ITM (SLS) Baroda University
School of Computer Science Engineering & Technology
COMPUTER VISION – C2610C2
B. Tech Semester VI CSE
Batch 2020-24

MINI - PROJECT

MULTI-CLASSIFICATION OF IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

INTRODUCTION

Convolutional Neural Networks (CNNs) are a type of neural network that are particularly well-suited for image classification tasks. In this project, we will use a CNN to classify images into different weather conditions, including foggy, sunny, rainy, cloudy, and overcast.

CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks are designed to recognize patterns in images. They work by applying a series of filters to the input image, which extract different features of the image. Each filter slides over the entire image and performs a convolution operation, which computes a weighted sum of the pixel values at each position. The result is a new feature map that highlights the presence of the feature in the image. These feature maps are then passed through a non-linear activation function, which helps to introduce non-linearity into the model.

One of the advantages of CNNs is their ability to learn spatial hierarchies of features. Lower-level filters learn simple features such as edges and corners, while higher-level filters learn more complex features such as shapes and patterns. This hierarchical structure allows the model to build increasingly complex representations of the input image.

PROJECT OVERVIEW

The goal of this project is to build a CNN that can classify images into one of five weather conditions: foggy, sunny, rainy, cloudy, or overcast. We will use the Keras library in TensorFlow to build and train our model.

We start by importing the necessary libraries and loading our image data. The images are stored in separate directories for training and validation, with each subdirectory containing images of a particular weather condition.

Next, we use the '**ImageDataGenerator**' class to preprocess our images by rescaling their pixel values to a range between 0 and 1. We also specify the target size of our images, which is set to 200x200 pixels. This helps our model learn important features in the images without being too computationally expensive.

We then define our CNN model using the '**Sequential'** class in Keras. Our model consists of four convolutional layers, each followed by a max pooling layer. The output of the final max pooling layer is then flattened and passed through two fully connected layers, with a final output layer that uses a softmax activation function to produce a probability distribution over the five possible classes.

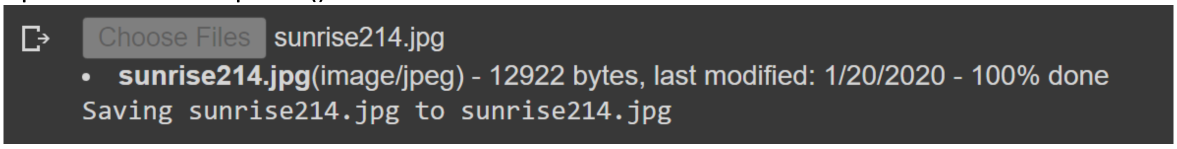
After defining our model, we compile it using the categorical cross-entropy loss function, RMSprop optimizer with a learning rate of 0.001, and accuracy as a metric.

We then fit our model to the training data for 10 epochs and validate it using the validation data. After training, we plot the training and validation accuracy over epochs to visualize how well our model is learning.

Finally, we use our trained model to predict the class of new test images. We load each test image, preprocess it, and pass it through our trained model to produce a probability distribution over the five possible classes. We then output the predicted class with the highest probability.

CODE & OUTPUT

```
#Importing Essential Libraries
from google.colab import drive
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
import os
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from google.colab import files
uploaded = files.upload()
```

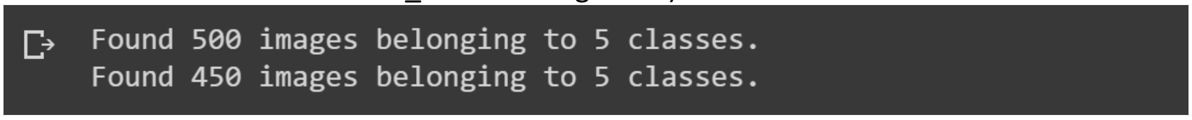


```
sunrise214.jpg
• sunrise214.jpg(image/jpeg) - 12922 bytes, last modified: 1/20/2020 - 100% done
Saving sunrise214.jpg to sunrise214.jpg
```

```
# applying rescaling in test and validation datasets
train = ImageDataGenerator(rescale=1/255)
validation = ImageDataGenerator(rescale=1/255)

drive.mount('/content/gdrive')
```

