EARIN 2023L PROJECT

VEHICLE CLASSIFICATION FROM AERIAL VIEW

Done by: Anatolii Do (309762)

Malik Amman (309188)

Group 13

PROBLEM DEFINITION

The purpose of this project is to develop an AI that distinguishes types of a vehicle from its aerial image.

Shortly, the goals are following:

- Develop a classification model and evaluate its performance.

- Develop a strategy to solve the class imbalance problem during training.

- Develop evaluation methods to understand the prediction results on the unknown validation set. It is strongly recommended to use the **confusion matrix** as one of the evaluation tools to understand class distribution from predictions.

The model will be trained on a large image dataset and evaluated on a validation set. The main challenges of this project include dealing with class imbalance in the dataset and achieving high accuracy and precision in the classification of the vehicles.

DATASET

Overview

The data set provided is available here.

This is training data set consisting of a big cluster of satellite vehicle images in black and white filter of 10 different types. The data set consists of approximately 300k .png images 32x32 each.

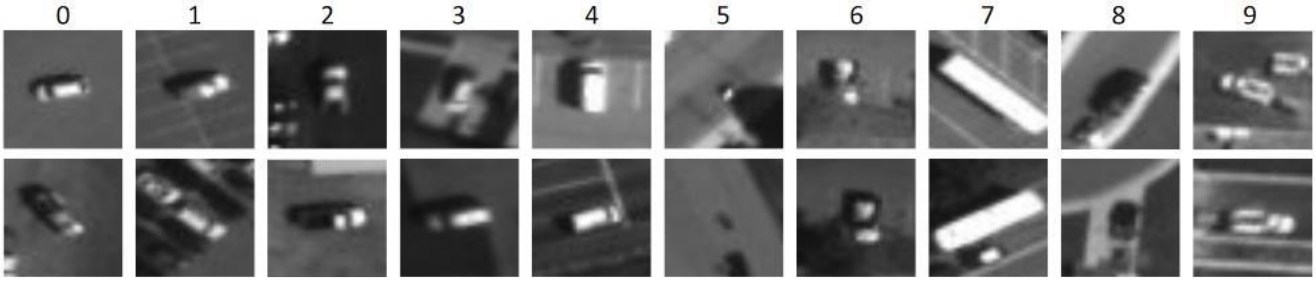


Fig1. Samples from data set

CLASS #	VEHICLE TYPE	# TRAIN	# VAL
0	Sedan	234,209	77
1	SUV	20,089	77
2	Pickup truck	15,301	77
3	Van	10,655	77
4	Box truck	1,741	77
5	Motorcycle	852	77
6	Flatbed truck	828	77
7	Bus	624	77
8	Pickup truck w/ trailer	840	77
9	Flatbed truck w/ trailer	633	77
	TOTAL	285,772	770

The dataset has a class imbalance problem, where the majority of the images belong to the car class, and the remaining classes have fewer samples.

If we try to train a machine learning model on the imbalanced dataset, it will not be effective because the model will tend to predict the majority class (Sedan in our case) most of the time and performance on other vehicle types will be poor.

To solve this problem, we will use under-sampling. The idea of this method is to reduce the number of samples in the majority classes. We will reduce the number of samples in all the classes to the number of images in a class with the smallest number of samples – Bus (624 images).

To ensure that each class is represented proportionally in both the training and validation sets, we will use stratified sampling to split the dataset. The dataset will be divided into groups based on the class labels, and then sampled from each group to create the training and validation sets. This will be done using the train_test_split function from scikit-learn library with the stratify parameter set to the class labels.

Pre-processing

Normalization actions on images to be done before the training: the pixel values of the images are normalized between 0 and 1 to help the model converge faster during training.

Post-processing

No post-processing actions to be done.

TECHNICAL APPROACH

Architecture

We will use a Convolutional Neural Network (CNN) as the base architecture for this project.

