Projet Stanford Dogs

Efkan TUREDI

Intro

Nous devons développer un algorithme de reconnaissance des races de chiens pour l'association pour laquelle nous travaillons qui n'a pas les ressources nécessaires pour mener à bien ces travaux de manière indépendante et à plus grande échelle.

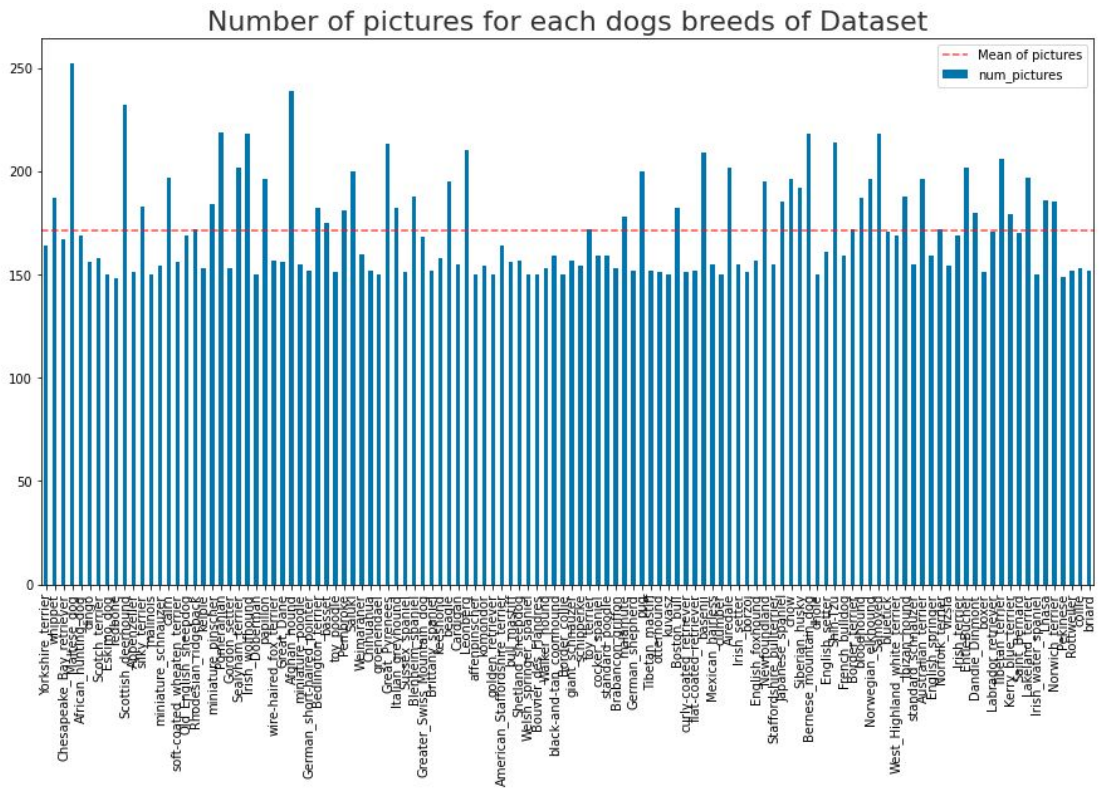
Nous utilisons la base de données Stanford dogs pour entraîner les modèles: elle contient 120 races de chiens.

Récupération des données

Récupération et Overview

- Nous stockons les deux folders qui vont nous être utiles pour cet exercice dans un bucket Amazon S3. Les folders Images et Annotations
- Il y a un total de 120 races de chiens (qui seront donc nos 120 classes). Chaques classes possède entre 150 et 200 photos. Pas de problème de classe imbalance à priori.
- A noter que nous ferons tourner nos modèles sur 20 classes uniquement pour de ressources temporelles.

