Running neural networks on Android devices

Panyala Sainath Reddy (B20CS040)
Neerukonda Venkata Srinivas (B20CS037)

Problem Statement:

To develop an API that can switch between AI models depending on factors like the available resources on the device, etc.

Motivation:

On-device processing of complex data and complex tasks requires machine learning algorithms, such as image classification, natural language processing, and speech recognition. By running these algorithms on the device, users should be able to enjoy the benefits of a fast, and personalized experience without relying on cloud-based servers, which would be slow and unavailable in real-time, so we need to run different ML models for the same task depending upon resource availability on the device.

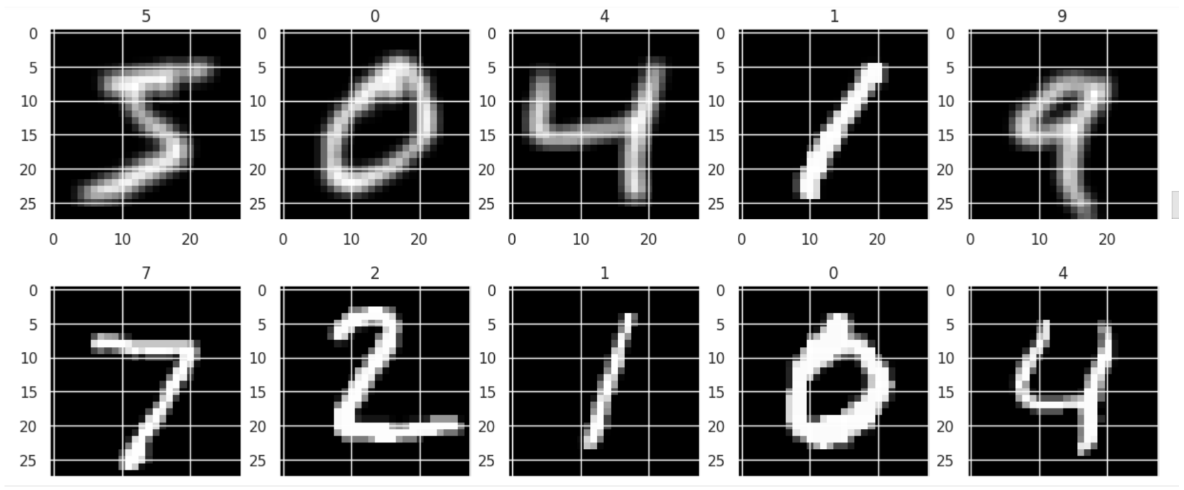
Proposed Methodology:

As we can't develop and deploy different applications for different devices, it would be complex. and would be a time-consuming process. The hardware specifications of different devices, such as CPU processing power, RAM, and GPU, have a direct impact on their AI score (ability to run neural networks on the device based on the memory it takes and the time it takes to run), which can be known by using the AI benchmark from NNAPI. Running different models on devices with varying specifications (different ranges of AI scores) allows for the most efficient use of the device's resources. The classification task can be performed more efficiently, with reduced latency and minimal resource consumption, by selecting the appropriate model based on the available hardware resources. This can result in a better user experience and higher performance. Using pre-trained models to train some task-based datasets and converting them to a liter version would be very useful to minimize resources. Moving further, NNAPI could be used to get better constraint-oriented applications.

NNAPI: To satisfy resource constraints, we have used NNAPI to run the Tflite file on mobile devices (Android) to predict the output for the given input image. As Decision making by neural networks involves a lot of computation, we need to minimize power consumption and memory usage to keep our device healthy. Tflite would be the lite version, which does the same thing as the original model but with less consumption of resources.

