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

Présenté par : NAMA NYAM Guy Anthony

Mentor : Julien Hendrick

22 Mars 2020

OPENCLASSROOMS

Sommaire

Contexte : Big data

- On assiste à une surabondance de données actuellement : images, vidéos, audio, texte, trace utilisateur, etc...
- Besoin évident d'accès, recherche ou classification de ses données regroupés sous la terminologie : ***reconnaissance*** en IA.
- Lorsque la donnée en entrée est une image à traiter, le champs de domaine est connu sous le nom de **vision par ordinateur**.

$$I : \begin{array}{ccc} \mathbb{R}^2 & \rightarrow & \mathbb{R}^3 \\ (x,y) & \mapsto & \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix} \end{array}$$

- Le problème soumit à cette thématique est de pouvoir détecter la race de chien sur une photo afin d'accélérer leur indexation.
- Cette tâche peut se formuler comme un problème de classification.

Contexte : classification

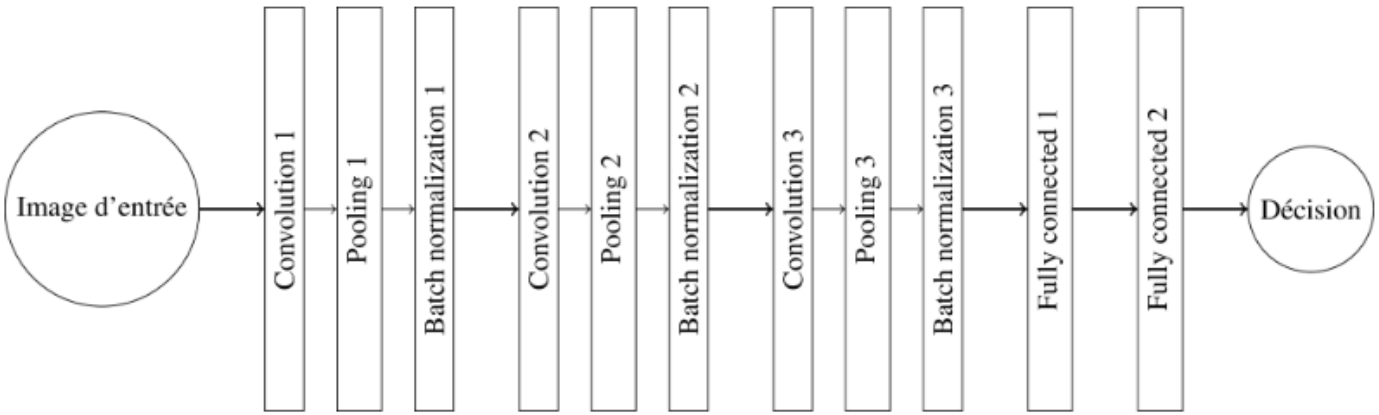
- La classification est très importante dans la reconnaissance et consiste : data(image chien) $\longrightarrow$ ensemble de classes prédéfinies(race de chien).
- La classification de chiens peut se faire à l'aide des algorithmes issus de machine learning, en particulier de l'apprentissage supervisé.
- Néanmoins, pour des performances comparables à celle de l'état de l'art, l'approche du ***Deep Learning*** est utilisée dans ce travail.
- Cas d'applications de reconnaissance visuelle : captioning(légende), localisation d'objets, réalité augmentée, imagerie médicale, conduite autonome, etc...

Objectif

Implémenter un algorithme capable de classer les chiens présents sur des images selon leur race.

Approche

- Le type d'apprentissage utilisé est supervisé
- L'apprentissage profond ou deep learning s'appuit sur des modèles appelés ***réseaux de neurones artificiels***(RNA).
- Il existe plusieurs types RNA(RNN, LSTM, GAN,..) ; le plus adapté au traitement de l'image est le RN ***convolutionnel*** (CNN).
- Un CNN classique comprend au moins 4 types de couches : la couche de ***convolution***, la couche de ***pooling***, la couche de correction ***ReLU*** et la couche ***fully-connected***



Un réseau profond complet

Architecture CNN

- Couche de convolution : composante clé et première couche CNN.

  (image, filtre) $\longrightarrow$ carte activation ou ***feature map***.

  Les filtres(e.g sobel) correspondent aux features que l'on souhaite retrouver dans les images(e.g bords, coins).

  La couche de convolution possède quatre hyper-paramètres :

  1. Le nombre de filtre
  2. La taille des filtres
  3. Le pas de glissement du filtre sur l'image.
  4. Le *zero-padding* pour le contour de l'image

- Couche d'activation : elle est pour la plupart couplée à la couche de convolution. La fonction d'activation non-linéaire utilisée par la suite : ReLU(Rectified Linear Unit).
  D'autres fonctions similaires : sigmoïde, TanH, les variantes à ReLU.

