Parcours Ingénieur Machine Learning

Session Mars 2021

OPENCLASSROOMS

Projet 6
Classez des images à l'aide d'algorithmes de Deep Learning

07/10/2021

Etudiante : QITOUT Kenza

Mentor : Maïeul Lombard

Evaluateur : Renel Cadin Demanou

CONTEXTE DU PROJET

Bénévole pour l'association de protection des animaux

Problématique : Base de données des animaux trop grande pour référencer toutes les images des animaux

Objectif : Créer un algorithme pour classer les images en fonction de la race du chien présent sur l'image

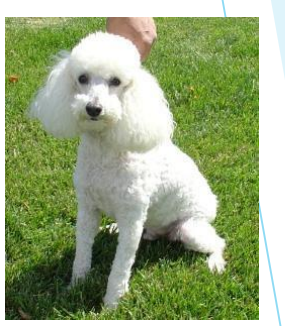
Missions :

- Pré-processer les images et réaliser une data augmentation
- Utiliser 2 approches de Réseaux de Neurones Convolutionnels (créer son propre réseau et utiliser le Transfer Learning)
- Présenter le modèle final sélectionné pour chaque approche, leur optimisation et leurs performances

BASES DE DONNEES

Jeu de données utilisé pour entraîner l'algorithme : **Stanford Dogs Dataset**

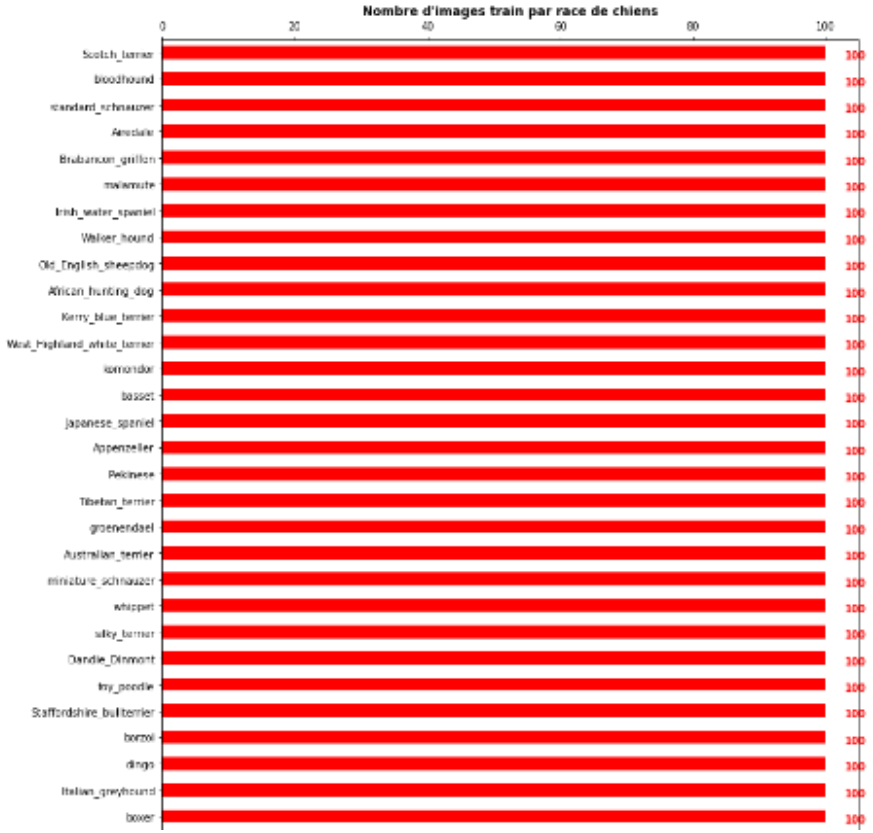


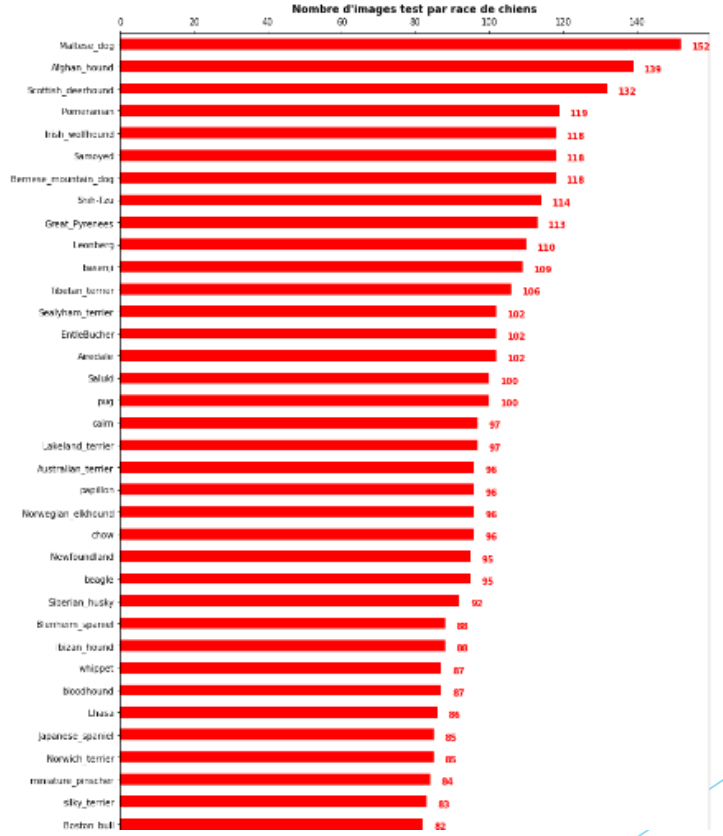



Base de données contenant 20 580 images de 120 races de chiens du monde entier (dossier de 757MB), les labels associés pour la catégorisation des images, séparées en 2 groupes train et test (liste des images train et test)

BASES DE DONNEES

Exploration des données :



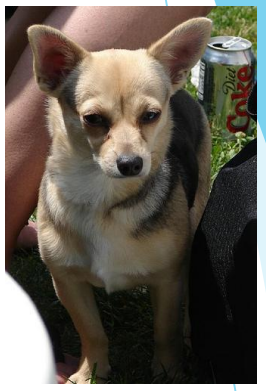


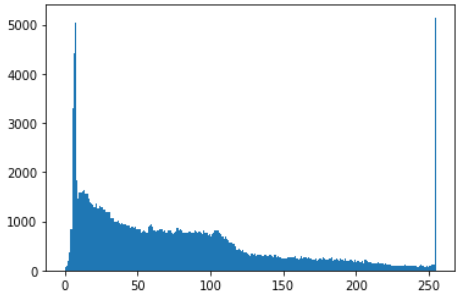
120 classes de chiens avec 100 images pour toutes les classes train et entre 162 et 48 images pour les classes test

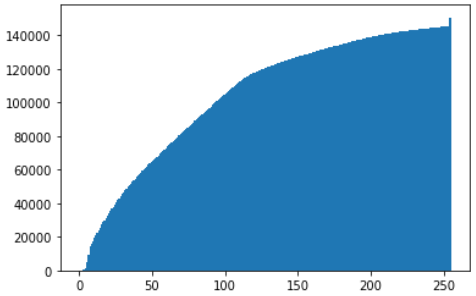
1. TRAITEMENT DES IMAGES

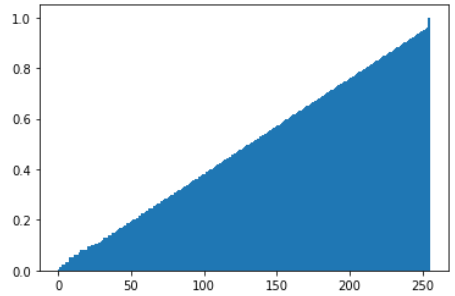
Traitements appliqués sur toutes les images les unes après les autres :

- Récupération des images dans le dossier images.tar
- Chargement de l'image en couleur avec *Image.open()*
- Redimensionner l'image en (224, 224)
- Egalisation de l'histogramme avec *ImageOps.equalize()*







