CLIENT-SIDE

Screens:

Welcome Screen:

```
import { View, Text, StyleSheet, Button } from "react-native";
import React, { useEffect, useRef, useState } from "react";
import { Camera } from "expo-camera";

const WelcomeScreen = ({ navigation }) => {
  let cameraRef = useRef();
  const [hasCameraPermsission, setHasCameraPermission] = useState(null);
  const [image, setImage] = useState(null);
  const [type, setType] = useState(Camera.Constants.Type.back);
  const [flash, setFlash] = useState(Camera.Constants.FlashMode.off);
  const [openCamera, setOpenCamera] = useState(false);
  useEffect(() => {
    (async () => {
      const cameraPermission = await
Camera.requestCameraPermissionsAsync();
      setHasCameraPermission(cameraPermission.status === "granted");
    })();
  });

  const takePicture = async () => {
    setOpenCamera(false);
    if (cameraRef) {
      try {
        const data = await cameraRef.current.takePictureAsync();
        console.log(data);
        setImage(data.uri);
      } catch (e) {
        console.log(e);
      }
    }
  };

  if (hasCameraPermsission === false) {
    return <Text>No access to Camera</Text>;
  }
  return (
    <View style={styles.container}>
      <Text style={styles.text}>PhotoApp</Text>
      <Button
        title="Take Picture"
        onPress={() => navigation.navigate("Camera")}
      ></Button>
    </View>
  );
};
```

```
export default WelcomeScreen;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "white",
    justifyContent: "center",
    alignItems: "center",
  },
  text: {
    fontSize: 50,
  },
});
```

Camera Screen:

```
import { View, Text, StyleSheet } from "react-native";
import React, { useEffect, useRef, useState } from "react";
import { Camera } from "expo-camera";
import Button from "../components/Button";

const CameraScreen = ({ navigation }) => {
  let cameraRef = useRef();
  const [hasCameraPermsission, setHasCameraPermission] = useState(null);
  const [image, setImage] = useState(null);
  const [type, setType] = useState(Camera.Constants.Type.back);
  const [flash, setFlash] = useState(Camera.Constants.FlashMode.off);

  useEffect(() => {
    (async () => {
      const cameraPermission = await
Camera.requestCameraPermissionsAsync();
      setHasCameraPermission(cameraPermission.status === "granted");
    })();
  });

  const takePicture = async () => {
    if (cameraRef) {
      try {
        const data = await cameraRef.current.takePictureAsync({
          quality: 1,
          base64: true,
          exif: false,
        });
        console.log(data);
        setImage(data.uri);
        navigation.navigate("Upload", { base64: data.base64 });
      } catch (e) {
        console.log(e);
      }
    }
  };

  return (
    <View style={styles.container}>
      <Camera
        style={styles.camera}
        type={type}
        FlashMode={flash}
        ref={cameraRef}
      ></Camera>
      <View>
        <Button
          title={"Take a Picture"}
          icon="circle"
          onPress={takePicture}
        ></Button>
      </View>
    </View>
  );
};

export default CameraScreen;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "black",
    justifyContent: "center",
  },
  camera: {
    flex: 1,
    borderRadius: 20,
  },
});
```

Upload Screen:

```
import { Button, StyleSheet, Text, View, Image, Alert } from "react-
native";
import React, { useState } from "react";
import DropDownPicker from "react-native-dropdown-picker";

const UploadScreen = ({ route, navigation }) => {
  console.log("route", route.params.base64);
  //   const [base64image,setBase64Image] = useState(null)
  const base64image = route.params.base64;
  const [upload, setUpload] = useState(false);
  const [open, setOpen] = useState(false);
  const [value, setValue] = useState(null);
  const [items, setItems] = useState([
    { label: "People", value: "People" },
    { label: "Animal", value: "Animal" },
    { label: "Object", value: "Object" },
    { label: "Misc", value: "Misc" },
  ]);

  const uploadHandler = async () => {
    const data = {
      category: value,
      image: base64image,
    };
    // console.log(data);

    const response = await fetch("http://192.168.0.225:4000/save_image", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });

    const json = await response.json();
    // console.log(json);
    setUpload(true);
    Alert.alert("The number is " + json['number']);
  };
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Captured Image:</Text>
      <Image
        style={{ height: 200, width: 200, marginBottom: 10 }}
        source={{ uri: "data:image/jpg;base64," + base64image }}
      ></Image>
      <Button
        style={{ marginTop: 30 }}
        title="Upload Image"
        onPress={uploadHandler}
      ></Button>
    </View>
  );
};

export default UploadScreen;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    bottom: 20,
  },
  text: {
    fontSize: 30,
    padding: 10,
  },
  dropdown: {
    width: "70%",
    alignSelf: "center",
    margin: 10,
  },
});
```

