ESGI
Paris
2018/
2019

Rapport Machine Learning

COMPETITION KAGGLE

AARGAN COINTEPAS - 4 IABD
JIHED RABIA - 4 IABD
SAID HAMIDOUCHE - 4 IABD

1) Pré-traitement (AarganC)

Nous avons commencé par télécharger le dataset directement sur la compétition Kaggle. Le fichier train.csv permet de connaître le nom de chaque fichier d'entraînement, nous construisons les différents chemins de la manière suivante :

```
filenames = ['../Data/data/train/' + fname for fname in train_csv['id'].tolist()]
```

Cette commande retourne une liste qui contient tous les chemins permettant de lire les images à l'aide de la librairie **`cv2.`** Nous appliquons ensuite les mêmes dimensions 32*32*3 afin d'avoir une uniformité et nous finissons par une division par 255 afin de ramener l'ensemble des valeurs entre 0 et 1. Cela aide à éliminer les distorsions causées par les lumières et les ombres d'une image :

```
train = []
for file_name in filenames:
    img = cv2.imread(file_name)
    img = img.reshape(32*32*3,)
    img = img/255
    train.append(img)
```

Ensuite nous récupérons l'ensemble des labels du fichier :

```
labels = train_csv['has_cactus'].tolist()
```

À l'aide de la fonction **`train_test_split`** nous répartissons les données entre les données d'entraînement et les données de test avec une répartition 90% train et 10% test :

```
x_train, x_test, y_train, y_test = train_test_split(train,
                                                    labels,
                                                    train_size=0.9)
```

Nous avons ensuite créé le tensor **`Input`** qui donnera au modèle le **`shape`** d'entré, nous avons pu remarquer que les modèles ResNet/LSTM nécessitent un **`reshape`** afin de pouvoir être utilisés. En effet, il faut lui préciser -1 comme premier champ car les modèles ont besoin de données avec un shape de 4 dimensions. Le -1 lui signifie qu'il aura N entrées.

```
x_train = np.array(x_train)
if name_modele != 'MLP':
    x_train = np.reshape(x_train, (-1, 32, 32, 3))
print(x_train.shape)

x_test = np.array(x_test)
if name_modele != 'MLP':
    x_test = np.reshape(x_test, (-1, 32, 32, 3))
print(x_test.shape)

if name_modele == 'MLP':
    inputs = Input(shape=train[0].shape)
else:
    inputs = Input(shape=(32, 32, 3))
print(inputs)
```

Nous finissons par convertir les vecteurs **`y_train`** et **`y_test`** en matrices binaires à l'aide de la fonction **`to_categrical`**. Les données d'apprentissage utilisent des classes sous forme de nombres, **`to_categorical`** transformera ces nombres en vecteurs appropriés à utiliser avec des modèles.

```
y_train = keras.utils.to_categorical(y_train, 2)
y_test = keras.utils.to_categorical(y_test, 2)
```

2) **Mise en place de GCP (AarganC)**

Afin de pallier la lenteur d'exécution des modèles sur une machine lambda j'ai fait le choix de monter une machine sur le service 'compute engine' de Google Cloud Plateform.

J'ai donc pris une machine avec les caractéristiques suivantes :

- type : n1-standard-2
- 2 vCPU
- 7,5 Go de mémoire
- 1 x NVIDIA Tesla V100

J'y ai installé les driver nvidia nécessaire à l'utilisation de la carte graphique ainsi que les packages nécessaires à l'exécution des différents modèles.

3) **Structure du code (AarganC)**

Pour pouvoir réaliser un maximum de test et pouvoir être le plus précis possible j'ai mis en place la structure suivante :

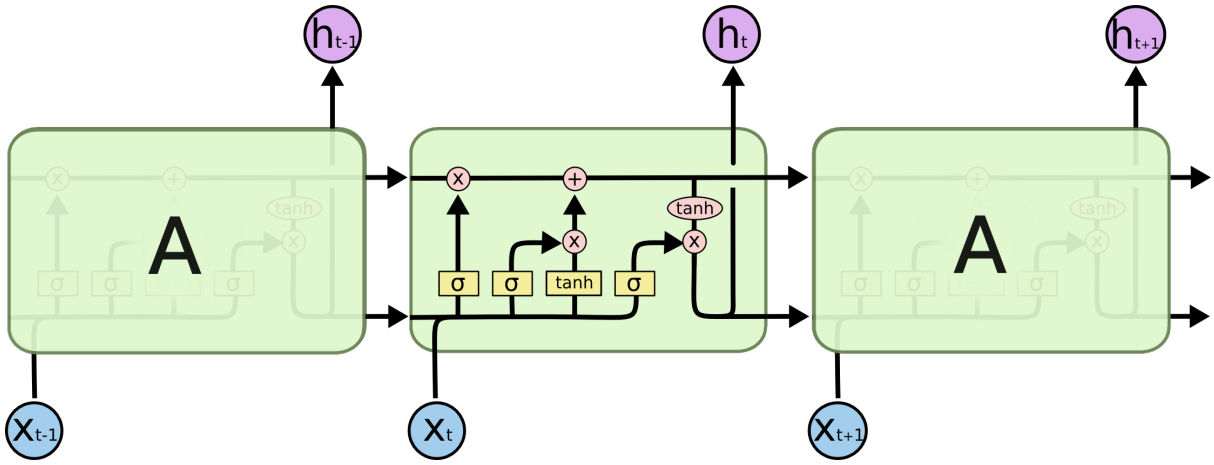
- Un template par modèle où seront provisionnées l'ensemble des fonctions nécessaires au lancement des différents modèles
- Un template_général ou nous allons retrouver : la récupération et l'organisation des hyper-paramètres, le prétraitement, le lancement des différents modèles, les paramètres Tensorboard ainsi que la prédiction
- Un script bash qui va lire un fichier csv où sont pré-remplis les hyper-paramètres et lancer le template_général autant de fois que nécessaire en lui passant les paramètres nécessaires à son exécution.
- Un fichier CSV où sont renseignés les hyper-paramètres.

