Chapitre 3

Contribution

Après avoir étudié l'état de l'art des différents outils du Machine Learning et Deep Learning, nous allons présenter dans cette partie une étude comparative entre les différents algorithmes utilisés pour la classification d'images, qui est la tâche d'attribuer à une image d'entrée X un label Y à partir d'un ensemble fixe de catégories. C'est l'un des problèmes fondamentaux de la vision par ordinateur qui a une grande variété d'applications pratiques.

3.1 Problématique étudiée

Nous allons nous intéresser à la classification des alphabets arabes manuscrits, c'est une classification multi-classe (nombre d'alphabets arabe : 28). Nous avons choisi de travailler avec la base de données AHCD1, cette base se compose de 16800 images grayscale de 32 pixels en largeur et hauteur. Chaque image est étiquetée par un nombre qui représente l'alphabet correspondant (0=alif, 1=bae, ...). Ces 16800 images sont divisées en un ensemble d'apprentissage de 13440 images et un ensemble de test de 3360 images.

FIGURE 3.1 – Les 28 classes de notre jeu de données

3.2 Software

3.2.1 Tensorflow

TensorFlow est un outil open source d'apprentissage automatique développé par Google Brain pour mener des recherches sur le Machine Learning et le Deep Learning. Ses avantages sont qu'il est supporté par Google c'est-à-dire qu'il y aura constamment des mises à jour, aussi qu'il fonctionne sur plusieurs processeurs ou GPU (multi-GPU) et même sur un système d'exploitation mobile. En plus il dispose d'une grande communauté d'utilisateurs, en fait Tensorflow attire la plus grande popularité sur GitHub par rapport aux autres Framework du deep learning.

3.2.2 Keras

Keras est une API de réseaux neuronaux de haut niveau, écrite en Python et capable de s'exécuter sur TensorFlow, CNTK ou Theano. Il est écrit et entretenu par Francis Chollet, un autre membre de l'équipe Google Brain. Il supporte les réseaux convolutionels et les réseaux récurrents, ainsi que les combinaisons des deux.

3.2.3 Scikitlearn

Scikit-learn est une bibliothèque d'apprentissage statistique en Python, il a été développé par David Cournapeau en 2007. C'est le moteur de beaucoup d'applications de machine learning, sa qualité est que ses algorithmes, ses interfaces, et sa documentation, sont universellement reconnus c'est pour cela qu'il dispose aussi d'une grande communauté d'utilisateurs.

3.3 Hardware

Le Machine Learning est un domaine avec des exigences en calculs intenses et la disponibilité des ressources (surtout en GPU) dédiées à cette tache vont fondamentalement influencer sur l'expérience de l'utilisateur car sans ses ressources, il faudra trop de temps pour apprendre de ses erreurs ce qui peut être décourageant. Les expérimentations ont toutes été effectuées sur l'outil Colaboratory fournit par GOOGLE, C'est un environnement de Notebook Jupyter gratuit qui ne nécessite aucune configuration et fonctionne entièrement dans le cloud.
Le cloud (ou cloud computing) est une technologie qui permet de stocker des données ou des logiciels qui sont généralement stockés sur l'ordinateur d'un utilisateur sur des serveurs distants. Cette virtualisation des ressources permet donc d'accéder aux données et ressources sans avoir à gérer une infrastructure informatique, souvent complexe et qui représente un certain coût.

3.4 Le modèle KNN

Nous allons commencer par l'algorithme le plus simple : KNN. La AHCD1 nous offre 13440 images pour l'entrainement et 3360 images pour le test, chaque image est représentée par une matrice (32,32) donc la forme des données est un tenseur 3D par contre le KNN ne peut travailler qu'avec des tenseurs 2D donc nous devons représenter l'image sous forme d'un vecteur d'où l'entrée du modèle aura la forme (1,1024). Pour le calcul de distance nous avons choisi la méthode euclidienne, et pour le choix du nombre des voisins K nous avons testé plusieurs valeurs pour choisir celle qui donne la meilleure précision, les résultats étaient :

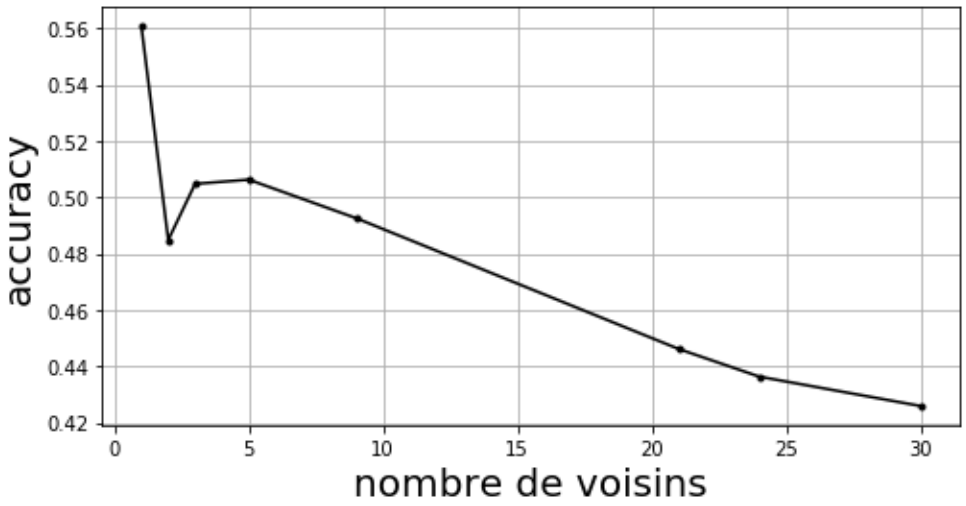

FIGURE 3.2 – La précision de l'algorithme en fonction de nombre de voisins

Malheureusement, un simple KNN ne peut pas classifier les alphabets arabes, la précision maximale qu'il peut atteindre est 56% pour un nombre de voisins égale à 1. Pour augmenter la précision nous avons pensé à utiliser la méthode de réduction des dimensions PCA. Le graphe suivant illustre la variation de la précision du modèle en fonction de nombre de composants PCA :

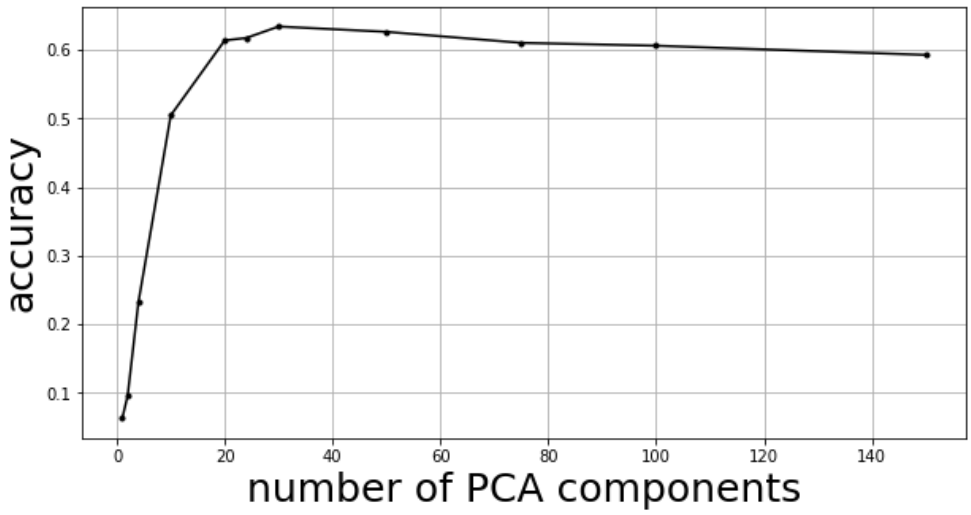

FIGURE 3.3 – La précision de l'algorithme en fonction de nombre de composants de PCA pour un K=5