Overview

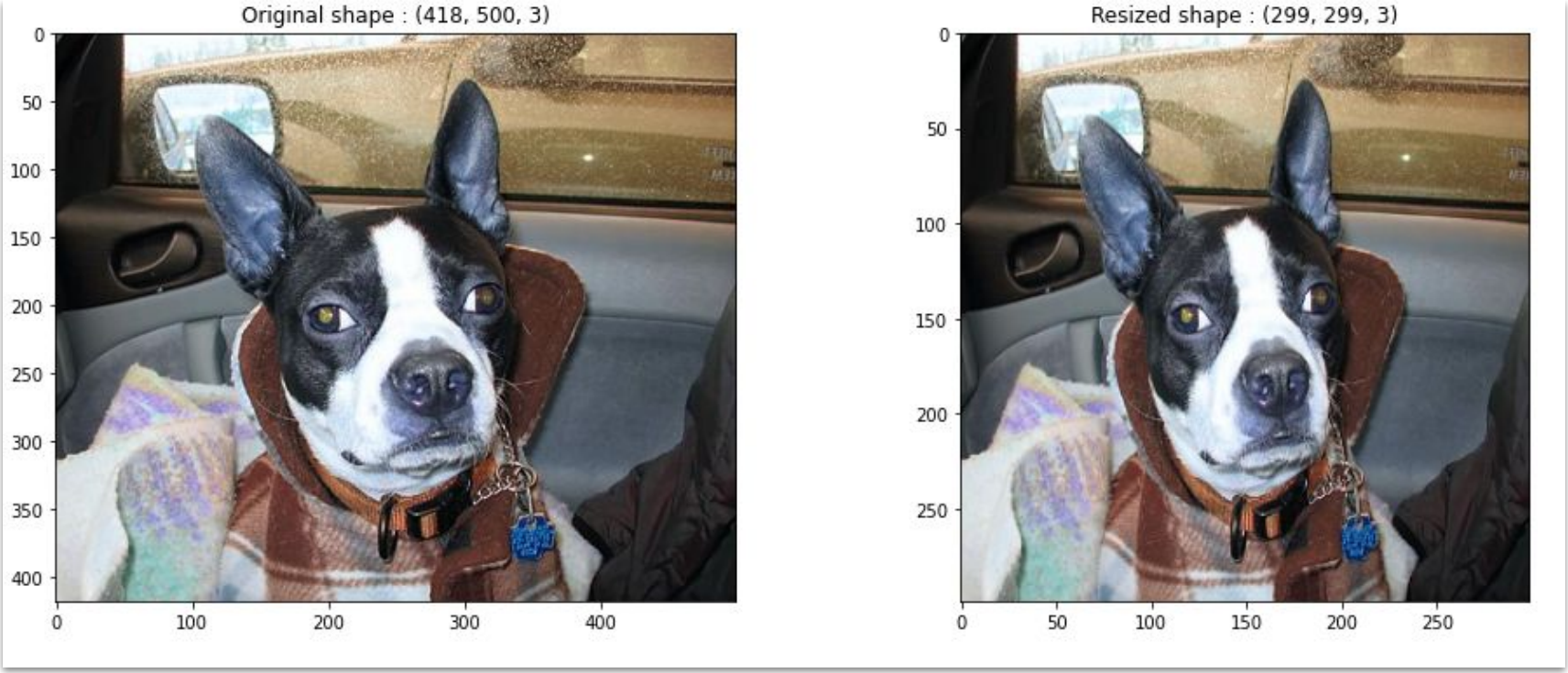

Number of pictures for each dogs breeds of Dataset
Mean of pictures
num_pictures
250
200
150
100
50
0

Préparation des données

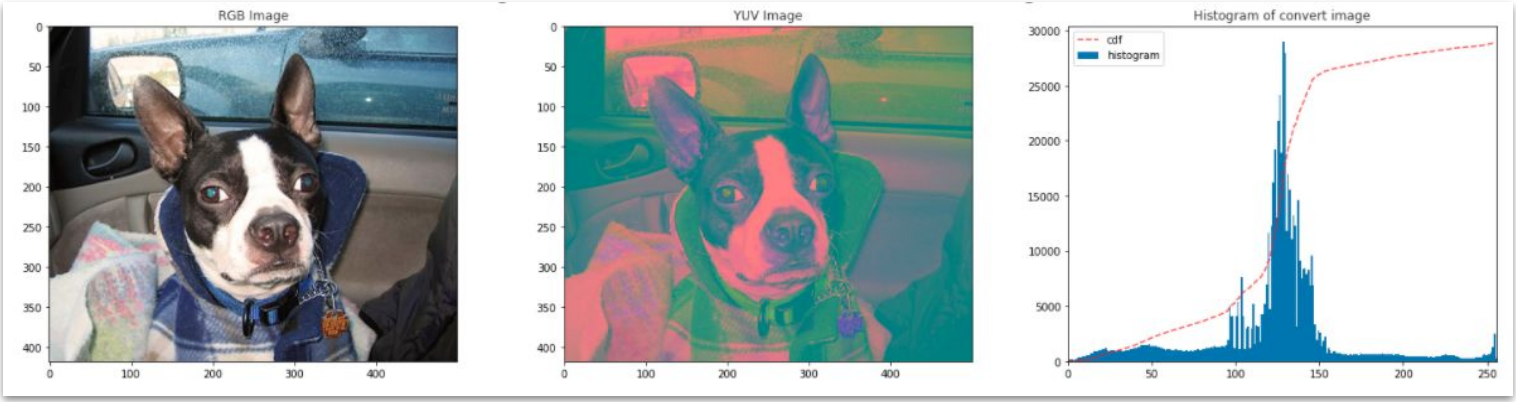
Nous allons appliquer les transformations suivantes:

- **Resizing**: Nous remettons les images à un format uniforme, afin de pouvoir les utiliser. Nous choisissons le format 299x299x3 qui ne déforme pas trop la plupart des images.
- **Egalisation**: Pour équilibrer la la luminosité des images, nous utilisons cette technique.
- **Debruitage/Filtre non lineaire:** Nous utilisons le non local means filter pour débruiter l'image, et la rendre plus "smooth" pour nos CNN
- **Data Augmentation:** Nous utilisons cette technique pour générer des "nouvelles" images pour améliorer la performance de nos modèles

