

Developing and Optimizing Convolutional Neural Networks for Traffic Sign Recognition: A Comparative Study Using Transfer Learning and Model Architecture Tuning on the Indian Traffic Sign Recognition Benchmark (ITSRB) Dataset

1) KARTHIK B M 2) NARENDRA

3) Athul 4) Vishal

Course: Online Course on AI and Machine Learning Course, IISC

Instructor: Prof. Sashikumaar Ganesan



Contents



- Problem Description
- Dataset
- Data Processing
- Model Architecture
- Model Training and Validation
- Model Testing and Results
- Pretrained Models, Mlops

Problem Description



This project addresses the problem of enhancing traffic sign recognition systems by developing a CNN-based approach and by using Pertained models and MLops. Despite recent advances, achieving near-perfect recognition accuracy remains a challenge, particularly in the context of complex, real-world driving environments.

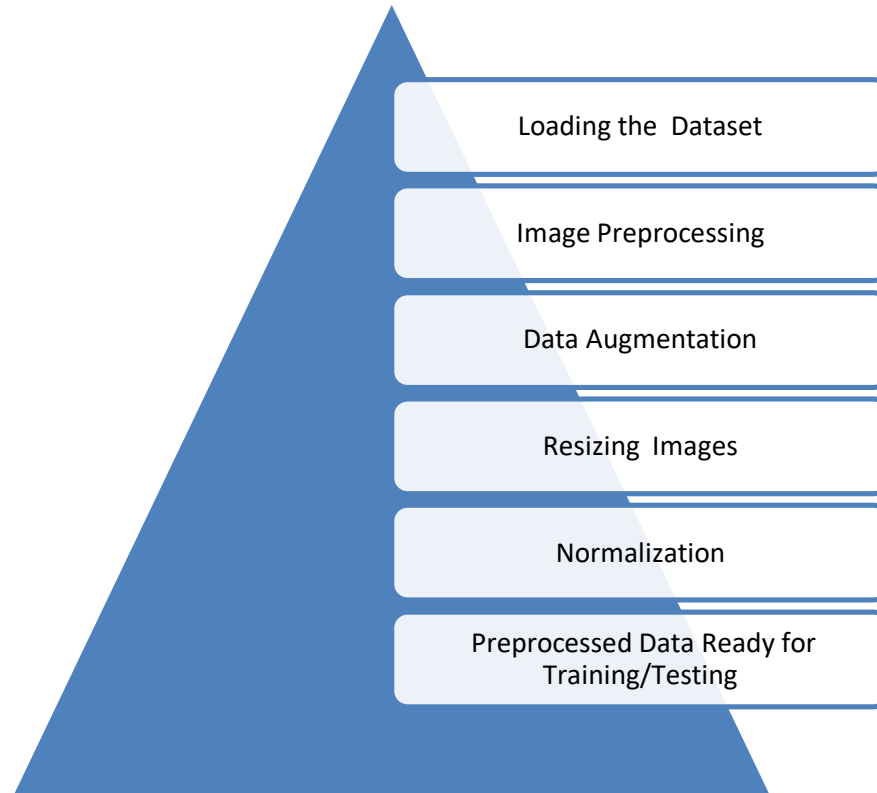
The proposed solution aims to overcome these limitations and contribute to the broader goal of improving road safety, supporting autonomous vehicle navigation, and aiding drivers, including those with disabilities, through more effective recognition and interpretation of traffic signs

Dataset

- Indian traffic sign dataset was used to work on a Traffic sign detection and recognition system.
- The dataset can be used for any Traffic Sign project based on Image Classification.
- Training : 10155 images belonging to 52 classes.
- Testing : 2510 images belonging to 52 classes
- height = 64 width = 64



Data Processing



Flow Diagram Description:

Load Dataset: Read the dataset (images) from the directory.

Image Preprocessing:

- *Converting images to gray scale.*
- *Enhance contrast using histogram equalization.*
- *Reduce noise with Gaussian blur.*
- *Sharpen details using a kernel.*
- *Convert images back to RGB.*

Data Augmentation:

- *Apply transformations like:*
- *Horizontal flipping.*
- *Brightness and contrast adjustment.*
- *Rotation for image diversity.*
- **Resize Images:** *Resizing images to a uniform size suitable for model input.*

Normalization: Scaling pixel values to the range [0, 1] for numerical stability during training.

Preparing for Modeling : Converting processed data into a NumPy array and split into training and testing sets.

Flow:

Input: Raw Images

Step 1: Preprocess images (grayscale → contrast → blur → sharpen → RGB).

