Confidential version 1

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

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 precess.

```
# (height, width, channels)
input_shape = (224, 224, 3)
batch_size = 8
learning_rate = 0.001
neurons = 128
path_dataset = 'dataset_cat_dogs'
folder_cat = 'Cat'
folder_dog = 'Dog'
folder_models = 'models'

# Path in Google Colab
# path = '/content/drive/MyDrive/Colab Notebooks/'
# path_dataset = path + '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
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)}')
```

```
Cat images found: 12491
Dog images found: 12470
```

```
def load_data(path, input_shape=input_shape, batch_size=batch_size,
seed=123, validation_split=0.2):
    height, width = input_shape[:2]
    datagen = ImageDataGenerator(rescale=1.0/255, zoom_range=0.0,
        horizontal_flip=True, vertical_flip=False,
        height_shift_range=0.0, width_shift_range=0.0,
        brightness_range=(0.9, 1.1), 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)
```

```
Found 19968 images belonging to 2 classes.
Found 4991 images belonging to 2 classes.
```

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

```
Classes found: {'Cat': 0, 'Dog': 1}
Training images: 19968
Validation images: 4991
```

```
# 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()
```


```
# Images dimentions
print(images.shape)
```

```
(8, 224, 224, 3)
```

Model

```
def create_resnet_model(input_shape=input_shape, neurons=neurons,
                        learning_rate=learning_rate):
    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):
    file_name = f'best_resnet50_v{version_model}.keras'
```

```
    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_acc',
save_best_only=True, verbose=1)
    ]

    history = model.fit(train_data, validation_data=val_data,
                #steps_per_epoch=train_data.samples //
train_data.batch_size,
                #validation_steps=val_data.samples //
val_data.batch_size,
                epochs=epochs, callbacks=callbacks,
                verbose=2)

    return model, history
```

Iteration 1: learning_rate = 1e-3, without fine-tunning

```
epochs = 10
version_model = 1
print(f"Parameters: batch_size = {batch_size}, learning_rate =
{learning_rate}, neurons = {neurons}, epochs = {epochs}")
```

```
Parameters: batch_size = 8, learning_rate = 0.001, neurons = 128,
epochs = 10
```

```
# Create and train the model v1
model = create_resnet_model()
model, history_stage1 = train_model(model, train, val, epochs=epochs,
version_model=version_model)
```

```
Epoch 1/10

2025-09-21 21:28:32.833196: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.
2025-09-21 21:28:33.682697: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.
2025-09-21 21:28:33.829896: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
26615808 exceeds 10% of free system memory.
2025-09-21 21:28:33.859257: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
25690112 exceeds 10% of free system memory.
2025-09-21 21:28:33.870281: W
tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
```

```
25690112 exceeds 10% of free system memory.
/home/ant/tensorflow3/env/lib/python3.8/site-packages/PIL/TiffImagePlu
gin.py:900: UserWarning: Truncated File Read
  warnings.warn(str(msg))

WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 4112s - loss: 0.6642 - accuracy: 0.6033 - val_loss: 0.6735
- val_accuracy: 0.6215 - lr: 0.0010 - 4112s/epoch - 2s/step
Epoch 2/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 2855s - loss: 0.6433 - accuracy: 0.6236 - val_loss: 0.6445
- val_accuracy: 0.6209 - lr: 0.0010 - 2855s/epoch - 1s/step
Epoch 3/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1853s - loss: 0.6295 - accuracy: 0.6452 - val_loss: 0.6227
- val_accuracy: 0.6474 - lr: 0.0010 - 1853s/epoch - 742ms/step
Epoch 4/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1821s - loss: 0.6214 - accuracy: 0.6520 - val_loss: 0.6042
- val_accuracy: 0.6702 - lr: 0.0010 - 1821s/epoch - 730ms/step
Epoch 5/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1800s - loss: 0.6109 - accuracy: 0.6599 - val_loss: 0.6598
- val_accuracy: 0.6209 - lr: 0.0010 - 1800s/epoch - 721ms/step
Epoch 6/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1802s - loss: 0.6051 - accuracy: 0.6699 - val_loss: 0.5899
- val_accuracy: 0.6796 - lr: 0.0010 - 1802s/epoch - 722ms/step
Epoch 7/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1796s - loss: 0.5980 - accuracy: 0.6773 - val_loss: 0.5970
- val_accuracy: 0.6792 - lr: 0.0010 - 1796s/epoch - 719ms/step
Epoch 8/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1809s - loss: 0.5949 - accuracy: 0.6785 - val_loss: 0.5946
- val_accuracy: 0.6788 - lr: 0.0010 - 1809s/epoch - 725ms/step
Epoch 9/10
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1762s - loss: 0.5904 - accuracy: 0.6837 - val_loss: 0.5777
- val_accuracy: 0.6975 - lr: 0.0010 - 1762s/epoch - 706ms/step
Epoch 10/10
```

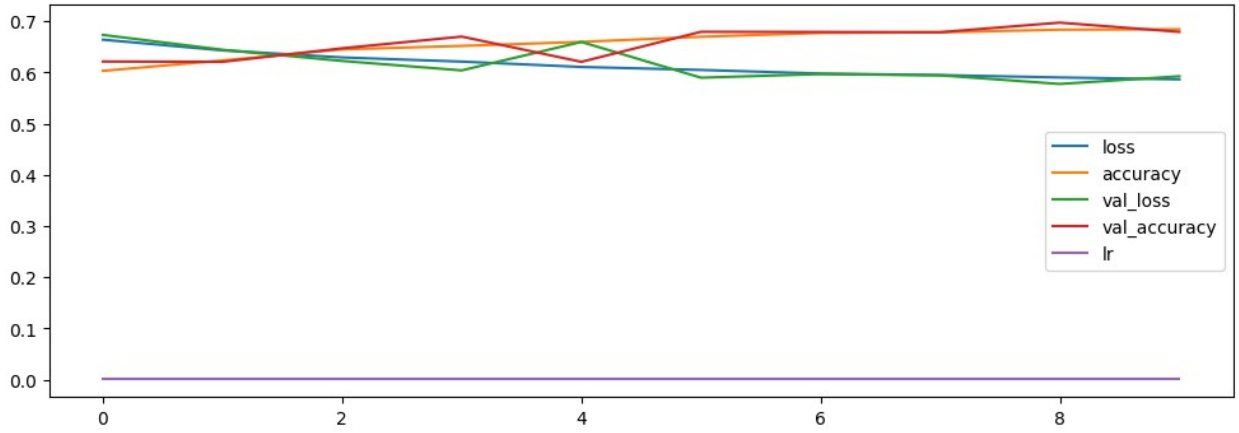
```
WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
2496/2496 - 1755s - loss: 0.5867 - accuracy: 0.6845 - val_loss: 0.5927
- val_accuracy: 0.6794 - lr: 0.0010 - 1755s/epoch - 703ms/step

/home/ant/tensorflow3/env/lib/python3.8/site-packages/keras/src/
engine/training.py:3000: UserWarning: You are saving your model as an
HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

Result 1: val_accuracy=67%.

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

<Axes: >
```




```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 128)               262272

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 23850113 (90.98 MB)
```

```
Trainable params: 262401 (1.00 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

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

```
import tensorflow as tf
print(f"TensorFlow Version: {tf.__version__}")
```

```
TensorFlow Version: 2.13.1
```