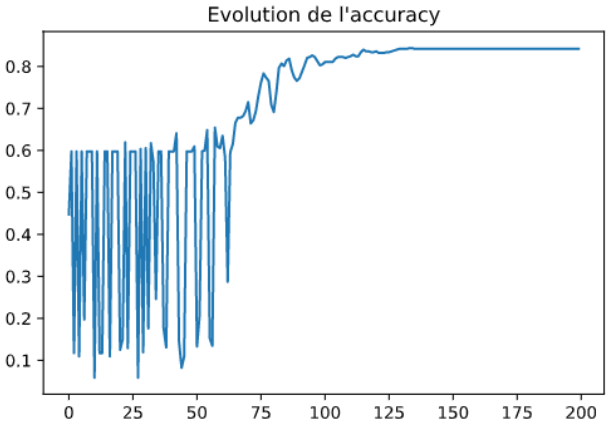
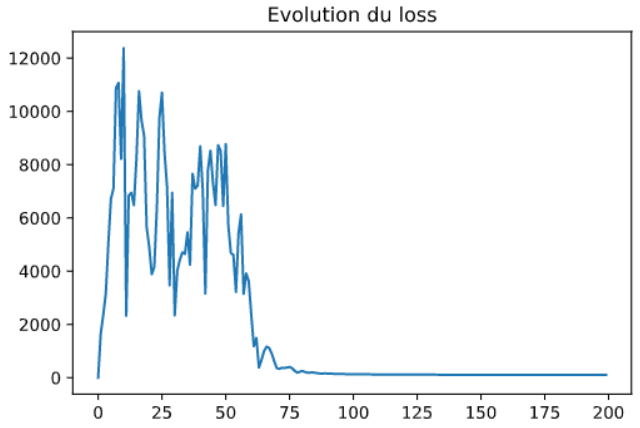
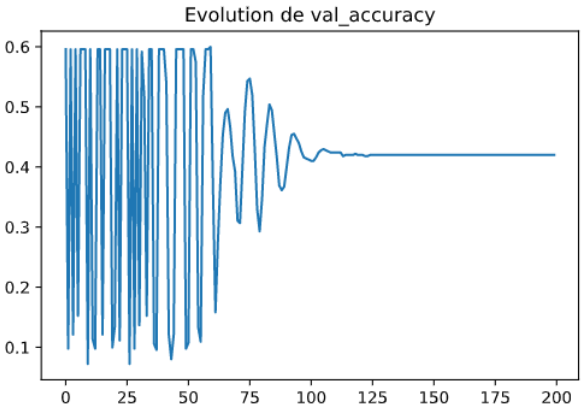
SUIVI KAGGLE

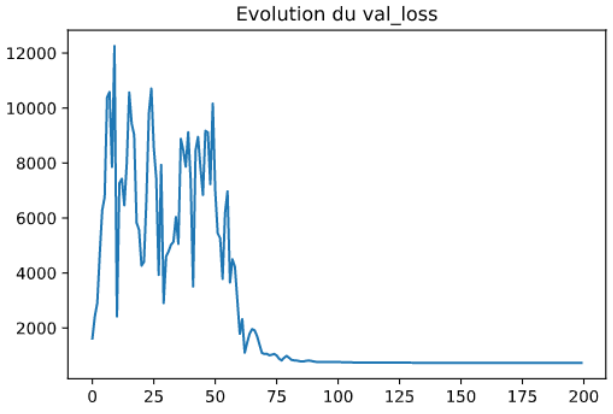
I. Model Linéaire

```
keras.optimizers.SGD(lr=0.1, momentum=0.80),
keras.losses.categorical_crossentropy,
epochs=200,
batch_size=512
```