Resizing

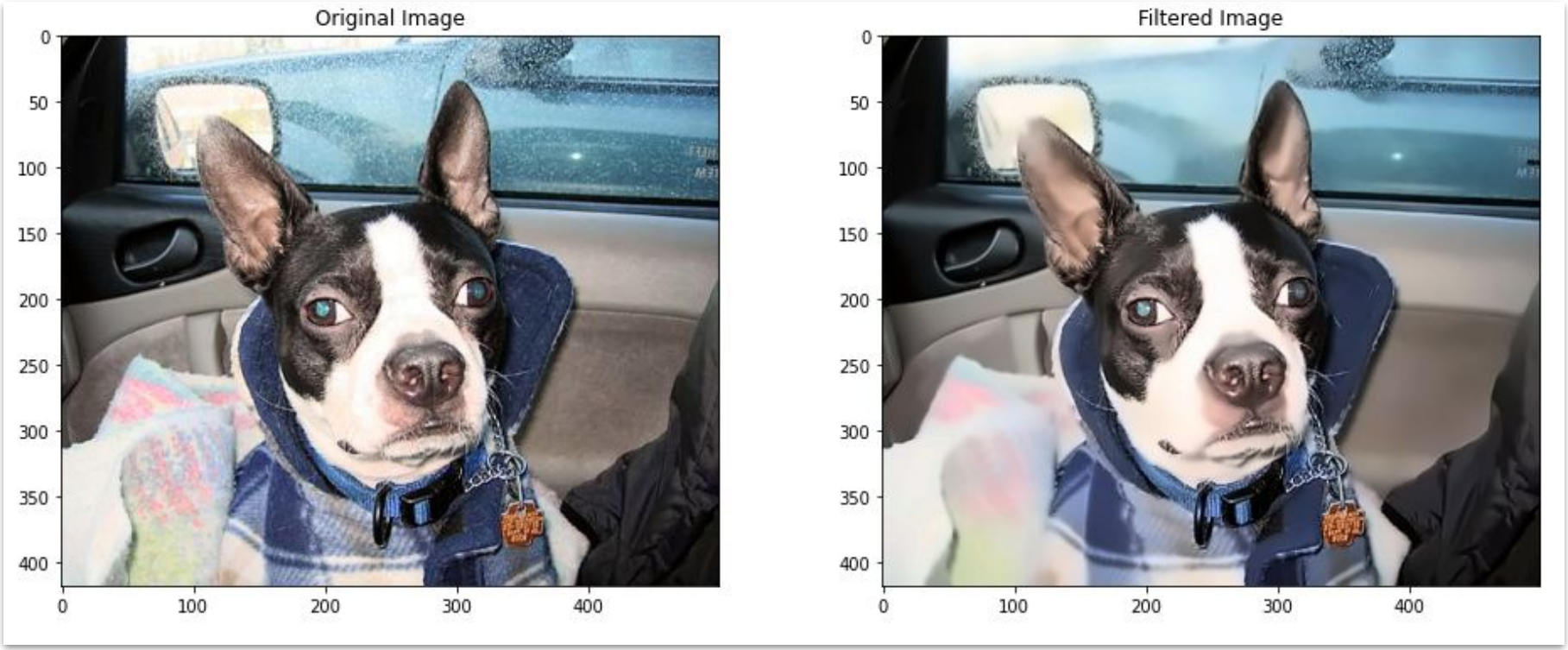

Original shape : (418, 500, 3)
0
50
100
150
200
250
300
350
400
0
100
200
300
400
Resized shape : (299, 299, 3)
0
50
100
150
200
250
0
50
100
150
200
250

