College of Computer and Information Sciences
Computer Science Department

CSC 462 Machine learning

Final Report

Prepared by:

Norah Alguraishi	442201375
Majd Alqahtani	442202282
Sarah Alajlan	442202314

Section: 72917

Dr. Sarab AlMuhaideb

Table of contents

1. Introduction

In this project proposal, we outline our plan for developing an image classification model for Weather Image Recognition. We will define the image classification task, discuss our dataset selection, review relevant literature, and present our proposed approach and methodology.

1.1 Problem Definition

Our main objective is to construct an image classification model capable of categorizing weather images into one of eleven distinct classes. The problem can be summarized as follows:
Task: Multi-class Weather Image Recognition.
Objective: To assign the most suitable weather class label to a given image from a predefined set of five classes.
Domain: Weather Image Recognition, encompassing a wide range of meteorological phenomena, including dew, fog/smog, glaze, rime, and sandstorm.

2. Dataset

2.1 Dataset Selection

We have selected the following dataset for our image classification task:
Dataset Name: Weather phenomenon database (WEAPD)
Description: The "Weather Phenomenon Database (WEAPD) " is a dataset for weather classification tasks it was published in 2021 by Haixia Xiao on Harvard Dataverse, comprising a diverse collection of 11 distinct weather categories with a total of 6862 images, each class represented by a varying number of JPG format images. This dataset covers a range of atmospheric conditions, including hail, snow, lightning, rainbow, rain, dew, sandstorm, rime, frost, glaze, and fog/smog. We obtained the dataset from [1]

2.2 Data Preprocessing

We have selected 5 classes from the dataset, comprising rime, fog/smog, dew, sandstorm, and glaze.
Resize and Normalization: To ensure uniformity, we will resize the images to 224x224 pixels. Additionally. Image pixel values are normalized by dividing them by 255.0 to scale them between 0 and1.
Label Encoding: The labels are initially encoded as integer indices in the y list during data collection.
Then, they are converted to categorical format using to_categorical to create one-hot encoded vectors for y_train, y_val, and y_test. We used this approach because it is common practice in classification tasks.
Data split: The dataset is initially split into training and testing sets using train_test_split with test_size=0.25, which means 25% of the data is allocated for testing while 75% is used for training.From the training set obtained above, an additional split is performed to create a validation set. This is done using train_test_split with test_size=0.2 on the training data, resulting in 80% for training and 20% for validation from the initial training set. So to summarize, 60% of the entire dataset is for training,
15% of the entire dataset is for validation and 25% of the entire dataset is for testing.

3. Literature Review

We have conducted a review of the relevant literature to gain insights into the state-of-the-art techniques and methodologies for image classification tasks. Key papers and sources we have consulted include:

[ResNet15: Weather Recognition on Traffic Road with Deep Convolutional Neural Network] - [Jingming Xia, Dawei Xuan , Ling Tan, and Luping Xing]

The research explores the importance of recognizing weather conditions in urban traffic management. It discusses how severe weather, like rain and snow, can cause traffic problems and accidents. The review also looks at systems and technologies that use weather information to improve traffic and driving safety.

Traditional methods for recognizing weather conditions often rely on sensors, but these have limitations. So, researchers are turning to machine learning, specifically deep learning with convolutional neural networks (CNNs), as a promising solution. CNNs have shown great accuracy in identifying different weather conditions. However, they can be resource-intensive, needing large datasets and powerful computers.

To overcome these challenges, the research introduces a new method that uses deep CNNs for weather recognition on roads. They've created a medium-sized dataset of weather images, "WeatherDataset-4," covering various severe weather conditions. They've also designed a simpler CNN model called ResNet15 that balances performance and computational efficiency, making it suitable for regular computers [2].

[Food Image Classification with Convolutional Neural Network] - [M. T. Islam, B. M. N. Karim Siddique, S. Rahman and T. Jabid]

The paper discusses the use of convolutional neural networks(CNNs) for classifying food images, highlighting the significance of this technology in the context of the widespread sharing of food images on social media. The authors emphasize that a food classification system can be valuable for restaurants, beverage companies and social media platforms to target their audience with relevant advertisements.

The paper outlines the methodology, including the use of the "Food-11" dataset, image preprocessing techniques and the architecture as shown in Figure 1. It also mentions the employment of a pre-trained inception V3 model.

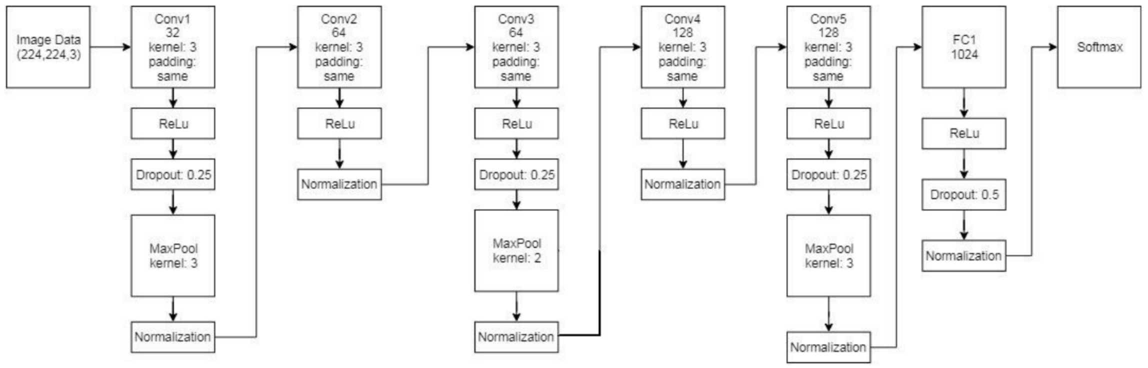


Figure 1 : Architecture of proposed Convolutional Neural Network Model.

Experimental results show that their proposed model achieved an accuracy of 74.70% while the pre-trained inception V3 model reached an accuracy of 92.86%, outperforming some other models (CNN,Alexnet and Caffenet). The paper concludes by suggesting potential improvements, such as expanding the CNN to handle a wider variety of food categories for enhanced social media food classification [3].

[Automated brain image classification based on VGG-16 and transfer learning] - [T. Kaur and T. K. Gandhi]

This paper investigates the effectiveness of transfer learning using a pre-trained VGG-16 deep neural network for classifying normal and abnormal MR brain images as presented in the dataset obtained from the Harvard Medical School repository. Earlier layers in VGG-16 capture general features, while later layers specialize in class-specific details. Thus, fine-tuning is concentrated on the last three layers of the VGG-16 model. In this approach, these layers were replaced with a fully connected layer, a softmax layer, and a classification output layer, with the fully connected layer's size adjusted to the new dataset's class count.

The study employs a 10-fold cross-validation technique to partition the dataset and evaluate the model's performance. The primary performance metrics used are sensitivity (Se), specificity (Sp), and accuracy (Acc). The results obtained illustrate the exceptional performance of the pre-trained VGG-16 model with transfer learning compared to existing state-of-the-art methods. In particular, the model achieves 100% Se, Sp, and Acc for the recognition of brain images, outperforming other techniques with lower accuracy values.

The conclusion of the paper highlights the success of the pre-trained VGG-16 model with transfer learning in achieving a perfect recognition rate for classifying abnormal MR brain images. The authors acknowledge the lengthy training process as a limitation and suggest that future research should explore alternative pre-trained models while assessing their computational complexities [4].

4. Proposed Approach and Methodology

4.1 Model Selection

For our multi-class image classification task, we intend to use a convolutional neural network (CNN) architecture. Specifically, we will adopt the ResNet15 architecture proposed in [2] as our chosen model for the task of weather image classification, a simplified and enhanced version of the ResNet50. The ResNet architecture has proven to be highly effective for image classification tasks. The ResNet architecture is based on the concept of residual modules, which enable the successful training of very deep networks. By introducing skip connections and residual connections, these modules address the vanishing gradient problem and allow for a straightforward increase in network depth without introducing additional parameters. This leads to improved convergence speed and accuracy.

4.2 Model Design

The architecture of ResNet50 starts with a 7×7 convolutional layer followed by a 3×3 MaxPooling layer and comprises residual modules. These modules come in two fundamental stacking modes: Identity Blocks (IB), in which input and output can be connected in series because of having the same dimensions. Convolution Blocks (CB), here input and output can not be connected in series because of different dimensions, and a function is needed to change the dimensions of the feature vector with a size of 1×1. Image classification and recognition are performed using the AveragePooling layer and a Softmax classifier.

ResNet15 is an optimized version of ResNet50, preserving vital components while enhancing efficiency. As shown in Figure 2 it retains the 7×7 convolutional layer, 3×3 MaxPooling layer, and specific Convolution Blocks for effective feature extraction. To simplify the architecture, some Identity Blocks are removed. Additionally, ResNet15 replaces AveragePooling with a fully connected layer featuring 512 dimensions, introducing a dropout layer to enhance generalization and reduce overfitting. The Softmax classifier remains the same, ensuring accurate classification.

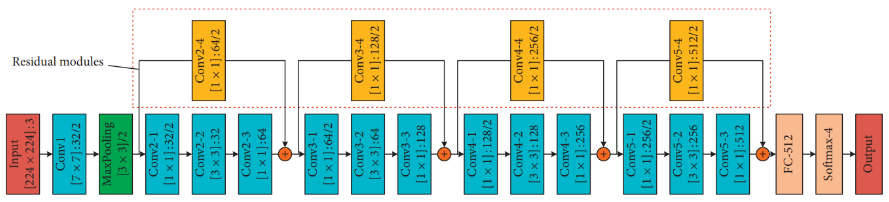


Figure 2. Architecture of ResNet15 [2].

4.3 Training and Evaluation

We will commence the model development process by dividing our dataset into three essential subsets: training, validation, and test sets. The training set will be utilized to train the model, while the validation set will serve as a means to monitor its performance during training. This iterative monitoring process will involve assessing various evaluation metrics, such as accuracy, precision, recall, and the generation of a detailed confusion matrix.

In addition to these steps, we recognize the significance of hyperparameter tuning in optimizing the model's performance. One key hyperparameter that we will focus on is the learning rate. Learning rate plays a crucial role in determining the size of the steps taken during the optimization process. To enhance the model's learning capabilities, we will systematically explore and fine-tune the learning rate through hyperparameter tuning.

The step decay scheduler dynamically adapts the learning rate at predefined intervals, fostering more effective convergence during training. This iterative process entails adjusting the learning rate based on the model's ongoing performance on the validation set, followed by subsequent model training and evaluation.

5. Experimental Results

In this section, presents an analysis of the experimental results obtained from our machine learning model. The model, trained on a designated dataset, has undergone evaluation on training, validation, and test sets. The metrics reported include training loss, accuracy, recall, and precision, as well as corresponding values for the validation and test sets. Additionally, we will provide a comparison based on the tuning of learning rates to offer insights into the impact of hyperparameter adjustments on the model's performance.

5.1 Performance Metrics

The machine learning model exhibits good performance across training, validation, and test datasets, as indicated by the following metrics:

Training Set:
As shown in Table 1. The model demonstrates a high degree of fit to the training data, accurately classifying instances while maintaining a good balance between identifying positive cases (recall) and avoiding false positives (precision).

Validation Set:
The model generalizes well to unseen data, with a slightly higher loss than in training, indicating reasonable performance on the validation set. Accuracy, recall, and precision metrics remain at acceptable levels.