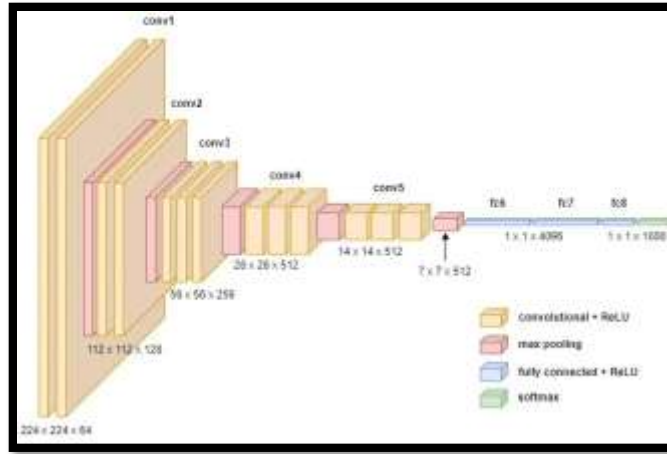
Step 2: Apply augmentation (flip, brightness, etc.).

Step 3: Resize images to the target shape.

Step 4: Normalize the data.

Output: Preprocessed Data Ready for Training/Testing

Model Architecture



Model Architecture Flow: Input \rightarrow Reflective Padding \rightarrow Conv2D + Swish \rightarrow MaxPooling2D \rightarrow BatchNormalization \rightarrow Dropout \rightarrow Conv2D Blocks (64, 128, 256 filters) \rightarrow Flatten \rightarrow Dense (512 units) \rightarrow Output (Softmax).

Model Architecture

```
model = keras.models.Sequential([ReflectivePadding2D(padding=(1, 1), input_shape=(height, width, 3)),
    keras.layers.Conv2D(filters= 64 , kernel_size=(5,5), activation='swish', kernel_initializer=initializers.HeNormal()),
    ReflectivePadding2D(padding=(1, 1)),
    keras.layers.Conv2D(filters= 64, kernel_size=(3,3), kernel_initializer=initializers.HeNormal(), activation='swish'),

    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),
    keras.layers.Dropout(rate=0.5),

    ReflectivePadding2D(padding=(1, 1)),
    keras.layers.Conv2D(filters=128, kernel_size=(3,3),kernel_initializer=initializers.HeNormal(), activation='swish'),

    ReflectivePadding2D(padding=(1, 1)),
    keras.layers.Conv2D(filters=128, kernel_size=(3,3), kernel_initializer=initializers.HeNormal(), activation='swish'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),
    keras.layers.Dropout(rate=0.5),

    ReflectivePadding2D(padding=(1, 1)),
    keras.layers.Conv2D(filters= 256, kernel_size=(3,3), kernel_initializer=initializers.HeNormal(), activation='swish'),
    ReflectivePadding2D(padding=(1, 1)),
    keras.layers.Conv2D(filters= 256, kernel_size=(3,3), kernel_initializer=initializers.HeNormal(), activation='swish'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Dropout(rate=0.5),
    keras.layers.Flatten(),
    keras.layers.Dense(512, kernel_initializer=initializers.HeNormal(), activation='swish'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),

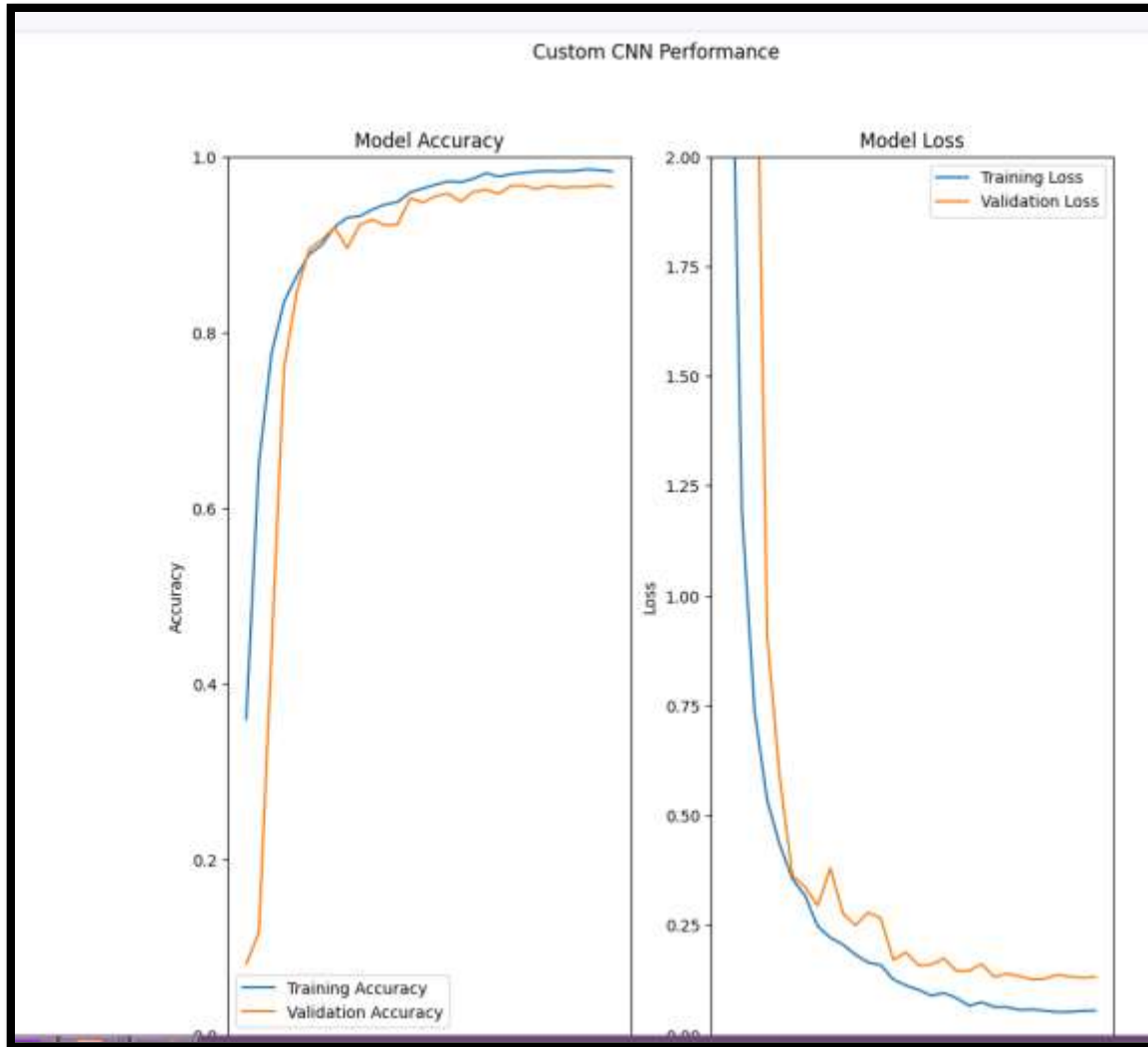
    keras.layers.Dense(52, activation='softmax')
])
```