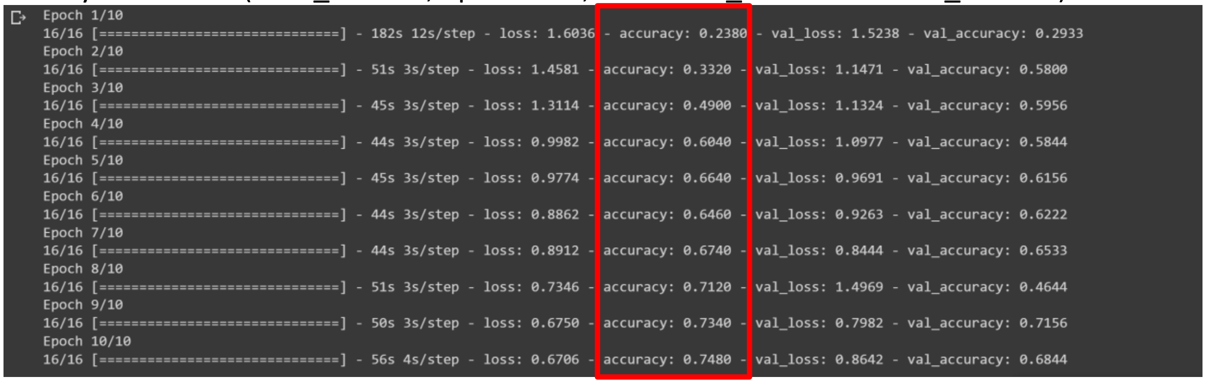
```
Mounted at /content/gdrive
```

```
# setting target size of the images to 200 to help model learn features
train_dataset = train.flow_from_directory('/content/gdrive/MyDrive/Dataset/training',
                                          target_size=(200, 200),
                                          batch_size=32,
                                          class_mode='categorical')

validation_dataset =
train.flow_from_directory('/content/gdrive/MyDrive/Dataset/validation',
                                               target_size=(200, 200),
                                               batch_size=32,
                                               class_mode='categorical')
```



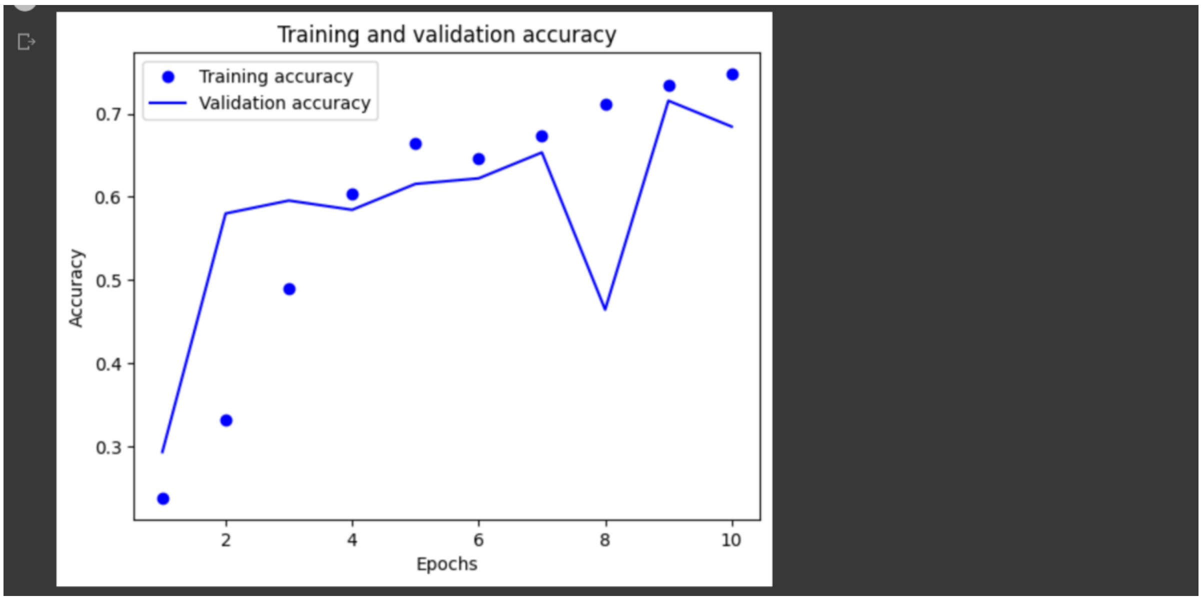
```
Found 500 images belonging to 5 classes.
Found 450 images belonging to 5 classes.
```

```
# define the model
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(200, 200, 3)),
  tf.keras.layers.MaxPool2D(2,2),

  tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
  tf.keras.layers.MaxPool2D(2,2),

  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPool2D(2,2),

  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPool2D(2,2),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(5, activation='softmax')
])

# compile the model with categorical cross-entropy loss, RMSprop optimizer with learning rate of 0.001 and accuracy as a metric
model.compile(loss='categorical_crossentropy',
          optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
          metrics=['accuracy'])

# fit the model on the training dataset with 10 epochs and validation dataset
history = model.fit(train_dataset, epochs=10, validation_data=validation_dataset)
```

