Winnie Leung
Professor Pham
EE104 Sec 01
April 30, 2022

LAB 8 README

Abstract

****

This lab allows users to utilize convolutional neural networks to recognize color images and try different methods to improve accuracy.

Objective

****

Fulfill all requirements listed in the table

Program or Requirement	Use Case	Earned Score / Max Score
Demonstration Video	You must submit a demonstration video or your score for this lab will be zero	
README file & Developer Documentation	README is a brief user guide and developer documentation so that the user can install the proper python packages and knows how to execute your program. The documentation section can contain sample screenshots with explanation.	_____ / 10
Check in your README file, documentation, and codes to Github	You will put all your work on your GitHub account and submit the same code to Canvas and share access to your GitHub with your instructor for grading purpose.	_____ / 5
CNN - Baseline + Increasing Dropout + Data Augmentation + Batch Normalization + Your own method	To achieve > 90% accuracy and recognize successfully 90% of the given test images. Video recording note: Because it will take a long time to run this test, you must submit a recording showing the result. The recording can be done by Zoom or any screen video capturing software. *Trim out the irrelevant contents and only submit the last few minutes showing you achieved your highest accuracy.* Achieve > 90% accuracy: Earn full 30 points Achieve > 87% accuracy: Earn 25 points Achieve > 84% accuracy: Earn 20 points Achieve > 80% accuracy: Earn 20 points Achieve > 77% accuracy: Earn 15 points Achieve > 74% accuracy: Earn 10 points Achieve > 70% accuracy: Earn 5 points Else: 0 point	_____ / 30
CNN - Challenge test	Recognize 5 unrecognizable images from the TEST IMAGES section below (3 points each) Hint: you can leverage and modify the test code from this GoogleColab page to test this lab: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/classification.ipynb#scrollTo=dC40sRITBSsQ	_____ / 15
Game Development – Balloon Flight	Leverage the base code from chapter Balloon Flight and add your own Hacks and Tweaks for any 4 of the options below: More High Scores, Lives, Speed It Up, Different Way to Score, File Handling, Add in Multiples of Each Obstacles, Level Up, Space Out the Obstacles	_____ / 40
	TOTAL	**100%**

Requirements

****

Google collab:

```
!pip install tensorflow
!pip install keras
!pip install h5py
!pip install Matplotlib
!pip install numpy
```

```
import tensorflow as tf
# baseline model with dropout and data augmentation on the cifar10 dataset
import sys
import numpy as np
from keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout
from keras.layers import BatchNormalization
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Pygame:

```
import pgzrun
import pygame
import pgzero
import random
from pgzero.builtins import Actor
from random import randint
```

Instruction

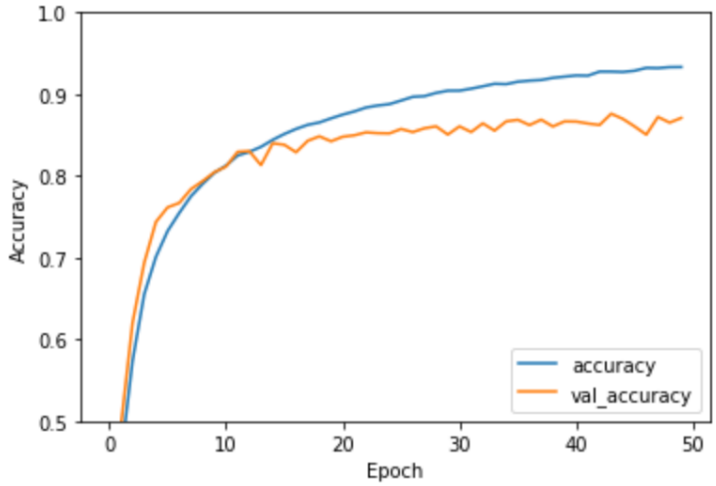
****

- Ensure all packages are installed prior to running an programs
- CNN
    - Download and run the Copyofcnn.ipynb
    - Observe the code and view the output
    - The goal of the program is to increase the classification accuracy but applying optimization techniques.
    - The output will look like the following

