

Classification cat et dog analysis

TP IA CNN

SEDJAL ABDELGHANI

UPEC

2022-2023

Table des matières

Présentation du dataset :	2
Importation des bibliothèques :	2
Prétraitement de données et création des classes :	2
Création du model (RELU,Softmax) lr= 0,01:	3

Présentation du dataset :

Le data set étudié dans ce tp est un ensemble d'image de chat et de chien ce data set a été créée pour pouvoir construire un modèle de classification de nouvelle image avec une précision élevé , ce dataset contient 25000 image répartie équitablement entre chiens et chats

Importation des bibliothèques :

```
import numpy as np
import pandas as pd
from tensorflow.keras.utils import load_img
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import tensorflow as tf
import random
import os
print(os.listdir("C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats"))
```

```
['dogscats', 'sample', 'test1', 'train', 'valid']
```

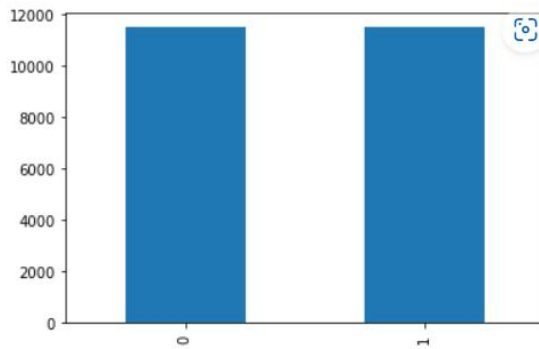
Prétraitement de données et création des classes :

```
filenames = os.listdir("C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats/train")
categories = []
for filename in filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append(1)
    else:
        categories.append(0)

df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})
```

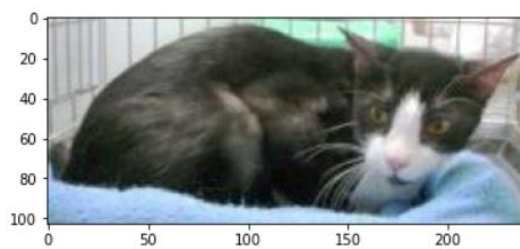
A travers ce petit script nous allons pouvoir répartir les données du dataset et attribuer a chaque image une catégorie tel qu'il est montré sur la figure ci-dessus.

Nous pouvons voir que les données sont réparties équitablement dans les données a utiliser dans le train et un exemple d'une image :



Entrée [7]: `sample = random.choice(filenamees)`
`image = load_img("C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats/train/"+sample)`
`plt.imshow(image)`

Out[7]: `<matplotlib.image.AxesImage at 0x29e000acc70>`



Création des modèles :

Modèle Avec 3 couche RELU et LR=0,001

Le premier modèle qu'on va le créer est le plus simple avec un Learning rate de 0,01 et de 3 couches avec la fonction activation Relu et la sortie avec la fonction softmax comme le nous pouvons visualiser :

```
Entrée [11]: from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

model.summary()
```

Notons que les 3 couches servent à extraire les features avec une entrée de taille 128*128

```

earlystop = EarlyStopping(patience=10)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                             patience=2,
                                             verbose=1,
                                             factor=0.5,
                                             min_lr=0.00001)

```

Afin de pouvoir éviter le over-fitting nous faisons stoppé l'entraînement du modèle a 10 epoques, puis nous réduisons le LR en fonction de la val_accuracy,

Après avoir configurer l'ensemble des paramètres du fitting, nous concentrerons sur la sélection des données pour la validation et le test,

```

callbacks = [earlystop, learning_rate_reduction]

```

```

df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})

```

```

train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)

```

Le remplacement du 0 et du 1 par chat et chien et due a l'utilisation de imagerator qui utilise les chaines de caractères.

```

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)

```

```

train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats/train/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)

```

Found 18400 validated image filenames belonging to 2 classes.