2]: `model.summary()`

Model: "sequential"

Layer (type)	Output Shape	Param #
reflective_padding2d (ReflectivePadding2D)	(None, 66, 66, 3)	0
conv2d (Conv2D)	(None, 62, 62, 64)	4864
reflective_padding2d_1 (ReflectivePadding2D)	(None, 64, 64, 64)	0
conv2d_1 (Conv2D)	(None, 62, 62, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 31, 31, 64)	0
batch_normalization (Batch Normalization)	(None, 31, 31, 64)	256
dropout (Dropout)	(None, 31, 31, 64)	0
reflective_padding2d_2 (ReflectivePadding2D)	(None, 33, 33, 64)	0
conv2d_2 (Conv2D)	(None, 31, 31, 128)	73856
reflective_padding2d_3 (ReflectivePadding2D)	(None, 33, 33, 128)	0
conv2d_3 (Conv2D)	(None, 31, 31, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 128)	0
batch_normalization_1 (Batch Normalization)	(None, 15, 15, 128)	512

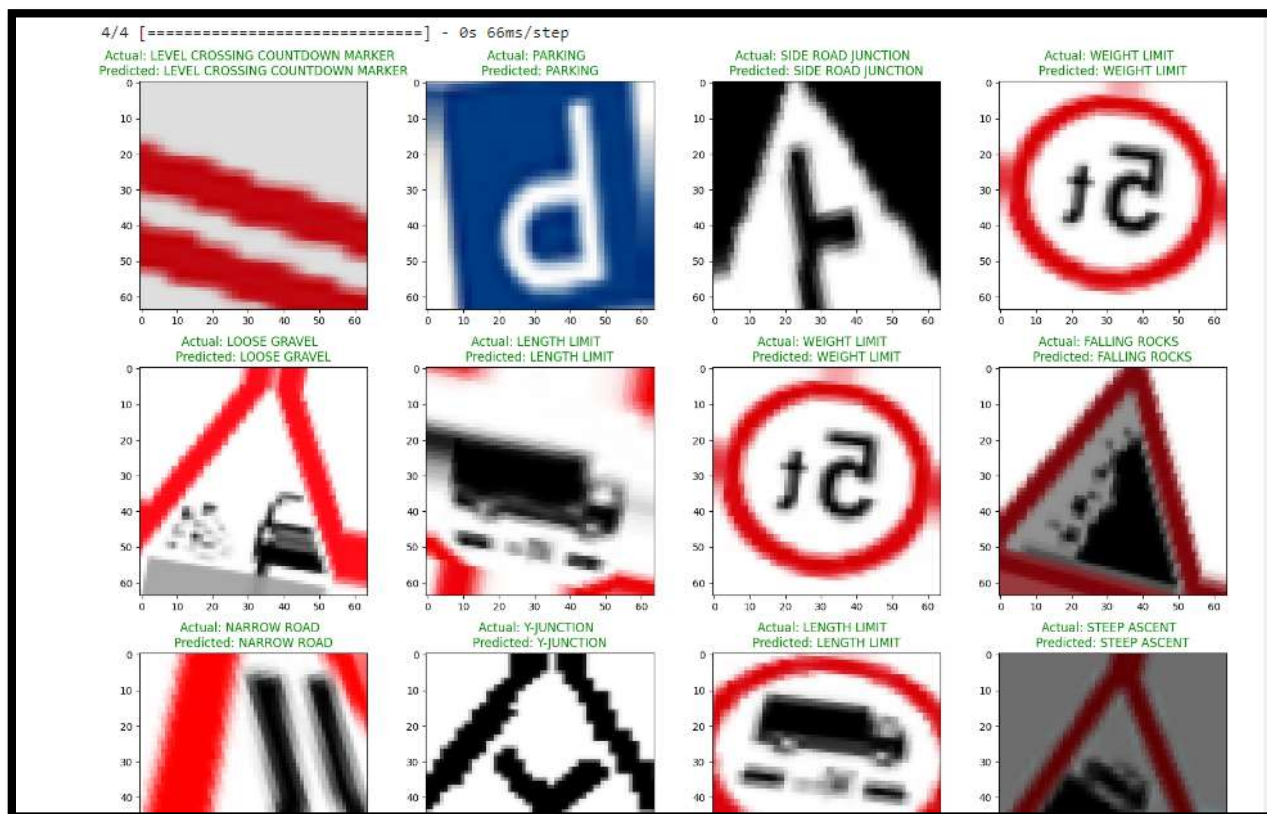
Model Training and Validation



```
43, 48, 16, 8, 10]  
f1 SCORE is: 0.8732453375199561  
Test Data accuracy: 87.02830188679245  
t[62]: 0.8702830188679245
```

