UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

RECONNAISSANCE D'OBJETS PAR APPRENTISSAGE MACHINE AVEC VISION LIMITÉE SUPPLÉMENTÉE PAR DONNÉES AUDIO ET TACTILES SIMULÉES

ESSAI PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

YANNICK DINEL

MARS 2026

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Cet essai intitulé :

RECONNAISSANCE D'OBJETS PAR APPRENTISSAGE MACHINE AVEC VISION LIMITÉE SUPPLÉMENTÉE PAR DONNÉES AUDIO ET TACTILES SIMULÉES

présenté par
Yannick Dinel

pour l'obtention du grade de maître ès science (M.Sc.)

doit être évalué par un jury composé des personnes suivantes :

Dr. Ana-Maria Cretu ..... Superviseure de l'essai
Dr. Mohand Allili ..... Membre du jury
Dr. Rokia Missaoui ..... Présidente du jury

Dépôt initial de l'essai le : 17 Mars 2026

Table des matières

Liste des figures

Liste des tableaux

Résumé

Le nombre de technologies employant les capacités de l'apprentissage machine est en constante évolution dans la société moderne, surtout au niveau de la reconnaissance et la classification d'objets. Ces modèles ont plusieurs applications en sécurité et en robotique, mais ceux-ci sont généralement entraînés que sur des données visuelles. Nous identifions alors la possibilité d'explorer l'utilisation de données tactiles et audio conjointement aux données visuelles lors de l'entraînement afin d'améliorer ces modèles. De plus, nous explorons l'importance d'utiliser des points saillants du point de vue de l'attention visuelle lors de la sélection des données d'entraînement.

Puisqu'il existe peu d'ensembles de données à modalités multiples sur le Web, nous décidons d'employer l'outil *ObjectFolder 2.0* afin de générer des images visuelles et tactiles simulées ainsi que des fichiers audio simulant des impacts sur divers objets. Nous documentons au passage le fonctionnement de cet outil en plus des modifications apportées afin de supporter le processus de préparation de données.

Nous entraînons ensuite une série de modèles de classification d'apprentissage profond basés sur *MobileNetV2*, dont certains utilisant une modalité unique et d'autres utilisant des modalités combinées. Les données d'entraînement sont simulées à partir de plusieurs sous-ensembles de points sur divers objets, dont un ensemble de points saillants contre cinq ensembles de points sélectionnés aléatoirement.

Nous démontrons alors que l'utilisation des points saillants permet de créer des modèles visuels plus précis et que leur utilisation avec les autres modalités permet d'accélérer la convergence des modèles lors de l'entraînement. Nous montrons aussi que la combinaison des données tactiles et visuelles dans un même modèle permet d'offrir une performance comparable au modèle purement visuel équivalent, peu importe l'ensemble de points choisi. Toutefois, nous observons que les données audio apportent une perte de performance lorsque combinées à ces modèles, induisant un niveau de confusion entre les objets avec une composition matérielle similaire.

Chapitre 1

Introduction

L'apprentissage machine est un aspect important de la société moderne. Des modèles de détection sont utilisés dans de nombreuses applications, tel qu'en sécurité pour la détection d'intrusion dans des réseaux informatiques, ou bien dans le téléphone intelligent des utilisateurs communs pour identifier des concepts dans leurs photos. Toutefois, certains de ces modèles n'atteignent pas toujours le même niveau de performance qu'un être humain, particulièrement ceux développés pour la reconnaissance visuelle.

En effet, selon un article de Rachel Gordon publié par le *Massachusetts Institute of Technology* [1], les humains sont plus efficaces pour la reconnaissance d'objets que des modèles entraînés sur les ensembles d'images communs. Cette difficulté est en partie causée par un manque de standardisation par rapport à la complexité des images et le temps qu'un humain prend à en identifier le contenu. La question est donc, que manque-t-il aux processus internes de nos modèles actuels pour mieux représenter les mécanismes du cerveau humain et améliorer leur performance dans les cas plus complexes ? Selon les chercheurs au *MIT Computer Science and Artificial Intelligence Laboratory* (*CSAIL*), ces modèles devraient comporter une composante temporelle, soit le temps minimum requis pour un être humain afin d'identifier le sujet d'une image.

Suivant une étude de Ghazal Rouhafzay, on y reconnaît que le mécanisme d'identification visuelle employé par l'être humain comporte des éléments qui ne sont pas toujours inclus à l'étape d'acquisition de données lors du développement de modèles d'apprentissage machine, n'utilisant souvent que les données visuelles [2, p. 5]. Où les chercheurs du *CSAIL* explorent le concept du temps requis pour identifier un objet, Ghazal Rouhafzay améliore l'efficacité de son modèle en y intégrant des données tactiles et un mécanisme d'identification de points visuels saillants.

1.1 Identification de la problématique

Les modèles de reconnaissance d'objets actuels sont principalement basés sur l'utilisation de données purement visuelles. Ces modèles, supposant généralement une vue complète des objets à détecter, obtiennent de bons résultats. Plusieurs exemples de modèles de reconnaissance d'objet utilisant l'apprentissage profond existent, tels que AlexNet [3], VGG [4], ResNet [5, 6] ou *MobileNet* [7, 8].

MobileNet, de sa petite taille, montre son utilité en robotique où les contrôleurs et capteurs ont une puissance de calcul limitée. Toutefois, il est possible que l'objet à identifier ne soit pas complètement visible et que davantage d'informations sur l'objet soit nécessaire pour le fonctionnement du robot. Dans ce cas, il serait possible d'utiliser une combinaison de données visuelles, tactiles et audio afin de bien identifier l'objet et sa composition. *MobileNet* reste par contre un modèle purement visuel, une adaptation de ce modèle avec l'apprentissage par transfert permettrait l'utilisation de ces modalités supplémentaires.

Entraîner un tel modèle sur un ensemble de données intégrant ces trois modalités montre une difficulté particulière, où la disponibilité de tels ensembles sur le Web est limitée. En effet, où l'acquisition de données visuelles en larges quantités est possible avec une caméra, les données tactiles et audio nécessitent l'emploi de capteurs spécialisés qui doivent toucher directement la surface de l'objet. De plus, la configuration de cet équipement pour l'extraction de telles données requiert un système de positionnement robuste afin d'éviter d'endommager les capteurs ou la surface de l'objet.

Faute d'accès à des données exhaustives, nous devons alors explorer le concept de simulation de données. Bien que l'utilisation de données entièrement simulées puisse soulever des questions par rapport à la généralisation de modèles entraînés à partir de celles-ci, elles nous permettent d'effectuer une démonstration de faisabilité avec un minimum de ressources. Certains projets existent à cette fin, tels que *ObjectFolder* [9], permettant de générer des données simulées visuelles, tactiles et audio à partir de points arbitraires sur la surfaces des objets. Une exploration initiale de cet outil révèle toutefois un processus d'utilisation contre-intuitif, une attention particulière devra donc être apportée à sa documentation.

De ce, la question de positionnement pour ce projet est donc la suivante : « Comment pouvons-nous améliorer la détection d'objets à l'aide de données multi-modales (visuel, tactile et audio) dans un contexte où l'accès à de telles données est limitée et doivent

donc être simulées ? ». De plus, nous espérons que ce projet puisse servir de preuve de concept pour une éventuelle validation utilisant des données réelles.

1.2 Objectifs de l'essai

Suivant l'identification de la problématique, nous décrivons ici six objectifs pour cet essai, soit :

1. Documenter le fonctionnement et les modifications apportées à l'outil de simulation d'objets *ObjectFolder* ;
2. Développer des modèles de reconnaissance d'objets entraînés à partir de données visuelles, tactiles et audio simulées avec *ObjectFolder* ;
3. Développer des méta-modèles combinant ces modalités individuelles suivant une stratégie de fusion basée sur le *Stacking* ;
4. Utiliser les points saillants d'attention visuelle identifiés dans les travaux de Ghazal Rouhafzay afin de raffiner les modèles en sélectionnant ces points lors de la simulation de données avec *ObjectFolder* ;
5. Évaluer la viabilité et l'efficacité de l'utilisation de ces points, comparativement à des points sélectionnés aléatoirement, pour la collecte de données visuelles, tactiles et audio ;
6. Comparer les résultats des différents modèles créés lors de ce projet afin d'évaluer l'impact de l'utilisation des points visuellement saillants et de données multimodales pour la reconnaissance d'objets.

1.3 Hypothèses

En entraînant un modèle d'apprentissage machine sur des données visuelles simulées, nous espérons obtenir un résultat comparable à la littérature.

En limitant la vision aux points saillants sur les objets, nous nous attendons à ce que les résultats subissent une perte de précision causée par la perte d'informations visuelles. Toutefois, cette perte devrait être limitée supposant l'importance de ces points selon les travaux de Ghazal Rouhafzay [2].

En ajoutant les composantes tactiles et audio au modèle, nous espérons que la précision du modèle sera améliorée par rapport à l'utilisation de données visuelles seules.

1.4 Structure de l'essai

Une étude de la littérature connexe sur le sujet est effectuée au chapitre 2, menant à une étude plus approfondie sur les différentes approches de reconnaissance d'objets, des travaux de Ghazal Rouhafzay [2] sur les techniques avancées d'identification, les ensembles de données disponibles ainsi que les différentes versions d'*ObjectFolder* offertes par Gao et al. [10–12] comme sources de données simulées.

Ensuite, le chapitre 3 aborde la méthodologie employée dans le développement d'un modèle de reconnaissance d'objets suivant la méthode *CRISP-DM* (« *CRoss Industry Standard Process for Data Mining* »). On y décrit le projet *ObjectFolder 2.0* [11] à la section 3.3, où l'exploration et les étapes nécessaires au fonctionnement du code sont expliquées. Une série d'images visuelles et tactiles ainsi que des fichiers audio sont générés à partir de points saillants fournis par Ghazal Rouhafzay afin d'entraîner des modèles d'apprentissage machine à la section 3.4.

Les résultats de ces modèles sont examinés au chapitre 4 afin de corroborer les découvertes de Ghazal Rouhafzay [2] et confirmer l'utilité des techniques « *Sim2Real* » [11] dans un pipeline d'apprentissage machine lorsque des données réelles sont difficilement accessibles.

Finalement, le chapitre 5 offre une conclusion au projet, discutant des objectifs atteints, des découvertes et apports à l'état de l'art de ce projet ainsi que des améliorations possibles.

Chapitre 2

Revue de la littérature

Cette section porte sur la littérature disponible dans le domaine de la reconnaissance d'objets à l'aide de l'apprentissage machine.

2.1 Reconnaissance d'objets avec l'apprentissage machine classique

La reconnaissance d'objets utilisant des algorithmes d'apprentissage classique repose sur un concept clé, soit la génération de caractéristiques à partir des images afin de détecter des objets et ensuite les classifier. Ces techniques emploient des transformateurs définis manuellement, tels que, par exemple, des fenêtres coulissantes afin d'effectuer la détection de contours [13].

En effet, selon Chinmoy Borah dans un article publié sur *Medium.com* en 2020, ce type de techniques atteint son plateau aux alentours de 2010, où les modèles d'apprentissage profond sont capables d'effectuer automatiquement cette étape d'extraction de caractéristiques, rendant les techniques manuelles obsolètes [14].

Néanmoins, il est important d'étudier ces approches classiques afin d'avoir une bonne compréhension de leur importance dans le développement de l'apprentissage profond. Les sections 2.1.1, 2.1.2 et 2.1.3 décrivent quelques exemples de ce type de méthodes.

2.1.1 Technique avec analyse de la forme de l'objet

Dans un travail de Zhi Kai Ku et al. [15], l'extraction de caractéristiques repose sur la détection de silhouette en format binaire (objet blanc, arrière-plan noir), la détection

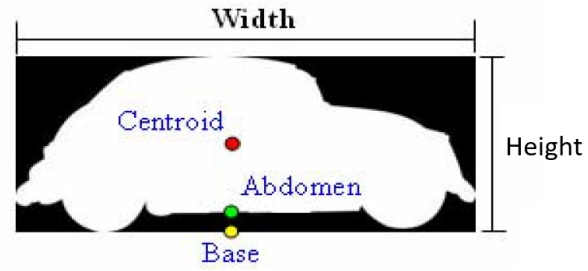

FIGURE 2.1 – Extraction de caractéristiques dans le travail de Zhi Kai Ku et al. [15], où on observe une silhouette d'un véhicule avec la définition de la largeur (« *Width* »), la hauteur (« *Height* ») et divers points d'importance. L'image a été modifiée afin de réintroduire le mot « *Height* », qui a été coupé par un image adjacente dans le texte original.

de points saillants tels que le centre de masse, ainsi que le calcul de ratios entre la largeur et la hauteur de l'objet à classifier. La classification est effectuée à partir d'un arbre de décision défini manuellement et est limité à quatre types, soit des formes humanoïdes, des formes animales quadrupèdes, des voitures et des formes divers. On observe un exemple des caractéristiques obtenues à la figure 2.1.

Bien que ce travail n'emploie pas de techniques d'apprentissage machine puisque l'arbre de décision est statique, il s'agît tout de même d'un exemple d'un exercice d'extraction de caractéristiques. Avec l'apprentissage machine, il est possible de générer automatiquement ces arbres de décision à partir des données. Ces méthodes classiques nous offrent un contexte important aux travaux de l'époque et montrent les idées de base qui mèneront éventuellement aux techniques plus modernes.

2.1.2 Technique avec descripteurs SIFT et modèle SVM

Les descripteurs *SIFT* (« *Scale Invariant Feature Transform* ») permettent d'identifier des ensembles de points localement saillants sur des images indépendamment de leur largeur et de leur rotation (d'où « *Scale Invariant* »). Ce type de descripteur est souvent utilisé comme méthode d'extraction de caractéristiques dans les images avant la montée de l'apprentissage profond.

Ai et al. [16] implémentent dans leur travail une variante de ce type de descripteur basé sur la distribution des couleurs de l'objet dans le but d'améliorer la séparation entre l'objet et l'arrière-plan dans une image. Plus spécifiquement, leur travail identifient que les *SIFT* existants implémentant une composante de couleurs souffrent d'un biais selon

la catégorie d'objets sur lesquelles ils sont appliqués. Les auteurs proposent alors un nouveau type de *SIFT* où la distribution des couleurs est normalisée selon la catégorie de l'objet à identifier, nommé *CIC-SIFT* (« *Color Independent Components SIFT* »).

Une fois les caractéristiques *SIFT* extraites, un modèle d'apprentissage classique de type *SVM* (« *Support Vector Machine* ») est utilisé pour classifier les objets. Les auteurs appliquent leur technique sur l'ensemble de données SIMPLIcity [17] et obtiennent un taux de classification de 82%, une amélioration d'environs 2% par rapport à l'utilisation du meilleur *SIFT* basée sur les couleurs (*HSV-SIFT*, « *Hue, Saturation, Value* ») et une amélioration de plus de 6% par rapport à un *SIFT* classique basé sur des images en tons de gris.

2.1.3 Technique avec détection de sous-composantes des objets

On observe dans le travail de Selvaraj et al. [18] l'idée de détecter et classifier des objets en entraînant un modèle détectant leurs sous-composantes, citant une difficulté de classification de l'objet entier lorsqu'il y a des obstructions dans l'image. Leur modèle est entraîné pour la détection de voitures et de piétons en détectant des sous-composantes telles que les roues, les fenêtres et les phares sur les voitures, ainsi que les visages, les bras, les jambes et le torse pour les piétons.

Il y a un parallèle intéressant avec cette technique et les algorithmes d'apprentissage machine profond, où l'extraction de caractéristiques décompose les images en hiérarchies afin de détecter les objets [19].

2.2 Reconnaissance d'objets avec l'apprentissage profond

Comme indiqué dans [14], les techniques classiques atteint rapidement une limite, où l'apprentissage profond fait ses preuves en utilisant la puissance de calcul accrue des ordinateurs modernes afin d'effectuer automatiquement l'étape d'extraction de caractéristiques avec des réseaux de neurones. Puisque l'objectif de ce projet est d'employer l'apprentissage profond pour effectuer la reconnaissance d'objets, nous explorons dans cette section un bref historique de modèles clés.

Ces modèles utilisent plusieurs techniques et ensembles de données, surtout pour les situations où les objets sont complètement visibles sur les images. Le dataset ImageNet

[20] étant l'un des plus importants, celui-ci fût utilisé pour alimenter divers modèles tels que *AlexNet* [3], *VGG* [4], *Resnet* [5,6] et *MobileNet* [7,8].

2.2.1 ImageNet

Le dataset d'images *ImageNet* [20], développé par Jia Deng et al., est disponible depuis 2009 et rassemble plus de trois millions d'images classifiés hiérarchiquement dans le but d'offrir une base solide pour l'avancement des technologies de reconnaissance d'objets. La hiérarchie est organisée de manière parallèle à *WordNet* [21], une base de donnée d'ensembles de mots reliés par concepts nommés « *Synonym Sets* », ou « synsets ».

Au moment de sa première mise en ligne, *ImageNet* offre 500 à 1000 images par *synset* pour plus de 5000 *synsets* sur les 80000 dans *WordNet*. Chaque année, le dataset est utilisé dans le « *ImageNet Large Scale Visual Recognition Challenge* » (*ILSVRC*) [22] où des développeurs de modèles d'apprentissage machine tentent de mettre au point le modèle le plus performant. Le dataset continue d'évoluer et comporte en août 2014 presque 22000 *synsets* de *WordNet* pour un total de plus de 14 millions d'images [22, p.3].

Parmi les nombreux modèles participant aux compétitions *ILSVRC* au fil des années, certains se démarquent, notamment *AlexNet* [3], *VGG* [4] et *ResNet* [5]. Ces modèles basés sur les réseaux neuronaux convolutifs profonds (« *Deep Convolutional Neural Networks* », *DCNN*) montrent l'efficacité de cette technique avec l'aide des processeurs graphiques de plus en plus puissant maintenant disponible sur le marché.

Erreur top-5 (top-k)

Les modèles explorés aux sections suivantes ayant participé au *ILSVRC* utilisent la métrique « *Top-5 Error* », ou l'erreur top-5, afin d'évaluer leur précision. Étant des modèles à prédiction multi-classes, chaque entrées dans ces modèles génèrent un ensemble de prédictions pour chaque classe possible dans l'ensemble de donnée d'entraînement.

Lorsque le modèle est testé, pour chaque image prédite, une erreur est comptée lorsque la classe de l'image n'est pas dans l'ensemble des cinq prédictions les plus élevées en sortie du modèle pour cette image. Le but de la compétition est donc de minimiser ce taux d'erreur.

Il est à noter que c'est métrique est applicable pour un nombre arbitraire k, mais les deux options les plus populaires sont top-1 (prédiction exacte) et top-5.

2.2.2 L'architecture d'un DCNN

Puisque les modèles explorés dans les sections suivantes sont basées principalement sur l'architecture d'un réseau de neurones convolutif profond, ou « *Deep Convolutional Neural Network* » (*DCNN*), il est important d'expliquer brièvement leur fonctionnement.

Suivant la figure 2.2, un *DCNN* est construit à partir des composantes suivantes :

- Une couche d'entrée ;
- Des couches de *convolution* et de *pooling* ;
- Des couches cachées complètement interconnectées ;
- Une couche de sortie, avec un neurone par classe.

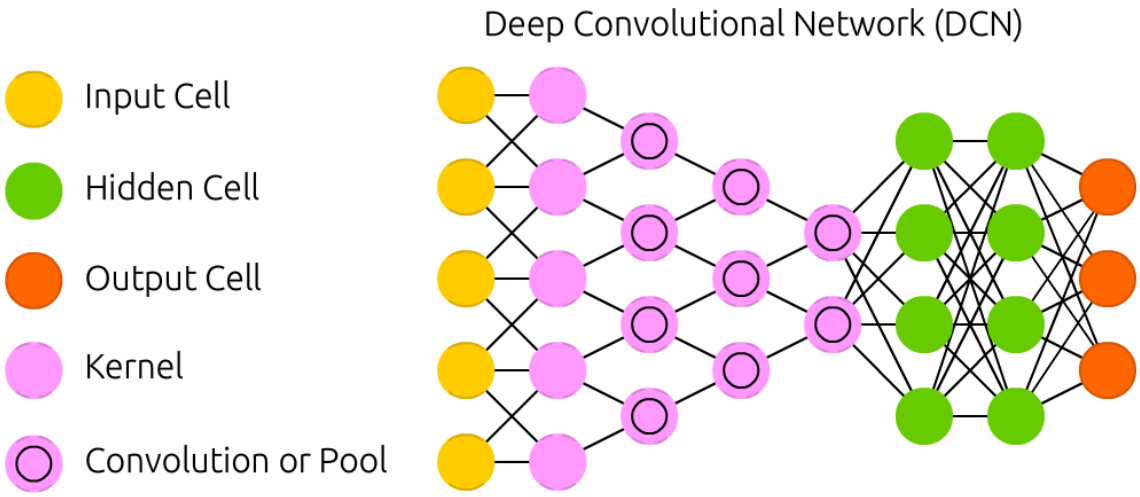

FIGURE 2.2 – Exemple de l'architecture d'un DCNN, tel que décrit dans [23].

Cette architecture est principalement utilisée dans le traitement d'image, où les couches convolutives et de *pooling* permettent d'extraire les caractéristiques de haut niveau de l'image avant de les envoyer dans les couches cachées, réduisant considérablement le nombre de poids à gérer dans ces couches ainsi que le risque de sur-apprentissage. Sans cette composante, les couches cachées devraient être entraînées pour prendre en entrée chaque pixels de l'image. Par exemple, pour une image de 256×256 pixels, ces couches auraient une taille de 65536 neurones.

Plus spécifiquement, la première couche prend en entrée l'image entière, suivi des couches convolutives où chaque neurones sont connectées à une petite section de l'image

(figure 2.3). Les poids de ces connexions représentent un filtre (ou « *Kernel* ») pouvant être entraîné afin de détecter les caractéristiques les plus importantes de l'image.

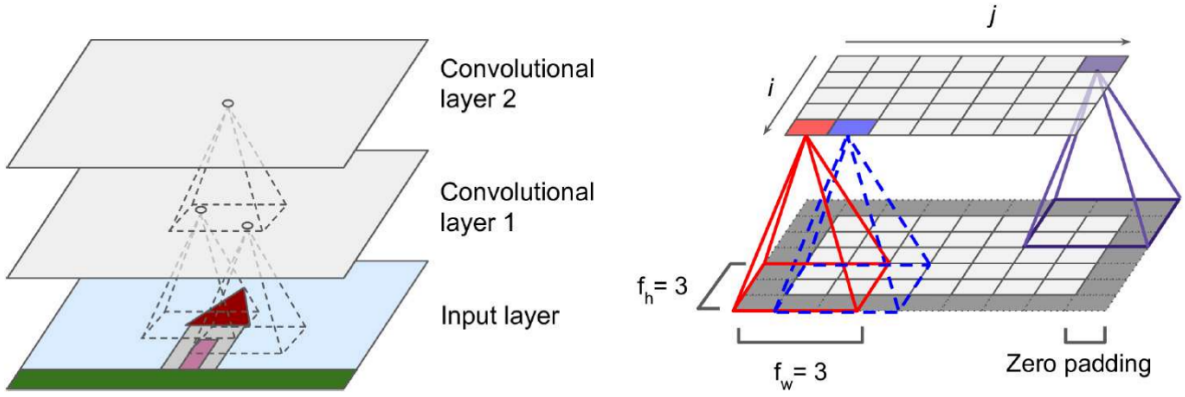

FIGURE 2.3 – Exemple de convolution dans un DCNN, tel que décrit dans [24].

Ensuite, les couches de *pooling* permettent de réduire la taille des entrées d'une manière plus simple que les couches convolutives. Chaque point de ces couches sont connectées à une petite fenêtre de l'entrée et une fonction est appliquée pour sélectionner la valeur en sortie pour ce point. La fonction choisie est à la discrétion du développeur du modèle et peut être, par exemple, la fonction *Max* (la pixel la plus intense est conservée), ou la fonction *Average* (la moyenne des intensités des pixels).

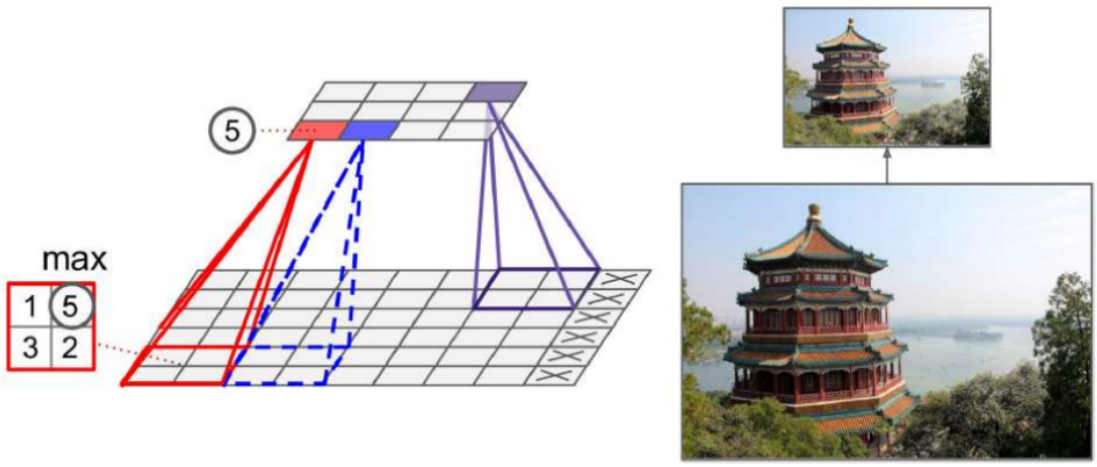

FIGURE 2.4 – Exemple de *pooling* avec la fonction *Max* dans un DCNN, tel que décrit dans [24].

2.2.3 AlexNet

AlexNet [3] est un *DCNN* développé dans le contexte de la compétition *ILSVRC-2012* par Alex Krizhevsky et al., gagnant avec un taux d'erreur top-5 de 15.3%, le deuxième meilleur ayant un taux de 26.2%. Le modèle est conçu à la base utilisant le dataset du *ILSVRC-2010*, puis adapté pour fonctionner sur le dataset de la compétition de 2012.

Parmi les avancés marquantes de leur projet, on observe l'utilisation de deux processeurs graphiques en parallèle pour entraîner les différentes couches du *DCNN*. Pour la plupart des couches, les processeurs graphiques sont indépendants, avec seulement certaines couches permettant la communication entre eux. Cette architecture permet d'améliorer l'efficacité du modèle tout en réduisant le temps requis pour son entraînement.

On observe aussi la mise en avant de la fonction linéaire rectifiée (« *Rectified Linear Unit* », ReLU, figure 2.5) comme fonction d'activation pour les neurones du réseau. Contrairement à d'autres fonctions nonlinéaires utilisées à l'époque, telles que *tanh(x)*, la fonction ReLU permet un apprentissage plus précis et rapide tout en réduisant les risques de surapprentissage, à condition que le modèle et l'ensemble de données d'entraînement soient suffisamment larges.

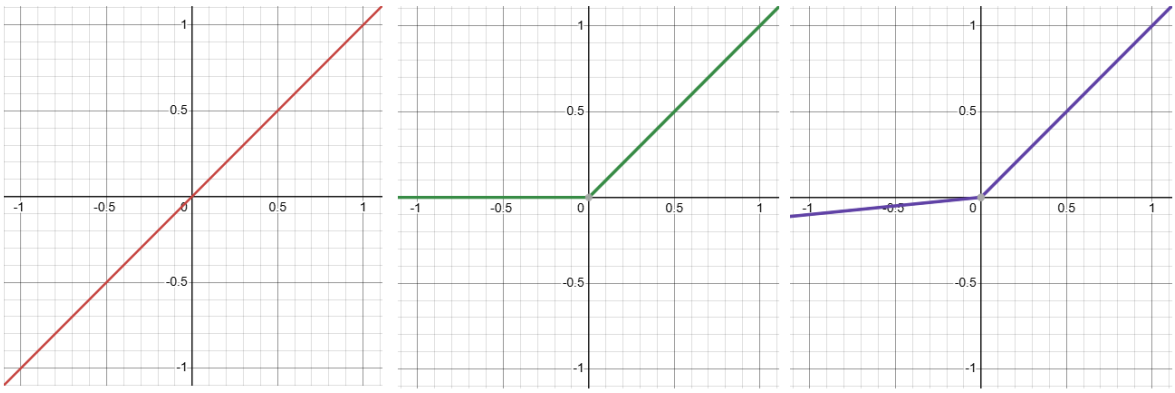

FIGURE 2.5 – Exemples de fonctions d'activation linéaires. En rouge, une fonction linéaire classique. En vert, une fonction linéaire rectifiée, où les valeurs négatives de x sont mises à 0. En mauve, une fonction linéaire rectifié avec fuite (« *leaky* »), où les valeurs de x passent dans une fonction linéaire avec une pente réduite lorsque x est négatif.

Finalement, lors du regroupement des neurones (*pooling*) entre les couches de convolution, le modèle dévie de la norme et permet aux unités de regroupement de se che-

vaucher (figures 2.6 et 2.7), réduisant légèrement les taux d'erreur et les risques de sur-apprentissage.

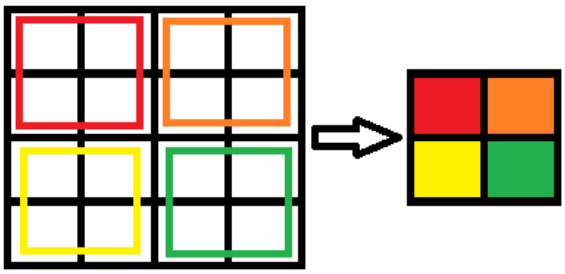

FIGURE 2.6 – Exemple d'une unité de regroupement avec une matrice 4×4 en entrée et une matrice 2×2 en sortie. Sans chevauchement, chaque cellule dans la matrice d'origine contribue à une seule cellule en sortie.

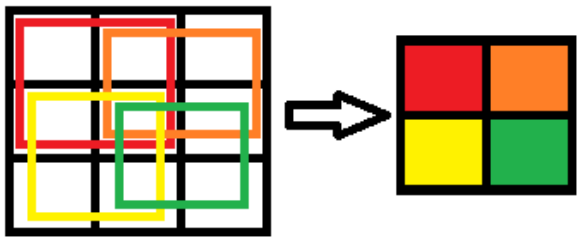

FIGURE 2.7 – Exemple d'une unité de regroupement avec une matrice 3×3 en entrée et une matrice 2×2 en sortie. Avec chevauchement, chaque cellule dans la matrice d'origine peut contribuer à plus d'une cellule en sortie.

2.2.4 VGG

Le modèle *VGG* [4] a été développé par Simonyan et Zisserman pour le *Visual Geometry Group* lors du *ILSVRC-2014*. Les auteurs prennent en compte les avancés apportées par AlexNet lors du *ILSVRC-2012* ainsi que certains participants de la compétition précédente, le *ILSVRC-2013*, pour créer un modèle de reconnaissance d'objets encore plus robuste, atteignant un taux d'erreur top-5 de 7.5% lors de la compétition de 2014.

Toujours basé sur un *DCNN*, le modèle *VGG* ajuste principalement les paramètres de profondeur du réseau et la taille des fenêtres de convolution. Où *AlexNet* utilise cinq couches de couches de convolutions avec des fenêtres de plus en plus petites (11x11, 5x5, 3x3 pour les trois dernières) [3, p.4,5], *VGG* utilise entre 8 à 16 couches de convolution 3x3 selon sa configuration.

Autre que les changements par rapport aux couches de convolution, *VGG* est configuré de façon similaire à *AlexNet*, utilisant la fonction d'activation *ReLU* pour les couches cachées du réseau. Finalement, les auteurs concluent alors que sacrifier la largeur des couches de convolution pour permettre un réseau convolutif plus profond est préférable pour améliorer la précision des modèles de reconnaissance d'objets.

2.2.5 ResNet

Entre 2014 et 2016, on observe une évolution des *DCNN* vers les réseaux profonds résiduels, ou *ResNets*. Suite aux résultat prometteurs de modèles très profonds tels que *VGG*, He et al. [5] se posent la question s'il est réellement aussi simple que de créer des modèles de plus en plus profond pour obtenir une meilleure performance dans les tâches de reconnaissance d'objets. Ils arrivent à la conclusion qu'il doit y avoir un changement à l'architecture de base des réseaux convolutifs conventionnels afin de supporter de telles profondeurs.

Fondamentalement, les couches d'un réseau de neurones prennent un vecteur de données en entrée et effectuent un mappage vers un nouveau vecteur en sortie suivant une étape d'entraînement. Dans le cas où les vecteurs en entrée et en sortie sont de la même dimension, une amélioration est identifiée par les auteurs de cet article. En effet, plutôt que d'entraîner les couches pour résoudre un mappage direct sur les données ($y := F(x)$, où $F(x)$ est la fonction contenant les poids à apprendre par la couche du réseau), on introduit une connexion de type « raccourci » pour le vecteur en entrée pour résoudre un mappage résiduel ($y := F(x) + x$). Le tout constitue une *unité résiduelle* (« *Residual Unit* »), voir la figure 2.8.

Ce changement d'architecture permet de réduire l'incidence de la dégradation des performances des réseaux de neurones convolutifs trop profonds en concentrant l'entraînement des couches sur la partie variable de la sortie. Ceci permet à He et al. de terminer l'*ILSVRC-2015* avec une erreur top-5 de 3.57%, remportant la première place [5, p. 776].

2.2.6 ResNetV2

En 2016, He et al. publient un article sur *ResNetV2* [6] peu de temps après leur travail précédent. Les auteurs parviennent à améliorer davantage l'unité résiduelle, permettant un entraînement plus rapide et une capacité de généralisation accrue. Un des changements principaux est de permettre au modèle de propager les mappages d'identité des unités résiduelles au travers le modèle en entier. Ceci est accompli en modifiant l'unité résiduelle pour appliquer la fonction d'activation seulement à la partie non-résiduelle de l'unité, ce qui permet d'éviter que la composante résiduelle ne soit pas tronquée d'une unité à la prochaine. La figure 2.9 montre la différence entre leur méthode originale et les modifications qu'ils ont apportées.

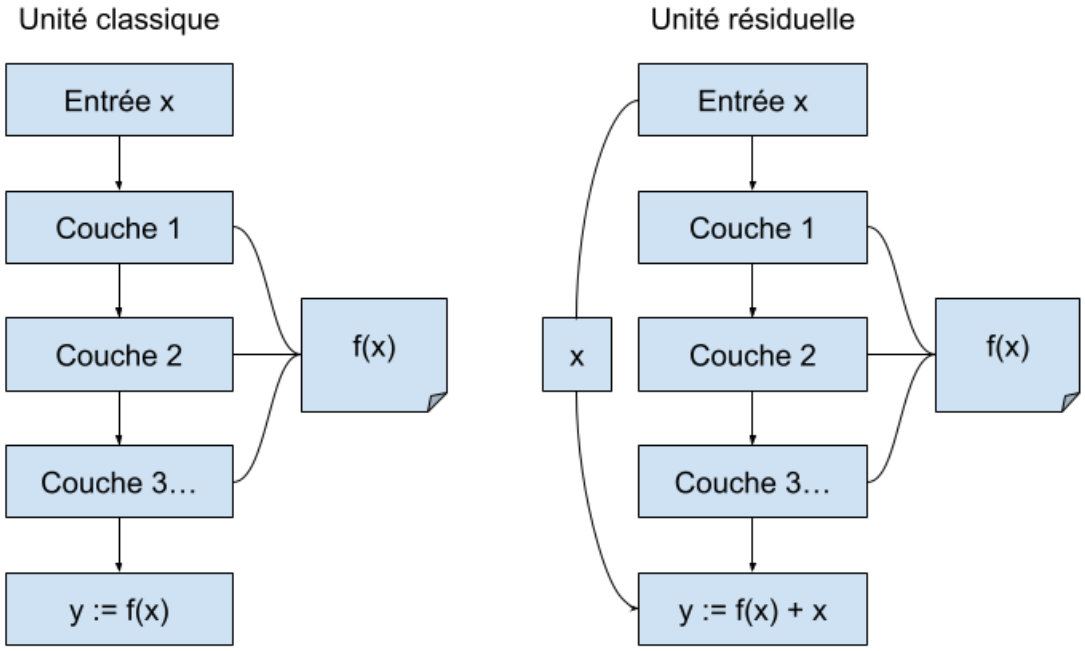

FIGURE 2.8 – Exemple d'une unité classique dans un modèle d'apprentissage machine (à gauche) contre une unité résiduelle. Les différentes couches peuvent représenter des convolutions, des fonctions d'activation ou des couches avec poids. Inspiré de la figure 2 dans [5]

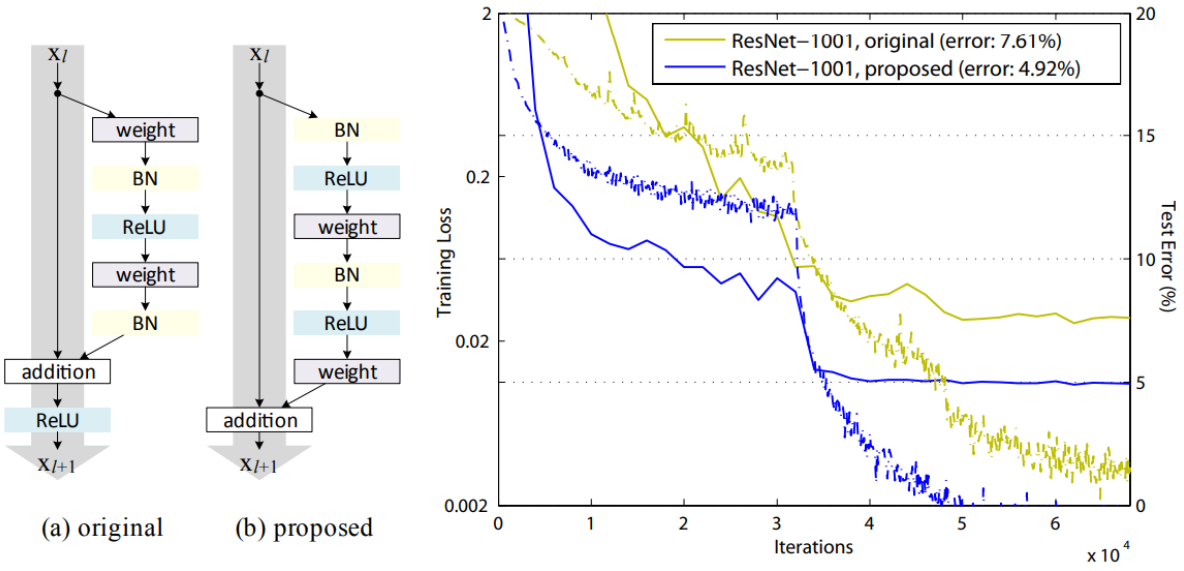

FIGURE 2.9 – Figure provenant du travail de He et al. [6, Fig. 1]. À gauche, on remarque le déplacement et la réorganisation des composantes de l'unité résiduelle afin que la fonction d'activation ReLU n'ait aucune interaction avec la partie non-résiduelle de l'unité. À droite, les auteurs montrent que lorsqu'ils utilisent cette nouvelle unité résiduelle dans le même contexte que leur travail précédent, une réduction du taux d'erreur sur les données de test de 2.69% est observée.

2.2.7 MobileNet

Dans un article d'Andrew Howard et al. [7] publié en 2017, on observe un problème par rapport au cheminement et à la conception des différents modèles précédents, tels que *AlexNet* [3], *VGG* [4] et *ResNet* [5] [6]. Depuis AlexNet, les *DCNN* deviennent plus populaires et la technique de choix pour améliorer leur précision est de les configurer avec des architectures de plus en plus profondes au détriment de l'efficacité et du temps d'exécution. Les auteurs identifient alors le besoin d'effectuer la reconnaissance d'objet rapidement sur des composantes moins performantes dans les domaines tels que la robotique ou l'utilisation par le grand public sur des téléphones intelligents.

Le principal changement apporté à l'architecture *MobileNet* est l'utilisation de convolutions basées sur la profondeur plutôt que des couches convolutives standard. Prenant comme exemple une image, plutôt que d'effectuer une convolution sur les trois couleurs (canaux) de l'image à la fois, la convolution est effectuée sur chaque canaux individuellement suivi d'une convolution de taille 1×1 sur ceux-ci obtenir un résultat similaire à la convolution standard. Pour *MobileNet*, ce changement apporte une réduction de 8 à 9 fois sur le temps de calcul du modèle [7, p. 3].

Suivant d'autres optimisations permises par l'utilisation des convolutions 1×1, où le modèle effectue la majorité de ses calculs, les auteurs obtiennent une précision comparable à *GoogLeNet* [25] avec 2,5 fois moins de calculs, de même que 27 fois moins contre *VGG16* [4], tout en étant un beaucoup plus petit modèle [7, p. 6].

Finalement, les auteurs adaptent ensuite leur modèle pour la détection dynamique d'objets dans des images en suivant la méthodologie décrite par les gagnants de la compétition *COCO 2016* (« *Common Objects in Context* »), Huang et al. [26]. *MobileNet* obtient des résultats prometteurs pour une fraction du temps d'exécution par rapport à la compétition, où, par exemple, des modèles *MobileNet*, *VGG* et *InceptionV2* [27] sont entraînés utilisant la méthode de détection *SSD* (« *Single Shot MultiBox Detector* ») [28] avec des images d'une résolution de 300×300 pixels. Les trois modèles ont une précision comparable, mais *MobileNet* obtient sa performance avec un peu moins du tiers de calculs et la moitié des paramètres par rapport à *InceptionV2*.

2.2.8 MobileNetV2

Un article de Mark Sandler et al. [8] publié en 2019 apporte des améliorations à *MobileNet* en réduisant davantage le volume de calculs requis à l'aide d'un nouveau module,

soit une manière de combiner des couches dans un réseau de neurones. *MobileNetV2* implémente un « *inverted residual with linear bottleneck* » (couches résiduelles inversées avec goulet linéaire), où un vecteur à faible dimensionalité peut être augmenté, filtré à l'aide d'une convolution basée sur la profondeur et finalement recalculé en un plus petit vecteur à l'aide d'une convolution linéaire. Une connexion existe entre l'entrée et la sortie du module, d'où la partie résiduelle.

En général, cette implémentation permet d'obtenir un rendement similaire au *MobileNet* original en utilisant en moyenne le quart des ressources [8, Table 3, p. 5]. En entraînant les modèles sur *ImageNet* 2015 [22], *MobileNetV2* parvient à obtenir une performance légèrement supérieure à *MobileNetV1* avec un temps d'exécution réduit de 33% pour la même tâche [8, Table 4, p. 7].

L'architecture de ce modèle sera donc retenue pour ce projet, car sa petite taille et sa performance élevée permettront une vitesse d'itération rapide.

2.3 L'utilisation de données multi-sensorielles pour la reconnaissance d'objets

Puisque ce travail porte sur la reconnaissance d'objets à l'aide de données multi-sensorielles, nous discutons alors d'une thèse de Ghazal Rouhafzay [2] portant sur les applications de l'apprentissage machine et la reconnaissance d'objets dans le contexte de la robotique. Une importance particulière est apportée à cette thèse car nous utilisons dans ce projet un ensemble de points saillants obtenus par l'application des techniques y étant décrites.

Plus précisément, on y explore la modélisation 3D en temps réel d'objets dans une scène afin de permettre à un robot d'identifier les objets et leur constitution ainsi que d'évaluer la séquence d'actions nécessaire pour saisir un objet. Ghazal Rouhafzay identifie les problèmes suivants :

- Les capacités de calcul graphique dans les robots sont souvent limitées, la modélisation en temps réel d'une scène doit donc être le plus simple possible sans compromettre la performance quant à l'identification des objets et le fonctionnement du mécanisme de saisie ;
- Les modèles d'identification d'objets ont souvent de la difficulté avec les objets partiellement visibles dans les scènes ayant un éclairage sub-optimal où l'addition

de modalités supplémentaires au robot, telles qu'un sens du toucher, permettrait de combler les données visuelles et améliorer ses capacités. Toutefois, les capacités de calcul requises pour ajouter des modalités supplémentaires restent un défi.

Suivant ces deux points, Ghazal Rouhafzay propose un modèle inspiré du fonctionnement du cerveau humain où un module d'identification des points saillants est employé lors de l'exploration d'objets. En utilisant les caractéristiques géométriques de l'objet, un *CNN* estime l'endroit où les yeux d'un humain ont la plus grande probabilité de fixer. Ceci nous offre alors des points candidats pour la modalité tactile et audio d'un robot, permettant de maximiser les capacités d'identification avec un minimum de touchers et de coups effectués sur les objets.

2.3.1 L'identification des points saillants suivant l'attention visuelle

Il existe plusieurs techniques contemporaines pour la détection de points saillants sur des objets 3D. Par exemple, un article publié en 2018 par Lavoué et al. [29] implémente un modèle utilisant un ensemble de points fixés par des sujets humains. D'autres chercheurs utilisent des techniques purement géométriques, comme la détection de coins ou la détection de points en moyenne plus distants des autres sur la surface de l'objet.

La technique implémentée par Ghazal Rouhafzay est principalement basée sur le calcul de mesures géométriques à partir de multiples projections 2D de l'objet, ensuite projeté sur la représentation 3D de l'objet. Ce modèle d'attention visuelle est particulièrement adapté à un contexte de détection d'objets en robotique, où les images 2D initiales proviennent des caméras installées sur un robot.

La création d'un modèle de détection des points saillants est effectuée selon les étapes suivantes [2, fig. 3.9, pg. 66] :

1. Un ensemble de données provenant d'un travail de Dutagaci et al. [30] est utilisé, contenant des objets 3D complimentés d'ensembles de points saillants obtenus par des utilisateurs humains ;
2. Des images de ces objets 3D sont captées à partir d'une multitude de points de vue ;
3. Un ensemble de neuf indicateurs est calculé pour chaque image, dont la courbature, la détection de lignes et l'entropie ;

4. En parallèle à l'étape précédente, les points saillants connus de l'ensemble de données sont projetés sur l'objet pour obtenir un indicateur cible, suivant une méthodologie d'attention visuelle dirigée décrite dans un travail de Hughes et Zimba [31] où ces points sont représentés sous forme de zones d'attention ;
5. Les indicateurs calculés sont comparés à l'indicateur cible en notant leurs indices de similarité et distance ;
6. Un modèle de type *SVM* (« *Support Vector Machine* ») est entraîné pour chaque objet à partir des neuf indicateurs et de l'indicateur cible ;
7. Après une étape de nettoyage des données, les n-points les plus saillants sont identifiés sur les images des projection 2D à 3D sur les objets initiaux sont effectuées afin d'identifier les sommets (« *vertices* ») saillants [2, fig. 3.18, pg. 80].

Toutefois, puisque qu'un modèle doit être entraîné pour chaque objet en connaissance de leurs points saillants, cette approche est difficilement applicable à de nouveaux objets sans modification. L'auteure identifie alors que la convexité d'un objet est un des facteurs principaux à la performance des modèles entraînés. Il est donc possible de calculer le niveau de convexité d'un objet arbitraire et de lui assigner le modèle pour l'objet ayant la convexité la plus proche parmis ceux entraînés ci-haut.

L'auteure explore aussi un modèle alternatif de détection des points saillants utilisant la technique de visualisation *Grad-CAM* (« *Gradient-weighted Class Activation Mapping* ») sur des réseaux de neurones pré-entraînés basés sur *ResNET* et *VGG16*. Les dernières couches des ces modèles sont remplacées afin de classifier selon des caractéristiques physiques (convexité, courbature, excentricité) et sémantiques (composition, utilité) de l'objet. Bien que les résultats soient plus précis, ces modèles ont tendance à identifier un petit nombre de points saillants les plus importants pour un objet donné tandis que l'approche précédente utilisant des caractéristiques géométrique offre une vue d'ensemble de l'objet.

Malgré la précision réduite de l'approche géométrique, cette approche plus générale est mieux adaptée au problème de la réduction des détails sur les modèles 3D.

2.3.2 L'identification d'objets par touchers à l'aide de l'attention visuelle

Ghazal Rouhafzay pointe vers l'importance de combiner les modalités visuelles et tactiles pour obtenir un meilleur rendement, citant un article publié par Amedi et al. [32].

On y identifie la rapidité avec laquelle nous utilisons la vision pour obtenir des informations superficielles sur les objets, mais avec une incapacité à obtenir des informations précises sur leur texture et leur composition. La possibilité de complimenter une vision réduite avec des données tactiles afin d'identifier les objets est explorée.

En effet, selon Allen et Roberts [33], il est possible d'identifier un objet avec un nombre limité de touchers rapides. La vision permet de guider ce nombre limité de touchers vers les points les plus importants pour maximiser les chances de bien reconnaître l'objet. Attaquant le problème d'un point de vue inverse, un article de Kennett et al. [34] montre que les points saillants au toucher attirent plus souvent l'attention visuelle. Ceci montre que les deux modalités sont complémentaires.

Suivant une méthode de simulation de capteur tactile pour explorer le profil d'objets virtuels décrite dans un travail de Pezzementi et al. [35], il y est alors implémenté un processus pour identifier un objet uniquement à partir de données tactiles obtenues selon son modèle d'attention visuelle. Les touchés sur les points les plus saillants sont filtrés pour retirer les données similaires et les coordonnées des touchers sont inclus pour entraîner un modèle d'apprentissage classique.

Pour améliorer le processus de détection des objets, l'auteure introduit un système utilisant les contours tactiles inspiré des observations de Lederman et Klatzky dans leur travail sur la perception tactile chez les humains [36]. En suivant la forme d'un objet selon les points visuellement saillants et en entrainant un modèle pour classifier ces données en temps que séries temporelles, des résultats plus prometteurs sont atteints. Il est aussi supposé que le robot possède une série de senseurs tactiles de différentes tailles pouvant être sélectionnés selon le niveau de précision requis pour sonder une section donnée de l'objet.

Il y est alors identifié la difficulté d'entraîner un modèle de détection d'objet à partir de données purement tactiles et l'auteure explore la possibilité d'utiliser des techniques d'apprentissage par transfert. Cinq modèles existants sont explorés, soit *AlexNet* [3], *GoogLeNet* [25], *VGG16* [4], *ResNet50* [5] et *MobileNetV2* [8]. Il est démontré que *ResNet50* et *MobileNetV2* sont les plus performants suivant un apprentissage par transfert en utilisant des données provenant d'un capteur tactile basé sur une technologie optique, tels que *GelSight* [37] et *BathTip* [38]. Dans le cas de données provenant d'autres types de capteurs, une dégradation considérable de la performance de ces modèles est observée.

De deux alternatives explorées, la première implique des données provenant d'un capteur de type *FSR* (« *Force-Sensing Resistor* ») [39], où la résolution du capteur

(16×16) s'avère trop limitée pour permettre une bonne classification. La deuxième est une main robotique composée de trois capteurs sur trois doigts nommée *BarrettHand* [40], touchant l'objet à trois points et prenant les données sur une durée de temps, où le format de ces données est incompréhensible au modèle de base et donc incapable d'effectuer la classification des objets.

L'auteure se concentre sur *MobileNetV2* car sa petite taille s'adapte bien au domaine de la robotique, où la capacité du processeur sur le robot est comparable à un appareil mobile. Une architecture hybride est développée où les couches du modèle sont entraînées par transfert en utilisant à la fois des données visuelles et des données tactiles provenant d'un senseur de type optique.

2.4 La génération de données multi-sensorielles avec ObjectFolder

Afin d'implémenter nos travaux dans ce projet, nous explorons l'ensemble de modèles *ObjectFolder*. Trois variantes existent, soit la version originale *ObjectFolder 1.0*, son évolution *ObjectFolder 2.0* ainsi qu'un ensemble de données construit à partir de données réelles, *ObjectFolder Real*.

2.4.1 ObjectFolder 1.0

Le projet *ObjectFolder* de Gao et al. [10] a été créé dans le but de répondre à un problème soulevé par les avancées en robotique et la reconnaissance d'objets à l'aide de données multi-sensorielles, soit qu'il existe peu de données pour entraîner de tels modèles, soit les jeux de données d'objets simulés existants ne sont pas assez réalistes ou les ensembles d'objets réels commerciaux sont trop dispendieux ou difficiles à utiliser dans ce contexte [10, p.1, résumé].

En effet, on y note l'existence de plusieurs jeux de données volumineux pour l'entraînement de modèles 2D, tels que *ImageNet* [20] et *ObjectNet* [41]. On y note aussi *ModelNet* [42] et *ShapeNet* [43] comme étant des ensembles d'objets modélisés en 3D, mais manquant de données pertinentes par rapport à la texture des objets. Finalement, bien que les ensembles *YCB* [44] et *BigBIRD* [45] offrent une gamme d'images et de données pour une multitude d'objets communs et tout aussi dans le but d'avancées en perception robotiques, les auteurs d'*ObjectFolder* notent que leur propre jeu de données

a pour but de donner un accès facile à des données visuelles, tactiles et auditives [10, p.2, Related Work, Object-Centric Datasets], contrairement à ces autres ensembles.

Un survol des articles décrivant les ensembles *YCB* et *BigBIRD* montre une approche commune. L'ensemble *YCB* réutilise l'appareil de numérisation conçu pour *BigBIRD* (figure 2.10) avec une sélection d'objets différente de ce dernier. De ce, les deux ensembles offrent pour chaque objet un ensemble de 600 images *RGB-Depth*, un ensemble de nuage de points ainsi qu'une modélisation 3D pour chaque objet. On y note l'absence de données tactiles et auditives. Ce format de données ce prête mal aux applications en robotique visée par les auteurs d'*ObjectFolder*.

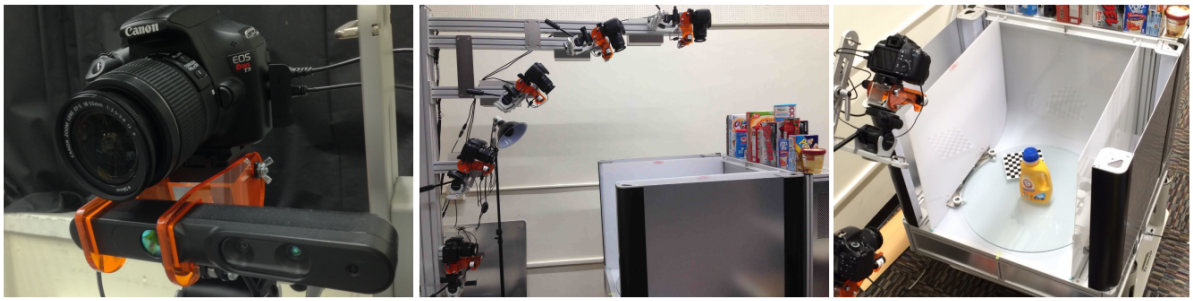

FIGURE 2.10 – L'appareil de numérisation d'objet 3D employé lors du développement des ensembles de données BigBIRD [45] et YCB [44].

La première itération d'*ObjectFolder* encode alors 100 différents objets virtuels à l'aide de réseaux de neurones, plus précisément des *MLP*s (Multi-Layer Perceptrons) basés sur les coordonnées. Ces réseaux offrant une représentation implicite des caractéristiques physiques de chaque objets permet la génération de données visuelles, tactiles et auditives simulées pour un point de vue et de contact arbitraire sur un objet donné.

Il est à noter que l'ensemble de matériaux possibles pour ces objets a été réduit au sous-ensemble de *céramique*, *vitre*, *bois*, *plastique*, *polycarbonate*, *fer* et *acier*. De plus, les objets choisis sont homogènes dans leur constitution pour assurer que la modalité auditive soit plus facile à simuler. 20 de ces objets proviennent du site web « *3D Model Haven* » [46] (maintenant « *Poly Haven* »), 28 du *YCB* [44] et 52 du *GSO* (« *Google Scanned Objects* ») [47].

Les différents *MLP*s sont entraînés à partir d'une série de simulations sur des fichiers de modèle 3D pour chaque modalité :

– Des rendus à l'aide de *Blender* [48] sont utilisés pour entraîner le modèle visuel *VisionNet* afin d'encoder une fonction *OSF* (« *Object-centric neural Scattering*

Function »). L'*OSF* prend en entrée un vecteur 7-dimensionnel encodant une coordonnée sur l'objet ainsi que les conditions lumineuses à cet endroit et donne en sortie une densité de volume et la fraction de lumière reflétée en direction de la caméra.

- Le réseau de points composant l'objet est converti en modèle volumétrique à base cubique. À l'aide de données sur la largeur et la composition matérielle de l'objet extrapolées à partir des trois sources de données (*3D Model Haven*, *YCB*, *GSO*), le modèle *AudioNet* est entraîné pour reproduire le spectrogramme généré par la vibration du volume dans plusieurs directions. Ces sous-images sont additionnées ensemble pour obtenir le spectrogramme final, pouvant ensuite être converti en fichier audio en sortie.
- Le modèle tactile est entraîné à partir d'un simulateur de toucher sous le nom de *TACTO* [49]. Une série de rayons sont projetés sur la surface de l'objet pour en évaluer la forme et inférer des images tactiles simulées dans un format Rouge-Vert-Bleu (*RGB*).

Les auteurs appliquent ensuite avec succès leurs modèles à, entres autres, des tâches de détection d'objets multi-modales ainsi que des simulations de saisie d'objets avec un bras robotique virtuel afin de montrer les avantages de l'utilisation conjointe des trois modalités décrites ci-dessus.

2.4.2 ObjectFolder 2.0

La plupart des auteurs d'*ObjectFolder 1.0* reviennent pour travailler sur la mise en place d'*ObjectFolder* 2.0 [11], visant à améliorer et corriger certains problèmes de la première version :

- Le nouveau dataset comporte 1000 éléments, incluant les 100 éléments de la version originale. La majorité, soit 855 objets, provient du dataset ABO (Amazon Berkeley Objects) [50]. Les 45 objets restants proviennent de l'ensemble *GSO* [47], spécifiquement que des objets de matériel « *polycarbonate* ».
- La qualité des images et des sons produits par la première version étaient considérées sous-optimales. Le nouveau dataset améliore grandement les rendus des trois modalités simulées, particulièrement au niveau du son et des images tactiles.
- Le temps de rendu des modalités simulées est grandement accéléré, permettant de générer des données en temps réel.

– Les auteurs montrent que leurs modèles s'appliquent à des tâches dans le monde réel, contrairement à *ObjectFolder 1.0* où les tests ont été effectués que sur des scénarios simulés.

Les objets sélectionnés sont de composition homogène et comportent les mêmes classes de matériaux que pour *ObjectFolder 1.0* (céramique, vitre, bois, plastique, polycarbonate, fer et acier).

On observe une grande amélioration des temps d'exécution au niveau de la vision entre *ObjectFolder 1.0* et *2.0*. Dans la version précédente, un simple *MLP* encodait la diffusion lumineuse pour l'entièreté de la scène. *ObjectFolder 2.0* implémente alors la technique *KiloOSF* (*Kilo Object-centric Neural Scattering Function*), inspirée des travaux de Christian Reiser et al. [51] où il est démontré qu'une scène peut être représentée par un ensemble de milliers de petits *MLP*s, chacun entraîné pour encoder une section précise de l'image. Cette technique permet d'avoir un résultat pratiquement autant précis en utilisant moins de ressources et en une fraction du temps d'exécution.

Pour la modalité audio, on observe certaines améliorations depuis la version originale d'*ObjectFolder*. Une représentation volumétrique à base de tétraèdres plutôt que de cubes est utilisée pour capturer davantage de détails dans la structure de l'objet. De plus, la représentation intermédiaire en spectrogrammes est évitée, puisque celle-ci force les sons générés à être d'une longueur et d'une résolution fixe. La nouvelle version d'*AudioNet* estime donc directement les modes de vibration de l'objet pour obtenir les fréquences sonores générées par un impact qui sont ensuite combinées pour obtenir un fichier audio. Le résultat final est un fichier plus clair représentant plus fidèlement le comportement de l'objet réel.

Finalement, la modalité tactile a été complètement repensée pour cette version d'*ObjectFolder*. La nouvelle version de *TouchNet* s'inspire des capteurs de type *GelSight* lors de la création des données simulées. Des fichiers de déformation sont générés pour différentes configurations de rotation et de position pour chaque point de l'objet qui sont ensuite utilisés pour entraîner un réseau *MLP*. Un modèle de simulation pour *GelSight*, nommé *Taxim* [52] est ensuite utilisé pour créer les images *RBG* correspondantes. Cette nouvelle méthode permet de générer plusieurs images pour le même point d'un objet selon des paramètres de rotation et de profondeur du capteur simulé.

2.4.3 ObjectFolder Real

Suite à la création d'*ObjectFolder 2.0*, Gao et al. continuent d'élargir les options disponibles avec *ObjectFolder Real* [12], un dataset de 100 objets offrant des données visuelles, auditives et tactiles dans un format similaire à *ObjectFolder 2.0*. Le but de ce projet est de servir de point de repère pour les modèles entraînés sur des données simulées et confirmer les capacités de transfert Sim2Real en évaluant leur performance sur des données réelles.

Puisque nous devons pouvoir sélectionner un point arbitraire sur la surface d'un objet, cet ensemble de données n'est pas considéré pour notre projet. Nous décidons donc d'utiliser *ObjectFolder 2.0* comme outil de simulation de données visuelles, tactiles et audio.

Chapitre 3

Méthodologie

Ce chapitre documente le processus de développement et la méthodologie employée pour développer un modèle de reconnaissance d'objets employant la méthode *CRISP-DM*. Une brève description du processus en général est visible à la figure 3.1, tandis que la figure 3.2 offre un diagramme indiquant les différentes composante de la méthodologie. Les sections à suivre explorent en détails les éléments de ces figures.

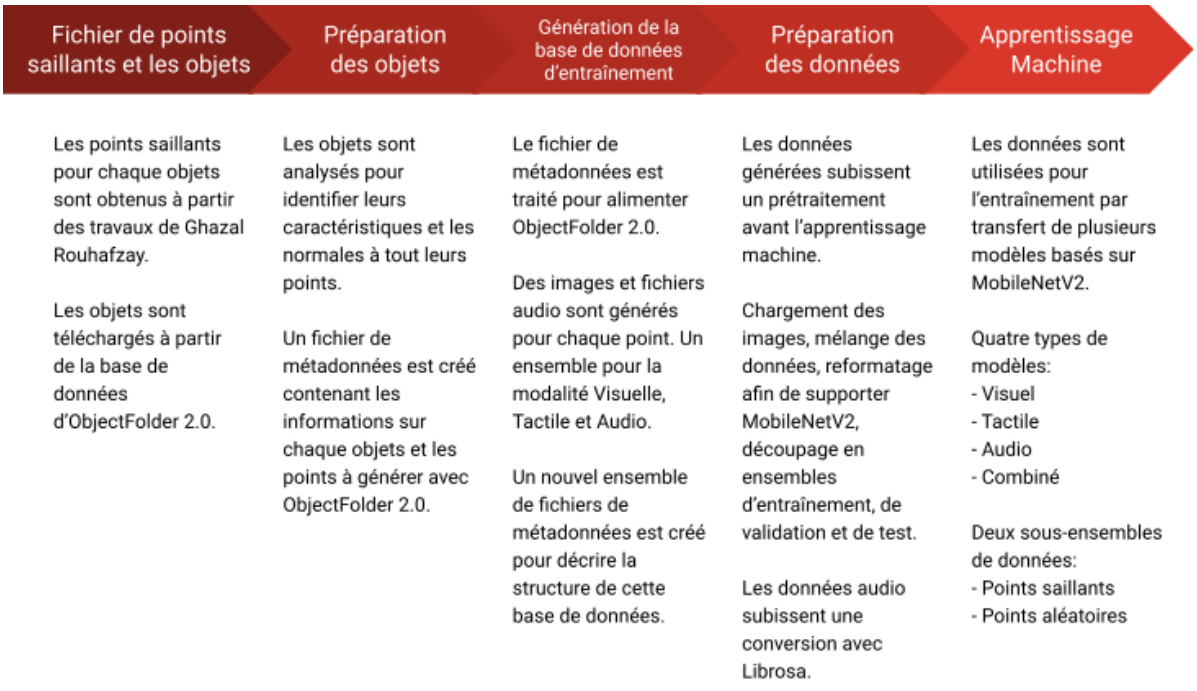

FIGURE 3.1 – Brève description de la méthodologie employée pour traiter les données et entraîner les modèles d'apprentissage profond.

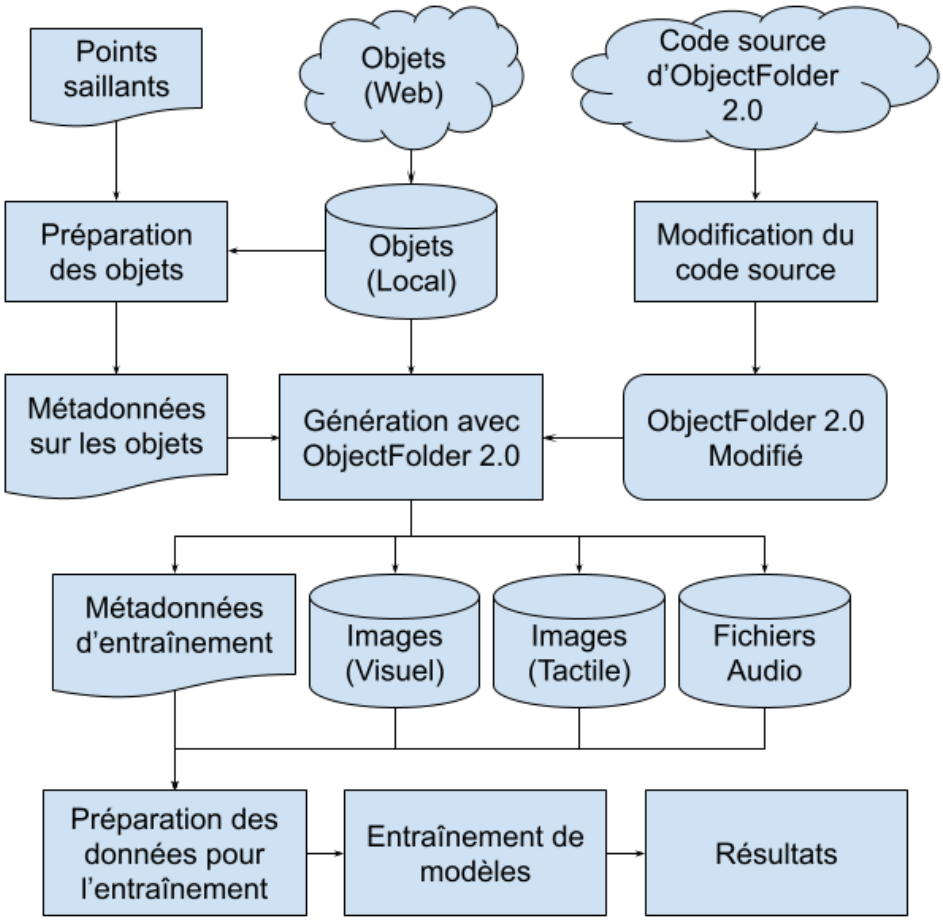

FIGURE 3.2 – Organigramme décrivant la structure générale de la méthodologie.

3.1 La méthode CRISP-DM

La méthode *CRISP-DM* définie le cycle de développement pour un projet de recherche en science des données. Suivant les ressources disponibles sur le site web d'IBM [53], un leader de cette industrie, la figure 3.3 offre un survol des différents éléments du processus.

Les six éléments de *CRISP-DM* sont les suivants :

- La compréhension de l'entreprise ;
- La compréhension des données ;
- La préparation des données ;
- La modélisation ;
- L'évaluation ;
- Le déploiement.

Dans le cas de cet essai, l'entreprise prend la forme de la recherche dans l'état de l'art et du contexte dans lequel nous travaillons. C'est-à-dire, la recherche dans le domaine de la détection et classification d'objets et son application en robotique. Les sections 2.3 et 2.4.2 offrent le contexte par rapport aux techniques et outils disponibles pour ce projet.

La compréhension des données comporte la phase d'exploration d'*ObjectFolder 2.0* à la section 3.3 de ce chapitre. On y documente le processus pour générer des données simulées et leur examen afin de bien comprendre et confirmer la documentation de cet outil.

La préparation des données est effectuée à la section 3.4.1, où l'on documente le processus final de génération de données à partir d'*ObjectFolder 2.0*, ainsi que les étapes de nettoyage et d'augmentation de données avant de débuter la modélisation.

Ensuite, la création de modèles de détection d'objets utilisant des données multisensorielles est documenté à la section 3.4.2.

Finalement, l'évaluation des modèles est effectuée au chapitre 4. Bien que le résultat de cet essai ne mènera pas directement à un déploiement dans le monde réel, les applications possibles des modèles et techniques développées seront abordées dans le chapitre 5.

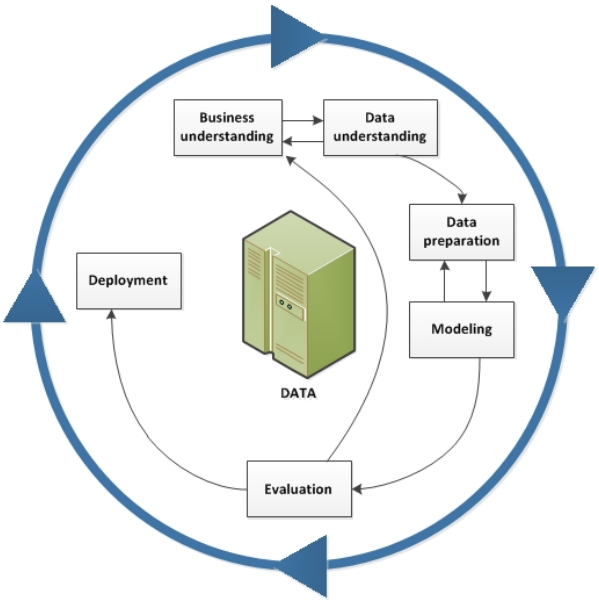

FIGURE 3.3 – Diagramme de développement selon CRISP-DM, tel que présenté sur le site d'IBM [53].

3.2 Configuration Matérielle

Ce projet a été effectué sur une machine comportant la configuration matérielle suivante :

- **Système d'exploitation :** Windows 10 Education 64-Bit ;
- **Processeur :** AMD Ryzen 7 5800X (8 cœurs, 3.8GHz) ;
- **Mémoire vive :** 32GB ;
- **Carte Graphique :** NVIDIA GeForce RTX 3080 ;
- **Mémoire Graphique :** 10GB.

La carte graphique est utilisée lors de l'exécution d'*ObjectFolder 2.0* à la section 3.3 ainsi que pour l'entraînement de modèles à la section 3.4.

3.3 Utilisation d'ObjectFolder 2.0

Le code source ainsi que les instructions d'utilisation pour *ObjectFolder 2.0* sont disponible sur GitHub [54], dont l'annexe B montre un exemple de son téléchargement. Il y est indiqué que le projet doit être exécuté sur une installation Ubuntu (Linux), mais il est possible de faire fonctionner le tout sur Windows avec quelques modifications.

Pour permettre l'utilisation du code source d'*ObjectFolder 2.0* sur une machine Windows 10, les changements apportés au projet sont les suivants :

- `environment.yml` : Le fichier indiquant les paquets Python/Anaconda à utiliser lors de l'exécution du code source. La combinaison de versions sélectionnée par les auteurs produit des erreurs sur une machine Windows, nécessitant d'expérimentater avec des versions alternatives et plus récentes.
  - `pytorch=1.8.1 -> pytorch=1.12.1` : Un problème au niveau de *pytorch* entre la version *1.8.1* et Windows empêche le bon fonctionnement du programme. La version 1.12.1 fonctionne sans problème.
  - `cudatoolkit=11.1.1 -> cudatoolkit=11.6` : Permet un meilleur fonctionnement de la carte graphique *Nvidia GeForce RTX 3080* avec *pytorch*, mais n'importe quelle version suivant et incluant 11.1 devrait fonctionner.
  - `torchvision, torchaudio -> 0.13.1, 0.12.1` : Important de spécifier les numéros de version pour assurer la compatibilité avec le changement vers `pytorch=1.12.1`.
  - `matplotlib -> matplotlib=3.6.3` : Important de spécifier une version compatible pour permettre à ObjectFolder 2.0 de générer des graphiques correctement.
- `TouchNet_utils.py` : Le changement de version de *pytorch* de *1.8.1* à *1.12.1* requiert la modification de certaines directives `import` suite à un changement d'interface à l'intérieur de ce paquet, tel que démontré dans la figure 3.4.
- `OF_render.py` : Ce fichier sert de point d'entrée pour l'exécution d'*ObjectFolder 2.0*. Des changements ont été apportés pour permettre d'exécuter le code à partir d'un nouveau script Python en encapsulant son contenu dans une méthode accessible depuis un fichier externe, tel que défini à la figure 3.5.
- L'ajout de `main.py` : Ce fichier sert de point de départ pour exécuter *ObjectFolder 2.0* à partir d'un script Python sans avoir à modifier `OF_render.py` davantage. Les arguments qui auraient été utilisés par invite de commande sont envoyés directement à `OF_render`, maintenant chargé sous la forme d'un module Python. Un exemple de ce script est visible à la figure 3.6.
- `VisionNet_utils.py` : Les images générées par la modalité Vision d'*ObjectFolder* sont créées sous le format `000.png, 001.png, 002.png...`. Or, les modalités Audio et Toucher sont sauvegardées sous les formats `1.wav, 2.wav, 3.wav...` et `1.png, 2.png, 3.png...`, ce qui ajoute une difficulté future quant à la récu-

pération et combinaison des fichiers afin d'entraîner un modèle. Une ligne de code est donc changée à la figure 3.7 pour assurer la constance entre les trois modalités en générant les noms d'images sous le format `1.png, 2.png, 3.png...`.

```
# ObjectFolder 2.0
from torch._six import container_abcs, string_classes, int_classes

# Modification
from torch._six import string_classes
import collections.abc as container_abcs
int_classes = int
```

FIGURE 3.4 – Modification apportée à `TouchNet_utils.py` suivant le changement de version de *pytorch*.

3.3.1 Détails techniques de l'utilisation d'ObjectFolder

Le modèle d'*ObjectFolder 2.0* implémente des réseaux de neurones qui prennent en entrée, tous en commun, un point dénoté par un vecteur 3D v_x, v_y, v_z. Les formats des fichiers en entrée sont des matrices *NumPy* avec les formes suivantes :

- Modalité visuelle : Une matrice 2D de taille $N \times 6$, où N est le nombre de points à tester (`vision_test_file`). Les 6 colonnes sont les coordonnées c_x, c_y, c_z de la caméra, suivi de la position de la lumière l_x, l_y, l_z.
  - Une fonction dans le code d'*ObjectFolder* s'occupe de calculer la direction de la camera selon sa position par rapport à l'objet, montrée en exemple à la figure 3.8. Le code devra être modifié pour avoir davantage de contrôle sur la caméra en permettant de définir manuellement la direction désirée.
- Modalité tactile : Deux matrices 2D de tailles $N \times 3$ chaque. La première (`touch_vertices_file`) contient les points à tester v_x, v_y, v_z et la deuxième (`touch_gelinfo_file`) contient des informations sur le capteur *GelSight* simulé, soit deux angles de rotation sur le capteur ainsi que sa profondeur g_θ, g_ϕ, g_d.
  - Il est à noter que dans les exemples offerts par les auteurs d'*ObjectFolder*, g_θ et g_ϕ sont toujours défini à 0.
- Modalité audio : Similaire à la modalité tactile, deux matrices 2D de tailles $N \times 3$ chaque. La première (`audio_vertices_file`) contient les points à tester v_x, v_y, v_z

```
# Remplacer le __main__ avec une méthode
#if __name__ =='__main__':
def run_OF_render(argarray=None):
    parser = config_parser()
    # S'il n'y a pas argarray, exécuter la ligne originale (donc
    ↪  arguments provenant de l'invite de commande)
    # Sinon, on utilise les arguments en entrée
    #args = parser.parse_args()
    if argarray==None:
        args = parser.parse_args()
    else:
        args = parser.parse_args(argarray)
    modalities = args.modality.strip().split(",")
    ...

# Si le fichier est exécuté directement comme script principal, on
↪  exécute le code original
if __name__ == '__main__':
    run_OF_render()
```

FIGURE 3.5 – Modifications apportées à `OF_render.py` afin de permettre au code d'être exécuté à partir d'un fichier externe. Le contenu du script original est encapsulé dans la méthode `run_OF_render()` avec l'option `argarray` permettant de fournir directement au programme les arguments tels qu'ils proviendraient depuis un invite de commandes. Il est toujours possible d'exécuter le script original comme s'il n'y avait eu aucun changement.

```
import importlib.util
import re


def module_from_file(module_name, file_path):
    spec = importlib.util.spec_from_file_location(module_name,
    ↪  file_path)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
    return module


args = """--modality vision,audio,touch --object_file_path
↪  demo/ObjectFile.pth
--vision_test_file_path demo/vision_demo.npy
--vision_results_dir demo/vision_results/
--audio_vertices_file_path demo/audio_demo_vertices.npy
--audio_forces_file_path demo/audio_demo_forces.npy
--audio_results_dir demo/audio_results/
--touch_vertices_file_path demo/touch_demo_vertices.npy
--touch_gelinfo_file_path demo/touch_demo_gelinfo.npy
--touch_results_dir demo/touch_results/"""

ofr = module_from_file("OF_Render", "OF_Render.py")

ofr.run_OF_render(re.split("\\s+", args))
```

FIGURE 3.6 – Le script du nouveau fichier `main.py` utilisé pour exécuter à partir d'un fichier externe l'exemple offert dans les instructions d'utilisation d'*ObjectFolder 2.0*. Le fichier `OF_render.py` est chargé comme module, important la méthode `run_OF_render` créée à la figure 3.5. Dans cet exemple initial, les arguments sont écrits directement comme texte dans une variable avant d'être envoyés à *ObjectFolder 2.0*, mais il y a maintenant une façon simple de manipuler ce texte à partir de Python.

```
# Avant (000.png, 001.png, 002.png...)
filename = os.path.join(savedir, '{:03d}.png'.format(i))

# Après (1.png, 2.png, 3.png...)
filename = os.path.join(savedir, f'{i+1}.png')

# Référence Audio (dans OF_render.py -> AudioNet_eval)
output_path = os.path.join(testsavedir, str(i+1) + '.wav')

# Référence Toucher (dans OF_render.py -> TouchNet_eval)
filename = os.path.join(testsavedir, '{}.png'.format(i+1))
```

FIGURE 3.7 – Le changement apporté à `VisionNet_utils.py` accompagné des lignes équivalentes trouvées dans `OF_render.py`. Bien que le code entre les modalités Audio et Toucher soit défini de deux façons différentes, le résultat est un nom de fichier numérique commençant à 1 avec aucun zéro en préfixe pour ces deux modalités. La modalité Vision génère des noms de fichier numériques commençant à 0 avec un remplissage de zéros jusqu'au minimum de trois caractères. Le code est modifié pour rendre les trois modalités constantes.

et la deuxième (`audio_forces_file`) contient la direction de la force appliquée à l'objet pour créer le son f_x, f_y, f_z.

— Dans les exemples offerts, f_x, f_y, f_z sont tous les trois toujours défini à 1 peu importe le point, il est donc possible qu'il ne soit pas nécessaire de modifier ces valeurs.

— Sinon, le vecteur de force pourrait être calculé selon le vecteur normal sur la surface de l'objet pour le point choisi. Ceci simulerait la direction avec laquelle le robot frappe sur l'objet.

3.3.2 Génération d'images à partir de données de points

Puisque l'un des objectifs de ce projet est d'utiliser une série de points visuellement saillants sur la surface d'un objet afin d'améliorer un modèle de reconnaissance d'objets, il est nécessaire de développer une technique permettant d'obtenir un rendu de l'objet regardant directement sur un point arbitraire.

En choisissant donc l'un des points définissant la surface d'un objet 3D, nous désirons en dériver une position de caméra offre une vue de ce point. Ceci peut être accompli en calculant le vecteur normal au point donné et en déplaçant la caméra le long de ce

```
def coordinates_to_c2w(x, y, z, r=2.5):
    theta = np.arccos(z / r)
    phi = np.arctan2(x, -y)
    Rx = ...theta...
    Rz = ...phi...
    R = Rz @ Rx
    c2w = R, x, y, z...
    return c2w
```

FIGURE 3.8 – Version simplifiée à titre d'exemple de la fonction Python calculant les angles de la caméra employée par la modalité Vision d'*ObjectFolder 2.0*. L'angle *theta* oriente la caméra vers le haut ou le bas selon la hauteur z, supposant que le point se trouve sur une sphère de rayon 2.5. L'angle *phi* pointe la caméra vers le centre de l'objet selon sa position sur le plan horizontal x, y. Cette fonction sera la cible d'une modification où les angles *theta* et *phi* sont calculés à partir d'un deuxième vecteur représentant la direction désirée.

vecteur selon la largeur de l'objet. Finalement, l'orientation de la caméra peut être fixée avec la négation de ce vecteur normal (Voir la figure 3.9).

Pour obtenir le vecteur normal à un point sur l'objet, les surfaces composant l'objet sont parcourues pour obtenir toutes les normales aux surfaces adjacentes à ce point. Ces normales sont additionnées et le résultat est normalisé (Voir la figure 3.10).

En pratique, puisque la méthode requiert de parcourir toutes les surfaces de l'objet, il est préférable de calculer la normale de chaque point composant l'objet d'un coup.

Suivant un objet dans le format 3D « *Wavefront* », on peut obtenir pour chaque triangle composant l'objet les trois points dans un ordre précis, nous permettant d'appliquer un produit croisé sur les trois points pour obtenir la normale. Le résultat est additionné à un accumulateur attaché à chaque point et le processus est répété sur toutes les faces de l'objet. Finalement, tous les accumulateurs sont normalisés, le résultat étant un mappage entre un index dans la liste des points son vecteur normal (Voir le code à la figure 3.11).

Un objet 3D défini dans le format *Wavefront* emploie normalement un vecteur vers le haut sur l'axe Y (figure 3.12). Toutefois, le programme *Blender* est conçu pour utiliser des fichiers dont le vecteur vers le haut est sur l'axe Z, avec un « sol » représenté par le plan composé des axes X et Y. Ce détail nécessite donc d'appliquer une rotation aux objets lors de l'importation afin d'avoir la bonne orientation de l'objet sur la scène (voir la figure 3.13). On peut confirmer la nécessité de cette étape en exécutant la modalité

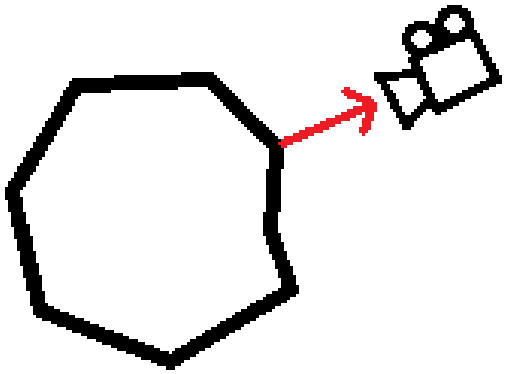

FIGURE 3.9 – Exemple en deux dimensions pour la sélection d'un point et d'une orientation pour la caméra.

