Phan Van Quy - 21020003 - 21CSE

Challenge 1: Build a model with different classes from QuickDraw.io by Google and deploy the model onto a Flask website.

This is my report about the AI challenge project, below will be steps i did to create the project:

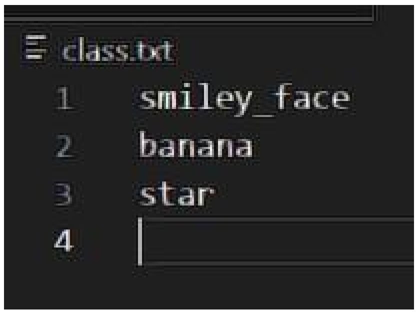
1. Data evaluate:

The first is to prepare data for ML model. I took data from Quick, Draw! Dataset by google creative lab which contain of 50 million drawings across 345 categories that contributed by players of the game. You can browse the recognized drawings on quickdraw.withgoogle.com/data. Each drawing is stored in ndjson format contain of categories:

```
{
  "key_id":"5891796615823360",
  "word":"nose",
  "countrycode":"AE",
  "timestamp":"2017-03-01 20:41:36.70725 UTC",
  "recognized":true,
  "drawing":[[[129,128,129,129,130,130,131,132,132,133,133,133,133,...]]]
}
```

2. Create train model class:

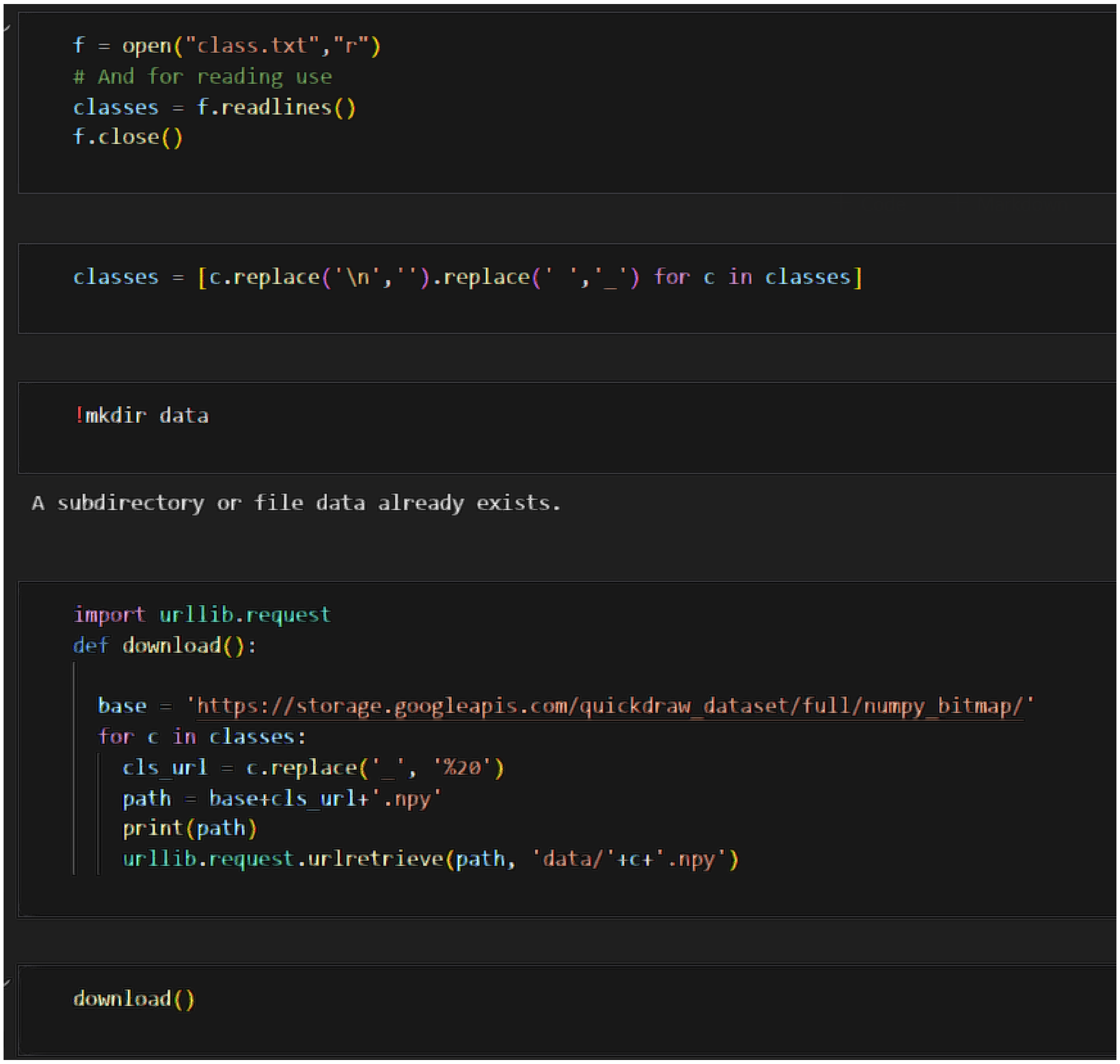
The next thing I did is to chose some out of the class categ in dataset. In this step I created a txt file format to store my 3 categ chosen:



```
class.txt
smiley_face
banana
star