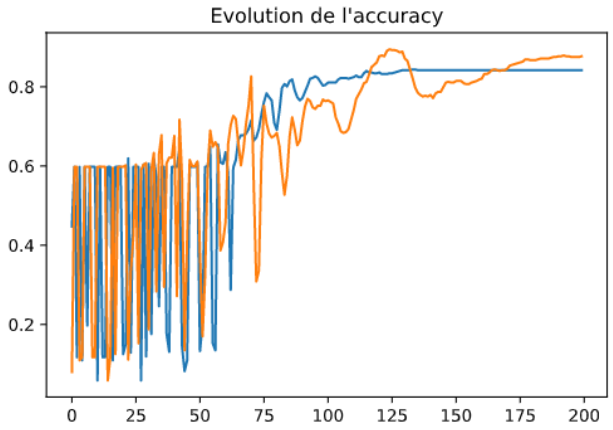


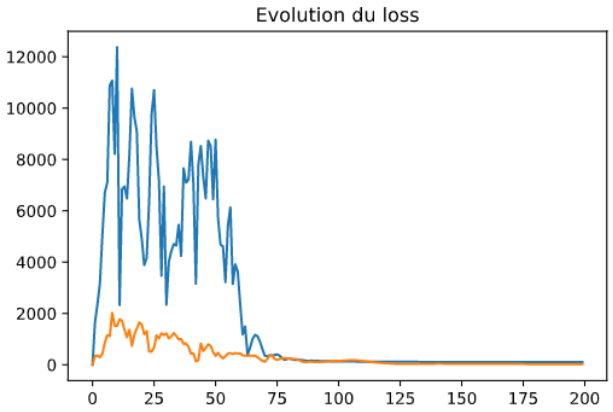


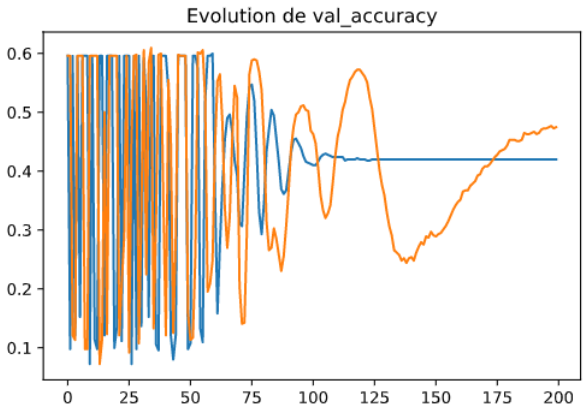

```
loss: 46.7217 - categorical_accuracy: 0.9004 - val_loss: 558.4800 -
val_categorical_accuracy: 0.4180
```

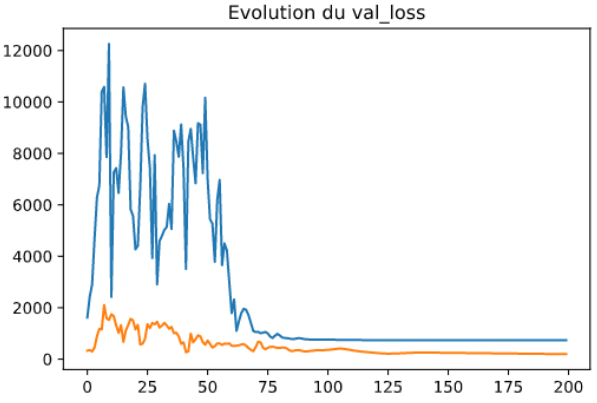
=> Convergence avant les 150 epochs, augmenter les epochs n'aidera pas à obtenir de meilleurs résultats
=> Nouvel essai avec des hyper paramètres différents

```
keras.optimizers.SGD(lr=0.01, momentum=0.95),
keras.losses.categorical_crossentropy,
epochs=200,
batch_size=512
```