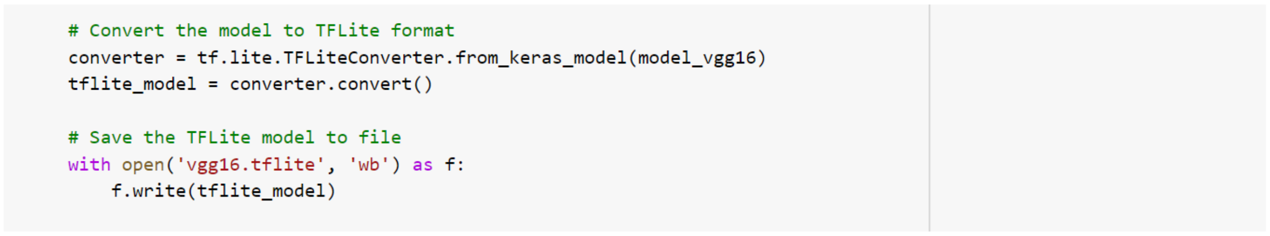
Task and Dataset:

The task we have chosen is the classification of images containing numbers into natural numbers. Classifying natural number images can be a useful task in various applications, such as digit recognition for check processing, postal automation, and automatic license plate recognition. For training the machine learning model, we have chosen the MNIST dataset, which is a collection of handwritten digit images used for training and testing image processing systems and machine learning models. The dataset consists of 60,000 training images and 10,000 test images. Each image is a grayscale image of 28x28 pixels, representing a handwritten digit from 0 to 9. We can develop an API on a model that is trained on this dataset, as it contains various types and styles of writing. It would be able to recognize any type of handwritten input. We have 10 classes into which we have to classify the given input. Some data points from test data and their labels

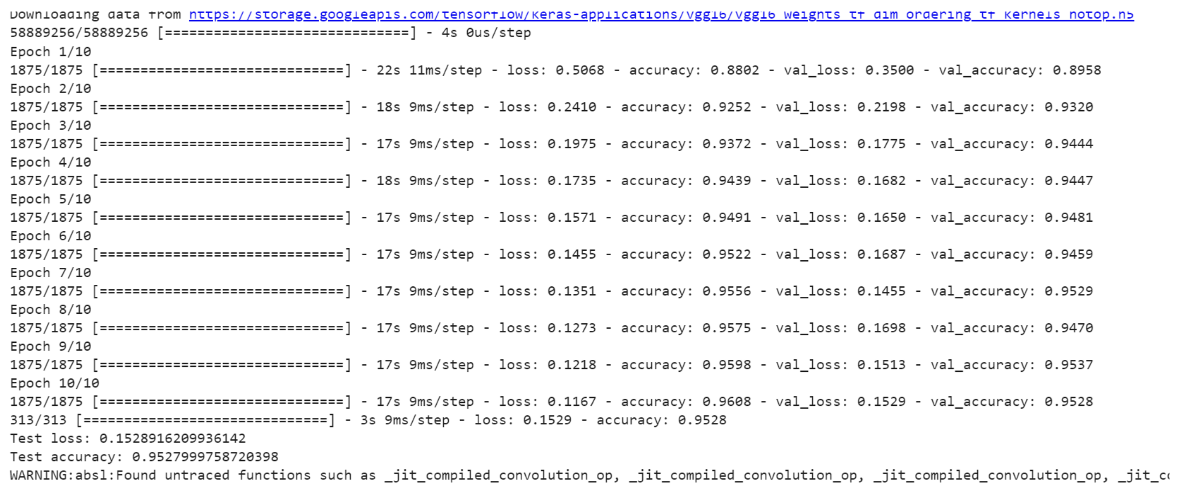


MODELS:

1. VGG-16: It is a CNN architecture for image classification, it consists of 16 layers and is trained on the ImageNet dataset, which has over 1 million images from 1,000 classes. The VGG16 model has achieved state-of-the-art performance on many image classification tasks, and its architecture has inspired many other models, such as VGG19, ResNet, and Inception.
It is trained on the same dataset to classify classes in the ImageNet dataset. It has achieved around 95% accuracy on this dataset, it is a very complex model compared to other models so it would have a bigger size. So, we have converted it into a .tflite file (a lite version) to deploy our model on Android devices. Generally, the pre-trained model takes around 600MB due to which mobiles may hang out. Here is how we converted our model to a light version.

