CMPE 255 - Final Project Report

(Application Option)

Devansh Modi[#1], Abhighna Kapilavai[#2]

[#]*Department of Software Engineering*
San Jose State University
1 Washington Sq, San Jose, CA 95129

I. Selected Problem

This project is an Application based idea following the Option #2 for the course final project.

II. Objective

Indoor scene recognition is a challenging open problem in high level vision. Most scene recognition models that work well for outdoor scenes perform poorly in the indoor domain. The main difficulty is that while some indoor scenes (e.g. corridors) can be well characterized by global spatial properties, others (e.g., bookstores) are better characterized by the objects they contain. More generally, to address the indoor scenes recognition problem we need a model that can exploit local and global discriminative information.

The objective of this project is to support indoor scene recognition for kitchen remodeling domain. It is a novel idea as there exists no datasets for kitchen layouts today, yet there is an ever-increasing need for kitchen remodeling services.

III. Key Techniques

This project pipeline uses the following key techniques:
In other words, the project aims to build:

1. A complete data mining pipeline application.
2. Data labeling processes (AWS SageMaker and S3)
3. Multiple image classification models (Google Colab)
4. Hosted inference infrastructure. (AWS EC2)
5. Mobile Application (Android / Tensorflow Lite)
6. Model Acceleration (Tensorflow Accelerations using GPU)

IV. Datasets

The dataset used for this project included a custom built and labeled data for classifying countertop material and kitchen layout.

Label Classifiers include:

- Wood Countertop

- Stone Countertop
- L-Shaped Layout
- U-Shaped Layout
- One-Wall Layout

As per data from major retailers, these classifiers comprise over 90% of the single family home kitchens in the United States.

In order to classify kitchen appliances and kitchen objects, we use MIT Indoor Scene Recognition open dataset and COCO Object Recognition dataset.

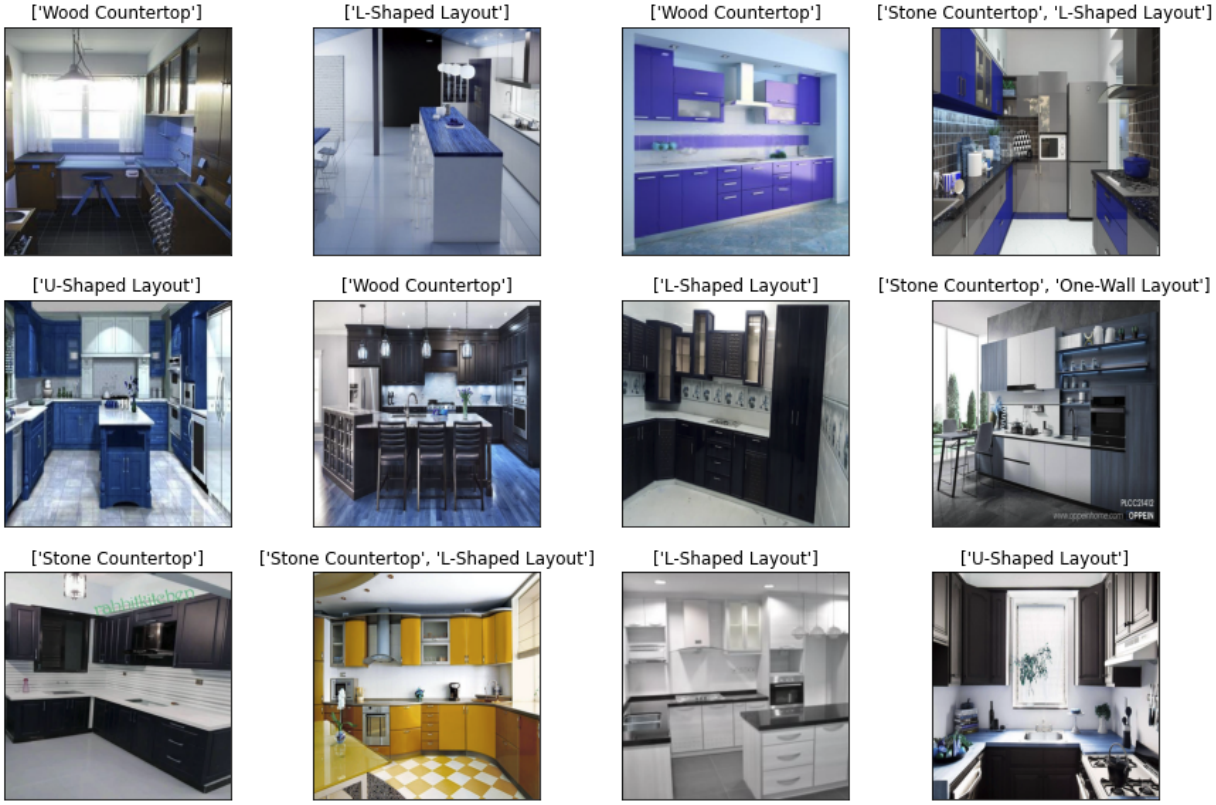


Figure: Kitchen Layout Labels

Figure: Cabinet and Objects

Data Preprocessing included steps such as:

- Image Resizing
- Shuffling of dataset
- Filtering of unsuitable images.
- Multiple iterations of label verification.