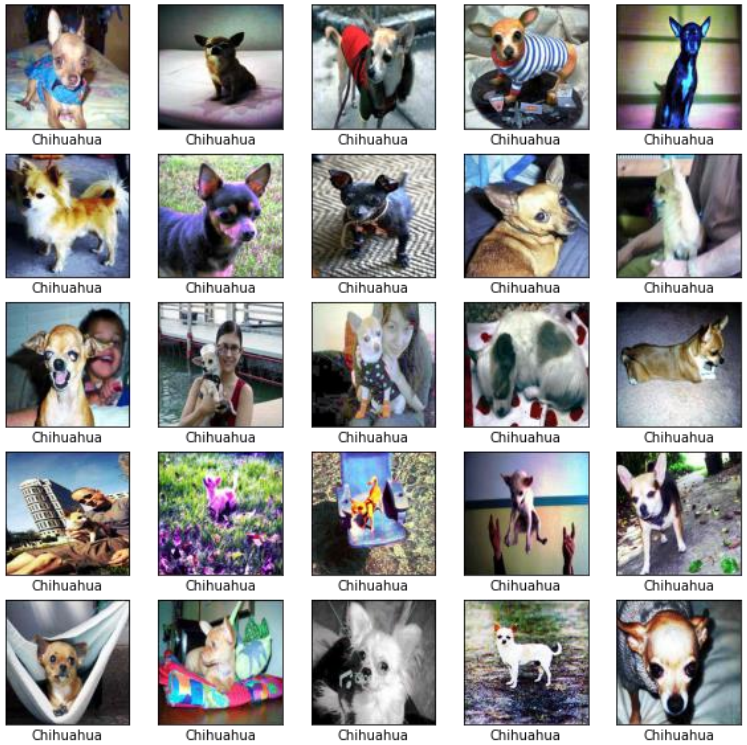


- Interpolation bilinéaire avec *rotate(0, resample=Image.BILINEAR)*
- Filtre moyenneur avec *filter(ImageFilter.BoxBlur(0))*

1. TRAITEMENT DES IMAGES

Création des jeux de données :

- Sauvegarde de la nouvelle image dans un dossier train/nom_race (12 000 images) ou dans un dossier test/nom_race (8 580 images)



- Création des matrices de pixels de toutes les images train et test

2. RESEAU DE NEURONES CNN

Réseau de neurones composé de différentes couches selon une architecture à définir et des hyperparamètres à optimiser (analyses réalisées sur 20 races de chiens)

Variables explicatives = Matrice des pixels des images train/255.0
Variables à expliquer = Vecteur des labels (Encodage de la liste des labels)

→ Création d'un jeu de données de validation stratifié X_valdation et y_validation avec train_test_split (0.2)

→ Définition de l'architecture du CNN à optimiser

filters couche de Conv = [32, 64, 128, 256, 512]
kernel_size de Conv = [3, 5]
pool_size de Pooling = [2, 3]
units de Dense = [32, 64, 128, 512, 1024]
learning_rate de Optimizers Adam = [0.1, 0.01, 0.001, 0.0001]
Loss de compile = ['mean_absolute_error', 'categorical_crossentropy']
nb_classes = 20
epochs = 50
batch_size = 64

→ Recherche des hyperparamètres optimaux en fonction de l'espace de recherche à tester avec BayesianOptimization sur X_train_train

→ Entrainement du meilleur modèle sur X_train_train

→ Calcul de l'**Accuracy**, de **Loss** et du **Temps de calcul** le jeu de données test X_test

→ Comparaison des résultats pour chaque modèle

2. RESEAU DE NEURONES CNN

Architecture du réseau de neurones convolutif de base :

```
model = Sequential()
model.add(Conv2D(filters, (kernel_size), input_shape=(224, 224, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(pool_size), strides=(2,2)))
```

X blocs de Convolution composés de **X** couches de Convolution et ReLu et de couches de Pooling

```
model.add(Flatten())
model.add(Dense(units, activation='relu'))
model.add(Dense(nb_classes, activation='softmax'))
```

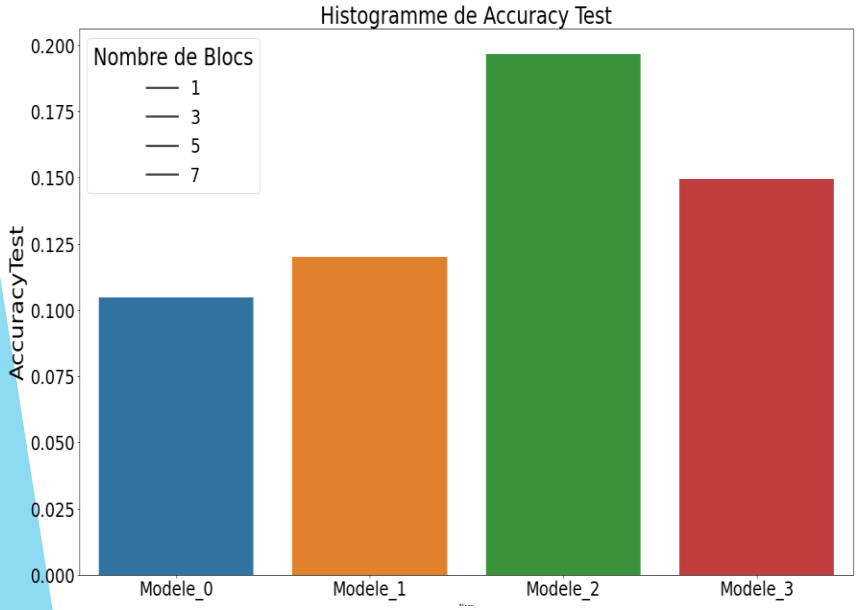
X couches Fully-connected

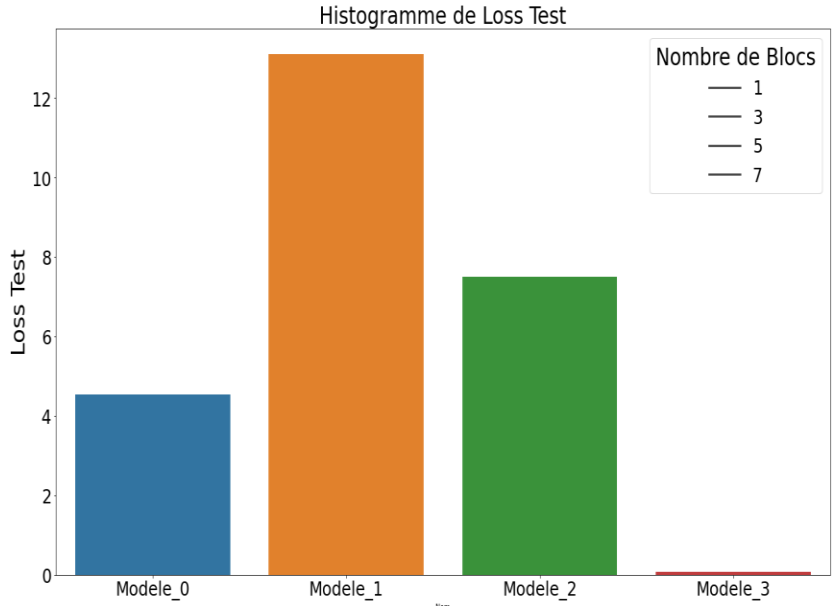
```
model.compile(optimizer=optimizers.Adam(learning_rate), loss=Loss,
 metrics=['accuracy'])
```

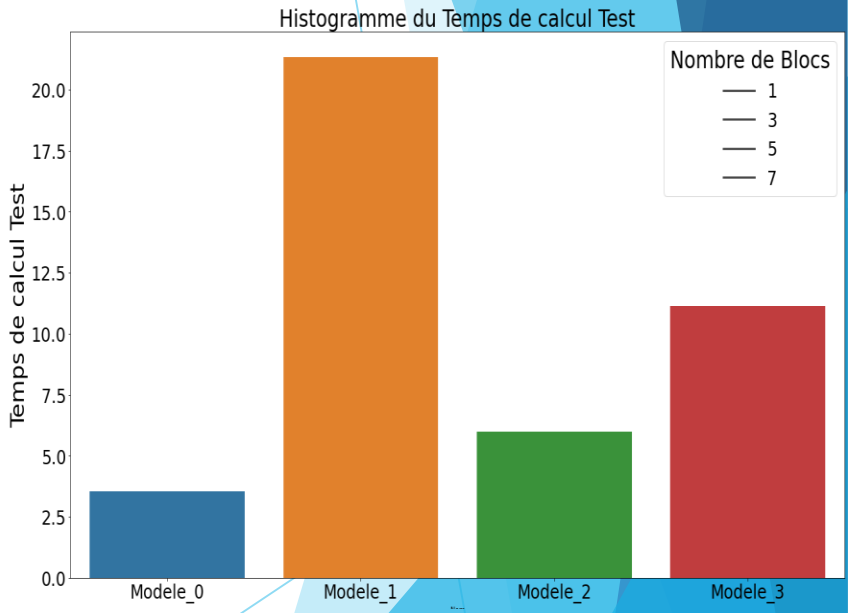
2. RESEAU DE NEURONES CNN

Comparaison des résultats :

