



Groupe 7

Plant Pathology

Compte Rendu

Meriem Nadege Souici, Doubalo Lamien, Victor Meyer, Cécile Cousin

Sommaire

I- Description du dataset

a - Classement

II- Modèle Linéaire

III- Perceptron Multicouche

IV- CNN

V- ResNet

VI- RNN

I- Description du dataset

Plant Pathology est un dataset rassemblant des données sur des maladies contractées par les plantes.

4 pathologies sont répertoriées: "combinations", "healthy", "rust", "scrab".

Entraînement sur 1456 photos.

Validation sur 365 photos.

Distribution de nos classes:

Entraînement:

combination = 27,8%

healthy = 5%

rust = 34 %

scrab = 33,2 %

Validation:

combinations = 31%

























healthy = 5,5%

rust = 35%

scrab = 28,5%

La classe "healthy" est sous représentée.

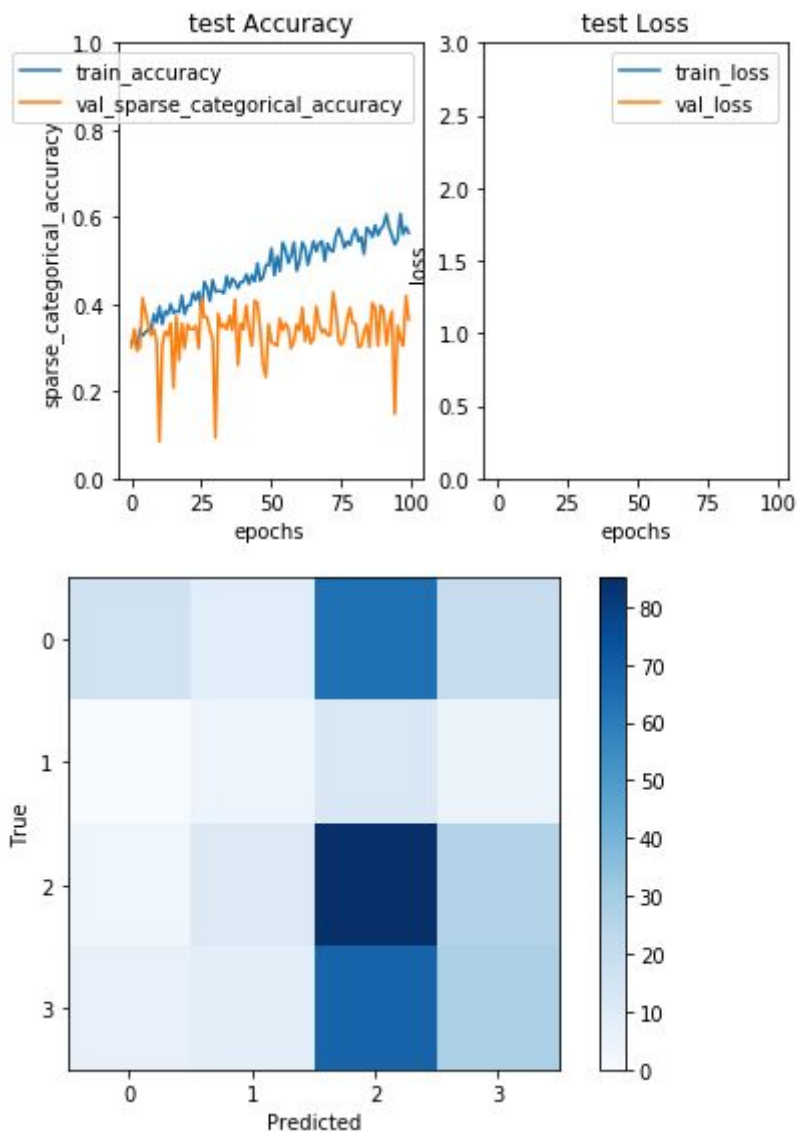
a - Classement

Overview	Data	Notebooks	Discussion	Leaderboard	Rules	Team	My Submissions	Late Submission
1178	▲8	k_z					 0.74521	1 2mo
1179	▲2	YBJJZHL				 	0.72955	2 1mo
1180	—	Adrian Fung					0.72817	1 24d
1181	▲8	sparse-shots					0.72201	3 3d
1182	▲5	Kuoyen Lo					0.72116	1 21d
1183	—	Shoaib Rain				 	0.71783	1 1mo
1184	▼2	Royston Tauro					0.71446	1 9d
1185	▲6	SriHarshaKonuru					0.71040	1 1mo
1186	▲4	Team6_IABD1				   	0.70918	2 6d
1187	▼2	cccccaiyuxuan					0.70642	3 1mo
1188	—	praveen kumar					0.70514	3 1mo
1189	▲5	KumarAbhinav					0.70074	2 1mo
1190	▼6	Rrohit_Kumar					0.69917	1 2mo
1191	▲1	Evgenii				 	0.69749	3 2mo
1192	▲1	Jose Muñoz			<> kernel37b4718519		0.69241	4 2mo
1193	▲2	AlexZhuravlev					0.67957	1 19d
1194	▲4	Luca Popescu					0.67597	2 23d
1195	▲6	Josephine Tsai					0.67004	1 2mo

II- Modèles Linéaires

Photos de dimensions 256 x 256

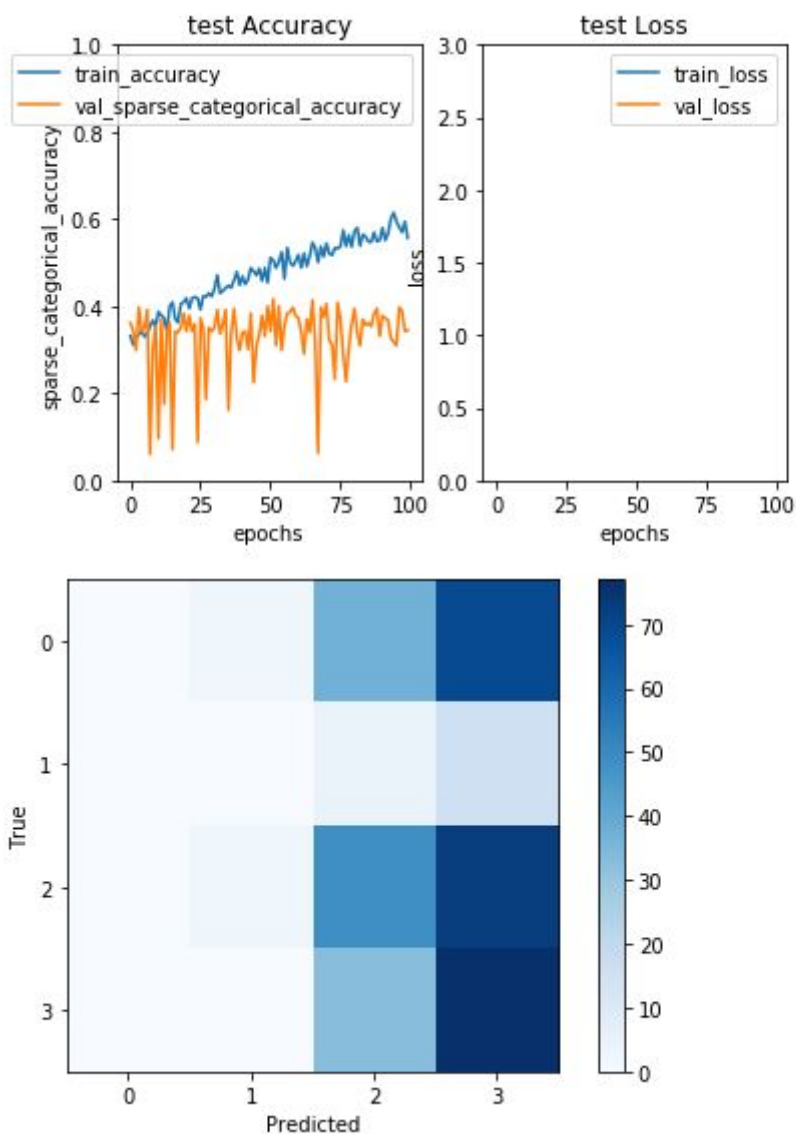
Modèle linéaire simple neurones de sortie avec activation softmax



Modèle overfité. Grande variance de l'accuracy de test. La matrice de confusion nous montre que le modèle retourne principalement "rust", la classe la plus représentée de notre dataset. Notre modèle est trop impacté par la disparité de notre dataset

Nous allons ajouter un dropout pour diminuer cette overfitting et essayé de ne moins être impacté par la composition du dataset