Due to the lack of data for kitchen layouts, we manually prepare a dataset of over 1200 images with a 20% split for validation. We used the following steps for preparing the manual dataset.

1. Compilation of royalty free kitchen images with a high resolution.
    a. Tools used: Envato Elements and Google Images.
2. Upload data to AWS S3
3. Start Data Labeling jobs on AWS SageMaker

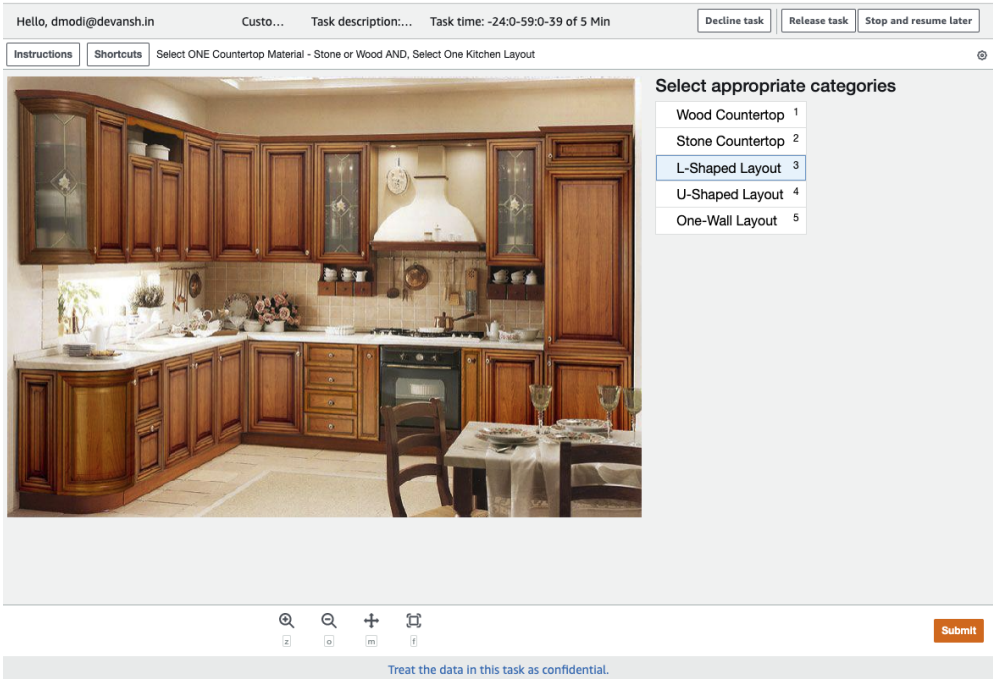


Figure: Data Labeling Job Created in AWS SageMaker

4. Completed manual data labeling with the help of both team members in a month's time.
5. Performed a second iteration of label verification.
6. Generated label manifest.
7. Resized and preprocessed the images in Google Colab to structure data in Tensorflow's friendly form. We resized the images to 512 x 512..