On répète la même chose pour la validation générateur ces deux variables seront introduit comme paramètres dans le model fitting

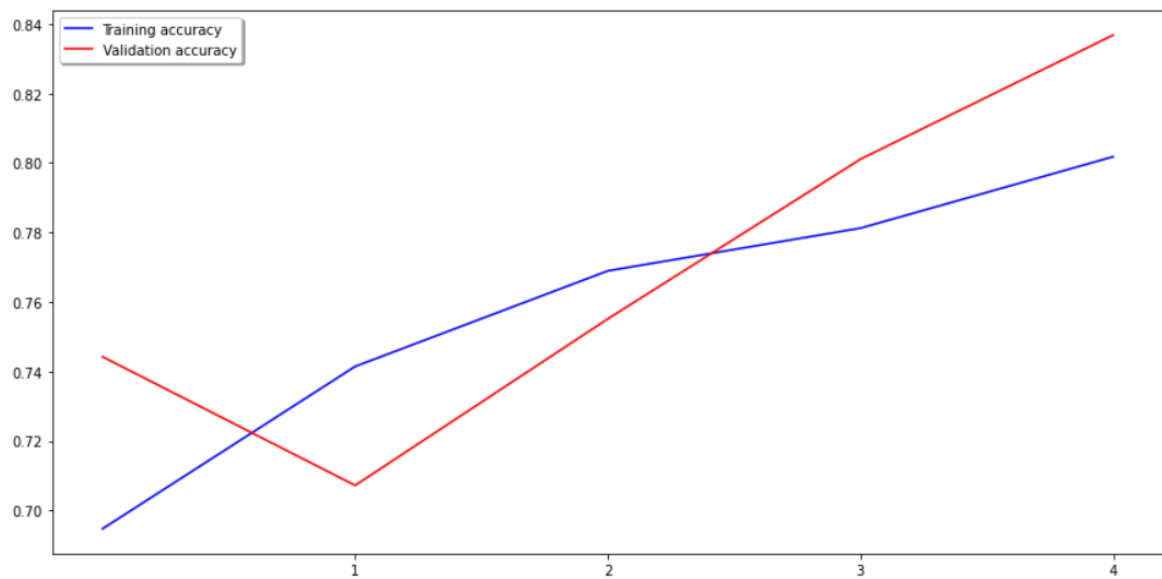
Nous trainons le modèle pour 5 epoques et a chaque fois on voit les métriques d'évaluation (accuracy et loss)

```

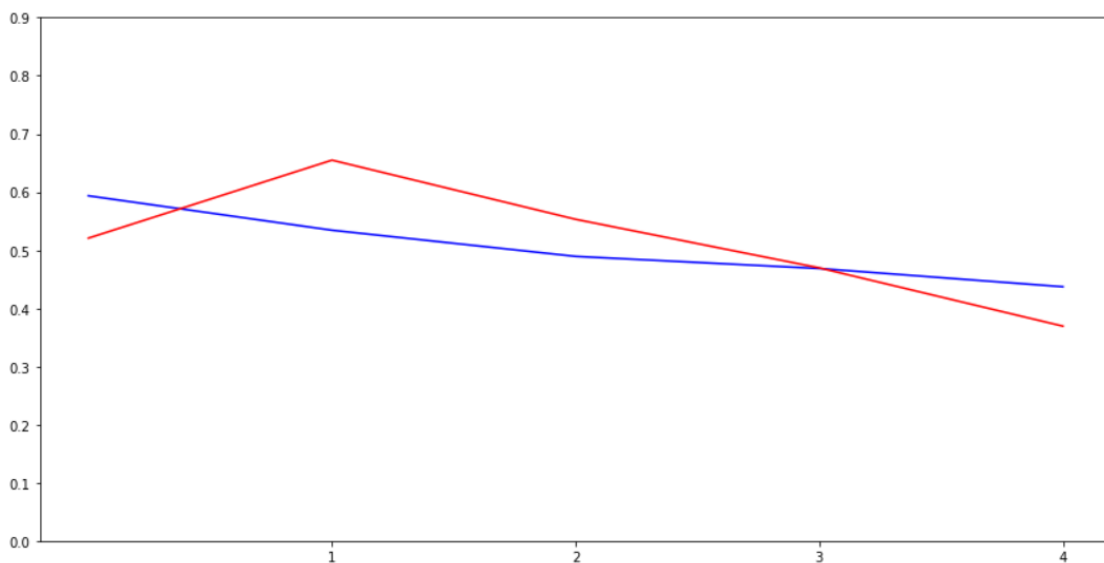
Epoch 1/5
C:\Users\ASUS ZenBook\AppData\Local\Temp\ipykernel_13376\1352027716.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(
1226/1226 [=====] - 1037s 846ms/step - loss: 0.5938 - accuracy: 0.6947 - val_loss: 0.5209 - val_accuracy: 0.7442 - lr: 0.0010
Epoch 2/5
1226/1226 [=====] - 1356s 1s/step - loss: 0.5346 - accuracy: 0.7414 - val_loss: 0.6550 - val_accuracy: 0.7072 - lr: 0.0010
Epoch 3/5
1226/1226 [=====] - 1180s 962ms/step - loss: 0.4896 - accuracy: 0.7689 - val_loss: 0.5533 - val_accuracy: 0.7551 - lr: 0.0010
Epoch 4/5
1226/1226 [=====] - 1075s 877ms/step - loss: 0.4688 - accuracy: 0.7812 - val_loss: 0.4699 - val_accuracy: 0.8011 - lr: 0.0010
Epoch 5/5
1226/1226 [=====] - 1080s 881ms/step - loss: 0.4374 - accuracy: 0.8018 - val_loss: 0.3697 - val_accuracy: 0.8368 - lr: 0.0010