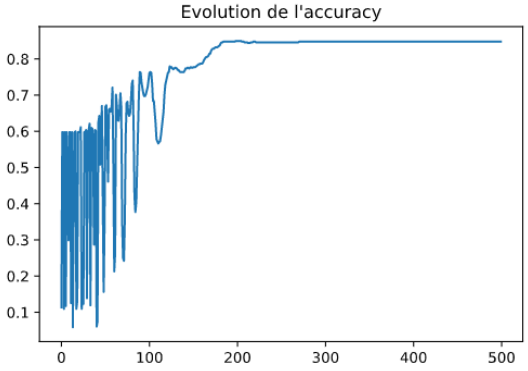


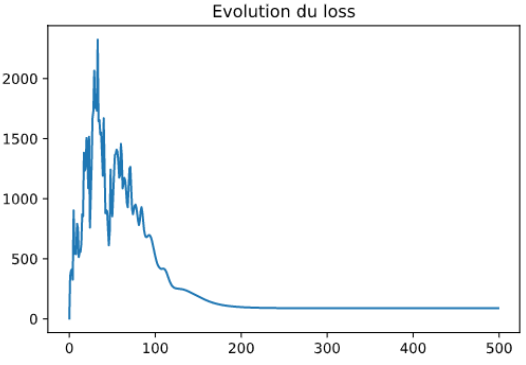


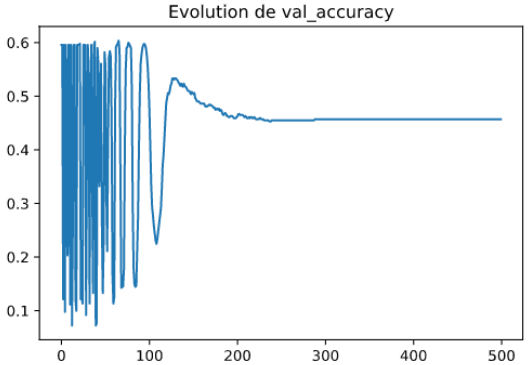


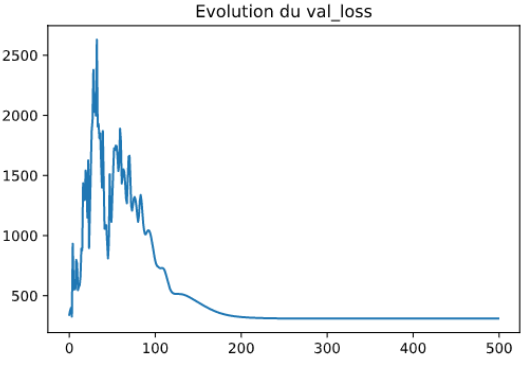


=> En baissant le lr et en augmentant le momentum, on voit en comparant les graphs que le loss converge plus vite mais par contre en 200 epochs l'accuracy est encore trop variables pour savoir si l'accuracy est meilleur.
=> Augmentation du nombre d'epochs









Après relance avec 400 epochs on obtient quelque chose de similaire :

```
loss: 88.3183 - categorical_accuracy: 0.8477 - val_loss: 309.2285 -
val_categorical_accuracy: 0.4570
```

Le modèle overfit énormément.
Pour résoudre ça, on a en premier lieu opté pour augmenter le dataset, mais nous avons rencontré beaucoup de problèmes à cette étape.

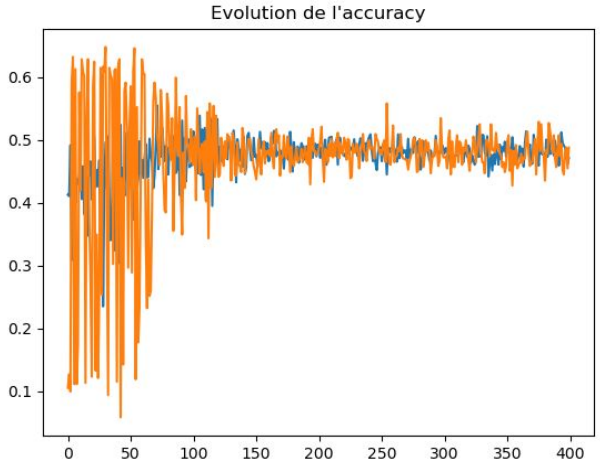
On a donc réalisé des rotations, des transformations, des découpages et l'ajout de flou aléatoire sur le dataset. Lors du chargement des images par la suite on remarque que les modèles sont plus longs que lors de l'utilisation de fichiers Tfrecord.
Ainsi on a donc voulu enregistrer nos images augmentés avec leur label dans des fichiers Tfrecord pour gagner du temps dans le train.
Cependant lors de l'import de ces derniers, des problèmes d'encodage apparaissent et les modèles ont des accuracy qui se bloquent sur certaines valeurs.

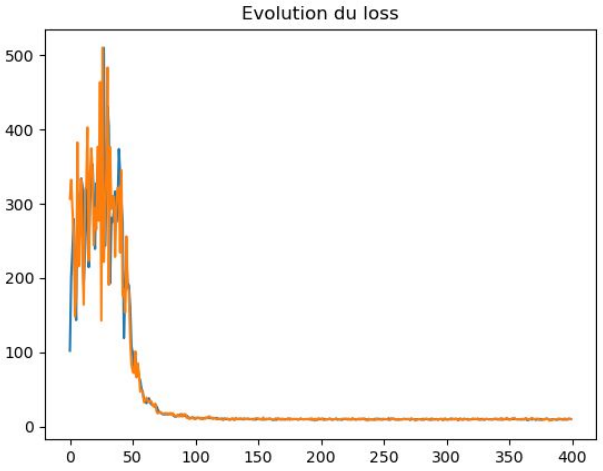
Finalement ce qui a été adapté a été d'utiliser le générateur d'image keras comme suivant :

```
train_datagen = keras.preprocessing.image.ImageDataGenerator(
      rescale= 1./255, validation_split=VALIDATION_SPLIT,
      shear_range= 0.2, zoom_range= 0.2, horizontal_flip= True,
      rotation_range=20, vertical_flip= True,)
```

Cependant malgré l'utilisation d'un ssd le chargement des données est plus long que l'entraînement du modèle. De plus nous ne disposions pas de GPU qui supportait le volume de données donc nous n'avons pu réaliser de training qu'avec les ressources de nos CPU.
Pour réduire les images, nous les avons redimensionnées en 128x128.