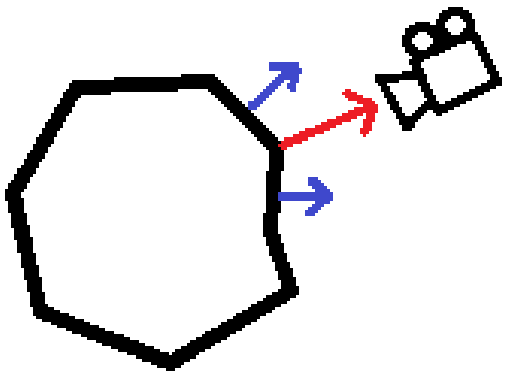

FIGURE 3.10 – Les vecteurs normaux des surface adjacentes au point sélectionné sont combinés pour obtenir le vecteur normal du point.

```python
def calculate_vertex_normals(faces, vertices):
    norms = {}
    for face in faces:
        v = [vertices[face[i]] for i in range(3)]
        norm = np.cross(v[1]-v[0], v[2]-v[0])
        for i in range(3):
            norms.setdefault(face[i], np.array([0., 0., 0.]))
            norms[face[i]] += norm
    # Normalize all vectors before returning
    for key, val in norms.items():
        norms[key] = val / np.linalg.norm(val)
    return norms
```

FIGURE 3.11 – Fonction en code *Python* prenant en entrée toutes les surfaces et les points d'un objet et retournant la somme normalisée de tous les vecteurs normaux des surfaces adjacentes à ces points.

visuelle d'*ObjectFolder 2.0* sur un des objets dans une orientation approximative à la figure 3.13, obtenant une image similaire à la figure 3.14.

Les fichiers 3D téléchargeables à partir de la base de données d'*ObjectFolder* semblent être créés de deux façons différentes. Des commentaires dans certains fichiers indiquent qu'ils ont été créés à l'aide de « *Katamari OBJ encoder* ». D'autres fichiers indiquent avoir été créés et exportés à partir de *Blender* (figure 3.16). Les deux sources de fichiers emploient différents vecteurs vers le haut, où ceux ayant été créés avec *Blender* nécessitent une rotation de 90° avant d'être importés tandis que ceux créés avec *Katamari* ne requièrent aucune rotation (figure 3.15). Les fichiers doivent être inspectés manuellement afin de déterminer la direction des objets avant de continuer.

Par rapport aux objets créés avec *Katamari*, aucune information a été trouvée quant à l'origine de l'outil. Toutefois, les modèles utilisant cet outil ont été retracés aux ensembles de données *YCB*, où des nuages de points à partir d'un scanneur 3D ont été utilisés lors de leur création [55, section 2.2]. Malheureusement, les auteurs de cet ensemble de données ne spécifient pas la méthode exacte pour effectuer cette conversion. De plus, les objets provenant de l'ensemble *GSO* montre les mêmes caractéristiques car les auteurs de cet ensemble utilisent le même scanneur [47, fig. 2(b)], où les détails de la conversion ne semblent pas expliqués.

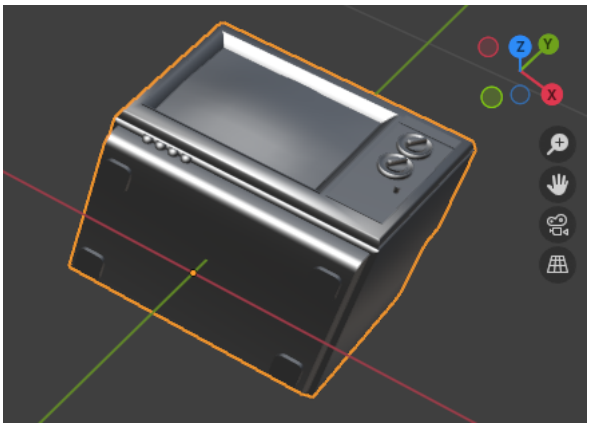

FIGURE 3.12 – Objet #4 (Télévision) importé dans *Blender* sans rotation, le dessus de la télévision pointe vers l'axe Y.

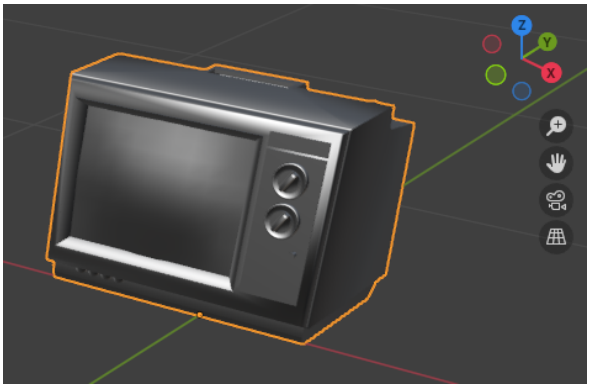

FIGURE 3.13 – Objet #4 (Télévision) importé dans *Blender* avec rotation de 90 deg, le dessus de la télévision pointe maintenant vers l'axe Z.

FIGURE 3.14 – Objet #4 (Télévision) rendu par *ObjectFolder 2.0* avec la caméra placée dans une position $-Y$ et $+X$, l'angle correspond approximativement à la figure 3.13.

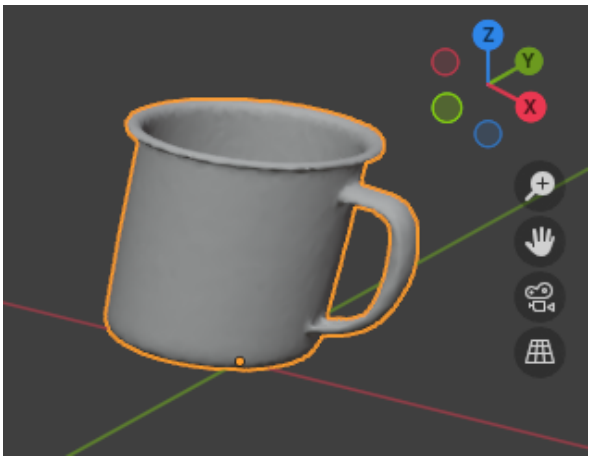

FIGURE 3.15 – Objet #23 (Tasse rouge) importé dans *Blender* sans rotation, le dessus de la tasse pointe vers l'axe Z car l'objet est enregistré dans l'orientation normale pour *Blender*. Ceci est techniquement incorrect, puisque l'objet est dans un format *Wavefront*.

```
# Blender v2.92.0 OBJ File: 'Television_01.blend'
# www.blender.org
mtllib model.mtl
o Television_01_Cube.021
v -0.286415 0.100000 0.154288
v -0.295746 0.497592 0.167872
...
```

FIGURE 3.16 – Objet #4 (Télévision), l'entête du fichier Wavefront montre que cet objet a été créé dans *Blender*.

```
# Generated by Katamari OBJ encoder.
# Vertices: 8188
# Faces: 16384
mtllib model.mtl
v -0.061331 0.017990 0.021480
v -0.061423 0.017642 0.018064
...
```

FIGURE 3.17 – Objet #23 (Tasse rouge), l'entête du fichier Wavefront montre que cet objet a été créé à l'aide de *Katamari OBJ encoder*.

Création d'images avec Py-Blender en tant que référence

Pour confirmer le fonctionnement de la méthode, nous employons le package Py-Blender (*bpy*) [56], permettant d'utiliser toutes les fonctionnalités de l'outil Blender à partir d'un environnement Python. Nous pouvons alors développer un programme pour générer des images à partir des points et des vecteurs calculés à l'étape précédente. Les images obtenues servent alors de points de repère à être répliqués en utilisant la modalité visuelle d'*ObjectFolder*.

Le rendu d'un objet à l'aide de *bpy* suit alors les étapes suivantes :

- La coordonnée du point sur l'objet est extrait (v_x, v_y, v_z), ainsi que la normale associée à ce point (n_x, n_y, n_z) ;
  - Un rotation est appliquée sur l'axe X à v et n selon l'objet, (voir le paragraphe portant sur la confusion des axes à la section 3.3.2).
- Un indicateur de la taille de l'objet, soit la diagonale de la « *bounding box* », est extrait (s) ;
- La position de la caméra est calculé comme le déplacement de la coordonnée du point le long de la normale, ajusté selon la taille de l'objet (ex. : $c_x = v_x + (s * n_x)$) ;
- La rotation de la caméra est calculé selon la formule employée par *ObjectFolder* ($C_{yaw} = \arctan 2(n_x, -n_y), C_{pitch} = \arccos(n_z)$) ;
- L'objet est importé dans la scène et des lumières sont créées autour de l'objet pour l'illuminer afin d'avoir une luminosité ambiante ;
- Une lumière additionnelle est créée au point de la caméra ;
- La procédure de rendu est lancée et répétée pour chaque point d'intérêt sur l'objet.

En appliquant cette méthode sur un sous-ensemble de 16 points visuellement saillants sur 20 extraits à partir du travail de Ghazal Rouhafzay, nous obtenons un résultat

tel que visible à la figure D.1. En apportant les bonnes modifications au code source d'*ObjectFolder 2.0*, nous devrions pouvoir prendre le contrôle de sa modalité visuelle afin de répliquer le point de vue de ces images *Blender* avec les images d'objets simulées en sortie.

Modifications apportées à ObjectFolder 2.0

Les points extrait à l'étape 3.3.2 sont utilisés avec la modalité visuelle d'*ObjectFolder* pour obtenir les images équivalentes. Tout d'abord, on observe les exemples offerts par les créateur d'*ObjectFolder* sur la façon de définir les coordonnées.

En examinant de plus près les coordonnées visuelles attendues par *ObjectFolder*, nous observons qu'une certaine normalisation a été appliquée aux objets. Notamment, dans tous les exemples offerts par les auteurs, les points se trouvent toujours sur une sphère de rayon 2.5, ou en termes mathématiques, $\sqrt{{c_x}^2 + {c_y}^2 + {c_z}^2} = 2.5$. Lorsque le programme est exécuté, la largeur apparente de l'objet dans l'image produite est ajustée par cette normalisation. Par exemple, la tasse rouge (Objet #23) et le baril (Objet #10) sont des modèles 3D de largeurs très différentes (fig. 3.18), mais prennent un espace d'écran comparable dans *ObjectFolder* pour la même coordonnée de caméra (figures 3.19 et 3.20).

Afin de pouvoir placer la caméra à un point arbitraire pour répliquer les images obtenues dans *Blender*, il est nécessaire de déterminer la normalisation exacte appliquée à l'objet. Pour la modalité visuelle, aucun indicateur dans le code d'*ObjectFolder* n'est présent. Toutefois, les modalités audio et tactiles requièrent en entrée des coordonnées qui sont directement attribuables à des points sur les modèles 3D des objets et qui sont ensuite normalisés par *ObjectFolder* avant d'être envoyées aux modèles respectifs. Les détails et problèmes liés à la normalisation appliquée sur la modalité visuelle sont explorés à la section 3.3.2.

Malgré les problèmes identifiés, supposant une bonne rétro-ingénierie de la normalisation appliquée à chaque objet, les modifications apportées à *ObjectFolder* nous permettent d'approximer de près les résultats obtenus à la section 3.3.2 en comparant les figures D.1 et D.2. Après avoir apporté les modifications décrites à la section 3.3, ceci est accompli en apportant les modifications supplémentaires suivantes :

- Tout d'abord, nous prenons le contrôle de la logique qui effectue la conversion entre un point de caméra. Nous ajoutons un paramètre de ligne de commande `vision_is_poses` (fig. 3.21).

FIGURE 3.18 – Les objets #10 (Baril bleu) et #23 (Tasse rouge) importés dans *Blender* ont une taille très différente.

FIGURE 3.19 – Démo de la modalité visuelle sur l'objet #10 (Baril bleu). La coordonnée employée pour la caméra est $1.76776695, 0., 1.76776695$.

FIGURE 3.20 – Démo de la modalité visuelle sur l'objet #23 (Tasse rouge). La coordonnée employée pour la caméra est $1.76776695, 0., 1.76776695$, même que pour la figure 3.19. Nous observons que l'objet apparait plus large malgré la différence de taille notée à la figure 3.18

– Ensuite, nous utilisons ce nouveau paramètre pour substituer nos propres données de caméra, au lieu de celles calculées par *ObjectFolder* (fig. 3.22).
– Nous créons une copie de la fonction `coordinates_to_c2w` (fig. 3.8) nommée `coordinates_to_c2w_direct` (fig. 3.23), où nous pouvons retirer le code traitant la coordonnée de la caméra comme vecteur normal et introduire le nôtre afin de permettre un contrôle direct des angles de la caméra.
– Nous exécutons la modalité visuelle sur les points et leurs normales associées suivant une procédure similaire à la section 3.3.2, avec comme différence une étape de normalisation appliquée au point de la caméra calculé avant l'envoi à la fonction `c2w` modifiée. Les facteurs de normalisation et les translations à appliqué doivent être déterminées par essai et erreur pour chaque objet, tel que décrit à la section 3.3.2.

```
# VisionNet options
parser.add_argument("--vision_test_file_path",
 ↪  default='data/vision_demo.npy', help='The path of the testing
 ↪  file for vision, which should be a npy file.')
parser.add_argument("--vision_results_dir", type=str,
 ↪  default='./results/vision/', help='The path of the vision
 ↪  results directory to save rendered images.')
parser.add_argument("--vision_is_poses", action="store_true") #
 ↪  Nouveau

...

def VisionNet_eval(args):
    ...
    # Nous donnons l'argument vision_is_poses à load_osf_data
    poses, hwf, i_split, metadata =
     ↪  load_osf_data(args.vision_test_file_path,
     ↪  args.vision_is_poses)
```

FIGURE 3.21 – Modifications apportées à `OF_render.py` afin de permettre la prise de contrôle de la modalité visuelle. Si l'argument `--vision_is_poses` est activé, le programme prendra alors en entrée une matrice de taille $N \times 19$, où les 16 premiers éléments sont le résultat de la fonction `coordinates_to_c2w` (fig. 3.8).

```
def load_osf_data(test_file_path, vision_is_poses=False):

all_poses = []
    all_metadata = []
    counts = [0]
    test_file = np.load(test_file_path)
    N = test_file.shape[0]
    for i in range(N):
        if vision_is_poses:
            # Si nous sommes ici, nous substituons nos propres "poses"
            lx, ly, lz = test_file[i][-3:]
            poses = test_file[i][:-3].reshape((4,4))
            poses = np.array(poses).astype(np.float32)
        else:
            # Original
            cx, cy, cz, lx, ly, lz = test_file[i]
            poses = coordinates_to_c2w(cx, cy, cz)
        metadata = np.array([[lx, ly, lz]]).astype(np.float32)
        all_poses.append(poses)
        all_metadata.append(metadata)

    poses = np.array(all_poses).astype(np.float32)

    metadata = np.concatenate(all_metadata, 0)
    counts.append(N)
    i_split = [np.arange(counts[0], counts[1])]

    H, W, focal = 256, 256, 355.5555419921875

    return poses, [H, W, focal], i_split, metadata
```

FIGURE 3.22 – Modifications apportées à `load_osf.py` afin de permettre la prise de contrôle de la modalité visuelle. Suite de la figure 3.21.

```
def coords_to_c2w_direct(x, y, z, nx, ny, nz):
    # Ensure normalized vector
    nx2, ny2, nz2 = ([nx, ny, nz] / np.linalg.norm([nx, ny, nz]))
    theta = np.arccos(nz2)
    phi = np.arctan2(nx2, -ny2)
    r_x = ...theta...
    r_z = ...phi...
    r_zx = r_z @ r_x
    c2w = r_zx, x, y, z
    return c2w
```

FIGURE 3.23 – Version simplifiée de la fonction Python modifiée calculant les angles de la caméra employée par la modalité Vision d'*ObjectFolder 2.0* (Fonction originale à la figure 3.8). Au lieu de traiter x, y, z comme un vecteur normal pour déterminer l'angle de la caméra, nous utilisons tout simplement notre propre vecteur normal nx, ny, nz.

Problèmes avec la modalité visuelle d'ObjectFolder 2.0

Rétro-ingénierie de la normalisation sur les objets En étudiant les détails de l'implémentation de *VisionNet* dans *ObjectFolder 1.0* [10], spécifiquement à la section 3.2, il est indiqué que chaque objet est normalisé pour être confiné à l'intérieur du cube unité de *Blender*. En théorie, ceci indique que chaque objet devrait être réduit uniformément pour que son axe le plus long ait un minimum de -0.5 et un maximum de 0.5, pour une longueur totale le long de cet axe de 1.0. De plus, l'objet serait déplacé afin de s'assurer que son point central se trouve à l'origine de ce cube unité.

Suivant ceci, nous pouvons appliquer cette normalisation à un point de caméra calculé précédemment afin d'obtenir des images qui s'approchent de *Blender*. Par contre, nous observons que les objets présentent une translation qui semble arbitraire dans l'espace. Par exemple, le baril bleu (objet #10) n'est pas centré sur le milieu de l'objet et en plaçant la caméra au « sol », soit un point avec la coordonnée Z à 0.0, l'objet apparaît plus bas. En déplaçant manuellement la caméra de haut en bas, nous identifions les translations sur les différents axes, où l'on estime les coordonnées où la caméra est alignée avec les différents côtés de l'objet (Tableau 3.1).

La différence entre le dessus et le dessous du baril bleu est d'environs $0.3325 - (-0.675) = 1.0075$, confirmant que les dimensions de l'objet sont normalisées sur sa hauteur, son axe le plus long. Une translation de $\frac{0.3325+-0.675}{2} = -0.17125$ dans l'espace de coordonnées normalisé est donc perceptible sur l'axe Z. Nous n'observons aucune

translation sur l'axe X (Côté gauche et droit) ou sur l'axe Y (Vue de côté), où le déplacement de la caméra est symétrique avec une coordonnée de ± 0.2759.

Nous déterminons alors que le facteur de normalisation pour un point donné est l'inverse de la différence entre le point minimal et maximal sur l'axe le plus long du modèle 3D d'origine. Ici, pour le baril, cette différence est de 0.88 sur sa hauteur (axe Z), le facteur de normalisation étant alors $\frac{1}{0.88} = 1.1\overline{36}$. La translation exacte à appliquer reste à être déterminée.

De retour avec l'exemple de la tasse rouge, nous appliquons la même technique qu'avec le baril bleu afin d'identifier les translations au tableau 3.2. L'axe le plus long de la tasse est à l'horizontal, aligné avec l'anse, soit l'axe Y. En regardant l'objet depuis l'axe X, on arrive à identifier le minimum à -0.435 et le maximum à 0.565, pour une différence de $0.565 - (-0.435) = 1$, concordant toujours avec les explications offertes dans [10]. Toutefois, on observe encore un décalage d'origine inconnue de $\frac{0.565+-0.435}{2} = 0.065$ sur cet axe Y, de plus que des décalages de $\frac{0.3775+-0.3175}{2} = 0.03$ sur l'axe Z et de $\frac{0.38805+-0.40805}{2} = -0.01$ sur l'axe X.

Ces translations semblant arbitraires appliquées aux objets lors de leur normalisation reste alors le problème principal quant à l'utilisation de la modalité visuelle d'*ObjectFolder*. Nous remarquons toutefois un point commun intéressant entre les objets, soit que le décalage est biaisé en direction de la partie de l'objet montrant une plus grande complexité géométrique. Pour le baril bleu, sur son axe Z, la moitié du haut de l'objet possède des parties concaves ainsi qu'un rebord détaillé. Pour la tasse rouge, sur l'axe Y, la présence de l'anse ajoute un niveau de complexité dans la direction positive de l'axe.

Nous revenons alors à l'hypothèse initiale de cette section, où nous supposons que l'objet devrait être contenu dans un cube où les points minimum et maximum sont entre -0.5 et 0.5. Afin de centrer l'objet, nous pouvons calculer sa « *Bounding Box* » (ou *Boîte Englobante*) en calculant les minimums et maximums des points pour chaque axe X, Y, Z. Le centre est donc la moyenne de ces valeurs, par exemple, $x_{cen\_bb} = \frac{x_{max}+x_{min}}{2}$.

Nous repensons alors à quelle autre façon quelqu'un utiliserait pour identifier le point *centroïde* d'un objet. Une telle alternative serait de prendre la moyenne de tous les points composant l'objet pour chaque axe, par exemple, $x_{cen} = \frac{1}{num_x} \sum_1^{num_x} x_i$. Ce point centroïde présenterait alors un biais vers les parties plus détaillées de l'objet, concordant à nos observations. Pour le baril bleu, nous calculons ce point à $x_{cen} \approx 1.04265 \times 10^{-5}, y_{cen} \approx 6.31649 \times 10^{-6}, z_{cen} \approx 0.59691$. Les axes X et Y étant ignorés

TABLE 3.1 – Estimation de la normalisation sur le baril bleu (Objet #10), lignes rouges ajoutées pour indiquer le centre de l'image.

Caméra (X, Y, Z)	Endroit	Image
0.0, 1.25, 0.0	Centré	
0.0, 1.25, 0.3325	Au dessus	
0.0, 1.25, -0.675	En dessous	
0.2759, 1.25, 0.0	Côté gauche	
-0.2759, 1.25, 0.0	Côté droit	
1.25, 0.0, 0.0	Centré (Vue de côté)	
1.25, -0.2759, 0.0	Côté gauche (Vue de côté)	
1.25, 0.2759, 0.0	Côté droit (Vue de côté)	

TABLE 3.2 – Estimation de la normalisation sur la tasse rouge (Objet #23), lignes rouges ajoutées pour indiquer le centre de l'image.

Caméra (X, Y, Z)	Endroit	Image
0.0 1.25 0.0	Centré	
0.0 1.25 0.3775	Au dessus	
0.0 1.25 -0.3175	En dessous	
0.38805 1.25 0.0	Côté gauche	
-0.40805 1.25 0.0	Côté droit	
1.25 0.0 0.0	Centré (Vue de côté)	
1.25 -0.435 0.0	Côté gauche (Vue de côté)	
1.25 0.565 0.0	Côté droit (Vue de côté)	

pour le moment, nous appliquons le facteur de normalisation identifié pour le baril bleu sur l'axe Z, nous donnant $z_{cen\_norm} \approx 0.59691 \times 1.1\overline{36} \approx 0.67831$.

Continuant cet exercice, les points minimums et maximums pour l'axe Z sur le baril sont $z_{min} = 0.004637, z_{max} = 0.884637$. En appliquant la normalisation, ces points sont $z_{min\_norm} \approx 0.0052693, z_{max\_norm} \approx 1.0052693$. Finalement, en appliquant le décalage, nous avons $z_{min\_norm\_dec} \approx 0.0052693 - 0.67831 \approx -0.673, z_{max\_norm\_dec} \approx 1.0052693 - 0.67831 \approx 0.327$. Ces valeurs sont très près des points identifiés dans le tableau 3.1 (-0.675 et 0.3325), et prenant en compte les erreurs de l'approche empirique, nous estimons que cette méthode est celle qui a été utilisée par les auteurs d'*ObjectFolder*.

En utilisant cette technique de normalisation et de calcul de centroïde, nous obtenons la série d'images à la figure D.2, étant directement comparable à la figure D.1. Suivant ceci, nous sommes donc en mesure de recréer un point de vue de caméra arbitraire équivalent entre *Blender* et *ObjectFolder 2.0*.

Confusion entre les axes X et Y pour certains objets Lors des tests effectués sur la conversion des coordonnées sur les objets vers un format utilisable avec la modalité Vision d'*ObjectFolder*, une erreur d'interprétation de la part des auteurs quant à l'orientation de certains objets lors du processus d'entraînement de leurs modèles semble s'être introduit. Ceci découle d'une confusion entre les objets créés dans *Blender* et ceux créés dans *Katamari*.

Les objets créés dans *Blender* respectent une orientation avec le vecteur avant à $-Z$ et un vecteur vers le haut à Y. En prenant un point sur l'objet et en appliquant la matrice de rotation à l'équation 3.1, une rotation de $+90\,\text{deg}$ autour de l'axe X est effectué et nous obtenons un point utilisant un vecteur avant Y et un vecteur vers le haut Z. Ceci nous permet de créer des images équivalentes entre *Blender* et *ObjectFolder* (figures 3.13 et 3.14).

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \tag{3.1}$$

Les objets créés avec *Katamari* sont toutefois déjà enregistrés sous le format $Avant = Y, Haut = Z$ et peuvent être importés dans *Blender* sans appliquer de rotation de correction (figure 3.24). Nous supposons alors qu'*ObjectFolder* respecte cette orientation, mais lors de l'exploration de son fonctionnement, nous remarquons que ces fichiers ont

été utilisés pour entraîner leur modèles avec une orientation $Avant = X, Haut = Z$ (tableau 3.2, première rangée, l'anse de la tasse pointe vers l'axe Y). Il est donc nécessaire d'appliquer la matrice à l'équation 3.2 pour effectuer une rotation de +90 deg autour de l'axe Z avant d'utiliser ces points avec *ObjectFolder* (figure 3.25).

$$\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

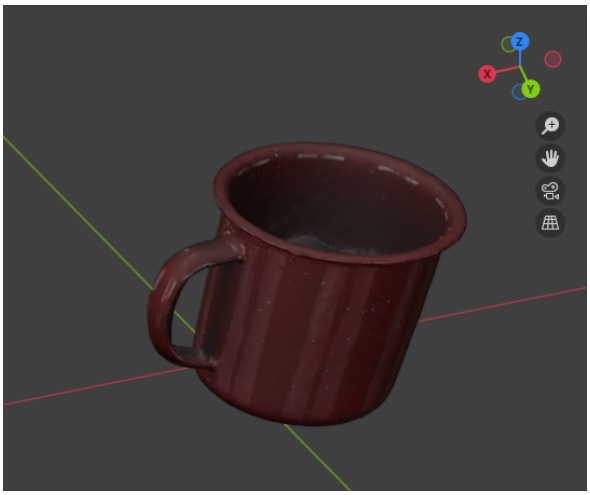

FIGURE 3.24 – Objet #23 (Tasse rouge) importé dans *Blender* sans rotation, point de vue alternatif à la figure 3.15.

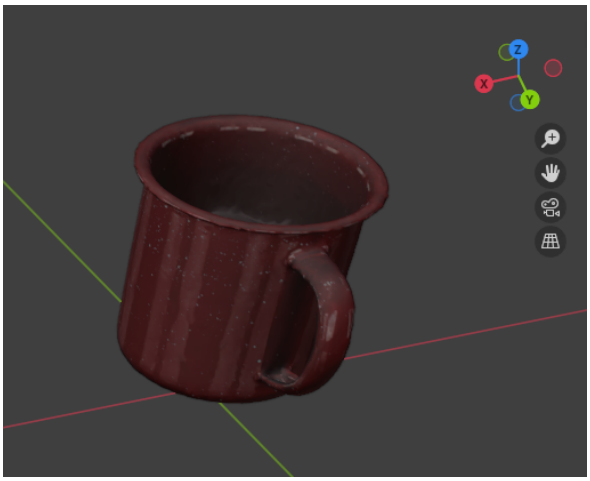

FIGURE 3.25 – Objet #23 (Tasse rouge) importé dans *Blender* avec rotation de 90 deg autour de l'axe Z, modifiant le vecteur avant de l'axe Y vers l'axe X.

Limites du niveau de détail sur la surface des objets La technique *KiloOSF* employée par *ObjectFolder 2.0*, tel que défini à la section 2.4.2, permet d'obtenir un rendu plus rapide de la modalité visuelle des objets en les séparant la tâche sur des milliers de *MLP*s. Toutefois, cette technique semble introduire un problème au niveau de la qualité des images générées lorsque l'utilisateur dévie de la distance optimale de l'objet, soit un point situé sur une sphère d'un rayon de 2.5 unités autour de l'objet.

En approchant la caméra de l'objet, il devient apparent que la résolution des *MLP*s ne change pas. Les images générées présentent alors une résolution floue et des artéfacts apparaissent dans l'image, où l'on perçoit une « grille » de séparation entre les *MLP*s

ayant contribué à l'image. On observe ce phénomène en comparant la figure 3.26 et la figure 3.27.

FIGURE 3.26 – Baril bleu (Object #10) vue de face. Caméra positionnée à 0.0, 1.25, 0.0 (X, Y, Z).

FIGURE 3.27 – Baril bleu (Object #10) vue rapprochée. Caméra positionnée à 0.0, 0.5, 0.2 (X, Y, Z).

Puisque l'un des objectifs de ce projet est d'entraîner des modèles de reconnaissance d'objets à l'aide d'images rapprochées sur des points saillants, ce problème risque d'avoir un impact sur la performance et doit donc être considéré lors de l'utilisation de la modalité visuelle d'*ObjectFolder 2.0*.

Éclairage intégré aux modèles des objets *ObjectFolder 2.0* permet à l'utilisateur de spécifier l'emplacement d'une source de lumière dans la scène. Toutefois, on observe une irrégularité non documentée, soit que le modèle entraîné comporte une illumination intrinsèque provenant d'un angle inconnu ne pouvant pas être modifiée.

La source lumineuse pouvant être contrôlée par l'utilisateur est capable d'illuminer les sections ombragées de l'objet, tel que dans le tableau 3.3, mais sans un contrôle complet de l'illumination de la scène, il existe un risque d'introduire un biais lors de l'entraînement d'un modèle de reconnaissance d'objets si l'on désire utiliser différentes conditions lumineuses lors du processus.

TABLE 3.3 – Interaction de la lumière sur le baril bleu (Objet #10), où on observe un ombrage constant sur l'objet malgré la position de la lumière.

Lumière (X, Y, Z)	Caméra (X, Y, Z)	Image
0.0, -0.707, 0.707	0.0 1.25 0.0	
-0.707, 0.0, 0.707	0.0 1.25 0.0	
0.0, 0.707, 0.707	0.0 1.25 0.0	
0.707, 0.0, 0.707	0.0 1.25 0.0	
0.0, -0.707, 0.707	0.0 -1.25 0.0	
-0.707, 0.0, 0.707	0.0 -1.25 0.0	
0.0, 0.707, 0.707	0.0 -1.25 0.0	
0.707, 0.0, 0.707	0.0 -1.25 0.0	

Problèmes avec la modalité tactile d'ObjectFolder 2.0

Explication du problème Un problème se présente lors de l'utilisation de la modalité tactile d'*ObjectFolder 2.0* où certains objets produisent des profils de toucher avec une orientation différente à celle attendue. En effet, les objets ayant été créés à l'aide de l'objet *Katamari*, tel que discuté à la section 3.3.2, sont enregistrés dans un format où l'objet est dans une orientation différente dont la modalité visuelle présente. Bien que les auteurs d'*ObjectFolder* aient corrigé cette discontinuité pour la composante visuelle, la même correction ne semble pas avoir été appliquée pour la modalité tactile.

Pour cette exploration, nous nous concentrons alors sur deux objets, soit l'objet #11 (une chaise berçante, fig. 3.28) et l'objet #23 (une tasse rouge, fig. 3.29). La chaise est un objet dont le fichier indique avoir été créé dans *Blender*, alors que la tasse a été créée avec l'outil *Katamari*.

En effectuant deux touchers sur la chaise (figures 3.30 et 3.32), *ObjectFolder* nous donne en retour des profils de toucher avec une orientation alignée à ce que l'on observe lorsque l'objet est debout (figures 3.31 et 3.33). Toutefois, en effectuant le même processus sur la tasse (figures 3.34 et 3.36), nous observons une rotation de 90° sur les profils de toucher produits par *ObjectFolder* (figures 3.35 et 3.37). Si nous présentons la tasse dans une orientation alternative, soit en utilisant la même configuration d'import que pour la chaise (fig. 3.38), les résultats de toucher obtenus avec *ObjectFolder* deviennent alors cohérents.

Correction du problème La composante tactile d'*ObjectFolder 2.0* fonctionne à l'aide d'un modèle d'apprentissage profond entraîné pour prédire la profondeur de chaque pixel sur un capteur de largeur 160×120. Le résultat de ce modèle est ensuite passé à l'outil *TAXIM*, qui génère une image *RGB* correspondante simulant l'apparence du résultat d'un capteur de type *GelSight*.

Afin de corriger le problème de l'orientation, deux modifications sont apportées au code d'*ObjectFolder*, soit :

- La matrice des profondeurs en sortie du modèle est coupée à la forme 120×120 afin d'avoir une image parfaitement carrée (fig. 3.40) ;
  - La composante *TAXIM* est modifiée pour fonctionner avec la nouvelle forme de la matrice (figures 3.41 et 3.42).
- Une rotation de 90° dans le sens horaire est appliqué à cette nouvelle matrice si l'utilisateur indique que l'objet doit être corrigé (figures 3.39 et 3.40).

FIGURE 3.28 – Vue en ensemble de l'objet #11 (Chaise berçante) dans Blender. L'objet est importé avec un vecteur avant $-Z$ et un vecteur haut Y.

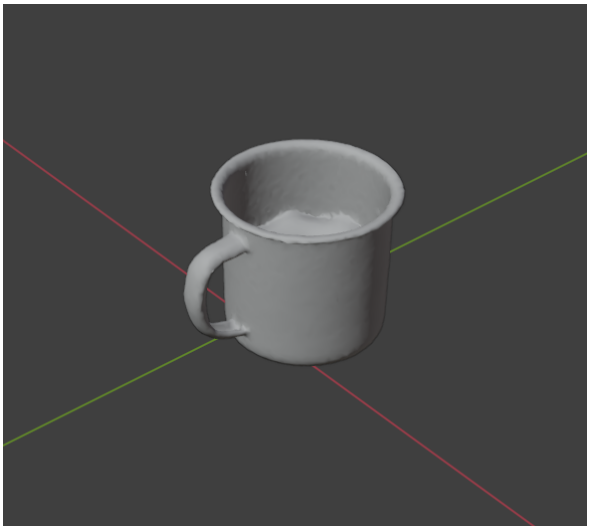

FIGURE 3.29 – Vue en ensemble de l'objet #23 (Tasse rouge) dans Blender. L'objet est importé avec un vecteur avant X et un vecteur haut Z.

FIGURE 3.30 – Vue de l'objet #11 (Chaise berçante) dans Blender. Le cube gris représente le premier test de toucher $(X, Y, Z = [0.249131, 0.553247, 0.179487]$, coordonnées locales).

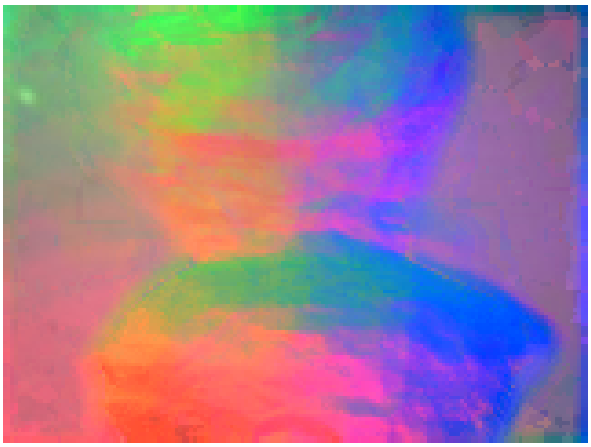

FIGURE 3.31 – Simulation avec la composante tactile d'ObjectFolder 2.0 sur l'objet #11 (Chaise berçante) $(X, Y, Z = [0.249131, 0.553247, 0.179487]$, coordonnées locales).

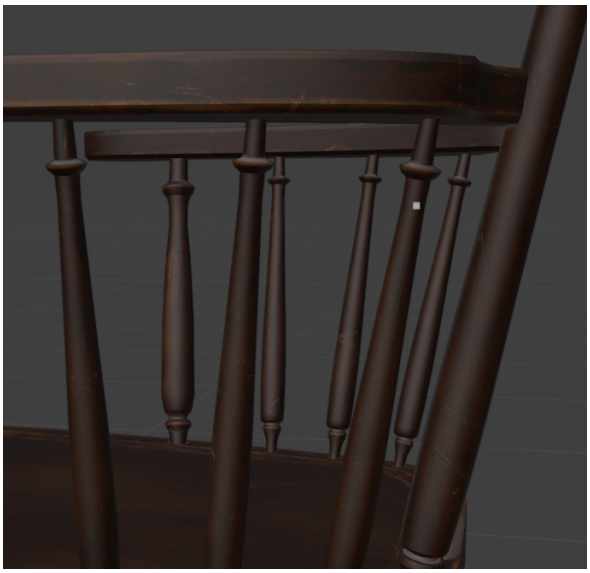

FIGURE 3.32 – Vue de l'objet #11 (Chaise berçante) dans Blender. Le cube gris représente le deuxième test de toucher ($X, Y, Z = [0.224512, 0.73938, -0.050904]$, coordonnées locales).

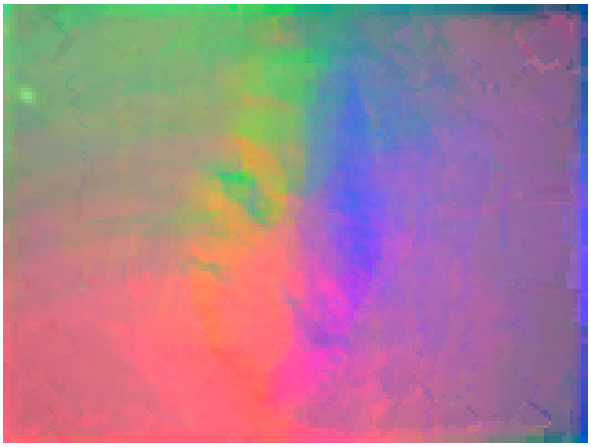

FIGURE 3.33 – Simulation avec la composante tactile d'ObjectFolder 2.0 sur l'objet #11 (Chaise berçante) ($X, Y, Z = [0.224512, 0.73938, -0.050904]$, coordonnées locales).

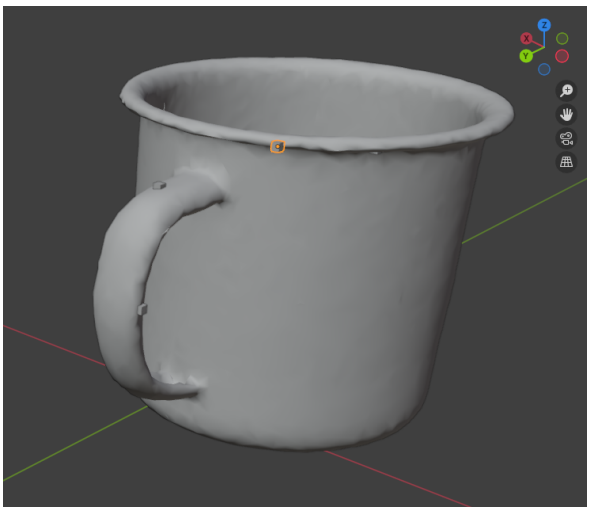

FIGURE 3.34 – Vue de l'objet #23 (Tasse rouge) dans Blender. Le cube gris surligné représente le premier test de toucher ($X, Y, Z = [0.021837, 0.035026, 0.0784]$, coordonnées locales).

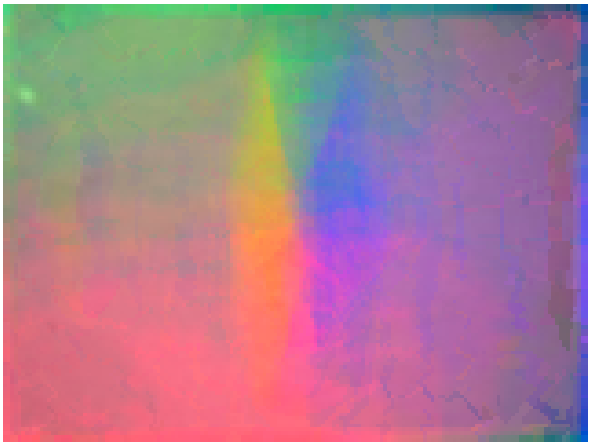

FIGURE 3.35 – Simulation avec la composante tactile d'ObjectFolder 2.0 sur l'objet #23 (Tasse rouge) ($X, Y, Z = [0.021837, 0.035026, 0.0784]$, coordonnées locales). Nous remarquons l'orientation verticale du rebord de la tasse dans le toucher, alors que nous nous attendions à un profil horizontal.

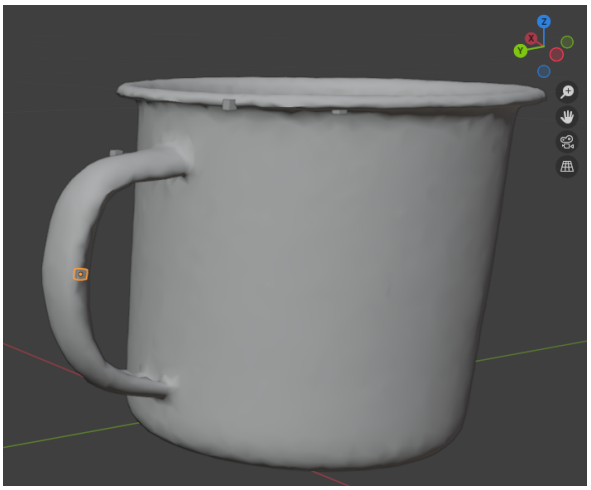

FIGURE 3.36 – Vue de l'objet #23 (Tasse rouge) dans Blender. Le cube gris surligné représente le deuxième test de toucher $(X, Y, Z = [0.046499, 0.020586, 0.044024]$, coordonnées locales).

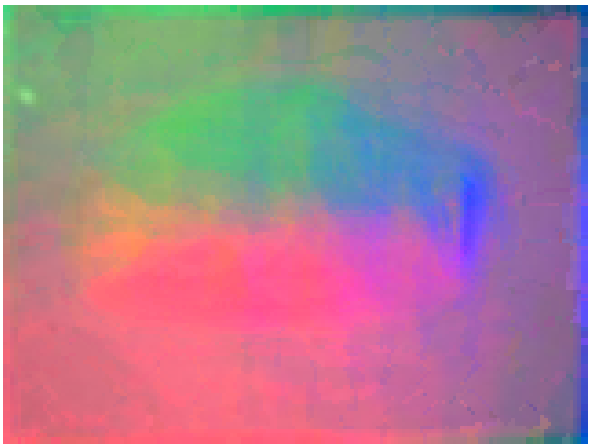

FIGURE 3.37 – Simulation avec la composante tactile d'ObjectFolder 2.0 sur l'objet #23 (Tasse rouge) $(X, Y, Z = [0.046499, 0.020586, 0.044024]$, coordonnées locales). Nous remarquons l'orientation horizontale de l'anse de la tasse dans le toucher, alors que nous nous attendions à un profil vertical.

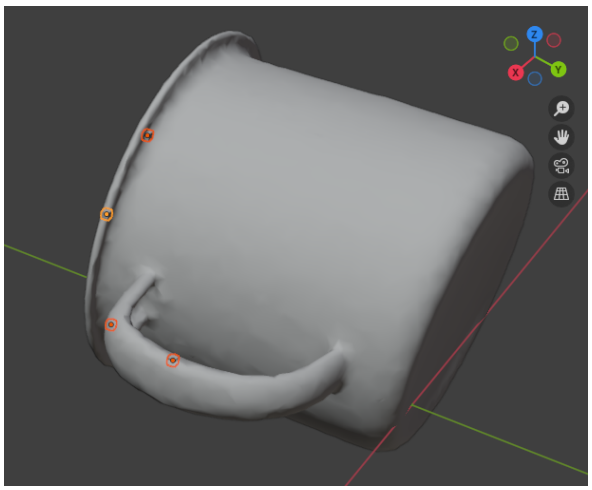

FIGURE 3.38 – Vue en ensemble de l'objet #23 (Tasse rouge) dans Blender. L'objet est importé avec un vecteur avant $-Z$ et un vecteur haut Y. Les images de toucher aux figures 3.35 et 3.37 s'alignent avec l'objet présenté dans cette orientation.

Le résultat de ces modifications peut être observé en comparant les figures 3.43 et 3.44 avec les figures 3.35 et 3.37.

```
# TouchNet options
parser.add_argument('--touch_vertices_file_path',
 ↪  default='./data/touch_demo_vertices.npy', help='The path of the
 ↪  testing vertices file for touch, which should be a npy file.')
parser.add_argument('--touch_gelinfo_file_path',
 ↪  default='./data/touch_demo_gelinfo.npy', help='The path of the gel
 ↪  configurations for touch, which should be a npy file.')
parser.add_argument('--touch_results_dir', type=str,
 ↪  default='./results/touch/', help='The path of the touch results
 ↪  directory to save rendered tactile RGB images.')
parser.add_argument("--touch_is_rotated", action="store_true") # Nouveau
```

FIGURE 3.39 – Modification apportée à `OF_render.py` afin de permettre d'activer la rotation de la matrice des profondeurs lorsque l'utilisateur en indique la nécessité pour l'objet à traiter.

3.4 Apprentissage Machine avec les données Sim2Real

3.4.1 Préparation des données

À partir de la méthodologie développé à la section 3.3, nous pouvons générer les données visuelles, tactiles et audio à partir des fichiers de points et d'*ObjectFolder 2.0*. Le processus de préparation des séparé en trois étapes principales, soit :

1. Le pré-traitement des fichiers de points saillants et des modèles des objets correspondants ;
   - Identification des caractéristiques clés des objets, telles que les valeurs de normalisation à appliquer (point centroïde, taille de l'objet, normale).
   - Le processus est appliqué pour chaque point saillant (20 par objet) ainsi qu'un nombre de points sélectionnés aléatoirement sur l'objet (100 par objet).
     - Les points aléatoires nous permettrons d'évaluer la performance du modèle selon si l'entraînement a été fait en utilisant les points visuellement saillants ou non.
     - Ces points sont sélectionnés en utilisant la fonction *Python* `random.sample()` [57] sur l'ensemble de tous les points uniques composant l'objet.

```
preds = preds  * (depth_max - depth_min) + depth_min
preds = np.transpose(preds.reshape((N, -1, 1)), axes = [0, 2,
↪  1]).reshape((N, rgb_width, rgb_height))

# Force 120x120 (Nouveau)
preds = preds[:, :, 20:-20]

# Rotate if argument present (Nouveau)
if args.touch_is_rotated:
    preds = np.rot90(preds, k=-1, axes=(1, 2))

taxim = TaximRender("./calibs/")
for i in trange(N):
    height_map, contact_map, tactile_map = taxim.render(preds[i],
    ↪  displacement[i])
    tactile_map = Image.fromarray(tactile_map.astype(np.uint8), 'RGB')
    filename = os.path.join(testsavedir, '{}.png'.format(i+1))
    tactile_map.save(filename)
```

FIGURE 3.40 – Modification apportée à `OF_render.py` afin de réduire la matrice de profondeurs en sortie du modèle tactile à 120×120 et ensuite appliquer une rotation de la matrice si l'utilisateur a activé l'indicateur ajouté à la figure 3.39.

```
# load depth bg
self.bg_depth = np.load(osp.join(calib_path,"depth_bg.npy"),
↪  allow_pickle=True)
self.bg_depth = self.bg_depth[:, 20:-20] # Nouveau
# load tactile bg
self.real_bg = np.load(osp.join(calib_path,"real_bg.npy"),
↪  allow_pickle=True)
self.real_bg = self.real_bg[:, 20:-20] # Nouveau
```

FIGURE 3.41 – Modification apportée à `taxim_render.py` afin de supporter une matrice de profondeurs de 120×120. Les 20 pixels les plus à gauche et les plus à droites sont retirées.

```
ball_radius = 4.00/2
pixmm = 0.1245  # 0.0295; # 0.0302
numBins = 120

# sensor setting
h = 120
# w = 160
w = 120
cam2gel = 0.04
```

FIGURE 3.42 – Modification apportée à `sensorParams.py` afin de supporter une matrice de profondeurs de 120×120. Nous passons de `w = 160` à `w = 120`.

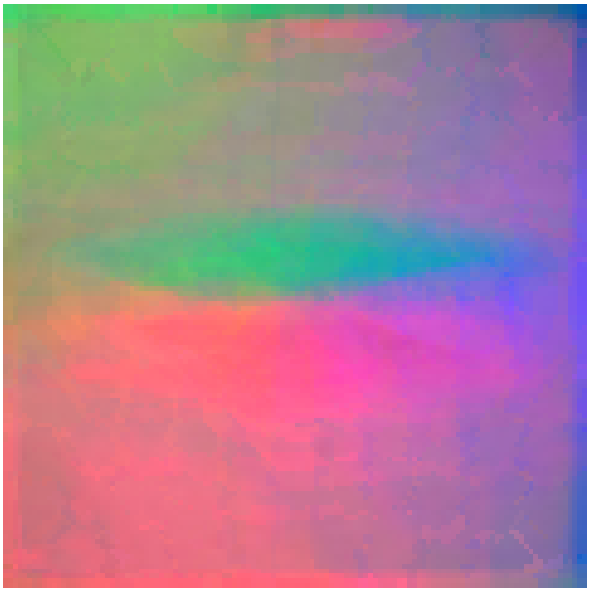

FIGURE 3.43 – Simulation avec la composante tactile d'ObjectFolder 2.0 sur l'objet #23 (Tasse rouge) ($X, Y, Z = [0.021837, 0.035026, 0.0784]$, coordonnées locales). L'orientation du résultat est maintenant corrigée par rapport à la figure 3.35.

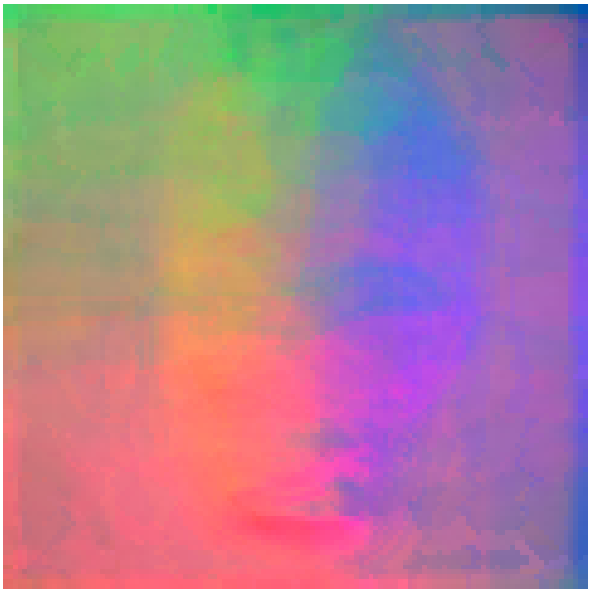

FIGURE 3.44 – Simulation avec la composante tactile d'ObjectFolder 2.0 sur l'objet #23 (Tasse rouge) ($X, Y, Z = [0.046499, 0.020586, 0.044024]$, coordonnées locales). L'orientation du résultat est maintenant corrigée par rapport à la figure 3.37.

- Création d'un fichier de définitions d'objets à être utilisé avec *ObjectFolder 2.0*. (Voir le tableau E.1)

2. La lecture du fichier de définitions avec *ObjectFolder 2.0* et la création de la base de données ;
   - Pour chaque point dans le fichier de définitions, les données sont réinterprétées dans le format attendu pour les différentes modalités d'*ObjectFolder*.
   - Pour la modalité visuelle, le point sur l'objet est normalisé, déplacé et réorienté. Le point de la caméra est calculé. Une augmentation est appliquée en sélectionnant des points de caméras alternatifs autour du point original, le processus étant nécessaire afin d'assurer une plus grande quantité de données lors de l'entraînement de modèles ainsi que d'avoir une bonne vue de chaque point.
   - Pour la modalité tactile, le point est utilisé directement et l'orientation de l'image générée est corrigée lorsque l'objet est l'un de ceux dont le modèle a été créé avec l'outil *Katamari*, suivant les explications à la section 3.3.2.
   - Pour la modalité audio, le point est utilisé directement, aucune autre manipulation est nécessaire.
3. La création d'un fichier de méta-données pour la base de données d'images et de fichiers audio.
   - Chaque fichier généré par les modalités visuelle, tactile et audio d'*ObjectFolder 2.0* sont décrits dans un ensemble de trois fichiers de méta-données (tableaux E.2, E.3 et E.4). Les détails de ces fichiers sont élaborés plus loin à la section 3.4.1.

Pré-traitement

Les fichiers des objets sont téléchargé et extraits à partir du lien `https://download.cs.stanford.edu/viscam/ObjectFolder/ObjectFolder1-100.tar.gz`. Pour chaque objet, nous avons un fichier *Wavefront* (*model.obj*) ainsi qu'un fichier « *Checkpoint* » (*ObjectFile.pth*), utilisé par la librairie *TensorFlow* à l'intérieur d'*ObjectFolder 2.0* et contenant tous les paramètres et poids des modèles visuels, tactiles et audio.

Les fichiers de points saillants sont obtenus à partir des travaux de Ghazal Rouhafzay [2], contenant les 20 points les plus saillants identifiés sur un ensemble de 11 objets, soit :

- Objet #2 : Un baril métallique jaune.
- Objet #4 : Une télévision, style rétro.
- Objet #5 : Une chaise en bois, style rétro.
- Objet #6 : Une commode en bois.
- Objet #7 : Une table de salon en bois.
- Objet #8 : Une guitare acoustique en bois.
- Objet #9 : Une table de salon en bois, style gothique.
- Objet #10 : Un baril bleu en plastique.
- Objet #53 : Un petit bol d'eau ou de nourriture en céramique pour animaux de compagnie.
- Objet #55 : Une petite assiette rectangulaire en céramique.
- Objet #77 : Un organisateur de fichiers métallique.

Une image représentative de ces objets est disponible à la figure C.1.

Chaque fichier de points est présenté sous le format *Excel* (ex. : *Object_2.xlsx*) ne contenant qu'une matrice de 20×3 valeurs, soit 20 ensembles de points x, y, z. Pour chaque point dans tous ces fichiers, les caractéristiques décrites à la figure E.1 sont extraites. De plus, 100 points sont sélectionnés aléatoirement pour chaque objet afin d'augmenter l'échantillon de données, avec un exemple de ces points pour l'objet #9 à la figure 3.45. Le tableau généré est enregistré sous format *CSV* (« *Comma Separated Values* »), prêt à être utilisé à l'étape suivante en tant que notre fichier de définitions d'objets.

Génération de la base de données avec ObjectFolder 2.0

Suivant les modifications et la technique développée à la section 3.3, nous chargeons le fichier *CSV* généré à l'étape précédente et appliquons les traitements nécessaires pour chacune des trois modalités.

Pour chaque point et pour chaque objet, nous obtenons la position du point dans l'espace d'origine (*vert_x,_y,_z*), la normale à ce point (*normal_x,_y,_z*, normalisée à un vecteur unitaire) ainsi que si l'objet est de type *Blender* (*is_blender_model*). Les trois modalités sont alors exécutés avec les traitements correspondants.

La modalité visuelle Pour la modalité visuelle, nous obtenons le point centroïde de l'objet (*cen_x,_y,_z*, une mesure de la largeur générale de l'objet (*obj_scale*) ainsi que le facteur de normalisation (l'inverse d'*obj_scale_3*).

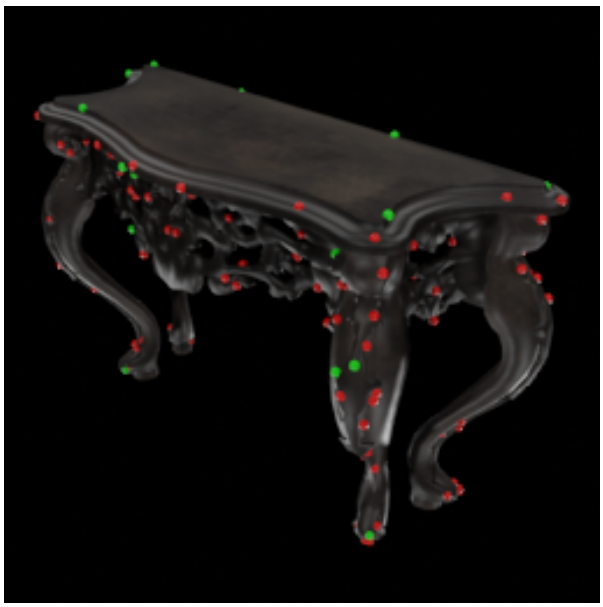

FIGURE 3.45 – Une image générée à partir de *PyBlender* démontrant en vert les points saillants et en rouge les points choisis aléatoirement.

Avec ces informations, nous soustrayons tout d'abord le point centroïde au point d'origine. Ensuite, ce point, ainsi que sa normale, reçoivent une rotation de 90 degrés tel que décrit à la section 3.3.2. Le point est par après déplacé le long de sa normale selon la largeur de l'objet divisée par deux. Finalement, les trois composantes sont multipliées par le facteur de normalisation.

Sans compter l'étape de rotation, ce processus est décrit à l'équation 3.3 pour la composante x comme exemple, où :

- x_{final} représente la coordonnée x à la fin de la transformation ;
- $x_{original}$ est le point initial tel que reçu à partir du fichier *CSV* ;
- x_{cen} est le point centroïde de l'objet ;
- x_{norm} est la normale à la surface de l'objet pour $x_{original}$;
- obj_scale est la longueur de la diagonale de la boîte englobante de l'objet ;
- obj_scale_3 est la longueur maximale entre les trois axes de la boîte englobante de l'objet ;

$$x_{final} = (x_{original} - x_{cen} + \frac{obj\_scale}{2} x_{norm}) \frac{1}{obj\_scale\_3} \tag{3.3}$$

Comme technique d'augmentation de données, nous effectuons plusieurs rotations sur le vecteur normal du point. Supposons ce vecteur étant un point sur une sphère de rayon

unitaire, nous appliquons la formule de conversion du point en coordonnées polaires θ, ϕ (eq. 3.4). Nous ajoutons ensuite des angles de $\pm 20\,\text{deg}$ et appliquons la formule de conversion inverse (eq. 3.5), nous donnant un total de 9 vecteurs normaux (incluant le vecteur normal original) à utiliser pour chaque point dans la formule 3.3.

$$\theta = \arccos(z), \phi = \arctan 2(y, x) \tag{3.4}$$

$$x = \sin(\theta)\cos(\phi), y = \sin(\theta)\sin(\phi), z = \cos(\theta) \tag{3.5}$$

Finalement, la matrice *c2w* (« *Camera-to-World* ») est calculée pour tous les points en utilisant le code à la figure 3.23. Les angles θ_c et ϕ_c sont calculés en utilisant le vecteur normal au point sélectionné $(x_{norm}, y_{norm}, z_{norm})$ à l'équation 3.6.

$$\theta_c = \arccos(z_{norm}), \phi_c = \arctan 2(x_{norm}, -y_{norm}) \tag{3.6}$$

Les matrices aux équations 3.7 et 3.8 sont calculées, représentant les matrices de rotation sur l'axe X (Rx) et l'axe Z (Rz). Celles si sont multipliées pour obtenir la matrice de rotation finale Rzx à l'équation 3.9.

$$Rx = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_c) & -\sin(\theta_c) \\ 0 & \sin(\theta_c) & \cos(\theta_c) \end{pmatrix} \tag{3.7}$$

$$Rz = \begin{pmatrix} \cos(\phi_c) & -\sin(\phi_c) & 0 \\ \sin(\phi_c) & \cos(\phi_c) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

$$Rzx = Rz \cdot Rx \tag{3.9}$$

Ensuite, les coordonnées de la caméra (x_c, y_c, z_c) sont ajoutées pour obtenir la matrice à l'équation 3.10, représentant la rotation de la caméra et sa position dans la scène, soit le format attendu par les composantes internes de la modalité visuelle d'*ObjectFolder 2.0*.

$$C2W = \begin{pmatrix} Rzx_{1,1} & Rzx_{1,2} & Rzx_{1,3} & x_c \\ Rzx_{2,1} & Rzx_{2,2} & Rzx_{2,3} & y_c \\ Rzx_{3,1} & Rzx_{3,2} & Rzx_{3,3} & z_c \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

Ces données sont enregistrés sous format *NumPy* dans un emplacement temporaire et la modalité visuelle d'*ObjectFolder 2.0* est invoquée à l'aide de la commande à la figure 3.46.

```
v_args = [
    # Endroit où se trouve ObjectFile.pth
    "--object_file_path", obj_file,
    # Modalité vision seulement
    "--modality", "vision",
    # Endroit où nous avons enregistrés les points à traiter (NumPy)
    "--vision_test_file_path", v_pts_file,
    # Dossier où enregistrer les images en sortie
    "--vision_results_dir", v_res_dir,
    # Nous donnons toujours les données de poses c2w directement
    "--vision_is_poses"
]
ofr.run_OF_render(v_args)
```

FIGURE 3.46 – Commande exécutée dans *Python* pour générer les données visuelles à l'aide d'ObjectFolder 2.0 pour chaque objet.

Les modalités tactiles et audio Pour les deux autres modalités, aucune transformation n'est requise au niveau des points x, y, z. Dans le cas de la modalité tactile, une augmentation de données est effectuée en itérant sur le facteur de profondeur (gel_d dans E.3) avec 9 valeurs équidistantes entre 0.0005 et 0.002. Ceci a pour but, comme pour la modalité visuelle, d'assurer un plus grand nombre de données disponibles pour l'entraînement et d'extraire un maximum d'information possible pour chaque point.

Les points sont enregistrés sous format *NumPy* et les commandes aux figures 3.47 et 3.48 sont invoquées pour les modalités tactiles et audio respectivement. De plus, la commande `--touch_is_rotated` est ajoutée à la modalité tactile si l'objet n'est pas d'origine *Blender* afin d'assurer la bonne orientation de l'image en sortie, tel que décrit à la section 3.3.2.

```
t_args = [
    # Endroit où se trouve ObjectFile.pth
    "--object_file_path", obj_file,
    # Modalité tactile seulement
    "--modality", "touch",
    # Endroit où nous avons enregistrés les points à traiter (NumPy)
    "--touch_vertices_file_path", t_pts_file,
    "--touch_gelinfo_file_path", t_gel_file,
    # Dossier où enregistrer les images en sortie
    "--touch_results_dir", t_res_dir
]
# Nous ajoutons cet argument s'il s'agît de l'un des objets Katamari
if not is_blender:
    t_args += ["--touch_is_rotated"]
ofr.run_OF_render(t_args)
```

FIGURE 3.47 – Commande exécutée dans *Python* pour générer les données tactiles à l'aide d'ObjectFolder 2.0 pour chaque objet.

```
a_args = [
    # Endroit où se trouve ObjectFile.pth
    "--object_file_path", obj_file,
    # Modalité audio seulement
    "--modality", "audio",
    # Endroit où nous avons enregistrés les points à traiter (NumPy)
    "--audio_vertices_file_path", a_pts_file,
    "--audio_forces_file_path", a_frc_file,
    # Dossier où enregistrer les fichiers audio en sortie
    "--audio_results_dir", a_res_dir
]
ofr.run_OF_render(a_args)
```

FIGURE 3.48 – Commande exécutée dans *Python* pour générer les données audio à l'aide d'ObjectFolder 2.0 pour chaque objet.

La base de données et les méta-données

Les données générées par *ObjectFolder 2.0* sont enregistrées dans un dossier suivant la structure suivante :

- Les dossiers « vision », « touch » et « audio » ;
  - Sous chaque dossier, un dossier par objet (« 2 », « 4 », « 5 », etc.) contenant tous les fichiers générés pour cet objet.
- Les fichiers de méta-données pour chaque modalité, soit `vision.csv`, `touch.csv` et `audio.csv`.
  - Les structures de ces fichiers sont décrites aux tableaux E.2, E.3 et E.4 respectivement.

De plus, des fichiers *CSV* supplémentaires sont décrits manuellement aux figures E.1 et E.2 afin de permettre l'association des identifiant des objets à des classes distinctes lors de l'entraînement de modèles à l'étape suivante.

3.4.2 Création des modèles

Un nouvel environnement Anaconda est créé à l'aide du fichier à la figure F.1 afin de supporter la création de modèles utilisant les données générées par *ObjectFolder 2.0*. Les paquets Python importants sont les suivants :

- *tensorflow* [58] : Ce paquet permet la création de modèles d'apprentissage profond. La version *2.8.4* est employée car les versions à partir de *2.11* ne supportent plus l'option d'exécuter les modèles sur Windows en utilisant une carte graphique. Les versions *2.9* et *2.10* comportent des problèmes par rapport au traitement des données et n'ont donc pas été considérées [59].
- *scikit-learn* [60] : Un paquet contenant plusieurs fonctions utilitaires nous permettant d'évaluer la performance des modèles.
- *librosa* [61] : Ce parquet permet l'analyse de fichiers audio et sera utilisé pour l'extraction de leurs caractéristiques sous forme de spectrogrammes pour le modèle Audio.

La structure de base du modèle est adaptée à partir d'un article mis en ligne par Alfred Weirich [62], indiquant comment effectuer l'apprentissage par transfert et le « *Fine-Tuning* » autour de *MobileNetV2* en utilisant *TensorFlow*. La plupart des paramètres utilisés par l'auteur ont été conservés pour ce projet. La structure générale du projet est décrite à la figure 3.49 et comporte trois étapes principales effectuées pour les modalités

Visuel, Tactile et Audio, soit le traitement de données, l'entraînement d'un modèle et l'évaluation de ce modèle. Finalement, un méta-modèle est créé en combinant les modèles entraînés afin d'effectuer des prédictions sur plusieurs modalités simultanément.

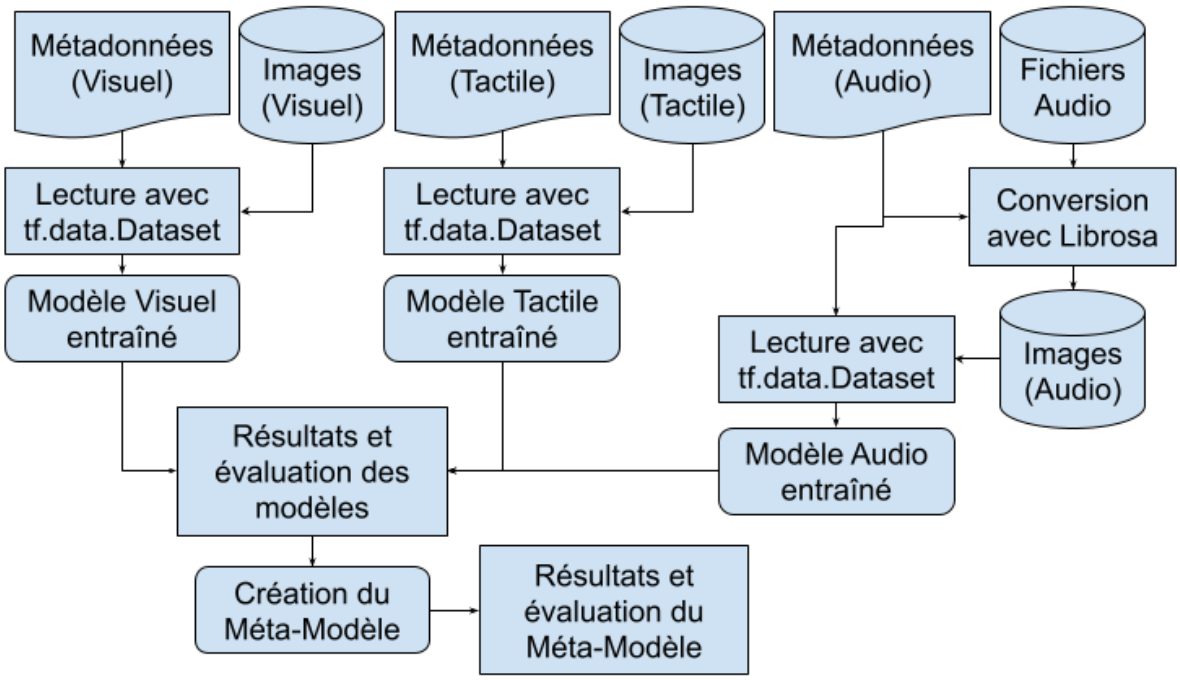

FIGURE 3.49 – La structure générale du projet d'apprentissage profond utilisant les données d'*ObjectFolder 2.0*. Les fichiers de méta-données sont utilisés pour charger les images à partir des bases de données et entraîner les modèles respectifs, tous basés sur *MobileNetV2*. La composante Audio comporte notamment une étape de conversion à l'aide de la librairie *Librosa*.

Pour les fichiers audio, les fichiers en format *.wav* sont convertis à l'aide de la librairie *Librosa*, nous permettant d'extraire des images de type *.png* représentant le spectrogramme Mel pour chaque fichier. Un exemple du résultat de cette extraction sur quelques points est visible à la figure 3.50. Ces images sont enregistrés dans un dossier « audio_mels » suivant la même structure que le dossier « audio » d'origine.

Le même processus est répété pour obtenir les images à la figure 3.51, représentant ces spectrogrammes sous format *MFCC* (« *Mel-frequency cepstral coefficients* ») dans un dossier « audio_mfcc ». Ce type d'encodage acoustique est commun dans le domaine de l'apprentissage machine pour sa capacité à fournir une représentation compressée des données, surtout au niveau de la reconnaissance vocale et de la détection d'anomalies [63]. Nous supposons que ceci s'étend au domaine de la classification d'objets.

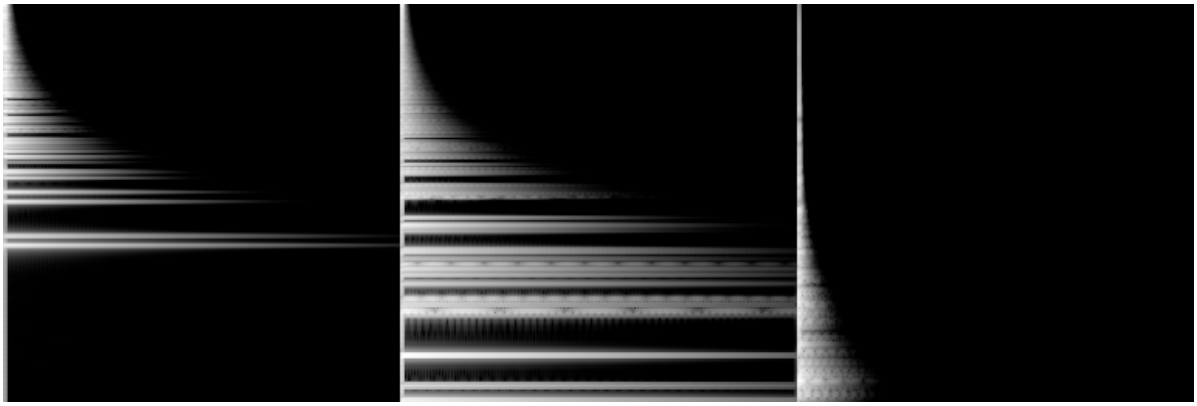

FIGURE 3.50 – Trois exemples de graphiques Mels extraits avec Librosa sur des fichiers audio générés pour les objets #2 (Baril jaune), #4 (Télévision) et #7 (Table en bois) respectivement.

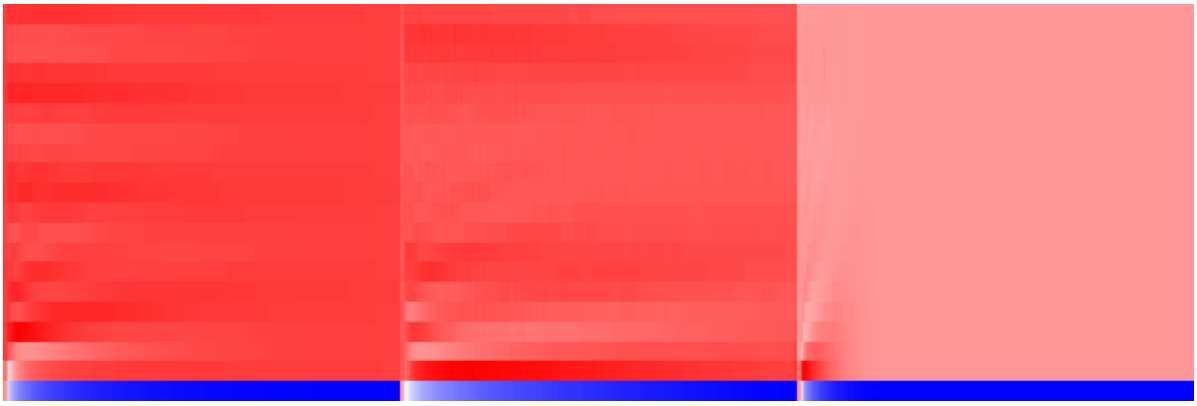

FIGURE 3.51 – Trois exemples de graphiques MFCC extraits avec Librosa sur des fichiers audio générés pour les objets #2 (Baril jaune), #4 (Télévision) et #7 (Table en bois) respectivement.

Traitement de données

L'étape de traitement de données est décrite en détails à la figure 3.52 où l'on crée un « *pipeline* » de lecture de données en utilisant les fonctionnalités offertes par l'objet *Dataset* de *TensorFlow* [64]. Ce « *pipeline* » comporte les étapes suivantes pour un modalité choisie :

- La lecture du fichier de méta-données contenant les informations sur la classe de chaque objet et l'emplacement de leur fichier *.png* respectif dans la base de données ;
- La conversion de la classe de l'objet en vecteur « *One-Hot* » de 9 éléments (voir la figure E.2 pour la liste des classes) ;
- Un mélange initial des données suivi de la séparation en trois sous-ensembles d'entraînement, de validation et de test ;
  - Le ratio entre les trois ensembles est de 70%, 15% et 15% respectivement.
  - La séparation est effectuée en sélectionnant les 70% premiers éléments pour les données d'entraînement, les 15% éléments suivant pour la validation et ce qui reste pour le test, d'où l'importance de devoir effectuer un mélange initial des données.
  - Le mélange initial utilise une graine fixe afin d'assurer que les trois sous-ensembles comportent toujours les mêmes données entre chaque exécution du script Python.
- Pour chaque élément des sous-ensembles, la donnée comportant l'emplacement du fichier *.png* est utilisée pour charger l'image, modifier sa taille à 224×224 et normaliser les valeurs *RGB* de $[0, 255]$ vers $[-1, +1]$, ces caractéristiques étant celles demandées pour l'utilisation avec *MobileNetV2* ;
- Les données d'entraînement et de validation ont une nouvelle couche de mélange ajoutée, assurant que chaque itération de l'entraînement des modèles soit effectuée en avec un ordre aléatoire des données ;
- Les données d'entraînement subissent une modification aléatoire pour chaque image comme étape d'augmentation ;
  - L'augmentation comporte, tous appliqués aléatoirement, une réflexion sur l'axe vertical, un ajustement de la luminosité et du contraste jusqu'à $\pm 10\%$ et une rotation de ± 0.05rad.
  - Cette augmentation est appliquée uniquement aux données de la modalité Visuelle, car nous supposons que la caméra d'un robot peut être affecté par

les conditions ambiantes tandis que les capteurs tactiles et audio produiront des données plus homogènes.

– Les données de chaque sous-ensemble sont séparées en lots de 32 éléments (« *Batching* »).

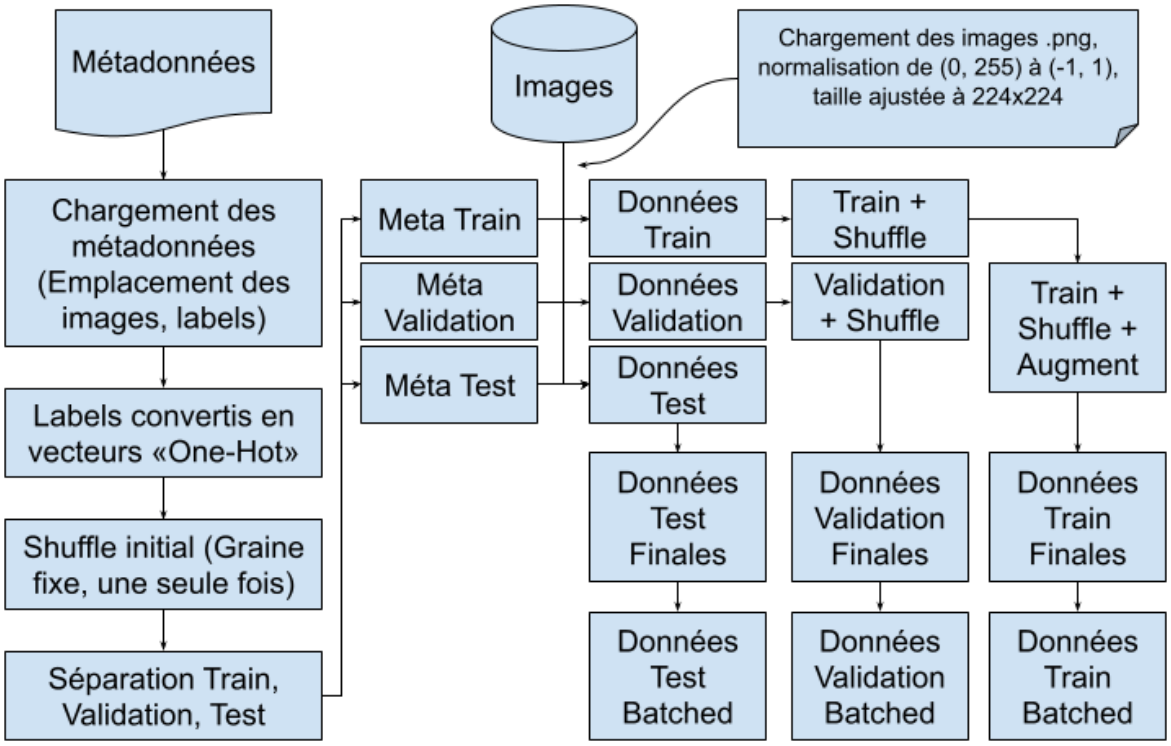

FIGURE 3.52 – L'étape de création du « *pipeline* » de lecture des méta-données et des images avant l'étape de l'entraînement.

Création et entraînement des modèles

La structure d'un modèle pour une modalité quelconque est décrite à la figure 3.53, où le modèle est composé de cinq couches distinctes :

– La couche d'entrée, où les images de taille 224×224 avec trois canaux pour les valeurs « *RGB* » (Rouge, Vert, Bleu) sont reçues ;

– Une instance de *MobileNetV2* [8], pré-chargée avec les poids de son entraînement sur ImageNet [20] et sa couche de sortie retirée ;

- Une couche de *Pooling* afin de réduire la sortie de *MobileNetV2* de $7 \times 7 \times 1280$ éléments à un ensemble de 1280 éléments en effectuant une opération de Moyenne sur chaque sous-groupe de taille 7×7 ;
- Une couche cachée entièrement connectée avec 256 neurones à être entraînés, utilisant la fonction d'activation *ReLU* ;
- Une couche de sortie entièrement connectée avec 9 neurones à être entraînés, utilisant la fonction d'activation *SoftMax*, nous donnant les prédictions sur les classes des objets.

Lors de l'entraînement du modèle, nous le compilons tout d'abord avec la fonction d'optimisation de type *Adam* [65] avec un taux d'apprentissage de 0.001. La perte du modèle utilise la fonction « *Categorical Cross-Entropy* » de *TensorFlow* [66]. Un modèle est entraîné de façon indépendante sur six sous ensembles de données d'entraînement pour chaque modalité, nous donnant un total de 24 modèles distincts :

- *Visual_Salient20* : Entraîné sur les données visuelles des 20 points visuellement saillants pour les 11 objets, avec 9 angles de vue par point saillant.
- *Touch_Salient20* : Les données tactiles pour les points visuellement saillants, avec 9 profondeurs pour le capteur tactile.
- *Audio_Salient20* : Les données audio (Mels) pour les points visuellement saillants.
- *AMFCC_Salient20* : Les données audio (MFCC) pour les points visuellement saillants.
- *Visual_NotSalient20* : Les données visuelles pour les 20 premiers points sélectionnés aléatoirement sur les objets, toujours avec les différents angles de vue.
- *Touch_NotSalient20* : Les données tactiles pour ces mêmes 20 points, toujours avec les 9 profondeurs.
- *Audio_NotSalient20* : Les données audio (Mels) pour ces mêmes 20 points.
- *AMFCC_NotSalient20* : Les données audio (MFCC) pour ces mêmes 20 points.
- *Modèles _NotSalient20_i* : Dont i allant de 2 à 5 pour chaque modalité.
  - *_NotSalient20_2* : Les points # 21 à 40 dans l'ensemble des 100 points sélectionnés aléatoirement pour chaque objets.
  - *_NotSalient20_3* : Les points # 41 à 60.
  - *_NotSalient20_4* : Les points # 61 à 80.
  - *_NotSalient20_5* : Les points # 81 à 100.

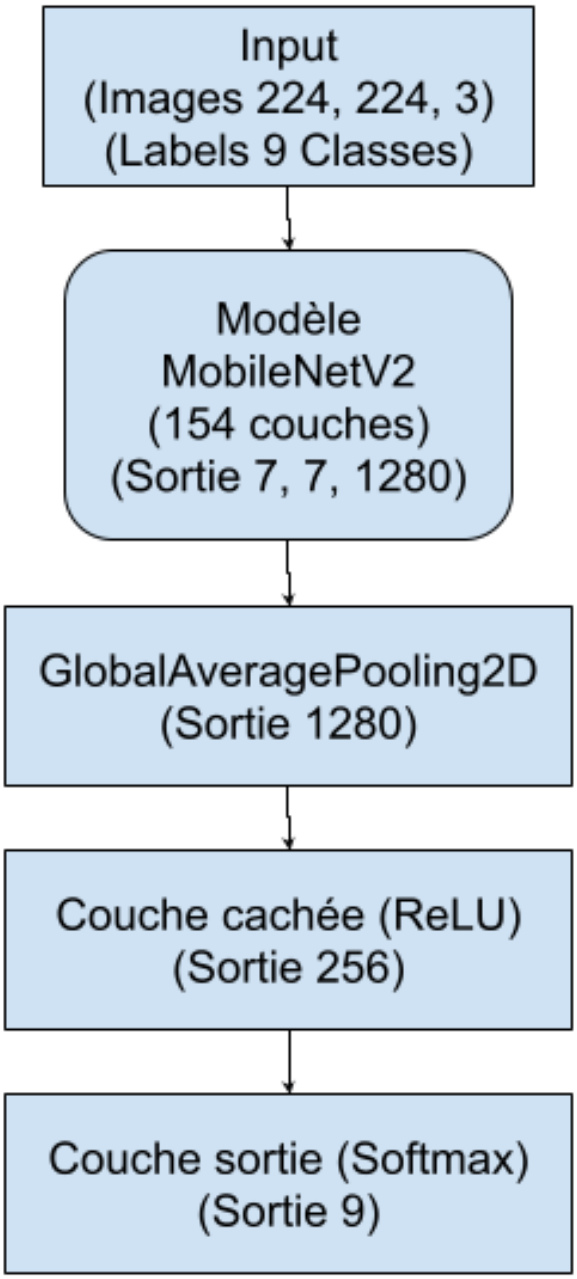

FIGURE 3.53 – La structure d'un modèle d'apprentissage profond utilisant MobileNetV2 comme modèle interne.

Dans le cas des données visuelles et tactiles, le nombre d'éléments dans les données d'entraînement, de validation et de test sont de 1386, 297 et 297 respectivement ($20 \times 11 \times 9 = 1980$). Pour les données audio, nous avons 154, 33 et 33 éléments ($20 \times 11 = 220$).

Chaque modèle est entraîné sur 20 époques initialement comme étape d'apprentissage par transfert, où le sous-modèle *MobileNetV2* est figé. Seul les deux dernières couches de notre modèle sont donc entraînées, soit la couche cachée de 256 neurones et la couche de sortie de 9 neurones. Par la suite, trois étapes de « *Fine-Tuning* » sur 5 époques chacune sont exécutées avec un taux d'apprentissage progressivement plus bas et en débloquant certaines couches du modèle *MobileNetV2* interne. Les étapes de l'entraînement sont donc :

1. *MobileNetV2* figé, le taux d'apprentissage est de 0.001, 20 époques ;
2. Les 92 dernières couches de *MobileNetV2* sont débloquées, le taux d'apprentissage est de 0.0001, 5 époques ;
3. Les 101 dernières couches de *MobileNetV2* sont débloquées, le taux d'apprentissage est de 0.00005, 5 époques ;
4. Les 110 dernières couches de *MobileNetV2* sont débloquées, le taux d'apprentissage est de 0.000025, 5 époques.

Les résultats de l'entraînement de ces modèles sont décrits au chapitre suivant à la section 4.1.

