Deep Learning (TensorFlow, Keras) with ResNet50: Image Binary Classifier (Part 1)

In this project, a model is trained to perform binary classifiaction for cats and dogs pictures. The pretrained model ResNet50 is used. This document is the first part of the whole training process.

The dataset can be found in:

https://www.kaggle.com/datasets/karakaggle/kaggle-cat-vs-dog-dataset

Iteration 1: Model creation and training (learning_rate=1e-4) without data augmentation (no fine-tuning yet)

```
# (height, width, channels)
input_shape = (224, 224, 3)
batch_size = 8
learning_rate = 1e-4
neurons = 128
path_dataset = '../dataset_cat_dogs'
folder_cat = 'Cat'
folder_dog = 'Dog'
folder_models = '../models'

# Path in Google Colab
# path_dataset = '/content/drive/MyDrive/Colab
Notebooks/dataset_cat_dogs'

# Mount Google Drive if using Google Colab
# from google.colab import drive
# drive.mount('/content/drive/')

import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint

# Find how many cats and dogs images exist
cat_imgs = os.listdir(os.path.join(path_dataset,folder_cat))
dog_imgs = os.listdir(os.path.join(path_dataset,folder_dog))
```

```
print(f'Cat images found: {len(cat_imgs)}')
print(f'Dog images found: {len(dog_imgs)}')
```

Classes are balanced.

No Data augmentation

```
def load_data(path, input_shape=input_shape, batch_size=batch_size,
seed=123, validation_split=0.2):
    """Function to create 2 ImageDataGenerators to split dataset into
train and validation datasets.
    Data augmentation is not implemented for the validation
dataset."""
    height, width = input_shape[:2]
    datagen = ImageDataGenerator(rescale=1.0/255, zoom_range=0,
        horizontal_flip=True, vertical_flip=False,
        height_shift_range=0, width_shift_range=0,
        brightness_range=(0.99, 1.0), rotation_range=0,
        validation_split=validation_split
    )
    train_data = datagen.flow_from_directory(path,
        target_size=(height, width), batch_size=batch_size,
        class_mode='binary', subset='training', seed=seed
    )
    val_datagen = ImageDataGenerator(rescale=1.0/255,
        validation_split=validation_split
    )
    val_data = val_datagen.flow_from_directory(path,
        target_size=(height, width), batch_size=batch_size,
        class_mode='binary', subset='validation', seed=seed
    )
    return train_data, val_data

# Split training and validation datasets
train, val = load_data(path_dataset)

print(f"Classes found: {train.class_indices}")
print(f"Training images: {train.samples}")
print(f"Validation images: {val.samples}")

# Obtain images and target
images, labels = next(train)

# Show 8 training images (batch_size=8)
figure, axes = plt.subplots(nrows=2,ncols=4, figsize=(8, 6))
for item in zip(axes.ravel(), images, labels):
    axes, image, target = item
    axes.imshow(image)
    axes.set_title(f'Target: {target:.0f}')
    axes.set_xticks([])
```

```
    axes.set_yticks([])
plt.tight_layout()
plt.show()

# Image dimensions
print(images.shape)
```

Model training

```
def create_resnet_model(input_shape=input_shape, neurons=neurons,
                        learning_rate=learning_rate):
    """Function to create the model using the pretrained model
    'ResNet50' and adding some final layers. The backbone is
'ResNet50',
    but it is freezed (not trained) in this iteration."""

    backbone = ResNet50(weights='imagenet', input_shape=input_shape,
                        include_top=False)

    # Freeze ResNet50 without the top
    backbone.trainable = False
    model = Sequential()
    model.add(backbone)
    model.add(GlobalAveragePooling2D())
    model.add(Dense(neurons, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='binary_crossentropy', metrics=['accuracy'])
    return model

def train_model(model, train_data, val_data, epochs, version_model):
    """Function to train the model and save the best one
    according to the validation accuracy."""
    file_name =
os.path.join(folder_models,f'binary_model_v{version_model}.h5')

    callbacks = [
        EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True, verbose=0),
        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=1e-6, verbose=0),
        ModelCheckpoint(file_name, monitor='val_accuracy',
save_best_only=True, verbose=1)
    ]

    history = model.fit(train_data, validation_data=val_data,
              epochs=epochs, callbacks=callbacks, verbose=2)

    return model, history
```

```
epochs = 20
version_model = 1
print(f"Parameters: batch_size = {batch_size}, learning_rate =
{learning_rate}, neurons = {neurons}, epochs = {epochs}")

# Create and train the model v1
model = create_resnet_model()
model.summary()

print(f"TensorFlow Version: {tf.__version__}")

# Ensure GPU is available
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0],True)
    print("GPU is available and memory growth is enabled.")
else:
    print("GPU not available, training will be on CPU.")

# Train the model
model, history_stage1 = train_model(model, train, val, epochs=epochs,
version_model=version_model)
```

Result 1: val_accuracy=?%.

```
pd.DataFrame(history_stage1.history).plot(figsize=(12, 4))
plt.show()

# Save model
#
model.save(os.path.join(folder_models,f'binary_model_v{version_model}.
keras'))
```

In the next iteration, the model will be retrained, data augmentation and fine-tuning will be performed.