UNIVERSITÉ DE NANTES

IAE NANTES
ÉCONOMIE & MANAGEMENT

Master 2 - Économétrie Appliquée et Statistiques
Petals to the Metal - Flower Classification on TPU
Dossier Big Data sous python

Préparé par : Sarah KERVRAN et Diana EL HAJJ SLEIMAN
Enseignant : Jeff ABRAHAMSON

Année Académique
2020-2021

Sommaire

I. Introduction

Depuis quelques années, un nouveau lexique lié à l'émergence de l'intelligence artificielle dans notre société s'est développé. Lorsqu'on parle d'intelligence artificielle, nous faisons souvent allusion aux technologies associées comme le machine learning ou le Deep learning.

Le machine learning est une technique de programmation informatique qui utilise des probabilités statistiques pour donner aux ordinateurs la capacité d'apprendre par eux-mêmes sans programmation explicite. Le machine learning n'est pas une technologie nouvelle. En effet, le premier neurone artificiel, appelé " Perceptron", a été inventé en 1958 par le psychologue américain Frank Rosenblatt.

Les chercheurs ont distingué trois catégories de machine learning. Tout d'abord, nous avons les techniques **supervisées** subdivisées en deux sous parties : la classification (où la variable de sortie est une catégorie) et la régression (où la variable de sortie est une valeur spécifique). Dans les principaux algorithmes d'apprentissage automatique supervisé nous retrouvons : **la régression logistique**, **les machines à vecteurs de supports (SVM)**, les **forêts aléatoires**, les **arbres de décision** et les **k plus proches voisins**.

La seconde classe de machine learning est celle des techniques **non supervisées**. Cette catégorie est représentée par exemple par le **clustering** et toute association qui peut être représentée avec les algorithmes suivants: les **K-means,** le regroupement hiérarchique ou encore la réduction de la dimensionnalité.
Enfin, la dernière catégorie est la méthode d'apprentissage automatique par renforcement composé de techniques comme le **Q-learning, Deep Q Network (DPQ) et SARSA**.

La base sur laquelle nous travaillons a été obtenue via le site de **Kaggle,** "https://www.kaggle.com". Plus précisément, le sujet qu'il nous avait été demandé de traiter concerne la compétition «**Petal to Metal : Flower classification on TPU** ».

Que sont les TPU ?

Un TPU est un puissant accélérateur matériel spécialisé dans les tâches d'apprentissage en profondeur.

La puce a été spécifiquement conçue pour le framework TensorFlow de Google , une bibliothèque mathématique symbolique qui est utilisée pour les applications d' apprentissage automatique telles que les réseaux de neurones. Les TPU sont utilisés notamment pour traiter de grandes bases de données d'images. Dans notre dossier nous n'utilisons pas les TPU, trop puissant à l'utilisation en local.

Le but de ce dossier est donc de construire une série de modèles d'apprentissage automatique permettant d'identifier et de catégoriser au mieux les différents types de fleurs par reconnaissance d'image.

Dans un premier temps, nous allons procéder à la présentation des données et au processus de traitement et préparation de la base. Ensuite, nous allons introduire de manière théorique les différents algorithmes d'apprentissage (**SVM, ANN, CNN**) que nous avons appliqué à nos données. Enfin, nous discuterons des résultats rendus par ces différents algorithmes sur notre base de données et nous comparerons ainsi leur taux de précision (accuracy) et leur taux d'erreur (loss) en considérant la volonté de maximiser le taux de précision et de minimiser l'erreur.

II. Présentation et traitement des données

Nous avons remarqué que les données étaient présentées sous le format nommé "**tfrecords**" c'est-à-dire sous forme de séquence binaire. Le package "**pandas_tfrecords"** nous a donc permis en particulier d'importer et de visualiser premièrement nos données tfrecords en Dataframe.

La base de données importée est constituée de 12 753 photos de fleurs sous format tfrecords associées à leurs “class” et leur “id”. Nous observons 104 types différents de fleurs à classer.

Les données étant conséquentes, nous avons calculé la médiane des effectifs de fleurs par classe pour pouvoir enlever toutes les classes de fleurs ayant une valeur inférieure à la médiane, c’est-à-dire 88. De plus, une classe de fleur n’ayant pas assez d’effectif peut-être problématique dans l’apprentissage des modèles pour la reconnaissance des images. Malgré l’enlèvement de certaines classes, les algorithmes d’apprentissage avaient un temps d’exécution encore particulièrement long. Nous avons donc, après plusieurs tentatives, décidé de ne garder que les classes avec un effectif supérieur à 190.

PCA

Nous avons tenté tout d’abord une réduction des dimensions des données via une application d’un algorithme dénommé **PCA** qui est un algorithme très performant.

Nous avons pour cela commencé par ajouter une colonne “features” à notre base de données contenant les valeurs des pixels (en 1 dimension) des images.

Le principe du PCA est de réduire les dimensions des “features” (nous sommes à 192x192x3) à des dimensions d’information beaucoup plus basses tout en gardant au maximum l’information sur nos pixels. Cette information gardée correspond au ratio de la variance expliquée.

