



Projet Final

Classification de visages réels ou artificiels à l'aide du
Deep Learning

Forage de données (8INF954)

Enseignant : Abdenour Bouzouane

Maîtrise en Informatique

Augustin Primous Prince POMALEGNI

Arthur Bachelet

Automne 2020

Table des matières

Introduction	3
1 Présentation des réseaux de neurones à convolution	4
1.1 Traitement des images	4
1.1.1 Couches de convolutions	5
1.1.2 Couches de pooling	6
1.2 Réseau pleinement connecté	7
1.2.1 Couche de flattening	8
1.2.2 Couches intermédiaires	8
1.2.3 Couche de sortie	9
2 Implémentations dans le cas de la détection de visages artificiels	12
2.1 Données utilisées pour l'étude	12
2.2 Mise en place des réseaux CNN	13
2.2.1 CNN basique	13
2.2.2 CNN avec dropout	15
2.2.3 CNN avec régulation de type L2	15
2.2.4 CNN inspiré d'un article	15
2.2.5 Réseaux CNN basés sur le Transfer Learning	16
3 Mesure et Analyse des résultats	21
3.1 Mesure des performances	21
3.1.1 CNN basique	21
3.1.2 CNN avec dropouts	22
3.1.3 CNN avec dropouts et optimisation L2	22
3.1.4 CNN basé sur l'article	22
3.1.5 CNN basé sur le transfer learning	23
3.2 Analyse et comparaison des résultats	23
3.2.1 CNN basique	24
3.2.2 CNN avec dropouts	25
3.2.3 CNN avec dropouts et régulation L2	26
3.2.4 CNN basé sur l'article	28
3.2.5 CNN basé sur le transfer learning	29
Conclusion	34
Bibliographie	35

A Annexe	36
A.1 Liens vers les googles collab	36
A.2 Article de référence	36

Introduction

Le développement des intelligences artificielles (IA) appliquées aux traitements d'image connaît un essor important depuis une décennie. En effet, aujourd'hui 49% des demandes de brevets déposés qui impliquent de l'IA concernent du traitement d'image [3]. Ce domaine suscite beaucoup de curiosité et beaucoup de moyens y sont déployés pour la recherche et le développement. Ces dernières années, un sujet particulier a fait beaucoup parler de lui, le "DeepFake". Cette technologie consiste à créer de fausses images ou vidéos, ou même encore des audios. Sa particularité est que ces fausses données peuvent paraître tout à fait réelles pour un être humain. Beaucoup d'outils permettent donc aujourd'hui de créer de fausses images, manuellement, ce qui reste cependant souvent visible, mais également assisté par une IA, permettant de créer des images parfaitement cohérentes pour l'oeil humain.

Le projet qui est présenté dans ce rapport a pour but d'explorer les méthodes et le fonctionnement du deep learning appliqué à la classification d'images. Le sujet sur lequel le projet se repose pour découvrir cette technologie est la classification de visages humains afin de découvrir si un visage est véritable ou s'il a été créé artificiellement. Ce document va donc proposer une méthode pour répondre à la problématique : **"Comment utiliser le deep learning pour détecter des visages humains artificiels ?"**.

Pour résoudre cette problématique, le dataset utilisé dans ce projet est un dataset Kaggle [5].

Pour ce faire, le document sera composé de trois parties. La première partie présentera la technologie utilisée pour répondre à ce problème. La seconde partie consistera en la présentation des différentes implémentations réalisées pour essayer de répondre à la problématique. La dernière partie comparera les différents résultats des implémentations. Et pour finir, le document conclura sur la performance de la solution apportée, et sur la méthode générale à utiliser pour ce genre de problème.

Présentation des réseaux de neurones à convolution

La problématique proposée dans ce document s'intéresse à l'utilisation du deep learning dans sa résolution. Le deep learning est un terme utilisé pour désigner une technique d'apprentissage automatique, cette technique d'apprentissage a la particularité d'accepter des données complexes qui seront prétraitées par l'algorithme de deep learning afin d'en extraire des caractéristiques différenciantes, pas forcément visibles par l'humain, et apprendra ensuite de ces données traitées.

Dans le cadre du problème traité, la méthode de deep learning qui semble la plus adaptée à de la classification d'images est la mise en place d'un "Réseau de neurones à convolution" (CNN). Ce type d'architecture permet en effet de faire de l'apprentissage supervisé à partir d'images. Cette architecture s'articule comme pour beaucoup d'architecture de deep learning en deux parties principales qui seront présentées dans ce chapitre.

1.1 Traitement des images

Comme expliqué dans l'introduction de ce chapitre, le deep learning permet l'utilisation de données complexes en incorporant dans son architecture un prétraitement permettant d'extraire des caractéristiques particulières, utile à l'apprentissage. Cette section va présenter le fonctionnement de cette partie de prétraitement dans le cadre des CNN, où elle joue le rôle de traitement d'images.

En effet, les CNN vont axer le prétraitement de leurs données autour du traitement d'images. Cette partie est composée de plusieurs couches, des couches de convolution et des couches de pooling. Ces couches sont en général mélangées entre elles afin de permettre un traitement plus efficace de l'image.

1.1.1 Couches de convolutions

Les couches de convolution permettent d'appliquer un filtre à l'image afin de la transformer, dans le but de faire ressortir les caractéristiques (features) qui permettront ensuite sa classification. Ce filtre (parfois appelé noyau ou kernel) est en fait une matrice d'une dimension donnée qui sera appliquée à notre image représentée sous une forme de matrice de pixel.

Appliquer le filtre sur une image revient à faire la multiplication des valeurs du filtre par les valeurs des pixels sur lesquels il est appliqué, et de faire l'addition de tous ces produits, afin de produire un pixel de sorti. L'application par convolution signifie que le filtre sera appliqué à toute l'image, par exemple en partant du coin supérieur gauche et en décalant colonne par colonne vers la droite, puis en reprenant au début de la ligne suivante quand le filtre a atteint la droite de l'image. La figure 1.1 représente l'application d'un filtre à une image. Elle montre que pour chaque application du filtre à une partie de l'image, on récupère la valeur d'un pixel de sortie (pixel construit à partir de 9 pixels de l'image de base). La convolution réduit donc légèrement la taille de l'image en supprimant une épaisseur d'un pixel sur tout le contour.

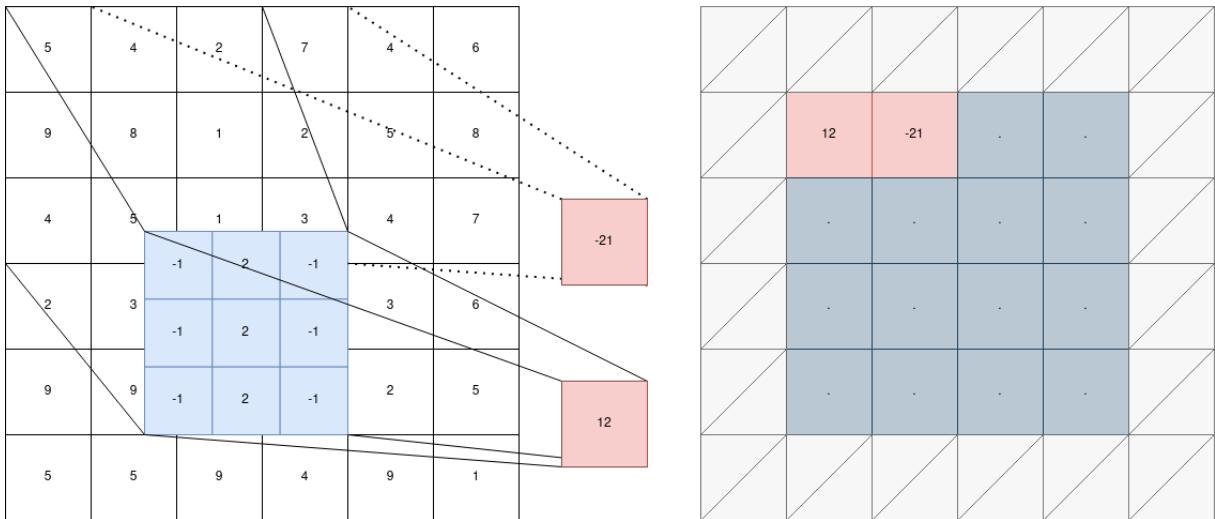


Figure 1.1 – Convolution d'une image par un filtre

En traitement d'images, certains filtres sont connus pour permettre la détection de caractéristiques, comme les contours des objets présents sur une image(filtre de

Sobel). Sur la figure 1.1 par exemple, le filtre permet la détection des lignes horizontales. Cependant, dans le cas du deep learning les caractéristiques permettant de classer les images ne sont pas définies, les filtres à utiliser ne peuvent donc pas être choisis à l'avance. Les CNN appliquent alors une méthode semblable à l'évolution des poids dans un réseau de neurones pleinement connectés. En se basant sur une fonction de perte, souvent la "cross entropy" entre les différentes classes de sortie, le réseau va effectuer une rétropropagation pour corriger le filtre et essayé d'en obtenir un plus performant [6].

De plus sur chaque couche de convolution, il est possible de tester plusieurs filtres en parallèle. Par exemple, pour une image de dimension 600px*600px en couleur, c'est à dire d'une dimension de 600*600*3 (car il y a les trois couches RGB) on peut configurer une couche de convolution pour obtenir une sortie de 579*579*32, le 32 signifiera que trente-deux filtres différents ont été testés en les répartissant sur les différentes couches de couleur de notre image, on obtient donc trente-deux images monochromes filtrées différemment. Cela est utile, car il peut être nécessaire d'avoir plusieurs versions de l'image filtrée différemment pour obtenir toutes les caractéristiques nécessaires à la classification.

1.1.2 Couches de pooling

Les couches de pooling ont une utilité quant à la quantité d'information traitée, il sera expliqué plus tard que le nombre de neurones nécessaire est directement lié à la dimension de l'image d'entrée. Il est donc nécessaire de réduire la quantité d'informations dans une image afin de permettre au réseau de ne pas utiliser une quantité de ressources trop importante. Les couches de pooling permettent de réduire la quantité de données que devra traiter notre système, en réduisant la résolution de l'image, sans modifier la profondeur (la profondeur correspond à la troisième dimension de nos données, par exemple les couches de couleurs RGB). Cependant, il est important de ne pas perdre les informations importantes de l'image en réduisant sa résolution. C'est pour cela que les CNN utilisent très souvent la fonction de Max-Pooling qui garde le pixel de valeur maximum parmi les pixels à réduire.

Comme pour la couche de convolution, le pooling dispose d'un noyau (kernel) qui définit la taille des zones de pixel à réduire. Par exemple, si le kernel a une dimension de 2×2 , il réduira la valeur d'un groupe de 4 pixels à un seul pixel. À la différence de la convolution, le noyau n'est pas glissé tout du long de notre image en retraitant des pixels traités dans les étapes précédentes, mais il est appliqué sur toute l'image par section distincte. La figure 1.2 montre l'application d'un max-pooling avec un noyau de 2×2 , divisant ainsi la taille de l'image par quatre.

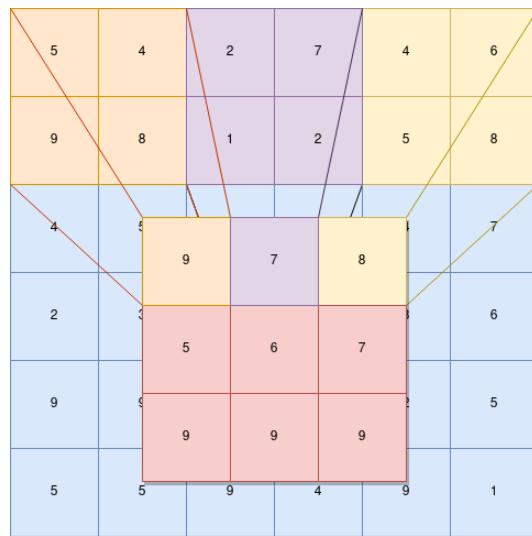


Figure 1.2 – Max-pooling de kernel 2×2

Cette opération de max-pooling permet, en gardant les valeurs élevées de pixels, de garantir un minimum de perte d'informations dans l'image, elle est donc très utilisée dans les CNN, car elle promet une bonne intégration entre les couches de convolution pour réduire la lourdeur du réseau tout en permettant quand même la découverte des caractéristiques utiles à la classification.

1.2 Réseau pleinement connecté

La seconde partie d'un CNN est constituée d'un réseau de neurones pleinement connecté, c'est cette partie qui permet de classifier les images à partir de leurs caractéristiques. Elle commence par une couche de flattening permettant de présenter les données arrivant de la partie traitement vers la partie de classification. Ensuite,

le réseau est constitué d'une ou plusieurs couches multi neuronales intermédiaires, finalement, une dernière couche permet de formater la sortie pour correspondre à la manière dont le réseau doit classifier les images.

