ESGI 2021-22

AZERAD ARIEL

MAMADOU SARAMBOUNOU

RAPPORT : PROJET ANNUEL

**Rapport :**

Les étapes du projet :

1. Choix du sujet
2. Organisation des taches
3. Constitution du dataset
4. Model linéaire
5. PMC
6. KERAS
7. SKL
8. Interface graphique

**Choix du sujet :**

A l’annonce du sujet du projet annuel, nous n’avons pas été très inspiré. Mais après quelques jours à réfléchir et à réétudier le thème du Projet Annuel, nous avons trouvé l’idée et nous avons pu démarrer nos recherches et nos premières esquisses de codes.

Nous avons décidé de faire une classification entre des images d’hommes et de femmes. Combien de fois dans notre vie avons-nous du remplir des formulaires nous demandant notre sexe ?

Un nombre incalculable de fois, et c’est ici que notre modèle prend tout son sens, grâce à un

simple upload d’une image, on pourrait remplir tous ces formulaires automatiquement. Alors évidemment il est très rare que l’on nous demande uniquement notre sexe, cependant avec le temps, des nouveaux modèles ou des nouvelles features du modèle pourront être ajoutées et ainsi remplir entièrement la plupart des paperasses qui sont souvent chronophages.

**Organisation des taches :**

* Travaux Ariel :
  + Dataset Femme
  + MLP
  + API
  + Interopérabilité (windows)
  + création du git
* Travaux Mamadou :
  + Dataset Homme
  + model linéaire
  + API
  + Interopérabilité (mac)
  + Scikit Learn

**Constitution du dataset :**

Notre dataset est constituée de 1065 images d’homme et 1065 images de femmes. Le tout est partagé en 1530 images d’entrainement, 384 images de test et 214 images de validation.

Pour collecter nos données, nous avons eu recourt à plusieurs sources, qui sont les suivantes :

* Generated Photos: 507 images dans chaque classe
* SITE INTERNET : Getty images, Google Image, Bing Image
* RESEAUX SOCIAUX : Instagram et Facebook

Au début de la constitution de notre dataset, nous comptions faire un scrapper qui allait prendre toutes les images dont on aurait besoin sur l’API Generated Photos. (Generated Photos est un site qui propose des images de personnes n’existant pas réellement et ayant été généré grâce à un algorithme de Data Generation.) Cependant, on s’est rendu compte que si on faisait cela notre dataset serait assez peu varié. Donc, nous avons décidé de prendre uniquement une moitié de notre dataset sur ce site à l’aide d’un algorithme de scrapping. Puis, nous avons constitué l’autre moitié du dataset à la main pour être sur que les images correspondent à nos attentes.

Mais la encore, un nouveau problème s’est présenté à nous. En effet, pour récupérer des images de bonne qualité, nous sommes allé sur gettyimages, cependant les images de gettyimages sont de trop bonne qualité, et ne colle pas avec notre problématique, qui comprenait que notre modèle devait fonctionner même sur des selfies fait à partir d’un téléphone portable. Donc, nous avons pris seulement quelques centaines d’images sur gettyimages. Puis, nous avons trouvé une nouvelle méthode pour récupérer des images qui correspondait plus à des selfies ou au moins à des photos moins professionnelles. Cette technique consistait à taper des prénoms au hasard sur google. L’intérêt de cette technique est que l’on récupère de photos plus personnelles et qui ont un aspect plus authentique, notamment par le biais des photos LinkedIn ou encore des photos sur des articles.

**Preprocessing :**

Pour faire le prétraitement de nos données, nous avons créé une fonction qui prend en paramètre 3 arguments :

* Le chemin vers les jeux de données : homme, femme
* La taille des images

Mettre en paramètre la taille des images nous permet d’avoir toutes les images sous le même format. Nous avons choisi la taille (56,56). Toutes les images du jeu de données sont d’abord ouvertes, puis mis sous format 2D et ensuite sous forme de liste. Pour les classes, nous avons distingué les mots « male » et « female » dans les chemins d’accès des dossiers respectives de chaque classe. Si le chemin de l’image qui est ouvert à l’instant contient « male » alors la valeur 1 lui est attribué. Sinon si c’est « female », alors c’est la valeurs -1 qui lui est attribué. On mettra ensuite toutes les classes récoltées dans une liste Y.