Test Set:
The model maintains consistent performance on the test set, further validating its generalization capabilities. Loss values and classification metrics align with expectations, suggesting reliable predictive capabilities.

Table 1. Performance Measures for Training, Validation, and Testing.

metric	Training	Validation	Testing
Accuracy	**92.57%**	**84.32%**	**84.75%**
Loss	**0.2267**	**0.4094**	**0.4405**
Recall	**91.79%**	**83%**	**83.27%**
Precision	**93.64%**	**86.28%**	**86.08%**

5.2 Comparisons

In order to enhance the model's performance, we fine-tuned the learning rate during training to achieve improved results and ensure better convergence and more accurate predictions. Through this process, we adjusted the learning rate dynamically using a step decay schedule with initial_learning_rate = 0.01, drop = 0.5, and epochs_drop = 5. Table 2 presents a detailed comparison of training accuracies over different epochs. It illustrates a gradual increase in model accuracy over consecutive five-epoch intervals, highlighting the effectiveness of the step decay learning rate schedule. Starting with an initial accuracy of 56.15% in the first epoch, the model consistently improves, reaching 92.57% by the 21st epoch. The patterns identified in Table 2 are further emphasized by the visual representation in Figure 3.

Table 2. Comparison between different learning rates.

Epochs	Learning rate	Average accuracies
1 - 5	**0.0100**	**70.99%**
6 - 10	**0.0050**	**83.49%**
11 - 15	**0.0025**	**88.16%**
16 - 20	**0.0012**	**91.60%**
21 - 25	**6.2500e-04**	**Early stop on epoch 21 with accuracy 92.57%**

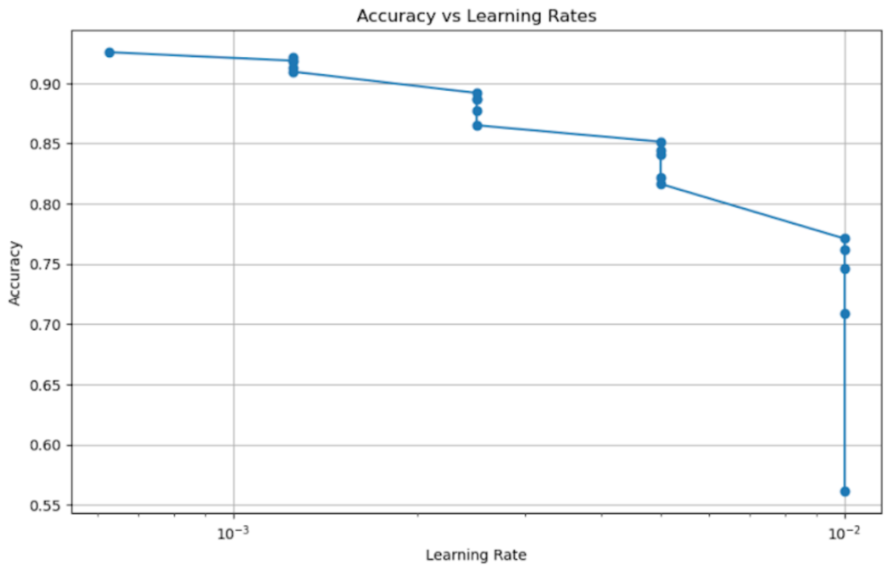


Figure 3. Accuracy vs Learning Rates

6. Discussion

6.1 Strengths

High Accuracy: The model demonstrates high accuracy on both the training and validation sets, indicating its effectiveness in making correct predictions.

Balanced Precision and Recall: The model achieves a good balance between precision and recall, crucial for applications where minimizing false positives and false negatives are equally important.
Generalization: The model generalizes well to the test set, suggesting that it can make accurate predictions on unseen data.

6.2 Limitations

Dataset Imbalance: The distribution of classes within the dataset is uneven, meaning that certain categories or labels have a higher or lower number of instances compared to others. This imbalance can impact the model's ability to learn equally from all classes.

Challenges with Certain Classes: Classes such as 'fogsmog,' 'dew,' and 'glaze' exhibit a notable frequency of misclassifications. It's worth emphasizing that the 'glaze' class encounters a high rate of misclassification, where the classifier often interprets it as 'rime,' and conversely, the 'rime' class experiences a significant misclassification rate, being erroneously labeled as 'glaze.' This highlights a specific challenge in the model's ability to accurately distinguish between 'glaze' and 'rime' classes.

6.3 Future Improvements

Hyperparameter Tuning: Further exploration of hyperparameter tuning, including dropout rates and batch sizes, may optimize the model's performance and make it less sensitive to these parameters.

Ensemble Methods: Exploring ensemble methods, combining predictions from multiple models, can enhance overall performance and provide more robust results.

Cross-Validation Techniques: Utilizing advanced cross-validation techniques, such as stratified k-fold cross-validation, can provide a more robust estimate of the model's performance, especially in cases of limited data.

Transfer Learning: Implementing transfer learning using pre-trained models on larger datasets can leverage knowledge from related domains, potentially improving the model's performance.

7. Visualizations

Here is the classifier predictions on random images

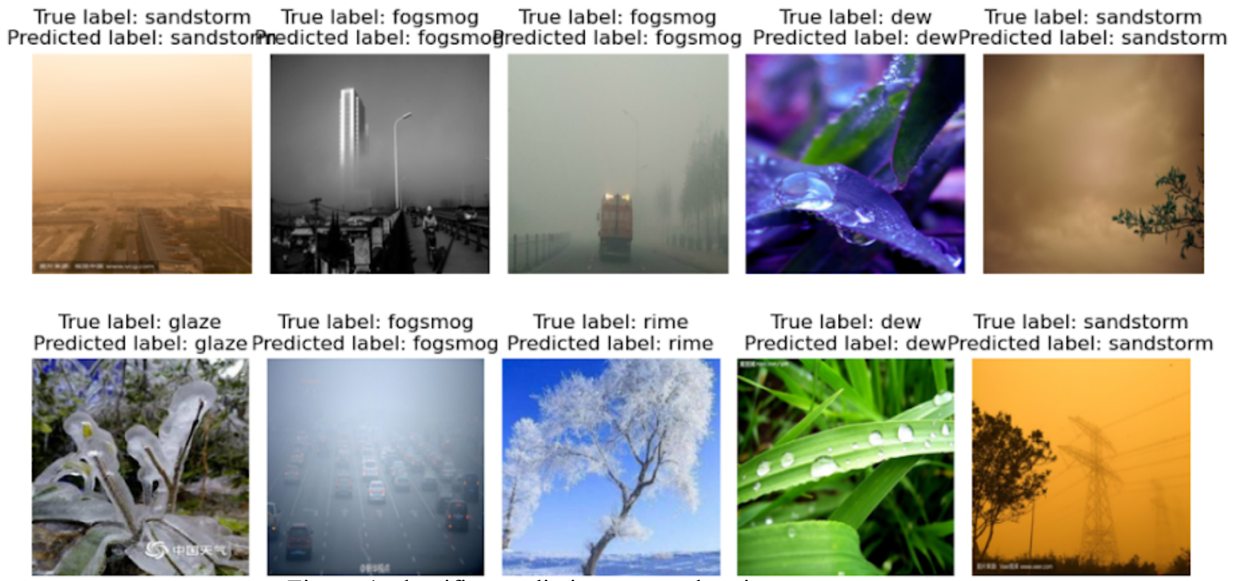


Figure 4. classifier predictions on random images

Confusion matrix:

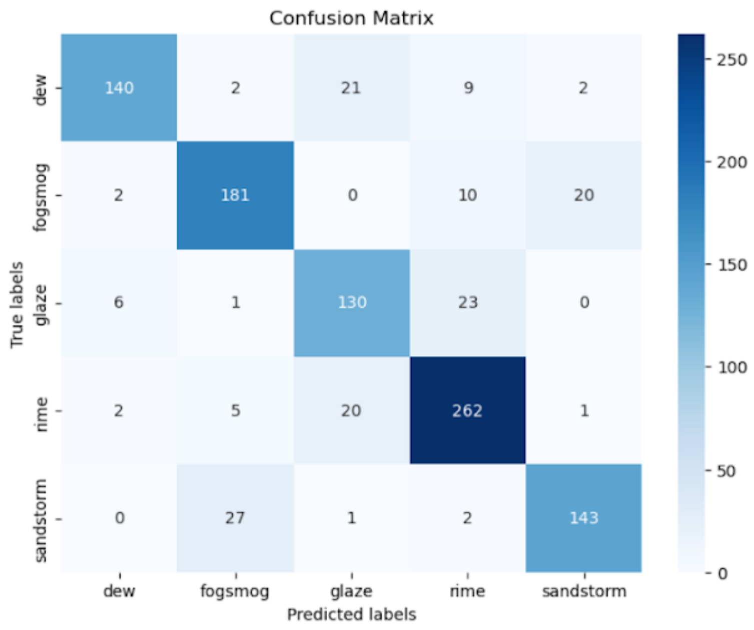


Figure 5. Confusion matrix

8. Conclusion

In general, the model demonstrates strong performance across various evaluation sets, showcasing its capacity to generalize well and provide accurate predictions. These findings are promising for the model's potential use in practical situations. To sum up, although the model displays commendable strengths, it is important to tackle its limitations by employing methods such as regularization and hyperparameter tuning. Additionally, exploring enhancements like data augmentation and ensemble methods could further improve the model's overall effectiveness and versatility in diverse real-world contexts.

9. References

[1] Jehan Bhathena, “Weather Image Recognition,” *Kaggle.com*, 2021.
https://www.kaggle.com/datasets/jehanbhathena/weather-dataset/data (accessed Nov. 05, 2023).

[2] J. Xia, D. Xuan, L. Tan, and L. Xing, “ResNet15: Weather recognition on traffic road with deep convolutional neural network,” Adv. Meteorol., vol. 2020, pp. 1–11, 2020.

[3] M. T. Islam, B. M. N. Karim Siddique, S. Rahman and T. Jabid, "Food Image Classification with Convolutional Neural Network," 2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Bangkok, Thailand, 2018, pp. 257-262, doi: 10.1109/ICIIBMS.2018.8550005

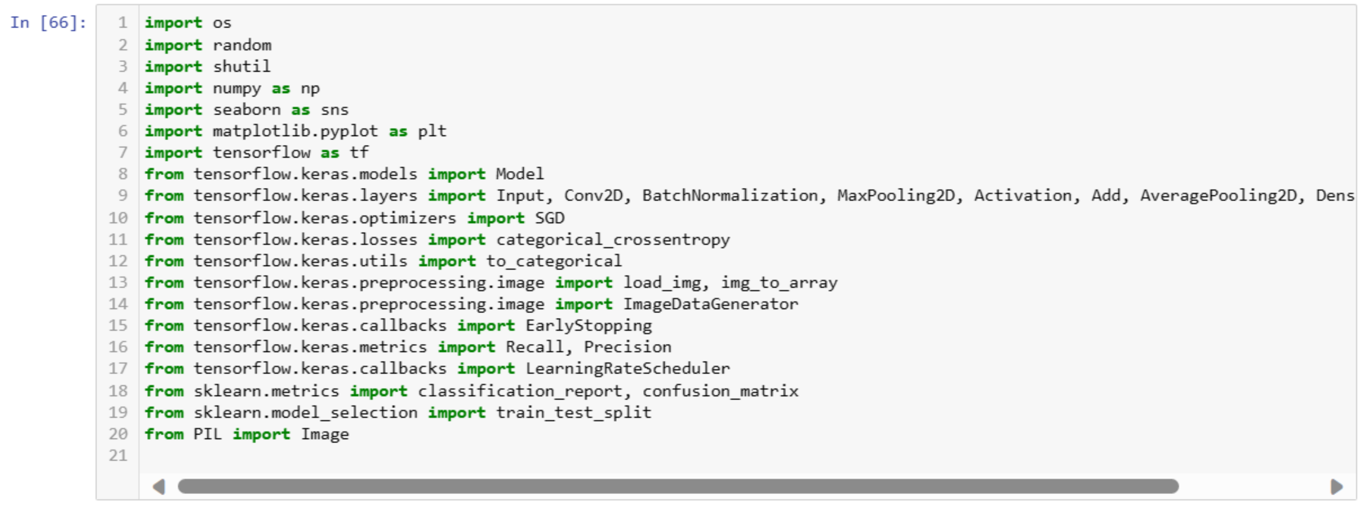
[4] T. Kaur and T. K. Gandhi, "Automated Brain Image Classification Based on VGG-16 and Transfer Learning," 2019 International Conference on Information Technology (ICIT), 2019, pp. 94-98, doi: 10.1109/ICIT48102.2019.00023.

