VNUK Institute for Research & Executive Education

Artificial Intelligence

::Challenge 1 (10%)

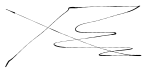
Due: 11 Nov 2023, 11:59 PM

Email to: vu.tran@vnuk.edu.vn

According to the VNUK *Academic integrity policy*, plagiarism is:

"Claiming and using the thoughts or writings or creative works of others without appropriate acknowledgement or attribution. It includes:
(a) copying part or all of another student's assignment;
(b) allowing another person to write some or all of an assignment;
(c) copying paragraphs, sentences or parts of sentences directly from texts or the internet without enclosing them in quotation marks or otherwise showing them to be copied - even if the source is acknowledged, this is still plagiarism;
(d) using concepts or developed ideas, even if paraphrased or summarised, from another person, from texts or the internet without acknowledging the source;
(e) copying graphics, architectural plans, multimedia works or other forms of intellectual property without appropriate acknowledgment."

The consequences of plagiarism (depending on the seriousness of the case) range from reducing your mark or failing the assignment up to a formal reference to a summary inquiry:

By signing below I certify that the attached assignment is my own work. Student ID: 21020006 Student Name: Nguyen Tran Xuan Tri Signature: 

Grade:

No.	Question	Grade
1	Question 1	
2	Question 2	
3	Question 3	
4	Question 4	
5	Question 5	
	Total gold coins	

This problem set will introduce you to using control flow in Python and formulating a computational solution to a problem.

Data

https://github.com/googlecreativelab/quickdraw-dataset

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. You can browse the recognized drawings on quickdraw.withgoogle.com/data.

Requirements:

No.	Criteria	Weight (%)
1	Train the model	20%
2	Deploy the model	30%
3	Explain the math/model	15%
4	Complete app	15%
5	Git usage	10%

Questions

1. **Train the model**

   Choose only 2 or 4 items to train your model

   Getting Data:

   + creating a new CNN class for implementing a Convolutional Neural Network model
   + loading three datasets (for example: car, fish and snowman)
   + splitting datasets into training and test data shuffling data

   Building the Model:

   + creating a sequential CNN model
   + adding layers to the model
   + compiling the model

   Training the Model

   + fetching batches of data
   + training, testing and evaluating the model
   + plotting graphs of the model loss and accuracy during training

   Reference:

   https://github.com/zaidalyafeai/Notebooks/blob/master/Sketcher.ipynb

   Explain about CNN layers:

   https://youtu.be/NL6eCtMjikQ

2. **Deploy the model**

   Use Web technology (for example Python Flask/NodeJS/PHP/Laverel) to code function same as https://quickdraw.withgoogle.com/

   Predicting Samples

   + fetching batches of samples
   + predicting fetched samples

   Drawing Doodles

   + creating a new Painter class to allow users to draw their own doodles with the mouse
   + defining painting objects: drawing area, bitmaps, pencil
   + adding a function for drawing a smooth line between two points using quadratic curves

   Recognizing Doodles

   + resizing doodle drawing to the required size of 28x28

+ normalizing array of pixels before passing it as the input of the CNN model predicting doodle

Debriefing Report :: Part 1

Part 1. Report on the challenge.

Download items:

In this challenge, I chose 4 items to train the model including: apple, rainbow, cloud, mountain, and store them in a list of classes.

```
classes = ['apple', 'rainbow', 'cloud', 'mountain']
```

In the next line, I define the base URL for the Google Cloud Storage where the Quick, Draw! dataset is stored.

