

Nuh Ersahin  
Professor Pham  
EE-104

## Lab 8 Documentation

### Part 1

In this part of the lab, the goal was to develop a convolutional neural network to recognize images by training a CIFAR-10 image dataset. The unique aspect of this dataset is that it contains thousands of images with multiple different classes. To train this model TensorFlow was utilized. Tensorflow is an open-source end-to-end machine learning platform that is a standard in the industry. Image 1 below shows the importing of TensorFlow into the project. Image 2 shows the multiple layers that are included in the program. These layers represent the model that is developed. The model starts with a simple 3X3 pooling layer, meaning that as images go down the multiple layers the image size will be shrunk by half. Then dropouts were included. These dropouts are layers that included random neurons to be dropped so the model stays to a certain size. Next, batch normalization is used to rescale and recenter the output of each layer. Finally, the flatten function of the code enables the files of each later to be laid out in a way for the final layer. Image 3 shows the plot of the final results, and image 4 shows the actual accuracy value. In the plot, it is important to notice that the general accuracy reaches over 90% but the final value accuracy is 83.5%.

```
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 Dense
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import BatchNormalization
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Image 1: Imports Needed

```
l = models.Sequential()
l.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(28, 28, 3)))
l.add(BatchNormalization())
l.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(MaxPooling2D((2, 2)))
l.add(Dropout(0.1))

l.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(MaxPooling2D((2, 2)))
l.add(Dropout(0.2))

l.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(MaxPooling2D((2, 2)))
l.add(Dropout(0.3))

l.add(Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
l.add(BatchNormalization())
l.add(MaxPooling2D((2, 2)))
l.add(Dropout(0.4))