```
Epoch 35/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2610 - accuracy: 0.9123 - val_loss: 0.4496 - val_accuracy: 0.8667
Epoch 36/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2527 - accuracy: 0.9152 - val_loss: 0.4461 - val_accuracy: 0.8686
Epoch 37/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2504 - accuracy: 0.9165 - val_loss: 0.4695 - val_accuracy: 0.8622
Epoch 38/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2433 - accuracy: 0.9174 - val_loss: 0.4414 - val_accuracy: 0.8688
Epoch 39/50
1563/1563 [==============================] - 39s 25ms/step - loss: 0.2378 - accuracy: 0.9200 - val_loss: 0.4656 - val_accuracy: 0.8605
Epoch 40/50
1563/1563 [==============================] - 39s 25ms/step - loss: 0.2363 - accuracy: 0.9215 - val_loss: 0.4467 - val_accuracy: 0.8667
Epoch 41/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2288 - accuracy: 0.9229 - val_loss: 0.4509 - val_accuracy: 0.8665
Epoch 42/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2280 - accuracy: 0.9226 - val_loss: 0.4554 - val_accuracy: 0.8641
Epoch 43/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2181 - accuracy: 0.9276 - val_loss: 0.4629 - val_accuracy: 0.8623
Epoch 44/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2170 - accuracy: 0.9276 - val_loss: 0.4316 - val_accuracy: 0.8761
Epoch 45/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2159 - accuracy: 0.9272 - val_loss: 0.4456 - val_accuracy: 0.8697
Epoch 46/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2107 - accuracy: 0.9287 - val_loss: 0.4900 - val_accuracy: 0.8605
Epoch 47/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.1996 - accuracy: 0.9322 - val_loss: 0.5334 - val_accuracy: 0.8505
Epoch 48/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.2028 - accuracy: 0.9319 - val_loss: 0.4515 - val_accuracy: 0.8721
Epoch 49/50
1563/1563 [==============================] - 38s 24ms/step - loss: 0.1971 - accuracy: 0.9331 - val_loss: 0.4747 - val_accuracy: 0.8652
Epoch 50/50
1563/1563 [==============================] - 39s 25ms/step - loss: 0.1975 - accuracy: 0.9332 - val_loss: 0.4474 - val_accuracy: 0.8709
```

-

```
313/313 - 2s - loss: 0.4474 - accuracy: 0.8709 - 2s/epoch - 7ms/step
```



- CNN Challenge
  - Run the code at the end of the copyofcnn.ipynb
  - Jaguar car

```
image_url = "https://images.all-free-download.com/images/graphiclarge/classic_jaguar_210354.jpg"
image_path = tf.keras.utils.get_file('classic_jaguar_210354.jpg', origin=image_url)

img = tf.keras.utils.load_img(
    image_path, target_size=(32, 32)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
print(score)
```

```
This image most likely belongs to automobile with a 99.61 percent confidence.
tf.Tensor(
[1.0455601e-04 9.9605584e-01 2.9018420e-06 3.7188004e-06 2.1646790e-06
 2.8373606e-06 2.3319881e-05 3.9860945e-05 1.3030920e-04 3.6344097e-03], shape=(10,), dtype=float32)
```

- Devel-motors sixteen

```
image_url = "https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/devel-motors-sixteen-1540564064.jpg"
image_path = tf.keras.utils.get_file('devel-motors-sixteen-1540564064.jpg', origin=image_url)

