Etude du dataset CIFAR-10

Victor Meyer, Zohir Malti, Cécile Cousin

Projet Deep Learning Rendu 1

2020

Sommaire

1.0 - Quelques bonne pratiques à utiliser avant l'entraînement :

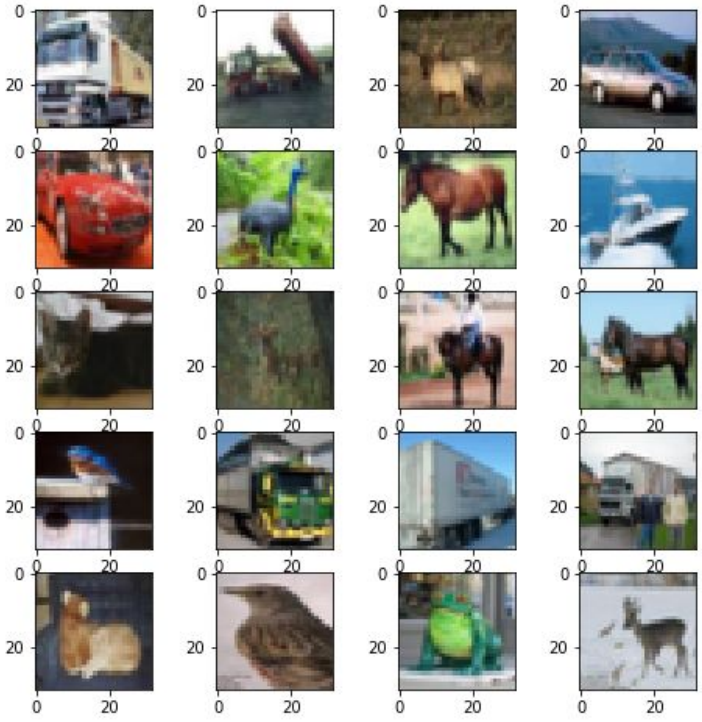
Feature Scaling

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

lr_scheduler: propose plusieurs méthodes afin de régler le taux d'apprentissage.
Early stopping: Arrêter l'entraînement lorsque celui ci arrête de s'améliorer.
Tensorboard: Avoir accès à des outils qui permettront un suivi de l'entraînement (loss, accuracy, visualisation).

1.1 - Présentation du dataset

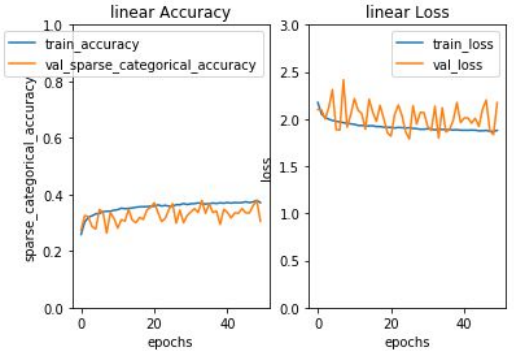
CIFAR10 est un dataset composé de 60000 images. 50000 pour l'entraînement et 10000 pour le test. Il est composé de 10 classes uniformément distribuées. Les images représentent des animaux et plusieurs types de véhicules.

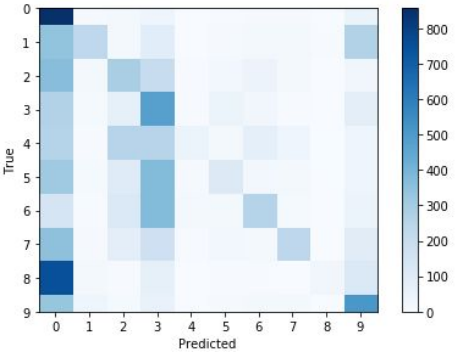


Modèle Linéaire

2.0 - Présentation

Tout d'abord, nous commençons par un réseau sans couches cachées: un paramètre par information et les biais. Avec une activation linéaire de la couche de sortie, les prédictions du modèle étaient majoritairement 1 et 9 et les performances dépassent à peine celles d'un random. Avec la softmax les prédictions tendent majoritairement vers 0, mais les résultats des sont assez cohérents.





On peut également souligner la variance très élevée des loss et accuracy de test.

Le linear discriminante analysis donne le moins bon résultat. Cependant les classes moyens de transports sont bien reconnus. On peut soupçonner un apprentissage par coeur.

Pour tous les modèles, un dataset en nuances de gris donnent des résultats moins satisfaisant.
Un modèle linéaire n'est pas adapté. On peut supposer que notre dataset de 10 classes d'images de 3072 paramètres n'est pas linéairement séparable.

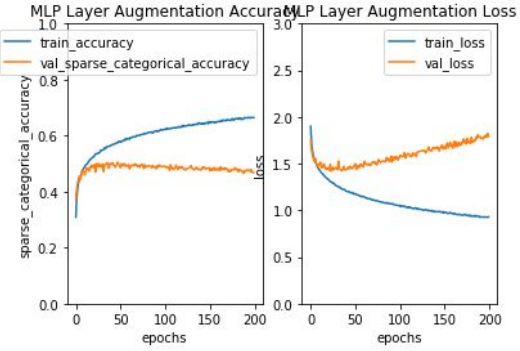
Perceptron multicouche

3.0 - Modèle de base

Nous rajoutons deux couches cachées de 32 neurones à notre réseau initial. Les résultats sont à peine plus satisfaisant qu'un modèle linéaire simple, mais la variance des loss a très nettement diminué. Les classes flagrantes sont toujours mieux reconnus que les animaux, notamment les oiseaux qui sont très mal reconnu.