Equalising

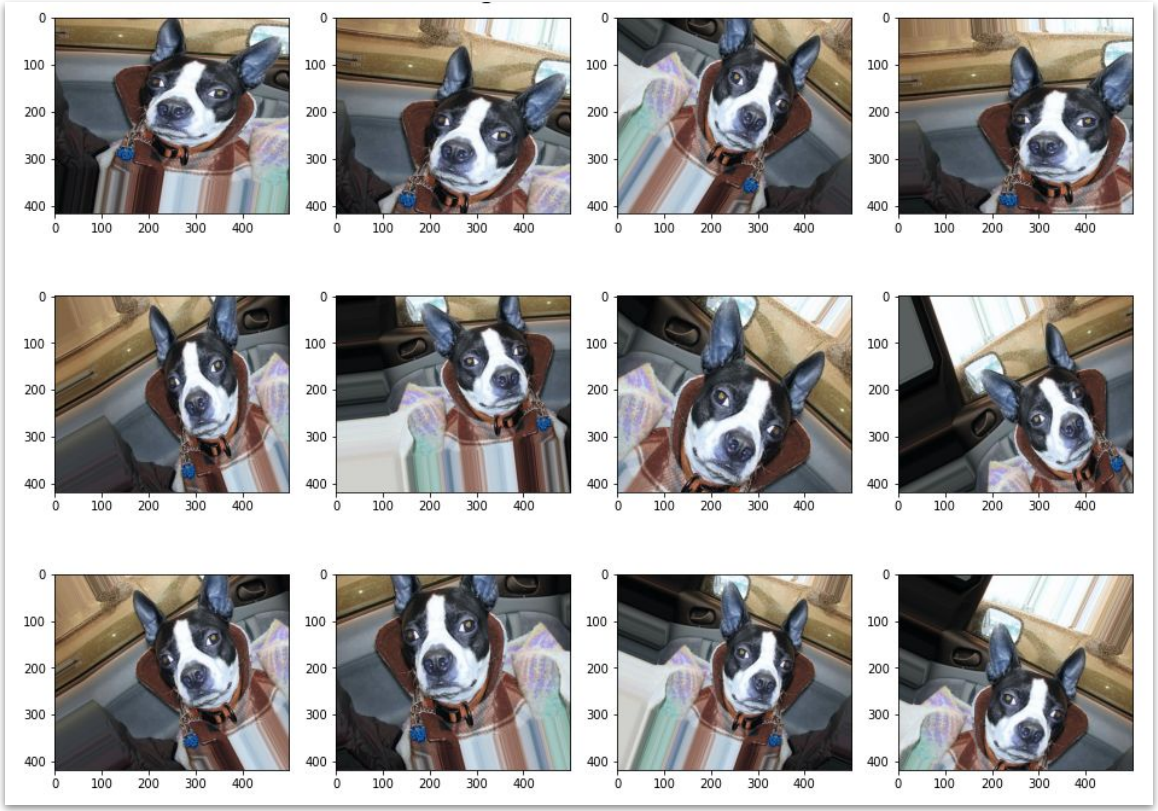

RGB Image
YUV Image
Histogram of convert image
cdf
histogram


RGB Image
Equalized Image
Histogram of convert image
cdf
histogram

Non Local means filter

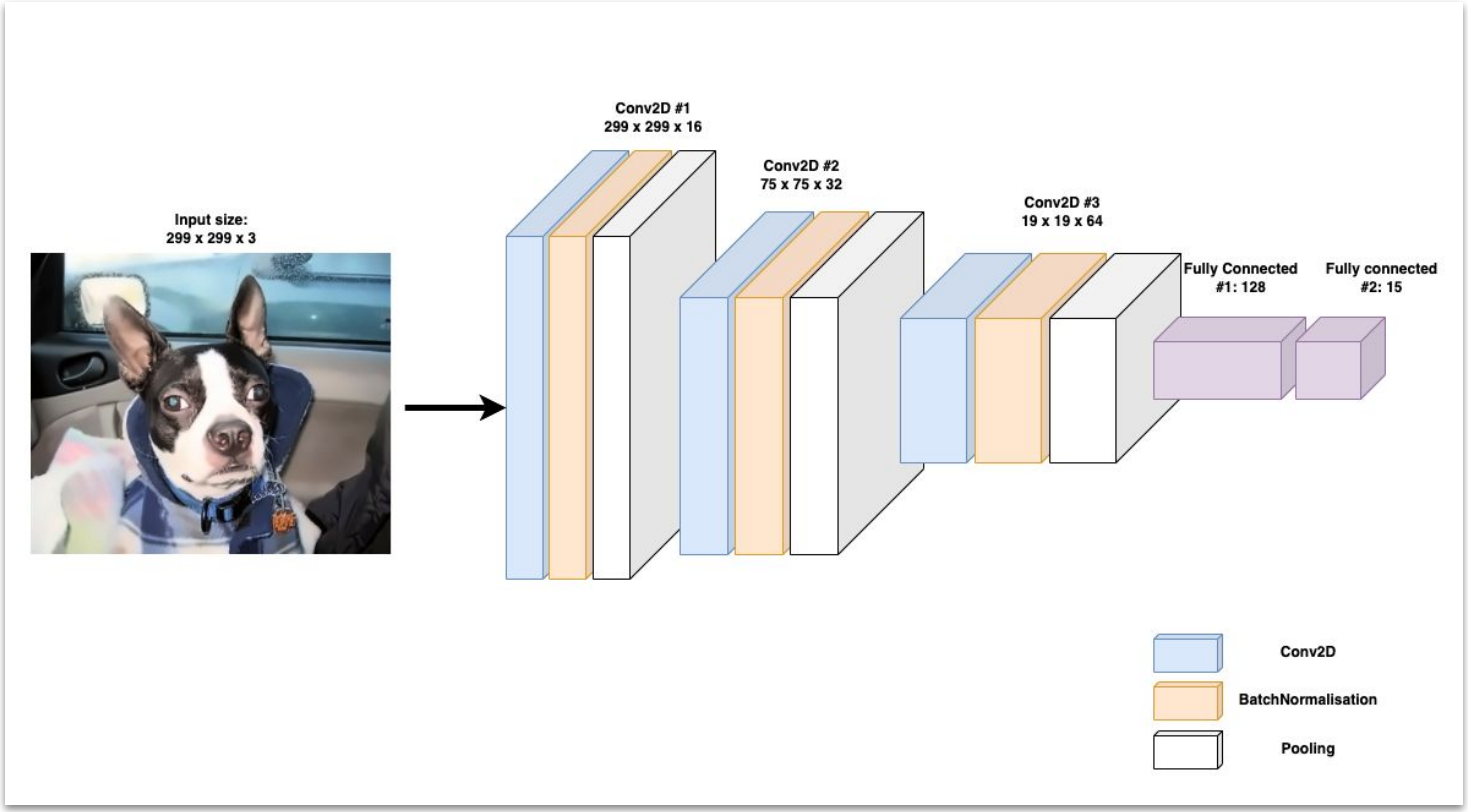


Data Augmentation



CNN “Fait maison”