4) Modèle RNN – LSTM (AarganC)

A) Définition

Le modèle LSTM est un type particulier de RNN, capable d'apprendre des dépendances à long terme.
Le LSTM a la capacité de supprimer ou d'ajouter des informations, soigneusement régulées par des structures appelées portes.

Les portes sont un moyen de laisser éventuellement passer l'information. Ils sont composés de couches de réseau neural sigmoïde et d'opérations de multiplications ponctuelles.



La fonction **`sigmoid`** a pour but de réduire la valeur d'entrée entre 0 et 1. En plus d'exprimer la valeur sous forme de probabilité, si la valeur en entrée est un très grand nombre positif, la fonction convertira cette valeur en une probabilité de 1. A l'inverse, si la valeur en entrée est un très grand nombre négatif, la fonction convertira cette valeur en une probabilité de 0.

La fonction **`tanh`** ressemble à la fonction **`sigmoid`**. La différence avec cette dernière est que la fonction **`tanh`** produit un résultat compris entre -1 et 1. La fonction **`tanh`** est centrée sur zéro. Les grandes entrées négatives tendent vers -1 et les grandes entrées positives tendent vers 1. Ce qui lui permet d'avoir une plus grande plage de filtrage que **`sigmoid`**.

B) Analyse

Afin d'obtenir une base de départ j'ai lancé 25 tests en faisant varier les hyper-paramètres suivants :

batch_size	epochs	LearningRate	nb_layer	nb_filtre
128	70	0.01	3	16
512	100	0.001	4	32
1024		0.0001	5	
			7	

J'ai par la suite sélectionné les 6 tests ayant obtenus le meilleur val_accuracy et le plus petit val_loss possible.

J’ai par la suite décidé d’ajouter une étape dropout afin de limiter les pics et d’essayer d’avoir des apprentissages plus linéaires et précis.
N’ayant jamais eu à me servir de cette fonction, j’ai cherché la manière dont elle est intégrée en fonction des modèles. Pour le modèle LSTM j’ai pu découvrir qu’on peut l'intégrer de plusieurs manières, j’ai retenu les trois positions suivantes :

- avant l’utilisation de la fonction **`BatchNormalization`**
- après l’utilisation de la fonction **`Flatten`**
- après le dernier **`Dense`**

J’ai également intégré deux variables qui me permettent de paramétrer les fonction dropout, de les combiner et de définir le taux de suppression :

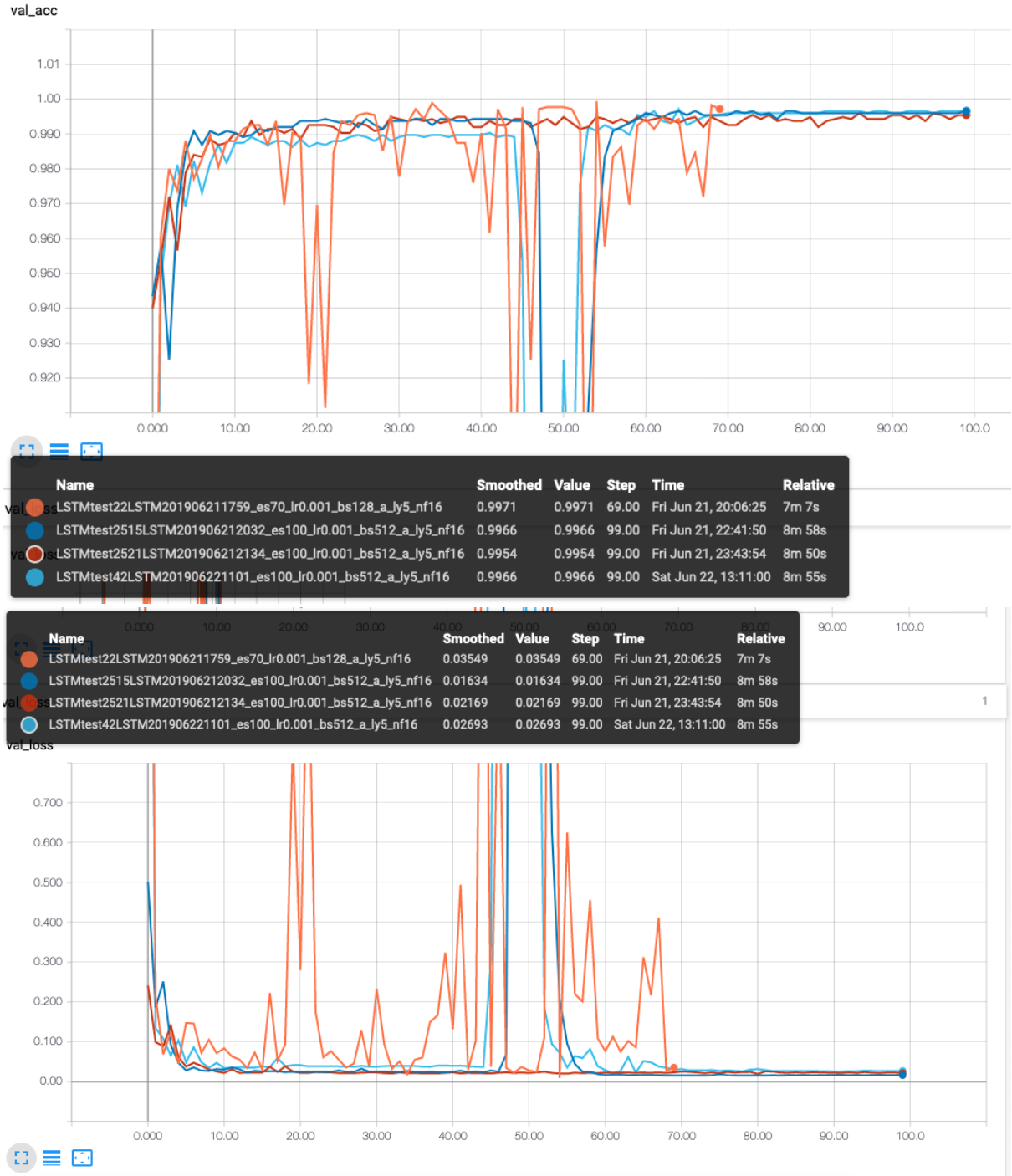
```
x = BatchNormalization()(outputs)

if dropout_flag == 1 or dropout_flag == 4 or dropout_flag == 5 or dropout_flag == 7:
    x = Dropout(dropout_value)(x)

x = keras.layers.add([x, outputs])

y = Flatten()(x)

if dropout_flag == 2 or dropout_flag == 4 or dropout_flag == 6 or dropout_flag == 7:
    y = Dropout(dropout_value)(y)

outputs = Dense(2, activation='softmax')(y)

if dropout_flag == 3 or dropout_flag == 5 or dropout_flag == 6 or dropout_flag == 7:
    outputs = Dropout(dropout_value)(outputs)

return outputs
```