1.2.1 Couche de flattening

La couche de flattening permet de mettre en forme correctement les données qui se présentent à l'entrée du réseau de neurones pour pouvoir les lui transmettre. Le principe est simple, la couche transforme la structure de données pour n'avoir plus qu'une seule dimension. Cette couche va donc pour chaque profondeur de notre structure de données transformer notre matrice en vecteur, et concaténer les vecteurs de chaque profondeur pour obtenir un vecteur contenant toutes les données en une seule dimension. La figure 1.3 propose une illustration de la méthode utilisée.

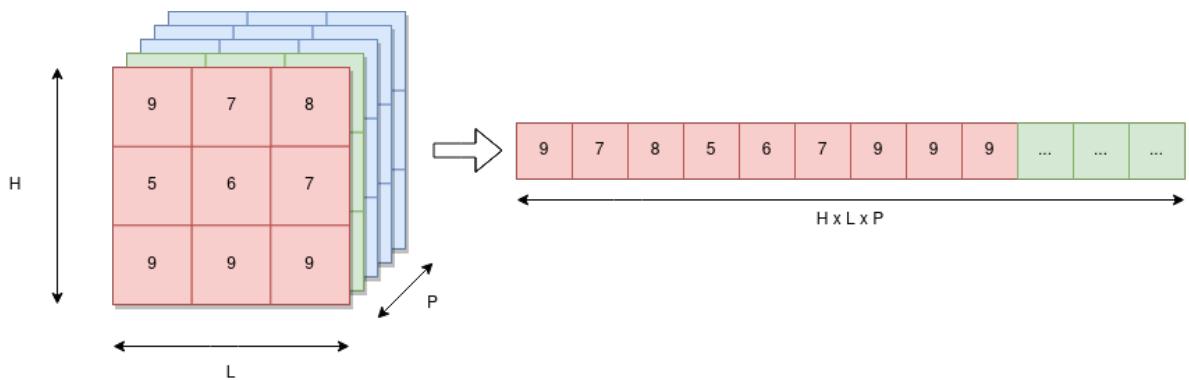


Figure 1.3 – Opération de flattening

1.2.2 Couches intermédiaires

Les couches intermédiaires sont les couches entre la couche de flattening et la couche de sortie, elles sont reliées entre elles, à la couche de flattening et à la couche de sortie par des liens pondérés. Ces pondérations permettent de prendre plus ou moins en compte les données provenant de certaines entrées. Chaque couche contient plusieurs neurones et les couches étant pleinement connectés, cela signifie que chaque neurone est connecté à tous les neurones de la couche précédente. Les

neurones fonctionnent ensuite de la même manière que pour un perceptron dont le fonctionnement a été présenté dans le cadre de ce cours. C'est-à-dire que toutes les entrées sont multipliées par le poids de leur lien et additionné, puis le résultat est passé dans une fonction d'activation qui définira la sortie du neurone. À chaque passage de données dans le réseau, celui-ci appliquera une rétropropagation pour corriger les poids. Cela se fait à l'aide de la descente du gradient qui permet de modifier petit à petit les poids pour trouver des optimums comme présentés dans le cours. Chaque itération va donc modifier la force des liens entre les neurones et c'est ce qui constituera "l'intelligence" (le classeur) de notre réseau. La figure 1.4 montre comment sont connectées les couches intermédiaires, le triangle représente la couche de sortie dont l'explication sera proposée dans la prochaine sous-section. L'épaisseur des liens représente leur pondération, et le vecteur d'entrée est le vecteur issu de la couche de flattening.

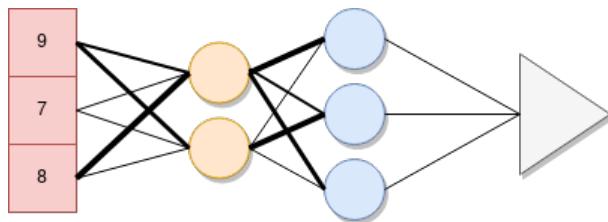


Figure 1.4 – Couche intermédiaire

Il existe une grande collection de fonctions d'activation utilisée par les neurones, il convient de chercher une fonction adaptée au problème traité. Beaucoup de personnes les ayant déjà utilisés, il est possible de trouver conseil dans des projets de nature similaire à celui que le réseau cherche à résoudre. Dans les CNN, les fonctions les plus utilisées semblent être des fonctions linéaires ou semi-linéaires.

1.2.3 Couche de sortie

La couche de sortie permet d'obtenir des valeurs compréhensibles pour la classification des données d'entrées. Elle dépend donc du nombre de classes de sortie possible. Elle utilise cependant presque toujours une fonction permettant de récupérer une valeur dans l'intervalle unitaire ($[0, 1]$).

On distingue donc deux types majeurs d'architecture. Les architectures à deux classes, avec une couche de sortie ne contenant qu'un seul neurone. La différenciation se fera alors dans la valeur, proche de 1 pour une classe et proche de 0 pour l'autre. L'autre type contient les architectures ayant plus de classes, dans ce cas la technique est de mettre en place un neurone par classe sur la couche de sortie. Si le neurone s'active (résultat proche de 1), la donnée sera donc étiquetée par la classe associée à ce neurone. La figure 1.5 montre les deux architectures de sortie possibles.

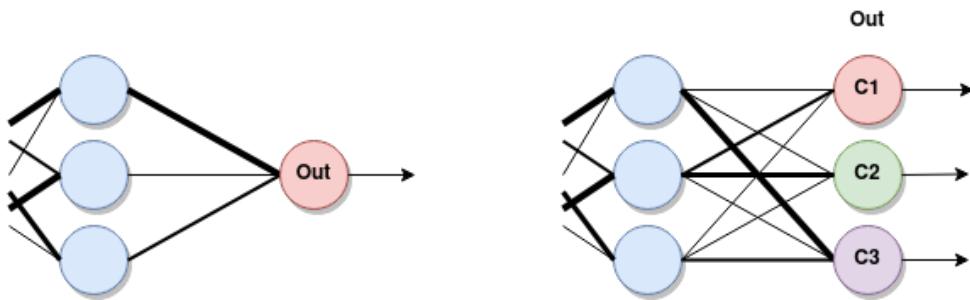


Figure 1.5 – Types de couches de sorties

La fonction sigmoïde est une fonction très souvent utilisée sur la couche de sortie, car elle permet d'obtenir des résultats dans l'échelon unitaire sans pour autant être binaire, ce qui permet d'être plus précis dans le calcul des taux d'erreur et donc dans la correction par rétropropagation.

Au final, la représentation d'un CNN complet est donnée figure 1.6. L'opération de flattening crée en réalité un vecteur cinq fois plus grand, mais pour que l'illustration reste claire, il n'est que partiellement représenté.

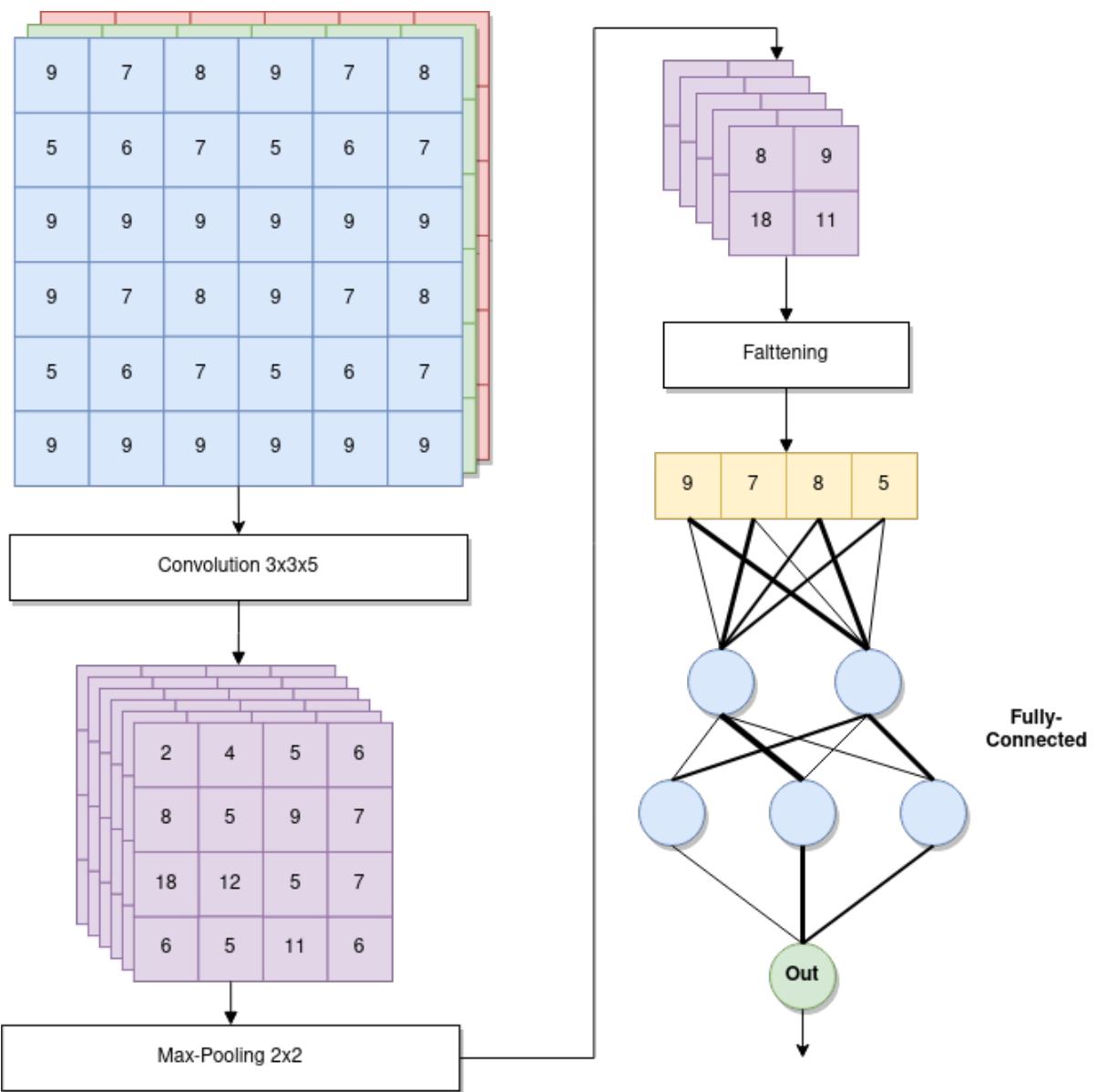


Figure 1.6 – Exemple de CNN complet

Implémentations dans le cas de la détection de visages artificiels

Ce chapitre se concentrera sur les différentes implémentations réalisées pour l'étude du problème. Dans un premier temps, il détaillera le dataset utilisé pour cette étude, puis il présentera les différentes architectures, de la plus simple à la plus complexe, qui ont été implémentées afin de les comparer.

2.1 Données utilisées pour l'étude

Dans cette partie, nous parlerons des données utilisées dans le cas de cette étude qui consiste en la classification d'images réelles de celles fausses ou modifiées (artificielles).

En effet, la collection et le choix des données est une partie importante dans le cadre d'une étude surtout lorsqu'elle est liée à l'utilisation d'images de personnes (respect des droits d'image). S'il avait été nécessaire de récolter des données dans le cadre de cette recherche, cela aurait nécessité l'accord des participants bien évidemment volontaires et ayant conscience que leur image servira dans le cadre d'une étude. Étude dont il faudrait leur spécifier les différents objectifs et la manière dont les images auraient été utilisées.

Cependant, dans le cadre de ce projet et compte tenu de la grande diversité de sources de données disponibles sur le net, il n'a pas été nécessaire de passer par cette étape. Nos données proviennent d'un dataset disponible sur "Kaggle" [5], disponible en open source et qui est composée d'images répondant aux critères recherchés. Ces critères ne sont rien d'autre que :

- la présence d'images réelles de visage humain n'ayant pas été modifiées que nous appellerons : *real faces* ;

— des images générées artificiellement, que nous nommerons : *fake faces*.

Un impératif pour le choix du dataset était l'équilibre entre les deux classes distinctes pour permettre une étude adéquate en garantissant un apprentissage équilibré. Le dataset sélectionné contient donc un total de 2041 images, dont 1081 réelles et 960 artificielles. La structure de celui-ci est très simple, dans un dossier sont placées les images réelles, dans un autre les images artificielles. Les images sont des images RGB de 600*600 pixels.

Au cours de l'étude, le dataset semble avoir montré cependant quelques limites en termes de quantité d'informations, impactant de forte manière les résultats.