SERVER-SIDE

app.py:

```
import io
import os

from flask import Flask, request, jsonify
import base64, random
from mnist_mentium_classifier.classification import MnistClassifier
import cv2
from PIL import Image


# file_name = "abc.png"
# file_path = os.path.join(os.getcwd(),file_name)
# # img = Image.open(io.BytesIO(base64.decodebytes(bytes(img_data, "utf-8"))))
# # img.save(file_path)
# img = cv2.imread('number_1.jpg', cv2.IMREAD_UNCHANGED)
# # resized = cv2.resize(img, (56, 56), interpolation=cv2.INTER_AREA)
# resized = img
#
app = Flask(__name__)
#
# clf = MnistClassifier()
# print(clf.classify(resized))

@app.route("/")
def hello_world():
    return jsonify({"Server":"Successsa"})
```

```
@app.route('/save_image', methods=['POST'])
def post():
    print("request")
    payload = request.get_json(force=True)
    # print(payload)
    category = payload.get("category")
    img_data = payload.get("image")
    img = Image.open(io.BytesIO(base64.decodebytes(bytes(img_data, "utf-8"))))
    img.save("abc.png")
    # print(img_data)
    file_name = "abc.png"
    file_path = os.path.join(os.getcwd(), file_name)
    img = cv2.imread('abc.png', cv2.IMREAD_UNCHANGED)
    resized = img
    clf = MnistClassifier()
    print("The number sent is:")
    num = int(clf.classify(resized))
    print(num)

    #saving to category
    parent_dir = os.getcwd()
    directory = str(num)
    path = os.path.join(parent_dir, directory)
    if not directory in os.listdir():
        os.mkdir(path)
    # directory = os.getcwd() + "/" + category + "/"
    file_name = ''.join(random.choice('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789') for _ in
range(10)) + ".png"
    file_path = os.path.join(path, file_name)
    img = Image.open(io.BytesIO(base64.decodebytes(bytes(img_data, "utf-8"))))
    img.save(file_path)
    return jsonify({"message": "Save successful","number":str(num), "error": None})

if __name__ == "__main__":
    app.run(host='0.0.0.0', port='4000')
```

Classification.py:

```
import os

import cv2
import numpy as np
from tensorflow import keras

# model = keras.models.load_model('mnist_mentium_classifier/model_preq')

class MnistClassifier:
    def __init__(self):
        # print(os.path.dirname(os.path.realpath(__file__)))
        self.model =
keras.models.load_model(os.path.dirname(os.path.abspath(__file__))+'/model_preq')

    def normalize(self,data):
        return data / 255.0


    def addChannel(self,data):
        return data.reshape((data.shape[0], data.shape[1], data.shape[1], 1))

    def classify(self,img_56):
        # img_56_gray shape = (56,56)
        img_56_gray = cv2.cvtColor(img_56, cv2.COLOR_BGR2GRAY)

        # img_28 shape = (28,28,1)
        img_28 = cv2.resize(img_56_gray, (28, 28), interpolation=cv2.INTER_AREA)

        # input_img shape = (1,28,28)
        input_img = np.expand_dims(img_28, axis=0)

        # sample shape = (1,28,28,1)
        sample = self.addChannel(input_img)

        # normalizing data
        sample = self.normalize(sample)

        #prediction
        prediction = self.model.predict(sample)

        number = np.argmax(prediction)
```

```
return number
```

```
import tensorflow as tf
import keras
import numpy as np
import matplotlib.pyplot as plt
import time
import pandas as pd
```

Importing MNIST dataset

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data(path="mnist.npz")
```

Checking if dataset is balanced - we can see the distribution of classes is even