En doublant le nombre de neurones par couche l'accuracy sur les données d'entraînement et de test grimpent nettement. Cependant notre modèle commence à overfitté.

En doublant le nombre de couches cachées, le modèle donne de bons résultats sur les données d'entraînement, la loss est pour la première fois inférieur à 1.



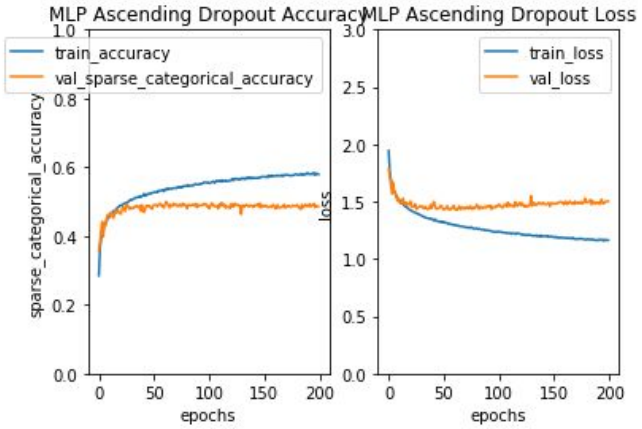
MLP 4 couches cachées 64 neurones

Mais notre modèle est clairement overfitté. Sur les données de test les résultats sont moins bons que le précédent paramétrage.

Nous allons donc tenter de réduire cet overfitting.

3.1 - DROPOUT

Nous testons l'ajout de dropout. Nous augmentons le taux à chaque couches.

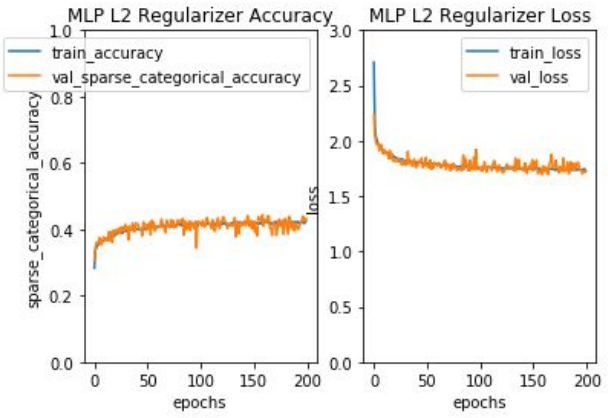


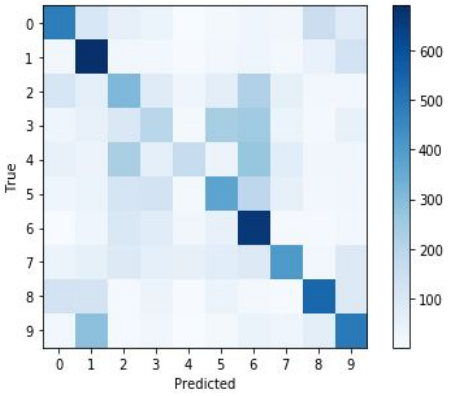
Notre modèle est moins overfitté, notre accuracy sur les données de test à augmenter et certaines catégories d'animaux commencent à être bien distingué.

Avec un dropout décroissant les résultats sont assez similaires bien qu'un peu moins performant.

3.2 - L1 / L2 Régularisation

L'ajout d'une L2 régularisation donne un modèle aussi efficace sur des données de tests que d'entraînement. Cependant les résultats sont proches de ceux des modèles simples initiaux et les prédictions sont, anormalement, trop souvent des grenouilles.





Les régularisations L1 ou L1/L2 fournissent de mauvais modèles, même en faisant varier le taux. Une itération supplémentaire n'affecte pas l'évolution du loss et de l'accuracy et reste à peine plus performante qu'un random.

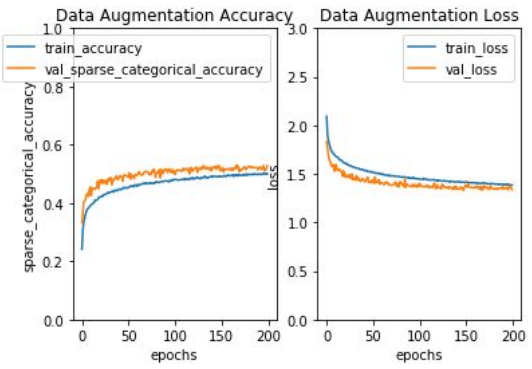
La régularisation par dropout croissant nous fournit le meilleur modèle. Les prédictions sont assez bonnes et il n'y a pas d'overfitting .Nous allons maintenant essayer de pousser nos performances en augmentant nos données.