Création et entraînement du méta-modèle combiné

Une fois les modèles à modalité unique entraînés, nous les combinons à l'aide d'un méta-modèle dont la structure est décrite à la figure 3.54. Cette structure est une forme de *Stacking* [67], où un modèle méta-apprenant est entraîné à partir des résultats des sous-modèles le composant, nous donnant la prédiction finale. Six éléments composent ce modèle, soit :

- La couche d'entrée, où les images pour chaque modalité entrent simultanément. Les dimensions des sous-images restent les mêmes que pour les sous-modèles ;
- Une couche *Lambda* où nous implémentons une fonction qui sépare les sous-images de chaque modalité afin de les envoyer à leurs sous-modèles respectifs ;
- L'ensemble de modèles pour chaque modalité acceptée par le méta-modèle ;
- Une couche de concaténation, où les prédictions de tous les sous-modèles sont combinées en un seul vecteur ;

- Une couche cachée entièrement connectée avec 9 neurones à être entraînés, utilisant la fonction d'activation $ReLU$;
- Une couche de sortie entièrement connectée avec 9 neurones à être entraînés, utilisant la fonction d'activation *SoftMax*, nous donnant les prédictions finales sur les classes des objets.

Le méta-modèle comporte les modalités Visuel, Tactile et Audio ($MFCC$), mais par cause de préoccupations quant à la qualité des données audio, nous avons choisi de créer un deuxième ensemble de méta-modèles n'incluant que les modalités Visuel et Tactile à la figure 3.55. Ce choix est élaboré à la section 4.1.3.

Comme à la section précédente, deux variantes de modèles sont entraînés pendant 20 époques sur les six différents sous ensembles de points avec un taux d'apprentissage de 0.001 où seulement les deux dernières couches sont débloquées. Nous n'exécutons pas d'étape de « *Fine-Tuning* » pour ces modèles. Nous obtenons alors les modèles suivants :

- *VT_Salient20* : Entraîné sur les données visuelles des 20 points visuellement saillants pour les 11 objets, avec 9 angles de vue par point saillant et l'une des 9 profondeurs choisie aléatoirement provenant des données tactiles pour chaque point.
- *VT_NotSalient20* : Les données visuelles et tactiles proviennent de 20 points sélectionnés aléatoirement sur les objets.
- *VTA_Salient20* : Entraîné sur les données visuelles des 20 points visuellement saillants pour les 11 objets, avec 9 angles de vue par point saillant, l'une des 9 profondeurs choisie aléatoirement provenant des données tactiles et la donnée audio ($MFCC$) associée pour chaque point.
- *VTA_NotSalient20* : Les données visuelles, tactiles et audio (avec MFCC) proviennent de 20 points sélectionnés aléatoirement sur les objets.
- *Modèles _NotSalient20_*i : Dont i allant de 2 à 5 pour chaque modalité.

Le nombre d'éléments dans les données d'entraînement, de validation et de test sont d'ailleurs de 1386, 297 et 297 respectivement. Les résultats de l'entraînement de ces modèles sont décrits au chapitre suivant à la section 4.2.

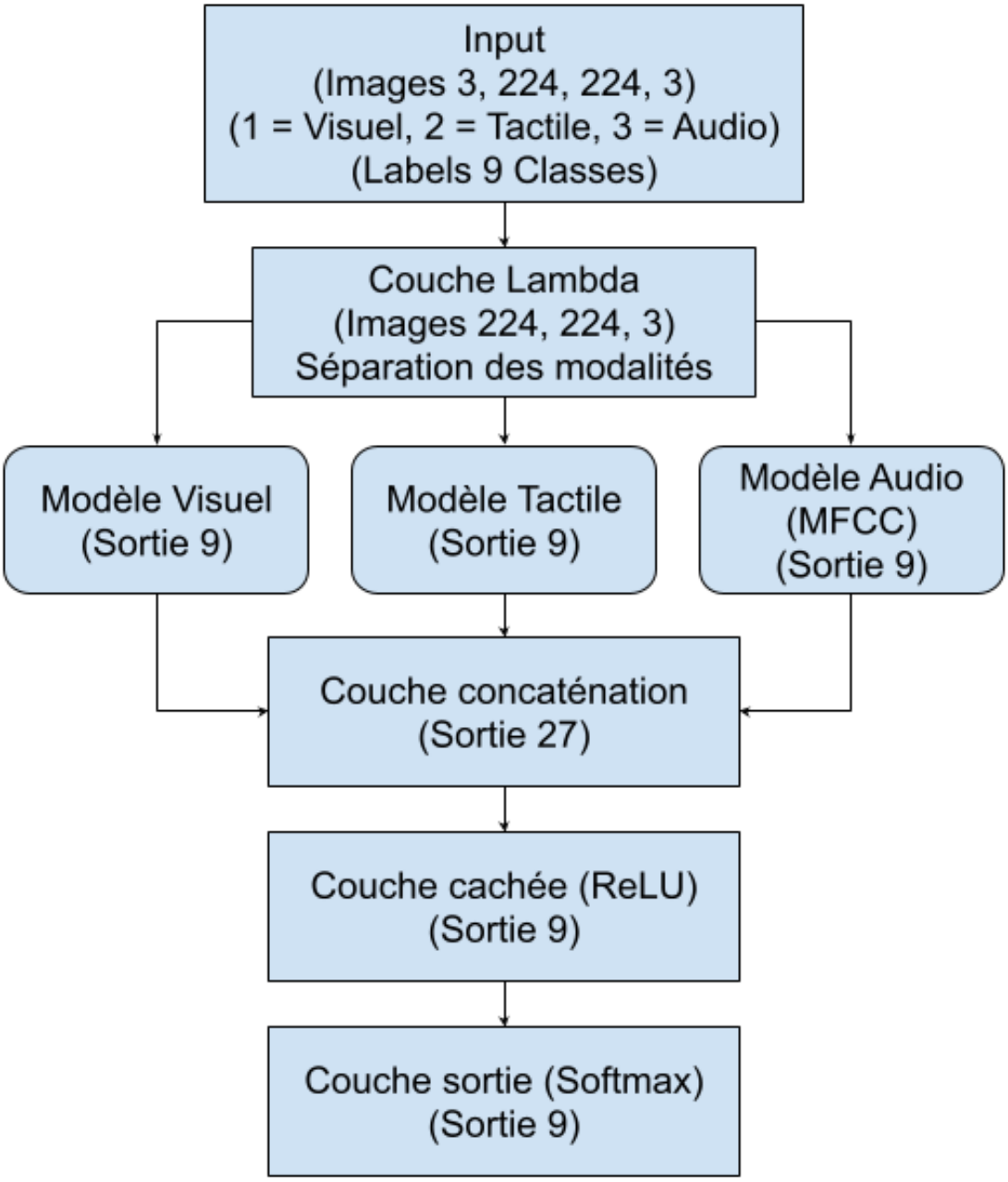

FIGURE 3.54 – La structure du méta-modèle combinant les modèles entraînés pour les modalités Visuel, Tactile et Audio (fig. 3.53).

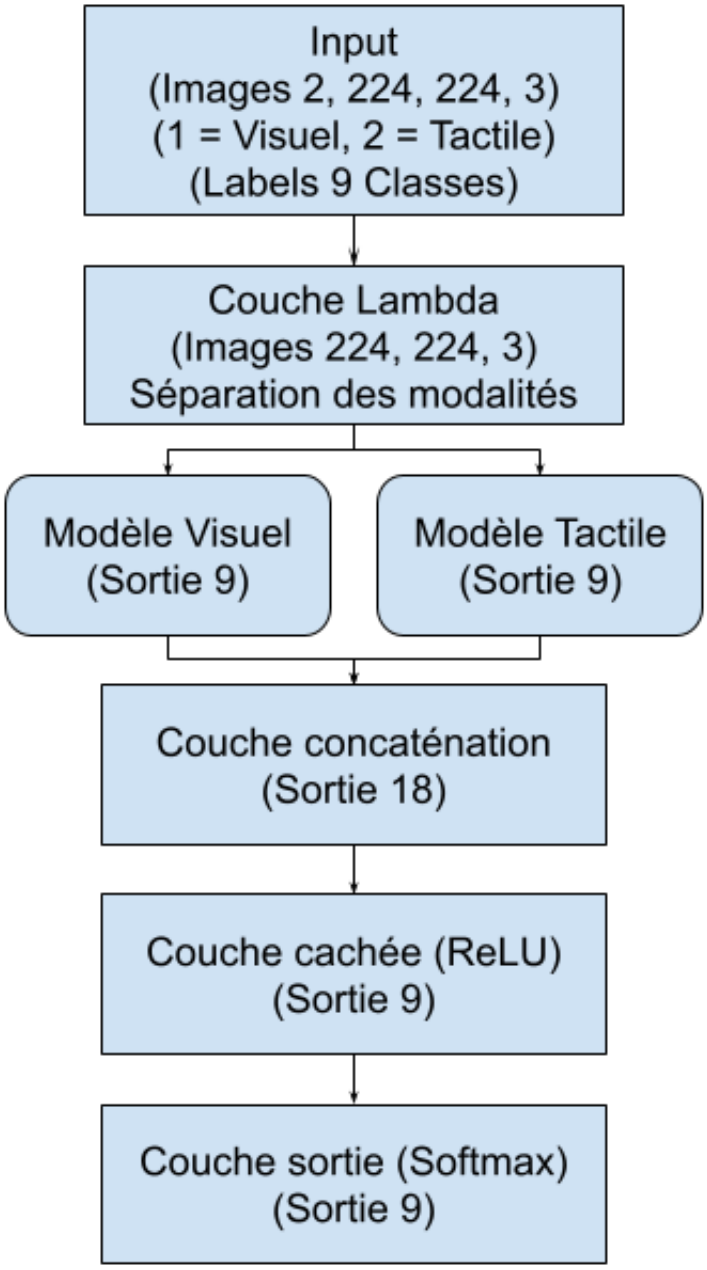

FIGURE 3.55 – La structure du méta-modèle combinant les modèles entraînés pour les modalités Visuel et Tactile seulement (fig. 3.53).

Chapitre 4

Résultats

Ce chapitre explore les résultats obtenus lors de l'entraînement des différents modèles utilisant la méthodologie présentée au chapitre 3. Nous comparons les performances entre les différentes modalités ainsi que l'effet observé entre l'utilisation des points saillants du point de vue de l'attention visuelle contre ceux choisis aléatoirement. Les métriques principales retenues pour évaluation sont le taux d'erreur, le score F1 Macro et le score F1 Micro. Il est à noter que sauf de spécification contraire, ces métriques sont calculées en exécutant les modèles entraînés sur les données de test.

Le taux d'erreur est défini à l'équation 4.1. Les fausses classifications pour une matrice de confusion donnée sont additionnés et comparés au total des vraies et fausses classifications de cette matrice. Une fausse classification FC représente une instance dont la classe a été prédite incorrectement (ex. : Un baril est prédit en tant que table) et une vraie classification VC est une instance prédite correctement. Cette métrique nous donne une idée du ratio d'éléments classifiés incorrectement par le modèle.

$$Erreur = \frac{FC}{VC + FC} \tag{4.1}$$

Le score F1 est défini par les équations 4.2, 4.3 et 4.4. La précision indique le ratio de prédictions réellement positives pour une classe donnée, tandis que le rappel indique le ratio d'éléments positifs correctement prédits. Le score F1 est une moyenne harmonique de ces deux ratios permettant de les combiner dans une seule métrique.

$$Precision_{classe} = \frac{VP_{classe}}{VP_{classe} + FP_{classe}} \tag{4.2}$$

$$Rappel_{classe} = \frac{VP_{classe}}{VP_{classe} + FN_{classe}} \tag{4.3}$$

$$F1Score_{classe} = \frac{2 \times Precision_{classe} \times Rappel_{classe}}{Precision_{classe} + Rappel_{classe}} \tag{4.4}$$

La précision, le rappel et le score F1 s'appliquent typiquement à des cas de classification binaire. Puisque nos modèles sont multi-classes, nous utilisons deux variantes de la formule du score F1 pour les évaluer. Le score F1 Micro représente simplement le score F1 calculé sur la somme des vrais positifs, faux positifs et faux négatifs pour toutes les classes confondues.

Le score F1 Macro est représenté à l'équation 4.5 par la moyenne des scores F1 pour chaque classe, nous permettant d'avoir une idée de l'impact sur la performance du modèle lorsque certaines classes présentent des problèmes de précision ou de rappel.

$$\frac{\sum_{i=1}^{num\_classes} F1Score_{classe\_i}}{num\_classes} \tag{4.5}$$

Les graphiques montrant les résultats de chaque modalité sont disponibles aux annexes suivantes :

- *Modalité visuelle* à l'annexe G ;
- *Modalité tactile* à l'annexe H ;
- *Modalité audio avec Mels* à l'annexe I ;
- *Modalité audio avec MFCC* à l'annexe J ;
- *Modalité visuelle/tactile combinée* à l'annexe K ;
- *Modalité visuelle/tactile/audio (MFCC) combinée* à l'annexe L.

À titre d'exemple, les sections 4.1 et 4.2 comparent en détails les résultats entre les modèles utilisant les points saillant (*_Salient20*) et ceux utilisant les 20 premiers points aléatoires (*_NotSalient20*) parmi les 100 sélectionnés lors de l'étape de pré-traitement à la section 3.4.1. Les résultats des modèles utilisant le reste des points aléatoires (*_NotSalient20_2* à *_NotSalient20_5*) sont explorés à la section 4.3 afin de confirmer les observations dans les sections précédentes.

4.1 Résultats des modèles à modalité unique

Cette section porte une attention particulière aux sous-modèles entraînés pour une modalité unique. Notamment, les modalités Visuel et Tactile sont retenues pour le méta-modèle à la section 4.2.1 tandis que les modalités Visuel, Tactile et Audio (*MFCC*) sont retenues pour le méta-modèle à la section 4.2.2.

4.1.1 Modèles Visuels

Les modèles entraînés sur les données visuelles montrent des taux de précision relativement élevés. Nous observons à la figure 4.1 la matrice de confusion pour les données de test sur le modèle visuel utilisant les points avec attention visuelle où le classement est presque parfait. Seulement 4 points sur 297 sont classifiés incorrectement, dont deux où une image d'une table (label 4) et de l'organisateur de fichiers (label 8) sont incorrectement prédits en tant que télévision (label 1). Nous observons un F1 Score Macro de 0.9837 et un F1 Score Micro de 0.9865 (figure G.3), une performance élevée attendue considérant que le sous-modèle *MobileNetV2* est spécialisé pour la classification de ce type d'images.

En comparant avec le modèle parallèle utilisant des points sélectionnés aléatoirement sur la surface des objets, nous remarquons une réduction de la performance. La figure 4.2 montrant la matrice de confusion pour ce modèle indique une classification incorrecte pour 12 sur 297 éléments avec des scores F1 Macro et Micro plus bas de 0.9565 et 0.9596 respectivement (figure G.6). Notamment, nous observons, comme pour le modèle précédent, une difficulté à classifier correctement les images de tables (label 4) et d'organisateurs de fichiers (label 8) avec chacun 4 instances incorrectes. Cette difficulté est toutefois beaucoup plus prononcée.

Nous comparons aussi l'évolution de l'exactitude lors de l'entraînement des deux modèles aux figures G.1 et G.4. Le modèle utilisant les points saillants converge aux alentours de l'époque 15, tandis que l'autre modèle ne commence à converger complètement qu'à partir de l'époque 21, lors de l'étape de « *Fine-Tuning* ».

Ceci confirme alors notre hypothèse que l'utilisation de données visuelle montrant un niveau d'attention élevé nous permet de conserver un taux de précision supérieur que lorsque les points sont choisis au hasard.

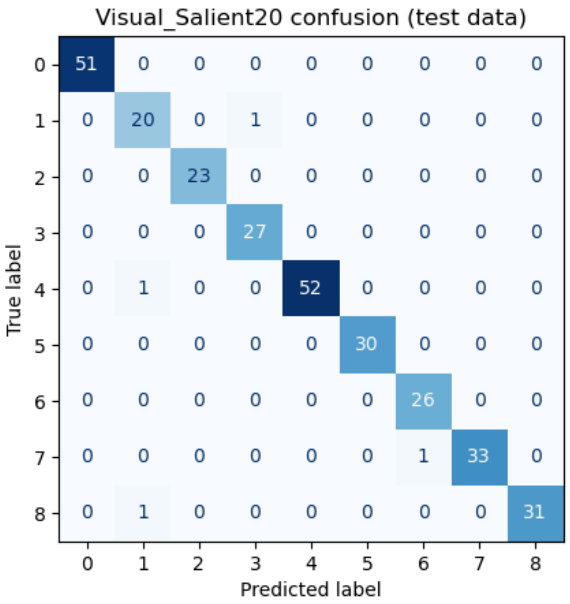

FIGURE 4.1 – Matrice de confusion sur les données de test du modèle visuel (Vision_Salient20).

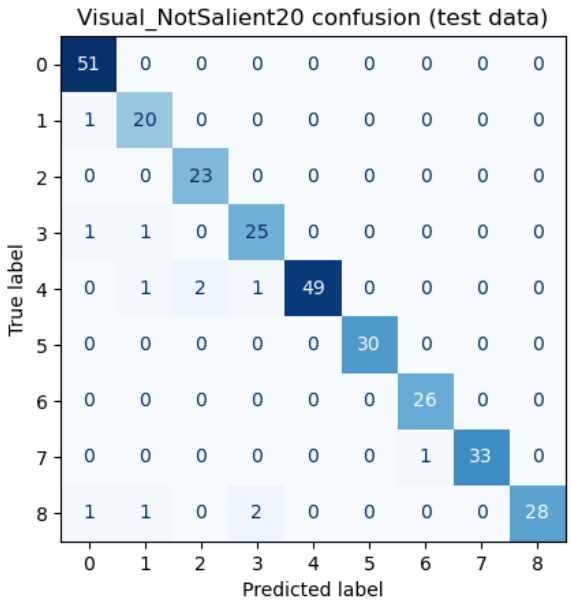

FIGURE 4.2 – Matrice de confusion sur les données de test du modèle visuel (Vision_NotSalient20).

4.1.2 Modèles Tactiles

Les modèles entraînés sur les données tactiles sont moins précis que ceux utilisant les données visuelles. Les matrices de confusion aux figures 4.3 et 4.4 indiquent un classement incorrect de 14 et 16 éléments sur 297 respectivement. Les F1 Score Macro et Micro sont de 0.9463 et 0.9529 pour le modèle utilisant les points saillants (fig. H.3) contre 0.9441 et 0.9461 pour celui avec les points aléatoires (fig. H.6).

Dans le modèle avec attention visuelle, les objets montrant la plus grande difficulté de classification sont la table (label 4) et la guitare acoustique. Ces deux objets sont parfois incorrectement classifiés en tant que chaise (label 2) ou télévision (label 1), de plus que certaines instance de la chaise sont classifiés en tant que table et guitare. Ceci est possiblement causé par des similarités au niveau de la texture de ces objets, surtout pour ceux en bois.

Le modèle utilisant les points aléatoires a comme difficulté principale la classification des barils (label 0), où quatre instances d'objets divers sont classifiées comme des barils et trois instances de barils sont classifiées comme télévision (label 1) ou table (label 4). L'assiette (label 7) montre aussi de la difficulté, avec deux instances de prédiction en

tant que bol (label 6), une en tant que chaise (label 2) et une autre en tant que baril (label 0).

Les courbes d'entraînement respectives aux figures H.1 et H.4 montrent une convergence similaire aux alentours de l'époque 13 avec un taux d'exactitude réduit sur les données de validation comparé aux modèles visuels, indiquant une plus grande difficulté à se généraliser aux données. Nous remarquons alors que l'utilisation de points saillants a un impact négligeable sur la performance du modèle tactile.

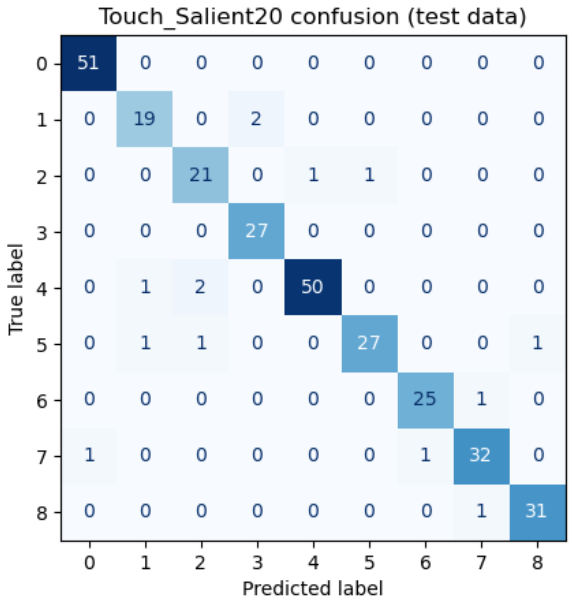

FIGURE 4.3 – Matrice de confusion sur les données de test du modèle tactile (Touch_Salient20).

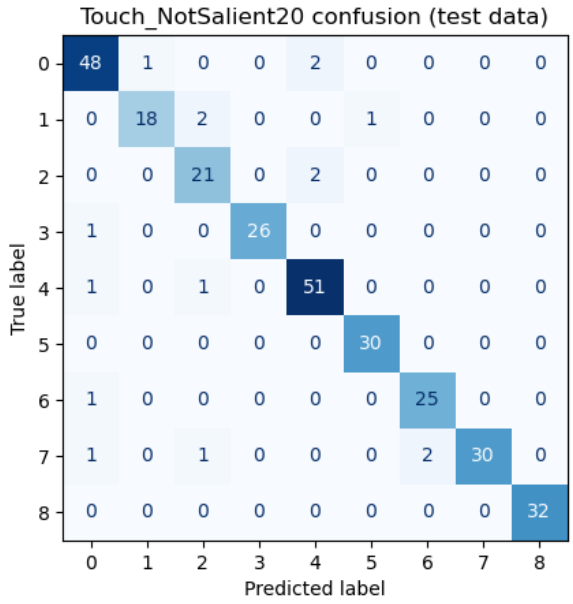

FIGURE 4.4 – Matrice de confusion sur les données de test du modèle tactile (Touch_NotSalient20).

4.1.3 Modèles Audio (Mels)

Les modèles audio utilisant les spectrogrammes Mel convergent très rapidement lors de leur entraînement, à partir de l'époque 8 pour le modèle avec attention visuelle (fig. I.1) et l'époque 5 pour celui avec les points aléatoires (fig. I.4), rendant l'incidence de l'utilisation des points saillants difficile à mesurer. En prenant un objet comme exemple à la figure 4.5, nous remarquons que toutes les images générées pour un objet donné présentent un niveau de similarité visuelle élevé, même entre les points ayant été choisis pour leur attention visuelle.

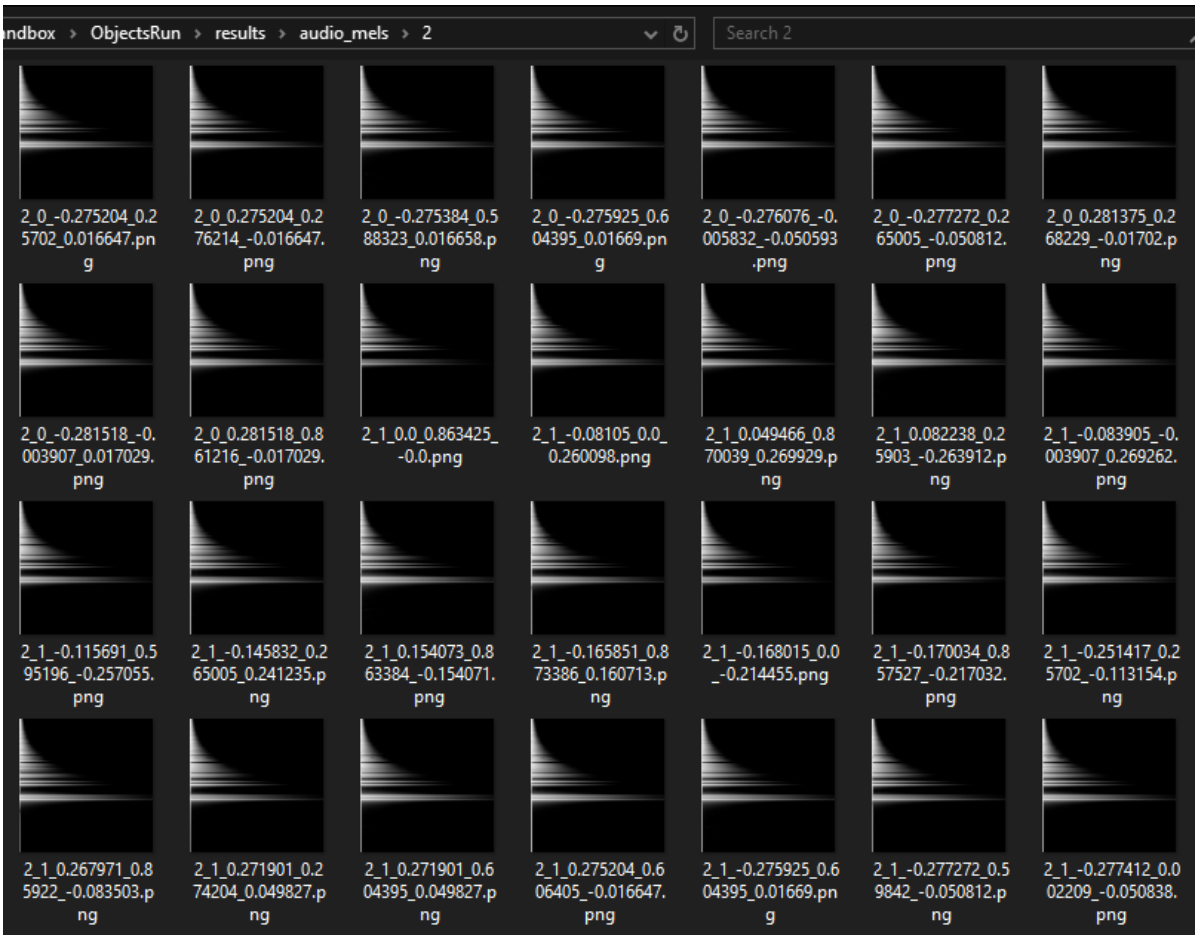

FIGURE 4.5 – Exemple des images Mels générées pour l'objet #2 (Baril jaune). La variance visuelle entre chaque image est très limitée.

De plus, ces modèles n'ayant pas été ciblés pour une augmentation de données, le nombre total d'éléments dans les données de test n'est que de 33, un nombre trop petit laissant certaines classes sous-représentées dans nos mesures, dont la classe 2 (Chaise) n'ayant aucun support lors de la sélection aléatoire des sous-ensembles. Même si les matrices de confusion aux figures 4.6 et 4.7 montrent un classement parfait sur les données de test, le volume total ne permet pas d'établir de conclusions sur l'efficacité des modèles. Les courbes ROC sont visibles aux figures I.3 et I.6, indiquant d'ailleurs des scores F1 Macro et Micro parfaits.

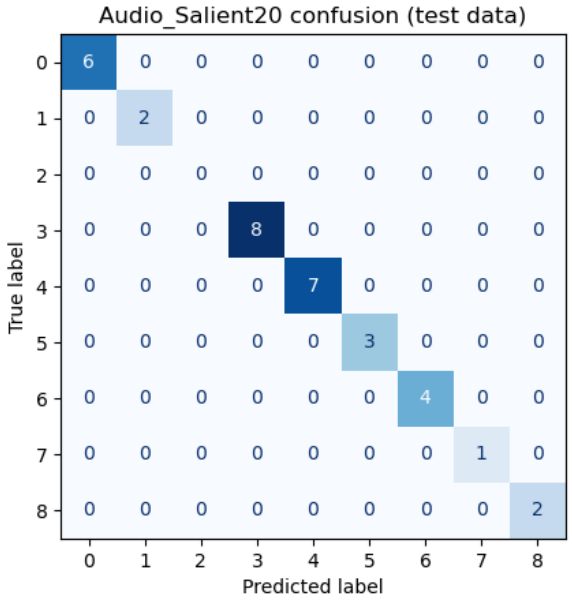

FIGURE 4.6 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_Salient20).

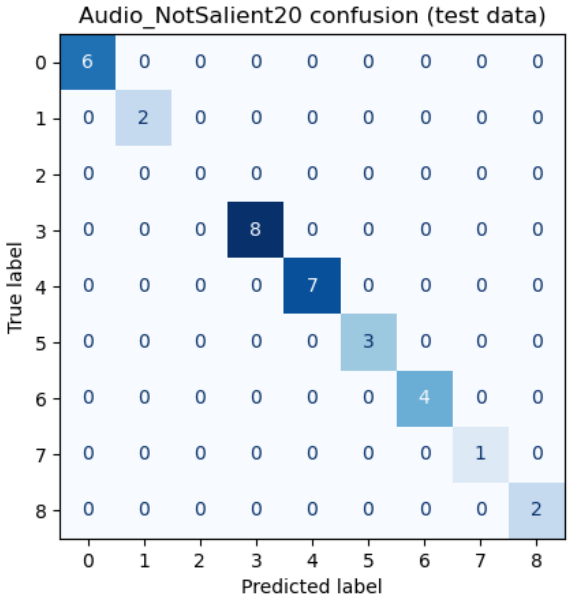

FIGURE 4.7 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_NotSalient20).

4.1.4 Modèles Audio (MFCC)

Les modèles audio utilisant les images MFCC ont plus de difficulté à converger que ceux utilisant les spectrogrammes Mel. Pour le modèle avec attention visuelle, la convergence est lente et le plafond n'est pas atteint avant l'époque 20 (fig. J.1), de même que pour le modèle utilisant les points aléatoires ne convergeant pas avant l'époque 21 (fig. J.4). Nous remarquons d'ailleurs que les courbes d'entraînement de ces modèles ont une caractéristique chaotique qui n'est pas présente pour les autres modèles.

Les images MFCC présentent le même problème que les spectrogrammes Mels, où chaque image pour un objet choisi un niveau de similarité visuelle élevé. La figure 4.8 équivalente met ce problème en évidence.

FIGURE 4.8 – Exemple des images MFCC générées pour l'objet #2 (Baril jaune). La variance visuelle entre chaque image est très limitée.

Puisque le même processus de sélection des sous-ensemble est en place pour tout les modèles, le même problème de classes sous-représentées est apparent dans les matrices de confusion aux figures 4.9 et 4.10 que pour les modèles audio avec spectrogrammes Mel. Ces matrices montrent toutefois une amélioration entre le modèle avec les points aléatoires et celui avec attention visuelle. Une instance de guitare acoustique (label 5) est incorrectement classifiée en tant que chaise (label 2) dans le modèle avec points aléatoires, tandis que cette erreur n'existe pas dans le modèle avec attention visuelle.

Nous remarquons aussi une amélioration au niveau des courbes ROC aux figures J.3 et J.6, ou le modèle avec attention visuelle obtient des scores F1 Macro et Micro de 0.9738 et 0.9697, tandis que le modèle avec les points aléatoire obtient des scores de 0.8444 et 0.9394 causés par une difficulté avec la classe de la guitare acoustique.

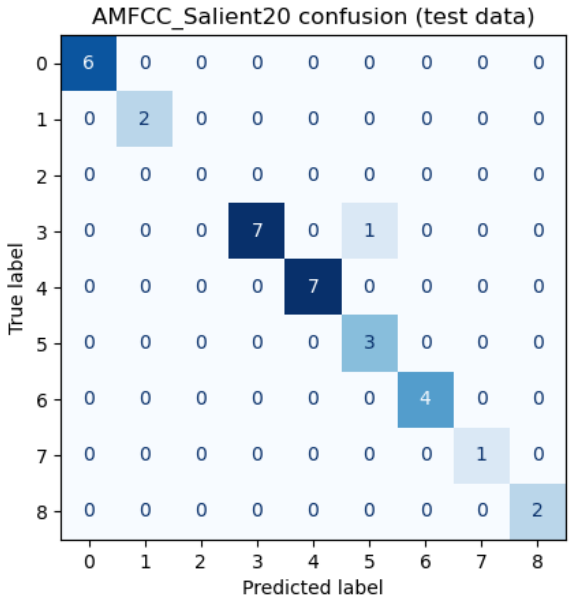

FIGURE 4.9 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_Salient20).

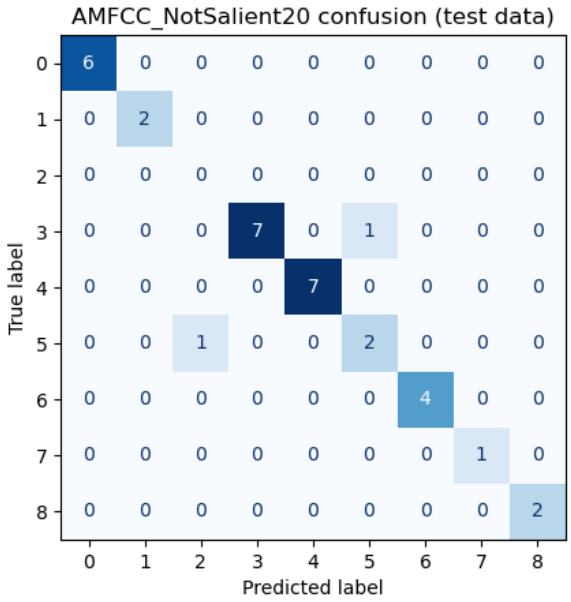

FIGURE 4.10 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20).

En analysant les performances des deux variantes de modèles audio, nous soupçonnons que ceux-ci effectuent un sur-apprentissage causé par une combinaison d'un trop petit nombre d'éléments et de la similarité entres les images associées à un objet quelconque. De plus, puisque le profil sonore d'un objet est principalement dicté par le matériel dont il est composé, la confusion entre différentes classes d'objets peut devenir un problème, comme pour l'erreur observée ici avec les classes *chaise* et *guitare* étant tout deux des objets en bois.

Nous décidons donc de développer deux ensembles de méta-modèles, dont un excluant l'audio et n'utilisant que les modèles Visuel et Tactile. Le deuxième ensemble inclut les modalités Visuel, Tactile et Audio (MFCC), dont la variante audio a été sélectionnée car l'effet du sur-apprentissage semble moins prononcé comparé à celui utilisant les spectrogrammes Mel.

4.2 Résultats des méta-modèles à modalités multiples

4.2.1 Modèles VT

Les méta-modèles combinés utilisant les données visuelles et tactiles montrent des résultats intéressants. Le modèle entraîné sur les données avec attention visuelle montre une performance pratiquement identique au modèle Visuel à modalité unique, avec 4 éléments sur 297 classifiés incorrectement (fig. 4.11) et des scores F1 Macro et Micro de 0.9852 et 0.9865 (fig. K.3). Il semble que lorsque les données visuelles sont concentrées sur les points d'attention, l'addition de données tactiles a un effet négligeable sur la performance du modèle.

Toutefois, pour le modèle utilisant les points choisis aléatoirement sur la surface des objets, nous observons une nette amélioration de la performance. En effet, le modèle classifie incorrectement que 6 éléments sur 297 (fig. 4.12), comparé aux 12 éléments pour le modèle Visuel équivalent. De plus, les scores F1 Macro et Micro sont de 0.9771 et 0.9798 (fig. K.6), se rapprochant du modèle combiné avec attention visuelle. Ceci viens confirmer notre hypothèse que la combinaison des modèles visuels et tactiles peut offrir une performance accrue, quoique l'effet n'est ici perceptible que lorsque les données visuelles sont de qualité réduite par rapport à celles utilisant l'attention visuelle.

Entre les deux modèles, nous remarquons aussi un fait intéressant au niveau de leurs matrices de confusion, où le modèle avec attention visuelle comporte un certain niveau de difficulté avec les guitares (label 5) étant incorrectement classifiées en tant que chaises (label 2) et télévisions (label 1) que le modèle avec points aléatoires ne présente pas. La difficulté de ce deuxième modèle semble plutôt de classifier incorrectement les chaises en commodes (label 3) ou en tables (label 4).

Malgré le manque d'amélioration de performance pour le modèle utilisant les données avec attention visuelle, nous observons quand même que celui-ci converge plus rapidement que le modèle avec les points aléatoires, où le premier converge à l'époque 8 (fig. K.1) tandis que le deuxième converge à l'époque 11 (fig. K.4). Ceci indique qu'il reste préférable d'utiliser les données provenant de points saillants.

4.2.2 Modèles VTA

Les méta-modèles combinés utilisant les données visuelles, tactiles et audio (MFCC) montrent une dégradation de la performance pour celui entraîné avec les points saillants,

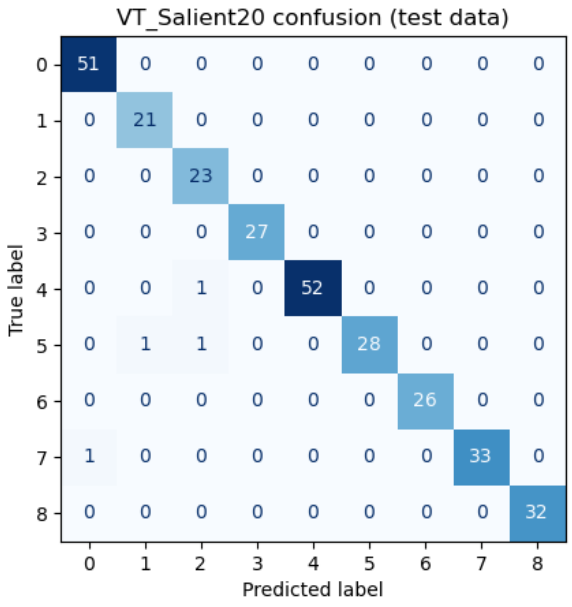

FIGURE 4.11 – Matrice de confusion sur les données de test du modèle combiné visuel/tactile (VT_Salient20).

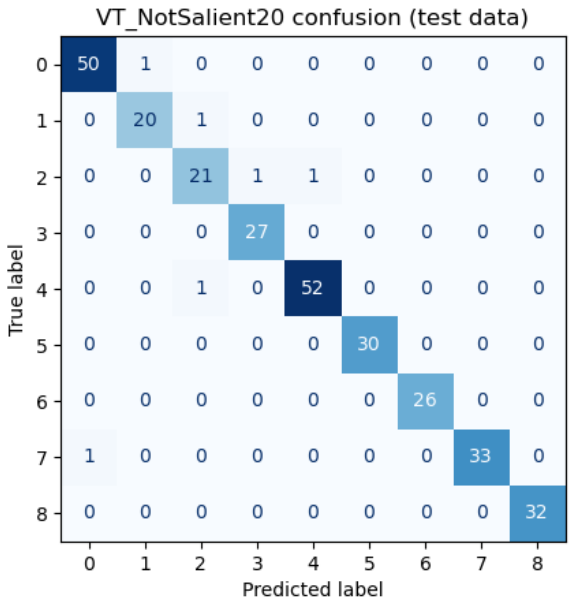

FIGURE 4.12 – Matrice de confusion sur les données de test du modèle combiné visuel/tactile (VT_NotSalient20).

classifiant incorrectement 14 éléments sur 297 (fig. 4.13). Les scores F1 Macro et Micro sont de 0.9468 et 0.9529 (fig. L.3), une chute marquée comparée au modèle équivalent utilisant seulement les modalités visuelle et tactile.

Le modèle utilisant les points choisis aléatoirement montre quant à lui une performance élevée, ne classifiant incorrectement que 3 éléments sur 297 (fig. 4.14) avec des scores F1 Macro et Micro de 0.9876 et 0.9899 (fig. L.6).

Entre les deux modèles, les erreurs proviennent surtout des classes 2, 3, 4 et 5, soit les chaises, commodes, tables et guitare, tous des objets en bois. Le modèle audio semble donc introduire de la confusion entre les objets partageant un matériel similaire. Pour ces modèles, nous observons toujours une converge légèrement plus rapide du modèle utilisant les points saillants contre celui avec les points aléatoires, où le premier converge à l'époque 10 (fig. L.1) tandis que le deuxième converge à l'époque 12 (fig. L.4).

Nous soupçonnons toujours que les problèmes identifiés avec les données audio sont présents dans ces méta-modèles combinés. Avec davantage de temps et de ressources, nous pourrions étudier et identifier la cause de ces problèmes de performance. La section 4.3 explore les résultats obtenus avec l'utilisation de différents points aléatoire,

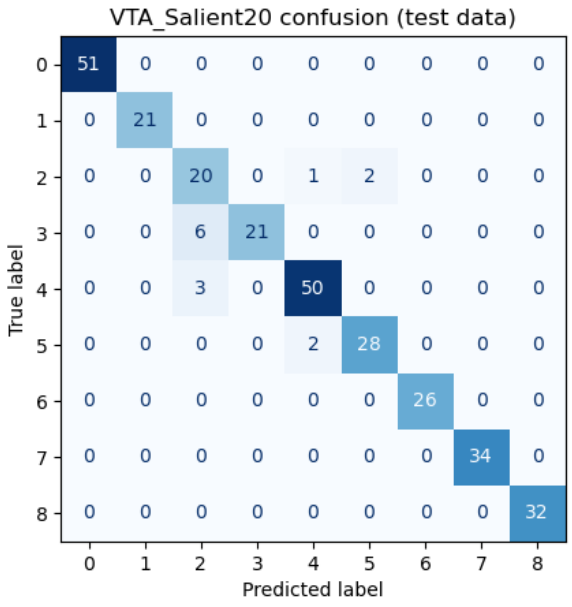

FIGURE 4.13 – Matrice de confusion sur les données de test du modèle combiné visuel/tactile (VTA_Salient20).

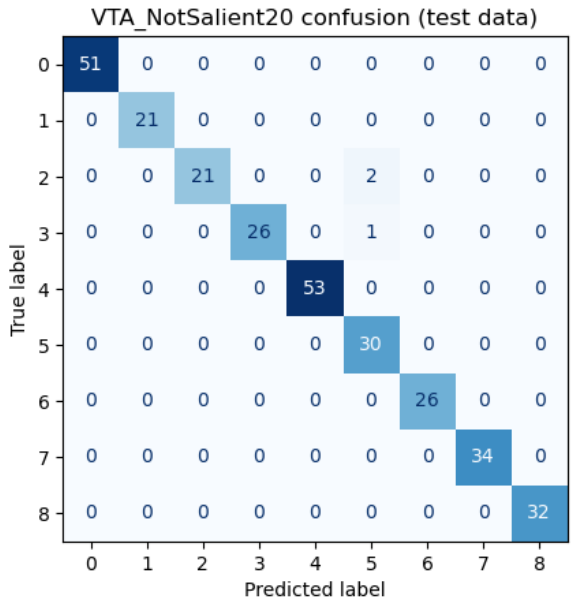

FIGURE 4.14 – Matrice de confusion sur les données de test du modèle combiné visuel/tactile (VTA_NotSalient20).

mettant en évidence un niveau de volatilité avec ces modèles qui rend difficile l'établissement d'une conclusion par rapport à leurs performances.

4.3 Récapitulatif des résultats

Le tableau 4.1 offre un survol des résultats discutés aux sections précédentes de ce chapitre ainsi que des résultats supplémentaires obtenus lors de l'entraînement des modèles *_NotSalient20_2* à *_NotSalient20_5*. Le tableau 4.2 donne la moyenne des résultats sur l'ensemble des cinq modèles entraînés à partir des points aléatoires.

Nous remarquons généralement l'augmentation de la vitesse de convergence lors de l'entraînement pour les modèles employant les points saillants. En comparant les moyennes sur les modèles utilisant les points aléatoires, cette conclusion reste vraie pour tous les modèles sauf ceux utilisant la modalité audio avec spectrogrammes Mel, où les modèles aléatoires convergence généralement plus rapidement.

Par rapport au taux d'erreur et aux scores F1 Macro et Micro, le verdict est clair pour la modalité visuelle, où la plupart des modèles aléatoires ont une performance réduite

TABLE 4.1 – Résultats de l'entraînement des différents modèles. La colonne « Cv. » indique le nombre approximatif d'époques avant la convergence du modèle lors de l'étape d'entraînement. Les colonnes « Macro » et « Micro » représentent les scores F1 respectifs.

Modalité	Ensemble de points	Erreur	Macro	Micro	Cv.
Visuel	Saillants	0.0135	0.9837	0.9865	15
	Aléatoires # 1 à 20	0.0404	0.9565	0.9596	21
	Aléatoires # 21 à 40	0.0135	0.9856	0.9865	22
	Aléatoires # 41 à 60	0.0236	0.9687	0.9764	15
	Aléatoires # 61 à 80	0.0168	0.9824	0.9832	22
	Aléatoires # 81 à 100	0.0236	0.9731	0.9764	22
Tactile	Saillants	0.0471	0.9463	0.9529	13
	Aléatoires # 1 à 20	0.0539	0.9441	0.9461	13
	Aléatoires # 21 à 40	0.0707	0.9221	0.9293	14
	Aléatoires # 41 à 60	0.0370	0.9615	0.9630	15
	Aléatoires # 61 à 80	0.0404	0.9587	0.9596	14
	Aléatoires # 81 à 100	0.0303	0.9689	0.9697	15
Audio (Mels)	Saillants	0.0000	1.0000	1.0000	8
	Aléatoires # 1 à 20	0.0000	1.0000	1.0000	5
	Aléatoires # 21 à 40	0.0000	1.0000	1.0000	8
	Aléatoires # 41 à 60	0.0000	1.0000	1.0000	5
	Aléatoires # 61 à 80	0.0000	1.0000	1.0000	6
	Aléatoires # 81 à 100	0.0303	0.8667	0.9697	5
Audio (MFCC)	Saillants	0.0303	0.9738	0.9697	20
	Aléatoires # 1 à 20	0.0606	0.8444	0.9394	21
	Aléatoires # 21 à 40	0.0909	0.8254	0.9091	25
	Aléatoires # 41 à 60	0.0303	0.9833	0.9697	24
	Aléatoires # 61 à 80	0.0909	0.8558	0.9091	23
	Aléatoires # 81 à 100	0.0303	0.9833	0.9697	26
Visuel/Tactile	Saillants	0.0135	0.9852	0.9865	8
	Aléatoires # 1 à 20	0.0202	0.9771	0.9798	11
	Aléatoires # 21 à 40	0.0168	0.9802	0.9832	13
	Aléatoires # 41 à 60	0.0067	0.9914	0.9933	12
	Aléatoires # 61 à 80	0.0067	0.9933	0.9933	15
	Aléatoires # 81 à 100	0.0303	0.9865	0.9697	13
V+T/Audio (MFCC)	Saillants	0.0471	0.9468	0.9529	10
	Aléatoires # 1 à 20	0.0101	0.9876	0.9899	12
	Aléatoires # 21 à 40	0.0471	0.9128	0.9529	14
	Aléatoires # 41 à 60	0.0303	0.9662	0.9697	14
	Aléatoires # 61 à 80	0.0572	0.9272	0.9428	15
	Aléatoires # 81 à 100	0.0471	0.9473	0.9529	19

TABLE 4.2 – Résultats de l'entraînement des différents modèles avec la moyenne des cinq modèles aléatoires provenant du tableau 4.1.

Modalité	Ensemble de points	Erreur	Macro	Micro	Cv.
Visuel	Saillants	0.0135	0.9837	0.9865	15
	Moyenne Aléatoires	0.0236	0.9733	0.9764	20
Tactile	Saillants	0.0471	0.9463	0.9529	13
	Moyenne Aléatoires	0.0465	0.9511	0.9535	14
Audio (Mels)	Saillants	0.0000	1.0000	1.0000	8
	Moyenne Aléatoires	0.0061	0.9733	0.9939	6
Audio (MFCC)	Saillants	0.0303	0.9738	0.9697	20
	Moyenne Aléatoires	0.0606	0.8984	0.9394	24
Visuel/Tactile	Saillants	0.0135	0.9852	0.9865	8
	Moyenne Aléatoires	0.0161	0.9857	0.9839	13
V+T/Audio (MFCC)	Saillants	0.0471	0.9468	0.9529	10
	Moyenne Aléatoires	0.0384	0.9482	0.9616	15

par rapport à celui employant les points saillants. La modalité audio (MFCC) présente un profil similaire, mais la variance élevée entre les modèles aléatoires remet en question cette observation, particulièrement avec ceux utilisant les points # 41 à 60 et # 81 à 100 ayant une performance très élevée et ceux utilisant les points # 21 à 40 et # 61 à 80 ayant une performance plus réduite. Pour les autres modalités, la performance est en moyenne équivalente entre les différents ensembles de points, autre que pour la vitesse de convergence, indiquant donc qu'il reste préférable d'utiliser les points saillants.

Comparant entre les modalités, le modèle Visuel a généralement une meilleure performance que le modèle Tactile. La combinaison de ces deux modalités permet d'améliorer considérablement la performance dans le cas où les points aléatoires sont utilisés, rendant les deux ensembles comparables. Toutefois, l'ajout de la modalité audio (MFCC) réduit la performance des modèles.

Les modèles Audio utilisant les spectrogrammes Mel obtiennent des scores parfait et convergent rapidement, mais nous soupçonnons que la qualité de l'ensemble de données ne nous permet pas d'établir de conclusions par rapport à ceux-ci. Nous observons toutefois une amélioration de la performance selon les points choisis pour les modèles employant les « *Mel-frequency cepstral coefficients* », mais davantage de données seraient nécessaire pour confirmer toute conclusion par rapport à cette observation. De plus, les matrices de confusion pour les méta-modèles incorporant ces données (figures L.2, L.5,

L.8, L.11, L.14 et L.17) montrent une tendance à classifier incorrectement les objets ayant un matériel similaire, indiquant que l'inclusion de ces données n'est potentiellement pas appropriée dans le contexte de la classification d'objets sur leur forme visuelle.

Chapitre 5

Conclusion et travaux futurs

L'objectif principal de cet essai était d'explorer le processus d'entraînement de modèles d'apprentissage profond utilisant des données visuelles, tactiles et audio et d'évaluer leurs performances selon l'utilisation de points avec un attention visuelle identifiés dans les travaux de Ghazal Rouhafzay [2].

En étudiant les options disponibles pour l'acquisition de données, nous identifions qu'il y a un manque de données facilement disponibles sur le Web pour les modalités tactiles et audio, surtout dans notre cas spécifique où nous désirons que les données soient inter-connectées. De ce, nous identifions *ObjectFolder 2.0* [11] comme outil afin de générer un ensemble de données à partir des points sur un objet. Toutefois, l'exploration de cet outil a demandé un effort de documentation considérable ainsi que plusieurs changements à son fonctionnement et ses interfaces afin de répondre à notre cas d'utilisation.

Une fois un processus en place pour générer notre base de données à partir des points d'intérêt sur les objets disponibles, nous avons pu entraîner une série de modèles afin d'évaluer la véracité de nos hypothèses. Nous avons pu montrer que les modèles entraînés sur les données visuelles simulées maintiennent une bonne performance, que l'utilisation de points visuellement saillants permettent d'améliorer l'exactitude des modèle visuels de même que la vitesse de convergence de tous les modèles et que la combinaison des données visuelles et tactiles en un modèle conjoint permet de réduire l'impact de l'utilisation de points aléatoires sur la performance des modèles.

Toutefois, l'ajout de données audio au modèles conjoints entraîne une perte de performance causée par l'introduction d'un niveau de confusion entre les objets partageant une composition matérielle similaire, tel que les objets en bois. Comme travaux futurs, il serait important d'approfondir notre compréhension de cette modalité en effectuant

des tests plus détaillés. Il serait aussi intéressant d'inclure la composante « Matériel » des objets et utiliser à la place les données audio pour la détection de la composition des objets.

De plus, nous devrions agrandir considérablement le nombre d'objets, de classes d'objets et de points par objets afin de s'assurer que nos conclusions restent valides lorsque nous travaillons avec des modèles plus volumineux. Nous pourrions aussi retravailler la méthode par laquelle les données audio sont produites en créant une couche d'augmentation ajoutant des sons ambiants aux données produites par *ObjectFolder* afin de simuler un niveau de pollution sonore, permettant potentiellement d'éliminer le problème de données visuellement trop similaires pour cette modalité.

Finalement, nous soulignons que ce projet emploie uniquement des données simulées et sert donc d'une démonstration de la faisabilité des techniques dont nous avons explorées, soulevant des questions quant à la généralisation de nos modèles sur des données réelles. Rappelons toutefois la difficulté d'obtenir des données multimodales réelles, dont un éventuel travail futur devra consacrer des ressources considérables afin d'obtenir en quantité raisonnable. Bien que le projet *ObjectFolder Real* [12] existe (voir la description à la section 2.4.3), nous n'avons aucun contrôle sur les points exacts sélectionnés sur les objets, rendant son utilité limitée dans le contexte de ce travail. Pour rappel, *ObjectFolder 2.0* a été choisi car nous pouvons, étant un outil de simulation, sélectionner des points de vue arbitraires pour répondre aux objectifs et aux hypothèses de cet essai.

Annexe A

Intégrité académique et déclaration de non-usage de l'IA générative

Nous déclarons que le contenu de cet essai est original et ne constitue pas un plagiat, que toutes les sources (textes, figures, idées, outils) sont référencées et qu'aucun outil d'IA générative a été utilisé lors de la rédaction de l'essai et du développement de la méthodologie.

Annexe B

Téléchargement d'ObjectFolder

Le projet est téléchargé localement à l'aide du *git CLI* (*Command Line Interface*) avec la commande `git clone` (figure B.1), le téléchargement peut aussi être effectué à l'aide de l'utilitaire en ligne sur le site web de GitHub.

```
D:\>git clone https://github.com/rhgao/ObjectFolder.git
Cloning into 'ObjectFolder'...
remote: Enumerating objects: 69, done.
remote: Counting objects: 100% (55/55), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 69 (delta 11), reused 46 (delta 10), pack-reused 14 (from
↪  1)
Receiving objects:  98% (68/69), 19.91 MiB | 9.95 MiB/s
Receiving objects: 100% (69/69), 24.45 MiB | 9.99 MiB/s, done.
Resolving deltas: 100% (12/12), done.
```

FIGURE B.1 – Téléchargement d'*ObjectFolder 2.0* à l'aide de l'utilitaire `git clone`.

Annexe C

Image représentative des objets disponibles

FIGURE C.1 – Image représentative des 11 objets utilisés dans cet essai. En ordre, de gauche à droite et haut en bas, les objets #2, #4, #5, #6, #7, #8, #9, #10, #53, #55 et #77.

Annexe D

Exploration initiale de la technique de rendu entre Blender et ObjectFolder

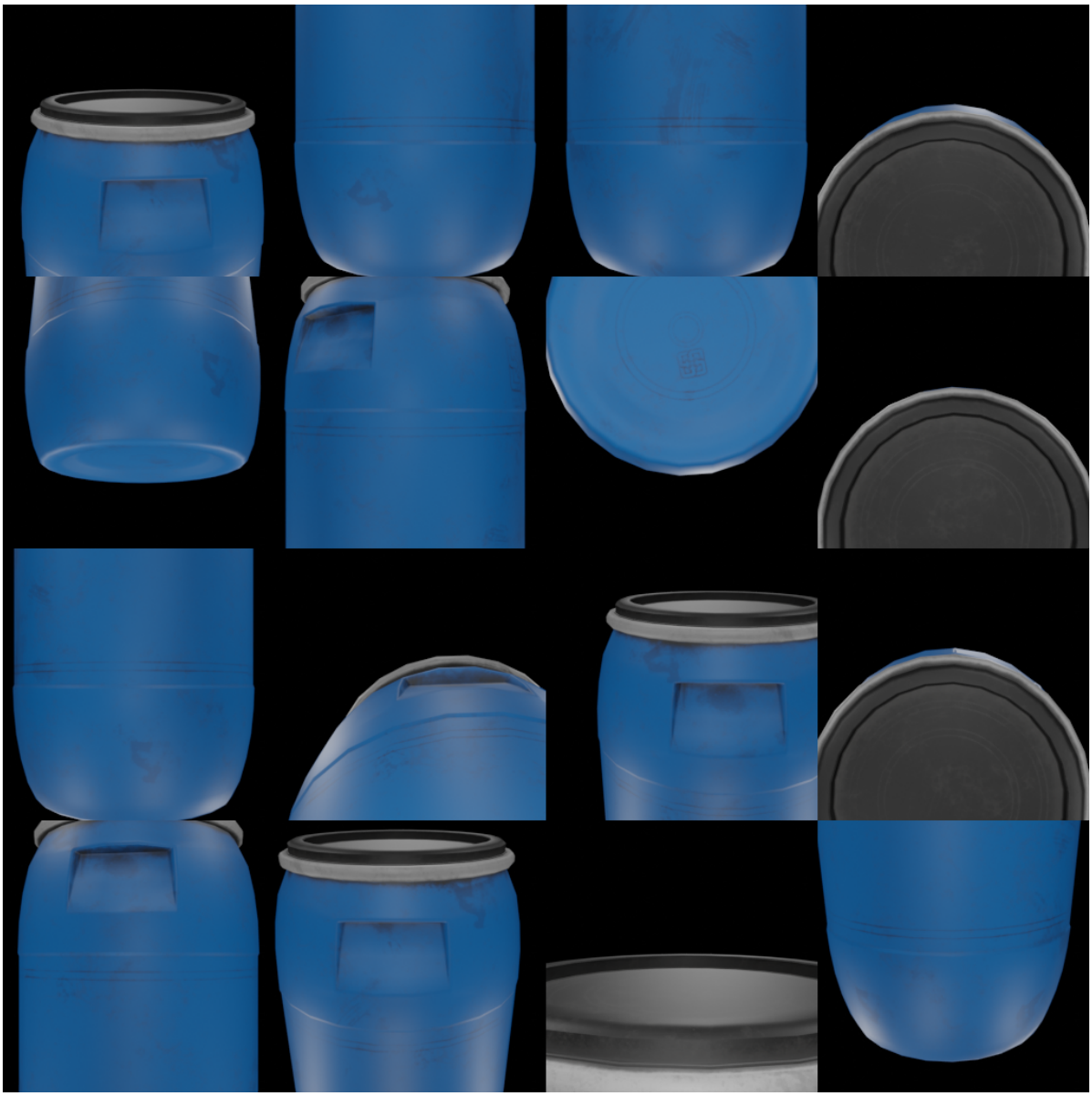

FIGURE D.1 – Rendu de 16 points d'intérêts sur l'objet #10 (Baril bleu) avec Blender à l'aide du package *bpy*. Ces images servent de point de repère à répliquer à la figure D.2.

FIGURE D.2 – Rendu de 16 points d'intérêts sur l'objet #10 (Baril bleu) avec la modalité visuelle d'ObjectFolder en suivant le plus près possible la technique développée à la figure D.1.

Annexe E

Définitions des fichiers de méta-données

Table E.1: Description des colonnes dans le fichier de pré-traitement.

Nom de la colonne	Description
obj_num	Le numéro de l'objet dans la base de données d'*ObjectFolder 2.0*.
is_blender_model	Vrai si l'objet a été créé dans *Blender*, faux si créé avec *Katamari*. Utilisé pour déterminer les rotations à appliquer aux objets lors de l'étape de traitement avec *ObjectFolder 2.0*.
is_salient	Vrai si le point provient du travail de Ghazal Rouhafzay, faux si le point a été choisi aléatoirement sur l'objet.
obj_scale	Distance de la diagonale de la « *Bounding Box* » englobant l'objet. $\sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2 + (z_{max} - z_{min})^2}$
obj_scale_2	Distance entre le minimum et le maximum, tous axes confondus. $\max(x_{max}, y_{max}, z_{max}) - \min(x_{min}, y_{min}, z_{min})$
obj_scale_3	Maximum de la distance entre les points minimums et maximums de chaque axes, dont l'inverse est utilisé comme facteur de normalisation pour modifier la taille de l'objet à un cube unité. $\max(x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min})$
point_vert_dist	La distance entre les variables *point* et *vert* plus bas. Devrait toujours être égale à zéro.

Suite à la page suivante

Table E.1: Description des colonnes dans le fichier de pré-traitement. (Suite)

Nom de la colonne	Description
min_x	La valeur minimale observée pour un point de l'objet sur l'axe X, x_{min}.
min_y	La valeur minimale observée pour un point de l'objet sur l'axe Y, y_{min}.
min_z	La valeur minimale observée pour un point de l'objet sur l'axe Z, z_{min}.
max_x	La valeur maximale observée pour un point de l'objet sur l'axe X, x_{max}.
max_y	La valeur maximale observée pour un point de l'objet sur l'axe Y, y_{max}.
max_z	La valeur maximale observée pour un point de l'objet sur l'axe Z, z_{max}.
cen_x	La valeur du point centroïde de l'objet sur l'axe X, suivant la moyenne calculée par $x_{cen} = \frac{1}{num_x} \sum_1^{num_x} x_i$.
cen_y	La valeur du point centroïde de l'objet sur l'axe Y, suivant la moyenne calculée par $y_{cen} = \frac{1}{num_y} \sum_1^{num_y} y_i$.
cen_z	La valeur du point centroïde de l'objet sur l'axe Z, suivant la moyenne calculée par $x_{cen} = \frac{1}{num_z} \sum_1^{num_z} z_i$.
point_x	Le point ciblé sur l'object, axe X.
point_y	Le point ciblé sur l'object, axe Y.
point_z	Le point ciblé sur l'object, axe Z.
vert_x	Le point le plus près à la cible, axe X. Devrait toujours être égal à *point_ x*.
vert_y	Le point le plus près à la cible, axe Y. Devrait toujours être égal à *point_ y*.
vert_z	Le point le plus près à la cible, axe Z. Devrait toujours être égal à *point_ z*.

Suite à la page suivante

Table E.1: Description des colonnes dans le fichier de pré-traitement. (Suite)

Nom de la colonne	Description
normal_x	Le valeur sur l'axe X de la normale au point ciblé, suivant le code à la figure 3.11.
normal_y	Le valeur sur l'axe Y de la normale au point ciblé.
normal_z	Le valeur sur l'axe Z de la normale au point ciblé.

Table E.2: Description des colonnes dans le fichier de méta-données pour la base de données Visuelle.

Nom de la colonne	Description
obj_id	Le numéro de l'objet dans la base de données d'*ObjectFolder 2.0*.
run_nbr	Le numéro de l'ordre d'exécution de l'image lors de sa création, avant l'application de techniques d'augmentation de données.
sub_run_nbr	Le numéro de l'ordre d'exécution de l'image lors de sa création, après l'application de techniques d'augmentation de données.
sub_run_nbr2	Le numéro de l'ordre d'exécution de l'image lors de sa création à l'intérieur d'une série d'application de techniques d'augmentation de données.
path	L'emplacement de l'image et son nom dans la base de données.
is_blender	Voir la colonne équivalente au tableau E.1.
is_salient	Voir la colonne équivalente au tableau E.1.
x	La composante sur l'axe X du point ciblé sur l'objet, avant normalisation et rotation.
y	La composante sur l'axe Y du point ciblé sur l'objet, avant normalisation et rotation.
z	La composante sur l'axe Z du point ciblé sur l'objet, avant normalisation et rotation.

Suite à la page suivante

Table E.2: Description des colonnes dans le fichier de méta-données pour la base de données Visuelle. (Suite)

Nom de la colonne	Description
rot_phi	Augmentation de données, rotation horizontale en degrés appliquée au vecteur normal lors du calcul de la position de la caméra.
rot_theta	Augmentation de données, rotation verticale en degrés appliquée au vecteur normal lors du calcul de la position de la caméra.
light_x	La composante sur l'axe X de l'orientation de la lumière sur l'objet.
light_y	La composante sur l'axe Y de l'orientation de la lumière sur l'objet.
light_z	La composante sur l'axe Z de l'orientation de la lumière sur l'objet.

Table E.3: Description des colonnes dans le fichier de méta-données pour la base de données Tactile.

Nom de la colonne	Description
obj_id	Le numéro de l'objet dans la base de données d'*ObjectFolder 2.0*.
run_nbr	Le numéro de l'ordre d'exécution de l'image lors de sa création, avant l'application de techniques d'augmentation de données.
sub_run_nbr	Le numéro de l'ordre d'exécution de l'image lors de sa création, après l'application de techniques d'augmentation de données.
sub_run_nbr2	Le numéro de l'ordre d'exécution de l'image lors de sa création à l'intérieur d'une série d'application de techniques d'augmentation de données.
path	L'emplacement de l'image et son nom dans la base de données.

Suite à la page suivante

Table E.3: Description des colonnes dans le fichier de méta-données pour la base de données Tactile. (Suite)

Nom de la colonne	Description
is_blender	Voir la colonne équivalente au tableau E.1.
is_salient	Voir la colonne équivalente au tableau E.1.
x	La composante sur l'axe X du point ciblé sur l'objet, avant normalisation et rotation.
y	La composante sur l'axe Y du point ciblé sur l'objet, avant normalisation et rotation.
z	La composante sur l'axe Z du point ciblé sur l'objet, avant normalisation et rotation.
gel_t	La composante *theta* θ utilisée par le capteur *GelSight* simulé.
gel_p	La composante *phi* ϕ utilisée par le capteur *GelSight* simulé.
gel_d	La profondeur utilisée par le capteur *GelSight* simulé.

Table E.4: Description des colonnes dans le fichier de méta-données pour la base de données Audio.

Nom de la colonne	Description
obj_id	Le numéro de l'objet dans la base de données d'*ObjectFolder 2.0*.
run_nbr	Le numéro de l'ordre d'exécution du fichier sonore lors de sa création.
path	L'emplacement du fichier sonore et son nom dans la base de données.
is_blender	Voir la colonne équivalente au tableau E.1.
is_salient	Voir la colonne équivalente au tableau E.1.
x	La composante sur l'axe X du point ciblé sur l'objet, avant normalisation et rotation.

Suite à la page suivante

Table E.4: Description des colonnes dans le fichier de méta-données pour la base de données Audio. (Suite)

Nom de la colonne	Description
y	La composante sur l'axe Y du point ciblé sur l'objet, avant normalisation et rotation.
z	La composante sur l'axe Z du point ciblé sur l'objet, avant normalisation et rotation.
frc_x	La composante sur l'axe X de la force de l'impact sur l'objet ayant généré le son.
frc_y	La composante sur l'axe Y de la force de l'impact sur l'objet ayant généré le son.
frc_z	La composante sur l'axe Z de la force de l'impact sur l'objet ayant généré le son.

```
obj_id,class_id
2,0
4,1
5,2
6,3
7,4
8,5
9,4
10,0
53,6
55,7
77,8
```

FIGURE E.1 – Fichier CSV de méta-données indiquant à quelle classe appartient chaque objet.

```
class_id,class_name
0,barrel
1,television
2,chair
3,commode
4,table
5,acoustic_guitar
6,bowl
7,tray
8,file_sorter
```

FIGURE E.2 – Fichier CSV de méta-données indiquant le nom pour chaque classe d'objet.

Annexe F

Fichiers et code relatifs à l'entraînement des modèles Vision, Toucher et Audio

```
name: Essay-MLTrain
channels:
    - conda-forge
    - defaults
    - nvidia
dependencies:
    - pip == 26.0.1
    - python == 3.10.*
    - numpy < 2
    - pandas
    - scipy
    - matplotlib
    - librosa
    - scikit-learn
    - scikit-image
    - cudatoolkit == 11.6
    - cudnn < 9
    - tensorboard = 2.8
    - setuptools < 82
    - protobuf < 5
    - pip:
        - tensorflow == 2.8.4
```

FIGURE F.1 – Fichier `environment.yml` pour la création des modèles Vision, Tactile et Audio à partir des données générées à l'aide d'ObjectFolder 2.0. Les numéros de versions des différents paquets ont été sélectionnés pour que l'environnement soit compatible avec le même système utilisé pour exécuter ObjectFolder 2.0.

Annexe G

Résultats de l'entraînement des modèles visuels

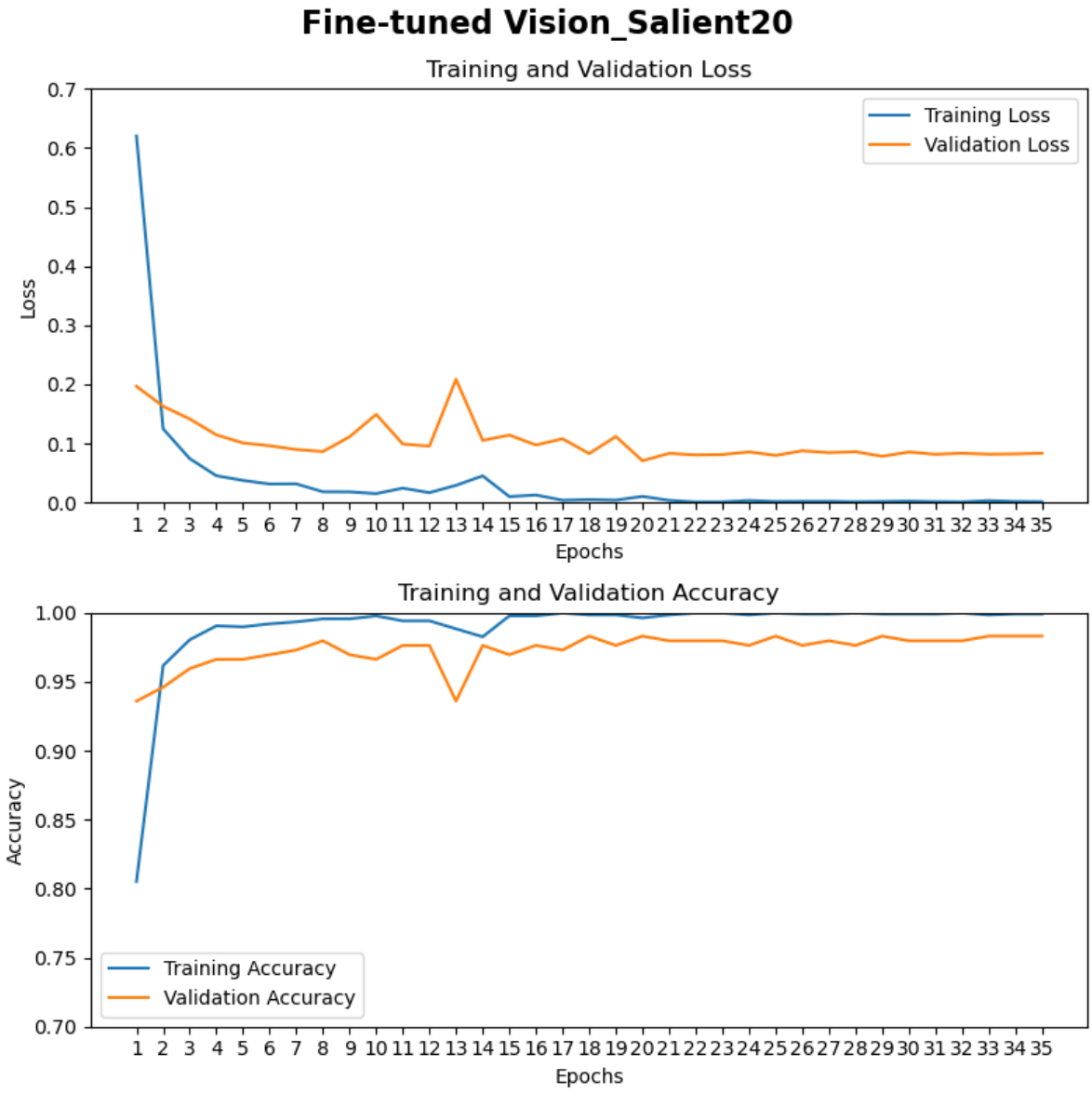

FIGURE G.1 – Historique de l'entraînement du modèle visuel utilisant les 20 points saillants par objet (Vision_Salient20).

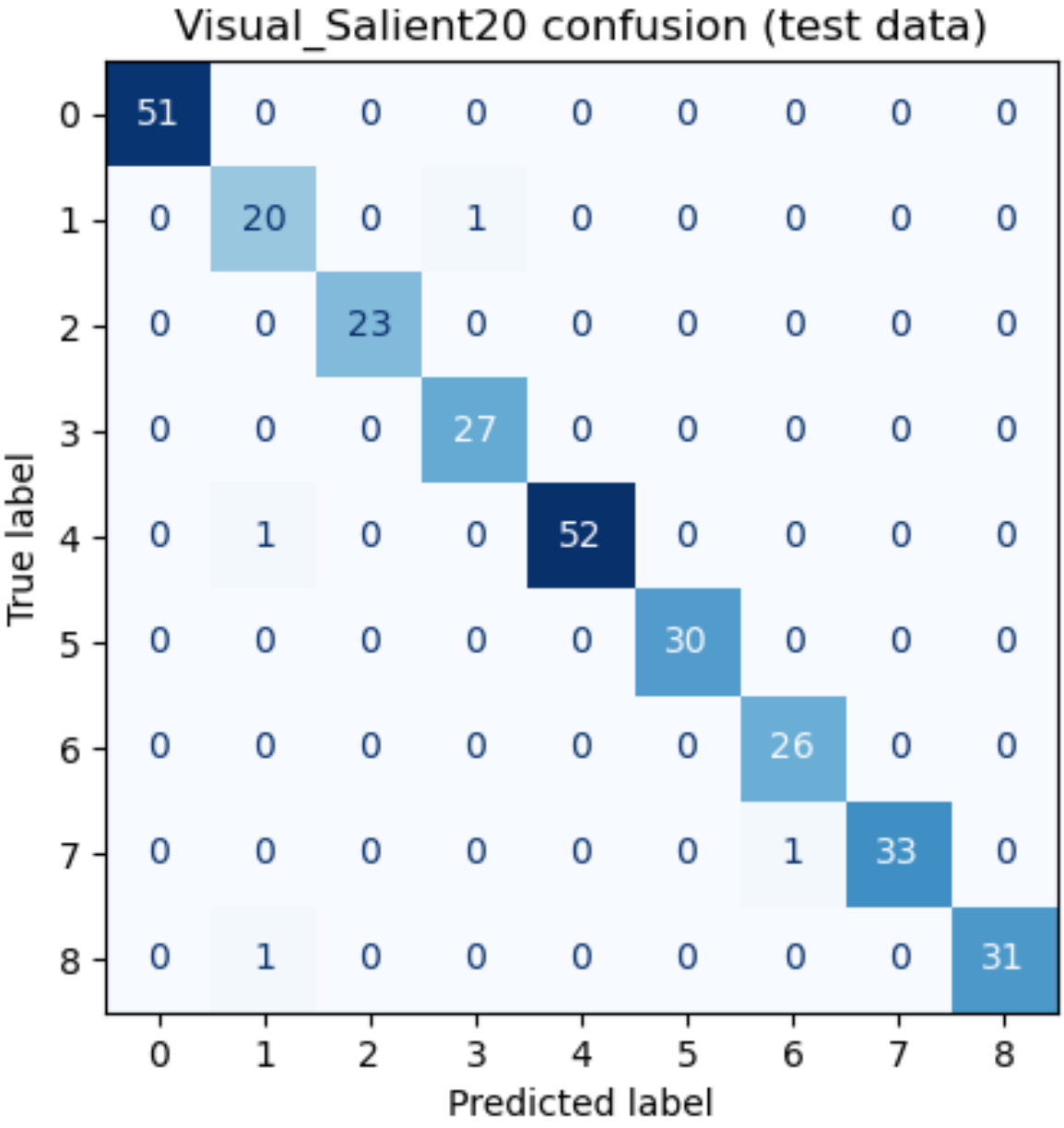

FIGURE G.2 – Matrice de confusion sur les données de test du modèle visuel (Vision_Salient20).

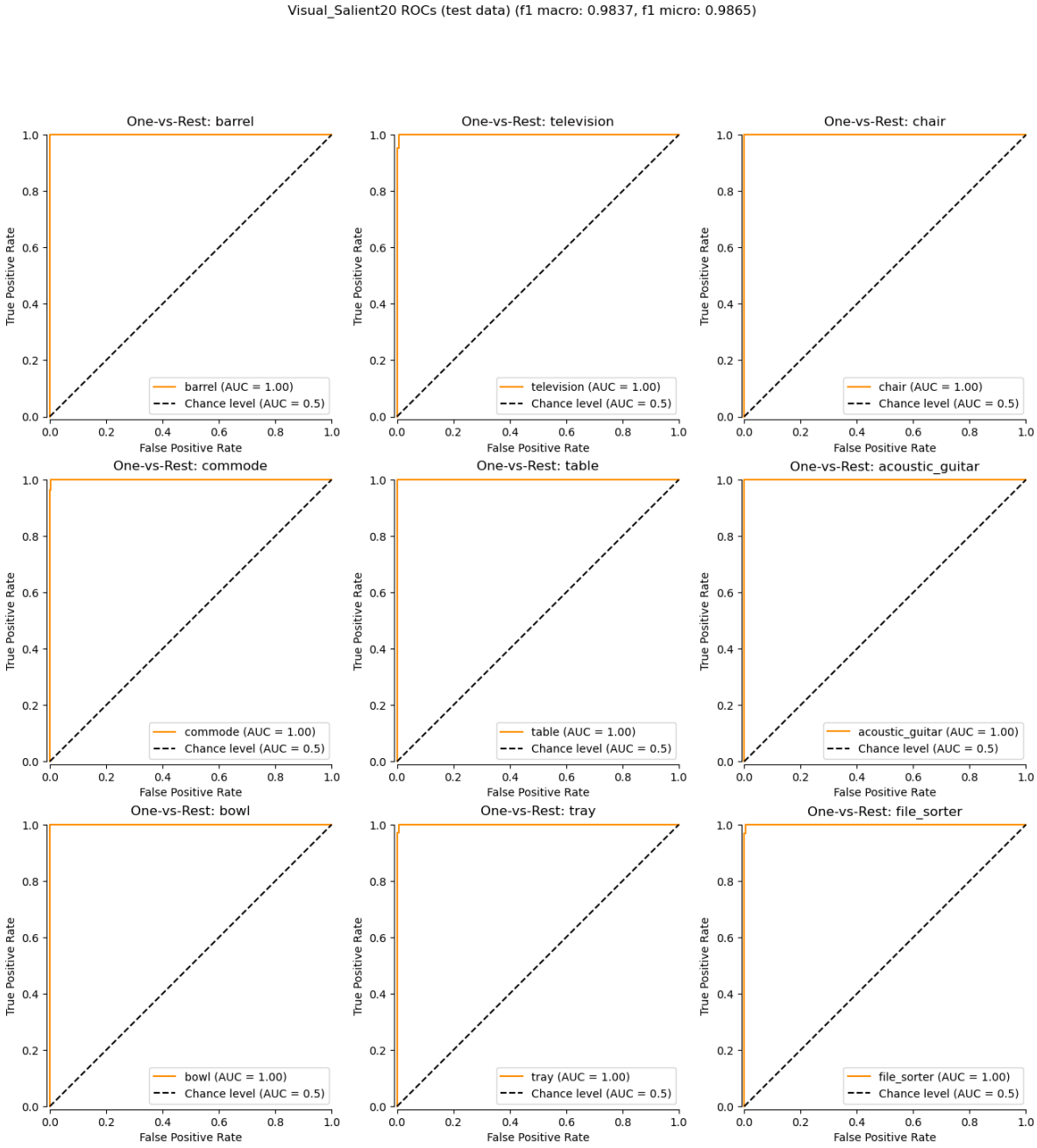

FIGURE G.3 – Courbes ROC et F1 Scores sur les données de test du modèle visuel (Vision_Salient20).

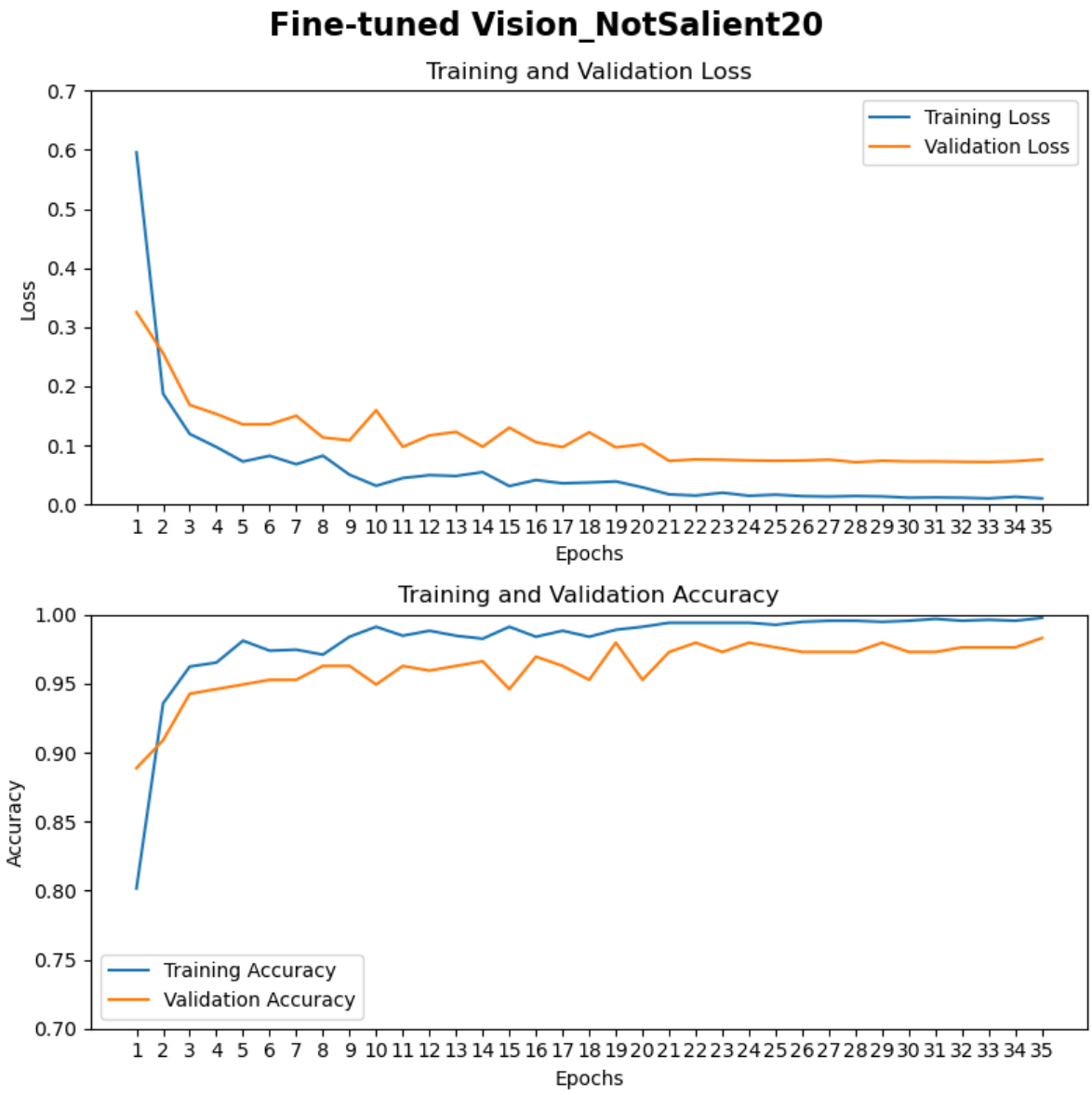

FIGURE G.4 – Historique de l'entraînement du modèle visuel utilisant les 20 premiers points sélectionnés aléatoirement par objet (Vision_NotSalient20).

