Quaid-i-Azam University, Islamabad

Project Report By:

Mr. Faizan Saleem Siddiqui

(BS-7th, Fall23, Dept of Electronics)

Submitted to:

Dr. Naeem Bhatti

Project Report on:

Image Classification Using CNN Model:

Student Name:	**Faizan Saleem Siddiqui**
Student ID:	**04102013033**
Subject:	**Digital Image Processing**
Course Code:	**EL-470**
Class:	**BS-7th**
Session:	**Fall2023**
Department:	**Electronics**

Date: ______________________ Sig:________________________________

Project Explanation:

In this project, a Convolutional Network based image classification model is designed, trained and tested to classify images according to the classes. The model takes total of 14000+ Training Images, 3000+ Testing Images and 7000+ Validation images to implement the idea. Exact details of the data is presented in the dataset section of this report. The project is part of BS-7th Semester, Department of Electronics, for the subject of Digital Image Processing.

In this project, the problem presented by Intel, a renowned microelectronics company, is solved. The problem is to classify the images into 6 categories. Therefore the model presented and designed in project classifies any given input image into following 6 categories.

1. Building
2. Street
3. Forrest
4. Glacier
5. Mountain
6. Sea

The model will take any input as image and might classify the input into any of the above mentioned classes. The details of model, kernels used, features extracted, all are presented in the next section. For the time being, just understand that this model can classify and predict the input image belonging to any of the above mentioned classes.

So in short, you give this model any image, and will tell you whether this image:

- Is an image of Forrest? Or
- Is an image of Mountain? Or
- Is an image of Street? Or
- Is an image of Glacier? Or
- Is an image of Sea? Or
- Is an image of a Building?

The project is implemented in Python Programming language, on Jupyter Notebook. The platform used is Kaggle. The reason for using Kaggle is the availability of and access to high efficiency GPU and CPU.

Dataset:

As earlier said, that this problem was presented by Intel, so the dataset was provided by the Intel. But, for this assignment, the dataset is used from the Kaggle website and online execution of training is carried out.

The dataset contains following three folders:

- Prediction (Seg_Pred)
- Testing (Seg_test)
- Training (Seg_Train)

The names in the parentheses describe the name of the folders in the dataset.

The testing and Training folders further contain following folders:

- Building
- Forrest
- Glacier
- Mountain
- Street
- Sea

The amount of images in the each folder is given as follows:

Training Folder		**Testing Folder**	
Name of Folder	**Number of Images**	**Name of Folder**	**Number of Images**
Building	2119	Building	437
Forrest	2271	Forrest	474
Glacier	2404	Glacier	553
Mountain	2512	Mountain	525
Sea	2274	Sea	510
Street	2382	Street	501
Total:	13962	**Total**	300