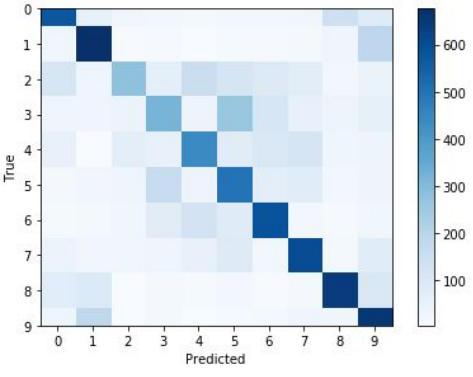
3.3 - DATA AUGMENTATION

Nous augmentons nos images avec la classe ImageDataGenerator. Nous mettons une rotation et un zoom assez faibles avec un décalage plutôt élevé.

Contrairement à nos précédent modèles, l'augmentation de nos données conduit à un sous entraînement. Nos données de test fournissent un bien meilleurs résultats que celles d'entrainement. L'ajout d'un dropout n'est donc pas nécessaire.

Nous essayons plutôt de doubler le nombre de nos neurones. Le modèle donne des résultats satisfaisant sur les données de test mais reste légèrement sous entraîné. Les moyens de transports sont très bien reconnus.





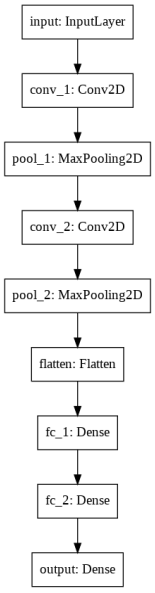
Le temps d'exécution de l'entraînement était en revanche long (2h30).

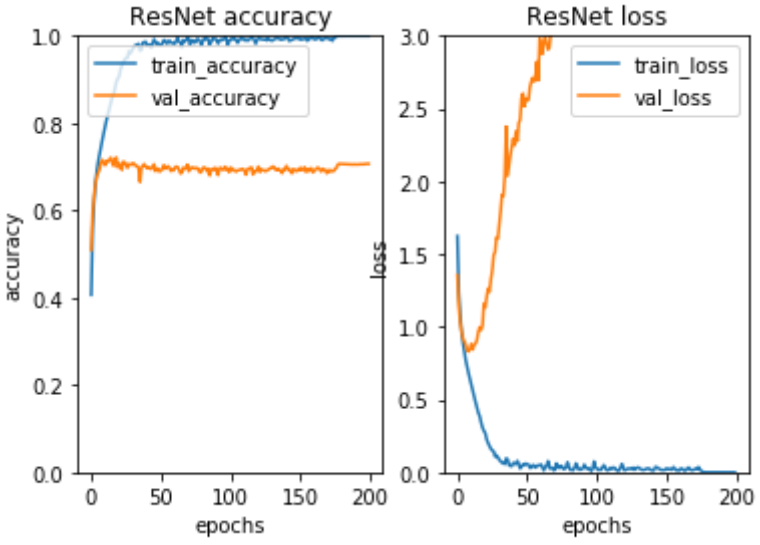
Convolutional neural network :

Le chapitre précédent a montré les limites d'un réseau de neurones artificiels. Arrivé à un certain seuil, ajouter des neurones, empiler les couches cachées ou même utiliser toutes les techniques de régularisations n'apportait pas de réelle amélioration pour notre modèle. C'est pour cela qu'un nouveau type de réseau de neurones est apparu : *Convolutional neural network.* Il apporte quelques nouvelles couches pour assurer une meilleure classification des images en extrant les features des images grâce à des couches de convolutions et pooling. Passons à l'implémentation :

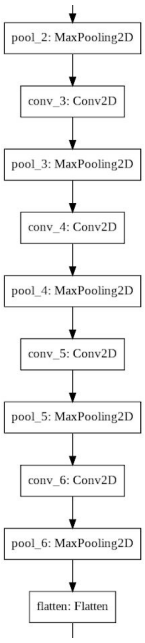
4.0 - Modèle de base :

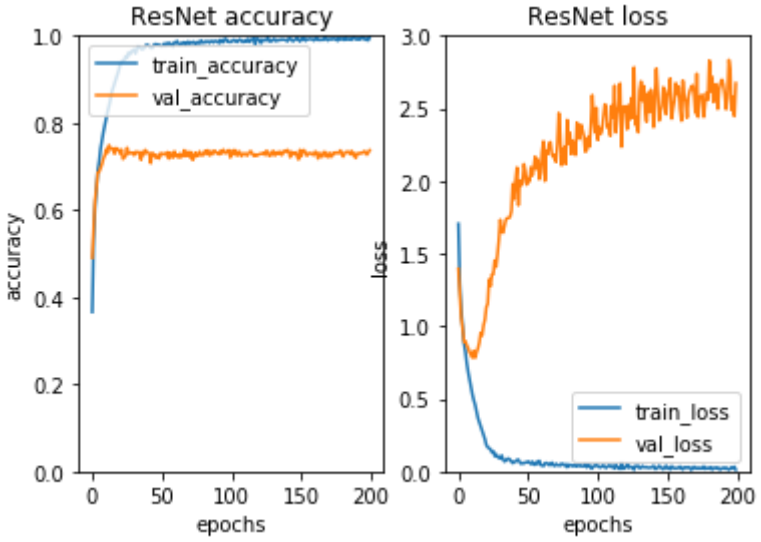
Le modèle défini ci-dessous montre une architecture simple d'un CNN. Les résultats de ses performances démontrent la difficulté du modèle à classifier les nouvelles images, et cela dès la 20éme epoch, où on se retrouve avec de l'overfitting.





