

Rapport de Synthèse du projet Cassiopée

Projet n°35

Réseau de neurones pour la vérification
d'identité par des signatures manuscrites ou
«hors-ligne»

Auteurs :

M. Marouane KACHOURI M^{elle} Fatma BOUZGHAÏA

Encadrants :

Pr. Sonia GARCIA Pr. Nesma HOUMANI

Année Universitaire : 2019/2020

Table des matières

Introduction	1
1 Etat de l'art	2
1.1 Etude des articles	2
1.2 Choix du modèle	4
1.3 Les bases de données	5
2 Implémentation de notre solution	7
2.1 Architecture du modèle	7
2.2 Pré-traitement	8
2.3 Entraînement	8
3 Expériences et Résultats	11
3.1 Protocole expérimental	11
3.2 Mesures de l'évaluation des performances	12
3.3 Résultats de l'entraînement du modèle	12
3.4 Résultats obtenus	13
Conclusion	16
Bibliographie	17
A Planning du projet	18

Table des figures

1.1	Résultats des tests de l'article [1]	3
1.2	Architecture et résultats de l'article [2]	3
1.3	Architecture et technique utilisées dans l'article [3]	4
1.4	Matrice de confusion des résultats obtenus de l'article [4]	4
1.5	Echantillon d'une paire de signatures	5
1.6	Paire de signatures de la base de données GPDS300	6
1.7	Paire de signatures de la base de données CEDAR	6
2.1	Architecture de SigNet	7
2.2	Implémentation de l'architecture de notre réseau	9
2.3	Définition de notre réseau	9
2.4	Entraînement de notre réseau sur la base de données CEDAR	10
3.1	Entraînement de notre modèle	13
3.2	Courbe de ROC	14
3.3	Matrice de confusion	15

Liste des tableaux

3.1	Répartition de chaque base de données pour notre modèle	11
3.2	Performance de l'entraînement du modèle	13
3.3	Comparaison de notre modèle sur diverses bases de données	14

Liste des sigles et acronymes

AUC	<i>Area Under The Curve</i>
F.C	<i>Fully Connected</i>
FAR	<i>False Acceptance Rate</i>
FRR	<i>False Rejection Rate</i>
ReLU	<i>Rectified Linear Unit</i>
RMSprop	<i>Root Mean Square Propagation</i>
ROC	<i>Receiver Operating Characteristics</i>
SPP	<i>Spatial Pyramid Pooling</i>
TNR	<i>True Negative Rate</i>
TPR	<i>True Positive Rate</i>
VGG	<i>Visual Geometry Group</i>
WD	<i>Writer Dependent</i>
WI	<i>Writer Independent</i>

Introduction

La signature manuscrite est un trait de comportement biométrique le plus populaire et le plus utilisé afin de vérifier l'identité d'une personne, cette caractéristique est généralement vérifiée dans les banques, les documents officiels et légaux, le domaine administratif, etc.

Il existe deux types de vérification d'une signature : online et offline. La vérification d'une signature online nécessite une tablette électronique et un stylet afin que ces derniers puissent enregistrer la séquence de coordonnées de la signature. Quant à la détection de signatures manuscrites falsifiées (offline), ça revient, généralement, à scanner des signatures sur des documents, ce qui produit alors une image de la signature à deux dimensions, et c'est cette signature qui nous servira comme entrée pour notre réseau de neurones.

La vérification d'une signature manuscrite peut être traitée de deux approches différentes : writer dependent et write independent. L'approche writer independent est préférable à celle du writer dependent. En effet, dans le cas du writer dependent, le système connaît l'auteur de la signature, il lui suffit alors juste de comparer les caractéristiques de la signature en entrée et la classer en authentique ou falsifiée. Mais, avec cette approche, il faut toujours mettre à jour le système de vérification de la signature manuscrite à chaque nouvel auteur de signature. Tandis que dans l'approche writer independent, la vérification d'identité devient plus difficile puisque le système ne connaît pas l'auteur de la signature et doit donc comparer les signatures afin de les classer en authentique et falsifiée indépendamment de leur auteurs. Ceci résulte en un système évolutif capable de faire des prédictions sur les futures entrées sans besoin d'informations supplémentaires.

Notre objectif durant ce projet est d'implémenter un modèle de réseau de neurones Siamois qui permet de détecter la falsification de signatures avec l'approche writer independent, pris en exemple dans un article étudié [4] et comparer nos résultats à ceux obtenus dans ce dernier article.

Chapitre 1

Etat de l'art

1.1 Etude des articles

Avant de mettre en œuvre un plan d'action nous avons fait la lecture et l'analyse de quatre articles pour avoir une meilleure idée sur les solutions existantes et les résultats obtenus. Les articles étudiés traitent tous le même sujet à savoir « La vérification de l'authenticité des signatures manuscrites dites Hors-ligne à travers des méthodes d'apprentissage profond ». Nous présentons ci-dessous brièvement les principaux points traités dans chacun des articles.

— **Offline Signature Verification with Convolutional Neural Networks [1]** :

L'architecture déployée dans le modèle de cet article est VGG-16, une architecture qui, comme son nom l'indique, contient exactement 16 couches de poids dont 13 sont des couches de convolution et 3 sont des couches denses. Le modèle comporte également 5 couches de Max pooling (expliqué dans la section : architecture du modèle) pour un total de 21 couches. La technique de « Transfer learning » qui consiste à agir sur les dernières couches d'un réseau pré-entraîné est utilisée. La fonction de perte adoptée est la « Categorical Cross-Entropy » de formule :

$$L_{cross_entropy}(\hat{y}, y) = - \sum_i y_i \log(\hat{y}_i) \quad (1.1)$$

Avec :

- y_i : est le label de la classe correcte.
- \hat{y}_i : est le label de la classe prédite.