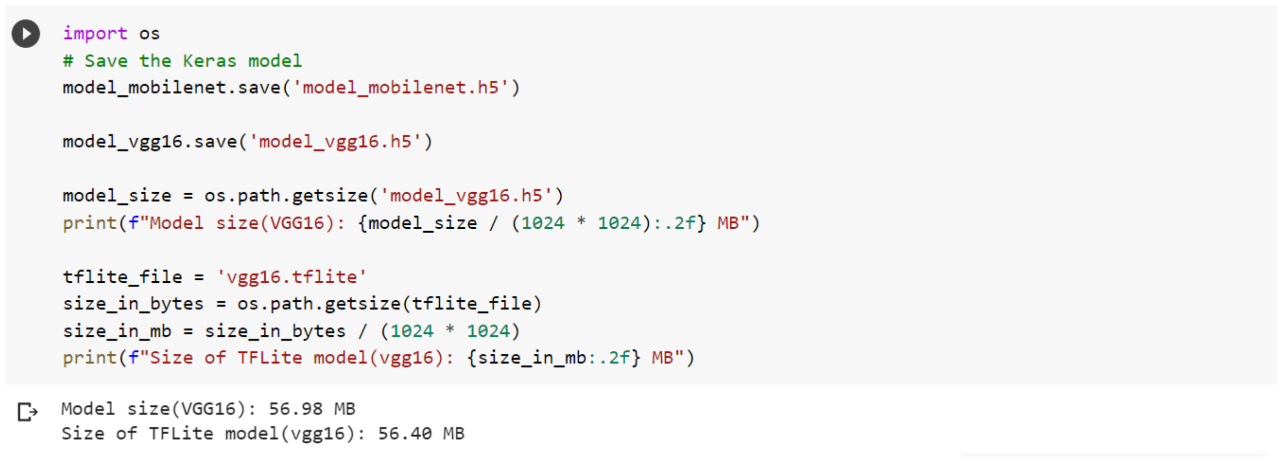


```
# Convert the model to TFLite format
converter = tf.lite.TFLiteConverter.from_keras_model(model_vgg16)
tflite_model = converter.convert()

# Save the TFLite model to file
with open('vgg16.tflite', 'wb') as f:
    f.write(tflite_model)
```



```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 4s 0us/step
Epoch 1/10
1875/1875 [==============================] - 22s 11ms/step - loss: 0.5068 - accuracy: 0.8802 - val_loss: 0.3500 - val_accuracy: 0.8958
Epoch 2/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.2410 - accuracy: 0.9252 - val_loss: 0.2198 - val_accuracy: 0.9320
Epoch 3/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1975 - accuracy: 0.9372 - val_loss: 0.1775 - val_accuracy: 0.9444
Epoch 4/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.1735 - accuracy: 0.9439 - val_loss: 0.1682 - val_accuracy: 0.9447
Epoch 5/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1571 - accuracy: 0.9491 - val_loss: 0.1650 - val_accuracy: 0.9481
Epoch 6/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1455 - accuracy: 0.9522 - val_loss: 0.1687 - val_accuracy: 0.9459
Epoch 7/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1351 - accuracy: 0.9556 - val_loss: 0.1455 - val_accuracy: 0.9529
Epoch 8/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1273 - accuracy: 0.9575 - val_loss: 0.1698 - val_accuracy: 0.9470
Epoch 9/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1218 - accuracy: 0.9598 - val_loss: 0.1513 - val_accuracy: 0.9537
Epoch 10/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1167 - accuracy: 0.9608 - val_loss: 0.1529 - val_accuracy: 0.9528
313/313 [==============================] - 3s 9ms/step - loss: 0.1529 - accuracy: 0.9528
Test loss: 0.1528916209936142
Test accuracy: 0.9527999758720398
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_c
```