- Choix du nombre de blocs de Convolution = [1, 3, 5, 7]





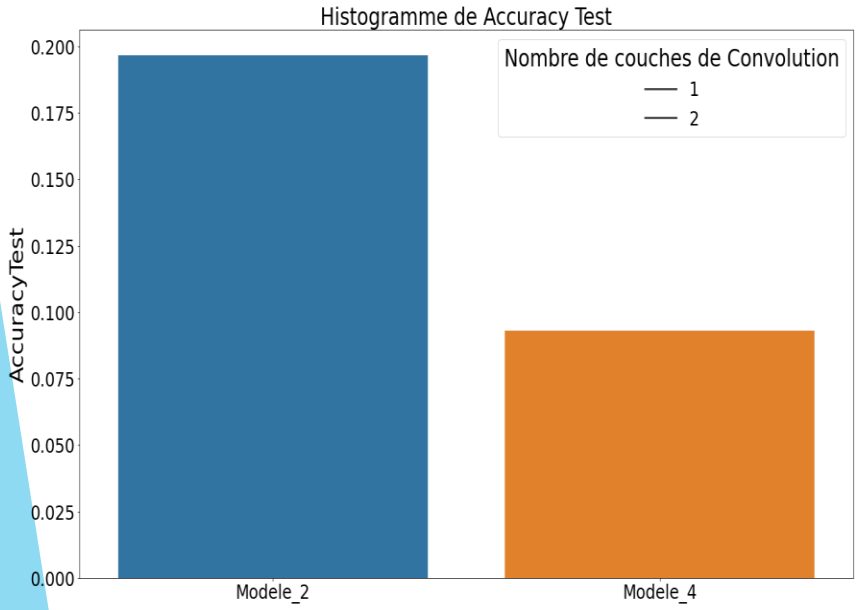


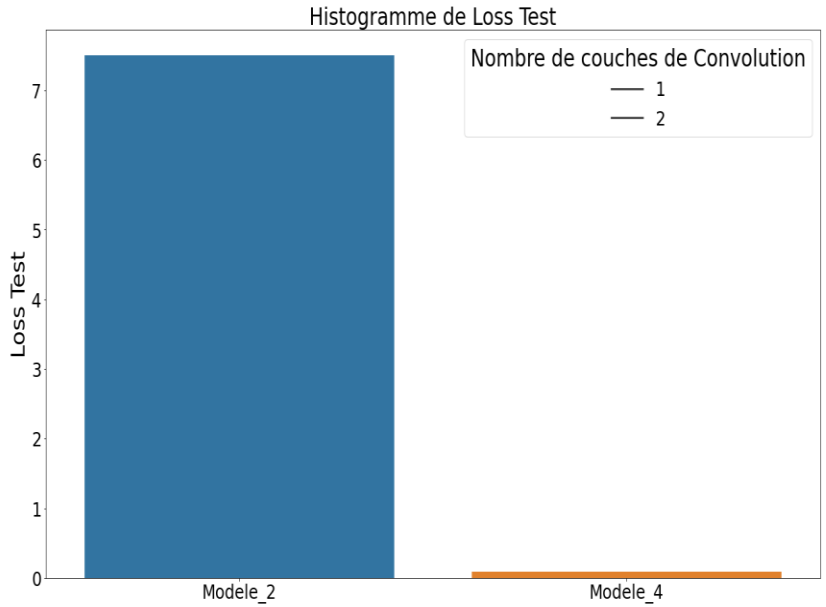
Meilleurs résultats obtenus pour 5 Blocs de Convolution : Accuracy = 0.196571, Loss = 7.499770 et Temps de calcul = 5.975756

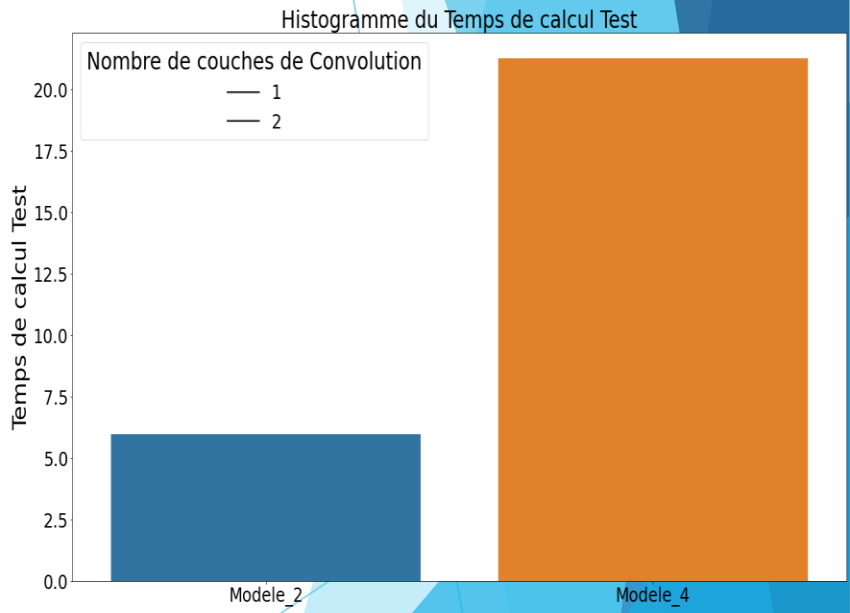
2. RESEAU DE NEURONES CNN

Comparaison des résultats :