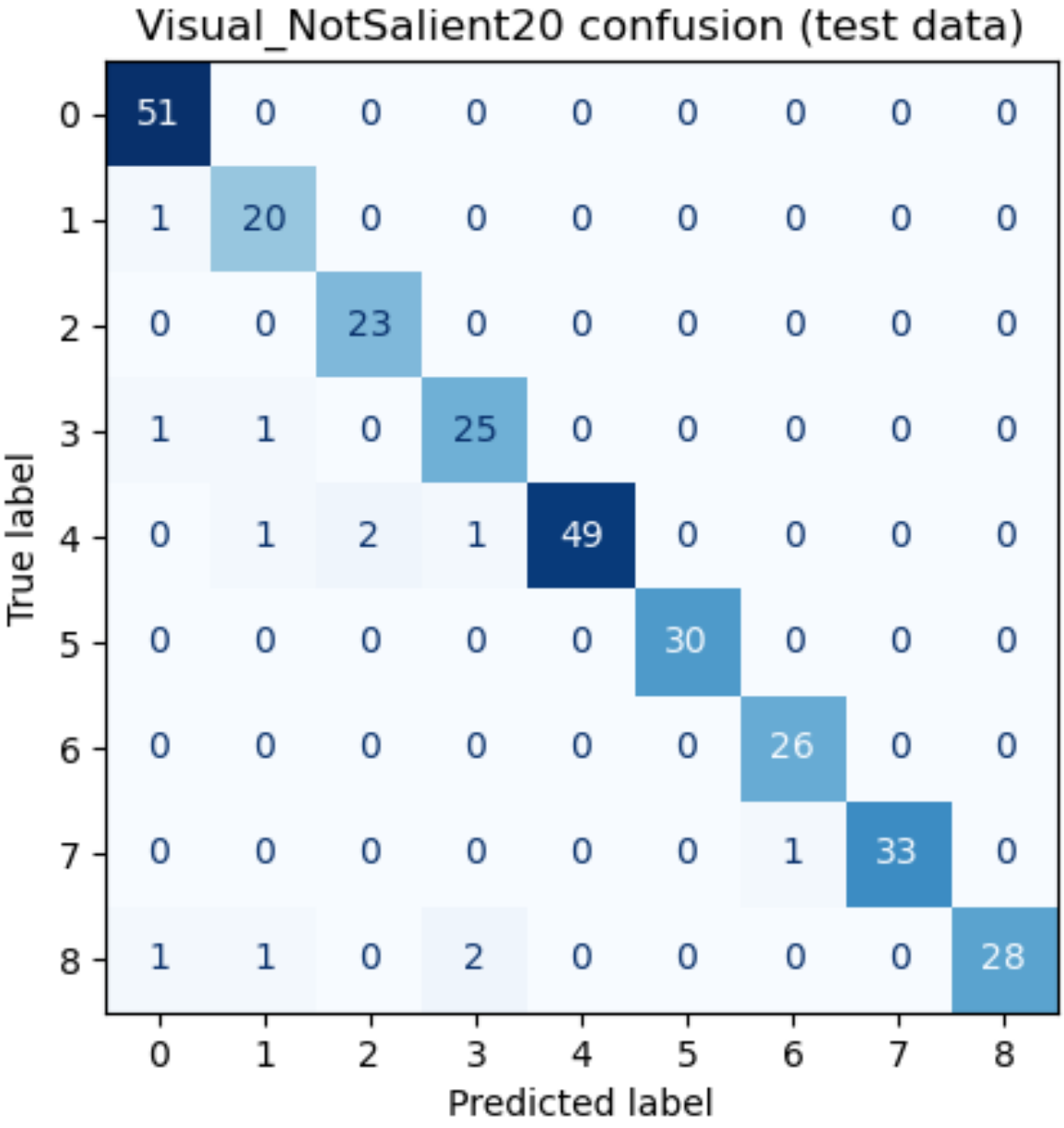

FIGURE G.5 – Matrice de confusion sur les données de test du modèle visuel (Vision_NotSalient20).

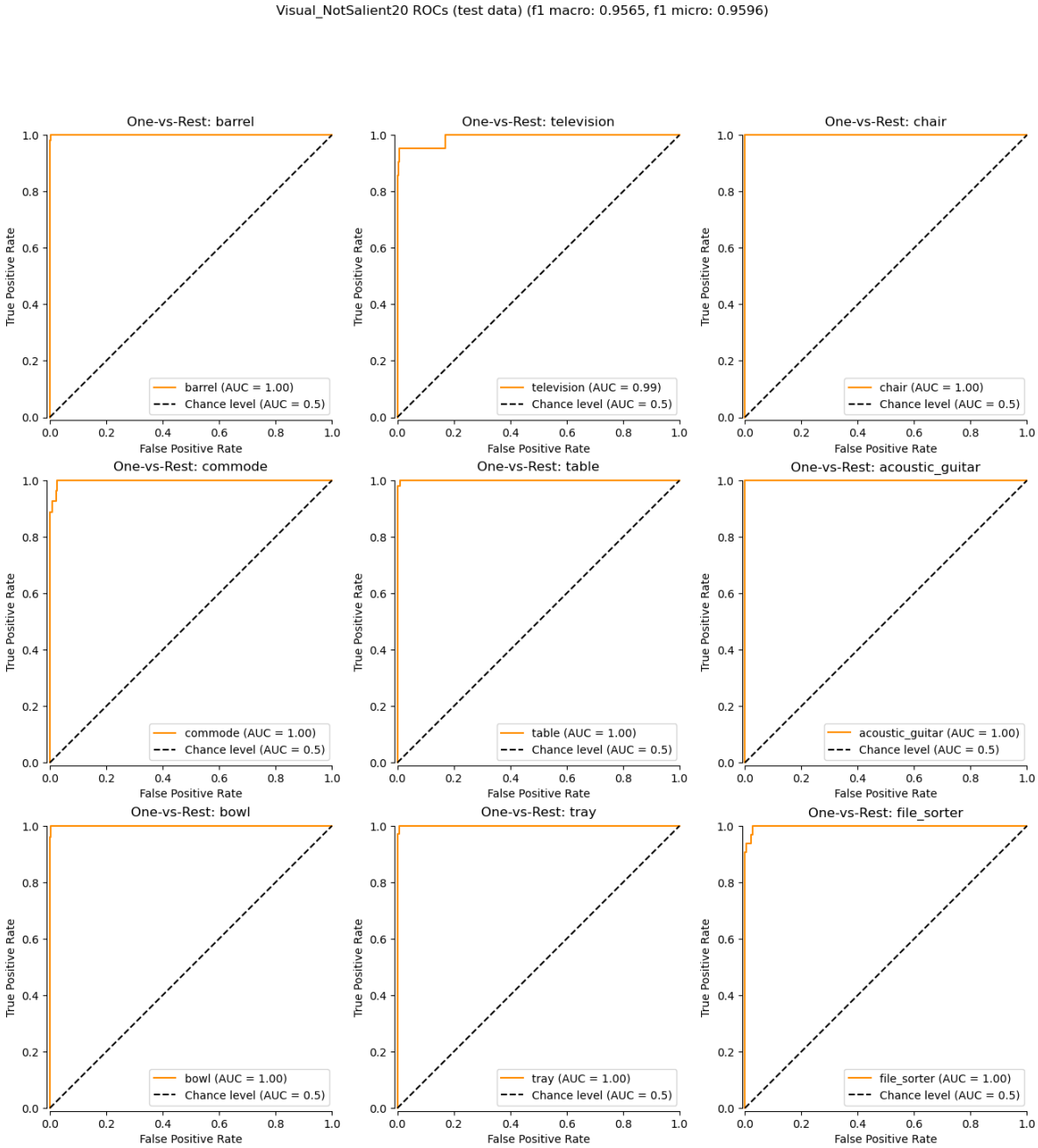

FIGURE G.6 – Courbes ROC et F1 Scores sur les données de test du modèle visuel (Vision_NotSalient20).

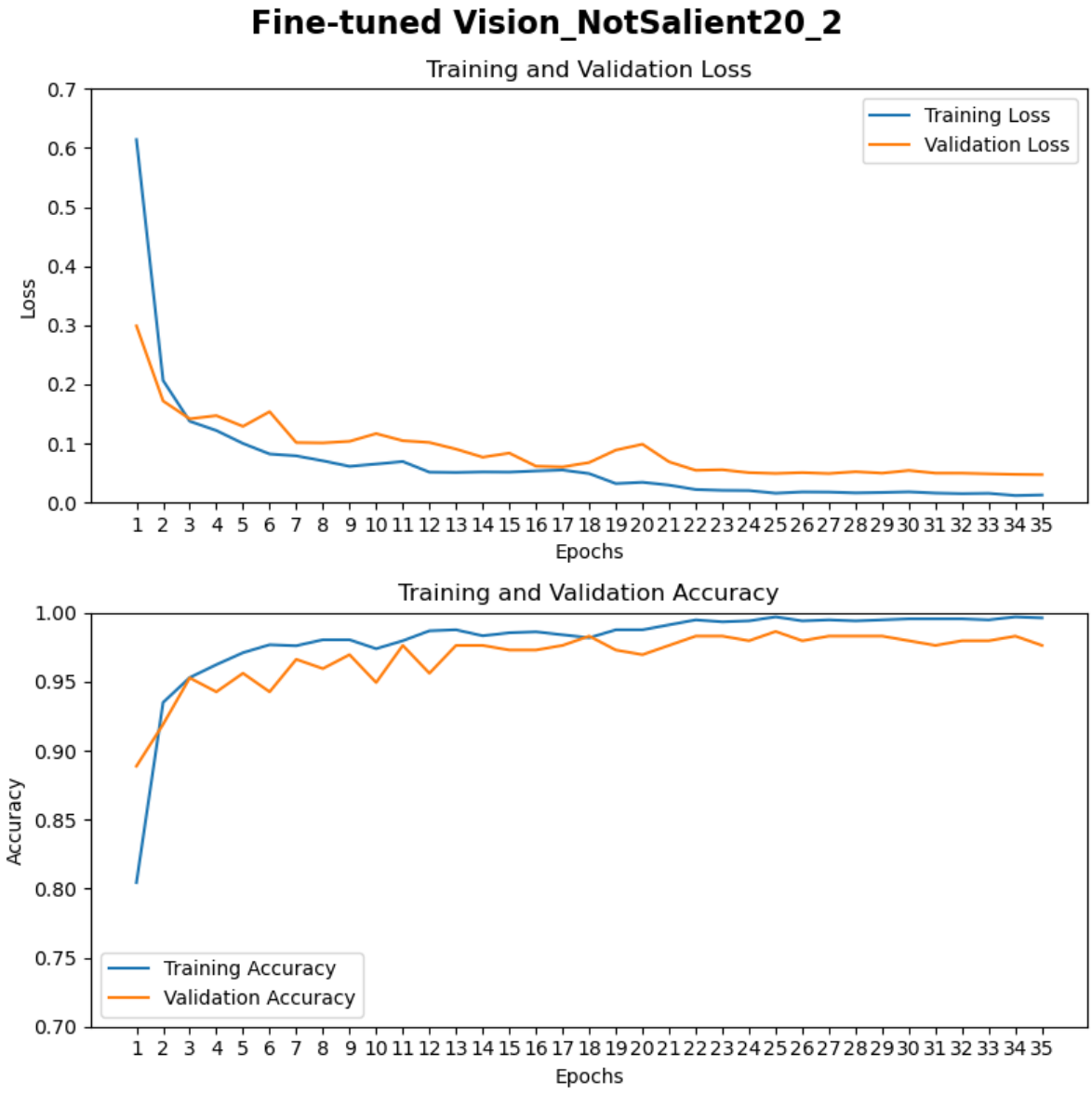

FIGURE G.7 – Historique de l'entraînement du modèle visuel utilisant les 20 points sélectionnés aléatoirement (# 21 à 40) par objet (Vision_NotSalient20_2).

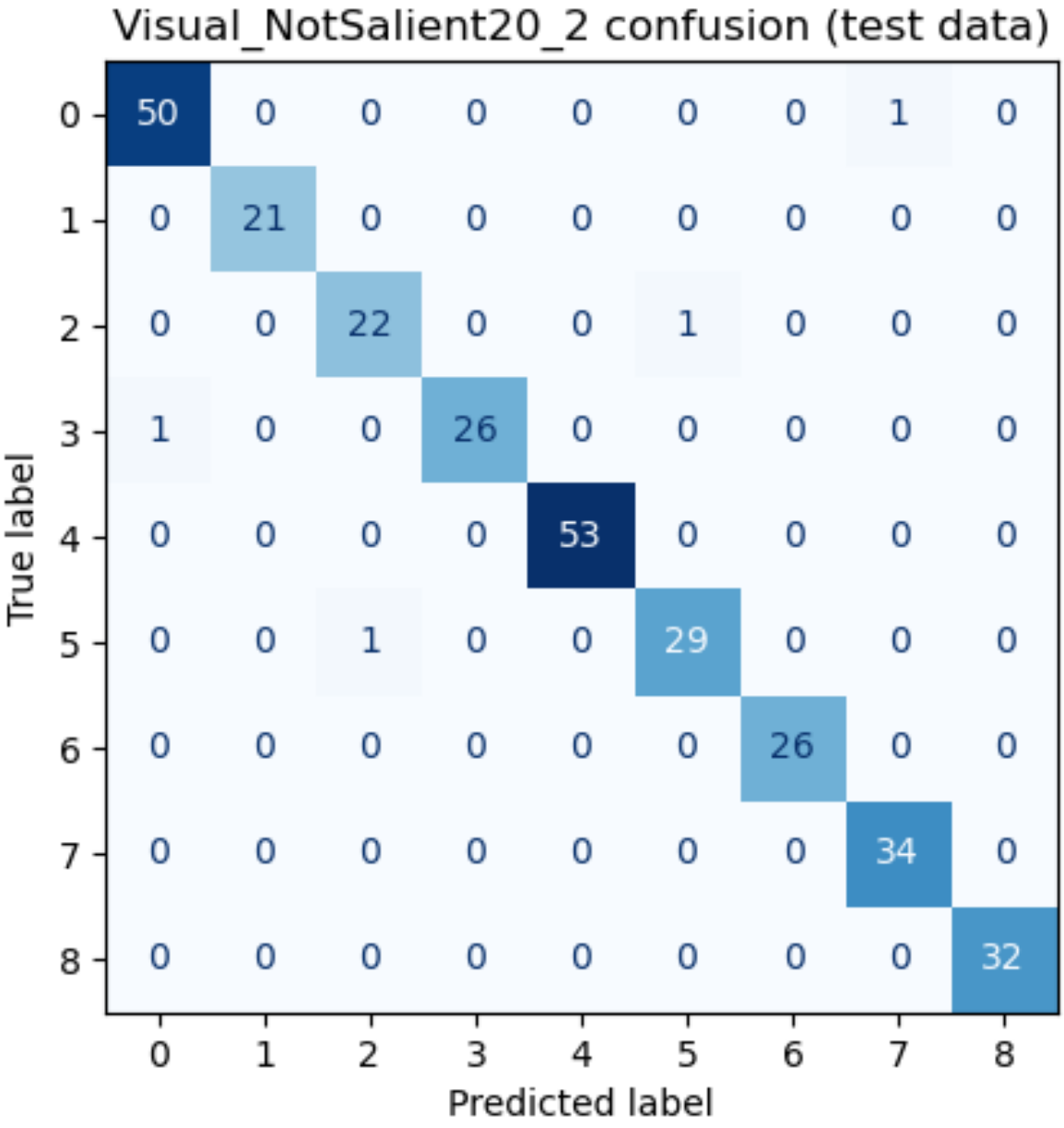

FIGURE G.8 – Matrice de confusion sur les données de test du modèle visuel (Vision_NotSalient20_2).

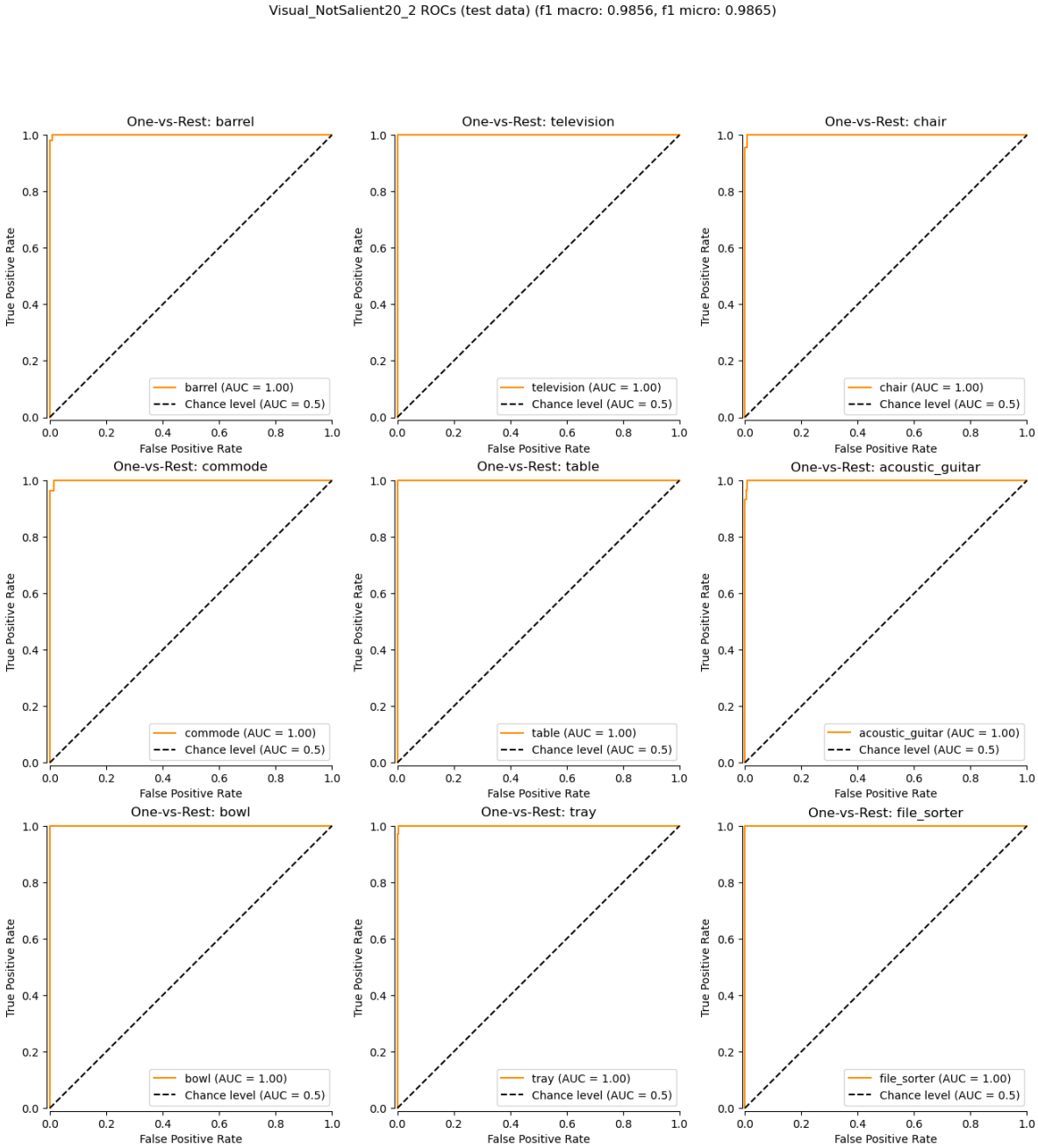

FIGURE G.9 – Courbes ROC et F1 Scores sur les données de test du modèle visuel (Vision_NotSalient20_2).

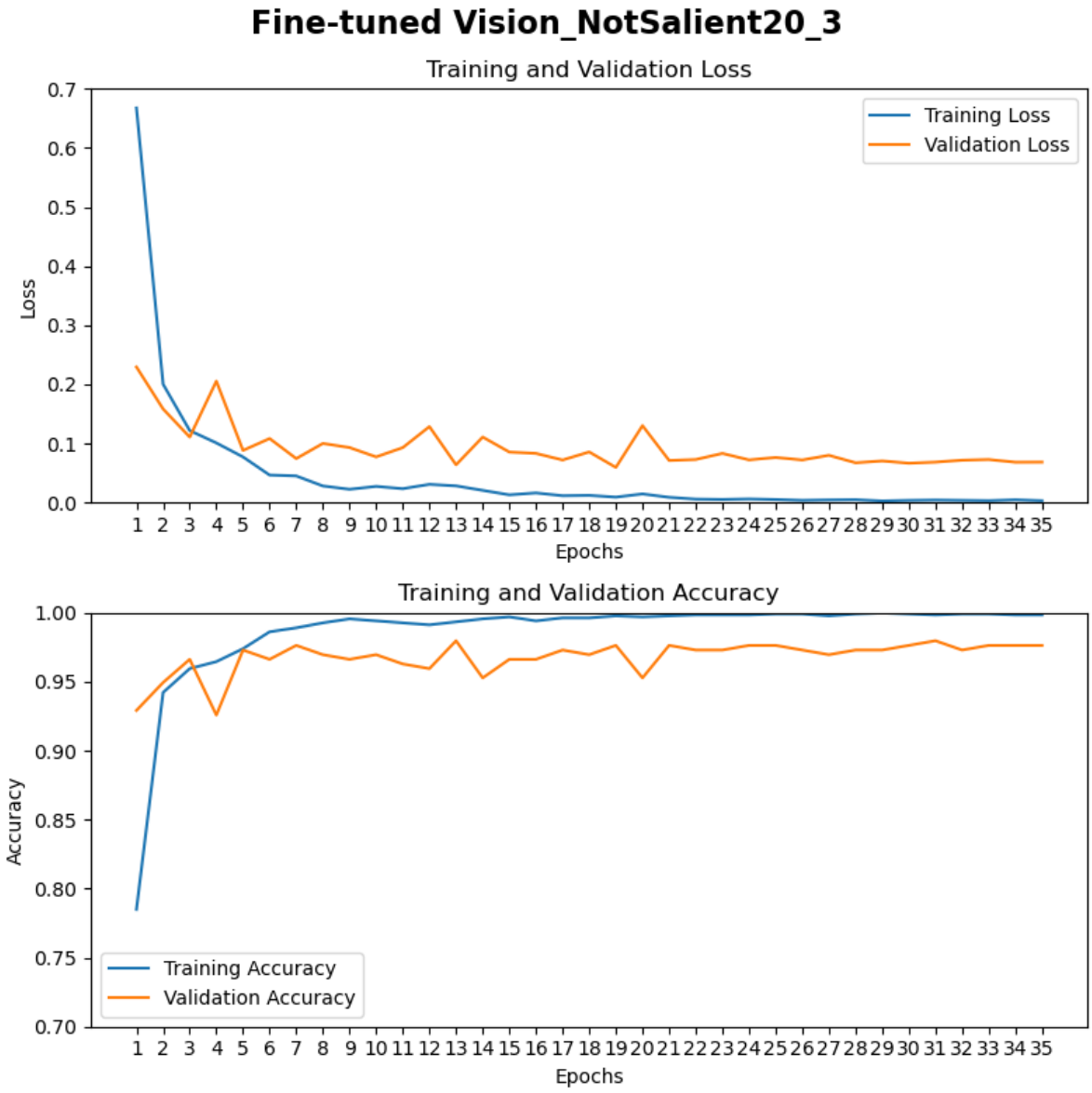

FIGURE G.10 – Historique de l'entraînement du modèle visuel utilisant les 20 points sélectionnés aléatoirement (# 41 à 60) par objet (Vision_NotSalient20_3).

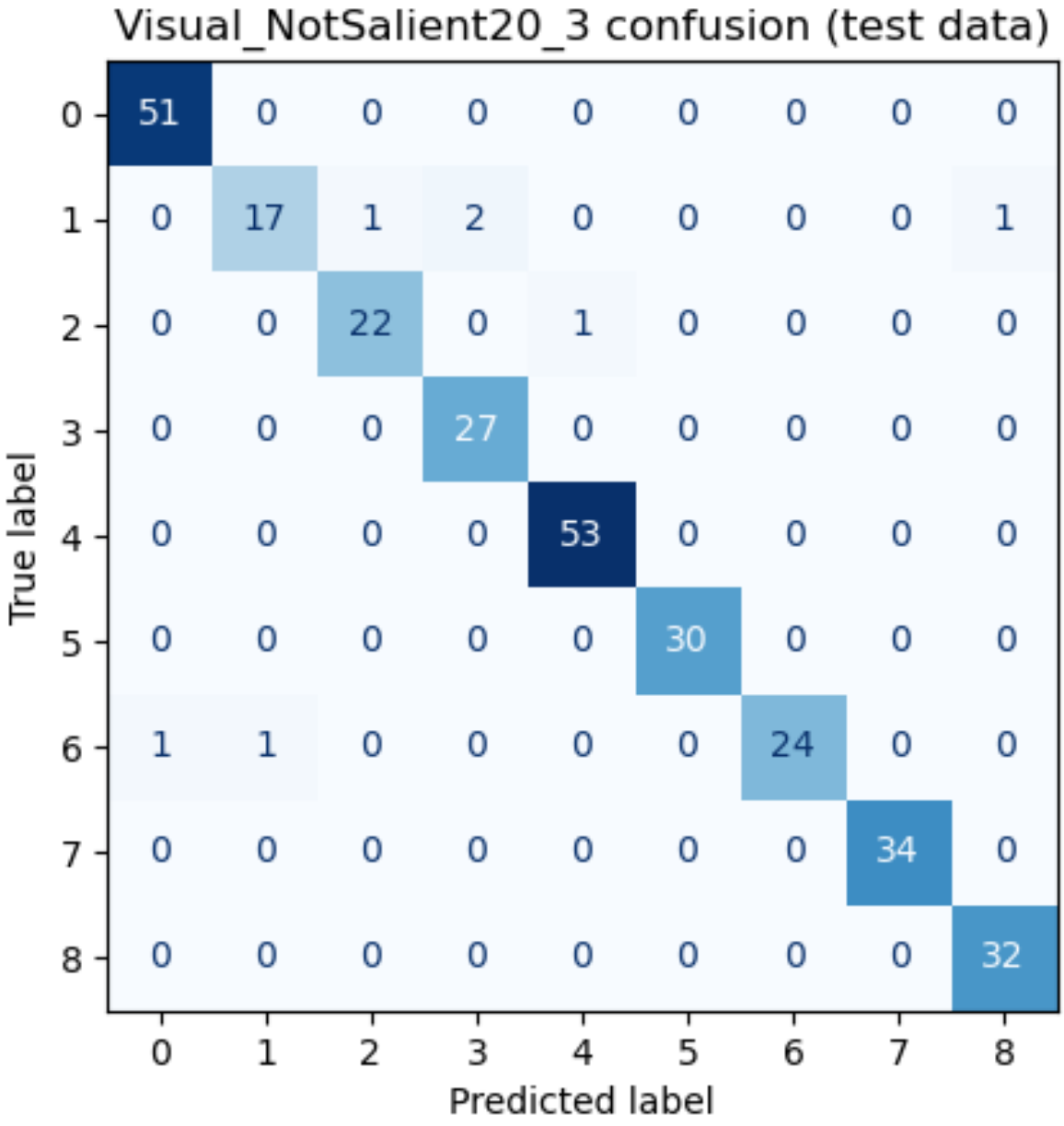

FIGURE G.11 – Matrice de confusion sur les données de test du modèle visuel (Vision_NotSalient20_3).

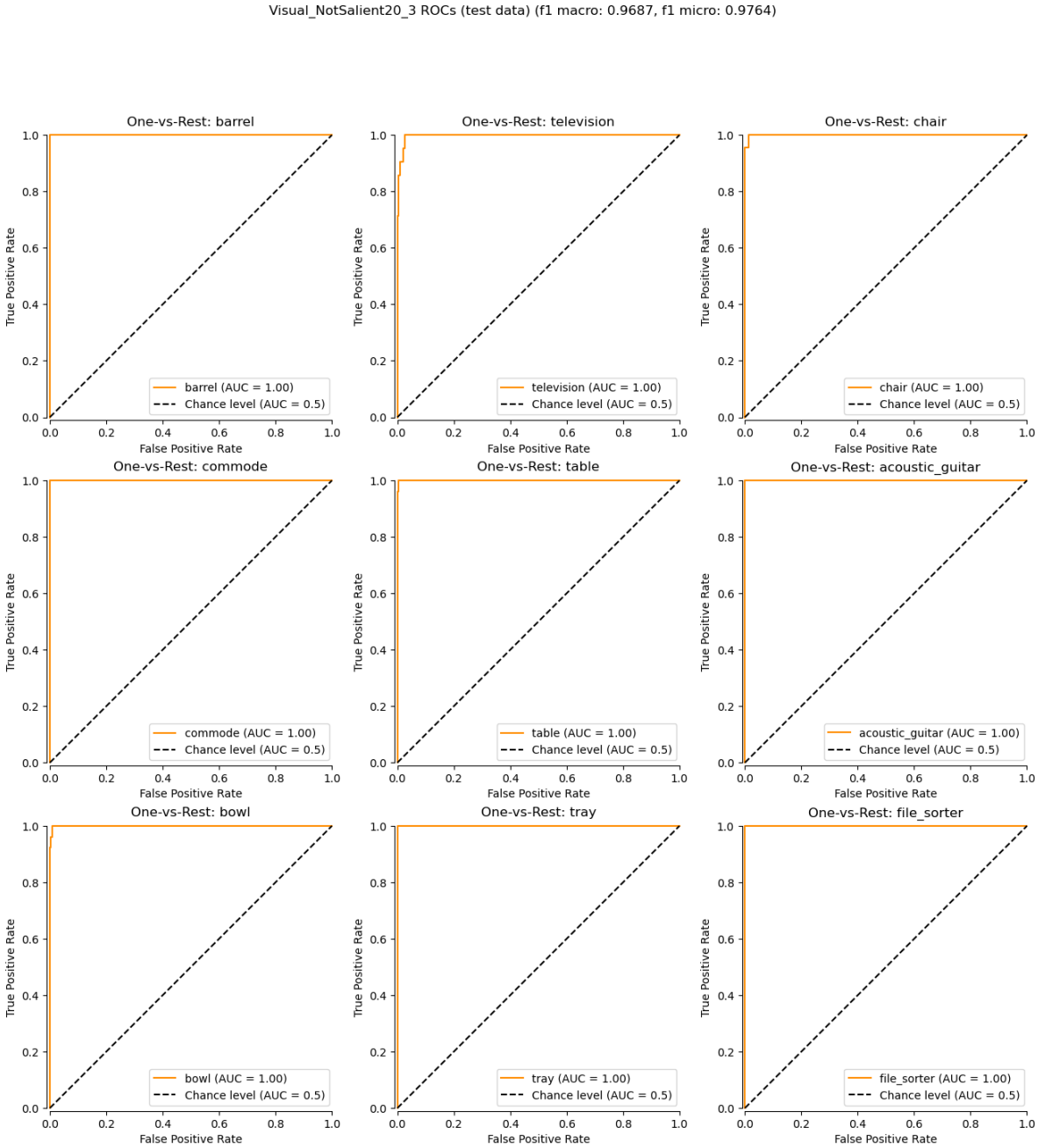

FIGURE G.12 – Courbes ROC et F1 Scores sur les données de test du modèle visuel (Vision_NotSalient20_3).

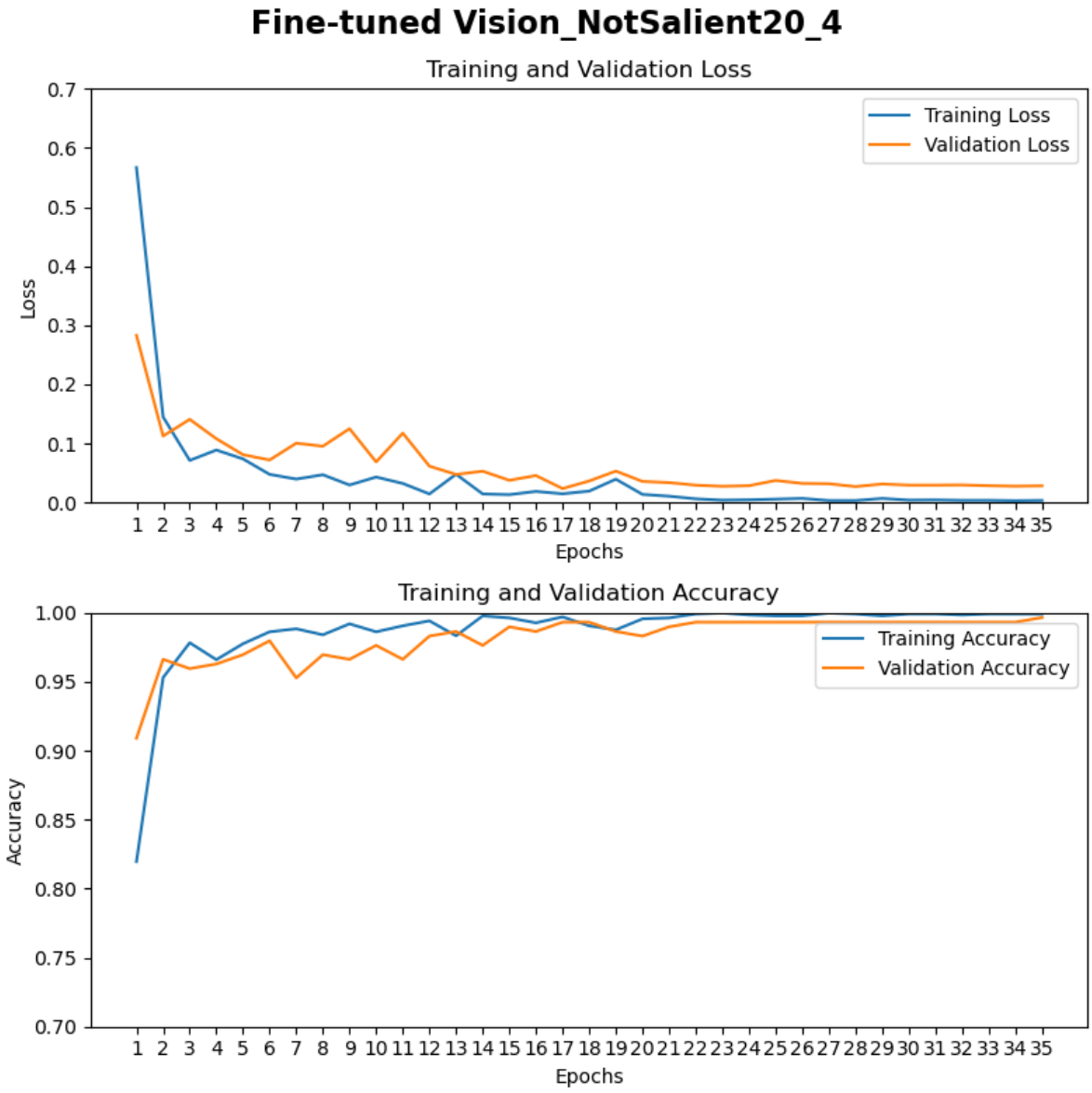

FIGURE G.13 – Historique de l'entraînement du modèle visuel utilisant les 20 points sélectionnés aléatoirement (# 61 à 80) par objet (Vision_NotSalient20_4).

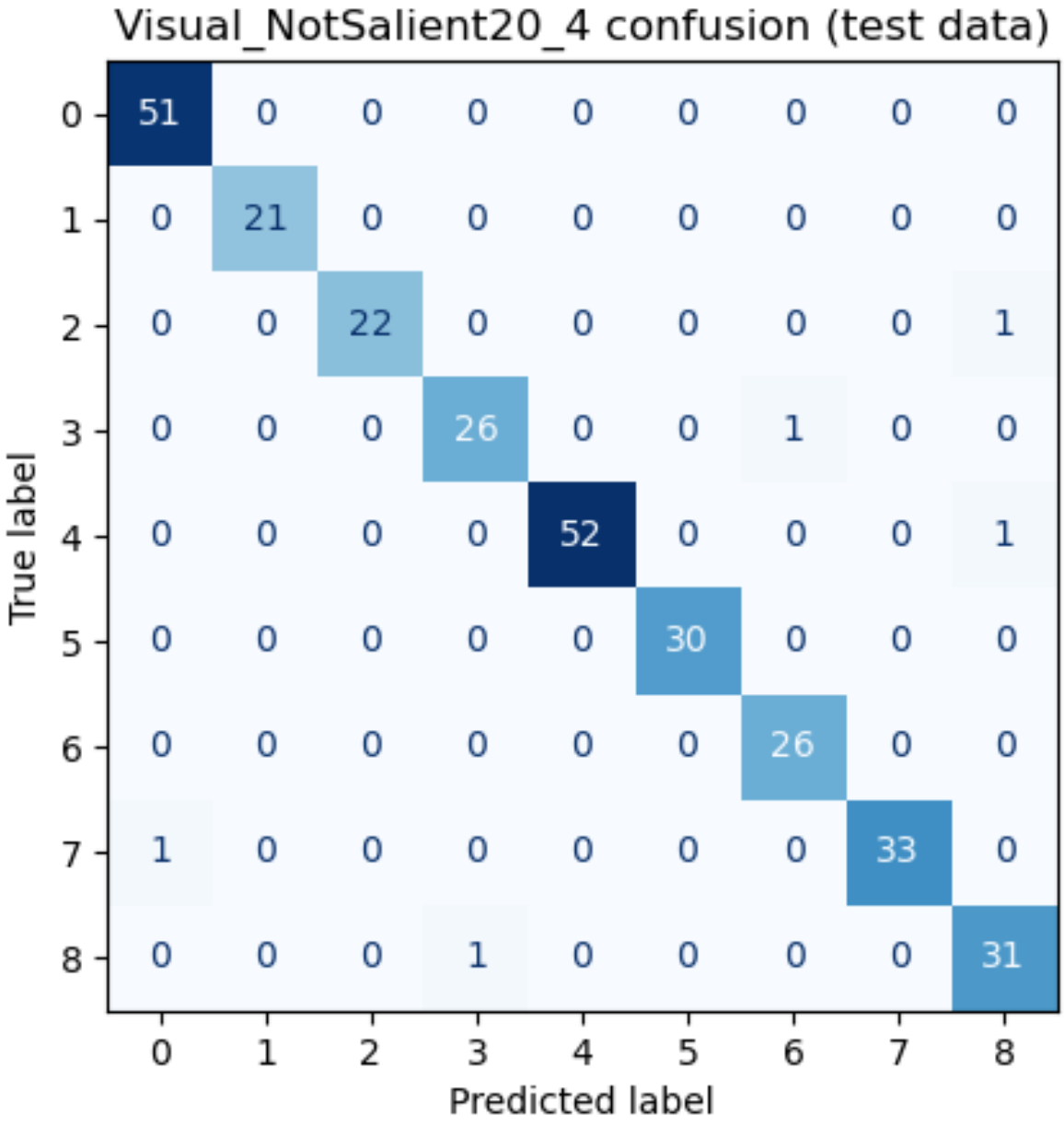

FIGURE G.14 – Matrice de confusion sur les données de test du modèle visuel (Vision_NotSalient20_4).

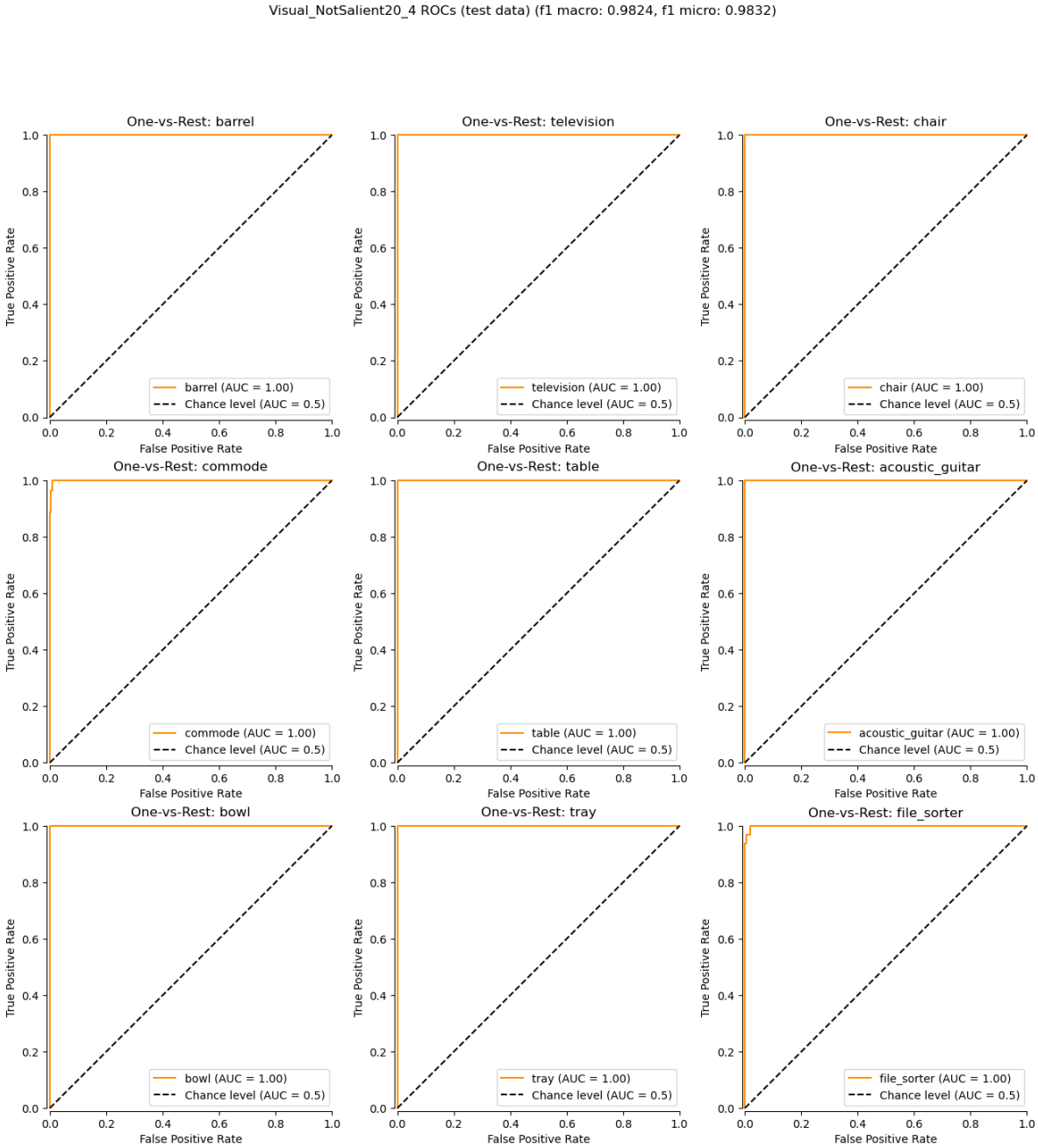

FIGURE G.15 – Courbes ROC et F1 Scores sur les données de test du modèle visuel (Vision_NotSalient20_4).

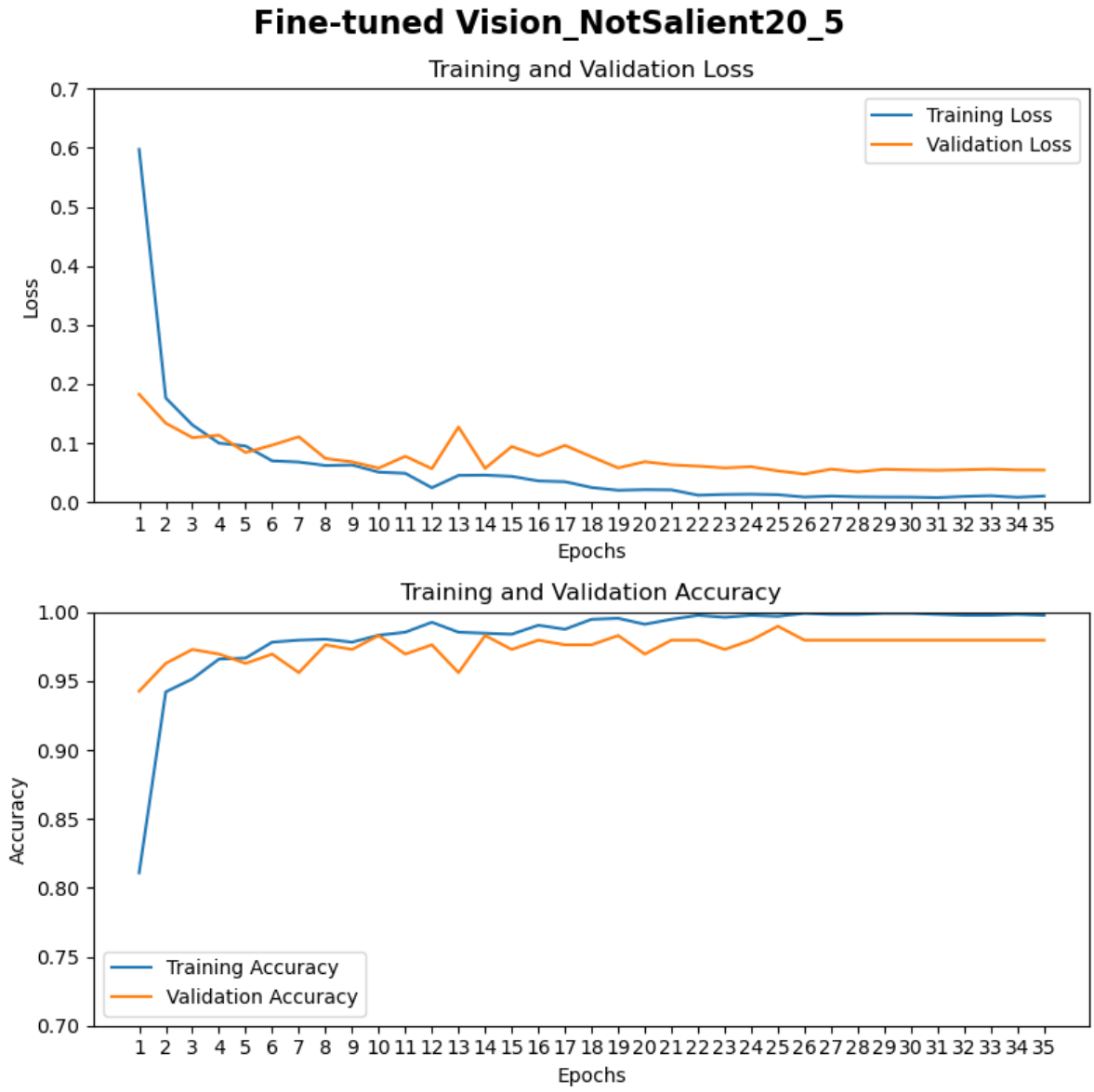

FIGURE G.16 – Historique de l'entraînement du modèle visuel utilisant les 20 points sélectionnés aléatoirement (# 81 à 100) par objet (Vision_NotSalient20_5).

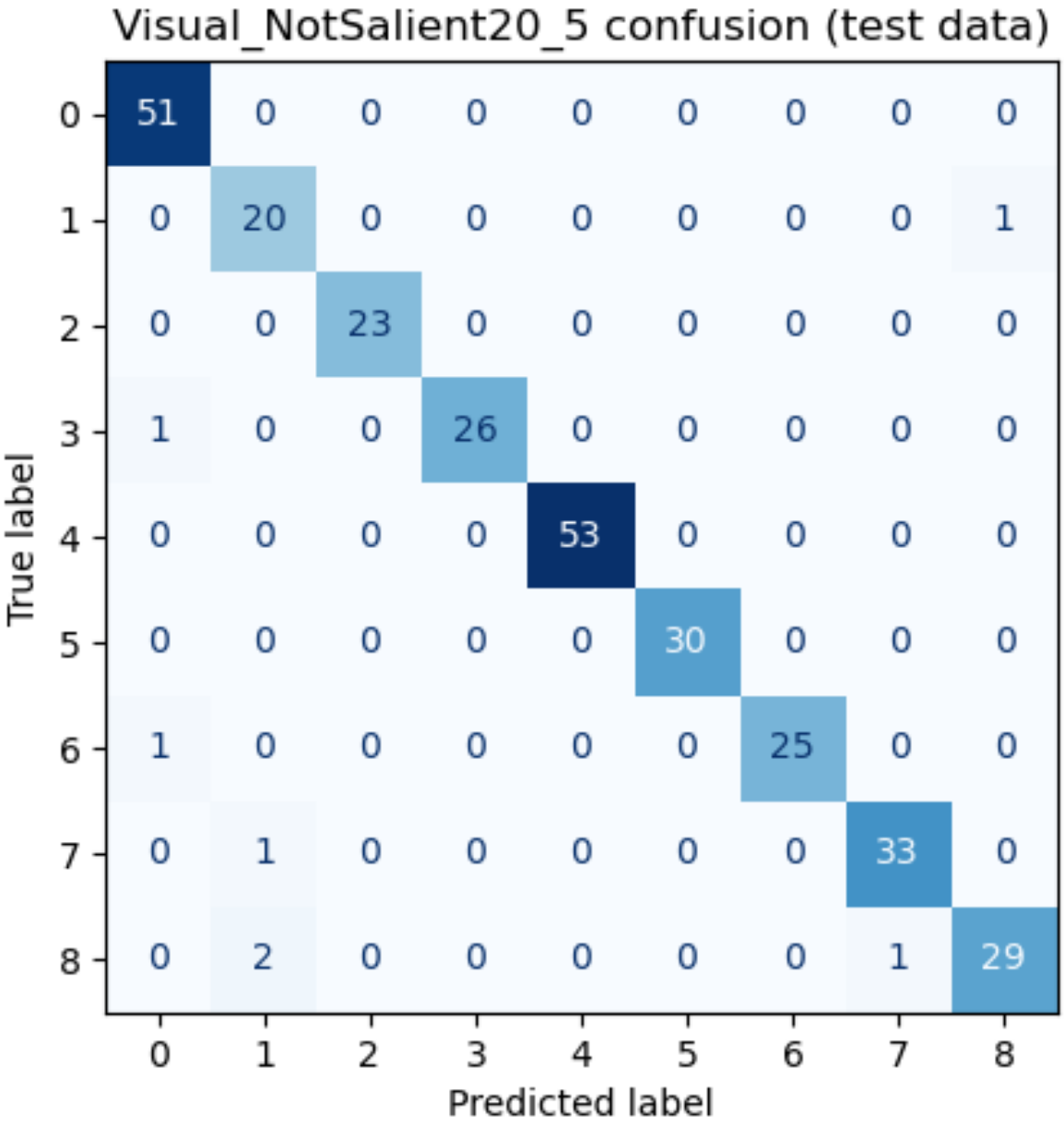

FIGURE G.17 – Matrice de confusion sur les données de test du modèle visuel (Vision_NotSalient20_5).

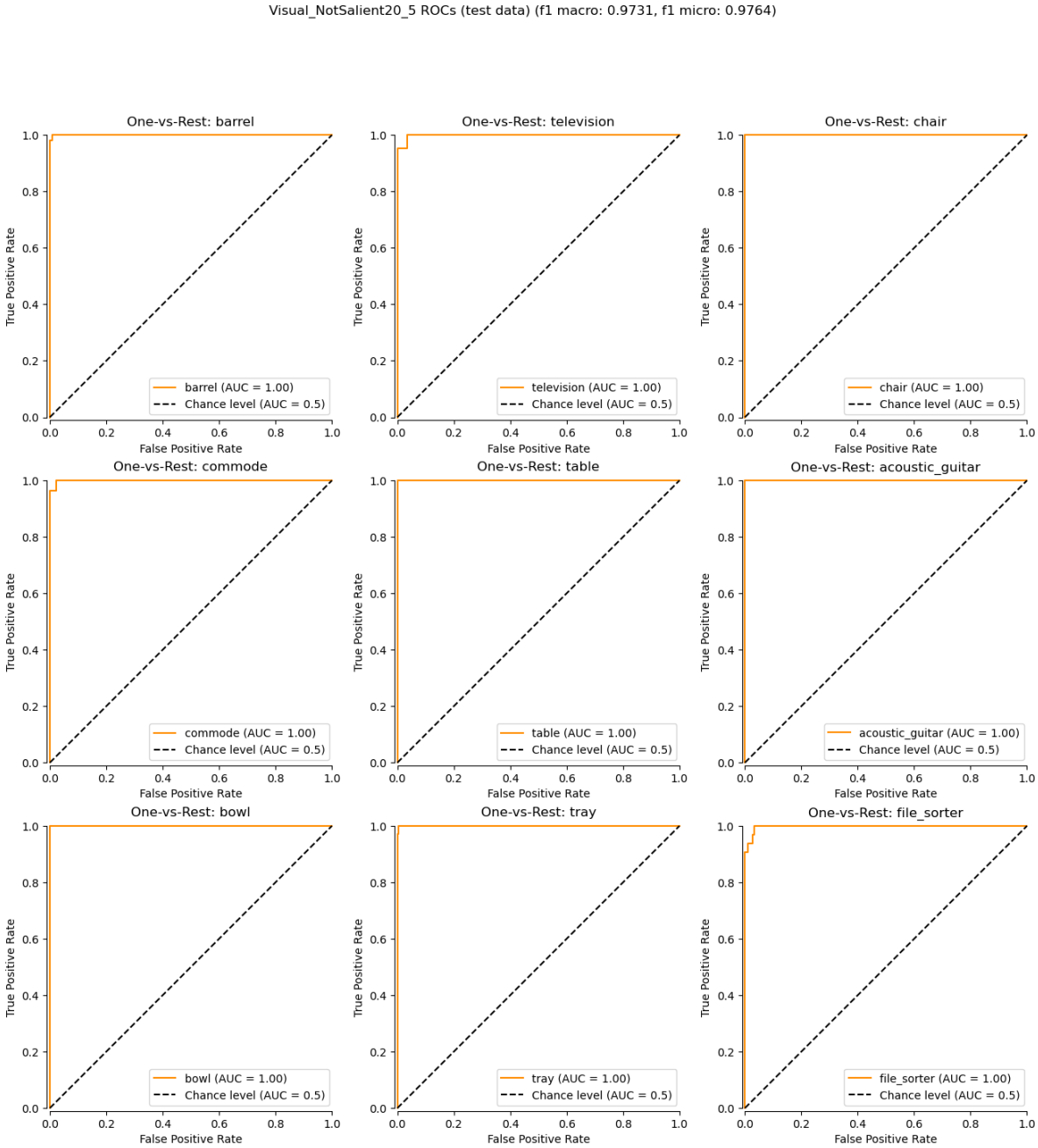

FIGURE G.18 – Courbes ROC et F1 Scores sur les données de test du modèle visuel (Vision_NotSalient20_5).

Annexe H

Résultats de l'entraînement des modèles tactiles

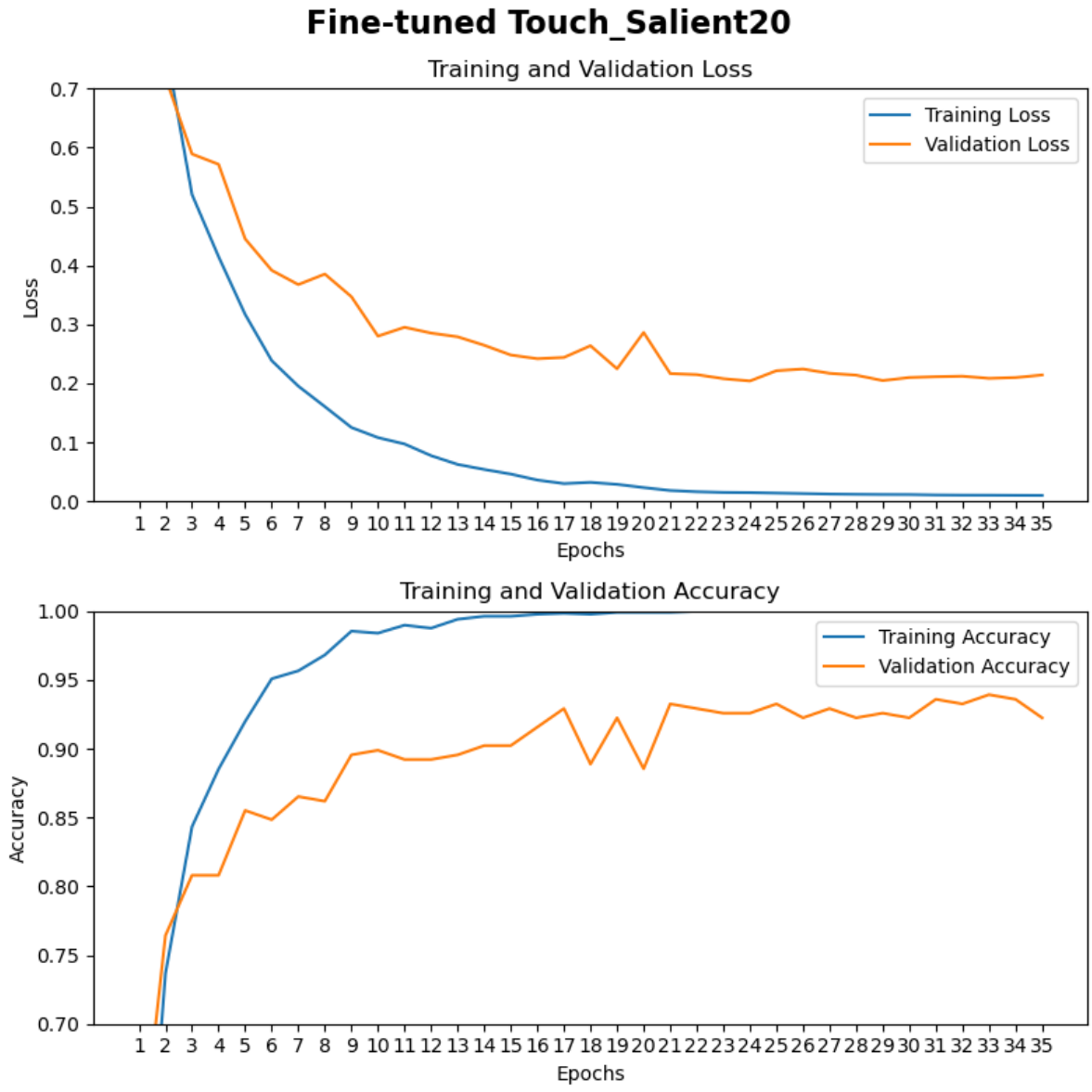

FIGURE H.1 – Historique de l'entraînement du modèle tactile utilisant les 20 points saillants par objet (Touch_Salient20).

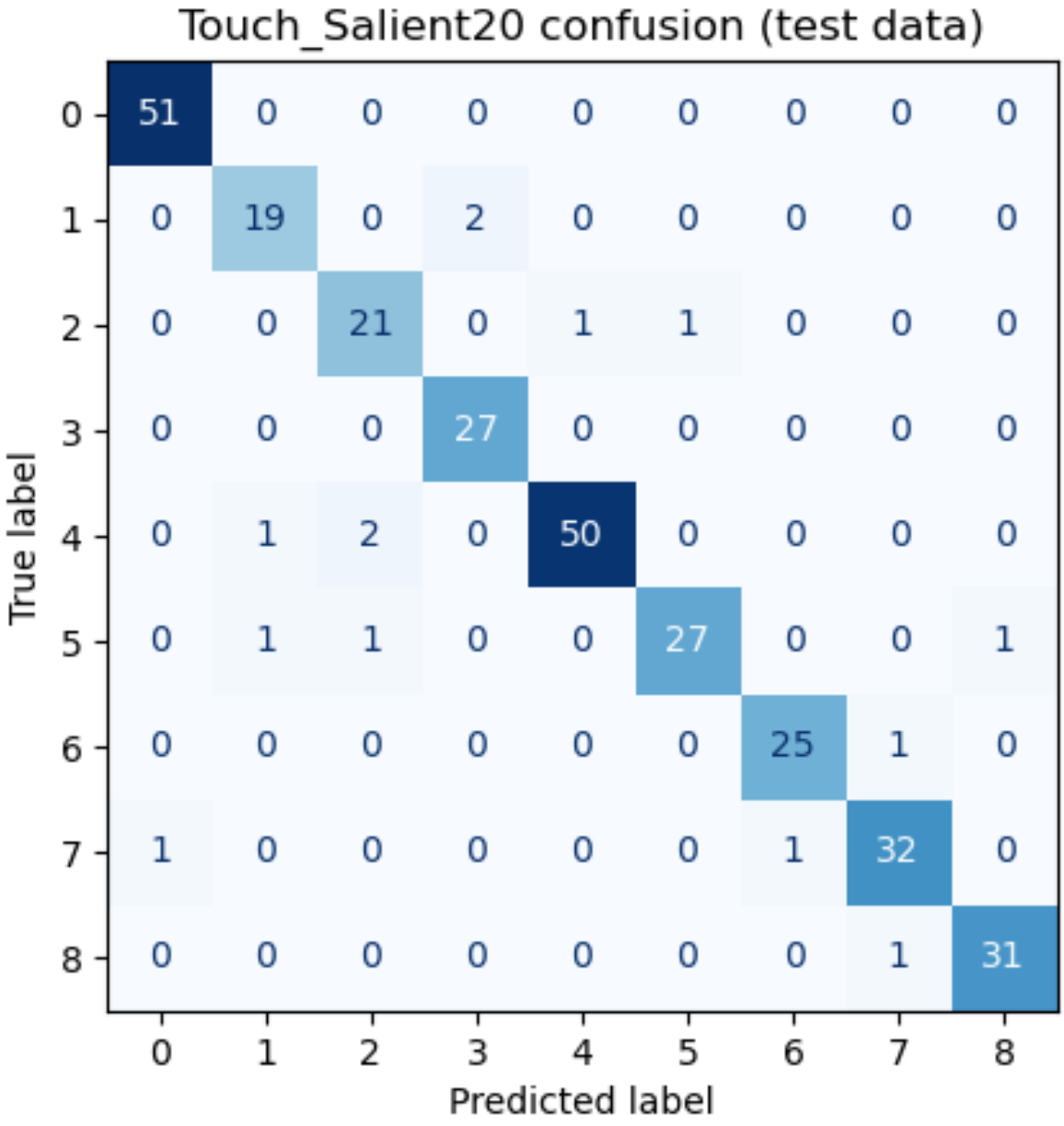

FIGURE H.2 – Matrice de confusion sur les données de test du modèle tactile (Touch_Salient20).

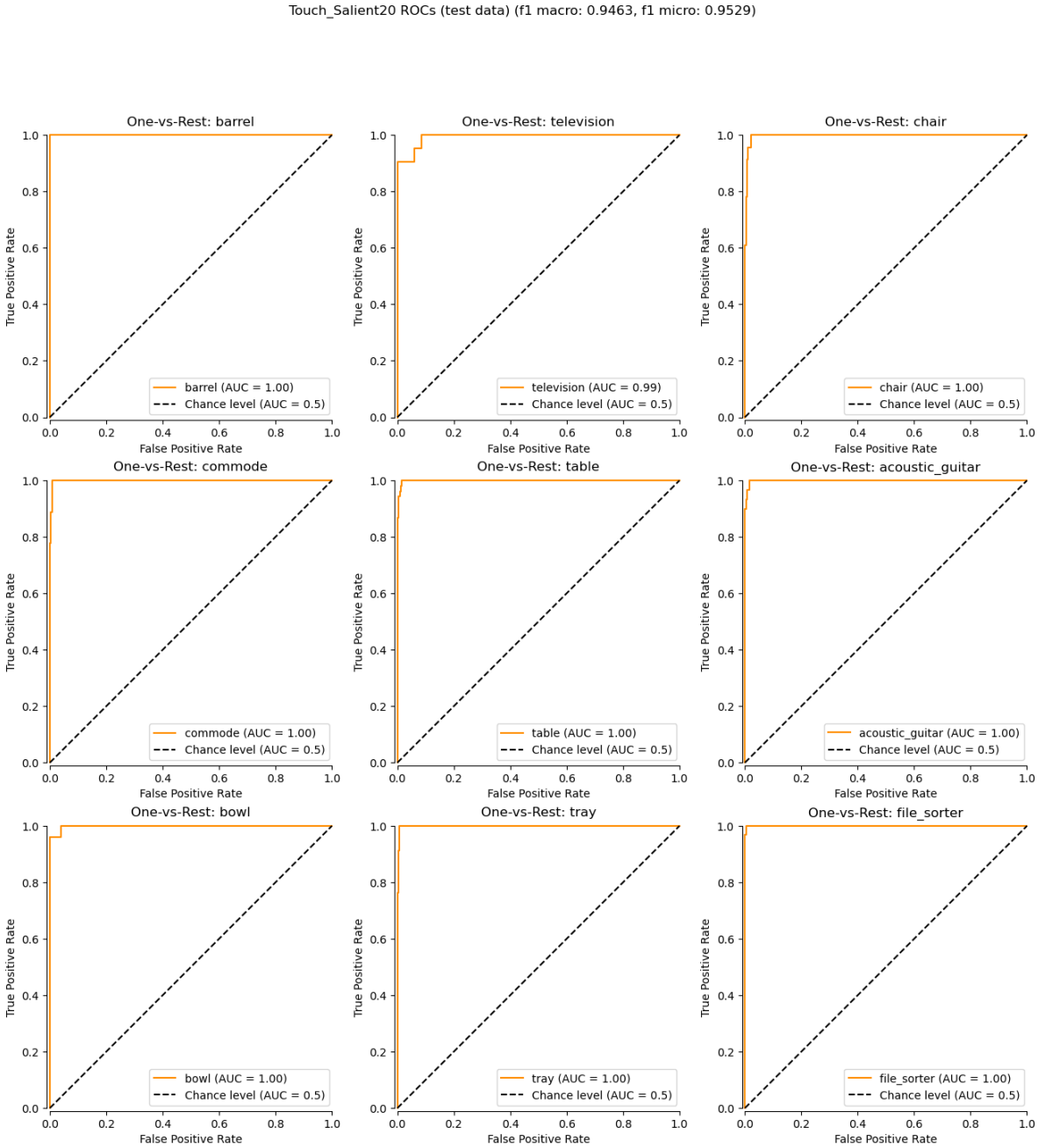

FIGURE H.3 – Courbes ROC et F1 Scores sur les données de test du modèle tactile (Touch_Salient20).

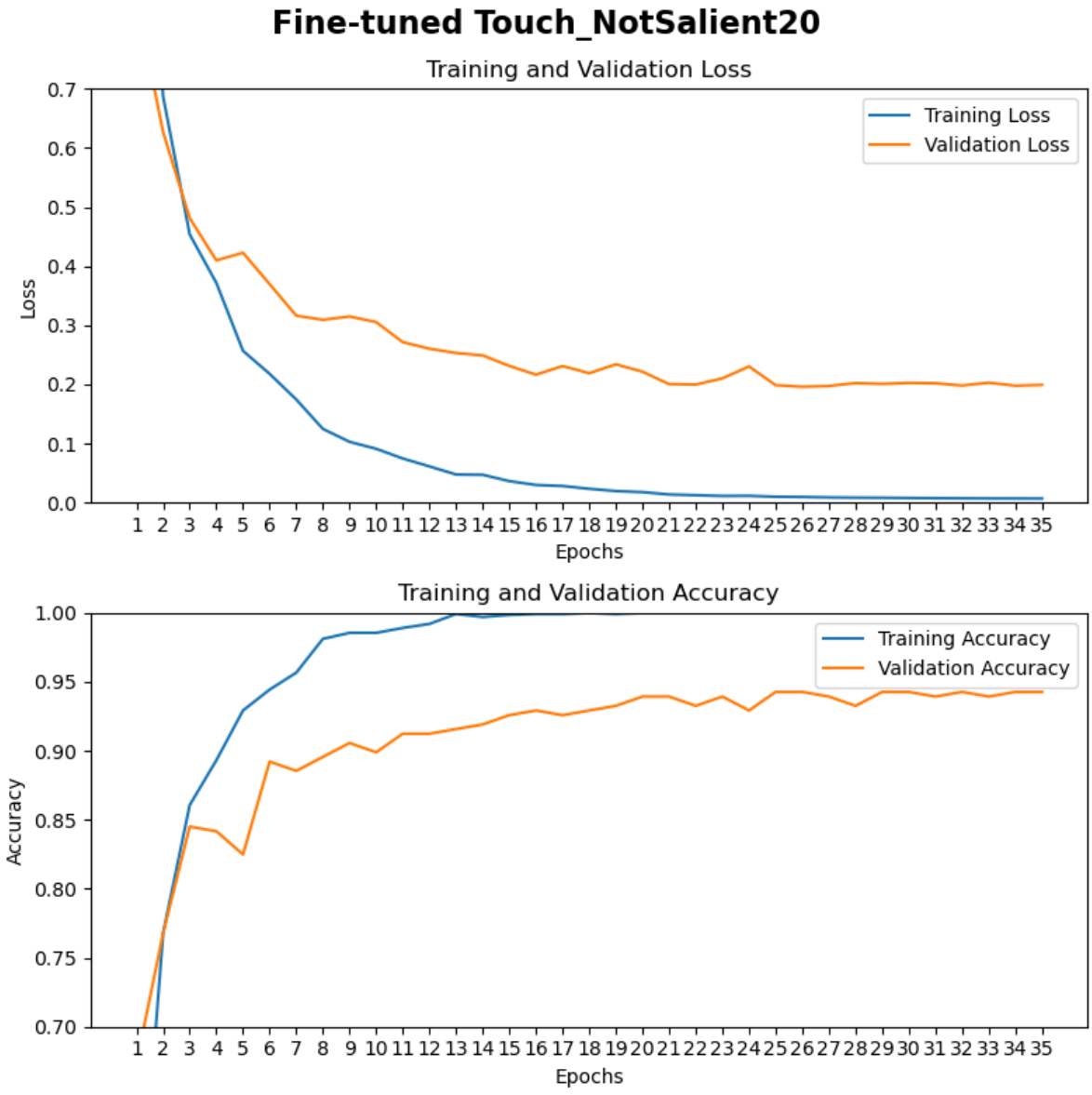

FIGURE H.4 – Historique de l'entraînement du modèle tactile utilisant les 20 premiers points sélectionnés aléatoirement par objet (Touch_NotSalient20).

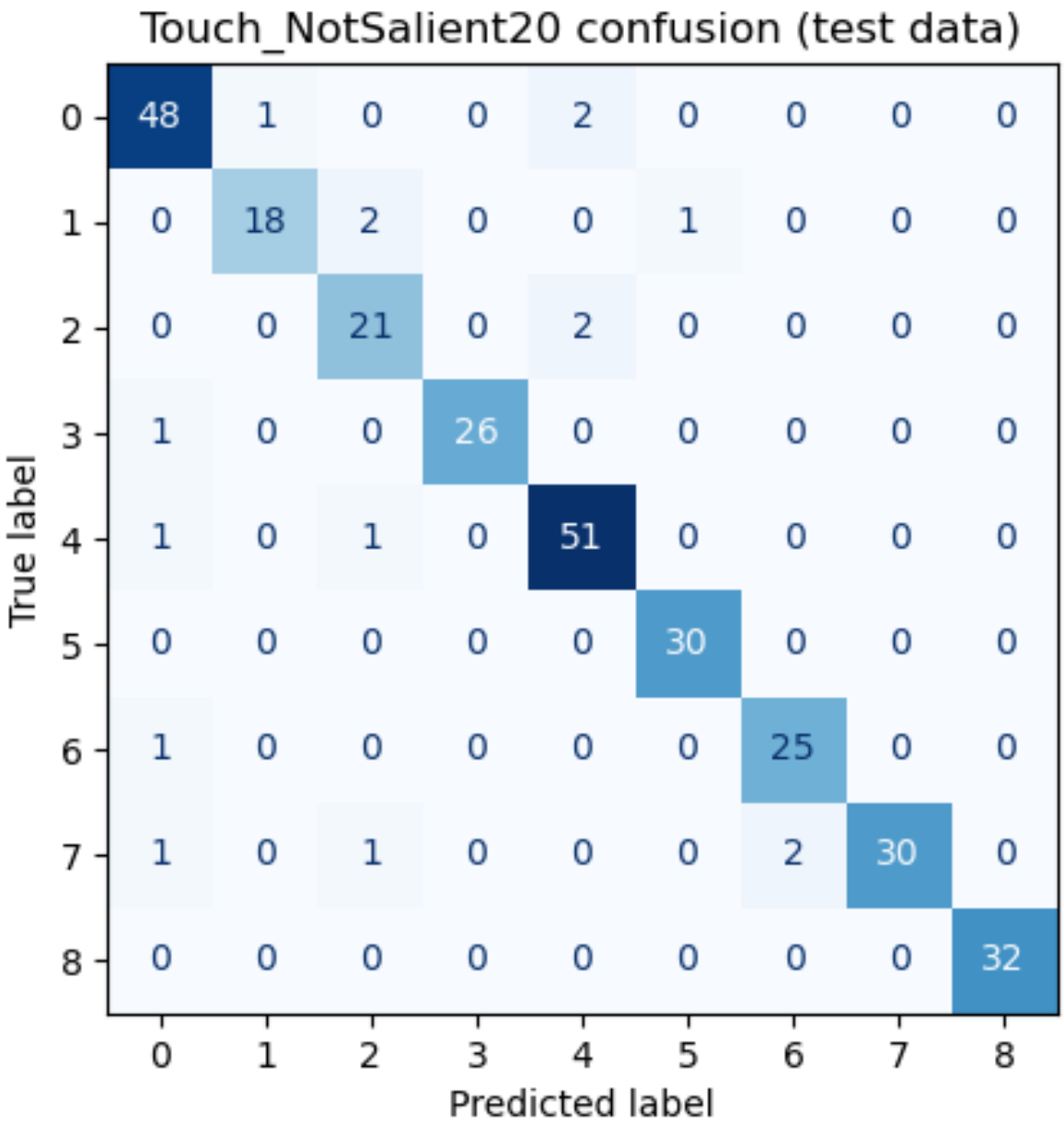

FIGURE H.5 – Matrice de confusion sur les données de test du modèle tactile (Touch_NotSalient20).

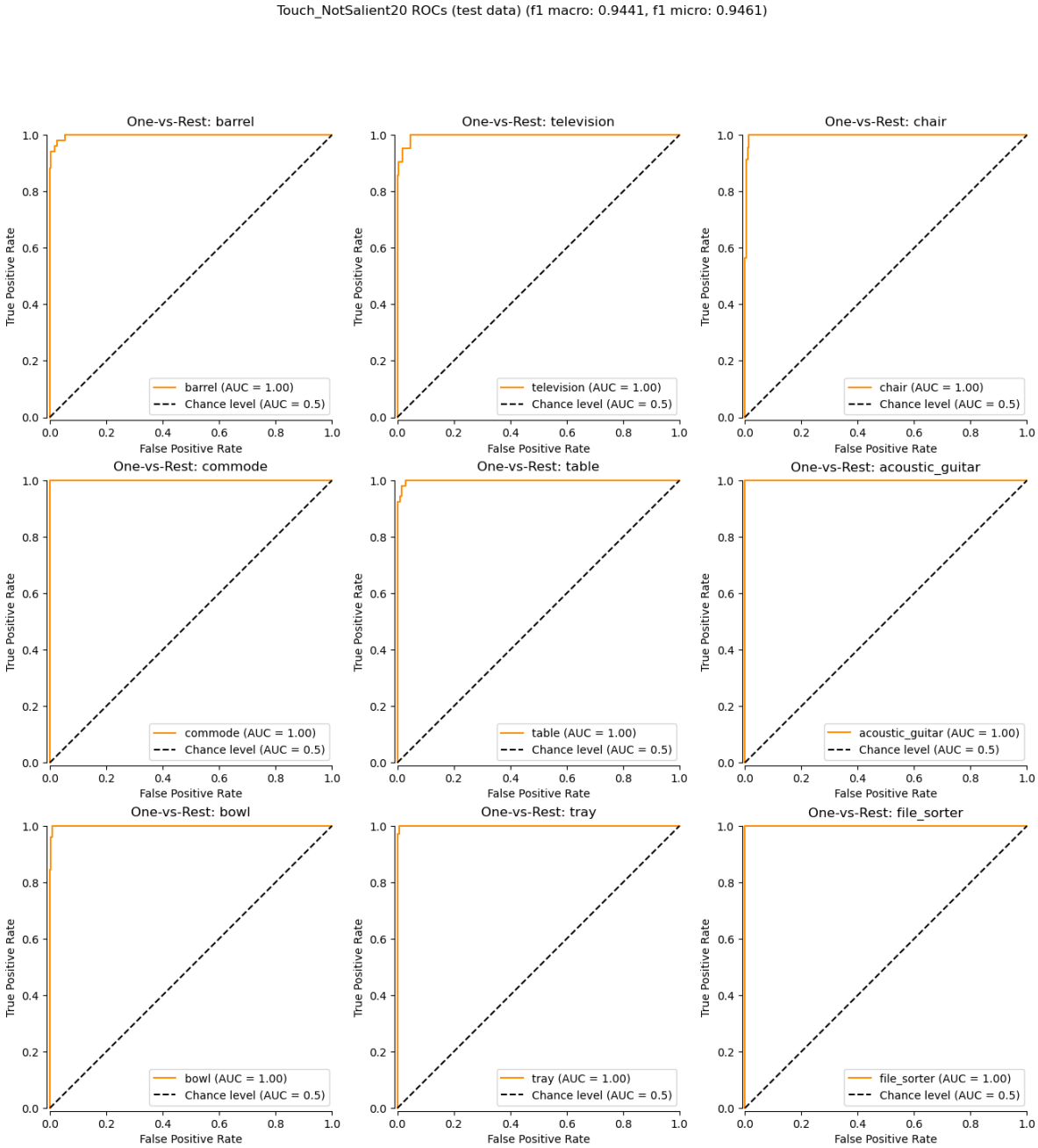

FIGURE H.6 – Courbes ROC et F1 Scores sur les données de test du modèle tactile (Touch_NotSalient20).

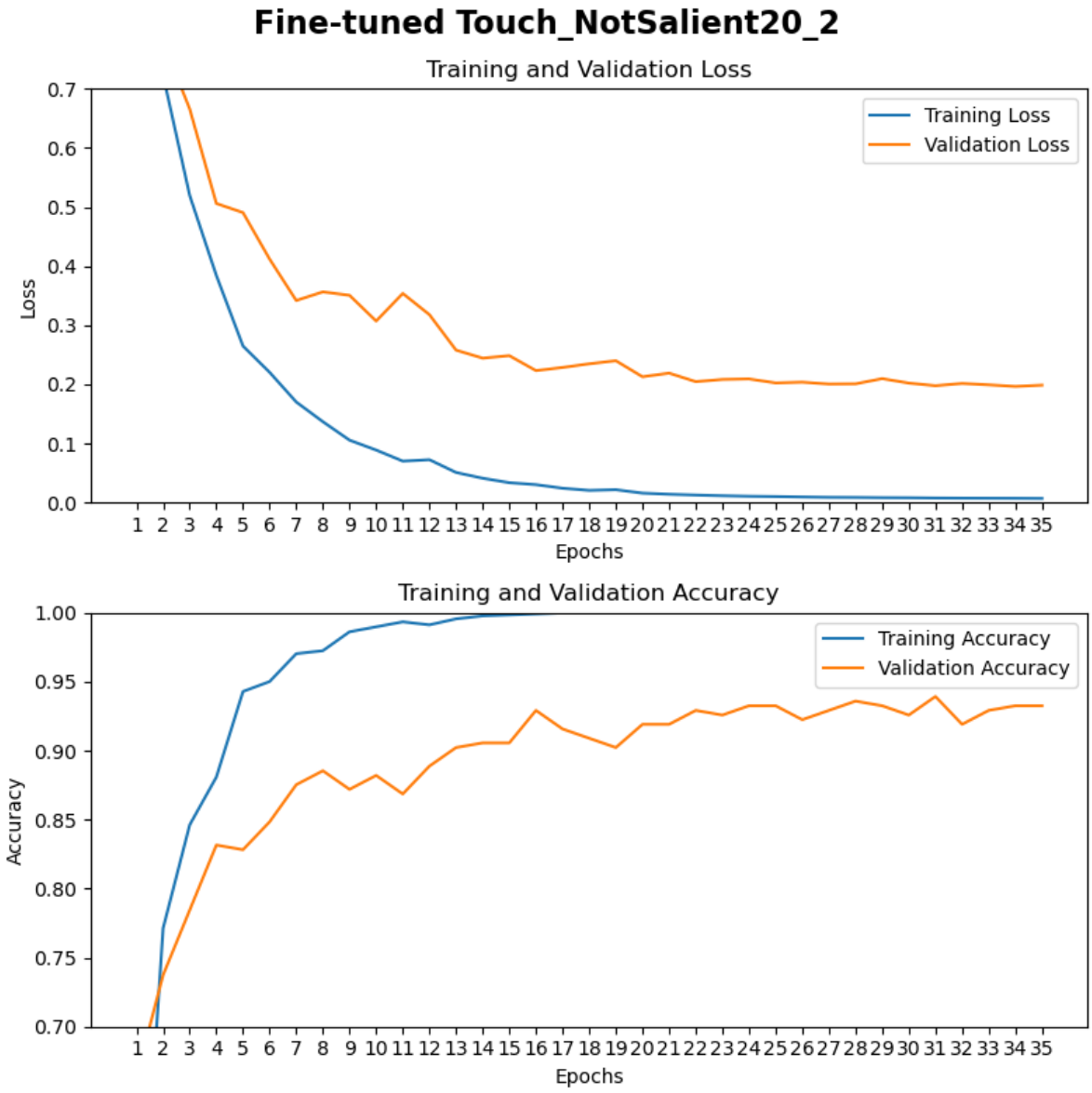

FIGURE H.7 – Historique de l'entraînement du modèle tactile utilisant les 20 points sélectionnés aléatoirement (# 21 à 40) par objet (Touch_NotSalient20_2).

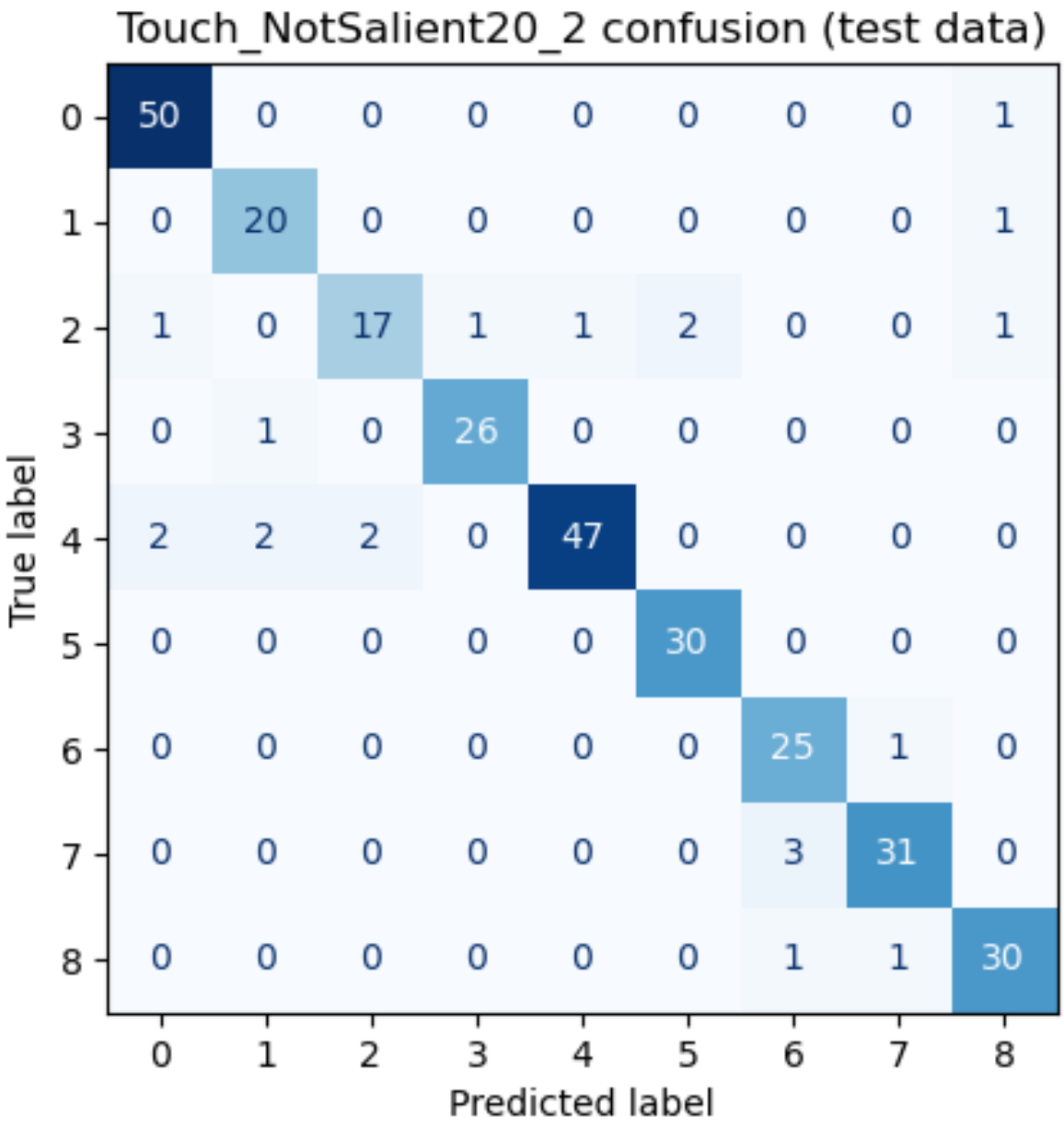

FIGURE H.8 – Matrice de confusion sur les données de test du modèle tactile (Touch_NotSalient20_2).