Il est nécessaire de prendre des valeurs de pixels déjà normalisées avant toute application du PCA. Les valeurs des pixels vont de 0 à 255 ce qui correspond aux nuances de couleurs. Nous avons donc divisé toutes les données par 255 pour obtenir des valeurs comprises entre 0 et 1, des valeurs normalisées.

Nous avons ensuite utilisé la fonction **“train_test_split”** du package **“sklearn.model_selection”** pour découper notre base de donnée en deux sous base ; une servant à l'entraînement de notre modèle SVM (**70 %** de la base), l’autre utilisé pour le test (**30%** de la base).

L'algorithme de réduction PCA va avoir lieu uniquement sur la base d'entraînement (train) et non sur la base entière car la transformation par le PCA relierait mathématiquement la base d'entraînement et de test (test) si nous faisions la PCA avant le “train_test_split”.

La dimension des features étant de égale à **110 592** (**192x192x3** avec 3 pour les couleurs “Rouge”, “Vert”, “Bleu” et 192x192 dimension de l’image) nous avons tenté de réduire nos données à 200 dimensions via le PCA car les dimensions des features étaient beaucoup trop conséquentes à donner à notre modèle d'entraînement.
Une fois l’application terminée, nous remarquons que le ratio de la variance expliquée est d’environ 76% ce qui implique que le PCA à 200 dimensions garde 76% d’information sur les données.
Nous appliquons ensuite la transformation à la base d'entraînement et celle de test et entraînons un modèle SVM à noyau linéaire et à pénalité 1 sur la base d’entraînement transformée. Le taux de bonne prédiction s’élève à environ 27%.
Il est important de noter que les algorithmes d’apprentissage SVM vont accueillir des données sous forme de tableau et non sous format PNG, jpeg.

Notons que l’application de l’algorithme du PCA prend beaucoup de temps. Nous nous sommes donc rabattus sur la création d’une fonction “Résolution” ayant pour fonctionnalité de réduire la dimension de nos données de 192x192 à 50x50. Nous utilisons donc les images à dimension réduite pour la suite de nos estimations ce qui va nous permettre aussi de diminuer drastiquement le temps d’exécution des modèles.
En effet, plus il y a d’image et de dimension de features, plus les estimations sont bonnes mais plus les algorithmes sont longs à l’exécution. Et le temps d’apprentissage sur nos ordinateurs serait interminable sans la réduction de la dimension des images.

III. Présentation théorique des algorithmes de classification

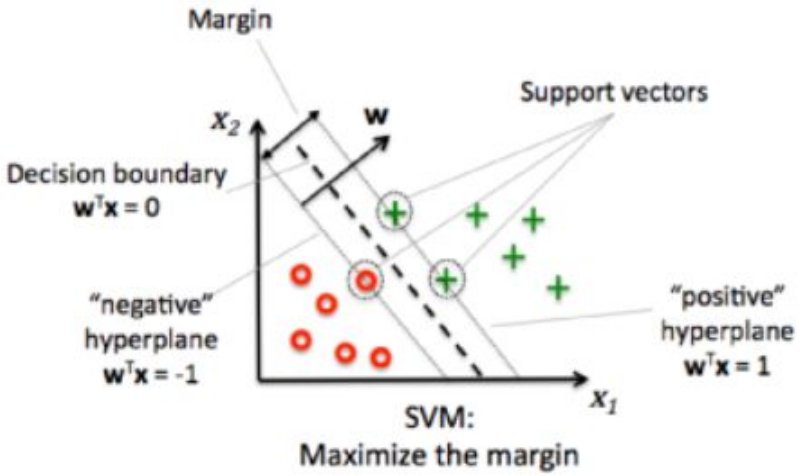
a) Régression logistique

La régression logistique est un type de classificateur linéaire. Nous pensons que ce genre de classificateur ne peut pas fonctionner dans la classification de 104 types de fleurs en raison de sa forme linéaire. Nous nous penchons donc directement sur les différents types de SVM.

b) SVM (Support Vector Machine ou Machines à vecteurs de support)

Les SVM sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression. Ils ont été développés dans les années 1990. Leur but est de séparer les données en classes à l'aide d'une frontière aussi "simple" que possible de façon à ce que la distance entre les différents groupes de données et la frontière qui les sépare soit maximale.

Figure 1 : Graphique de données linéairement séparables



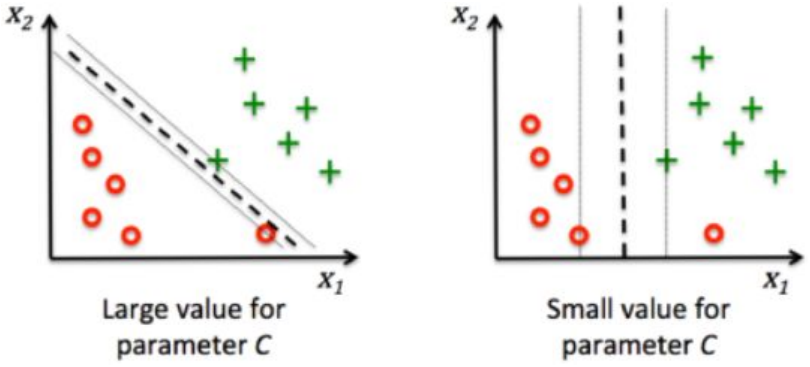
Prenons par exemple deux groupes de fleurs à classer en fonction de leur largeur de pétales (x_1) et de leur longueur de pétales (x_2). Les tulipes sont représentées en ronds rouges et les pâquerettes en croix vertes.