- Choix du nombre de couches Conv et ReLu = [1, 2] pour 5 Blocs de Convolution





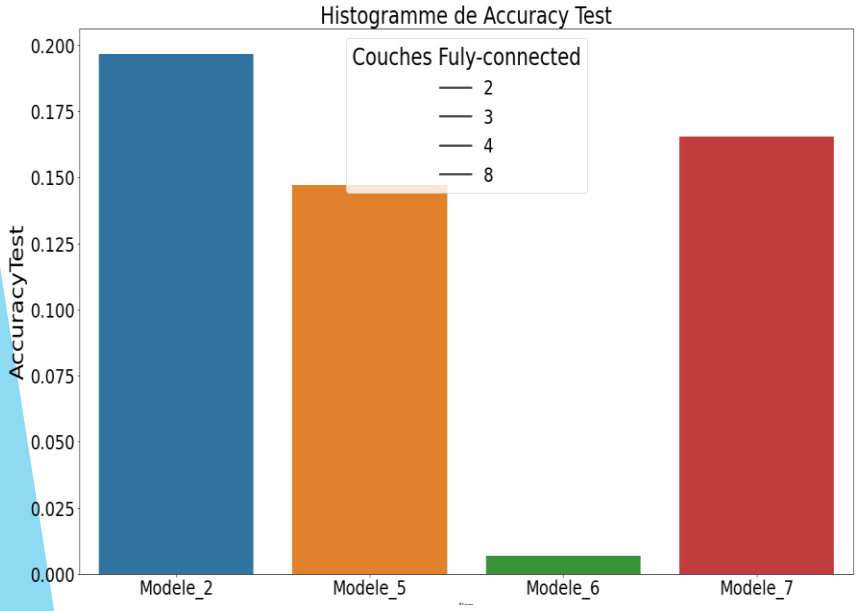


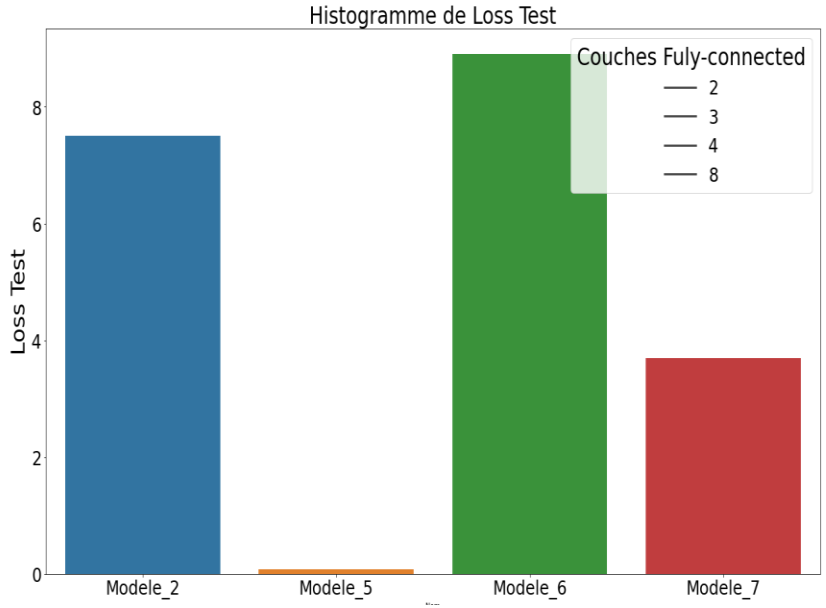
Meilleurs résultats obtenus pour 1 couche de Convolution et de correction ReLu pour 5 Blocs : Accuracy = 0.196571, Loss = 7.499770 et Temps de calcul = 5.975756

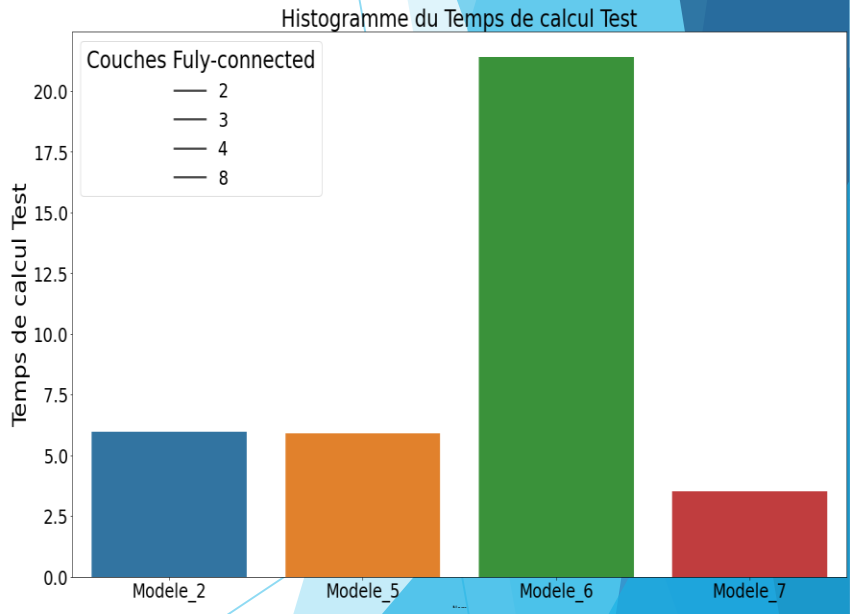
2. RESEAU DE NEURONES CNN

Comparaison des résultats :

- Choix du nombre de couches Dense = [2, 3, 4, 8] pour 5 Blocs d'1 couche de Convolution





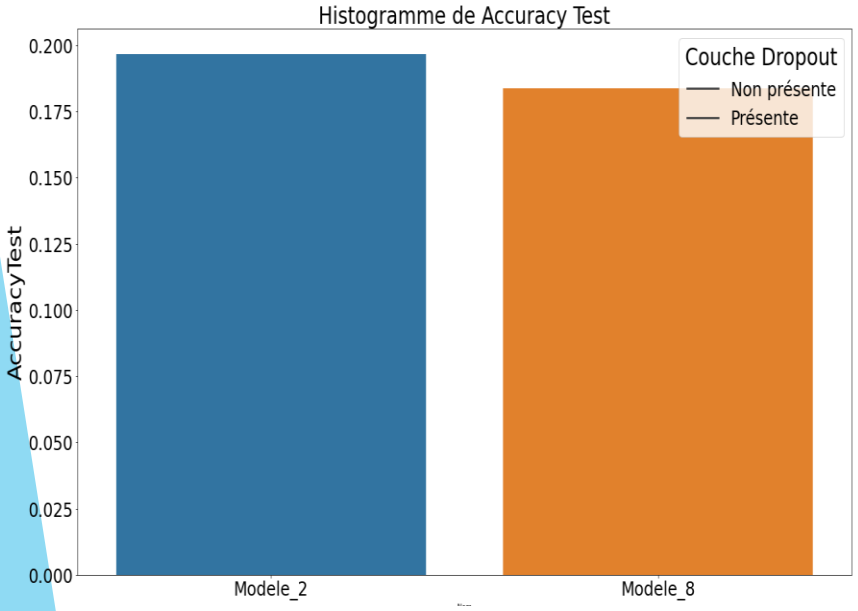


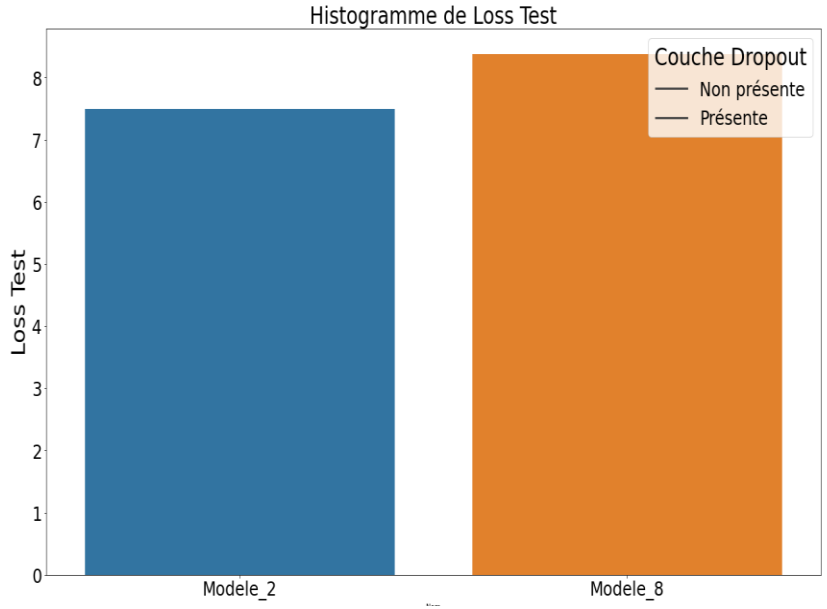
Meilleurs résultats obtenus pour 2 couches Fully-connected pour 5 Blocs d'1 couche de Convolution : Accuracy = 0.196571, Loss = 7.499770 et Temps de calcul = 5.975756

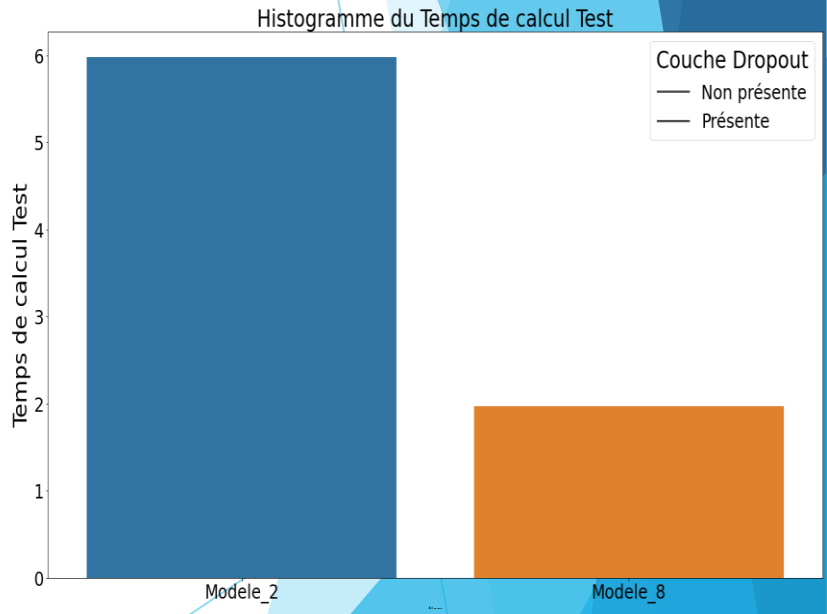
2. RESEAU DE NEURONES CNN

Comparaison des résultats :