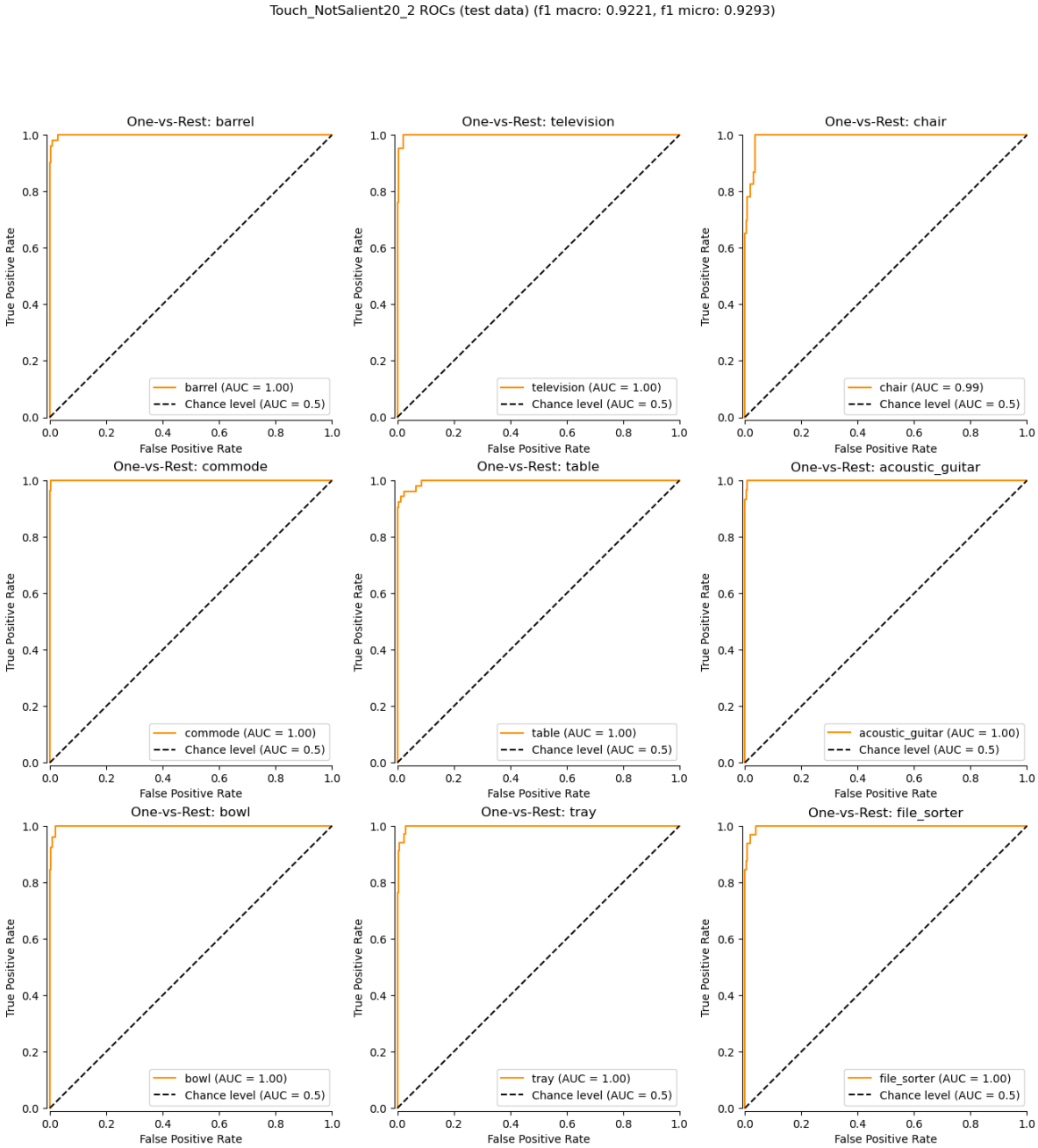

FIGURE H.9 – Courbes ROC et F1 Scores sur les données de test du modèle tactile (Touch_NotSalient20_2).

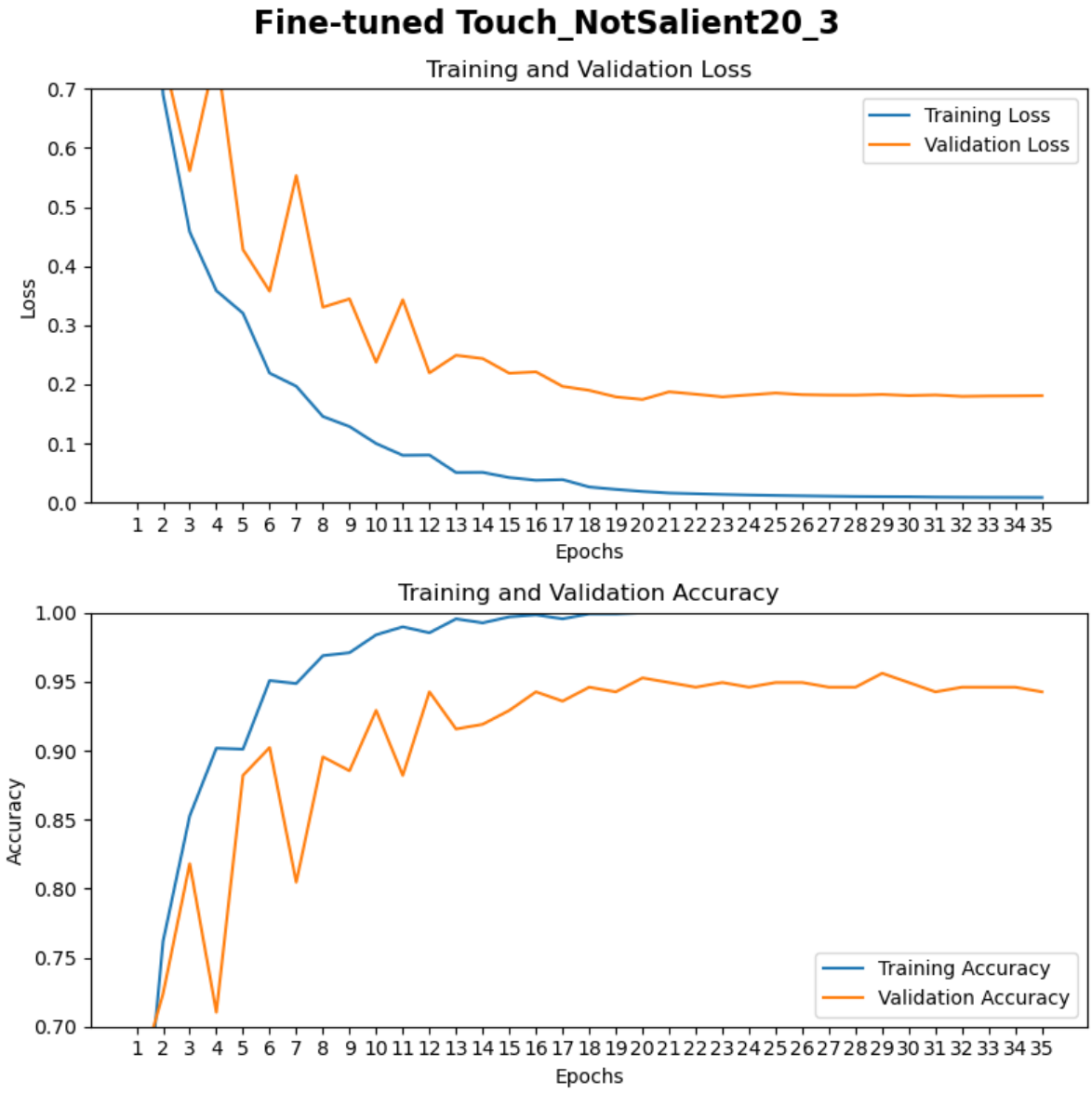

FIGURE H.10 – Historique de l'entraînement du modèle tactile utilisant les 20 points sélectionnés aléatoirement (# 41 à 60) par objet (Touch_NotSalient20_3).

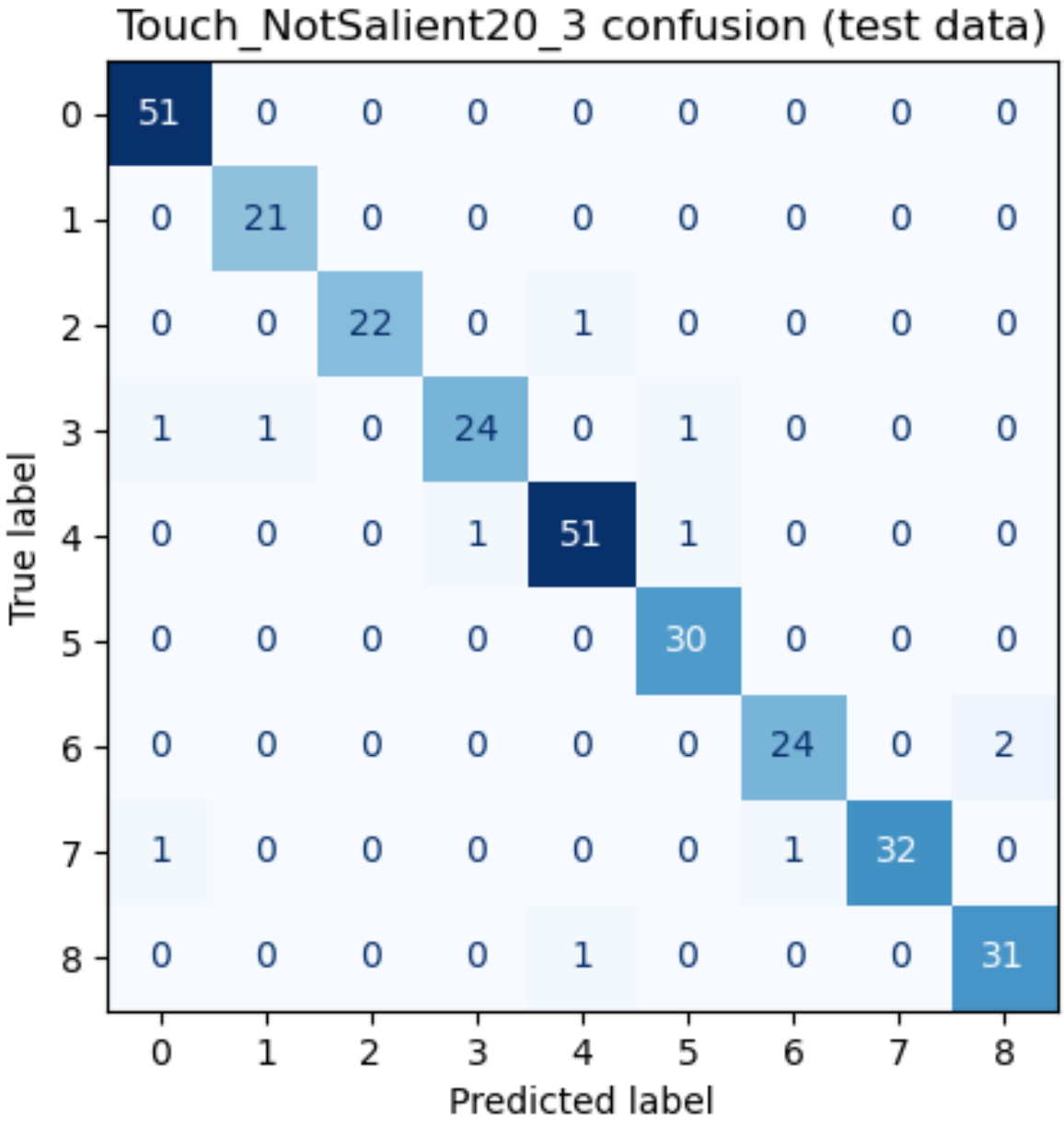

FIGURE H.11 – Matrice de confusion sur les données de test du modèle tactile (Touch_NotSalient20_3).

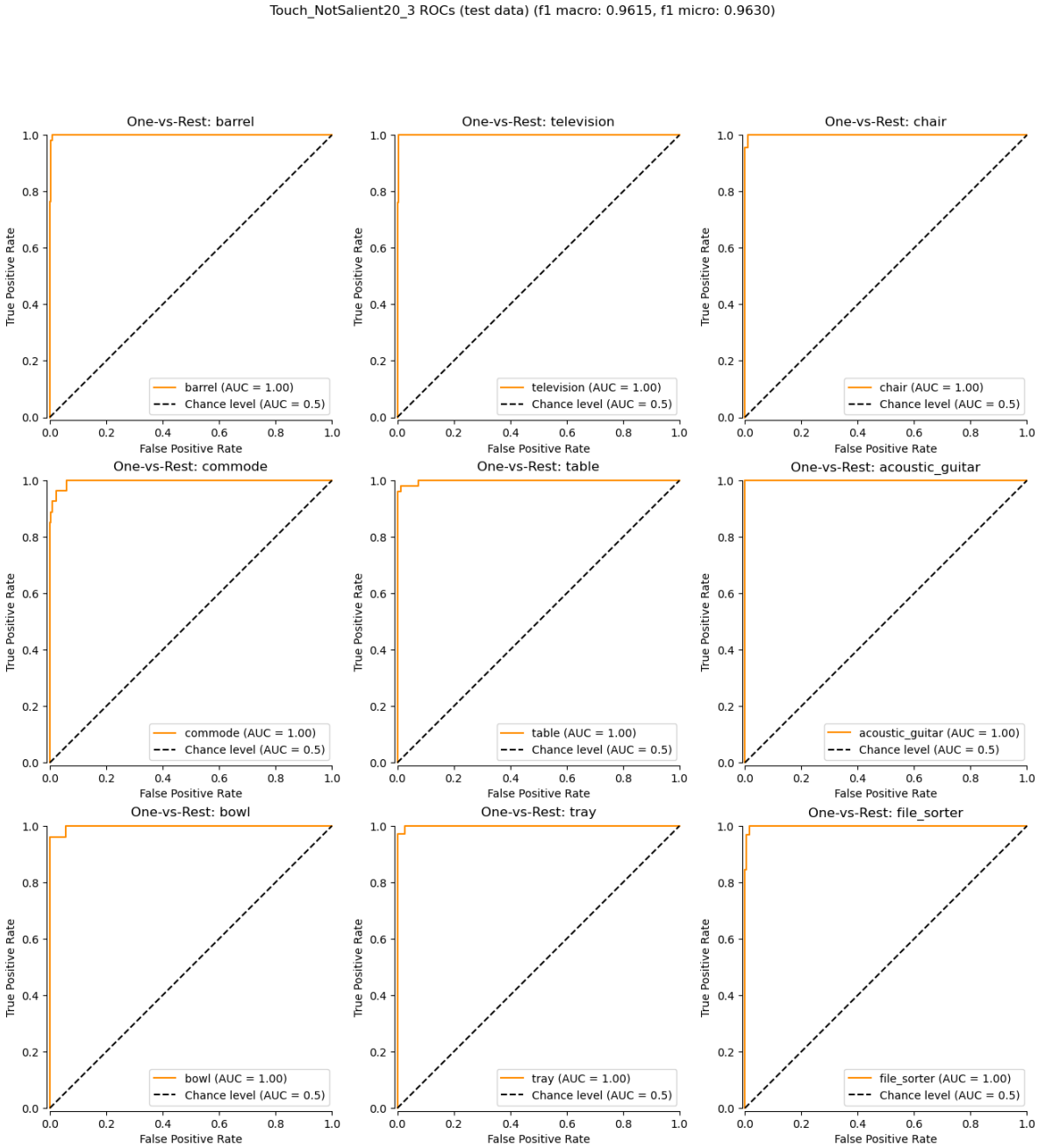

FIGURE H.12 – Courbes ROC et F1 Scores sur les données de test du modèle tactile (Touch_NotSalient20_3).

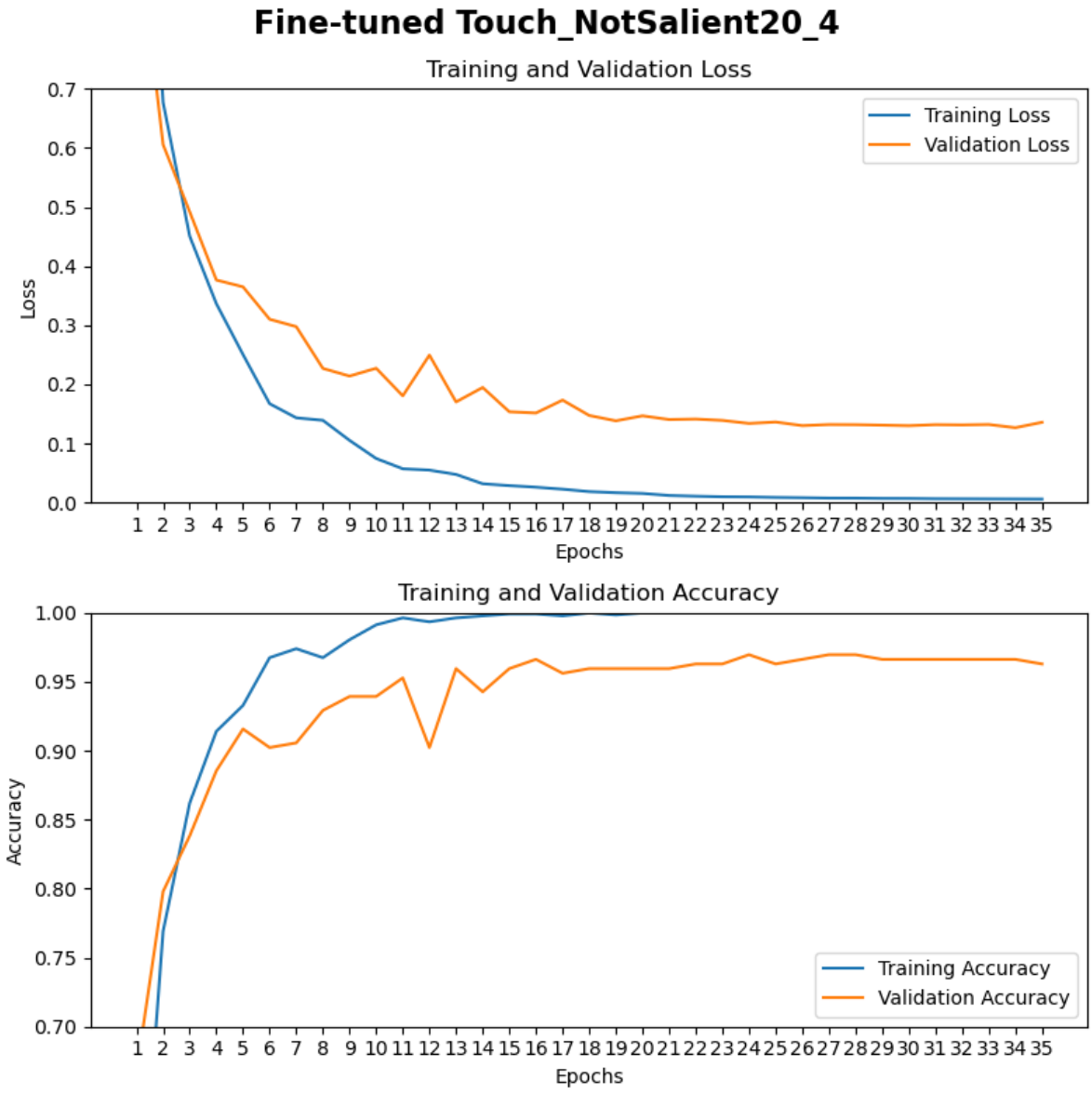

FIGURE H.13 – Historique de l'entraînement du modèle tactile utilisant les 20 points sélectionnés aléatoirement (# 61 à 80) par objet (Touch_NotSalient20_4).

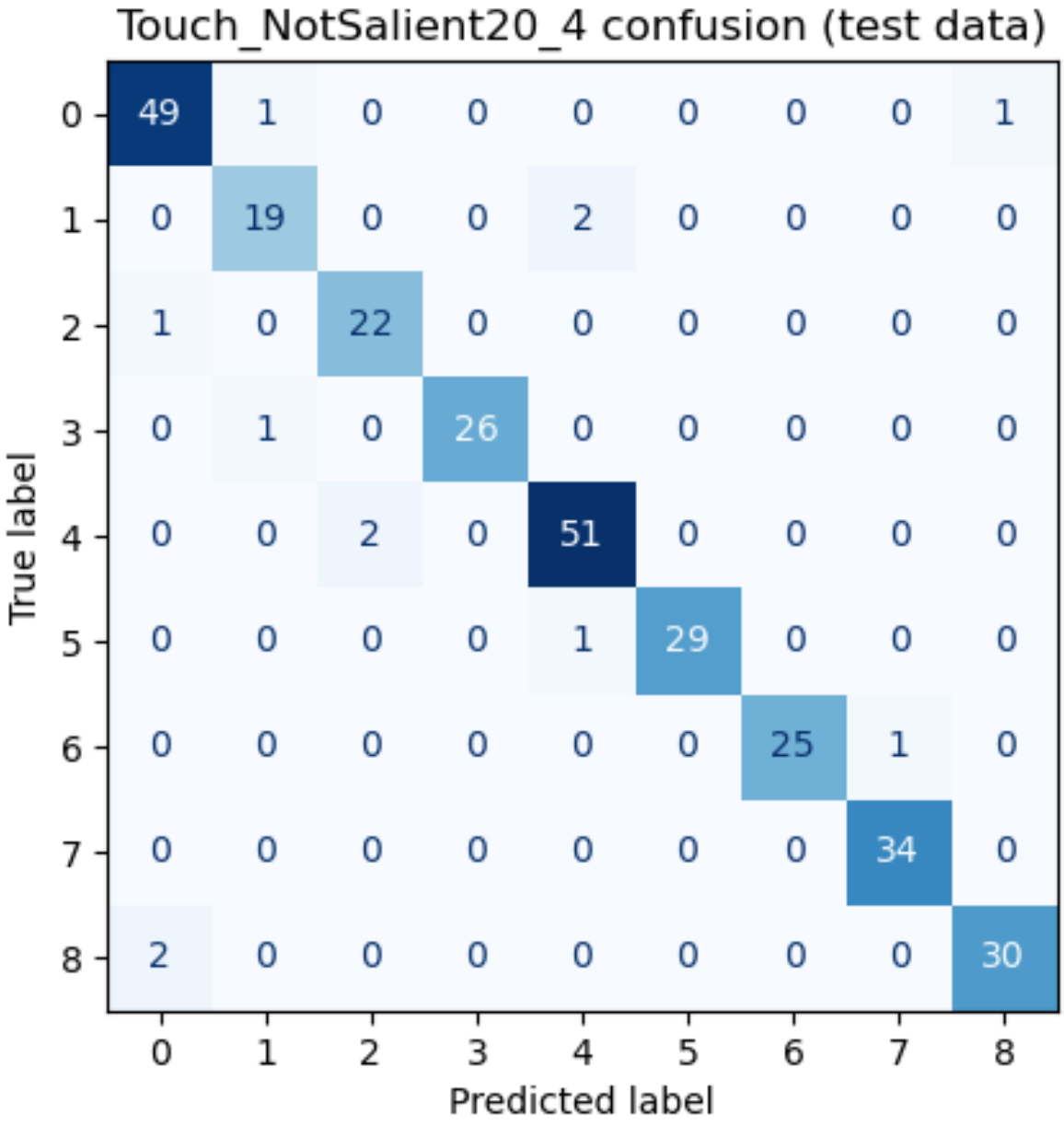

FIGURE H.14 – Matrice de confusion sur les données de test du modèle tactile (Touch_NotSalient20_4).

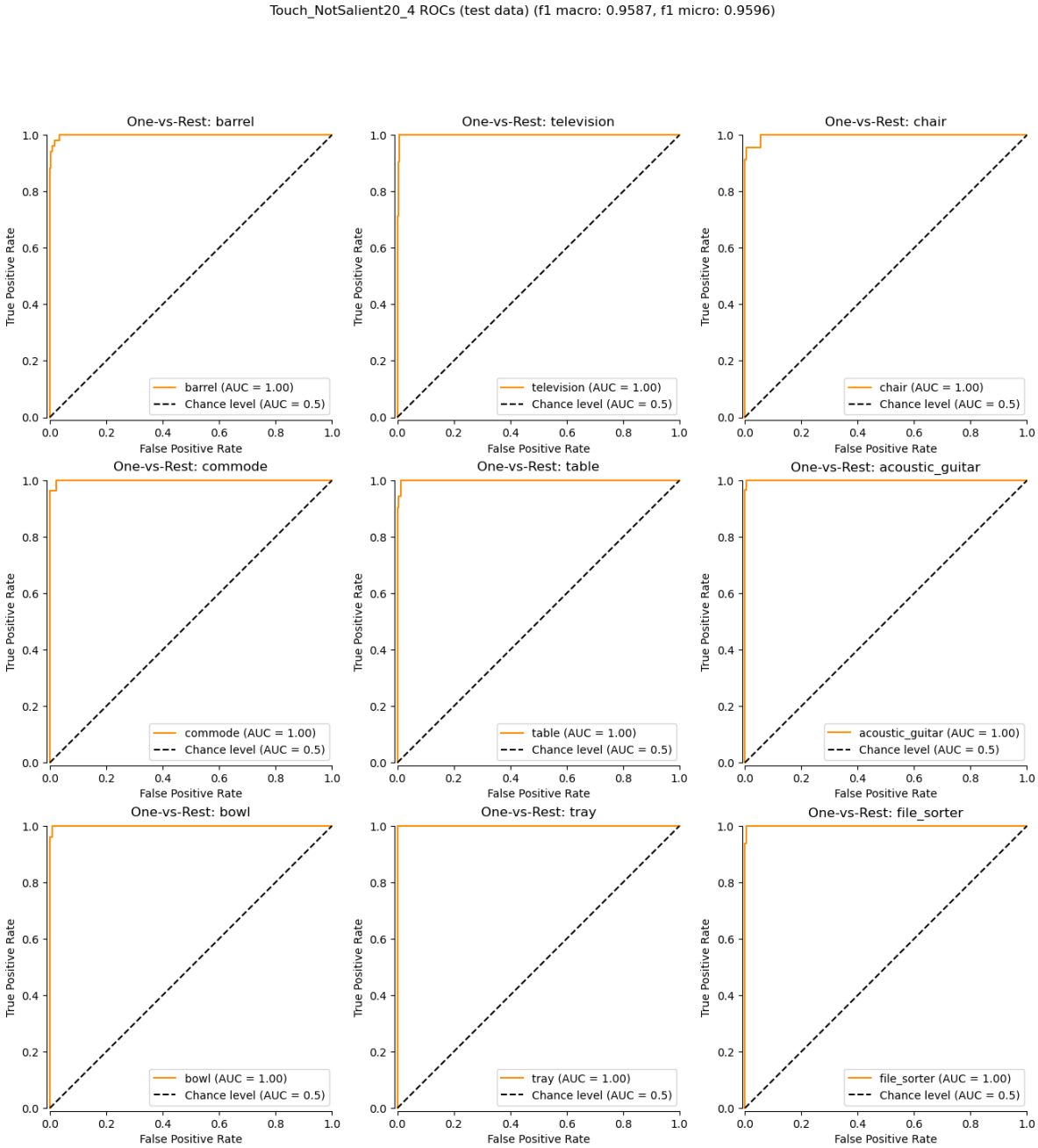

FIGURE H.15 – Courbes ROC et F1 Scores sur les données de test du modèle tactile (Touch_NotSalient20_4).

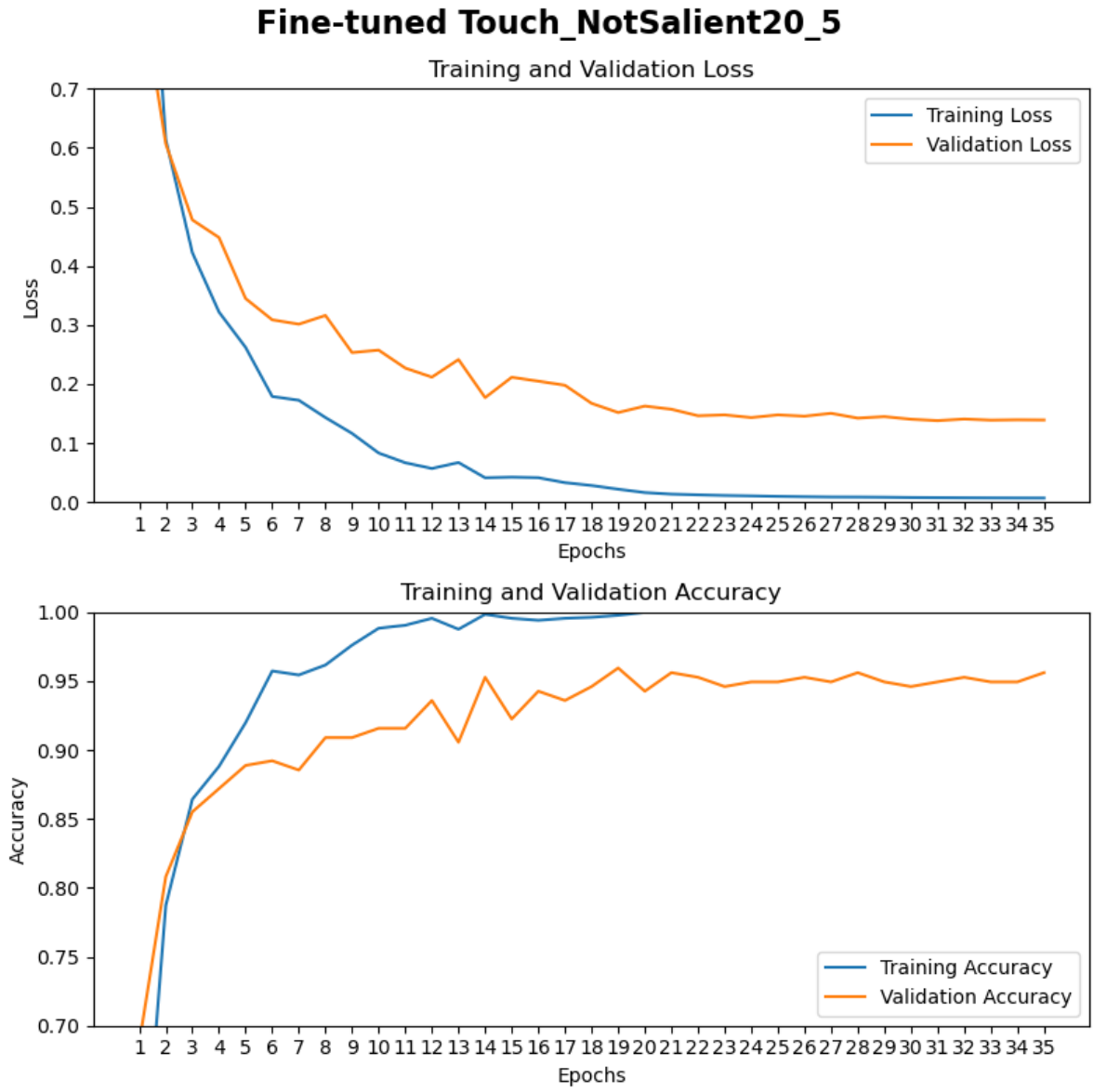

FIGURE H.16 – Historique de l'entraînement du modèle tactile utilisant les 20 points sélectionnés aléatoirement (# 81 à 100) par objet (Touch_NotSalient20_5).

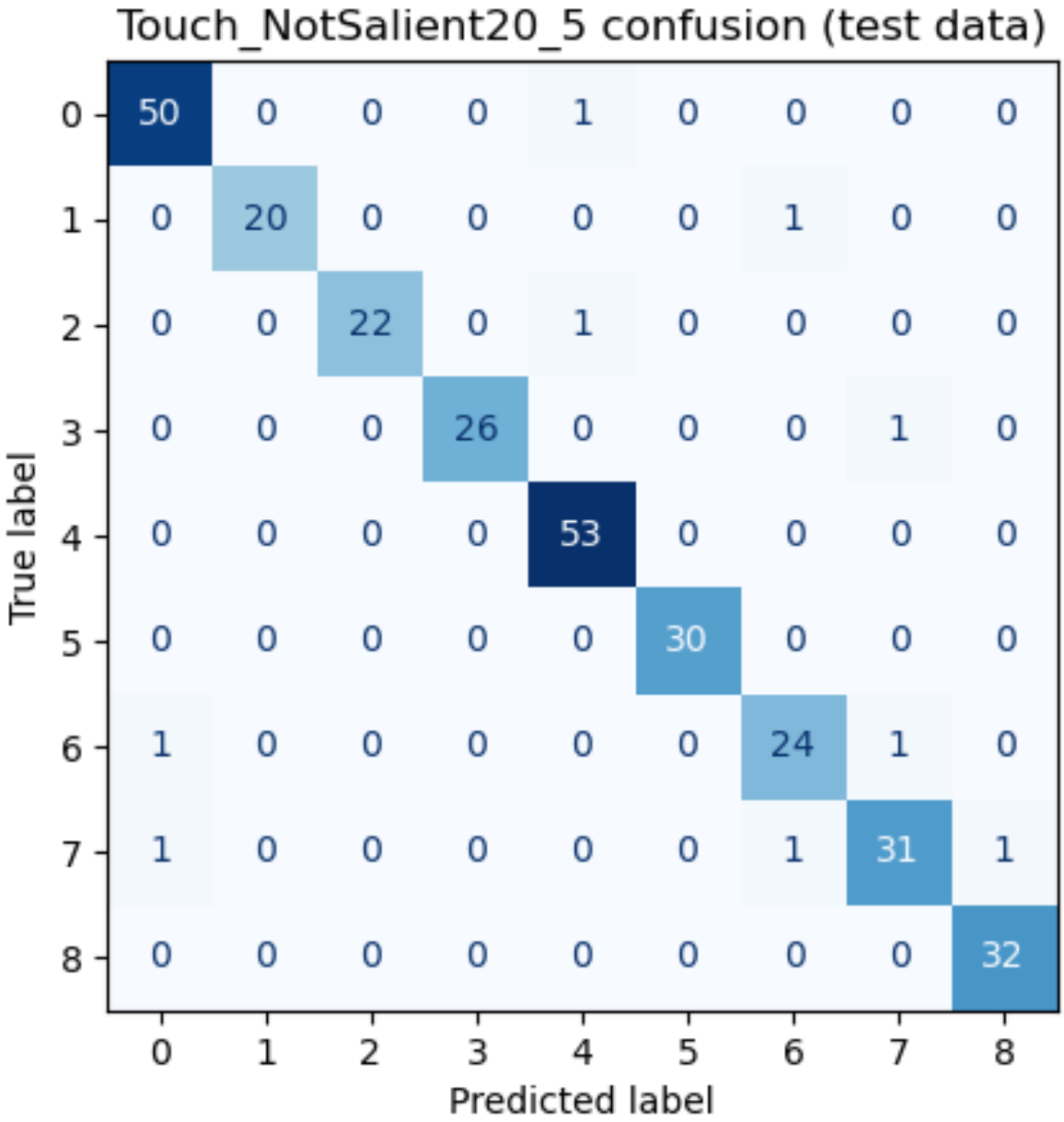

FIGURE H.17 – Matrice de confusion sur les données de test du modèle tactile (Touch_NotSalient20_5).

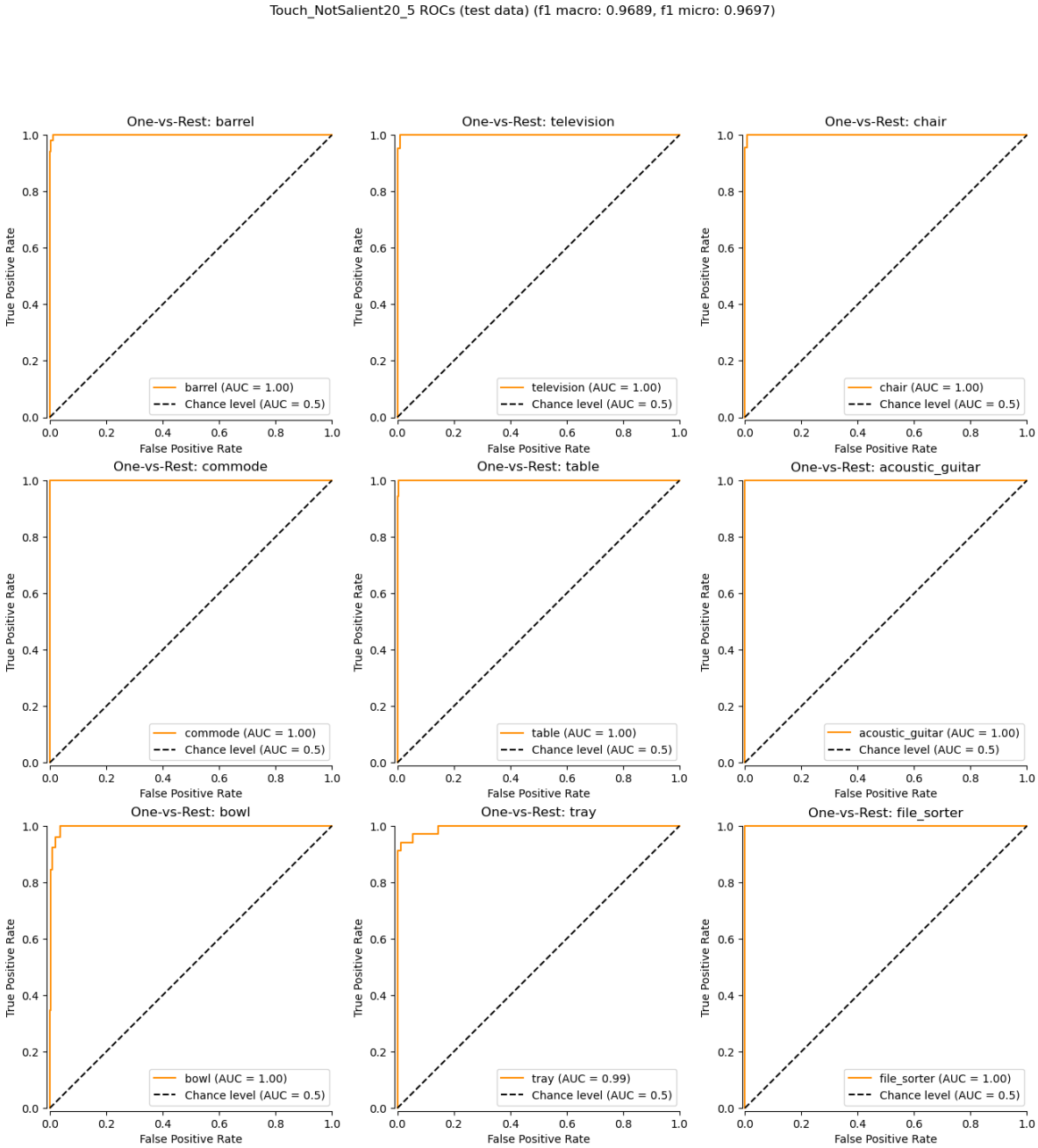

FIGURE H.18 – Courbes ROC et F1 Scores sur les données de test du modèle tactile (Touch_NotSalient20_5).

Annexe I

Résultats de l'entraînement des modèles Audio (Mels)

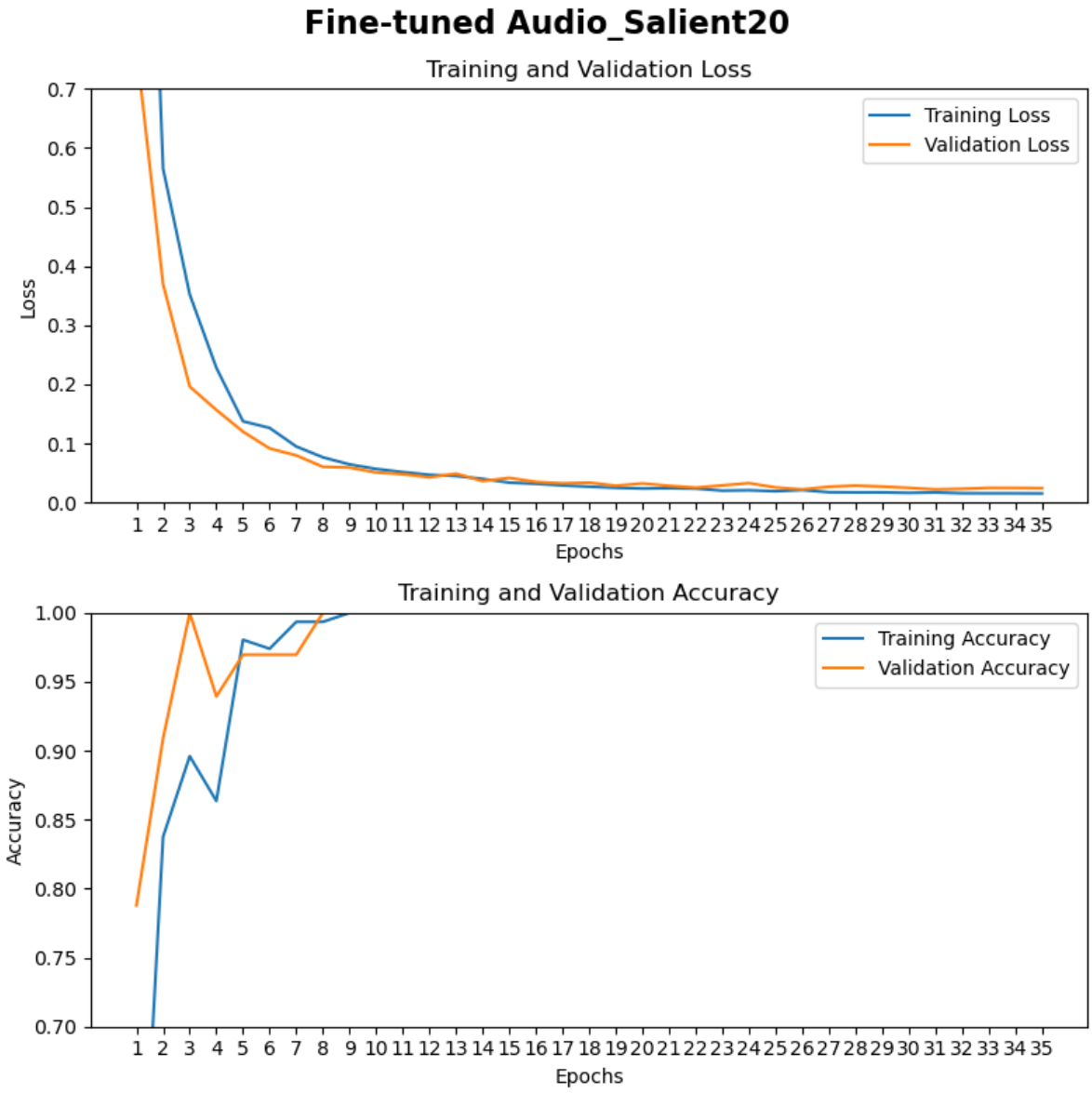

FIGURE I.1 – Historique de l'entraînement du modèle audio (Mels) utilisant les 20 points saillants par objet (Audio_Salient20).

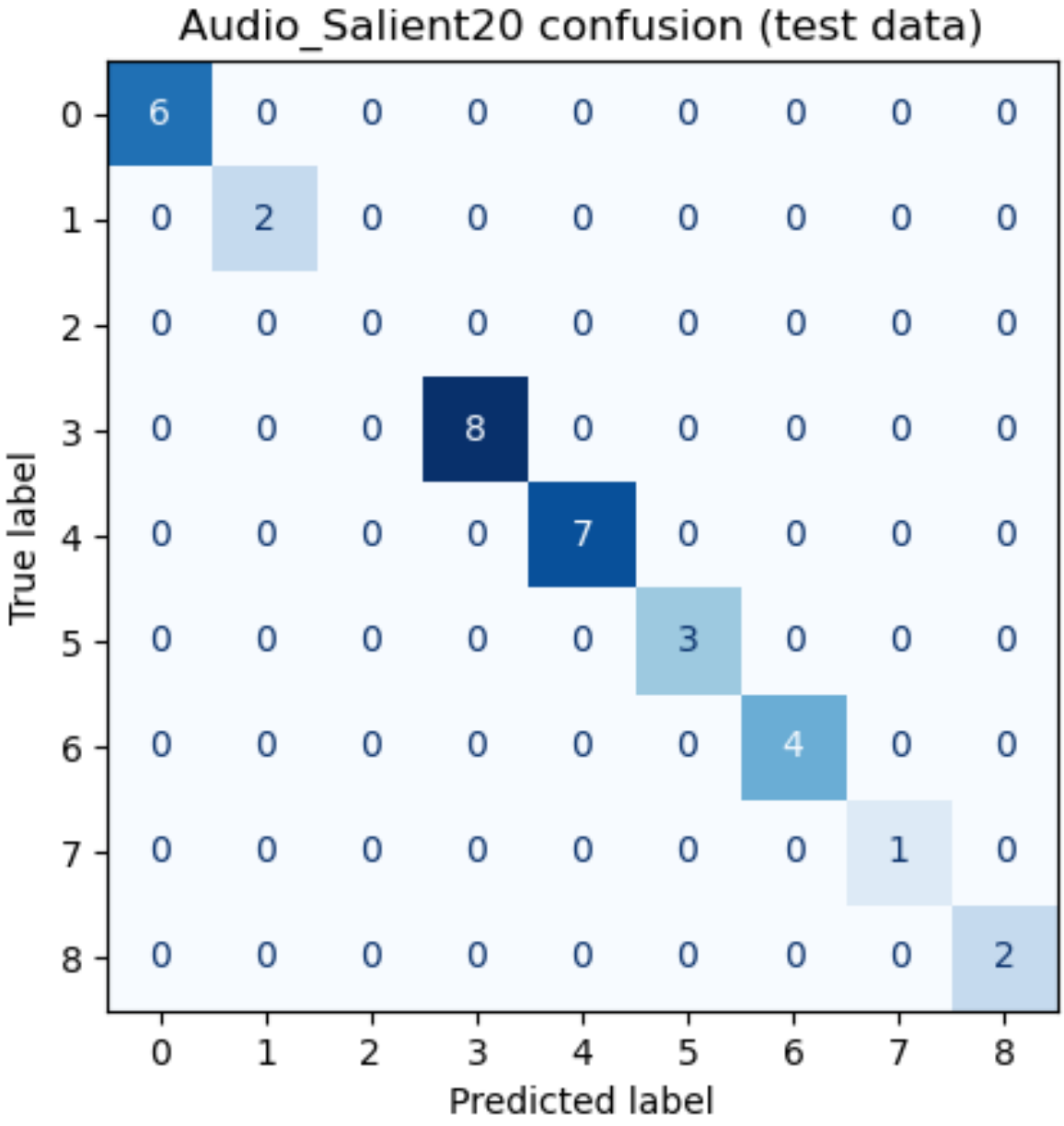

FIGURE I.2 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_Salient20).

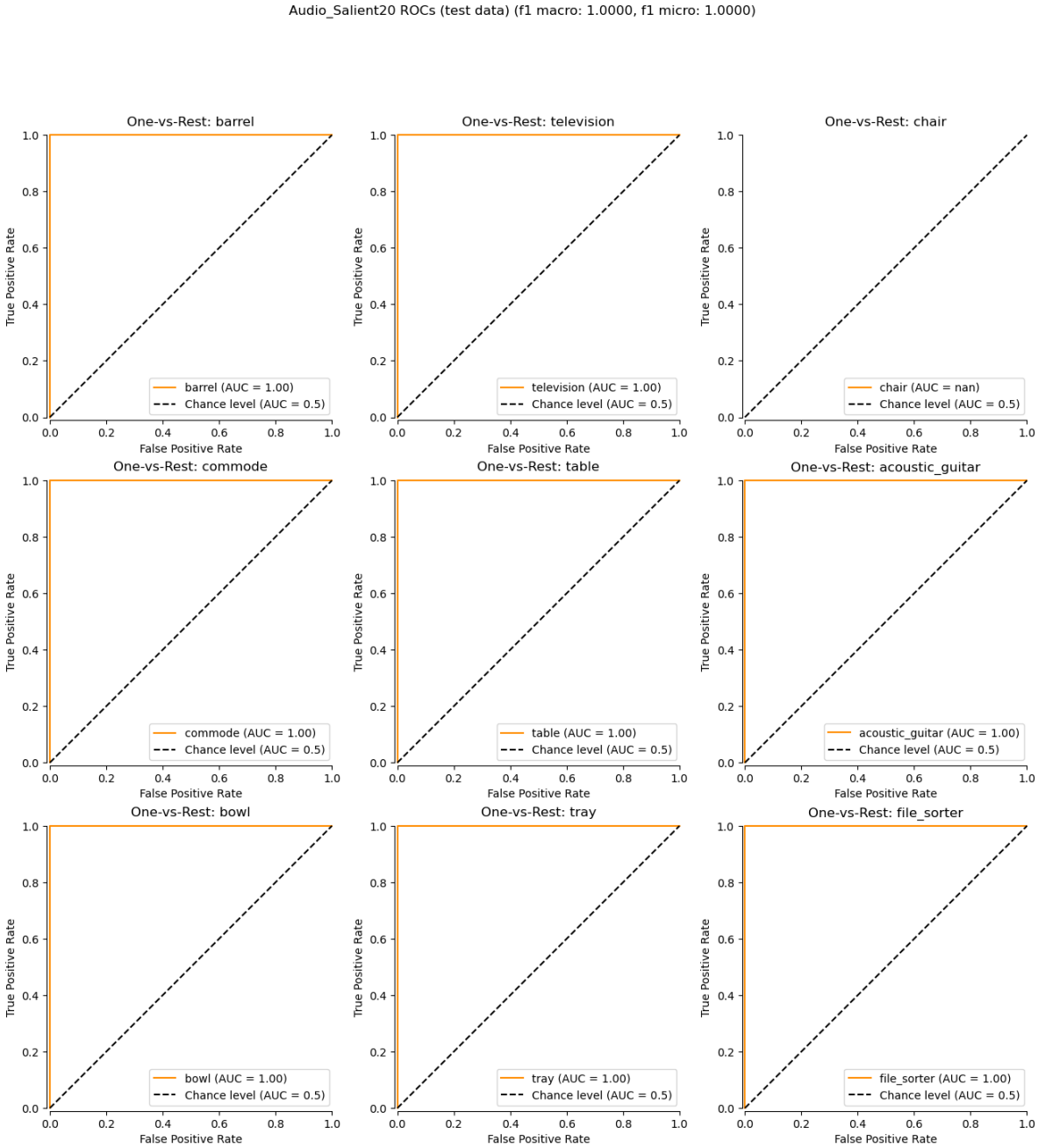

FIGURE I.3 – Courbes ROC et F1 Scores sur les données de test du modèle audio (Mels) (Audio_Salient20).

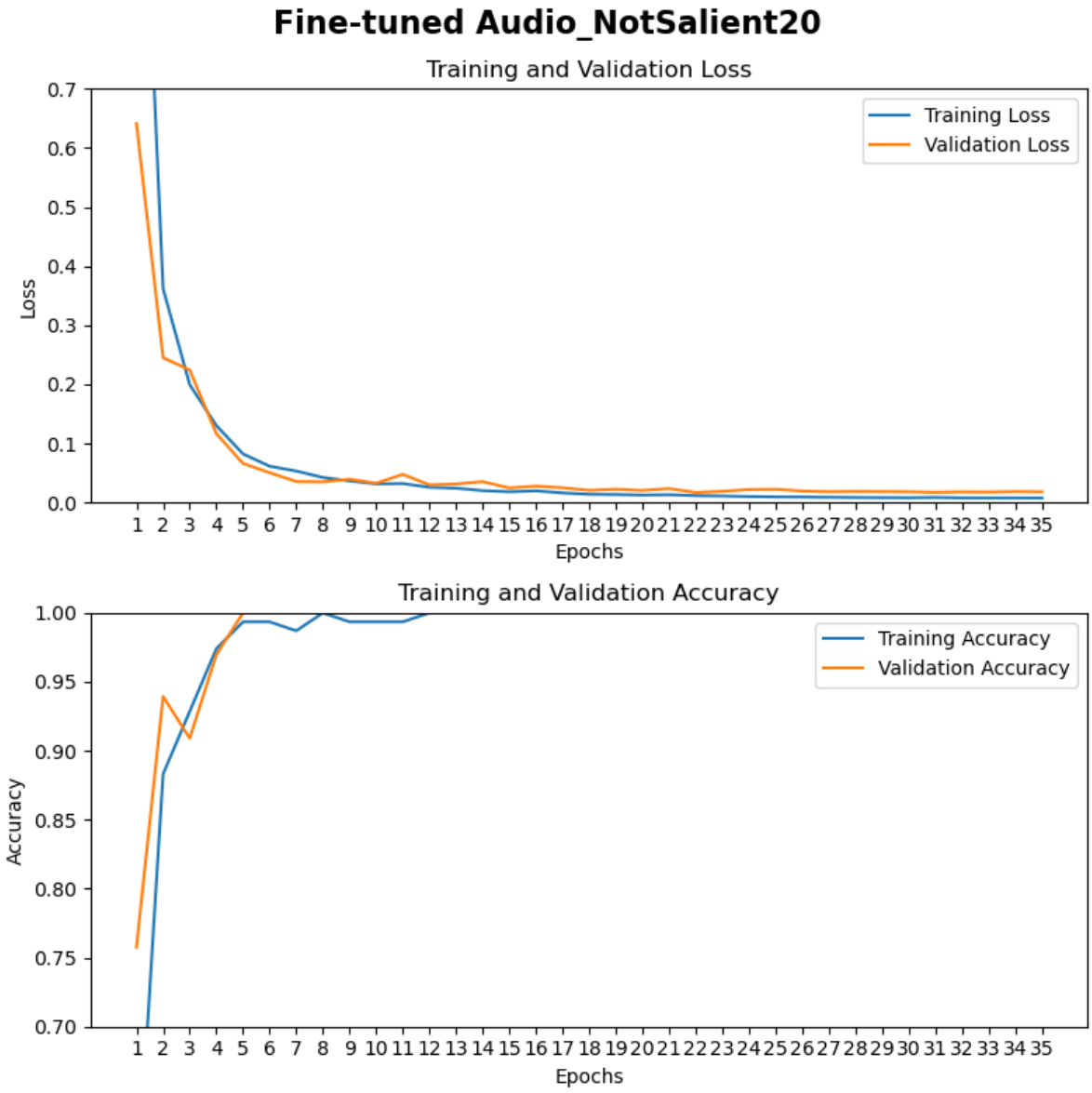

FIGURE I.4 – Historique de l'entraînement du modèle audio (Mels) utilisant les 20 premiers points sélectionnés aléatoirement par objet (Audio_NotSalient20).

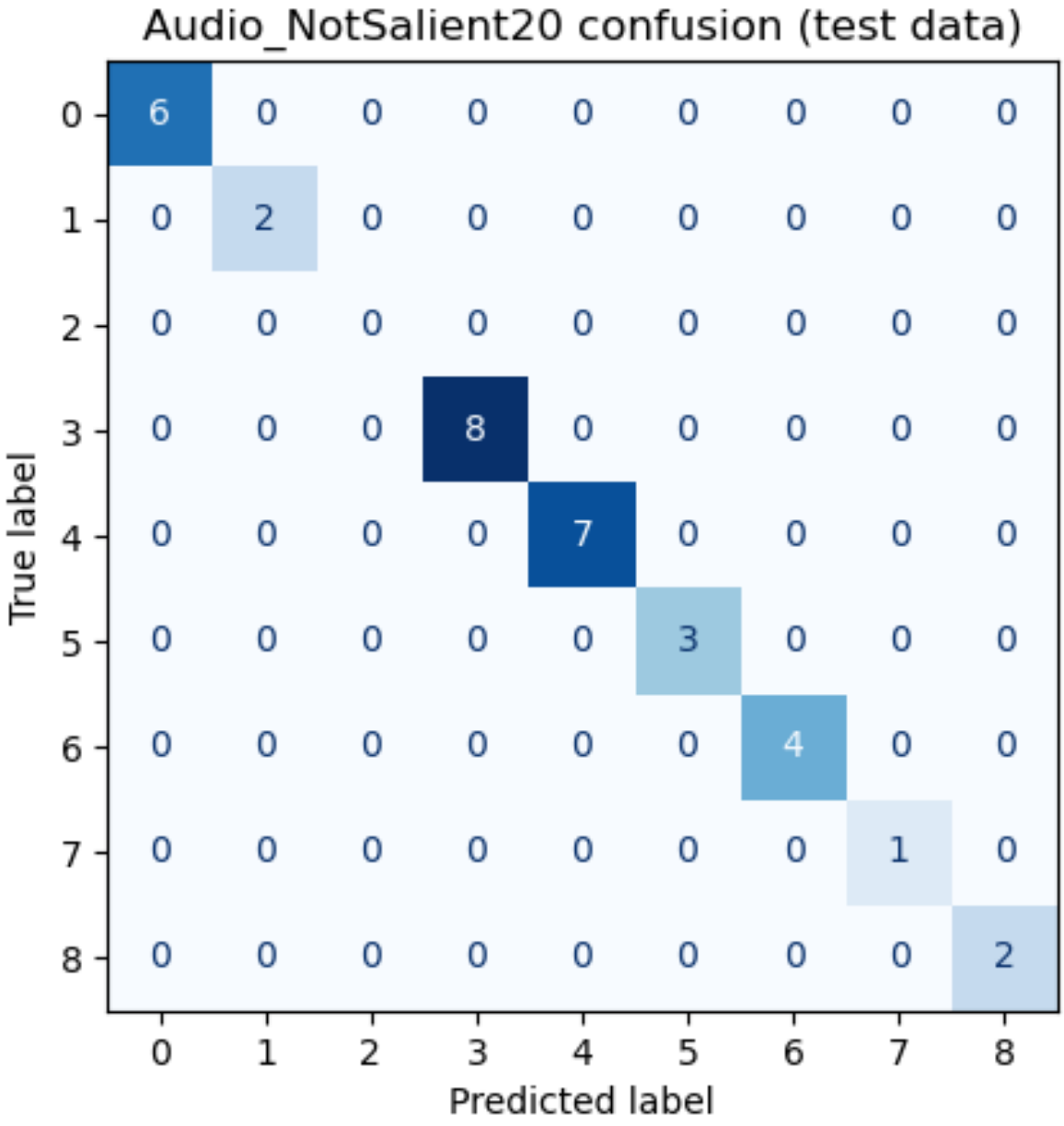

FIGURE I.5 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_NotSalient20).

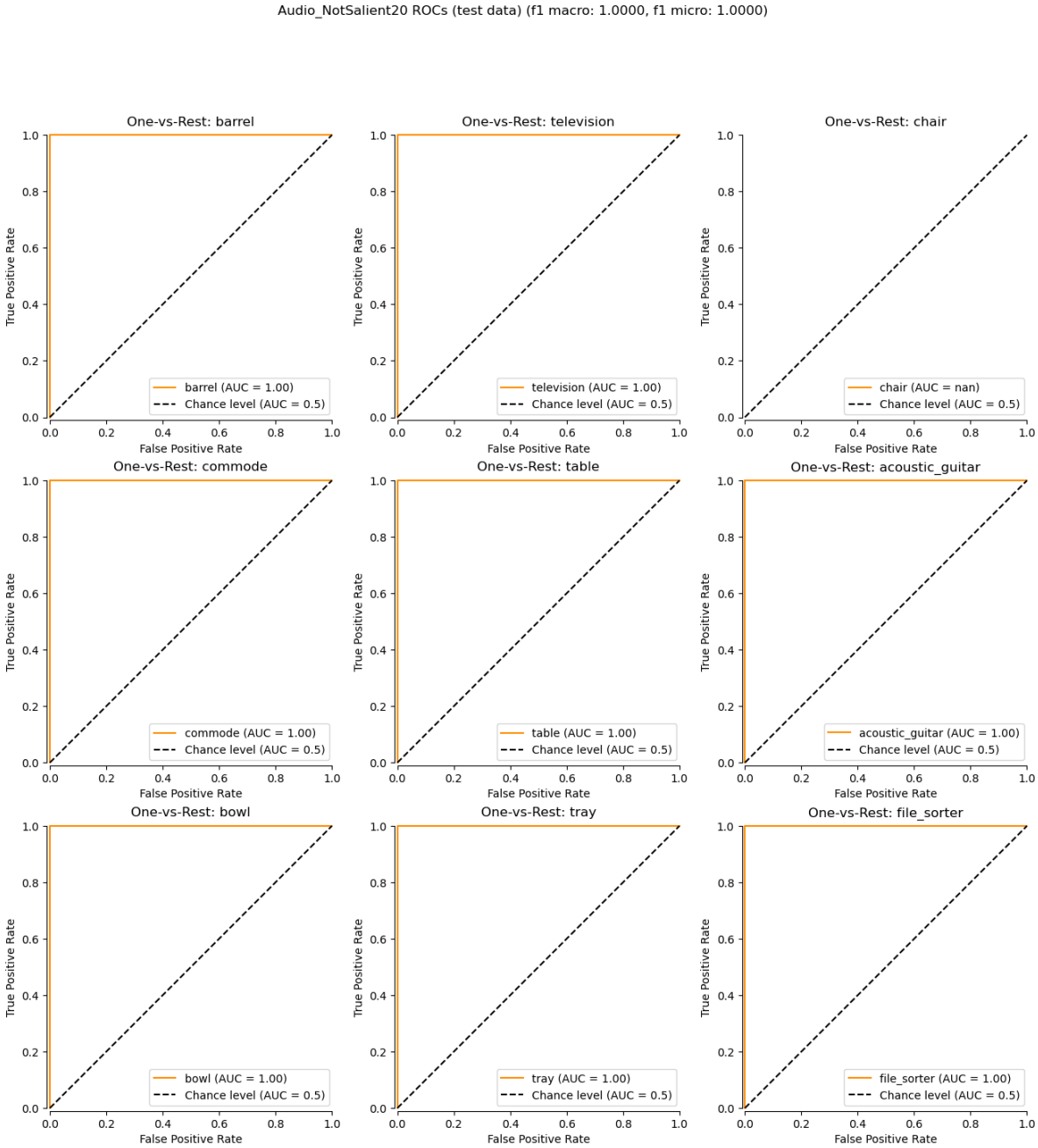

FIGURE I.6 – Courbes ROC et F1 Scores sur les données de test du modèle audio (Mels) (Audio_NotSalient20).

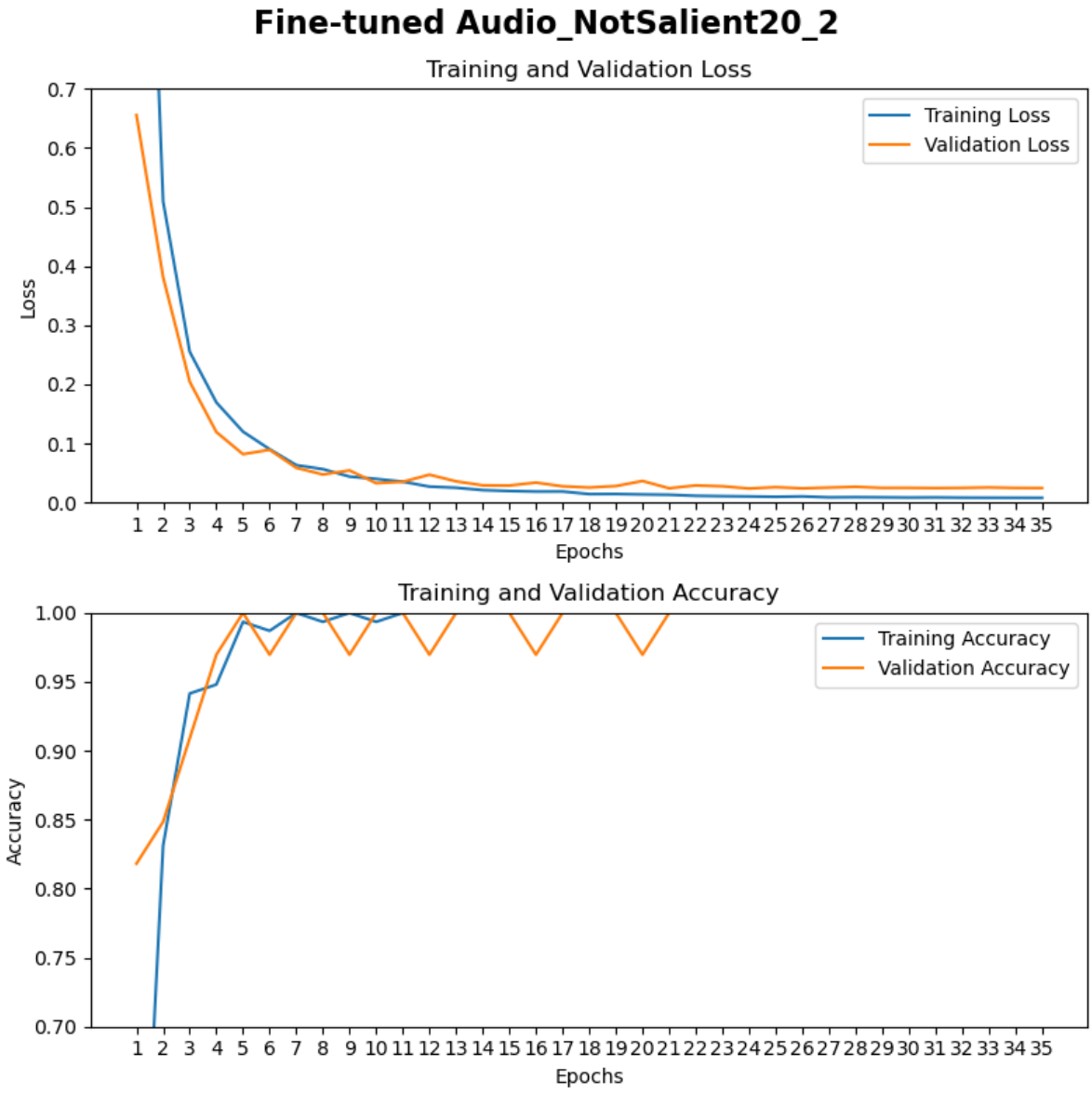

FIGURE I.7 – Historique de l'entraînement du modèle audio (Mels) utilisant les 20 points sélectionnés aléatoirement (# 21 à 40) par objet (Audio_NotSalient20_2).

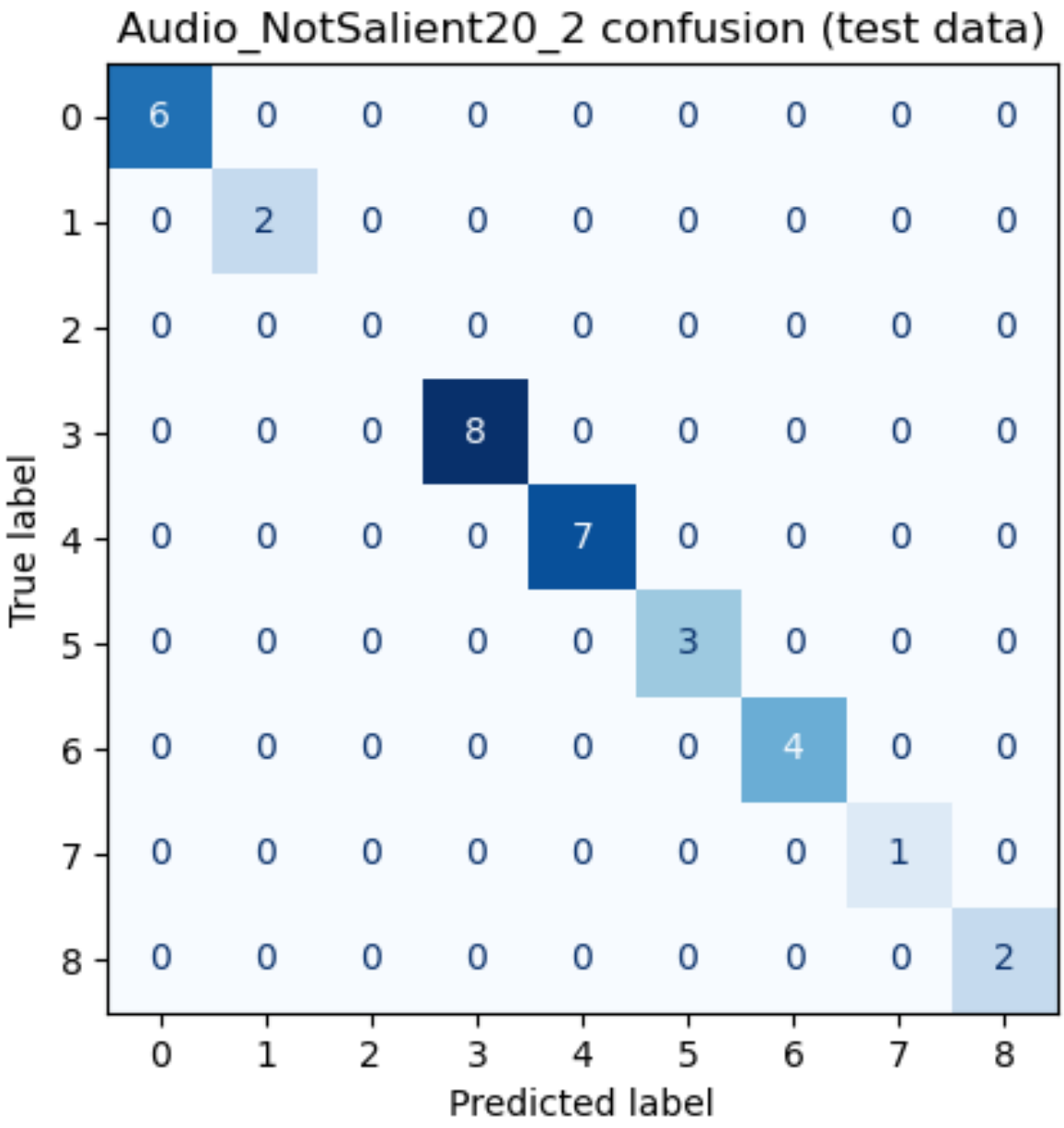

FIGURE I.8 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_NotSalient20_2).

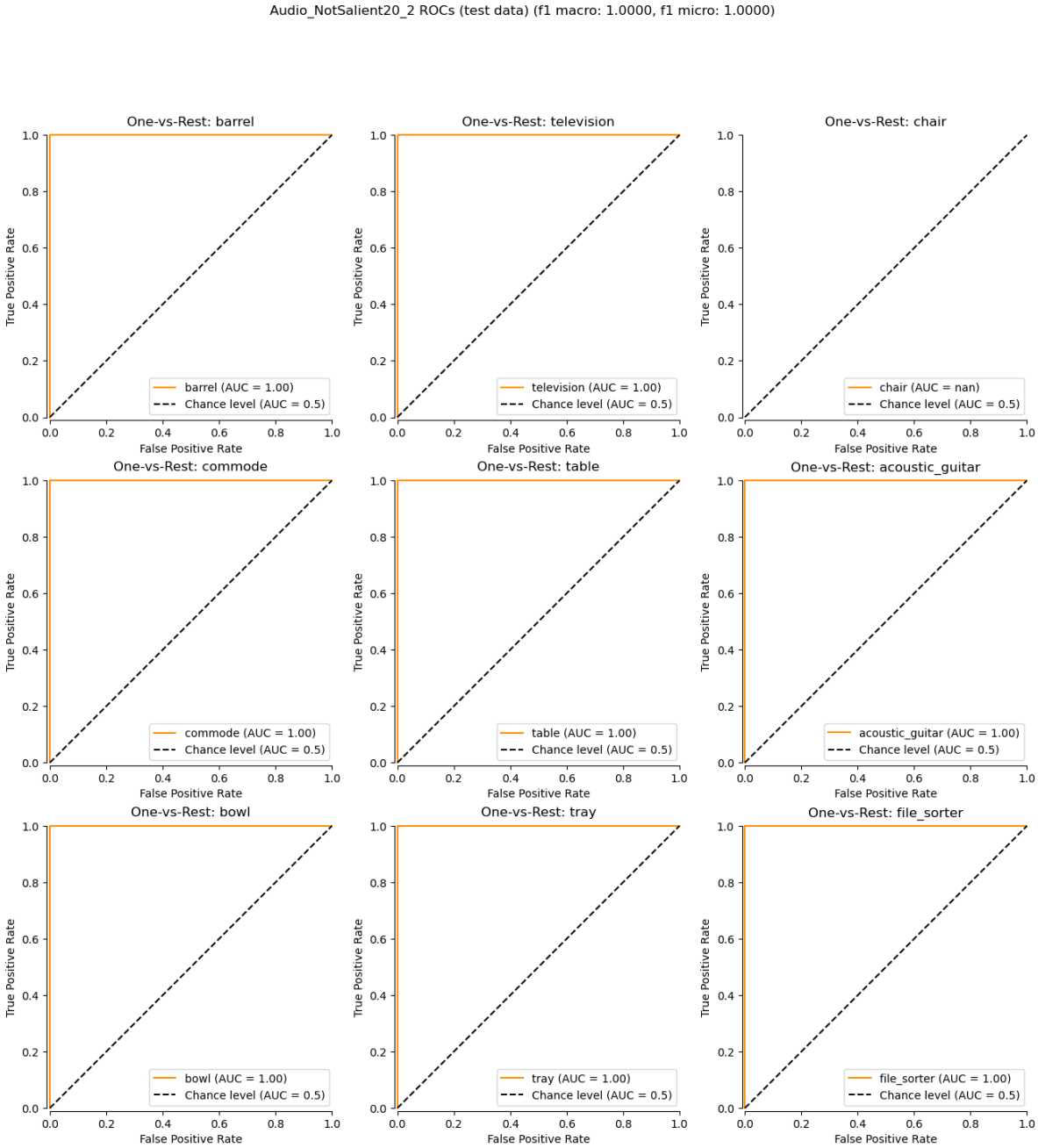

FIGURE I.9 – Courbes ROC et F1 Scores sur les données de test du modèle audio (Mels) (Audio_NotSalient20_2).

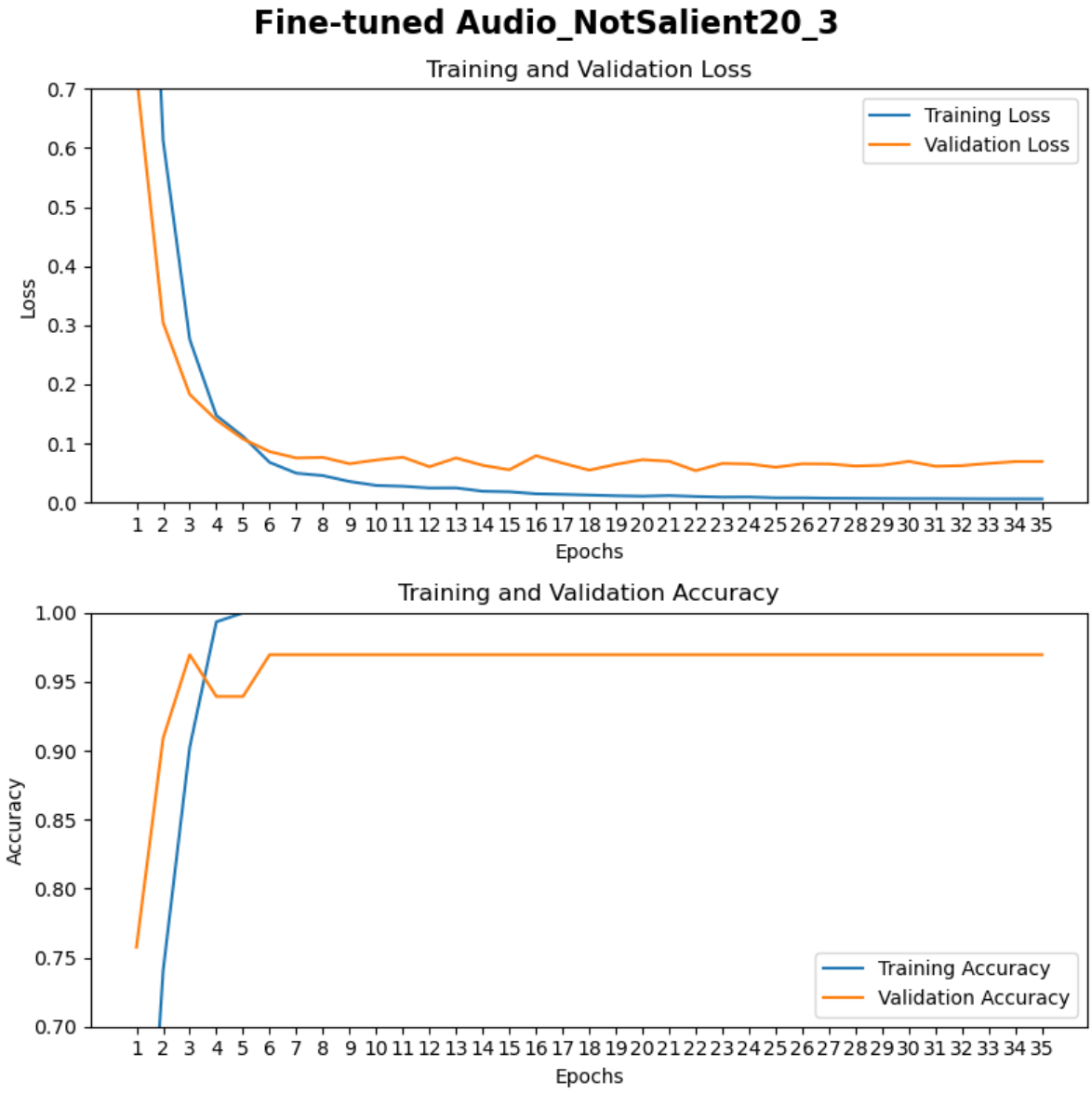

FIGURE I.10 – Historique de l'entraînement du modèle audio (Mels) utilisant les 20 points sélectionnés aléatoirement (# 41 à 60) par objet (Audio_NotSalient20_3).

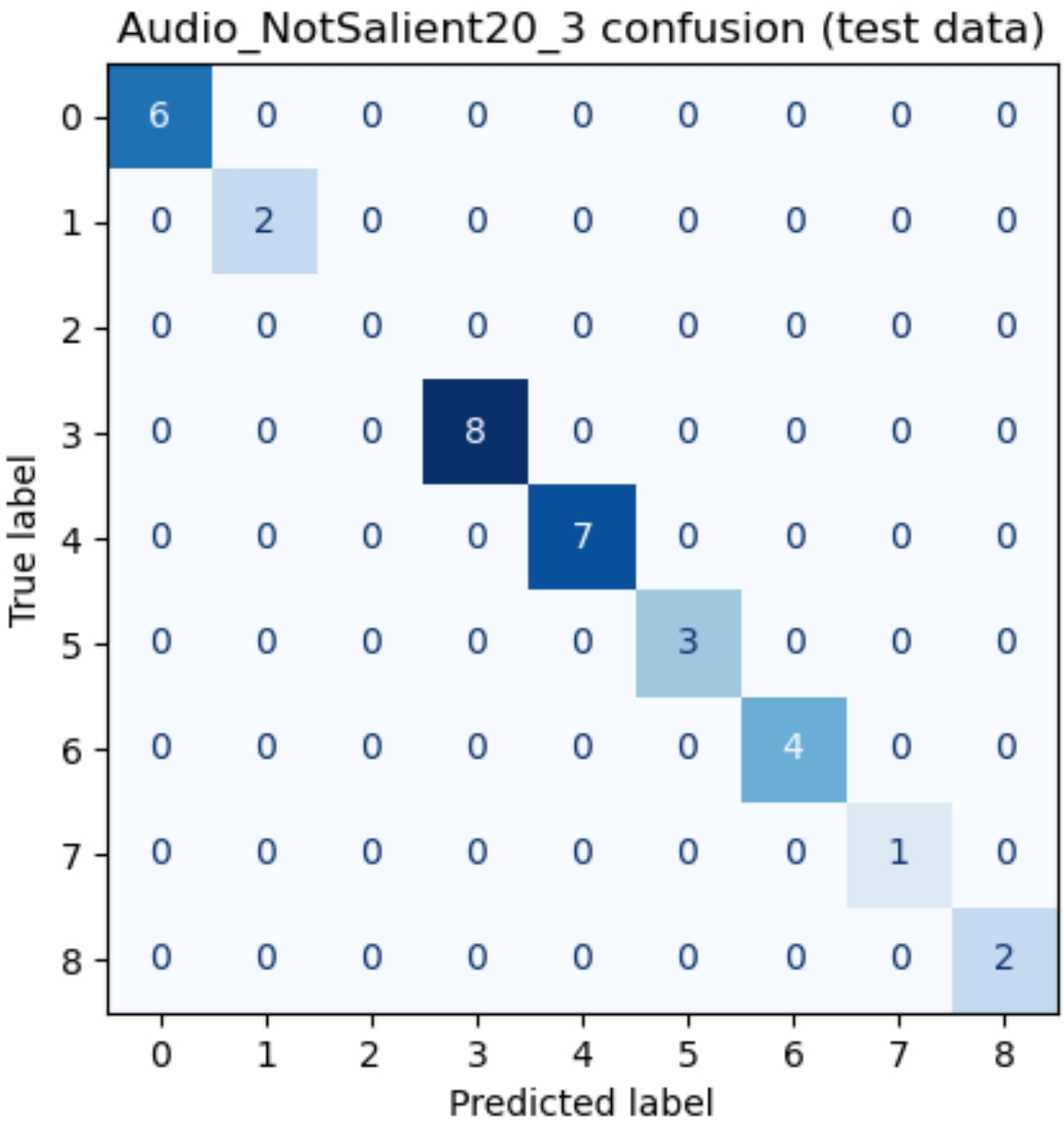

FIGURE I.11 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_NotSalient20_3).