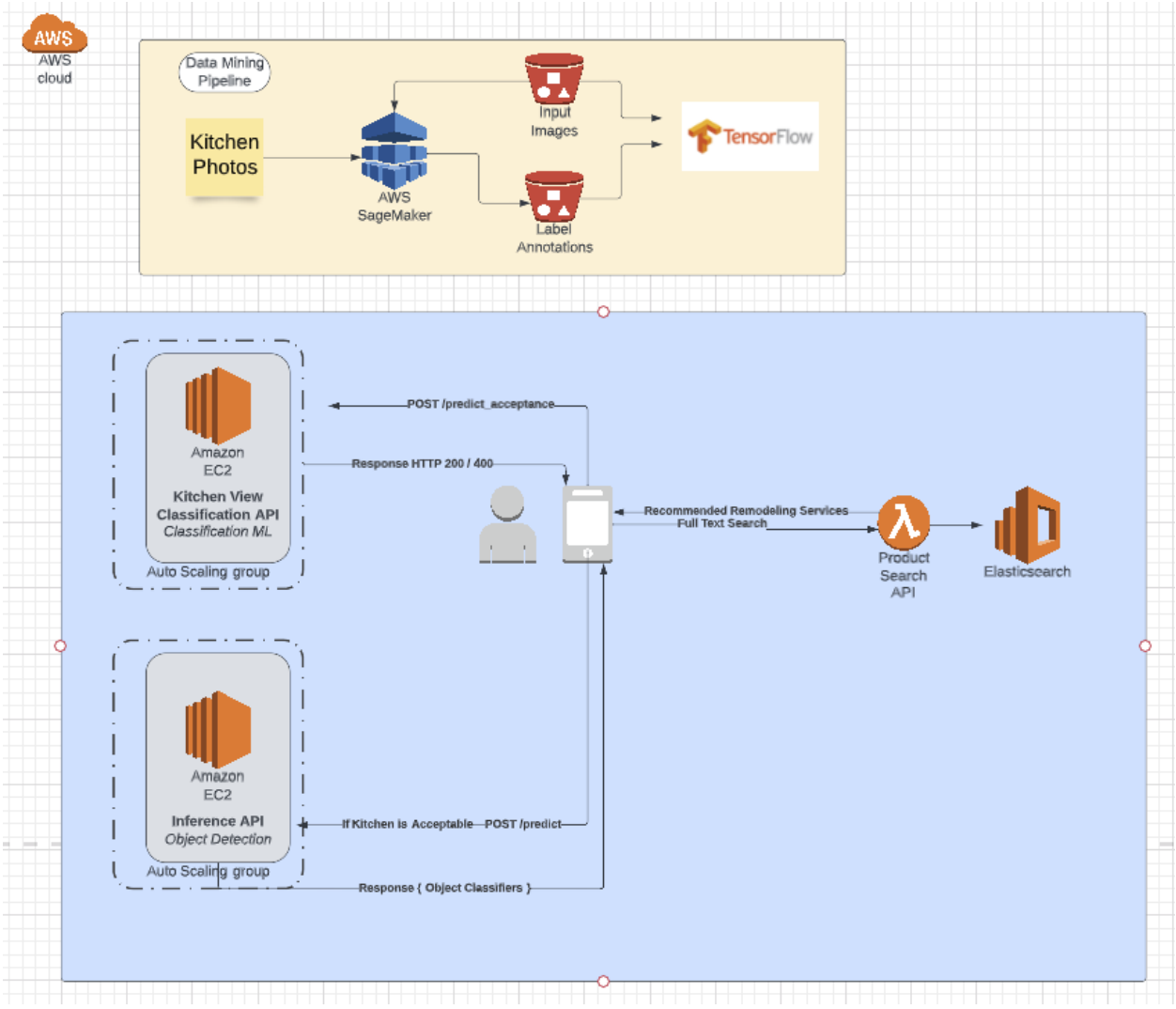
V. Architecture

The architecture of our platform is built using AWS and open-source components.

We have designed our inference and application to be scalable in both input data images and to be able to handle a large volume of inference requests from the mobile application. This is possible because, we are using AWS Auto Scaling group for adding identical clones of EC2 instances based on high resource usage in the servers.

We also use Android for adding Tensorflow Lite locally so we can offer increased performance to the users.

The architecture also uses AWS S3 for storing the model training and validation dataset securely and in an easily accessible manner. Loading the data from S3 can be done simply by using AWS CLI, which we have demonstrated in our Jupyter Notebook.



VI. MODELING

We used multiple models to compare performance. Here is the list:

1. Self-built CNN Model
2. EfficientNetB0
3. MobileNetV2
4. AWS SageMaker Auto Algorithm

Self-Built CNN Model

The self-built CNN model uses 3 convolution layers, max pooling layers, ReLU activation function along with a dense fully connected layer. This model was tweaked with multiple parameters.

We performed experiments with different layering, activation functions, learning rate, loss function, and optimizers. We achieved the best result with Adam optimizer.

Using a CNN image classification model is a good fit as the multiple convolution layers allow us to predict patterns in the kitchen images in a human-like

EfficientNetB0 Model

EfficientNet is a pre-trained model that leverages transfer learning and improved architecture over CNN. The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2, in addition to squeeze-and-excitation blocks.

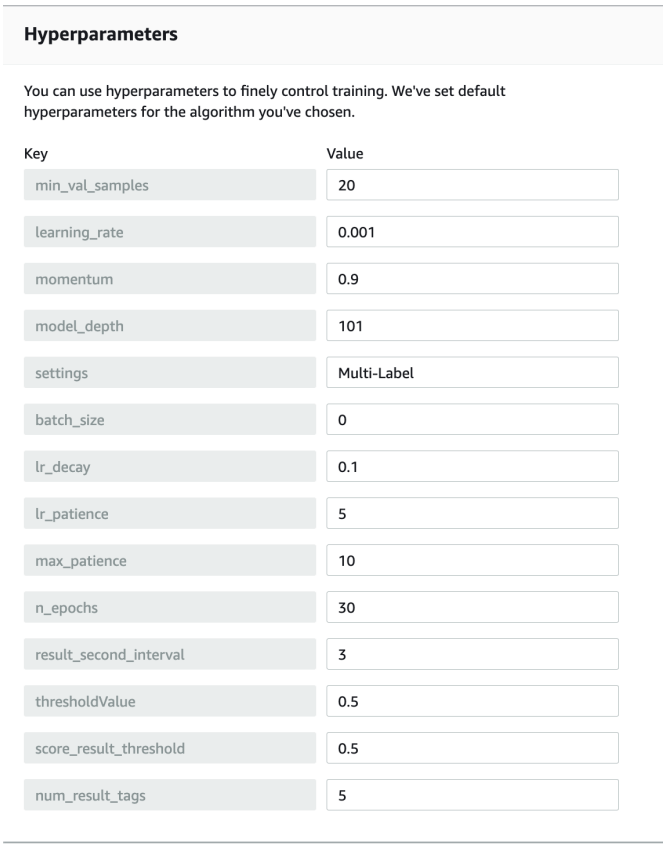
MobileNetV2

MobileNetV2 is a convolutional neural network architecture that can perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers.