- Ajout d'une couche Dropout = [0, 0.1, 0.2, 0.3, 0.4, 0.5] pour 5 Blocs d'1 couche de Convolution et 2 couches Fully-connected





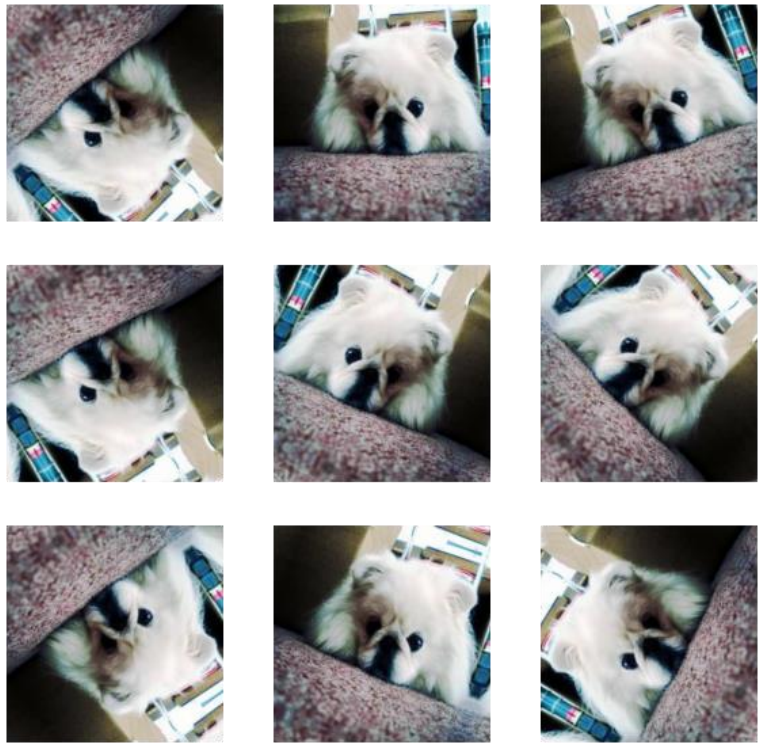


Pas de meilleurs résultats en ajoutant une couche Dropout pour 5 Blocs d'1 couche de Convolution et 2 couches Fully-connected

2. RESEAU DE NEURONES CNN

Data augmentation : Augmentation de la taille du jeu d'entraînement avec :

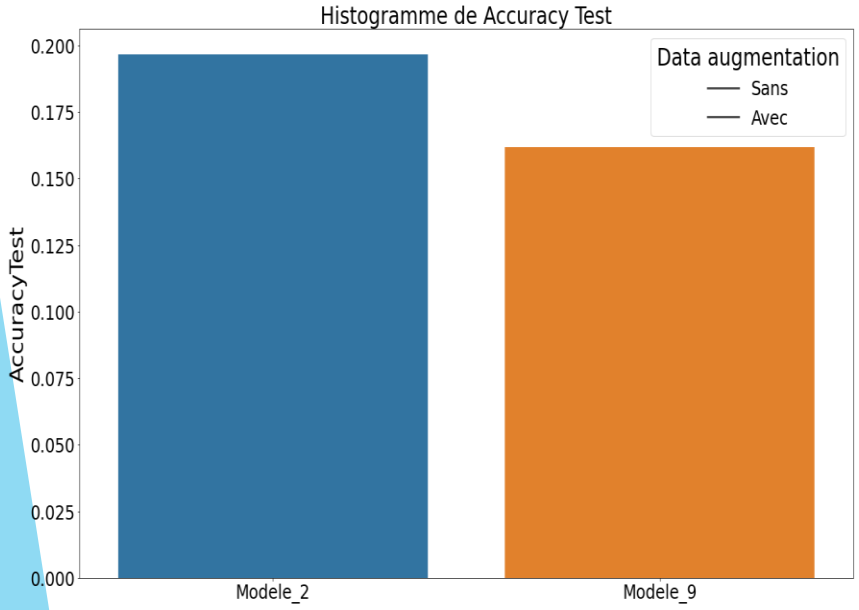
- Rotation de l'image horizontale et verticale
- Rotation de l'image = [0.1, 0.3, 0.5, 0.7, 0.9]

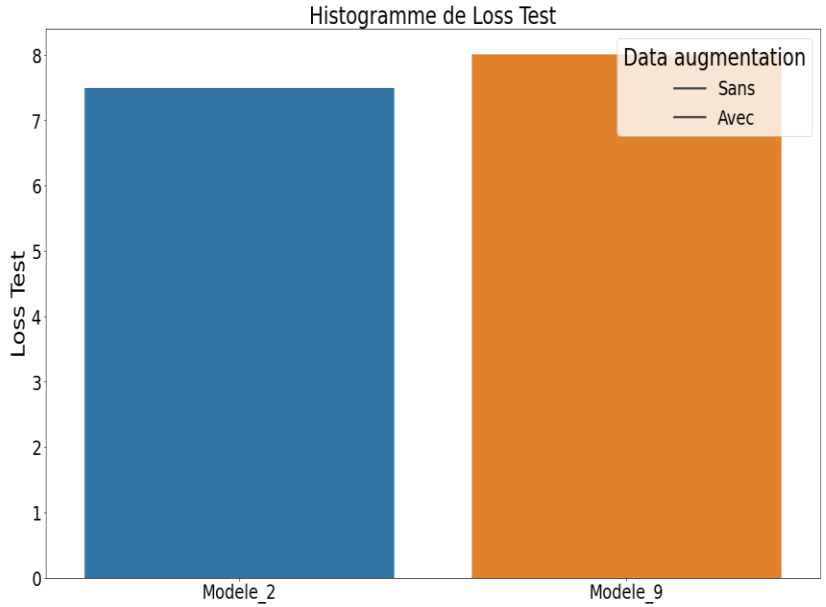


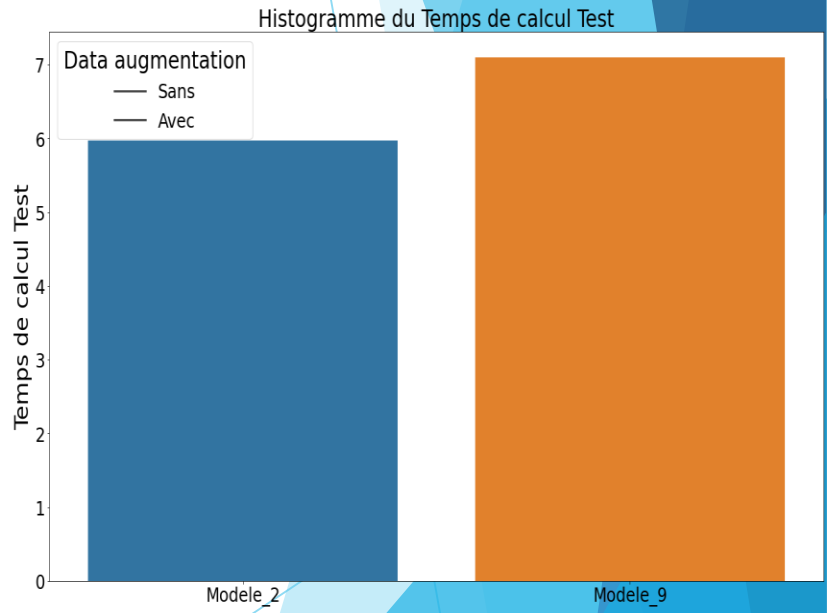
2. RESEAU DE NEURONES CNN

Comparaison des résultats :

- Data augmentation pour 5 Blocs d'1 couche de Convolution, 2 couches Fully-connected







Meilleurs résultats obtenus sans Data augmentation : Accuracy = 0.196571, Loss = 7.499770 et Temps de calcul = 5.975756

2. RESEAU DE NEURONES CNN

Modèle sélectionné : Architecture composée de 5 Blocs d'1 couche de Convolution et de correction ReLu, 2 couches Fully-connected

Recherche des hyperparamètres optimaux epochs et batch_size :

- Estimation de l'epochs optimal avec val_accuracy max = **22 epochs**
- Choix de batch_size sur [10, 20, 32, 40, 60, 80 et 96] = **96 batch_size**