The model architecture used in the code consists of the following layers:

1. Convolutional Layer (Conv2D): This layer applies a 2-dimensional convolution operation to the input images. It uses 32 filters with a size of 5x5 and applies the ReLU activation function. The input shape of the layer is (32, 32, 1), indicating that the input images have a height and width of 32 pixels and a single channel (grayscale).
2. Max Pooling Layer (MaxPooling2D): This layer performs max pooling operations on the output of the previous convolutional layer. It uses a pooling window of size 2x2 to reduce the spatial dimensions of the feature maps.
3. Flatten Layer: This layer flattens the 2-dimensional feature maps into a 1-dimensional vector. It transforms the output of the previous layer into a format suitable for feeding into the fully connected layers.
4. Dense Layer: This layer is a fully connected layer with 128 neurons. It applies the ReLU activation function to introduce non-linearity in the model.
5. Dense Layer: This is the output layer of the model with a number of neurons equal to the number of classes. It uses the softmax activation function to produce class probabilities for each input image.

The model follows a sequential architecture, where each layer is added in sequence using the tf.keras.models.Sequential class. The model takes the input images with a shape of (32, 32, 1) and outputs class probabilities for the input images based on the defined architecture.

Training details

The model will be trained on a GPU (Graphics Processing Unit) using the **TensorFlow** deep learning framework. We will use the **cross-entropy loss function**, which is suitable for multi-class classification problems. The **Adam optimizer** will be used to update the weights of the model during training. For the baseline model, the following hyperparameters will be used:

- Batch size: 64
- Number of epochs: 12
- Learning rate: 0.001

We will also use **Keras**, a popular deep learning library that provides a user-friendly interface for building and training deep neural networks. We will use **NumPy** for data manipulation and **Matplotlib** for data visualization. Also, we use **Sklearn** to split the data into training and validation sets and for model evaluation.

Evaluation details

The performance of the model will be evaluated on a validation set using the following metrics:

- Accuracy: The percentage of correctly classified images.
- Precision: The ratio of true positive predictions to the total number of positive predictions.
- Recall: The ratio of true positive predictions to the total number of positive samples in the dataset.
- F1 Score: The harmonic mean of precision and recall.

The confusion matrix will also be used to evaluate the performance of the model. The confusion matrix shows the number of true positive, false positive, true negative, and false negative predictions for each class. It will help us understand the distribution of predicted classes and identify potential sources of errors.

IMPLEMENTATION

See main.py in Appendix.

RESULTS

	PRECISION	RECALL	F1-SCORE	SUPPORT
Sedan	0.86	0.73	0.80	150
SUV	0.83	0.92	0.97	125
Pickup truck	0.87	0.90	0.89	123
Van	0.85	0.96	0.90	112
Box truck	0.98	0.99	0.98	132
Motorcycle	0.99	0.98	0.99	103
Flatbed truck	1.00	0.99	1.00	120
Bus	0.99	0.99	0.99	135
Pickup truck w/ trailer	1.00	0.99	1.00	111
Flatbed truck w/ trailer	0.99	0.99	0.99	137
Accuracy	0.93			