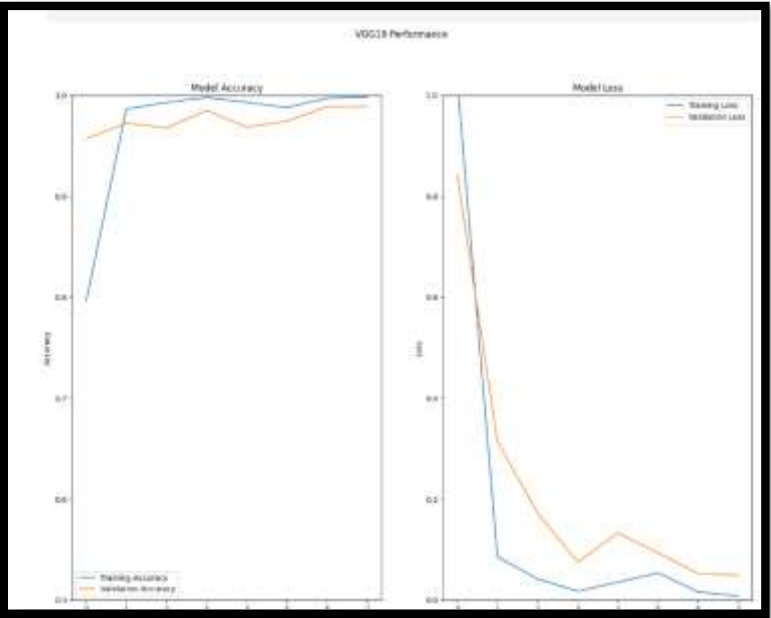
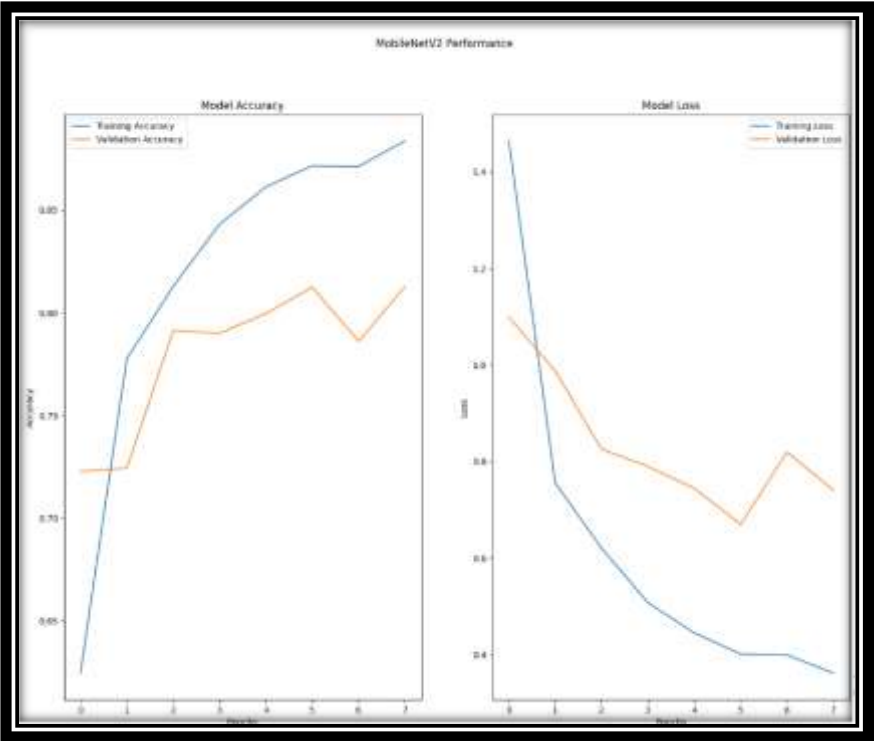
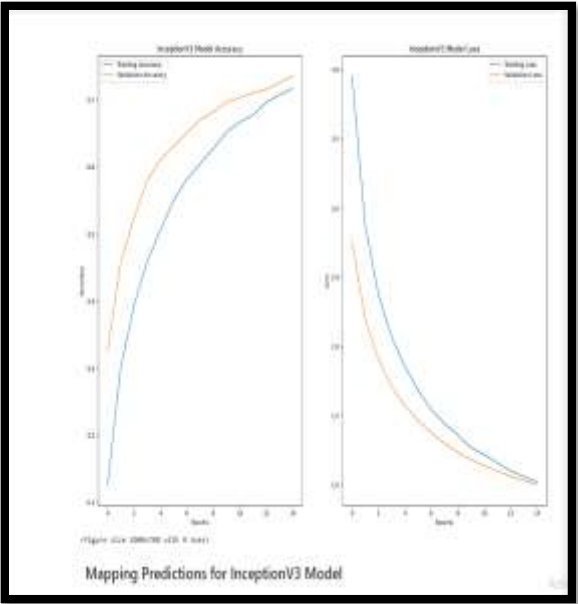