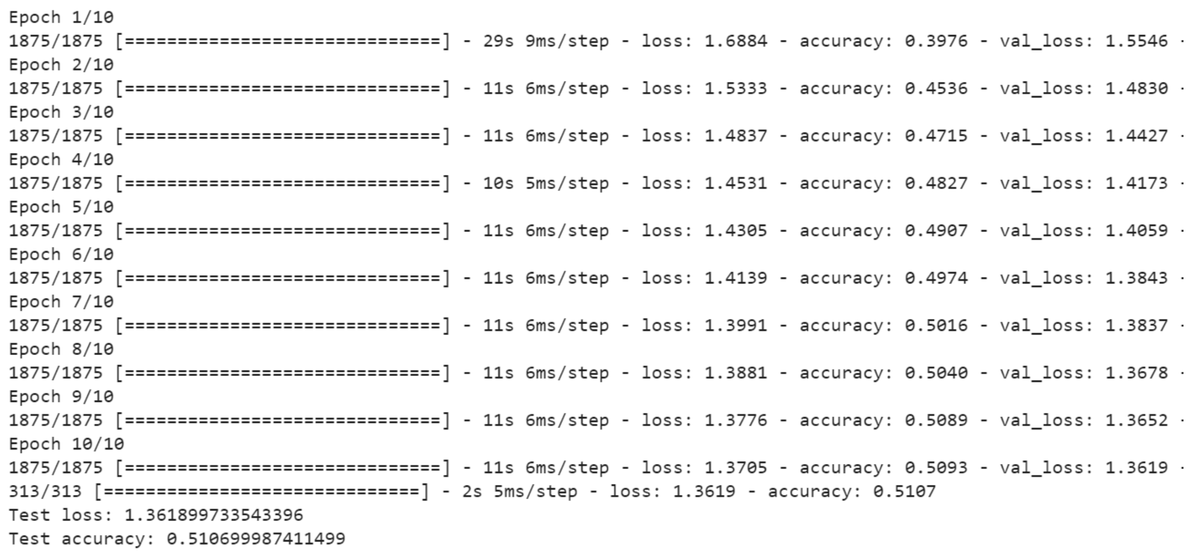
The memory that is taken by the model and its lite versions (Tflite):



```
import os
# Save the Keras model
model_mobilenet.save('model_mobilenet.h5')

model_vgg16.save('model_vgg16.h5')

model_size = os.path.getsize('model_vgg16.h5')
print(f"Model size(VGG16): {model_size / (1024 * 1024):.2f} MB")

tflite_file = 'vgg16.tflite'
size_in_bytes = os.path.getsize(tflite_file)
size_in_mb = size_in_bytes / (1024 * 1024)
print(f"Size of TFLite model(vgg16): {size_in_mb:.2f} MB")
```

```
Model size(VGG16): 56.98 MB
Size of TFLite model(vgg16): 56.40 MB
```

2. MOBILENET: It is a pre-trained model on object classification, CNN, which is trained on the ImageNet dataset, which contains millions of images belonging to thousands of different classes. The MobileNet architecture was designed to be efficient and lightweight, making it well-suited for use in mobile and embedded devices with limited computational resources. So we can train our dataset (MNIST DATA) on mobilenet, which contains grayscale images of siz28X28X1, but we need to give three-channel input for mobilenet model we have tripled the single channel over others to fit into the model After training our data with the dataset, we have converted our Tensorflow model to its lite version to be deployed on mobile (Android) devices. We can observe a decrease in the size of the model that we have to run to classify the given image. We have got up to 98% accuracy using PyTorch and up to 97% accuracy on TensorFlow.