10. Statement of Work Distribution

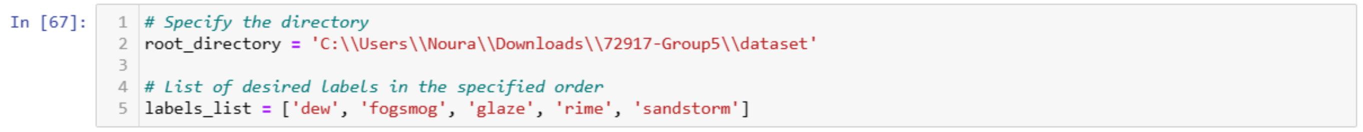
Table 3. Task distribution.

Shared responsibilities	Task	Students
Classification task, Testing and Literature Review.	Splitting the data, building residual blocks,define learning rate schedule,callbacks, displaying the confusion matrix and Discussion	**Norah Alguraishi**
	Converting label to categorical, building the model, early stopping,fitting the model, showing few test images and their predicted class and Model Design	**Majd Alqahtani**
	Normalizing and generate batches of image, model compilation, plot the accuracy vs. learning rate,printing the model evaluation, Data Configuration and Visualizations	**Sarah Alajlan**

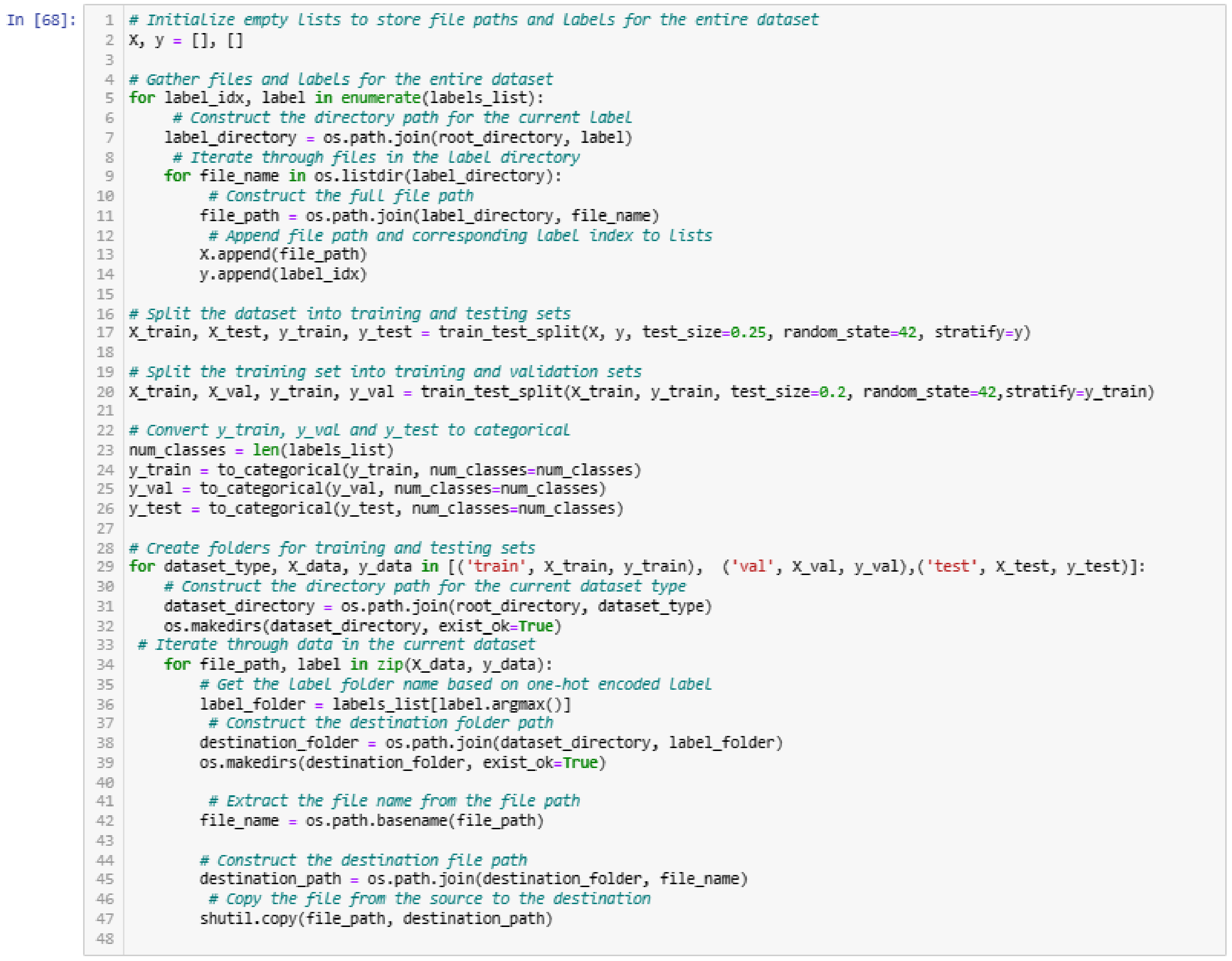
11. Appendix



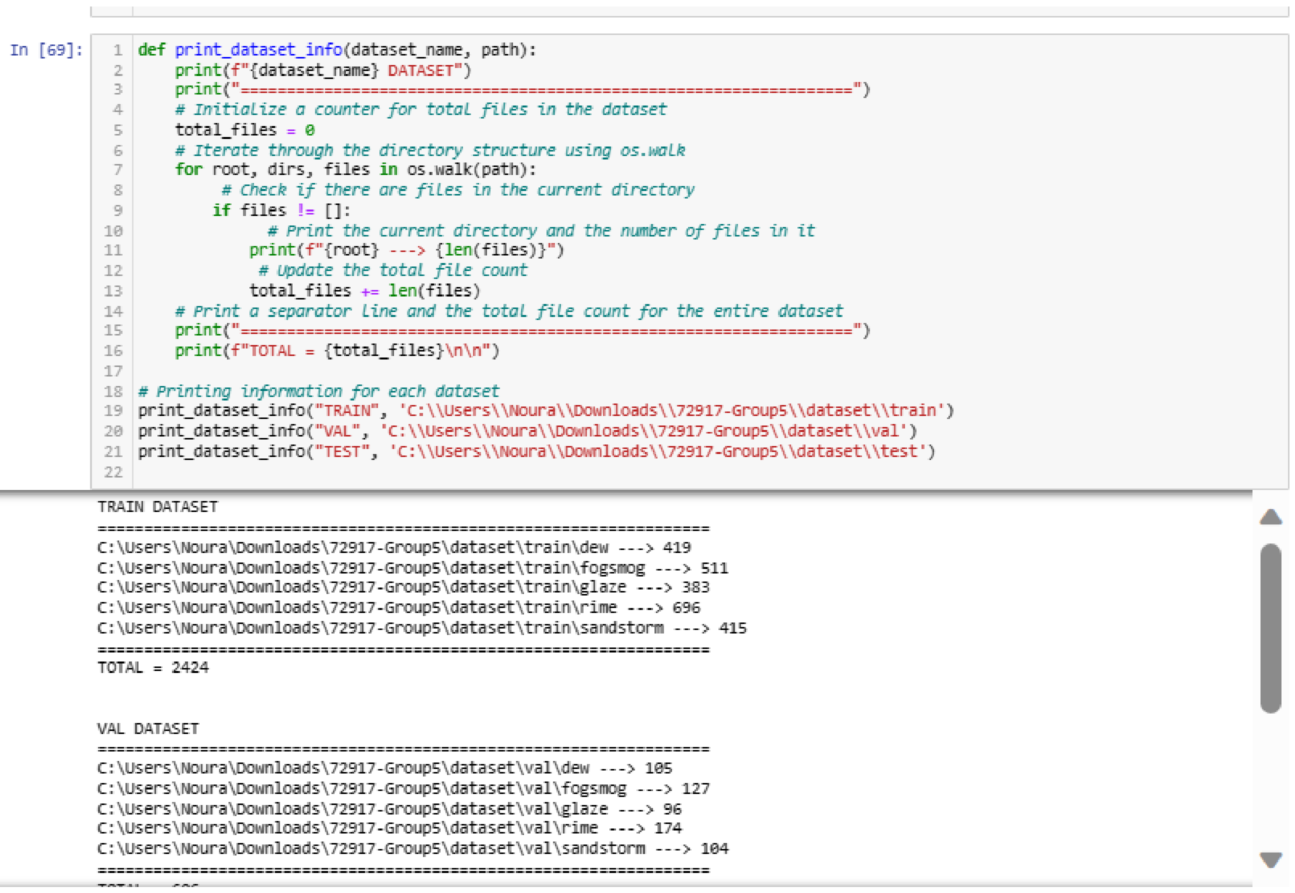
```
In [66]:
import os
import random
import shutil
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, MaxPooling2D, Activation, Add, AveragePooling2D, Dens
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import Recall, Precision
from tensorflow.keras.callbacks import LearningRateScheduler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from PIL import Image

```



```
In [67]:
# Specify the directory
root_directory = 'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset'

# List of desired labels in the specified order
labels_list = ['dew', 'fogsmog', 'glaze', 'rime', 'sandstorm']
```