La base de données utilisée est SigComp 2011 qui contient des signatures pour 10 personnes chinoises et 10 personnes néerlandaises. Les résultats de test sur des signatures non existantes dans la base d'entraînement sont résumés ci-dessous :

	Dutch Result
Accuracy	76%
FAR	28.6%
FRR	21.8%

FIGURE 1.1 – Résultats des tests de l’article [1]

— **Convolutional Neural Network Based Offline Signature Verification Application [2] :**

Cet article porte sur les approches « Writer Dependent » et « Writer Independent ». Nous nous intéressons seulement à la partie « Writer Independent ». La base de données traitée dans cet article est GPDS Synthetic. L’architecture du modèle ainsi que les résultats trouvés sont présentés par la figure 1.2.

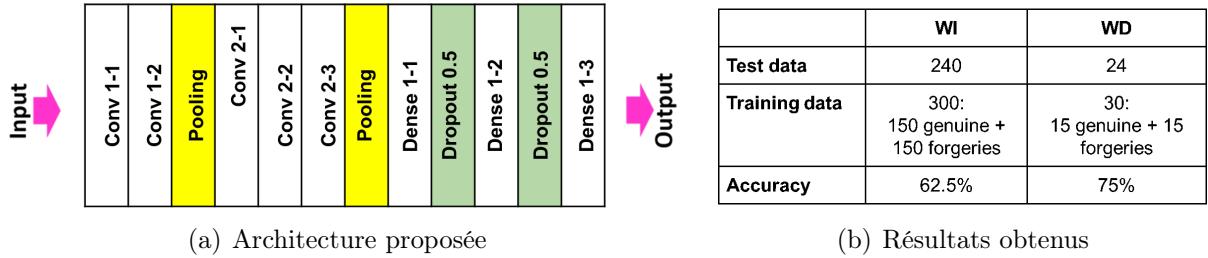


FIGURE 1.2 – Architecture et résultats de l’article [2]

— **Fixed-sized representation learning from Offline Handwritten Signatures of different sizes [3] :**

Dans cet article, on a l’introduction d’une nouvelle technique de pooling appelée Spatial Pyramid Pooling (ou SPP). Le SPP de taille $N \times N$ consiste à diviser l’image d’entrée en des sous-matrices de taille $N \times N$ et retourner la valeur maximale de chaque sous matrice. Les valeurs sont par la suite concaténées dans un vecteur de taille $M \times 1$ où M est le nombre de sous-matrices obtenu suite à la division de l’image d’entrée. Une variété de bases de données a été utilisée pour l’entraînement et le test telles que GPDS, MCYT, CEDAR etc. Parmi les meilleurs résultats, un taux d’erreur de 3% sur les signatures hors-ligne. Ci-dessous nous représentons un schéma de SPP et l’architecture utilisée.

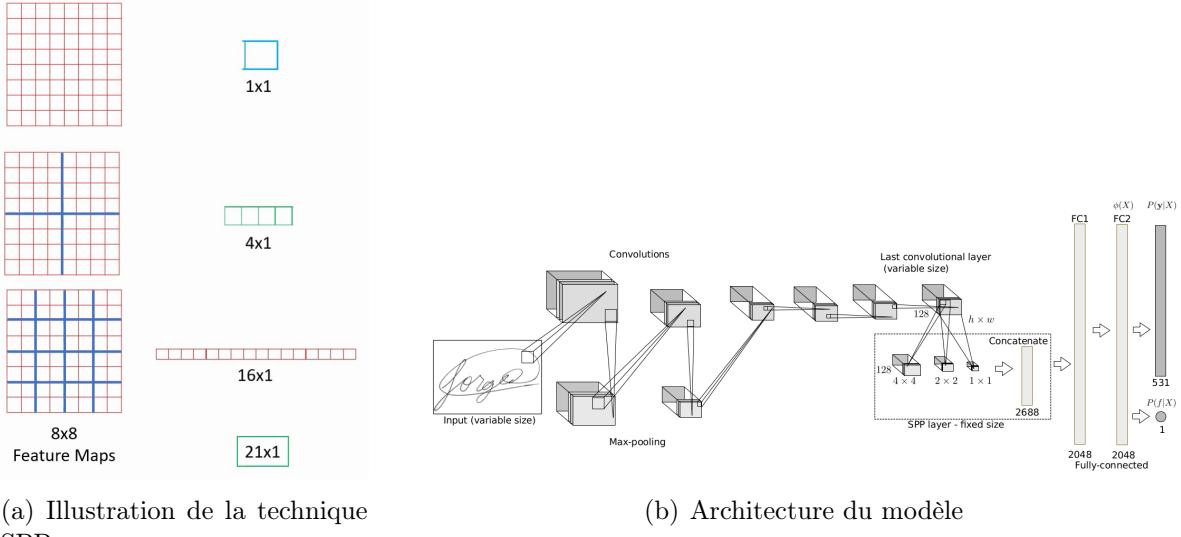


FIGURE 1.3 – Architecture et technique utilisées dans l'article [3]

— SigNet : Convolutional Siamese Network for Writer Independent Offline Signature Verification [4] :

Le contenu de cet article sera présenté dans la suite du rapport puisqu'il fait l'objet de référence pour notre projet. Nous présentons, cependant, les résultats obtenus afin d'effectuer une comparaison avec nos résultats.