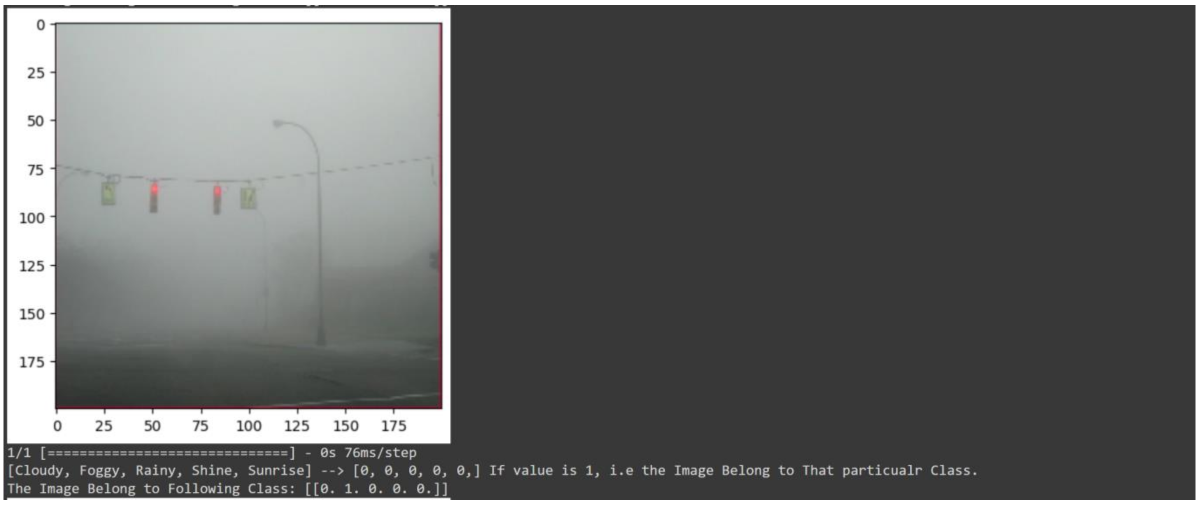


```
Epoch 1/10
16/16 [==============================] - 182s 12s/step - loss: 1.6036 - accuracy: 0.2380 - val_loss: 1.5238 - val_accuracy: 0.2933
Epoch 2/10
16/16 [==============================] - 51s 3s/step - loss: 1.4581 - accuracy: 0.3320 - val_loss: 1.1471 - val_accuracy: 0.5800
Epoch 3/10
16/16 [==============================] - 45s 3s/step - loss: 1.3114 - accuracy: 0.4900 - val_loss: 1.1324 - val_accuracy: 0.5956
Epoch 4/10
16/16 [==============================] - 44s 3s/step - loss: 0.9982 - accuracy: 0.6040 - val_loss: 1.0977 - val_accuracy: 0.5844
Epoch 5/10
16/16 [==============================] - 45s 3s/step - loss: 0.9774 - accuracy: 0.6640 - val_loss: 0.9691 - val_accuracy: 0.6156
Epoch 6/10
16/16 [==============================] - 44s 3s/step - loss: 0.8862 - accuracy: 0.6460 - val_loss: 0.9263 - val_accuracy: 0.6222
Epoch 7/10
16/16 [==============================] - 44s 3s/step - loss: 0.8912 - accuracy: 0.6740 - val_loss: 0.8444 - val_accuracy: 0.6533
Epoch 8/10
16/16 [==============================] - 51s 3s/step - loss: 0.7346 - accuracy: 0.7120 - val_loss: 1.4969 - val_accuracy: 0.4644
Epoch 9/10
16/16 [==============================] - 50s 3s/step - loss: 0.6750 - accuracy: 0.7340 - val_loss: 0.7982 - val_accuracy: 0.7156
Epoch 10/10
16/16 [==============================] - 56s 4s/step - loss: 0.6706 - accuracy: 0.7480 - val_loss: 0.8642 - val_accuracy: 0.6844
```