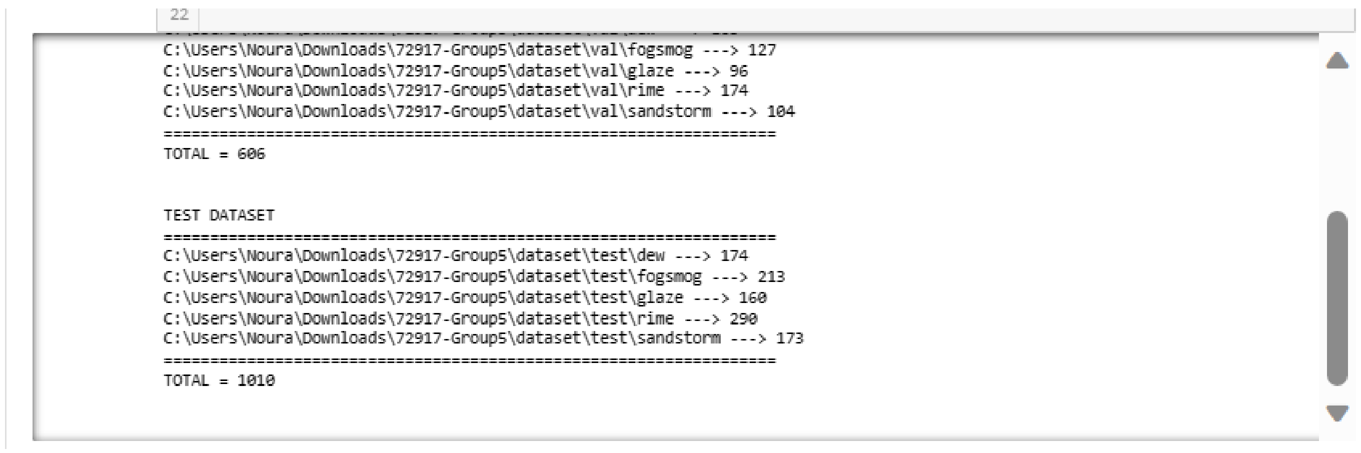


```
In [68]:
# Initialize empty lists to store file paths and labels for the entire dataset
X, y = [], []

# Gather files and labels for the entire dataset
for label_idx, label in enumerate(labels_list):
     # Construct the directory path for the current label
    label_directory = os.path.join(root_directory, label)
     # Iterate through files in the label directory
    for file_name in os.listdir(label_directory):
         # Construct the full file path
        file_path = os.path.join(label_directory, file_name)
         # Append file path and corresponding label index to lists
        X.append(file_path)
        y.append(label_idx)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)

# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42,stratify=y_train)

# Convert y_train, y_val and y_test to categorical
num_classes = len(labels_list)
y_train = to_categorical(y_train, num_classes=num_classes)
y_val = to_categorical(y_val, num_classes=num_classes)
y_test = to_categorical(y_test, num_classes=num_classes)

# Create folders for training and testing sets
for dataset_type, X_data, y_data in [('train', X_train, y_train),  ('val', X_val, y_val),('test', X_test, y_test)]:
    # Construct the directory path for the current dataset type
    dataset_directory = os.path.join(root_directory, dataset_type)
    os.makedirs(dataset_directory, exist_ok=True)
 # Iterate through data in the current dataset
    for file_path, label in zip(X_data, y_data):
        # Get the label folder name based on one-hot encoded label
        label_folder = labels_list[label.argmax()]
         # Construct the destination folder path
        destination_folder = os.path.join(dataset_directory, label_folder)
        os.makedirs(destination_folder, exist_ok=True)

         # Extract the file name from the file path
        file_name = os.path.basename(file_path)

        # Construct the destination file path
        destination_path = os.path.join(destination_folder, file_name)
         # Copy the file from the source to the destination
        shutil.copy(file_path, destination_path)

```



```
In [69]:
def print_dataset_info(dataset_name, path):
    print(f"{dataset_name} DATASET")
    print("=================================================================")
    # Initialize a counter for total files in the dataset
    total_files = 0
    # Iterate through the directory structure using os.walk
    for root, dirs, files in os.walk(path):
         # Check if there are files in the current directory
        if files != []:
              # Print the current directory and the number of files in it
            print(f"{root} ---> {len(files)}")
             # Update the total file count
            total_files += len(files)
    # Print a separator line and the total file count for the entire dataset
    print("=================================================================")
    print(f"TOTAL = {total_files}\n\n")

# Printing information for each dataset
print_dataset_info("TRAIN", 'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\train')
print_dataset_info("VAL", 'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\val')
print_dataset_info("TEST", 'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\test')
```

```
TRAIN DATASET
=================================================================
C:\Users\Noura\Downloads\72917-Group5\dataset\train\dew ---> 419
C:\Users\Noura\Downloads\72917-Group5\dataset\train\fogsmog ---> 511
C:\Users\Noura\Downloads\72917-Group5\dataset\train\glaze ---> 383
C:\Users\Noura\Downloads\72917-Group5\dataset\train\rime ---> 696
C:\Users\Noura\Downloads\72917-Group5\dataset\train\sandstorm ---> 415
=================================================================
TOTAL = 2424


VAL DATASET
=================================================================
C:\Users\Noura\Downloads\72917-Group5\dataset\val\dew ---> 105
C:\Users\Noura\Downloads\72917-Group5\dataset\val\fogsmog ---> 127
C:\Users\Noura\Downloads\72917-Group5\dataset\val\glaze ---> 96
C:\Users\Noura\Downloads\72917-Group5\dataset\val\rime ---> 174
C:\Users\Noura\Downloads\72917-Group5\dataset\val\sandstorm ---> 104
=================================================================
```



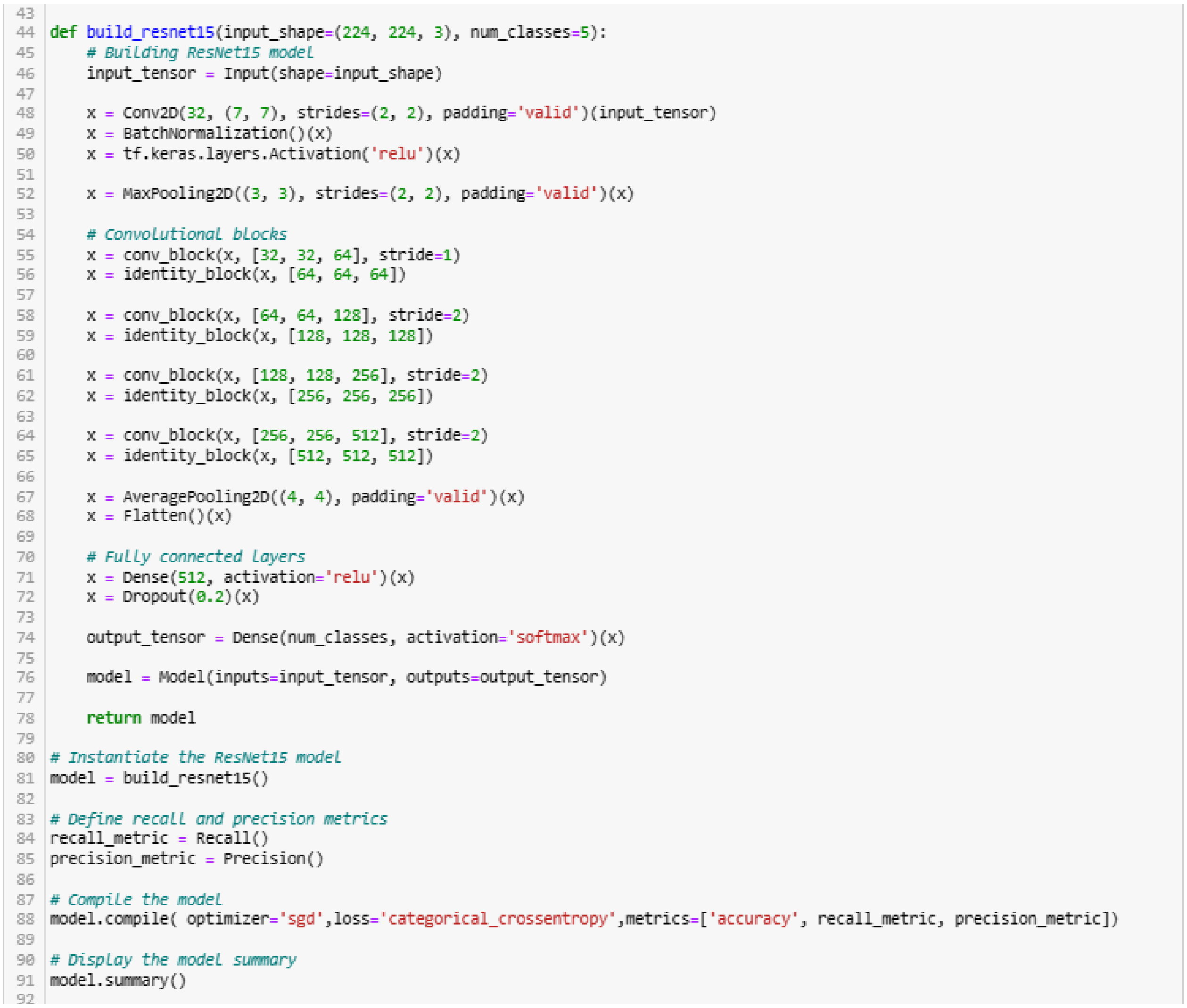
```
C:\Users\Noura\Downloads\72917-Group5\dataset\val\fogsmog ---> 127
C:\Users\Noura\Downloads\72917-Group5\dataset\val\glaze ---> 96
C:\Users\Noura\Downloads\72917-Group5\dataset\val\rime ---> 174
C:\Users\Noura\Downloads\72917-Group5\dataset\val\sandstorm ---> 104
=================================================================
TOTAL = 606


TEST DATASET
=================================================================
C:\Users\Noura\Downloads\72917-Group5\dataset\test\dew ---> 174
C:\Users\Noura\Downloads\72917-Group5\dataset\test\fogsmog ---> 213
C:\Users\Noura\Downloads\72917-Group5\dataset\test\glaze ---> 160
C:\Users\Noura\Downloads\72917-Group5\dataset\test\rime ---> 290
C:\Users\Noura\Downloads\72917-Group5\dataset\test\sandstorm ---> 173
=================================================================
TOTAL = 1010
```

In [70]:
```python
# Define data generators for training, validation, and testing sets
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Training Data Generator
train_generator = train_datagen.flow_from_directory(
   'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\train',
    target_size=(224, 224), # Resize images to (224, 224) during loading
    batch_size=32, # Number of samples per batch
    class_mode='categorical', # Categorical labels for multi-class classification
    shuffle=True  # Shuffle the data for better training
)
# Validation Data Generator
val_generator = val_datagen.flow_from_directory(
   'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\val',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False  # No need to shuffle the validation data
)
# Test Data Generator
test_generator = test_datagen.flow_from_directory(
   'C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False  # No need to shuffle the test data
)