Architecture CNN

- Couche de pooling : généralement placée entre les couches de convolutions. Elle consiste à réduire les feature map tout en préservant les feature importantes(opérateur moyenne ou max).

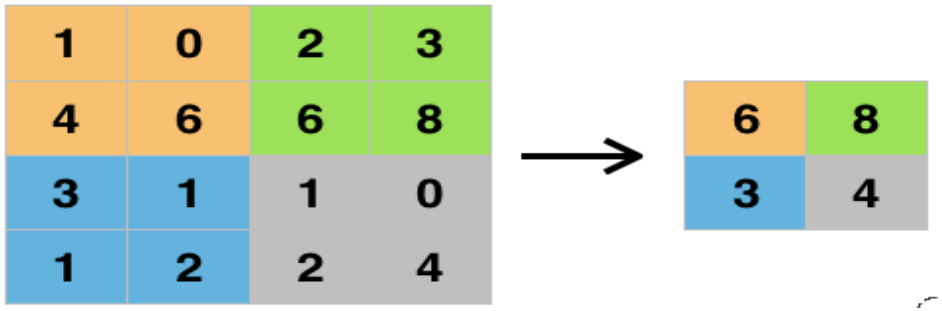


FIGURE – MaxPooling - source : wikimedia commons

La couche de pooling possède deux hyper-paramètres :

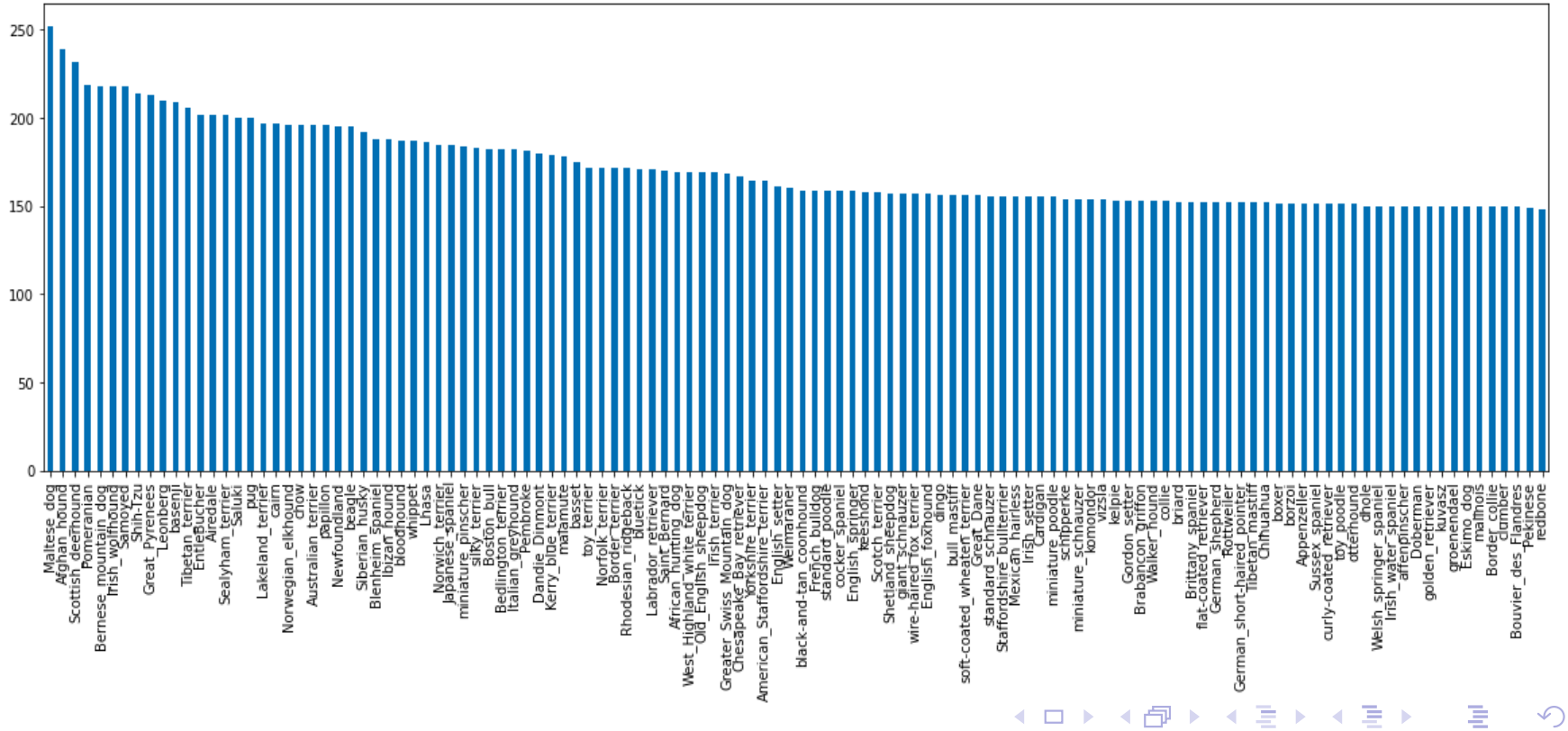
1. La taille de cellule
2. Le pas des cellules.