Table 1. Classification report of baseline model

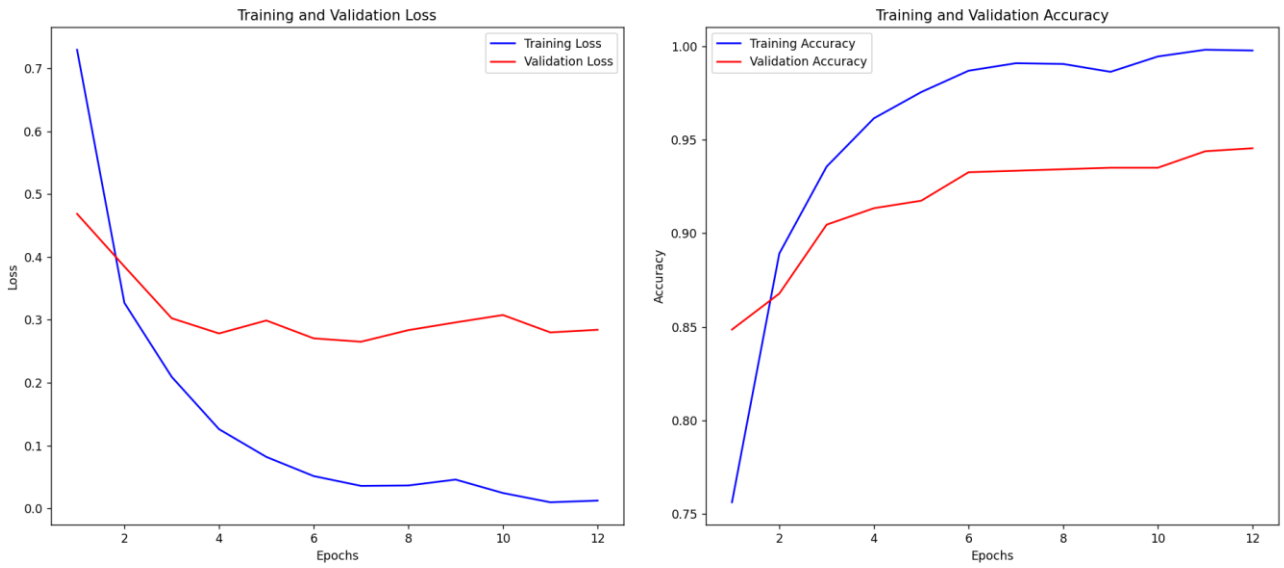


Fig 2. Visualization of training process

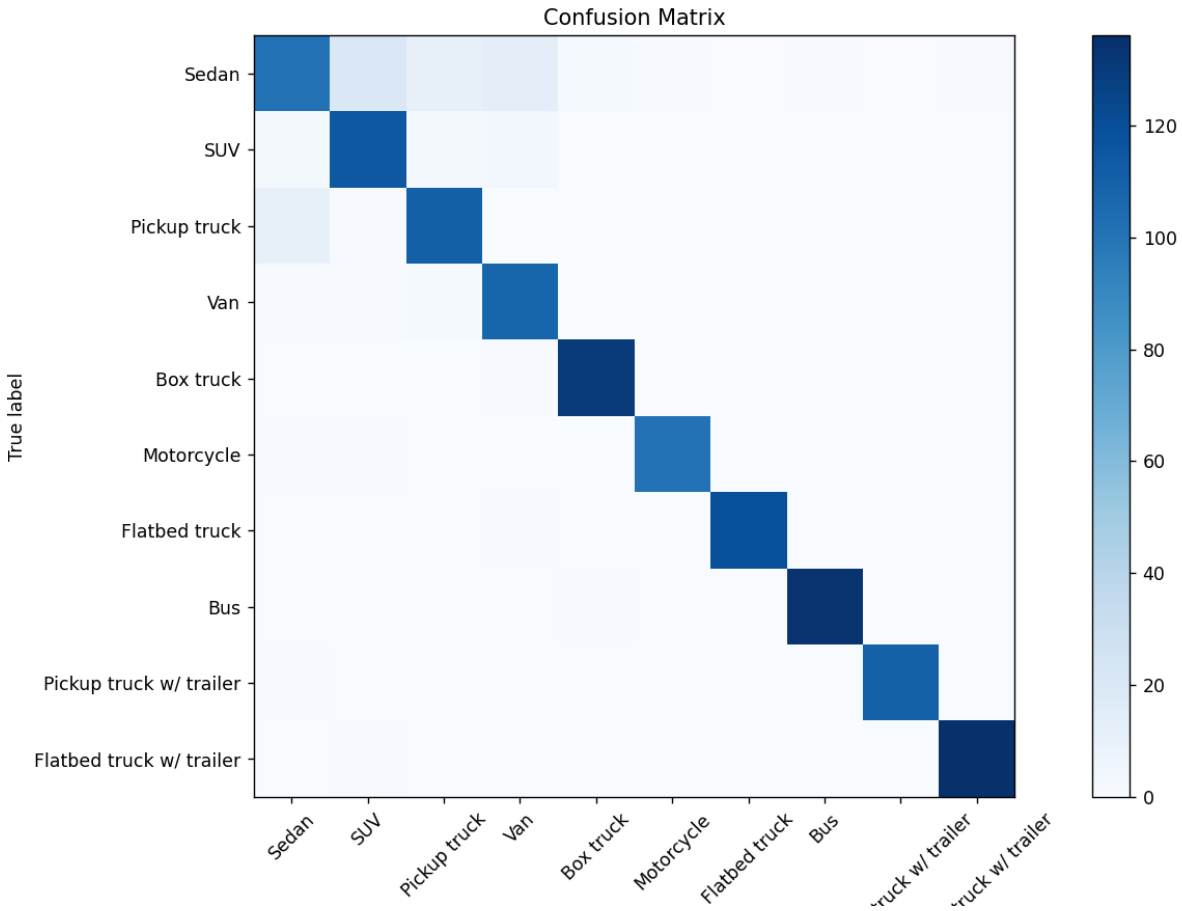


Fig 3. Confusion matrix

The model achieved an overall accuracy of 93.4% in 12 epochs, which is a good result: this percentage of the images was classified correctly.
Also the precision is good: in range of 83 – 100 percents.
Model also captures most positive samples, as we can observe from Recall column.
Finally, The F1 score provides the harmonic mean of precision and recall, providing a balanced measure of a model's performance. Confusion matrix also shows that the baseline model, in general, predicts vehicles correctly.

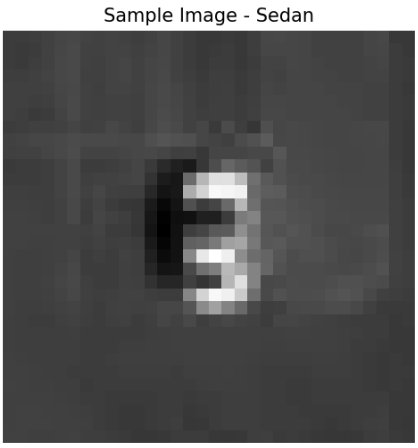
However, we can observe that for Sedan vehicle class, the performance of the model is lower: the recall is just 0.67.

a) Let's adjust and experiment with hyperparameters to obtain better result:

Experiment №	Learning rate	Epochs	Batch size	Accuracy	Comment
	0.001	12	64	0.93	The baseline
1: Change the # of convolutional layers	0.001	12	32	0.94	Adding another layer with 64 filters makes model more complex, longer time to train, result did not improve.