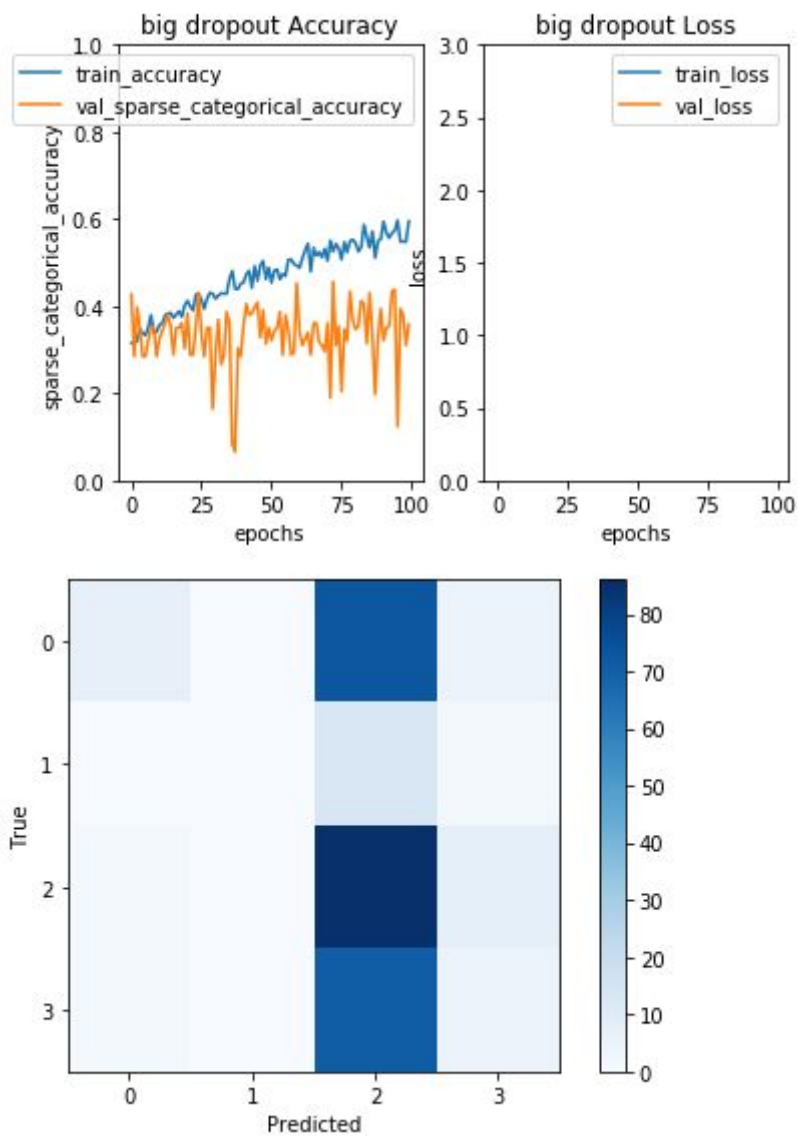
dropout 0.01



Le modèle bien appris 2 classes, les plus représenté dans les données d'entraînement, mais a complètement oublié la première classe, qui est pourtant assez bien représenté. La variance de l'accuracy de test a augmenté.

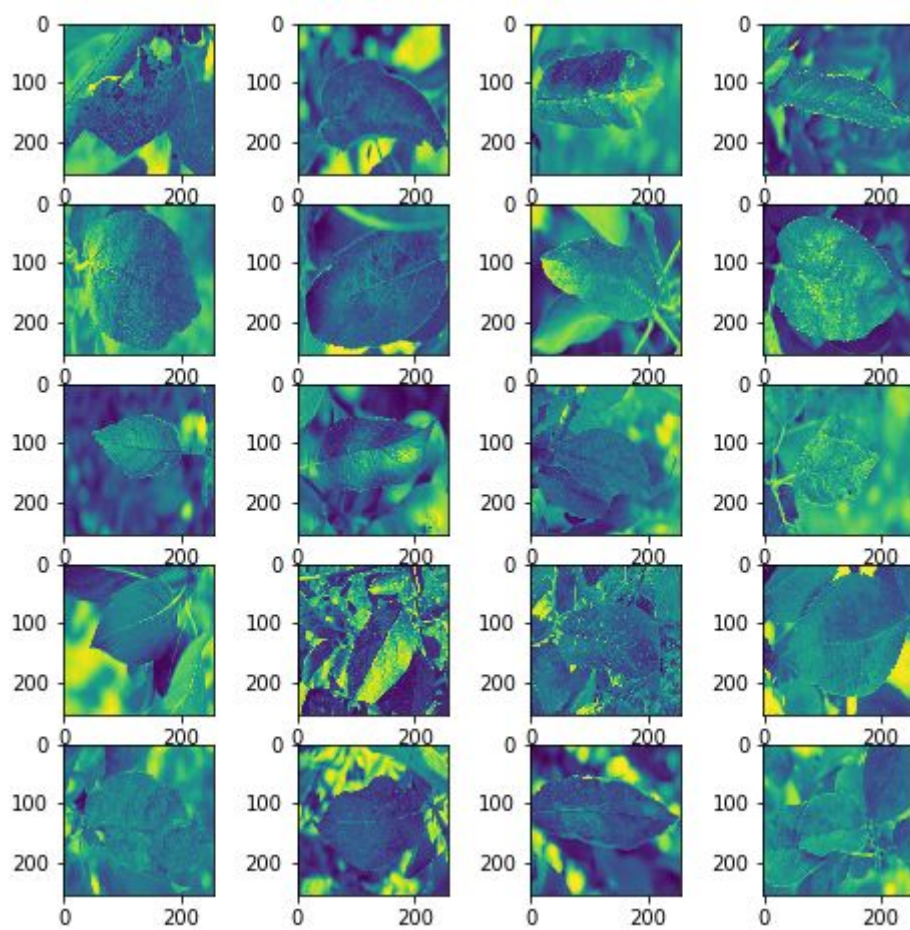
Nous allons augmenter le dropout

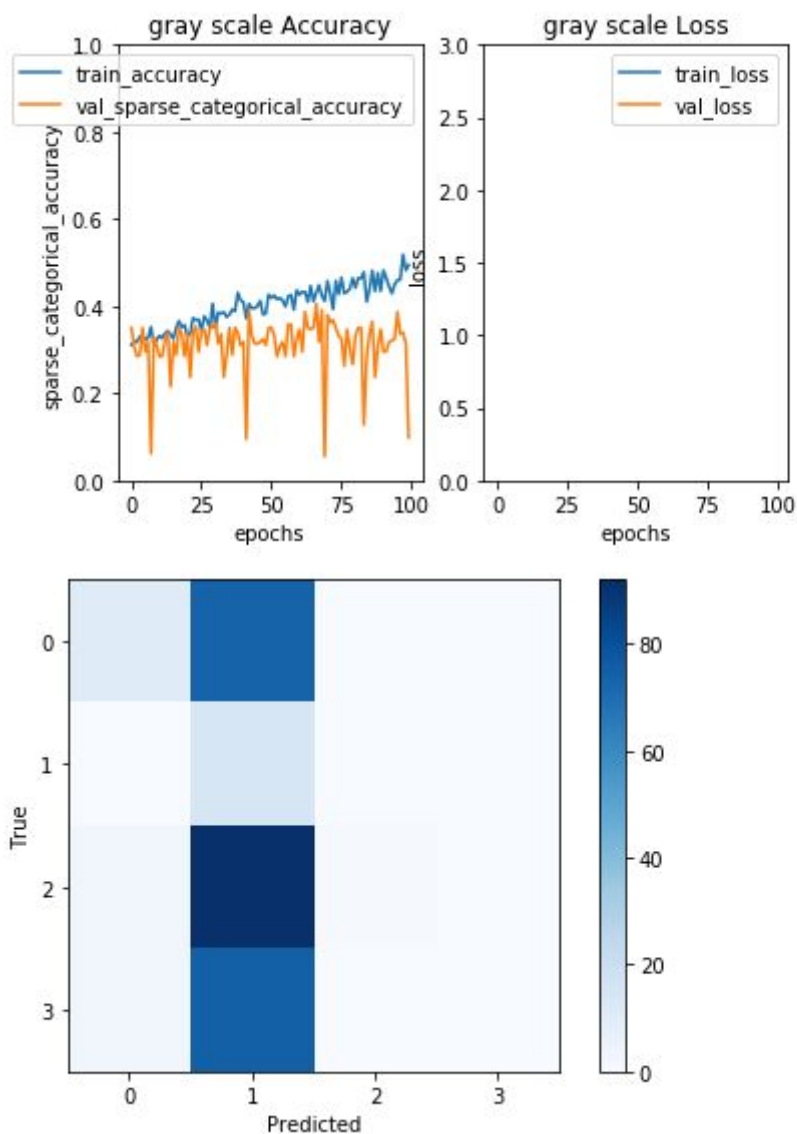
dropout 0.5



On retrouve le problème du premier modèle. Il se focalise sur la 3 ème classe qui est sur représenté