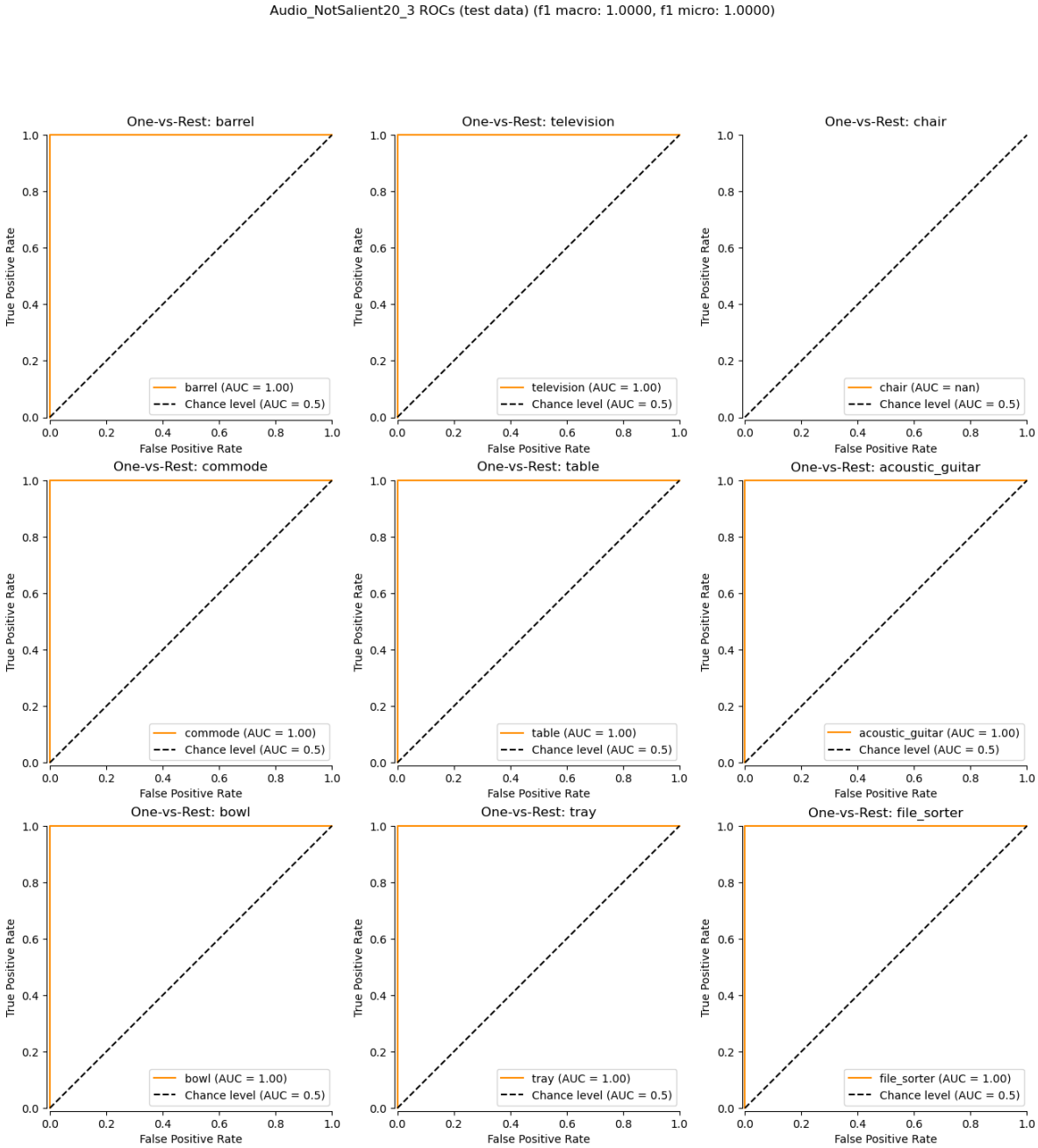

FIGURE I.12 – Courbes ROC et F1 Scores sur les données de test du modèle audio (Mels) (Audio_NotSalient20_3).

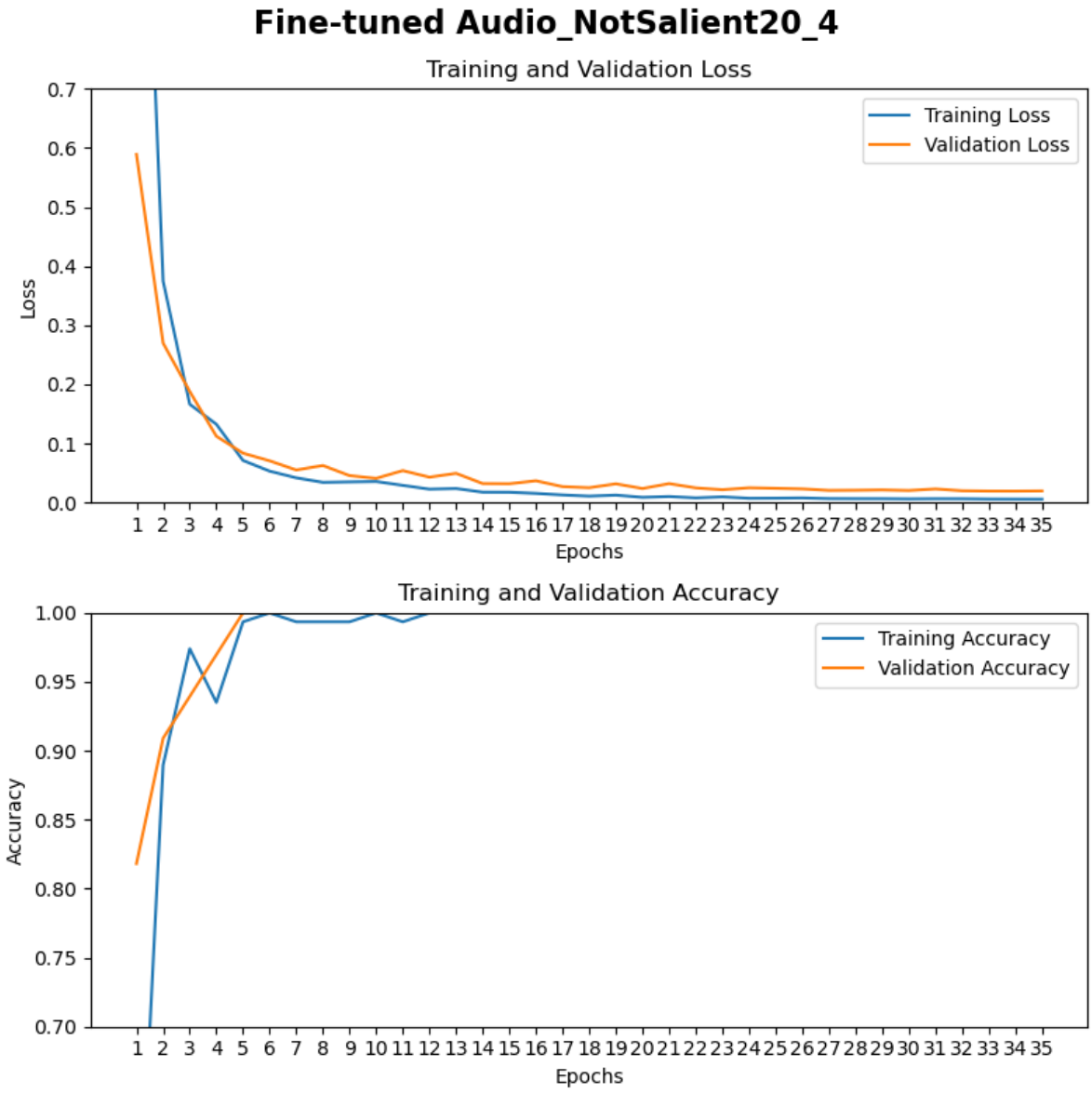

FIGURE I.13 – Historique de l'entraînement du modèle audio (Mels) utilisant les 20 points sélectionnés aléatoirement (# 61 à 80) par objet (Audio_NotSalient20_4).

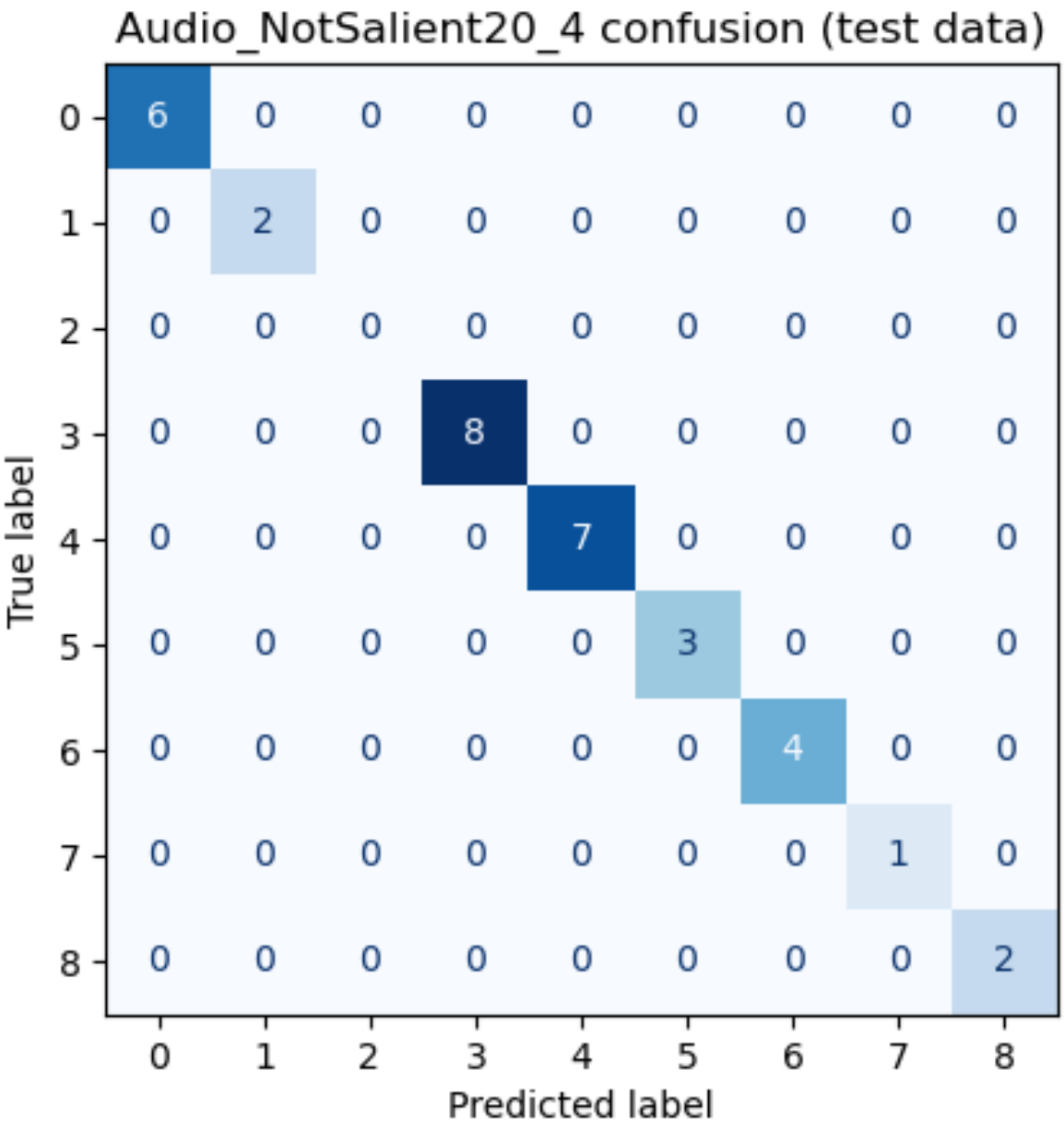

FIGURE I.14 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_NotSalient20_4).

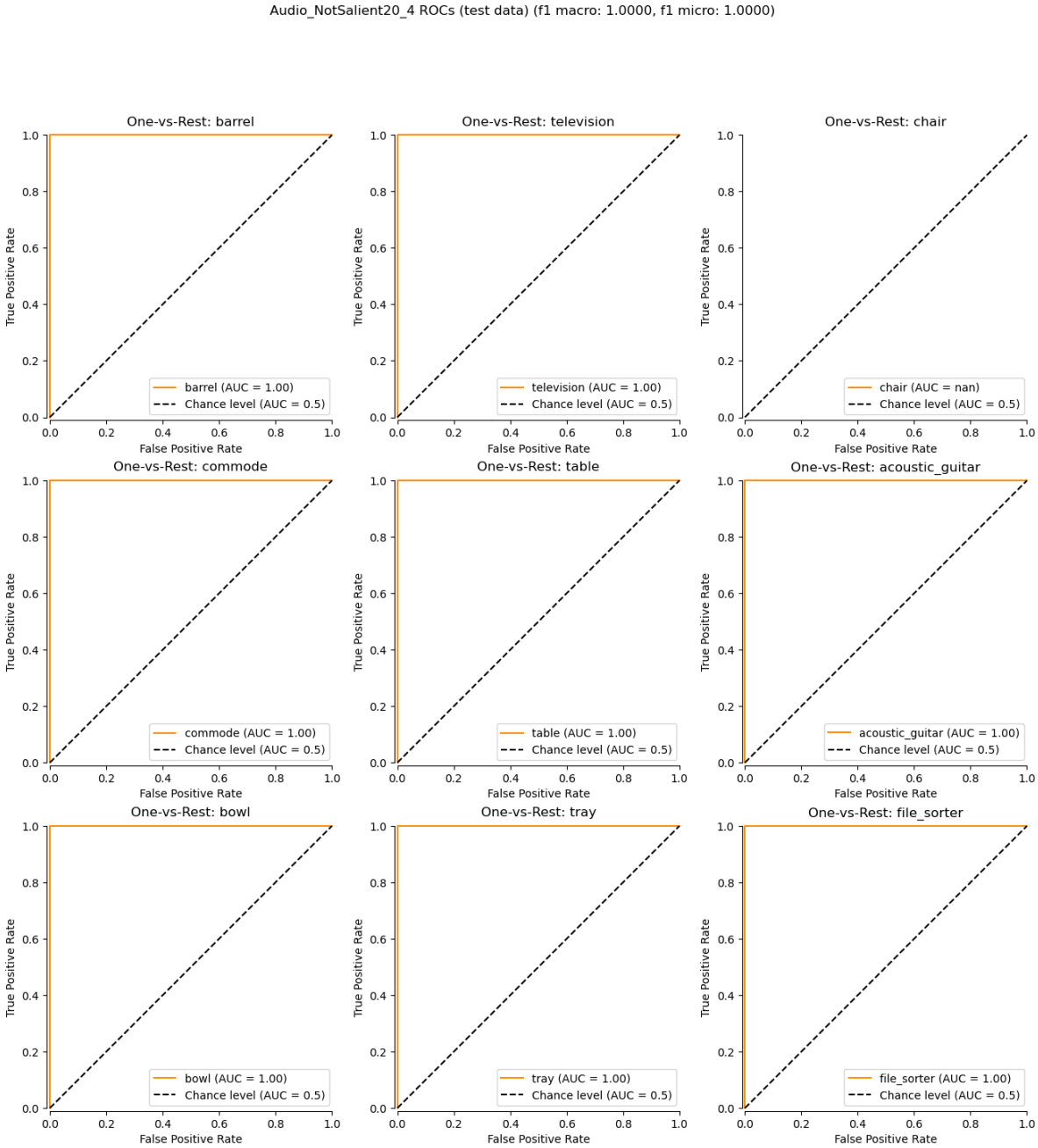

FIGURE I.15 – Courbes ROC et F1 Scores sur les données de test du modèle audio (Mels) (Audio_NotSalient20_4).

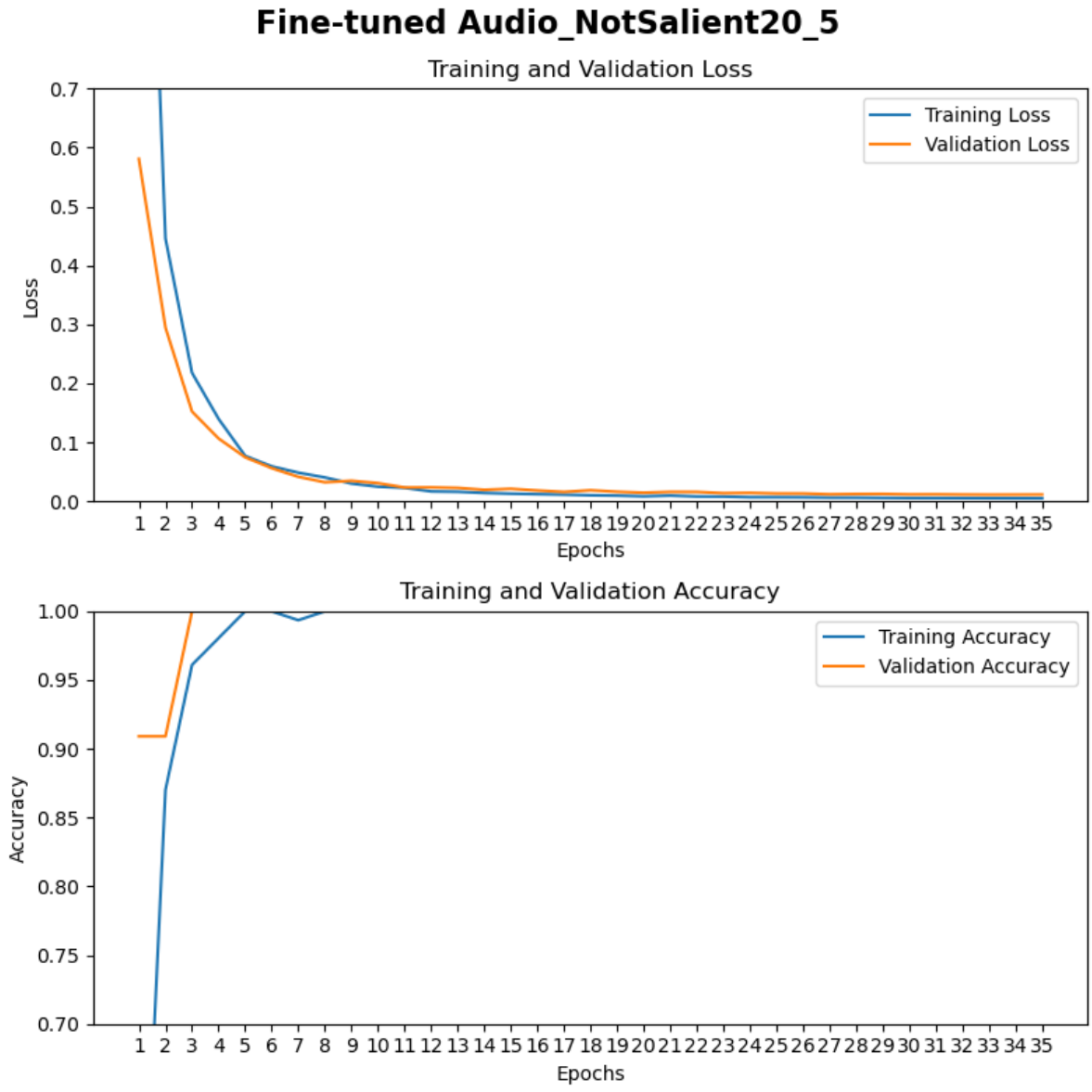

FIGURE I.16 – Historique de l'entraînement du modèle audio (Mels) utilisant les 20 points sélectionnés aléatoirement (# 81 à 100) par objet (Audio_NotSalient20_5).

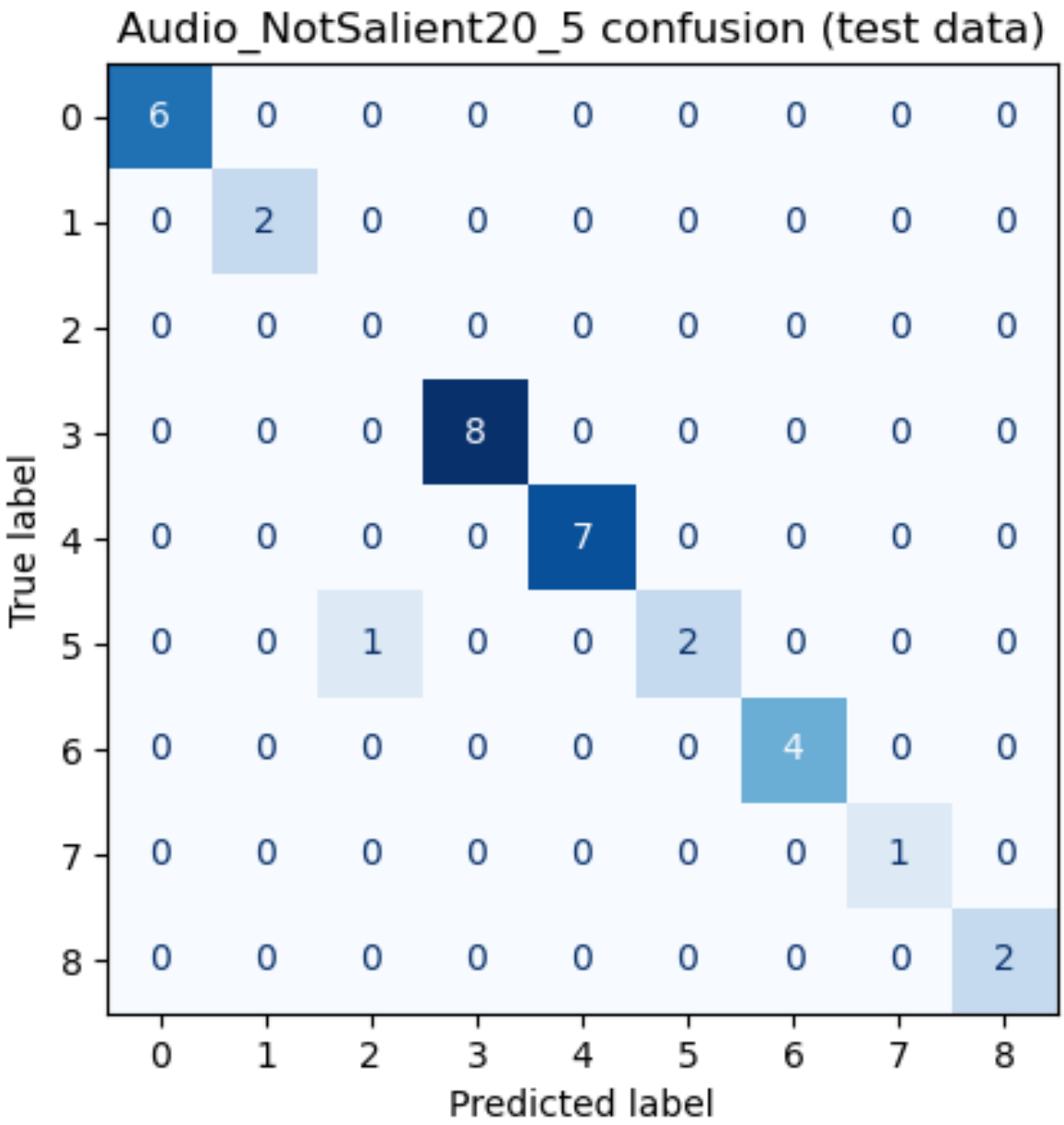

FIGURE I.17 – Matrice de confusion sur les données de test du modèle audio (Mels) (Audio_NotSalient20_5).

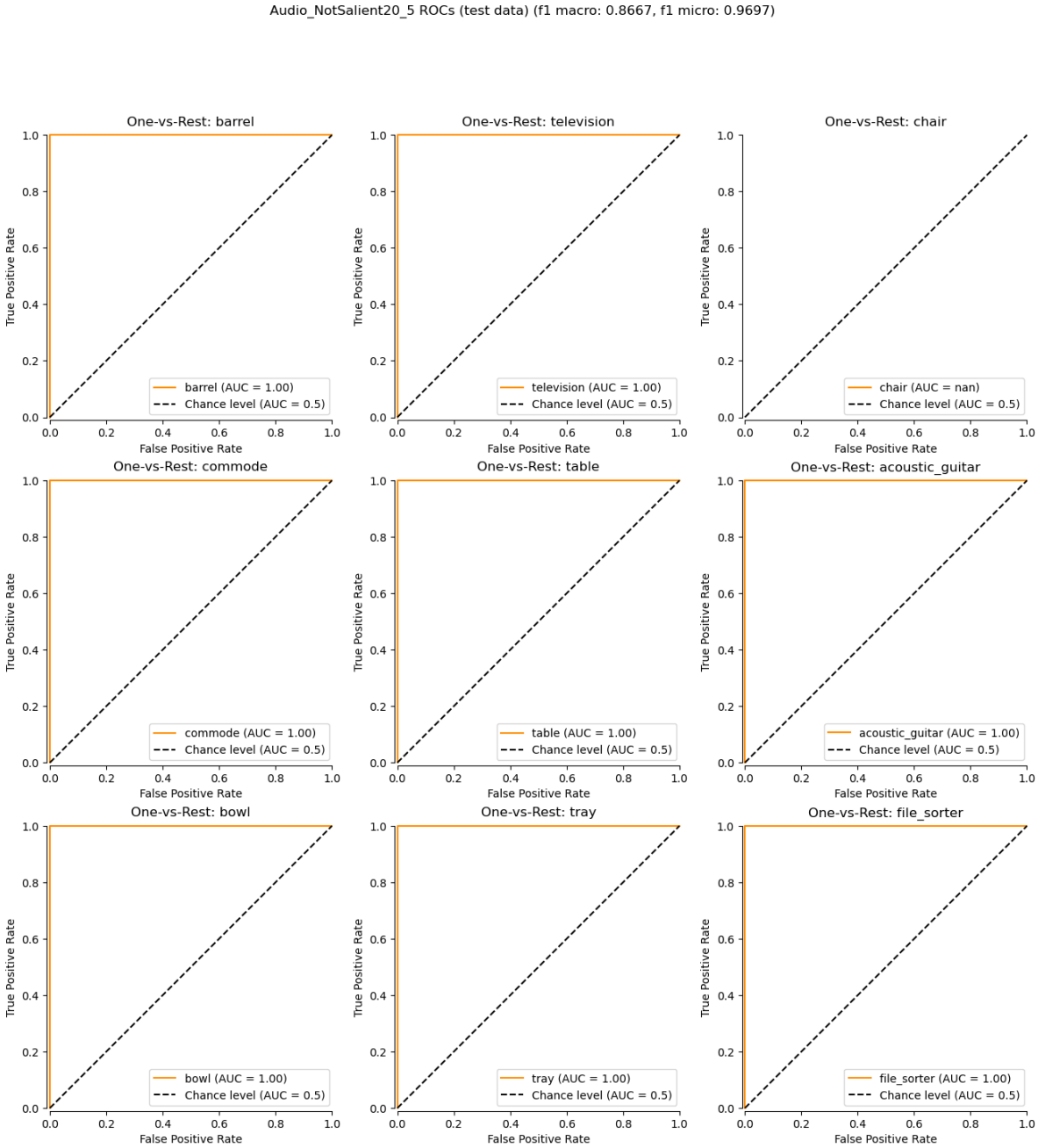

FIGURE I.18 – Courbes ROC et F1 Scores sur les données de test du modèle audio (Mels) (Audio_NotSalient20_5).

Annexe J

Résultats de l'entraînement des modèles Audio (MFCC)

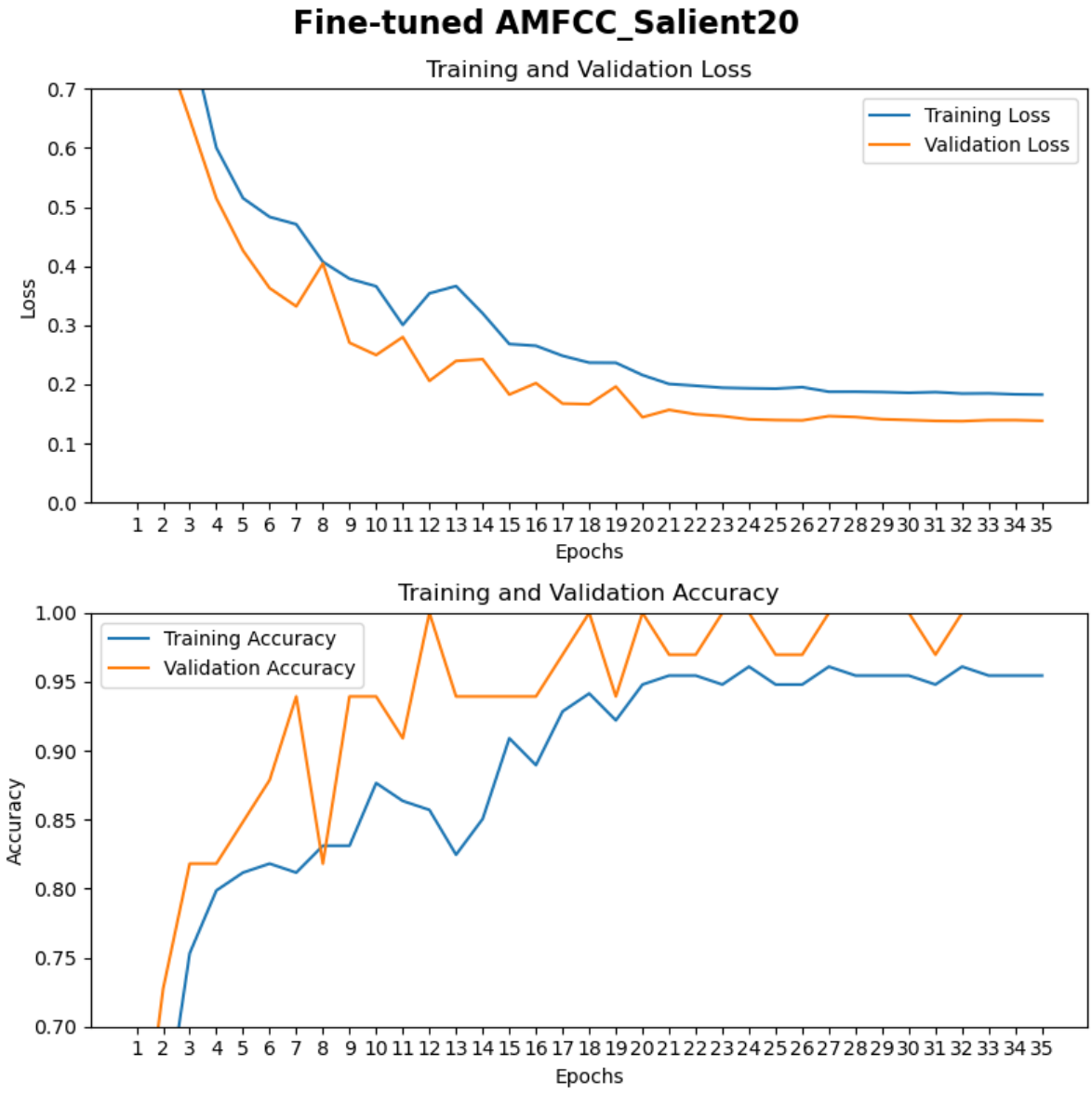

FIGURE J.1 – Historique de l'entraînement du modèle audio (MFCC) utilisant les 20 points saillants par objet (AMFCC_Salient20).

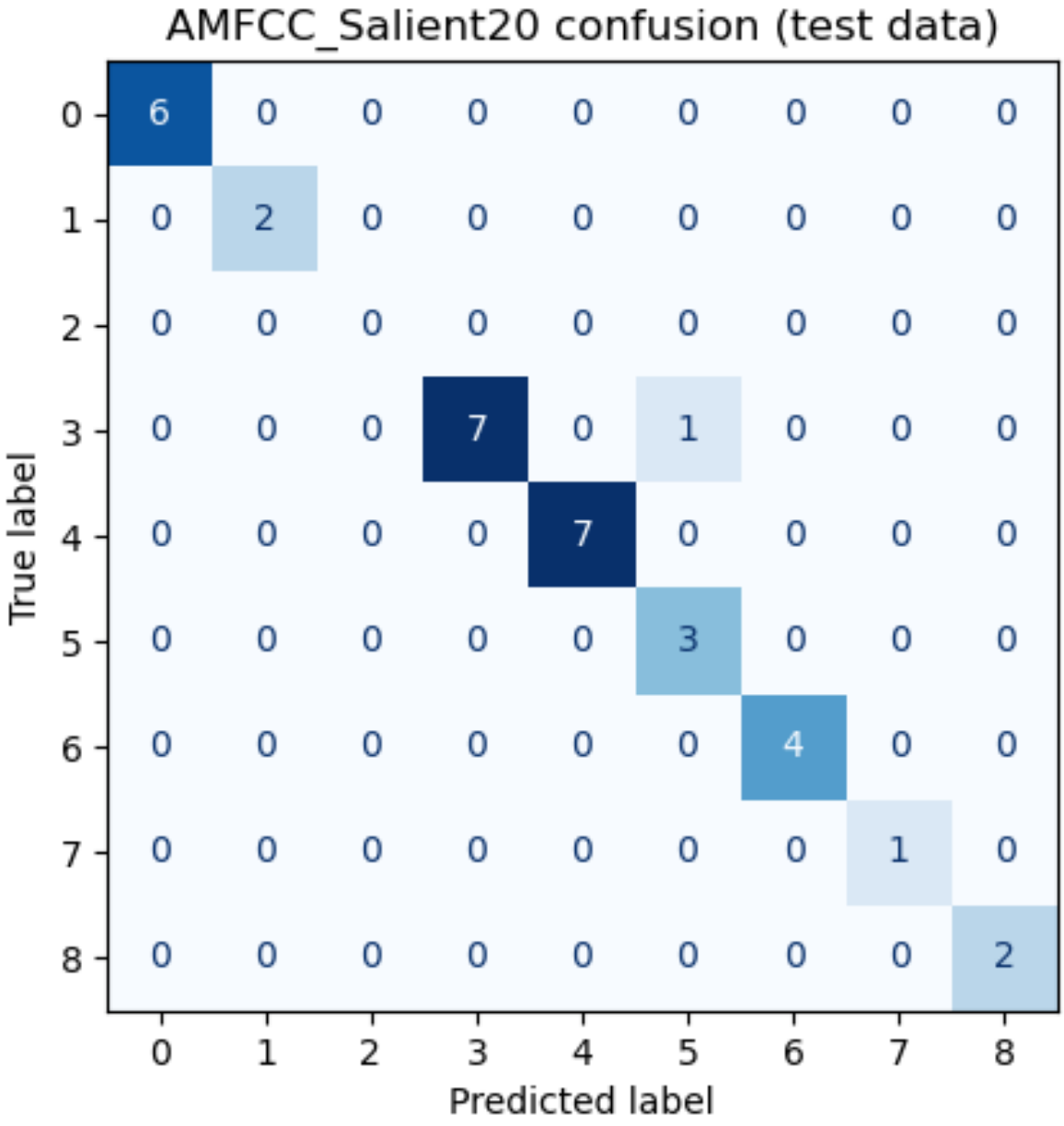

FIGURE J.2 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_Salient20).

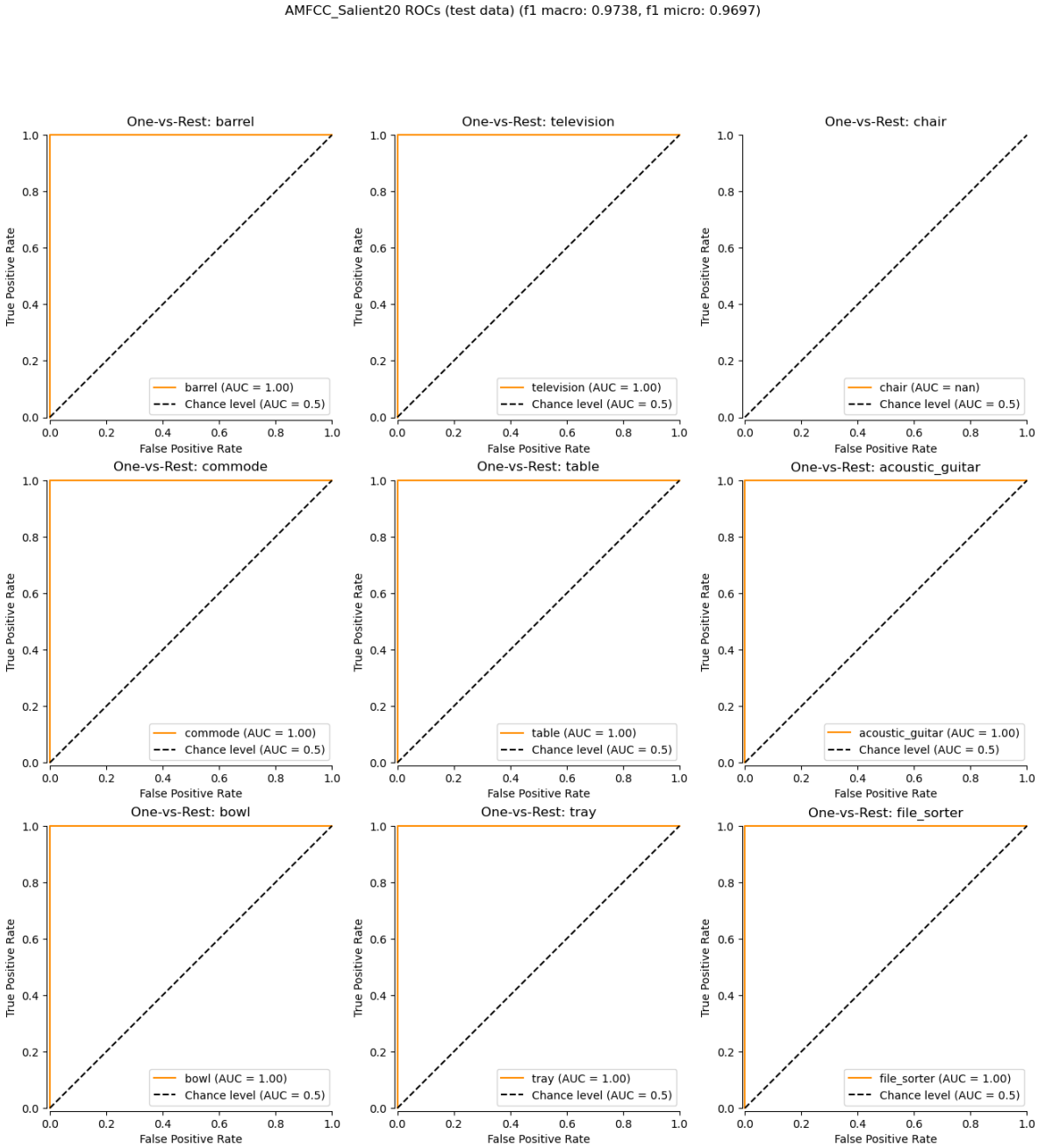

FIGURE J.3 – Courbes ROC et F1 Scores sur les données de test du modèle audio (MFCC) (AMFCC_Salient20).

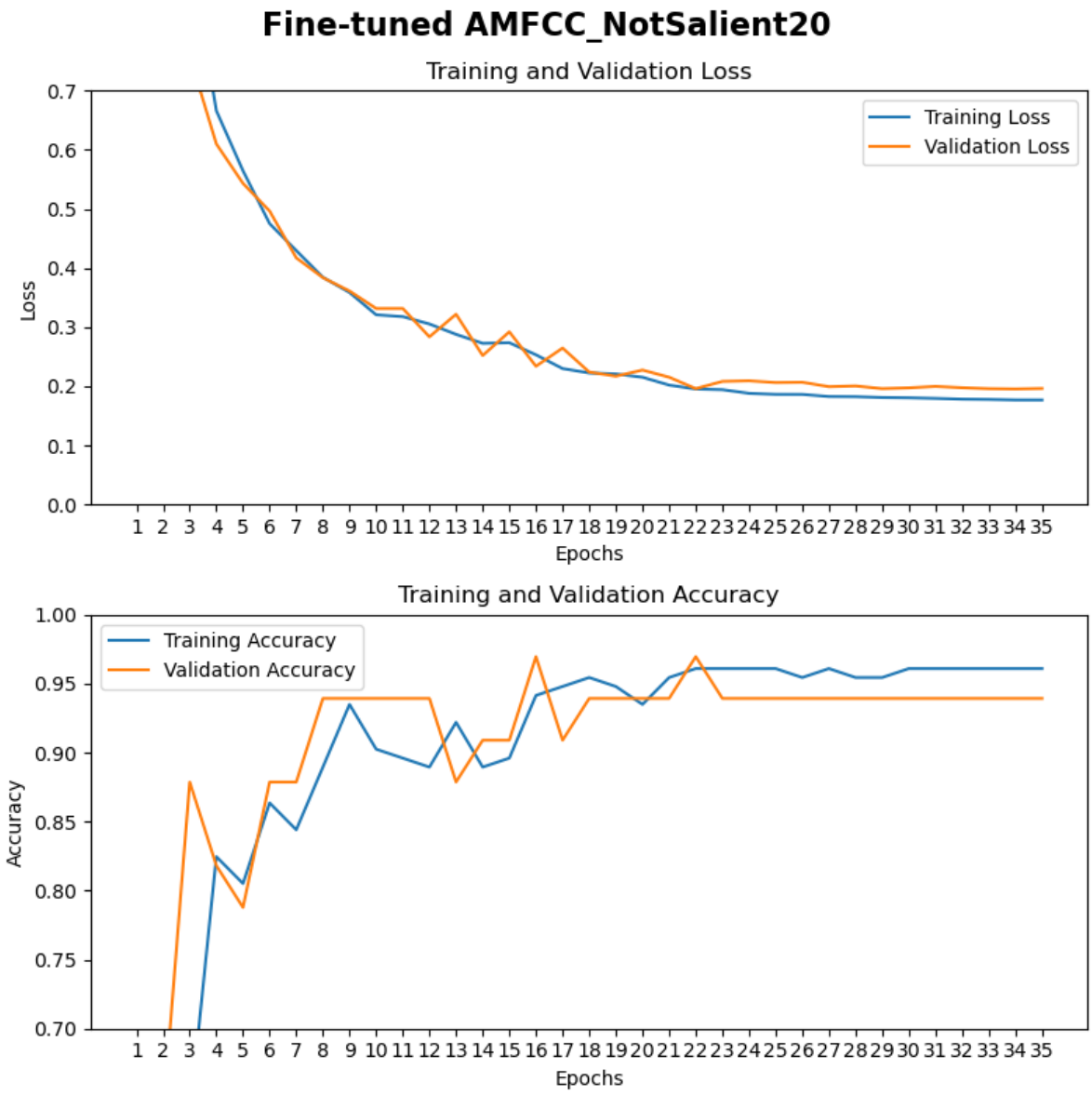

FIGURE J.4 – Historique de l'entraînement du modèle audio (MFCC) utilisant les 20 premiers points sélectionnés aléatoirement par objet (AMFCC_NotSalient20).

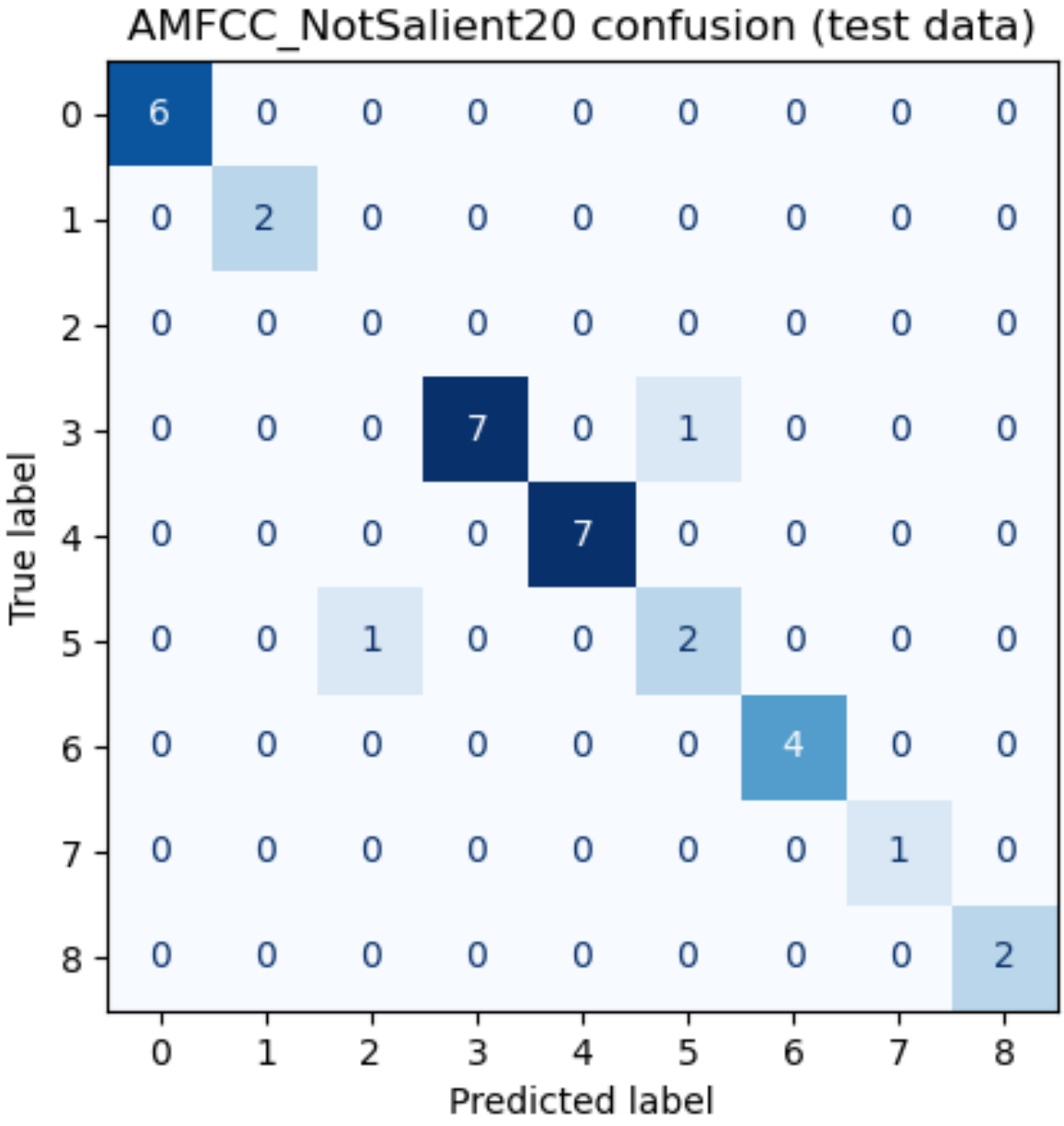

FIGURE J.5 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20).

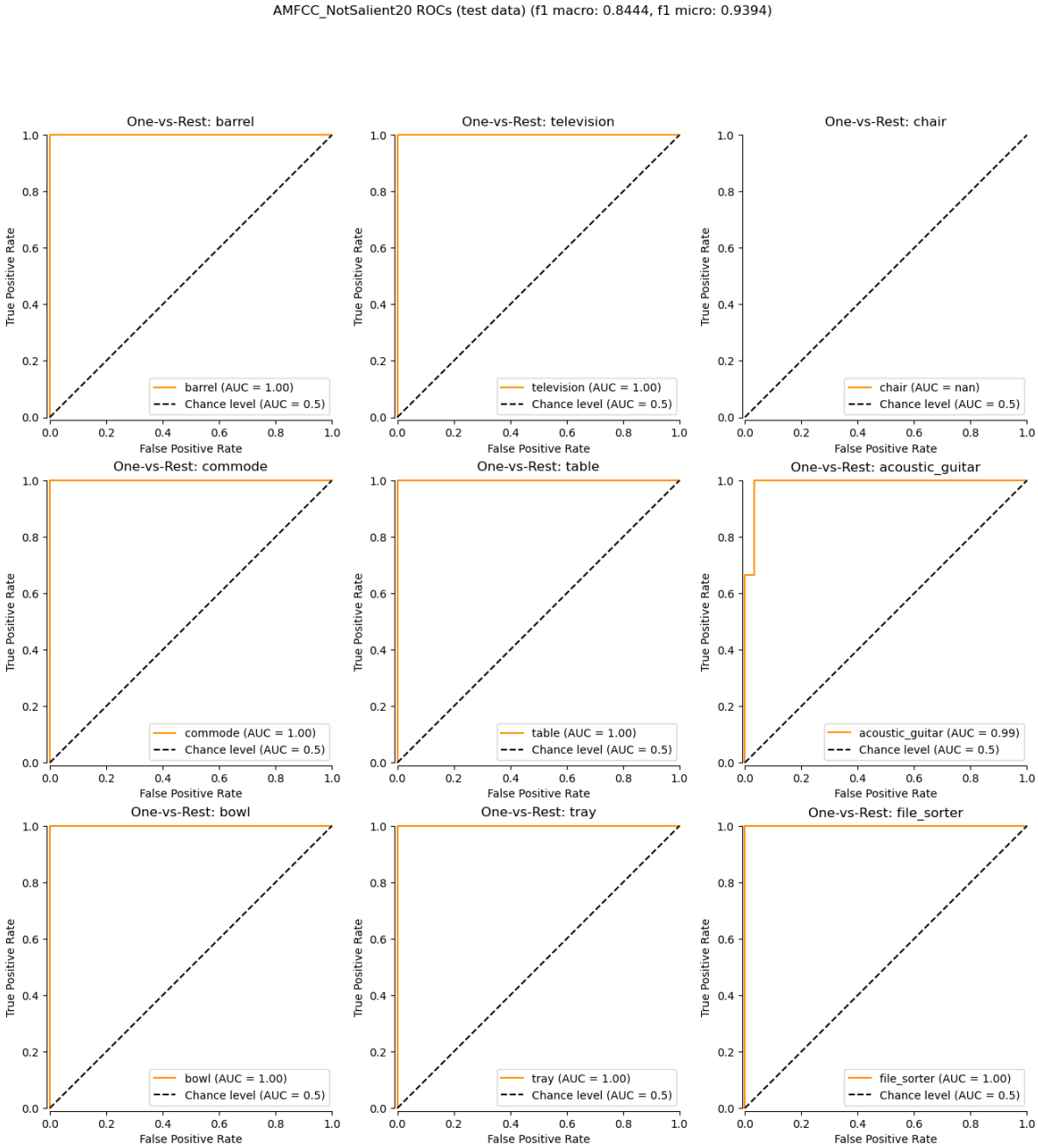

FIGURE J.6 – Courbes ROC et F1 Scores sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20).

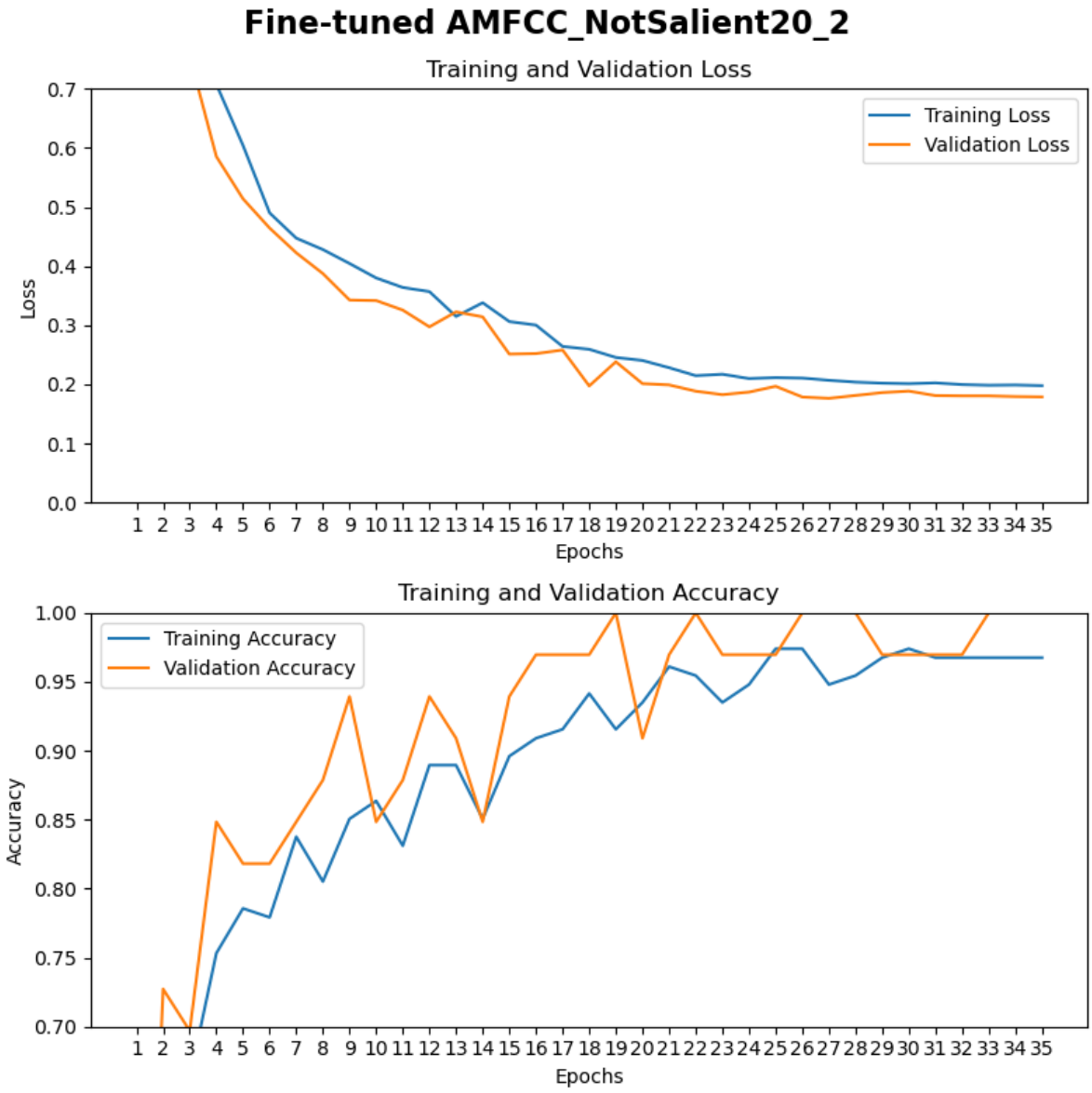

FIGURE J.7 – Historique de l'entraînement du modèle audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 21 à 40) par objet (AMFCC_NotSalient20_2).

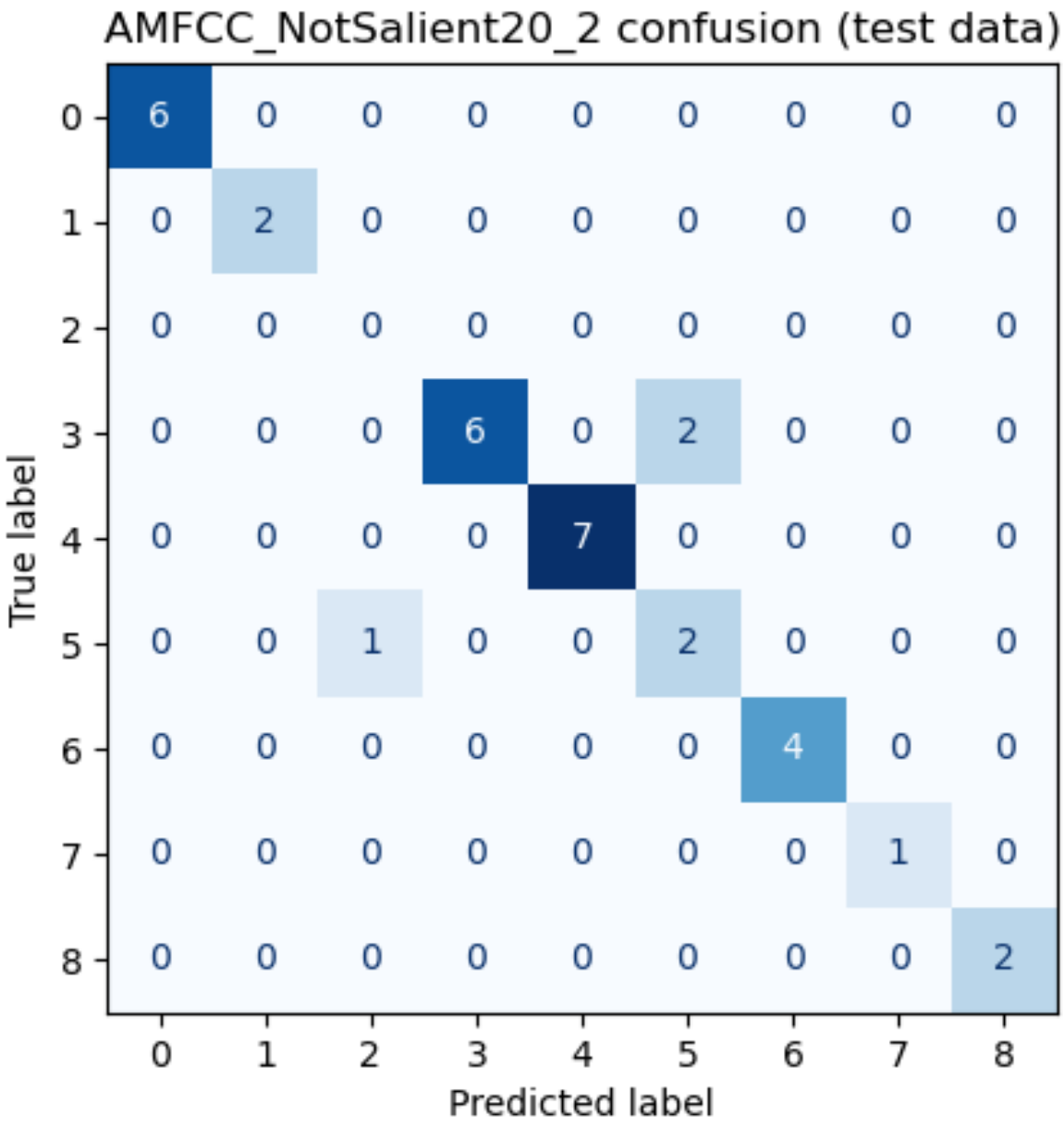

FIGURE J.8 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_2).

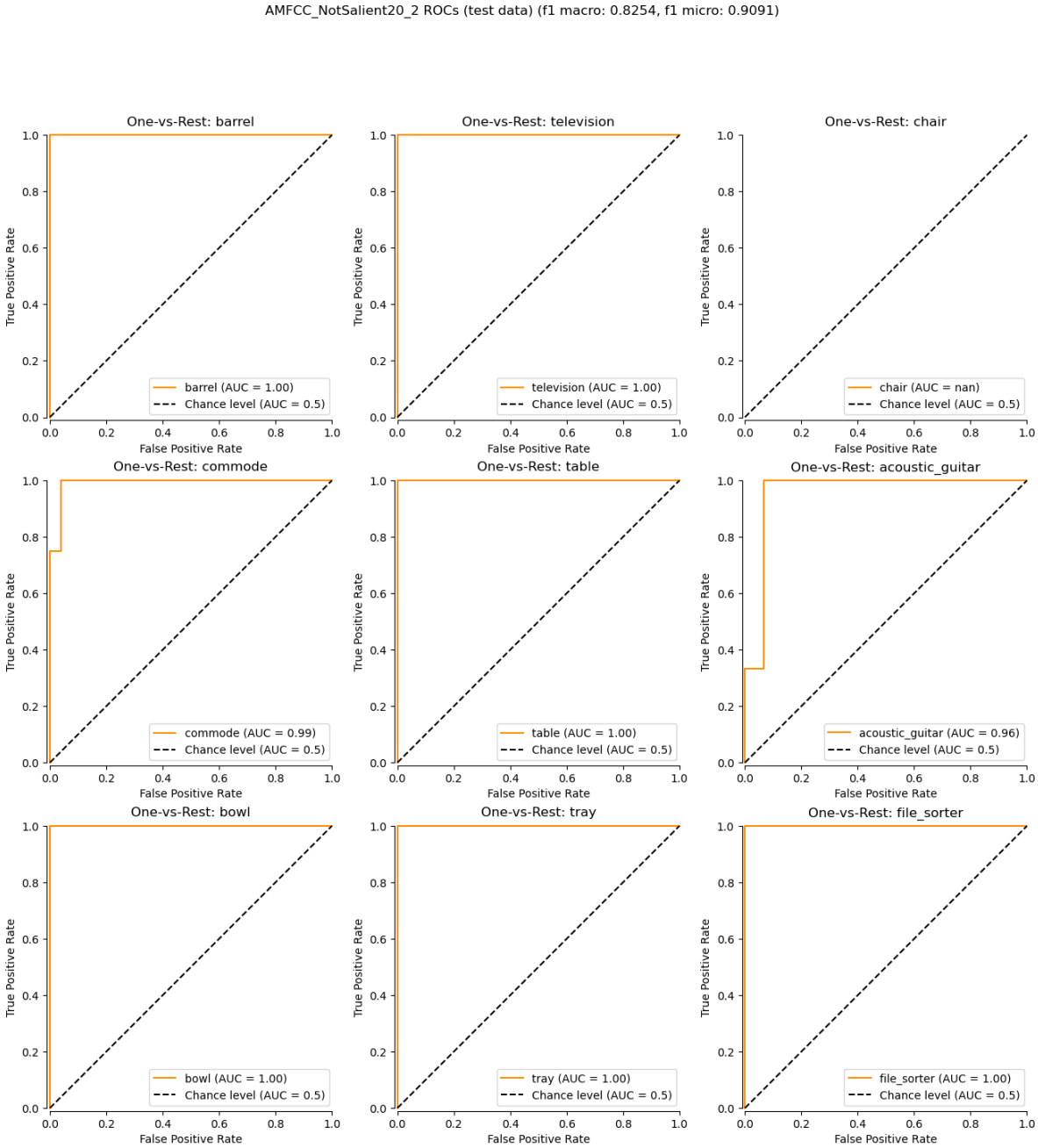

FIGURE J.9 – Courbes ROC et F1 Scores sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_2).

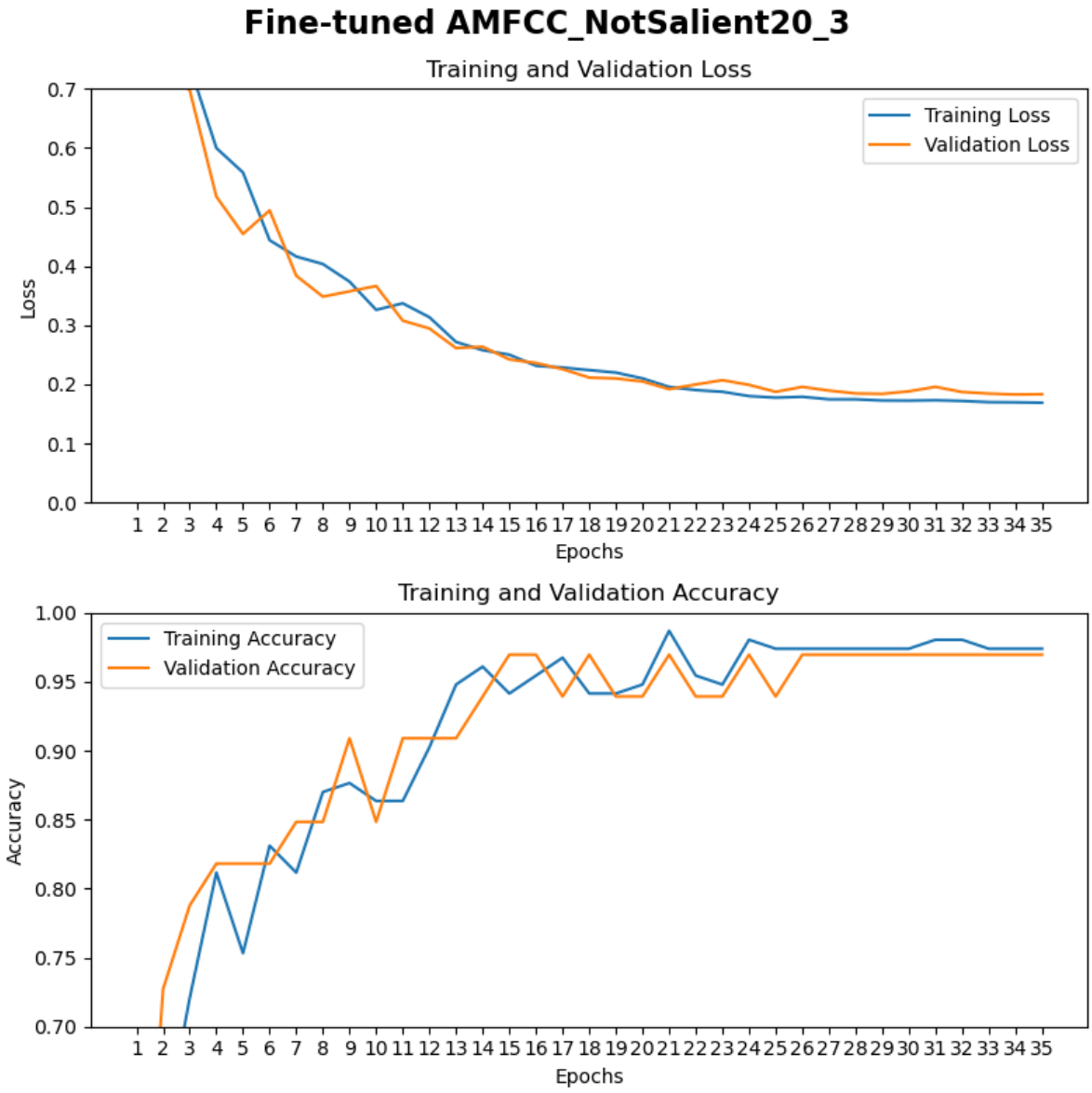

FIGURE J.10 – Historique de l'entraînement du modèle audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 41 à 60) par objet (AMFCC_NotSalient20_3).

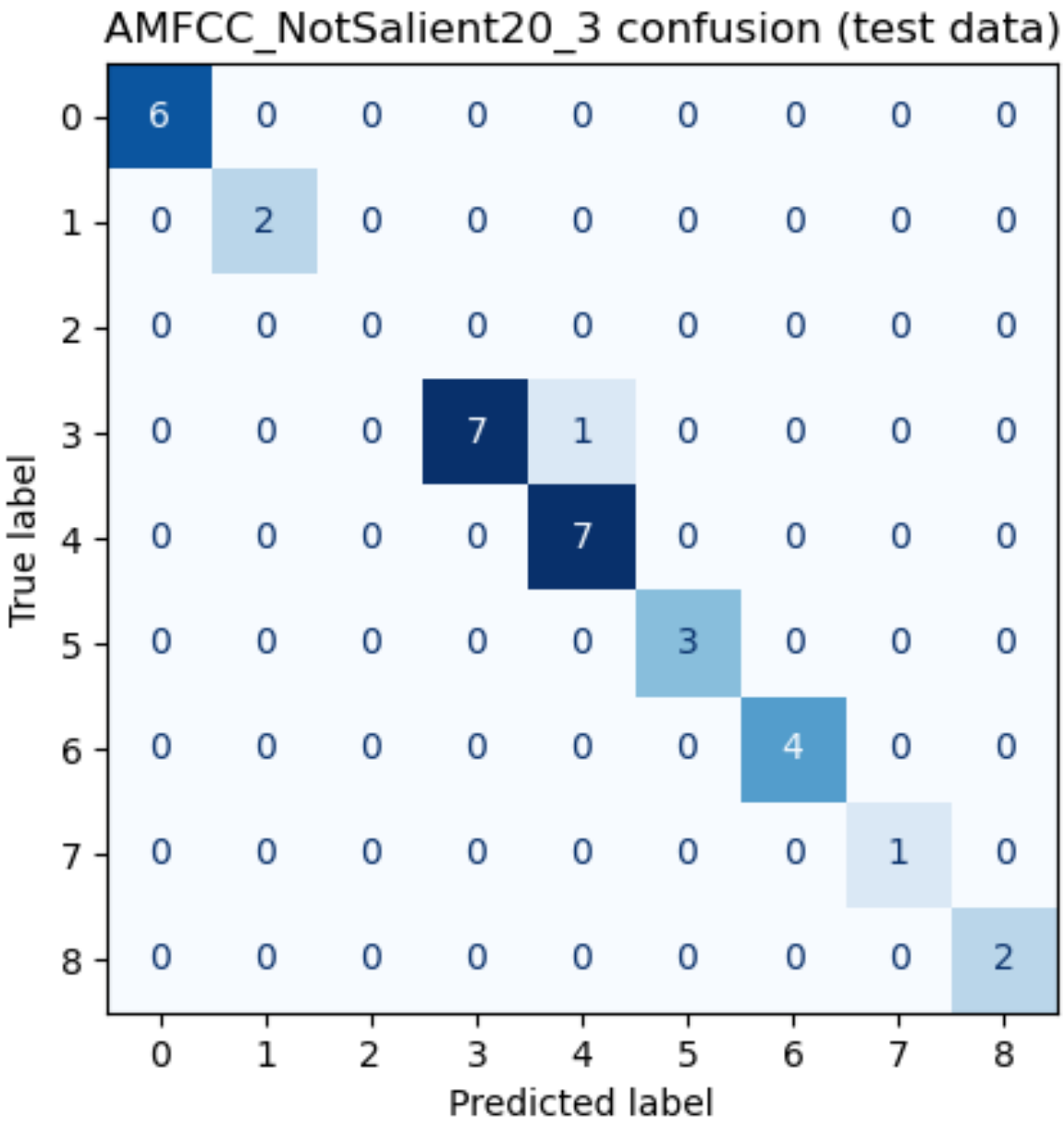

FIGURE J.11 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_3).

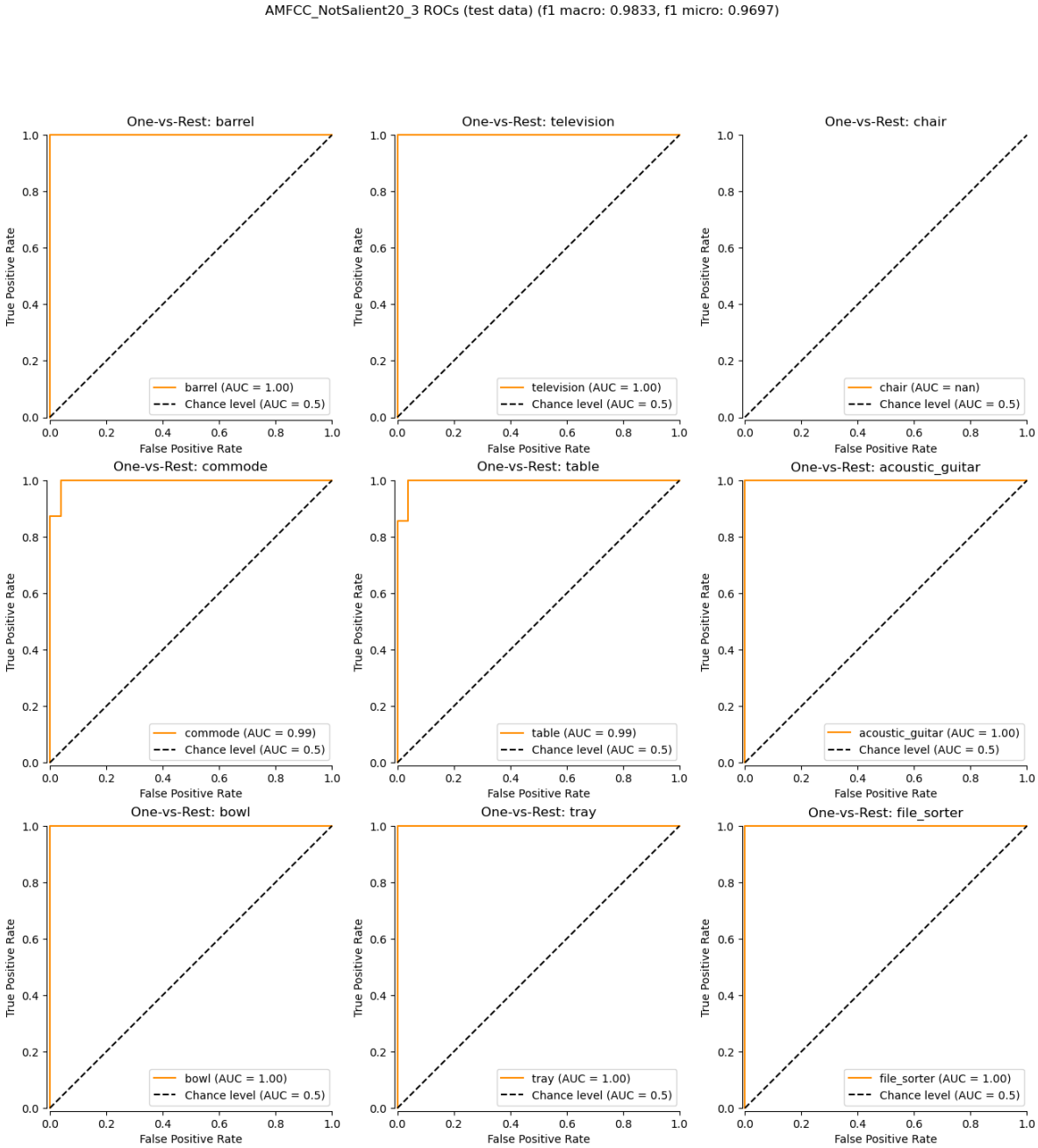

FIGURE J.12 – Courbes ROC et F1 Scores sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_3).

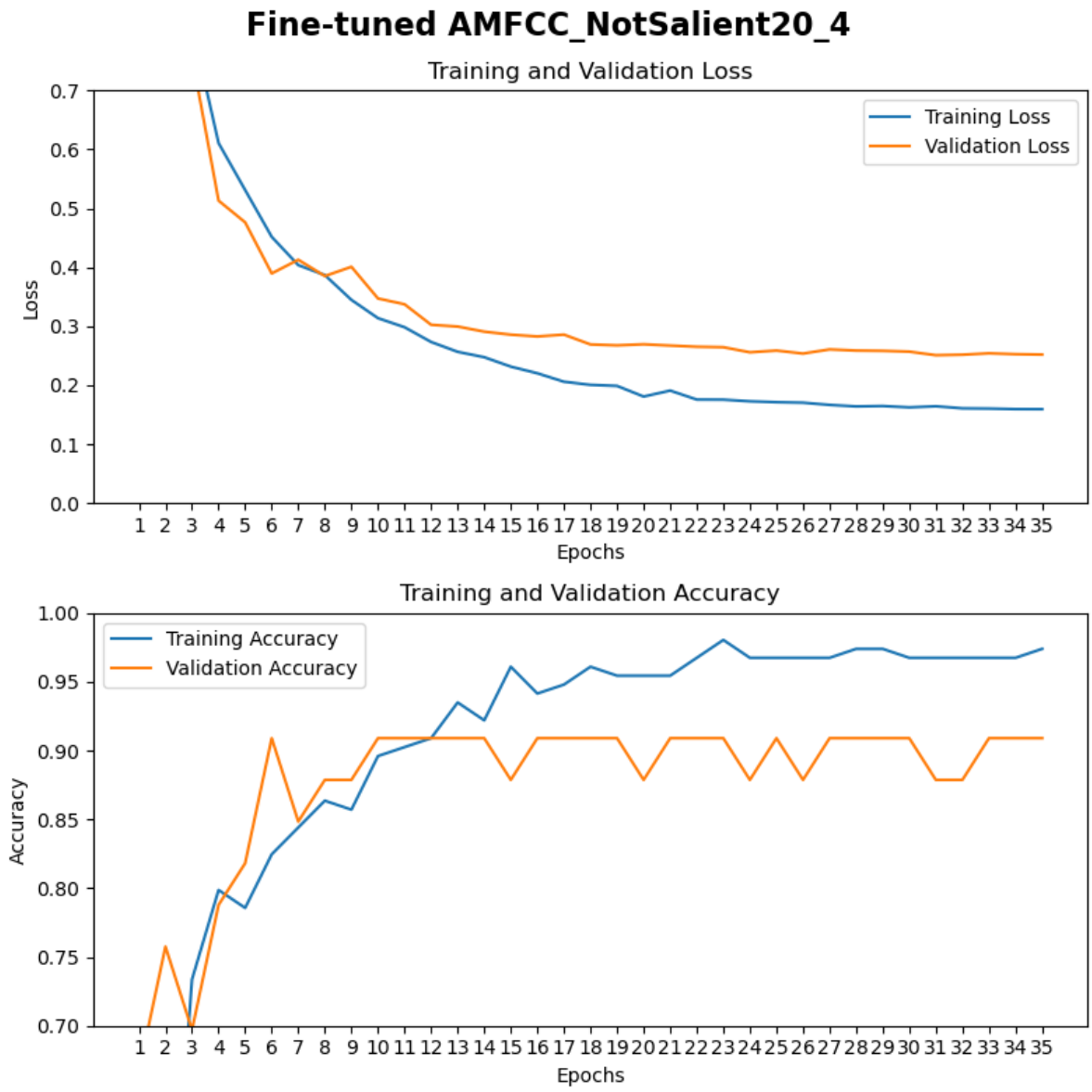

FIGURE J.13 – Historique de l'entraînement du modèle audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 61 à 80) par objet (AMFCC_NotSalient20_4).

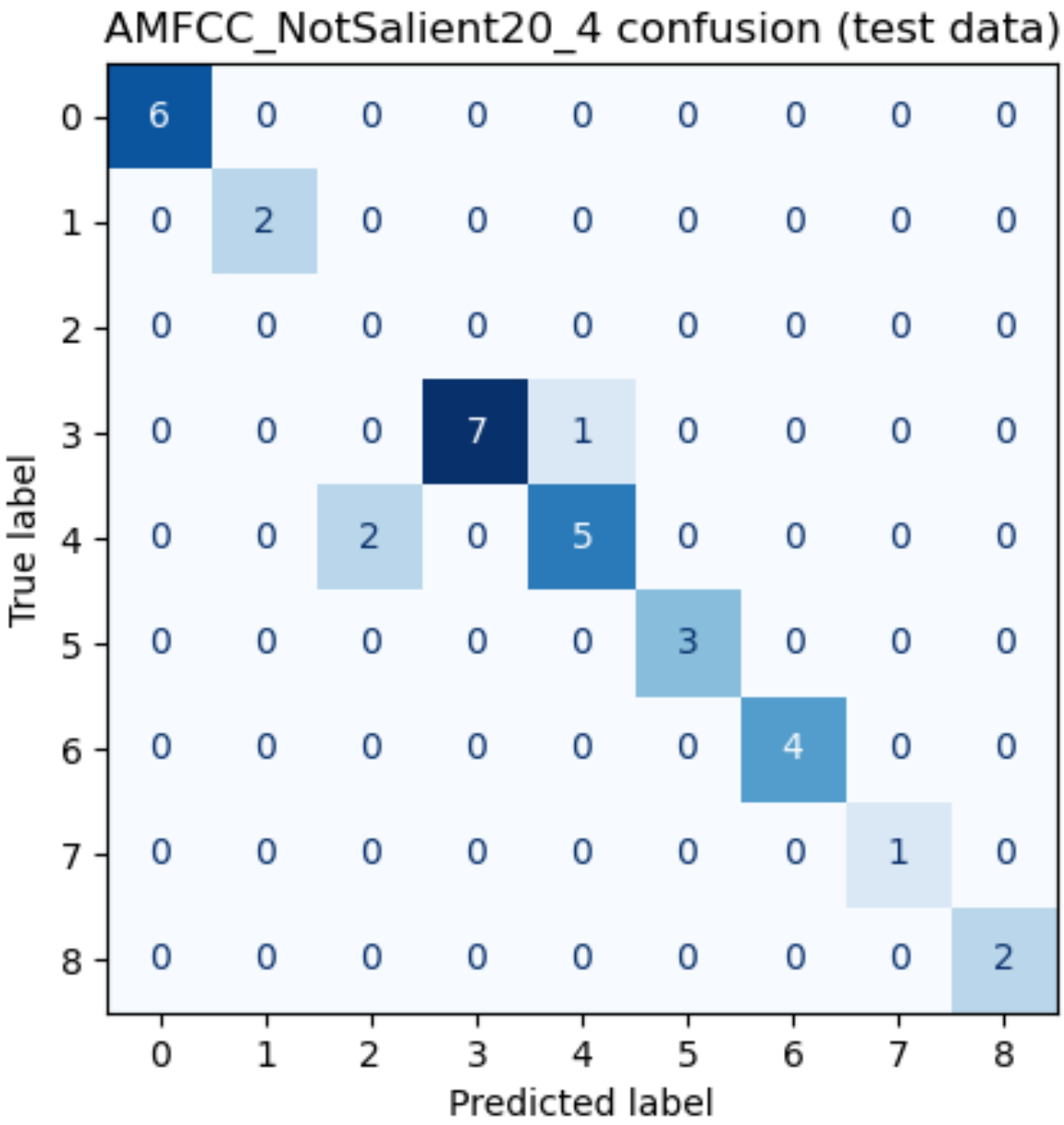

FIGURE J.14 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_4).

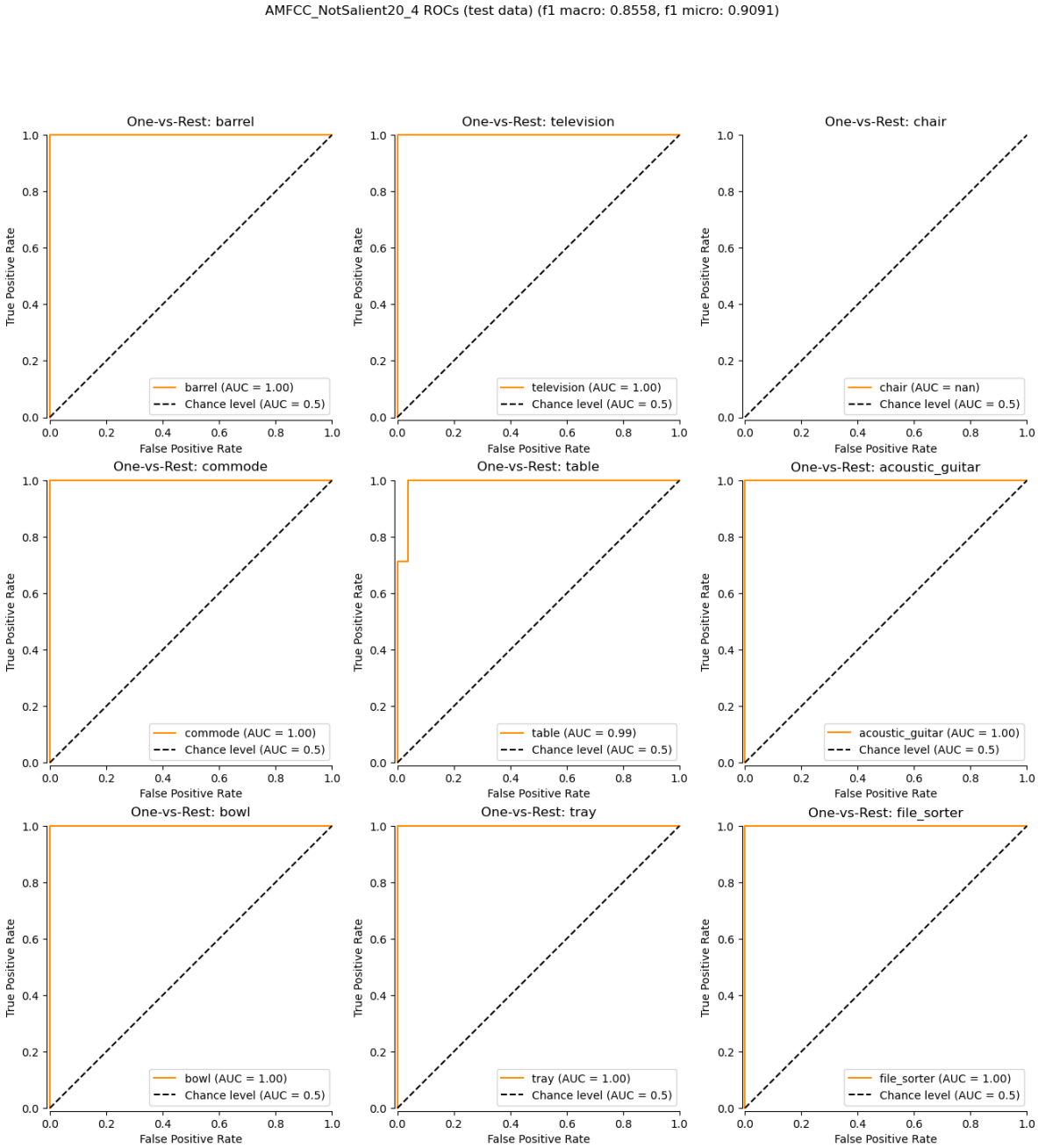

FIGURE J.15 – Courbes ROC et F1 Scores sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_4).