A la fin de notre fonction, on va retourner une liste X, composée de toutes les images du jeu de données. Et une liste Y composée de la classe attribuée à chaque image.

Et pour finir, on procède à une formalisation de toutes nos valeurs X en les divisant par 255 au carré, dans le but de ne pas saturer nos fonctions d’activation.

**Librairies :**

Nous avons 4 model implémenté dont 3 qui fonctionne correctement.

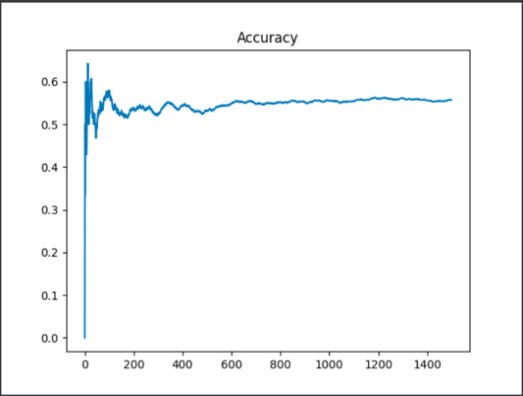
**Model linéaire :**

Pour notre modèle linéaire, nous avons finalement une accuracy de 0.57. Avec un learning rate de 0.01 et 1500 itération.

Une image contenant texte

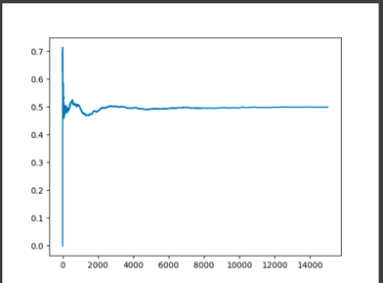
Description générée automatiquement

Voici un aperçu de la courbes obtenue.



On voit bien que l’accuracy comment par une valeur faible et tant vers 1 à la fin de l’itération.

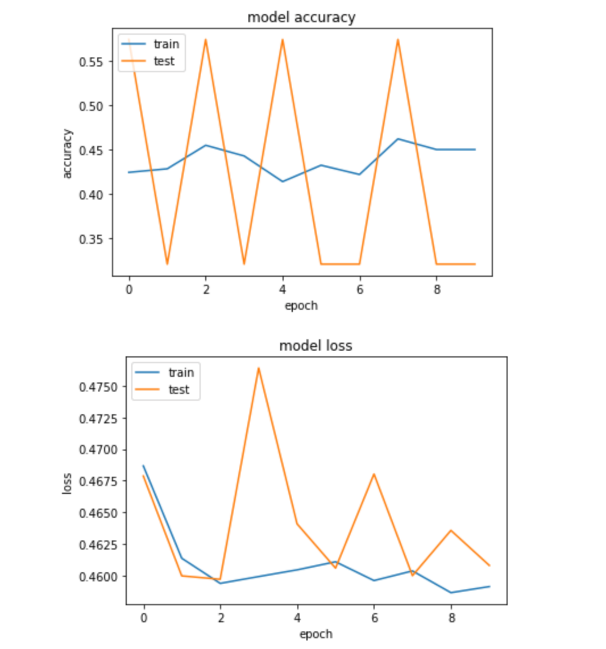
Mais avant d’avoir ces valeurs, nous avons testé différents hypermaramètre. Avec 150 000 itération et un learning rate de 0.1, on observe le graphique suivant.



Sur ce graphique, on constate que, l’accuracy augmente bien au début de l’itération. Puis diminue légèrement pour enfin stagner avec 0.5 d’accuracy.