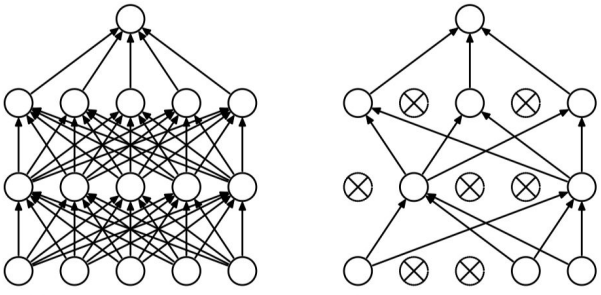
Afin de permettre au modèle d'avoir de meilleures performances, on essaie d'ajouter quelques couches de convolutions suivies de pooling, pour lui permettre de reconnaître les moindres détails de nos images. Cette complexité du modèle lui permet d'apprendre beaucoup plus vite, mais en gardant l'overfitting..





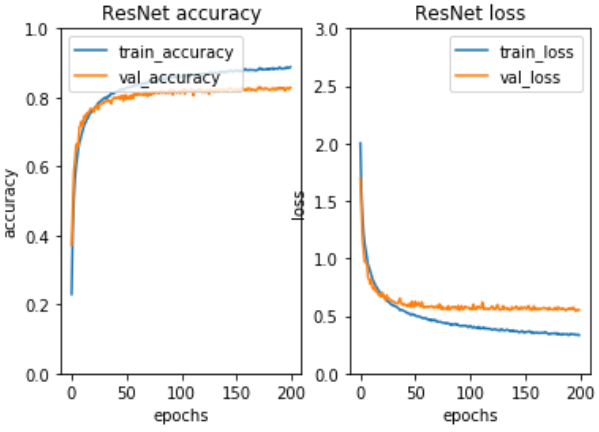
Pour remédier à ce problème, plusieurs techniques de régularisations existent:

4.1 - Dropout : une technique très utilisée, permet de désactiver quelques neurones lors de l'entraînement afin de forcer les autres à mieu apprendre et ainsi éviter le sur-apprentissage.

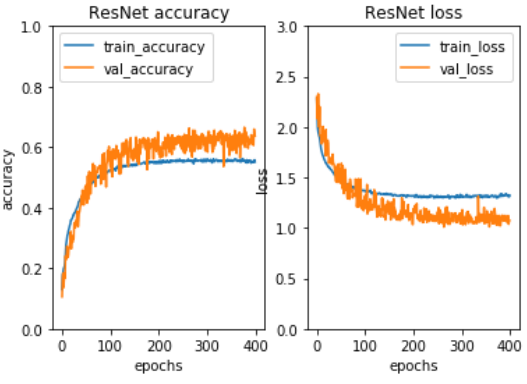


(a) Standard Neural Net (b) After applying dropout.

En ajoutant du dropout de 0.2, ce qui signifie désactiver 20% des neurones lors de l'entraînement, on constate un réel progrès. En effet, le sur-apprentissage a nettement diminué. on peut aussi constater un rapprochement des accuracy et loss entre le train et la validation.

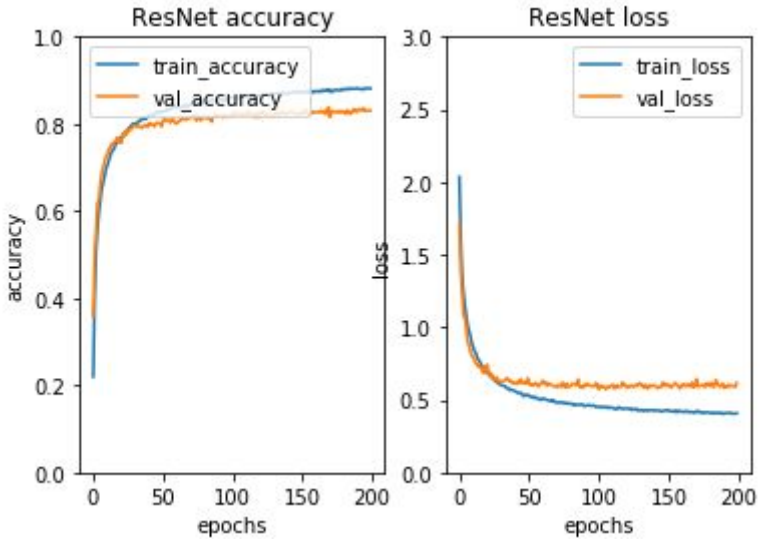


Cependant, en essayant un très grand dropout, de 0.6 par exemple, on peut remarquer que le modèle a beaucoup de mal à s'entraîner avec le reste des neurones.