J’ai fait varier les hyper-paramètres avec les valeurs suivantes sur 98 entrainements :

batch_size	epochs	LearningRate	nb_layer	nb_filtre	dropout_flag	dropout_value
128	70	0.001	3	32	1	0.2
256	100	0.0001	5	16	2	0.4
512					3	0.6
1024					4	
2048					5	
2056					6	
					7	

J’ai ensuite choisi de conserver les résultats ayant obtenus les meilleurs scores de val_accuracy ainsi que le val_loss le plus bas :

name_param	name_modele	batch_size	epochs	LearningRate	nb_layer	nb_filtre	dropout_flag	dropout_value	val_acc	val_loss
LSTMtest2515	LSTM	512	100	0.001	5	16	1	0.4	0.9966	0.01634
LSTMtest2521	LSTM	512	100	0.001	5	16	4	0.4	0.9954	0.02169
LSTMtest22	LSTM	128	70	0.001	5	16	0	0	0.9997	0.03549
LSTMtest42	LSTM	512	100	0.001	5	16	0	0	0.9966	0.02693



On observe que le dropout ne permet pas réellement d'améliorer l'accuracy dans le cas d'un LSTM, mais il arrive à réduire et à stabiliser la valeur de loss. En effet le but du dropout est de supprimer une partie des neurones afin d'éviter que le modèle n'apprenne par cœur le dataset d'entraînement, ce qui lui permet d'éviter les erreurs sur le dataset de test.

C) Submission

J'ai soumis les quatre prédictions générées par les tests analysés précédemment et j'ai obtenu les résultats suivants :

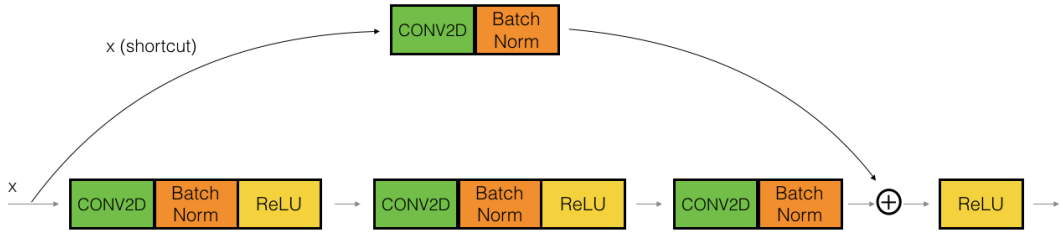
	val_acc training	val_loss training	Résultat Kaggle
LSTMtest22	0.9997	0.03549	0.9931
LSTMtest42	0.9966	0.02693	0.9920
LSTMtest2515	0.9966	0.01634	0.9860
LSTMtest2521	0.9954	0.02169	0.9838

5) Modèle ResNet (AarganC)

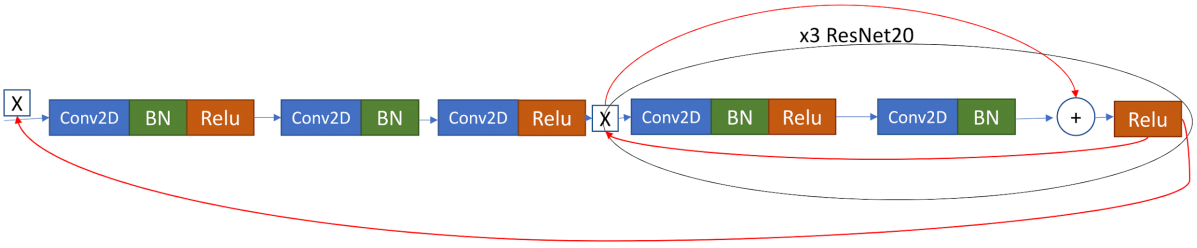
A) Définition

Un modèle ResNet consiste à établir des connexions de raccourci ignorant la création d'une ou plusieurs couches afin de créer un bloc résiduel. Cela signifie que les valeurs d'entrée sont ajoutées après l'ajout de couches. Cela permet essentiellement d'éviter une dégradation trop rapide et importante du gradient. Il permet donc de créer des modèles plus profonds sans perte de précision au cours de l'apprentissage.