**Keras :**

Nous avons implémenté un model Sequential avec la librairie Keras avec d’abord 3136 neurones en entrée, une couches cachée avec 10 neurones et enfin une couche avec 3 neurones en sortie. Le tout avec des fonctions d’activation tanh. Nous avons remarqué que le model n’apprenait pas bien avec seulement une accuracy de 30%. Par la suite nous avons fait un autre test avec cette fois-ci 3136 neurones en entrée ce qui correspond au nombre de pixel d’une image importer. Suivit d’une couche cachée de 8 neurones et enfin une couche de sortie avec une fonction d’activation softmax. Grace à la fonction softmax on, on a une liste en sortit de la taille du nombre de classe.



**Perceptron Multi Couches:**

Pour implémenter le perceptron multi couche et faire l’interop avec Python, il aura fallu plusieurs mois. Ariel s’est occupé de toutes les parties de cette algorithme.

MLP en C++ :

A la répartition initiale des tâches, l’initialisation du PMC devait être effectué par Mourad, cependant il a décidé d’abandonner les cours en plein milieu d’année sans plus donner aucun signe. Donc Ariel a du reprendre cette partie parce qu’il avait déjà commencé le predict\_mlp et le training\_mlp, mais sans create\_mlp, il ne pouvait rien tester.

Heureusement, grâce aux implémentations python des algorithmes qui nous ont été fourni, nous avons pu comprendre assez bien la logique des algorithmes.

Cependant, pour l’implémentation ça a été assez compliqué, car, Ariel n’ayant jamais codé en C++ et que très peu en C, il n’est pas familier avec les langages qui demande qu’on alloue et qu’on libère manuellement la mémoire que l’on veut utiliser.

Après quelques semaines, notre MLP fonctionnait sur C++. C’est-à-dire qu’avant de faire les tests sur Python, nous avons fait des tests sur C++ avec les cas de tests pour vérifier que nos algos fonctionnaient.

Interopérabilité avec Python :

Après vérifier que la librairie fonctionnait sur C++, est venu le temps de l’interopérabilité sur Python.

Cependant, problème, ayant réalisé toute la lib sur MingW, le fichier DLL ne s’est pas généré. Une image contenant texte, capture d’écran, portable

Description générée automatiquement

Donc, ne trouvant pas de solutions après plusieurs heures de recherche, nous avons demandé de l’aide à M.Vidal.

Pour solutionner ce problème, il fallait en fait changer de compilateur et prendre un compilateur Visual Studio.

On a ensuite pu avoir un fichier DLL qui fonctionnait et ainsi faire fonctionner notre Interopérabilité.

Entraînement de notre modèle avec notre dataset :

Après avoir fait les premiers tests avec notre dataset, on s’est rendu compte que notre librairie ne marchait pas si bien que ce que l’on pensait. En effet, une exécution sur 2, nous avions un ‘acces violation reading’, autrement dit un problème d’allocation de mémoire C++.

Une image contenant texte, capture d’écran, moniteur, noir

Description générée automatiquement

Cependant, Ariel ne l’avait pas compris au moment où il a vu l’erreur, il lui a fallu plusieurs journées.

Puis une fois ce problème résolu, ce fut la consécration.