Nous tentons de passer nos images en nuances de gris





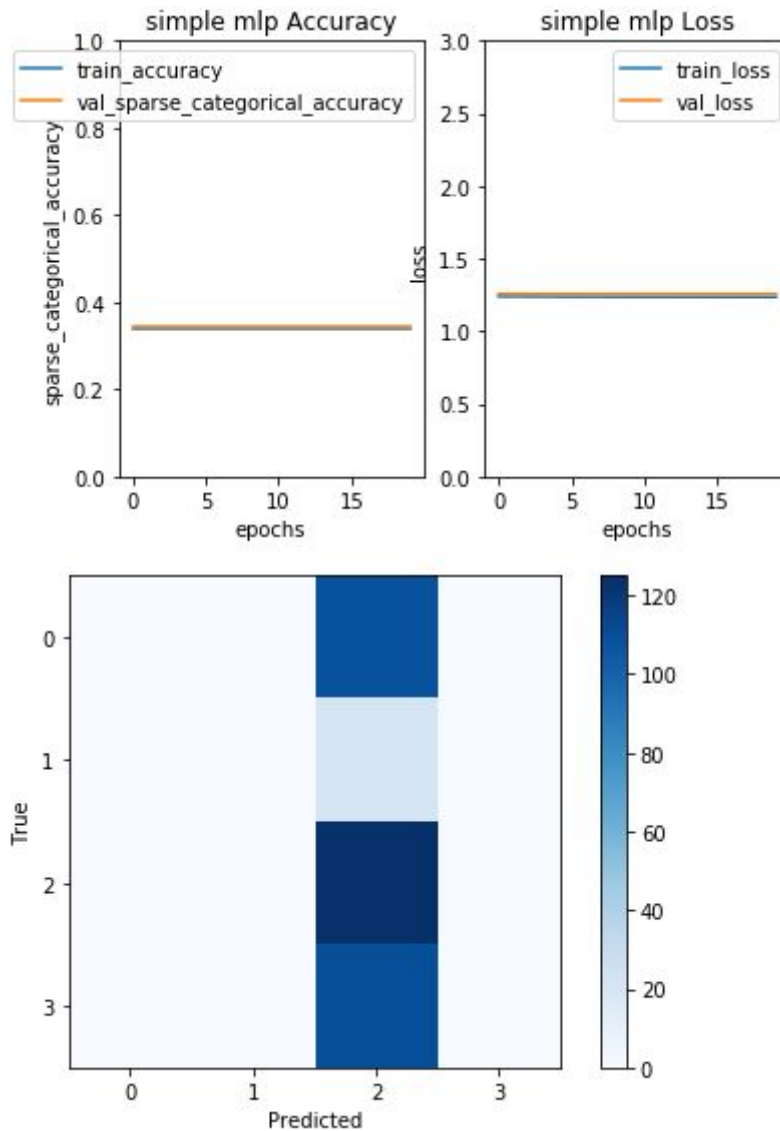
Le modèle ne retourne que “healthy” la classe sous représenté.

La nuance de gris a certainement uniformisé les feuilles et rendu invisible une pathologie.

Le modèle linéaire est à peine plus performant qu’un random. Nos modèles sont bien trop influencés par l’hétérogénéité de notre dataset.

III- MLP

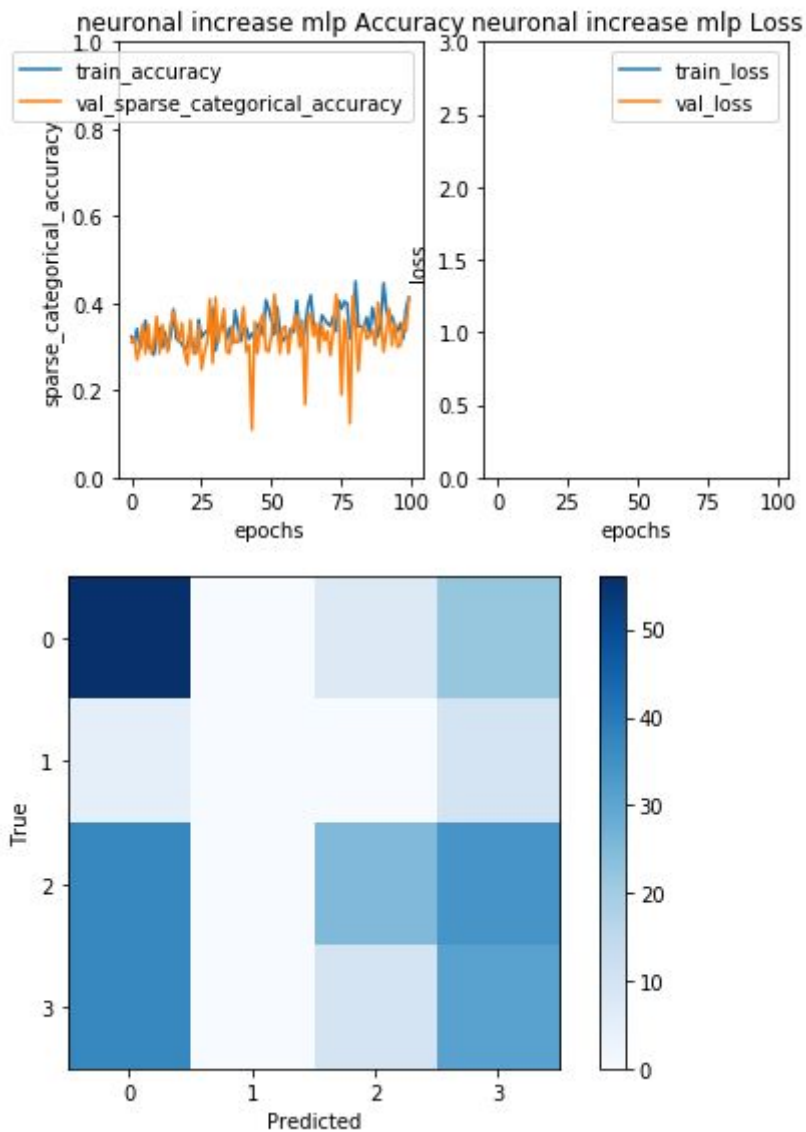
Une couche cachée de 20 neurones activation relu



Le modèle stagne. Il ne retourne que la classe rust. La couche caché semble augmenter l'impact de l'hétérogénéité du dataset.

Nous allons augmenter le nombre de neurones pour améliorer l'apprentissage

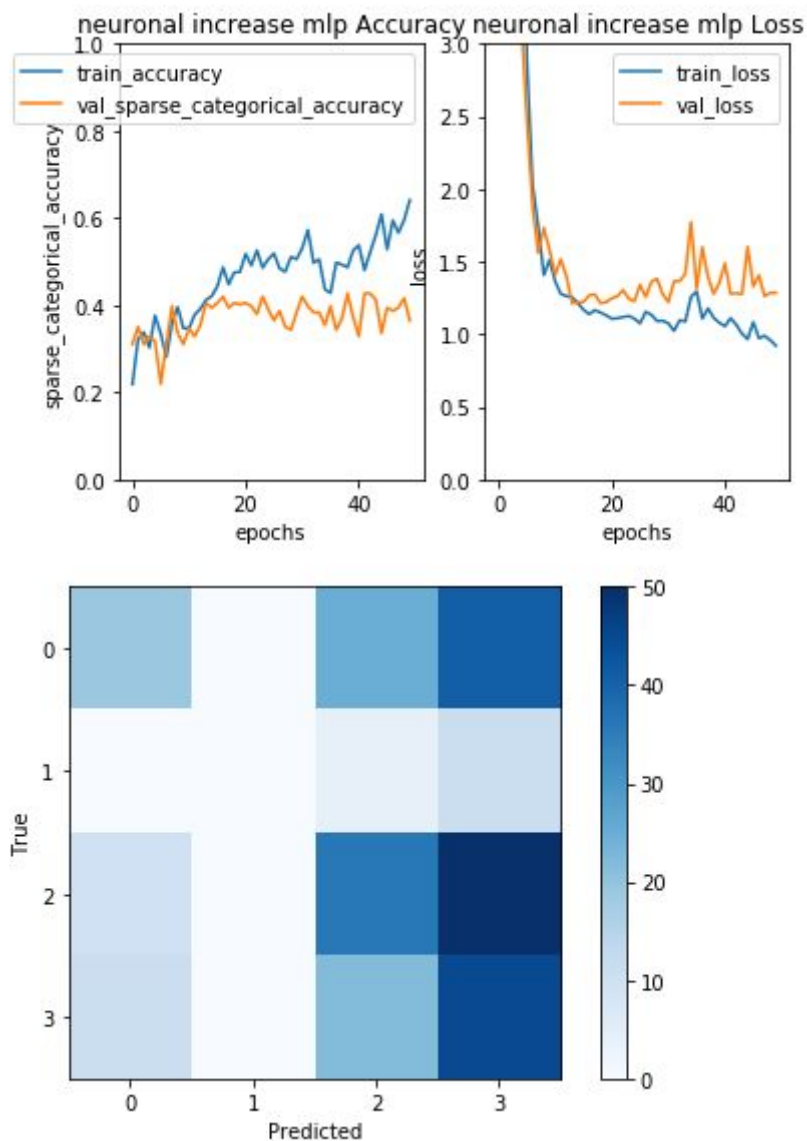
Une couche cachée de 132 neurones activation relu



Le modèle distingue pour la première fois 3 classes. La classe sous représenté "healthy" n'est toujours pas assimilée par le modèle. Il n'y a pas d'overfitting mais l'accuracy suit une forte variance.