On a donc réessayer avec le modèle linéaire en effectuant ces transformations aléatoires, on obtient ces résultats :



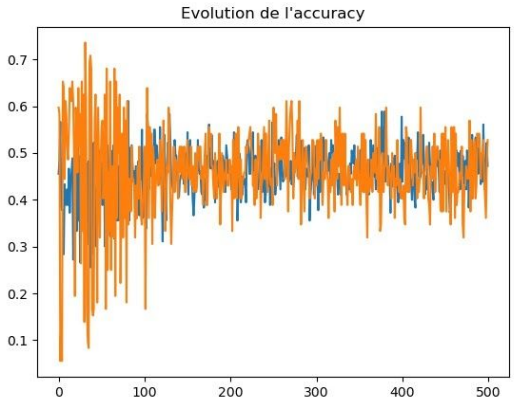


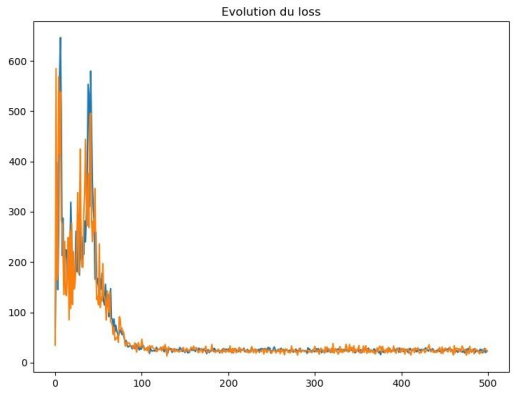
=>

on voit que le modèle généralise mieux, on a

une prédiction homogène sur le dataset de validation et d'entrainement. L'augmentation du dataset a été bénéfique et notre model semble plus cohérent.

Nous avons également fait des tests avec un mini batch size (= 12) et un grand nombre d'epochs pour voir si le modèle converge bien :



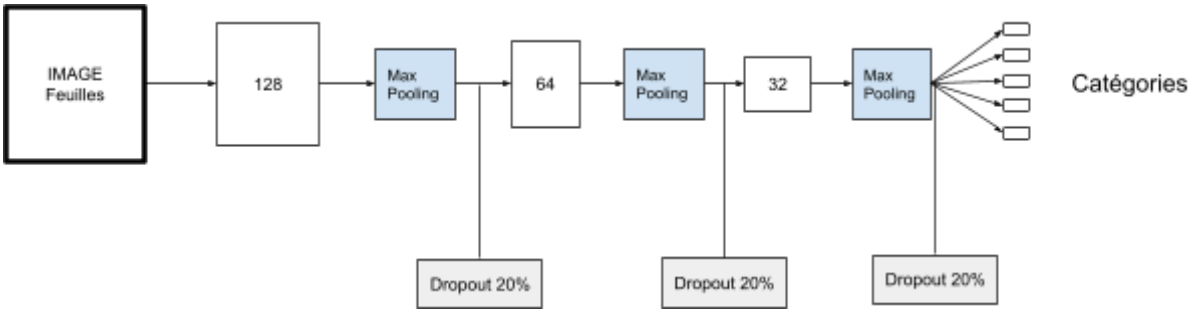


Le modèle converge encore avec une bonne généralisation même si on peut le voir, il y a beaucoup de bruit sur les courbes, ce qui rend leur lecture plus complexe.

II. Modèle CNN

Ainsi après avoir entraîné un modèle linéaire, nous sommes passés sur un modèle plus complexe qui est reconnu comme efficace pour la détection d'image. Nous avons ainsi testé plusieurs modèles CNN plus ou moins complexes avec différents hyper-paramètres.

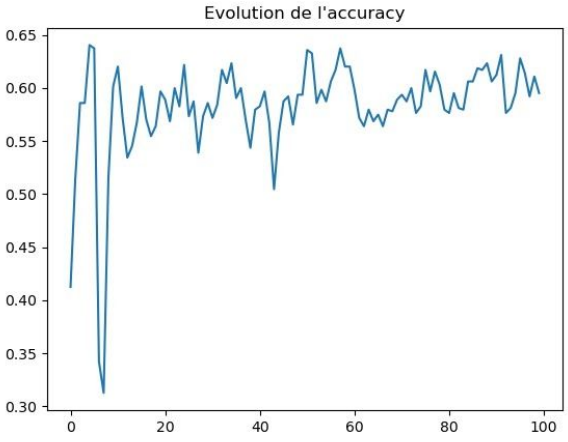
Tout d'abord un modèle avec 3 convnet simple et une sortie Dense en 5 catégorie :