Pour contourner ce problème, nous avons pensé à générer de nouvelles données à partir de celles que nous avions, grâce aux opérations qui peuvent être faites sur une image. Un deuxième dataset a été mis en place faisant le double du premier. Pour créer ce dataset, les données du premier ont simplement été recopiées, puis pour chaque image une copie contenant une modification de zoom ou de légère rotation a été ajoutée de manière à doubler la taille du dataset de base, sans récolter de nouvelles données.

2.2 Mise en place des réseaux CNN

2.2.1 CNN basique

Dans un premier temps, avec les informations présentées dans le premier chapitre, un premier CNN a été développé. Le développement de tous les CNN de cette étude a été réalisé avec tensorflow et keras sur la plate-forme "google collab". Ce premier CNN a donc une architecture particulièrement simple.

La partie traitement d'images est constituée de trois couches de convolution de noyaux 3*3, en augmentant la profondeur de sortie (et donc le nombre de filtres essayé), la première couche donne une profondeur de 32, la seconde de 64 et la troisième de 128 (ces choix ont été effectués après plusieurs essais et paraissent assez adaptés). Ces trois couches de convolutions sont séparées par des couches de Max-

Pooling chacune avec un noyau de dimension 2*2. En sortie de la partie de traitement d'images, la dimension des données est donc de 128 fois l'image de 73*73 pixels filtrée différemment.

Côté réseau de neurones, une couche de flattening vient présenter ces 682112 pixels au réseau de neurones. Celui-ci dispose d'une couche intermédiaire de 64 neurones utilisant une fonction d'activation de type leakyRELU. Cette fonction d'activation est proposée figure 2.1 et a été choisie, car contrairement à la fonction RELU qui est largement utilisée dans les CNN, elle présente l'avantage de ne pas mettre la sortie à 0 en cas de non-activation d'un neurone, ce qui permet malgré tout corriger le poids de ce neurone lors de la rétropropagation. Cela évite de se retrouver dans une situation où une majorité des neurones ne sont pas activés et propose une valeur nulle en sortie, entraînant une modification d'une toute petite partie des poids lors de la rétropropagation.

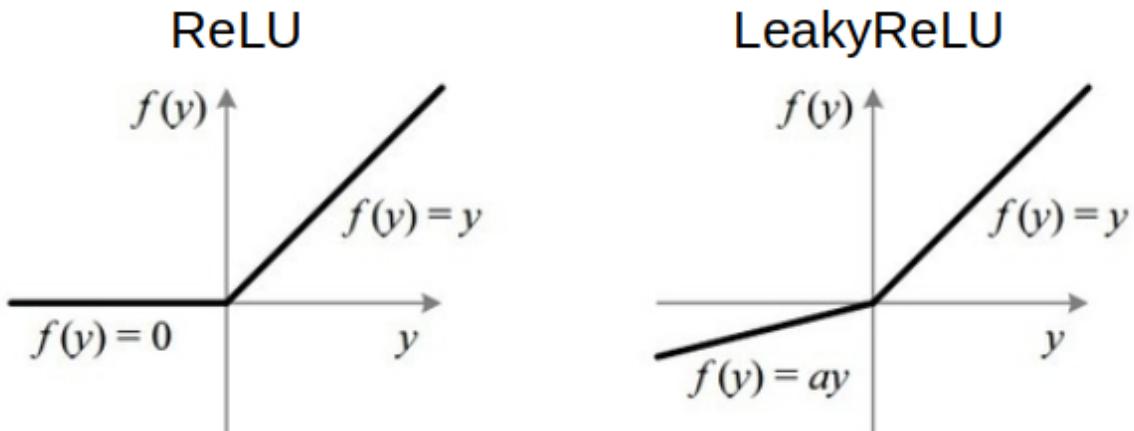


Figure 2.1 – Fonction ReLU et LeakyReLU

Enfin, il dispose d'une couche de sortie contenant un seul neurone permettant de répondre au problème puisqu'il ne contient que deux classes. Cette couche utilise la fonction sigmoïde pour ramener la valeur dans l'intervalle unitaire.

Le réseau se base sur la "cross entropy" pour corriger ses poids.

2.2.2 CNN avec dropout

La deuxième architecture rajoute un élément, afin d'éviter le surapprentissage dans les CNN, il s'agit d'un "dropout". Le principe du dropout est de désactiver aléatoirement un pourcentage de neurone à chaque itération, c'est-à-dire que sa sortie vaudra "0". Cette partie aléatoire permet en fait de sortir des optimums locaux dans lesquels la méthode de la descente du gradient peut parfois se retrouver coincée. L'architecture qui a été implémentée contient alors un "Dropout" après chaque couche de maxpooling et entre le flattening et la couche intermédiaire. Cette architecture aura pour but de déterminer l'impact des dropouts sur le CNN afin de vérifier si une amélioration est constatée.

2.2.3 CNN avec régulation de type L2

De la même manière que pour le dropout, cette troisième architecture va simplement venir rajouter une régulation de type L2 (connue sous le nom de Ridge regression [4]). Cette opération va en fait venir ajouter une pénalité, qui dépend directement du carré de la différence entre la valeur obtenue et la valeur attendue, lors du calcul de notre fonction de perte basée sur la "cross entropy". Cette modification permettra de modifier légèrement la correction lors de la rétropropagation et permettra d'éviter des modifications parfois trop violentes des poids. Un coefficient permet de définir avec quelle intensité cette régulation a lieu, car la régulation apporte un biais plutôt élevé. Il est donc important de l'atténuer suffisamment avec ce paramètre pour éviter qu'elle n'empêche une rétropropagation correcte.

2.2.4 CNN inspiré d'un article

À titre de comparaison, cette nouvelle implémentation se base sur un travail réalisé dans l'article "Fake Faces Identification via Convolutional Neural Network" disponible en annexe, où le but était similaire à celui de cette étude, mais basé sur un dataset beaucoup plus gros. Une nouvelle implémentation a été mise en place en utilisant la structure du CNN utilisé dans cet article. La structure de la partie traitement

d'images est la même que dans le CNN basique réalisé plus tôt, car dans le contexte de cette étude le filtre de type "High Pass" n'a pas été implémenté pour rester dans de l'apprentissage automatisé.

Ce que l'article apporte en termes de modification, comparé à la structure du CNN basique réalisé, est la mise en place de deux couches intermédiaires dans le réseau de neurones de 1024 et 512 neurones respectivement. La fonction d'activation de l'article était softmax, mais la sigmoïde a été conservée dans notre cas. Le but sera de constater l'importance du bon nombre de couches et de neurones par couche pour la résolution de notre problème. Finalement, on notera que l'article impose l'utilisation de adam comme optimiseur.

2.2.5 Réseaux CNN basés sur le Transfer Learning

Comme mentionné précédemment, les réseaux de neurones apprennent les features sur les données d'entraînement et en profitent pour définir les différents paramètres du réseau notamment les valeurs des poids qui permettront au réseau de faire de bonnes prédictions. Cette phase d'entraînement est d'autant plus efficace qu'elle nécessite une grande puissance de calcul pour faire des traitements rapides : puissance de calcul dont on ne peut pas tous se prévaloir.[1] [2] C'est là qu'intervient le transfer learning (apprentissage par transfert).

Le transfer learning [8] permet donc de faire du deep learning (construire des réseaux) en optimisant la puissance de calcul de l'ordinateur utilisé.

Un autre avantage du transfer learning en plus d'accélérer l'entraînement du réseau, est la capacité qu'il offre d'éviter le surapprentissage. En effet, en utilisant cette méthode, nous ne partons plus d'un ensemble de valeurs de paramètres(poids) nulles. Celles-ci sont acquises directement du réseau servant de source dans le transfert d'apprentissage. Ainsi, en fonction de la stratégie de transfer learning adoptée, ces différents paramètres pourront être ajustés au cours de son entraînement pour donner des résultats qui pourront convenir aux features (caractéristiques) correspondantes à notre source de données. Les stratégies de transfer learning disponibles sont donc : fine-tuning total, extraction de features et le fine-tuning partiel.

Stratégie 1 : Le fine-tuning total

Cette stratégie consiste à remplacer la dernière couche fully-connected du réseau pré entraîné par un classeur adapté au nouveau problème (SVM, régression logistique...) et initialisé de manière aléatoire. Toutes les couches sont ensuite entraînées sur les nouvelles images.

Elle est beaucoup plus optimale lorsqu'elle est utilisée sur une collection d'images de grande taille. Dans le cas échéant, il est possible d'entraîner le réseau sans courir le risque de surapprentissage. De plus, comme les paramètres de toutes les couches (sauf de la dernière) sont initialement ceux du réseau pré entraîné, la phase d'apprentissage sera faite plus rapidement que si l'initialisation avait été aléatoire.

Stratégie 2 : L'extraction de features

Cette stratégie consiste à se servir des features du réseau pré entraîné pour représenter les images du nouveau problème. Pour cela, on retire la dernière couche fully-connected et on fixe tous les autres paramètres. Ce réseau tronqué va ainsi calculer la représentation de chaque image en entrée à partir des features déjà apprises lors du pré entraînement. On entraîne alors un classeur, initialisé aléatoirement, sur ces représentations pour résoudre le nouveau problème.

Elle doit être utilisée lorsque la nouvelle collection d'images est petite et similaire aux images de pré entraînement. En effet, entraîner le réseau sur petite quantité d'images est dangereux puisque le risque d'overfitting (surapprentissage) est important.

Stratégie 3 : Le fine-tuning partiel

Cette stratégie est la combinaison des deux précédentes permettant ainsi d'avoir un équilibre si l'on se retrouve dans les deux cas de situations présentées ci-dessus.

Ainsi, en fonction du cas d'étude et des capacités de calcul à disposition, on peut décider de partir sur la mise en place d'un réseau CNN classique ou utiliser le transfer learning en adoptant une des stratégies qu'il offre.

Dans le cadre de cette étude, la stratégie de transfer learning qui sera implémentée est le **fine-tuning total** qui contrairement aux autres stratégies permet de s'assurer que le réseau apprenne les caractéristiques liées aux real et fake faces

pour une meilleure efficacité. Il est intéressant de noter que s'il était question d'une classification d'objets, l'intuition la plus correcte aurait été d'utiliser les stratégies 2 ou 3 pour avoir de meilleurs résultats.

2.2.5.1 Transfer learning en utilisant le réseau VGG-16

VGG-16 est une version de réseau de neurones convolutif très connu (également sous le nom de VGG-Net). Il est constitué de plusieurs couches, dont 13 couches de convolution et 3 fully-connected. Il apprend l'ensemble des poids des 16 couches. Dans sa première implémentation, il a été utilisé et entraîné sur la base de données ImageNet composée de 1000 classes d'images(chien, chat, etc...). Ainsi, il prend en entrée une image en couleurs de taille 224*224 px et la classifie dans une des 1000 classes renvoyant ainsi un vecteur de taille 1000 qui contient les probabilités d'appartenance à chacune des classes. Dans notre implémentation son vecteur de sortie aura une taille de 2.[8]

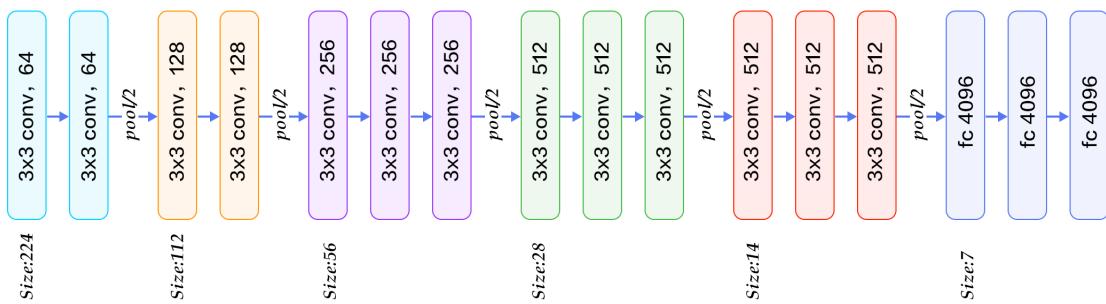


Figure 2.2 – Architecture de VGG-16

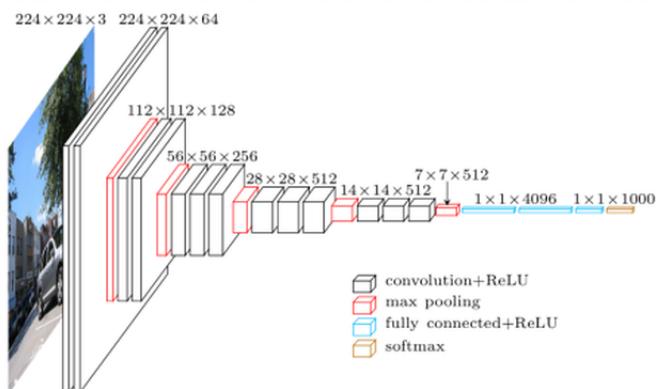


Figure 2.3 – Architecture de VGG-16 en représentation 3D

Sur cette architecture, il faut savoir que :

- les couches de convolution utilisent des filtres en couleurs de taille 3×3 px, déplacés avec un pas de 1 pixel. Un paramètre de biais est associé au produit de convolution pour chaque filtre. Aussi, la fonction d'activation des couches de convolution est une ReLU ;
- l'opération de pooling est réalisée avec des cellules de taille 2×2 px ;
- les deux premières couches fully-connected sont suivies d'une couche ReLU et calculent chacune un vecteur de taille 4096. La dernière renverra un vecteur de probabilités de taille 2 (nombre de classes de notre étude) en appliquant la fonction softmax. Un paramètre de biais est également associé pour chaque élément de leur vecteur en sortie.