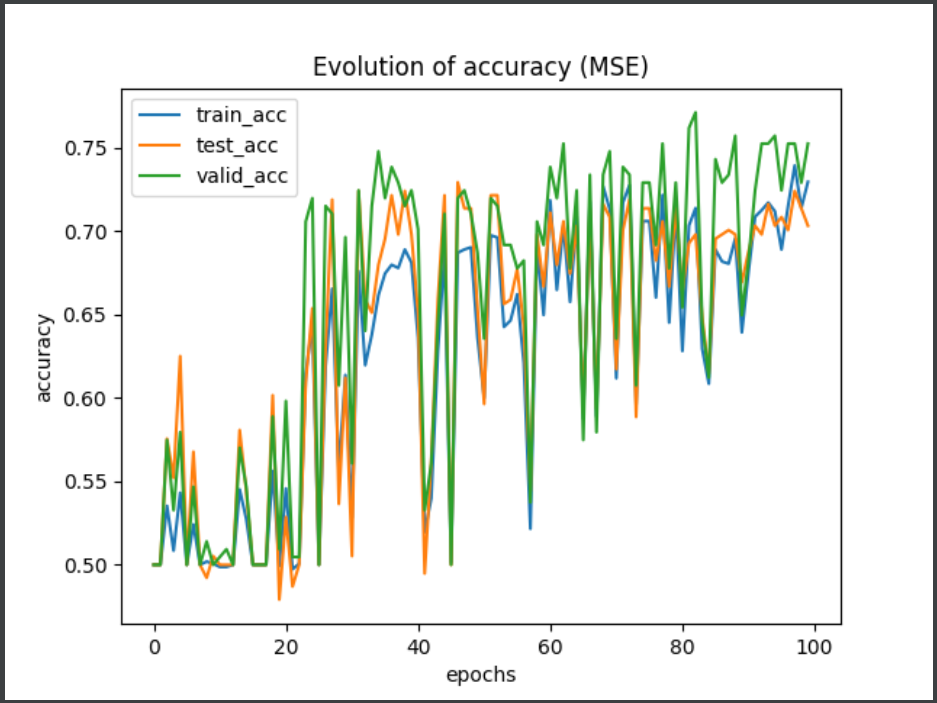
En plus d’avoir un modèle qui marche, nous avons eu du premier coup une accuracy d’environ 0.75.

Une image contenant texte

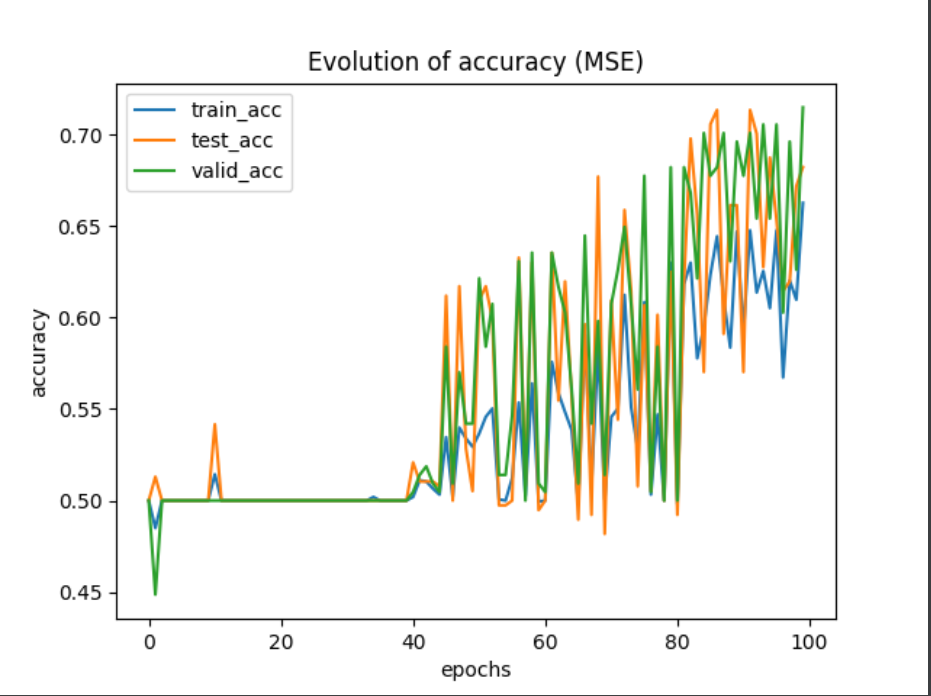
Description générée automatiquement

Cependant, afin d’être sur de la structure de notre modèle, nous avons voulu visualiser les accuracy de chaque Epochs.