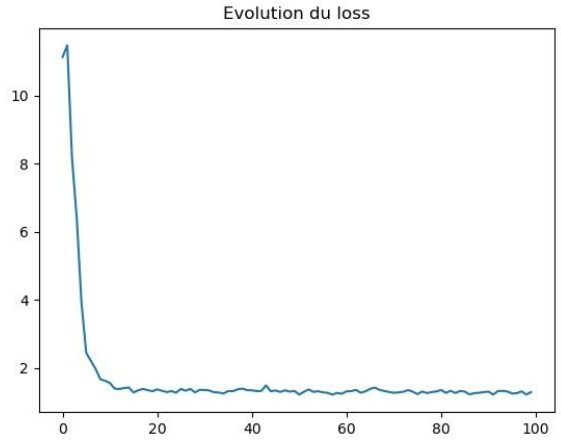


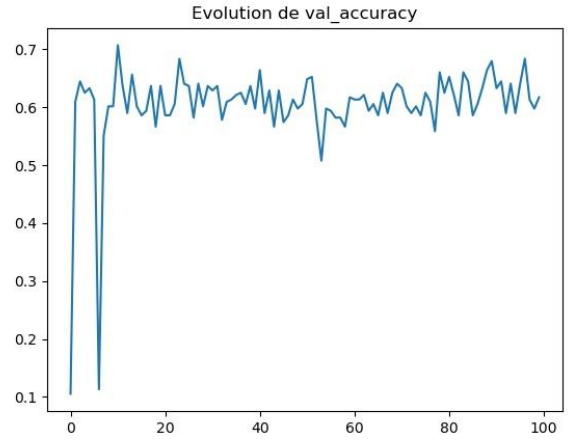
On reprend les mêmes hyper-paramètres que le dernier modèle linéaire pour pouvoir comparer :

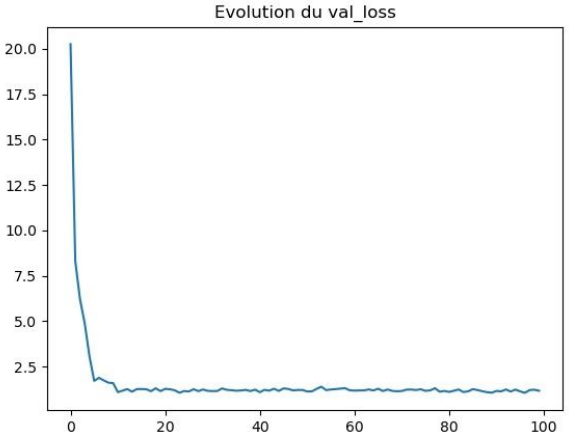
```
keras.optimizers.SGD(lr=0.01, momentum=0.95),
keras.losses.categorical_crossentropy,
epochs=100,
batch_size=256,
```

```
steps_per_epoch=10,
val_steps=3
```





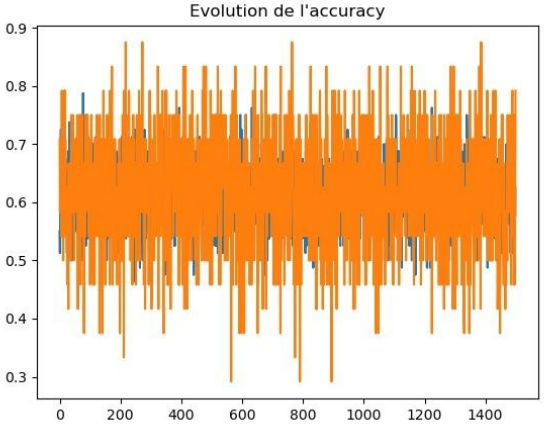


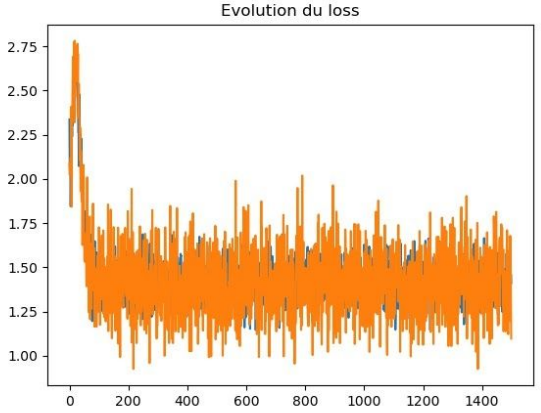


=> On peut voir que les résultats sont dans l'ensemble meilleurs et ne présentent pas d'over fitting.