AWS SageMaker Auto Algorithm

We also experimented with auto algorithms from AWS SageMaker. These algorithms allow you to plug and play a machine learning algorithm developed by SageMaker simply by analyzing your dataset.

Here is an example configuration of SageMaker auto algorithm.



VII. Model Comparison

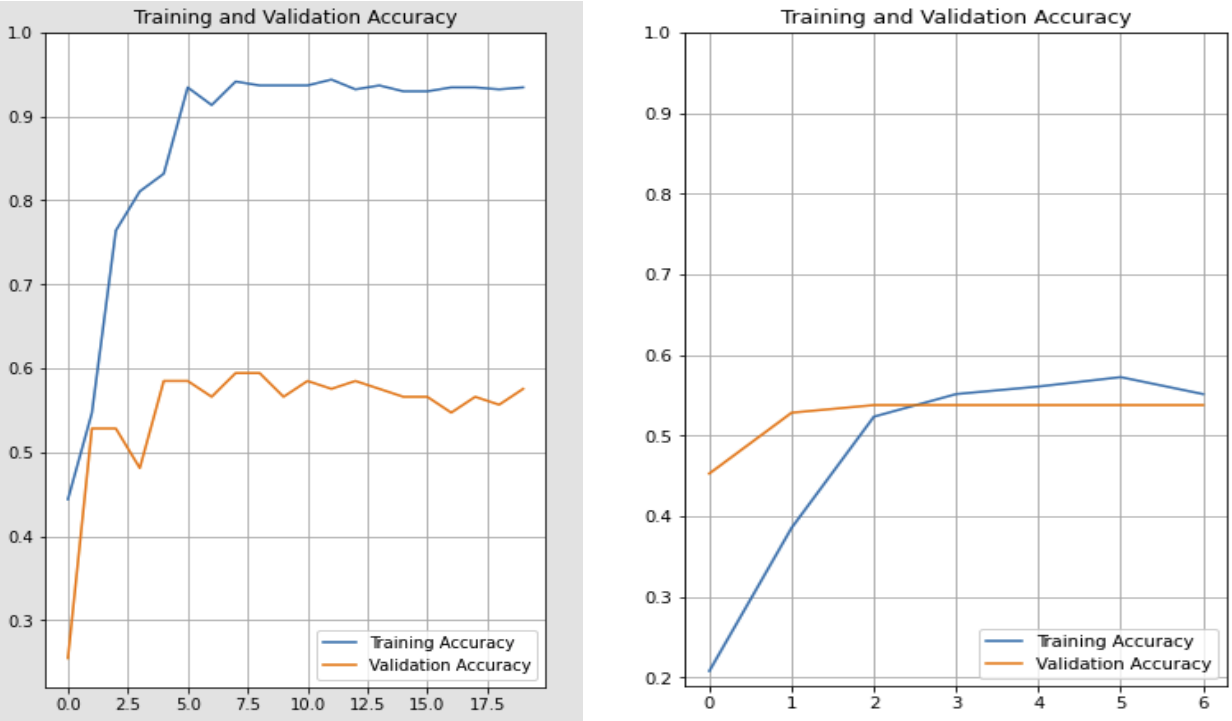


Figure : Custom CNN Model (Left) and MobileNetV2 Model (Right)

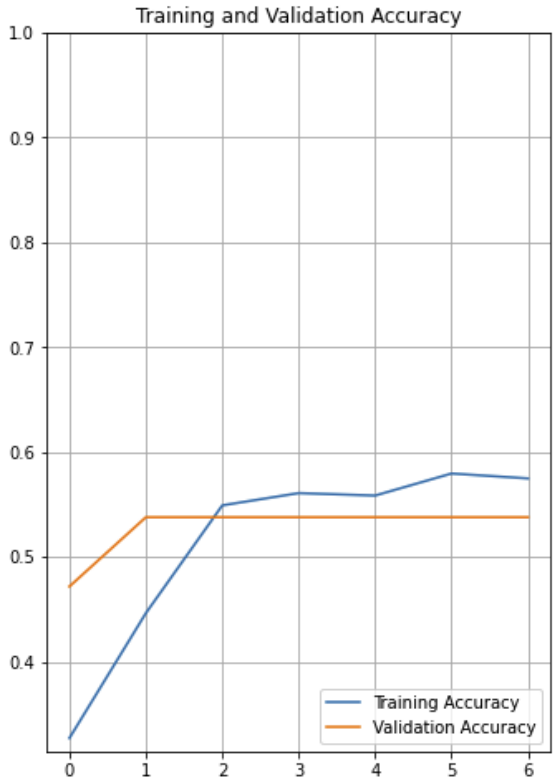


Figure : EfficientNetB0 Model

The accuracy of these models has varied significantly. The custom model we have built uses 3 convolution layers and ReLU activation function. We also experimented with MobileNetV2 and EfficientNetB0 models, which did not offer satisfactory results.