- Couche fully-connected : constitue la dernière couche du CNN. Elle permet de classifier l'image en entrée du CNN en renvoyant un vecteur de taille N, où N est le nombre de classes du problème.

Sommaire

Description

- Pour entraîner notre algorithme supervisé de classification de chiens par race, nous utilisons le jeu de données ***Stanford Dogs***.
- Jeu de données avec 20580 images pour 120 classes dont la figure ci-dessous donne la distribution.



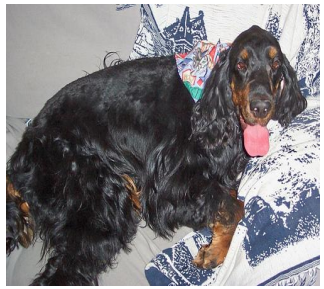
Visualisation échantillon Standford Dogs dataset

(a) n02088364_161 - race : beagle

(b) n02091134_145 - race : whippet

(c) n02116738_1815 - race : ?

(d) n02112137_517 - race : chow



(e) n02101006_22 - race : Gordon setter

Préparation de la donnée

- Formater le jeu de données dans la structure DataFrame.

```
data
```

	directory	filename	category
0	n02110185-Siberian_husky	n02110185_5973.jpg	Siberian_husky
1	n02110185-Siberian_husky	n02110185_2614.jpg	Siberian_husky
2	n02110185-Siberian_husky	n02110185_8327.jpg	Siberian_husky
3	n02110185-Siberian_husky	n02110185_4133.jpg	Siberian_husky
4	n02110185-Siberian_husky	n02110185_14650.jpg	Siberian_husky
...	...	...	...
20575	n02093647-Bedlington_terrier	n02093647_120.jpg	Bedlington_terrier
20576	n02093647-Bedlington_terrier	n02093647_2585.jpg	Bedlington_terrier
20577	n02093647-Bedlington_terrier	n02093647_2068.jpg	Bedlington_terrier
20578	n02093647-Bedlington_terrier	n02093647_3219.jpg	Bedlington_terrier
20579	n02093647-Bedlington_terrier	n02093647_2349.jpg	Bedlington_terrier

20580 rows × 3 columns

- split data 80% : training data

```
train_df = data.groupby('category').head(137)
train_df.shape[0]
```

```
16440
```

- split data 20% : test data

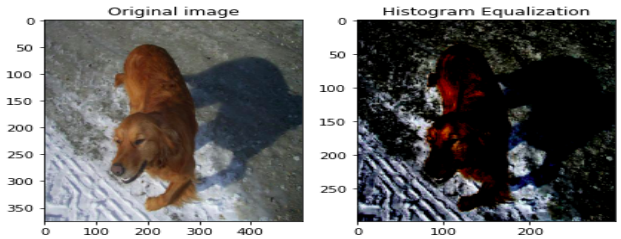
```
test_df = data.loc[data.index.difference(train_df.index)]
test_df.shape[0]
```

```
4140
```

Image preprocessing

- Deux techniques de prétraitements ont été implémentées :
  1. L'égalisation de l'histogramme grâce à la bibliothèque ***OpenCV***