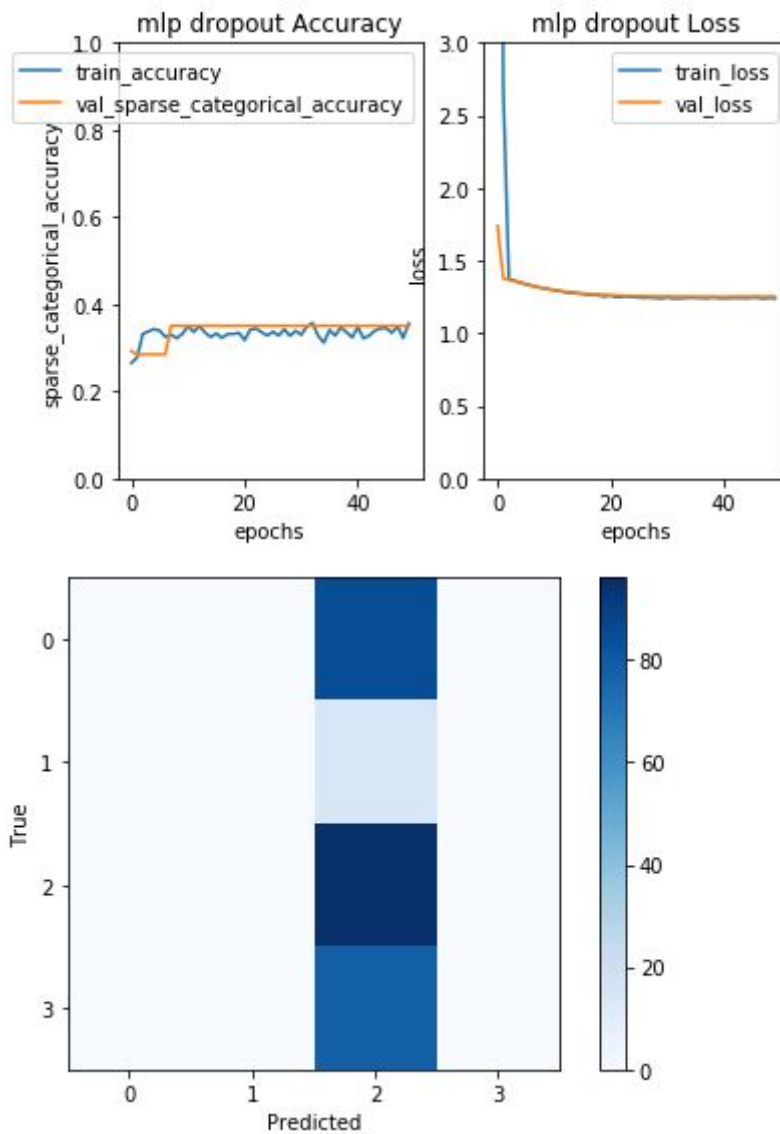
Le modèle distingue la pathologie dans 40% des cas, il commence à être plus performant qu'un random.

4 couches cachée de 64 neurones activation relu



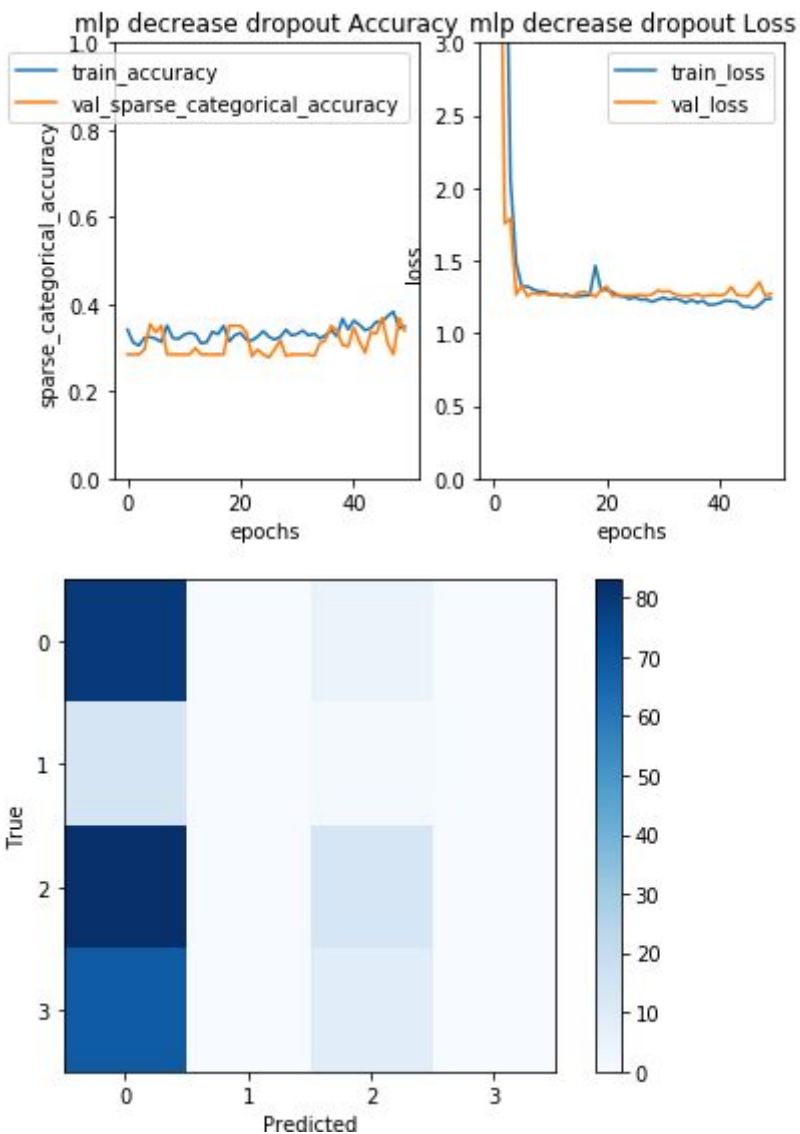
La classe healthy est toujours inconnue pour notre modèle. Mais les 3 autres classes commencent à être bien détectées. Les performances sur les données d'entraînement sont assez bonnes, mais l'accuracy de test reste bloqué à 40%. Nous allons essayer de rajouter de la régularisation pour augmenter la performance du modèle sur données de test et diminuer l'overfitting.

4 couches cachée de 64 neurones activation relu avec dropout de 0.2 à chaque couche



Nous retournons aux résultats d'un modèle linéaire. Le modèle est focalisé sur la classe surreprésenté.