Pour batch_size = 10 : Loss = 8.48, Accuracy = 0.15 et Temps de calcul = 5.22

Pour batch_size = 20 : Loss = 8.53, Accuracy = 0.16 et Temps de calcul = 5.63

Pour batch_size = 32 : Loss = 8.51 Accuracy = 0.15 et Temps de calcul = 3.97

Pour batch_size = 40 : Loss = 8.41, Accuracy = 0.16 et Temps de calcul = 5.27

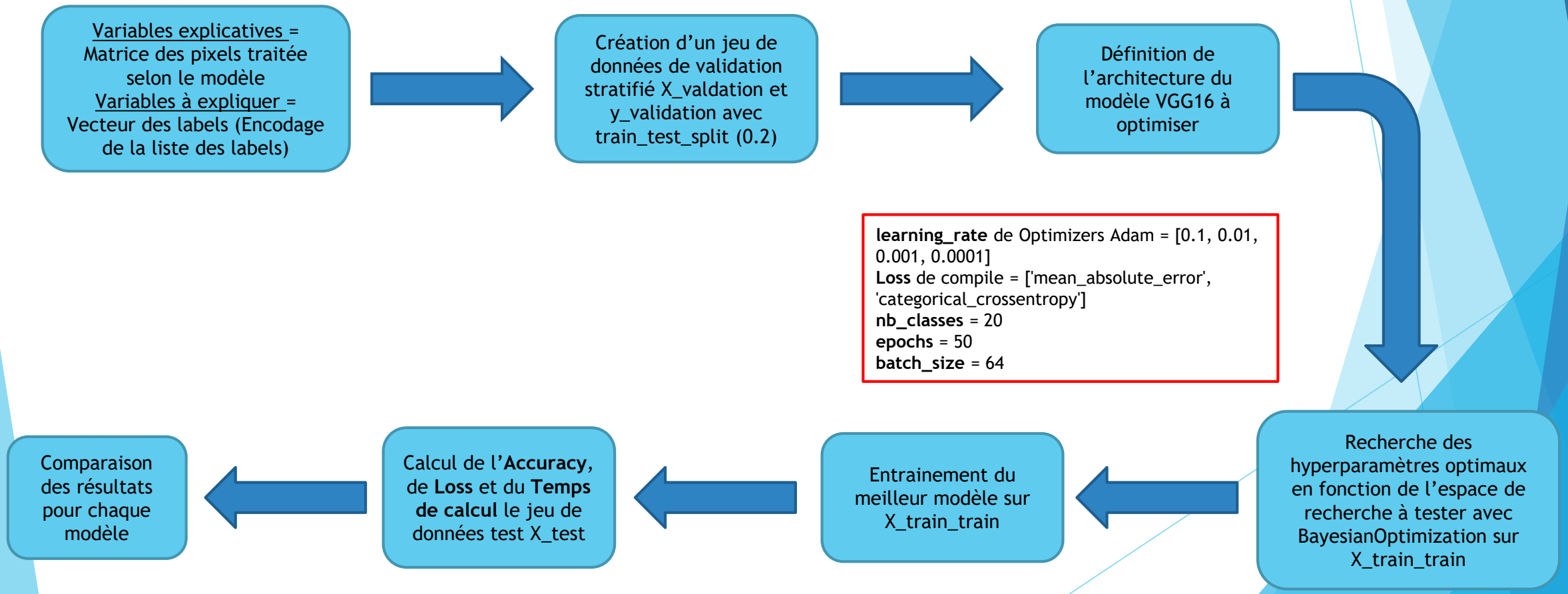
Pour batch_size = 60 : Loss = 8.57 Accuracy = 0.16 et Temps de calcul = 4.14

Pour batch_size = 80 : Loss = 8.81, Accuracy = 0.16 et Temps de calcul = 3.06

Pour batch_size = 96 : Loss = 10.15, Accuracy = 0.17 et Temps de calcul = 3.81

3. TRANSFER LEARNING

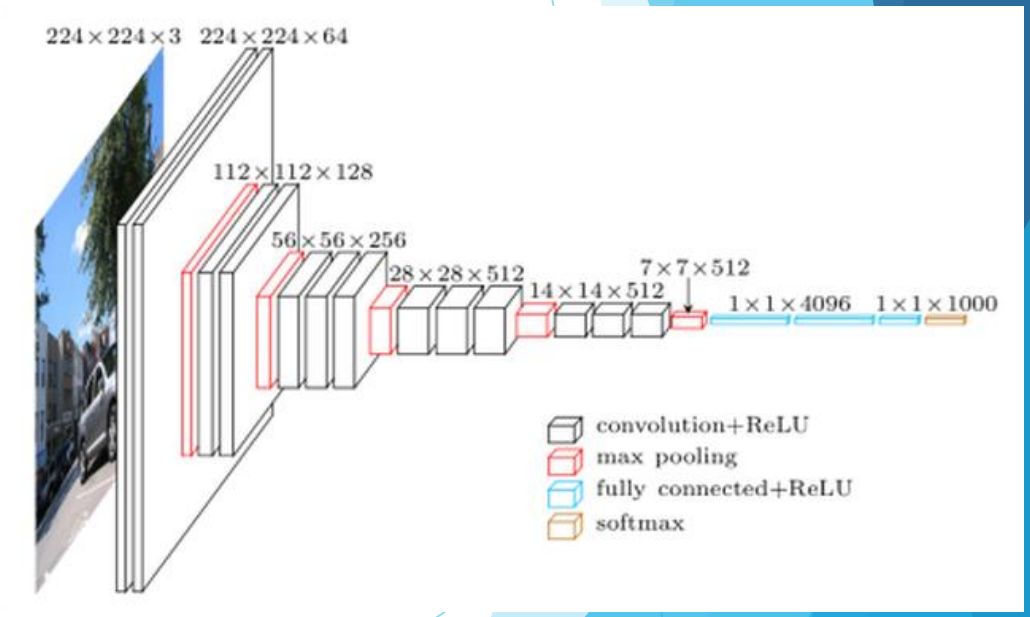
Réseau de neurones **VGG16** déjà pré-entraîné sur ImageNet à optimiser selon la stratégie à adopter et les hyperparamètres (analyses réalisées sur 20 races de chiens)



3. TRANSFER LEARNING

Architecture du réseau de neurones VGG16 de base (Stratégie d'extraction de features) :

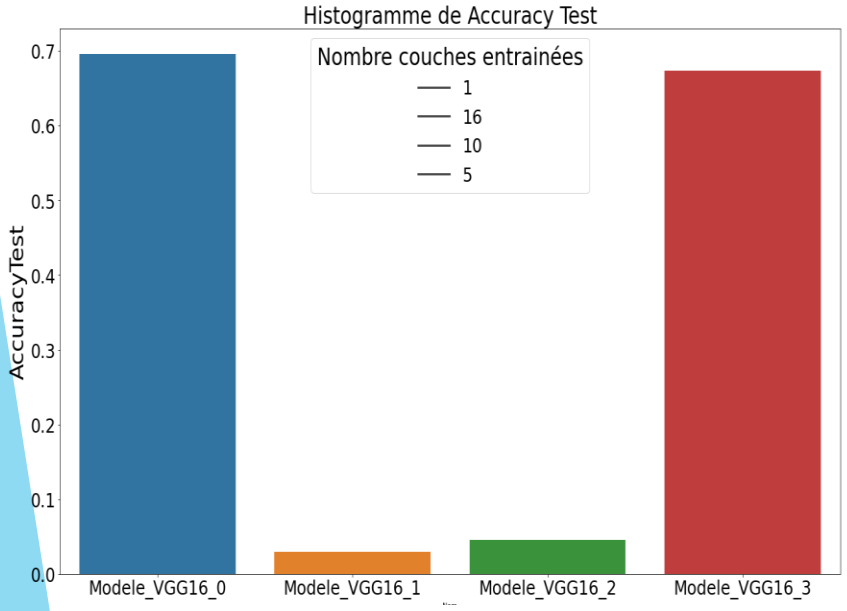
```
model = VGG16(weights="imagenet", include_top=False, inp
ut_shape=(224, 224, 3))

x = model.output

for layer in model_VGG16.layers:

  layer.trainable = False

predictions = Dense(nb_classes, activation='softmax')(x)

model = Model(inputs=model.input, outputs=predictions)

model.compile(optimizer=optimizers.Adam(learning_rate),
loss=Loss, metrics=['accuracy'])
```