2.2.5.2 Transfer learning en utilisant le réseau MobileNetV2

Ce réseau de neurones a été spécialement conçu pour les appareils mobiles compte tenu des faibles puissances dont ils sont dotés. Il s'agissait en effet de fournir un réseau de neurones CNN capable d'être utilisé sur un CPU n'ayant pas une grande puissance et ceci dans le but d'avoir des réseaux qui demeureront néanmoins optimaux.

L'architecture de MobileNetV2 se présente comme suit :

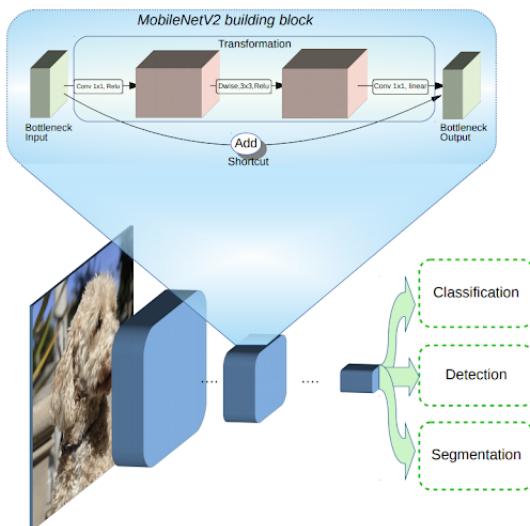


Figure 2.4 – Architecture de MobileNetV2 [7]

MobileNetV2 est une architecture de réseau neuronal à convolution qui cherche à être performante sur les appareils mobiles. Elle est basée sur une structure résiduelle inversée où les connexions résiduelles se trouvent entre les couches du goulot d'étranglement. La couche d'expansion intermédiaire utilise des convolutions légères en profondeur pour filtrer les caractéristiques comme source de non-linéarité. Dans l'ensemble, l'architecture de MobileNetV2 contient la couche initiale à convolution complète avec 32 filtres, suivie de 19 couches résiduelles de goulot d'étranglement.

Mesure et Analyse des résultats

Ce chapitre présentera les résultats de chacun des CNN que nous avons mis en place dans notre étude, une première partie se concentrera sur les paramètres choisis pour l'apprentissage de notre réseau, une seconde analysera et comparera les résultats.

3.1 Mesure des performances

Cette partie a pour but d'expliquer dans quelles conditions les apprentissages et les évaluations des CNN de cette étude ont été réalisés. Pour chaque CNN, il sera précisé avec quels paramètres il a été entraîné. Tous les CNN utilisent dans notre cas la "cross entropy" comme fonction de perte pour améliorer le réseau, et les mesures de performance sont la mesure de la précision du réseau et la cross-entropy.

3.1.1 CNN basique

Dans le CNN de base, les données ont été formatées sous la forme de batch de 26 images, ce qui signifie que la rétropropagation n'aura lieu que lorsque le réseau aura traité 26 images. Cela permet notamment d'éviter une correction des poids qui serait trop liée à une image, mais permet de la moyenner.

Le réseau de neurones utilise la fonction d'activation de type leakyRelu pour les couches intermédiaires, le paramètre "a" qui gère la pente de la fonction dans les négatifs comme présenté figure 2.1 est de 0.1, car il semble le plus adapté après plusieurs essais, en promettant une pente assez douce.

Sur les couches de convolution, la fonction Relu est utilisée comme fonction d'activation.

Le réseau est testé avec deux types d'optimiseurs, adam et sgd, afin de déterminer lequel des deux semble le plus adapté dans ce cas d'étude. Les optimiseurs

sont des versions de la descente du gradient qui modifie légèrement son fonctionnement, sgd permet l'application par paquet de données, et adam ajoute la variance des poids dans sa correction. Chaque optimiseur peut avoir ses caractéristiques, adam est notamment réputé pour être rapide et efficace, sgd pour être plus lent, mais plus robuste. C'est pourquoi un test avec les deux a été effectué.

Le CNN sera testé à chaque fois sur 20 époques qui semblent suffisantes pour remarquer la tendance.

3.1.2 CNN avec dropouts

Le CNN avec dropouts reprend les mêmes paramètres que le CNN basique, à la différence qu'il ajoute des dropouts à 2% de probabilité, ce qui signifie que sur chaque couche de dropout, 20% neurones sont désactivés aléatoirement.

Ce réseau a aussi été testé sur 20 époques.

3.1.3 CNN avec dropouts et optimisation L2

Dans ce CNN, les paramètres sont les mêmes que pour le CNN avec dropouts. Cependant, l'optimisation de type L2 disposera d'un coefficient d'application de 0.0001 permettant une application équilibrée qui ne risque pas de trop biaiser la correction des poids lors de la rétropropagation.

3.1.4 CNN basé sur l'article

Dans le CNN basé sur l'article, les paramètres étaient donc imposés par l'article. L'optimiseur est donc adam. Le réseau est entraîné sur 10 époques avec des batchs de 64 images, sans aucun dropouts. La fonction leakyRelu propose cette fois un coefficient de pente à 0.3 et est utilisée en plus comme fonction d'activation sur les couches de convolution à la place de Relu dans les cas précédent.

Sur les deux couches intermédiaires du réseau de neurones, des optimisations de type L2 sont mises en place avec un coefficient de 0.0005 légèrement plus élevé que

précédemment.

3.1.5 CNN basé sur le transfer learning

Les choix de paramètres fait sur les deux exemples de réseaux CNN basé sur le transfert de connaissance présenté cette étude sont :

- le choix de l'optimiseur ;
- du batch size ;
- des tailles des images sur lesquelles les réseaux seront entraînés qui sont différentes suivant le réseau ;
- et la valeur de l'époque ou du nombre de générations qui peut influencer le résultat et conduire au sur apprentissage quand celle-ci est trop grande.

En ce qui concerne le choix de l'optimiseur, nous avons décidé de le porter sur les optimiseurs SGD (Stochastic gradient descent) et Adam qui est une extension du premier avec un taux d'apprentissage fixé à 0.0001. Le batch size quant à lui a été fixé à 32 images. Pour le nombre d'époques, l'étude a été limitée à 10. Le choix du paramètre taille des images d'entrées sera précisé dans les sous-sections de chaque réseau notamment parce qu'il ne dépend pas d'un choix aléatoire, mais plutôt d'une condition fixée pour faire du transfer learning dans le cas de ces réseaux.

3.1.5.1 Transfer learning de VGG-16

Dans le cas d'un réseau VGG-16, la taille des images à l'entrée (input_shape) est de (224,224).

3.1.5.2 Transfer learning de MobileNetV2

Pour ce qui est du réseau MobileNetV2 la taille du input_shape doit être de (160,160).

3.2 Analyse et comparaison des résultats

3.2.1 CNN basique

Pour le CNN basique, on remarque que l'apprentissage fonctionne légèrement et atteint à peu près les 60%, cependant la validation ne suit pas du tout l'apprentissage constaté à l'entraînement, en effet celle-ci reste à peu près constante aux alentours de 53%.

Les résultats sur la fonction de perte (cross entropy) montrent le même comportement, une entropie qui diminue bien pendant l'apprentissage, mais qui stagne lors de la validation.

Ces résultats, visibles figure 3.1 et 3.2 indiquent un surapprentissage du réseau, puisque la validation montre que le modèle que le réseau a adapté ne permet pas d'obtenir des résultats corrects lors de la validation.

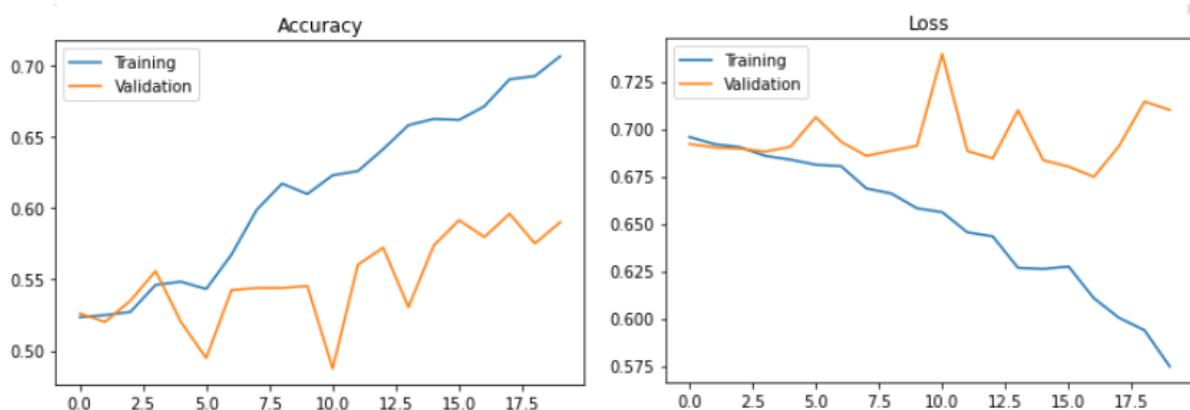


Figure 3.1 – CNN basic avec sgd

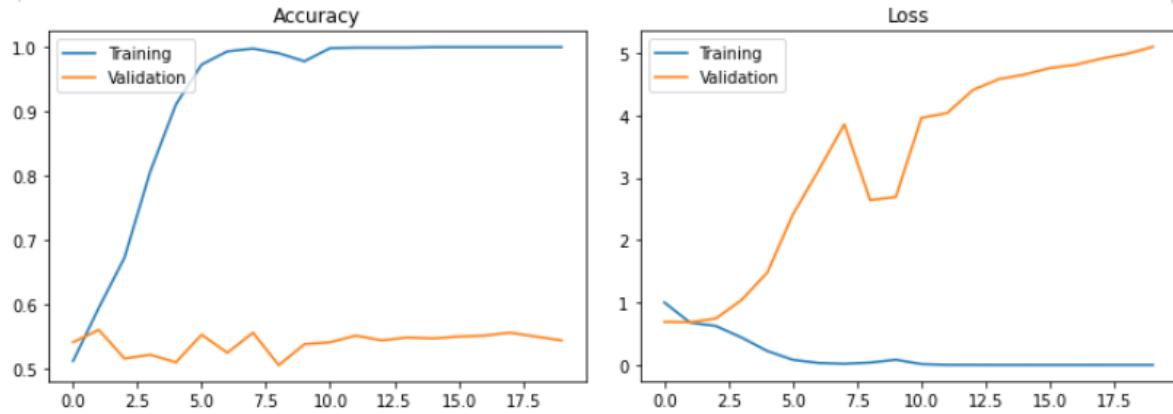


Figure 3.2 – CNN basic avec adam

On remarque qu'adam permet d'obtenir rapidement une meilleure précision à l'entraînement, mais que la validation ne suit pas du tout la tendance, il en est de même pour la perte.

Du côté de sgd, la précision de l'apprentissage est plus faible, mais la courbe de validation suis la même tendance, avec toujours un décalage cependant.

3.2.2 CNN avec dropouts

Sur l'architecture avec les dropout, avec les deux optimiseurs on remarque une très légère amélioration sur les courbes. Cependant, il existe encore un gros sur apprentissage comme le montre l'écart qui se creuse entre les courbes.