Toutefois ils présentent encore une accuracy à revoir environ 0.6, ce qui n'est pas satisfaisant dans le cadre d'une compétition kaggle.

De même que pour le modèle linéaire nous avons essayé avec un mini batch size et un grand nombre d'epochs d'entraîner notre modèle.



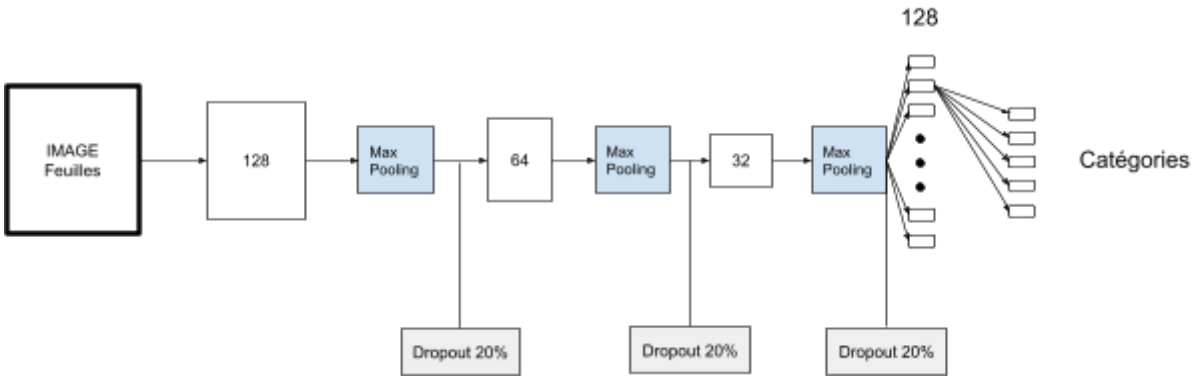


=> Le modèle généralise toujours aussi bien mais à cause du petit batch size, l'accuracy fluctue beaucoup

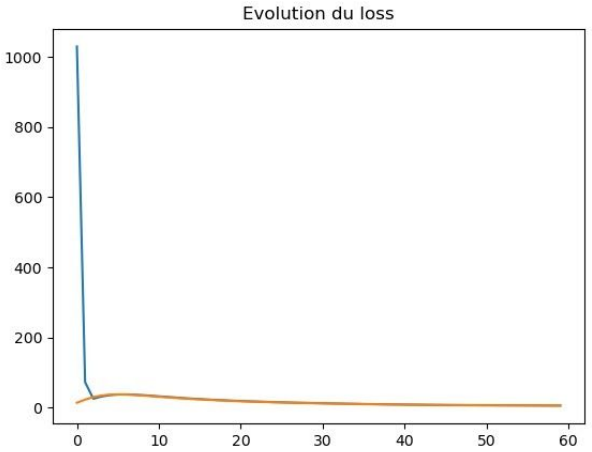
On est revenu alors sur un plus gros batch size (256) pour la suite afin de pouvoir analyser plus clairement les résultats.
En gardant globalement le même modèle mais en ajoutant une couche dense à la fin des convnet. De plus, un nouvel optimizer est testé : Adam, qui est meilleur sur les résultats possédant du bruit. De plus nous avons plus d'étape de génération d'images.
On utilise alors ces hyper-paramètres :

```
keras.optimizers.Adam(lr=0.01), keras.losses.categorical_crossentropy,
epochs=60,
batch_size=256,
steps_per_epoch=20,
val_steps=6
```

et le model comme suit

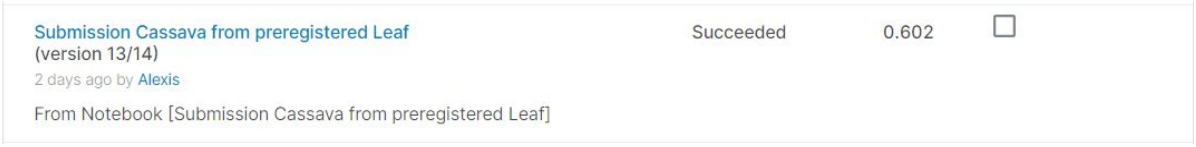


Ce qui a donné :



=> On remarque peu de changements
=> On pourrait dire que l'accuracy semble légèrement plus groupé.

A partir de cette étape nous avons essayé de réaliser une soumission à la compétition. Pour ce faire, nous sauvegardons le dernier modèle présenté après l'entraînement.
Puis nous le chargeons sur un notebook qui va appliquer le modèle sur tout le dossier d'images "/test_images/". Enfin le notebook sauvegarde les résultats dans un fichier csv qui sera analysé pour produire le score.
Les captures suivantes montrent le résultat de notre soumission ainsi que le bloc du notebook soumis.

