Algorithme de détection de la race du chien sur une photo

OPENCLASSROOMS

colab

kaggle

Android Studio

Outline

- Introduction
  - Objectif de l'étude
  - Jeux de données
- Feature engineering
  - Traitement des images
  - Augmentation
- Modélisation
  - Modelés simples – numéro réduit des races
  - Transfer-learning et batch processing (modélisation complète)
  - Optimisation du modèle (couches denses, vitesse apprentissage)
  - Tensorflow-lite model maker : peu de code et petite taille
- Conclusions
  - Choix du modèle (serveur web/ smartphone)

Modalités de la soutenance

5 min - Présentation de la problématique, de son interprétation et des pistes de recherche envisagées.

5 min - Présentation du cleaning effectué, du feature engineering et de l'exploration.

10 min - Présentation des différentes pistes de modélisation effectuées.

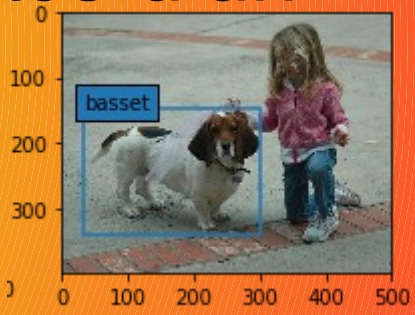
5 min - Présentation du modèle final sélectionné et résultats.

5 à 10 minutes de questions-réponses.

Objectif du modèle

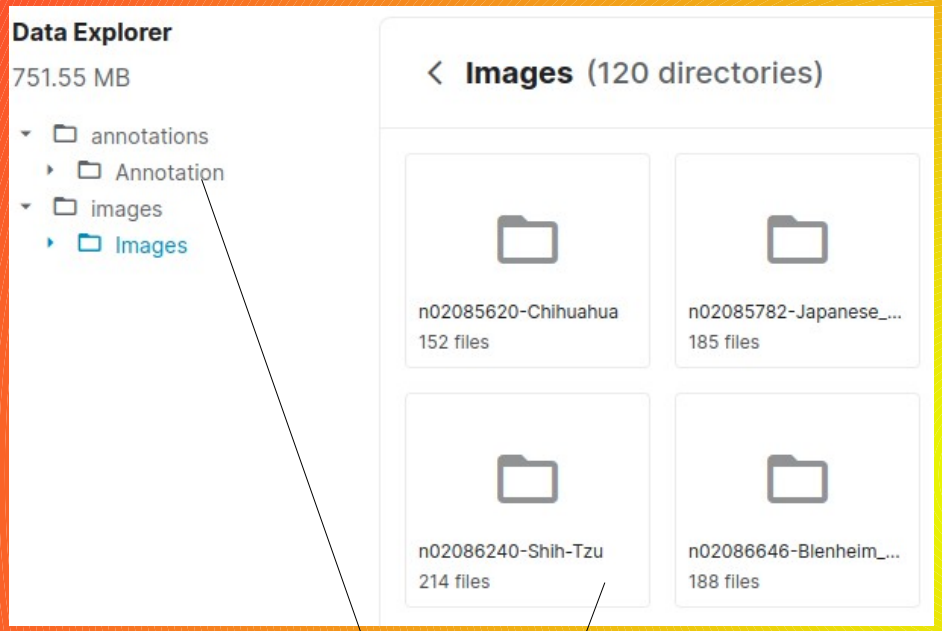
Dans cette étude avec un apprentissage supervisé, on utilise une réseaux des neurones convolutionelle pour