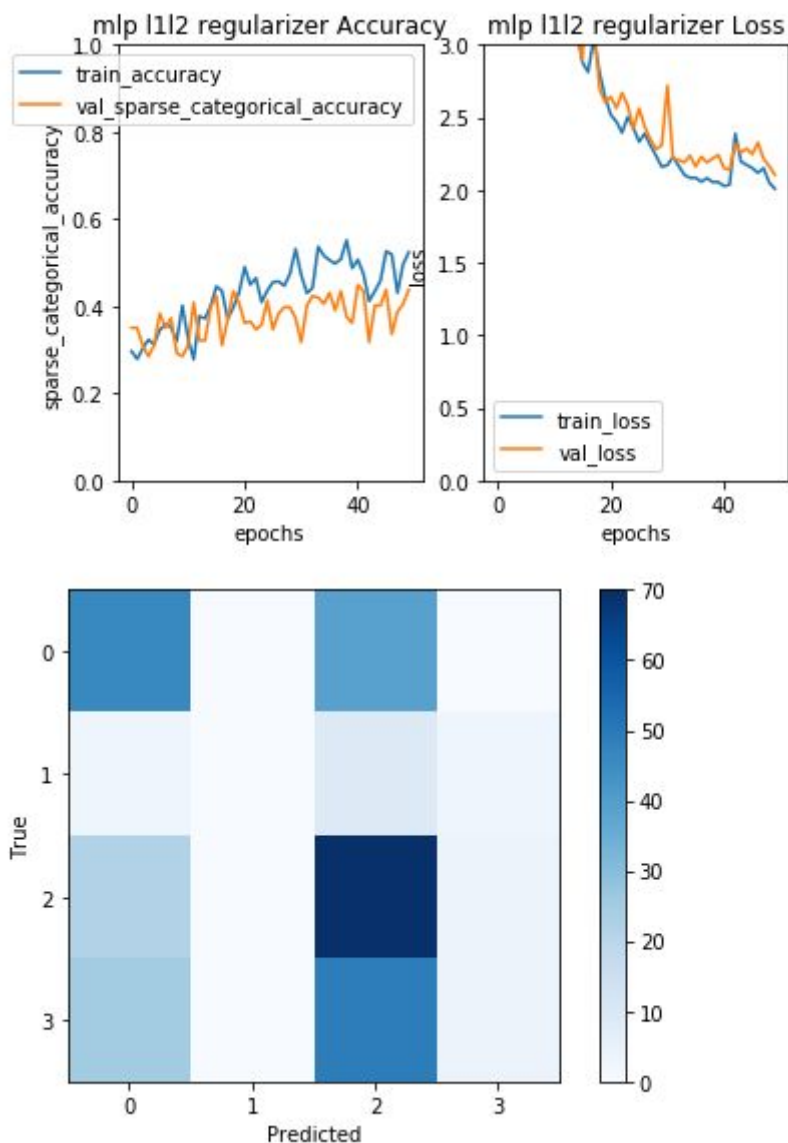
4 couches cachée de 64 neurones activation relu avec dropout de 0.05 à chaque couche



En diminuant le dropout, 2 classes sont distingués, mais le modèle retourne bien trop souvent la première classe.

Jusqu'à présent nous avons vu que les meilleurs modèles sont ceux qui ont le plus de paramètres. Les dropout ne sont donc pas une bonne solution pour pousser les performances de nos données de test.

4 couches cachée de 64 neurones activation relu avec L2 régularisation de 0.01 a chaque couche

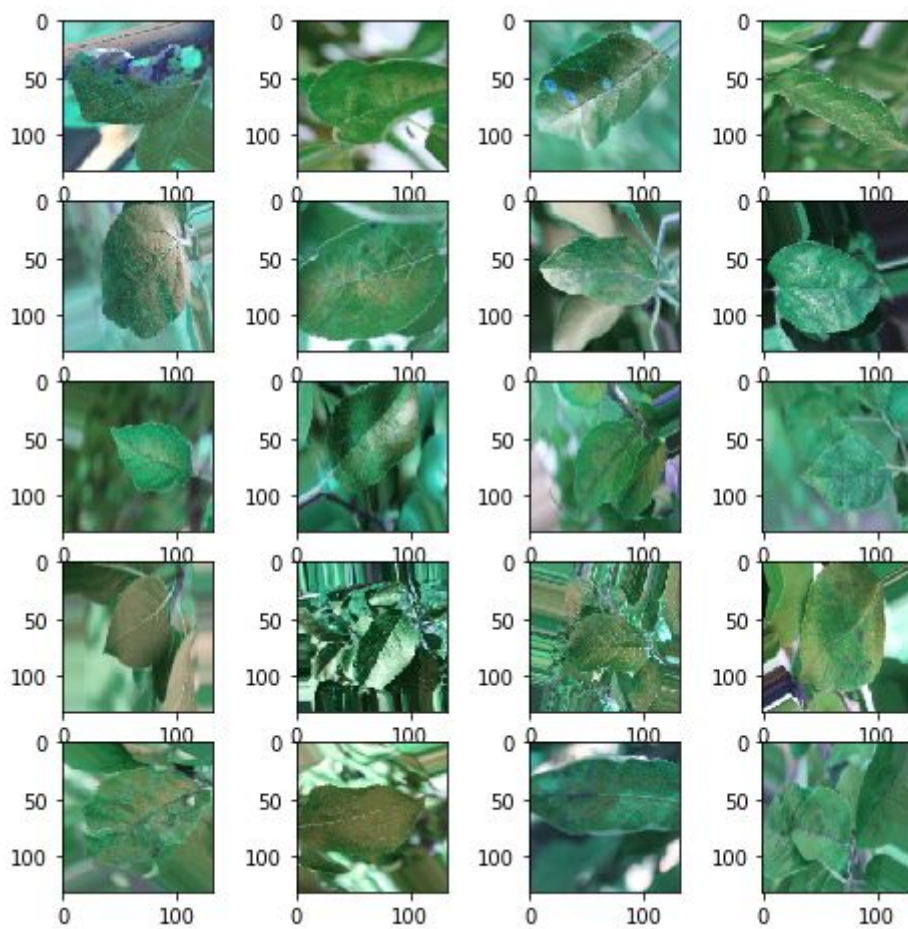


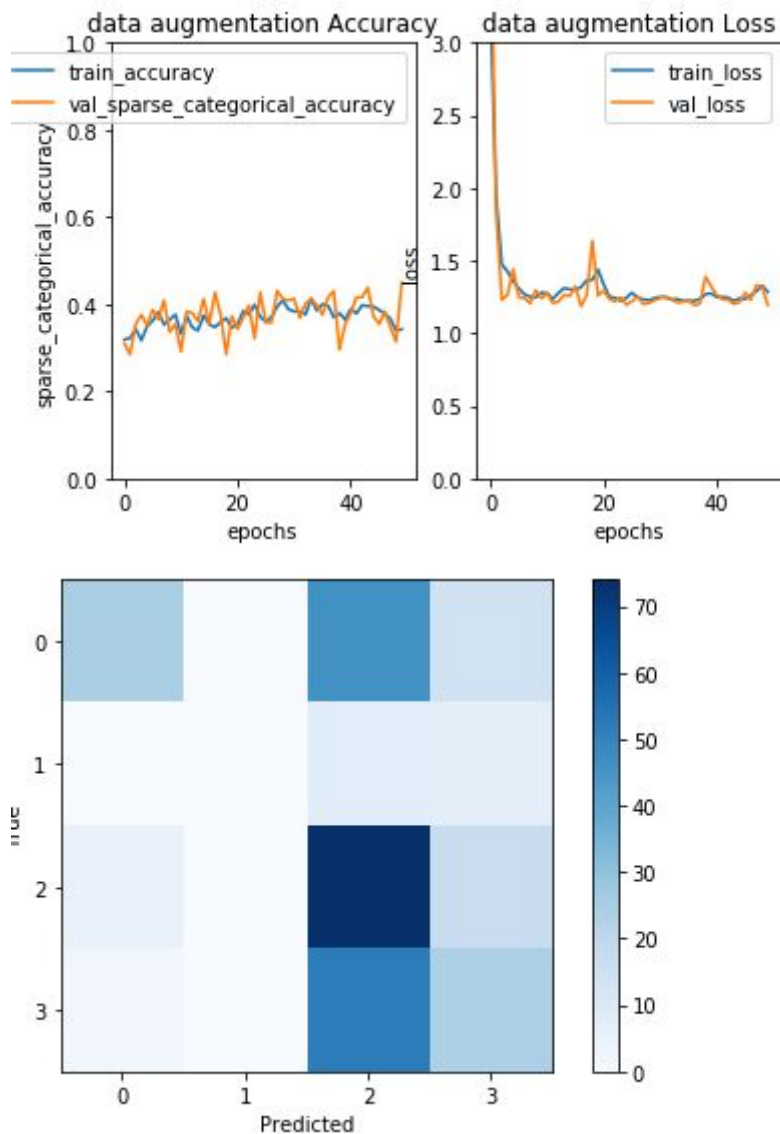
Le modèle est moins overfitté, mais les performances des données de test n'ont que très légèrement augmenter. Mais cette amélioration est certainement expliquée par la composition du dataset. Notre modèle retourne quasiment tout le temps la première et la 3eme classe.

Les différentes techniques de régularisation ne fournissent pas de résultats satisfaisants. Nous allons essayer d'accroître nos performances avec l'augmentation de données.