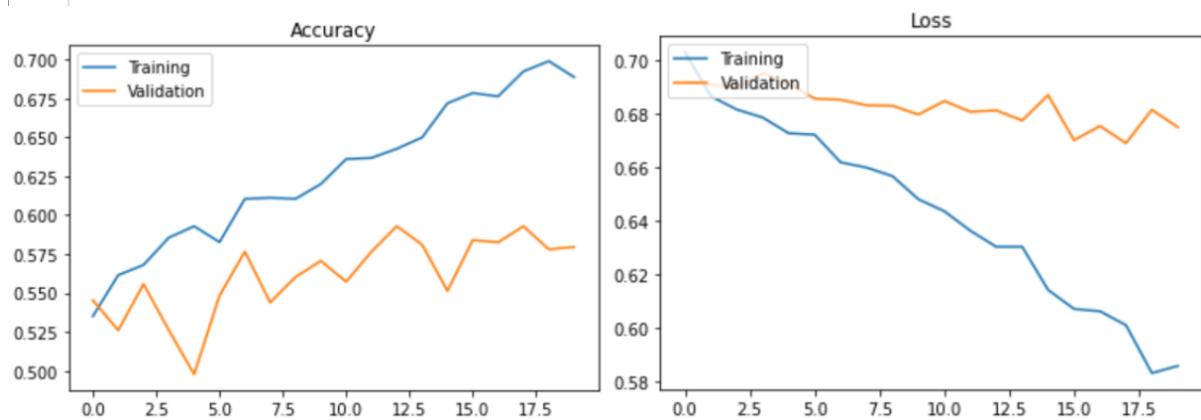


Figure 3.3 – CNN avec dropouts et sgd

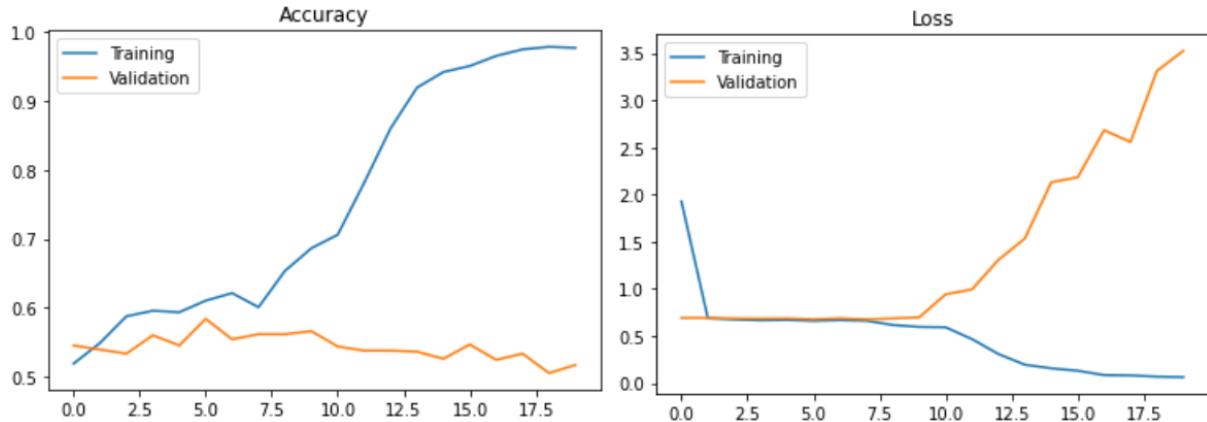


Figure 3.4 – CNN avec dropouts et adam

Ce CNN permet donc de confirmer que le dropout peut parfois aider à sortir d'un surapprentissage lors de la correction des poids, mais que ce n'est pas une solution parfaite, car un sur apprentissage persiste. Le problème peut donc provenir d'autres paramètres.

3.2.3 CNN avec dropouts et régulation L2

En rajoutant l'optimisation de type L2 sur la couche intermédiaire du CNN avec les dropouts, on remarque une amélioration visible 3.5. La couche L2 semble permettre à la validation de validée l'apprentissage en proposant des courbes très similaires sur les premières époques, mais encore un écart sur les dernières époques, toujours signe d'un surapprentissage. Comme le montre la courbe, le meilleur taux de précision atteint par le CNN en validation est de 60%.

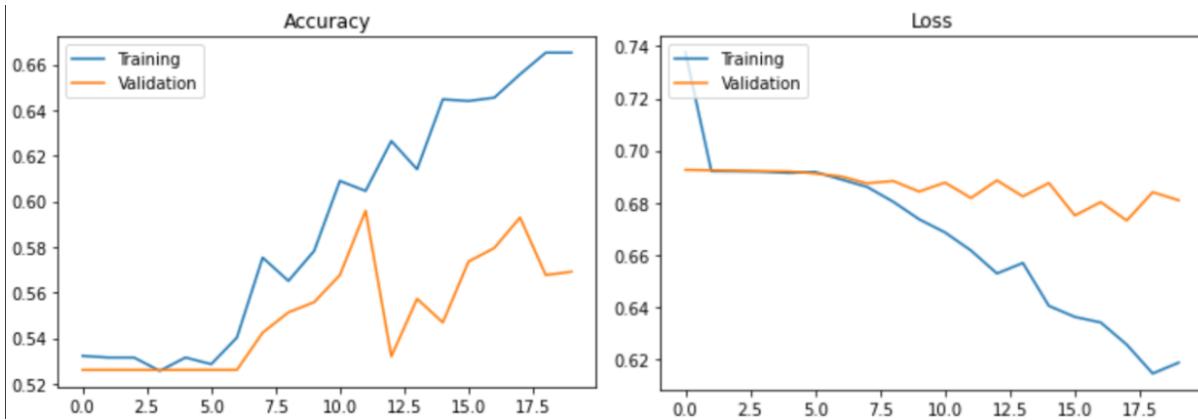


Figure 3.5 – CNN avec dropouts, L2 et sgd

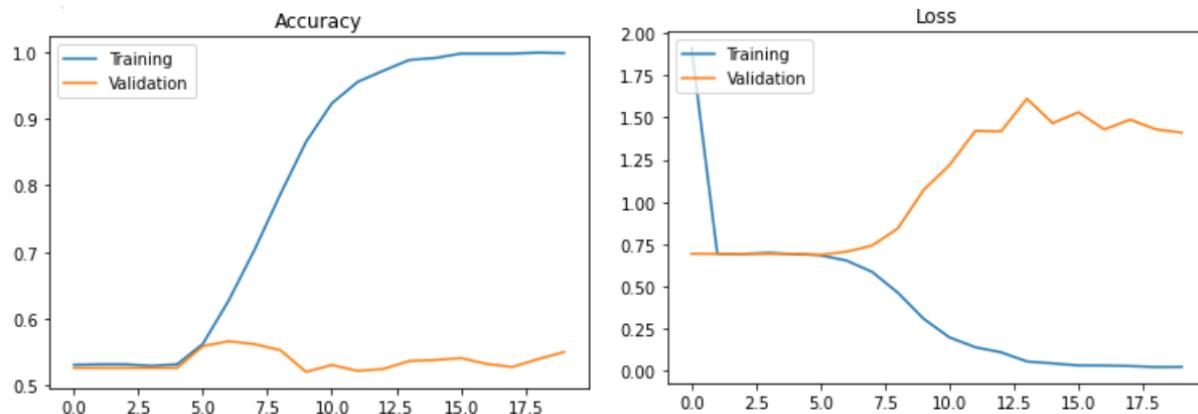


Figure 3.6 – CNN avec dropouts, L2 et adam

Ces courbes (figure 3.5) qui semblent se dégrader après un certain nombre d'époques peuvent sous-entendre une quantité trop faible de données dans le dataset. En effet, le modèle étant correct sur un certain nombre d'époques mais ce dégradant seulement pour la validation ensuite indique que le réseau commence à pratiqué le sur apprentissage.

Une manière d'améliorer la précision sur les dernières époques serait donc d'utiliser un dataset plus grand. Un deuxième test a donc été réalisé avec le dataset augmenté contenant le double des images de base.

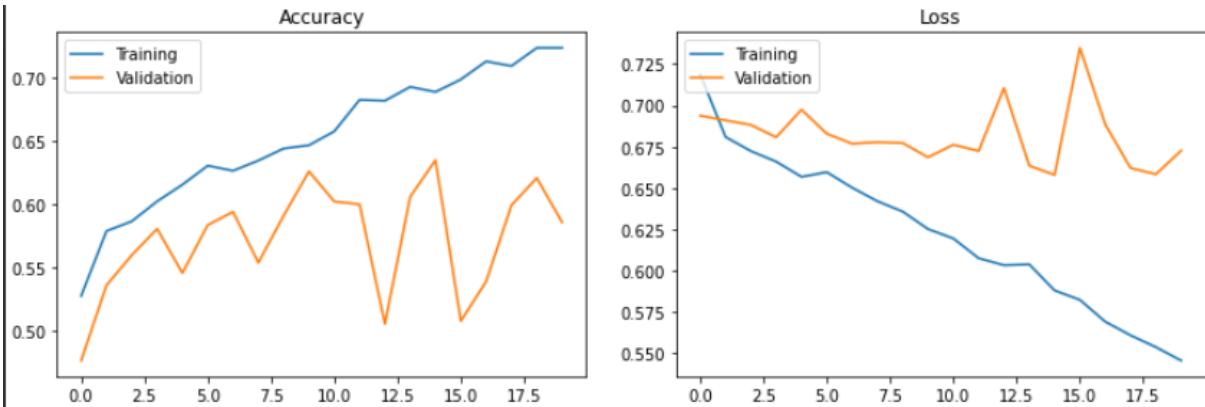


Figure 3.7 – CNN avec dropouts, L2, sgd et dataset augmenté

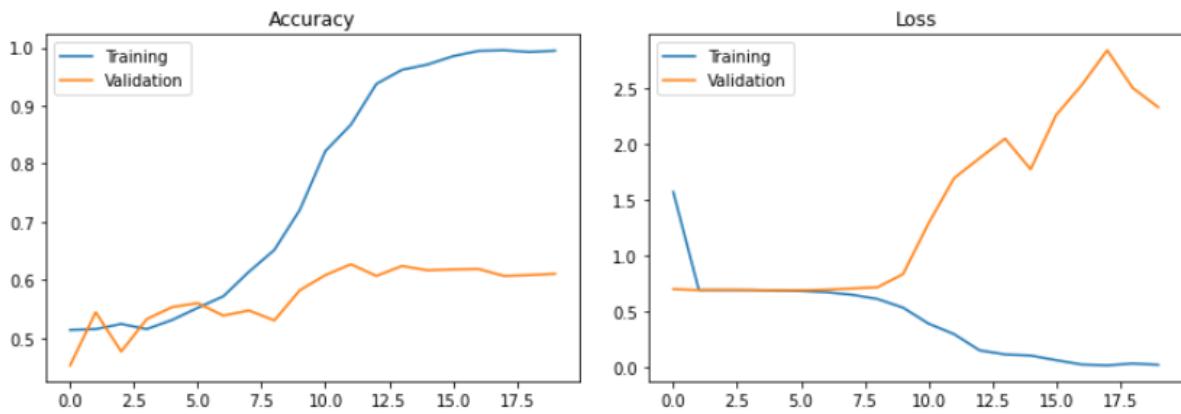


Figure 3.8 – CNN avec dropouts, L2, adam et dataset augmenté

Les résultats, disponible figures 3.7 et 3.8 montre une légère amélioration de la précision de l'algorithme avec sgd, qui permet à présent d'atteindre les 63% de précision, cela reste cependant assez faible. De plus la cross entropy reste assez élevé avec 67% environ.

3.2.4 CNN basé sur l'article

En appliquant le CNN présenté par l'article utilisé comme référence, les résultats sont très similaires au CNN précédent comme le montre la figure 3.9.

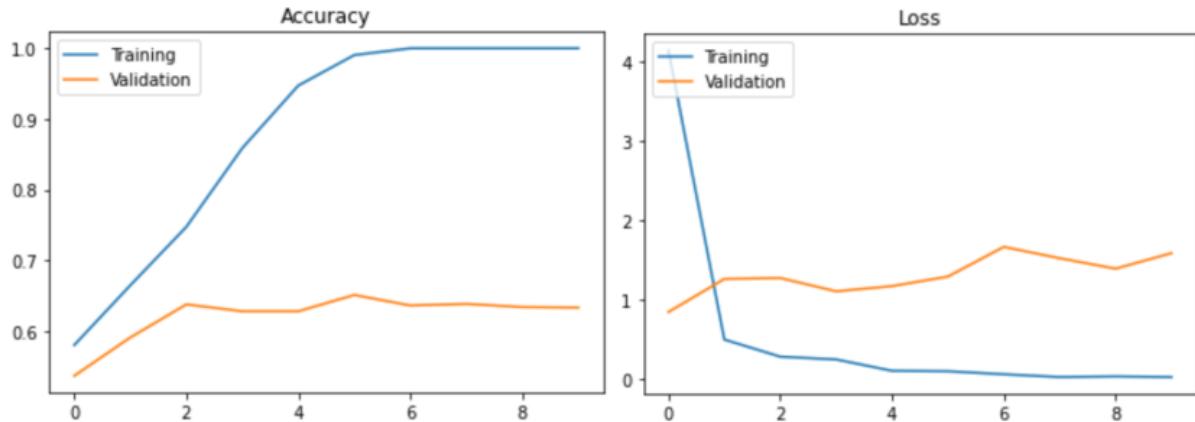


Figure 3.9 – CNN de l'article

La différence majeure entre le cas de l'article et celui de notre étude étant la taille du dataset renforce l'idée que le dataset utilisé ici ne contient pas assez de donnée ce qui empêche un apprentissage complet sans surapprentissage.