Ce graphe nous montre que l'intervalle [20,75] et celui qui représente la majorité de caractéristiques. Nous constatons que la précision atteint sa limite 63,39% pour un nombre de composants égale à 30. Le PCA a augmenté la précision mais nous n'avons pas encore atteint un résultat satisfaisant. La transformation des images matricielles en un vecteur nous a fait perdre une information importante c'est le voisinage des pixels. Cette information nous aide à extraire des caractéristiques pertinentes comme les bords, les contours.... Pour bénéficier du voisinage des pixels nous avons utilisé l'histogramme de gradient orienté. Cette méthode prend comme paramètre une matrice et retourne un vecteur contenant les fréquences des orientations du gradient. Le tableau ci-dessous montre les différents résultats que nous avons obtenu en changeant les paramètres du HOG et en cherchant le meilleur nombre de voisins :

Pixels par cellule	Cellules par bloc	Orientation	Meilleure K	Précision
(8,8)	(2,2)	9	5	63.27%
(8,8)	(2,2)	6	9	62.02%
(8,8)	(2,2)	4	9	58.72%
(4,4)	(2,2)	9	5	63.27%
(4,4)	(2,2)	6	9	70,80%
(4,4)	(2,2)	4	5	72.05%

3.5 Le modèle random forest

3.5.1 Out-Of-Bag (OOB)

Avant de passer à l'expérience des Random Forest nous expliquerons une méthode pour mesurer l'erreur de prédiction des forêts aléatoires (généralement, toutes les méthodes qui utilisent la technique Bootstrap). Cette mesure s'appelle Out-Of-Bag. [3]
Pour la description du calcul du score OOB, supposons qu'on a cinq DT (Decision Tree) dans l'ensemble du Random Forest étiquetés de 1 à 5. Pour simplifier, supposons que nous disposons d'un ensemble de données d'entraînement simple comme ci-dessous.

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

FIGURE 3.4 – Exemple d'un ensemble de données d'entraînement

Soit le premier échantillon de Bootstrap composé des trois premières lignes de cet

ensemble de données. Cet exemple de Bootstrap sera utilisé comme données d'entraînement pour le DT «1». Ensuite, la dernière ligne qui est « laissée à côté » dans les données d'origine est appelée échantillon Out-Of-Bag. Cette ligne ne sera pas utilisée comme données d'entraînement pour DT 1. Noter bien qu'en réalité, plusieurs de ces lignes seront laissées hors du sac, ici pour plus de simplicité, une seule est affichée.

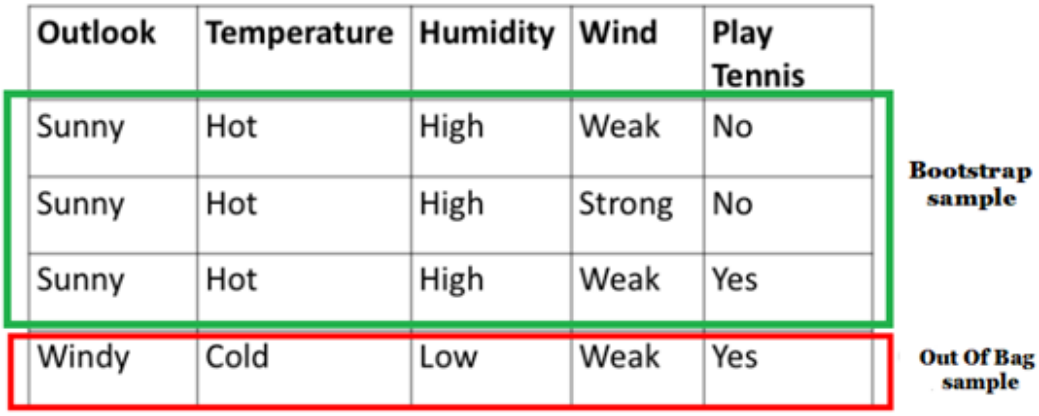

FIGURE 3.5 – Illustration de l'OOB

Une fois que le modèle sera formé, cette ligne restante ou l'échantillon OOB sera donné comme données invisibles au DT 1. Le DT 1 prédira le résultat de cette ligne. De même, cette ligne sera transmise à tous les DT qui ne contenaient pas cette ligne dans leurs données de formation Bootstrap. Le score OOB est calculé comme le nombre de lignes correctement prédites par les DT à partir de l'échantillon OOB.

Puisque nous avons compris comment le score OOB est estimé, essayons de comprendre en quoi il diffère du score des données de validation. Le score OOB est calculé sur des données qui n'ont pas nécessairement été utilisées dans l'apprentissage du modèle. Alors que pour le score des données de validation, une partie de l'ensemble de données d'apprentissage original est en fait mise à côté avant l'apprentissage du modèle. De plus, le score OOB est calculé en utilisant uniquement un sous-ensemble de DT ne contenant pas l'échantillon OOB dans leur ensemble de données d'apprentissage Bootstrap. Alors que le score de validation est calculé en utilisant tous les DT de l'ensemble. Cependant, le jeu de données n'est parfois pas assez volumineux et, par conséquent, la réservation d'une partie pour la

validation n'est pas abordable. Donc dans les cas où nous n'avons pas un grand ensemble de données et que nous voulons tout consommer comme ensemble de données d'apprentissage, le score OOB offre un bon compromis.

3.5.2 Élagage des arbres de décision

La première stratégie utilisable pour éviter un sur-apprentissage des arbres de décision consiste à proposer des critères d'arrêt lors de la phase d'expansion. C'est le principe du pré-élagage. Lorsque le groupe est d'effectif trop faible, ou lorsque l'homogénéité d'un sous-ensemble a atteint un niveau suffisant, on considère qu'il n'est plus nécessaire de séparer l'échantillon. Un autre critère souvent rencontré dans ce cadre est l'utilisation d'un test statistique pour évaluer si la segmentation introduit un apport d'informations significatif pour la prédiction de la variable-cible.

Min impurity decrease : Un nœud sera divisé si ce fractionnement induit une diminution de l'homogénéité supérieure ou égale à une valeur donnée.

Max depth : La profondeur maximale de l'arbre.

Min samples split : Le nombre minimum d'échantillons requis pour fractionner un nœud interne.

3.5.3 La réalisation de random forest

Après l'expérience du KNN nous avons passé à l'algorithme Random Forest. Donc nous avons créé à l'aide de la bibliothèque scikit-learn un modèle de classification avec un nombre d'arbres égale à 500, pour mesurer l'homogénéité des données nous nous sommes appuyés sur le critère de Gini, et pour calculer le nombre de variables à utiliser pour entrainer chaque arbre nous avons choisis la fonction $\sqrt{x}$. le tableau suivant représente les taux de bonne classification dans les données d'apprentissage, OOB et données de validation :

Données	Score
Données d'apprentissage	100%
Out Of Bag	67.63%
Données de validation	63,48%

Il est bien clair que le modèle surapprend les particularités des images traitées. Pour remédier à ce problème nous avons utilisé les techniques suivantes :

- L'histogramme des gradients orientés HOG (feature extraction).
- Le pré-élagage des arbres de decision.