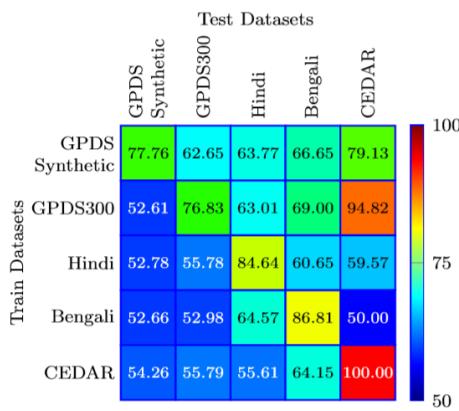


FIGURE 1.4 – Matrice de confusion des résultats obtenus de l'article [4]

1.2 Choix du modèle

Nous avons choisi de travailler avec le modèle siamois pour sa compatibilité avec notre problème. En effet, un réseau de neurones siamois (parfois appelé réseau neuronal jumeau) est un réseau de neurones artificiel qui utilise les mêmes poids tout en travaillant en tandem

sur deux vecteurs d'entrée différents pour calculer des vecteurs de sortie comparables. Souvent, l'un des vecteurs de sortie est précalculé, formant ainsi une ligne de base par rapport à laquelle l'autre vecteur de sortie est comparé. De ce fait, le réseau siamois est adéquat à ce qu'on vise de réaliser puisqu'il crée une corrélation entre les deux images d'entrées et décide sur leur différence au lieu de traiter chaque image séparément de l'autre comme le font les autres réseaux mentionnés (VGG-16, AlexNet, etc.). Le réseau siamois est donc idéal pour notre cas puisqu'on cherche à identifier les signatures imitées en fournissant des images authentiques de ces signatures.

Dans notre cas, une paire de signatures est donnée en entrée avec la première signature qui est toujours authentique et la deuxième qui est soit authentique soit falsifiée. Pour conclure sur la nature de la deuxième signature, un score de différence est calculé entre les deux vecteurs de sortie fournis par le modèle selon la formule de distance euclidienne et un seuil est déterminé par le modèle. Dans le cas où le score dépasse le seuil nous décidons que la signature est falsifiée et dans le cas contraire la signature est libellée comme authentique.

1.3 Les bases de données

Nous avons découvert plusieurs bases de données contenant des signatures authentiques et falsifiées d'un nombre variable d'auteurs. Les bases de données sont assez variées quant à l'origine des signatures (Inde, Bangladesh, Europe) et donc leur style ainsi que dans la taille des images fournies. Les bases de données traitées dans ce projet sont :

- **BHSig260** : Cette base de données contient des signatures authentiques et imitées pour 260 auteurs dont 100 sont des personnes de l'Inde (Hindi) et 160 sont des signataires du Bengaldesh (Bengali). Pour cette base de données, nous avons, pour chaque auteur, 24 signatures authentiques et 30 signatures falsifiées. De ce fait, nous nous trouvons avec $100 \times 24 = 2400$ signatures authentiques et $100 \times 30 = 3000$ signatures falsifiées pour Bengali, et $160 \times 24 = 3840$ signatures authentiques et $160 \times 30 = 4800$ signatures imitées pour Hindi. Notons que, nous avons séparé cette base de données en Hindi et Bengali pour expérimenter notre modèle.

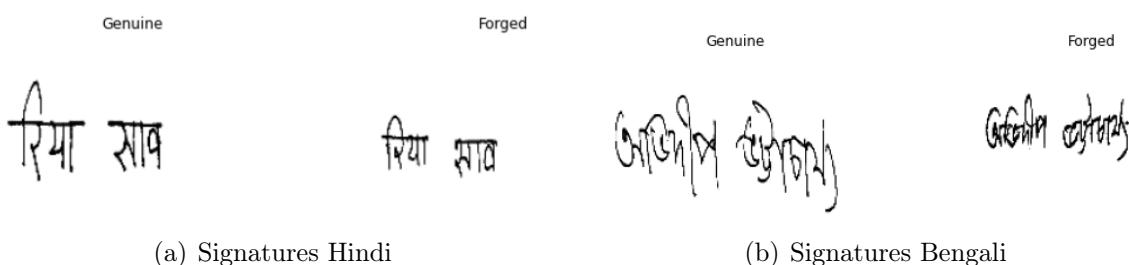


FIGURE 1.5 – Echantillon d'une paire de signatures

- **GPDS300** : Cette base de données contient des signatures européennes authentiques et imitées pour 300 auteurs. Pour chaque auteur, nous avons 24 signatures authentiques et 30 signatures imitées. En total, nous avons $300 \times 24 = 7200$ signatures originales et $300 \times 30 = 9000$ signatures falsifiées.



FIGURE 1.6 – Paire de signatures de la base de données GPDS300

- **CEDAR** : Cette base de données contient des signatures européennes authentiques et imitées pour 55 auteurs. Pour chaque auteur, nous avons 24 signatures authentiques et 24 signatures imitées. Cette base de données comprends $55 \times 24 = 1320$ signatures authentiques, de même pour les signatures falsifiées.

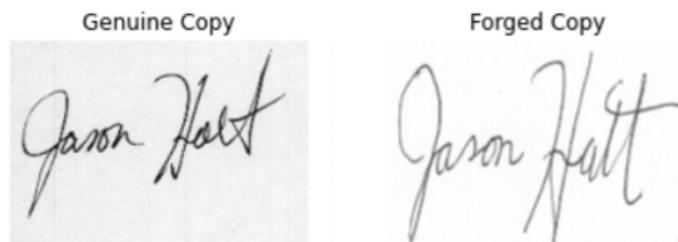


FIGURE 1.7 – Paire de signatures de la base de données CEDAR

D'autres bases de données ont été abordées lors de l'étude des articles mais que nous n'avons pas traité telles que DS-GET, GPDS Synthetic, SigComp, etc.