3.2.5 CNN basé sur le transfer learning

Les résultats obtenus sur la phase d'entraînement et de validation sont présentés sous la forme de courbe montrant l'évolution de la précision et de la perte notées au cours de son exécution. Ils ont été obtenus avec un nombre d'époques fixé à 10 et avec deux types d'optimiseurs qui sont SGD et Adam.

3.2.5.1 Transfer learning de VGG-16

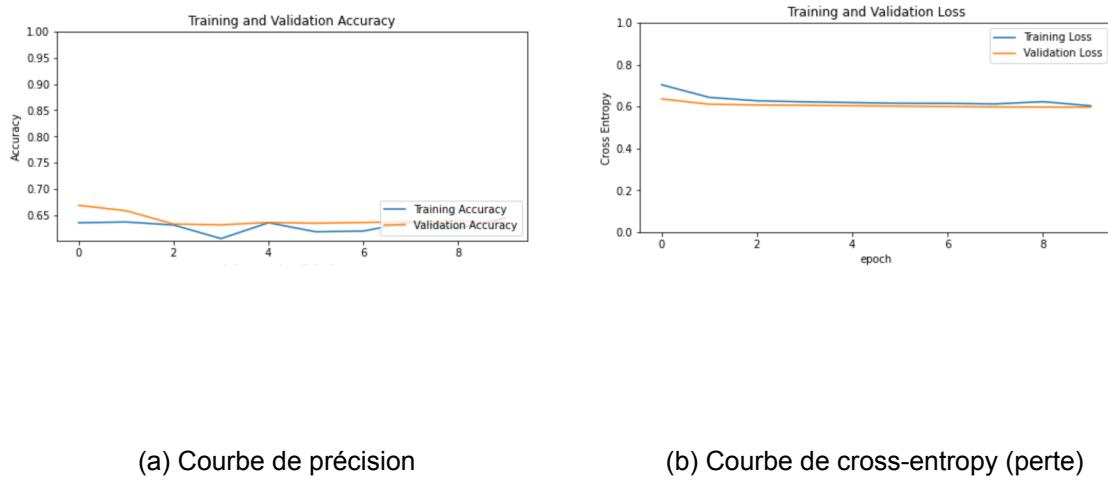
Sur la base des différentes courbes présentes sur les figures 3.10 et 3.12, on peut remarquer qu'en termes de précisons avec l'optimiseur Adam, la validation oscille d'une époque à l'autre en ayant un pourcentage entre 60% et 70% contrairement au pourcentage de la phase d'entraînement qui culmine à plus de 70% au bout des 10 époques tandis que pour l'optimiseur SGD, le la précision avoisine les 65% pour les deux phases.

Pour ce qui est de la perte, avec SGD, on a l'impression que pour les deux phases elles sont équivalentes, mais la figure 3.11 montre que la perte au cours de la validation est légèrement plus basse que celle au cours de l'entraînement (différence

allant de 4% à 2%). Tandis qu'avec Adam cette différence est visible.

Résultats obtenus en utilisant VGG-16

Optimiseur SGD



(a) Courbe de précision

(b) Courbe de cross-entropy (perte)

Figure 3.10 – Transfer learning : Résultats de VGG-16 avec SGD

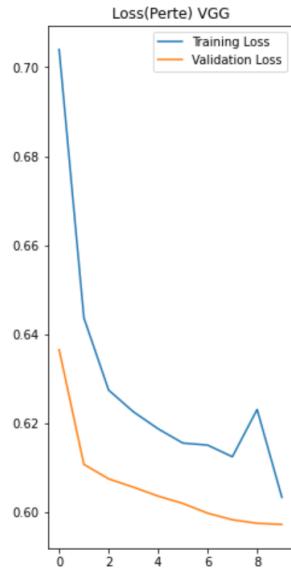
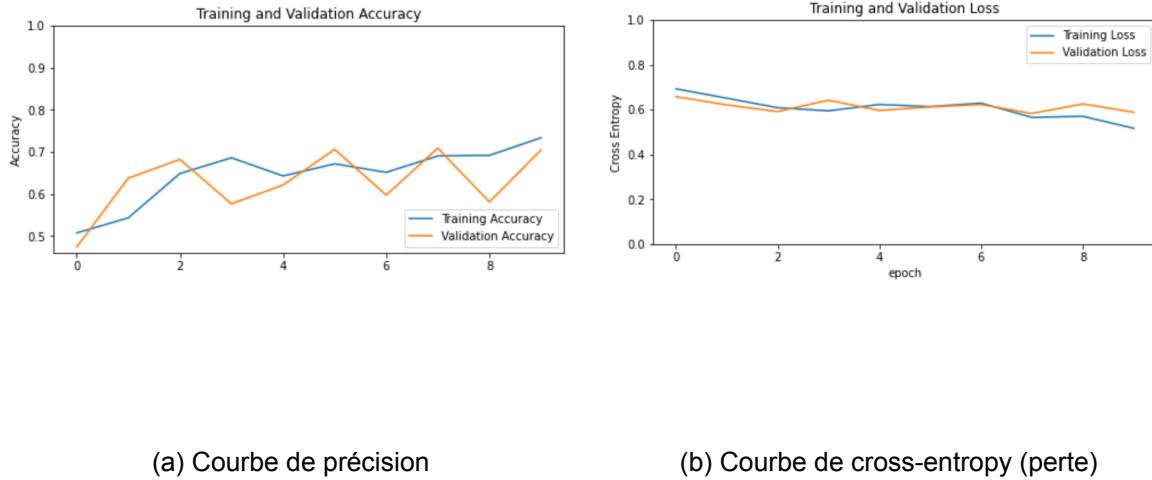


Figure 3.11 – Autre vue de la perte avec SGD

Optimiseur Adam



(a) Courbe de précision

(b) Courbe de cross-entropy (perte)

Figure 3.12 – Transfer learning : Résultats de VGG-16 avec Adam

3.2.5.2 Transfer learning de MobileNetV2

On peut voir sur les figures 3.14 et 3.13 les courbes de précision et de perte (cross-entropy) des deux optimiseurs que nous avons décidé d'utiliser pour voir les performances du transfer learning ayant modèle de réseau le MobilenetV2. Sur ces courbes, on peut voir qu'en termes de précision en utilisant l'optimiseur SGD, la phase de validation présente un léger avantage à la phase d'entraînement. Mais cet avantage est assez minime, car les deux avoisinent les 70% pour la précision tout au long des 10 époques. Quant à l'optimiseur Adam, sa précision sur les deux phases est plutôt dans une phase ascendante passant de 50% dans les premières générations jusqu'à atteindre les 70% de précision à la fin des 10 époques pour ce qui de la validation. Notons que ce pourcentage reste supérieur à celui obtenu à la phase d'entraînement qui reste au-dessous de 60%.

En ce qui concerne le cross-entropy avec Adam, la perte obtenue lors de la validation est inférieure à celle de l'entraînement avec une différence sur la courbe assez visible alors qu'avec SGD, les deux courbes s'entrelacent presque.

On peut donc dire qu'avec le transfert de connaissance faite avec MobilenetV2,

l'optimiseur Adam présente l'avantage de donner un résultat positif à son homologue SGD.

Optimiseur SGD

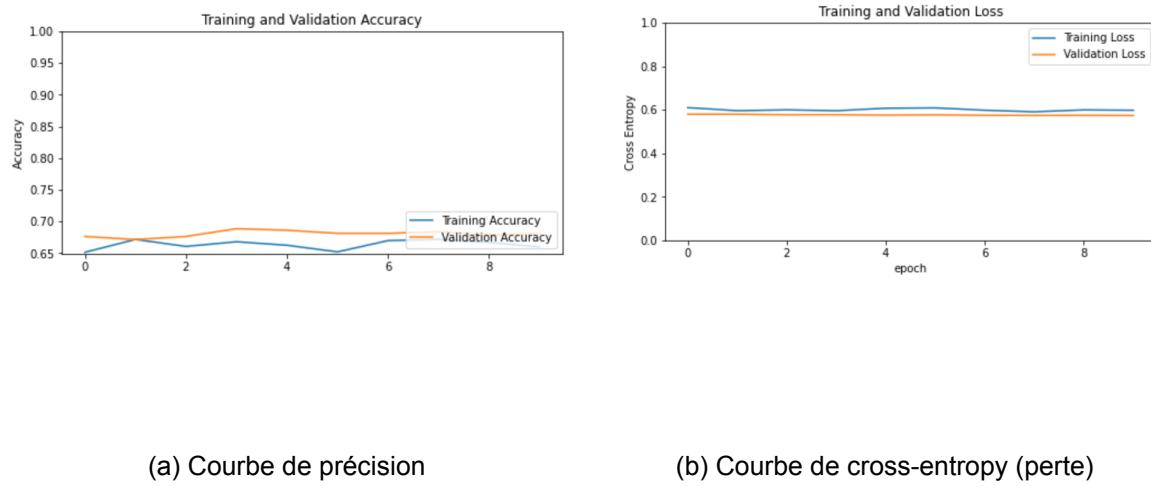


Figure 3.13 – Transfer learning : Résultats de MobileNetV2 avec SGD

Optimiseur Adam

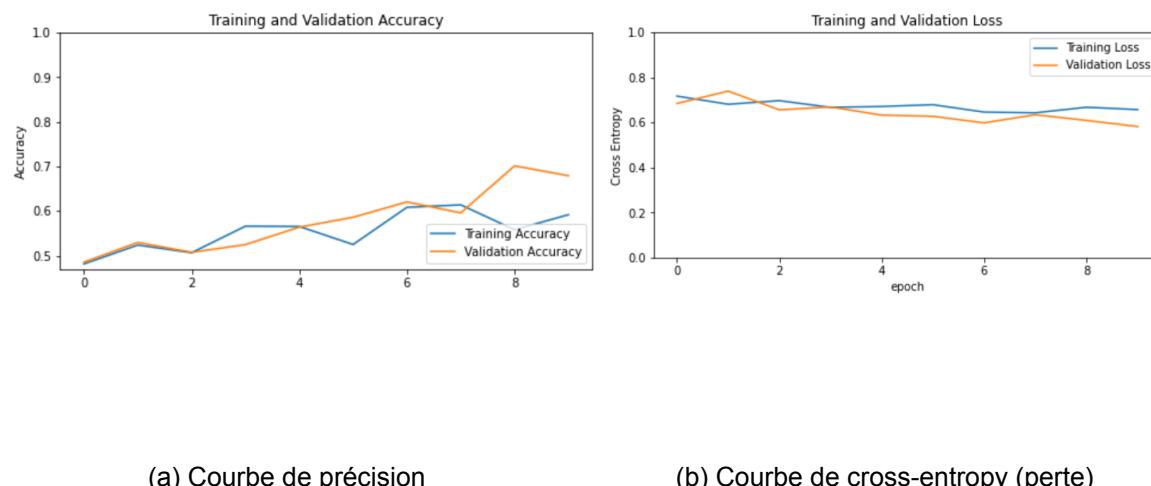


Figure 3.14 – Transfer learning : Résultats de MobileNetV2 avec Adam

Après les tests avec ces différentes architectures, et après analyse des résultats il apparaît évident que le dataset est une limite très forte pour la réalisation du projet, car il aurait fallu qu'il comporte beaucoup plus de données pour pouvoir entraîner complètement notre réseau. Le transfert learning permet cependant d'améliorer les résultats et d'obtenir un classeur un minimum fonctionnel. Il est à noter que l'utilisation de dataset plus grand était difficile dans le cadre de cette étude étant donnée la puissance à disposition qui implique des temps de traitement très long.

Conclusion

À travers ce projet, plusieurs notions ont été découvertes. Premièrement, ce projet permet de comprendre l'utilité du deep learning qui est très utile dans le cadre de traitement de données complexes où les caractéristiques ne sont pas facilement visibles. Plus particulièrement, il permet de comprendre le fonctionnement d'un CNN tant dans le prétraitement des données que dans la méthode de classification. Le transfert learning est une notion qui vient compléter les réseaux de type CNN en permettant d'utiliser des parties de réseau déjà entraîné pour améliorer son réseau sans trop de puissance. Cette notion s'intègre parfaitement dans ce projet, car elle permet de mettre en avant les côtés négatifs du deep learning, qui sont la quantité de paramètres à définir à travers de nombreux test, la puissance et la quantité de données qu'il peut nécessiter pour créer un classeur fonctionnel.