2: Adjust the kernel size in convolutional layers	0.001	12	32	0.94	Kernel size was set to (5,5). The accuracy did not really improve. Similar training time.
3: Apply regularization techniques	0.001	12	32	0.95	Introduced dropout rate =0.2. This improved the performance on the Sedan class and now it has recall 0.85.
4: Change the optimizer	0.001	12	32	0.95	Changing the optimizer (to RMSprop in this case) also improved the overall performance on the problematic Sedan class.
5: Adjust the learning rate	0.0001	12	32	0.92	The training this time was done in smaller steps(smoother learning graph), however, it did not really impact the performance.
6: Increase the batch size	0.001	12	128	0.93	Increasing batch size only increases the training time, but does not really impact on result.
7: Increase no. of epochs	0.001	50	32	0.95	Increasing number of epochs has improved the model performance slightly, however, training took a longer time. We can increase the learning rate and batch size to get a faster converging model.

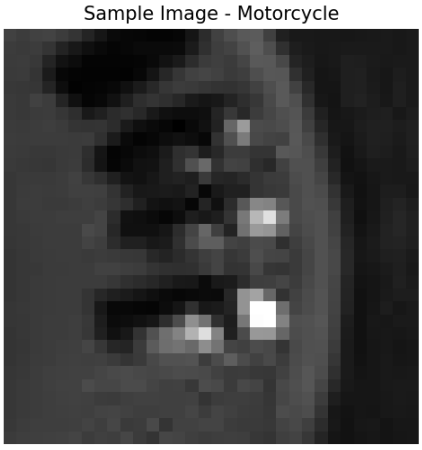
b) Results on samples:

EO_57.png (Sedan)




```
1/1 [==============================] - 0s 85ms/step
Predicted Class: Sedan
```