Chapitre 2

Implémentation de notre solution

2.1 Architecture du modèle

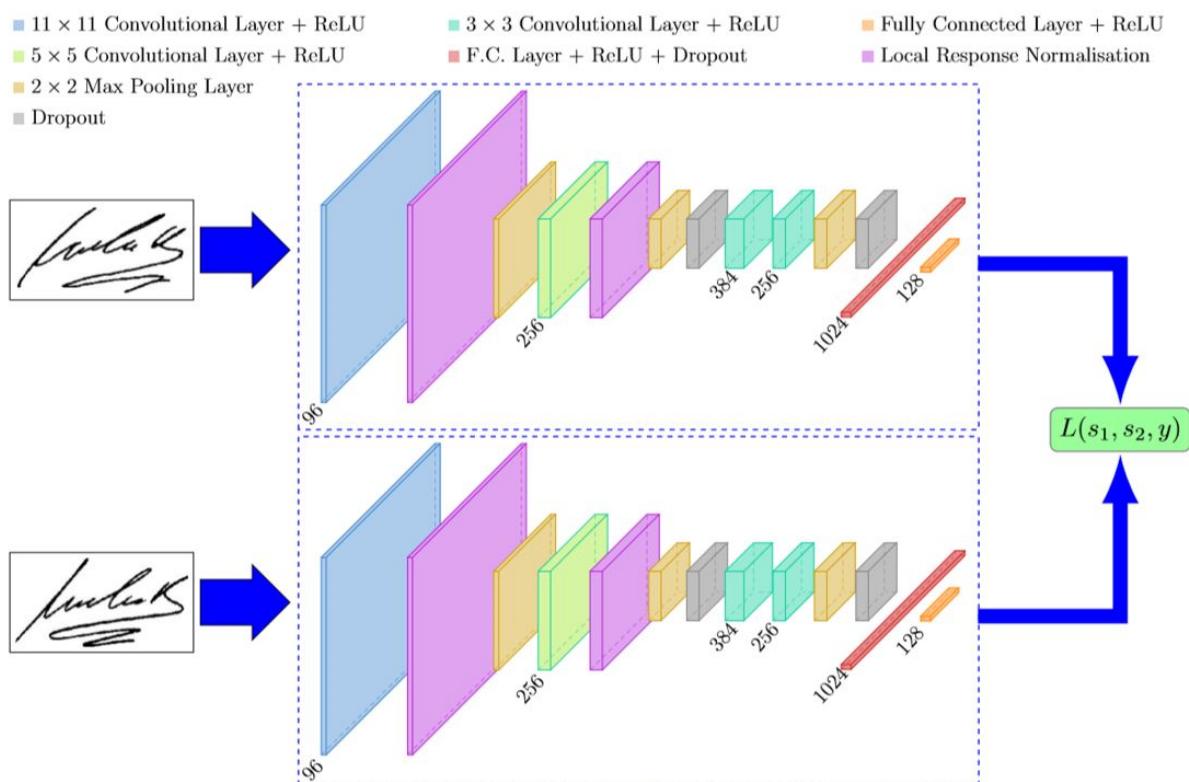


FIGURE 2.1 – Architecture de SigNet

Comme représenté dans la figure 2.1, nous avons deux réseaux identiques qui prennent en entrée une signature afin de calculer la différence de score entre les deux vecteurs de sorties. Notre modèle est composé de :

- **Convolutional Layer** : Les couches convolutives traitent une image d'entrée en faisant glisser un certain nombre de petits filtres à travers chaque région possible et en sortant le produit scalaire du filtre et l'image au niveau de chaque région. Les paramètres apprenables pour une couche convolutionnelle sont les poids de chaque filtre et une valeur du biais pour chaque filtre. Au niveau de notre modèle, on réalise en succession l'opération de convolution de l'image en entrée avec un filtre de taille précisée pour chaque couche (11×11 , 3×3 , 5×5).
- **ReLU** : ou Rectified Linear Unit est une fonction d'activation ayant pour expression :

$$f(x) = \max(x, 0) \quad (2.1)$$

Les couches ReLu n'ont pas de paramètres apprenables.

- **Max pooling Layers** : Pour chaque sous matrice de taille 2×2 le maximum des 4 pixels est choisi comme valeur de sortie. L'objectif de cette couche est de sous-échantillonner l'image en réduisant sa dimension.
- **Fully Connected Layer** : C'est une couche où toutes les entrées d'une couche sont connectées à chaque unité d'activation de la couche suivante.
- **Dropout** : Nous ignorons des connections avec une probabilité p . C'est une technique utilisée pour résister au phénomène de sur-apprentissage. Dans notre cas, la probabilité de Dropout $p = 0.3$ lorsque la couche de Dropout suit celle de Max pooling et $p = 0.5$ lorsqu'on suit une Fully Connected Layer.
- **Local Response Normalisation** : C'est une opération de normalisation sur les valeurs de pixels de l'image en entrée.

2.2 Pré-traitement

Dans le but de maximiser les performances de notre modèle, un pré-traitement sur les images d'entrées s'impose. Dans notre cas, notre pré-traitement se présente sous la forme de rogner les images selon une taille fixe étudiée : 155×220 . Ce pré-traitement nous permet d'introduire moins de facteurs de bruit sur le modèle afin qu'il apprenne les vraies caractéristiques d'une signature imitée.