Model Testing and Results

- Testing data consisted of 2510_test images, with their labels indicating the corresponding traffic signs. Testing F1 Score came out to be 0.88, and the test accuracy was 88.56%, which is considered a good result for the dataset and problem. The testing results validate the model's ability to generalize to unseen data, aligning closely with the validation metrics observed during training.



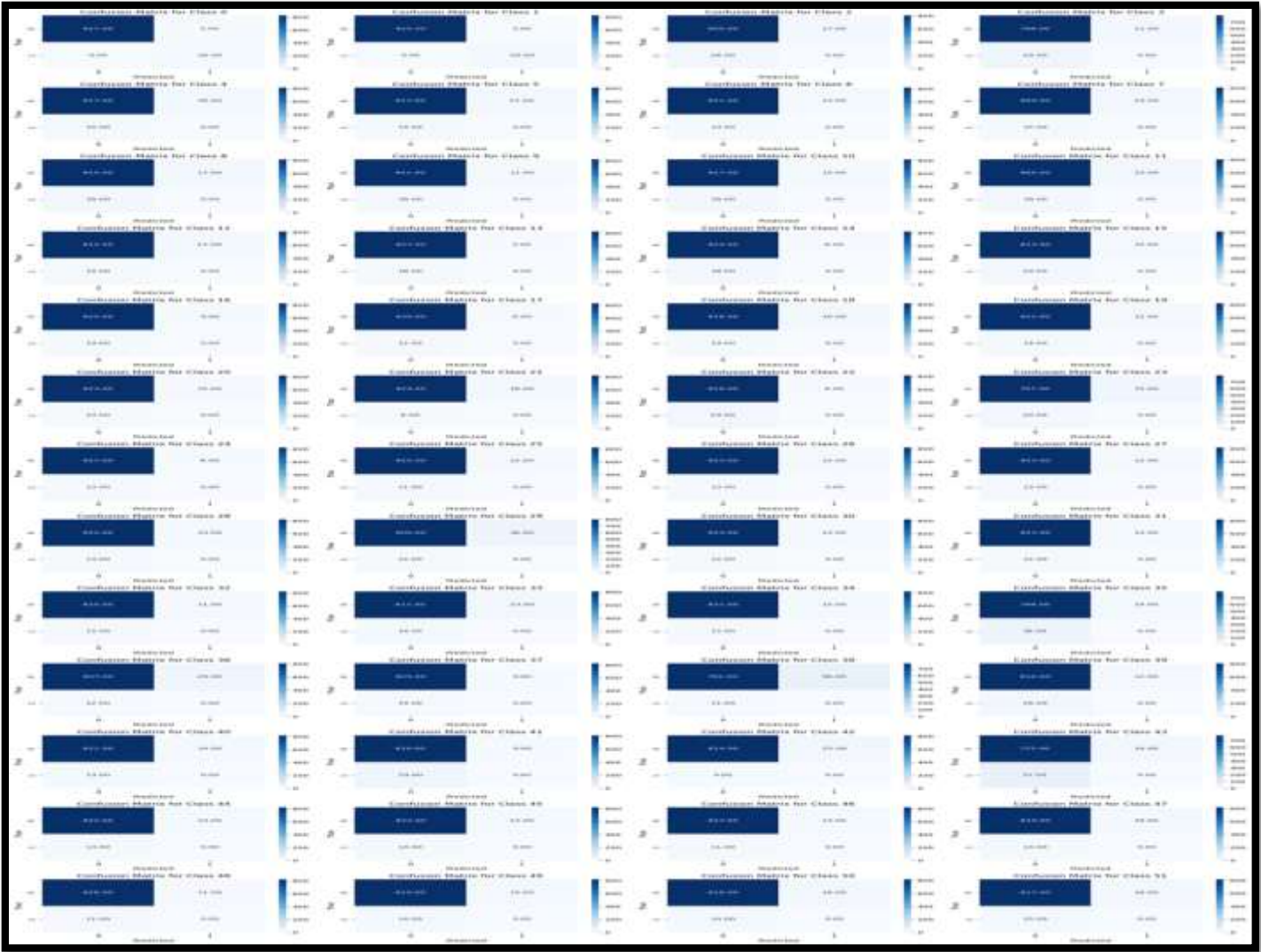
Mapping Predictions for pertained Models