```

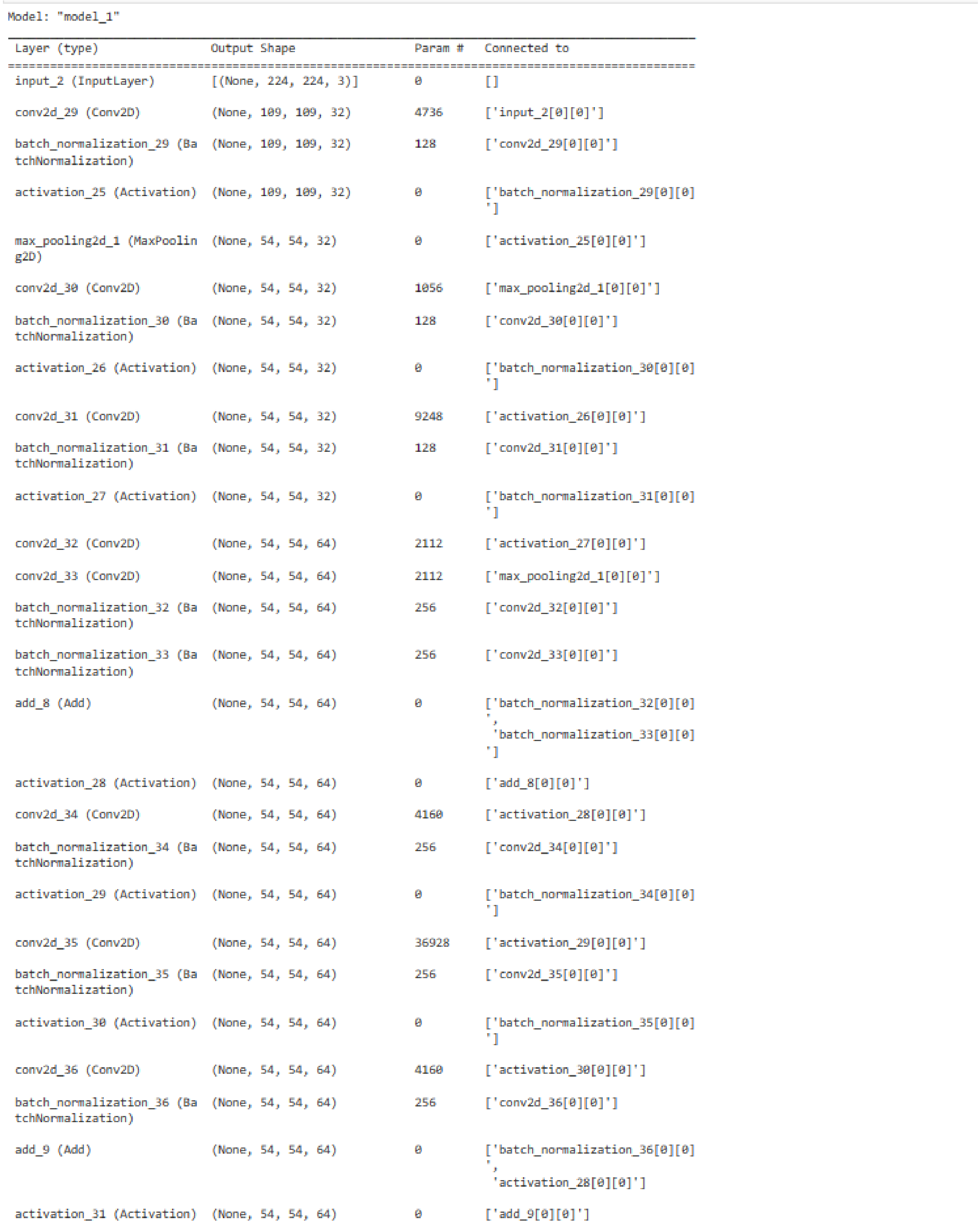
```
Found 2424 images belonging to 5 classes.
Found 606 images belonging to 5 classes.
Found 1010 images belonging to 5 classes.
```

In [71]:
```python
# Check GPU availability
# Get a list of available physical devices (e.g., GPUs)
physical_devices = tf.config.list_physical_devices('GPU')
# Check if there are any GPUs available
if physical_devices:
    # Enable dynamic GPU memory growth to allocate memory as needed
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

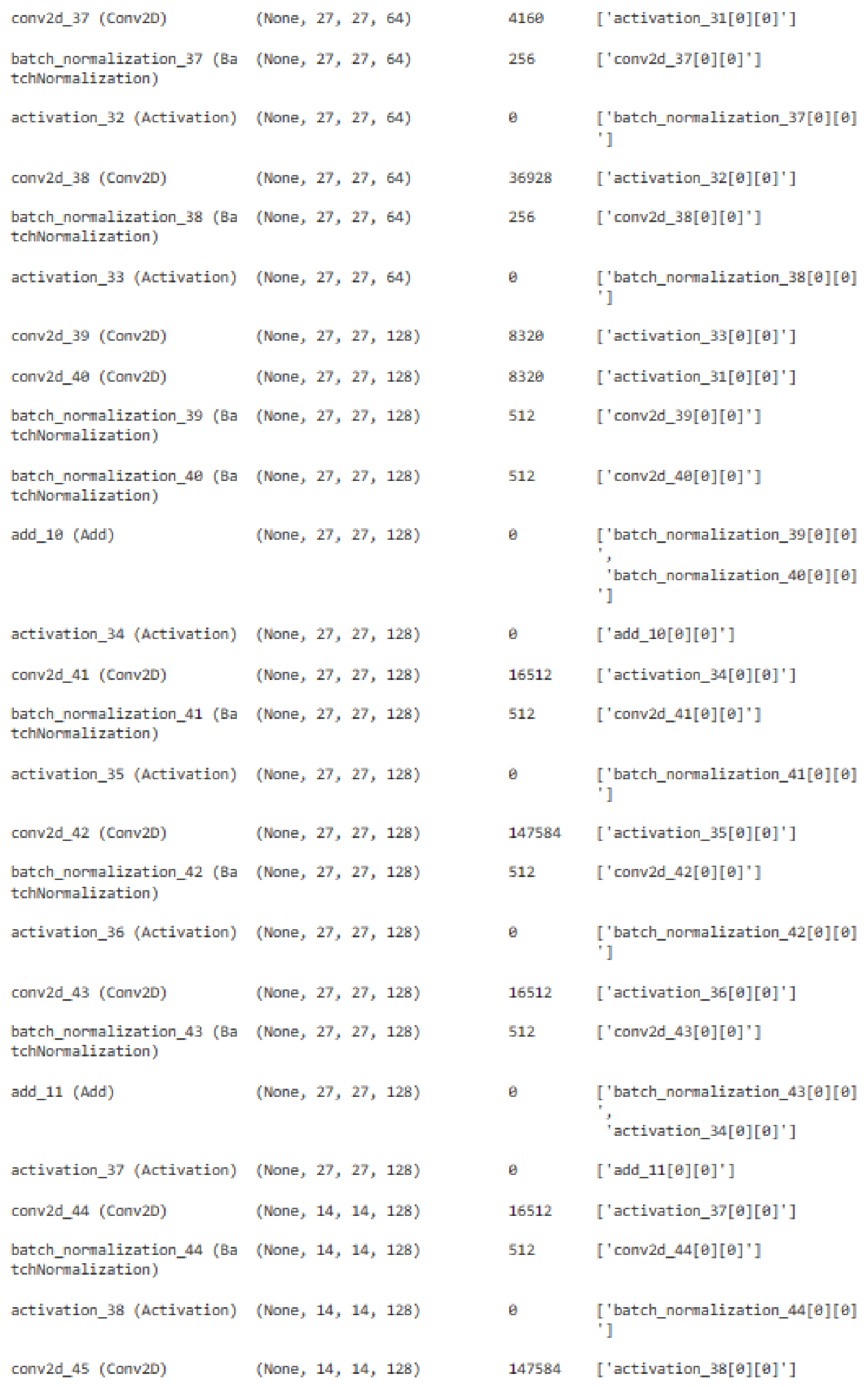
In [72]:
```python
def identity_block(x, filters):
    # Identity block with three convolutional layers
    x_shortcut = x

    x = Conv2D(filters[0], (1, 1), strides=(1, 1), padding='valid')(x)
    x = BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)

    x = Conv2D(filters[1], (3, 3), strides=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)

    x = Conv2D(filters[2], (1, 1), strides=(1, 1), padding='valid')(x)
    x = BatchNormalization()(x)

    x = Add()([x, x_shortcut])
    x = tf.keras.layers.Activation('relu')(x)

    return x

def conv_block(x, filters, stride):
    # Convolutional block with shortcut connection
    x_shortcut = x

    x = Conv2D(filters[0], (1, 1), strides=(stride, stride), padding='valid')(x)
    x = BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)

    x = Conv2D(filters[1], (3, 3), strides=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)

    x = Conv2D(filters[2], (1, 1), strides=(1, 1), padding='valid')(x)
    x = BatchNormalization()(x)

    x_shortcut = Conv2D(filters[2], (1, 1), strides=(stride, stride), padding='valid')(x_shortcut)
    x_shortcut = BatchNormalization()(x_shortcut)

    x = Add()([x, x_shortcut])
    x = tf.keras.layers.Activation('relu')(x)

    return x
```



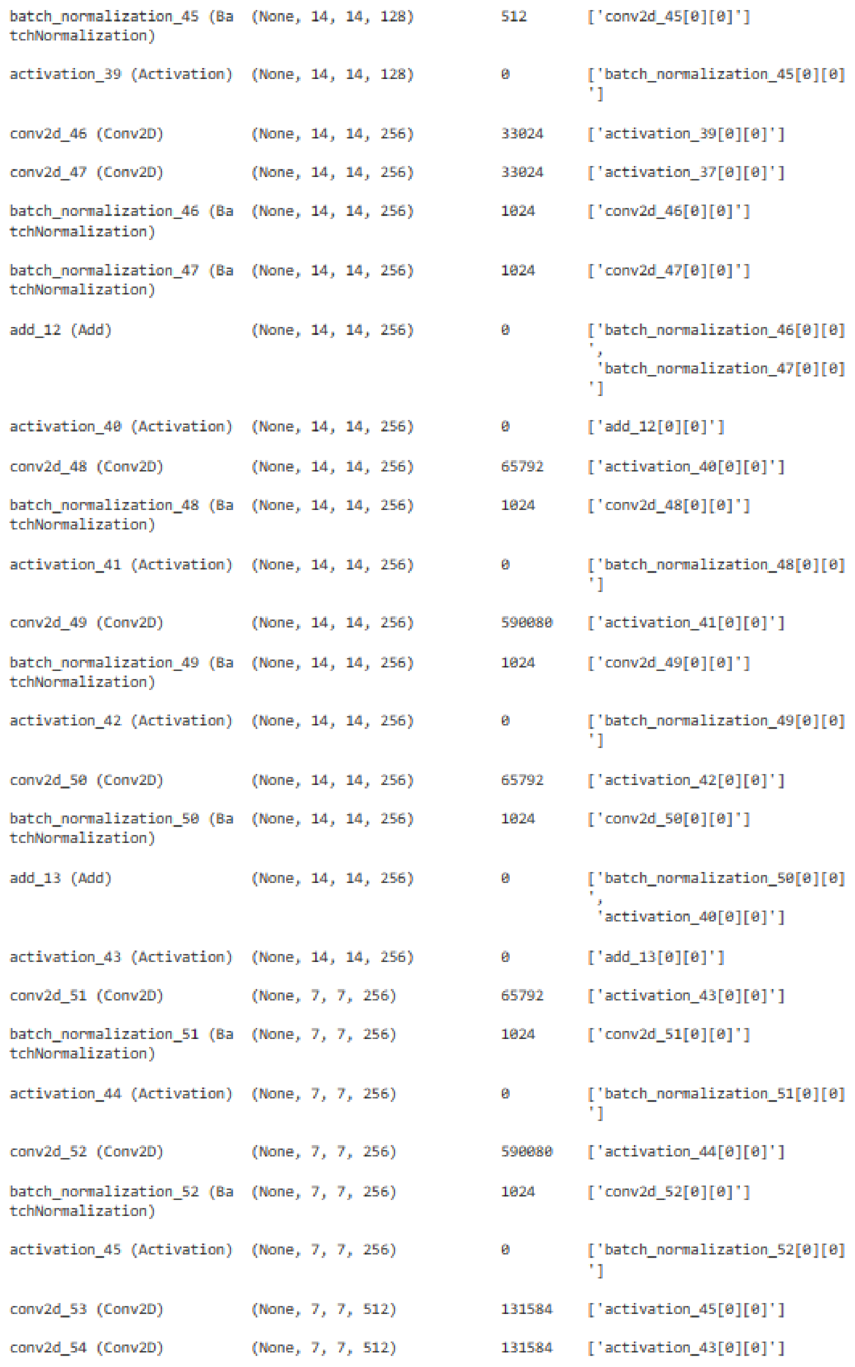
```python

def build_resnet15(input_shape=(224, 224, 3), num_classes=5):
    # Building ResNet15 model
    input_tensor = Input(shape=input_shape)

    x = Conv2D(32, (7, 7), strides=(2, 2), padding='valid')(input_tensor)
    x = BatchNormalization()(x)
    x = tf.keras.layers.Activation('relu')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='valid')(x)