There is scope to improve upon the pretrained models by performing fine tuning. For the scope of this project, we have focused on application improvements.

The accuracy achieved by the different models is as follows:

1. Custom CNN Model - 93.46%
2. MobileNetV2 Model - 57.24%
3. EfficientNetB0 Model - 53.7%

As we can see, the best model for classifying kitchen layouts is our custom CNN model at 93.46% accuracy.

VIII. Model Acceleration

TensorFlow Lite supports several hardware accelerators to speed up inference on your mobile device. GPU is one of the accelerators that TensorFlow Lite can leverage through a delegate mechanism and it is fairly easy to use.

In addition we have experimented with acceleration on both **Android and EC2.**

Here is the comparison of performance:

Platform	Acceleration Type	Response Time (P95)	Requests Performed (Per Minute)
Android	None	675ms	10
EC2 Flask API	None	4.25s	10
Android	GPU	533ms	10
EC2 Flask API	GPU	4.1s	10

As evident, we are able to see a 3.4% increase in performance in EC2 Flask API and a tremendous 26.6% improvement in performance in Android Tensorflow acceleration using a GPU delegate.

IX. Model Inference

For model inference, we have determined our approach based on user friendliness.

The three main methods we have attempted to perform inference on are:

1. EC2 Hosted Flask API
2. EC2 Hosted Flask API with Acceleration
3. TensorFlow Lite on Android

TensorFlow Lite is the lightweight version which is specifically designed for the mobile platform and embedded devices. It provides machine learning solutions for mobile with low latency and small binary size.

The TensorFlow Lite interpreter, which runs specially optimized models on many different hardware types, including mobile phones, embedded Linux devices, and micro controllers.

The TensorFlow Lite converter, which converts TensorFlow models into an efficient form for use by the interpreter, and can introduce optimizations to improve binary size and performance. The trained TensorFlow model on the disk will convert into TensorFlow Lite file format (.tflite) using the TensorFlow Lite converter. Then we can use that converted file in the mobile application.

Benefits of Tensorflow Lite:

1. Low latency
2. Faster response time
3. Higher customer satisfaction due to increased performance.

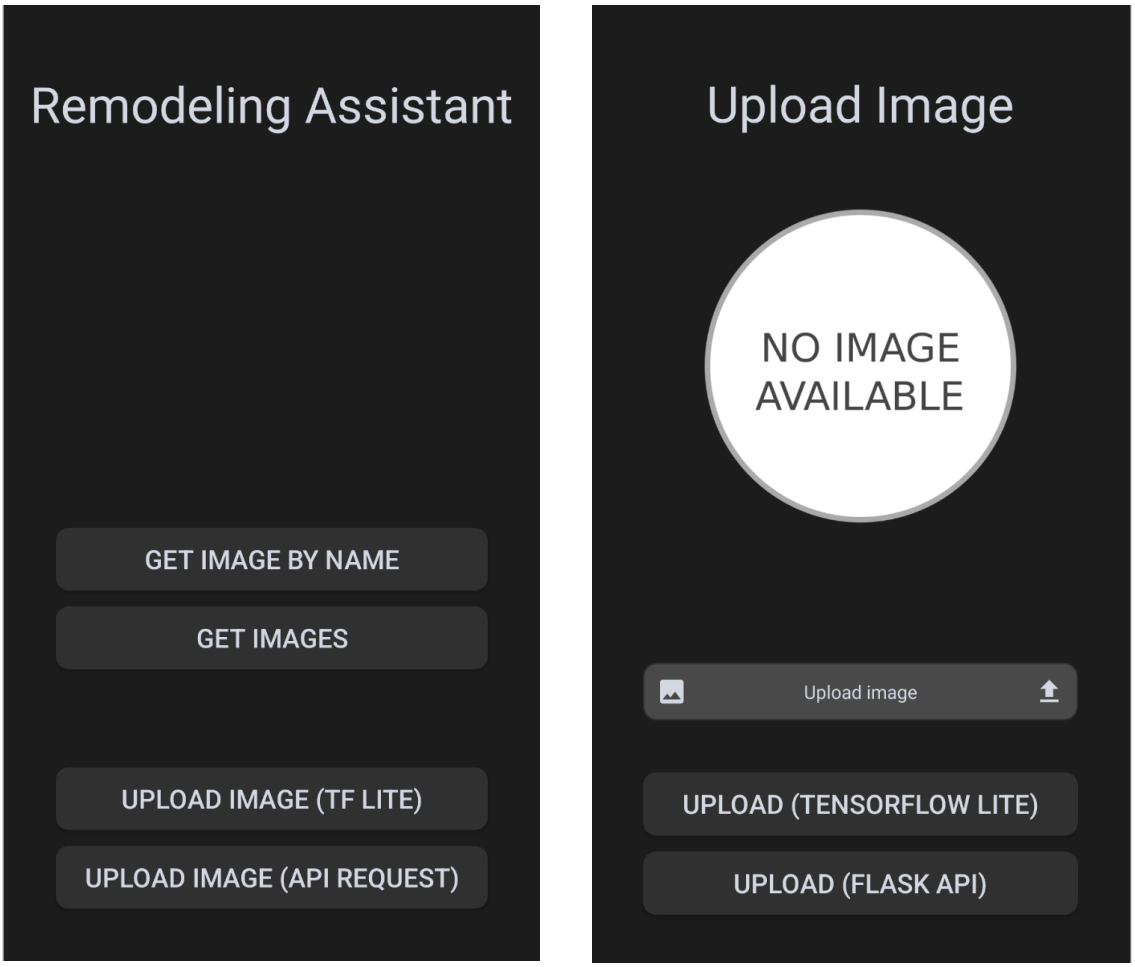


Figure: Android Application Screens

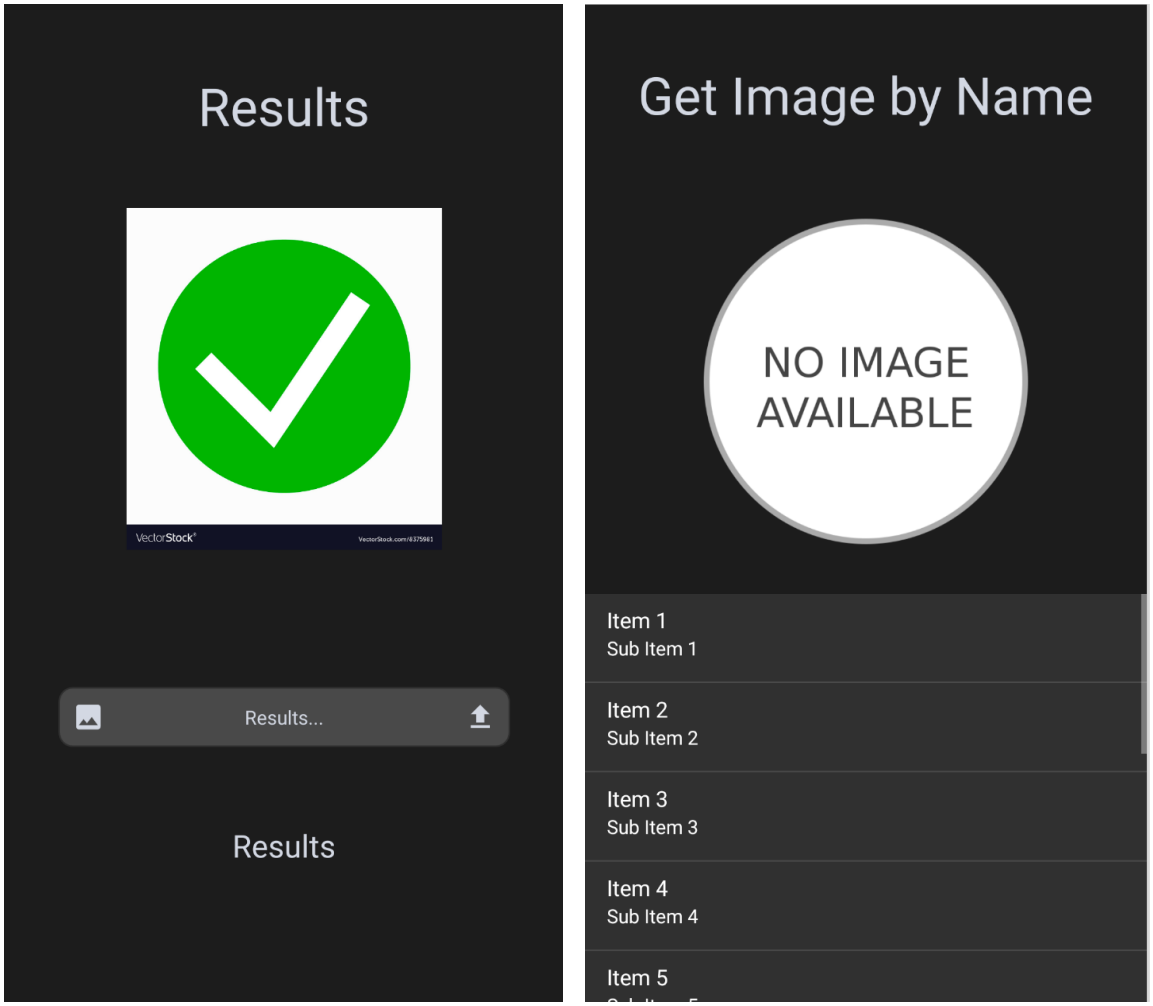


Figure: Android Application Screens

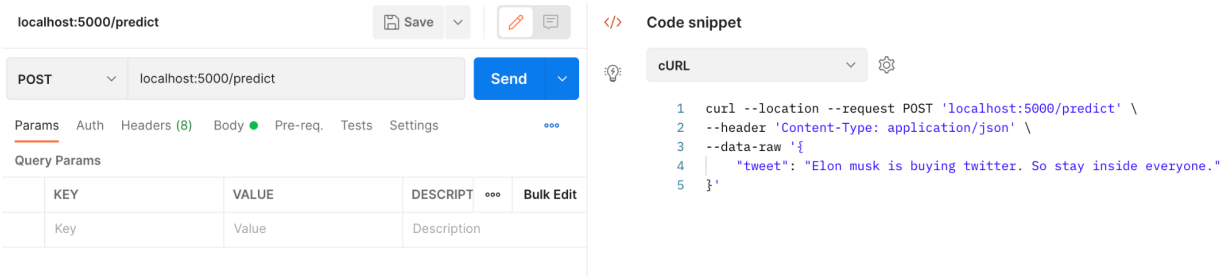


Figure: Flask REST API Calls

X. Code Demonstration

Our code is easy to walk through. We have achieved most functionality of machine learning and inference in two languages - Python and Java, both using Tensorflow.