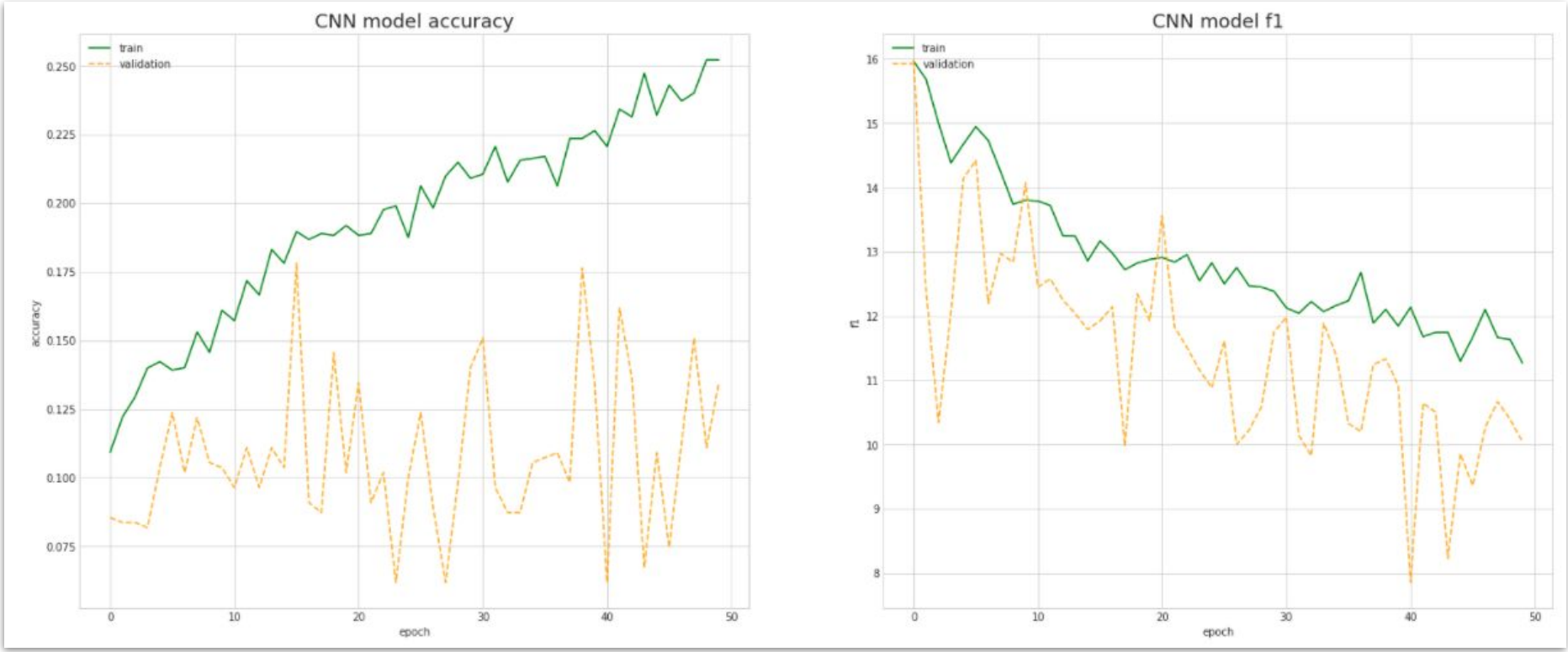
Structure de notre CNN (1/2)



Structure de notre CNN (2/2)

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 299, 299, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 299, 299, 16)      2352
_________________________________________________________________
batch_normalization (BatchNo (None, 299, 299, 16)      48
_________________________________________________________________
activation (Activation)      (None, 299, 299, 16)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        12800
_________________________________________________________________
batch_normalization_1 (Batch (None, 75, 75, 32)        96
_________________________________________________________________
activation_1 (Activation)    (None, 75, 75, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 19, 19, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 19, 19, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 19, 19, 64)        18432
_________________________________________________________________
batch_normalization_2 (Batch (None, 19, 19, 64)        192
_________________________________________________________________
activation_2 (Activation)    (None, 19, 19, 64)        0
_________________________________________________________________
global_average_pooling2d (Gl (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 128)               8320
_________________________________________________________________
dense_1 (Dense)              (None, 20)                2580
=================================================================
Total params: 44,820
Trainable params: 44,596
Non-trainable params: 224
_________________________________________________________________
```

Nos résultats



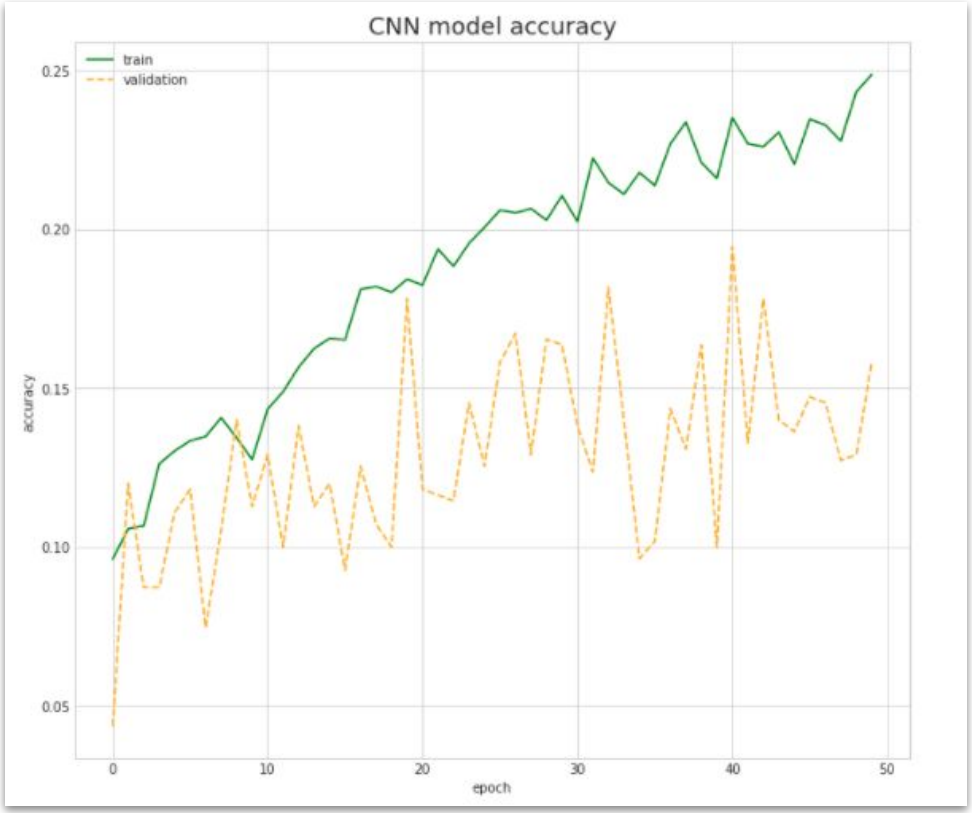
Hyperparamétrage de notre CNN

Hyperparamétrage de notre CNN

```
[33]: def hpt_cnn_model_builder(hp):
          model = custom_cnn_builder()