La distance entre ces deux groupes, comme on peut le voir dans le graphique, va être nommée de **"marge"** et les données les plus proches de la frontières sont ainsi qualifiées de " **séparateurs à vaste marge**" (**SVM**) ou encore de **" vecteurs de supports "**. C'est de par cette marge que nous allons pouvoir tenter de classer les nouveaux éléments arrivant. Les facteurs se trouvant au-dessus de "**l'espace de décision borné**" seront placés dans les pâquerettes et ceux se trouvant en dessous de cet espace (hyperplan), se retrouveront dans la catégorie "tulipe".

Il est important de parler également du paramètre de pénalisation C qui a une incidence directe sur la maximisation de la marge. C'est un paramètre considéré comme une contrainte dans notre modèle d'optimisation de la marge. Il désigne le coût que va

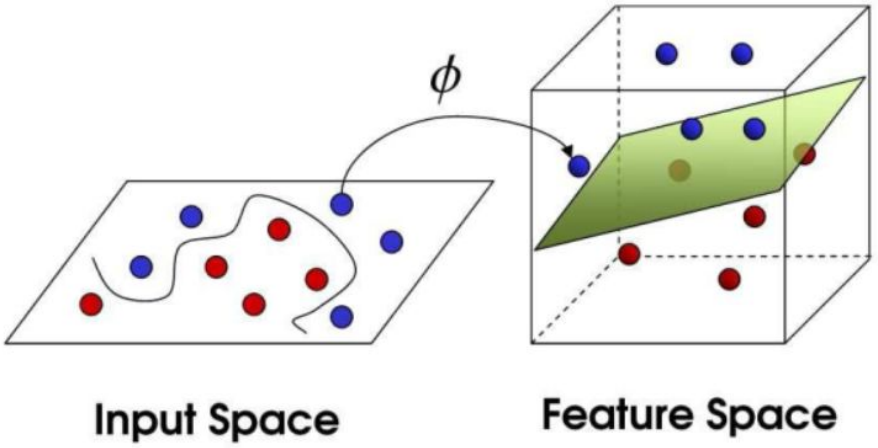
susciter un élément mal classé par le classificateur. Plus ce paramètre est élevé, plus la marge sera petite et plus le classificateur payera cher son erreur de classement. En revanche, une petite valeur du paramètre de pénalisation engagera un plus grand laxisme sur le taux d'erreur de classement.

Figure 2 : Pénalisation C



Ici les données sont linéairement séparables, ce qui est rarement le cas. Afin de gérer ce problème, les SVM se reposent sur l'utilisation des noyaux (kernel). Les noyaux sont des fonctions mathématiques qui permettent de projeter les données dans un **feature space** (un espace vectoriel d'une dimension supérieure à l'espace initial). Ces fonctions peuvent être "linéaire", "polynomiale","gaussienne" ou encore "sigmoïde".

Figure 3 : Projection des données sur un feature space



Dans le premier plan à gauche, nous remarquons que les données ne sont pas linéairement séparables. Grâce au noyau, une simulation de projection des données est faite sur un espace à **n** dimension où les données peuvent être catégorisées via un hyperplan linéaire de dimension **n-1**. Dans l'exemple, nous passons d'un espace **2D** à un espace **3D** et l'hyperplan permettant le classement des données est bien de dimension n-1 c'est-à-dire **2D**.

Les SVM sont utilisés dans de nombreux domaines tels que la bioinformatique, la recherche d'informations, la vision par ordinateur ou encore la finance.

c) Le Réseau de Neurones Artificiels (ANN) ou le Perceptron Multicouche

Les réseaux de neurones artificiels sont des systèmes inspirés du fonctionnement des neurones biologiques. Le plus célèbre d'entre eux est le perceptron multicouche , un système artificiel capable d'apprendre par l'expérience. Cet algorithme est développé en 1957 par Robert Rosenblat.