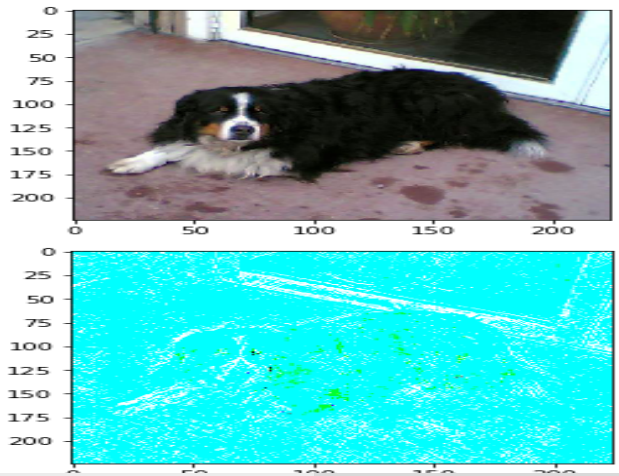
```
def equalHist(img, adaptive=True):
    img = img.astype(np.uint8)
    ycrcb=cv2.cvtColor(img,cv2.COLOR_RGB2YCR_CB)
    channels=cv2.split(ycrcb)
    if adaptive:
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        channels[0] = clahe.apply(channels[0])
    else:
        channels[0] = cv2.equalizeHist(channels[0])
    ycrcb = cv2.merge(channels)
    img = cv2.cvtColor(ycrcb,cv2.COLOR_YCR_CB2RGB)
    return img
```



  2. Le blanchiment(whitening). Ce dernier ne sera pas utilisé lié au contrainte de ressources disponible.

```
def zca_whitening(x):
    x = x - x.mean(axis=0)
    sigma = x.dot(x.T) / float(x.shape[0])
    U, S, Vh = linalg.svd(sigma)
    xRot = U.T.dot(x)
    epsilon = 1e-5
    xPCAWhite = np.diag(1.0 / np.sqrt(S + epsilon)).dot(U.T).dot(x)
    xZCAWhite = U.dot(xPCAWhite)
    return xZCAWhite
```

```
def whitening(img):
    channels = cv2.split(img)
    channels[0] = zca_whitening(channels[0])
    #channels[1] = zca_whitening(channels[1])
    #channels[2] = zca_whitening(channels[2])
    img = cv2.merge(channels)
    return img
```



Utilisation : callbacks, losses, optimizers

- Un callback : ensemble de fonction à appliquer à des étapes données de l'entraînement du modèle.
- Nous avons utilisé(pertinents pour ce projet) :
  1. EarlyStopping - arrêter l'entraînement lorsque la métrique surveillée n'évolue plus.
  2. ReduceLROnPlateau - réduire le learning rate lorsqu'une métrique n'évolue plus.
  3. TensorBoard - visualisation des métriques de base.
- La fonction de perte(ou objective) utilisée pour l'estimation des paramètres de nos réseaux de neurones est ***categorical_crossentropy*** (classification multi-classes)
- Nous avons défini notre propre optimiseur de mise à jour des paramètres lors de la phase de rétro-propagation du gradient : tf.keras.optimizers.RMSprop(learning=0.01).

Sommaire

InceptionResNetV2

- Le ***Transfer Learning*** permet de faire du deep learning en accélérant l'apprentissage du réseau.
- Cette technique nécessite un réseau déjà entraîné, de préférence sur un problème proche du nôtre(classification de chiens).
- ***InceptionResNetV2*** est un modèle de classification d'images avec des poids entrainé sur ***ImageNet***. Modèle avec 55,873,736 paramètres et top-1 Accuracy de 0.803

```
inceptionResNetV2_model = tf.keras.applications.InceptionResNetV2(weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219062272/219055592 [==============================] - 2s 0us/step
```

```
x = inceptionResNetV2_model.output
```

```
x = GlobalAveragePooling2D()(x)
```

```
predictions = Dense(120, activation='softmax', name='predictions')(x)
```

```
my_inceptionResNetV2 = tf.keras.models.Model(inputs=inceptionResNetV2_model.input, outputs=predictions)
```

```
# On entraîne seulement le nouveau classifieur
for layer in inceptionResNetV2_model.layers:
    layer.trainable = False
```

Fit model

- Avant d'entraîner le modèle, on commence par le compiler.

```
my_inceptionResNetV2.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

- Entraîner le modèle sur le jeu de données 16440

```
history = my_inceptionResNetV2.fit(train_generator,
                                   epochs=30,
                                   steps_per_epoch=SIZE_TRAIN//32,
                                   validation_data=test_generator,
                                   validation_steps=SIZE_TEST//32,
                                   callbacks=[learning_rate_reduction, earlystop, tensorboard_callback]
                                   )
```