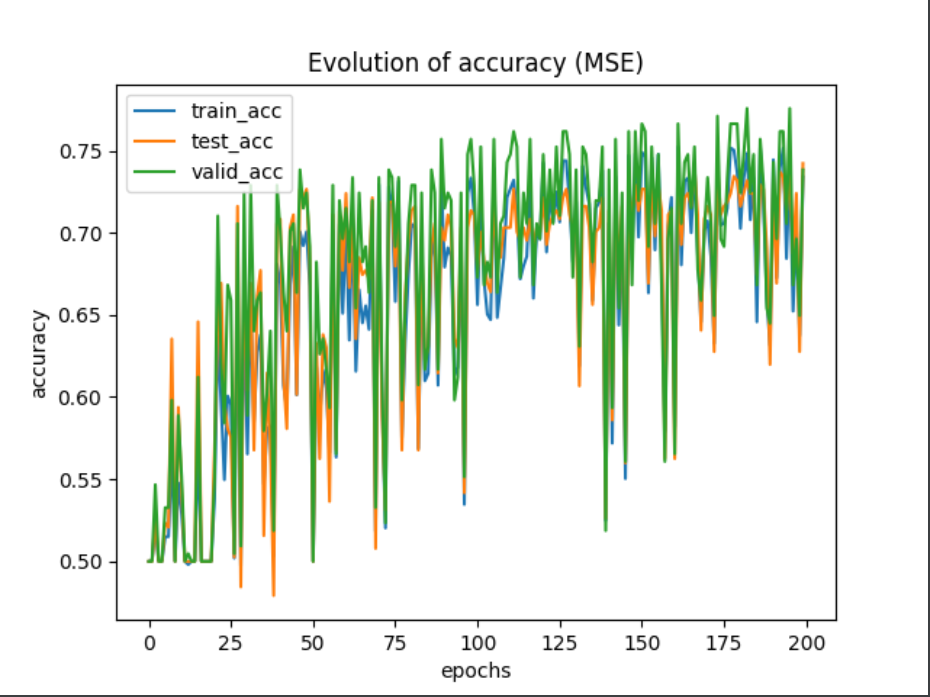
Pour avoir ces accuracies, nous avons utilisé un modèle avec 2 couches cachées et voici comment le modèle au cours des epochs.



En regardant une première fois, on s’est dit que notre accuracy variait beaucoup donc nous avons essayé de réduire le learning rate avec le même nombre d’itération et voici ce que ça a donné :



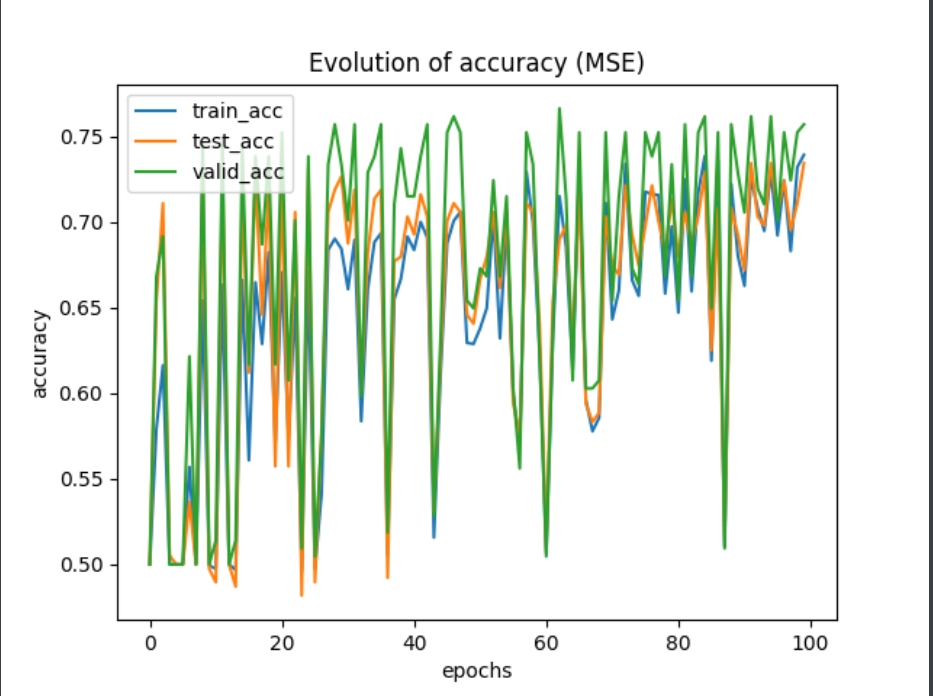
En visualisant ce graphique, on voit que le learning rate 0.01 et moins efficace que le 0.1 et on voit qu’il augmente uniquement à partir de 40.000 itération et se maximise à 0.7.