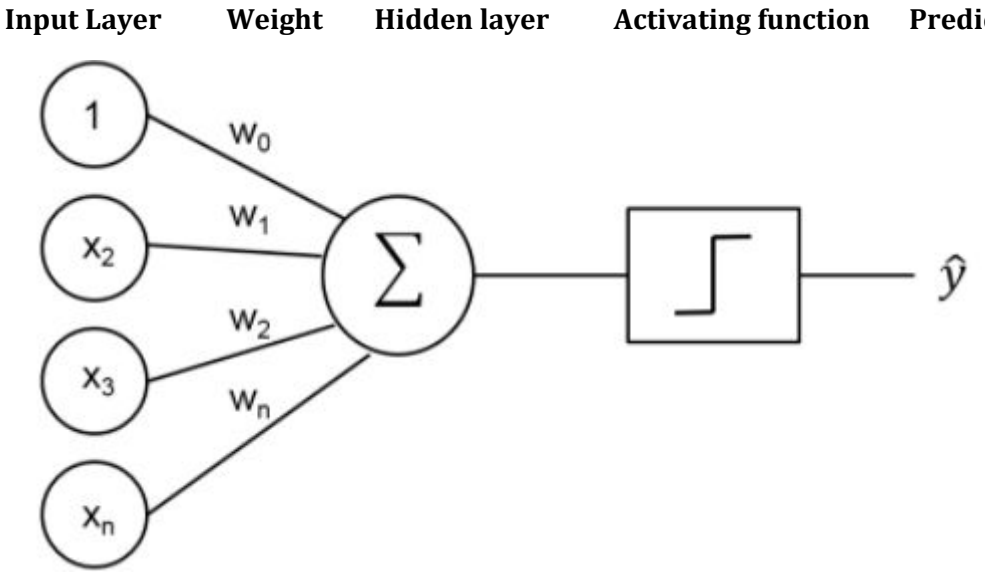
Le perceptron est un cas particulier de réseau de neurones . Il prend en entrée un vecteur à plusieurs dimensions (1 par neurone) et opère une séparation entre les données pour fournir une sortie.

En partant du principe biologique du neurone, le perceptron décrit un ensemble de neurones organisés en couches.

Prenons un perceptron organisé en trois parties. Tout d'abord, nous avons la couche d'entrée (Input Layer) qui correspond à l'ensemble des neurones portant le signal d'entrée. Tous les neurones de cette couche sont reliés à celui de la couche suivante.

La couche qui suit représente ce que nous appelons la couche cachée (**Hidden layer**). Il y a généralement plusieurs couches cachées. Il s'agit du cœur de notre perceptron, là où les relations entre les variables sont prises en compte. Par contre, le choix du bon nombre de neurones et du bon nombre de couches est très difficile. Par ailleurs, deux couches suffisent pour la majorité des problèmes, et mettre un nombre au-delà de 6 à 10 va poser des problèmes d'**overfitting.**

La troisième partie correspond à la couche de sortie, cette couche représente le résultat final de notre réseau, sa prédiction.



La figure ci-dessus représente un perceptron simple à une couche cachée. Les caractéristiques de nos images (les valeurs des pixels allant de x_1 à x_n avec b=1 une constante) sont amenés dans l'input layer et associés à des poids (w_0 à w_n). Les poids sont à définir, et en général il est préférable de commencer avec des poids proches de zéro. La couche cachée accueille la somme de poids appliqués aux valeurs que la fonction de la couche cachée soit $\Sigma x_n w_n + b w_o$. Ces données passent ensuite par ce que nous appelons la "fonction d'activation" qui va permettre de n'obtenir que des résultats

normalisés pour les valeurs des prédictions ^y. Ces fonctions d'activation sont souvent sigmoïdes tandis que la fonction d'activation de la couche cachée est nommée "Relu". Ce principe ne se fait pas qu'une seule fois. En effet, pour que le réseau puisse apprendre il faut faire une mise à jour des poids de par la descente du Gradient pour pouvoir obtenir de meilleures prédictions.
Nous appliquons donc ce schéma autant de fois que nous le décidons pour ainsi faire une mise à jour des poids, faire baisser la perte du modèle et augmenter sa qualité de précision.

d) Les réseaux de neurones convolutifs (CNN)

Les réseaux de neurones convolutifs sont développés en 2012 par Alex Krizhevsky. Nous allons présenter brièvement la théorie et l'outil mathématique cachés derrière le CNN, ainsi que toutes les étapes de l'analyse faite par l'algorithme.
L'être humain peut visualiser les images via les formes et les couleurs, par contre les machines et surtout les ordinateurs , les visualisent comme des nombre organisés. Nous pouvons en tant qu'être humain avoir une vision d'ensemble des images mais ce n'est pas le cas pour les ordinateurs.
Prenons par exemple une image de résolution 16x16x3 pixels c'est-à-dire une image composée de 16x16 pixels et basée sur les nuances de couleurs de base "Rouge","Vert","Bleu". A noter que dans le cas d'images de format "PNG" il existe une quatrième dimension de couleur (α) correspondant à la transparence de l'image. Nous nous pencherons sur une dimension de 3 en ce qui concerne les images que nous donnerons aux CNN dans notre dossier.
Le réseau de neurones convolutif va donc recevoir l'image (de l'exemple présenté un peu avant) sous forme de matrice et va appliquer plusieurs opérations comme les filtres et les simplifications.

Le CNN est composé de couches tout comme l'ANN mais elle diffère tout de même. Dans un premier temps, nous pouvons retrouver **la couche de convolution classique** détenant trois paramètres : la taille de la convolution (appelé Kernel de dimension nxm), le "**stride**" qui représente le décalage du kernel entre chaque calcul et le

“**padding**” est la manière dont on peut “dépasser “ de l’image pour appliquer la convolution.
Nous pouvons également retrouver la **couche** de **convolution dilatée**, qui est similaire à la convolution à ceci près que le kernel est éclaté. Cette couche va prendre en compte un paramètre supplémentaire : le **dilation rate** , qui représente le nombre de pixels à ignorer tout au long de l’apprentissage et de la mise à jour de l’algorithme. C’est un paramètre intéressant car il peut permettre d’éviter le sur-apprentissage.
Les fonctions d’activation des couches convolutives sont également de forme “ReLU” tout comme les ANN, et la fonction de sortie utilisée sera “softmax”.