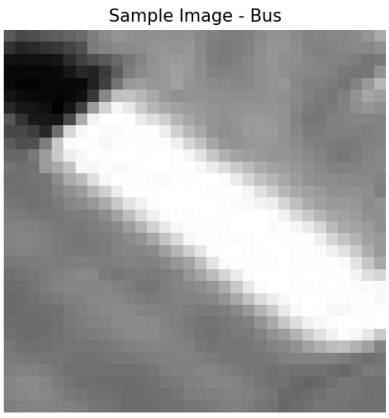
EO_438096.png (Sedan)




```
1/1 [==============================] - 0s 119ms/step
Predicted Class: Motorcycle
```

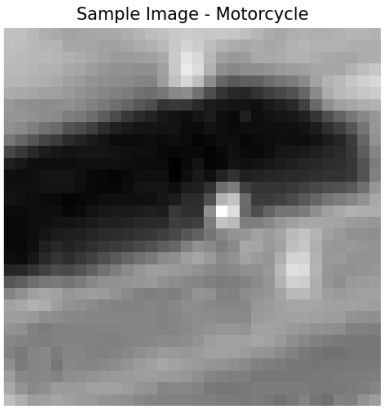
The second picture of from class Sedan was predicted incorrectly since there are multiple cars present on the sample. In addition, the model struggles to distinguish key features on dark vehicles, so that it was recognized to be most likely a motorcycle. In contrast, the model predicts pretty accurately samples with light vehicles since it can see the shape and positions of windows better.

EO_44671.png (Bus)




```
1/1 [==============================] - 0s 124ms/step
Predicted Class: Bus
```

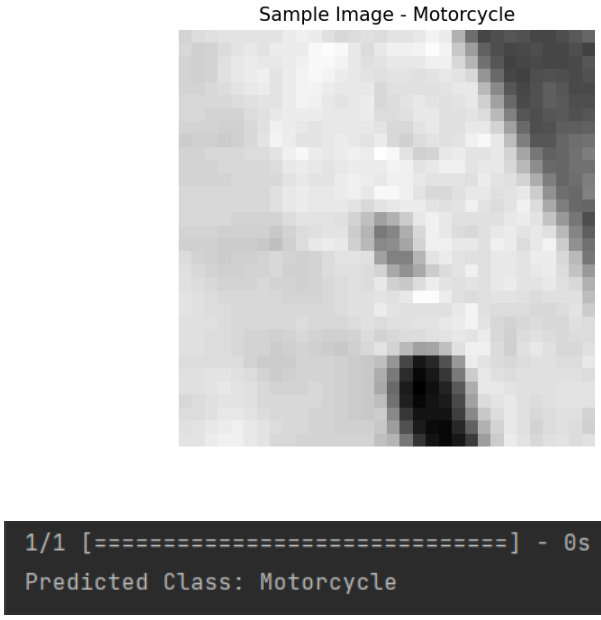
EO_44717.png (Bus)




```
1/1 [==============================] - 0s 119ms/step
Predicted Class: Motorcycle
```

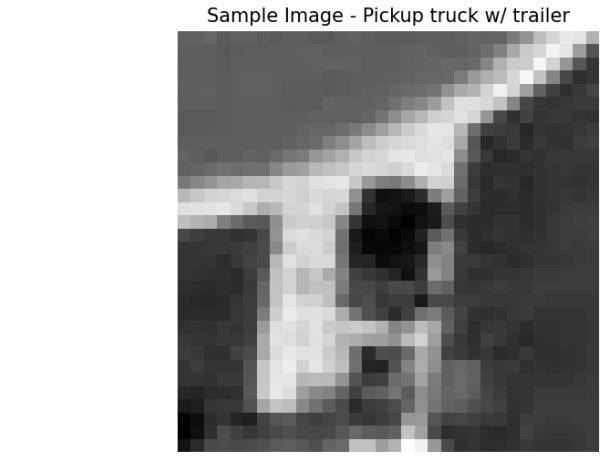
Again, the second image taken from the bus data set was predicted incorrectly because the input data consisted mostly of the pictures of “big white rectangular”, which was determined as a bus correctly in first example. The samples like second one are dark and unclear, so that we get a wrong prediction.