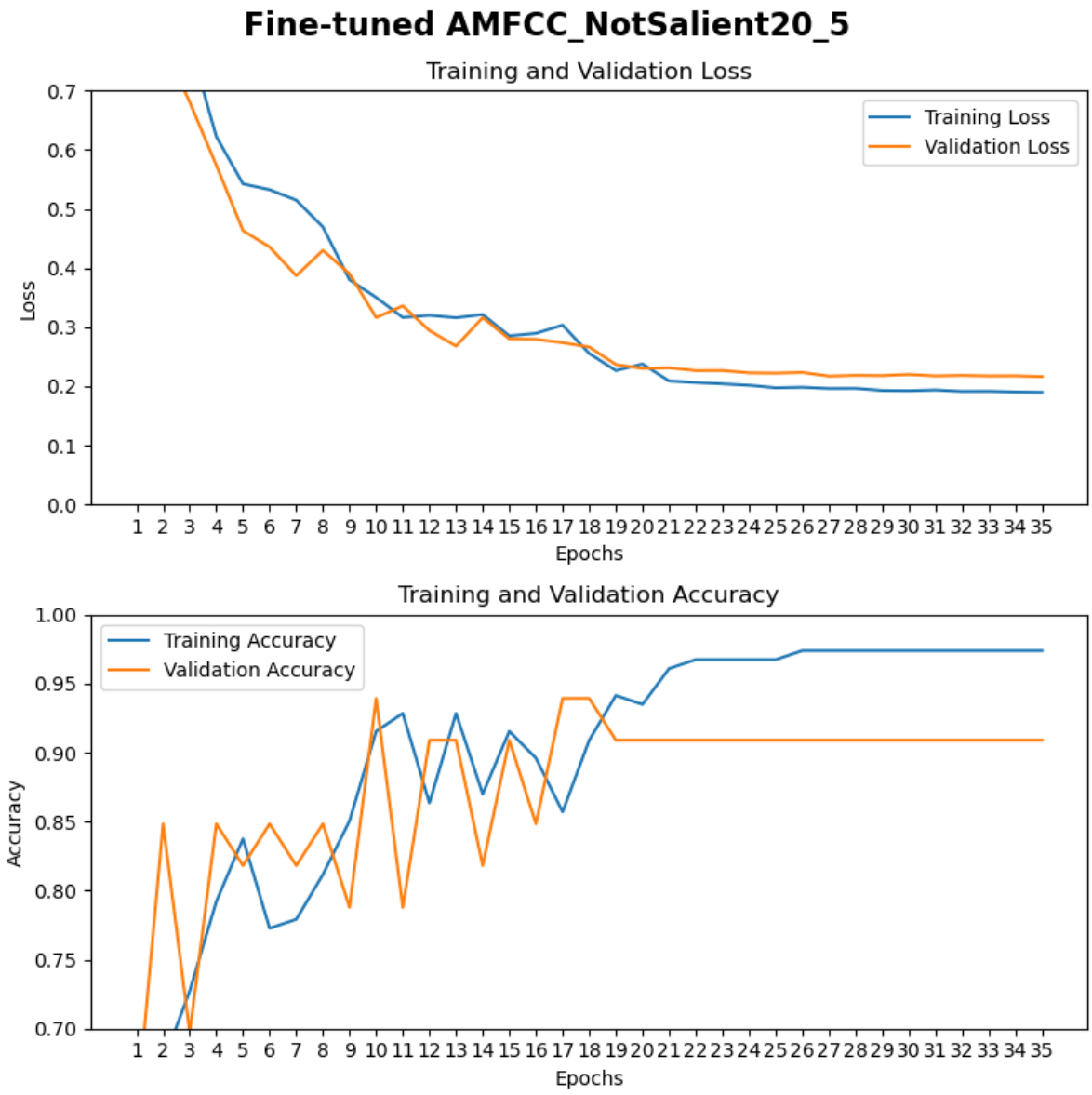

FIGURE J.16 – Historique de l'entraînement du modèle audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 81 à 100) par objet (AMFCC_NotSalient20_5).

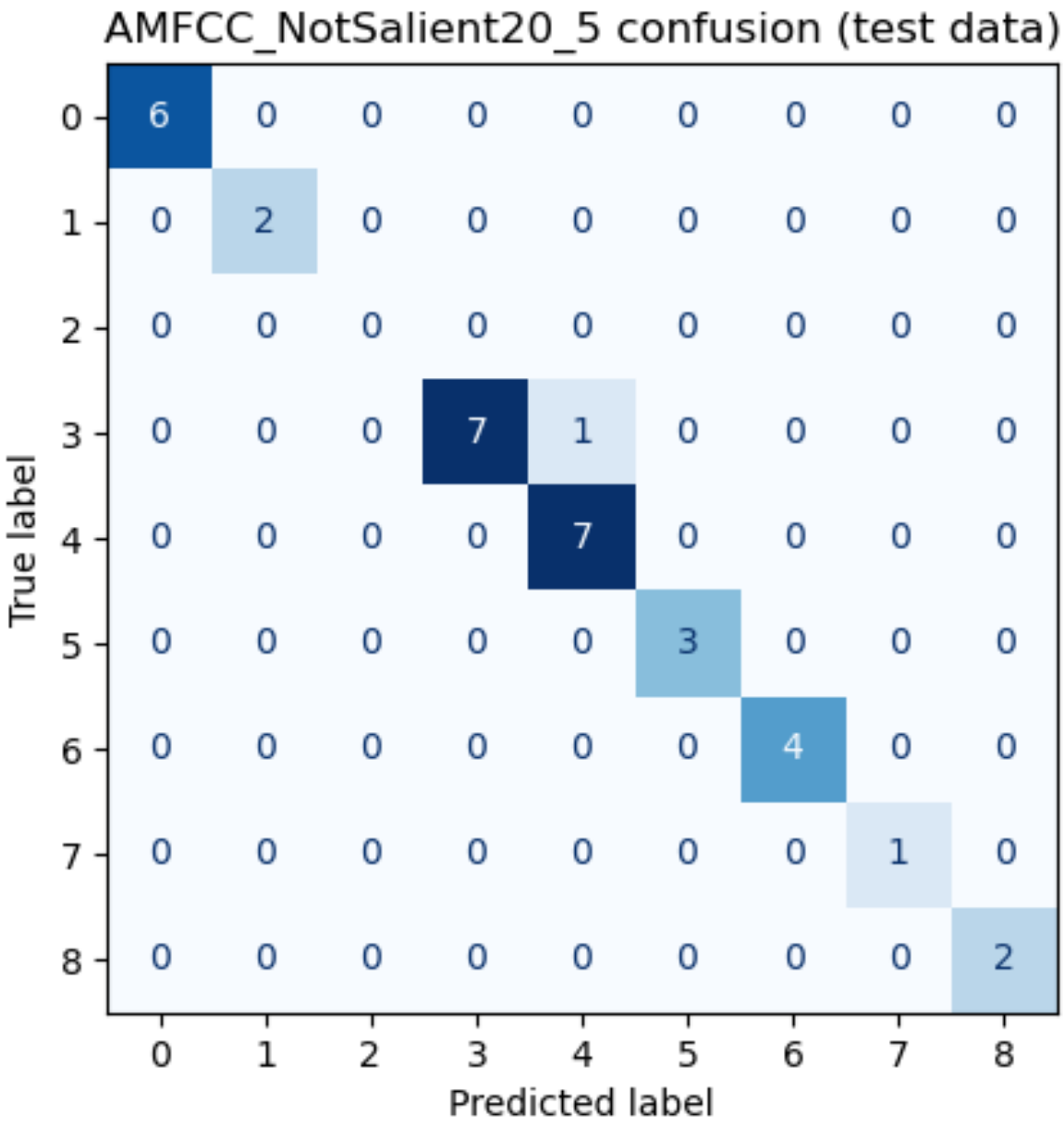

FIGURE J.17 – Matrice de confusion sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_5).

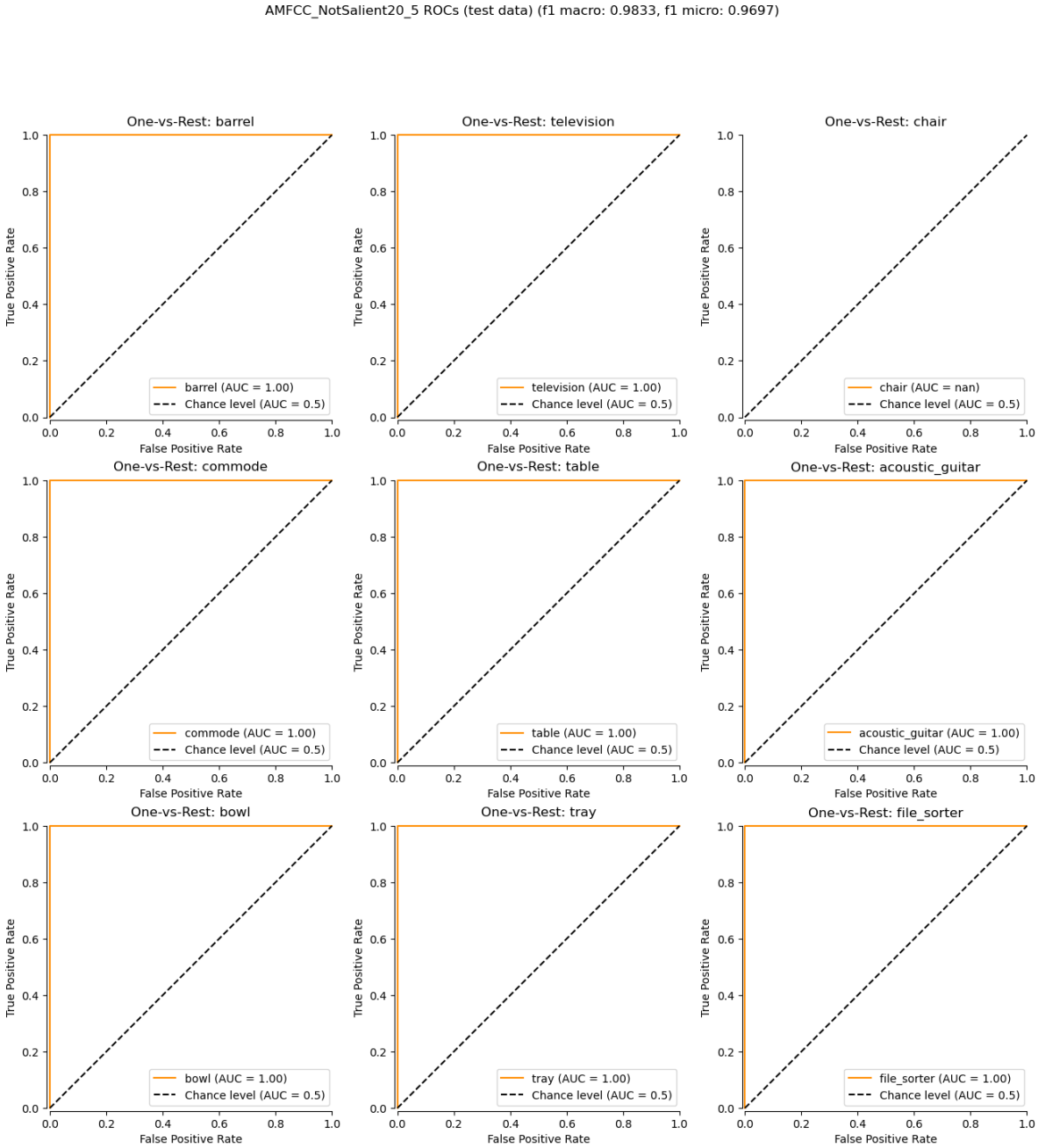

FIGURE J.18 – Courbes ROC et F1 Scores sur les données de test du modèle audio (MFCC) (AMFCC_NotSalient20_5).

Annexe K

Résultats de l'entraînement des méta-modèles Visuel+Tactile

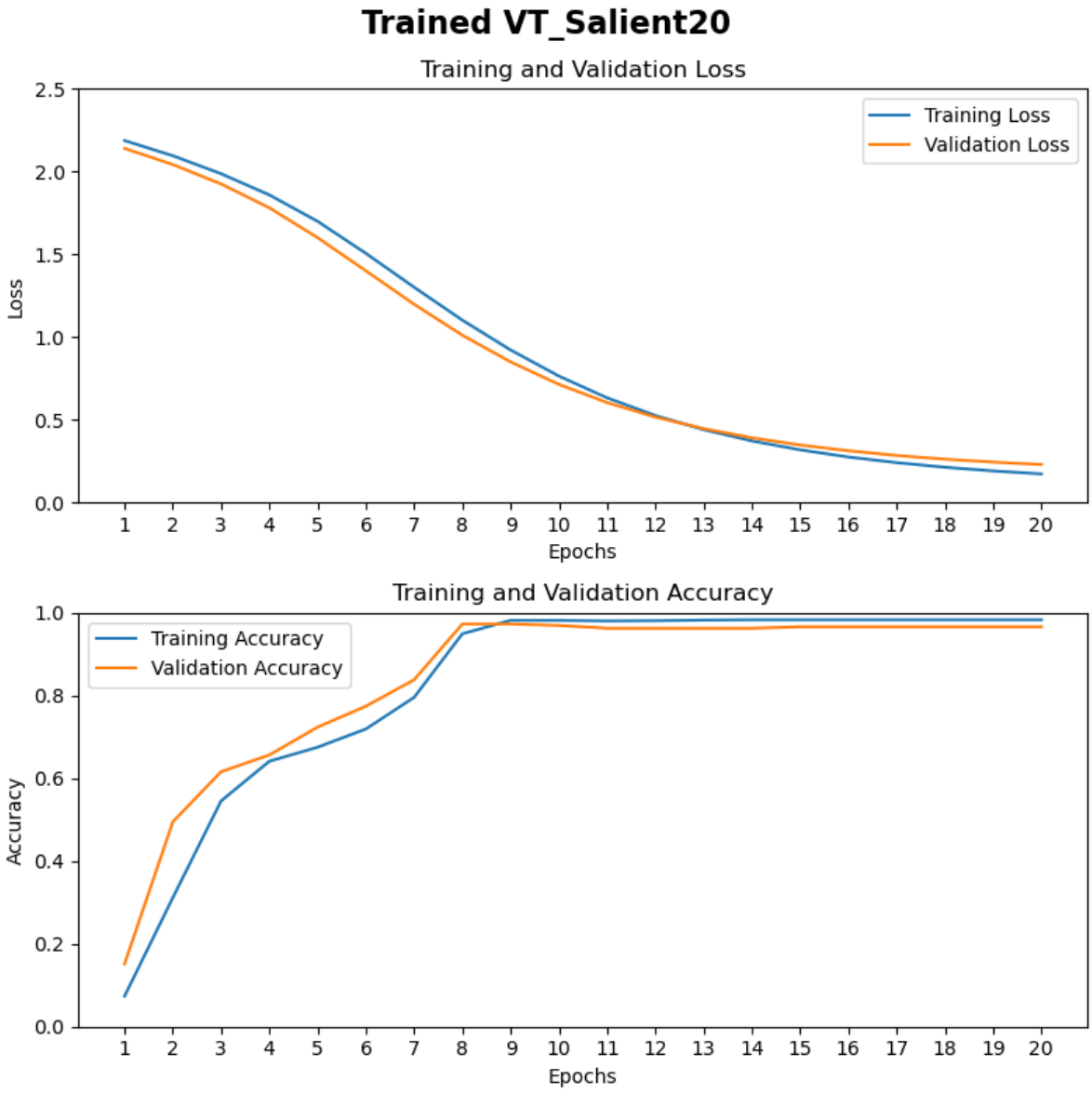

FIGURE K.1 – Historique de l'entraînement du méta-modèle combiné visuel/tactile utilisant les 20 points saillants par objet (VT_Salient20).

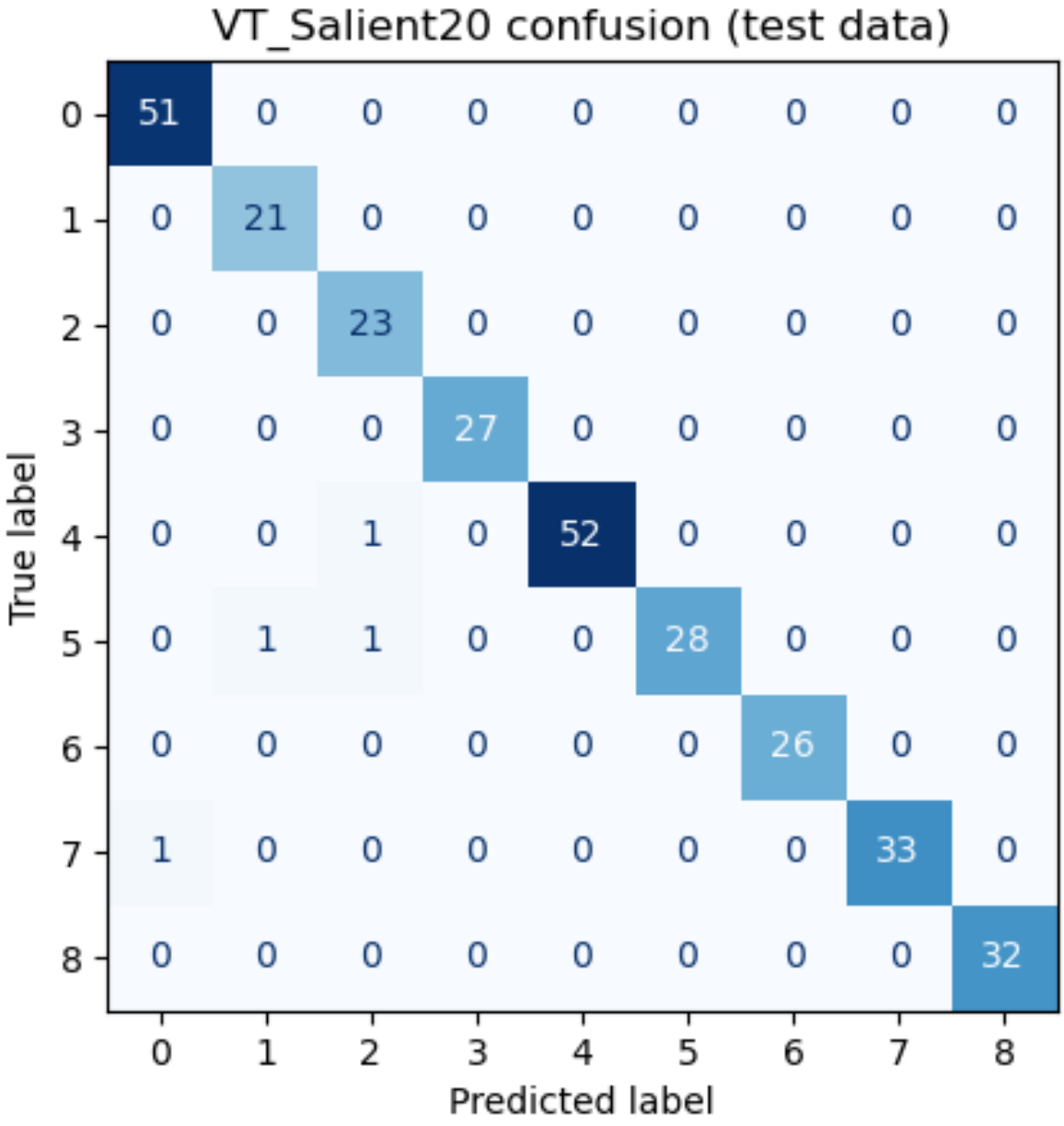

FIGURE K.2 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile (VT_Salient20).

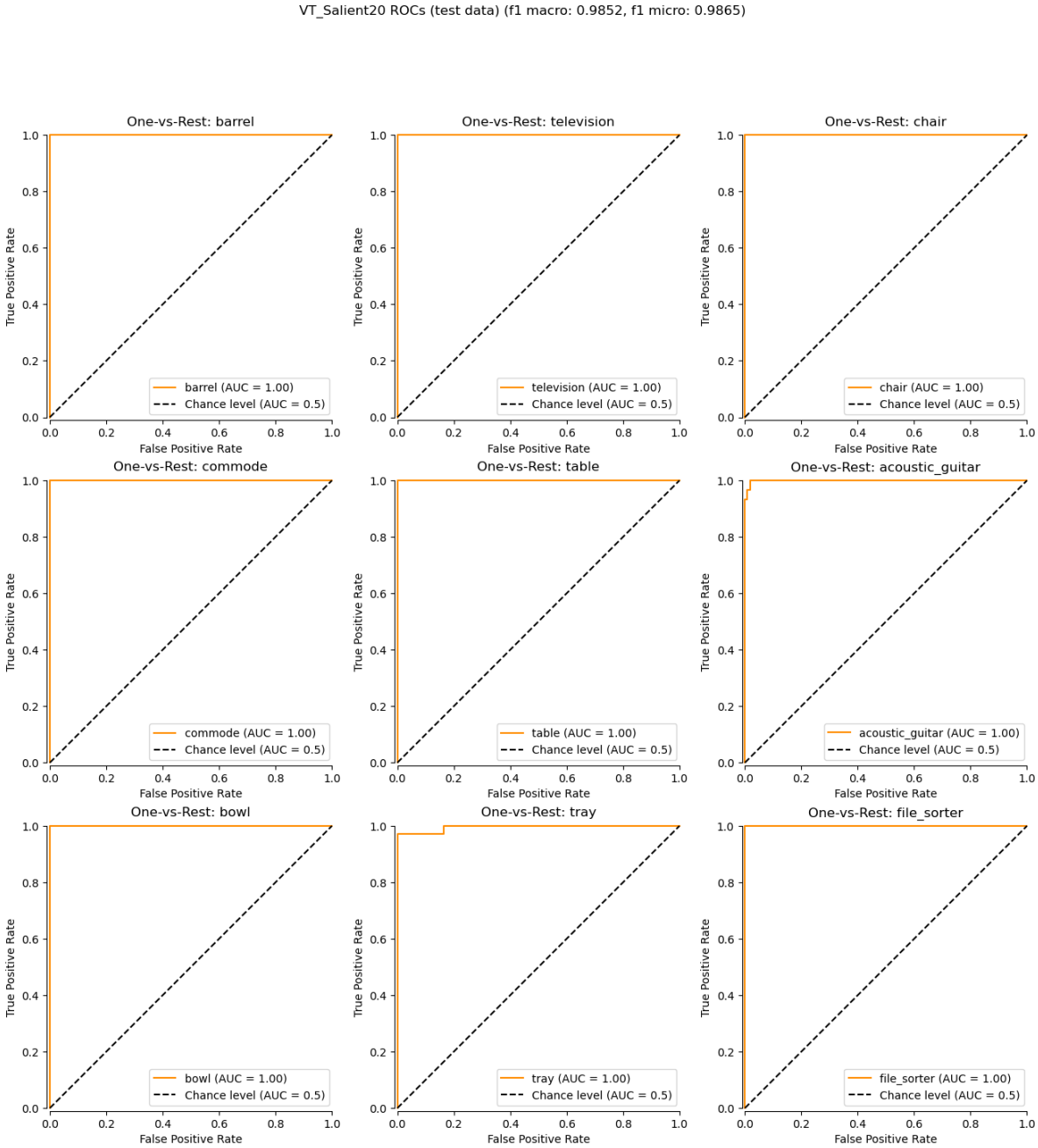

FIGURE K.3 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile (VT_Salient20).

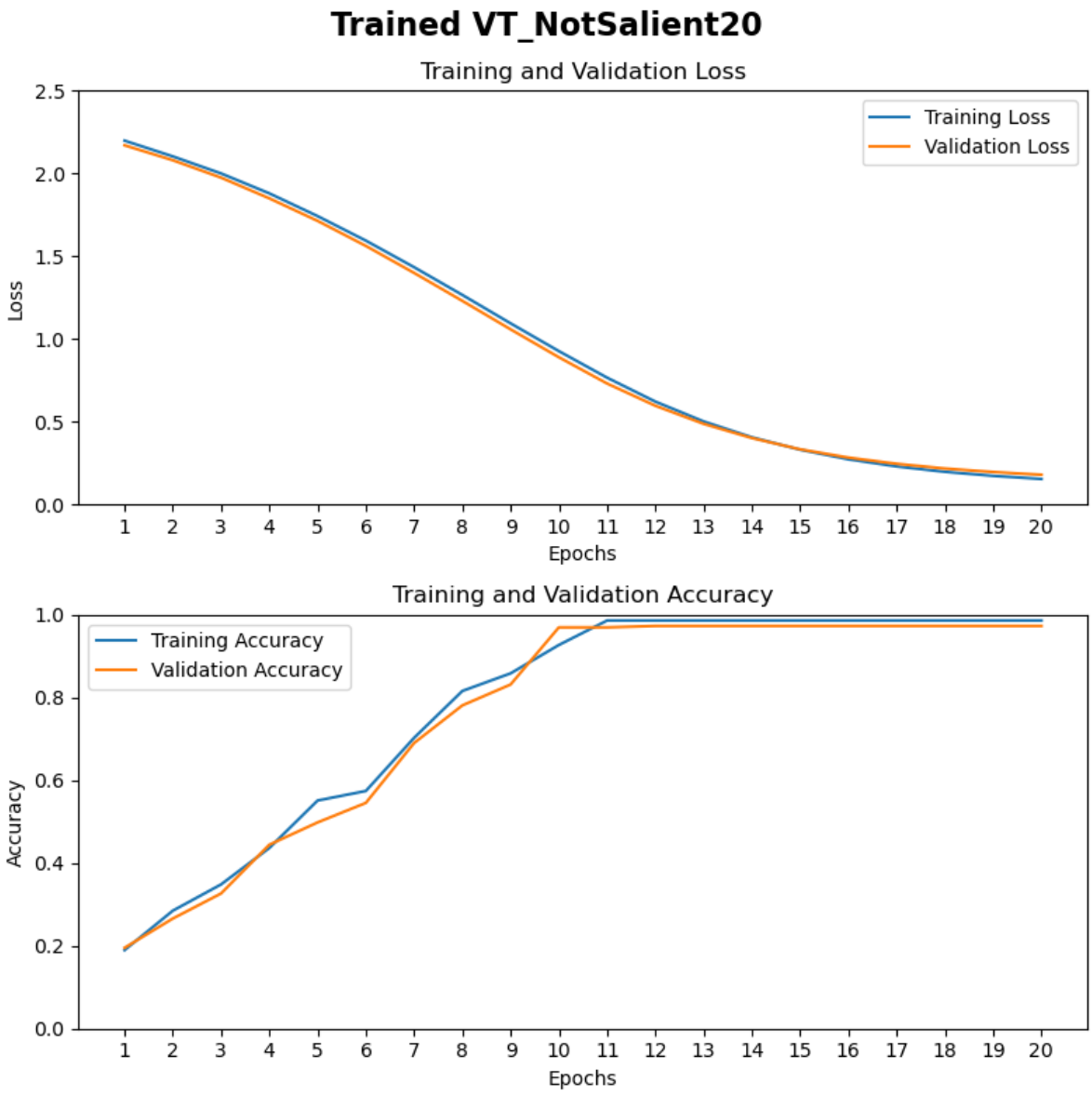

FIGURE K.4 – Historique de l'entraînement du méta-modèle combiné visuel/tactile utilisant les 20 premiers points sélectionnés aléatoirement par objet (VT_NotSalient20).

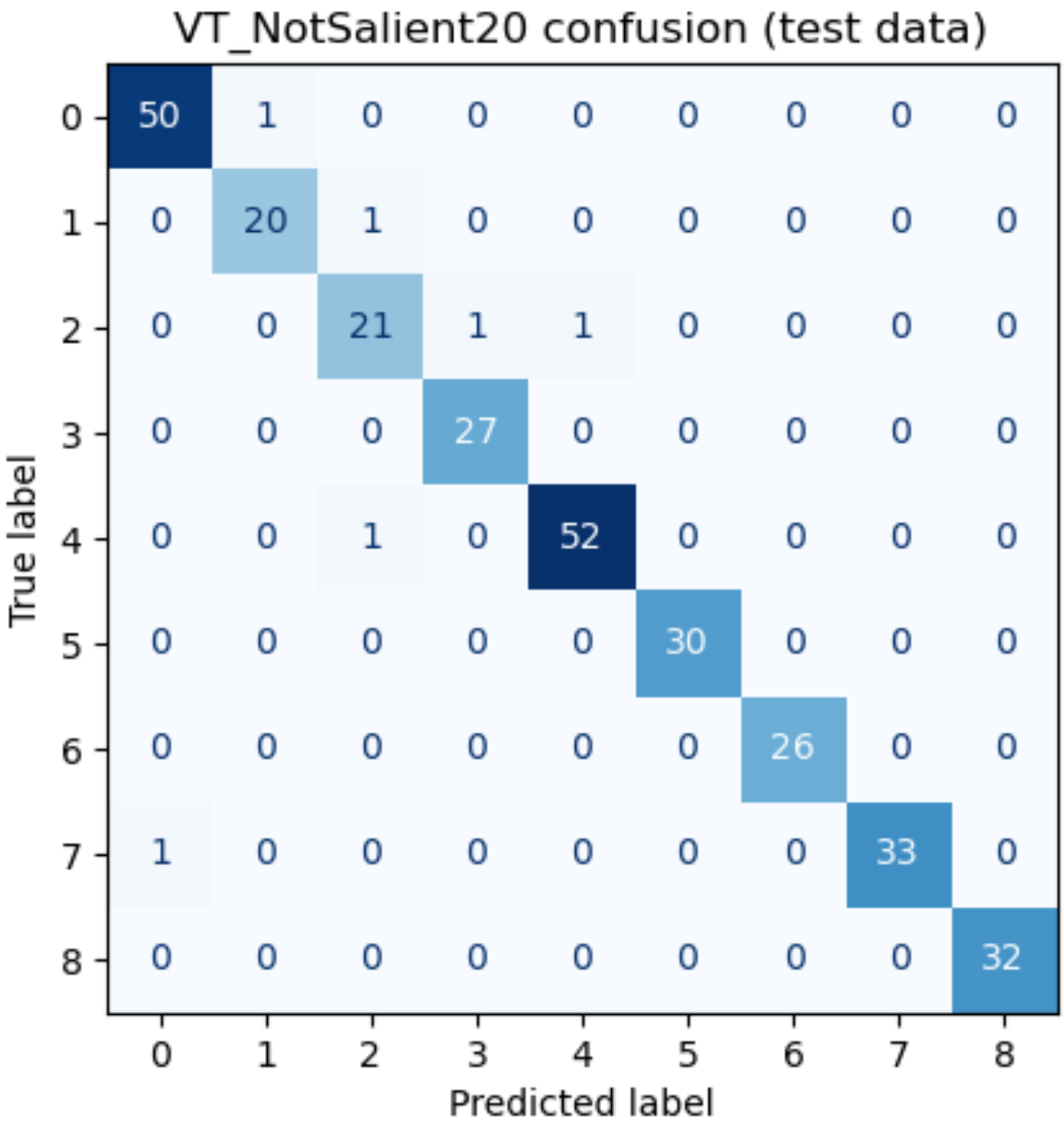

FIGURE K.5 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20).

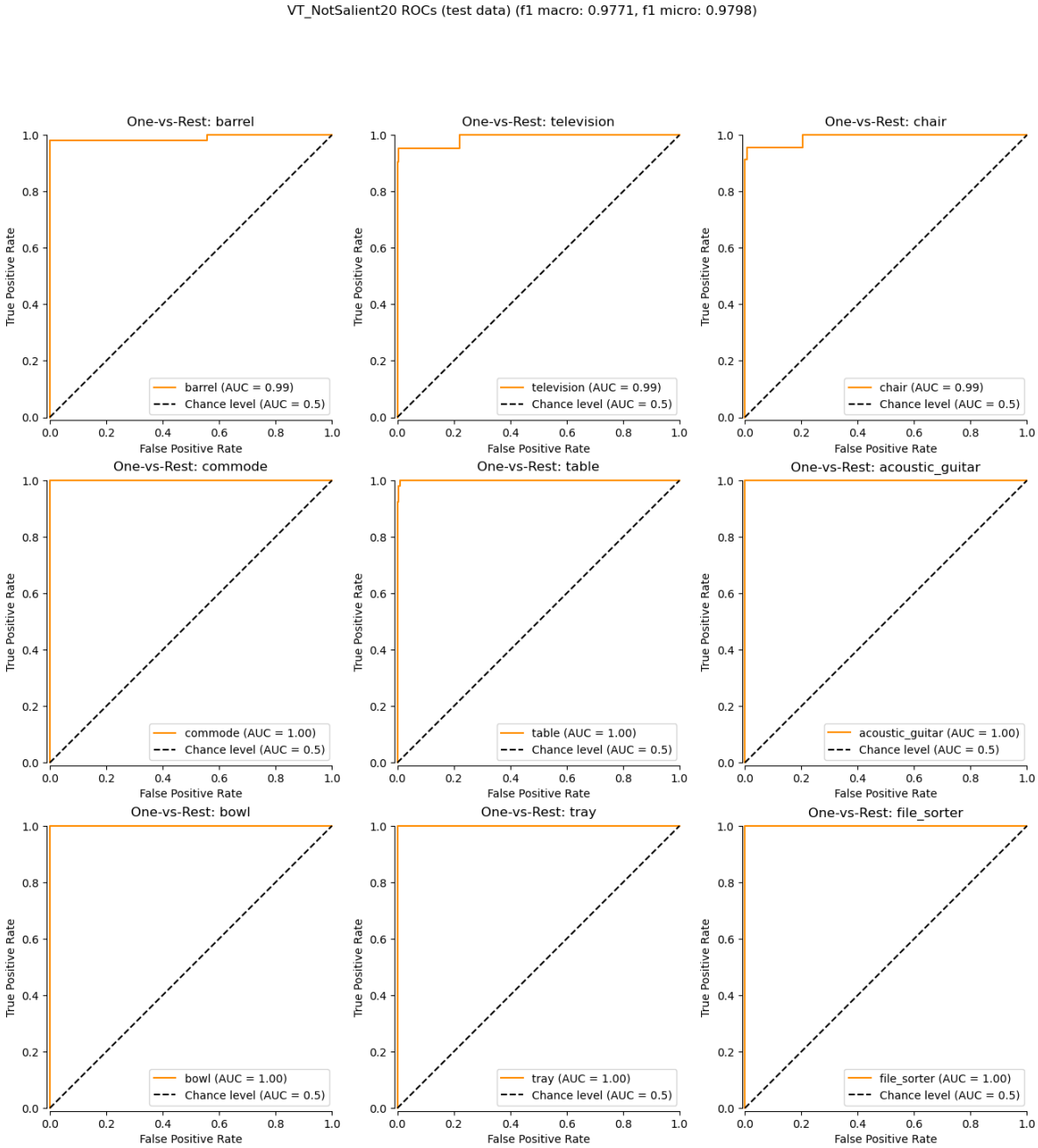

FIGURE K.6 – Courbes ROC et F1 Scores sur les données de test du combiné visuel/-tactile (VT_NotSalient20).

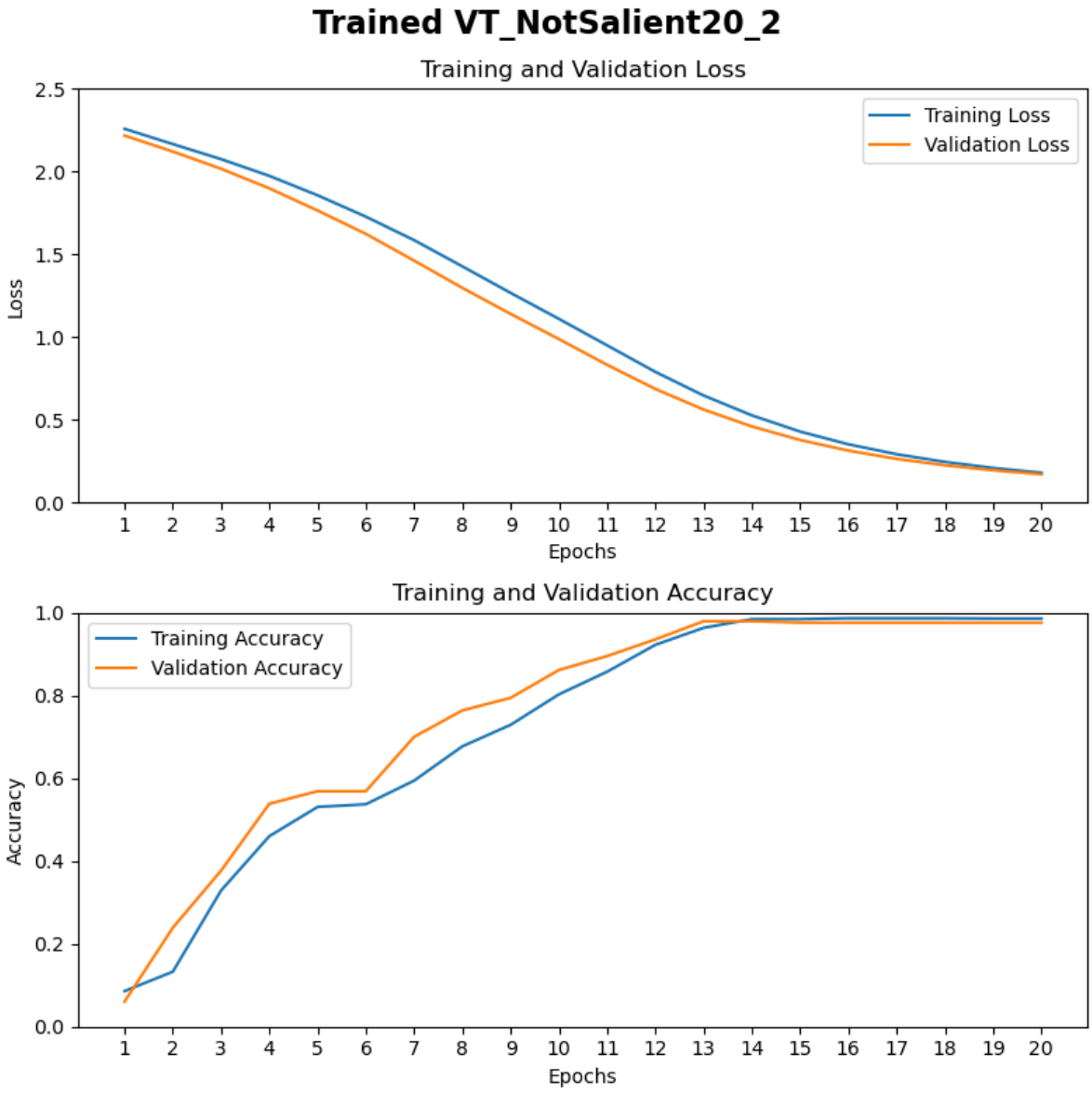

FIGURE K.7 – Historique de l'entraînement du méta-modèle combiné visuel/-tactile utilisant les 20 points sélectionnés aléatoirement (# 21 à 40) par objet (VT_NotSalient20_2).

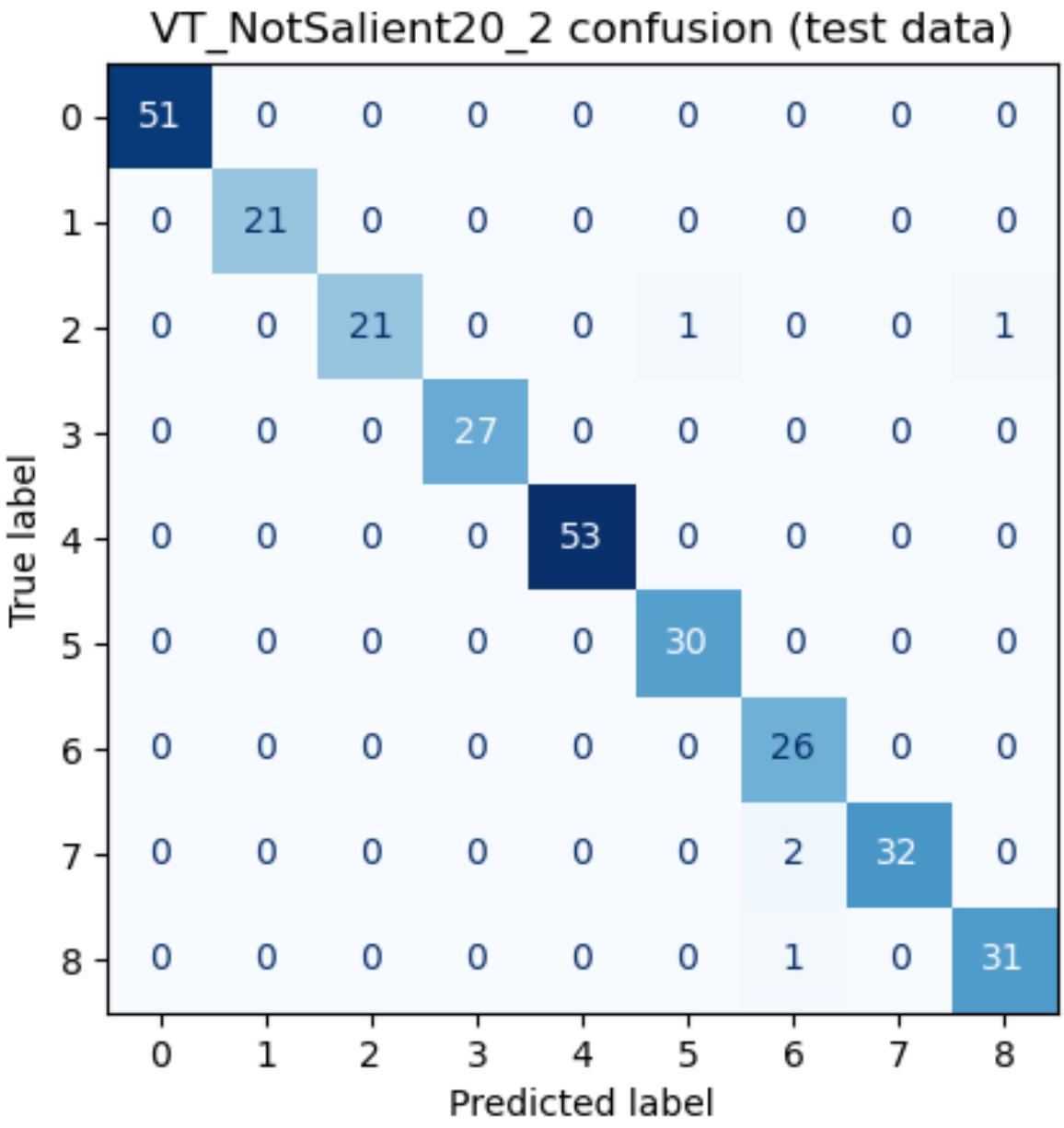

FIGURE K.8 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_2).

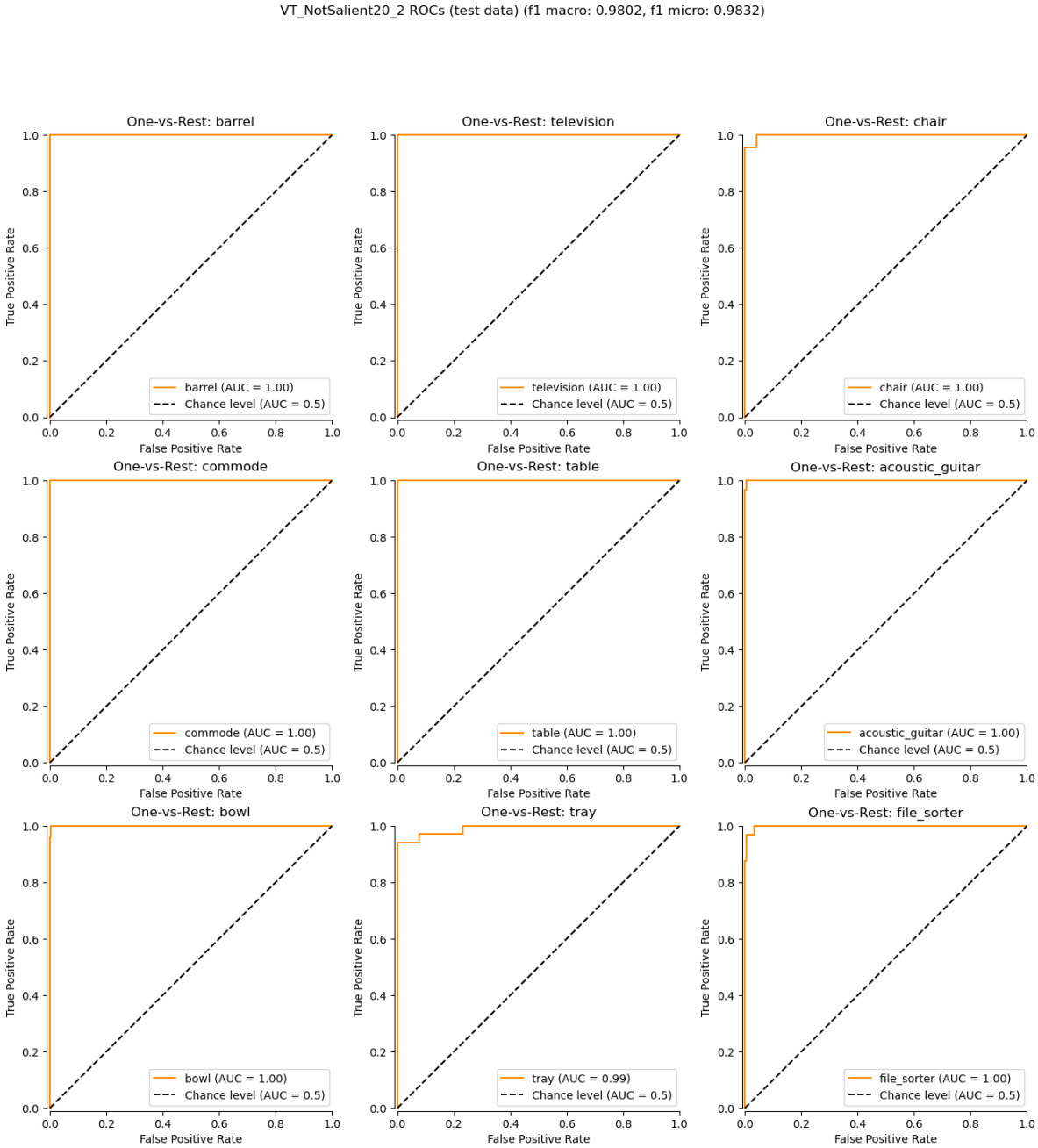

FIGURE K.9 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_2).

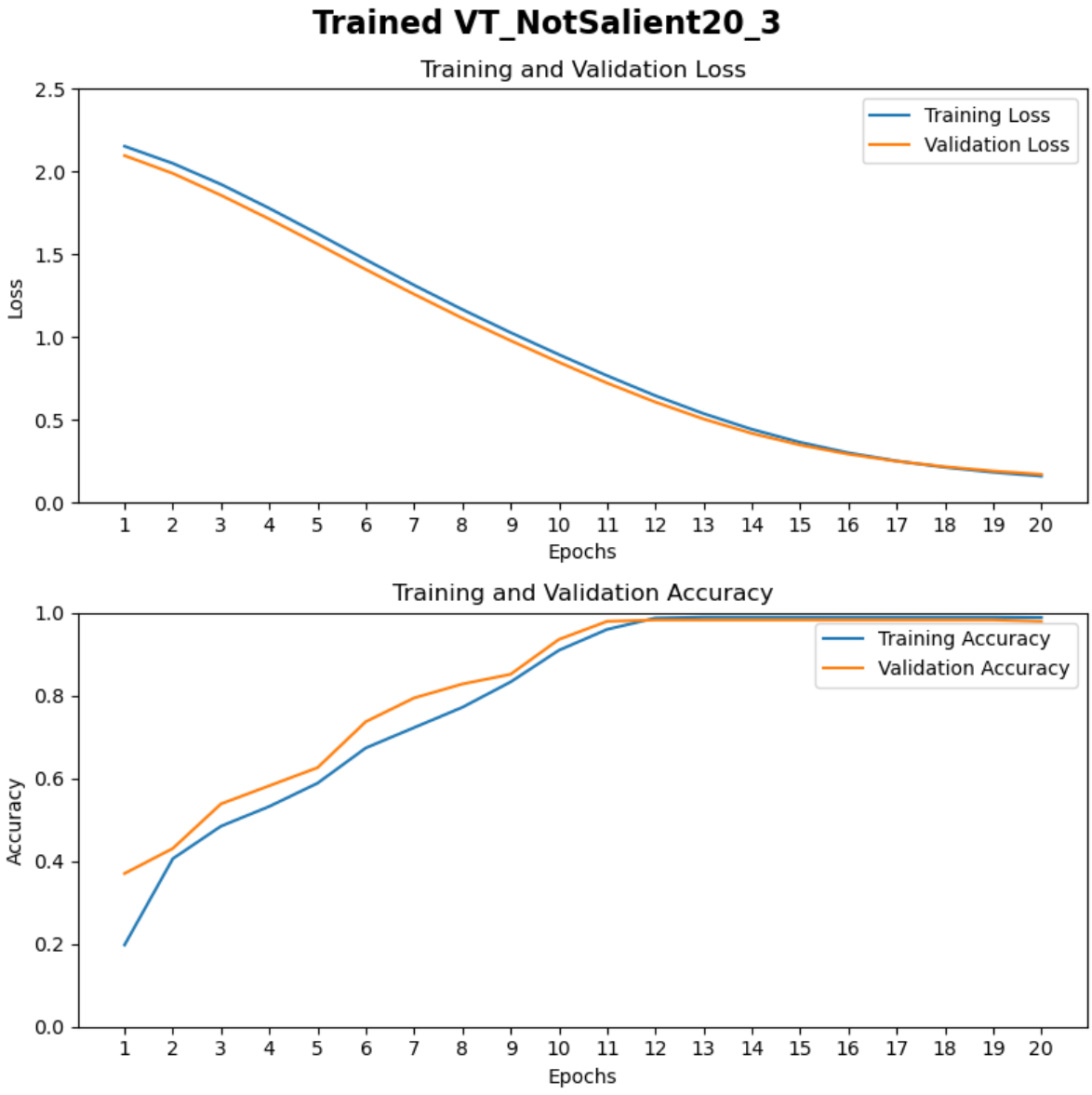

FIGURE K.10 – Historique de l'entraînement du méta-modèle combiné visuel/-tactile utilisant les 20 points sélectionnés aléatoirement (# 41 à 60) par objet (VT_NotSalient20_3).

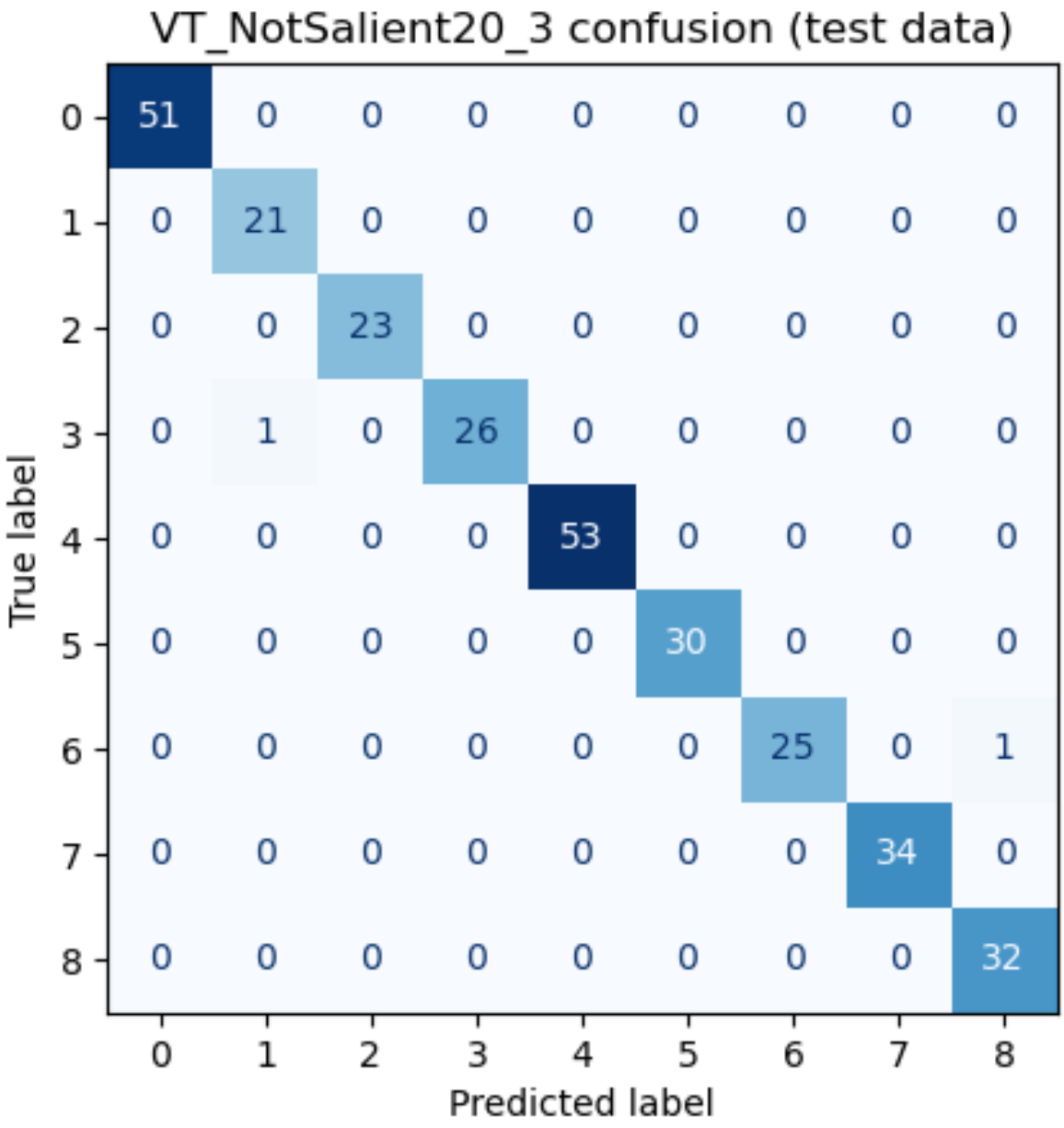

FIGURE K.11 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_3).

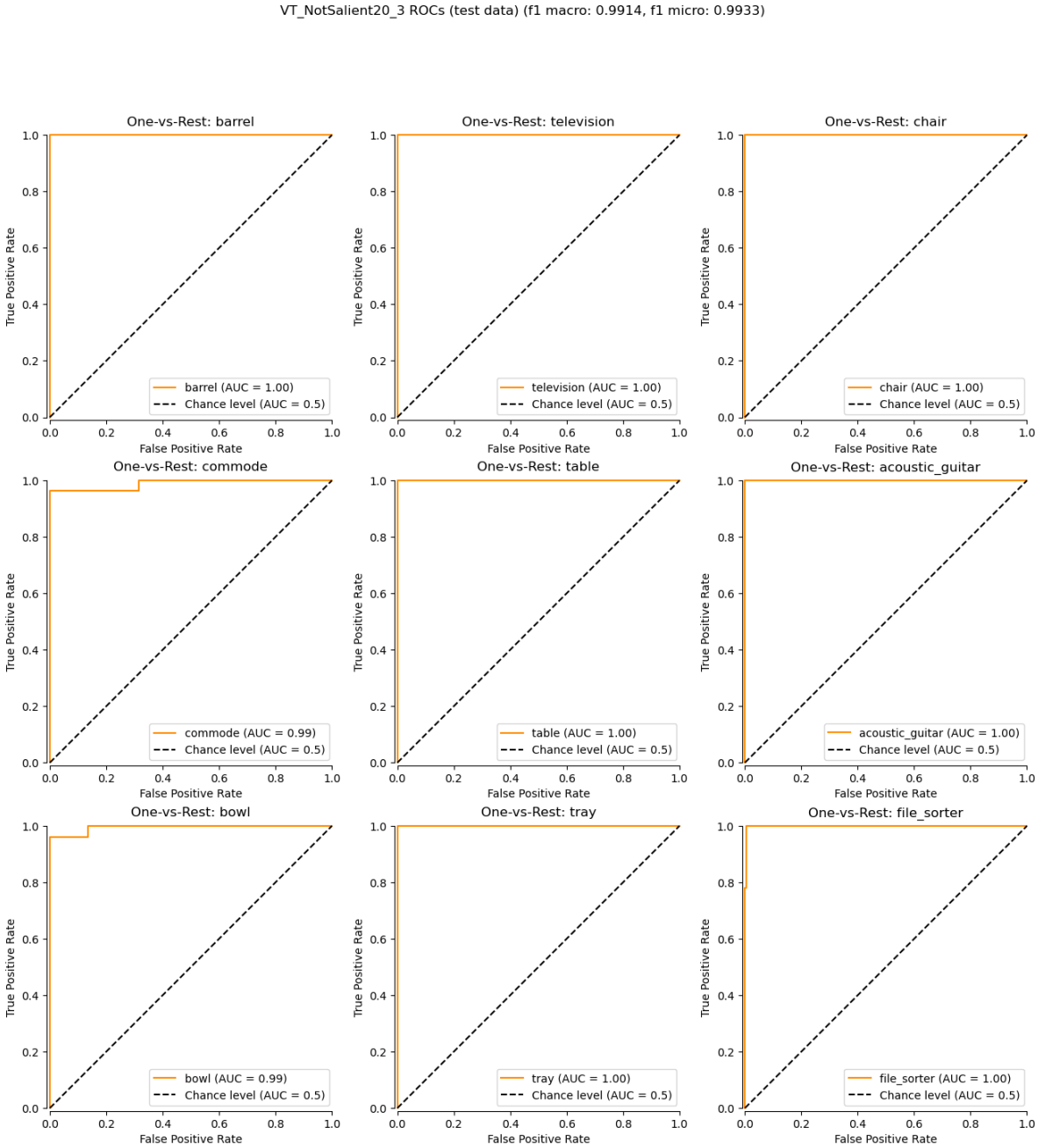

FIGURE K.12 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_3).

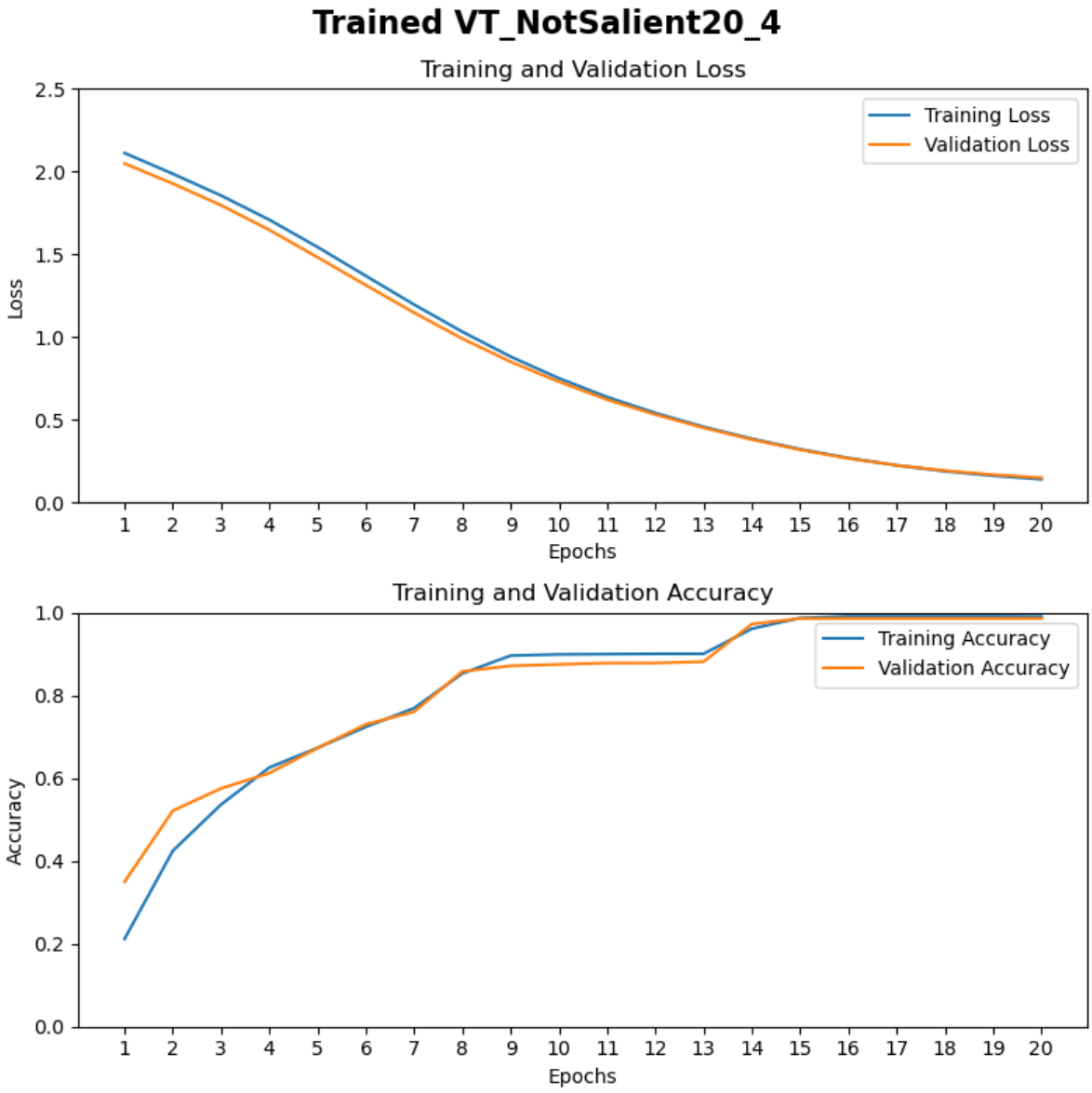

FIGURE K.13 – Historique de l'entraînement du méta-modèle combiné visuel/-tactile utilisant les 20 points sélectionnés aléatoirement (# 61 à 80) par objet (VT_NotSalient20_4).

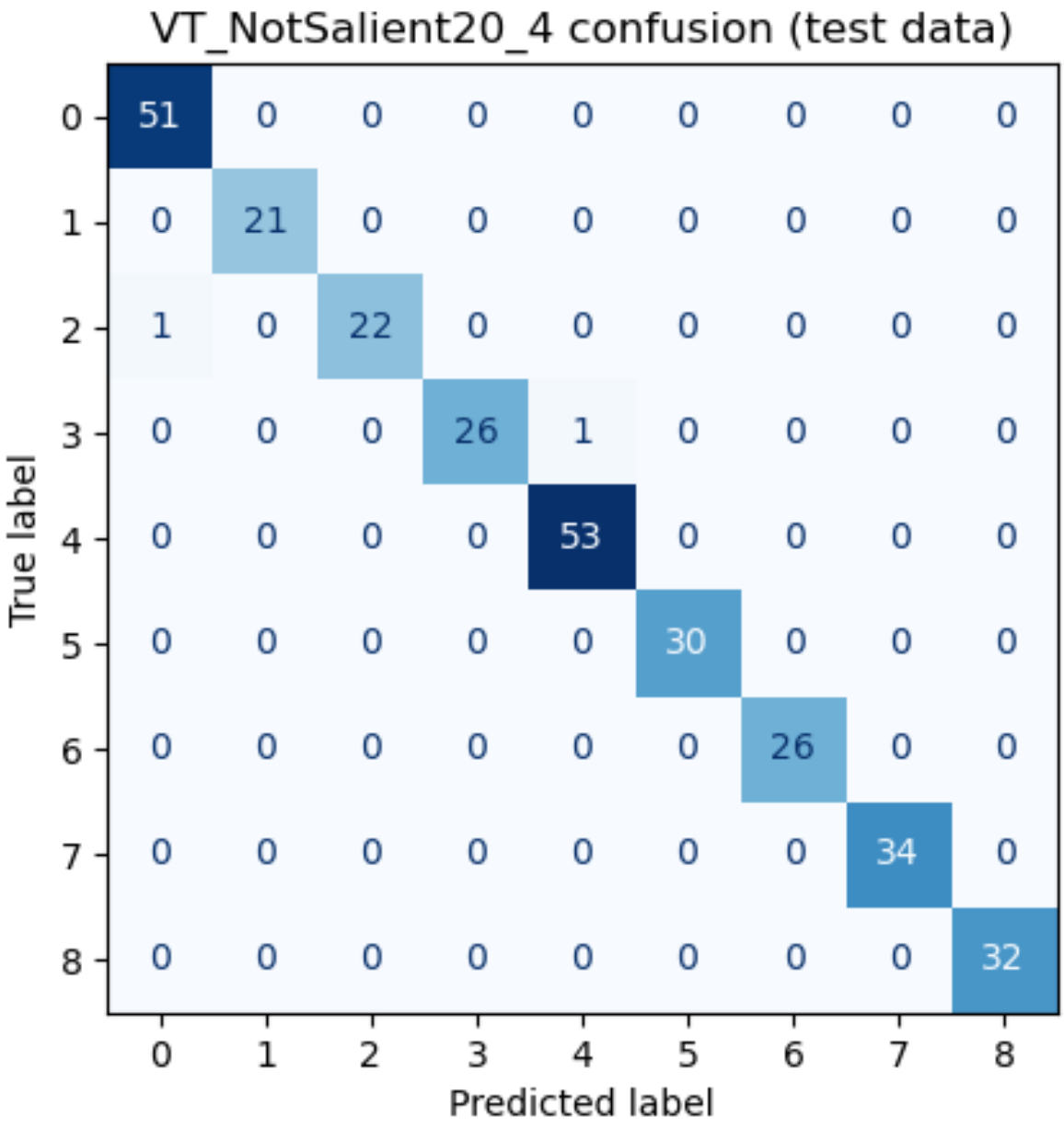

FIGURE K.14 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_4).

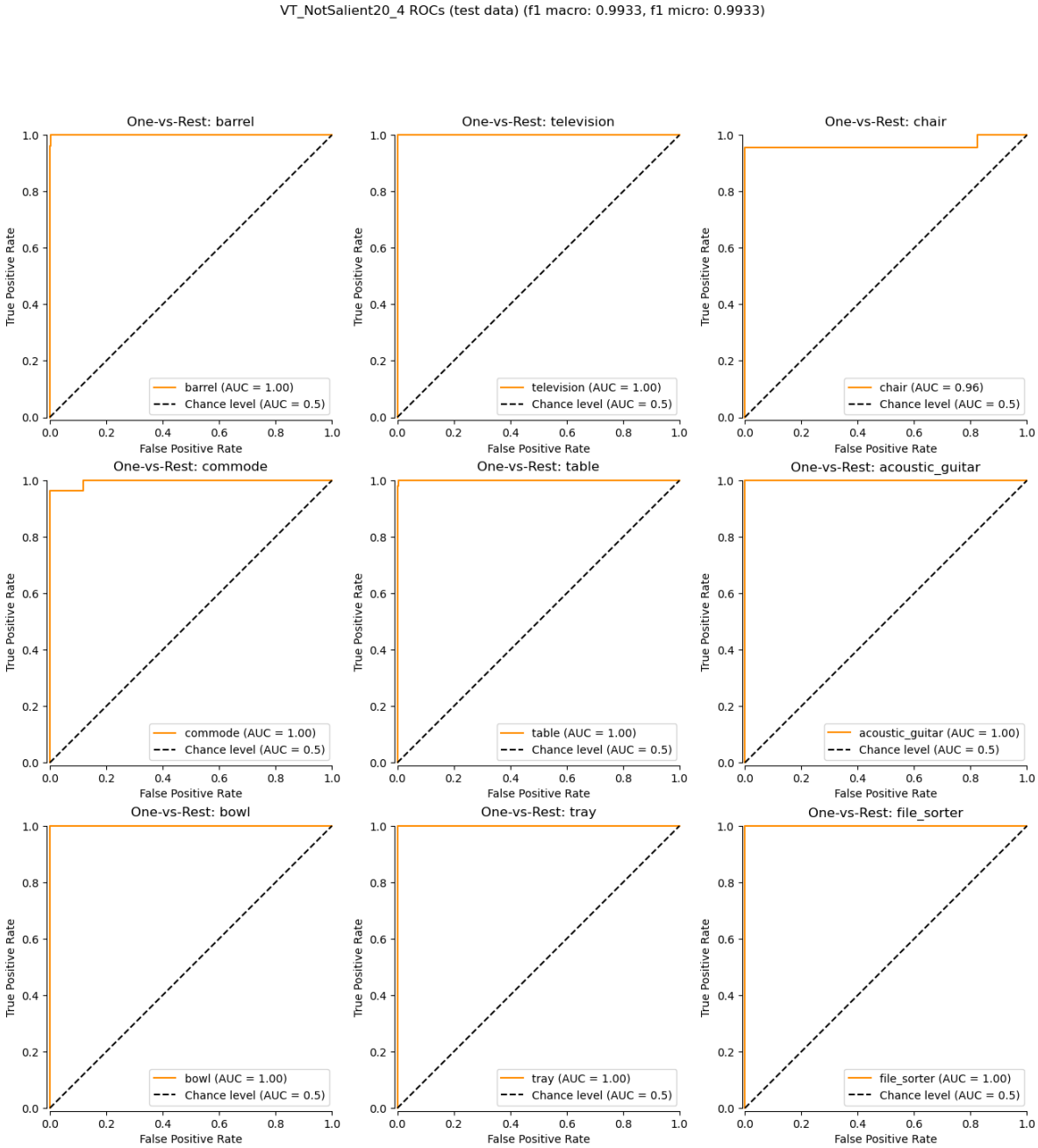

FIGURE K.15 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_4).

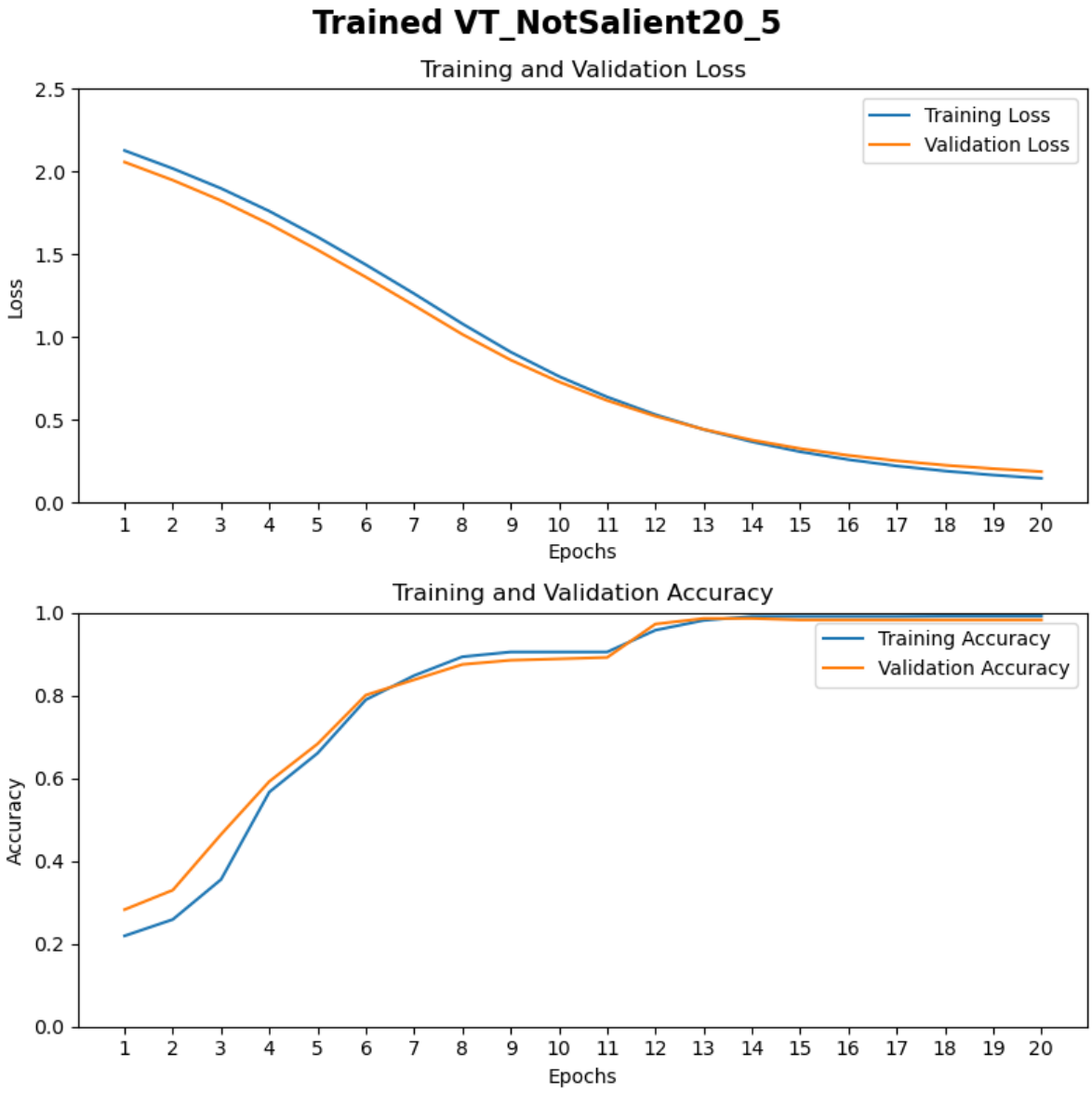

FIGURE K.16 – Historique de l'entraînement du méta-modèle combiné visuel/-tactile utilisant les 20 points sélectionnés aléatoirement (# 81 à 100) par objet (VT_NotSalient20_5).

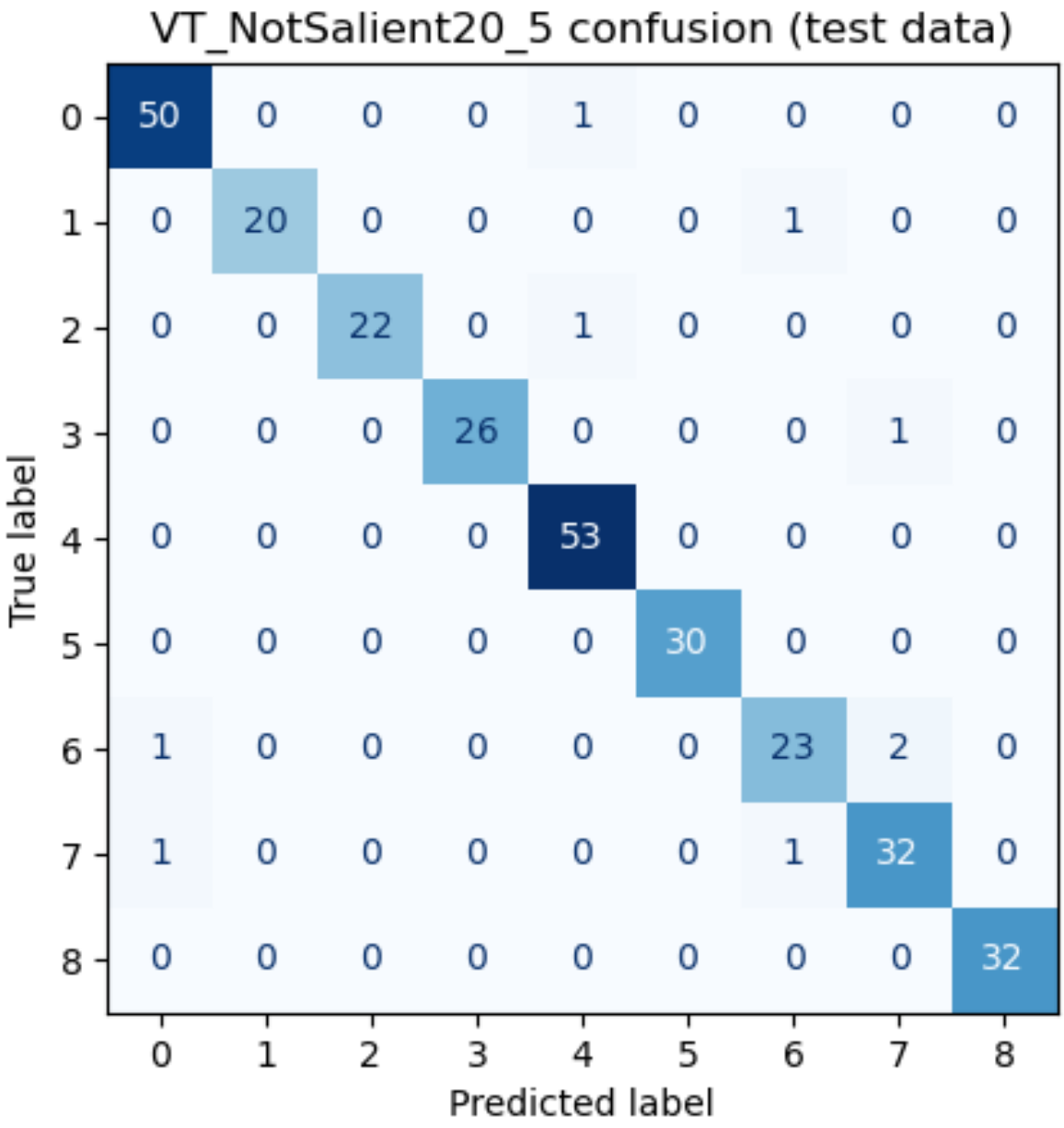

FIGURE K.17 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_5).

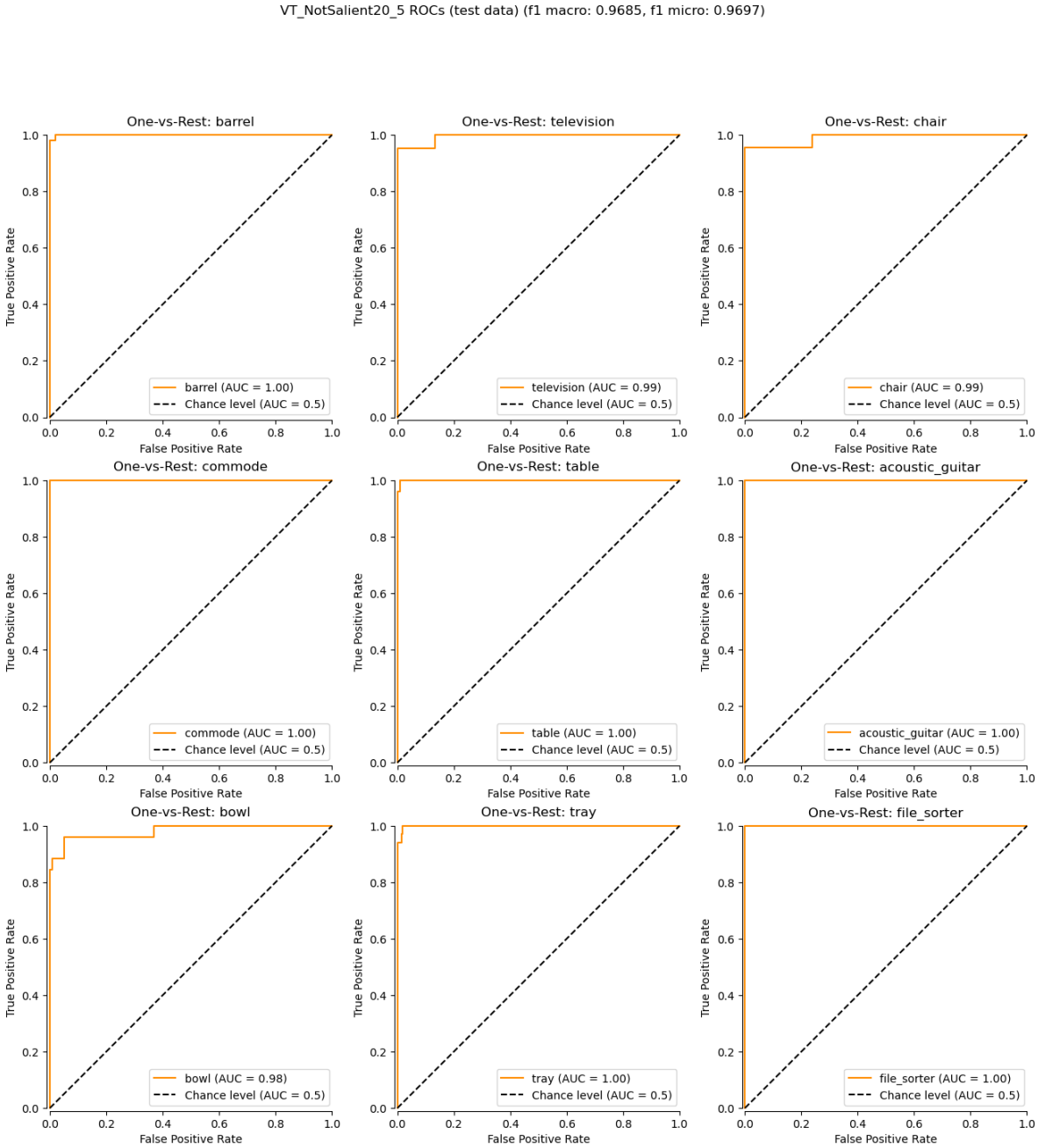

FIGURE K.18 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile (VT_NotSalient20_5).

Annexe L

Résultats de l'entraînement des méta-modèles Visuel+Tactile+Audio (MFCC)

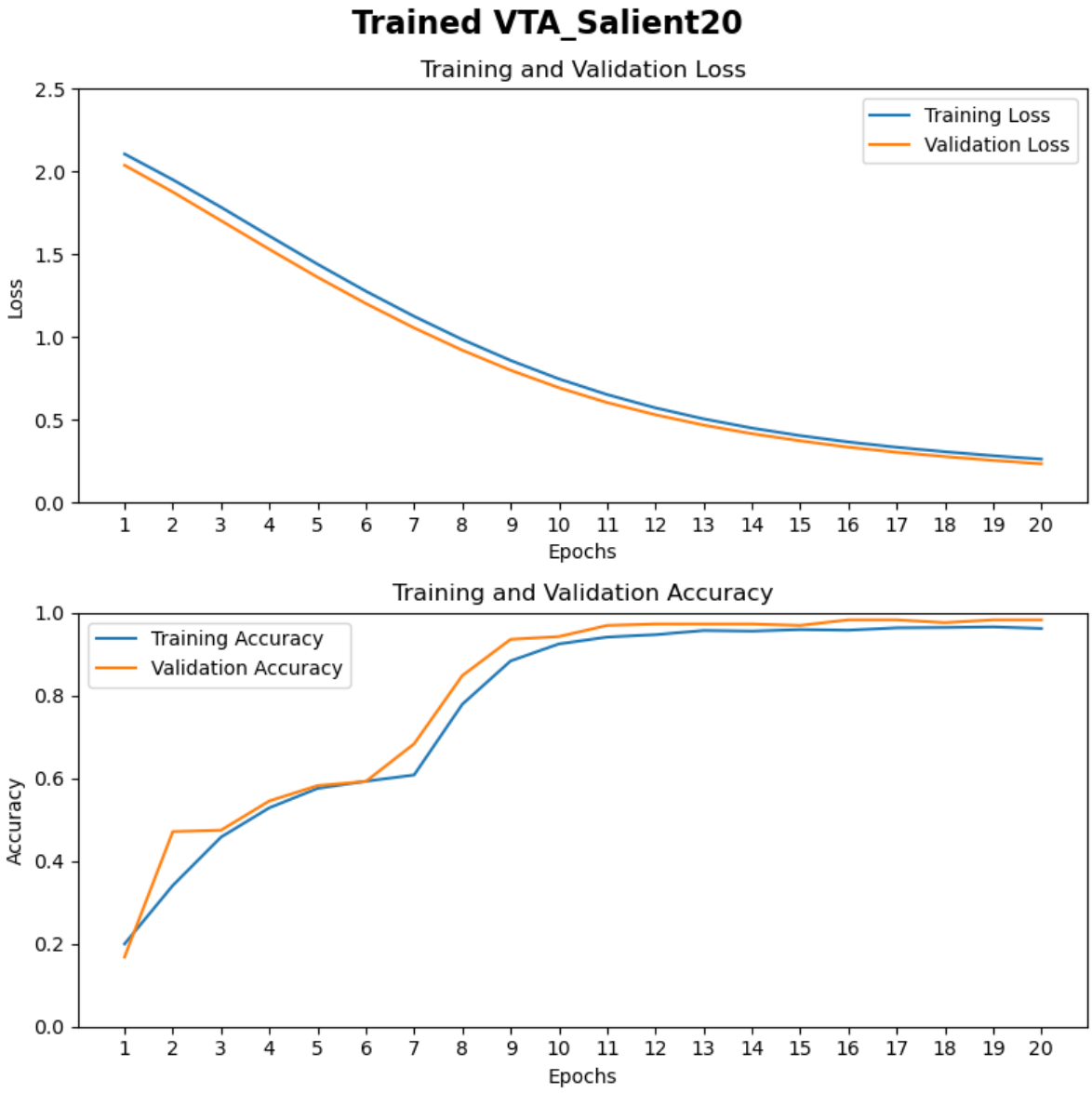

FIGURE L.1 – Historique de l'entraînement du méta-modèle combiné visuel/tactile/audio (MFCC) utilisant les 20 points saillants par objet (VTA_Salient20).