```

Après avoir effectué le fitting nous traçons les graphes accuracy et loss pour le modèle et le résultat est montré dans l'image ci-dessous :



A travers le graphe accuracy nous voyons qu'il y'a pas d'overfitting ou bien d'underfitting dans le modèle ainsi sa val_accuracy est autour de 0,81 dans 5 époques ce qui est bien,



Le graphe du loss,

Ensuite nous allons préparer les données de test ainsi le générateur ainsi lancer la prédiction du modèle :

```
test_filenames = os.listdir("C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats/test1")
test_df = pd.DataFrame({
    'filename': test_filenames
})
nb_samples = test_df.shape[0]
```

```
test_gen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
test_generator = test_gen.flow_from_dataframe(
    test_df,
    "C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats/test1/",
    x_col='filename',
    y_col=None,
    class_mode=None,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    shuffle=False
)
```

Found 12500 validated image filenames.

```
predict = model.predict_generator(test_generator, steps=np.ceil(nb_samples/batch_size))
```

C:\Users\ASUS ZenBook\AppData\Local\Temp\ipykernel_13376\3090721588.py:1: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, which supports generators.

```
predict = model.predict_generator(test_generator, steps=np.ceil(nb_samples/batch_size))
```

```
test_df['category'] = np.argmax(predict, axis=-1)
```

Après avoir effectué la prédiction elle sera accompagnée de la probabilité de chaque catégorie. Nous choisirons donc la catégorie qui a la probabilité la plus élevée avec numpy average max,

```
label_map = dict((v,k) for k,v in train_generator.class_indices.items())
test_df['category'] = test_df['category'].replace(label_map)
test_df['category'] = test_df['category'].replace({ 'dog': 1, 'cat': 0 })
```

```
sample_test = test_df.head(18)
sample_test.head()
plt.figure(figsize=(12, 24))
for index, row in sample_test.iterrows():
    filename = row['filename']
    category = row['category']
    img = load_img("C:/Users/ASUS ZenBook/Pictures/UPEC_IA/archive/dogscats/test1/"+filename, target_size=IMAGE_SIZE)
    plt.subplot(6, 3, index+1)
    plt.imshow(img)
    plt.xlabel(filename + '(' + "{}".format(category) + ')')
plt.tight_layout()
plt.show()
```

nous reconvertirons la catégorie prédite en classes de notre générateur en utilisant `train_generator.class_indices`. Il s'agit des classes que le générateur d'images mappe lors de la conversion des données en vision par ordinateur, puis nous visualisons le résultat prédit :