Premièrement, nous avons utilisé l'histogramme de gradient orienté avec les paramètres suivants : Pixel par cellule=(4,4) , Cellule par block=(2,2) , Nombre d'orientation=4
Les taux de bonne classification dans les données d'apprentissage, OOB et données de validation en utilisant HOG sont représentés dans le tableaux suivant :

Données	Score
Données d'apprentissage	100%
Out Of Bag	60.76%
Données de validation	75.11%

Nous remarquons qu'il y a une bonne augmentation de précision sur les données de validation. Pourtant nous souffrons encore de sur-apprentissage, donc nous avons passé à l'utilisation des paramètres d'élagages :

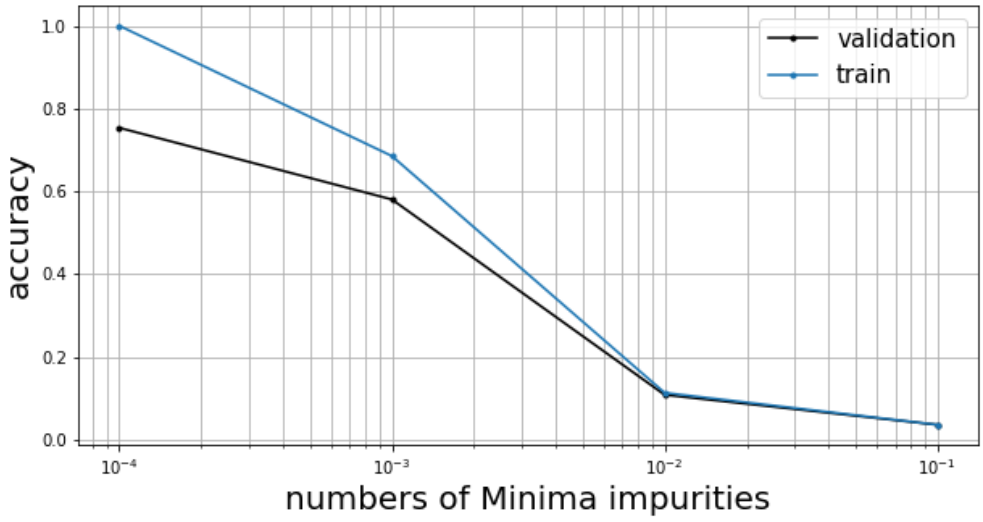

FIGURE 3.6 – La précision du modèle sur les données de validation en fonction de la valeur minimale de l'indice de Gini

L'élagage a baissé la précision du modèle sur les données d'apprentissage cela est traduit par une augmentation du biais, mais ce biais s'applique aussi sur les données de validation donc la variance du modèle reste inchangeable après l'élagage.
Pour réduire la variance du modèle nous allons se contenter par augmenter le nombre d'arbres de décision utilisé avec l'utilisation d'histogramme de gradient orienté. Avec 1500 arbres nous avons eu ces taux de bonne classification sur les differents types de données :

Données	Score
Données d'apprentissage	100%
Out Of Bag	81.76%
Données de validation	78.11%
Données de test	83,55%

La matrice de confusion representée dans la figure qui suit nous, montre qu'il y a des confusions entre plusieurs classes . Par exemple nous voyons bien que 19 (tae) qui sont représentées par la classe 2, ont été prédit comme des (thae) représentées par la classe 3, et vice versa. Aussi il y a des confusions entre les classes (7,8), (4,5,6), (13,14), (15,16) etc. Généralement, la seule différence entre ces classes est un "point" ou l'emplacement de ce "point". Pourtant les classes 0(alif), 21(kaf), 22(mim) qui ont une forme différente des autres classes ont été bien classifié. Donc c'est la forme similaire entre les classes qui provoque ces confusions.

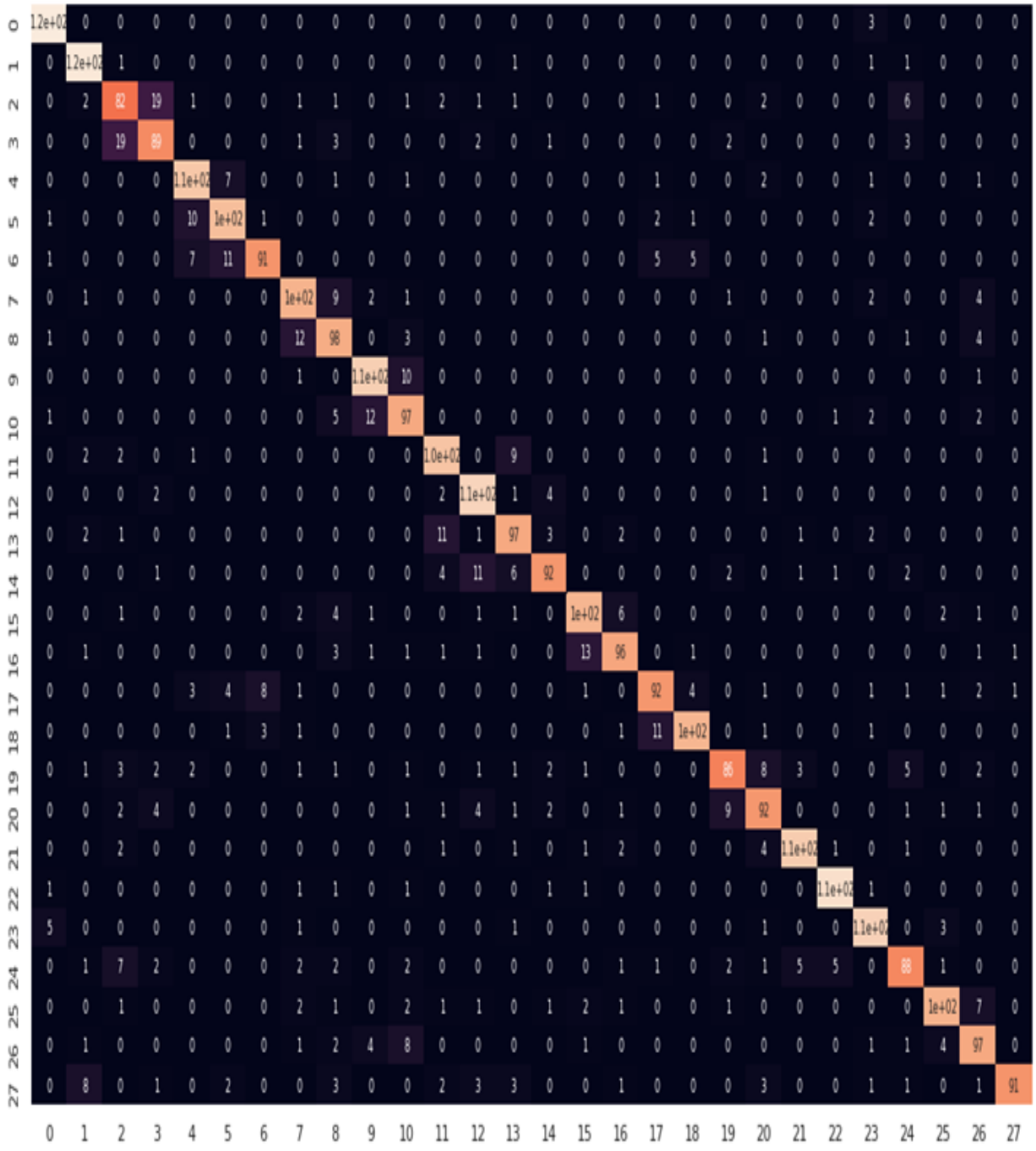

FIGURE 3.7 – La matrice de confusion pour Random forest : les colonnes représentent les classes à prédire et les lignes représentent les prédictions

3.6 Le modèle SVM

Après le Random Forest nous avons expérimenté les Support Vector machine en utilisant la méthode d'extraction des caractéristiques HOG avec les mêmes paramètres utilisés dans le random forest. Donc nous avons créé un modèle SVM avec le kernel RBF (Gaussien) et pour choisir les paramètres de régularisation C et gamma nous avons utilisé l'algorithme Grid Search.

param_grid= 'C' : [1, 10, 100, 1000], 'gamma' : [0.1, 1, 10, 100]

Le Grid Search [9] cherche la combinaison de C et gamma qui donne la meilleure précision en testant chaque combinaison possible et en utilisant la technique du K-fold cross validation et voilà les résultats :