A. Sample Requests to Flask API

Request:

```
curl --location --request POST 'localhost:5000/predict' \
        --form 'image1=@"sample.jpeg"'
```

Response

```
{
   "classifiers": "One-Wall Layout",
   "prediction_id": "ee1b8482-4892-4a5e-b0e2-2481dfacf184"
}
```

Internally, the model generates the probability output of, as an example, "[[1.9729896 -3.0941513 -2.8888257]]" for kitchen layout classifiers, which in turn uses 1-Hot Encoding to get the right label.

B. Flask Tensorflow Inference Code

```python
def make_summary(file):
    # model inference here
    CLASSES = sorted(['One-Wall Layout', 'L-Shaped Layout', 'U-Shaped Layout'])

    # Load the TFLite model and allocate tensors.
    interpreter = tf.lite.Interpreter(model_path="api/tensorflow/converted_model.tflite")
    interpreter.allocate_tensors()

    # Get input and output tensors.
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    input_shape = input_details[0]['shape']

    image = Image.open(file).resize((512, 512))
    image = np.array(image)
    input=image[np.newaxis, ...]

    input_data = np.array(np.random.random_sample(input_shape), dtype=np.float32)

    input_data = np.array(input, dtype=np.float32)

    interpreter.set_tensor(input_details[0]['index'], input_data)

    interpreter.invoke()

    output_data = interpreter.get_tensor(output_details[0]['index'])

    return {
        'prediction_id': uuid.uuid4(),
        'classifiers': str(CLASSES[np.argmax(output_data[0], axis=-1)])
    }
```

C. Flask Rest API Routing

```python
def configure_inference_handlers(app):
    @app.route('/predict', methods=['POST'])
    def predict():
        if 'image1' not in request.files:
            return 'there is no image1 in the request!'

        file1 = request.files['image1']

        data = make_summary(file1)
        response = app.response_class(
            response=json.dumps(data),
            status=200,
            mimetype='application/json'
        )
        return response
```

D. Android API Request

```
// Upload the image to the remote database
public void uploadImage(View view) throws IOException {
    Bitmap bitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), selectedImage);
    ByteBuffer buffer = ByteBuffer.allocate(bitmap.getByteCount()); //Create a new buffer
    bitmap.copyPixelsToBuffer(buffer);
    //File imageFile = new File(part_image);                                                    /,
    RequestBody reqBody = RequestBody.create(MediaType.parse("multipart/form-file"), buffer.array());
    MultipartBody.Part partImage = MultipartBody.Part.createFormData( name: "image1", UUID.randomUUID().
    API api = RetrofitClient.getInstance().getAPI();
    Call<ResponseBody> upload = api.uploadImage(partImage);
    upload.enqueue(new Callback<ResponseBody>() {
        @Override
        public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
            if(response.isSuccessful()) {
                try {
                    Toast.makeText( context: Upload.this,  text: "Kitchen Photo Accepted" + response.body().
                    Intent i = new Intent( packageContext: Upload.this, Results.class);
                    i.putExtra( name: "key", response.body().string());
                    startActivity(i);                    // Start the activity to get all images

                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        @Override
        public void onFailure(Call<ResponseBody> call, Throwable t) {
            Toast.makeText( context: Upload.this,  text: "Request failed", Toast.LENGTH_SHORT).show();
        }
    });
}
```

E. Android Tensorflow Lite Request

```
//ConvertedModel model = ConvertedModel.newInstance(this);
Interpreter interpreter = new Interpreter(FileUtil.loadMappedFile( context: this,  filePath: "converted_model.tflite"), options);

// Creates inputs for reference.
TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, 512, 512, 3}, DataType.FLOAT32);
Bitmap bitmap = Bitmap.createScaledBitmap(imageBitmap,  dstWidth: 512,  dstHeight: 512,  filter: true);
ByteBuffer byteBuffer = convertBitmapToByteBuffer(bitmap);
inputFeature0.loadBuffer(byteBuffer);

float[][] result = new float[1][3];
interpreter.run(byteBuffer, result);
Toast.makeText( context: Upload.this,  text: "Kitchen Photo Accepted" + getResult(result), Toast.LENGTH_LONG).show();

long endTime = SystemClock.uptimeMillis();
String runTime = String.valueOf(endTime - startTime);

Log.d( tag: "Result",  msg: "kitchenClassifer: " + runTime + "ms");

Thread.sleep( millis: 3000);

Intent i = new Intent( packageContext: Upload.this, Results.class);
i.putExtra( name: "key", getResult(result));
startActivity(i);
```