4 couches cachées de 64 neurones activation relu avec data augmentation (photos 132 x 132)

Nous avons mis l'accent sur le zoom et la rotation





Les résultats sont un peu plus satisfaisant que lors de l'entraînement sur les données originales. La performance de nos machines nous ont obligé à diminuer la taille de nos images.

Un perceptron multicouche peut surement fournir de bons résultats en augmentant le nombre de paramètres. Mais nos machines ne nous permettent pas de pousser les modèles sans trop réduire la dimensions des images. Avec d'autre architecture comme un CNN ou un Resnet, nous pourrons peut être obtenir de bons résultats avec moins de traitement.

IV- CNN

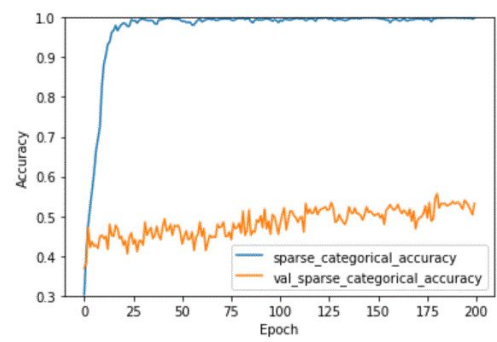
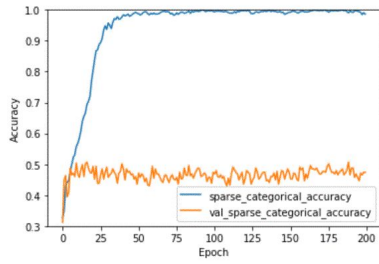
Paramètres qui varie selon les modèles :

	Nombre de Conv2D	Nombre de filtre + Kernel_size	Nombre de Maxpooling2D	Nombre de DropOut	Valeurs des DropOut	Nombre de couche Dense	Units	Nombre d'epoch	Résultat
Modèle 1	3	(64,(32,32))	3	3	0.3 0.4 0.5	2	128 4	200	Accuraccy : 0.9863 Val_Accuraccy : 0.4740
Modèle 2	2	(128,(32,32))	2	2	0.1 0.1	1	4	200	Accuraccy : 0.9904 Val_Accuraccy : 0.4466
Modèle 3	3	(64,(32,32))	3	3	0.2 0.2	2	128 4	180	Accuraccy : 0.9986 Val_Accuraccy : 0.3945
Modèle 4	3	(64,(32,32))	3	3	0.3 0.3 0.3	1	4	200	Accuraccy : 0.9959 Val_Accuraccy : 0.5370
Modèle 5	3	(64,(32,32))	3	3	0.3 0.3 0.3	2	128 4	200	Accuraccy : 0.9993 Val_Accuraccy : 0.4548
Modèle 6	3	(64,(32,32))	3	3	0.4 0.4 0.4	2	512 4	200	Accuraccy : 0.9993 Val_Accuraccy : 0.3973
Modèle 7	3	(64,(32,32))	3	3	0.4 0.4 0.4	2	128 4	200	Accuraccy : 0.9993 Val_Accuraccy : 0.4658
Modèle 8	3	(64,(32,32))	3	3	0.5 0.5	1	4	200	Accuraccy : 0.9904 Val_Accuraccy : 0.6767

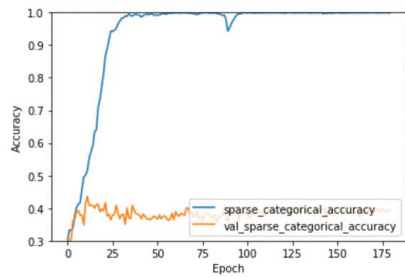
					0.5				
Modèle 9	2	(128,(32,32))	2	2	0.4 0.4 0.4	2	512 4	200	Accuraccy : 0.9986 Val_Accuraccy : 0.4767
Modèle 10	3	(128,(32,32))	3	4	0.4 0.4 0.4 0.5	2	512 4	200	Accuraccy : 0.9986 Val_Accuraccy : 0.4767
Modèle 11	3	(128,(32,32))	3	4	0.5 0.5 0.5 0.5	2	256 4	200	Accuraccy : 0.9828 Val_Accuraccy : 0.4849
Modele 12	3	(64,(32,32))	3	3	0.5 0.5 0.5	1	4	1000	Accuraccy : Val_Accuraccy :
Modél 13	3	(64,(32,32))	3	3	0.5 0.5 0.5	1	4	1500	Accuraccy : Val_Accuraccy :
Modél 14	3	(128,(32,32))	3	3	0.5 0.5 0.5	1	4	400	Accuraccy :0.9993 Val_Accuraccy :0.6932