Représentation du modèle ResNet Classique :



J'ai utilisé le model Keras ResNet CIFAR-10 (https://keras.io/examples/cifar10_resnet/) qui est le suivant :



B) Analyse

J’ai lancé 35 tests en faisant varier les hyper-paramètres suivant afin d’obtenir une première idée des valeurs des différents paramètres :

batch_size	epochs	LearningRate	activation	nb_layer
128	70	0.01	relu	3
512		0.001	selu	5
1024		0.0001	tanh	

J’ai choisi la fonction **`ReLU`** qui est l’une des fonctions d’activation les plus utilisées. Elle est interprétée par la formule : f(x) = max(0, x). Si l'entrée est négative, la sortie est 0 et si elle est positive alors la sortie est x. Cette fonction évite la saturation du modèle.

J’ai également choisi la fonction **`SeLu`** qui va approcher les valeurs moyennes proches de 0, ce qui va avoir comme impact d’améliorer les performances d’entrainements. Elle a un paramètre ALPHA prédéfini ce qui évite les problèmes d’explosion et de disparition de gradients en s’auto normalisant et gardant les mêmes variances pour les sorties de chaque couche, et ce tout au long de l'entraînement.

J’ai donc sélectionné les résultats suivants :

name_param	modele	batch_size	epochs	Learning Rate	Activation	nb_layer
ResNettest24	ResNet	128	70	0.01	selu	3
ResNettest25	ResNet	512	70	0.0001	relu	3
ResNettest28	ResNet	512	70	0.0001	selu	3
ResNettest31	ResNet	512	70	0.0001	tanh	3

Tout comme pour le modèle LSTM j'ai choisie d'ajouter une étape dropout afin de limiter les pics et d'essayer d'avoir des apprentissages plus linéaires et précis. J'ai ajouté l'étape dropout de la même manière que dans le modèle LSTM :

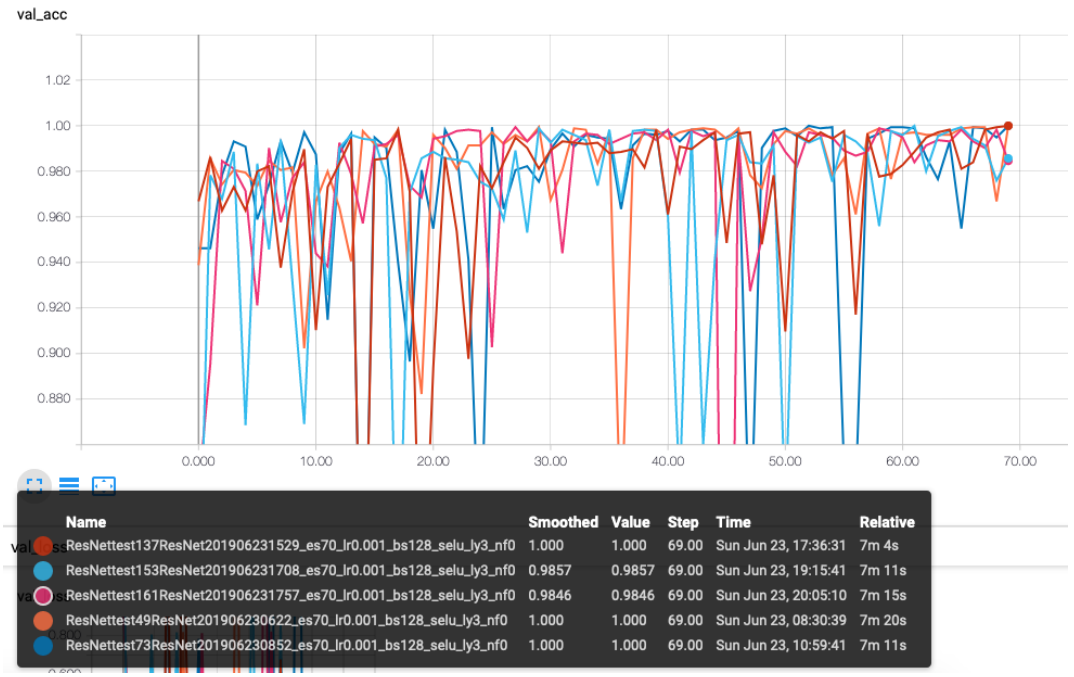
- avant l'utilisation de la fonction BatchNormalization
- après l'utilisation de la fonction Flatten
- après le dernier Dense