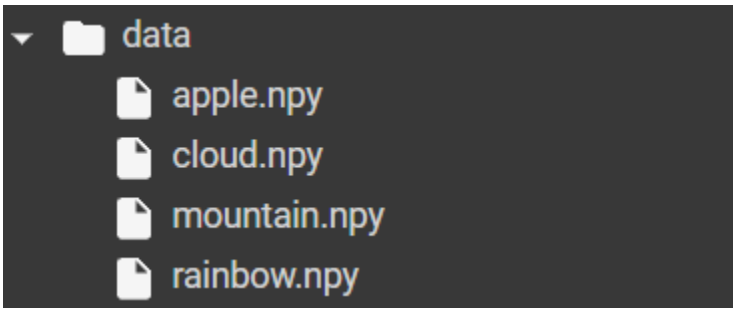
```
base = 'https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/'
```

Both of the lines of code above are written in a function named download with a loop that defines a path to load the data of each of the initially selected items.

```python
import urllib.request

def download():
    # List of class names want to download
    classes = ['apple', 'rainbow', 'cloud', 'mountain']

    base = 'https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/'

    for c in classes:
        cls_url = c.replace('_', '%20')
        path = base + cls_url + '.npy'
        print("Downloading:", path)
        urllib.request.urlretrieve(path, 'data/' + c + '.npy')

# Call the download function to start downloading the selected items
download()
```

```
Downloading: https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/apple.npy
Downloading: https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/rainbow.npy
Downloading: https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/cloud.npy
Downloading: https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/mountain.npy
```

As a result, the items' data files are downloaded and saved in the data path defined during the download function.



Loading the data:

After having datasets, the next step is loading the data into the kernel.

A load_data function is created with three parameters:

- root: The path to the data files' directory.
- vfold_ratiro: The ratio of the data to be used for validation/testing.
- max_item_per_class: The maximum number of items to be taken from each class.

Here, the default value for vfold_ratio is 0.2 (means 20% of the dataset will be used for testing/validation, while 80% will be used for training, as is common in machine learning for evaluating model performance), and that of max_item_per_class is 4000.

```
def load_data(root, vfold_ratio=0.2, max_items_per_class= 4000 ):
```

In the first line of the function, it uses glob.glob to gather all files with .npy extension in root directory and store them in an all_files variable.

```
    all_files = glob.glob(os.path.join(root, '*.npy'))
```

Then initialize an empty x and y array with x of the form [0, 784] and y of the form [0] for passing data in, and a class_name array to contain the class names.

```
#initialize variables
x = np.empty([0, 784])
y = np.empty([0])
class_names = []
```

The data is then loaded by looping through each file in the.npy file list that has been collected from the all_files variable. It selects a subset of the data to limit the number of items per class. It then generates an array of labels, with each label corresponding to the index of the current layer in the overall list of layers. The data and labels are concatenated to the existing x and y arrays. In addition, the loop extracts the class name from the file name and adds it to a list class_names. This process is repeated for each file, efficiently loading, preprocessing, and organizing the data into a format suitable for machine learning tasks, where x contains input features and y contains output features. In short, this process is repeated for each file, efficiently loading, preprocessing, and organizing data into a format suitable for machine learning tasks, where x represents input features, y represents labels, and class_names represents class names.

```
#load each data file
for idx, file in enumerate(all_files):
    data = np.load(file)
    data = data[0: max_items_per_class, :]
    labels = np.full(data.shape[0], idx)

    x = np.concatenate((x, data), axis=0)
    y = np.append(y, labels)

    class_name, ext = os.path.splitext(os.path.basename(file))
    class_names.append(class_name)
```

The dataset contained within the x and y array is then randomized, preventing any potential bias during training by permuting the array's indices.

```
#randomize the dataset
permutation = np.random.permutation(y.shape[0])
x = x[permutation, :]
y = y[permutation]
```

The dataset is divided into a training set and a test set after being randomized based on the vfold_size value explained at the beginning.

```
#separate into training and testing
vfold_size = int(x.shape[0]/100*(vfold_ratio*100))

x_test = x[0:vfold_size, :]
y_test = y[0:vfold_size]

x_train = x[vfold_size:x.shape[0], :]
y_train = y[vfold_size:y.shape[0]]
```

Finally, the function returns training data, training labels, test data, test labels, and a list of class names.