```
Epoch 1/30
513/513 [==============================] - 19400s 38s/step - loss: 1.1864 - accuracy: 0.8090 - val_loss: 0.8889 - val_accuracy: 0.8740 - lr: 0.0100
Epoch 2/30
513/513 [==============================] - 183s 358ms/step - loss: 0.9112 - accuracy: 0.8722 - val_loss: 0.6911 - val_accuracy: 0.9051 - lr: 0.0100
Epoch 3/30
513/513 [==============================] - 183s 357ms/step - loss: 0.8135 - accuracy: 0.8847 - val_loss: 0.9748 - val_accuracy: 0.8830 - lr: 0.0100
```

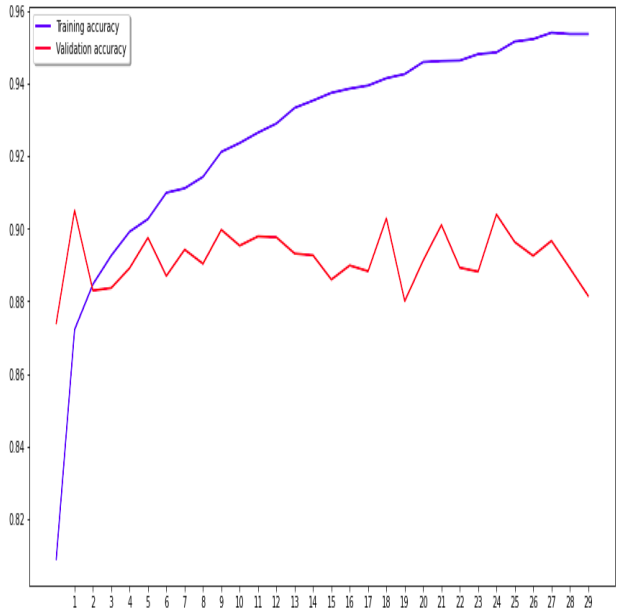
- Sauvegarder le modèle et les indices des classes.

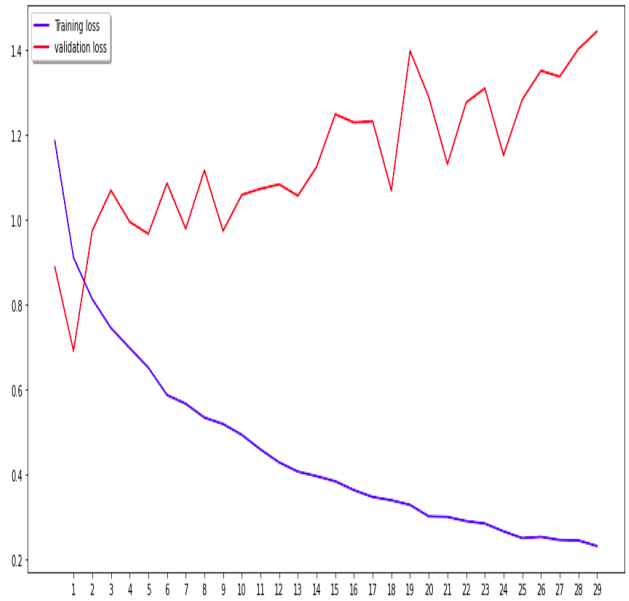
```
# Save model
tf.keras.models.save_model(my_inceptionResNetV2, dir_model + "my_inceptionResNetV2.h5")
```

```
# Save class indices
save_obj(train_generator.class_indices, 'class_indices')
```

Evaluate model

- 'Accuracy' et 'loss' pendant la phase d'entraînement.





- Évaluation exactitude du modèle sur le jeu de 4 140 échantillons.

```
[52] loss_and_metrics = my_inceptionResNetV2.evaluate(test_generator, batch_size=BATCH_SIZE_TEST, steps=SIZE_TEST//32)

129/129 [==============================] - 45s 350ms/step - loss: 1.4479 - accuracy: 0.8814

[53] loss_and_metrics

[1.4479219913482666, 0.8813744783401489]
```

Misclassifications

- Extraction des images mal classées

```
collections = np.array([img_to_array(load_img(dir_test + "/" + row['filename'], target_size=(299, 299))) for i, row in test_df.iterrows()])
```

```
collections = np.array([preprocessing_image(collection) for collection in collections])
```