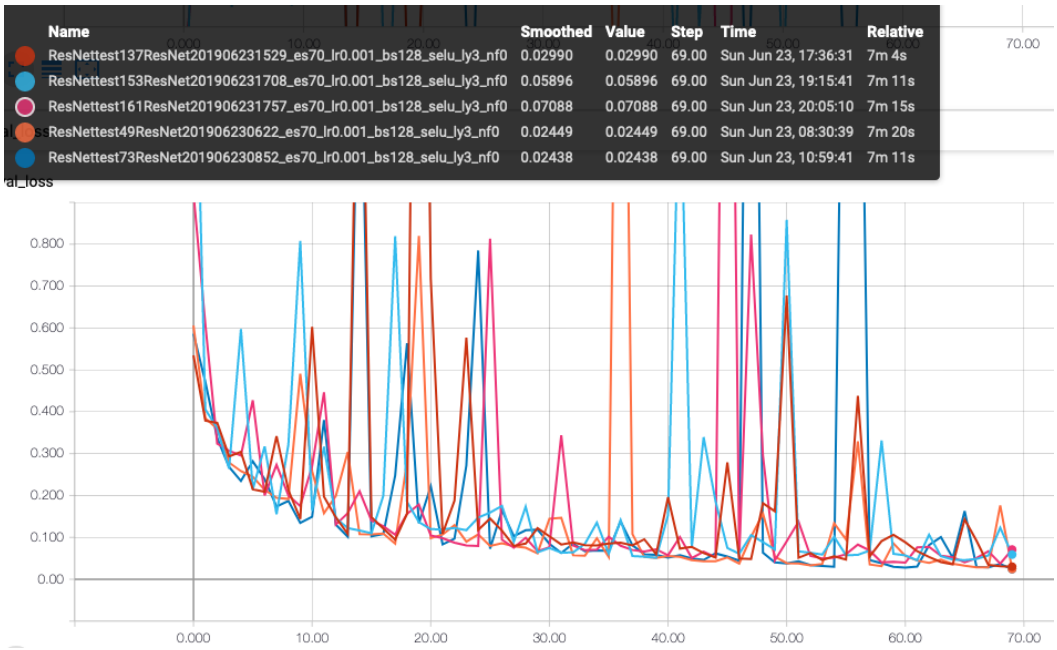
J’ai par la suite fait varier les hyper-paramètres avec les valeurs suivantes sur 167 entrainements :

batch_size	epochs	LearningRate	activation	nb_layer	dropout_flag	dropout_value
128	70	0.001	relu	20	1	0.2
512		0.0001	selu		...	0.4
1024			tanh		7	0.7

J’ai ensuite choisi de conserver les résultats ayant obtenus les meilleurs scores de val_accuracy ainsi que le val_loss le plus bas :

name_param	name_modele	batch_size	epochs	LearningRate	nb_layer	nb_filtre	dropout_flag	val_acc	val_loss
ResNettest49	ResNet	128	70	0.001	selu	3	2	1.00	0.02449
ResNettest73	ResNet	128	70	0.001	selu	3	5	1.00	0.02438
ResNettest137	ResNet	128	70	0.001	selu	3	6	1.00	0.02990
ResNettest153	ResNet	128	70	0.001	selu	3	1	0.9996	0.03227
ResNettest161	ResNet	128	70	0.001	selu	3	2	0.9997	0.03046





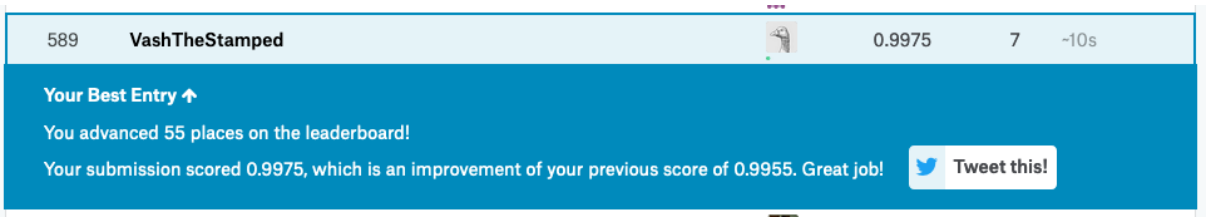
C) Submission

J'ai soumis les cinq prédictions générées par les tests analysés précédemment et j'ai obtenu les résultats suivants :

	val_acc training	val_loss training	Résultat Kaggle
ResNettest49	1.00	0.02449	0.9955
ResNettest73	1.00	0.02438	0.9975
ResNettest137	1.00	0.02990	0.9905
ResNettest153	0.9996	0.03227	0.9811
ResNettest161	0.9997	0.03046	0.9700

6) Conclusion

J'ai donc réussi à me glisser à la 589e place grâce au modèle ResNet.

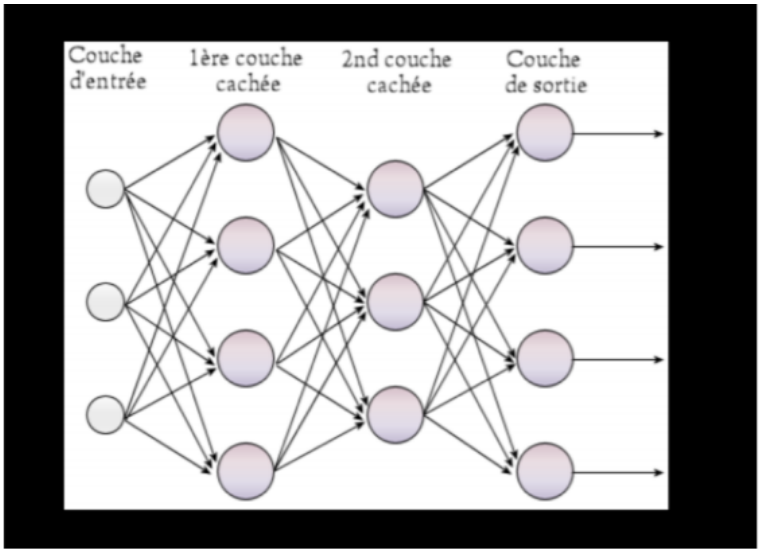


Le modèle LSTM est normalement approprié pour tout ce qui est lié à une notion de temps, comme c'est le cas d'avec les données audio, vidéo et aussi robotique.
Alors que le modèle ResNet est quant à lui optimisé pour la reconnaissance d'image.
Ce qui explique que j'ai réussis à obtenir de meilleur résultat avec le modèle ResNet avec ce dataset.