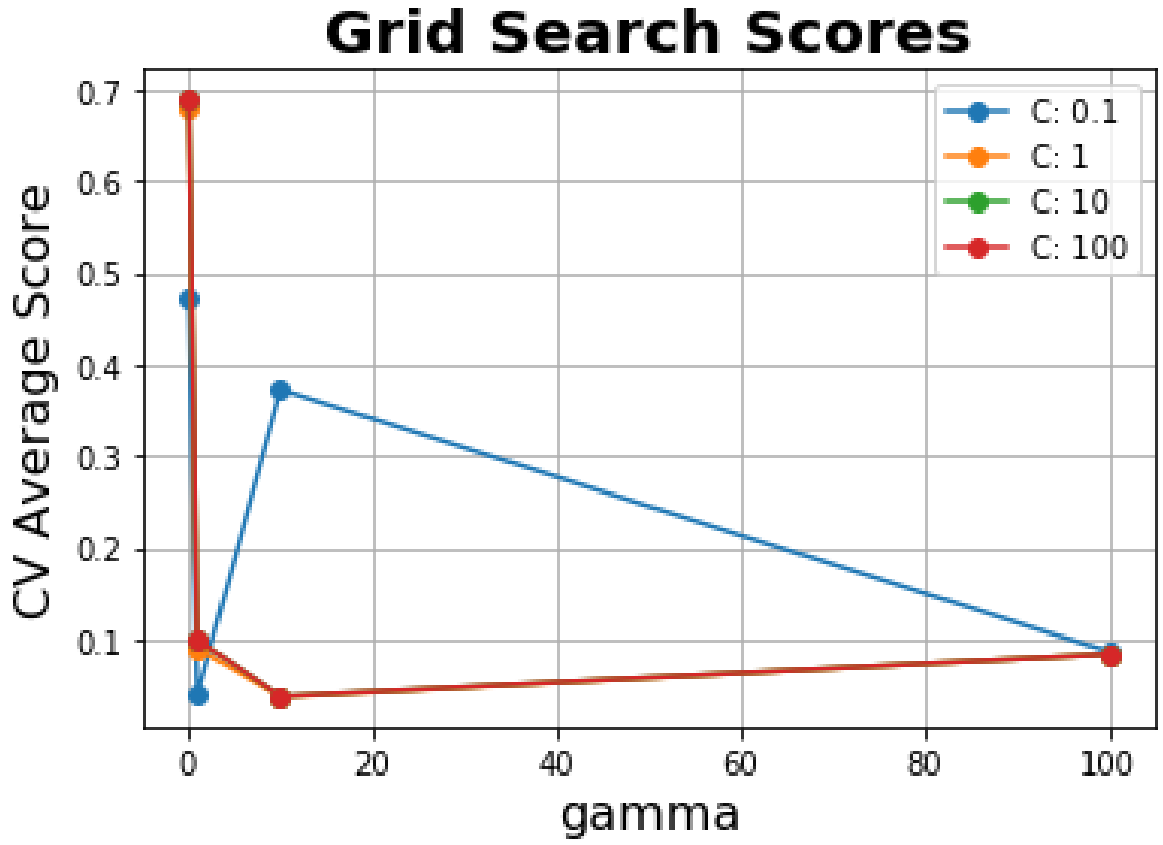

FIGURE 3.8 – La précision du SVM en fonction de gamma et C

Ce graphe montre que pour les valeurs C >= 1 la performance varie en fonction de gamma : si nous prenons un gamma autour de 0.1 la précision sera élevée, pourtant si nous prenons un gamma plus grand le modèle tend à sous-apprendre (underfitting). La meilleure combinaison est C = 10 et gamma= 0.1, elle donne une précision de 85,78% sur les données de test.

Pour une meilleure investigation des résultats nous avons utilisé la matrice de confusion pour calculer le taux de classification de chaque classe :

Classe	Précision	Classe	Précision	Classe	Précision
0 (أ)	98,36%	10(ز)	86,20%	20(ق)	75,00%
1 (ب)	93,22%	11(س)	86,20%	21(ك)	88,00%
2 (ت)	71,66%	12(ش)	88,70%	22(ل)	94,00%
3(ث)	80,67	13(ص)	85,47%	23(م)	96,00%
4(ج)	86,20%	14(ض)	75,83%	24(ن)	79,16%
5(ح)	86,95%	15(ط)	86,20%	25(ه)	90,90%
6(خ)	81,66%	16(ظ)	86,95%	26(و)	80,83%
7(د)	86,95%	17(ع)	83,33%	27(ي)	92,34%
8(ذ)	81,66%	18(غ)	90,16%		
9(ر)	89,43%	19(ف)	75,00%		

Aussi cette matrice de confusion présentée dans la figure 3.9 nous montre qu'il y a des confusions entre les classes qui ont une forme semblable. Par exemple nous voyons bien que 10,83%(13) des (tha2) sont prédits comme des (ta2) et 15,12%(18) pour l'inverse. Par rapport au Random Forest nous remarquons une diminution d'erreur de classification dans les données de test.

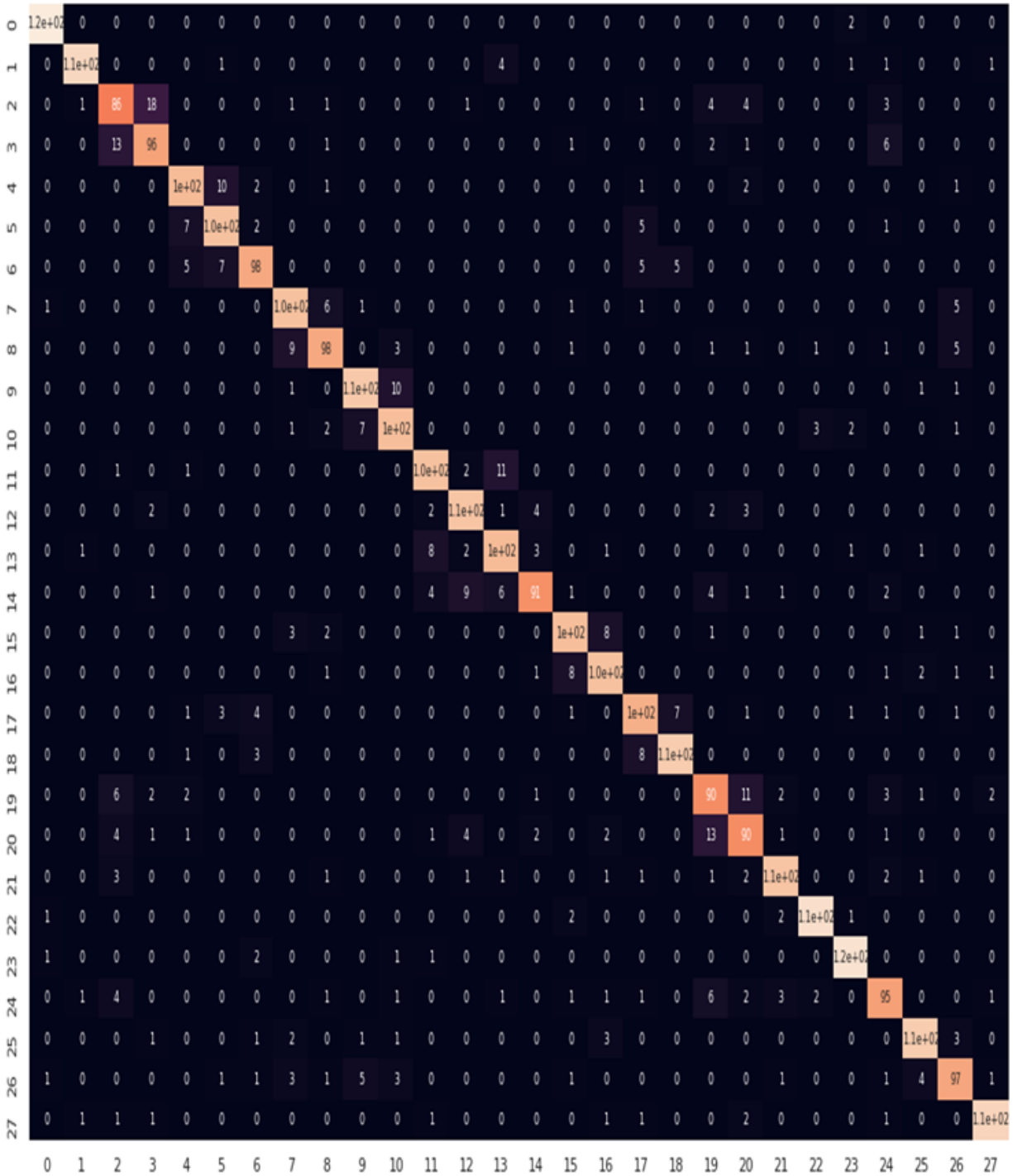

FIGURE 3.9 – La matrice de confusion pour le modéle SVM : les colonnes représentent les classes à prédire et les lignes représentent les prédictions