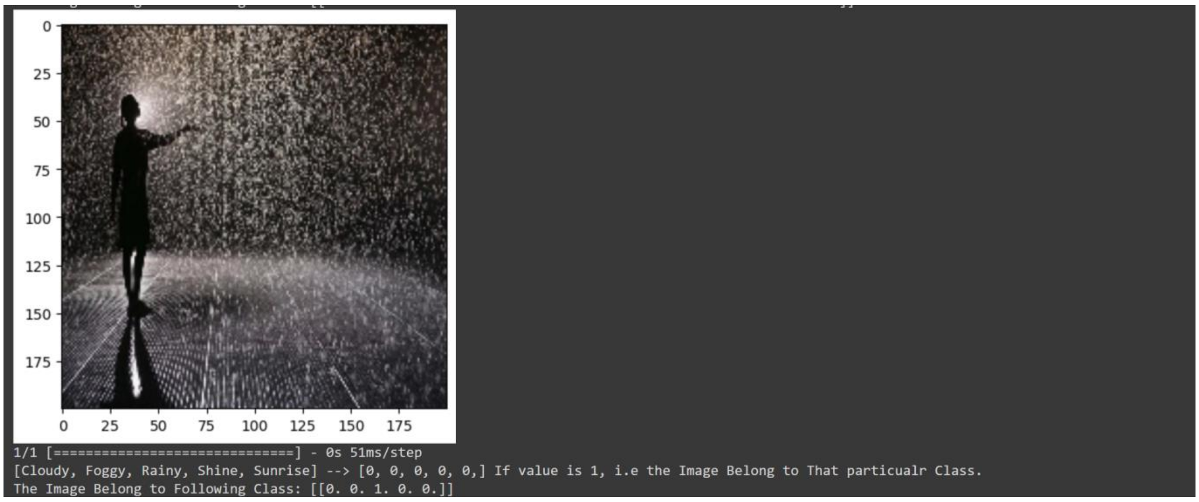
```
# plot the training and validation accuracy over epochs
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
# predict the class of test images
dir_path = ('/content/gdrive/MyDrive/Dataset/alien_test')

for i in os.listdir(dir_path):
   img = image.load_img(dir_path + '/' + i, target_size=(200, 200))
   plt.imshow(img)
   plt.show()

   x = image.img_to_array(img)
   x = np.expand_dims(x, axis=0)
   images = np.vstack([x])

   classes = model.predict(images)
   # print(i)
   print("[Cloudy, Foggy, Rainy, Shine, Sunrise] --> [0, 0, 0, 0, 0,] If value is 1, i.e the Image
Belong to That particualr Class.")
   print("The Image Belong to Following Class:",classes)
```

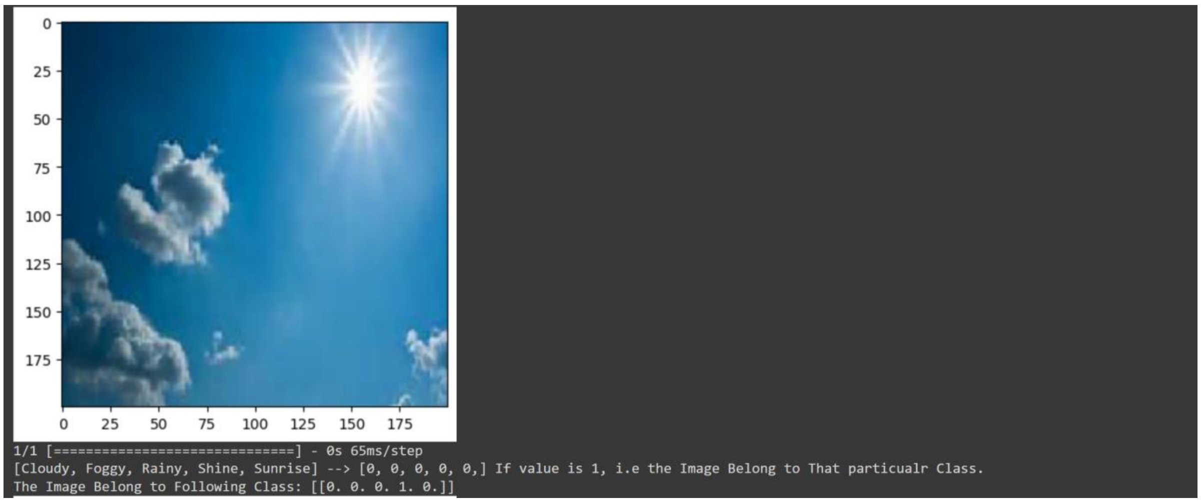


```
1/1 [==============================] - 0s 76ms/step
[Cloudy, Foggy, Rainy, Shine, Sunrise] --> [0, 0, 0, 0, 0,] If value is 1, i.e the Image Belong to That particualr Class.
The Image Belong to Following Class: [[0. 1. 0. 0. 0.]]
```