img = tf.keras.utils.load_img(
    image_path, target_size=(32, 32)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
print(score)
```

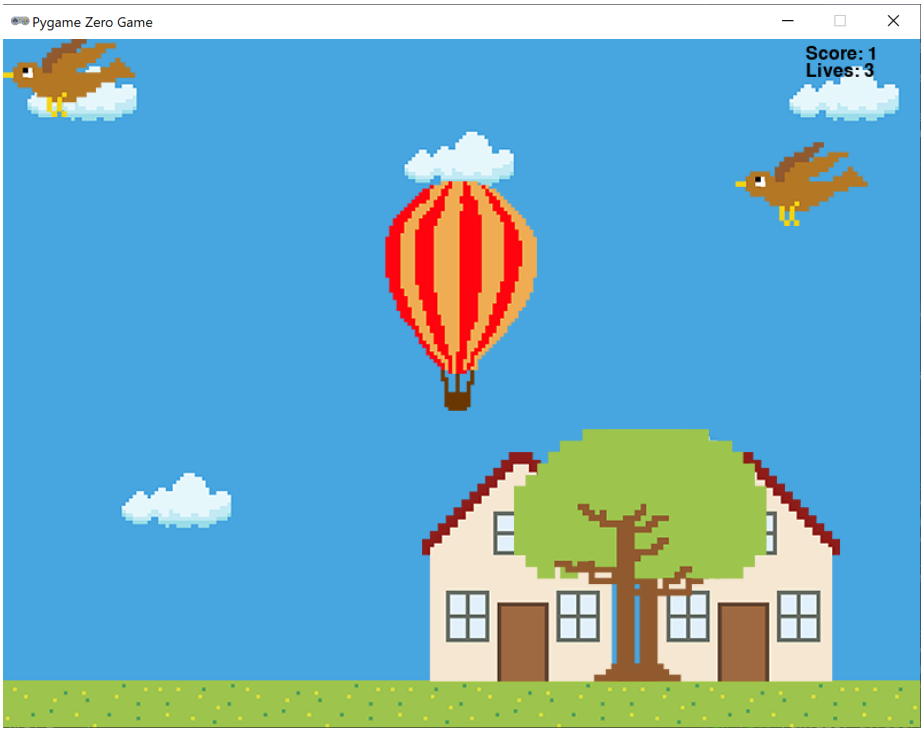
```
Downloading data from https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/devel-motors-sixteen-1540564064.jpg
1335296/1333618 [==============================] - 0s 0us/step
1343488/1333618 [==============================] - 0s 0us/step
This image most likely belongs to automobile with a 99.90 percent confidence.
tf.Tensor(
[6.44805550e-05 9.98974562e-01 2.19593244e-06 2.68521694e-06
 1.84617898e-06 2.09927771e-06 1.92146726e-05 2.66038642e-05
 1.03100574e-04 8.03270203e-04], shape=(10,), dtype=float32)
```

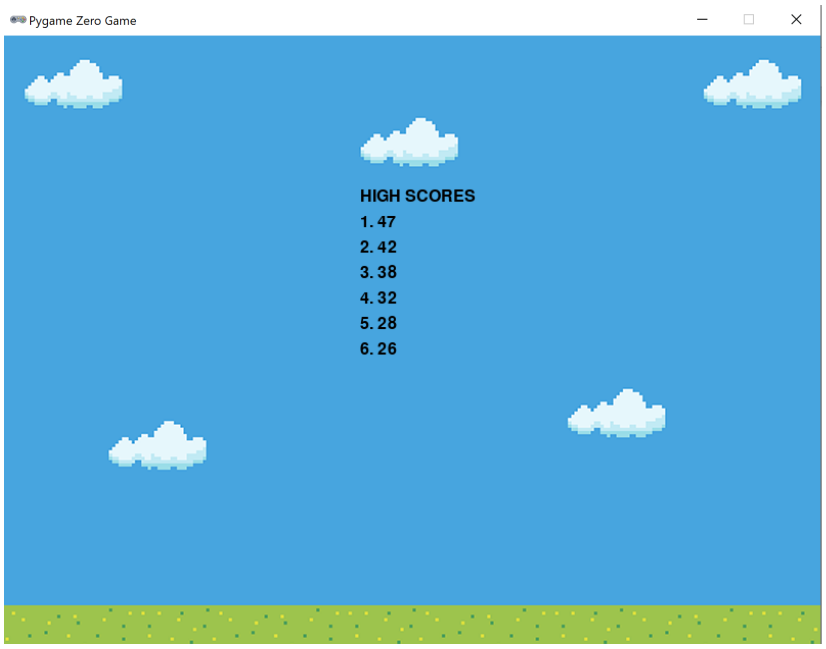
- Valkyrie-spider

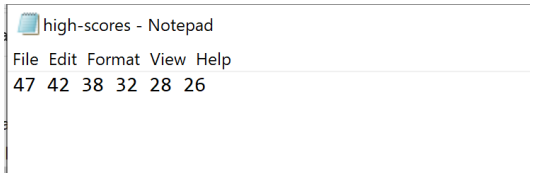
```
image_url = "https://amsc-prod-cd.azureedge.net/-/media/aston-martin/images/default-source/models/valkyrie/new/valkyrie-spider_f02-169v2.jpg?mw=1980&rev=-1&hash=
image_path = tf.keras.utils.get_file('valkyrie-spider_f02-169v2.jpg', origin=image_url)

img = tf.keras.utils.load_img(
    image_path, target_size=(32, 32)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
print(score)
```

```
Downloading data from https://amsc-prod-cd.azureedge.net/-/media/aston-martin/images/default-source/models/valkyrie/new/valkyrie-spider_f02-169v2.jpg?mw=1980&rev
729088/727309 [==============================] - 0s 0us/step
737280/727309 [==============================] - 0s 0us/step
This image most likely belongs to automobile with a 99.90 percent confidence.
tf.Tensor(
[6.4726781e-05 9.9897408e-01 2.2255635e-06 2.7941962e-06 1.8732994e-06
 2.2226513e-06 1.9458708e-05 2.7087044e-05 1.0330604e-04 8.0223172e-04], shape=(10,), dtype=float32)
```

- Pygame- Balloon Fight
  - Download the program BaloonFight.py run and observe the code
  - Results should look like the following:







.txt file updates to display the scores in the game