Modèle 1

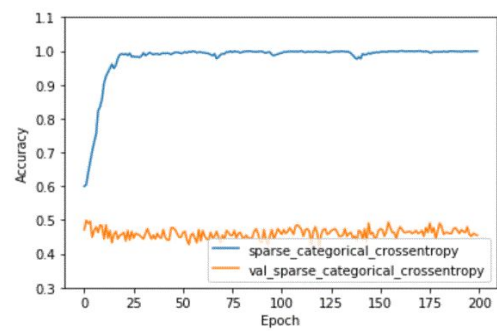
Modèle 2



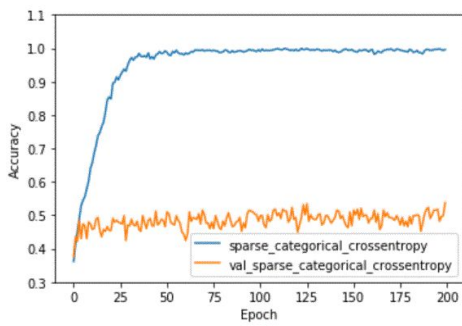
Modele 3



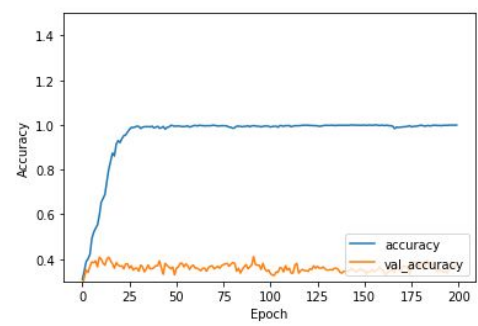
Modele 4



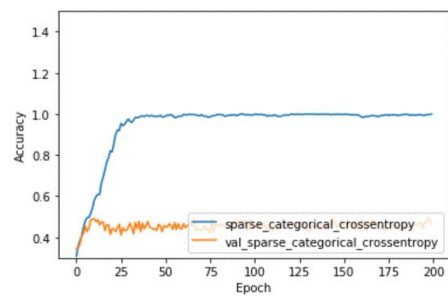
Modele 5



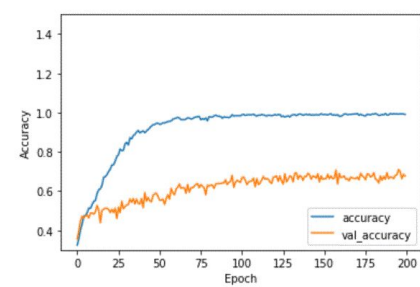
Modele 6



Modele 7

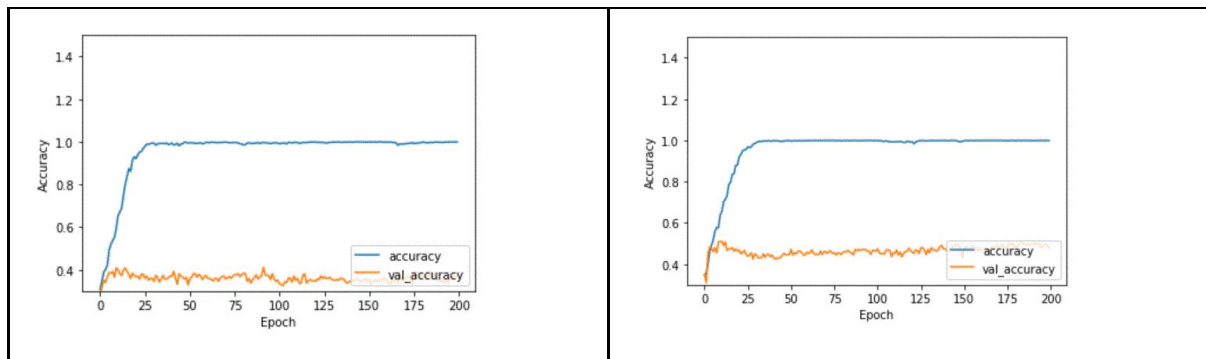


Modele 8



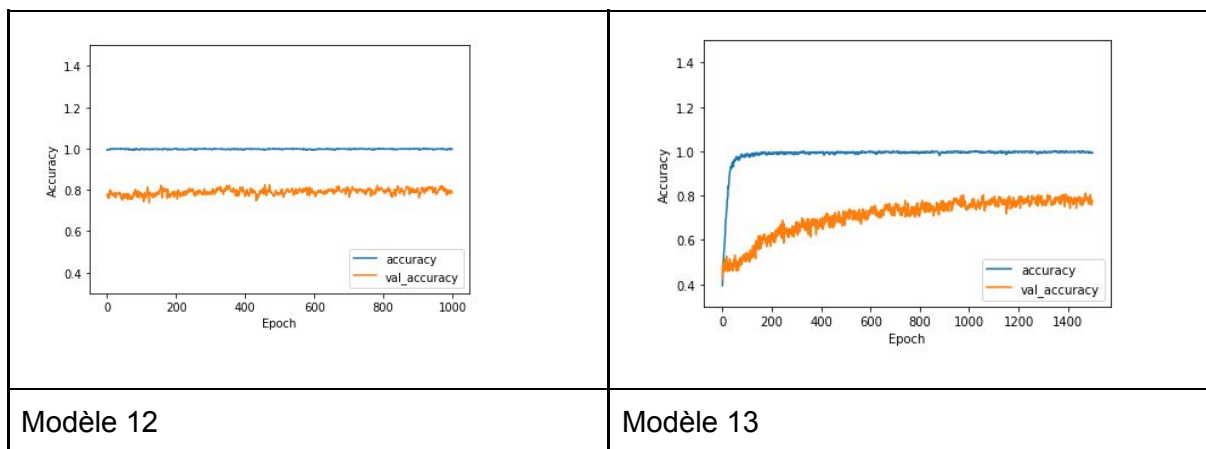
Modele 9

Modele 10

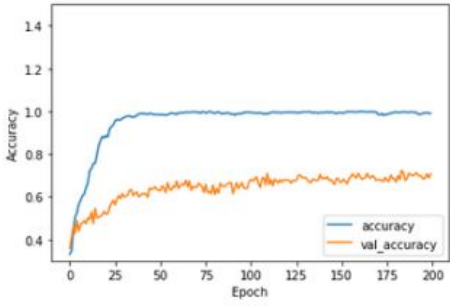
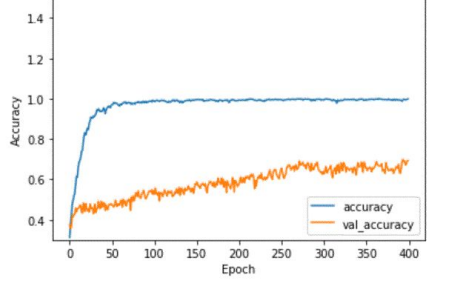


Après avoir tester les modèles de manière assez aléatoire, modèles qui sont dans le tableau au dessus nous avons constaté que le modèle 8 est celui qui overfit le moins, pour le coup on à décider d'augmenter le nombre d'époch à 1000 puis 1500 et on remarqué une amélioration que le voit dans les graphes ci-dessous.

Modèle



suite à ces résultat on a procédé à l'agrandissement des images à (128X128) mais nous n'avons pas pu aller plus loin que les 400 epoch faute de matériel, mais nous n'avons pas constaté une réel amélioration par rapport au modèle 8, on en a conclu qu'il faudrait peut être plus d'epochs.

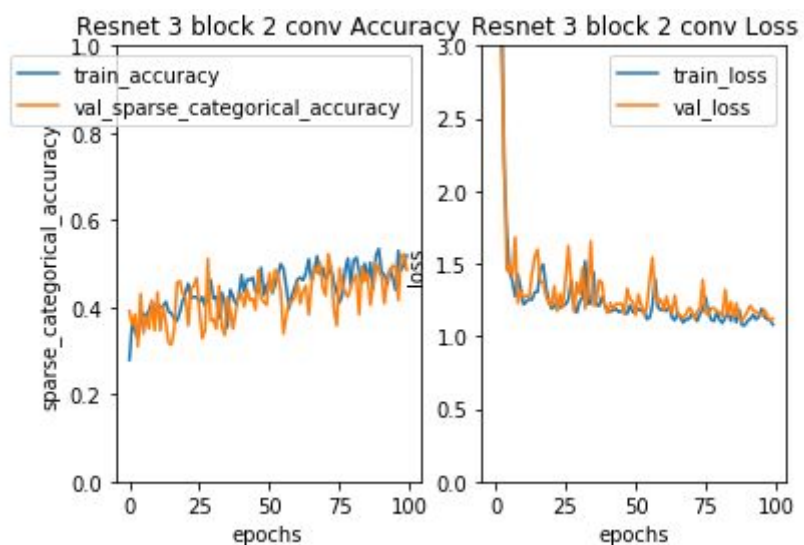
 <p>This plot shows the training and validation accuracy for 'Modèle 11' over 200 epochs. The x-axis is labeled 'Epoch' and ranges from 0 to 200. The y-axis is labeled 'Accuracy' and ranges from 0.4 to 1.4. The training accuracy (blue line) starts at approximately 0.4 and rises sharply to about 1.0 by epoch 25, then remains stable. The validation accuracy (orange line) starts at approximately 0.4 and rises to about 0.7 by epoch 25, then fluctuates between 0.65 and 0.75.</p>	 <p>This plot shows the training and validation accuracy for 'Modèle 14' over 400 epochs. The x-axis is labeled 'Epoch' and ranges from 0 to 400. The y-axis is labeled 'Accuracy' and ranges from 0.4 to 1.4. The training accuracy (blue line) starts at approximately 0.4 and rises sharply to about 1.0 by epoch 50, then remains stable. The validation accuracy (orange line) starts at approximately 0.4 and rises to about 0.6 by epoch 50, then fluctuates between 0.55 and 0.7.</p>
Modèle 11	Modèle 14

V- RESNET

Le resnet permet de créer des modèles très profonds tout en convergeant rapidement vers les bonnes prédictions. Il peut potentiellement être bien plus efficaces que nos perceptrons multicouches limités en paramètre par nos machines

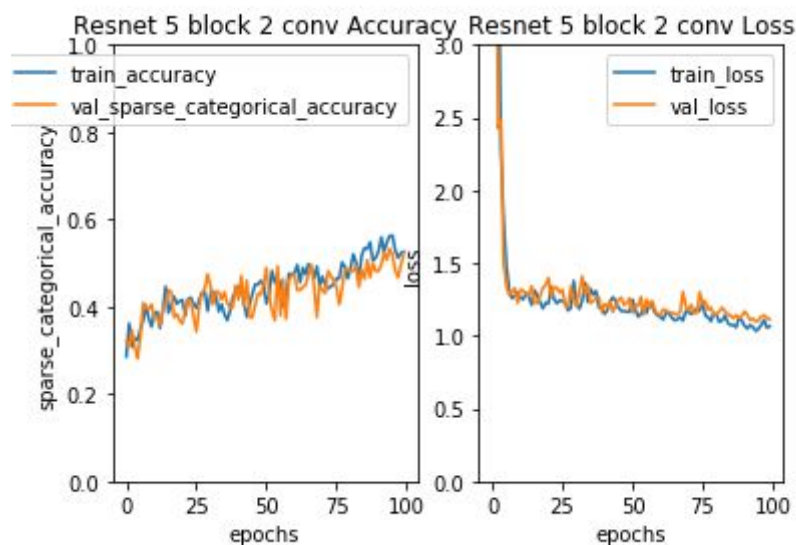
Nous commençons par un modèle simple de 3 blocs de 2 couche de convolutions

Resnet 3 blocs 2 couches de convolutions (photos 64 x 64)



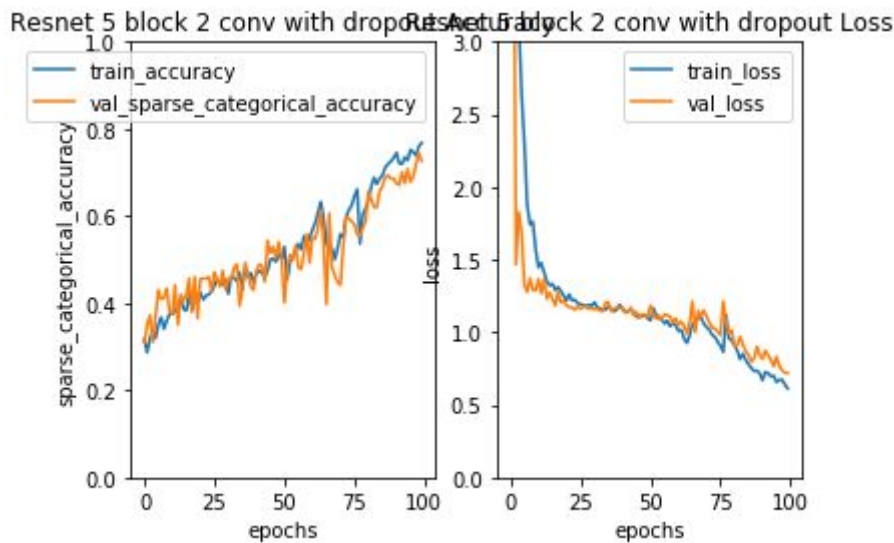
Les résultats sont encourageant, l'accuracy sur les données de test est de 48%. Les performances continue de croître à chaque epochs contrairement aux perceptrons multicouche qui stagnaient rapidement..