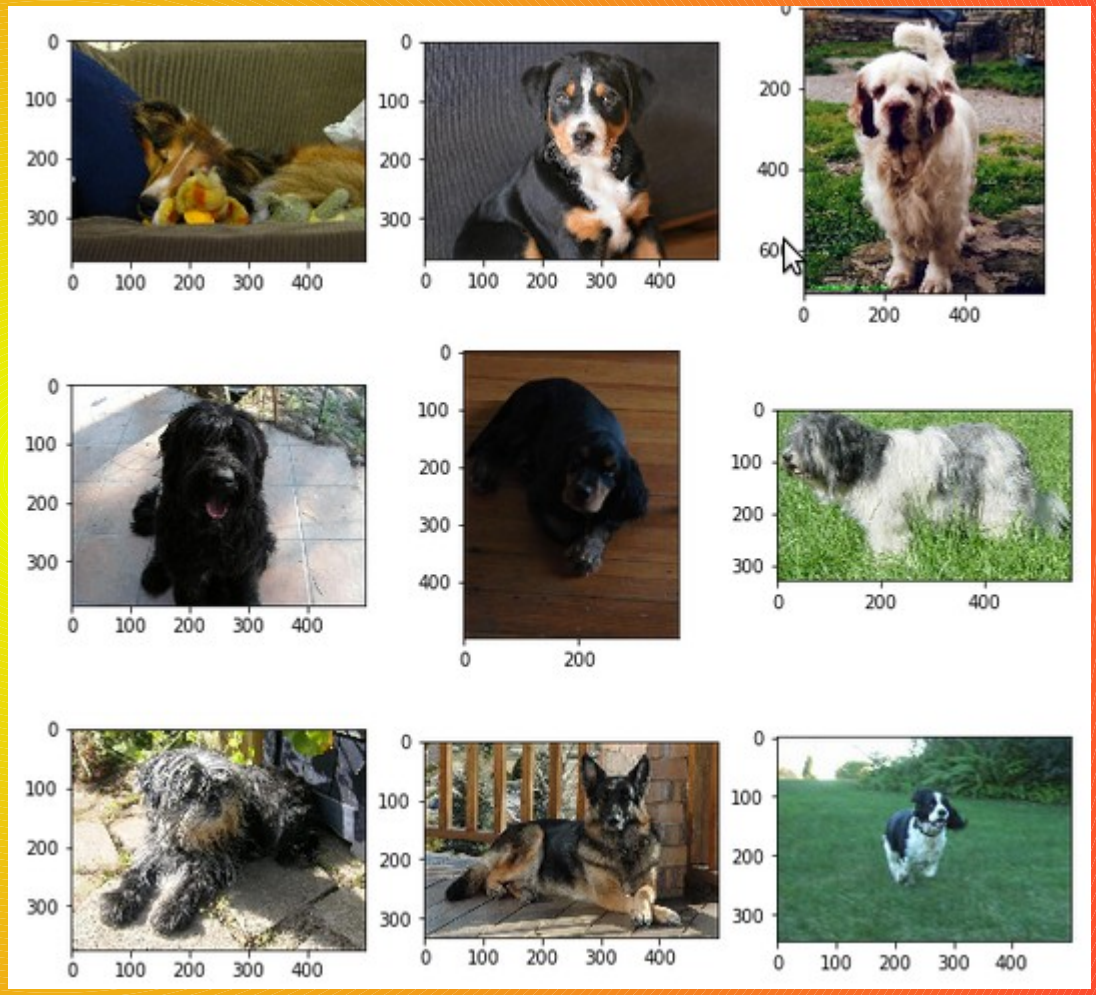
- Déterminer précisément (P > 80 %) la race d'un chien à partir d'une photo



- Intégrer une application sur un smartphone (taille < 20 MB) ou site web

Données du départ





On a des annotations mais aussi le nom de la race pour chaque dossier d'image

Pre-traitement des données

- Equalisation
- Augmentation des images
- Réduction de la taille (format pour réseaux neurones)
- Segmentation des données : entraînement, test

Pre-traitement des données

Observations
X

Réduction à la taille adaptée
pour une réseaux des neurones

```
#resize image to 227 x 227 because the input
resized_img = img_array.resize((227, 227))

img_lst.append(np.array(resized_img))

labels.append(filtered_breeds[index])
```

Normalisation

```
#Normalization for the images
images = np.array(images)
images = images.astype(np.float32)
#labels = labels.astype(np.int32)
X_norm = images/255
```

Cible
y

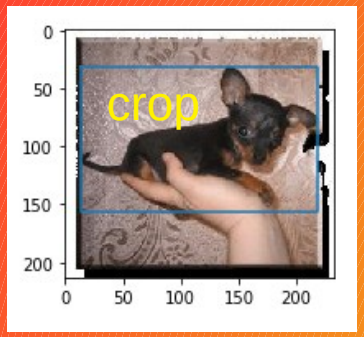
One-hot-encoding pour les categories

```
# replace numbers with names
le = LabelEncoder()
nlabels = le.fit_transform(labels) # encode labels as number values. Thi
Y=to_categorical(nlabels,num_classes = num_breeds) # category encoding
```