Modèle avec 3 couches SELU et SIGMOID 8 époques et LR=0,01:

Pour l'ensemble de données nous avons travaillé avec les mm DF que dans le modèle 1 le code de création du modèle est le suivant :

CHANGER QUELQUES PARAMETRES (FONCTION ACTIVATION, LR = 0,01) et voir Les accuracies et Les loss

```
from keras.models import Sequential
import keras
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model2 = Sequential()

model2.add(Conv2D(32, (3, 3), activation='selu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

model2.add(Conv2D(64, (3, 3), activation='selu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

model2.add(Conv2D(128, (3, 3), activation='selu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))

model2.add(Flatten())
model2.add(Dense(512, activation='selu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.5))
model2.add(Dense(2, activation='sigmoid')) # 2 because we have cat and dog classes
opt = keras.optimizers.Adam(learning_rate=0.01) #learning rate a 0,01

model2.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

model2.summary()
```

Puis comme le premier modèle nous le passons par le fitting


```

epochs=3 if FAST_RUN else 8
history2 = model2.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size
)

```

L'output :

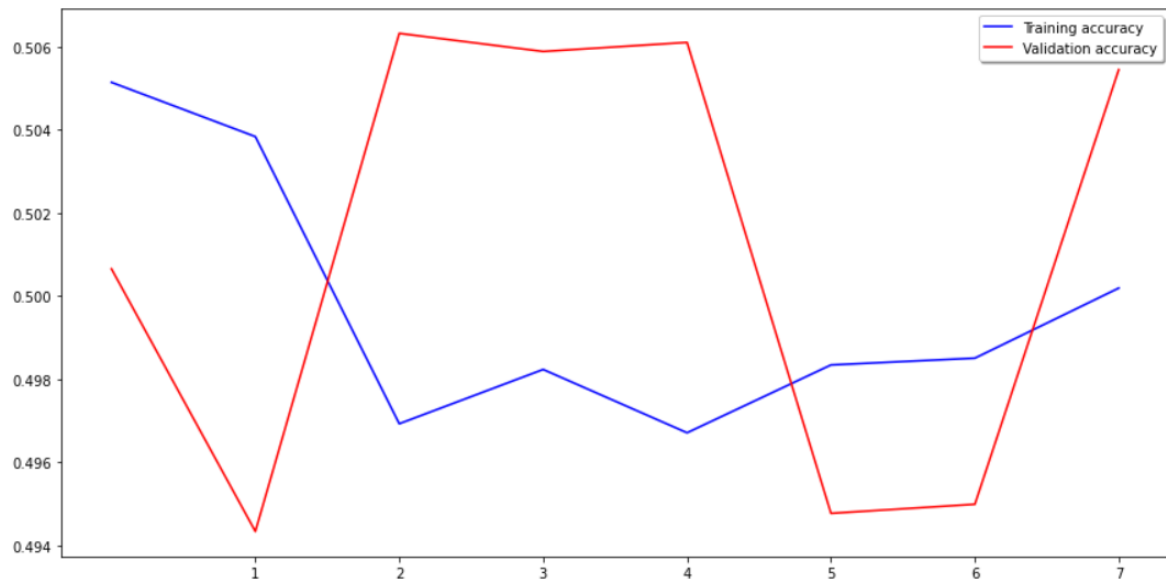
```

Epoch 1/8
C:\Users\ASUS ZenBook\AppData\Local\Temp\ipykernel_13376\2016724532.py:2: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history2 = model2.fit_generator(
1226/1226 [=====] - 1445s 1s/step - loss: 8.0913 - accuracy: 0.5051 - val_loss: 0.7214 - val_accuracy: 0.5007
Epoch 2/8
1226/1226 [=====] - 1753s 1s/step - loss: 0.9891 - accuracy: 0.5038 - val_loss: 0.7060 - val_accuracy: 0.4943
Epoch 3/8
1226/1226 [=====] - 1459s 1s/step - loss: 1.1654 - accuracy: 0.4969 - val_loss: 1.5661 - val_accuracy: 0.5063
Epoch 4/8
1226/1226 [=====] - 1253s 1s/step - loss: 1.4341 - accuracy: 0.4982 - val_loss: 0.8163 - val_accuracy: 0.5059
Epoch 5/8
1226/1226 [=====] - 1310s 1s/step - loss: 1.6054 - accuracy: 0.4967 - val_loss: 1.0462 - val_accuracy: 0.5061
Epoch 6/8
1226/1226 [=====] - 1365s 1s/step - loss: 1.5739 - accuracy: 0.4983 - val_loss: 0.9731 - val_accuracy: 0.4948
Epoch 7/8
1226/1226 [=====] - 1237s 1s/step - loss: 1.6821 - accuracy: 0.4985 - val_loss: 0.7413 - val_accuracy: 0.4950
Epoch 8/8
1226/1226 [=====] - 1391s 1s/step - loss: 1.6484 - accuracy: 0.5002 - val_loss: 1.5423 - val_accuracy: 0.5054

model.save_weights("model_SELU_001LR.h5")