3.7 Le modèle CNN

3.7.1 Architecture

Maintenant nous examinerons l'algorithme le plus performant dans le domaine de vision par ordinateur : les réseaux de neurones artificiels. Nous allons commencer par entraîner un réseau de neurones convolutif des plus classiques. Le réseau possède 1,203,708 paramètres entraînables et il consiste en 4 couches convolutives, une couche de Max pooling est placé après chaque série de 2 couches convolutives et enfin 3 couches denses. Pour un apprentissage plus rapide, ReLu sera la fonction d'activation.

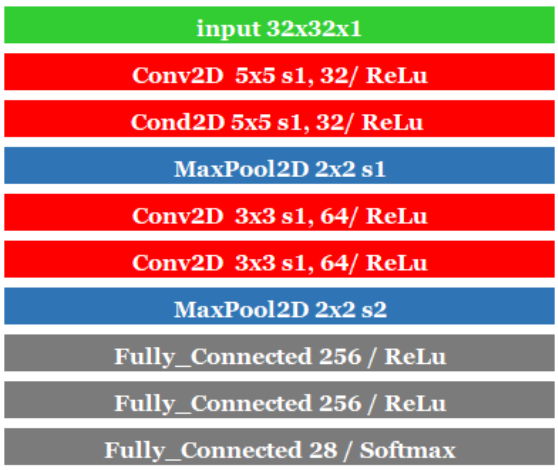

FIGURE 3.10 – L'architecture du 1èr modèle

3.7.2 Discussion

Afin de choisir un algorithme d'apprentissage parmi (SGD, Adagrad, Adam, RMSprop) nous allons expérimenter chacun d'eux avec le même modèle décrit par la figure 3.10. Le nombre d'époques sera le même pour les 6 algorithmes. Le taux d'apprentissage est fixé à 10^{-3}. Les hyperparamètres de ADAM et RMSprop sont initialisés par leurs valeurs par défaut. La taille du batch est de 100. La fonction de coût est Categorical_Cross_Entropy :

$$\mathbf{H}(p,q) = -\sum_x p\,(x)\,log\,(q\,(x))$$

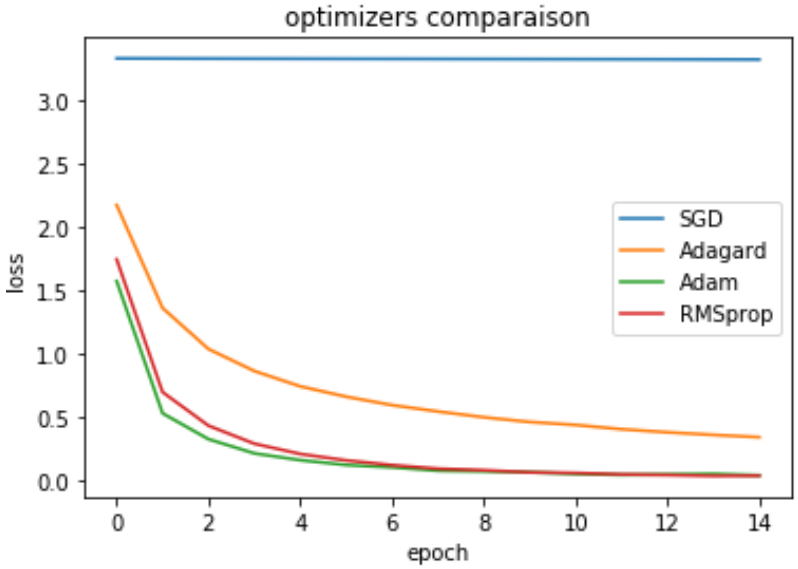

FIGURE 3.11 – Comparaison de différents optimiseurs en fonction du taux d'erreur dans 15 époques

Les deux algorithmes RMSprop et Adam progressent rapidement et minimisent l'erreur de l'apprentissage dès le début du processus. le tableau suivant représente le taux d'erreur après 15 époques pour chaque optimiseur :

Optimiseurs	Erreur
SGD	3.31
Adagrad	0.33
Adam	0.03
RMSprop	0.03

Toutes les prochaines expérimentations se feront avec RMSprop comme algorithme d'apprentissage. Maintenant nous allons voir la courbe d'erreur de la phase de validation et de la phase d'apprentissage :

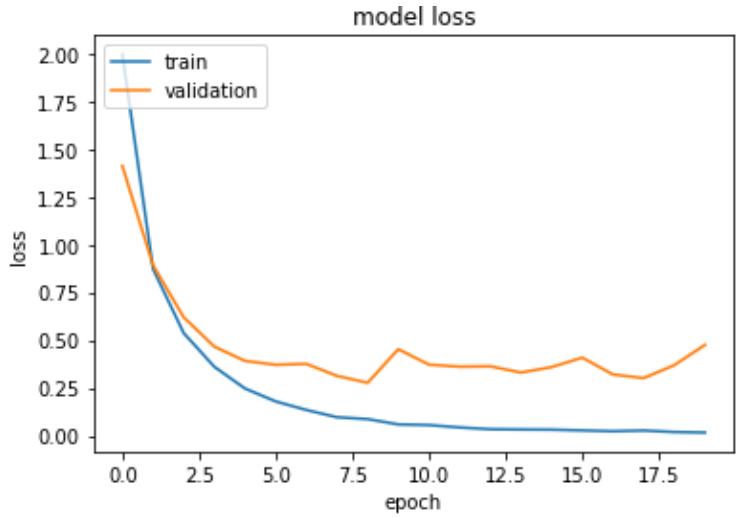

FIGURE 3.12 – La courbe d'erreur de validation vs la courbe d'erreur d'apprentissage

L'erreur sur les données de validation est bien minimisée au début et atteint sa plus faible valeur après 8 époques, puis elle commence à se diverger alors qu'au même moment l'erreur de l'apprentissage continue de diminuer. C'est le sur-apprentissage. Le tableau suivant représente l'erreur et la précision des données de validation :

Erreur	Précision
0.34	0.93

3.7.3 Data Augmentation

Le sur-apprentissage est dû au fait d'avoir peu de données d'apprentissage, ce qui empêche le modèle de se généraliser à des nouvelles données. Si nous disposons des données infinies, notre modèle ne tombera jamais dans le sur-apprentissage. L'augmentation des données est une approche qui consiste à générer plus de données d'apprentissage à partir des données d'apprentissage existantes, en augmentant ces dernieres via un certain nombre de transformations aléatoires qui produisent des nouvelles images crédibles. L'objectif est qu'au moment de l'apprentissage, notre modèle ne voit jamais deux fois la même image exacte.