Dans notre dossier nous avons appliqué ce que l’on appelle le “**pooling**”. C’est une opération simple qui consiste à remplacer un carré de pixels (on va se déplacer sur l’image) par une valeur unique. De cette manière, l’image diminue en taille et se retrouve lissée. Il existe plusieurs types de pooling :

- Le “ **max pooling** “ qui revient à prendre la valeur maximale de la sélection. C’est le type le plus utilisé car il est rapide à calculer (immédiat) et permet de simplifier efficacement l’image.
- Le “ **mean pooling** “ (ou average pooling) , soit la moyenne des pixels de la sélection. Nous calculons la somme de toutes les valeurs et nous divisons par le nombre de valeurs. On obtient aussi une valeur intermédiaire pour représenter ce lot de pixels.
- Le “**sum pooling** “, nous calculons seulement la somme des pixels.

Le **mean pooling** et le **sum pooling** sont quasi-identiques. Cependant il y a une différence notable entre **max** et **mean**. En effet, le “ max-pooling” va avoir tendance à retenir les caractéristiques les plus marquées et simples dans une image (comme par exemple une arête verticale). Au contraire, “mean” étant une moyenne, seules les caractéristiques moins marquées ressortiront.
Nous nous concentrerons donc sur l’application du “max pooling” puisqu’ il se diffère de “mean pooling” dans les cas extrêmes mais se retrouve être quasiment similaire au “mean pooling” dans les autres cas.

IV. Application des différentes approches

Dans cette section, nous allons appliquer les différents algorithmes que nous avons présentés précédemment sur nos propres données. Ensuite, nous allons comparer ces différents algorithmes et regarder quelle technique obtient la meilleure qualité de prédiction sur nos données entre tous nos modèles.

SVM

Nous allons appliquer dans un premier temps l'algorithme des machines à supports vecteurs (**SVM**). Avant tout commencement, nous importons plusieurs librairies nécessaires à l'application de cet algorithme. La librairie de **"scikit learn"** est donc insérée dans le code ainsi que quelques fonctions importantes de ce package : **"SVC" , "svm", "metrics", "train_test_split" , "classification_report"** et **"confusion matrix"**.
En utilisant la fonction **"train_test_split",** nous pouvons découper notre base en deux sous bases ; l'une correspondant à la base d'entraînement représentant **70%** de la base initiale et l'autre destinée à la base test qui correspond à **30%** de la base initiale.

Tout d'abord, nous appliquons deux SVM en tenant compte d'un noyau linéaire et de différentes valeurs de pénalité pour observer s'il existe une différence de qualité d'estimation.
Dans le premier modèle, nous imposons une pénalité élevée de valeur **1000** (le paramètre de pénalité est représenté par **C**). Nous calculons ensuite le rapport de classification utilisé pour mesurer la qualité de prédiction de l'algorithme.
A partir de ce rapport, nous obtenons trois métriques principales de classification; **le taux de précision (accuracy)**, le **recall et** le **F1-score** pour chaque classe. Ces métriques sont calculées en se basant sur les **vraies** et les **faux positifs** ainsi que les **vraies** et les **faux négatifs**. La colonne **"accuracy"** mesure le taux de bonne prédiction pour chaque classe. Le taux de précision pour chaque classe est défini comme étant défini le **rapport** entre les vraies positives sur la somme des vraies positives et des faux négatifs.
Le **recall**, lui, représente le pourcentage de vrais positifs bien prédits ou identifiés. Pour finir, le **F1-score** représente la **moyenne harmonique pondérée de la précision** et du **rappel.** Le meilleur score est donc de 1 et le pire de 0. En règle générale, la moyenne

pondérée du score F1 doit être utilisée pour comparer les algorithmes de classification et non la précision globale.
En se basant sur les résultats obtenus dans le rapport de classification, nous trouvons que le taux de précision pour ce modèle pour les différentes classes varient entre 0.06 et 0.39. Nous remarquons aussi que le meilleur F1-score est pour la classe 73. En plus, le taux de précision globale de ce modèle est de **26%**. Il prédit donc correctement **26%** des cas.

Le deuxième modèle que nous entraînons implique également un noyau linéaire mais cette fois-ci avec une pénalité C de 1. Suite aux résultats obtenus dans le rapport de classification, nous constatons une légère amélioration des taux de précision pour les différentes classes qui varient entre 10 et 47%. Le taux de précision globale ou moyen, quant à lui, est égale à **27%** (ce qui est finalement similaire au résultat obtenu après la transformation via le PCA).
Nous observons donc une légère augmentation de 1 % de la qualité de prédiction comparé au SVM avec pénalité à 1000. Ceci nous montre qu'il ne faut peut-être pas imposer une pénalité trop élevée sur nos données car nous pouvons remarquer que cela peut diminuer la qualité de prédiction. A noter également qu'une trop forte pénalité peut induire un sur-ajustement.