The overview of the function:

```python
def load_data(root, vfold_ratio=0.2, max_items_per_class= 4000 ):
    all_files = glob.glob(os.path.join(root, '*.npy'))

    #initialize variables
    x = np.empty([0, 784])
    y = np.empty([0])
    class_names = []

    #load each data file
    for idx, file in enumerate(all_files):
        data = np.load(file)
        data = data[0: max_items_per_class, :]
        labels = np.full(data.shape[0], idx)

        x = np.concatenate((x, data), axis=0)
        y = np.append(y, labels)

        class_name, ext = os.path.splitext(os.path.basename(file))
        class_names.append(class_name)

    data = None
    labels = None

    #randomize the dataset
    permutation = np.random.permutation(y.shape[0])
    x = x[permutation, :]
    y = y[permutation]

    #separate into training and testing
    vfold_size = int(x.shape[0]/100*(vfold_ratio*100))

    x_test = x[0:vfold_size, :]
    y_test = y[0:vfold_size]

    x_train = x[vfold_size:x.shape[0], :]
    y_train = y[vfold_size:y.shape[0]]
    return x_train, y_train, x_test, y_test, class_names
```

In the end, call the function to receive data and labels for training and testing the model, and also get the number of classes and set the image size to 28x28.

```
x_train, y_train, x_test, y_test, class_names = load_data('data')
num_classes = len(class_names)
image_size = 28
```

Preprocess the data:

Preprocess data to ensure it is fit for training and will perform better when used to build models.

Reshape: The input data (x_train and x_test) is reshaped to a 4D tensor in this step.Many deep learning frameworks, including TensorFlow and Keras, expect input data to be in the form of a 4D tensor, with the dimensions corresponding to the number of samples, height, width, and channels.

Normalize:

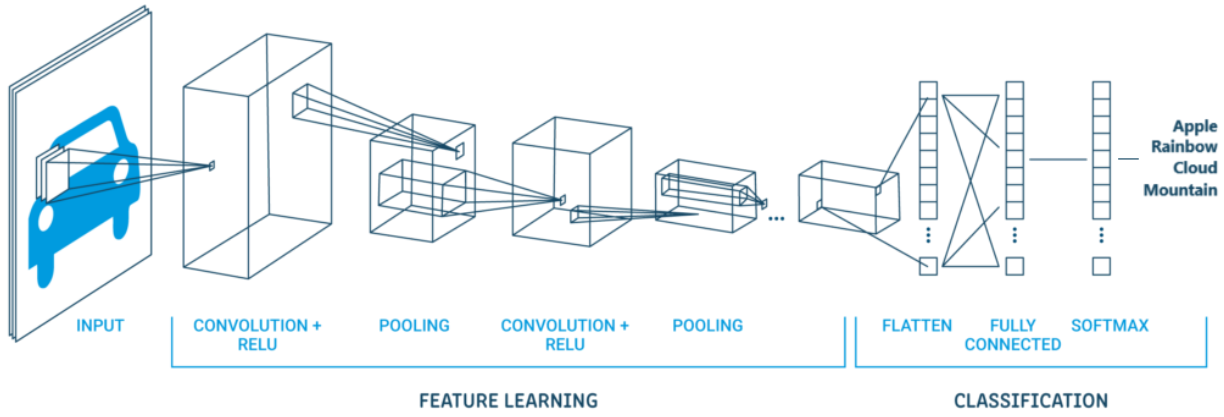
After reshaping, the pixel values of the images are normalized. Dividing each pixel value by 255 achieves this normalization.

As well as to answer the question of what and why normalization, Normalization of input data is a common practice in machine learning because it aids in the convergence of the training process, as neural networks often perform better and converge faster when input values are within a smaller, more uniform range. The goal is to transform the data into a standard range while preserving the differences between the individual feature ranges.

```
# Reshape and normalize
x_train = x_train.reshape(x_train.shape[0], image_size, image_size, 1).astype('float32')
x_test = x_test.reshape(x_test.shape[0], image_size, image_size, 1).astype('float32')

x_train /= 255.0
x_test /= 255.0

# Convert class vectors to class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

The Model:

To make the model for training, I define a Convolutional Neural Network (CNN) using the Keras library with a TensorFlow backend. Below is the illustration of how CNN works in the model with the classes (apple, rainbow, cloud, mountain):



First, initializes a sequential model, which is a linear stack of layers. In Keras, the sequential model is a simple and common way to build neural networks layer by layer.