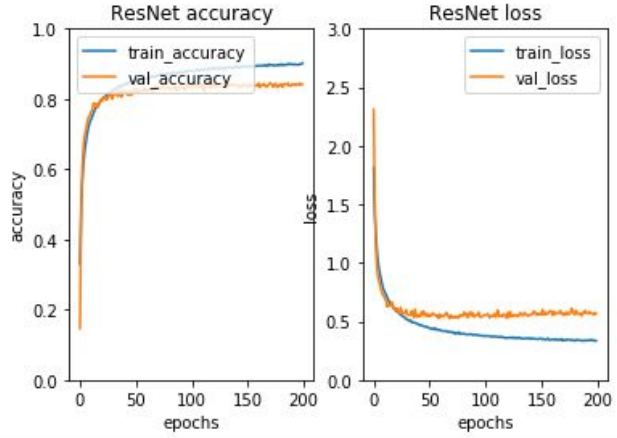
4.2 - L2 regularization :

Nous appliquons donc une régularisation L2 à 0.00001 sur les couches de convolution. De cette manière,nous allégeons les poids des neurones.
Nous nous retrouvons toujours avec un overfitting, plus léger que celui du modèle précédent.



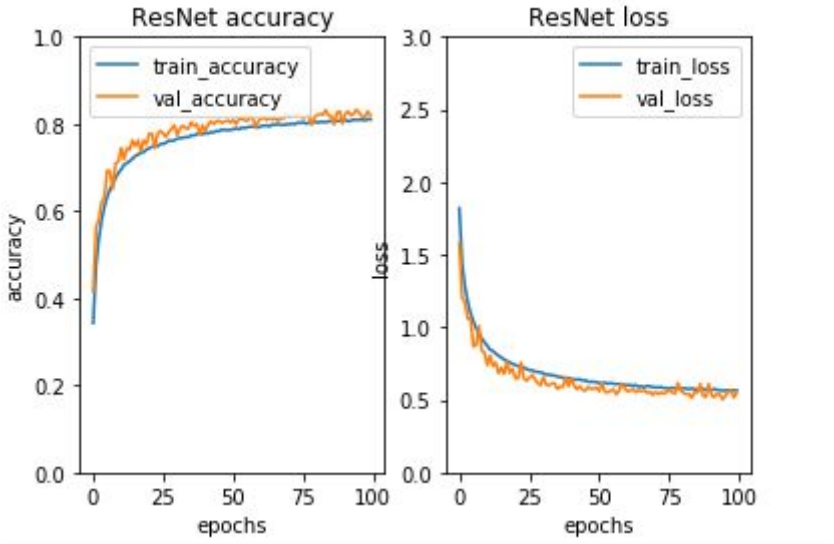
4.3 - L2 + Batchnorm:

Afin de rendre notre entraînement plus performant et d'initialiser les poids, nous ajoutons un batchnorm avec notre régularisation L2. Sans oublier de l'ajouter à la couche flatten.
On constate une très légère augmentation de l'overffiting comparé au modèle précédent.



4.4 - Data Augmentation :

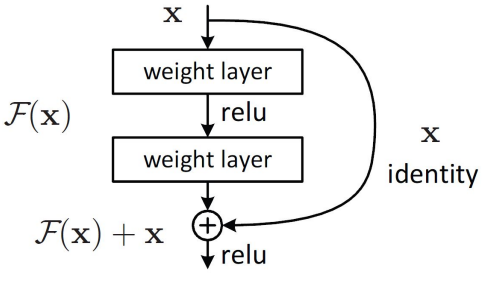
Afin que notre modèle capture mieux les features des images, nous réglons notre data augmentation de la manière suivante. On effectue une rotation de manière horizontale tout en faisant légèrement tourner la longueur et largeur de nos images.
Après cette action, nous avons l'accuracy de train et de test quasiment identiques (0.81).



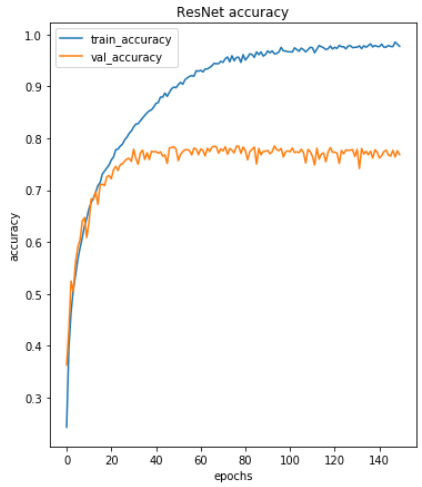
Residual neural network (ResNet) :

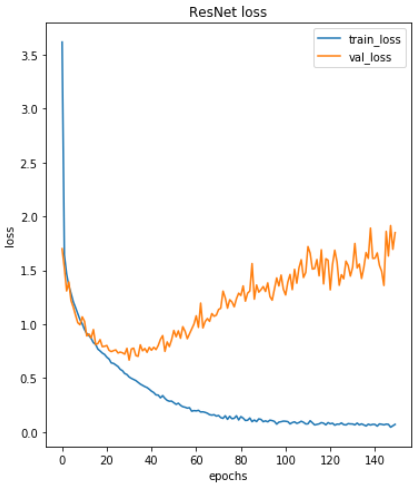
5.0 - Modèle de base

Pour faire face à la problématique du vanishing gradient qui se produit lors de l'entraînement d'un réseau très profond (plus de 30 couches par exemple), un nouveau type de réseau de neurones est apparu, appelé *residual neural network.* Son architecture permet de contourner le problème avec des skip connections (ou shortcut connections), en additionnant la matrice d'identité et la matrice résiduelle (ayant subi des opérations), représentée par la formule suivante : **H(x) = F(x) + x** (x étant la matrice d'identité)