EO_380306.png (Motorcycle)



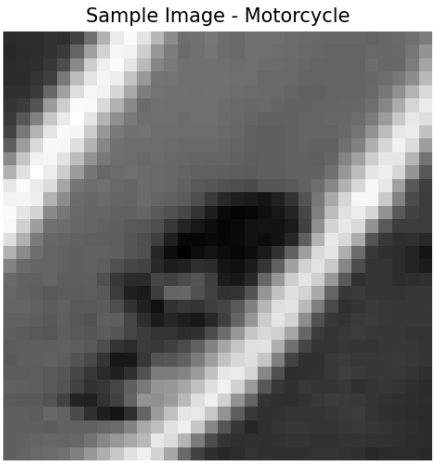

```
1/1 [==============================] - 0s 119ms/step
Predicted Class: Motorcycle
```

EO_302937.png (Pickup truck w/ trailer)



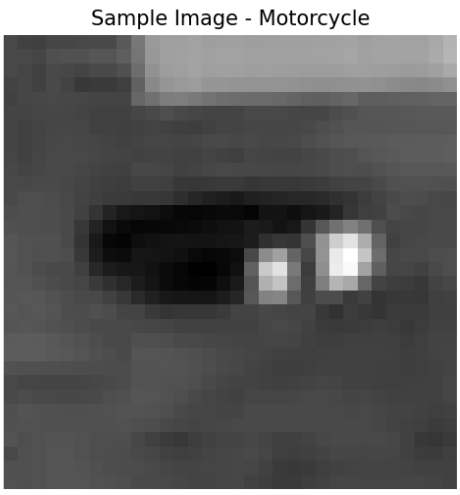

```
1/1 [==============================] - 0s 107ms/step
Predicted Class: Pickup truck w/ trailer
```

EO_58261.png (Pickup truck w/ trailer)



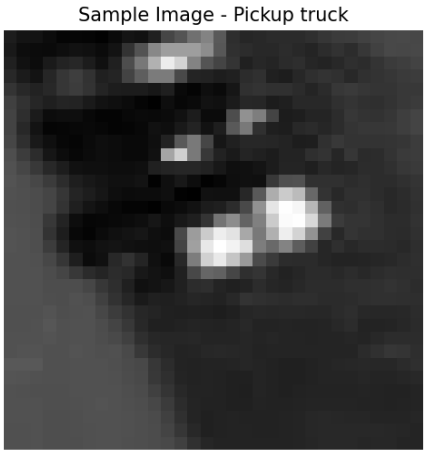

```
1/1 [==============================] - 0s 119ms/step
Predicted Class: Motorcycle
```

EO_781.png (Pickup truck)




```
1/1 [==============================] - 0s 119ms/step
Predicted Class: Motorcycle
```

EO_2175.png (Pickup truck)




```
1/1 [==============================] - 0s 101ms/step
Predicted Class: Pickup truck
```

EO_337658.png (Flatbed)


```
1/1 [==============================] - 0s 108ms/step
Predicted Class: Flatbed truck
```

For flatbed truck, the prediction was correct even with corrupted sample.

CONLCUSION

In this project, we tackled the task of multi-class image classification for vehicle recognition. We built a deep learning model using a Convolutional Neural Network (CNN) architecture and trained it on a dataset consisting of various vehicle classes.

Through extensive experimentation and optimization, we achieved promising results in terms of accuracy, performance, and learning time. However, there is still plenty of room for experimentation because the model is not ideal and may give incorrect predictions sometimes.

Prediction success strongly depends on the input sample: the image should be clear, be bright enough, preferably, should consist of a single vehicle in a frame.

The evaluation of our model on a validation set demonstrated its effectiveness in accurately classifying different types of vehicles. The classification report and confusion matrix provided insights into the model's performance for each vehicle class, showing competitive performance across most categories.