F. Python Flask GPU Acceleration

```python
def create_app(config=None, app_name=None, blueprints=None):
    # Create a Flask app

    if app_name is None:
        app_name = DefaultConfig.PROJECT
    if blueprints is None:
        blueprints = DEFAULT_BLUEPRINTS

    app = Flask(app_name,
                instance_path=INSTANCE_FOLDER_PATH,
                instance_relative_config=True)

    configure_app(app, config)
    configure_hook(app)
    configure_blueprints(app, blueprints)
    configure_extensions(app)
    configure_logging(app)
    configure_template_filters(app)
    configure_error_handlers(app)
    configure_inference_handlers(app)

    gpus = tf.config.list_physical_devices('GPU')
    if gpus:
    # Restrict TensorFlow to only allocate 1GB of memory on the first GPU
        try:
            tf.config.set_logical_device_configuration(
                gpus[0],
                [tf.config.LogicalDeviceConfiguration(memory_limit=1024)])
            logical_gpus = tf.config.list_logical_devices('GPU')
            print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
        except RuntimeError as e:
            # Virtual devices must be set before GPUs have been initialized
            print(e)

    return app
```

G. Android Tensorflow Lite GPU Acceleration

```java
// Initialize interpreter with GPU delegate
Interpreter.Options options = new Interpreter.Options();
CompatibilityList compatList = new CompatibilityList();

if(compatList.isDelegateSupportedOnThisDevice()){
    // if the device has a supported GPU, add the GPU delegate
    GpuDelegate.Options delegateOptions = compatList.getBestOptionsForThisDevice();
    GpuDelegate gpuDelegate = new GpuDelegate(delegateOptions);
    options.addDelegate(gpuDelegate);
    Log.d( tag: "Result",  msg: "kitchenClassifer: GPU Enabled for Tensorflow.");
} else {
    // if the GPU is not supported, run on 4 threads
    options.setNumThreads(4);
}
```

XI. Cloud Technologies

We have made heavy use of Cloud Computing and AWS specifically.

Our backend architecture is completely based on AWS, SageMaker, and Flask along with custom code. The frontend mobile app uses Android along with Tensorflow.

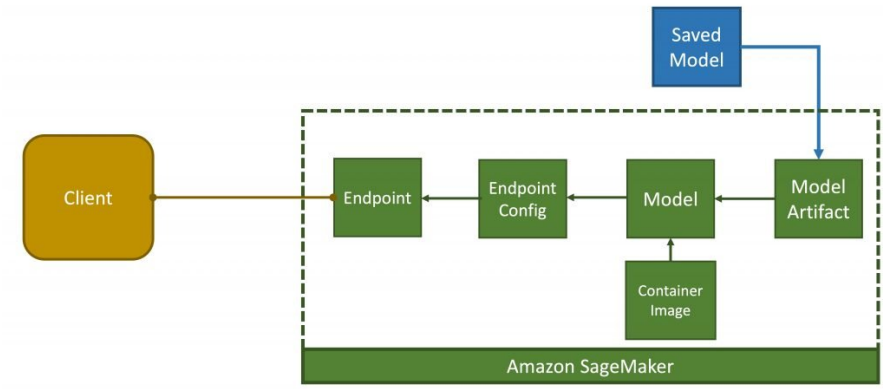


Figure: Use of AWS SageMaker

XII. Conclusion

All in all, this project was exciting as it allowed us to work on a complete data mining pipeline. We were able to evaluate multiple models and test inference response time via multiple mechanisms. Our end choice was to use our custom model instead of SageMaker and other models due to cost optimization and higher performance using GPU.

We achieved an accuracy of 93% with the CNN model. Moreover, the project includes a fully working pipeline of data labeling, preprocessing, modeling, inference, mobile application, and cloud infrastructure.

These capabilities can be leveraged by a user or startup interested in kitchen remodeling space. Furthermore, the pipeline can be improved further with improved datasets and applying additional machine learning techniques.

XIII. Key References

Gpiosenka. (2021, December 7). *Utensils F1 test score =98%*. Kaggle. Retrieved May 25, 2022, from https://www.kaggle.com/code/gpiosenka/utensils-f1-test-score-98

Image recognition experiment: Finding furniture on kitchen photos. Abto Software. (2022, January 4). Retrieved May 25, 2022, from https://www.abtosoftware.com/blog/image-recognition-kitchen-furniture-appliances

Madhab, N. (2021, November 10). *Let's develop an Android app to upload files and images on cloud*. Medium. Retrieved May 26, 2022, from https://medium.com/javarevisited/lets-develop-an-android-app-to-upload-files-and-images-on-cloud-f9670d812060

Madhab, N. (2021, January 3). *Let's add products in Android for E-Commerce App*. Medium. Retrieved May 27, 2022, from https://medium.com/webtutsplus/lets-add-products-in-android-for-e-commerce-app-b8468e055001