Assignment 9.1 Convolutional Neural Network

Name: Jomarie Dupaya

Course and Section: CPE019, CPE32S3

Instructor: Engr. Roman Richard

Date Performed: 4/25/24

Date Submitted: 4/27/24

Title: Lions or Cheetahs - Image Classification

Dataset Link: https://www.kaggle.com/datasets/mikoajfish99/lions-or-cheetahs-image-classification

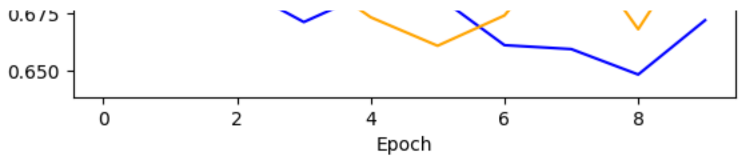
The problem is to predict which animal classifies the follwing images if the image is Lion or Cheetah since both have almost the same color but has different structure and skin pattern.

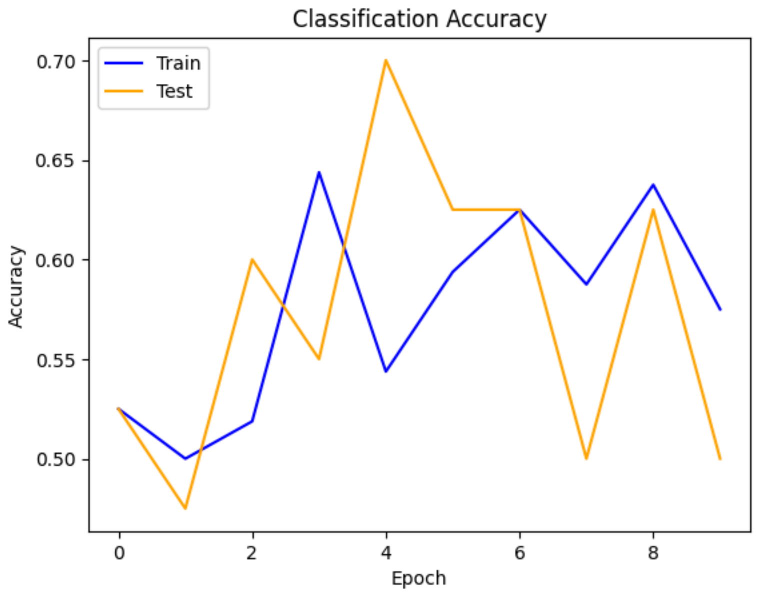
Using your dataset, create a baseline model of the CNN

```
import time
start_time = time.time()
from google.colab import drive
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import SGD
from sklearn.model_selection import KFold
from numpy import mean, std
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf

# Mount Google Drive
drive.mount('/content/drive')

# Define the image dimensions
img_width, img_height = 28, 28

# Define paths for the data folders for lions and cheetahs
data_directory = "/content/drive/My Drive/Colab Notebooks/dataset/lioncheetah"

lions_directory = os.path.join(data_directory, "Lions")
cheetahs_directory = os.path.join(data_directory, "Cheetahs")

# Get the list of file paths for lions and cheetahs images
lions_file_paths = [os.path.join(lions_directory, filename) for filename in os.listdir(lions_directory)]
cheetahs_file_paths = [os.path.join(cheetahs_directory, filename) for filename in os.listdir(cheetahs_directory)]

# Create labels for lions (0) and cheetahs (1)
y_lions = np.zeros(len(lions_file_paths))
y_cheetahs = np.ones(len(cheetahs_file_paths))

# Concatenate paths and labels
X_paths = lions_file_paths + cheetahs_file_paths
y = np.concatenate([y_lions, y_cheetahs])

# Convert y to strings
y = y.astype(str)

# Create a DataFrame with paths and labels
df = pd.DataFrame({'filename': X_paths, 'label': y})

# Split the data into train and test sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

def preprocess_image(image):
    resized_image = tf.image.resize(image, [img_width, img_height])
    resized_image = tf.reshape(resized_image, [img_width, img_height, 3])
    resized_image = tf.cast(resized_image, tf.float32)
    return resized_image

# Define data generators for training and testing with preprocessing function
train_datagen = ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=preprocess_image,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=preprocess_image
)

train_generator = train_datagen.flow_from_dataframe(
    train_df,
    x_col='filename',
    y_col='label',
    target_size=(img_width, img_height),
    batch_size=9,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_dataframe(
    test_df,
    x_col='filename',
    y_col='label',
    target_size=(img_width, img_height),
    batch_size=9,
    class_mode='binary'
)
```