    # Convolutional blocks
    x = conv_block(x, [32, 32, 64], stride=1)
    x = identity_block(x, [64, 64, 64])

    x = conv_block(x, [64, 64, 128], stride=2)
    x = identity_block(x, [128, 128, 128])

    x = conv_block(x, [128, 128, 256], stride=2)
    x = identity_block(x, [256, 256, 256])

    x = conv_block(x, [256, 256, 512], stride=2)
    x = identity_block(x, [512, 512, 512])

    x = AveragePooling2D((4, 4), padding='valid')(x)
    x = Flatten()(x)

    # Fully connected layers
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.2)(x)

    output_tensor = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=input_tensor, outputs=output_tensor)

    return model

# Instantiate the ResNet15 model
model = build_resnet15()

# Define recall and precision metrics
recall_metric = Recall()
precision_metric = Precision()

# Compile the model
model.compile( optimizer='sgd',loss='categorical_crossentropy',metrics=['accuracy', recall_metric, precision_metric])

# Display the model summary
model.summary()

```



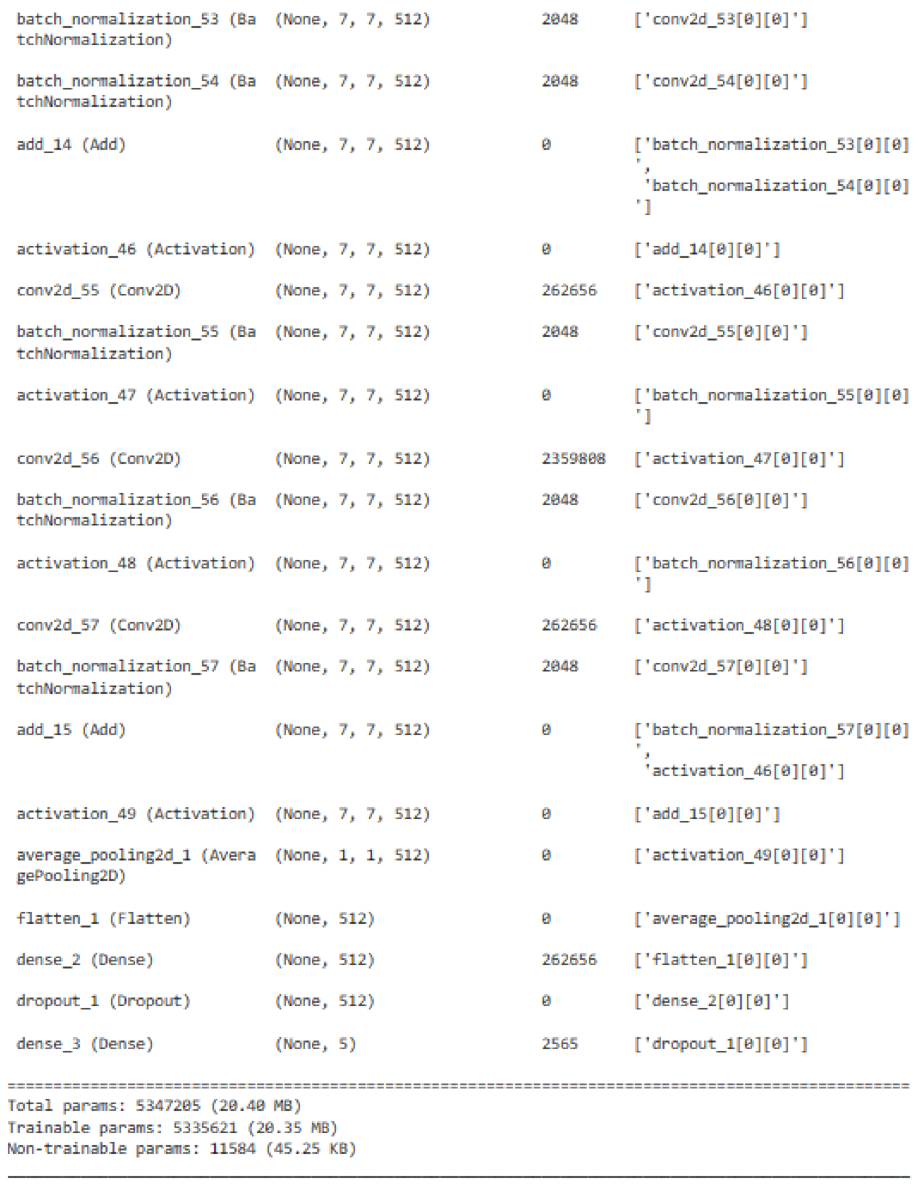
```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 224, 224, 3)]  0          []

 conv2d_29 (Conv2D)             (None, 109, 109, 32)  4736        ['input_2[0][0]']

 batch_normalization_29 (Ba     (None, 109, 109, 32)  128         ['conv2d_29[0][0]']
 tchNormalization)

 activation_25 (Activation)     (None, 109, 109, 32)  0           ['batch_normalization_29[0][0]
                                                                  ']

 max_pooling2d_1 (MaxPoolin     (None, 54, 54, 32)   0           ['activation_25[0][0]']
 g2D)

 conv2d_30 (Conv2D)             (None, 54, 54, 32)   1056        ['max_pooling2d_1[0][0]']

 batch_normalization_30 (Ba     (None, 54, 54, 32)   128         ['conv2d_30[0][0]']
 tchNormalization)

 activation_26 (Activation)     (None, 54, 54, 32)   0           ['batch_normalization_30[0][0]
                                                                  ']

 conv2d_31 (Conv2D)             (None, 54, 54, 32)   9248        ['activation_26[0][0]']

 batch_normalization_31 (Ba     (None, 54, 54, 32)   128         ['conv2d_31[0][0]']
 tchNormalization)

 activation_27 (Activation)     (None, 54, 54, 32)   0           ['batch_normalization_31[0][0]
                                                                  ']

 conv2d_32 (Conv2D)             (None, 54, 54, 64)   2112        ['activation_27[0][0]']

 conv2d_33 (Conv2D)             (None, 54, 54, 64)   2112        ['max_pooling2d_1[0][0]']

 batch_normalization_32 (Ba     (None, 54, 54, 64)   256         ['conv2d_32[0][0]']
 tchNormalization)

 batch_normalization_33 (Ba     (None, 54, 54, 64)   256         ['conv2d_33[0][0]']
 tchNormalization)

 add_8 (Add)                    (None, 54, 54, 64)   0           ['batch_normalization_32[0][0]
                                                                  ',
                                                                   'batch_normalization_33[0][0]
                                                                  ']

 activation_28 (Activation)     (None, 54, 54, 64)   0           ['add_8[0][0]']

 conv2d_34 (Conv2D)             (None, 54, 54, 64)   4160        ['activation_28[0][0]']

 batch_normalization_34 (Ba     (None, 54, 54, 64)   256         ['conv2d_34[0][0]']
 tchNormalization)

 activation_29 (Activation)     (None, 54, 54, 64)   0           ['batch_normalization_34[0][0]
                                                                  ']

 conv2d_35 (Conv2D)             (None, 54, 54, 64)   36928       ['activation_29[0][0]']

 batch_normalization_35 (Ba     (None, 54, 54, 64)   256         ['conv2d_35[0][0]']
 tchNormalization)

 activation_30 (Activation)     (None, 54, 54, 64)   0           ['batch_normalization_35[0][0]
                                                                  ']

 conv2d_36 (Conv2D)             (None, 54, 54, 64)   4160        ['activation_30[0][0]']

 batch_normalization_36 (Ba     (None, 54, 54, 64)   256         ['conv2d_36[0][0]']
 tchNormalization)

 add_9 (Add)                    (None, 54, 54, 64)   0           ['batch_normalization_36[0][0]
                                                                  ',
                                                                   'activation_28[0][0]']

 activation_31 (Activation)     (None, 54, 54, 64)   0           ['add_9[0][0]']
```



```
conv2d_37 (Conv2D)          (None, 27, 27, 64)       4160      ['activation_31[0][0]']

batch_normalization_37 (Ba  (None, 27, 27, 64)       256       ['conv2d_37[0][0]']
tchNormalization)

activation_32 (Activation)  (None, 27, 27, 64)       0         ['batch_normalization_37[0][0]
                                                                '[

conv2d_38 (Conv2D)          (None, 27, 27, 64)       36928     ['activation_32[0][0]']

batch_normalization_38 (Ba  (None, 27, 27, 64)       256       ['conv2d_38[0][0]']
tchNormalization)

activation_33 (Activation)  (None, 27, 27, 64)       0         ['batch_normalization_38[0][0]
                                                                ']

conv2d_39 (Conv2D)          (None, 27, 27, 128)      8320      ['activation_33[0][0]']

conv2d_40 (Conv2D)          (None, 27, 27, 128)      8320      ['activation_31[0][0]']

batch_normalization_39 (Ba  (None, 27, 27, 128)      512       ['conv2d_39[0][0]']
tchNormalization)

batch_normalization_40 (Ba  (None, 27, 27, 128)      512       ['conv2d_40[0][0]']
tchNormalization)

add_10 (Add)                (None, 27, 27, 128)      0         ['batch_normalization_39[0][0]
                                                                ',
                                                                 'batch_normalization_40[0][0]
                                                                ']

activation_34 (Activation)  (None, 27, 27, 128)      0         ['add_10[0][0]']

conv2d_41 (Conv2D)          (None, 27, 27, 128)      16512     ['activation_34[0][0]']

batch_normalization_41 (Ba  (None, 27, 27, 128)      512       ['conv2d_41[0][0]']
tchNormalization)

activation_35 (Activation)  (None, 27, 27, 128)      0         ['batch_normalization_41[0][0]
                                                                ']

conv2d_42 (Conv2D)          (None, 27, 27, 128)      147584    ['activation_35[0][0]']

batch_normalization_42 (Ba  (None, 27, 27, 128)      512       ['conv2d_42[0][0]']
tchNormalization)

activation_36 (Activation)  (None, 27, 27, 128)      0         ['batch_normalization_42[0][0]
                                                                ']

conv2d_43 (Conv2D)          (None, 27, 27, 128)      16512     ['activation_36[0][0]']

batch_normalization_43 (Ba  (None, 27, 27, 128)      512       ['conv2d_43[0][0]']
tchNormalization)

add_11 (Add)                (None, 27, 27, 128)      0         ['batch_normalization_43[0][0]
                                                                ',
                                                                 'activation_34[0][0]']

activation_37 (Activation)  (None, 27, 27, 128)      0         ['add_11[0][0]']

conv2d_44 (Conv2D)          (None, 14, 14, 128)      16512     ['activation_37[0][0]']

batch_normalization_44 (Ba  (None, 14, 14, 128)      512       ['conv2d_44[0][0]']
tchNormalization)

activation_38 (Activation)  (None, 14, 14, 128)      0         ['batch_normalization_44[0][0]
                                                                ']

conv2d_45 (Conv2D)          (None, 14, 14, 128)      147584    ['activation_38[0][0]']
```