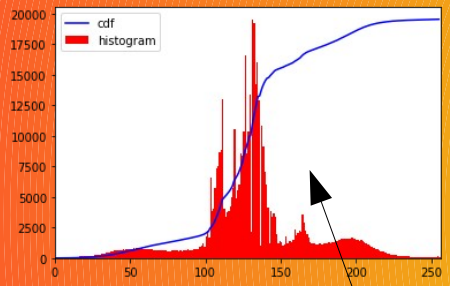
Données Entraînement, validation , test

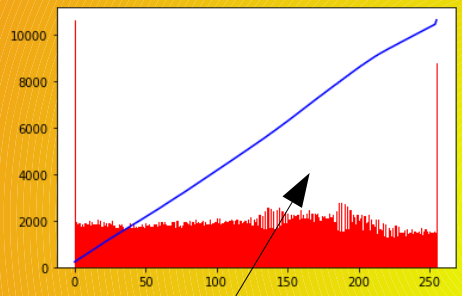
```
x_train, x_test, y_train, y_test = train_test_split(X_norm, Y, test_size = 0.2, random_state = 42)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=1)
```

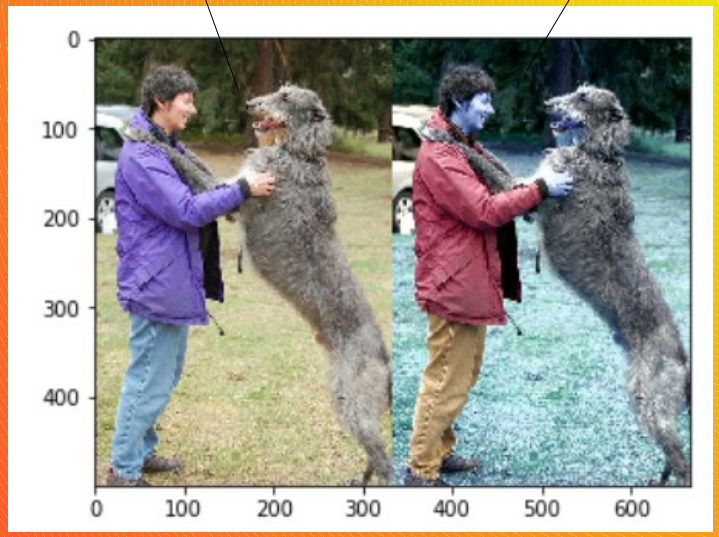
Pre-traitement des images