Total Images: Testing + Training Images = 13962+300 = 16962 Images

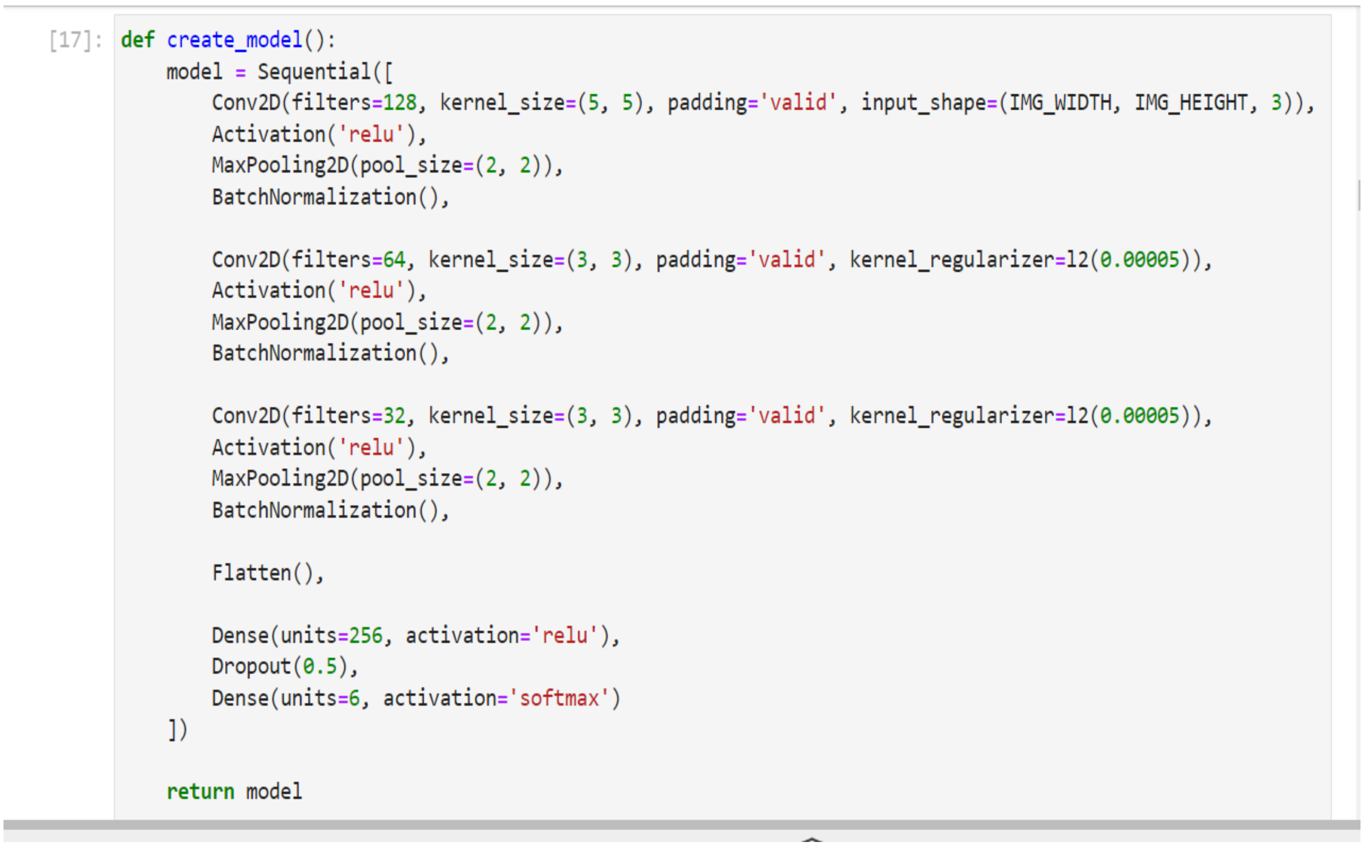
The prediction folder contains : 7301 Images.

Model Explanation:

This model is based on the convolutional neural network. The model is implemented using Keras with Tensorflow. A 2-dimensional neural network is implemented using "Sequential()" structure of Keras. In the first 2-D layer of network, total 128 filters are used, and kernel size of 5-x-5 is convolved to the input image. Next, in the first layer, padding is nullified and no extra padding is used. The input image shape is resize to 150x150 with 3 channels. After this, the activation "ReLu" is applied following the MaxPooling of 2x2 and a Batch-Normalization layer. These ingredients are for the first layer.

Next in the second 2D-layer, 64 filters are used, without any padding with kernel size of 3x3. Then a regularization of L1 = 0.00005 is added. Again an activation "ReLu" is applied with MaxPooling of size 2x2 and a batch normalization layer.