Donc, ayant compris que le learning rate de 0.1 était le plus efficace, nous avons décidé d’augmenter le nombre d’itération pour voir si le modèle s’améliorerai plus avec plus d’itération.

Voici la courbe des accuracy avec 200.000 itérations et on se rend compte qu’à partir d’environ 90.000 itérations l’accuracy commence à stagner autour des 0.7. On comprend donc qu’il n’est pas forcément nécéssaire d’augmenter le nombre d’itérations.

Après cela, il nous restait à tester le nombre de couches cachées.

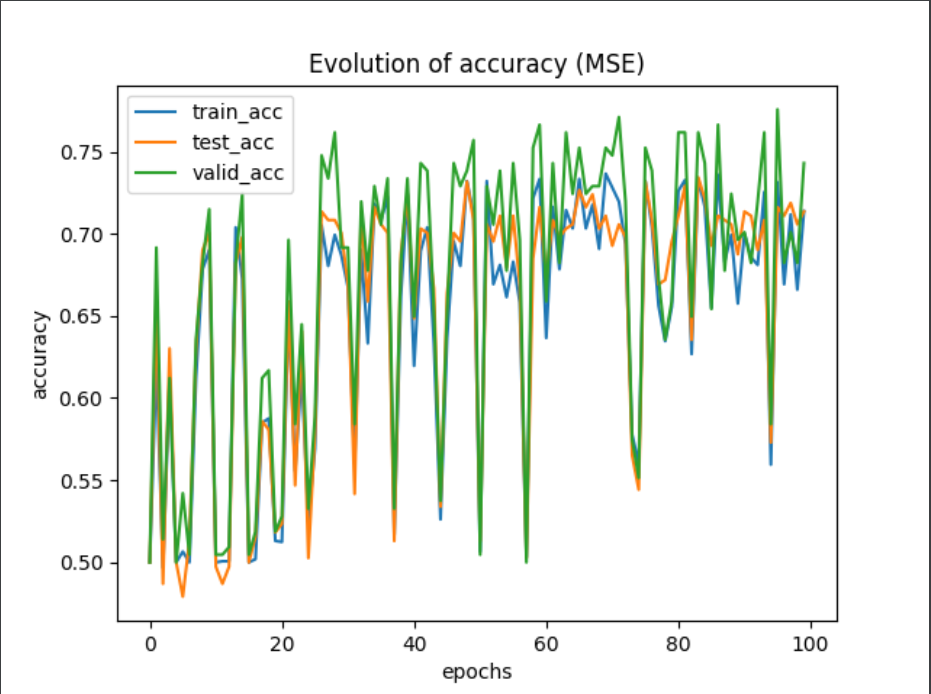
Dans tous les précédentes expériences, on mettait 2 couches cachées, voici ce qui se passe quand on en a 10 :



Avec ce modèle on peut voir une amélioration de l’accuracy plus tôt et on peut voir également que le bas de la courbe est plus haut, ce qui signifie qu’on a plus souvent plus de 0.7 d’accuracy.

Donc ce modèle à l’air plus performant.