Equalisation



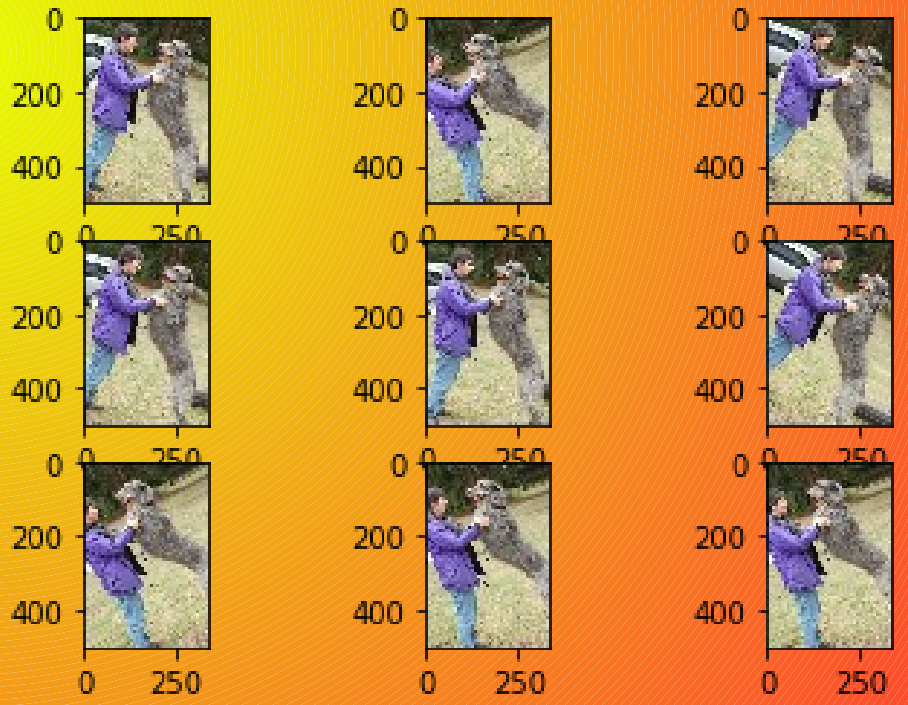




Augmentation

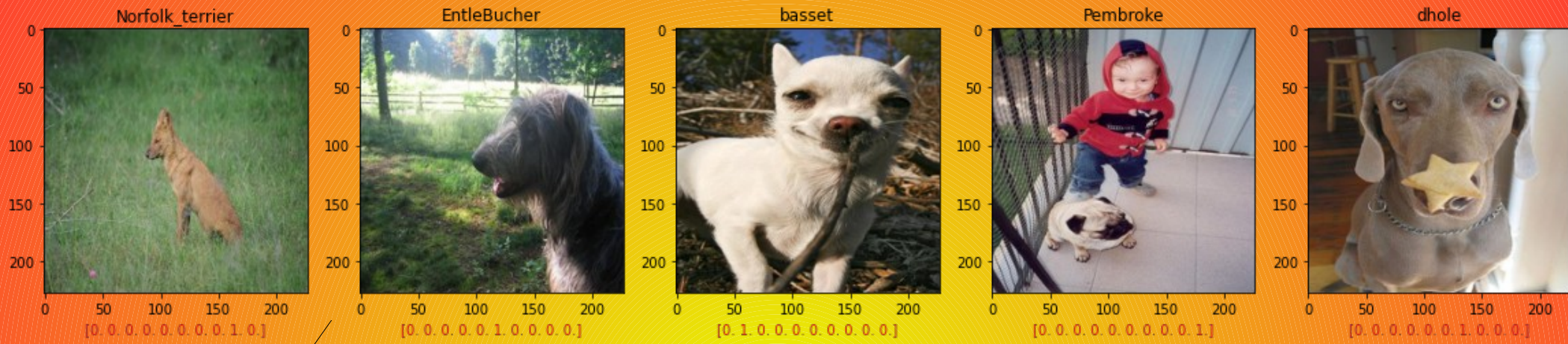
miroir

rotation



Que pour les données d'entrainement X_{train}

Batch processing



Images + etiquettes

Scale
Normalize
Encode

Train

Val | test

Pre-process :
moyennage,
crop,
augmentation

Pre-process :
moyennage,
crop,

Modélisation
(Kaggle GPU)