```
batch_normalization_45 (Ba  (None, 14, 14, 128)         512       ['conv2d_45[0][0]']
tchNormalization)

activation_39 (Activation)  (None, 14, 14, 128)         0         ['batch_normalization_45[0][0]
                                                                  ']

conv2d_46 (Conv2D)          (None, 14, 14, 256)         33024     ['activation_39[0][0]']

conv2d_47 (Conv2D)          (None, 14, 14, 256)         33024     ['activation_37[0][0]']

batch_normalization_46 (Ba  (None, 14, 14, 256)         1024      ['conv2d_46[0][0]']
tchNormalization)

batch_normalization_47 (Ba  (None, 14, 14, 256)         1024      ['conv2d_47[0][0]']
tchNormalization)

add_12 (Add)                (None, 14, 14, 256)         0         ['batch_normalization_46[0][0]
                                                                  ',
                                                                   'batch_normalization_47[0][0]
                                                                  ']

activation_40 (Activation)  (None, 14, 14, 256)         0         ['add_12[0][0]']

conv2d_48 (Conv2D)          (None, 14, 14, 256)         65792     ['activation_40[0][0]']

batch_normalization_48 (Ba  (None, 14, 14, 256)         1024      ['conv2d_48[0][0]']
tchNormalization)

activation_41 (Activation)  (None, 14, 14, 256)         0         ['batch_normalization_48[0][0]
                                                                  ']

conv2d_49 (Conv2D)          (None, 14, 14, 256)         590080    ['activation_41[0][0]']

batch_normalization_49 (Ba  (None, 14, 14, 256)         1024      ['conv2d_49[0][0]']
tchNormalization)

activation_42 (Activation)  (None, 14, 14, 256)         0         ['batch_normalization_49[0][0]
                                                                  ']

conv2d_50 (Conv2D)          (None, 14, 14, 256)         65792     ['activation_42[0][0]']

batch_normalization_50 (Ba  (None, 14, 14, 256)         1024      ['conv2d_50[0][0]']
tchNormalization)

add_13 (Add)                (None, 14, 14, 256)         0         ['batch_normalization_50[0][0]
                                                                  ',
                                                                   'activation_40[0][0]']

activation_43 (Activation)  (None, 14, 14, 256)         0         ['add_13[0][0]']

conv2d_51 (Conv2D)          (None, 7, 7, 256)           65792     ['activation_43[0][0]']

batch_normalization_51 (Ba  (None, 7, 7, 256)           1024      ['conv2d_51[0][0]']
tchNormalization)

activation_44 (Activation)  (None, 7, 7, 256)           0         ['batch_normalization_51[0][0]
                                                                  ']

conv2d_52 (Conv2D)          (None, 7, 7, 256)           590080    ['activation_44[0][0]']

batch_normalization_52 (Ba  (None, 7, 7, 256)           1024      ['conv2d_52[0][0]']
tchNormalization)

activation_45 (Activation)  (None, 7, 7, 256)           0         ['batch_normalization_52[0][0]
                                                                  ']

conv2d_53 (Conv2D)          (None, 7, 7, 512)           131584    ['activation_45[0][0]']

conv2d_54 (Conv2D)          (None, 7, 7, 512)           131584    ['activation_43[0][0]']
```



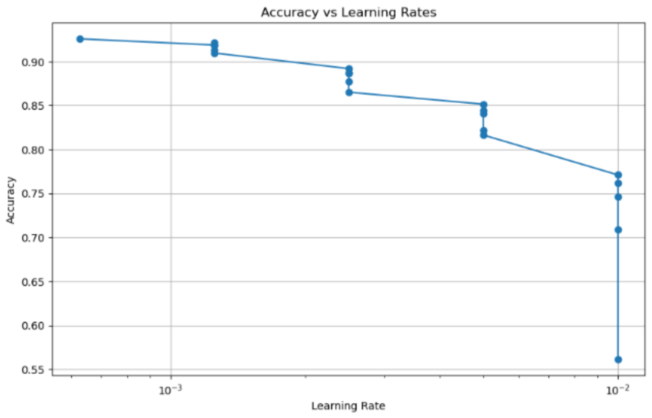
```
 batch_normalization_53 (Ba  (None, 7, 7, 512)            2048      ['conv2d_53[0][0]']
 tchNormalization)

 batch_normalization_54 (Ba  (None, 7, 7, 512)            2048      ['conv2d_54[0][0]']
 tchNormalization)

 add_14 (Add)                (None, 7, 7, 512)            0         ['batch_normalization_53[0][0]
                                                                    ',
                                                                     'batch_normalization_54[0][0]
                                                                    ']

 activation_46 (Activation)  (None, 7, 7, 512)            0         ['add_14[0][0]']

 conv2d_55 (Conv2D)          (None, 7, 7, 512)            262656    ['activation_46[0][0]']

 batch_normalization_55 (Ba  (None, 7, 7, 512)            2048      ['conv2d_55[0][0]']
 tchNormalization)

 activation_47 (Activation)  (None, 7, 7, 512)            0         ['batch_normalization_55[0][0]
                                                                    ']

 conv2d_56 (Conv2D)          (None, 7, 7, 512)            2359808   ['activation_47[0][0]']

 batch_normalization_56 (Ba  (None, 7, 7, 512)            2048      ['conv2d_56[0][0]']
 tchNormalization)

 activation_48 (Activation)  (None, 7, 7, 512)            0         ['batch_normalization_56[0][0]
                                                                    ']

 conv2d_57 (Conv2D)          (None, 7, 7, 512)            262656    ['activation_48[0][0]']

 batch_normalization_57 (Ba  (None, 7, 7, 512)            2048      ['conv2d_57[0][0]']
 tchNormalization)

 add_15 (Add)                (None, 7, 7, 512)            0         ['batch_normalization_57[0][0]
                                                                    ',
                                                                     'activation_46[0][0]']

 activation_49 (Activation)  (None, 7, 7, 512)            0         ['add_15[0][0]']

 average_pooling2d_1 (Avera  (None, 1, 1, 512)            0         ['activation_49[0][0]']
 gePooling2D)

 flatten_1 (Flatten)         (None, 512)                  0         ['average_pooling2d_1[0][0]']

 dense_2 (Dense)             (None, 512)                  262656    ['flatten_1[0][0]']

 dropout_1 (Dropout)         (None, 512)                  0         ['dense_2[0][0]']

 dense_3 (Dense)             (None, 5)                    2565      ['dropout_1[0][0]']

==================================================================================================
Total params: 5347205 (20.40 MB)
Trainable params: 5335621 (20.35 MB)
Non-trainable params: 11584 (45.25 KB)
__________________________________________________________________________________________________
```

In [73]:
```python
# Define a learning rate schedule
def lr_schedule(epoch):
    initial_lr = 0.01
    drop = 0.5
    epochs_drop = 5
    lr = initial_lr * (drop ** (epoch // epochs_drop))
    return lr

# Store learning rates and accuracy
learning_rates = []   # List to store learning rates during training
accuracies = []  # List to store accuracy values during training

# Create a learning rate scheduler callback
lr_scheduler = LearningRateScheduler(lr_schedule)


# Custom callback to log learning rates and accuracy
class LearningRateLogger(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        lr = float(tf.keras.backend.get_value(self.model.optimizer.lr))
        learning_rates.append(lr)

    def on_epoch_end(self, epoch, logs=None):
        accuracies.append(logs.get('accuracy'))
```

In [74]:
```python
# Create an early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