Ces limites se sont aussi retrouvées dans les résultats des différentes implémentations qui ont pu être réalisées. Ces différentes implémentations ont permis de comprendre quelques techniques d'optimisation permettant de lutter contre les points faible du deep learning, mais il est finalement assez visible que même avec tous ces outils il n'est pas toujours simple de trouver le bon réglage pour obtenir un classeur fonctionnel. La quantité de données pour l'apprentissage est aussi une problématique bien mise en avant par ce projet, car elle est au centre des problèmes rencontrés dans chacune des implémentations réalisées.

Pour conclure, il semble bien que le deep learning propose une méthode puissante pour le traitement de données, mais elle reste cependant très spécifique et demande beaucoup de temps de configuration. Pour aller plus loin, il pourrait être intéressant de comparer les CNN à une autre méthode de classification d'image afin de comparer les résultats de ces deux méthodes en fonction du temps nécessaire à leurs mises en place.

Bibliographie

- [1] A. Géron (2017) . Deep learning avec tensorflow : mise en oeuvre et cas concrets. Dunod.
- [2] A. Géron et H. Soulard (2020) . Deep learning avec avec keras et tensorflow : mise en oeuvre et cas concrets. Dunod.
- [3] Alexandre Bertin . Intelligence artificielle : la reconnaissance d'images prend de l'ampleur. <https://medium.com/>.
- [4] Anuja Nagpal . L1 and l2 regularization methods. <https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>.
- [5] Computational Intelligence and Yonsei University Photography Lab. Real and fake face detection. <https://www.kaggle.com/ciplab/real-and-fake-face-detection>.
- [6] cs231n . Convolutional neural networks (cnns / convnets). <https://cs231n.github.io/convolutional-networks/>.
- [7] Mark Sandler et Andrew Howard . Mobilenetv2 : The next generation of on-device computer vision networks. <https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html>.
- [8] Pascal Monasse et Kimia Ndjahi . Classez et segmentez les données visuelles. <https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn>.

Annexe

A.1 Liens vers les googles collab

A ce lien vous trouverez le dataset utilisé et les fichiers google collab contenant notre code : <https://colab.research.google.com/drive/1eCHKah7WmLx4K-XBaFEI9DXPEKBCTRfk?usp=sharing>

A ce lien vous trouverez le code pour le transfert learning : <https://colab.research.google.com/drive/1akV93v5yM1mCt7YQ3M8sYaoNKIz5jQkp?usp=sharing>

A.2 Article de référence

A la suite vous trouverez l'article à partir duquel nous avons implémenté l'un de nos CNN.

Fake Faces Identification via Convolutional Neural Network

Huaxiao Mo

Sun Yat-sen University

Guangzhou, China

mohx5@mail2.sysu.edu.cn

Bolin Chen

Sun Yat-sen University

Guangzhou, China

chenbl8@mail2.sysu.edu.cn

Weiqi Luo*

Sun Yat-sen University

Guangzhou, China

luoeweiqi@mail.sysu.edu.cn

ABSTRACT

Generative Adversarial Network (GAN) is a prominent generative model that are widely used in various applications. Recent studies have indicated that it is possible to obtain fake face images with a high visual quality based on this novel model. If those fake faces are abused in image tampering, it would cause some potential moral, ethical and legal problems. In this paper, therefore, we first propose a Convolutional Neural Network (CNN) based method to identify fake face images generated by the current best method [20], and provide experimental evidences to show that the proposed method can achieve satisfactory results with an average accuracy over 99.4%. In addition, we provide comparative results evaluated on some variants of the proposed CNN architecture, including the high pass filter, the number of the layer groups and the activation function, to further verify the rationality of our method.

KEYWORDS

Image Forensics, Deep Learning, Generative Adversarial Networks (GAN), Convolutional Neural Network (CNN)

ACM Reference Format:

Huaxiao Mo, Bolin Chen, and Weiqi Luo. 2018. Fake Faces Identification via Convolutional Neural Network. In *IH&MMSec '18: 6th ACM Workshop on Information Hiding and Multimedia Security, June 20–22, 2018, Innsbruck, Austria*. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3206004.3206009>

1 INTRODUCTION

With the rapid development of image processing technology, modifying an image without obvious visual artifacts becomes much easier. Nowadays, seeing is no longer believing. Image forensics have attracted widely attention in the last decade, and many forensic methods based on hand-crafted features such as [4, 15, 16, 19] have been proposed until now.

Different from conventional methods based on hand-crafted features, deep learning can exploit cascaded layers to adaptively learn hierarchical representations from input data. Some novel models in deep learning such as CNN and GAN have been extensively studied and have achieved great success in many image related applications, such as image style transfer [8, 11], image super-resolution [13, 18],

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IH&MMSec'18, June 20–22, 2018, Innsbruck, Austria

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5625-1/18/06...\$15.00

<https://doi.org/10.1145/3206004.3206009>

image inpainting [10, 22] and image steganalysis [6, 21]. Up to now, several deep learning based works have been proposed for image forensics. For instance, Chen et al. [5] proposed a median filtering forensic method based on CNN; Bayar et al. [2] proposed a new CNN architecture to detect several typical image manipulations; Rao et al. [17] proposed a CNN based method to detect image splicing and copy-move; Choi et al. [7] proposed a CNN based method to detect composite forgery detection. Recently, some studies have shown that we can obtain fake face images with high visual quality (refer to Section 2 for details) based on GAN model. Since these fake face images can successfully cheat our eyes, identifying fake images becomes an very important issue in image forensics. In this paper, we propose a CNN based method to identify the fake images generated by the work [20]. In our method, we carefully design the CNN architecture, with particular attention to the high pass filter for the input image, the number of layer groups and the activation function, and then provide extensive experimental results to show the effectiveness and rationality of the proposed method. To the best of our knowledge, this is the first work to investigate this forensic problem.

The rest of the paper is organized as follows. Section 2 describes two recent GAN based works on face generation. Section 3 gives the proposed detection method based on CNN. Section 4 shows experimental results and discussions. Finally, the concluding remarks of this paper and the future works are given in Section 5.

2 FACE GENERATION WITH GAN

Generative adversarial networks (GAN) [9] is a prominent generative model, which learns the distribution form high-dimension data and produces novel sample. Typically, a GAN contains two parts: a generator and a discriminator. The generator learns to create fake data indistinguishable from the real data, while the discriminator learns to determine whether the input data is real or fake. They contest against each other during training until the generator can produce high quality fake data.

Recently, several GAN based methods are proposed to generate high quality fake face images. For instance, in [3], Berthelot et al. proposed a novel equilibrium method for balancing the two parts of GAN to generate visually pleasing face images. However, this method can only produce fake face images in low resolutions, such as 256×256 . In [20], Karras et al. proposed a progressive strategy to construct and train GAN for generating high quality images. The progressive strategy is illustrated in Fig. 1. Instead of training the whole GAN on high-resolution image, they first construct a simple GAN training on low-resolution images in the beginning, and then gradually add more layers to adapt the model to high-resolution images during the training stage. Based on the experiments, most fake face images (1024×1024) generated by this method are difficult to identify with the naked eye, as illustrated

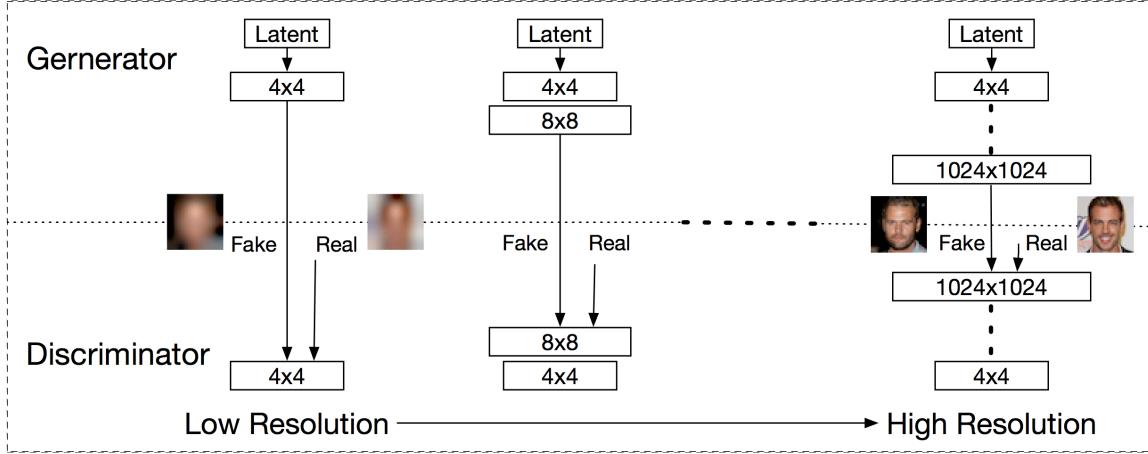


Figure 1: The progressive training strategy employed in [20]. Here $N \times N$ refers to layers operating on images of $N \times N$ resolution.

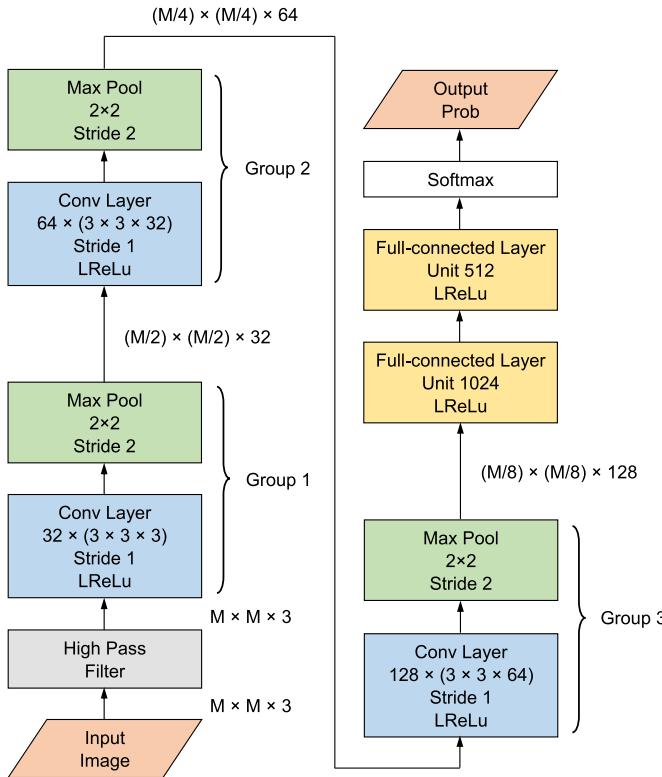


Figure 2: The proposed architecture

in the first row of Fig.8. However, some poorer results are also obtained using this method, as illustrated in the second row of Fig. 8. In this paper, we first propose a method to identify those good fake face images generated using the method [20].

3 THE PROPOSED DETECTION METHOD

Since the generator and discriminator employed in [20] are mainly based on CNN, it is natural to use a CNN based method to detect the resulting fake face images. To this end, we carefully design the architecture of the proposed CNN model, as illustrated in Fig. 2. The model input is an RGB color image with size $M \times M \times 3$. Since the contents of fake and true face images are quite similar, it is expected that the main difference between the two kinds of images would be reflected on the residual domain according to the previous research [14]. Therefore, we first transform the input images into residuals using a high pass filter. The resulting residuals are then fed into three layer groups. Each group includes a convolutional layer (3×3 size, 1×1 stride) equipped with LReLU and a max pooling layer (2×2 size, 2×2 stride). The output feature map number of the convolutional layer in the first group is 32, while that of the other convolutional layers is twice the corresponding input feature map number. The output feature maps of the last group are then aggregated and fed into two fully-connected layers. They both equipped with LReLU and consist of 1024, 512 units, respectively. Finally, the softmax layer is used to produce the output probability.

In our experiments, we implement the proposed CNN model using Tensorflow [1] and train it using Adam [12] with a learning rate of 0.0001. All the weights are initialized using a truncated Gaussian distribution with zero mean and standard deviation of 0.01. The biases is initialized as zero. L2 regularization is enabled in the fully-connected layer with the λ of 0.0005. In the training stage, we use a batch size of 64 and train the proposed CNN for 20 epochs. In addition, we shuffle the training data between epochs.

4 EXPERIMENTAL RESULTS

In this section, we first describe the image data set used in our experiments. Then we present some experiments to show the effectiveness of the proposed method in identifying fake face images. In addition, we conduct extensive experiments to show the rationality of the proposed model.