7) Perceptron multicouche (jiji38)

A) Définition

Le perceptron multicouche (multilayer perceptron MLP) est un type de réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau à propagation directe (feedforward). Chaque couche est constituée d'un nombre variable de neurones, les neurones de la dernière couche (dite « de sortie ») étant les sorties du système global. Les prétraitements utilisés sont les mêmes que pour le perceptron simple.

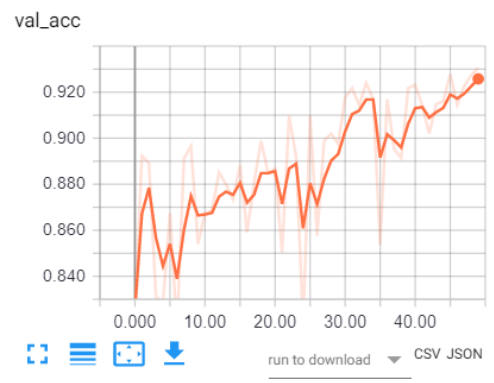


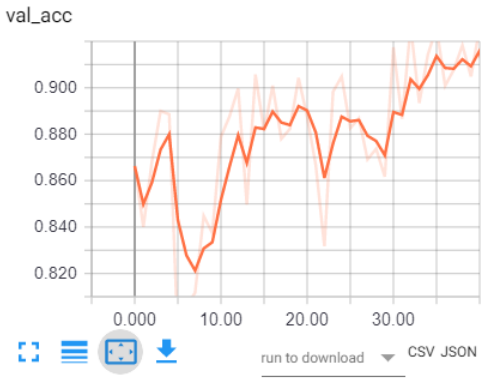
B) Tests

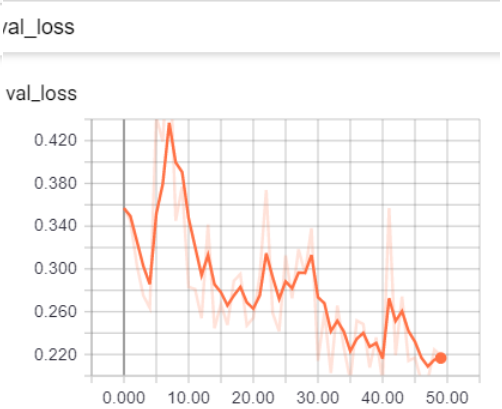
Pour le MLP nous avons une liste de 5 hyperparamètres que j'ai décidé de modifier : Batch-size, learning rate, fonction d'activation, nombre de layer et nombre de neurones.
Dans un premier temps j'ai décidé de lancer une première phase de test en modifiant uniquement la fonction d'activation le nombre de layer et le nombre de couche.
Voici un exemple ci-dessous des résultats.

NOM_TEST	batch-size	epoch	LR	ACTIVATION	NB_LAYER	NB_NEURONES	LOSS	ACC	VAL_LOSS	VAL_ACC
MLPTEST1	64	50	0.01	relu	3	128	0.2157	0.9107	0.2443	0.9020
MLPTEST2	64	50	0.01	relu	4	128	0.1857	0.9281	0.2362	0.909
MLPTEST3	64	50	0.01	relu	5	128	0.1904	0.9261	0.2552	0.904
MLPTEST4	64	50	0.01	relu	3	256	0.1987	0.9217	0.2345	0.9106
MLPTEST5	64	50	0.01	relu	4	256	0.2147	0.9139	0.2316	0.9069
MLPTEST6	64	50	0.01	relu	5	256	0.1974	0.9241	0.2609	0.8860
MLPTEST7	64	50	0.01	relu	3	512	0.1977	0.9210	0.2221	0.9126
MLPTEST10	64	50	0.01	selu	3	128	0.1731	0.9298	0.1981	0.9309
MLPTEST11	64	50	0.01	selu	4	128	0.1731	0.9324	0.2606	0.9003
MLPTEST12	64	50	0.01	selu	5	128	0.1663	0.9358	0.2225	0.9166
MLPTEST13	64	50	0.01	selu	3	256	0.1647	0.9359	0.2279	0.9117
MLPTEST14	64	50	0.01	selu	4	256	0.1791	0.9312	0.1837	0.9314
MLPTEST15	64	50	0.01	selu	5	256	0.1825	0.9299	0.1942	0.9251
MLPTEST16	64	50	0.01	selu	3	512	8.0097	0.4999	8.0590	0.5000
MLPTEST19	64	50	0.01	tanh	3	128	0.5636	0.7501	0.5596	0.7554
MLPTEST20	64	50	0.01	tanh	4	128	0.5635	0.7501	0.5563	0.7554

Après les premiers tests ou ont été modifiés aléatoirement les 3 paramètres ci-dessous. On se rend compte que la fonction d'activation la plus efficace est 'selu'.
Nous retenons deux modèles le 10 et le 14 qui ont une val acc de 0.93





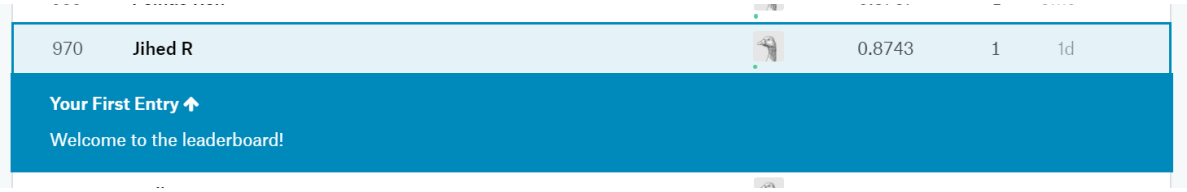


Pour essayer d'avoir de meilleur résultat nous nous basons sur ces deux modèles pour les prochains tests ou le batch-size et le lr seront modifiés.