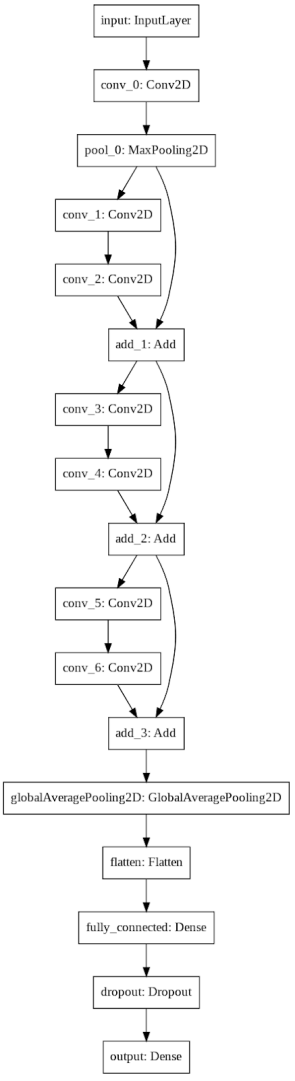


Pour commencer, nous décidons de tester des architectures différentes afin de choisir la plus optimisée. La première est composée de trois blocs de deux couche à convolution.

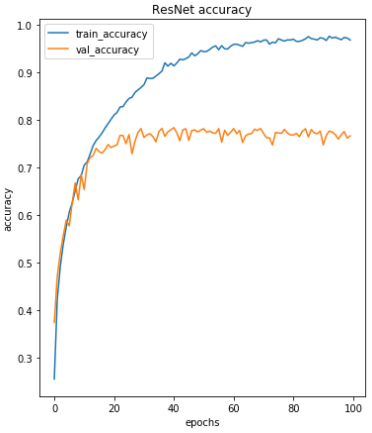


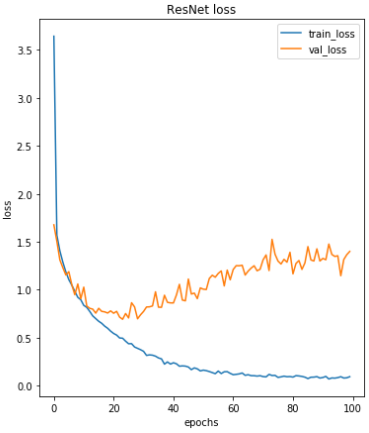


Le résultat précédent montre l'overfitting flagrant de notre modèle, en effet, il a beaucoup de mal à généraliser et reconnaître les nouvelles images du test, c'est pour cela qu'on va essayer de rajouter plus de blocs similaires aux précédents.



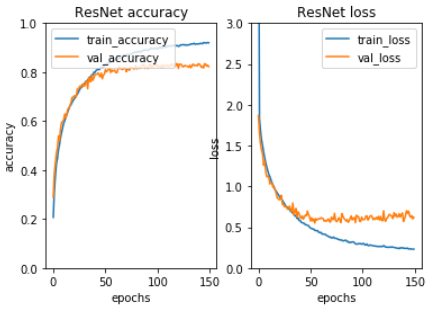
On constate que le loss de validation résiste un peu mieu mais présente toujours un cas d'overfitting, on peut dire que la complexité du modèle n'a pas apporté grand chose, on va essayer de l'accompagner de quelques techniques de régularisations.





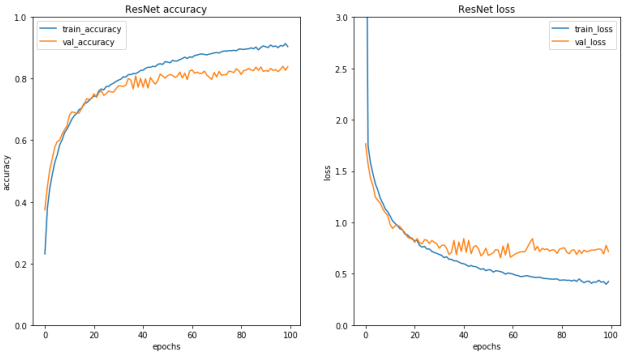
5.1 - Ajout de dropout :

Pour combattre l'overfitting, nous ajoutons un dropout de 0.2 à chaque couche de convolution.
Le résultat s'est nettement amélioré et le modèle se comporte beaucoup mieu avec les nouvelles images, il arrive beaucoup mieu à généraliser et reconnaître 4 images sur 5.