```
predict = my_inceptionResNetV2.predict(collections)
```

```
test_df['predicted'] = np.argmax(predict, axis=1)  # position of max probability
```

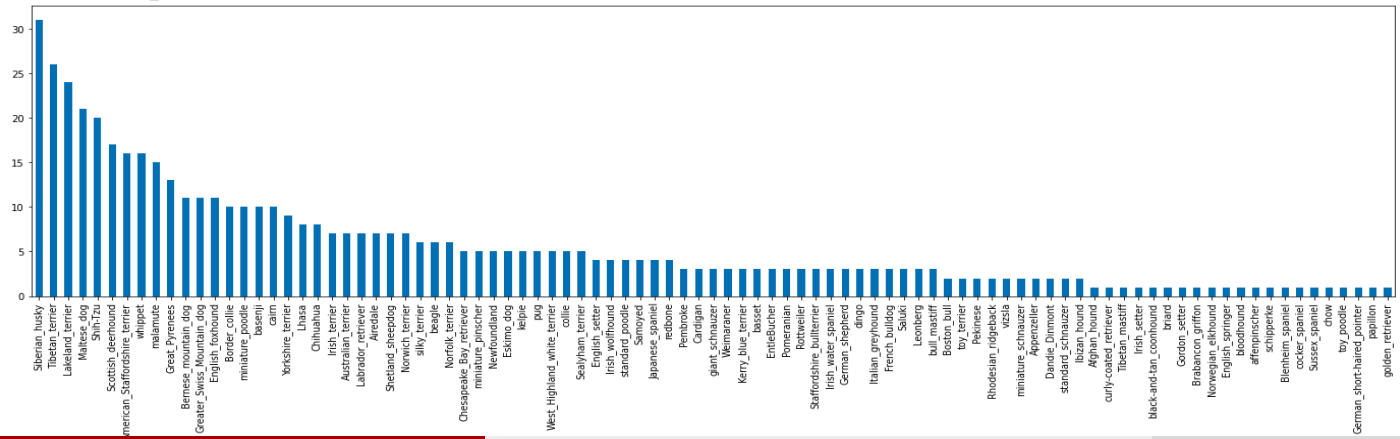
- Mal classés

```
# Misclassifications
error_df = test_df[test_df['category'] != test_df['predicted']]
error_df.shape[0]
```

```
521
```

```
plt.figure(figsize=(24, 6))
error_df['category'].value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2967076128>
```

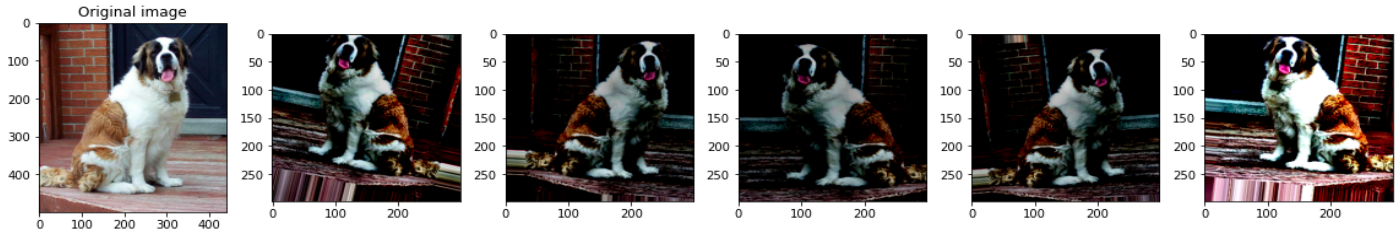


Data augmentation with ImageDataGenerator

- Preprocessing + Data augmentation avec ***ImageDataGenerator***

```
# Parameters estimated from misclassifications that detect them(empirical)
train_datagen_aug = ImageDataGenerator(rotation_range=30,
                                       width_shift_range=0.1,
                                       height_shift_range=0.1,
                                       shear_range=0.01,
                                       zoom_range=0.1,
                                       horizontal_flip=True,
                                       brightness_range=[0.5, 1.5],
                                       preprocessing_function=preprocessing_image
                                       )
```

- Effets de la Data augmentation



- Amélioration sensible : 0.88 à 0.91

```
loss_and_metrics_aug = my_inceptionResNetV2.evaluate(test_generator_aug, batch_size=BATCH_SIZE_TEST, steps=SIZE_TEST//32)
```

```
129/129 [==============================] - 46s 360ms/step - loss: 1.0751 - accuracy: 0.9057
```

```
loss_and_metrics_aug
```

```
[1.0750921964645386, 0.9056580662727356]
```

Sommaire

Mise au point

- Égalisation d'histogramme unique opération de preprocessing
- Usage de la data augmentation

```
# use of the pre-processing function of the model inceptionResNetV2
vgg16_train_datagen = ImageDataGenerator(rotation_range=20,
                                         width_shift_range=0.1,
                                         height_shift_range=0.1,
                                         shear_range=0.01,
                                         zoom_range=0.1,
                                         horizontal_flip=True,
                                         brightness_range=[0.5, 1.5],
                                         preprocessing_function=preprocessing_image_from_scratch
                                         )
```

```
vgg16_test_datagen = ImageDataGenerator(rescale=1./255)
```

```
vgg16_train_generator = vgg16_train_datagen.flow_from_dataframe(dataframe=train_df,
                                                                directory=dir_train,
                                                                x_col='filename',
                                                                y_col='category',
                                                                target_size=(224, 224),
                                                                class_mode='categorical',
                                                                batch_size=BATCH_SIZE_TRAIN
                                                                )
```