Comparison of pertained models for **InceptionV3 Pre-trained Model, Mobile Net v2 and VGG19**



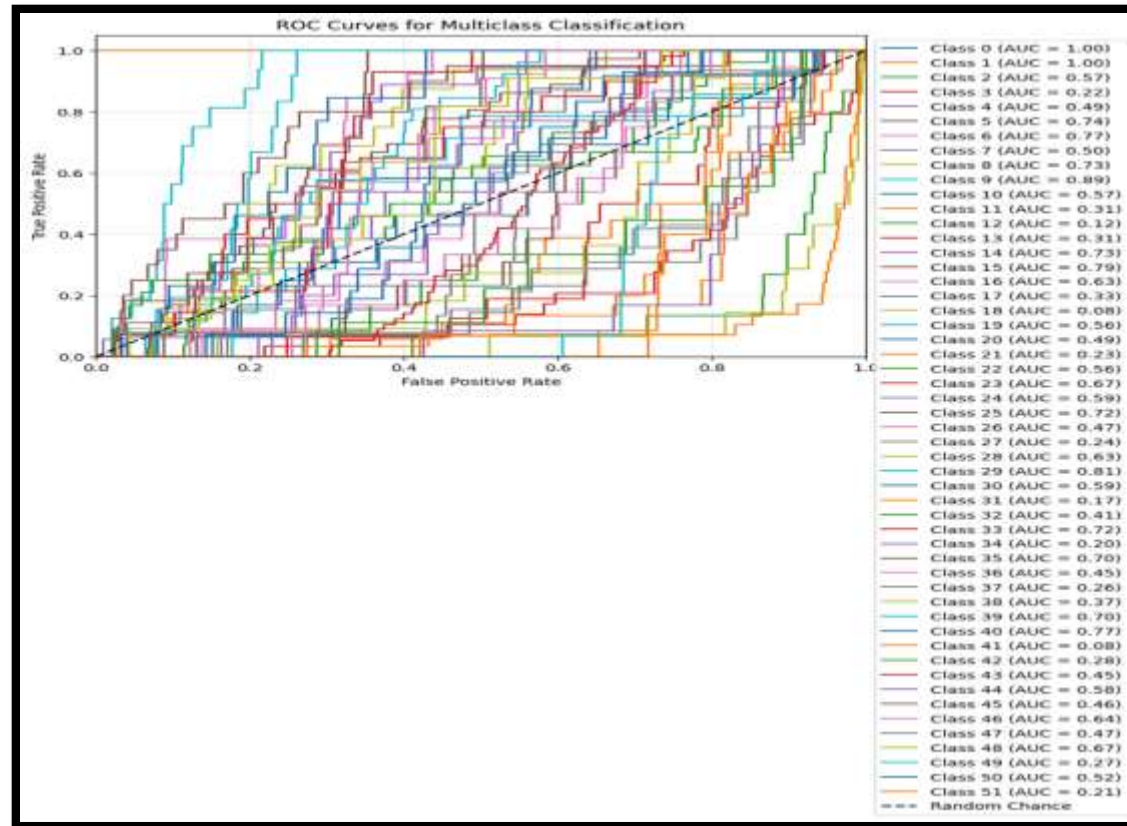
Confusion matrix for each class

Confusion matrix



Combined ROC for all Classes

Combined ROCS FOR ALL CLASSES



Mlops Workflow

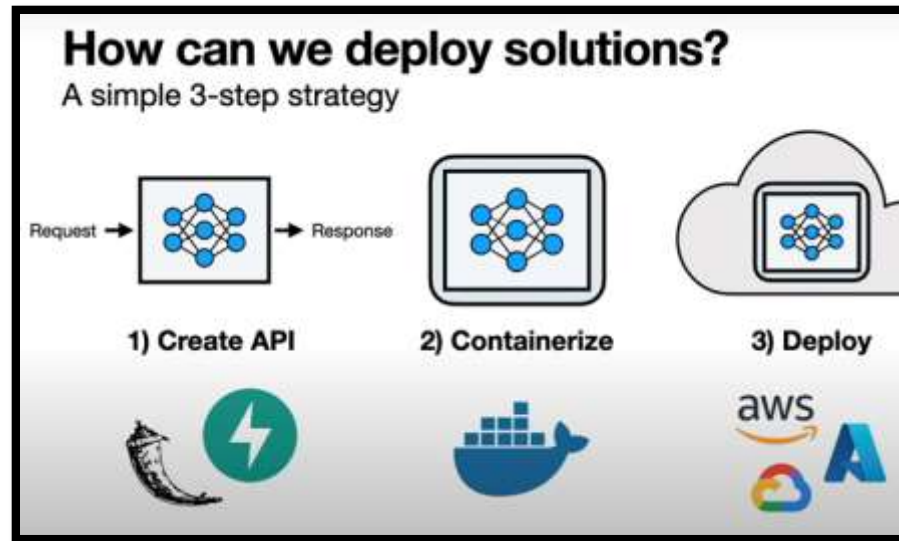
Data Preparation - Version Control -Github