APPENDIX

main.py:

```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import sklearn.metrics as metrics
import matplotlib.pyplot as plt


# Set the path to the dataset
data_path = "C:/Users/Dell 7567/Desktop/EO"

# Define the classes and their respective labels
classes = {
    0: "Sedan",
    1: "SUV",
    2: "Pickup truck",
    3: "Van",
    4: "Box truck",
    5: "Motorcycle",
    6: "Flatbed truck",
    7: "Bus",
    8: "Pickup truck w/ trailer",
    9: "Flatbed truck w/ trailer"
}

# Define a function to load and preprocess the images
def load_image(file_path):
    image = tf.io.read_file(file_path)
    image = tf.image.decode_png(image, channels=1)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, (32, 32))
    return image

# Load the file names of the images
image_files = []
labels = []
for class_id, class_name in classes.items():
    class_path = os.path.join(data_path, str(class_id))
    file_names = os.listdir(class_path)
    file_names = [f for f in file_names if f.endswith(".png")]
    image_files.extend([os.path.join(class_path, f) for f in file_names])
    labels.extend([class_id] * len(file_names))

# Perform under-sampling
min_samples = min([labels.count(class_id) for class_id in classes])
```

```
undersampled_files = []
undersampled_labels = []
for class_id in classes.keys():
    class_files = [f for f, label in zip(image_files, labels) if label ==
class_id]
    np.random.shuffle(class_files)
    undersampled_files.extend(class_files[:min_samples])
    undersampled_labels.extend([class_id] * min_samples)

# Split the data into training and validation sets
train_files, val_files, train_labels, val_labels = train_test_split(
    undersampled_files, undersampled_labels, test_size=0.2,
random_state=42
)

# Load and preprocess the training images and labels
train_images = np.array([load_image(file) for file in train_files])
train_labels = np.array(train_labels)

# Load and preprocess the validation images and labels
val_images = np.array([load_image(file) for file in val_files])
val_labels = np.array(val_labels)

# Normalize the images
train_images = (train_images - np.mean(train_images)) /
np.std(train_images)
val_images = (val_images - np.mean(val_images)) / np.std(val_images)

# Build the model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation='relu', input_shape=(32,
32, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
   # tf.keras.layers.Dropout(0.15),  # Dropout layer added
    tf.keras.layers.Dense(len(classes), activation='softmax')
])

# Compile the model with a specific learning rate
learning_rate = 0.001
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model and track the training history
history = model.fit(train_images, train_labels, epochs=12, batch_size=64,
validation_data=(val_images, val_labels))

# Get the training history
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
training_accuracy = history.history['accuracy']
```

```
validation_accuracy = history.history['val_accuracy']

# Plot the training and validation curves
epochs = range(1, len(training_loss) + 1)

plt.figure(figsize=(12, 4))

# Plot loss curves
plt.subplot(1, 2, 1)
plt.plot(epochs, training_loss, 'b-', label='Training Loss')
plt.plot(epochs, validation_loss, 'r-', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot accuracy curves
plt.subplot(1, 2, 2)
plt.plot(epochs, training_accuracy, 'b-', label='Training Accuracy')
plt.plot(epochs, validation_accuracy, 'r-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

# Evaluate the model
val_predictions = model.predict(val_images)
val_predictions = np.argmax(val_predictions, axis=1)

# Calculate the confusion matrix
confusion_mtx = confusion_matrix(val_labels, val_predictions)

# Generate the classification report
classification_rep = metrics.classification_report(val_labels,
val_predictions, target_names=classes.values())

# Print the classification report
print("Classification Report:\n", classification_rep)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(confusion_mtx, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes.values(), rotation=45)
plt.yticks(tick_marks, classes.values())
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
# Save the trained model
model.save("model12.h5")


# Load the saved model
model = tf.keras.models.load_model("model12.h5")

# Load a sample image for prediction
sample_image_path = "C:/Users/Dell 7567/Desktop/EO/8/EO_302937.png"
sample_image = load_image(sample_image_path)
sample_image = np.expand_dims(sample_image, axis=0)

# Make predictions on the sample image
predictions = model.predict(sample_image)
predicted_label = np.argmax(predictions)

# Get the predicted class name
predicted_class = classes[predicted_label]

# Print the prediction result
print("Predicted Class:", predicted_class)

# Display the sample image
plt.imshow(sample_image[0], cmap='gray')
plt.axis('off')
plt.title("Sample Image - " + predicted_class)
plt.show()
```