```

Next up is to imported & downloaded the data file:

Phan Van Quy - 21020003 - 21CSE



```
f = open("class.txt","r")
# And for reading use
classes = f.readlines()
f.close()
```

```
classes = [c.replace('\n','').replace(' ','_') for c in classes]
```

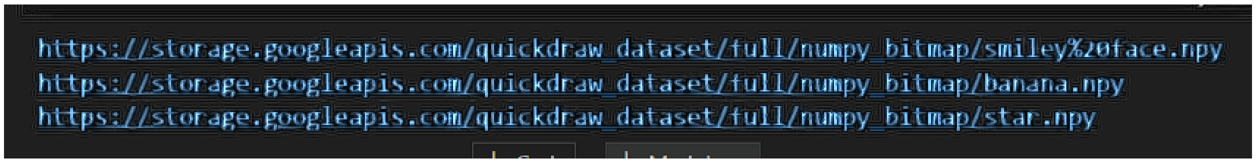
```
!mkdir data
```

```
A subdirectory or file data already exists.
```

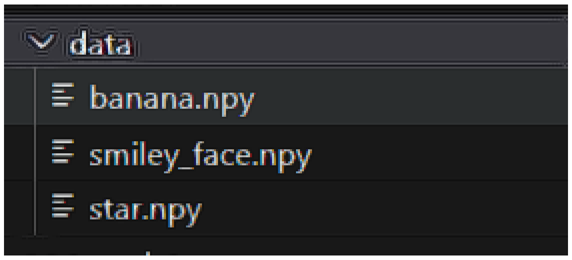
```
import urllib.request
def download():

  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(path)
    urllib.request.urlretrieve(path, 'data/'+c+'.npy')
```

```
download()
```

The process would located the layers and all of photograph refs in desired portion.
All ptgs is kept at ‘data’ folder in root file, with the ref images in npy format.



```
https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/smiley%20face.npy
https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/banana.npy
https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/star.npy
```



```
data
    banana.npy
    smiley_face.npy
    star.npy
```

3. Approach image:

The code below defines load_data function to loads and preprocesses a dataset stored in multiple npy files from root directory. The dataset appears to consist of image data where each image is represented as a 1D array of size 784 (28x28 pixels flattened).

```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
```

- Required libs & modules for the function.

```
import os
import glob
import numpy as np
from tensorflow.keras import layers
from tensorflow import keras
import tensorflow as tf
```

In summary, the function loads image data from multiple files, preprocesses it by selecting a subset and randomizing the order, and then splits it into training and testing sets. The function also returns the list of class names corresponding to the loaded data.

The next is to synchronize the img dataset into one form:

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

Data set to 28x28 format

Then normalize the img value to range (0, 1) and label each layer:

```
# 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)
```

4. Define model:

```
# Define model
model = keras.Sequential()
model.add(layers.Convolution2D(16, (3, 3),
                        padding='same',
                        input_shape=x_train.shape[1:], 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)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
# Train model
adam = tf.optimizers.Adam()
model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['accuracy'])
print(model.summary())
```

In this step I create a CNN model with convolutional, pooling, and fully connected layers for a classification task with three output classes. The Adam optimizer and categorical crossentropy loss function are used for training.
The code would need TensorFlow & Keras libs imported to work.

Phan Van Quy - 21020003 - 21CSE

```
from tensorflow.keras import layers
from tensorflow import keras
import tensorflow as tf
```

Output:

```
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, 3)                 387

...
Trainable params: 97539 (381.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Next is to trained & tested the defined model using the training data (x_train and y_train):

```
history= model.fit(x = x_train, y = y_train, validation_split=0.2, batch_size =
256, verbose=2, epochs=5)
```

Output:

```
Epoch 1/5
30/30 - 5s - loss: 0.5762 - accuracy: 0.7754 - val_loss: 0.2532 - val_accuracy:
0.9208 - 5s/epoch - 169ms/step
Epoch 2/5
30/30 - 5s - loss: 0.2102 - accuracy: 0.9316 - val_loss: 0.1834 - val_accuracy:
0.9417 - 5s/epoch - 153ms/step
Epoch 3/5
30/30 - 5s - loss: 0.1463 - accuracy: 0.9520 - val_loss: 0.1532 - val_accuracy:
0.9458 - 5s/epoch - 152ms/step
Epoch 4/5
30/30 - 4s - loss: 0.1192 - accuracy: 0.9605 - val_loss: 0.1269 - val_accuracy:
0.9563 - 4s/epoch - 135ms/step
Epoch 5/5
30/30 - 4s - loss: 0.1002 - accuracy: 0.9677 - val_loss: 0.1092 - val_accuracy:
0.9625 - 4s/epoch - 129ms/step
```

5. Performance testing:

Next is to evaluates the trained model's performance on the test dataset (x_test and y_test). The evaluation is done using the evaluate method, which computes the model's loss and metrics specified during compilation (in this case, accuracy):

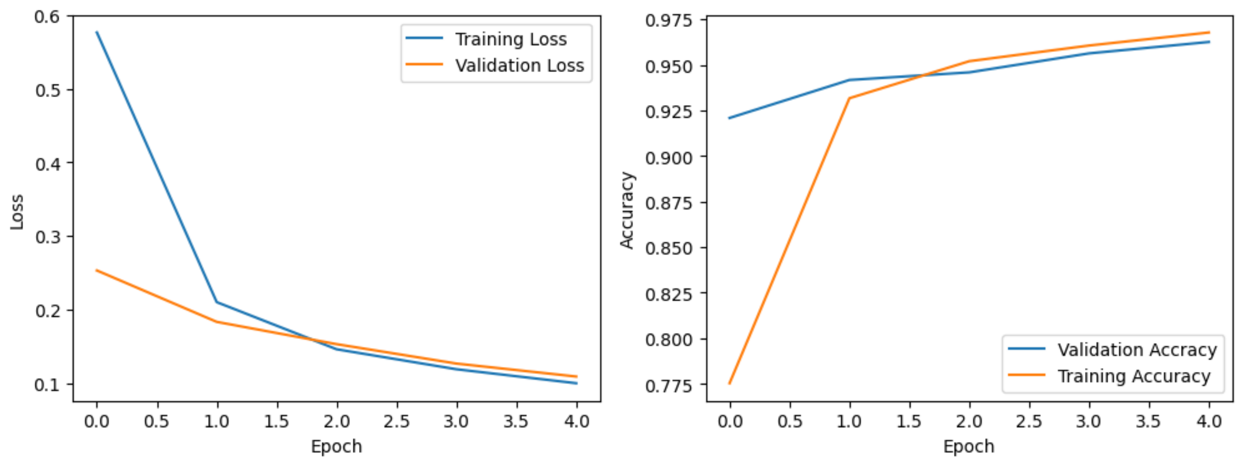
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuarcy: {:0.2f}%'.format(score[1] * 100))
```

```
Test accuarcy: 96.29%
```

Then I create two graphs to assess the parameters of loss and accuracy to retrieves the training loss (loss), validation loss (val_loss), validation accuracy (accuracy), and training accuracy (val_accuracy) values:

```
# Extract training history
loss = history.history['loss']
val_loss = history.history['val_loss']
accuracy = history.history['val_accuracy']
val_accuracy = history.history['accuracy']

# Plot training and validation loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(accuracy, label='Validation Accracy')
plt.plot(val_accuracy, label='Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

Graph outcome:



6. Testing result:

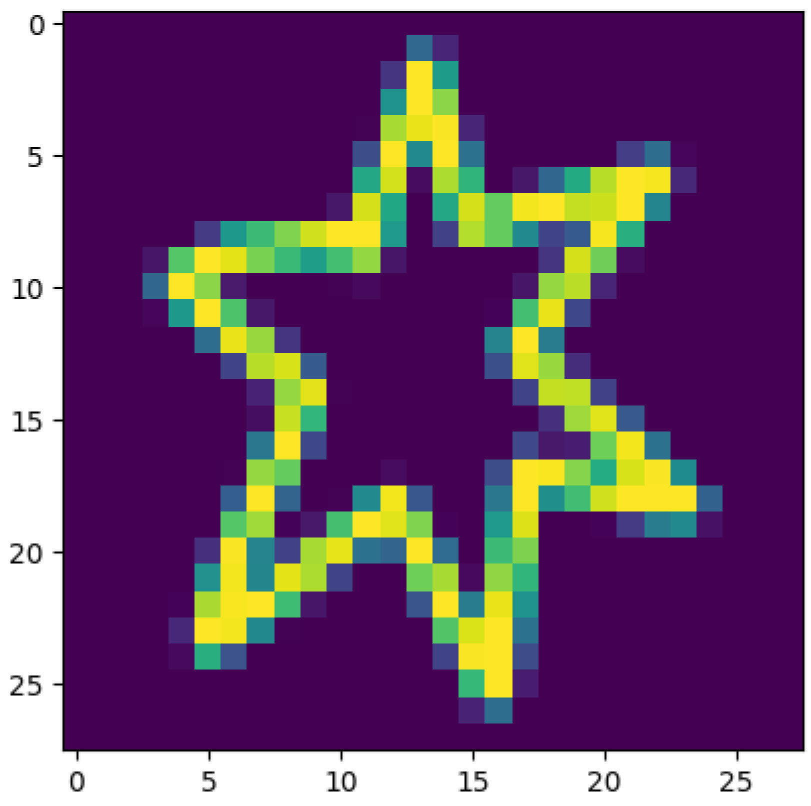
```
import matplotlib.pyplot as plt
from random import randint
%matplotlib inline
idx = randint(0, len(x_test))
img = x_test[idx]
plt.imshow(img.squeeze())
pred = model.predict(np.expand_dims(img, axis=0))[0]
ind = (-pred).argsort()[:5]
```