Model Training – Experiment Tracking -

Model Packaging – Containerization - Docker

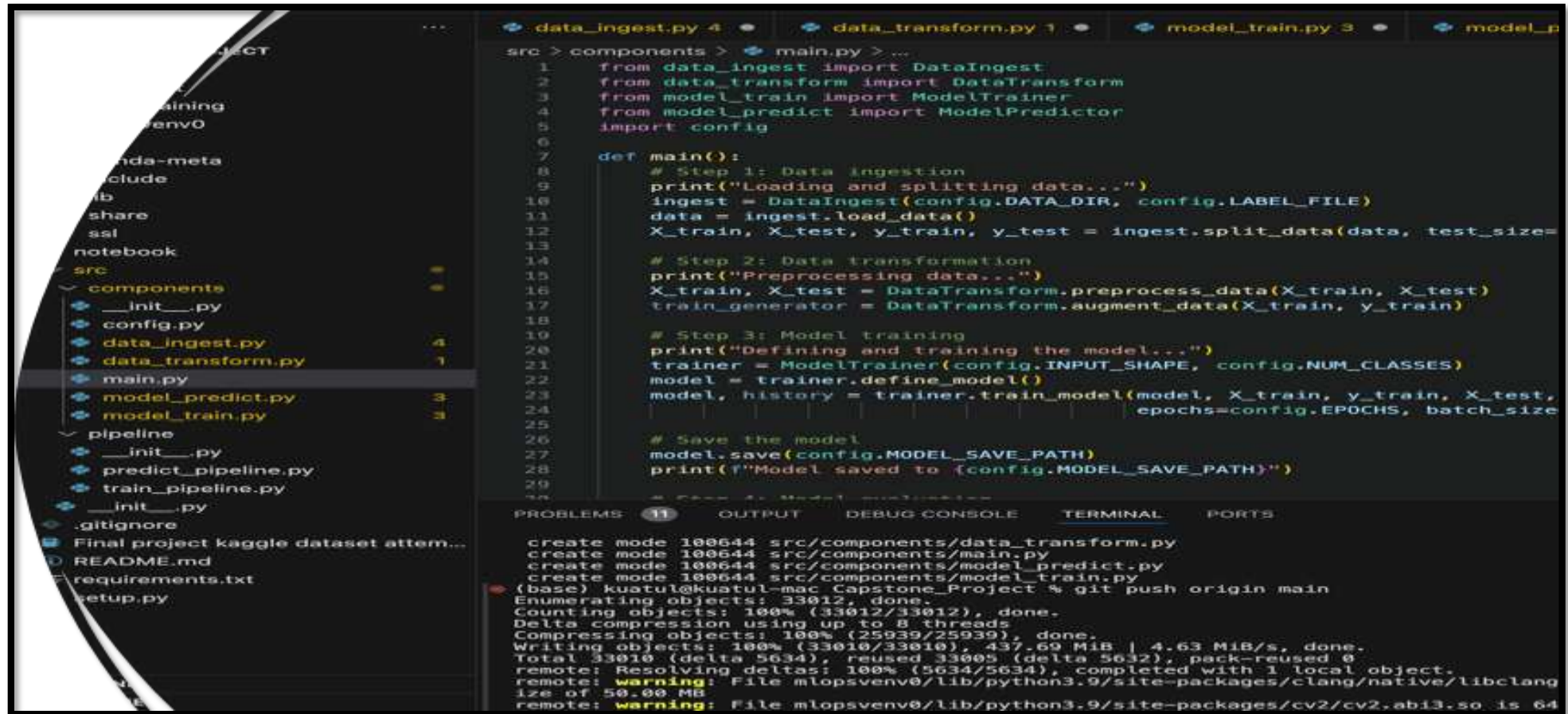
Model Deployment – Serving API - FlaskAPI