2.3 Entraînement

Dans le but d'entraîner notre modèle, nous mettons en place, tout d'abord, l'architecture de notre modèle (Section 2.1). Nous créons ainsi une fonction qui nous permet de combiner les différentes couches de notre modèle et qui nous retourne notre réseau de neurones complet.

```

def create_base_network_signet(input_shape):
    '''Base Siamese Network'''

    seq = Sequential()
    seq.add(Conv2D(96, kernel_size=(11, 11), activation='relu', name='conv1_1', strides=4, input_shape=input_shape,
                  init='glorot_uniform', dim_ordering='tf'))
    seq.add(BatchNormalization(epsilon=1e-06, mode=0, axis=1, momentum=0.9))
    seq.add(MaxPooling2D((3,3), strides=(2, 2)))
    seq.add(ZeroPadding2D((2, 2), dim_ordering='tf'))

    seq.add(Conv2D(256, kernel_size=(5, 5), activation='relu', name='conv2_1', strides=1, init='glorot_uniform', dim_ordering='tf'))
    seq.add(BatchNormalization(epsilon=1e-06, mode=0, axis=1, momentum=0.9))
    seq.add(MaxPooling2D((3,3), strides=(2, 2)))
    seq.add(Dropout(0.3))# added extra
    seq.add(ZeroPadding2D((1, 1), dim_ordering='tf'))

    seq.add(Conv2D(384, kernel_size=(3, 3), activation='relu', name='conv3_1', strides=1, init='glorot_uniform', dim_ordering='tf'))
    seq.add(ZeroPadding2D((1, 1), dim_ordering='tf'))

    seq.add(Conv2D(256, kernel_size=(3, 3), activation='relu', name='conv3_2', strides=1, init='glorot_uniform', dim_ordering='tf'))
    seq.add(MaxPooling2D((3,3), strides=(2, 2)))
    seq.add(Dropout(0.3))# added extra
    seq.add(Flatten(name='flatten'))
    seq.add(Dense(1024, W_regularizer=l2(0.0005), activation='relu', init='glorot_uniform'))
    seq.add(Dropout(0.5))

    seq.add(Dense(128, W_regularizer=l2(0.0005), activation='relu', init='glorot_uniform')) # softmax changed to relu

    return seq

```

FIGURE 2.2 – Implémentation de l'architecture de notre réseau

Conformément à la définition d'un réseau siamois, notre modèle est défini par deux sous-réseaux identiques. De plus, nous avons les deux vecteurs de sorties qui nous permettent par la suite de calculer la différence de score par la méthode de la distance euclidienne que nous verrons par l'équation (2.2).

```

#network definition
base_network = create_base_network_signet(input_shape)

input_a = Input(shape=(input_shape))
input_b = Input(shape=(input_shape))

# because we re-use the same instance `base_network` ,
# the weights of the network
# will be shared across the two branches
processed_a = base_network(input_a)
processed_b = base_network(input_b)

# compute the Euclidean distance between the two vectors in the latent space
distance = Lambda(euclidean_distance, output_shape=eucl_dist_output_shape)([processed_a, processed_b])

model = Model(input=[input_a, input_b], output=distance)

```

Deux sous-réseaux identiques conformément à la définition d'un réseau siamois

Les deux vecteurs de sortie

Calcul de la différence de score (distance euclidienne)

FIGURE 2.3 – Définition de notre réseau

Nous compilons notre modèle en utilisant l'optimiseur RMSProp et la fonction de perte appelée *Contrastive Loss*.

L'idée centrale de RMSprop est de conserver la moyenne mobile des gradients carrés pour chaque poids. Et puis, nous divisons le gradient par la racine carrée du carré moyen.

C'est pourquoi il s'appelle RMSprop (Root Mean Square) [5].

La fonction de perte utilisée dans notre modèle est appelée *Contrastive Loss* [6] et elle est définie par l'expression suivante :

$$L = \frac{1}{2N} \left(\sum_{n=1}^N y_n d_n^2 + (1 - y_n) \max(\text{margin} - d_n^2, 0)^2 \right) \quad (2.2)$$

avec :

- y_n est la sortie réelle. Elle indique si pour une paire d'échantillons, ces signatures sont considérées comme similaires ou non.
- d_n^2 est la distance euclidienne entre deux paires de signatures paramétrées par les poids appris dans une couche particulière du réseau.
- margin est la marge > 0 . La marge définit un rayon autour de l'espace d'intégration d'un échantillon de sorte que des paires d'échantillons différents ne contribuent à la fonction de perte contrastive que si la distance d_n^2 est dans la marge.
- N est le nombre d'échantillons

Lorsque nous avons compilé notre modèle, nous faisons en sorte d'enregistrer les poids de notre modèle pour chaque époque dans le but d'utiliser le meilleur poids pour le test et la validation de notre modèle. De plus, nous réduisons le taux d'apprentissage d'un facteur 0,1 si la perte de validation ne s'améliore pas après 5 époques.