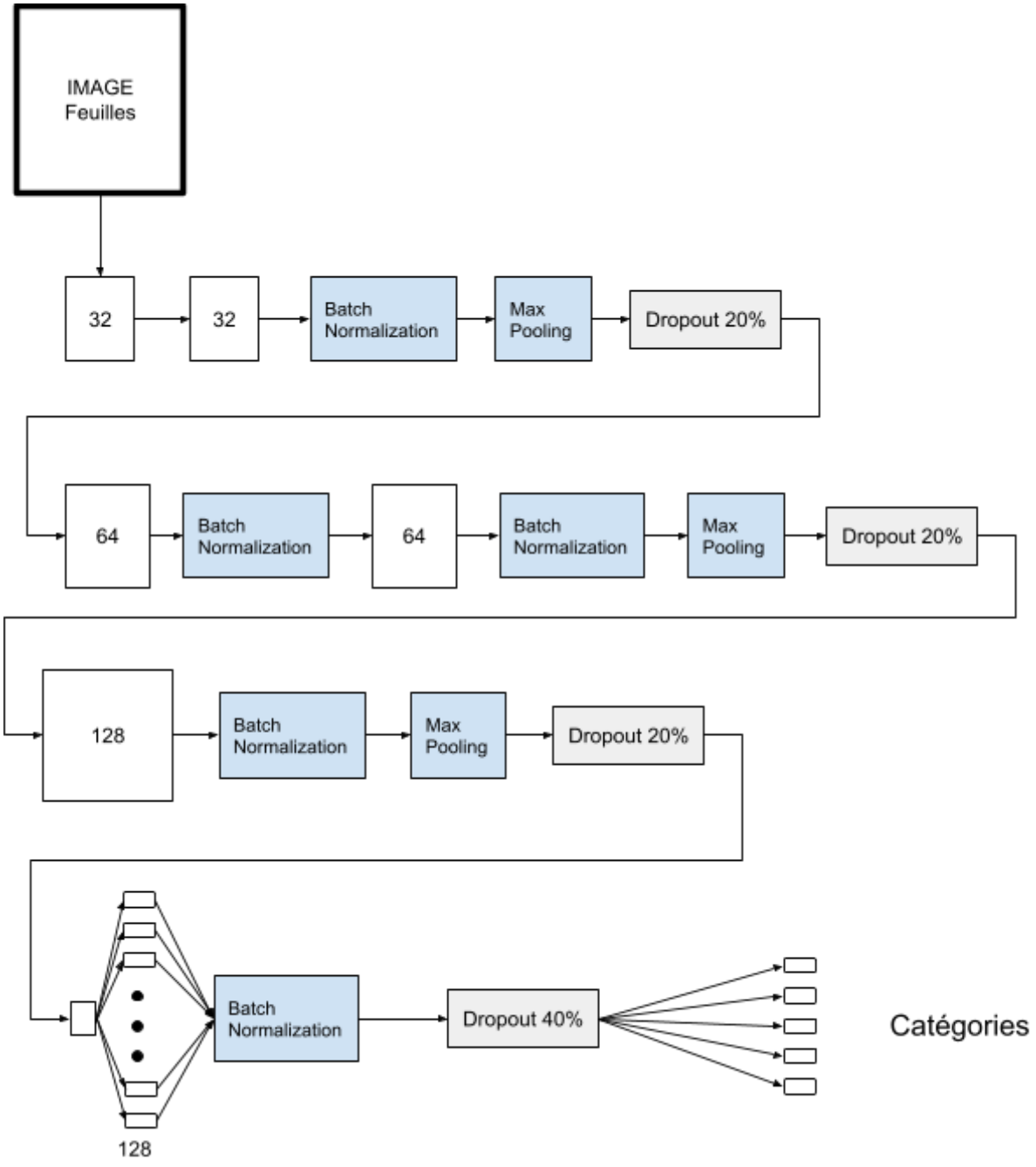



```
preds = []
origin = "../input/cassava-leaf-disease-classification"
# sample_sub = pd.read_csv(origin + '/train.csv')
sample_sub = pd.read_csv(origin + '/sample_submission.csv')
model = keras.models.load_model("../input/model-convnet/convnet_aerial")

for image in sample_sub.image_id:
    img = tf.keras.preprocessing.image.load_img(origin + '/test_images/' + image)
    img = tf.keras.preprocessing.image.img_to_array(img)
    img = tf.keras.preprocessing.image.smart_resize(img, (256, 256))
    img = np.expand_dims(img, 0)
    prediction = model.predict(img)
    preds.append(np.argmax(prediction))

my_submission = pd.DataFrame({'image_id': sample_sub.image_id, 'label': preds})
my_submission.to_csv('submission.csv', index=False)
my_submission
```