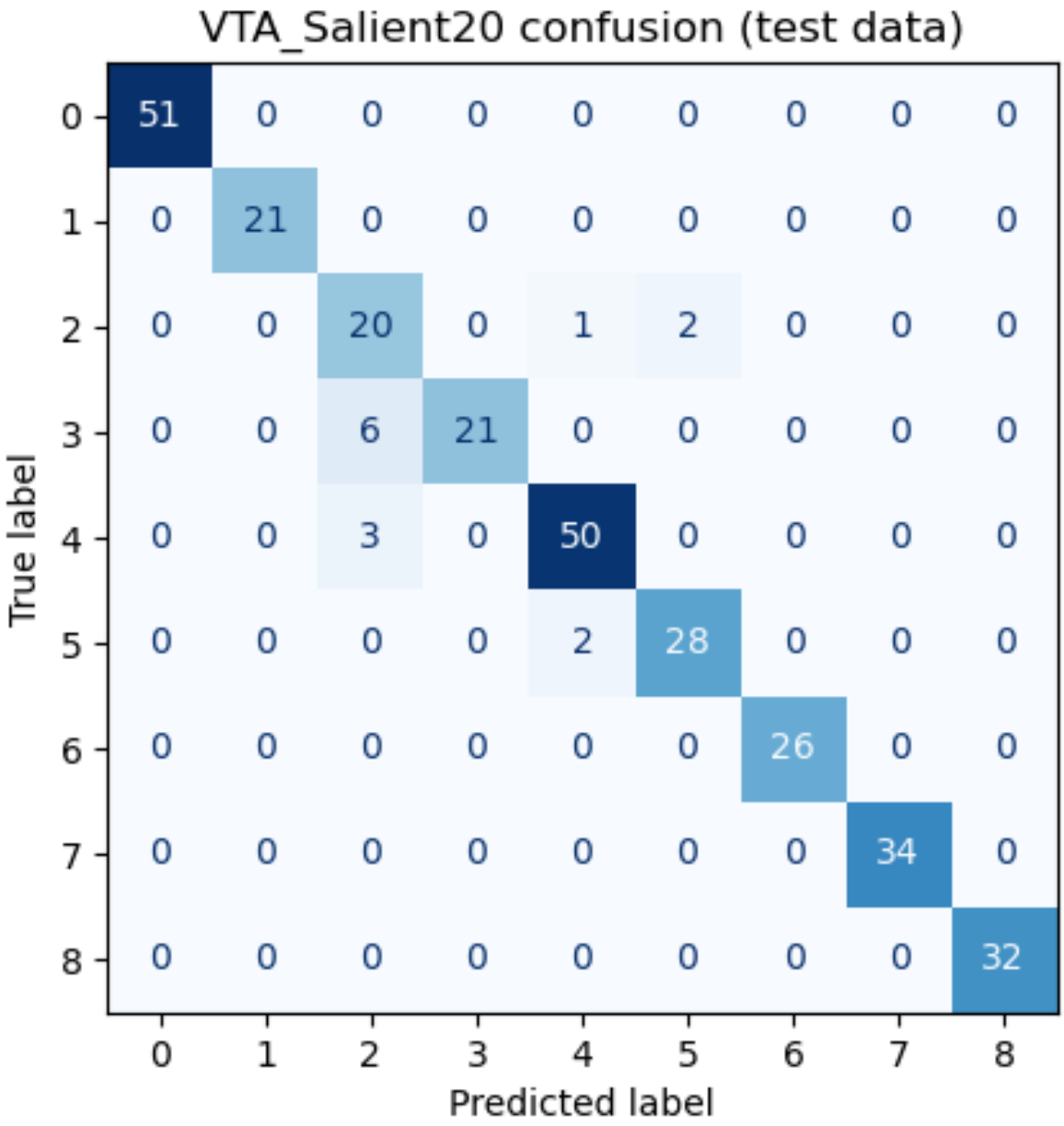

FIGURE L.2 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_Salient20).

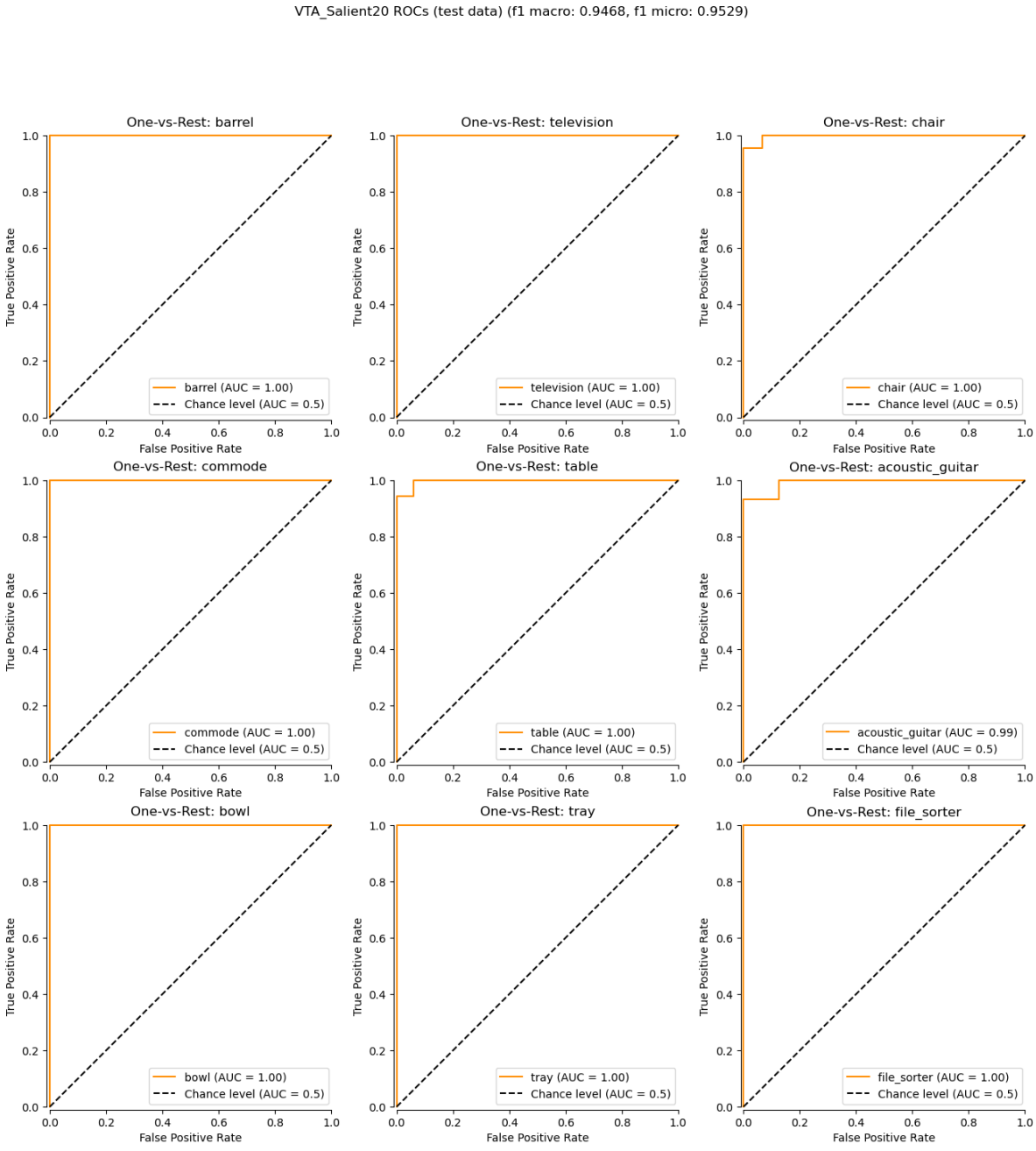

FIGURE L.3 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_Salient20).

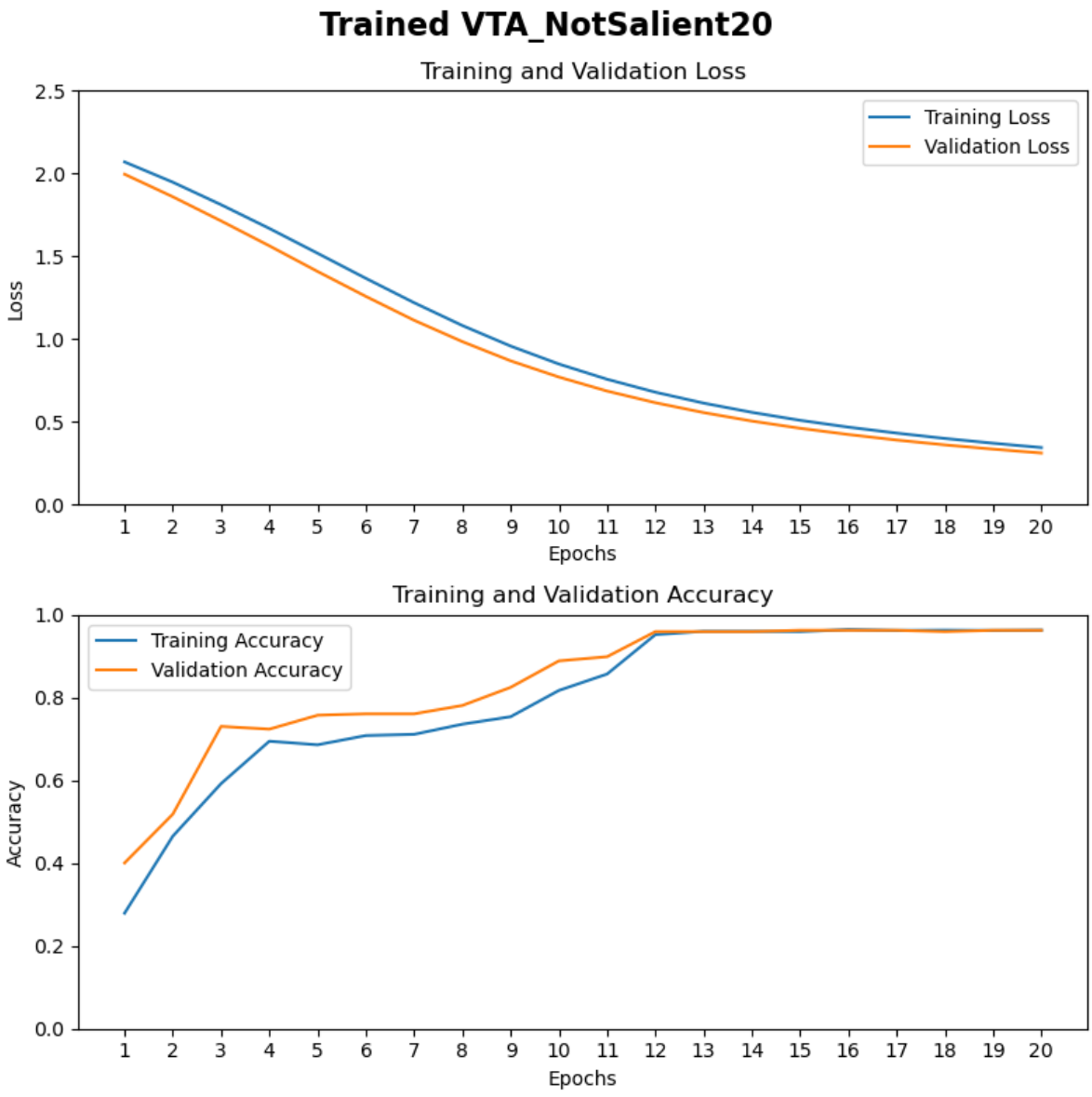

FIGURE L.4 – Historique de l'entraînement du méta-modèle combiné visuel/tactile/audio (MFCC) utilisant les 20 premiers points sélectionnés aléatoirement par objet (VTA_NotSalient20).

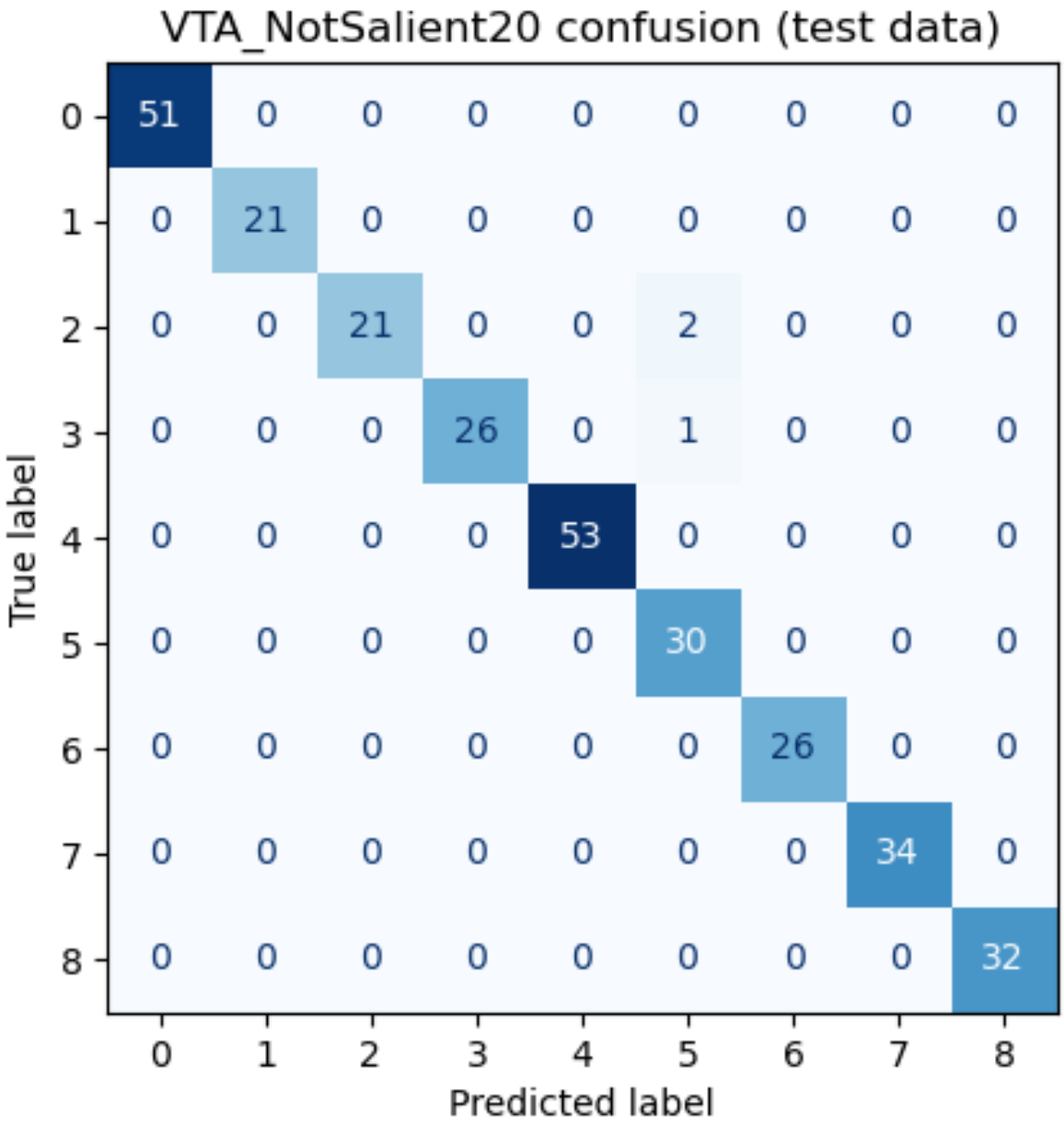

FIGURE L.5 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20).

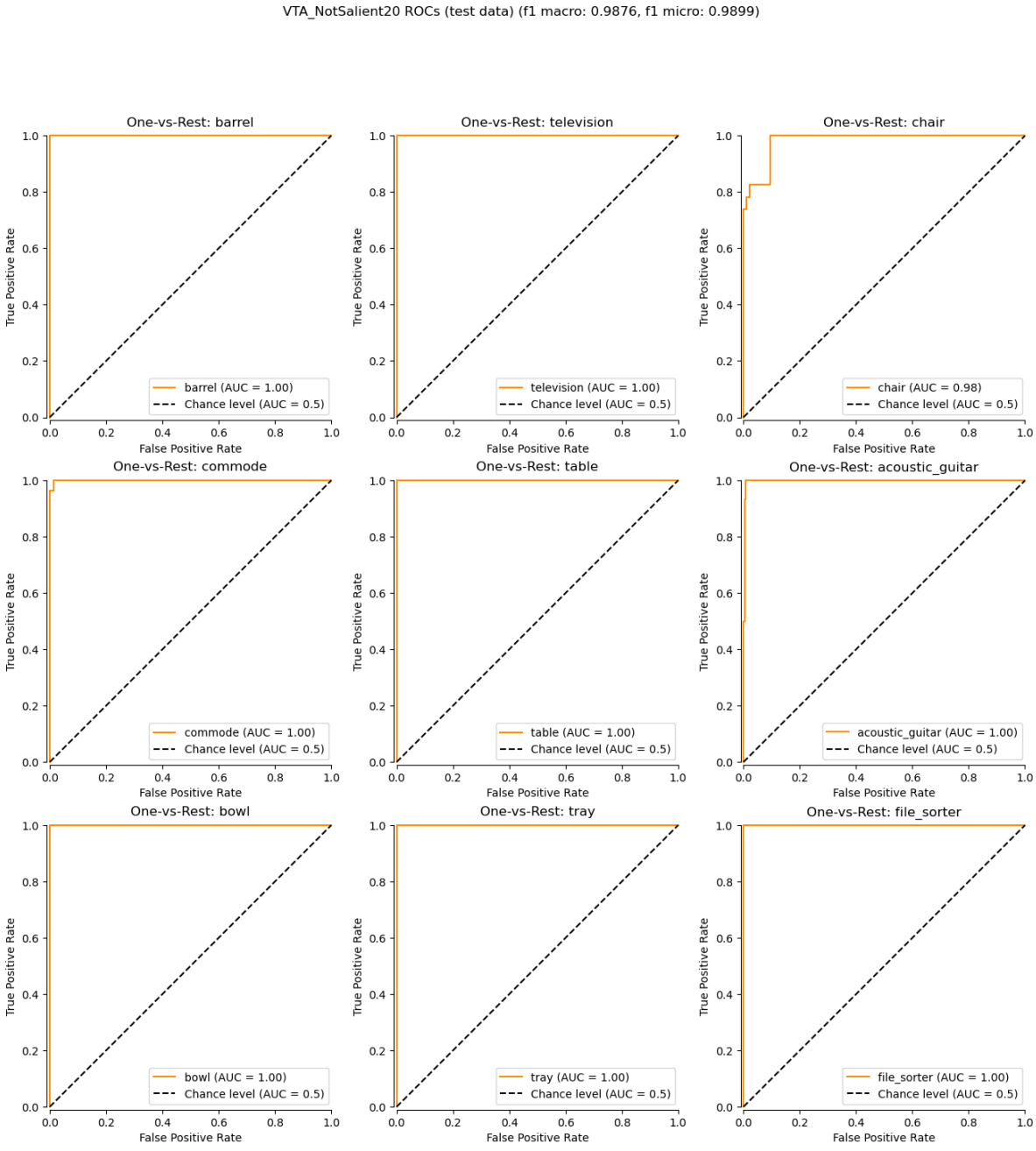

FIGURE L.6 – Courbes ROC et F1 Scores sur les données de test du combiné visuel/-tactile/audio (MFCC) (VTA_NotSalient20).

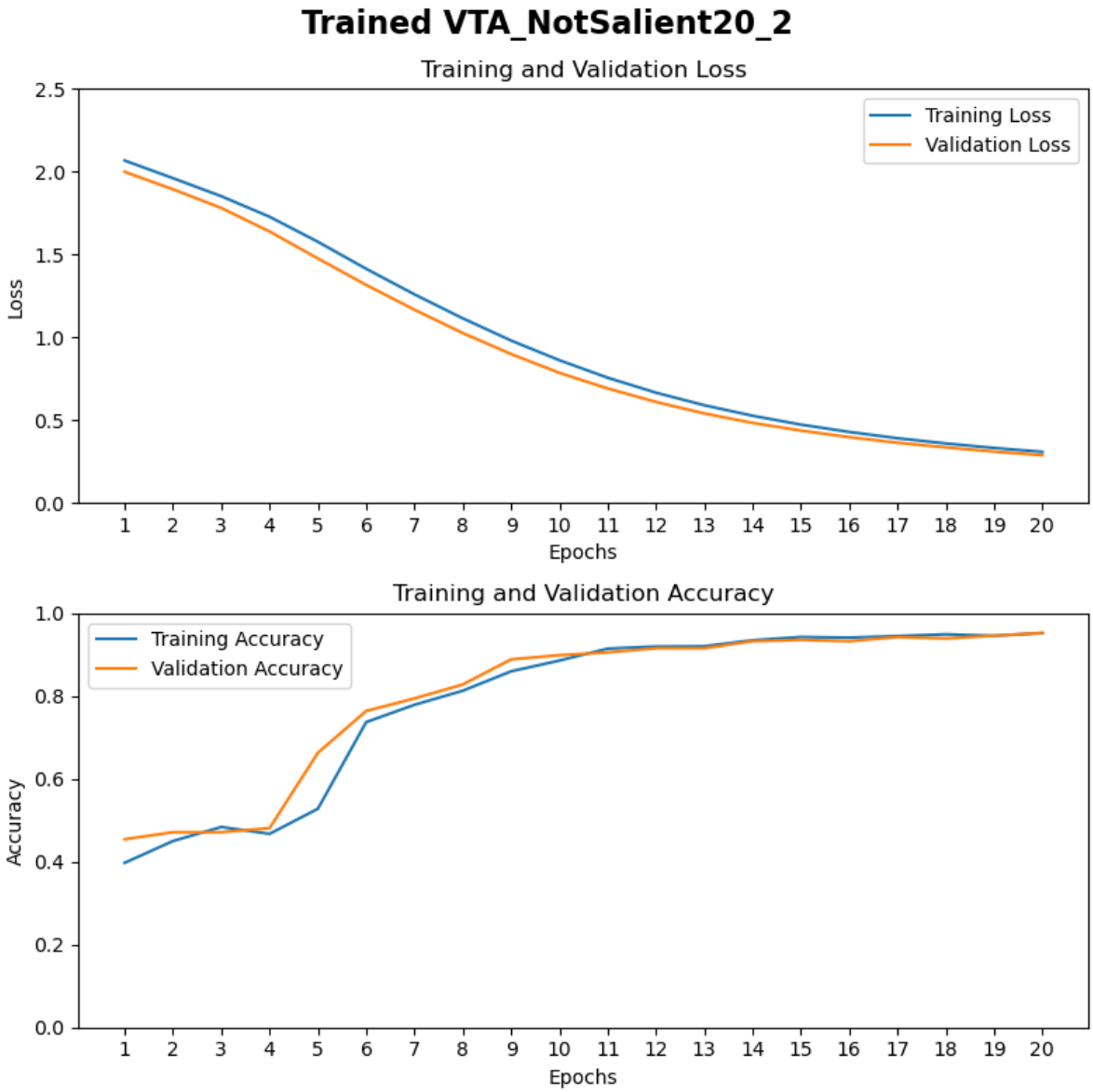

FIGURE L.7 – Historique de l'entraînement du méta-modèle combiné visuel/tactile/audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 21 à 40) par objet (VTA_NotSalient20_2).

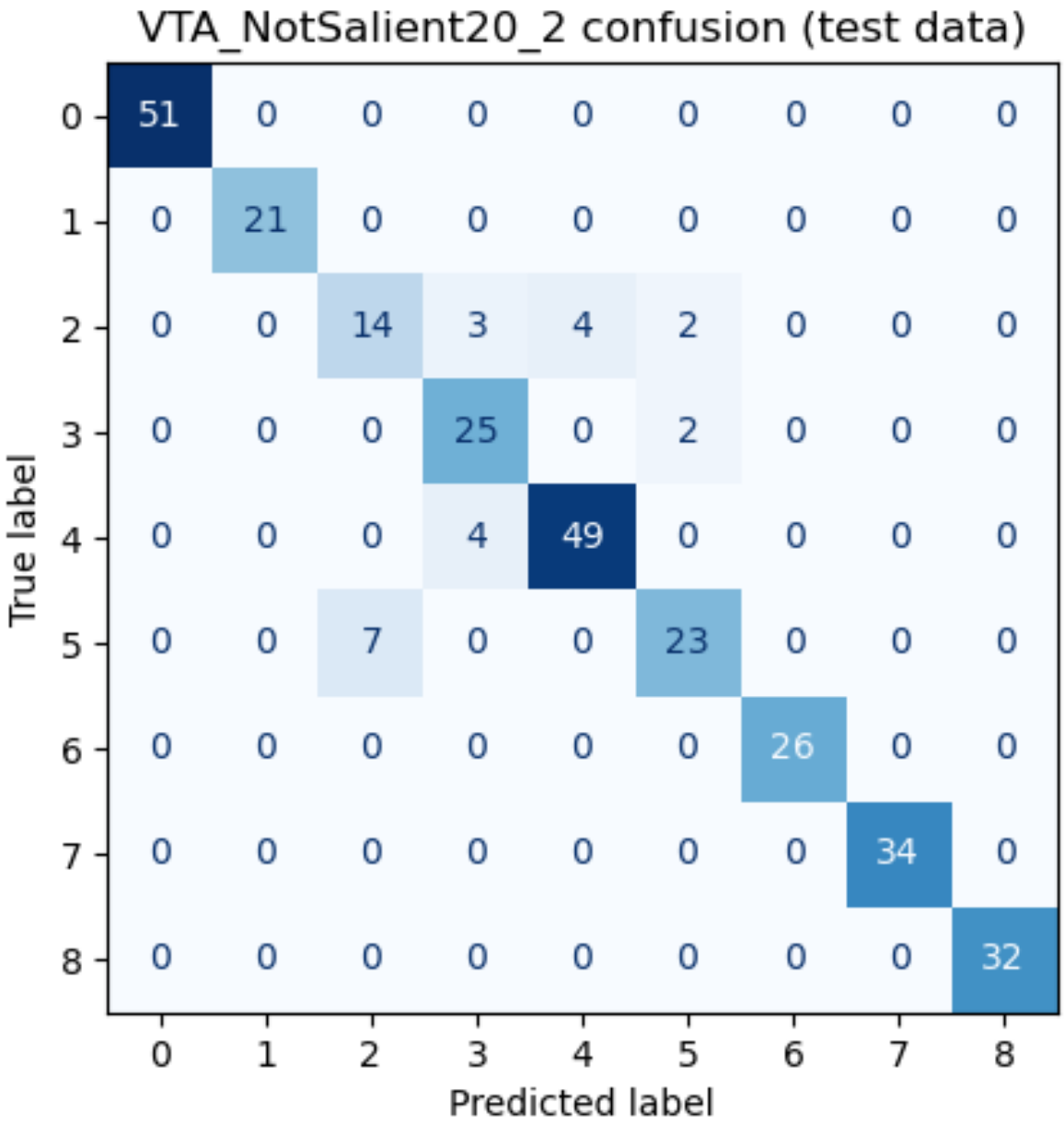

FIGURE L.8 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_2).

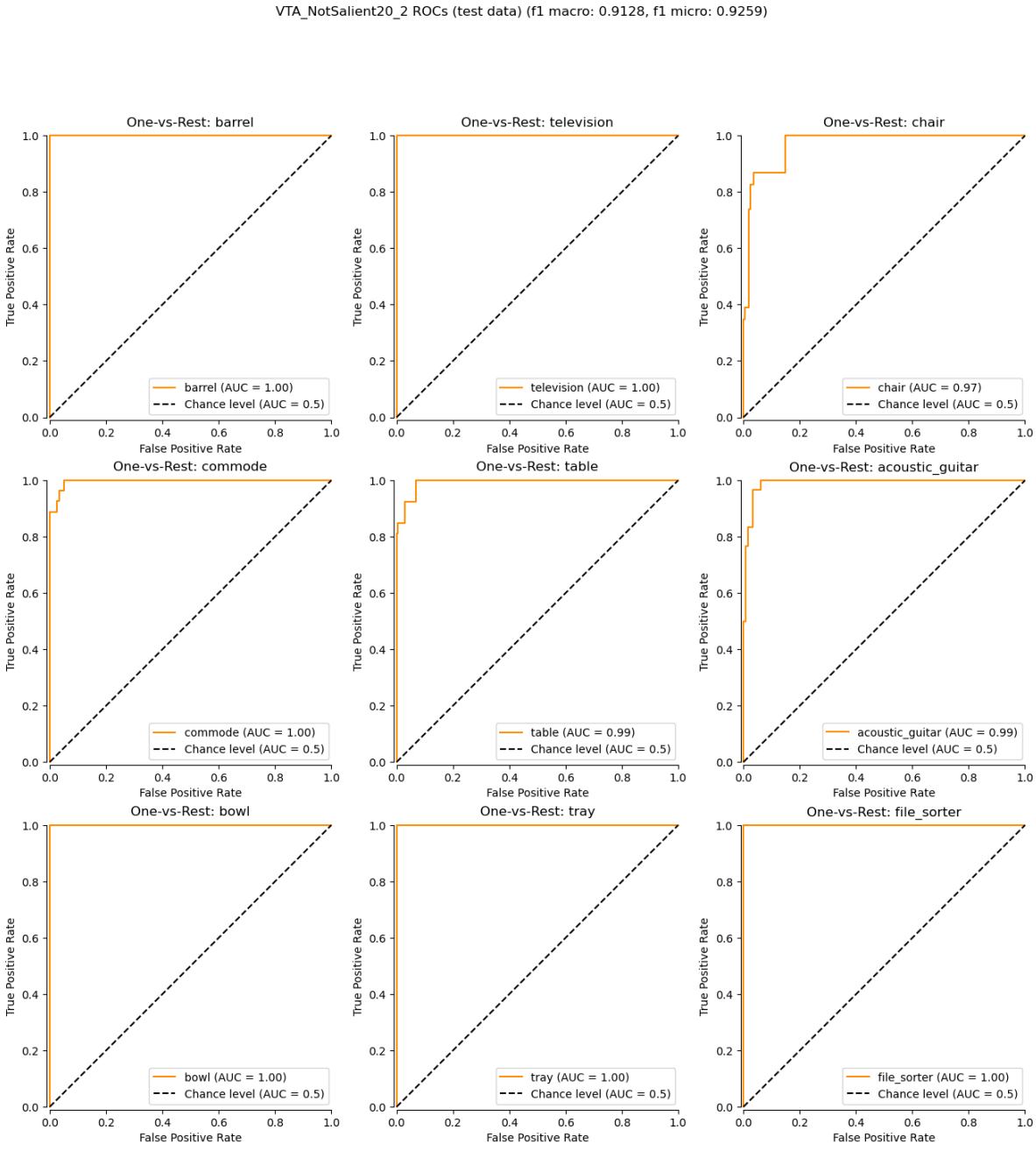

FIGURE L.9 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_2).

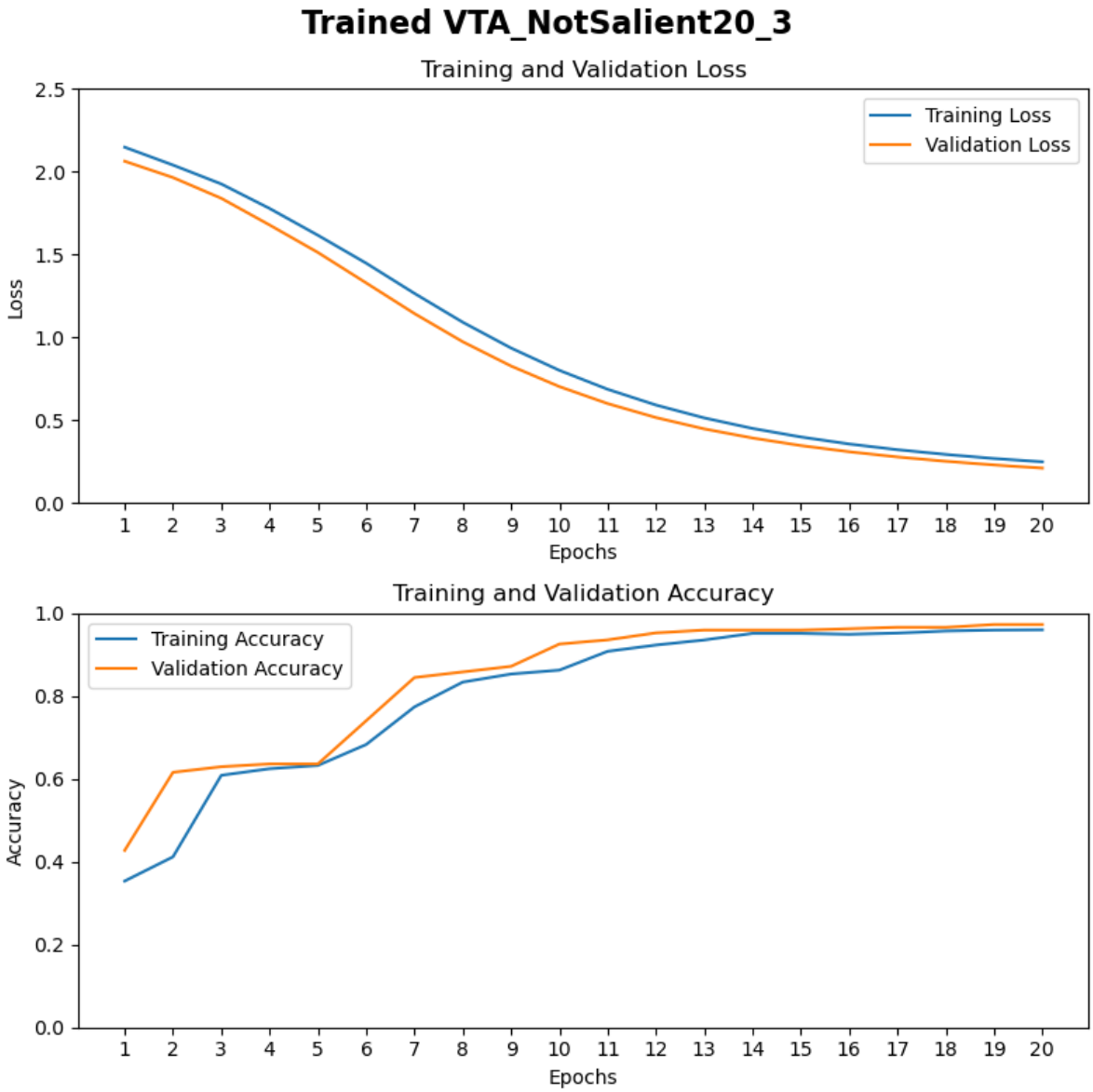

FIGURE L.10 – Historique de l'entraînement du méta-modèle combiné visuel/tactile/audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 41 à 60) par objet (VTA_NotSalient20_3).

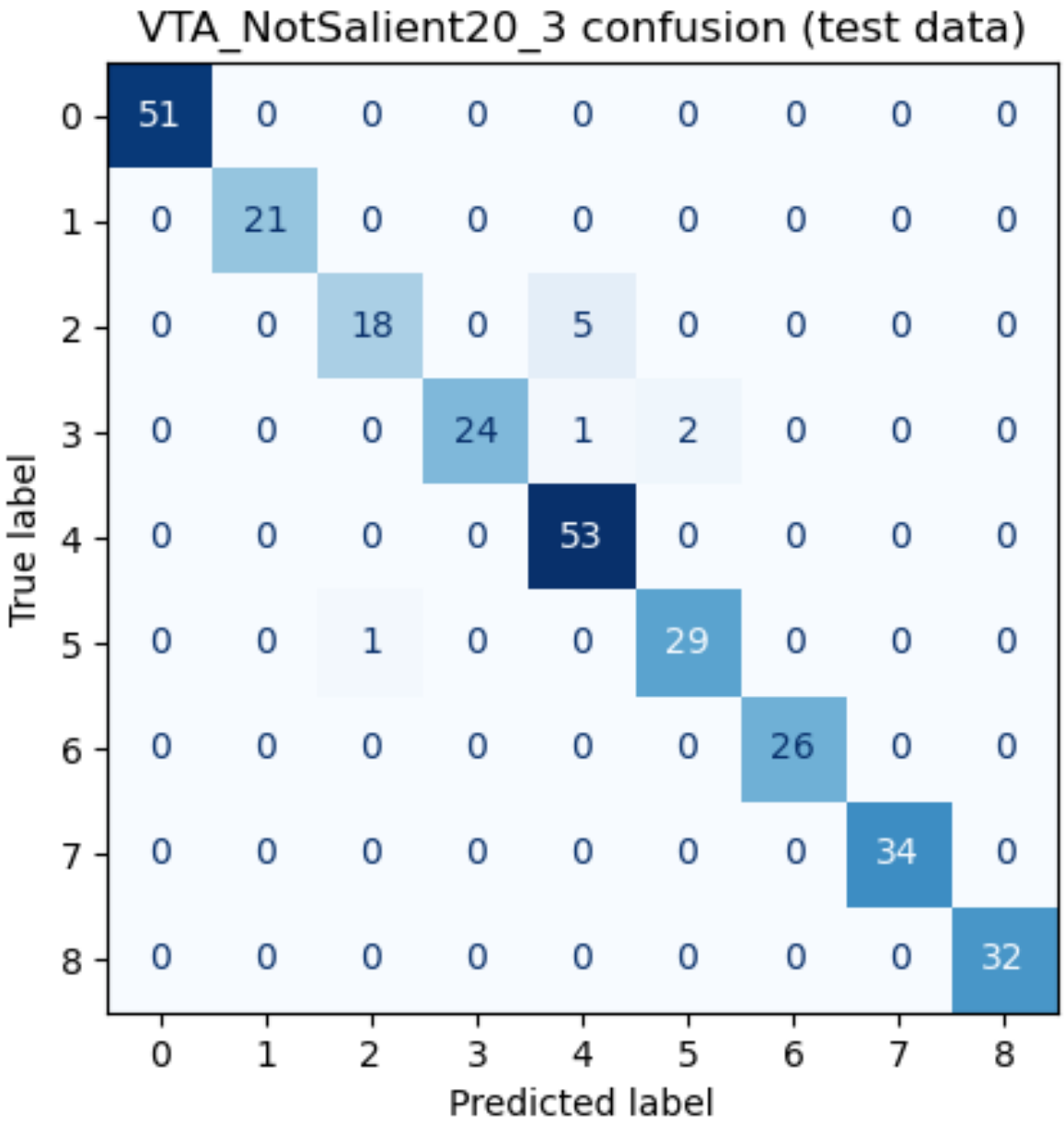

FIGURE L.11 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_3).

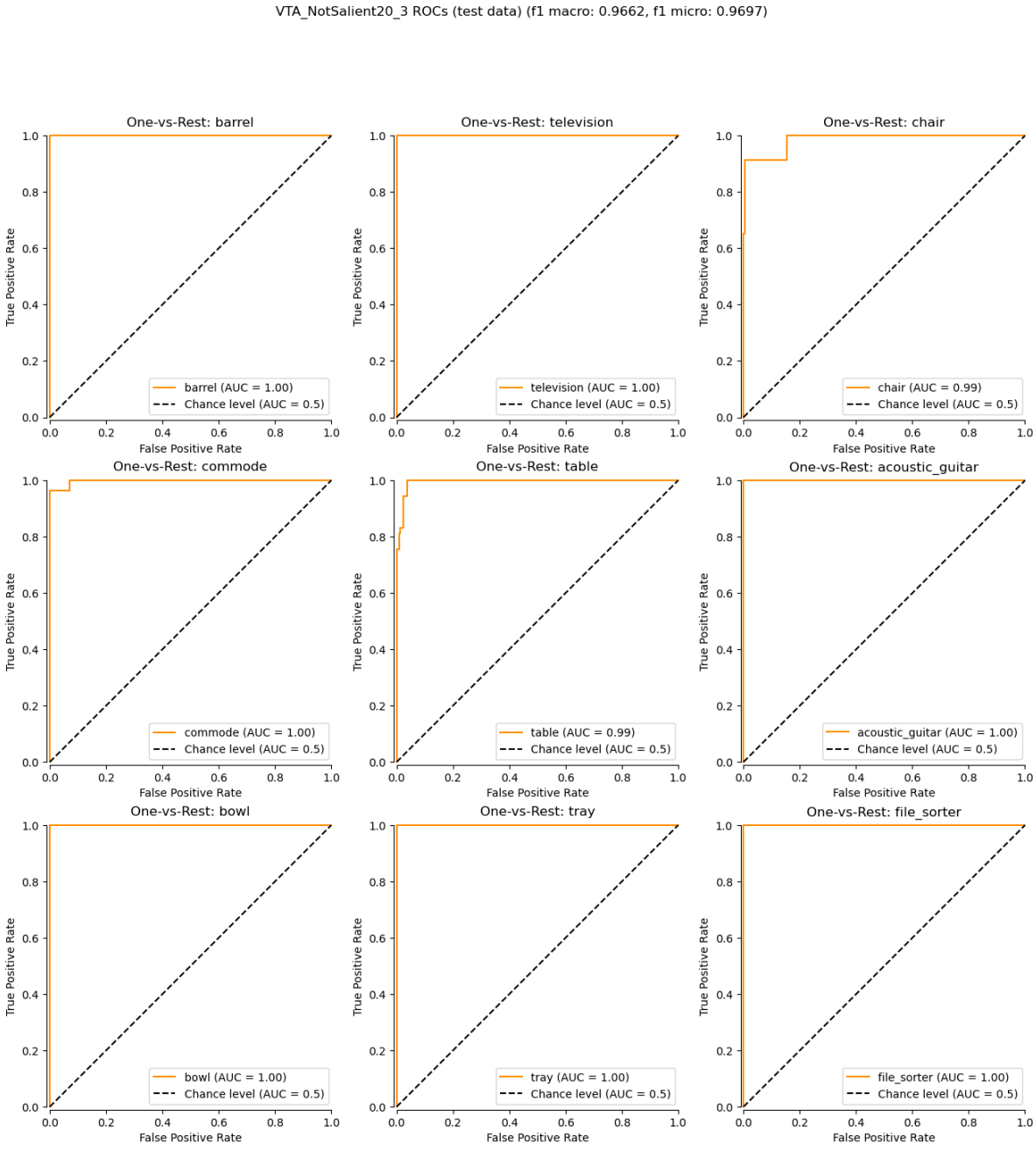

FIGURE L.12 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_3).

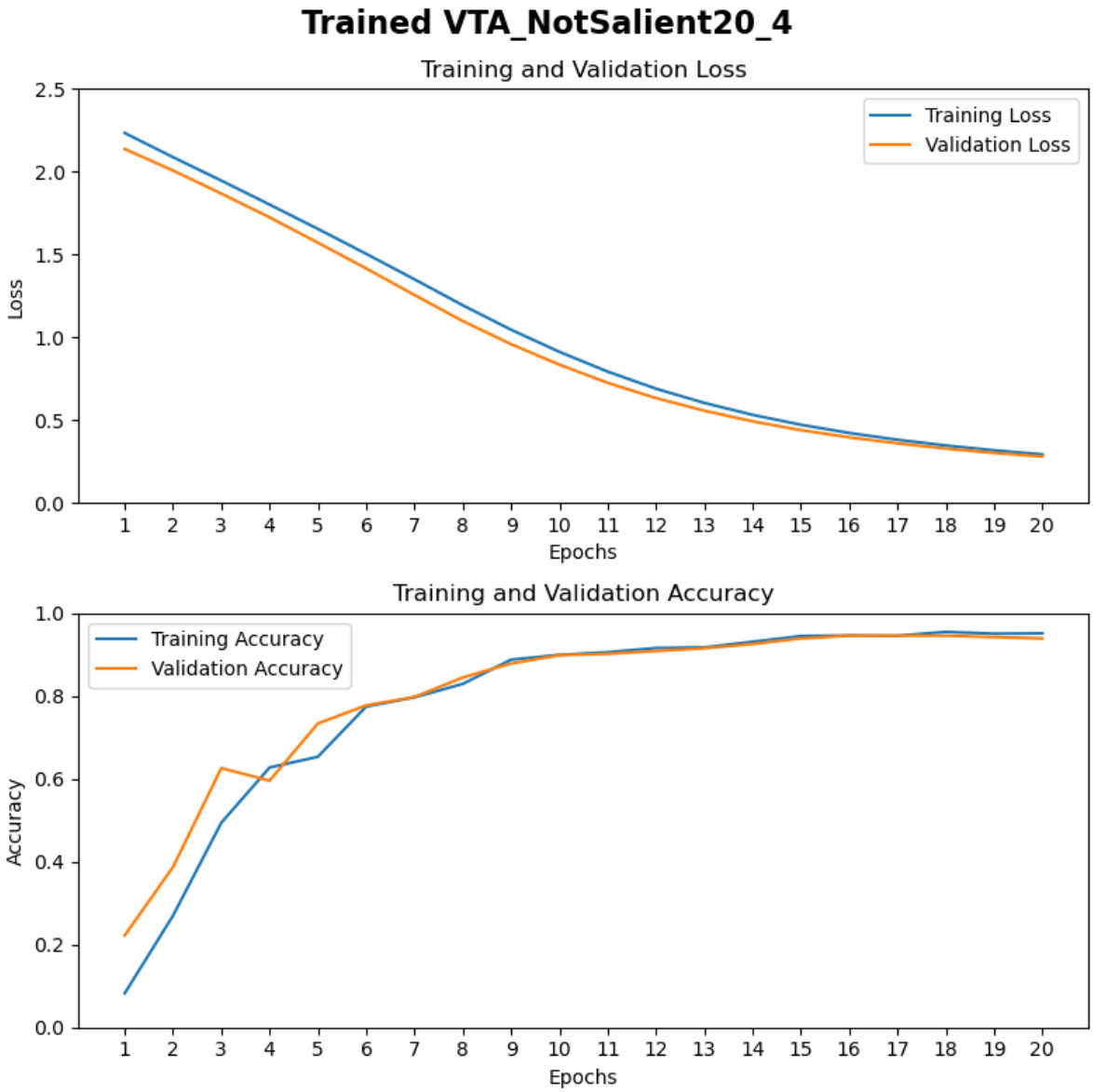

FIGURE L.13 – Historique de l'entraînement du méta-modèle combiné visuel/tactile/audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 61 à 80) par objet (VTA_NotSalient20_4).

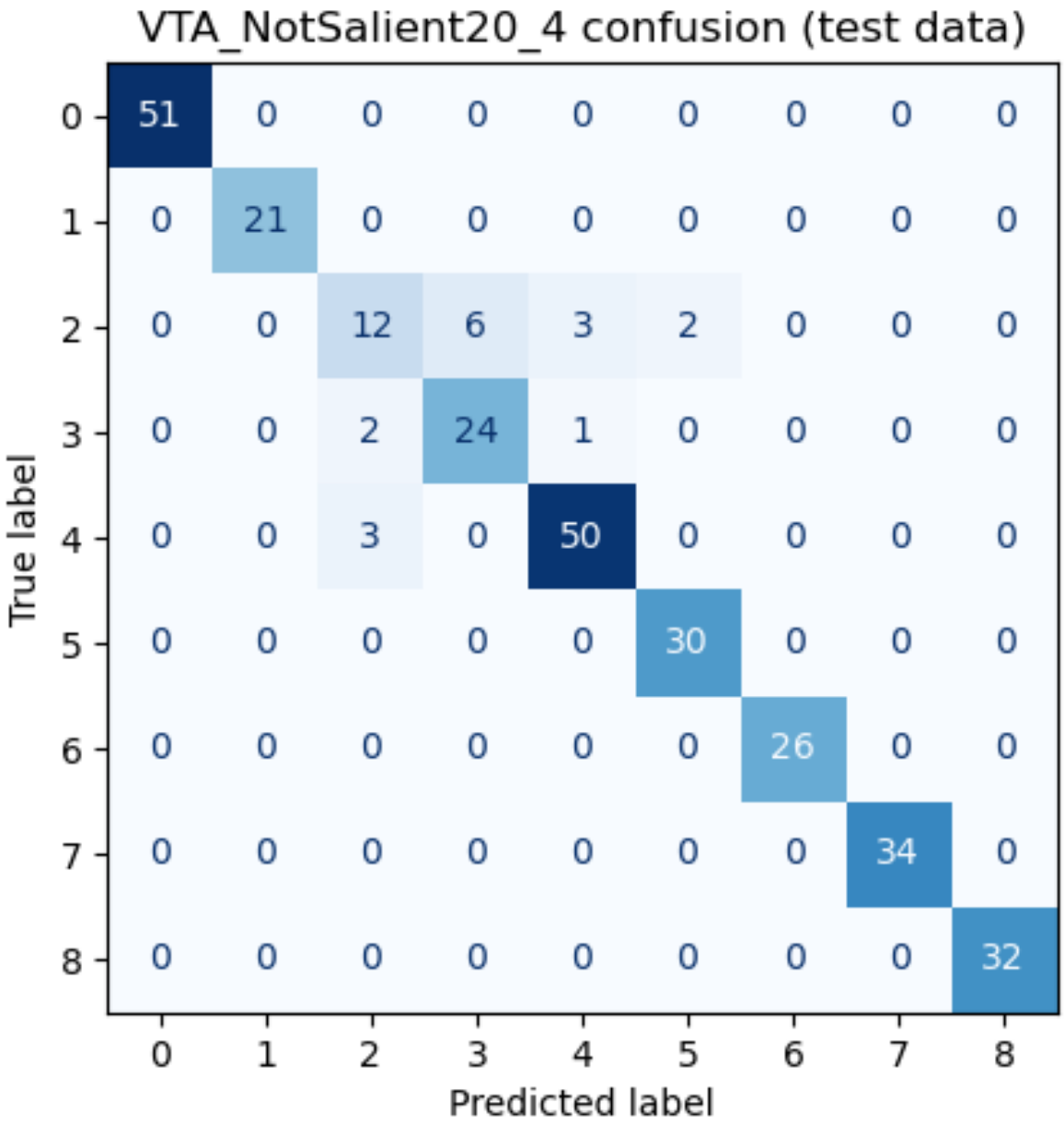

FIGURE L.14 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_4).

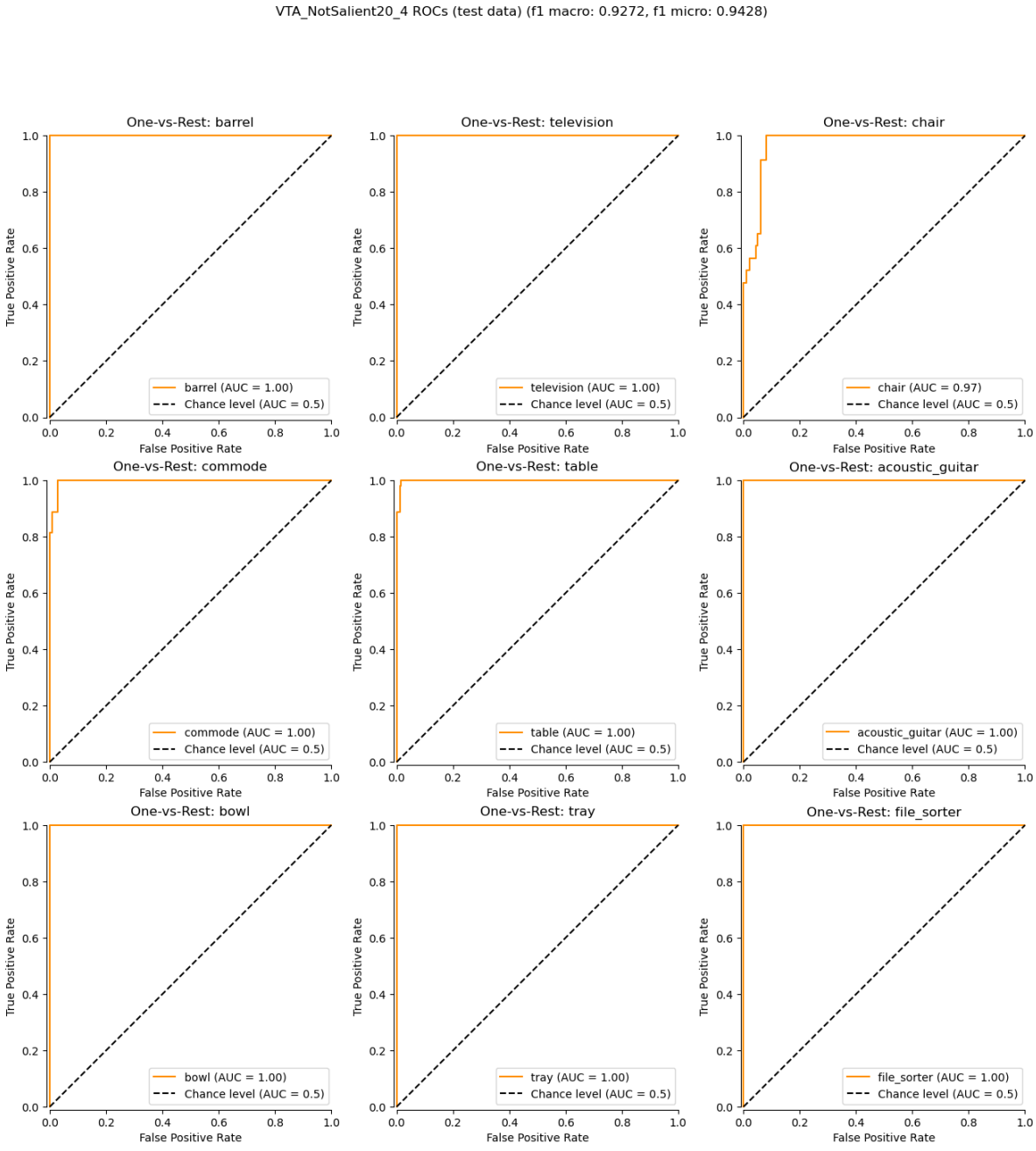

FIGURE L.15 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_4).

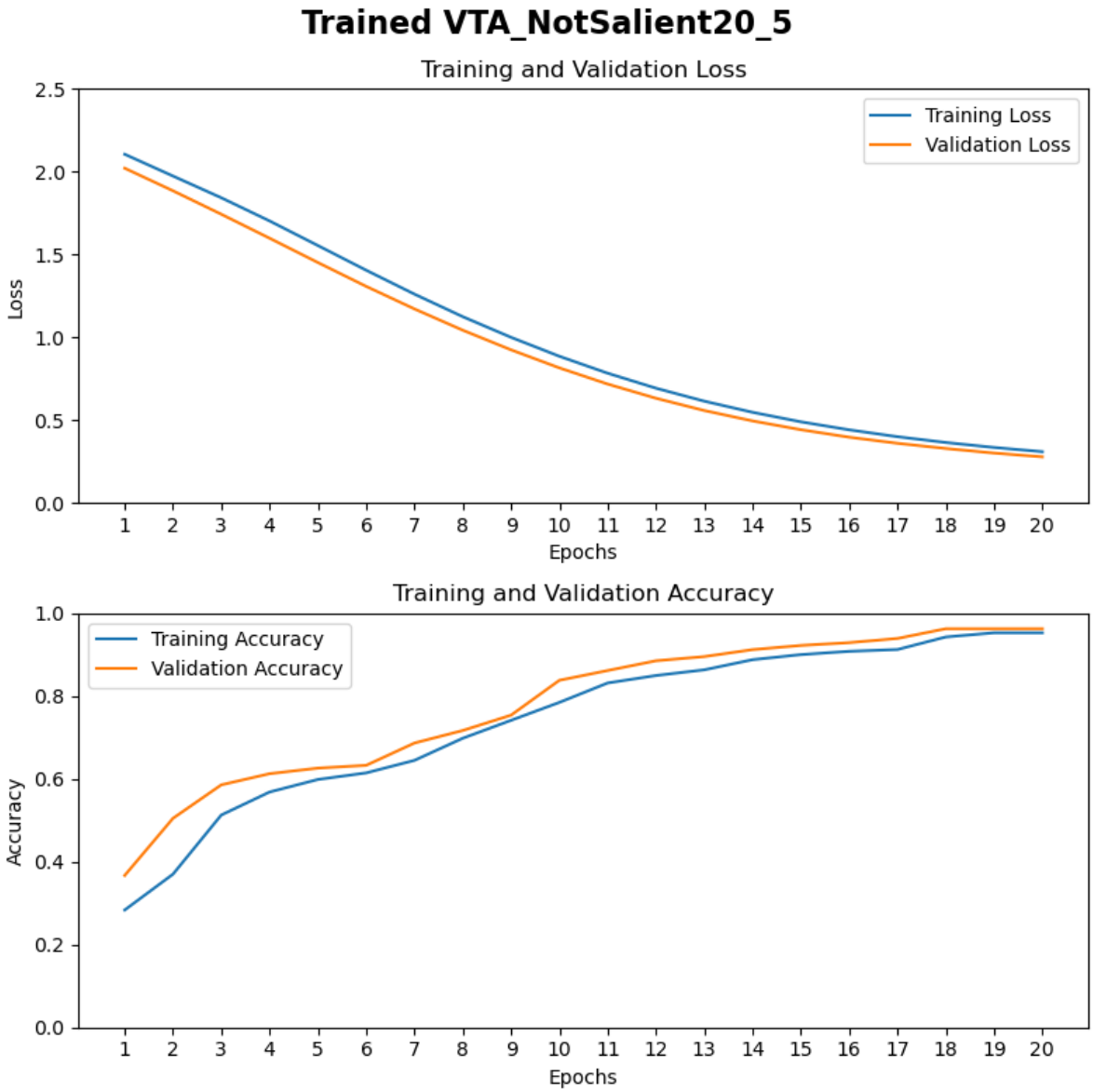

FIGURE L.16 – Historique de l'entraînement du méta-modèle combiné visuel/tactile/audio (MFCC) utilisant les 20 points sélectionnés aléatoirement (# 81 à 100) par objet (VTA_NotSalient20_5).

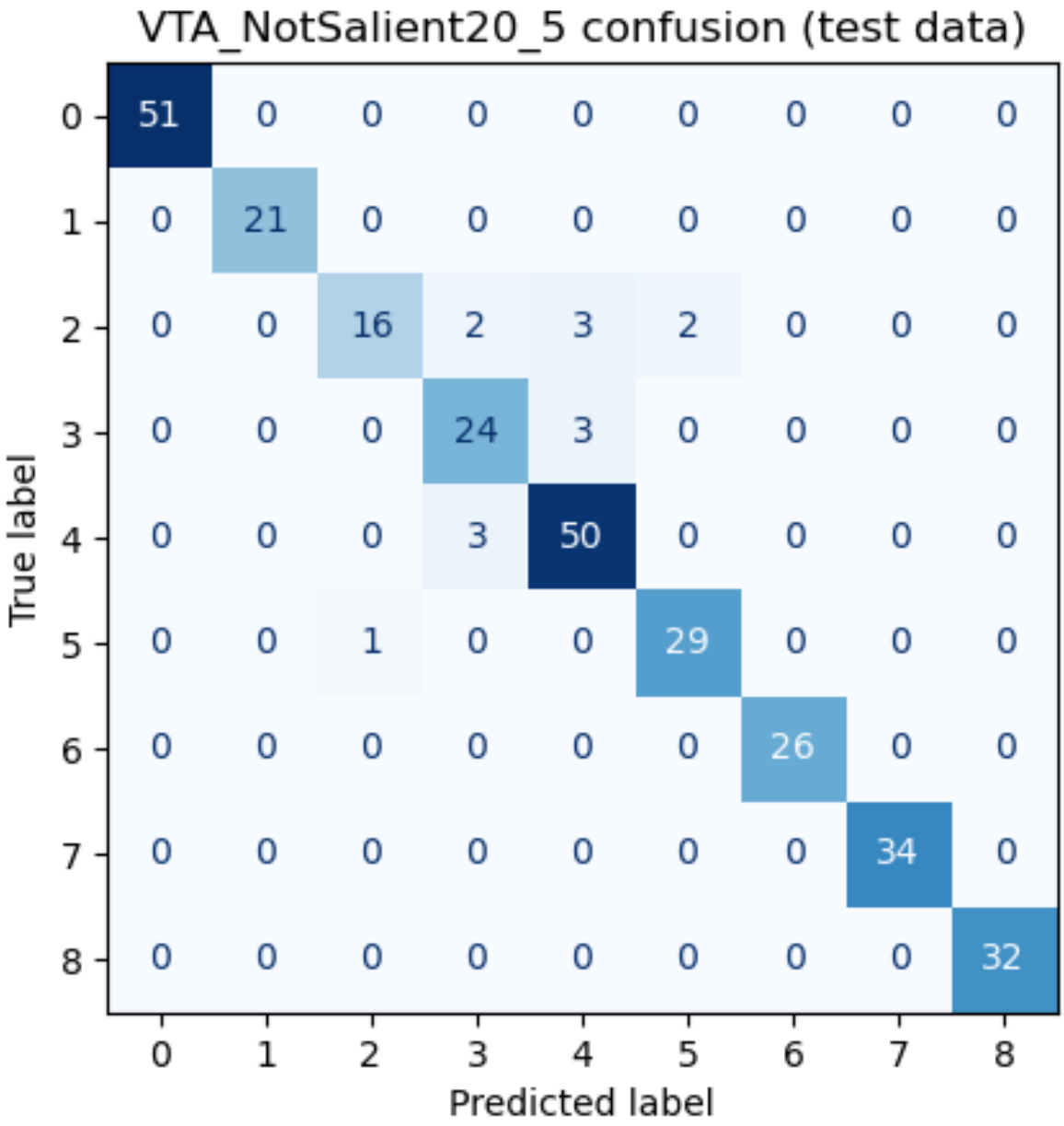

FIGURE L.17 – Matrice de confusion sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_5).

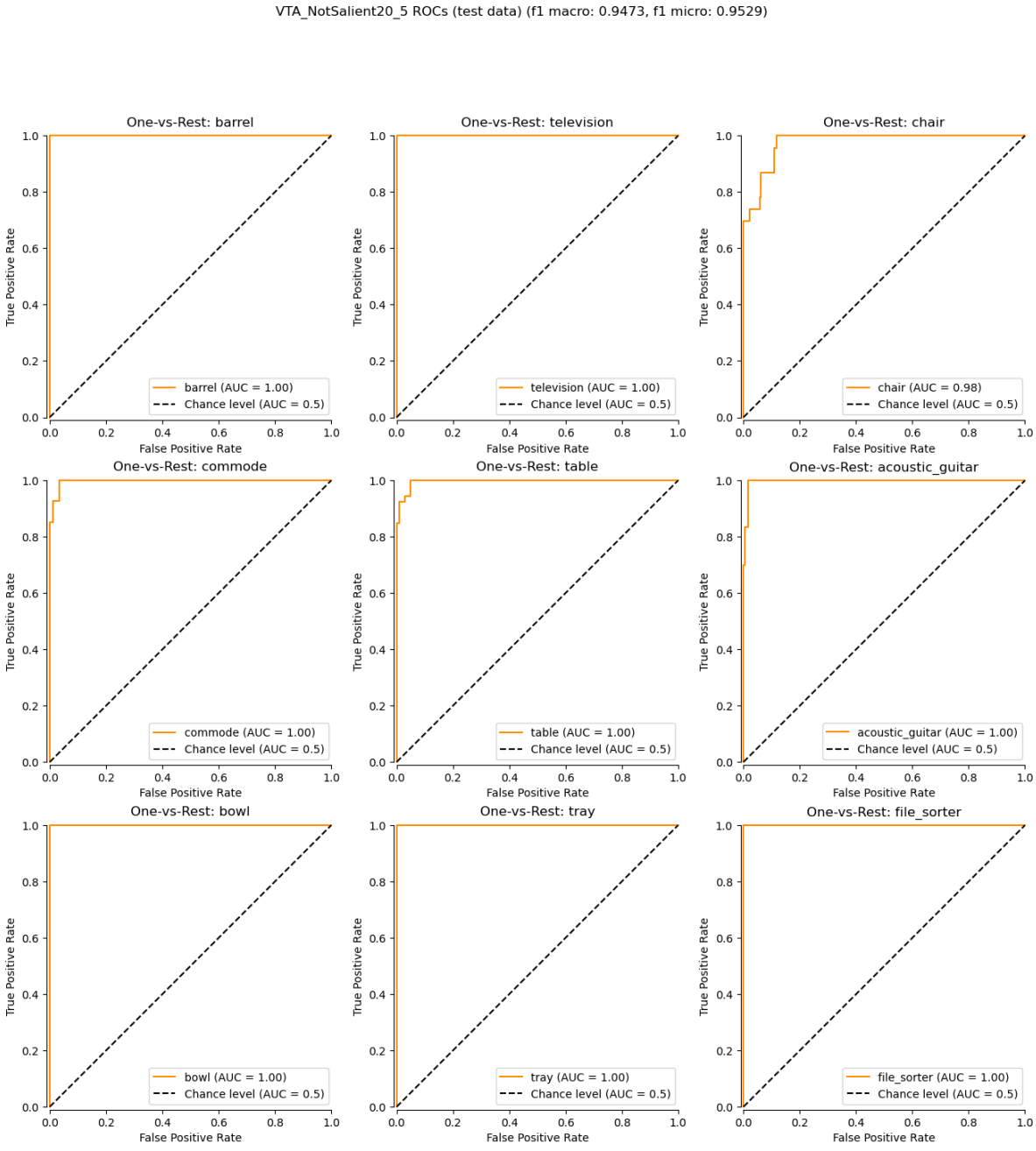

FIGURE L.18 – Courbes ROC et F1 Scores sur les données de test du méta-modèle combiné visuel/tactile/audio (MFCC) (VTA_NotSalient20_5).

Bibliographie

[1] R. Gordon. (2023) Image recognition accuracy : An unseen challenge confounding today's AI. Massachusetts Institute of Technology. [En Ligne]. Disponible sur : https://news.mit.edu/2023/image-recognition-accuracy-minimum-viewing-time-metric-1215 [Visité le : 2024-05-22]

[2] G. Rouhafzay, "3d object representation and recognition based on biologically inspired combined use of visual and tactile data," Ph.D. dissertation, University of Ottawa, Mai 2021. doi : 10.20381/ruor-26344

[3] A. Krizhevsky, I. Sutskever, et G. E. Hinton, "Imagenet classification with deep convolutional neural networks," dans *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, et K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [En Ligne]. Disponible sur : https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf [Visité le : 2024-05-24]

[4] K. Simonyan et A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [En Ligne]. Disponible sur : https://arxiv.org/abs/1409.1556 [Visité le : 2024-09-08]

[5] K. He, X. Zhang, S. Ren, et J. Sun, "Deep residual learning for image recognition," dans *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi : 10.1109/CVPR.2016.90 pp. 770–778.

[6] ——, "Identity mappings in deep residual networks," 2016. [En Ligne]. Disponible sur : https://arxiv.org/abs/1603.05027 [Visité le : 2024-09-08]

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, et H. Adam, "Mobilenets : Efficient convolutional neural networks for mobile vision applications," 2017. [En Ligne]. Disponible sur : https://arxiv.org/abs/1704.04861 [Visité le : 2024-09-08]

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, et L.-C. Chen, "Mobilenetv2 : Inverted residuals and linear bottlenecks," 2019. [En Ligne]. Disponible sur : https://arxiv.org/abs/1801.04381 [Visité le : 2024-09-08]

[9] R. Gao, Y. Dou, H. Li, Y.-Y. Chang, Z. Si, T. Agarwal, S. Clarke, S. Mall, Y. Li, J. Bohg, W. Yuan, L. Fei-Fei, et J. Wu. (2023) ObjectFolder – A dataset of multisensory neural and real objects and a benchmark suite for multisensory

object-centric learning, centered around object recognition, reconstruction, and manipulation with sight, sound, and touch. Stanford Vision and Learning Lab. [En Ligne]. Disponible sur : https://objectfolder.stanford.edu/ [Visité le : 2024-03-11]

[10] R. Gao, Y.-Y. Chang, S. Mall, L. Fei-Fei, et J. Wu, "Objectfolder : A dataset of objects with implicit visual, auditory, and tactile representations," dans *CoRL*, 2021. [En Ligne]. Disponible sur : https://arxiv.org/abs/2109.07991 [Visité le : 2025-09-30]

[11] R. Gao, Z. Si, Y.-Y. Chang, S. Clarke, J. Bohg, L. Fei-Fei, W. Yuan, et J. Wu, "Objectfolder 2.0 : A multisensory object dataset for sim2real transfer," dans *CVPR*, 2022. [En Ligne]. Disponible sur : https://arxiv.org/abs/2204.02389 [Visité le : 2025-09-30]

[12] R. Gao, Y. Dou, H. Li, T. Agarwal, J. Bohg, Y. Li, L. Fei-Fei, et J. Wu, "The objectfolder benchmark : Multisensory learning with neural and real objects," dans *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 276–17 286. [En Ligne]. Disponible sur : https://arxiv.org/abs/2306.00956 [Visité le : 2025-09-30]

[13] P. Viola et M. Jones, "Rapid object detection using a boosted cascade of simple features," dans *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001. doi : 10.1109/CVPR.2001.990517 pp. I–I.

[14] C. Borah. (2020, Nov.) Evolution of object detection. Medium – Analytics Vidhya. [En Ligne]. Disponible sur : https://medium.com/analytics-vidhya/evolution-of-object-detection-582259d2aa9b [Visité le : 2025-11-02]

[15] Z. K. Ku, C. F. Ng, et S. W. Khor, "Shape-based recognition and classification for common objects - an application in video scene analysis," dans *2010 2nd International Conference on Computer Engineering and Technology*, vol. 3, 2010. doi : 10.1109/ICCET.2010.5485747 pp. V3–13–V3–16.

[16] D. Ai, X. Han, X. Ruan, et Y.-W. Chen, "Adaptive color independent components based sift descriptors for image classification," dans *2010 20th International Conference on Pattern Recognition*, 2010. doi : 10.1109/ICPR.2010.596 pp. 2436–2439.

[17] J. Z. Wang, "Simplicity : a region-based retrieval system for picture libraries and biomedical image databases," dans *Proceedings of the Eighth ACM International Conference on Multimedia*, ser. MULTIMEDIA '00. New York, NY, USA : Association for Computing Machinery, 2000. doi : 10.1145/354384.376395. ISBN 1581131984 p. 483–484.

[18] K. Selvaraj, A. A. Fathima, et V. Vaidehi, "Multi-class object detection by part based approach," dans *2012 International Conference on Recent Trends in Information Technology*, 2012. doi : 10.1109/ICRTIT.2012.6206837 pp. 114–118.

[19] T. Dettmers. (2022, Août) Deep learning in a nutshell : Core concepts. NVIDIA Technical Blog. [En Ligne]. Disponible sur : https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/ [Visité le : 2025-11-09]

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, et L. Fei-Fei, "Imagenet : A large-scale hierarchical image database," dans *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. doi : 10.1109/CVPR.2009.5206848 pp. 248–255.

[21] C. Fellbaum, *WordNet : An Electronic Lexical Database*. Bradford Books, 1998. [En Ligne]. Disponible sur : https://mitpress.mit.edu/9780262561167/ [Visité le : 2024-05-25]

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, et L. Fei-Fei, "Imagenet large scale visual recognition challenge," 2015. [En Ligne]. Disponible sur : https://arxiv.org/abs/1409.0575 [Visité le : 2024-09-08]

[23] S. L. Fjodor van Veen. (2016, Sep.) The Neural Network Zoo. The Asimov Institute. [En Ligne]. Disponible sur : https://www.asimovinstitute.org/neural-network-zoo/ [Visité le : 2026-02-10]

[24] A.-M. Cretu, "INF6333 – Intelligence artificielle appliquée : Réseaux de neurones. Apprentissage profond. Architectures modernes de réseaux de neurones convolutifs." Université du Québec en Outaouais, 2024.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, et A. Rabinovich, "Going deeper with convolutions," 2014. [En Ligne]. Disponible sur : https://arxiv.org/abs/1409.4842 [Visité le : 2024-09-08]

[26] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," 2017. [En Ligne]. Disponible sur : https://arxiv.org/abs/1611.10012 [Visité le : 2025-11-10]

[27] S. Ioffe et C. Szegedy, "Batch normalization : Accelerating deep network training by reducing internal covariate shift," 2015. [En Ligne]. Disponible sur : https://arxiv.org/abs/1502.03167 [Visité le : 2025-11-10]

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, et A. C. Berg, *SSD : Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. doi : 10.1007/978-3-319-46448-0_2. ISBN 9783319464480. ISSN 1611-3349

[29] G. Lavoué, F. Cordier, H. Seo, et M.-C. Larabi, "Visual attention for rendered 3d shapes," *Computer Graphics Forum*, vol. 37, no. 2, pp. 191–203, 2018. doi : 10.1111/cgf.13353. [En Ligne]. Disponible sur : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13353 [Visité le : 2024-07-01]

[30] H. Dutagaci, C. P. Cheung, et A. Godil, "Evaluation of 3d interest point detection techniques via human-generated ground truth," *The Visual Computer*, vol. 28, no. 9, p. 901–917, Juin 2012. doi : 10.1007/s00371-012-0746-4. ISSN 1432-2315. [En Ligne]. Disponible sur : http://dx.doi.org/10.1007/s00371-012-0746-4 [Visité le : 2025-09-30]

[31] H. Hughes et L. Zimba, "Natural boundaries for the spatial spread of directed visual attention," *Neuropsychologia*, vol. 25, no. 1, p. 5–18, 1987. doi :

10.1016/0028-3932(87)90039-x. ISSN 0028-3932. [En Ligne]. Disponible sur : http://dx.doi.org/10.1016/0028-3932(87)90039-X [Visité le : 2025-09-30]

[32] A. Amedi, R. Malach, T. Hendler, S. Peled, et E. Zohary, "Visuo-haptic object-related activation in the ventral visual pathway," *Nature Neuroscience*, vol. 4, no. 3, p. 324–330, Mar. 2001. doi : 10.1038/85201. ISSN 1546-1726. [En Ligne]. Disponible sur : http://dx.doi.org/10.1038/85201 [Visité le : 2025-09-30]

[33] P. Allen et K. Roberts, "Haptic object recognition using a multi-fingered dextrous hand," dans *Proceedings, 1989 International Conference on Robotics and Automation*. IEEE Comput. Soc. Press, 1989. doi : 10.1109/robot.1989.100011 p. 342–347. [En Ligne]. Disponible sur : http://dx.doi.org/10.1109/ROBOT.1989.100011 [Visité le : 2025-09-30]

[34] S. Kennett, M. Eimer, C. Spence, et J. Driver, "Tactile-visual links in exogenous spatial attention under different postures : Convergent evidence from psychophysics and erps," *Journal of Cognitive Neuroscience*, vol. 13, no. 4, p. 462–478, Mai 2001. doi : 10.1162/08989290152001899. ISSN 1530-8898. [En Ligne]. Disponible sur : http://dx.doi.org/10.1162/08989290152001899 [Visité le : 2025-09-30]

[35] Z. Pezzementi, E. Plaku, C. Reyda, et G. D. Hager, "Tactile-object recognition from appearance information," *IEEE Transactions on Robotics*, vol. 27, no. 3, pp. 473–487, 2011. doi : 10.1109/TRO.2011.2125350

[36] S. J. Lederman et R. L. Klatzky, "Haptic perception : A tutorial," *Attention, Perception & Psychophysics*, vol. 71, no. 7, p. 1439–1459, Sep. 2009. doi : 10.3758/app.71.7.1439. ISSN 1943-393X

[37] W. Yuan, S. Dong, et E. Adelson, "Gelsight : High-resolution robot tactile sensors for estimating geometry and force," *Sensors*, vol. 17, no. 12, p. 2762, Nov. 2017. doi : 10.3390/s17122762. ISSN 1424-8220

[38] T. M. Corradi, "Integrating visual and tactile robotic perception," Ph.D. dissertation, University of Bath, 2018. [En Ligne]. Disponible sur : https://files01.core.ac.uk/download/pdf/161920381.pdf [Visité le : 2026-01-04]

[39] E. M. Petriu, P. Payeur, A.-M. Cretu, et C. Pasca, "Complementary tactile sensor and human interface for robotic telemanipulation," dans *2009 IEEE International Workshop on Haptic Audio visual Environments and Games*. IEEE, Nov. 2009. doi : 10.1109/have.2009.5356117 p. 164–169.

[40] BarrettHand – Multi-fingered programmable grasper. Barrett Technology, LLC. [En Ligne]. Disponible sur : https://barrett.com/barretthand [Visité le : 2026-01-04]

[41] A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, et B. Katz, "Objectnet : A large-scale bias-controlled dataset for pushing the limits of object recognition models," dans *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, et R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [En

Ligne]. Disponible sur : https://proceedings.neurips.cc/paper_files/paper/2019/file/97af07a14cacba681feacf3012730892-Paper.pdf [Visité le : 2024-05-23]

[42] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, et J. Xiao, "3d shapenets : A deep representation for volumetric shapes," 2015. [En Ligne]. Disponible sur : https://arxiv.org/abs/1406.5670 [Visité le : 2024-09-08]

[43] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, et F. Yu, "ShapeNet : An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv :1512.03012 [cs.GR], 2015.

[44] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, et A. M. Dollar, "The ycb object and model set : Towards common benchmarks for manipulation research," dans *2015 International Conference on Advanced Robotics (ICAR)*, 2015. doi : 10.1109/ICAR.2015.7251504 pp. 510–517.

[45] A. Singh, J. Sha, K. S. Narayan, T. Achim, et P. Abbeel, "Bigbird : A large-scale 3d database of object instances," dans *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014. doi : 10.1109/ICRA.2014.6906903 pp. 509–516.

[46] Poly Haven Models. Poly Haven. [En Ligne]. Disponible sur : https://polyhaven.com/models [Visité le : 2025-11-10]

[47] L. Downs, A. Francis, N. Koenig, B. Kinman, R. Hickman, K. Reymann, T. B. McHugh, et V. Vanhoucke, "Google scanned objects : A high-quality dataset of 3d scanned household items," 2022. [En Ligne]. Disponible sur : https://arxiv.org/abs/2204.11918 [Visité le : 2025-11-10]

[48] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2024. [En Ligne]. Disponible sur : http://www.blender.org [Visité le : 2024-05-23]

[49] S. Wang, M. Lambeta, P.-W. Chou, et R. Calandra, "Tacto : A fast, flexible, and open-source simulator for high-resolution vision-based tactile sensors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, p. 3930–3937, Avr. 2022. doi : 10.1109/lra.2022.3146945. ISSN 2377-3774. [En Ligne]. Disponible sur : http://dx.doi.org/10.1109/LRA.2022.3146945 [Visité le : 2025-11-10]

[50] J. Collins, S. Goel, K. Deng, A. Luthra, L. Xu, E. Gundogdu, X. Zhang, T. F. Y. Vicente, T. Dideriksen, H. Arora, M. Guillaumin, et J. Malik, "Abo : Dataset and benchmarks for real-world 3d object understanding," dans *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. doi : 10.1109/CVPR52688.2022.02045 pp. 21 094–21 104.

[51] C. Reiser, S. Peng, Y. Liao, et A. Geiger, "Kilonerf : Speeding up neural radiance fields with thousands of tiny mlps," 2021. [En Ligne]. Disponible sur : https://arxiv.org/abs/2103.13744 [Visité le : 2024-09-08]

[52] Z. Si et W. Yuan, “Taxim : An example-based simulation model for gelsight tactile sensors,” 2021. [En Ligne]. Disponible sur : https://arxiv.org/abs/2109.04027 [Visité le : 2025-09-30]

[53] IBM. (2021, Août) Présentation générale de Crisp-DM. IBM. [En Ligne]. Disponible sur : https://www.ibm.com/docs/fr/spss-modeler/saas?topic=dm-crisp-help-overview [Visité le : 2025-12-20]

[54] R. Gao. (2022) ObjectFolder [GitHub Repo]. GitHub. [En Ligne]. Disponible sur : https://github.com/rhgao/ObjectFolder [Visité le : 2024-10-21]

[55] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, et A. M. Dollar, “Yale-cmu-berkeley dataset for robotic manipulation research,” *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017. doi : 10.1177/0278364917700714. [En Ligne]. Disponible sur : https://doi.org/10.1177/0278364917700714 [Visité le : 2025-11-10]

[56] Blender Authors. (2025) Blender 5.0 Python API Documentation. Blender. [En Ligne]. Disponible sur : https://docs.blender.org/api/current/ [Visité le : 2025-12-26]

[57] Mar. 2026. [En Ligne]. Disponible sur : https://docs.python.org/3/library/random.html#random.sample [Visité le : 2026-03-12]

[58] M. Abadi *et al.*, “TensorFlow : Large-scale machine learning on heterogeneous systems,” 2015. [En Ligne]. Disponible sur : https://www.tensorflow.org/ [Visité le : 2026-02-19]

[59] AndrzejO, Sep. 2022. [En Ligne]. Disponible sur : https://stackoverflow.com/a/73751638 [Visité le : 2026-02-20]

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, et E. Duchesnay, “Scikit-learn : Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[61] B. McFee *et al.*, “librosa/librosa : 0.11.0 (0.11.0),” 2025. doi : 10.5281/ZENODO.15006942

[62] A. Weiric. (2024, Juil.) Finetuning tensorflow/keras networks : Basics using mobilenetv2 as an example. Medium. [En Ligne]. Disponible sur : https://medium.com/@alfred.weirich/finetuning-tensorflow-keras-networks-basics-using-mobilenetv2-as-an-example-8274859dc232 [Visité le : 2026-02-19]

[63] Z. K. Abdul et A. K. Al-Talabani, “Mel frequency cepstral coefficient and its applications : A review,” *IEEE Access*, vol. 10, pp. 122 136–122 158, 2022. doi : 10.1109/ACCESS.2022.3223444

[64] Avr. 2024. [En Ligne]. Disponible sur : https://www.tensorflow.org/api_docs/python/tf/data/Dataset [Visité le : 2026-02-20]

[65] D. P. Kingma et J. Ba, “Adam : A method for stochastic optimization,” Jan. 2017. doi : 10.48550/arXiv.1412.6980

[66] Juin 2024. [En Ligne]. Disponible sur : https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy [Visité le : 2026-02-20]

[67] B. Soni. (2023, Mai) Stacking to improve model performance : A comprehensive guide on ensemble learning in python. Medium. [En Ligne]. Disponible sur : https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28 [Visité le : 2026-05-24]