Finalement, nous débutons l'entraînement de notre modèle pour un nombre d'époques défini à 100. Néanmoins, nous arrêtons l'entraînement en utilisant un arrêt précoce si la perte de validation ne s'améliore pas au-delà de 12 époques.

```
# compile model using RMSProp Optimizer and Contrastive loss function defined above
rms = RMSprop(lr=1e-4, rho=0.9, epsilon=1e-08)
model.compile(loss=contrastive_loss, optimizer=rms)

# Using Keras Callbacks, save the model after every epoch
# Reduce the learning rate by a factor of 0.1 if the validation loss does not improve for 5 epochs
# Stop the training using early stopping if the validation loss does not improve for 12 epochs
callbacks = [
    EarlyStopping(patience=12, verbose=1),
    ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.000001, verbose=1),
    ModelCheckpoint('/content/drive/My Drive/projetCassiopee/Weights/CEDAR/CEDAR-{epoch:03d}.h5', verbose=1, save_weights_only=True)
]

results = model.fit_generator(generate_batch(orig_train_CEDAR, forg_train_CEDAR, batch_sz),
                             steps_per_epoch = num_train_samples//batch_sz,
                             epochs = 100,
                             validation_data = generate_batch(orig_val_CEDAR, forg_val_CEDAR, batch_sz),
                             validation_steps = num_val_samples//batch_sz,
                             callbacks = callbacks)
```

FIGURE 2.4 – Entraînement de notre réseau sur la base de données CEDAR

Chapitre 3

Expériences et Résultats

Dans cette section, nous parlons de nos expériences et les résultats que nous avons obtenus. Nous avons entraîné notre modèle sur BHSig260-Hindi et l'avons testé sur les autres bases de données. Ensuite, nous avons pris la même structure de ce modèle et l'avons entraîné sur la base de données CEDAR. Nous présentons nos résultats par rapport à ces deux expériences et nous comparons les résultats de ces expériences.

3.1 Protocole expérimental

Nous avons entraîné notre modèle, une première fois, sur la base de données BHSig260-Hindi, ensuite sur la base de données CEDAR. Nous avons, alors, décidé de diviser nos bases de données pour l'entraînement, la validation et le test de ces deux bases de données comme suite :

Datasets	Training	Validation	Testing
CEDAR	35	10	10
BHSig260-Hindi	120	20	20

TABLE 3.1 – Répartition de chaque base de données pour notre modèle

Base de données de l'entraînement

BHSig260-Hindi : Nous entraînons le modèle sur les signatures de 120 auteurs aléatoires parmi les 160. Pour chaque auteur, nous prenons la totalité des signatures authentiques, soit 24 signatures, et 12 signatures imitées aléatoirement tirées des 30 existantes. Donc, pour chaque auteur nous avons $C_{24}^2 = 276$ paires « Genuine, Genuine » possibles et $12 \times 24 = 288$ paires « Genuine, Forged » possibles. Au total nous avons un nombre d'échantillons égal à $120 \times (276 + 288) = 67680$.

CEDAR : Nous entraînons le modèle sur les signatures de 35 auteurs aléatoires parmi les 55. Pour chaque auteur, nous prenons la totalité des signatures authentiques, soit 24 signatures, et 12 signatures imitées aléatoirement tirées des 24 existantes. Donc, pour chaque auteur nous avons $C_{24}^2 = 276$ paires « Genuine, Genuine » possibles et $12 \times 24 = 288$ paires « Genuine, Forged » possibles. Au total nous avons un nombre d'échantillons égal à $35 \times (276 + 288) = 19740$.

3.2 Mesures de l'évaluation des performances

Nous évaluons la performance de notre modèle siamois en utilisant trois mesures :

- **Classification Accuracy** : C'est le nombre de prédictions correctes faites par notre modèle divisé par le nombre total de prédictions faites.

$$\text{Accuracy} = \frac{\text{true_positive} + \text{true_negative}}{\text{total_pred}} \quad (3.1)$$

- **True Positive Rate (TPR)** : C'est le nombre moyen de signatures authentiques prédites correctement par notre modèle.

$$TPR = \frac{\text{true_positive}}{\text{true_positive} + \text{false_positive}} \quad (3.2)$$

- **True Negative Rate (TNR)** : C'est le nombre moyen de signatures falsifiées prédites correctement par notre modèle.

$$TNR = \frac{\text{true_negative}}{\text{true_negative} + \text{false_negative}} \quad (3.3)$$

3.3 Résultats de l'entraînement du modèle

Comme expliqué dans la section 2.3, nous enregistrons les poids correspondant à chaque époque. Afin de choisir le meilleur poids, nous nous basons sur la *val_loss* qui est la perte sur l'ensemble de validation [7]. Ceci est calculé par époque et est donné sous forme de métrique sur l'ensemble de validation dans sa totalité.

Nous avons entraîné notre modèle sur un total de 18 époques sur la base de données BHSig260-Hindi afin d'avoir le meilleur poids correspondant à l'époque 5 qui nous permet d'avoir de bonnes prédictions des signatures. Pour la base de données CEDAR, nous avons entraîné notre modèle sur un total de 15 époques pour avoir le meilleur poids correspondant à l'époque 9.