Et voici ce qui se passe quand le nombre de couches cachées augmente encore jusqu’à 50 cette fois-ci :



Ici, on voit que la courbe est plus stable, cependant elle commence à décroître à partir de 70.000 itérations environ.

Pour conclure, on peut dire que la structure de modèles avec 10 couches cachées, un learning rate de 0.1 et 100.000 itérations est potentiellement l’une des meilleurs pour ce modèle.

**SKLEARN :**

Une image contenant texte

Description générée automatiquement

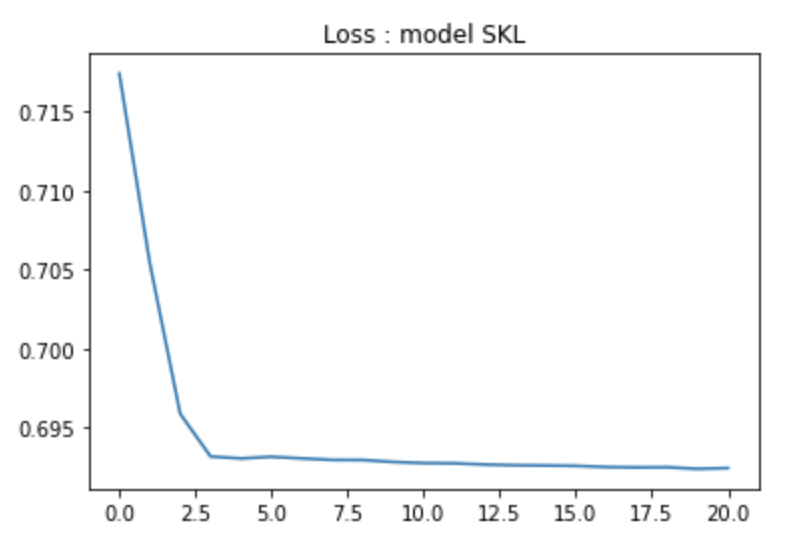
Le 4e modèle que nous avons implémenté est grâce à la librairie sckit-learn. Une librairie de python. Étant donné que Scikit-learn, choisit grâce au données envoyé le nombre de neurones dans la couche d’entrée et la couche de sortit. Nous n’avons donc que les couches cachées au choix. Nous avons choisi donc une couche cachée de 10 neurones avec 1000 itérations et un learning rate de 0.01.

Grace à ce modèle, nous avons de 0.7 sur les jeu de d’entrainement et 0.9 sur les jeu de test.

Une image contenant texte

Description générée automatiquement

Voici un aperçu de la courbe des valeurs des métriques de loss :



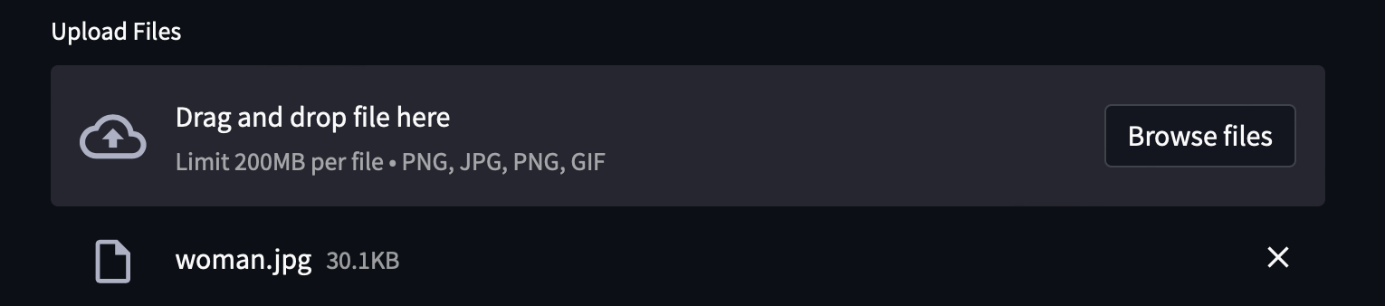
On observe que la courbes suit bien le résultat attendu, savoir au départ des valeurs élevé, qui au fur et à mesure comme à baisser jusqu’à tendre vers 0.