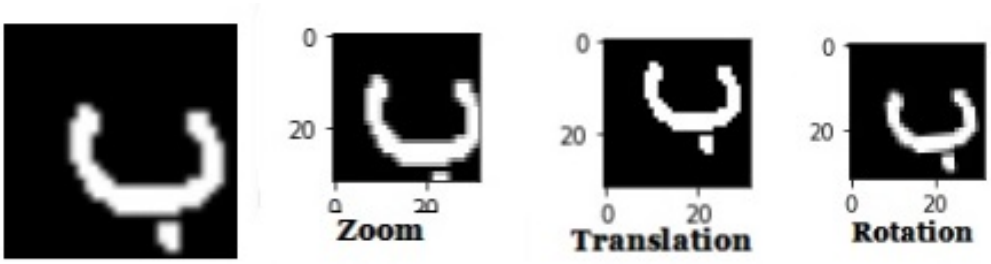

FIGURE 3.13 – Exemple du data augmentation

En utilisant cette technique nous avons diminué l'erreur sur les données de validation. La figure suivante represente le taux d'erreur sur les données d'apprentissage et sur les données de validation après 60 époques :

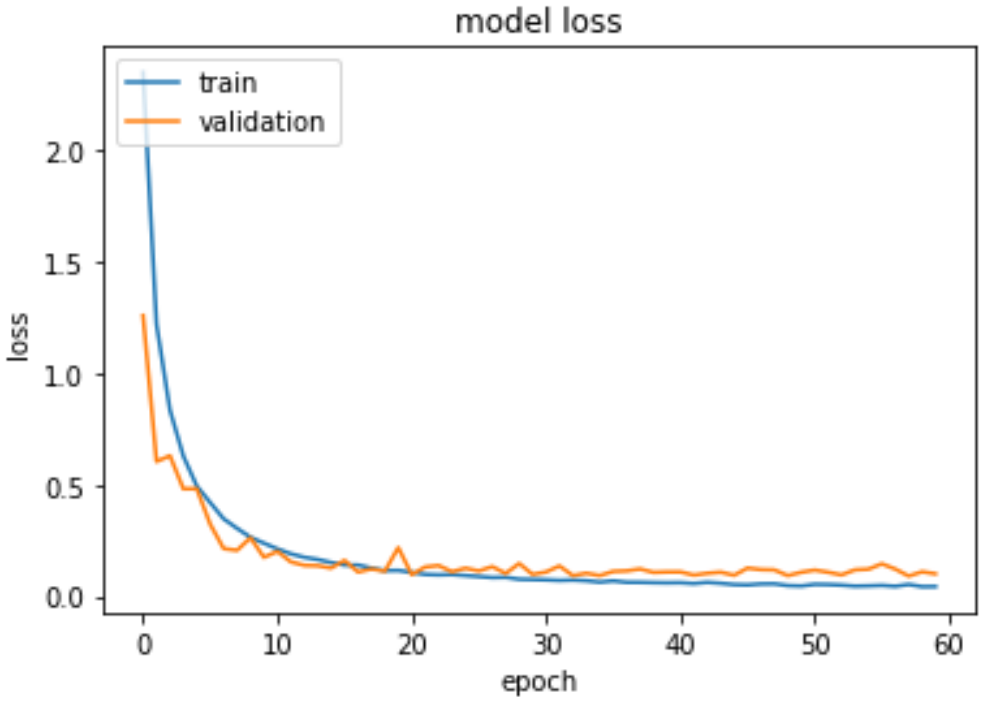

FIGURE 3.14 – La courbe d'erreur de la phase de validation et de la phase d'apprentissage aprés utilisation du data augmentation

Erreur	Précision
0.17	0.97

Nous remarquons que l'erreur de validation est bien minimisée après l'augmentation des données et aussi la précision a été augmentée, mais nous observons encore un sur-apprentissage. La prochaine méthode va éliminer définitivement le problème de sur-apprentissage en même temps elle va encore baisser le taux d'erreur sur les données de validation.

3.7.4 La régularisation

Le mot régulariser signifie rendre les choses régulières ou acceptables. C'est exactement pourquoi nous l'utilisons. Les régularisations sont des techniques utilisées pour réduire l'erreur de test et pour éviter le sur-apprentissage. Nous allons nous intéresser à une technique particulièrement puissante : le **Dropout**.

3.7.4.1 Dropout

Le Dropout est une technique où des neurones sélectionnés au hasard sont ignorés (temporairement) pendant l'apprentissage. Cela signifie que leur contribution à l'activation des neurones qui leur succèdent est temporairement supprimée lors de la phase de propagation et toutes les mises à jour de poids ne sont pas appliquées au neurone lors de la phase de rétro-propagation. Lorsque des neurones sont supprimés au hasard du réseau pendant l'apprentissage, les autres neurones devront intervenir et gérer la représentation requise pour faire des prédictions pour les neurones manquants. Lors de la phase d'apprentissage, pour chaque itération, un neurone est gardé avec une probabilité p, sinon il est supprimé. Lors de la phase de test, tous les neurones sont gardés, nous voulons donc que les sorties des neurones au moment du test soient identiques à leurs sorties au moment de l'apprentissage. Par exemple, dans le cas ou $p = 0.5$, les neurones doivent réduire de moitié leurs sorties au moment du test pour avoir la même sortie que pendant l'apprentissage. [4]

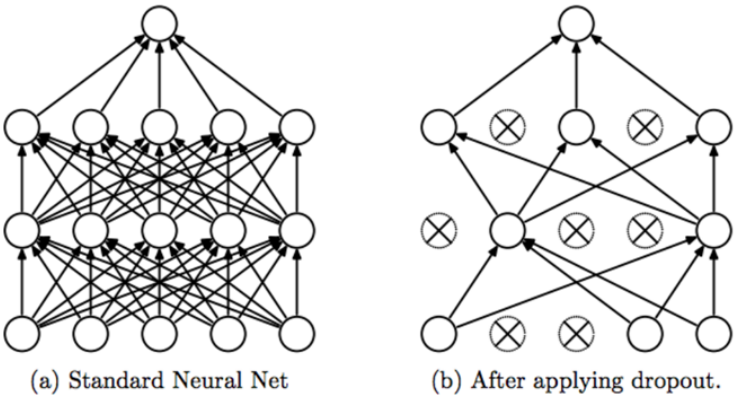

FIGURE 3.15 – Un réseau de neurones avant et après l'application du dropout

Le dropout force le réseau de neurones à apprendre des features plus robustes qui sont utiles en conjonction avec de nombreux sous-ensembles aléatoires différents des autres neurones.

Maintenant que nous savons comment fonctionne le dropout, nous allons l'introduire dans notre réseau et voir comment il impacte les performances.

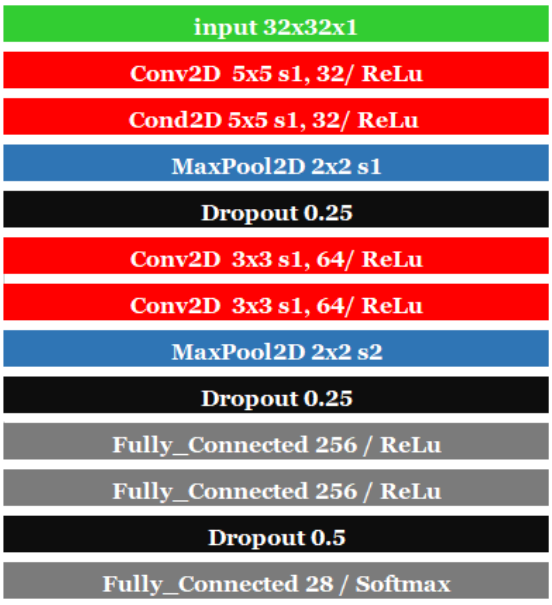

FIGURE 3.16 – L'architecture du 2ème modèle

les 2 figures suivantes montrent l'erreur et la précision du 2ème modèle après 15 époques :