          # Tune the learning rate and beta1 for the optimizer
          # Choose an optimal value from 0.01, 0.001, or 0.0001
          hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
          hp_beta1 = hp.Choice('beta_1', values=[0.9,0.95,0.99])

          model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate,beta_1=hp_beta1),
                        loss="sparse_categorical_crossentropy",
                        metrics=['accuracy',f1_m])
          return model
```

- Nous allons optimiser les paramètres de learning rate et beta1 de notre descente de gradient

Hyperparamétrage de notre CNN



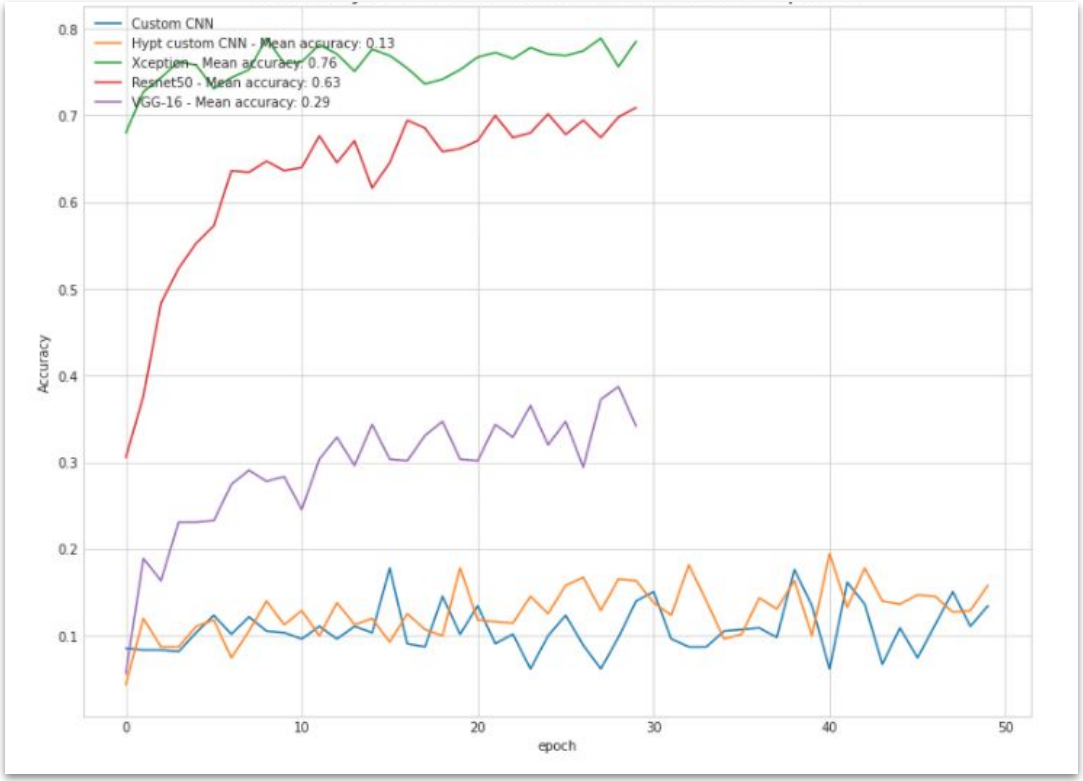
- Nos résultats sont relativement mauvais, il nous faut soit optimiser d'autres paramètres, ou utiliser du transfer learning
- Très légère amélioration grâce à notre hyperparamétrage

Transfer Learning

Nous allons utiliser les CNN suivants

- Nous allons utiliser 3 CNN bien connues qui ont été entrainés sur la base de données ImageNet:
  - VGG-16 (2014)
  - Xception (2017)
  - ResNet50 (2015)
- Nous gelons les couches des différents CNN afin de ne pas les réentrainer, car nous voulons garler les poids optimaux obtenus avec ImageNet

Résultats sur set de validation et choix de CNN



- Xception sera notre choix pour la suite de notre exercice, car il offre la meilleure performance en se basant sur l'accuracy, par rapport aux autres CNN

Hyperparametrage de Xception

Hyperparametrage et réentrainement

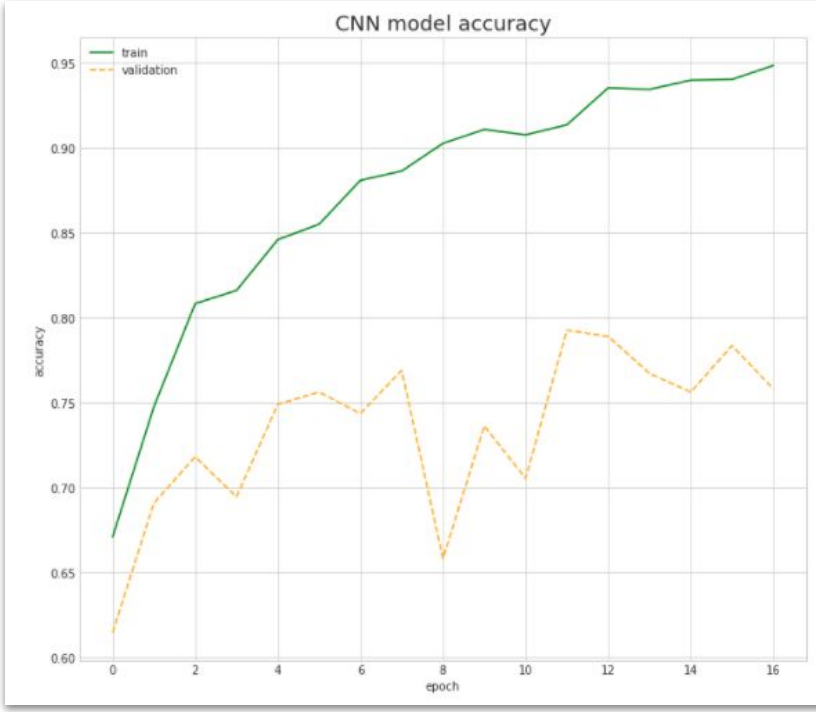
- Nous gelons les 115 premières couches de Xception afin de ne pas les entrainer car nous souhaitons uniquement ré-entrainer la couche de sortie.
- Nous mettons en place un callback avec une patience de 5 epochs afin de ne pas surentrainer notre modèle.

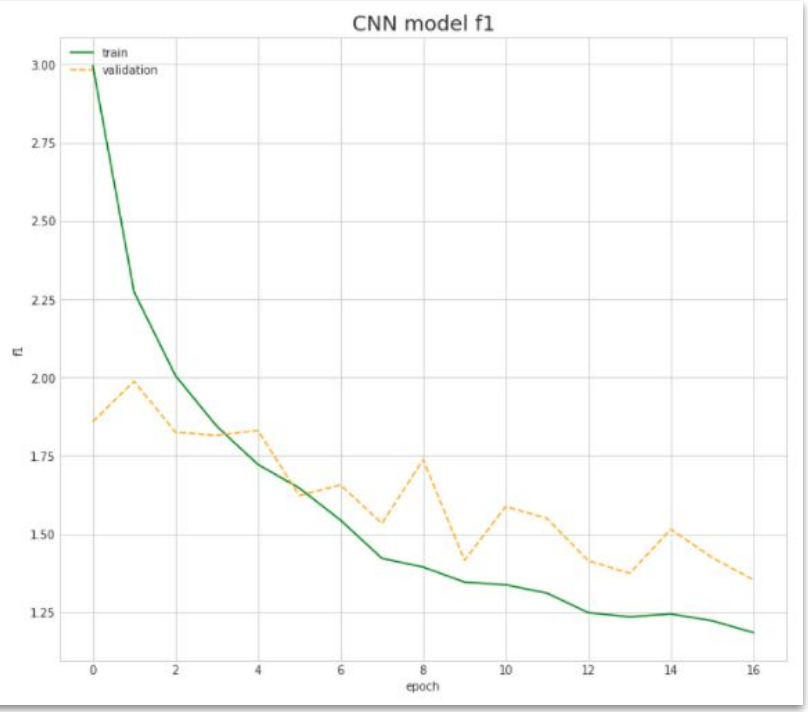
Hyperparametrage et réentrainement

[74]:
```python
def xception_fine_tune(nb_layers):
    """
    Raison d'être: Training some parts only of model and returning the trained model
    Args:
        nb_layer: number of layers NOT to be trained
    Returns:
        hypermodel_t: the newly trained model
    """
    # Load the pre trained model
    hypermodel_t = load_model('./xception_hypermodel.h5', custom_objects={"f1": f1})