Application aux nouvelles images

Modélisation des données

- Simple multi-layer perceptron
- Alexnet
- Xception transfer learning

Modéle 0 : 2-couches CNN

Input

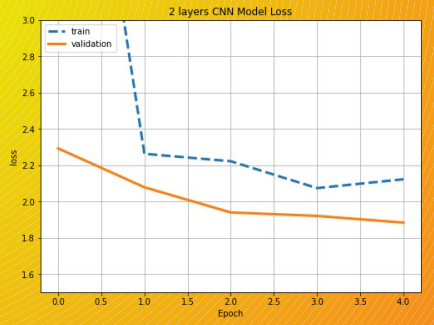
227 X
227 X
3

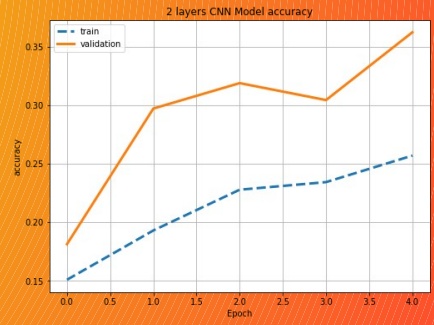
Conv1 Pool 1 flatten

Fully Connected

Output (n_classes)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              multiple                  432
_________________________________________________________________
max_pooling2d (MaxPooling2D) multiple                  0
_________________________________________________________________
flatten (Flatten)            multiple                  0
_________________________________________________________________
dropout (Dropout)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  6654080
_________________________________________________________________
dense_1 (Dense)              multiple                  1290
=================================================================
Total params: 6,655,802
Trainable params: 6,655,802
Non-trainable params: 0
```





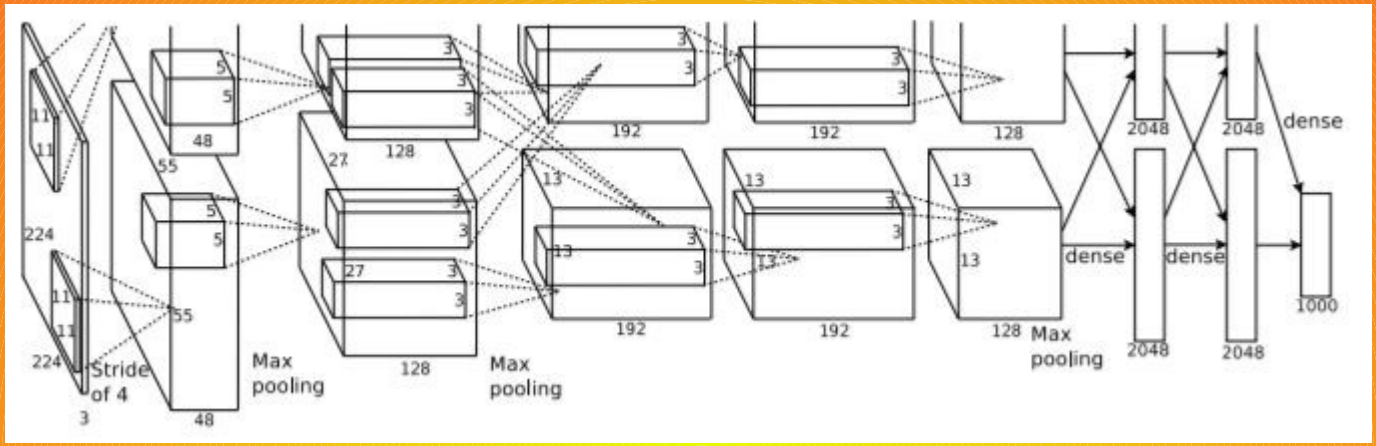
Features extraction

Classification

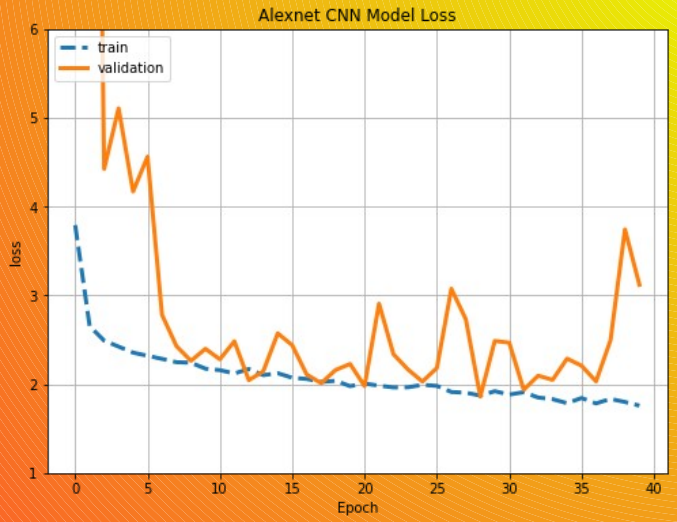
Underfitting

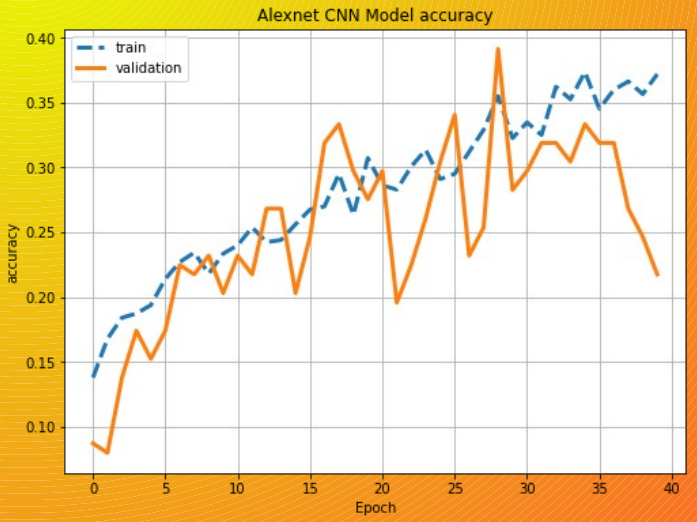
Test accuracy : 34 %

Modéle 1 : Alexnet



Avec plus des couches on essaye de réduire l'underfitting