Le troisième modèle que nous décidons de mettre en avant concerne un SVM polynomial, c'est-à-dire un modèle détenant une fonction de projection polynomiale dans l'espace supérieur.
Après quelques essais, nous choisissons un degré polynomial de 2 pour notre noyau. Nous remarquons que les taux de précision pour les différentes classes gardées dans notre échantillon varient entre 10 et 50%. Cependant, la classe 63 est ici la classe détenant le meilleur F1 score (0.47), ce n'est donc plus la classe 73 . En plus de cela, nous retrouvons un taux de prédiction globale des cas, pour ce modèle, égal à **33.47%**.
En comparant ce modèle avec les modèles précédents, nous pouvons observer une augmentation entre **7** et **6%**.

Les algorithmes suivants sont deux SVM gaussien (en prenant en compte un noyau "kernel radial basis") avec une régularisation de 1. Nous ajoutons également à ce

modèle le paramètre γ indiquant la dispersion de la fonction gaussienne. Plus le γ est petit, plus nous retrouvons une fonction de projection gaussienne aplatie. Dans le cas contraire où γ se retrouve grand, nous obtenons une projection des données, dans l'espace de dimension supérieur, composée de plusieurs fonctions très étroites. Elles projettent individuellement chaque donnée.

Dans le premier SVM gaussien, nous avons pris un gamma " auto" et dans le second un gamma "scale". Pour le SVM gaussien avec gamma auto, et en se basant sur les résultats du rapport de classification, nous trouvons que le taux de précision varie entre 0 et 1 pour les différentes classes. Le meilleur F1 score est pour la classe 53 pour ce dernier modèle nous trouvons que le modèle prédit correctement **29.09%** des cas.

Pour le second SVM gaussien avec gamma "scale", nous trouvons que le taux de précision varie entre 0 et 0.62 pour les différents classes. Le meilleur F1 score dans ce dernier modèle est pour la classe 67 et nous retrouvons le même taux de prédiction que le modèle gaussien avec un gamma "auto".

Nous avons pu faire la remarque qu'une pénalité trop grande peut conduire à un sur-ajustement et il est aussi important de faire attention au paramètre gamma. Nous nous demandons donc comment trouver les meilleures combinaison de C et γ à appliquer à notre modèle SVM gaussien pour maximiser sa qualité de prédiction et minimiser son erreur quadratique.

Tout d'abord, nous avons essayé d'appliquer cette technique sur une résolution de 50x50, ce qui a pris énormément de temps pour donner les résultats. Pour cette raison, nous avons décidé de réduire la résolution de nos images à 30x30 et de réappliquer le grid search.

Comment avons- nous procédé pour appliquer cette technique ?

Tout d'abord, nous avons créé une grille de paramètres de régularisation C et de dispersion gamma à tester puis nous avons entraîné notre modèle gaussien avec un nombre maximal d'itérations égale à 50. Nous avons appliqué sur ce modèle la fonction grid search avec une cross-validation. Nous avons introduit les paramètres de pénalisation dans ce modèle afin de déterminer à partir de la cross-validation les meilleures combinaisons des paramètres C et γ minimisant l'erreur quadratique moyenne et maximisant la qualité de prédiction. Après cela nous avons entraîné le

modèle final avec les paramètres obtenus en cross-validation. Et nous avons donc appliqué ensuite les prédictions.

En se basant sur les résultats obtenus dans le rapport de classification, nous remarquons que le taux de précision varie entre 0.07 et 0.59 pour les différentes classes. Le meilleur **F1-score** cette fois ci est pour la classe 53 qui est équivalent à 0.50. Le taux de précision globale pour le SVM gaussien (grid search) quant à lui est de **30%**. C'est un taux supérieur aux résultats des modèles SVM à noyau linéaire (**26** et **27**%) mais inférieur au taux de prédiction du SVM à noyau polynomial de degré 2 (**33.47%**). Cela est sûrement dû au fait qu'on ait réduit nos données à 30x30.