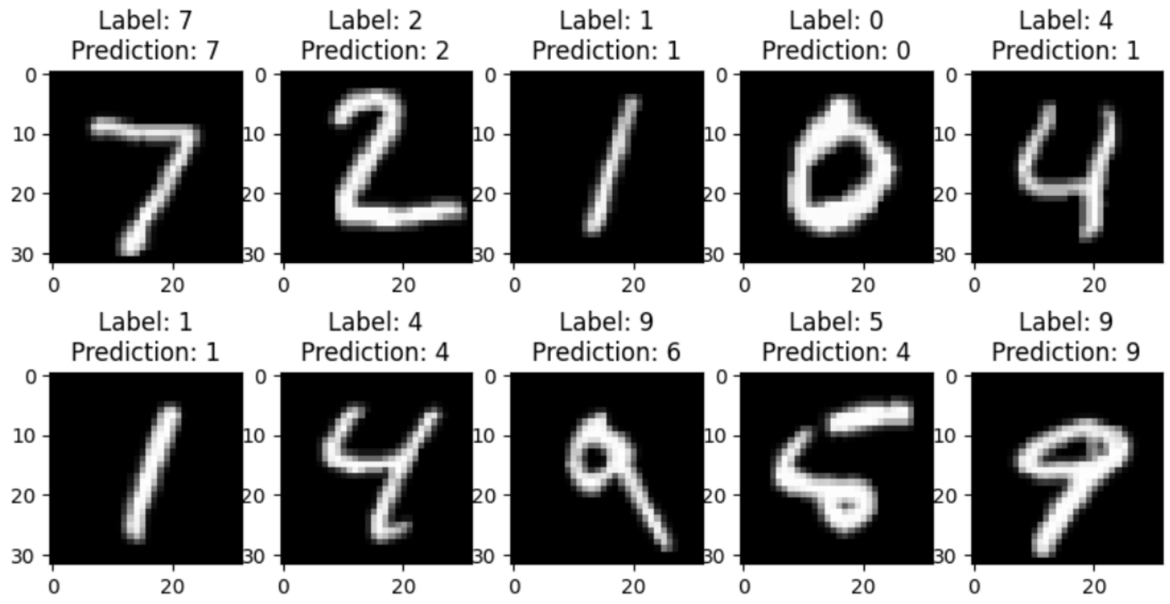


```
Epoch 1/10
1875/1875 [==============================] - 29s 9ms/step - loss: 1.6884 - accuracy: 0.3976 - val_loss: 1.5546 -
Epoch 2/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.5333 - accuracy: 0.4536 - val_loss: 1.4830 -
Epoch 3/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.4837 - accuracy: 0.4715 - val_loss: 1.4427 -
Epoch 4/10
1875/1875 [==============================] - 10s 5ms/step - loss: 1.4531 - accuracy: 0.4827 - val_loss: 1.4173 -
Epoch 5/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.4305 - accuracy: 0.4907 - val_loss: 1.4059 -
Epoch 6/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.4139 - accuracy: 0.4974 - val_loss: 1.3843 -
Epoch 7/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.3991 - accuracy: 0.5016 - val_loss: 1.3837 -
Epoch 8/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.3881 - accuracy: 0.5040 - val_loss: 1.3678 -
Epoch 9/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.3776 - accuracy: 0.5089 - val_loss: 1.3652 -
Epoch 10/10
1875/1875 [==============================] - 11s 6ms/step - loss: 1.3705 - accuracy: 0.5093 - val_loss: 1.3619 -
313/313 [==============================] - 2s 5ms/step - loss: 1.3619 - accuracy: 0.5107
Test loss: 1.361899733543396
Test accuracy: 0.510699987411499
```

Converting our model to a lite version

```
[12] import os
     model_size = os.path.getsize('model_mobilenet.h5')
     print(f"Model size(MOBILENET): {model_size / (1024 * 1024):.2f} MB")
     tflite_file = 'model_mobilenet.tflite'
     size_in_bytes = os.path.getsize(tflite_file)
     size_in_mb = size_in_bytes / (1024 * 1024)
     print(f"Size of TFLite model(MOBILENET): {size_in_mb:.2f} MB")

     Model size(MOBILENET): 14.09 MB
     Size of TFLite model(MOBILENET): 12.71 MB
```