```
model = keras.Sequential()
```

Then, with max-pooling, add three convolutional layers. Each Convolution2D layer represents a convolutional operation with a specified number of filters (16, 32, and 64, respectively), a filter size of 3x3, and the same padding, which means the output feature map has the same spatial dimensions as the input.

Rectified Linear Unit (ReLU) activation function is used, which introduces non-linearity into the model. A MaxPooling2D layer is added after each convolutional layer to perform max pooling with a pool size of 2x2. Max pooling reduces the spatial dimensions of the feature map, which aids in the extraction of key features and lowers computational load.

```
model.add(layers.Convolution2D(16, (3, 3),
                        padding='same',
                        input_shape=input_shape, activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Convolution2D(32, (3, 3), padding='same', activation= 'relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Convolution2D(64, (3, 3), padding='same', activation= 'relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
```

Before proceeding from the convolutional layer to the fully connected layer, add a flattening layer, which converts the multidimensional output from the convolutional layers into a 1D array.

```
model.add(layers.Flatten())
```

After that, two dense layers are added. The hyperbolic tangent (tanh) activation function is used in the first dense layer, which has 128 units. The second dense layer has the same number of units as the number of classes in the classification problem and employs the softmax activation function, which is commonly used for multi-class classification problems because it converts the raw output of the model into probability distributions across multiple classes.

```
model.add(layers.Dense(128, activation='tanh'))
model.add(layers.Dense(num_classes, activation='softmax'))
```

Finally, compile the model with the categorical cross-entropy loss function, Adam optimizer, and a custom metric called top_k_categorical_accuracy and return the model. Adam is a neural network training optimization algorithm. The metric top_k_categorical_accuracy measures top-k accuracy, where k is usually set to 5, and is useful for multi-class classification problems. And wrap those code lines in a function named create_model.

```
adam = tf.optimizers.Adam()
model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['top_k_categorical_accuracy'])
return model
```

As the model is done, call create_model function to create an instance of a CNN with the data shape and number classes.

```
def create_model(input_shape, num_classes):
```

```
# Create the model with data shape and number of classes
model = create_model(x_train.shape[1:], num_classes)
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2  (None, 14, 14, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          18496

 max_pooling2d_2 (MaxPoolin  (None, 3, 3, 64)          0
 g2D)

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 128)               73856

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 97668 (381.52 KB)
Trainable params: 97668 (381.52 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

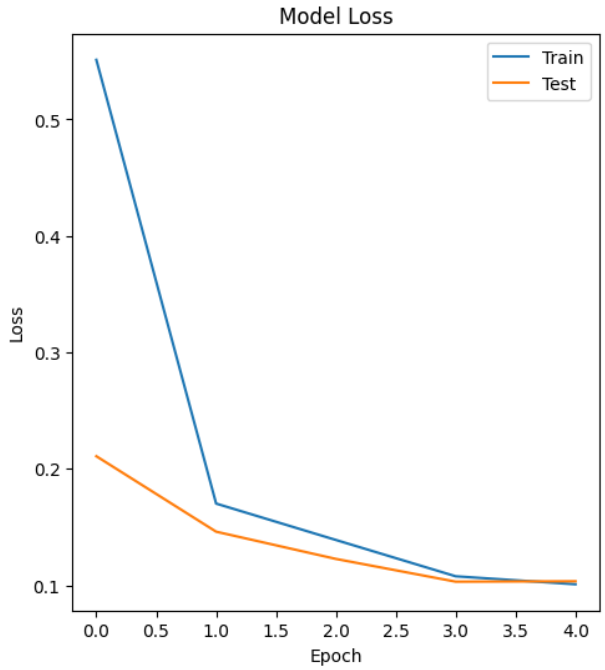
Training:

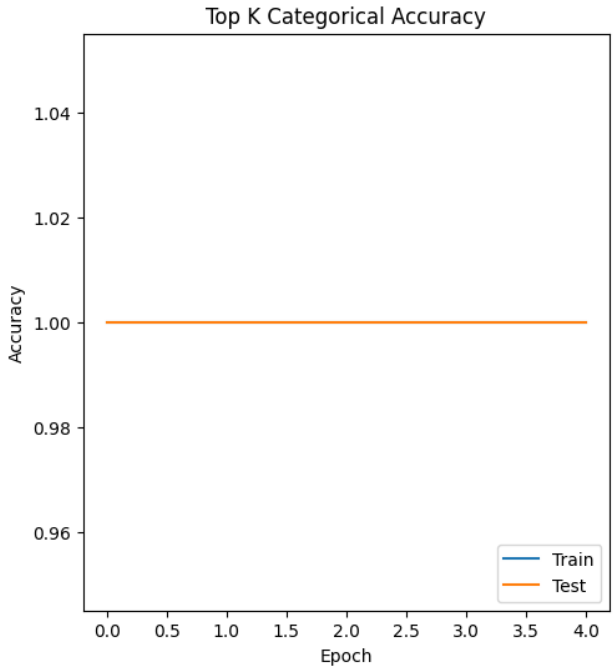
Train the CNN model using the training data (x_train, y_train) with batch_size = 256 and epochs = 5. There ares graphs to visualize the model loss and accuracy during training.

```
batch_size = 256  # You can adjust this value as needed
epochs = 5

history = model.fit(x = x_train, y = y_train, validation_split=0.1, batch_size = batch_size, verbose = 2, epochs = epochs)
```

```
Epoch 1/5
45/45 - 10s - loss: 0.5040 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.2046 - val_top_k_categorical_accuracy: 1.0000 - 10s/epoch - 228ms/step
Epoch 2/5
45/45 - 6s - loss: 0.1511 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.1545 - val_top_k_categorical_accuracy: 1.0000 - 6s/epoch - 142ms/step
Epoch 3/5
45/45 - 9s - loss: 0.1144 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.1503 - val_top_k_categorical_accuracy: 1.0000 - 9s/epoch - 199ms/step
Epoch 4/5
45/45 - 10s - loss: 0.1070 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.1199 - val_top_k_categorical_accuracy: 1.0000 - 10s/epoch - 216ms/step
Epoch 5/5
45/45 - 11s - loss: 0.0891 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.1135 - val_top_k_categorical_accuracy: 1.0000 - 11s/epoch - 249ms/step
```

```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(history.history['top_k_categorical_accuracy'])
plt.plot(history.history['val_top_k_categorical_accuracy'])
plt.title('Top K Categorical Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')

plt.show()
```


Model Loss
Train
Test
Loss
Epoch
0.1
0.2
0.3
0.4
0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Top K Categorical Accuracy
Accuracy
Epoch
Train
Test
0.96
0.98
1.00
1.02
1.04
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Testing:

In this step, the trained model is evaluated on the testing data (x_test, y_test). And print the test accuracy.

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuarcy: {:0.2f}%'.format(score[1] * 100))
```