MLPTEST21	128	50	0.01	selu	5	256	0.2304	0.9049	0.2353	0.9060
MLPTEST22	128	50	0.01	selu	4	256	0.1639	0.9374	0.1867	0.9300
MLPTEST23	128	50	0.01	selu	3	128	0.1755	0.9302	0.1985	0.9226
MLPTEST24	128	50	0.01	relu	3	256	0.2519	0.8953	0.2639	0.8980
MLPTEST25	128	50	0.1	selu	4	256	0.2261	0.9065	0.4038	0.8211
MLPTEST26	128	50	0.001	selu	4	256	0.1785	0.9306	0.2662	0.8963
MLPTEST27	64	50	0.1	selu	4	256	0.1854	0.9262	0.2159	0.9211
MLPTEST28	64	50	0.001	selu	4	256	0.1423	0.9471	0.2036	0.9323
MLPTEST30	128	50	0.1	selu	3	128	0.1809	0.9293	0.2432	0.9020
MLPTEST31	128	50	0.001	selu	3	128	0.1042	0.9617	0.2962	0.8971
MLPTEST32	64	50	0.1	selu	3	128	0.1747	0.9309	0.2020	0.9240
MLPTEST33	64	50	0.001	selu	3	128	0.1565	0.9388	0.2508	0.9006

Après avoir modifier le batch size j'ai modifié le lr et le batch size et je n'ai pas modifié le nombre d'epoch car je n'estimais pas cela nécessaire. Après avoir modifié ces paramètres nous avons des résultats peu convaincants. Un seul test se démarque c'est le numéro 28 qui possède une val acc de 0.93.

Pour finir j'ai décidé de submit le test 14 qui contient les meilleurs résultats.



J'ai obtenu le score de 0.87 ce qui est très faible mais le MLP n'est pas le meilleur dans le traitement d'image.
Source définition : Wilkipédia

8) Convolutional neural network (saidteste)

Le but de l’analyste :

Comprendre la répartition de deux classes (pas de cactus / a de cactus) Regardez certaines caractéristiques de l’image (distribution des canaux RVB, luminosité moyenne, etc.)

Les étapes :

1. 1- Utilisation des générateurs d'image pour le prétraitement des images d'entrée Les générateurs d'image ont été utilisés pour augmenter les données existantes. L'ensemble de formation est divisé en 80:20 en ensemble de formation et de validation. Des générateurs sont créés pour chaque groupe.
2. 2- Construire le modèle J'ai utilisé trois modèles, Le_Conv, Custom_Conv et Complex_model, et formé les modèles individuellement avant d'utiliser les prédictions du modèle. Tous les modèles utilisent 50 Epoques et arrêt précoce. La formation du modèle se déroule dans une simple boucle for

Quelques rappels standard fournis par keras. :

EarlyStopping: Arrête le processus de formation si le paramètre surveillé cesse de s'améliorer avec le nombre de périodes de «patience». RéductionLROnPlateau: Réduit le taux d'apprentissage d'un facteur si le paramètre surveillé cesse de s'améliorer avec le nombre de périodes de «patience». Cela aide à mieux adapter les données d'entraînement. TensorBoard: Aide à la visualisation. ModelCheckpoint: Stocke les meilleurs poids après chaque époque dans le chemin fourni. Pour plus de détails, consultez ce lien. Former les modèles en boucle et stocker les modèles formés

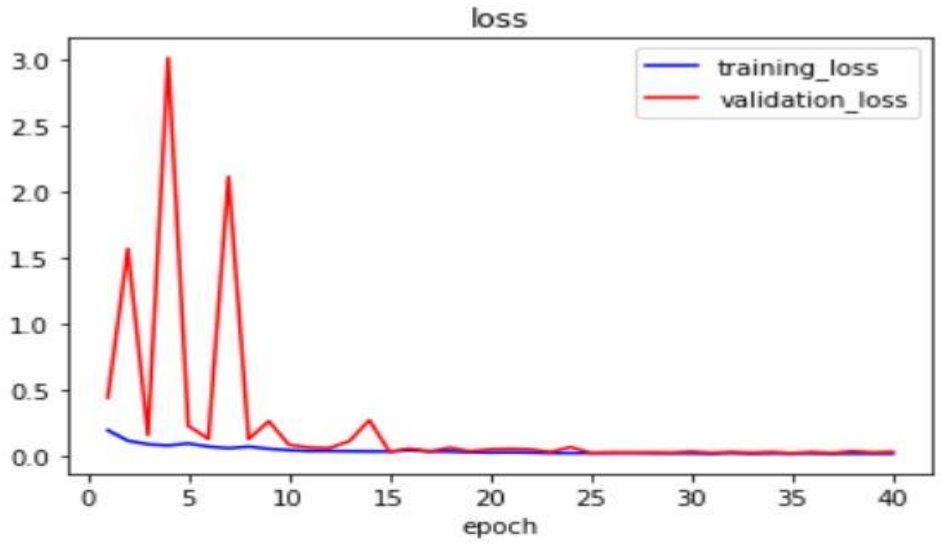
3- Visualisation Train vs Validation Ces parcelles peuvent aider à réaliser des cas de sur ajustement.