Monitoring and Retraining – Automated Pipelines AWS



GIT versioning

GIT versioning



```
PROJECT
├── .env
├── .envO
├── .meta
├── .include
├── .lib
├── .share
├── .ssl
├── .notebook
├── .src
├── components
│   ├── __init__.py
│   ├── config.py
│   ├── data_ingest.py
│   ├── data_transform.py
│   ├── main.py
│   ├── model_predict.py
│   ├── model_train.py
├── pipeline
│   ├── __init__.py
│   ├── predict_pipeline.py
│   ├── train_pipeline.py
├── __init__.py
├── .gitignore
├── Final project kaggle dataset attem...
├── README.md
├── requirements.txt
├── setup.py
```

```
src > components > main.py > ...
1  from data_ingest import DataIngest
2  from data_transform import DataTransform
3  from model_train import ModelTrainer
4  from model_predict import ModelPredictor
5  import config
6
7  def main():
8      # Step 1: Data ingestion
9      print("Loading and splitting data...")
10     ingest = DataIngest(config.DATA_DIR, config.LABEL_FILE)
11     data = ingest.load_data()
12     X_train, X_test, y_train, y_test = ingest.split_data(data, test_size=
13
14     # Step 2: Data transformation
15     print("Preprocessing data...")
16     X_train, X_test = DataTransform.preprocess_data(X_train, X_test)
17     train_generator = DataTransform.augment_data(X_train, y_train)
18
19     # Step 3: Model training
20     print("Defining and training the model...")
21     trainer = ModelTrainer(config.INPUT_SHAPE, config.NUM_CLASSES)
22     model = trainer.define_model()
23     model, history = trainer.train_model(model, X_train, y_train, X_test,
24                                         epochs=config.EPOCHS, batch_size
25
26     # Save the model
27     model.save(config.MODEL_SAVE_PATH)
28     print(f"Model saved to {config.MODEL_SAVE_PATH}")
29
30     # Case 4: Model prediction
```

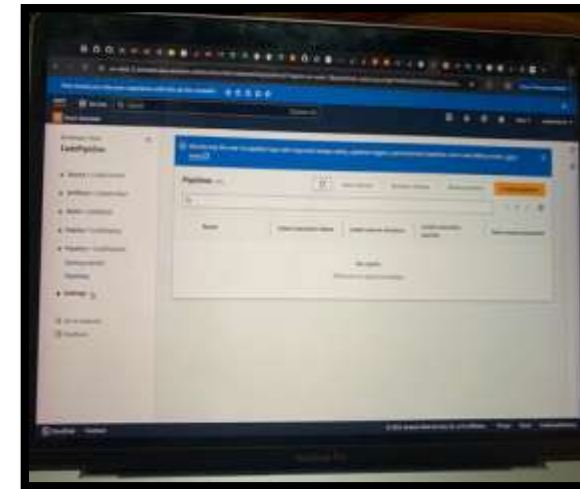
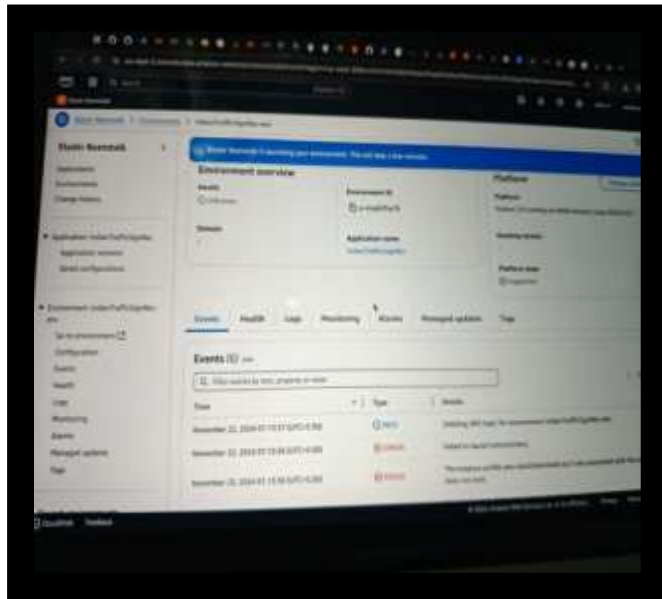
```
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS
create mode 100644 src/components/data_transform.py
create mode 100644 src/components/main.py
create mode 100644 src/components/model_predict.py
create mode 100644 src/components/model_train.py
(base) kuatul@kuatul-mac Capstone_Project % git push origin main
Enumerating objects: 33012, done.
Counting objects: 100% (33012/33012), done.
Delta compression using up to 8 threads
Compressing objects: 100% (25939/25939), done.
Writing objects: 100% (33010/33010), 437.69 MiB | 4.63 MiB/s, done.
Total 33010 (delta 5634), reused 33005 (delta 5632), pack-reused 0
remote: Resolving deltas: 100% (5634/5634), completed with 1 local object.
remote: warning: File mlopsvenv0/lib/python3.9/site-packages/clang/native/libclang
ize of 50.00 MB
remote: warning: File mlopsvenv0/lib/python3.9/site-packages/cv2/cv2.abi3.so is 64
```

AWS Beanstalk, Codepipeline

AWS Beanstalk, Codepipeline and EC3 instance set up in progress.

We are able to modularize the code in VS and push code to github branch as part of versioning.

However, we are facing issues with AWS Beanstalk, codepipeline setup. Flask api and Docker are yet to start.





Thank you



Team member contribution

1)KARTHIK B M 2) NARENDRA 3) Athul 4) Vishal

Data cleanup and acquisition - 100 / 100

ML Model Selection/ Training - 100/ 100

Hyper parameter tuning - 100 /100

Metrics - 100 / 100

Presentation - 100 / 100

Documentation - 100 / 100