```
Test accuarcy: 100.00%
```

Store and save

Store the name of classes into a text file named class_names.

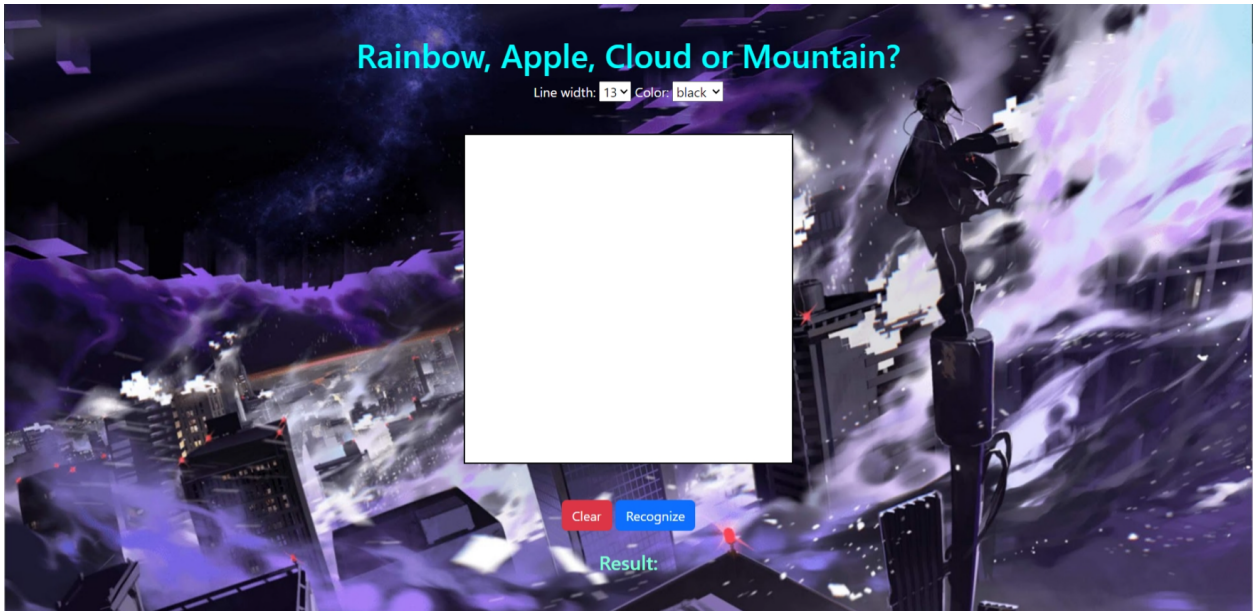
```
with open('class_names.txt', 'w') as file_handler:
    for item in class_names:
        file_handler.write("{}\n".format(item))
```

Save the model with the name ‘cnn.h5’.

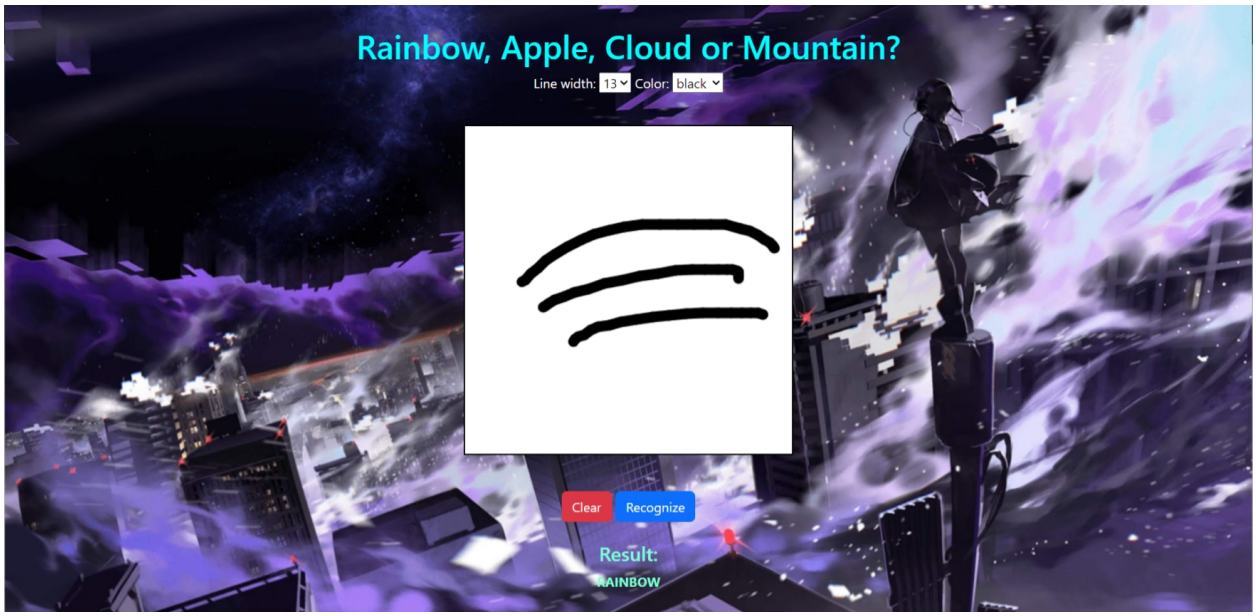
```
model_save_name1 = 'cnn.h5'
path1 = F"/content/gdrive/My Drive/Collab - Artificial Intelligence/model/{model_save_name1}"