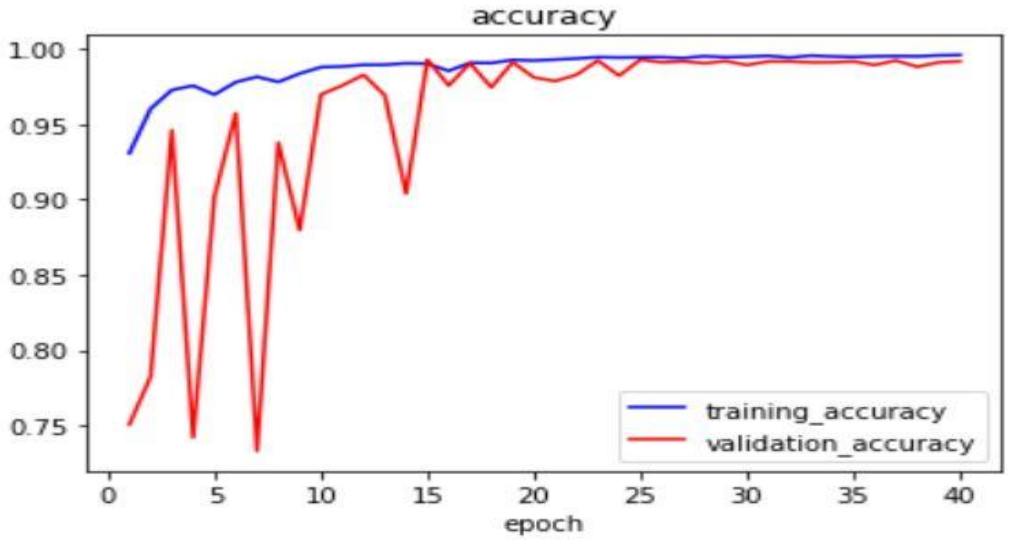
Note importante :

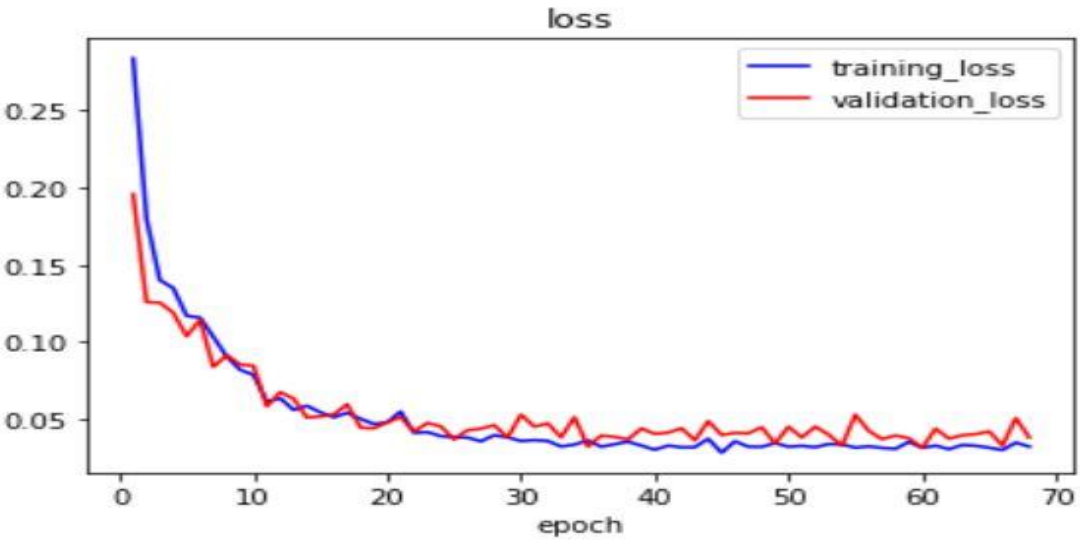
Vous remarquerez peut-être cela en ligne avec # !!! il n’est pas très clair 128 * 11 * 11. C’est la dimension de l’image avant les couches FC (H xlx C), puis vous devez la calculer manuellement (en keras, par exemple, .Flatten () fait tout pour vous).

Cependant, il y a une vie bidouille - il suffit de créer print (x.shape) dans forward () (ligne commentée). Vous verrez la taille (batch_size, C, H, W) - vous devez tout multiplier sauf le premier (batch_size), ce sera la première dimension de Linear (), et c'est dans CHW que vous devez "développer" x avant d’alimenter Linear ().

```
# Training plots
for history in histories :
    epochs = [i for i in range(1, len(history.history['loss'])+1)]
    plt.plot(epochs, history.history['loss'], color='blue', label="training_l
oss")
    plt.plot(epochs, history.history['val_loss'], color='red', label="validat
ion_loss")
    plt.legend(loc='best')
    plt.title('loss')
    plt.xlabel('epoch')
    plt.show()
    plt.plot(epochs, history.history['acc'], color='blue', label="training_ac
curacy")
    plt.plot(epochs, history.history['val_acc'], color='red',label="validatio
n_accuracy")
    plt.legend(loc='best')
    plt.title('accuracy')
    plt.xlabel('epoch')
    plt.show()
```




accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0
5
10
15
20
25
30
35
40
epoch
training_accuracy
validation_accuracy


loss
0.25
0.20
0.15
0.10
0.05
0
10
20
30
40
50
60
70
epoch
training_loss
validation_loss