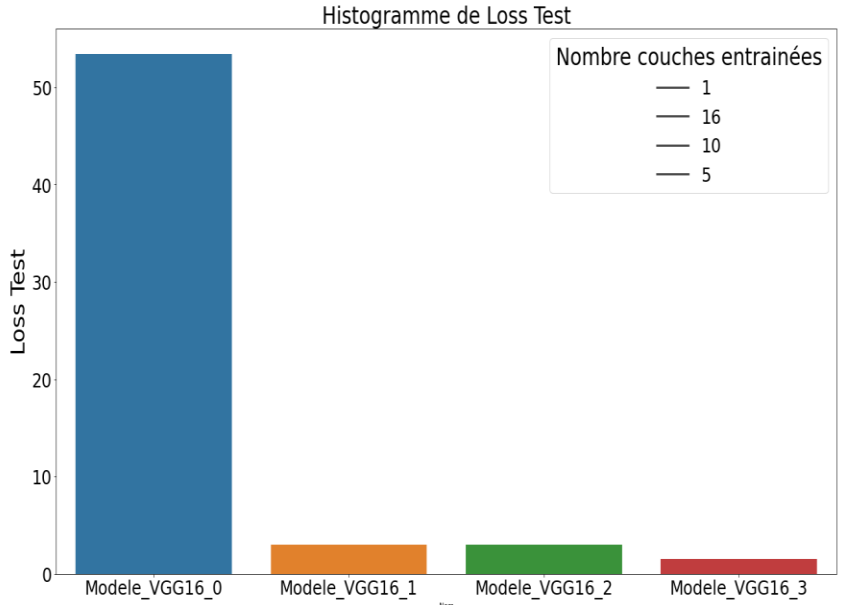


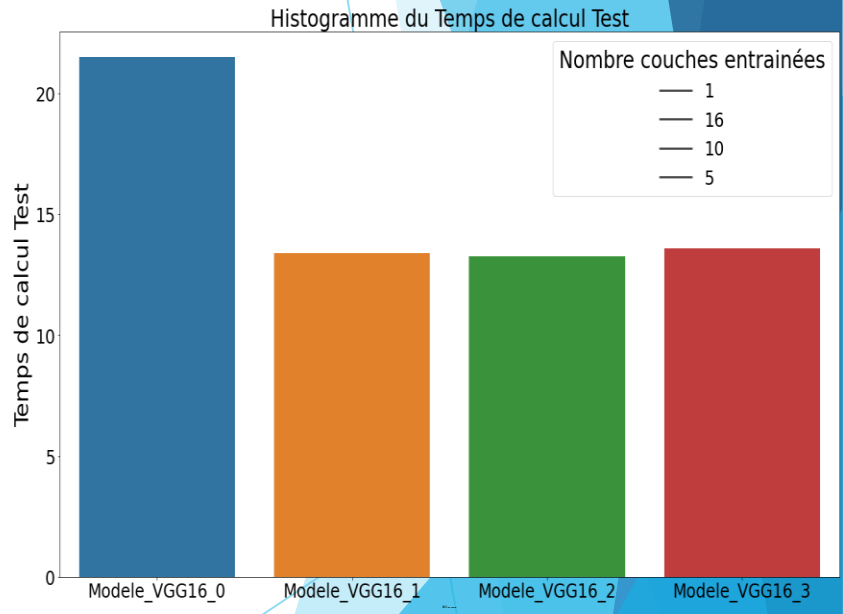
3. TRANSFER LEARNING

Comparaison des résultats :

- Choix du nombre de couches hautes à entraîner = [1, 5, 10, 16]





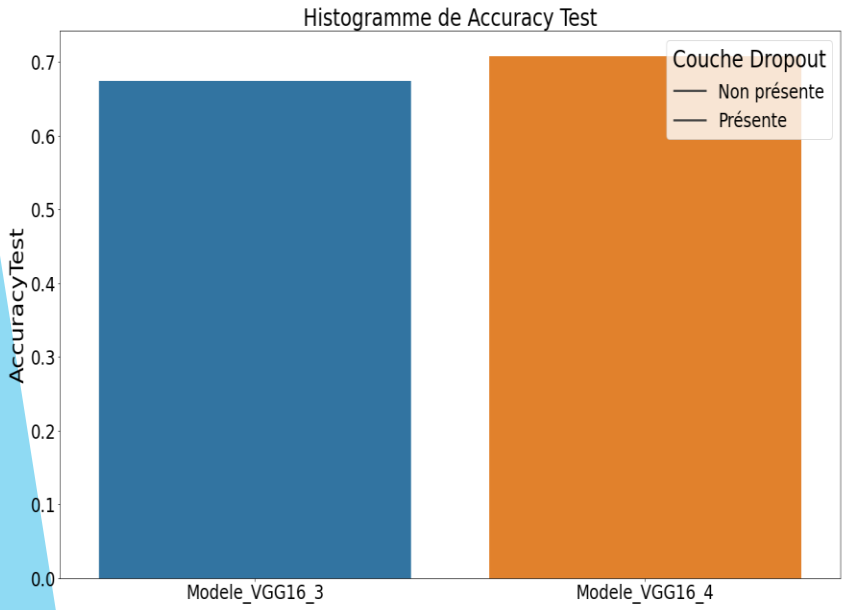


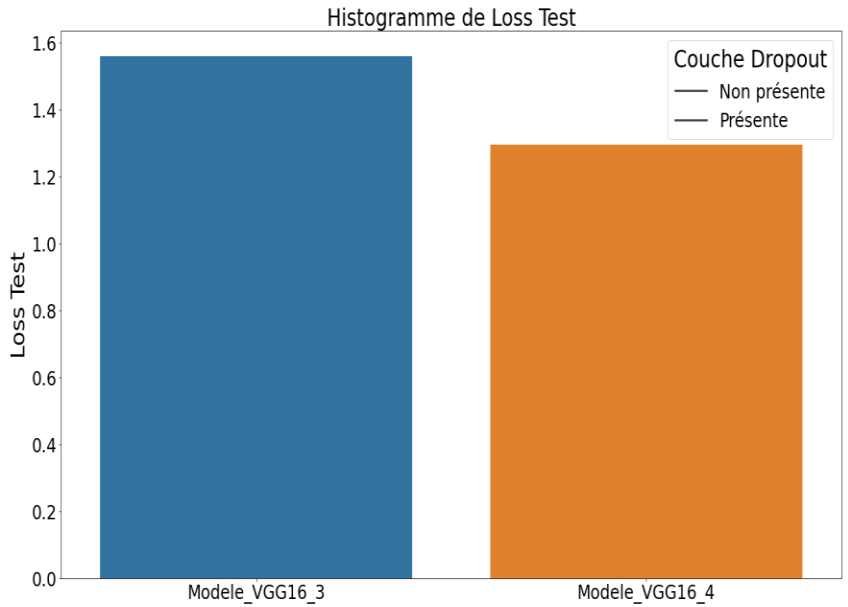
Meilleurs résultats obtenus en entraînant les 5 couches les plus hautes : Accuracy = 0.673607, Loss = 1.558877 et Temps de calcul = 13.578077

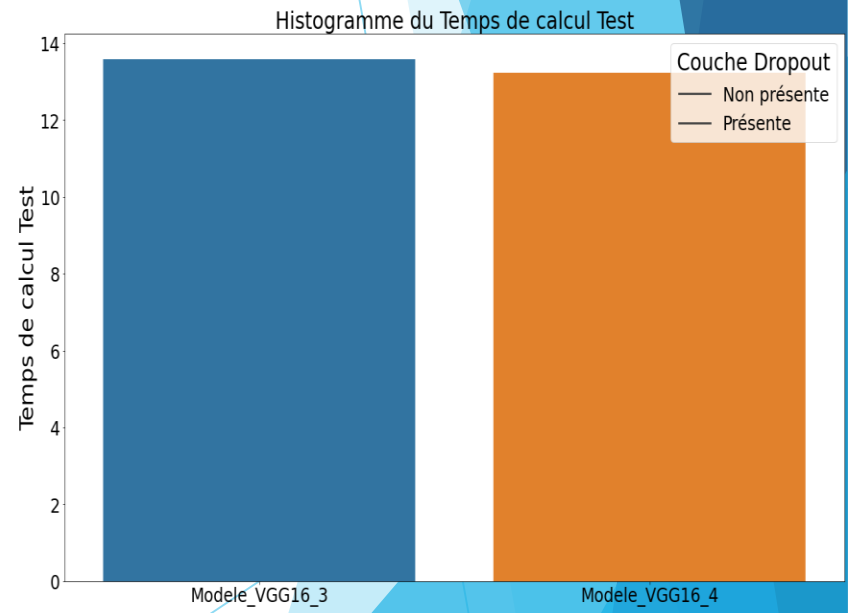
3. TRANSFER LEARNING

Comparaison des résultats :