```
Found 16440 validated image filenames belonging to 120 classes.
```

```
Xception_train_datagen = ImageDataGenerator(rotation_range=20,
                                            width_shift_range=0.1,
                                            height_shift_range=0.1,
                                            shear_range=0.01,
                                            zoom_range=0.1,
                                            horizontal_flip=True,
                                            brightness_range=[0.5, 1.5],
                                            preprocessing_function=preprocessing_image_from_scratch
                                            )
```

```
Xception_test_datagen = ImageDataGenerator(rescale=1./255)
```

```
Xception_train_generator = Xception_train_datagen.flow_from_dataframe(dataframe=train_df,
                                                                      directory=dir_train,
                                                                      x_col='filename',
                                                                      y_col='category',
                                                                      target_size=(299, 299),
                                                                      class_mode='categorical',
                                                                      batch_size=BATCH_SIZE_TRAIN
                                                                      )
```

- Comparaison sur une compétition sur ***kaggle*** pour le même jeu de données des modèles 'from scratch'.
- Métrique évaluée : ***'loss'***. Benchmark : 4.78749

my_vgg16 : architecture

- Architecture CNN

```
my_VGG16 = tf.keras.models.Sequential()
```

```
# Block 1
my_VGG16.add(Conv2D(64,(3, 3), input_shape=(224, 224, 3), padding='same', activation='relu'))

my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# Block2
my_VGG16.add(Conv2D(128, (3, 3), padding='same', activation='relu'))

my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# Block3
my_VGG16.add(Conv2D(256, (3, 3), padding='same', activation='relu'))

my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# Block4
my_VGG16.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

my_VGG16.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# Block5
my_VGG16.add(Conv2D(512, (3, 3), padding='same', activation='relu'))

my_VGG16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# Fully-connected classifier
my_VGG16.add(Flatten())
my_VGG16.add(Dense(1024, activation='relu'))
my_VGG16.add(Dense(2048, activation='relu'))
my_VGG16.add(Dense(120, activation='softmax'))
```

- Nombre de paramètres : 34,306,808

my_vgg16 : fit and evaluate

- Fit model

```
my_VGG16.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
my_VGG16.fit(vgg16_train_generator, epochs=30, steps_per_epoch=SIZE_TRAIN//32, callbacks=[learning_rate_reduction, earlystop])
```

```
Epoch 1/30
513/513 [==============================] - 7407s 14s/step - loss: 4.8548 - accuracy: 0.0105 - lr: 1.0000e-04
Epoch 2/30
513/513 [==============================] - 334s 652ms/step - loss: 4.4804 - accuracy: 0.0369 - lr: 1.0000e-04
Epoch 3/30
513/513 [==============================] - 321s 625ms/step - loss: 4.1788 - accuracy: 0.0680 - lr: 1.0000e-04
Epoch 4/30
513/513 [==============================] - 320s 624ms/step - loss: 3.9525 - accuracy: 0.1010 - lr: 1.0000e-04
Epoch 5/30
513/513 [==============================] - 319s 623ms/step - loss: 3.7281 - accuracy: 0.1357 - lr: 1.0000e-04
```

- Évaluation

```
loss_and_metrics_v = my_VGG16.evaluate(vgg16_test_generator, batch_size=BATCH_SIZE_TEST, steps=SIZE_TEST//32)
```

```
129/129 [==============================] - 1967s 15s/step - loss: 6.9645 - accuracy: 0.0058
```

```
loss_and_metrics_v
```

```
[6.964511394500732, 0.005828071851283312]
```

my_Xception : architecture

- Architecture CNN

```
# Entry flow
main_input = tf.keras.Input(shape=(299, 299, 3), name='main_input')

x = Conv2D(32, (3, 3), strides=(2,2), activation='relu', padding="same")(main_input)
x = Conv2D(64, (3, 3), activation='relu', padding="same")(x)

tower_1 = Conv2D(1, (1, 1), strides=(2,2))(x)

x = SeparableConv2D(128, (3, 3), padding='same')(x)

x = Activation('relu')(x)
x = SeparableConv2D(128, (3, 3), padding='same')(x)

x = MaxPooling2D(pool_size=(3, 3), strides=(2,2), padding="same")(x)

x = concatenate([x, tower_1], axis=-1)

tower_2 = Conv2D(1, (1, 1), strides=(2,2))(tower_1)

x = Activation('relu')(x)
x = SeparableConv2D(256, (3, 3), padding="same")(x)