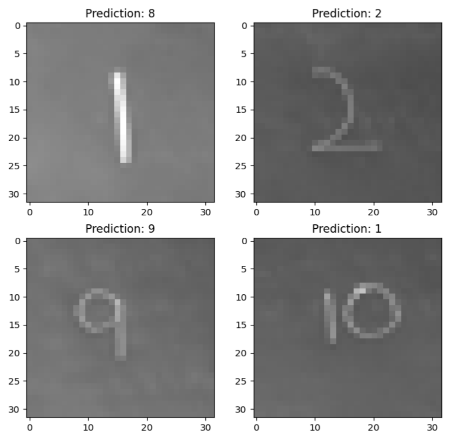
Testing our models on handwritten inputs is mandatory, as we would be giving our input directly to the API without any preprocessing, and testing on our data would also be mandatory. some data samples and our model predictions for them.

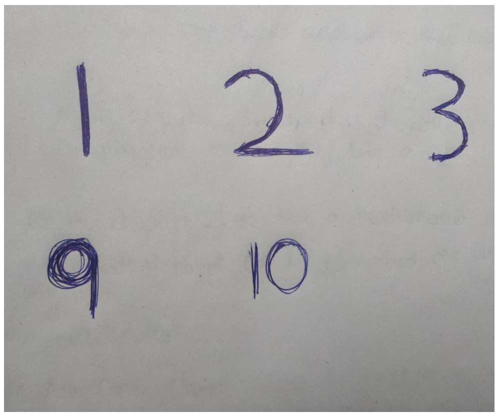


If data is given as handwritten input, it might not give a good result for a direct colored image, as we have only grayscale images in our dataset. At first, we have to resize the image to 32 x 32, as we have trained our model with images of that size.

We converted the input image into grayscale, tripled the channel, and inverted the color of pixels to have better accuracy, similar to the way we have trained our model. some of the predictions for manual, handwritten inputs by both models

Results:

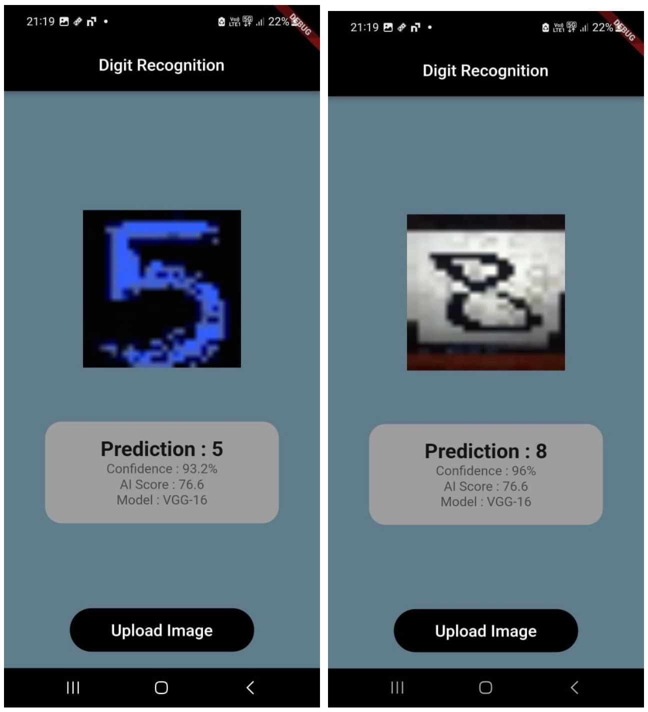




Digit Recognition App:

We have made the app using **Flutter.** On opening the app, we are detecting the device name using the **device_name** flutter package, and depending on the name of the device, using the **ai-benchmark** API, we are detecting the AI-Score of the device. We have set a threshold of 60. If the score is greater than 60, we have considered it a mobile device with a good amount of resources available to run large models (here VGG16), and if the score is less than 60, then we consider it a mobile device with fewer resources to run a large model. So we are using MobileNet. Here, one device has a value of 55.3 and another has a value of 76.6. These are clearly seen in the demo video.