**Interface graphique :**

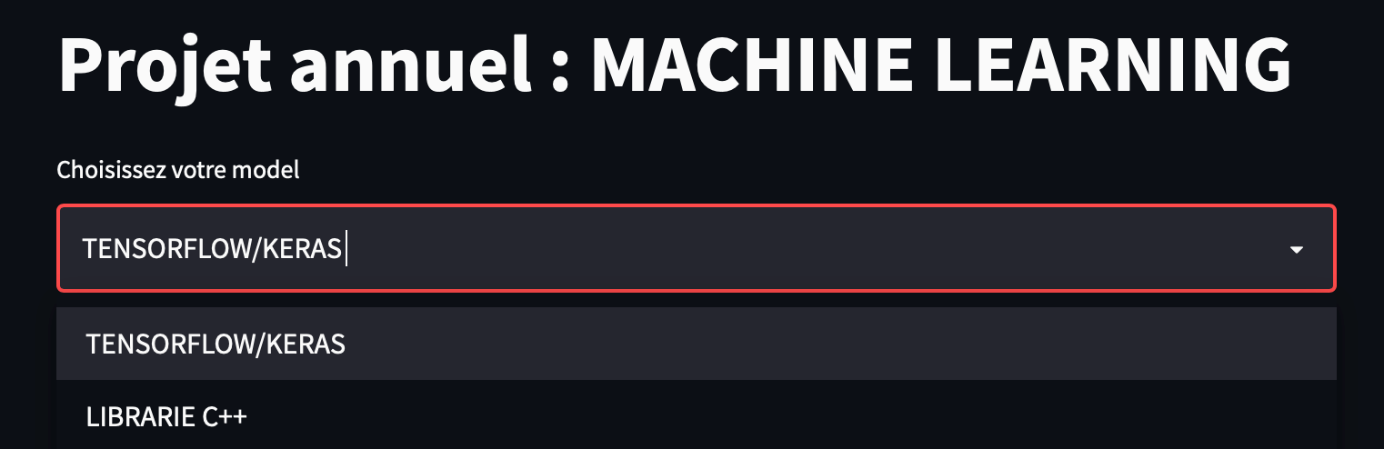
Pour faire notre interface graphique, on avait une liste de 3 choix : Flask, Streamlit et Flutter. Flask et Streamlit sont des libraire Python pour une interface graphique sur le web et Flutter nous permettrait de créer une application mobile. Nous avons d’abord implémenté notre interface graphique avec Flask, car chaque membre de notre groupe avait déjà utilisé cette librairie ce qui serait plus opportun pour le groupe. Cependant, nous avons eux quelque problème avec Flask, pour différentes raisons :

Par la suite, étant donné le temps qu’il nous restait pour le projet, vous avons logiquement essayé streamlit puisque c’est une librarie sur Python. Or pour pouvoir implémenté sur Flutter, il aurait fallu se familiarisé avec le langage d’abord et puis ensuite commencer à implémenter le projet.

Streamlit propose des APIs afin de s’intégrer à d’autres applications informatiques comme Python, et TensorFlow. Notre interface streamlit est composé d’une option qui permet d’uploader ou de drag and drop une photo avec des format déjà présélectionner (par exemple : png, jpeg, jpg…).



Une fois l’image envoyé à l’API, elle sera affichée sous un normal standardisé.Par la suite, l’image sera envoyée dans une fonction, qui contient le model. Le choix du model en question ce fait grâce à une liste déroulante qu’on propose depuis l’API

****

Ensuite le model prédira à quelle classe correspond l’image uploadé. Et pour finir le l’API affichera le nom correspondant à la classe retourné.Une image contenant texte, intérieur, capture d’écran, personne

Description générée automatiquement