```
pd.Series(y_train).value_counts()
```

```
1    6742
7    6265
3    6131
2    5958
9    5949
0    5923
6    5918
8    5851
4    5842
5    5421
dtype: int64
```

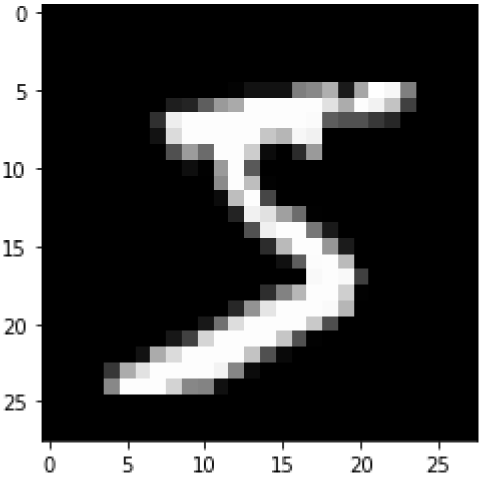
```
pd.Series(y_test).value_counts()
```

```
1    1135
2    1032
7    1028
3    1010
9    1009
4     982
0     980
8     974
6     958
5     892
dtype: int64
```

Sampling one image/sample from the dataset

```
plt.imshow(X_train[0],cmap='gray')
```

```
    <matplotlib.image.AxesImage at 0x7f8d314b35d0>
```



Now we are going to normalize the image(divide each pixel by 255.0) so that training happens faster (0 to 255 ----> 0 to 1)

```
def normalize(data):
  return data/255.0
```

```
X_train = normalize(X_train)
X_test = normalize(X_test)
```

Checking the shape of input image

```
X_train[0].shape
```

```
    (28, 28)
```

Mnist dataset is grayscale - > so we add a channel

```
def addChannel(data):
  return data.reshape((data.shape[0],data.shape[1],data.shape[1],1))
```

```
X_train = addChannel(X_train)
X_test = addChannel(X_test)
```

```
X_train[0].shape
```

```
    (28, 28, 1)
```

Creating a simple model to classify image to 10 categories

```
from keras import Sequential
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import Dense
from tensorflow.keras.utils import plot_model
from keras.layers import MaxPool2D
from keras.layers import Dropout
from tensorflow.keras.utils import to_categorical
from keras.layers  import BatchNormalization
```

```
model = Sequential([])
model.add(Conv2D(32 ,(3,3),activation = 'relu',input_shape=(28,28,1)))
model.add(MaxPool2D((2, 2)))
model.add(Conv2D(48, (3,3), activation='relu'))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

MODEL SUMMARY

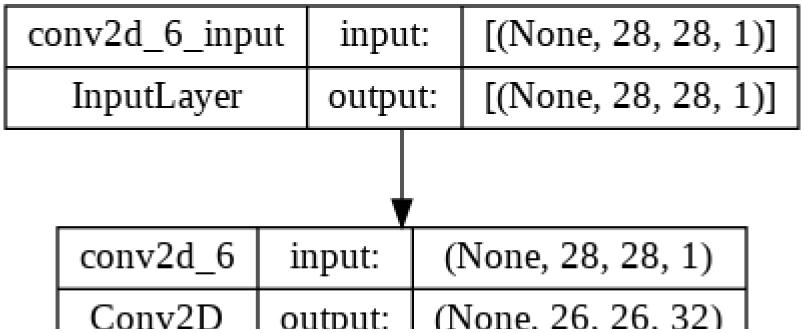
```
model.summary()

new_mode=Model(input=model.inputlayer,output=model.layers[-2])
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d_6 (MaxPooling  (None, 13, 13, 32)       0         
 2D)                                                             
                                                                 
 conv2d_7 (Conv2D)           (None, 11, 11, 48)        13872     
                                                                 
 max_pooling2d_7 (MaxPooling  (None, 5, 5, 48)         0         
 2D)                                                             
                                                                 
 dropout_3 (Dropout)         (None, 5, 5, 48)          0         
                                                                 
 flatten_3 (Flatten)         (None, 1200)              0         
                                                                 
 dense_6 (Dense)             (None, 500)               600500    
                                                                 
 dense_7 (Dense)             (None, 10)                5010      
```

```
=================================================================
Total params: 619,702
Trainable params: 619,702
Non-trainable params: 0
_________________________________________________________________
```

```
plot_model(model,show_shapes=True)
```



```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy',tf.keras.m
```