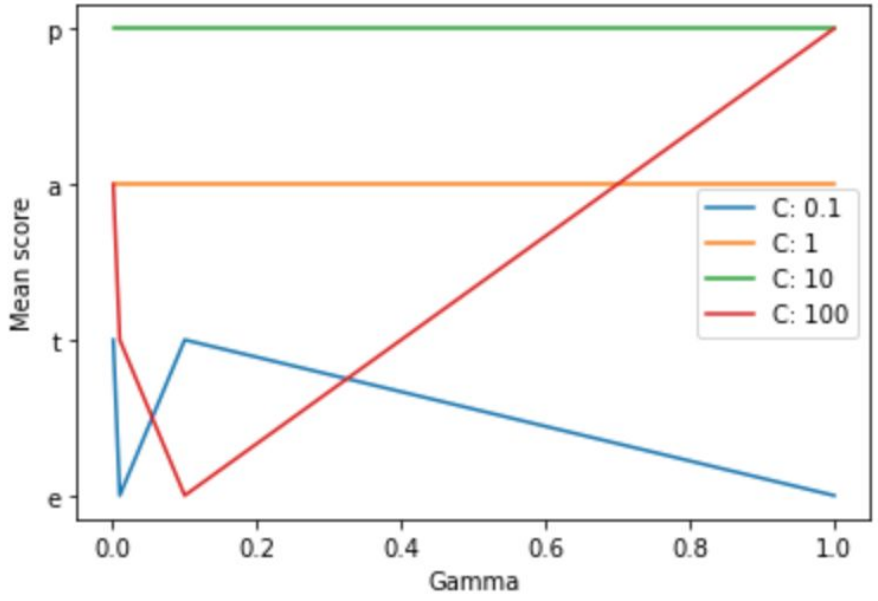


Figure 5 : Effet des paramètres C et gamma sur le SVM gaussien

En observant le graphique de la figure 5 ci-dessus, nous trouvons par exemple pour C = 0.1(courbe bleue) que le taux de précision diminue pour un paramètre gamma égale à 0. En revanche, avec un gamma qui est entre 0 et 0.1, le taux de précision commence à augmenter, par contre, pour un gamma supérieur à 0.1, le taux de précision décroît linéairement avec un C =0.1.

En ce qui concerne une pénalisation C = 1 (courbe jaune), nous trouvons que le taux de précision est constant. Il en est de même pour C = 10 (courbe verte), avec cependant un taux de précision plus élevé. Enfin, pour C = 100(courbe rouge), le taux de précision diminue lorsque le paramètre gamma est entre 0 et 0.1 mais commence à augmenter linéairement pour un gamma supérieur à 0.1.

ANN

L'algorithme d'apprentissage sur lequel nous nous penchons maintenant est celui du "réseau de neurones artificiel". A noter que nous détenons toujours une sous base train de 70% des données et une autre test de 30% des données. Les données ont également été standardisées, via une fonction créée nommée **"Standardisation"**, avant l'application de l'ANN.

Nous importons "keras" de la librairie **"tensorflow"** qui nous ait nécessaire à la construction de notre réseau de neurones et qui sera nécessaire aux réseaux de neurones convolutifs que nous verrons par la suite.

La structure de données de base de Keras est un modèle , un moyen d'organiser les couches. Le modèle principal est le modèle séquentiel , une pile linéaire de couches.

Nous construisons donc, pour commencer, un réseau de neurones séquentiel composé d'une seule couche cachée de 50 nœuds et prenant en entrée le nombre de composants des "features" c'est-à-dire 7500 entrées.

Nous appliquons la fonction d'activation "**Relu**" sur les valeurs des nœuds de la couche cachée. Cette fonction sortira le résultat maximum entre 0 et la valeur des nœuds tel que

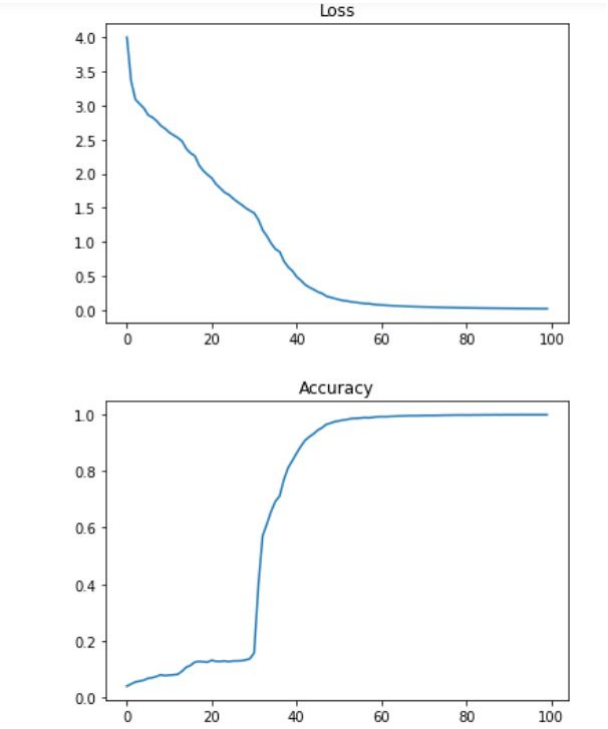
$f(x) = max(0, x)$, x étant la valeur d'un nœud.

Nous ajoutons ensuite la dernière couche appelée "couche de sortie". Les sorties devant être des valeurs comprises entre 0 et 1, nous appliquons la fonction d'activation dite **"sigmoid"** permettant d'obtenir des valeurs sur l'intervalle [0,1].

La dernière étape consiste à ajouter une fonction de coût et l'optimiseur qui nous permettront de mettre à jour les poids au fur et à mesure de l'apprentissage. L'optimisation se fait par la descente du Gradient. Nous entraînons ensuite notre modèle avec un nombre **"epochs"** égal à 100 c'est-à-dire que nous faisons passer toutes

nos données 100 fois dans notre modèle avec, à chaque itération, une mise à jour des poids.

Les deux figures ci-dessous nous montrent la progression de la fonction de perte et de la qualité de précision durant l'entraînement du modèle. Nous pouvons remarquer que le notre réseau de neurones s'est entraîné jusqu'à ne faire quasiment plus d'erreur.