```

Le graphe tracé sera le suivant :



On remarque qu'il y'a de l'overfitting

Modèle Avec 4 couches fonction activation Relu et LR=0,0001 :

Même chose nous travaillerons sur les même données, pour le modèle :

```

from keras.models import Sequential
import keras

from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

model3 = Sequential()

model3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS)))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(128, (3, 3), activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Flatten())
model3.add(Dense(512, activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
model3.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes

```

Après avoir conçu le modèle nous trainons notre data sur ce modèle :

```

epochs=3 if FAST_RUN else 8
history3 = model3.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size
)

```

Après avoir passé presque toutes les époques :

```

Epoch 1/8
1226/1226 [=====] - 720s 584ms/step - loss: 0.9995 - accuracy: 0.5871 - val_loss: 1.5464 - val_accu-
racy: 0.6150
Epoch 2/8
1226/1226 [=====] - 743s 606ms/step - loss: 0.7832 - accuracy: 0.6269 - val_loss: 2.0369 - val_accu-
racy: 0.5786
Epoch 3/8
1226/1226 [=====] - 762s 622ms/step - loss: 0.6763 - accuracy: 0.6562 - val_loss: 1.8202 - val_accu-
racy: 0.6078
Epoch 4/8
1226/1226 [=====] - 781s 637ms/step - loss: 0.6198 - accuracy: 0.6776 - val_loss: 1.5119 - val_accu-
racy: 0.6312
Epoch 5/8
1226/1226 [=====] - 755s 616ms/step - loss: 0.5688 - accuracy: 0.7098 - val_loss: 1.2929 - val_accu-
racy: 0.6712
Epoch 6/8
1226/1226 [=====] - 816s 665ms/step - loss: 0.5442 - accuracy: 0.7270 - val_loss: 1.8203 - val_accu-
racy: 0.6479
Epoch 7/8
473/1226 [=====>.....] - ETA: 7:47 - loss: 0.5265 - accuracy: 0.7436

```

L'accuracy est aux alentours de 75%

Synthèse :

A travers l'étude d'un problème de classification (chat et chien) nous avons vu l'impact des paramètres en entrée du cnn (nb couches , Learning rate, fonction activation, nb epoques) sur le comportement du modèle ciblé.