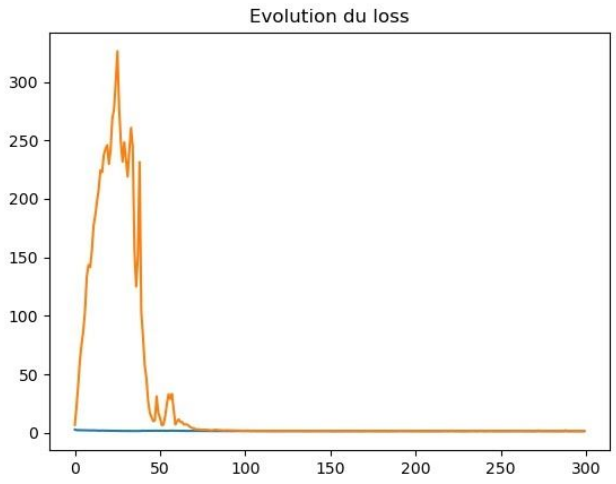
Pour finir nous avons employé un modèle plus complexe qui nous a pris bien plus de temps à entraîner pour espérer obtenir le meilleur résultat possible avec nos ressources et nos connaissances.

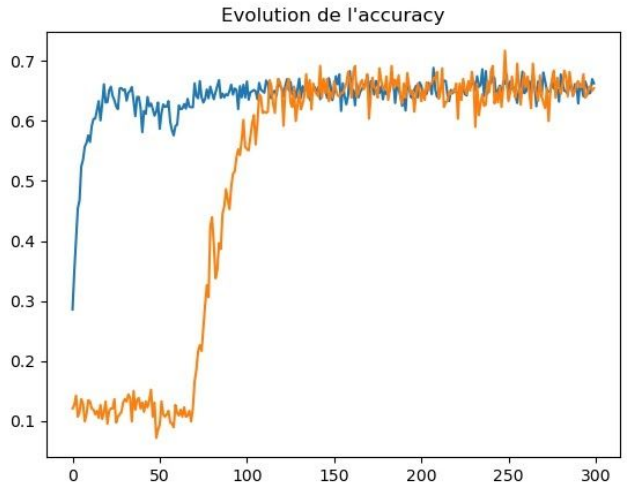


Cette fois les filtres des layers conv2D vont de 32 à 128, en doublant les couches à 32 et 64 pour réduire l'overfitting. Les dropout de 20% permettent également de renouveler l'apprentissage et augmenter l'apprentissage.
A chaque couche conv2D, on réalise un BatchNormalization pour la sortie de la couche précédente qui permet d'assurer une stabilité dans l'apprentissage en ajoutant des poids sur les extrêmes.

Ensuite on passe par des couches Dense pour classer les résultats dans nos 5 catégories.

Les résultats sont ainsi les suivants :






```
Epoch 300/300

1/5 [=====>........................] - ETA: 0s - loss: 1.4937 - categorical_accuracy: 0.6328
2/5 [===========>..................] - ETA: 1:17 - loss: 1.4748 - categorical_accuracy: 0.6582
3/5 [=================>............] - ETA: 1:09 - loss: 1.4587 - categorical_accuracy: 0.6706
4/5 [=======================>......] - ETA: 38s - loss: 1.4808 - categorical_accuracy: 0.6670
5/5 [==============================] - ETA: 0s - loss: 1.5036 - categorical_accuracy: 0.6625
5/5 [==============================] - 231s 46s/step - loss: 1.5036 - categorical_accuracy: 0.6625 - val_loss: 1.5861 - val_c
ategorical_accuracy: 0.6543
```

Ceci sont les meilleurs résultats que l'on ait pu obtenir depuis le début de l'étude.
Pour terminer notre participation à la compétition kaggle, nous soumettons ce modèle ci.

On a amélioré légèrement le score.

III. Possibilité d'amélioration des résultats

Il peut être intéressant de faire tourner nos modèles bien plus longtemps. Mais ici afin d'au mieux couvrir toute la partie de recherches avec nos ressources, nous nous sommes plutôt limités tout en maîtrisant ce que l'on faisait.

Pour aller plus loin, il est possible d'importer un modèle pré-entraîné disponible sur internet type CNN ou EfficientNet afin d'aller plus loin. A ce modèle importé on y ajoute des traitements et d'autres couches afin d'affiner les résultats et de faire correspondre ces modèles pré-entraînés à nos catégories.