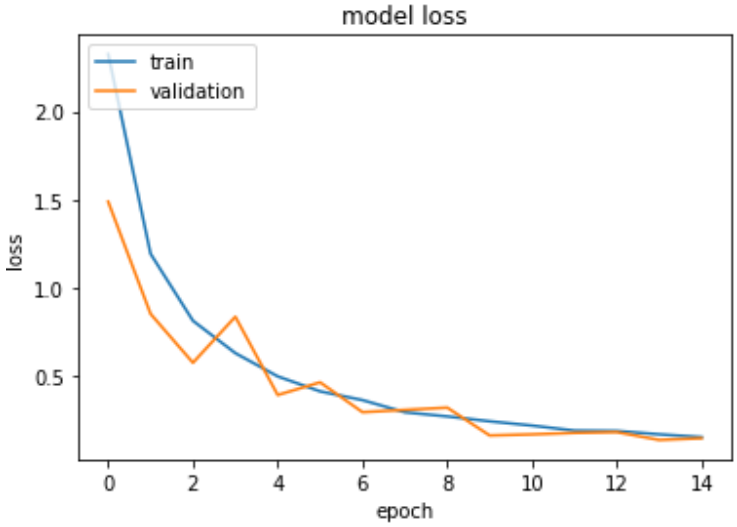

FIGURE 3.17 – L'erreur de validation et d'entraînement aprés utilisation du dropout

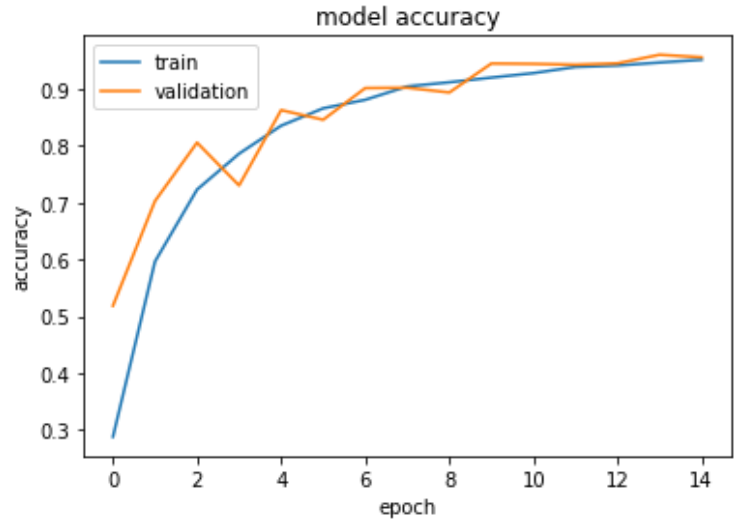

FIGURE 3.18 – La précision de validation et d'entraînement aprés utilisation du dropout

À partir des deux figures on voit tout de suite que le modèle avec dropout élimine totalement le sur-apprentissage. Avec le modèle 2 la précision atteint 94% après 10 époques. Cette précision continue d'augmenter doucement jusqu'à atteindre 95% après 15 époques. mais voyons l'impacte du dropout sur l'erreur d'apprentissage :

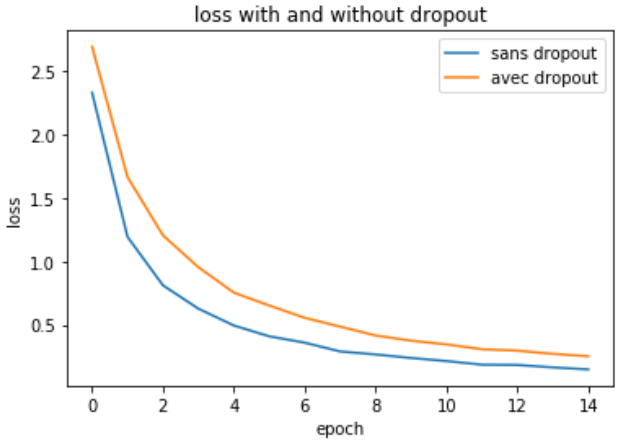

FIGURE 3.19 – La courbe d'erreur d'apprentissage avec et sans dropout

Le fait de supprimer aléatoirement et temporairement des neurones, ralentit considérablement l'apprentissage et le modèle 2 n'arrive pas à minimiser l'erreur d'apprentissage aussi bien que le faisait le modèle 1. Pour cela, on a plusieurs solutions :

- Laisser durer l'apprentissage.
- Augmenter les probabilités du dropout : Cette stratégie n'est pas sûre car le fait d'augmenter les probabilités de garder les neurones pourrait ne pas protéger aussi efficacement du sur-apprentissage.

Maintenant nous avons obtenu notre modèle final. Puisque nous comptons tirer le maximum de notre réseau, nous prévoyons de laisser durer l'apprentissage très longtemps pour prévoir le nombre d'époques où le modèle stagne. Après 30 époques le modèle atteint l'erreur minimale 9% et la précision maximale 96% sur les données d'apprentissage, sur les données de test nous avons obtenu une erreur de 8% et une précision de 98%. Pour une meilleure investigation des résultats nous presenterons le taux de bonne classification de chaque classe que nous avons calculée à partir de la matrice de confusion montrée dans la figure 3.20 :

Classe	Précision	Classe	Précision	Classe	Précision
0 (أ)	99,17%	10(ز)	95,23%	20(ق)	96,77%
1 (ب)	98,36%	11(س)	100%	21(ك)	99,17%
2 (ت)	97,56%	12(ش)	100%	22(ل)	100%
3(ث)	97,56%	13(ص)	98,36%	23(م)	98,36%
4(ج)	99,17%	14(ض)	96,77%	24(ن)	97,56%
5(ح)	99,17%	15(ط)	100%	25(ه)	97 ,56%
6(خ)	98,36%	16(ظ)	96,77%	26(و)	96%
7(د)	96,00%	17(ع)	98,36%	27(ي)	100%
8(ذ)	96,77%	18(غ)	100%		
9(ر)	100%	19(ف)	96,77%		

Comme vous pouvez voir dans ce tableau et dans la matrice de confusion le CNN classifie presque parfaitement les 28 classes du AHCD1.