Defining a callback to stop training when there is no improvement in val_loss

```
callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience = 5
)
```

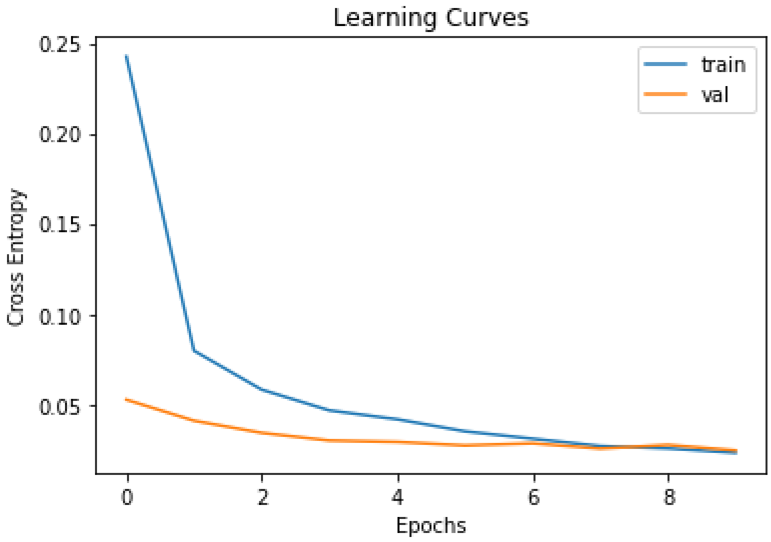
Since I am using categorical cross entropy instead of sparse,I need to one hot encode the labels

```
y_train  = to_categorical(y_train)
y_test  = to_categorical(y_test)
```

```
history = model.fit(X_train,y_train,epochs=10,batch_size = 128,verbose=2,validation_split = 0
```

```
Epoch 1/10
422/422 - 3s - loss: 0.2425 - accuracy: 0.9259 - auc_3: 0.9952 - val_loss: 0.0526 - val_
Epoch 2/10
422/422 - 2s - loss: 0.0797 - accuracy: 0.9750 - auc_3: 0.9989 - val_loss: 0.0409 - val_
Epoch 3/10
422/422 - 2s - loss: 0.0581 - accuracy: 0.9818 - auc_3: 0.9992 - val_loss: 0.0342 - val_
Epoch 4/10
422/422 - 2s - loss: 0.0466 - accuracy: 0.9849 - auc_3: 0.9994 - val_loss: 0.0299 - val_
Epoch 5/10
422/422 - 2s - loss: 0.0417 - accuracy: 0.9870 - auc_3: 0.9994 - val_loss: 0.0291 - val_
Epoch 6/10
422/422 - 2s - loss: 0.0350 - accuracy: 0.9891 - auc_3: 0.9996 - val_loss: 0.0272 - val_
Epoch 7/10
422/422 - 2s - loss: 0.0309 - accuracy: 0.9900 - auc_3: 0.9997 - val_loss: 0.0284 - val_
Epoch 8/10
422/422 - 2s - loss: 0.0269 - accuracy: 0.9909 - auc_3: 0.9997 - val_loss: 0.0255 - val_
Epoch 9/10
422/422 - 2s - loss: 0.0256 - accuracy: 0.9917 - auc_3: 0.9997 - val_loss: 0.0275 - val_
Epoch 10/10
422/422 - 2s - loss: 0.0231 - accuracy: 0.9924 - auc_3: 0.9998 - val_loss: 0.0245 - val_
```

```
plt.title('Learning Curves')
plt.xlabel('Epochs')
plt.ylabel('Cross Entropy')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.legend()
plt.show()
```



Checking Testing metrics

```
loss, accuracy,auc = model.evaluate(X_test, y_test, verbose=0)
print(f'Accuracy: {accuracy*100}')
print(f'AUC score: {auc*100}')
```

```
Accuracy: 99.37000274658203
AUC score: 99.97484087944031
```