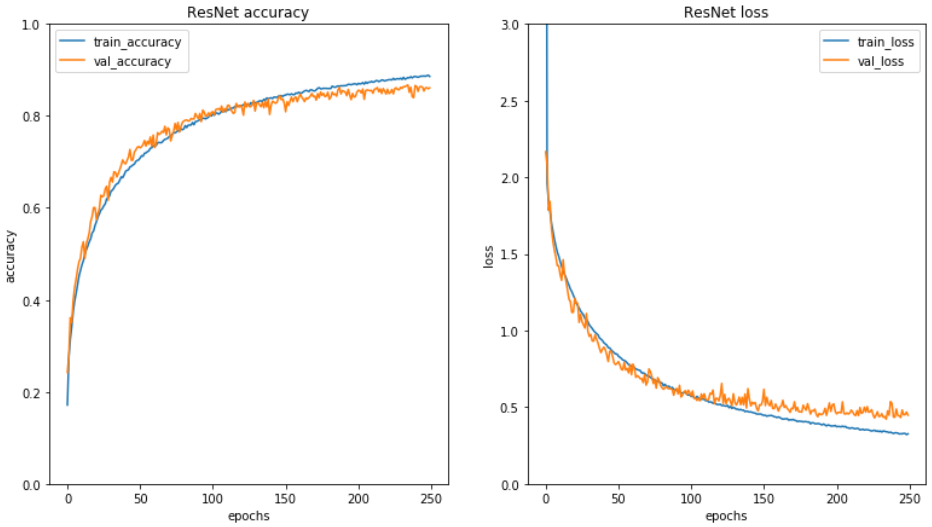
5.2 - Ajout de L2 :

Maintenant que le dropout a bien amélioré notre modèle, on essaie de rajouter la régularisation L2 à chaque couche de convolution.
Pour rappel, la régularisation L2 ajoute une pénalité à la somme des carrés de tous les poids des neurones, elle permet de forcer ces derniers à devenir de plus en plus petits (sans les réduire à zéro). Plus le taux de L2 est grand, plus les poids sont plus petits et plus robustes.
En appliquant une L2 à 0.0001, on peut remarquer ainsi une légère amélioration pour le loss.



5.3 - Ajout de la data augmentation :

Avec l'ajout de la data augmentation à notre modèle, le risque d'overfitting disparaît et cela a permis d'atteindre 87% de validation accuracy à l'epoch 250, en espérant d'avoir de meilleurs résultats en prolongeant les epochs.



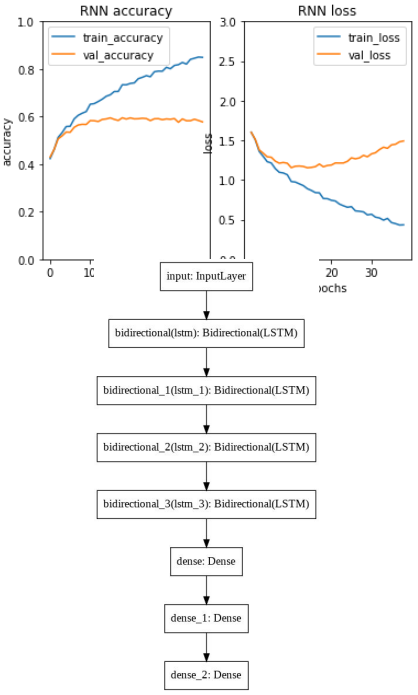
5.4 - Résumé :

Pour résumer, on peut dire que le Resnet apporte beaucoup plus de flexibilité au modèle à convolutions (CNN), pour ainsi nous permettre de travailler avec de plus gros modèles (plus profonds) sans se soucier du problème du *vanishing gradient*.

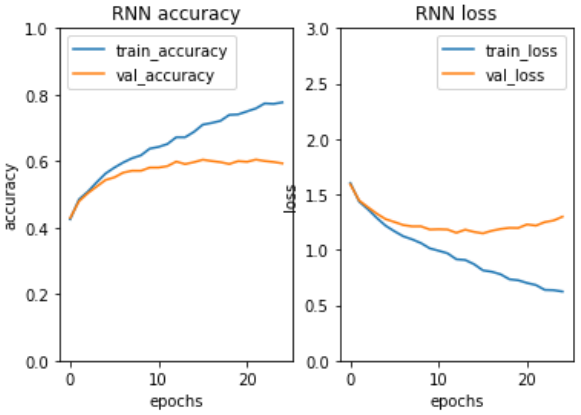
Recurrent neural network (RNN) :

On va à présent essayer le réseau de neurones récurrent, souvent utilisé pour traiter du texte, comme générer un poème, traduction du texte ou encore proposer un titre à des images de bande dessinée.
Le but est d'apprendre ce modèle récurrent à classifier des images à partir d'une séquence de pixels.
Pour notre analyse, on va commencer avec un petit modèle composé de 4 couches LSTM de 32 unités chacunes.

le résultat de la classification est représenté dans la figure suivante, on constate que le modèle présente un cas d'overfitting et ne dépasse pas 55% d'accuracy.