Le taux de prédiction sur nos données est de **28.74%** (ce taux est évalué via la fonction "sparse_categorical_accuracy") avec une perte égale à **4.359** (évalué par la fonction "sparse_categorical_crossentropy"). Nous remarquons que ce taux ne change pas tellement des résultats que nous avons pu observer précédemment, il est même plus bas que celui du SVM gaussien ou polynomial. A noter que cela peut se comprendre étant donné que nous avons créé un simple réseau de neurones artificiel à une couche cachée.

CNN

Passons maintenant à la création de différents réseaux de neurones convolutifs (CNN).

Il est important de noter qu'un CNN, contrairement aux autres modèles, va pouvoir accueillir les tableaux des données dans leur dimension réelle et non aplatie en une dimension (c'est-à-dire en 50x50x3 pour la taille réelle). Les "features" doivent cependant être également normalisées.
L'idée d'une couche de convolution est de détecter des formes récurrentes retrouvées dans les différentes images (comme une forme plutôt ronde, carrée, des traits spécifiques aux images). Pour se faire nous désignons un filtre d'une dimension spécifique (en pixel : nxm) qui va scanner les images pour trouver des caractéristiques spécifiques à celles-ci et permettre ainsi de ne se concentrer que sur ces traits et transformer les valeurs des pixels des images.
Le CNN va se créer de la même manière que l'ANN (le principe est le même), seulement nous utilisons la fonction **"Conv2D"** du package "**keras"** pour pouvoir ajouter et appliquer nos couches convolutives dans notre réseau de neurones.

Tableau des différents CNN

	Qualité de prévision
CNN 3 couches	37.33%
CNN 3 couches avec "Max Pooling"	44,67%
CNN 5 couches avec "Max Pooling"	52,64%

Le premier CNN séquentiel que nous avons créé est composé de trois couches convolutives. La première couche recherche jusqu'à 128 formes de tailles 3x3, puis avec ces formes nous en construisons 64 autres de tailles 5x5 pour finir avec la troisième couche qui construit 32 autres formes de taille 7x7. La fonction d'activation pour la couche de sortie est cette fois-ci **"softmax"**.
Les deux fonctions pour la perte et la qualité de prévision restent les mêmes mais cette fois-ci nous utilisons un optimiseur du nom de **"adam".** Ce sera également le cas pour les deux autres CNN suivants.
Après un entraînement avec un **"epochs"** de 20 et un **"batch_size"** égal à 100, nous remarquons une amélioration du taux de prédiction à **37,33%** (contre 28.74% pour les ANN) et une perte de **6.6062** (plus élevé que celle des ANN).

Dans deuxième modèle de CNN nous ajoutons une couche appelée **"pooling"** permettant de réduire la taille de l'image, donc de se concentrer uniquement sur des formes de plus en plus grandes et d'effacer les petits détails qui ne sont pas forcément nécessaires à l'interprétation.En

d'autres termes elle essaye de mettre en avant les traits principaux des images, ce qui permet de réduire les problèmes de sur-apprentissage.
Cette couche va être amenée dans le code par la fonction **"MaxPooling2D"** de keras.
Dans le deuxième modèle donc, nous ajoutons une couche "pooling" de taille 3x3. Comme peut le présenter le tableau, nous obtenons une augmentation de la qualité de prévision (**44,67%**) et en plus de cela une baisse de la perte (**4,9610**) comparé au modèle précédent.

Pour le troisième modèle nous avons tenté d'augmenter le nombre de couches convolutives au nombre de 5. Ainsi la première couche cherchent 128 formes de taille 3x3, la deuxième 64 formes de tailles 5x5, la troisième 32 formes de tailles 7x7, la quatrième 16 formes de taille 9x9 et la dernière 8 formes de tailles 12x12. Nous gardons cependant une couche de "pooling" de dimension 7x7.
Après application sur la base test, nous remarquons encore une amélioration dans le taux de prévision (**52,64%**) et une très grosse baisse de la perte (**1.6411**).

Ci-dessous, le récapitulatif des taux de qualité de prévision de tous les modèles mis en marche.

Tableau récapitulatif des qualités de prévisions des modèles

Modèles	Qualité de prévision
SVM linéaire	27%
SVM polynomial	33.47%
SVM gaussien (grid search)	30%

ANN	28.74%
CNN 3 couches sans “MaxPooling”	37.33%
CNN 3 couches avec “Max Pooling”	44,67%
CNN 5 couches avec “Max Pooling”	52,64%

V. Conclusion

Les qualités de prévisions des différents modèles concordent avec ce que nous pensions obtenir comme résultats. Cependant, nous sommes tout de même étonnés que le taux de prédiction s'avère plus élevé pour le SVM polynomial que pour le SVM gaussien étant donné que nous avons appliqué le grid search sur ce dernier. Nous pensions également visualiser de meilleurs résultats avec l’ANN, ce qui n’est finalement pas le cas. A noter que nous avons construit un simple réseau de neurones à une couche cachée, ce qui a certainement une grande influence sur les résultats. Enfin, concernant les CNN nous obtenons de meilleurs résultats, ce qui est en accord avec ce que nous pensions étant donné que ce genre de réseaux de neurones convient mieux à la reconnaissance d’image.