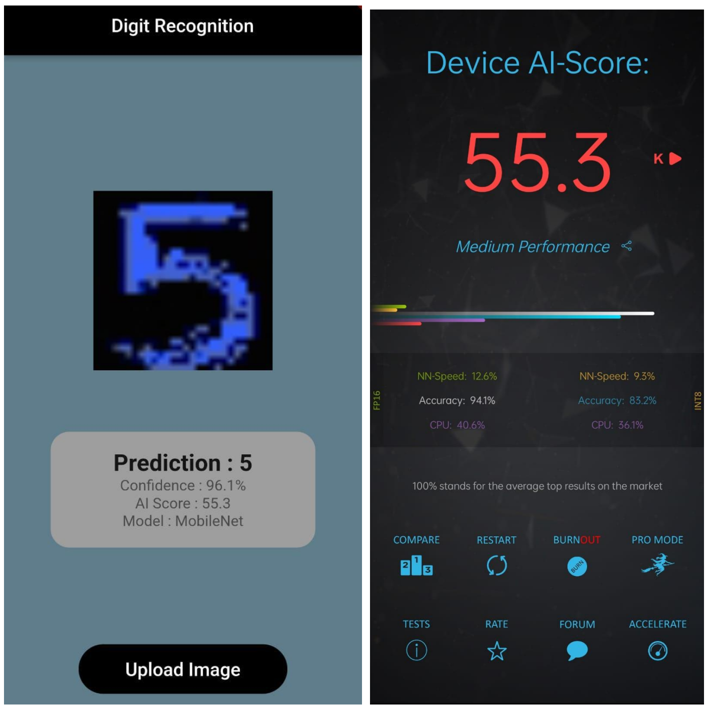
We can upload the image either by using the camera or can take it from the gallery, After the image is uploaded, depending on the device's AI-Score, the image will be recognized with the respective model corresponding to the device.
And then the image, prediction, confidence, AI-Score of the device, and model used will be displayed on the app.



Prediction : 5
Confidence : 93.2%
AI Score : 76.6
Model : VGG-16

Prediction : 8
Confidence : 96%
AI Score : 76.6
Model : VGG-16

Images taken from Oneplus Nord (AI-score: 76.6)
(Model-used: VGG-16)



Prediction : 5
Confidence : 96.1%
AI Score : 55.3
Model : MobileNet

NN-Speed: 12.6% Accuracy: 94.1% CPU: 40.6%
NN-Speed: 9.3% Accuracy: 83.2% CPU: 56.1%

Image taken from Realme 7 Pro & its AI-score (55.3)
(Model-used: MobileNet)

Conclusion:

We can use our app very efficiently in terms of memory and time if we use different AI models for different devices (devices with different mobile AI scores). We have used only two models with our devices, as we were unable to find a large range of devices. We can use many models in the app and deploy them for large-scale requirements.

Resources:

1. https://arxiv.org/pdf/1810.01109.pdf AI Benchmark: Running Deep Neural Networks on Android Smartphones
2. https://developer.android.com/ndk/guides/neuralnetworks Neural Networks API
3. https://www.kaggle.com/code/xhlulu/training-mobilenet-v2-in-4-min Kaggle for Training MobileNet v2
4. https://www.kaggle.com/code/amithasanshuvo/mnist-classification-using-tensorflowMNIST classification using TensorFlow
5. https://pub.dev/packages/device_name Flutter package for getting the device name
6. https://ai-benchmark.com/ranking.html Details about Benchmark and scores

The models used for classification are MobileNet and vGG16 (.tflite files).