l.add(Flatten())
l.add(Dense(10, activation='softmax'))
```

Image 2: Showing the model code

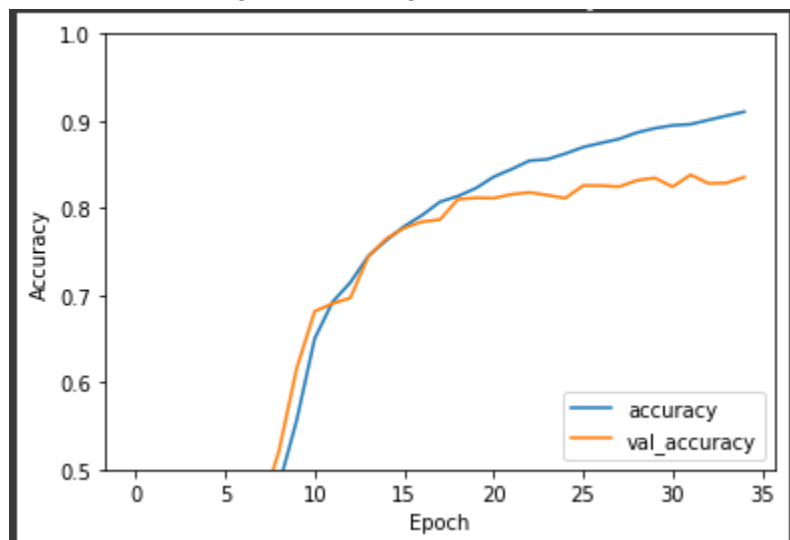


Image 3: Output Plot

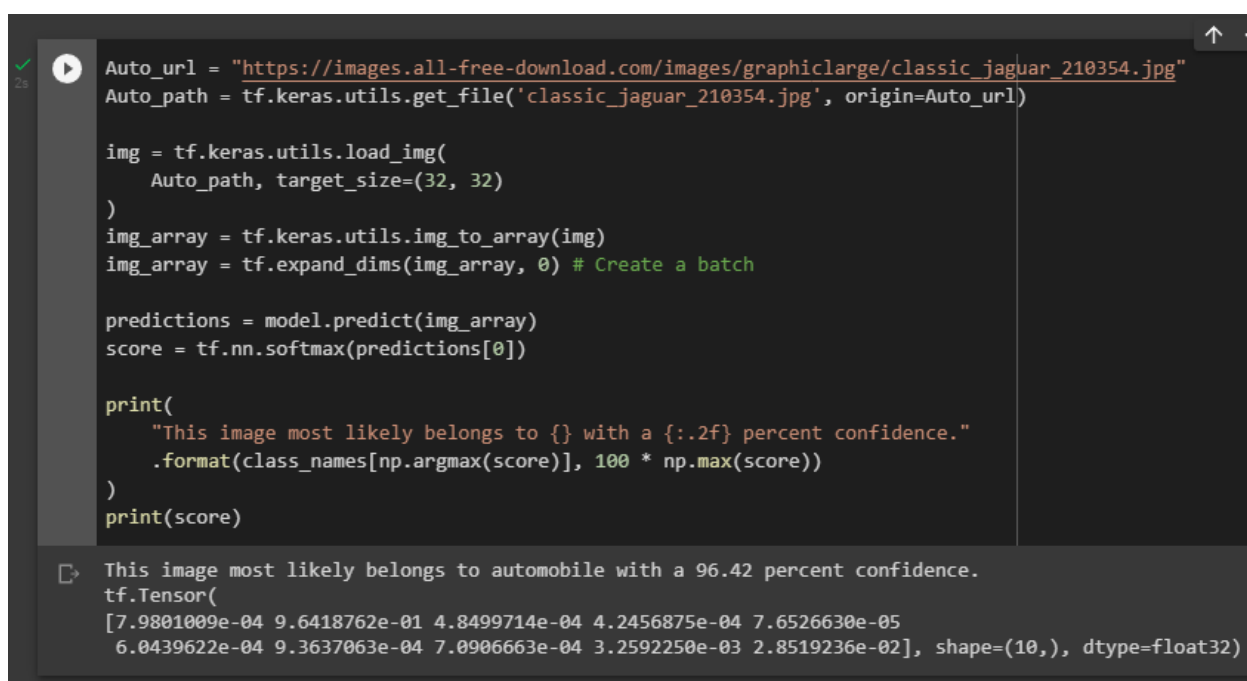
```
print(test_acc)
```

0.8355000019073486

## Image 4: Final Value Accuracy

### Part 2:

This part of the lab utilizes the same base code as the previous part with a few new added functions. 5 new cells were added to the code and in these cells, the goal was to be able to detect unrecognizable images using the dataset and the AI. In the image below it can be seen the code is able to detect an automobile in the code that the object is an automobile with 96.42 percent confidence. This was the only one I was able to classify as the others continually had problems I was unsure how to resolve.



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

img = tf.keras.utils.load_img(
    Auto_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 96.42 percent confidence.

```
tf.Tensor(
[7.9801009e-04 9.6418762e-01 4.8499714e-04 4.2456875e-04 7.6526630e-05
 6.0439622e-04 9.3637063e-04 7.0906663e-04 3.2592250e-03 2.8519236e-02], shape=(10,), dtype=float32)
```

### Part 3:

For this part of the lab, 4 hacks and tweaks were added to the game. The first one was to display more high scores. To do this I opened the .txt file that is in the game files folder and added two more zeros. These zeros are placeholders for future high scores. This enabled me to display 5 high scores at the end of the game instead of just 3.

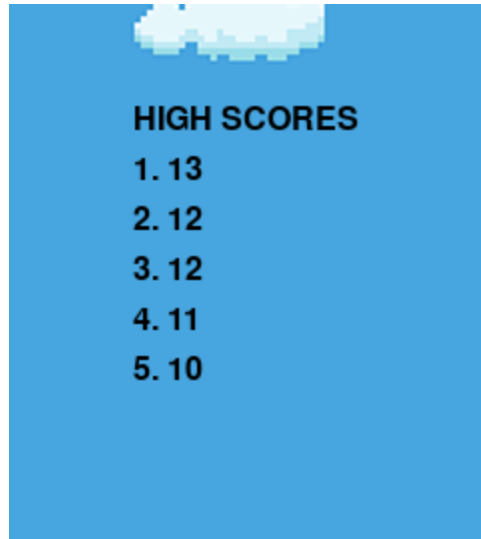


Image 1: High Scores Displayed

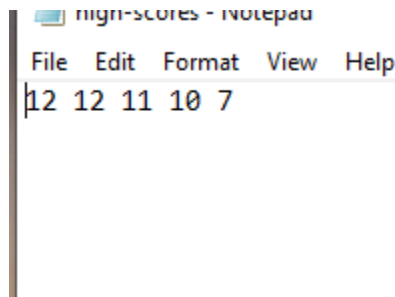


Image 2: Corresponding .txt file

The next tweak that was made was to speed up the bird. To do this the change in the bird's x plane was changed from 2 to 7. This can be seen in the file below.