x = Activation('relu')(x)
x = SeparableConv2D(256, (3, 3), padding="same")(x)

x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="same")(x)

x = concatenate([x, tower_2], axis=-1)

tower_3 = Conv2D(1, (1, 1), strides=(2,2))(tower_2)

x = Activation('relu')(x)
x = SeparableConv2D(768, (3, 3), padding="same")(x)

x = Activation('relu')(x)
x = SeparableConv2D(768, (3, 3), padding="same")(x)

x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="same")(x)

x = concatenate([x, tower_3], axis=-1)
```

```
# Middle flow : repeat 4 fois
for i in range(4):
    y = Activation('relu')(x)
    y = SeparableConv2D(728, (3, 3), padding="same")(y)

    y = Activation('relu')(y)
    y = SeparableConv2D(728, (3, 3), padding="same")(y)

    y = Activation('relu')(y)
    y = SeparableConv2D(728, (3, 3), padding="same")(y)

    x = concatenate([x, y])
```

```
# Exit flow
tower_4 = Conv2D(1, (1, 1), strides=(2,2))(x)

x = Activation('relu')(x)
x = SeparableConv2D(728, (3, 3), padding="same")(x)

x = Activation('relu')(x)
x = SeparableConv2D(1024, (3, 3), padding="same")(x)

x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding="same")(x)

x = concatenate([x, tower_4])

x = SeparableConv2D(1536, (3, 3), activation='relu', padding="same")(x)
x = SeparableConv2D(2048, (3, 3), activation='relu', padding="same")(x)

x = GlobalAveragePooling2D()(x)

x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(120, activation='softmax')(x)
```

- Nombre de paramètres : 21,172,063

my_Xception : fit and evaluate

- Fit model

```
my_Xception = tf.keras.Model(inputs=main_input, outputs=x)
```

```
my_Xception.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
my_Xception.fit(Xception_train_generator, epochs=30, steps_per_epoch=SIZE_TRAIN//32, callbacks=[learning_rate_reduction, earlystop])
```

```
Epoch 1/30
513/513 [==============================] - 508s 989ms/step - loss: 4.7887 - accuracy: 0.0074 - lr: 0.0010
Epoch 2/30
513/513 [==============================] - 508s 991ms/step - loss: 4.7885 - accuracy: 0.0071 - lr: 0.0010
Epoch 3/30
513/513 [==============================] - ETA: 0s - loss: 4.7886 - accuracy: 0.0064
Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
513/513 [==============================] - 509s 991ms/step - loss: 4.7886 - accuracy: 0.0064 - lr: 0.0010
Epoch 4/30
513/513 [==============================] - 507s 988ms/step - loss: 4.7880 - accuracy: 0.0069 - lr: 1.0000e-04
Epoch 5/30
```

- Évaluation

```
loss_and_metrics_x = my_Xception.evaluate(Xception_test_generator, batch_size=BATCH_SIZE_TEST, steps=SIZE_TEST//32)
```

```
129/129 [==============================] - 54s 415ms/step - loss: 4.7876 - accuracy: 0.0075
```

```
loss_and_metrics_x
```

```
[4.787564277648926, 0.007527926005423069]
```

Sommaire

Conclusion

- Nombreux sont les concepts étudiés et maîtrisés au cours du projet ;
- Traitement d'une image numérique du point de vue ordinateur
- Nous avons découvert les réseaux de neurones, en particulier les CNN.
- Comprendre l'approche de 'deep learning' pour la classification d'images et bien d'autres tâches.
- L'importance de la phase du pré-traitement et les outils utilisables(OpenCV, ImageDataGenerator, etc.)
- Réaliser la data augmentation(mirroring, brightness, etc.) pour améliorer le modèle.

Conclusion

- Concevoir nos propres architectures CNN(from scratch)
- Ré-utiliser les modèles pré-entraînés(Transfer Learning) pour résoudre des problèmes spécifiques.
- Fine-tuning les hyper-paramètres pour améliorer les performances des modèles d'apprentissage.
- Organiser et préparer la data depuis les serveurs Google(Drive)
- Approfondir l'usage de Google Colab et l'utilisation de leur GPU.

Perspectives

- Faire du ***cropping*** pour améliorer le modèle issu du Transfer Learning
- Re-tuning les hyper-paramètres des différents modèles.
- Rendre le modèle plus robuste au modifications
- Intégrer la hiérarchie dans la détection des features(capsule network)
- Tester l'effet du blanchiment sur les performances des modèles
- Utiliser le modèle retenu pour des vidéos cracking.
- Explorer d'autres bibliothèques pour la vision par ordinateur tel que PyTorch