Resnet 5 blocs 2 couches de convolutions (photos 64 x 64)



L'accuracy sur les données de test dépassent les 50%. Augmenter le nombre d'epochs est envisageable pour augmenter nos performances. Nous allons avant essayé des techniques de normalisation.

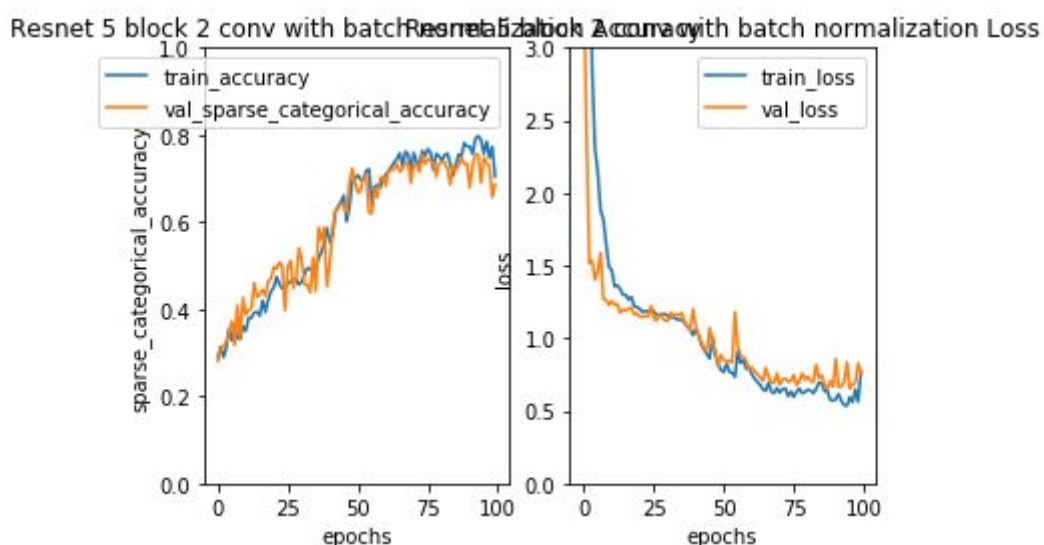
Resnet 5 blocs 2 couches de convolutions avec dropout 0,1 à chaque couche (photos 64 x 64)



Les performances ont très nettement augmenter. Cette augmentation est difficilement explicable, notre modèle n'était pas overfitté. Il semble y avoir eu une cassure lors de l'epochs 80 où la diminution marginale de notre loss semble augmenter. Augmenter le nombre d'epochs pourrait considérablement augmenter nos performances.

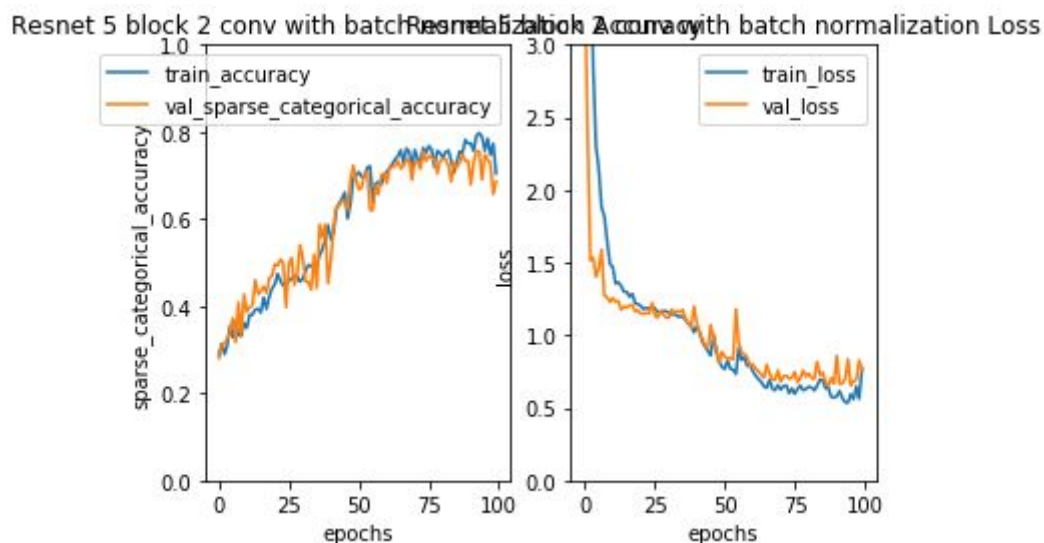
Nous rajoutons une batch normalisation pour stabiliser le modèle.

Resnet 5 blocs 2 couches de convolutions avec dropout 0,1 et batch norm à chaque couche (photos 64 x 64)



Le modèle semble atteindre un plateau à l'épochs 60. Les résultats sont un peu moins satisfaisant que le précédent modèle.

Resnet 5 blocs 2 couches de convolutions avec dropout 0.1 à chaque couche (photos 64 x 64) 200 epochs



Ici, l'augmentation du nombre d'epochs n'améliore pas le modèle tout en accentuant l'overfitting.

VI-RNN

Pour ce modèle, nous avons implémenté une classe de structure qui nous permet de facilement tester un grand nombre de modèle avec différents paramètres :

```
Entrée [36]: class Lstm:
    def __init__(self):
        self.nb_layers = 3
        self.units = 50
        self.activation = 'tanh'
        self.output_activation = 'softmax'
        self.recurrent_activation = 'sigmoid'
        self.dropout_value = 0.6
        self.kernel_regularizer = None
        self.recurrent_regularizer = None
        self.output_regularizer = None
        self.l1_value = 0.0
        self.l2_value = 0.0
        self.loss = 'sparse_categorical_crossentropy'
        self.optimizer = 'Adam'
        self.metrics = ['sparse_categorical_accuracy']
```

Et Pour créer le modèle, nous avons ensuite implémenté une fonction `create_lstm(lstm_struct: Lstm)` qui va prendre la structure de du modèle et le construire. On a ensuite effectué une serie de test afin d'identifié le meilleur modèle.

1/ LSTM entre 3 et 9 couches sans dropout et régularisation

Pour cette première serie de test, nous avons lancé des modèles avec entre 3 et 9 couches cachés, avec 50 unités de mémoire en les entrainant sur 200 epochs.

nb_layers	accuracy_train	accuracy_val
3	0.9990	0.6265
3	0.9990	0.6406
4	0.9990	0.6305
6	0.9990	0.6044
9	0.9663	0.6104

On remarque très vite que notre modèle donne une excellente accuracy sur le train, mais le modèle overfit beaucoup.

Nous allons donc essayer de lancer des modèles avec la même mémoire, mais cette fois-ci avec un dropout.

2/ LSTM entre 3 et 9 couches avec dropout et régularisation

Cette fois-ci, nous avons essayé d'appliquer différentes valeurs d'epochs et de dropout sur nos couches cachés et aussi :

Nb_layers	Epochs	dropOut	Train_accuracy	val accuracy
3	100	0.2	0.9542	0.6185
3	100	0.4	0.8799	0.6084
3	100	0.6	0.7447	0.5904
4	100	0.4	0.8683	0.6365
4	100	0.6	0.7111	0.6004

6	100	0.2	0.8965	0.5884
6	100	0.4	0.7950	0.6145
6	100	0.6	0.6693	0.5482
6	100	0.8	0.4940	0.3899
9	100	0.4	0.7095	0.6064
9	100	0.6	0.9416	0.5837
9	100	0.8	0.4241	0.3896

On constate qu'avec le dropOut, on arrive à réduire considérablement l'overfitting, mais les modèles avec 3 couches overfit beaucoup. Les modèles avec 6 et 9 couches nous donnent de bons résultats, mais on a quand même certains modèles qui overfit.

On tire quand même un des modèles qui nous donne une bonne accuracy sans trop d'overfit avec 9 couches cachées et un dropout à 0.4.