```
def update():
    global game_over, score, number_of_updates
    if not game_over:
        if not up:
            balloon.y += 1
        if bird.x > 0:
            bird.x -= 7 # speed up on the bird
            if number_of_updates == 14:
                flap()
            number_of_updates = 0
    else:
```

Image 3: Code showing sped up bird

The next function I added was multiples of each object so the bird the house and the tree all have two of them on the screen. They all follow the same functionality as the originals such as speed and placement and point-scoring(discussed below). To do this multiple lines of code were added, the images below show the changes made.

```
def draw():
    screen.blit("background", (0, 0))
    if not game_over:
        balloon.draw()
        bird.draw()
        bird2.draw()
        house.draw()
        tree.draw()
        house2.draw()
        tree2.draw()
```

```
bird = Actor("bird-up")
bird.pos = randint(800, 1600), randint(10, 200)

bird2 = Actor("bird-up")
bird2.pos = randint(800, 1600), randint(10, 200)

house = Actor("house")
house.pos = randint(800, 1600), 460

house2 = Actor("house")
house2.pos = randint(800, 1600), 460

tree = Actor("tree")
tree.pos = randint(800, 1600), 450

tree2 = Actor("tree")
tree2.pos = randint(800, 1600), 450
```

```

    if balloon.top < 0 or balloon.bottom > 560:
        game_over = True
        update_high_scores()
    if (balloon.collidepoint(bird.x, bird.y) or
        balloon.collidepoint(bird2.x, bird2.y) or
        balloon.collidepoint(house.x, house.y) or
        balloon.collidepoint(house2.x, house2.y) or
        balloon.collidepoint(tree.x, tree.y) or
        balloon.collidepoint(tree2.x, tree2.y)):
        game_over = True
        update_high_scores()

```

```

141         score += 1
142         number_of_updates = 0
143
144         if bird2.x > 0:
145             bird2.x -= 3 # speed up on the bird
146             if number_of_updates == 6:
147                 flap()
148                 number_of_updates = 0
149             else:
150                 number_of_updates += 1
151         else:

```

```

2         score += 1
3
4     if house2.right > 0:
5         house2.x -= 2
6         add_score()
7     else:
8         house2.x = randint(800, 1600)
9         score += 1
10
11     if tree.right > 0:

```

```

176         score += 1
177
178     if tree2.right > 0:
179         tree2.x -= 2
180         add_score()
181     else:
182         tree2.x = randint(800, 1600)
183         score += 1
184
185     if balloon.top < 0 or balloon.bottom > 560:

```

The last part was to change the way the scoring is handled. The base game has it set so the moment the balloon reaches an object you get a point. The way I have it set is the moment the balloon passes the object, the tree, or the house, not the bird, it will increment the score by one. The image below shows how the code was added.

```

95     up = False
96
97     def add_score():
98         global score
99         if house.right == 400 or house.right == 399:
100             score += 1
101         if tree.right == 400 or tree.right == 399:
102             score += 1
103         if house2.right == 400 or house2.right == 399:
104             score += 1
105         if tree2.right == 400 or tree2.right == 399:
106             score += 1

```