In this lab a Neural Network was made from scratch on google colab to look at a dataset and allow AI to guess what kind of image is being shown. The code for this lab is quite complicated, and due to time issues the accuracy of the CNN was only 77%. The code begins by importing a data set of thousands of images and defining 25 classes that these images belong to. Most of this is done using tensor flow.

Import TensorFlow

```
import tensorflow as tf

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

Download and prepare the CIFAR10 dataset

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

```
[3] (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

    # Normalize pixel values to be between 0 and 1
    train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 14s 0us/step
```

Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image:

```
[4] class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                   'dog', 'frog', 'horse', 'ship', 'truck']

    plt.figure(figsize=(10,10))
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(train_images[i])
        # The CIFAR labels happen to be arrays,
        # which is why you need the extra index
        plt.xlabel(class_names[train_labels[i][0]])
    plt.show()
```

After the data is uploaded layers are created for the neural network model. These layers help to guess the image.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))
```

Once the layers are created for the model, the model is tested by uploading a certain amount of images with the correct label and then allowing the AI model to guess on the rest of the images. This allows us to see the accuracy of the model.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Once the epochs or number of test runs are done the data can be plotted and the accuracy shown.

```
Epoch 1/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4958 - accuracy: 0.8277 - val_loss: 0.6652 - val_accuracy: 0.7867
Epoch 2/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4808 - accuracy: 0.8331 - val_loss: 0.5975 - val_accuracy: 0.8054
Epoch 3/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4679 - accuracy: 0.8377 - val_loss: 0.6261 - val_accuracy: 0.8043
Epoch 4/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4661 - accuracy: 0.8348 - val_loss: 0.6535 - val_accuracy: 0.7954
Epoch 5/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4564 - accuracy: 0.8397 - val_loss: 0.6246 - val_accuracy: 0.8023
Epoch 6/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4535 - accuracy: 0.8435 - val_loss: 0.6576 - val_accuracy: 0.7953
Epoch 7/10
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4492 - accuracy: 0.8432 - val_loss: 0.6633 - val_accuracy: 0.7942
Epoch 8/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4424 - accuracy: 0.8458 - val_loss: 0.6463 - val_accuracy: 0.8031
Epoch 9/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4260 - accuracy: 0.8508 - val_loss: 0.6463 - val_accuracy: 0.7972
Epoch 10/10
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4288 - accuracy: 0.8516 - val_loss: 0.6839 - val_accuracy: 0.7897
```

+ Code + Text

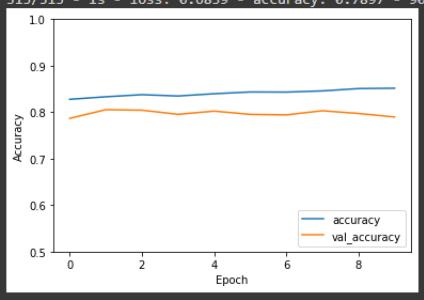
Evaluate the model

+ Code + Text

```
[16] plt.plot(history.history['accuracy'], label='accuracy')
     plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.ylim([0.5, 1])
     plt.legend(loc='lower right')

     test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.6839 - accuracy: 0.7897 - 900ms/epoch - 3ms/step
```




```
[ ] print(test_acc)
```

```
0.7896999716758728
```

The model is then saved and can be downloaded for future use. The saved model can be uploaded and used to check test images to see if the AI can guess them correctly.

```
# You will use this trained model to test the images
model.save('MyGroup_CIFARmodel_baseline.h5')

## Save file to your local computer if you will test it locally
##  using the provided file test_image.py
#https://neptune.ai/blog/google-colab-dealing-with-files
from google.colab import files
files.download('MyGroup_CIFARmodel_baseline.h5')

#########################################################################
## If you run on GoogleColab, then add the following code on GoogleColab ##
#########################################################################

# load the trained CIFAR10 model
from keras.models import load_model
model = load_model('MyGroup_CIFARmodel_baseline.h5')

def load_image(filename):
  img = load_img(filename, target_size=(32, 32))
  img = img_to_array(img)
  img = img.reshape(1, 32, 32, 3)
  img = img / 255.0
  return img

#https://stackoverflow.com/questions/72479044/cannot-import-name-load-img-from-keras-preprocessing-image
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.models import load_model

# get the image from the internet
URL = "https://wagznwhiskerz.com/wp-content/uploads/2017/10/home-cat.jpg "
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)


# show the picture
image = plt.imread(picture_path)
plt.imshow(image)

# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])

# get the image from the internet
URL = "https://wagznwhiskerz.com/wp-content/uploads/2017/10/home-cat.jpg"
picture_path  = tf.keras.utils.get_file(origin=URL)
img = load_image(picture_path)
result = model.predict(img)


# show the picture
image = plt.imread(picture_path)
plt.imshow(image)

# show prediction result.
print('\nPrediction: This image most likely belongs to ' + class_names[int(result.argmax(axis=-1))])


############################################################
## This website has everything to improve your accuracy  ###
############################################################
```

For the balloon game most of the code remained the same from the textbook with changes to number of highscores displayed at the end, lives given to the player to be able to continue for an extra 3 turns, changing the score to increase right after the player passes an obstacle as to as the obstacle exits the screen, and spacing out the obstacle so they don’t show up on top of each other.