Précision faible : 23 %

Résultats Alexnet

Lakeland_terrier

whippet 80%, Border_terrier 10%, Lakeland_terrier 0%

affenpinscher

Irish_setter 30%, Weimaraner 30%, affenpinscher 20%

Weimaraner

Weimaraner 100%, Lakeland_terrier 0%, wire-haired_fox_terrier 0%

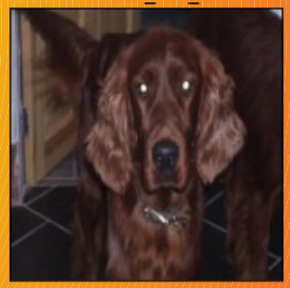
Irish_setter

black-and-tan_coonhound 100%, Lakeland_terrier 0%, wire-haired_fox_terrier 0%

black-and-tan_coonhound

black-and-tan_coonhound 100%, Lakeland_terrier 0%, wire-haired_fox_terrier 0%

wire-haired_fox_terrier



Irish_setter 50%, wire-haired_fox_terrier 30%, black-and-tan_coonhound 20%

black-and-tan_coonhound

black-and-tan_coonhound 80%, silky_terrier 10%, whippet 10%

whippet

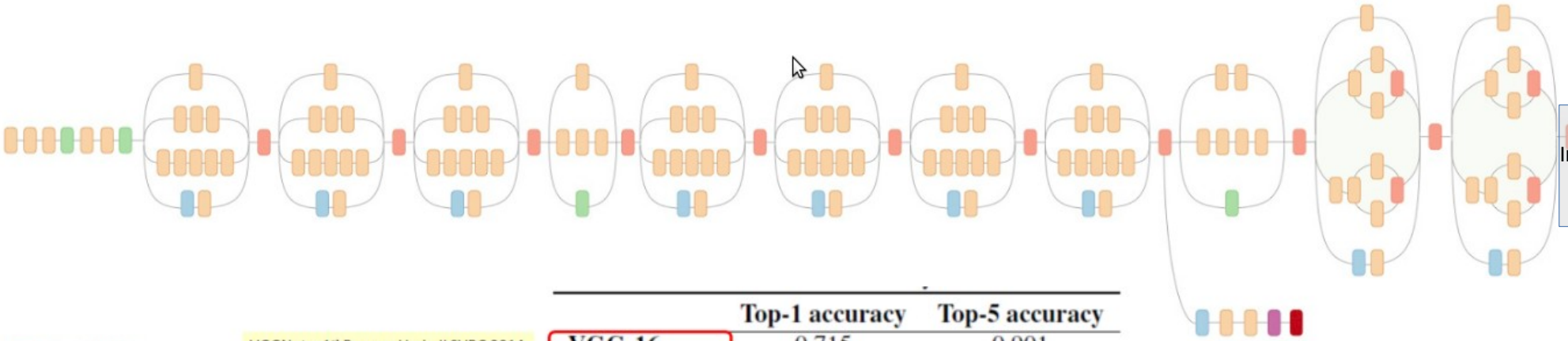
Weimaraner 40%, black-and-tan_coonhound 20%, Irish_setter 10%

Irish_setter

black-and-tan_coonhound 60%, silky_terrier 10%, whippet 10%

Parfois correct

Transfer learning : Xception



Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax

VGGNet – 1st Runner Up in ILSVRC 2014
ResNet – Winner in ILSVRC 2015
Inception-v3 – 1st Runner Up in ILSVRC 2015

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

Input

299 X
299 X
3

Xception :
weights pretrained
with Imagenet

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception
/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 1s 0us/step
```