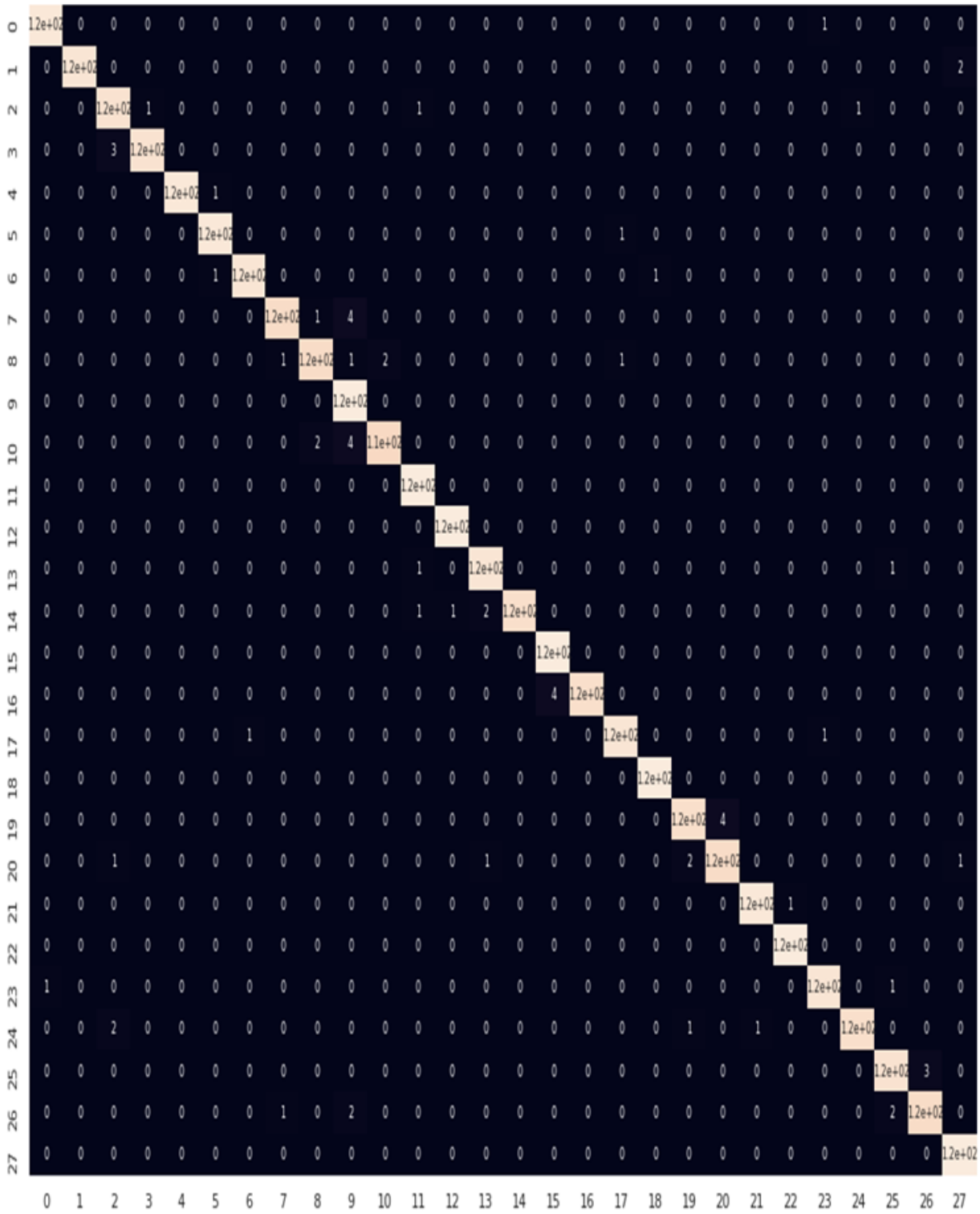

FIGURE 3.20 – La matrice de confusion pour CNN : les colonnes représentent les classes à prédire et les lignes représentent les prédictions

3.8 Conclusion

Voici les meilleures précisions que nous avons obtenues en testant chaque modèle sur les données de test :

Modèle	Précision
K-Near Neighbors	72%
Random Forest	83%
Support Vector machine	85%
Convolutional Neural Network	98%

Dans cette partie, nous avons décrit les démarches suivies afin d'obtenir le classifieur le plus précis possible. Nous avons expérimenté plusieurs classifieurs en observant le comportement de chacun grâce aux graphes et nous avons fait changer les paramètres un par un afin de comprendre l'impact de chacun sur la performance finale.

3.9 Interface

Nous avons tenu à introduire les deux modèles : réseau de neurones(CNN) et SVM dans une interface graphique afin de pouvoir tester leurs performances facilement sans être obligé de passer à chaque fois par les lignes de codes. Les détails de l'interface sont montrés dans la figure 3.21.

1 .Le bouton pour importer l'image du caractére à classifier.

2 . l'image insérée.

3 . les 3 plus grandes probabilités calculées par CNN.

4 . La prédiction du SVM.

5 . Un bouton qui affiche les features extraites par la 1ére couche de convolution du CNN.

6 . L'histogramme des probabilités calculées par le CNN.

7 . L'image résultante par HOG.

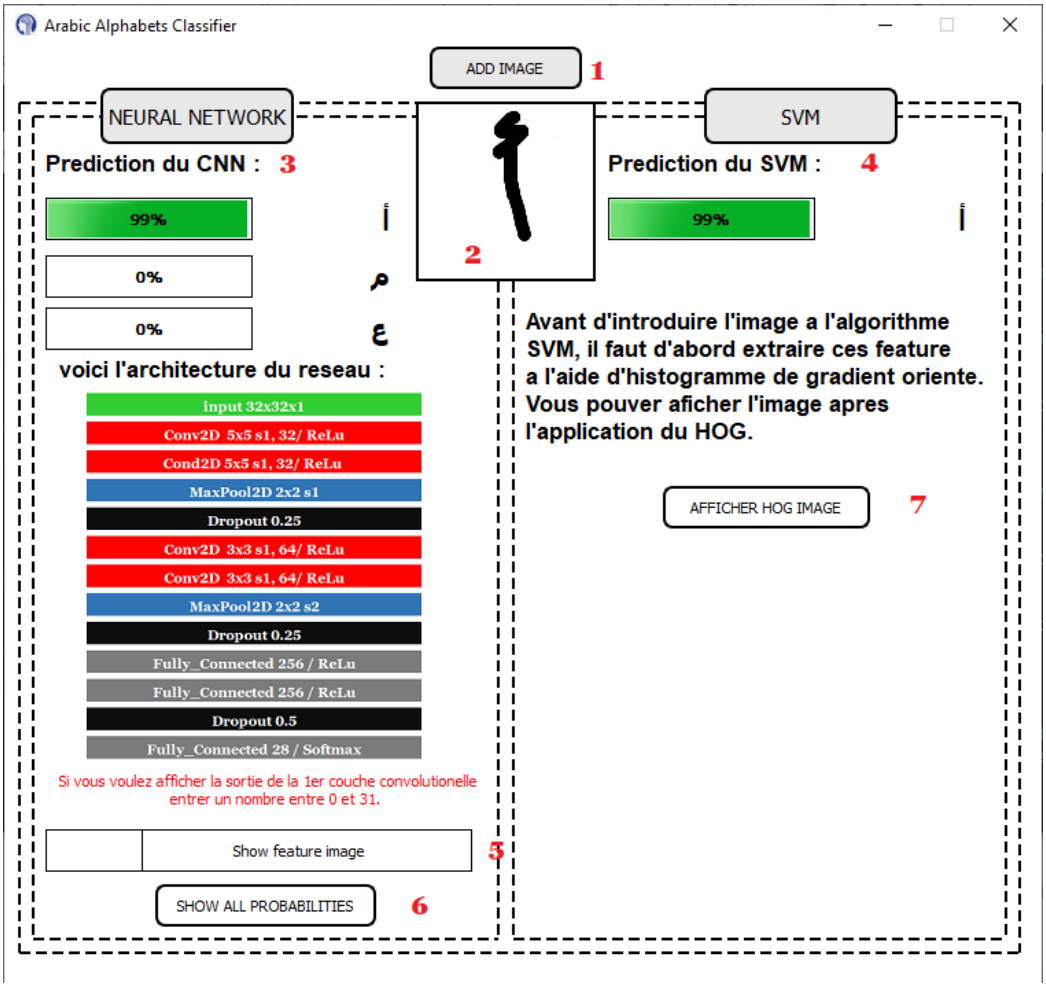

FIGURE 3.21 – Image de l'interface Graphique

Cette interface est conçue à l'aide de la bibliothèque PyQt5 et de l'outil de QT Designer.

QT est une API orientée objet développée en C++, Qt offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc...

PyQt est un module libre qui permet de lier le langage Python avec la bibliothèque Qt distribué sous deux licences : une commerciale et la GNU GPL. Il permet ainsi de créer des interfaces graphiques en Python.

QtDesigner est Une extension de QT qui permet de concevoir visuellement des interfaces graphiques et générer leurs code Python.

Conclusion générale

Ces résultats montrent que grâce au deep Learning l'avenir de l'intelligence artificielle est prometteur. Dans ce travail, nous avons exploré le domaine de la classification d'image qui est comme tous les autres domaines de l'intelligence artificielle ont connu une évolution majeure depuis l'apparition du Deep Learning. Afin d'aboutir à ces résultats, nous avons passé beaucoup de temps à lire et à étudier les publications et les articles concernant ce domaine pour voir ce qui se fait de mieux en matière de classification. Pour finir, avant de passer aux perspectives, ce travail nous a permis de découvrir le monde du machine learning et du deep learning et surtout de voir l'avenir prometteur de l'intelligence artificielle et de ses applications dans le monde réel. Comme perspective nous pouvons travailler sur l'implémentation d'un système de reconnaissance d'écriture manuscrite. C'est un problème qui fait appel à la vision par ordinateur, mais également au traitement automatique du langage naturel. Cela veut dire que le système, tout comme le cerveau humain, reconnaît des mots et des phrases existants dans un langage connu plutôt qu'une succession de caractères.