In [75]:
```python
# Fit the model with training and validation data
history = model.fit(train_generator, epochs=50, validation_data=val_generator, callbacks=[early_stopping, lr_scheduler, Lear

# Plotting accuracy against learning rates
plt.figure(figsize=(10, 6))
plt.plot(learning_rates, accuracies, marker='o')
plt.title('Accuracy vs Learning Rates')
plt.xlabel('Learning Rate')
plt.ylabel('Accuracy')
plt.xscale('log')  # Set x-axis to logarithmic scale for better visualization if needed
plt.grid(True)
plt.show()
```

```
Epoch 1/50
76/76 [==============================] - 96s 1s/step - loss: 1.0705 - accuracy: 0.5615 - recall_1: 0.3998 - precision_1: 0.6916 - val_loss: 1.8322 - val_accuracy: 0.2096 - val_recall_1: 0.0000e+00 - val_precision_1: 0.0000e+00 - lr: 0.0100
Epoch 2/50
76/76 [==============================] - 84s 1s/step - loss: 0.7333 - accuracy: 0.7092 - recall_1: 0.6415 - precision_1: 0.7608 - val_loss: 2.5815 - val_accuracy: 0.2822 - val_recall_1: 0.2657 - val_precision_1: 0.2875 - lr: 0.0100
Epoch 3/50
76/76 [==============================] - 84s 1s/step - loss: 0.6557 - accuracy: 0.7463 - recall_1: 0.6968 - precision_1: 0.7794 - val_loss: 1.5751 - val_accuracy: 0.4158 - val_recall_1: 0.3597 - val_precision_1: 0.4570 - lr: 0.0100
Epoch 4/50
76/76 [==============================] - 84s 1s/step - loss: 0.5927 - accuracy: 0.7616 - recall_1: 0.7219 - precision_1: 0.7926 - val_loss: 0.9249 - val_accuracy: 0.6700 - val_recall_1: 0.4521 - val_precision_1: 0.7919 - lr: 0.0100
Epoch 5/50
76/76 [==============================] - 83s 1s/step - loss: 0.5810 - accuracy: 0.7710 - recall_1: 0.7347 - precision_1: 0.8008 - val_loss: 0.8176 - val_accuracy: 0.6914 - val_recall_1: 0.6056 - val_precision_1: 0.7459 - lr: 0.0100
Epoch 6/50
76/76 [==============================] - 83s 1s/step - loss: 0.4797 - accuracy: 0.8164 - recall_1: 0.7880 - precision_1: 0.8366 - val_loss: 1.4184 - val_accuracy: 0.5314 - val_recall_1: 0.4901 - val_precision_1: 0.5940 - lr: 0.0050
Epoch 7/50
76/76 [==============================] - 85s 1s/step - loss: 0.4597 - accuracy: 0.8218 - recall_1: 0.7970 - precision_1: 0.8448 - val_loss: 0.6679 - val_accuracy: 0.7541 - val_recall_1: 0.7063 - val_precision_1: 0.7955 - lr: 0.0050
Epoch 8/50
76/76 [==============================] - 81s 1s/step - loss: 0.4141 - accuracy: 0.8408 - recall_1: 0.8185 - precision_1: 0.8585 - val_loss: 0.5141 - val_accuracy: 0.7987 - val_recall_1: 0.7723 - val_precision_1: 0.8239 - lr: 0.0050
Epoch 9/50
76/76 [==============================] - 87s 1s/step - loss: 0.4207 - accuracy: 0.8441 - recall_1: 0.8210 - precision_1: 0.8630 - val_loss: 0.4994 - val_accuracy: 0.8086 - val_recall_1: 0.7673 - val_precision_1: 0.8501 - lr: 0.0050
Epoch 10/50
76/76 [==============================] - 102s 1s/step - loss: 0.3822 - accuracy: 0.8515 - recall_1: 0.8337 - precision_1: 0.8670 - val_loss: 0.9758 - val_accuracy: 0.6238 - val_recall_1: 0.5891 - val_precision_1: 0.6491 - lr: 0.0050
Epoch 11/50
76/76 [==============================] - 102s 1s/step - loss: 0.3465 - accuracy: 0.8651 - recall_1: 0.8498 - precision_1: 0.8796 - val_loss: 0.5398 - val_accuracy: 0.7838 - val_recall_1: 0.7591 - val_precision_1: 0.8244 - lr: 0.0025
Epoch 12/50
76/76 [==============================] - 101s 1s/step - loss: 0.3427 - accuracy: 0.8771 - recall_1: 0.8597 - precision_1: 0.8933 - val_loss: 0.8505 - val_accuracy: 0.6964 - val_recall_1: 0.6683 - val_precision_1: 0.7271 - lr: 0.0025
Epoch 13/50
76/76 [==============================] - 98s 1s/step - loss: 0.2961 - accuracy: 0.8870 - recall_1: 0.8725 - precision_1: 0.8969 - val_loss: 0.4057 - val_accuracy: 0.8383 - val_recall_1: 0.8267 - val_precision_1: 0.8564 - lr: 0.0025
Epoch 14/50
76/76 [==============================] - 99s 1s/step - loss: 0.2983 - accuracy: 0.8870 - recall_1: 0.8758 - precision_1: 0.9015 - val_loss: 0.4899 - val_accuracy: 0.8168 - val_recall_1: 0.8020 - val_precision_1: 0.8336 - lr: 0.0025
Epoch 15/50
76/76 [==============================] - 99s 1s/step - loss: 0.2943 - accuracy: 0.8919 - recall_1: 0.8795 - precision_1: 0.9034 - val_loss: 0.6476 - val_accuracy: 0.7624 - val_recall_1: 0.7459 - val_precision_1: 0.7807 - lr: 0.0025
Epoch 16/50
76/76 [==============================] - 102s 1s/step - loss: 0.2663 - accuracy: 0.9097 - recall_1: 0.8952 - precision_1: 0.9203 - val_loss: 0.4025 - val_accuracy: 0.8482 - val_recall_1: 0.8399 - val_precision_1: 0.8656 - lr: 0.0012
Epoch 17/50
76/76 [==============================] - 101s 1s/step - loss: 0.2551 - accuracy: 0.9125 - recall_1: 0.8981 - precision_1: 0.9276 - val_loss: 0.4458 - val_accuracy: 0.8350 - val_recall_1: 0.8218 - val_precision_1: 0.8542 - lr: 0.0012
Epoch 18/50
76/76 [==============================] - 101s 1s/step - loss: 0.2452 - accuracy: 0.9216 - recall_1: 0.9076 - precision_1: 0.9298 - val_loss: 0.4062 - val_accuracy: 0.8449 - val_recall_1: 0.8300 - val_precision_1: 0.8702 - lr: 0.0012
Epoch 19/50
76/76 [==============================] - 87s 1s/step - loss: 0.2503 - accuracy: 0.9179 - recall_1: 0.9051 - precision_1: 0.9281 - val_loss: 0.4097 - val_accuracy: 0.8399 - val_recall_1: 0.8267 - val_precision_1: 0.8535 - lr: 0.0012
Epoch 20/50
76/76 [==============================] - 86s 1s/step - loss: 0.2285 - accuracy: 0.9187 - recall_1: 0.9109 - precision_1: 0.9293 - val_loss: 0.4847 - val_accuracy: 0.7970 - val_recall_1: 0.7805 - val_precision_1: 0.8113 - lr: 0.0012
Epoch 21/50
76/76 [==============================] - 89s 1s/step - loss: 0.2267 - accuracy: 0.9257 - recall_1: 0.9179 - precision_1: 0.9364 - val_loss: 0.4094 - val_accuracy: 0.8432 - val_recall_1: 0.8300 - val_precision_1: 0.8628 - lr: 6.2500e-04
```



In [76]:

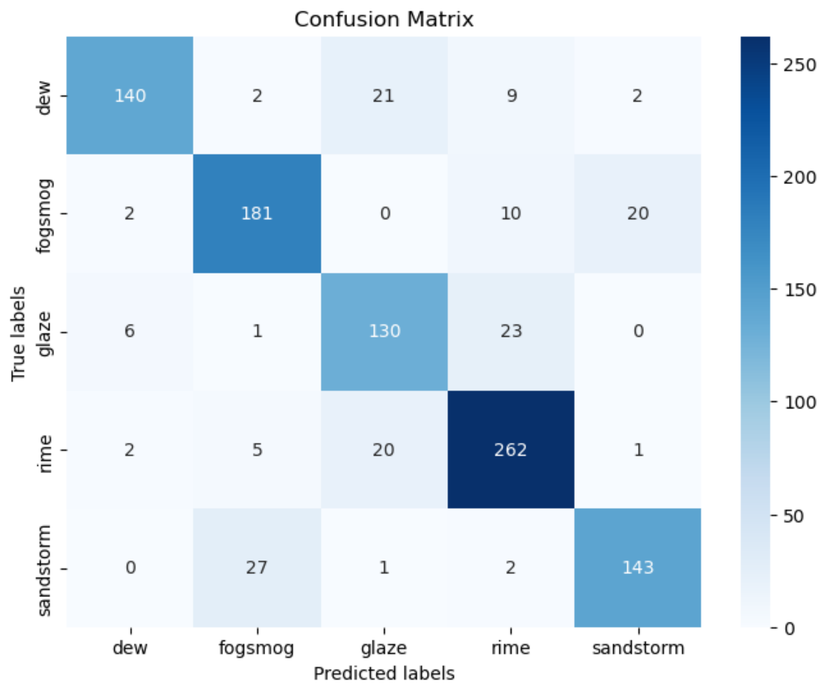
```python
# Evaluate the model on the test generator
test_loss, test_accuracy, test_recall, test_precision = model.evaluate(test_generator)
# Print evaluation metrics
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
print(f'Test Precision: {test_precision:.4f}')
print(f'Test Recall: {test_recall:.4f}')
```

```
32/32 [==============================] - 9s 262ms/step - loss: 0.4405 - accuracy: 0.8475 - recall_1: 0.8327 - precision_1: 0.86
08
Test Loss: 0.4405
Test Accuracy: 0.8475
Test Precision: 0.8608
Test Recall: 0.8327
```

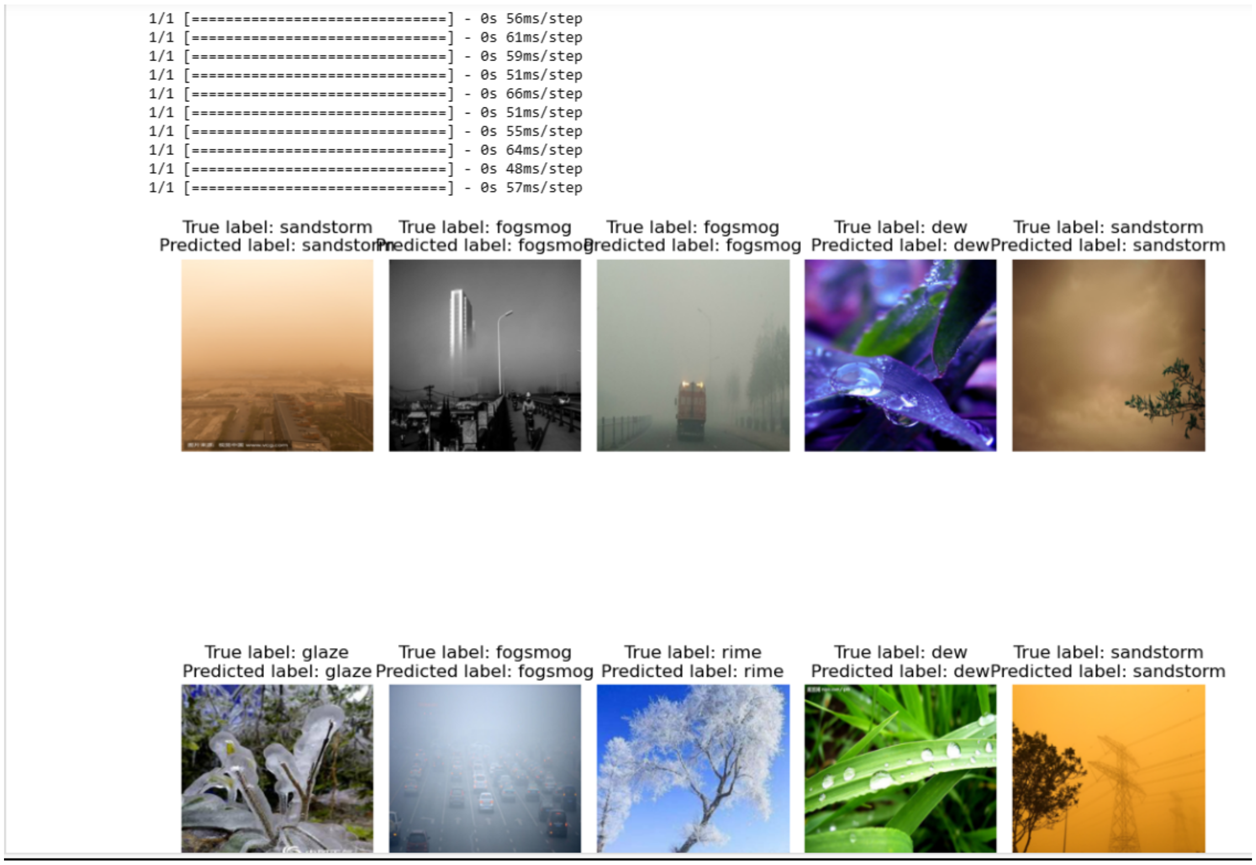
In [83]:

```python
# Generate predictions
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = test_generator.classes

# Create a confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# Display confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=labels_list, yticklabels=labels_list)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

```

```
32/32 [==============================] - 8s 250ms/step
```



In [86]:
```python
# Display a couple of test images with true and predicted labels
num_images_to_display = 10  # Specify the number of images to display
num_cols = 5  # Number of columns in the grid
num_rows = (num_images_to_display + num_cols - 1) // num_cols  # Calculate number of rows needed
indices = np.random.choice(len(test_generator.filenames), num_images_to_display)

# Create subplots in a grid
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10, 10))  # Adjust figsize as needed

for i, idx in enumerate(indices):
    row = i // num_cols
    col = i % num_cols

    image_path = os.path.join('C:\\Users\\Noura\\Downloads\\72917-Group5\\dataset\\test', test_generator.filenames[idx])

    # Load and resize the image
    img = Image.open(image_path)
    img = img.resize((224, 224))
    img_array = np.array(img) / 255.0  # Normalize pixel values

    # Reshape and predict
    img_array = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction)
    true_class = test_generator.classes[idx]

    # Display image and labels in subplots
    axes[row, col].imshow(img)
    axes[row, col].axis('off')
    axes[row, col].set_title(f"True label: {labels_list[true_class]}\nPredicted label: {labels_list[predicted_class]}")

# Hide empty subplots if num_images_to_display is less than num_cols * num_rows
for i in range(num_images_to_display, num_rows * num_cols):
    row = i // num_cols
    col = i % num_cols
    axes[row, col].axis('off')

plt.tight_layout()  # Adjust spacing between subplots
plt.show()

```



```
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 57ms/step
```