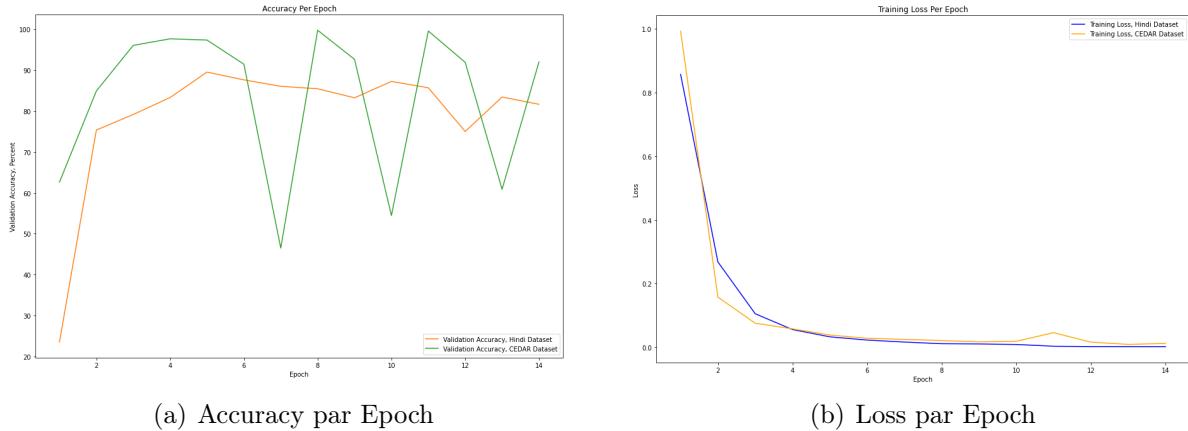


FIGURE 3.1 – Entraînement de notre modèle

Nous entraînons notre modèle sur deux bases de données, à savoir BHSig260-Hindi et CEDAR, une à la fois. Pour l’entraînement sur la base de données Hindi, la *classification accuracy* atteint 80%. La performance de notre modèle est assez bonne pour prédire les signatures.

Lors du *training* sur CEDAR, la *classification accuracy* atteint 100%, ce modèle promet alors de très bonnes prédictions des signatures lors des tests.

Category	Training on BHSig260-Hindi	Training on CEDAR
Classification Accuracy	80.86	100
TPR	86.12	100
TNR	75.25	100

TABLE 3.2 – Performance de l’entraînement du modèle

3.4 Résultats obtenus

Le tableau 3.3 présente les résultats des *accuracy*, *TPR* et *TNR* de notre réseau de neurones. Une des raisons possible qui permet d’expliquer la baisse des performances quand le modèle est entraîné sur la base de données CEDAR, est que les signatures originales sont bruitées (l’arrière plan des signatures authentiques est gris), et donc notre modèle se base sur ce critère pour classer les signatures imitées et originales.

Une deuxième raison pourrait être le nombre de signatures dans cette base de données qui est très faible (55 signataires), ce qui peut influer sur les performances du modèle.

Enfin, il est à noter que nous n'avons testé notre modèle que sur seulement 20 signatures de chaque base de données.

Databases	Trained on	#Signers	Accuracy	TPR	TNR
BHSig260 - Hindi	BHSig260-Hindi	160	80.68	58.79	75.25
	CEDAR	160	56.54	54.29	56.54
CEDAR	BHSig260-Hindi	55	64.00	88.63	40.52
	CEDAR	55	100.00	100.00	100.00
BHSig260 - Bengali	BHSig260-Hindi	100	59.66	75.00	44.33
	CEDAR	100	59.66	75.00	44.33
GPDS300	BHSig260-Hindi	300	53.53	75.70	31.37
	CEDAR	300	51.30	49.07	53.53

TABLE 3.3 – Comparaison de notre modèle sur diverses bases de données

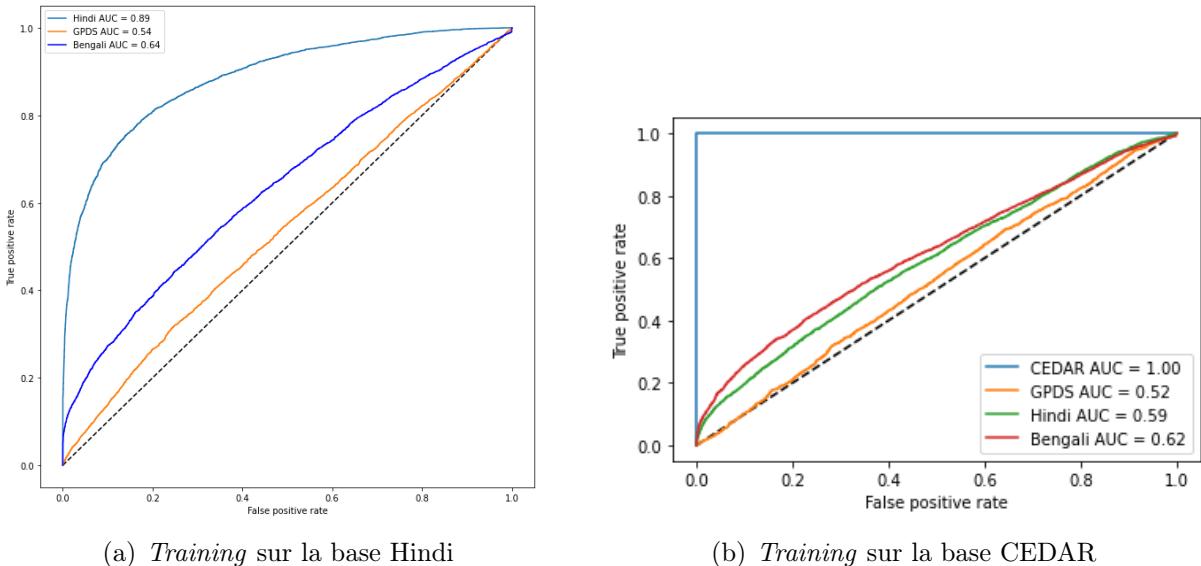


FIGURE 3.2 – Courbe de ROC

La *classification accuracy* des tests dépend de la façon dont les tests distinguent une signature authentique d'une signature falsifiée. La *classification accuracy* est mesurée par l'aire sous la courbe ROC. Une zone de 1 représente un test parfait ; une zone de 0,5 représente un test inutile.

L'aire sous la courbe de ROC mesure la discrimination, c'est-à-dire la capacité du test à classer correctement les signatures originales ou imitées [8]. Le critère AUC_ROC permet de mesurer la performance de notre modèle à l'aide d'une variable seuil qui, pour notre cas est de 50%.