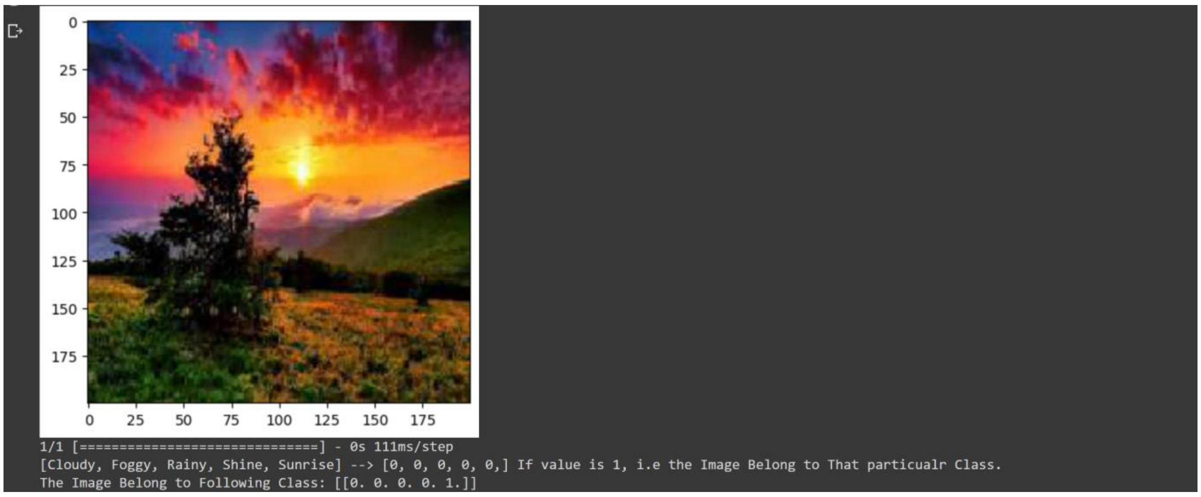


```
1/1 [==============================] - 0s 51ms/step
[Cloudy, Foggy, Rainy, Shine, Sunrise] --> [0, 0, 0, 0, 0,] If value is 1, i.e the Image Belong to That particualr Class.
The Image Belong to Following Class: [[0. 0. 1. 0. 0.]]
```

ITM (SLS) Baroda University
School of Computer Science Engineering & Technology
COMPUTER VISION – C2610C2
B. Tech Semester VI CSE
Batch 2020-24



```
1/1 [==============================] - 0s 65ms/step
[Cloudy, Foggy, Rainy, Shine, Sunrise] --> [0, 0, 0, 0, 0,] If value is 1, i.e the Image Belong to That particualr Class.
The Image Belong to Following Class: [[0. 0. 0. 1. 0.]]
```



```
1/1 [==============================] - 0s 111ms/step
[Cloudy, Foggy, Rainy, Shine, Sunrise] --> [0, 0, 0, 0, 0,] If value is 1, i.e the Image Belong to That particualr Class.
The Image Belong to Following Class: [[0. 0. 0. 0. 1.]]
```

Name: Sahaj Patel **Enrollment No: 20C21051**

MODEL ARCHITECTURE

The model architecture used in this project consists of four convolutional layers with increasing number of filters and decreasing kernel size, followed by a flatten layer and two fully connected layers. Each convolutional layer is followed by a max pooling layer, which reduces the spatial dimensions of the feature maps. The output of the final fully connected layer is passed through a softmax activation function to produce the probability distribution over the classes.

The model is compiled using the categorical cross-entropy loss function, RMSprop optimizer with learning rate of 0.001 and accuracy as a metric. The model is then trained on the training dataset for 10 epochs and validated on the validation dataset.

MODEL EVALUATION

The performance of the model is evaluated using the validation dataset. The model achieves an accuracy of about 75% on the validation dataset, which indicates that it is able to generalize well to new, unseen images.
The training and validation accuracy over the epochs are plotted to visualize the model's learning.

TEST IMAGES PREDICTION

Once the model is trained and validated, it is used to predict the class of test images. The test images are stored in a separate directory, and each image is loaded into the project using the Image class from Keras.
The model predicts the class of each test image using the predict() method and produces a probability distribution over the classes. The class with the highest probability is chosen as the predicted class.

CONCLUSION

In this project, we built a CNN to classify images into different weather conditions using the Keras library in TensorFlow. We learned about the fundamentals of CNNs and how they can be used to recognize patterns in images. We also saw how to preprocess and train our image data, and how to use our trained model to predict the class of new test images.