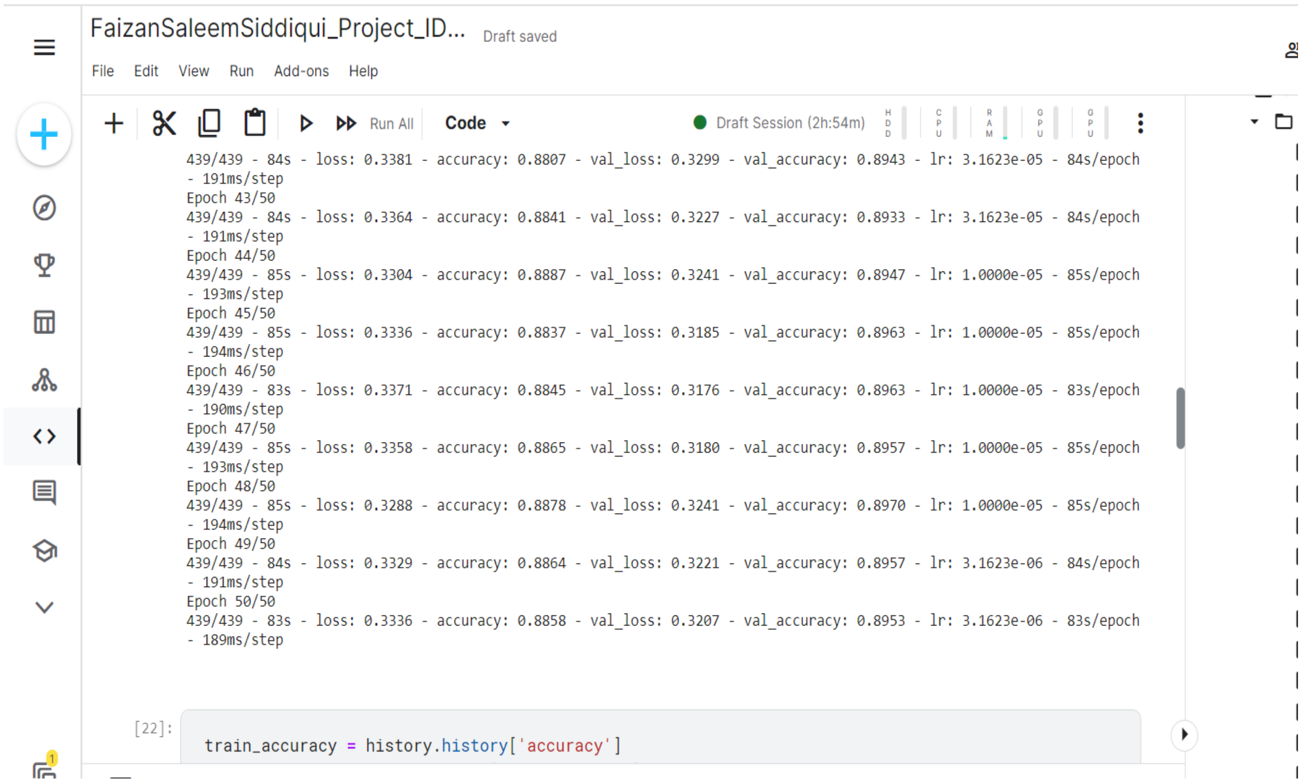
After this, in the third layer, the image is flattened to give it as input to the dense layer or fully connected layer. The dense layer then contains total 256 Neurons with again activation of Relu. After this dropout layer is added. Dropout value is 0.5, it means total 50% Neurons will be dropped out after this layer. Finally, another dense layer with 6 neurons is added. This is actually the output layer. The probabilities of the output are obtained using the softmax function. And this brings us to the end of model explanation. Our model is ready to perform.



```
[17]: def create_model():
          model = Sequential([
              Conv2D(filters=128, kernel_size=(5, 5), padding='valid', input_shape=(IMG_WIDTH, IMG_HEIGHT, 3)),
              Activation('relu'),
              MaxPooling2D(pool_size=(2, 2)),
              BatchNormalization(),

              Conv2D(filters=64, kernel_size=(3, 3), padding='valid', kernel_regularizer=l2(0.00005)),
              Activation('relu'),
              MaxPooling2D(pool_size=(2, 2)),
              BatchNormalization(),

              Conv2D(filters=32, kernel_size=(3, 3), padding='valid', kernel_regularizer=l2(0.00005)),
              Activation('relu'),
              MaxPooling2D(pool_size=(2, 2)),
              BatchNormalization(),

              Flatten(),

              Dense(units=256, activation='relu'),
              Dropout(0.5),
              Dense(units=6, activation='softmax')
          ])

          return model
```

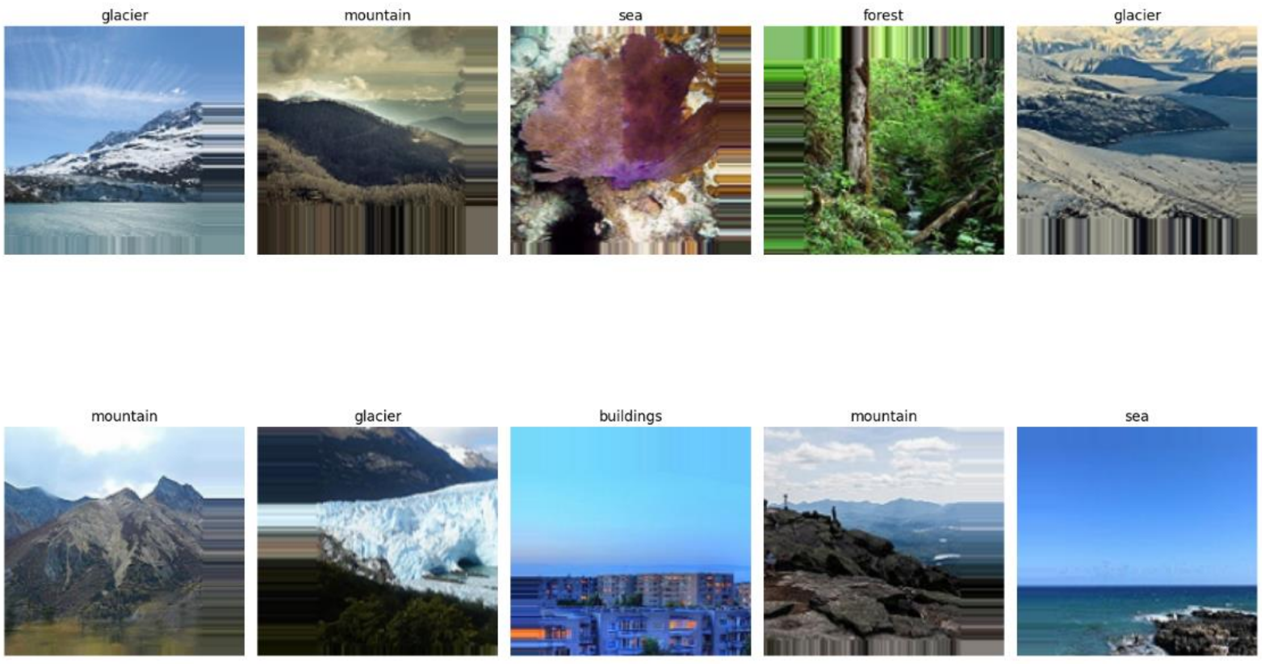
Training the Model:

Finally the model is compiled and trained. To compile the model, the "Adam" optimize is used with learning rate of 0.001 and loss function used is "categoricalCrossEntropy loss". Next, the model is trained. To train the model, total 50 epochs are used with verbose of 2 and a callback function is called to reduce the learning rate after every epoch. This helps in enhancing the accuracy and generalization of the model.



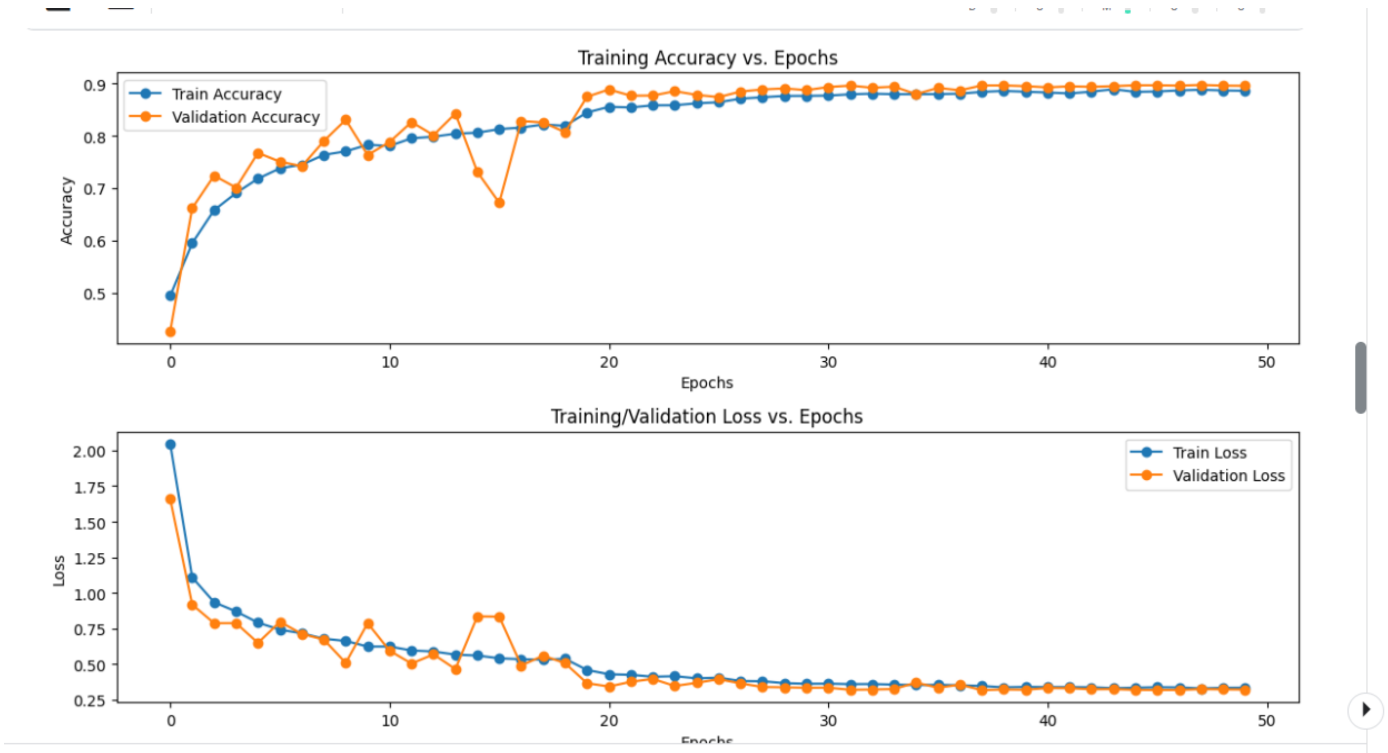
```
439/439 - 84s - loss: 0.3381 - accuracy: 0.8807 - val_loss: 0.3299 - val_accuracy: 0.8943 - lr: 3.1623e-05 - 84s/epoch
- 191ms/step
Epoch 43/50
439/439 - 84s - loss: 0.3364 - accuracy: 0.8841 - val_loss: 0.3227 - val_accuracy: 0.8933 - lr: 3.1623e-05 - 84s/epoch
- 191ms/step
Epoch 44/50
439/439 - 85s - loss: 0.3304 - accuracy: 0.8887 - val_loss: 0.3241 - val_accuracy: 0.8947 - lr: 1.0000e-05 - 85s/epoch
- 193ms/step
Epoch 45/50
439/439 - 85s - loss: 0.3336 - accuracy: 0.8837 - val_loss: 0.3185 - val_accuracy: 0.8963 - lr: 1.0000e-05 - 85s/epoch
- 194ms/step
Epoch 46/50
439/439 - 83s - loss: 0.3371 - accuracy: 0.8845 - val_loss: 0.3176 - val_accuracy: 0.8963 - lr: 1.0000e-05 - 83s/epoch
- 190ms/step
Epoch 47/50
439/439 - 85s - loss: 0.3358 - accuracy: 0.8865 - val_loss: 0.3180 - val_accuracy: 0.8957 - lr: 1.0000e-05 - 85s/epoch
- 193ms/step
Epoch 48/50
439/439 - 85s - loss: 0.3288 - accuracy: 0.8878 - val_loss: 0.3241 - val_accuracy: 0.8970 - lr: 1.0000e-05 - 85s/epoch
- 194ms/step
Epoch 49/50
439/439 - 84s - loss: 0.3329 - accuracy: 0.8864 - val_loss: 0.3221 - val_accuracy: 0.8957 - lr: 3.1623e-06 - 84s/epoch
- 191ms/step
Epoch 50/50
439/439 - 83s - loss: 0.3336 - accuracy: 0.8858 - val_loss: 0.3207 - val_accuracy: 0.8953 - lr: 3.1623e-06 - 83s/epoch
- 189ms/step
```