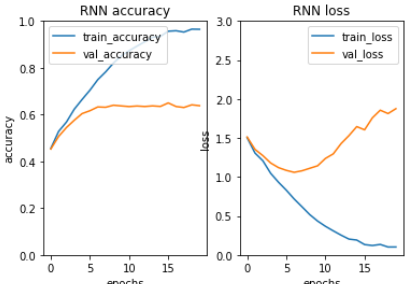


On va à présent ajouter 2 couches Dense (fully connected). L'architecture ci-dessous résume cela.



On peut voir que cela reduit l'ecart entre le loss de train et de validation, en d'autre terme cela a réduit l'overfitting, on peut déduire aussi que les couches fully connected permettent au modèle une meilleure stabilisation et un assure un meilleur contrôle (pour ne pas deraper en overfitting).

On va à présent passer à 128 unités pour les LSTM.



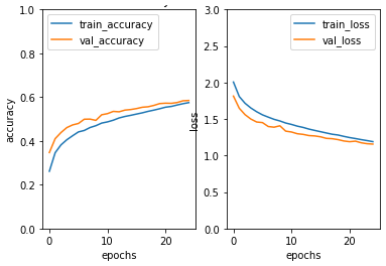
on peut voir qu'en augmentant le nombre d'unités des LSTM, cela offre plus de possibilités de reconnaître les features de nos images (amélioration de train accuracy), cependant, cela fait exploser le modèle dans un overfitting conséquent.

L'étape suivante consiste à utiliser les techniques de régularisations pour ainsi diminuer le risque de sur-apprentissage.

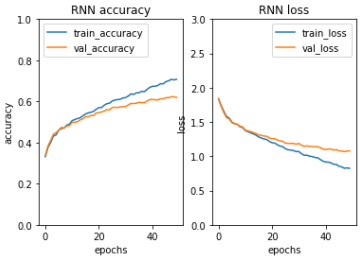
Ajout de dropout :

Maintenant qu'on a vu que le modèle réagit mieu avec 128 unités de LSTM plus qu'avec 32, on va appliquer différents taux de dropout :

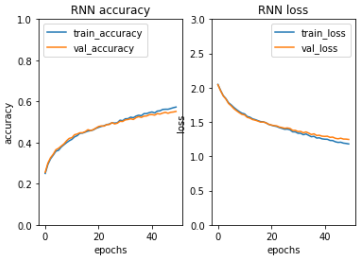
dropout 0.1



dropout 0.2



dropout 0.4

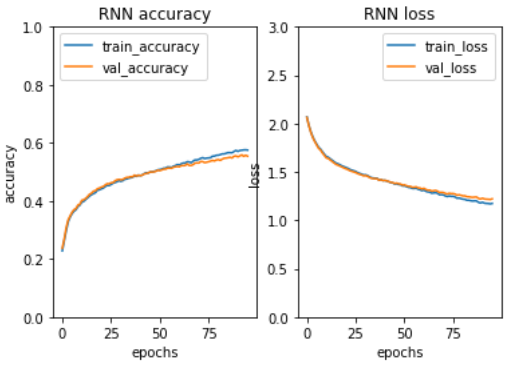


Analyse : en désactivant 10% des neurones, on remarque un léger underfitting tandis qu'avec un taux égale à 20%, le modèle commence à overfitter après 20 epochs, alors qu'avec 40%, ce qui est considérable, le modèle ne présente plus aucun risque d'overfitting/underfitting mais a tout de même du mal à dépasser les 60% de val accuracy.

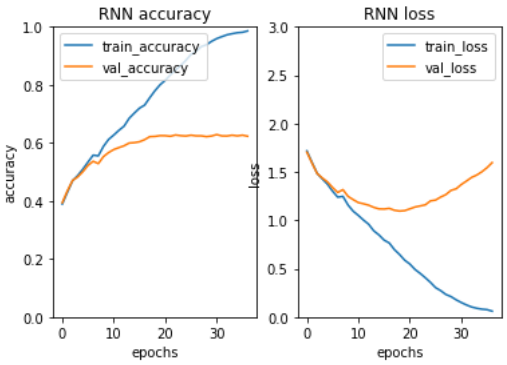
Varier les unités LSTM avec le dropout :

Cette nouvelle étape consiste à varier le nombre d'unités pour les couches LSTM, on ainsi essayer 64 et 512 unités.

64 unités



512 unités



Analyse : on peut s'en apercevoir qu'en augmentant les unités LSTM permet d'avoir une meilleure val_accuracy, cela est tout à fait normal vu qu'avec 512 on a plus de chance

d’attrapper les moindres détails de nos images et ainsi mieu classifier nos images, cependant, le modèle a appris par coeur la dataset du train en atteignant 100% d’accuracy.

Conclusion :

Après l’essaie de 5 différents types de modèles sur notre dataset d’images en RGB à 3072 pixels, du linéaire au réseau de neurone récurrent, on peut déduire que notre cas d’usage n’est pas linéairement séparable, le MLP n’a pas très efficace vu le nombre d’inputs de nos images, l’idéal était une architecture comprenant des couches de convolutions et la meilleure était celle du modèle résiduel qui a pu apporter de très bon résultats grâce à son architecture qui permet d'implémenter des modèles très profonds, composés de dizaines de couches, sans pour autant perdre de l’information et ainsi converger rapidement vers de bonnes prédictions.