flatten

Fully
Connected

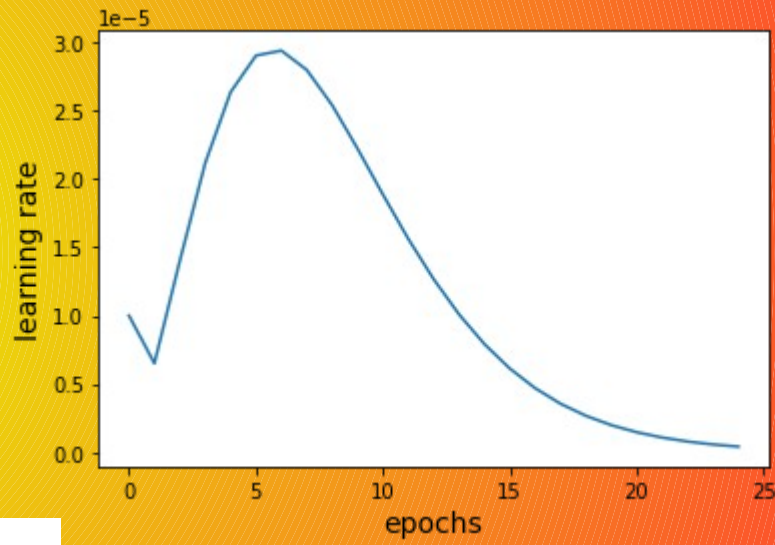
Classification

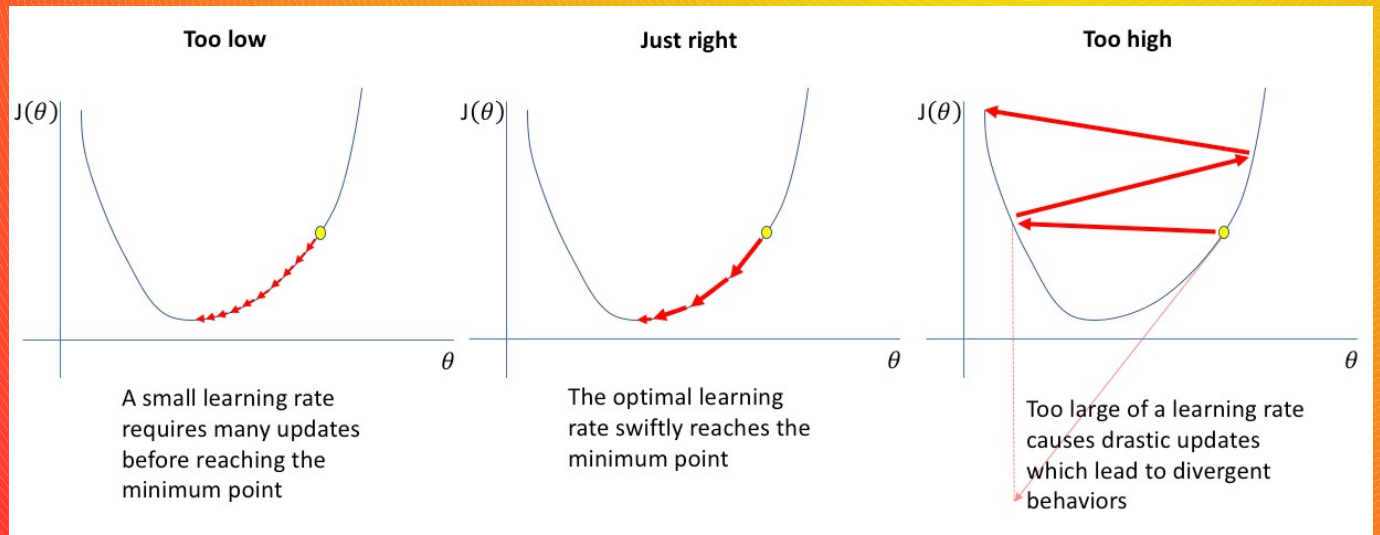
Output (120 classes)

Export trained model :
my_model.h5

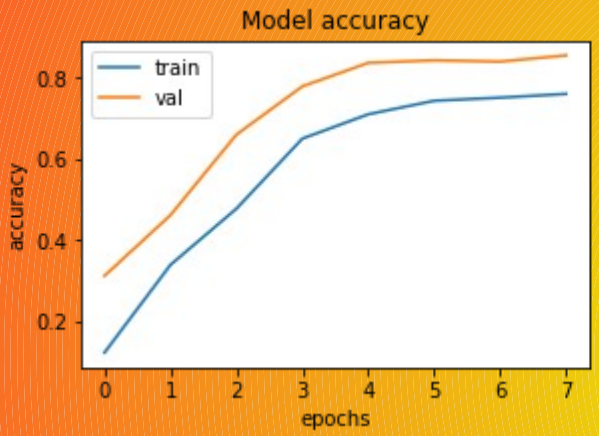
Optimisation de l'entraînement

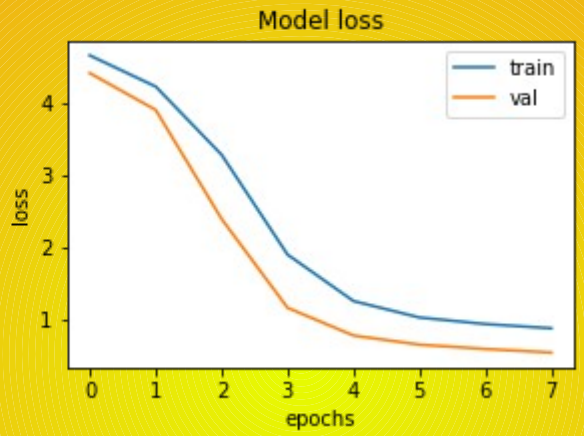
- n couches cachées
- dropout
- n epochs / steps per epoch
- vitesse d'apprentissage α