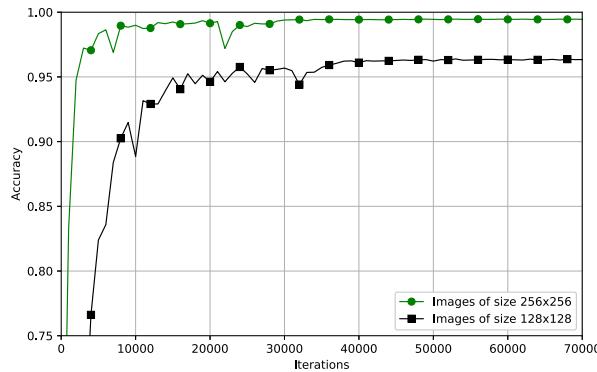


Figure 3: Comparison of different image sizes

4.1 Image Data Set

In our experiments, we use 30,000 true face images from CELEBA-HQ dataset and select 30,000 fake ones with good visual quality from the fake face image database¹ generated by [20]. All images are with 1024×1024 and stored in PNG format. In our experiments, we resize all images into 256×256 using bilinear interpolation, and compress them using lossy JPEG compression with a quality factor of 95. Finally, we divide the resulting images into training set (12,000 pairs of true-fake face images), validation set (3,000 pairs) and test set (15,000 pairs). To achieve convincing results, we randomly split the training, validation, and test set three times and report the average results in the following experiments.

4.2 Fake Face Identification

In this section, we aim to identify whether a given face image is real or generated one. As shown in the blue box in Fig. 9, we found that some background regions in some fake face images looks unnatural, which may help increase the detection performance. To reduce the influence of image backgrounds, we crop a small segment (128×128) from every image in the original image set (256×256), and ensure that each cropped segment mainly includes some facial key-points (such as eyes, nose, and mouth), as illustrated in red box in Fig. 9. Finally, we obtain two different image data set for experiments, that is original ones including face and background, and the cropped ones just including main facial region.

The experimental results evaluated on the two validation set are shown in Fig. 3. From Fig. 3, we observe that during training stage, the proposed model on both original images (green line) and cropped images (black line) can converge within 70,000 iterations, and both detection accuracies would be over 95% after 40,000 iterations. For a more convincing result, we evaluate the trained model on the test set, and obtain accuracies of over 99.4% and 96.3% on the original images and cropped images respectively, which means that we can still obtain satisfactory results after removing unnatural parts in the background.

¹Available at: <https://drive.google.com/drive/folders/0B4qLcYyJmiz0TXY1NG02bzZVRGs>

4.3 Comparison of Variants of the Proposed Model

In this section, we present some results to validate the rationality of the proposed model in Fig. 2. Three parts of our model are considered, including the high pass filter, the number of the layer groups and the activation function. The corresponding results evaluated on the validation set are shown in the following subsections.

4.3.1 High Pass Filter: In this experiment, three following high pass filters are evaluated in the model.

$$\begin{array}{lll} \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{bmatrix} & \begin{bmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{bmatrix} \\ \text{(a) filter A} & \text{(b) filter B} & \text{(c) filter C} \end{array}$$

Figure 4: Three high pass filters

The corresponding results are shown in Fig. 5. From Fig 5, we observe the proposed model (i.e. using the filter B) can achieve highest accuracy among the three test filters. We also observe that the model using the filter A can achieve similar results with our proposed model, while the model with filter C has the lowest detection accuracy. In addition, the detection accuracy of removing the high pass filter is nearly 98%, which means that using suitable high pass filter can help improve detection performance.

4.3.2 Number of layer groups: In this experiment, we evaluate the influence of adding/removing one layer group in the proposed model. The results are shown in Fig. 6. From Fig. 6, we observe that adding one group to the proposed model does not increase the detection performance, while removing one group decreases the detection performance slightly, which means that using three layer groups in the proposed model is sufficient for the investigated problem.

4.3.3 Activation functions: Activation function is another important factor in CNN. In this experiment, six commonly used activation functions are considered in the proposed model. They are TanH, ReLu, and four variants of ReLu, including PReLU, LReLU, ELU and ReLU6. The experimental results are shown in Fig. 7. From Fig. 7, we observe that LReLU ReLU, PReLU, and ELU can achieve similar accuracy. Among six activation functions, LReLU obtains the best performance while TanH shows the worst performance.

5 CONCLUSION

In this paper, we first propose a CNN based method to identify fake face images generated with the state-of-the-art method [20], and provide extensive experimental results to show that the proposed method can effectively identify fake face images with a high visual quality from real ones. Our experimental results also indicate that even though the current GAN based methods can generate realistic looking faces (or other image objects and scenes), some obvious statistical artifacts would be inevitably introduced and can serve as evidences for fake ones.

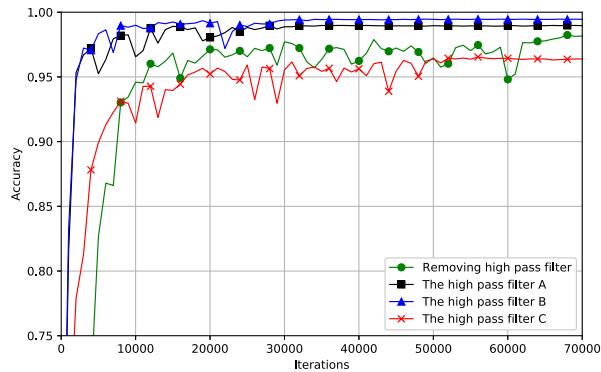


Figure 5: Comparison of using different high pass filters/without high pass filter

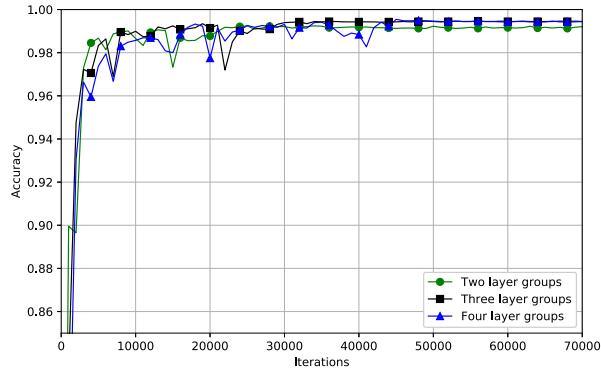


Figure 6: Comparison of different numbers of groups

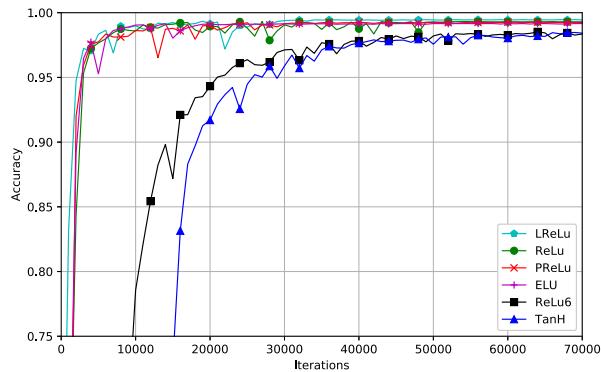


Figure 7: Comparison of different activation functions

In future, we will further investigate some inherent artifacts left by the GAN in [20] for image forensics. On the other side, we

will try to propose a wise face generation method that can avoid detection.

ACKNOWLEDGMENTS

This work is supported in part by the NSFC (61672551), the Special Research Plan of Guangdong Province under Grant 2015TQ01X365, and the Guangzhou Science and Technology Plan Project under Grant 201707010167.

REFERENCES

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
- [2] Belhassen Bayar and Matthew C Stamm. 2016. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security*. 5–10.
- [3] David Berthelot, Tom Schumm, and Luke Metz. 2017.Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717* (2017).
- [4] Gang Cao, Yao Zhao, Rongrong Ni, and Xuelong Li. 2014. Contrast enhancement-based forensics in digital images. *IEEE transactions on information forensics and security* 9, 3 (2014), 515–525.
- [5] Jiansheng Chen, Xiangui Kang, Ye Liu, and Z Jane Wang. 2015. Median filtering forensics based on convolutional neural networks. *IEEE Signal Processing Letters* 22, 11 (2015), 1849–1853.
- [6] Mo Chen, Vahid Sedighi, Mehdi Boroumand, and Jessica Fridrich. 2017. JPEG-Phase-Aware Convolutional Neural Network for Steganalysis of JPEG Images. In *ACM Workshop on Information Hiding and Multimedia Security*. 75–84.
- [7] Hak-Yeol Choi, Han-Ul Jang, Dongkyu Kim, Jeongho Son, Seung-Min Mun, Sunghie Choi, and Heung-Kyu Lee. [n. d.]. Detecting composite image manipulation based on deep neural networks. In *IEEE International Conference on Systems, Signals and Image Processing*. 1–5.
- [8] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. 2017. A learned representation for artistic style. In *Proceedings of International Conference on Learning Representations*.
- [9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
- [10] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2017. Globally and locally consistent image completion. *ACM Transactions on Graphics* 36, 4 (2017), 107:1–107:14.
- [11] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*. Springer, 694–711.
- [12] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 <http://arxiv.org/abs/1412.6980>
- [13] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4681–4690.
- [14] Haodong Li, Weiqi Luo, Xiaoqing Qiu, and Jiwu Huang. 2018. Identification of various image operations using residual-based features. *IEEE Transactions on Circuits and Systems for Video Technology* 28, 1 (2018), 31–45.
- [15] Li Li, Jianru Xue, Zhiqiang Tian, and Nanning Zheng. 2013. Moment feature based forensic detection of resampled digital images. In *Proceedings of the 21st ACM international conference on Multimedia*. ACM, 569–572.
- [16] Xiaoqing Qiu, Haodong Li, Weiqi Luo, and Jiwu Huang. 2014. A universal image forensic strategy based on steganalytic model. In *Proceedings of the 2nd ACM workshop on Information hiding and multimedia security*. ACM, 165–170.
- [17] Yuan Rao and Jiangqun Ni. 2016. A deep learning approach to detection of splicing and copy-move forgeries in images. In *IEEE International Workshop on Information Forensics and Security*. 1–6.
- [18] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. 2017. Amortised map inference for image super-resolution. In *Proceedings of International Conference on Learning Representations*.
- [19] Matthew Stamm and KJ Ray Liu. 2008. Blind forensics of contrast enhancement in digital images. In *IEEE International Conference on Image Processing*. IEEE, 3112–3115.
- [20] Samuli Laine Jaakko Lehtinen Tero Karras, Timo Aila. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *International Conference on Learning Representations* (2018). <https://openreview.net/forum?id=Hk99zCeAb> accepted as oral presentation.



Figure 8: Fake face examples from the work [20]. The first row shows examples with a good visual quality, while the second shows ones with a poor visual quality that would be removed in our experiments.

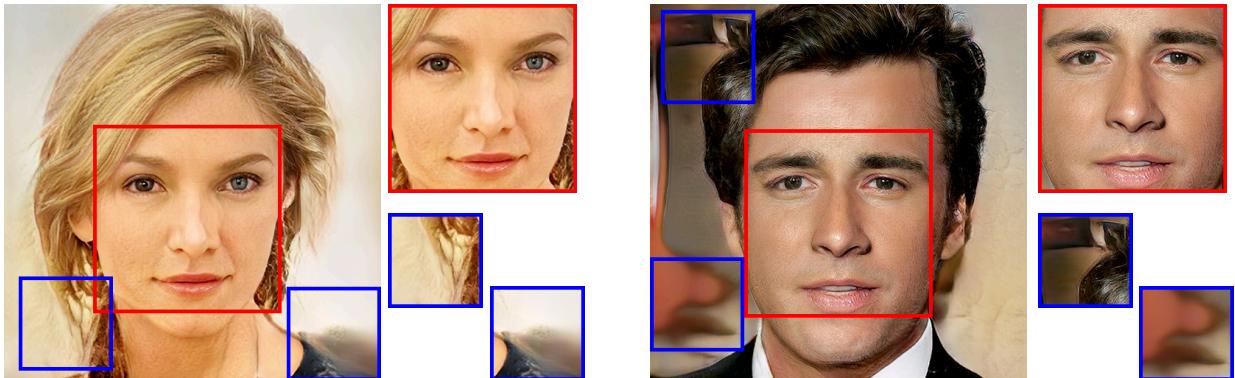


Figure 9: Fake face vs. Background. The region in the red box includes some facial key-points; while the blue ones are located at the background with poor visual artifacts.

[21] Guanshuo Xu, Han Zhou Wu, and Yun Qing Shi. 2016. Structural Design of Convolutional Neural Networks for Steganalysis. *IEEE Signal Processing Letters* 23, 5 (2016), 708–712.

[22] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. 2017. High-resolution image inpainting using multi-scale neural patch synthesis. In *The IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 1. 3.