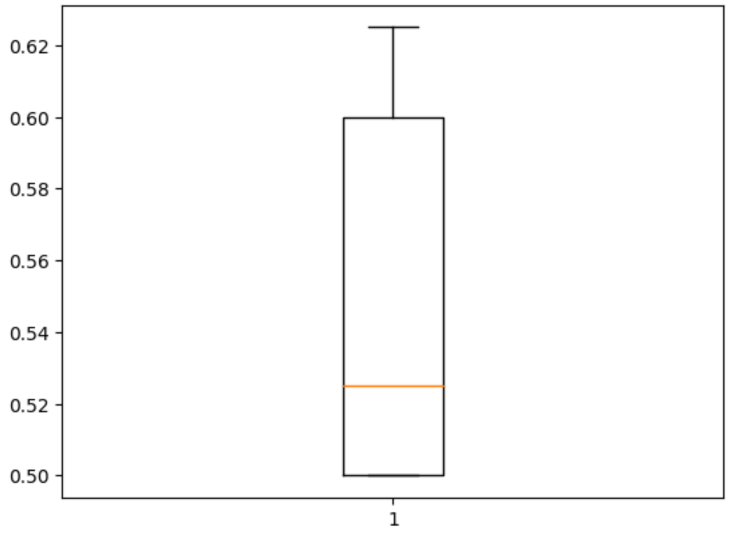
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 160 validated image filenames belonging to 2 classes.
Found 40 validated image filenames belonging to 2 classes.
```

```
# Define CNN model with baseline architecture
def define_baseline_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(img_width, img_height, 3)),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu', kernel_initializer='he_uniform'),
        Dense(1, activation='sigmoid')  # Output layer for binary classification
    ])

    # Compile model
    opt = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model

def evaluate_model(train_generator, test_generator, model_builder, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(train_generator):
        # define model
        model = model_builder()
        # fit model
        history = model.fit(train_generator, steps_per_epoch=len(train_generator), epochs=10, validation_data=test_generator, validation_ste
        # evaluate model
        _, acc = model.evaluate(test_generator, steps=len(test_generator), verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories

# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for history in histories:
        # plot loss
        plt.plot(history.history['loss'], color='blue', label='train')
        plt.plot(history.history['val_loss'], color='orange', label='test')
        plt.title('Cross Entropy Loss')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

        # plot accuracy
        plt.plot(history.history['accuracy'], color='blue', label='train')
        plt.plot(history.history['val_accuracy'], color='orange', label='test')
        plt.title('Classification Accuracy')
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
    # box and whisker plots of results
    plt.boxplot(scores)
    plt.show()

# entry point, run the test harness
scores, histories = evaluate_model(train_generator, test_generator, define_baseline_model)
# learning curves
summarize_diagnostics(histories)
# summarize estimated performance
summarize_performance(scores)
```





```
Accuracy: mean=55.000 std=5.244, n=5
```



Remarks: The baseline model cannot predict mostly of the images and its finding a hard time to process since the preprocessing is heavy on computing power of my devices therefore there are limits to the preprocessing hence the low accuracy.

Data Image Augmentation

```
# Random Rotations
train_datagen = ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=preprocess_image,
    rotation_range=20,  # Random rotations between -20 and +20 degrees
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True
)

# Generate augmented images
augmented_images = train_datagen.flow_from_dataframe(
    train_df,
    x_col='filename',
    y_col='label',
    target_size=(img_width, img_height),
    batch_size=9,
    class_mode='binary'
)

# Plot a grid of augmented images
fig, ax = plt.subplots(3, 3, figsize=(6, 6))
for i in range(3):
    for j in range(3):
        image, label = augmented_images.next()
        # Ensure pixel values are within the valid range for imshow
        image = np.clip(image[0], 0, 1)
        ax[i, j].imshow(image.astype('uint8'))
        ax[i, j].axis('off')
plt.tight_layout()
plt.show()
```