Transfer learning : Résultats





Underfitting
(un peu)

```
test_loss, test_accuracy = my_model.evaluate_generator(generator=test_ds,steps=int(100))

print("Test results \n Loss:",test_loss,'\n Accuracy',test_accuracy)
```

```
Test results
 Loss: 0.5387319326400757
 Accuracy 0.8543750047683716
```

85 %
accuracy

```
93.55%    malinois
1.69%     kelpie
1.32%     German_shepherd
0.72%     Doberman
0.65%     dingo
```

```
download_and_predict("https://
cdn.pixabay.com/photo/
2018/08/12/02/52/belgian-mallinois-
3599991_1280.jpg",
                     "test_1.jpg")
```

Output

543.9 MB

dog_breed_xcept_weights.h5

my_model.h5

my_model.h5 (111.95 MB)

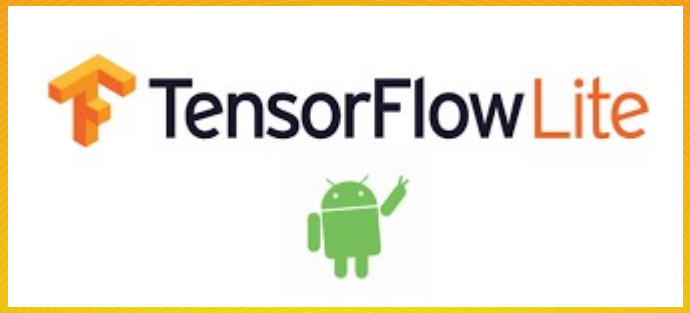
Modèle sauvé dans un fichier assez petit pour un serveur web mais trop lourd pour un smartphone

Développent pour Android

- Presque pas des réglage
- Précis (85%)
- Léger (smartphone app, pas des serveurs)

Modélisation pour smartphone

Modèle Tensorflow Imageclassifier

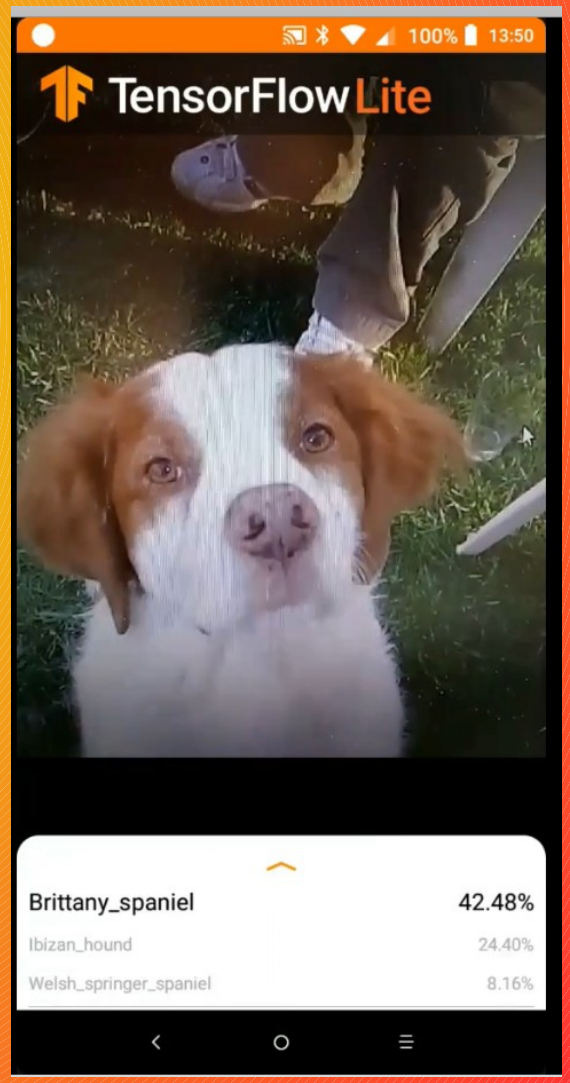


Entraînement sur base de données choisie (Colabs GPU)

85 % accuracy

Tensorflow-Lite converter

Model file (.tflite)



Conclusions

- Les modèles multi-couche perceptron donnent underfitting et faible précision
- Le meilleur résultat était obtenu avec une réseaux plus complexe et une optimisation des paramétrés(vitesse apprentissage…)
- Avec Tensorflow-lite model maker un modèle très léger mais performant était crée pour Android