model.save(path1)
```

Deploy the model:

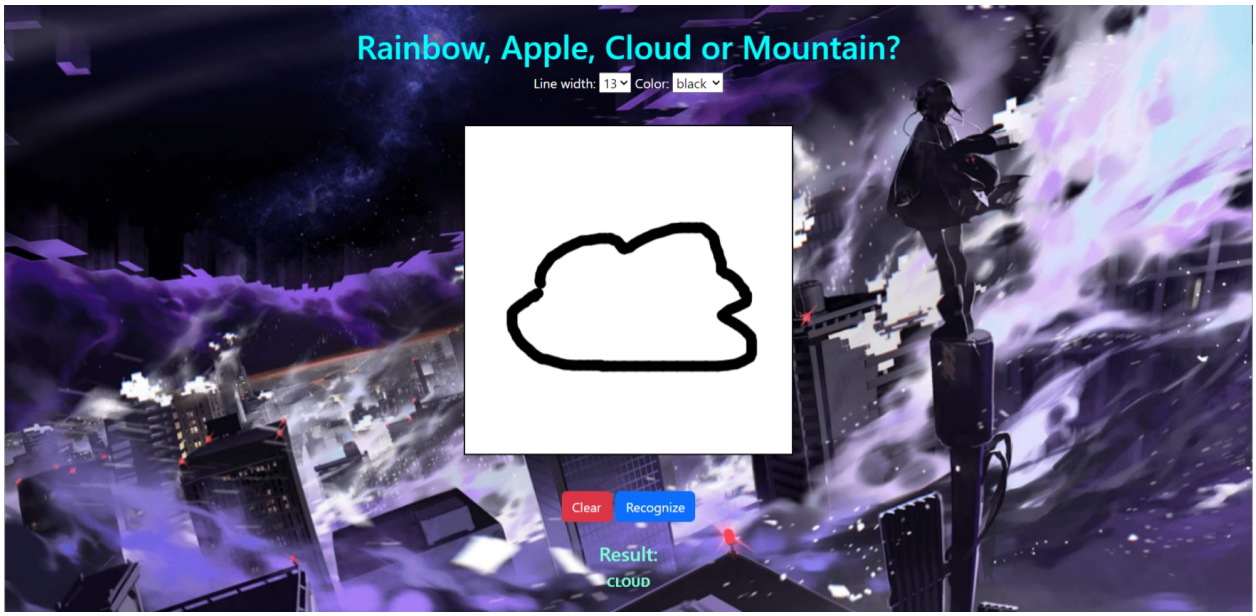
The layout is based on the digital recognition assignment



Try if the model runs correctly:


Rainbow, Apple, Cloud or Mountain?
Line width: 13 Color: black
Clear
Recognize
Result:
RAINBOW


Rainbow, Apple, Cloud or Mountain?
Line width: 13 Color: black
Clear
Recognize
Result:
CLOUD



Github's Link: https://github.com/VOIDkuugeki/Challenge-1-AIML-Quick-Draw