    # re train the last layers
    for i, layer in enumerate(hypermodel_t.layers):
        if i < nb_layers:
            layer.trainable = False
        else:
            layer.trainable = True

    # Compile model
    hypermodel_t.compile(
        optimizer='adam',
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy", f1])

    return hypermodel_t
```

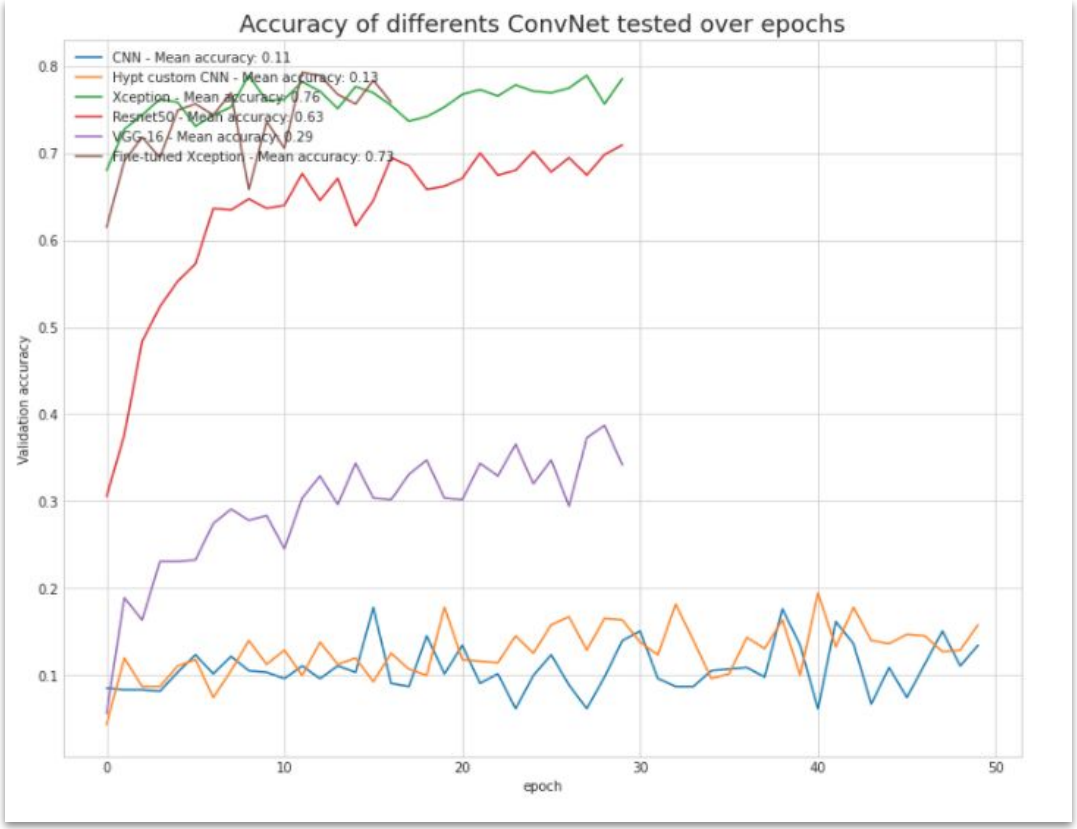
[75]:
```python
# Define a early stopping
stop_early = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    patience=5)