- Ajout d'une couche Dropout = [0, 0.1, 0.2, 0.3, 0.4, 0.5] pour la stratégie fine-tuning partiel





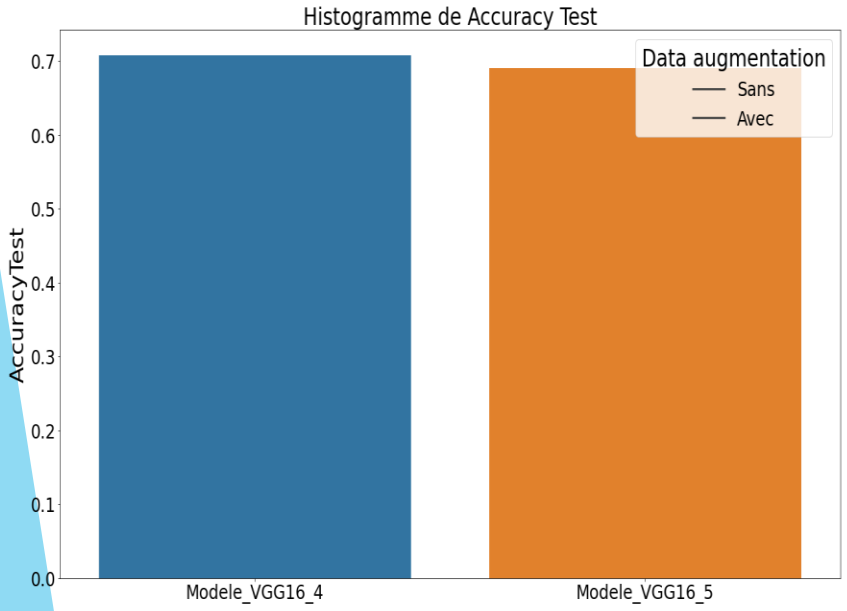


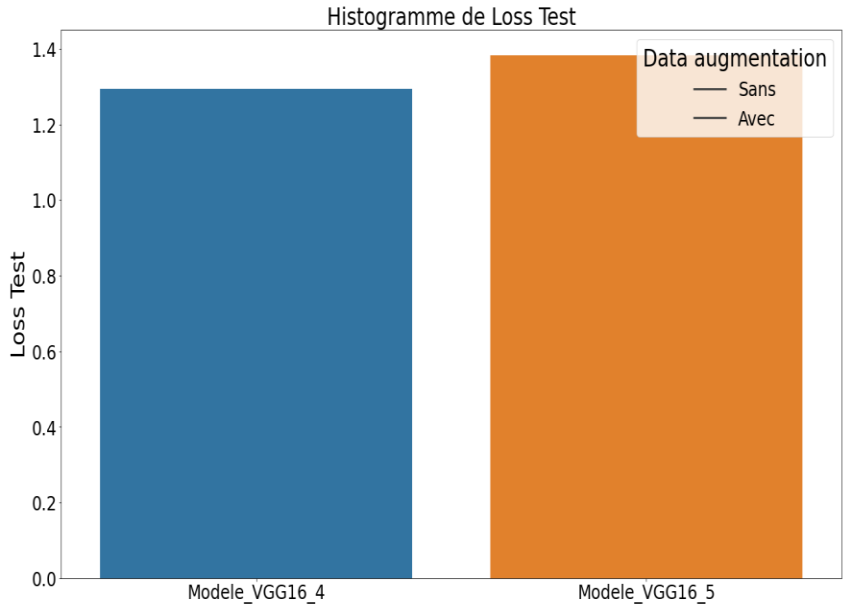
Meilleurs résultats obtenus en ajoutant une couche Dropout et en entraînant les 5 couches les plus hautes : Accuracy = 0.707287, Loss = 1.295118 et Temps de calcul = 13.240761

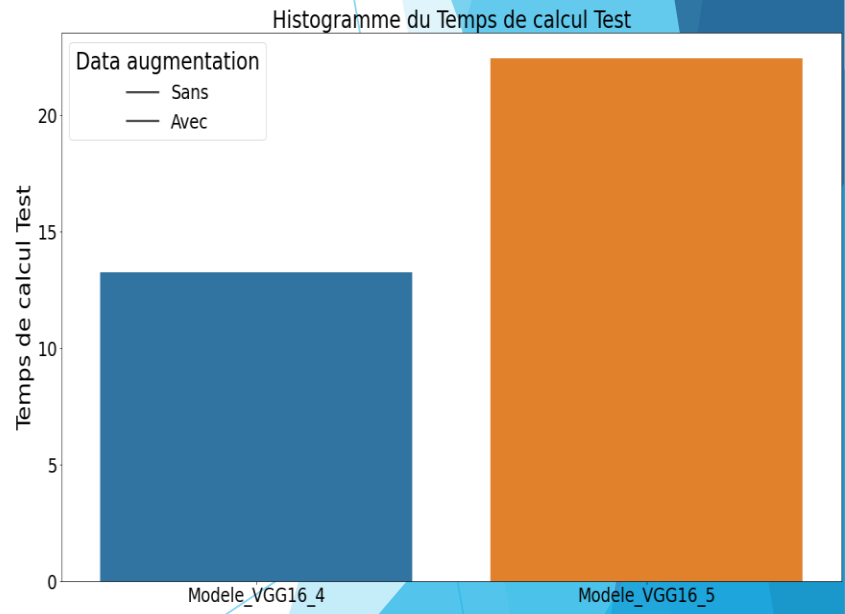
3. TRANSFER LEARNING

Comparaison des résultats :

- Data augmentation pour la stratégie fine-tuning partiel avec une couche Dropout







Pas de meilleurs résultats avec la Data augmentation : Accuracy = 0.707287, Loss = 1.295118 et Temps de calcul = 13.240761

3. TRANSFER LEARNING

Modèle sélectionné : Stratégie Fine-tuning partiel du modèle VGG16 en utilisant une couche Dropout

Recherche des hyperparamètres optimaux epochs et batch_size :

- Estimation de l'epochs optimal avec val_accuracy max = **50 epochs**
- Choix de batch_size sur [10, 20, 32, 40, 60, 80 et 96] = **96 batch_size**

Pour batch_size = 10 : Loss = 2.36, Accuracy = 0.69 et Temps de calcul = 23.41

Pour batch_size = 20 : Loss = 2.76, Accuracy = 0.70 et Temps de calcul = 24.14

Pour batch_size = 32 : Loss = 1.37, Accuracy = 0.70 et Temps de calcul = 14.25

Pour batch_size = 40 : Loss = 1.73, Accuracy = 0.69 et Temps de calcul = 25.80

Pour batch_size = 60 : Loss = 1.63, Accuracy = 0.70 et Temps de calcul = 22.62

Pour batch_size = 80 : Loss = 1.48, Accuracy = 0.72 et Temps de calcul = 14.44

Pour batch_size = 96 : Loss = 1.47, Accuracy = 0.73 et Temps de calcul = 15.93

CONCLUSION

Objectif : Réaliser un algorithme de détection de la race du chien à partir d'une image

Tester 2 approches de CNN sur un sous-jeu de données :

- Construire un réseau de neurones convolutionnels et optimiser son architecture et ses hyperparamètres
- Utiliser un modèle pré-entraîné VGG16 et tester la stratégie d'entraînement optimale

→ Meilleure approche = **Transfer Learning** (Accuracy de 0.73 sur le jeu de test vs proche de 0.2 sur le modèle construit)

Programme pour prédire la race la plus probable du chien présent sur l'image en utilisant le modèle sélectionné entraîné sur le jeu de données d'entraînement

MERCI DE VOTRE ATTENTION

QUESTIONS – REPONSES