```
latex = [class_names[x] for x in ind]
print(latex)
```

The code visualize img from the test dataset (x_test). It then uses the trained model to make predictions on the image and retrieves the predicted classes based on the model's output probabilities.
It further translates the class indices into their corresponding class names using the class_names list and prints predicted class names (latex). This provides a quick visual and textual representation of the model's predictions for a randomly selected test image.

Output:

```
1/1 [==============================] - 0s 201ms/step
['star', 'banana', 'smiley_face']
```



7. Deploying model:

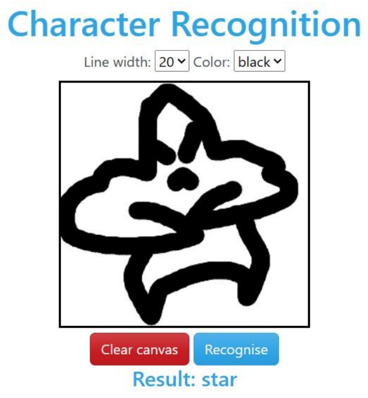
```
app = Flask(__name__)

model = tf.keras.models.load_model('model4.h5')
model.make_predict_function()
with open('class_names.txt') as file:
    class_labels = file.read().splitlines()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/recognize', methods = ['POST'])
def recognize():
    if request.method =='POST':
        print('Receive image and predict what it is')
        data = request.get_json()
        imageBase64 = data['image']
        imgBytes = base64.b64decode(imageBase64)

        with open('temp.jpg', 'wb') as temp:
            temp.write(imgBytes)
        image = cv2.imread('temp.jpg')
        image = cv2.resize(image,(28,28),interpolation = cv2.INTER_AREA)
        image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

        image_prediction = np.reshape(image_gray,(28,28,1))
        image_prediction = (255-image_prediction.astype('float')) /255
        prediction = np.argmax(model.predict(np.array([image_prediction])),axis =
-1)

        #run prd
        return jsonify({
            'prediction':class_labels[prediction[0]],
            'status': True

        })
```

Phan Van Quy - 21020003 - 21CSE

```python
if __name__ == '__main__':
    app.run(debug = True)
```

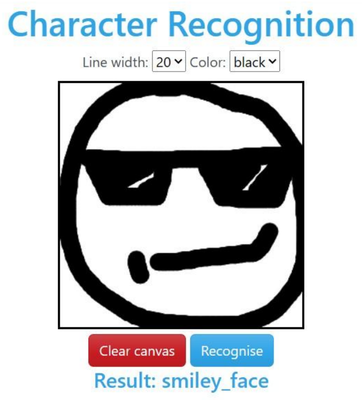
The code compile along with an html file to deploy simple web page similar to Quick, Draw using a pre-trained deep learning model to predict and recognised the img made by users. Users can upload an image or using hand draw, the server will then take the model to predict and return the most relative class label for the content of the image.

Overall result of web deployed:


Character Recognition

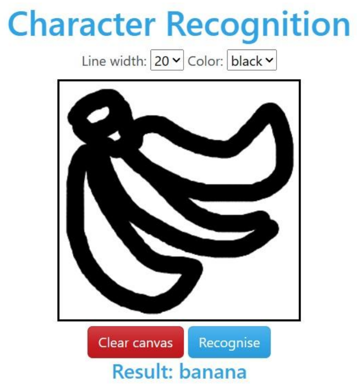
Line width: 20 Color: black

Result: star


Character Recognition

Line width: 20 Color: black

Result: smiley_face


Character Recognition

Line width: 20 Color: black

Result: banana

This is the github link of the project:
To run the project:

- Download & unzip folder
- Installed required libraries
- Open & run the main.py file
- Access via port: {{ * Running on http://127.0.0.1:5000 }}