# Dont train the 115 first layers. Last layer excluding the exit flow is layer 115. Then we get in the exit flow.
my_tuned_xcept_model = xception_fine_tune(115)
fine_tuned_history = my_tuned_xcept_model.fit(
    train_datagen.flow(
        x_train, y_train,
        batch_size=BATCH_SIZE,
        shuffle=False,
        subset='training'),
    epochs=EPOCHS,
    validation_data=train_datagen.flow(
        x_train, y_train,
        batch_size=BATCH_SIZE,
        shuffle=False,
        subset='validation'),
    callbacks=[stop_early],
    verbose=2)
```

Résultats d’hyperparamétrage (1/2)





Résultats d’hyperparamétrage (2/2)



- Encore une fois peu d’apports significatifs de notre hyperparamétrage de la couche de sortie de notre Xception

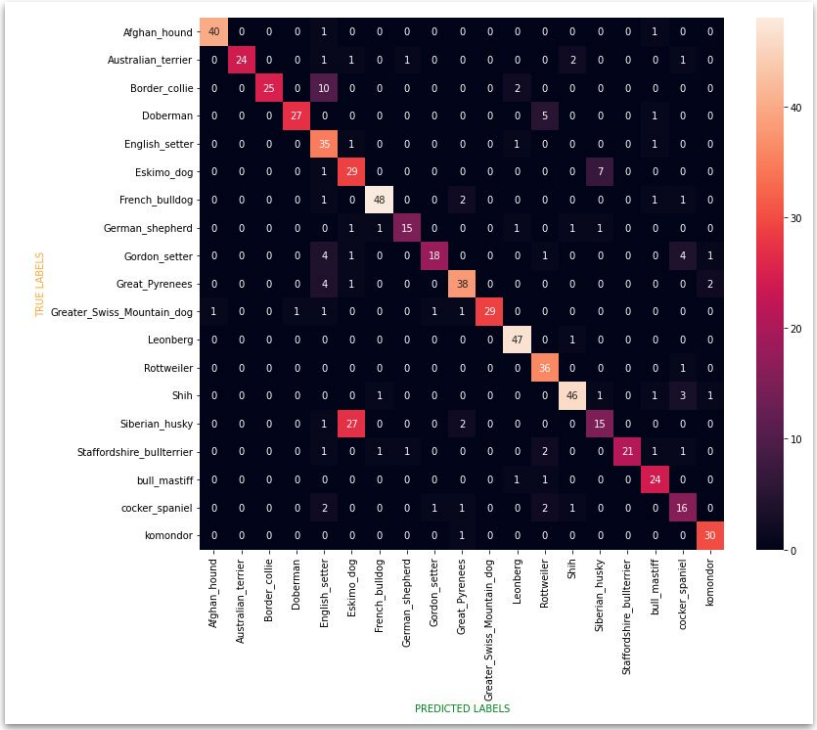
Xception sur Test set

Code pour faire tourner notre modèle sur le test set

[78]:
```
# Model evaluation on test set
xception_eval = fine_tuned_history.model.evaluate(
    test_datagen.flow(
        x_test, y_test,
        batch_size=BATCH_SIZE,
        shuffle=False),
    verbose=1)
print("-" * 50)
print("Xception model evaluation :")
print("-" * 50)
print('Test Loss: {:.3f}'.format(xception_eval[0]))
print('Test Accuracy: {:.3f}'.format(xception_eval[1]))
print('Test F1 score: {:.3f}'.format(xception_eval[2]))
```

```
44/44 [==============================] - 5s 113ms/step - loss: 0.7701 - accuracy: 0.8171 - f1: 1.3076
--------------------------------------------------
Xception model evaluation :
--------------------------------------------------
Test Loss: 0.770
Test Accuracy: 0.817
Test F1 score: 1.308
```

Matrice de Confusion et Résultats par classe



	precision	recall	f1-score	support
Afghan_hound	0.98	0.95	0.96	42
Australian_terrier	1.00	0.80	0.89	30
Border_collie	1.00	0.68	0.81	37
Doberman	0.96	0.82	0.89	33
English_setter	0.56	0.92	0.70	38
Eskimo_dog	0.48	0.78	0.59	37
French_bulldog	0.94	0.91	0.92	53
German_shepherd	0.88	0.75	0.81	20
Gordon_setter	0.90	0.62	0.73	29
Great_Pyrenees	0.84	0.84	0.84	45
Greater_Swiss_Mountain_dog	1.00	0.85	0.92	34
Leonberg	0.90	0.98	0.94	48
Rottweiler	0.77	0.97	0.86	37
Shih	0.90	0.87	0.88	53
Siberian_husky	0.62	0.33	0.43	45
Staffordshire_bullterrier	1.00	0.75	0.86	28
bull_mastiff	0.80	0.92	0.86	26
cocker_spaniel	0.59	0.70	0.64	23
komondor	0.88	0.97	0.92	31
accuracy			0.82	689
macro avg	0.84	0.81	0.81	689
weighted avg	0.84	0.82	0.82	689

Merci de votre attention!