Nous observons, que pour la figure 3.2, les tests sont loin d'être parfait. En effet, tous les

tests ont une valeur aux alentours des 50% pour l'aire sous la courbe de ROC. Cependant, pour les tests de notre modèle entraîné sur la base de données Hindi, nous avons d'assez bons résultats puisque l'AUC_ROC est légèrement au dessus de notre seuil du 0,5.

		Test Datasets			
		Hindi	Bengali	CEDAR	GPDS
Train Datasets	Hindi	80.86%	59.66%	64%	53.53%
	CEDAR	56.54%	59.66%	100%	51.30%

FIGURE 3.3 – Matrice de confusion

Nous observons, selon la figure 3.3, que les meilleures performances du modèle sont obtenues lorsque le modèle est entraîné sur la même base de données.

Etant donné que la base de données BHSig260-Hindi a des *features* bien particuliers, puisque toutes les signatures sont en langage Hindi et la signature d'une personne est son nom complet. Dans ce cas, il est fort probable que notre réseau modélise ces *features* pour permettre d'identifier les signatures authentiques des signatures falsifiées.

L'une des raisons qui pourront expliquer la baisse des performances de notre modèle est le fait que les bases de données utilisées pour le *training* de notre réseau sont assez limitées en nombre de signataires (160 et 55 signataires respectivement pour Hindi et CEDAR).

Par ailleurs, nous concluons que notre modèle n'est pas résistant au bruit. Comme nous avons vu dans le cas où notre réseau est entraîné sur la base de données CEDAR, le modèle s'entraîne sur des signatures bruitées, mais il n'arrive pas à prédire correctement les signatures des autres bases de données lors du test.

Toutefois, le réseau entraîné sur la base de données Hindi nous a donné de bons résultats lors de la prédiction des signatures bruitées de la base CEDAR. Avec un taux de 64%, nous concluons alors, par rapport à ce modèle, qu'il est assez résistant au bruit.

Conclusion

Pour conclure, nous avons appliqué des méthodes d'apprentissage profond pour analyser des signatures manuscrites et détecter les éventuelles tentatives de fraude à travers les signatures imitées. Le réseau siamois était le réseau le plus adapté pour une telle application en considérant sa caractéristique différentielle qui crée une corrélation entre deux images au lieu de traiter chaque image séparément.

L'obtention d'un réseau performant pour toutes les bases de données s'est avéré être une tâche compliquée à mettre en œuvre. Nous expliquons ceci par le fait que les différences entre les caractéristiques de chaque base de données peuvent être source de bruit et de perturbation pour le modèle mis en place. Un exemple facilement remarquable est celui des images de la base CEDAR dans laquelle la majorité des signatures authentiques ont un fond gris tandis que la majorité des signatures imitées ont un arrière plan blanc. Cette spécificité nous permet d'obtenir une précision de 100% du modèle entraîné sur la base CEDAR et testé sur CEDAR.

Ceci est prévisible du fait que la caractéristique majeure de distinction extraite est le fond, ce qui facilite la prise de décision du réseau mais l'affaiblit lors des tests sur les autres bases de données. Cependant, si un pré-traitement de l'arrière-plan est effectué, nous pensons que ça pourrait améliorer la précision des tests sur les autres bases. Faute de temps, nous n'avons pas eu l'occasion de mettre en œuvre cette expérimentation pour prouver l'augmentation de l'*accuracy* du réseau après un pré-traitement des signatures de la base de données CEDAR.

Ceci dit, même avec un tel pré-traitement, nous sommes bien loin d'un modèle performant pour toutes les signatures. Pour obtenir un tel résultat, nous envisageons un modèle plus résistant au bruit et avec des caractéristiques plus générales. Nous avons trouvé des résultats très proches de l'état de l'art [4].

Ce rapport clôture notre travail pour le projet Cassiopée. Ce dernier nous a permis d'acquérir des compétences avancées de l'intelligence artificielle et en particulier celui de l'apprentissage profond. Grâce à ce projet, nous avons pu nous familiariser avec la bibliothèque Keras ainsi que TensorFlow mais aussi nous avons découvert l'outil Google Colab qui nous a permis de travailler ensemble durant la période de confinement due au Covid-19.

Bibliographie

- [1] Gabe Alvarez, Blue Sheffer, and Morgan Bryant. Offline signature verification with convolutional neural networks. http://cs231n.stanford.edu/reports/2016/pdfs/276_Report.pdf, 2016.
- [2] Adem Tekerek, Nurettin Topaloglu, and Mutlu Yapıcı. Convolutional neural network based offline signature verification application. December 2018.
- [3] Luiz Gustavo Hafemann, Robert Sabourin, and Luiz Soares de Oliveira. Fixed-sized representation learning from offline handwritten signatures of different sizes. *Document Analysis and Recognition*, 21 :219–232, April 2018.
- [4] Sounak Dey, Anjan Dutta, J. Toledo, Suman Ghosh, Josep Lladós, and Umapada Pal. Signet : Convolutional siamese network for writer independent offline signature verification. July 2017.
- [5] <https://keras.io/api/optimizers/rmsprop/>. [consulté le 07/06/2020].
- [6] <https://towardsdatascience.com/contrastive-loss-explained-159f2d4a87ec>. [consulté le 03/06/2020].
- [7] https://www.reddit.com/r/learnmachinelearning/comments/7b11pj/keras_val_loss/, 2018. [consulté le 07/06/2020].
- [8] <http://gim.unmc.edu/dxtests/roc3.htm>. [consulté le 03/06/2020].

Annexe A

Planning du projet