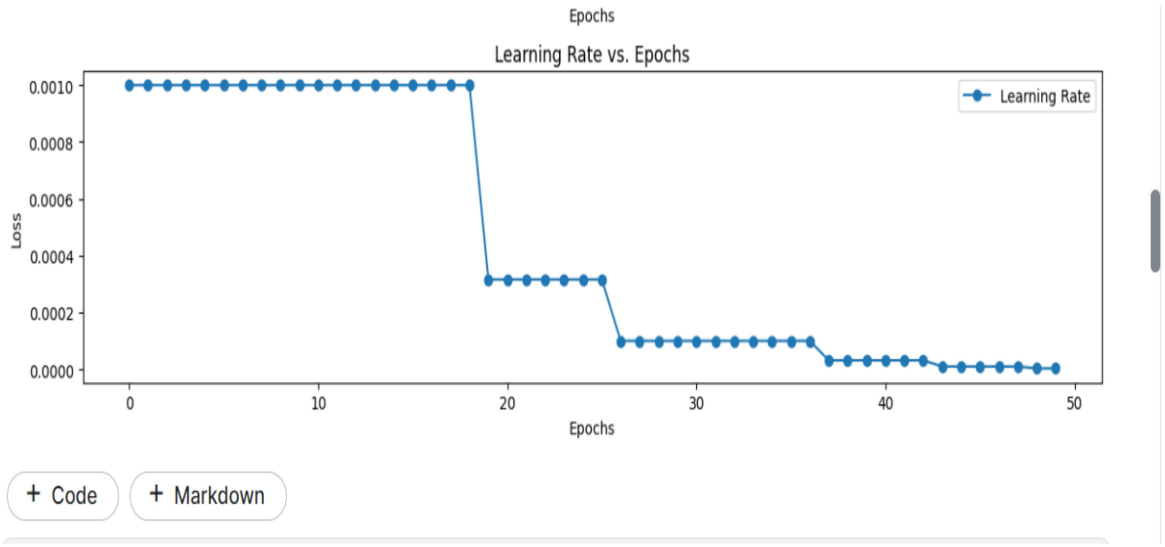
```
[22]: train_accuracy = history.history['accuracy']
```



Output Results:

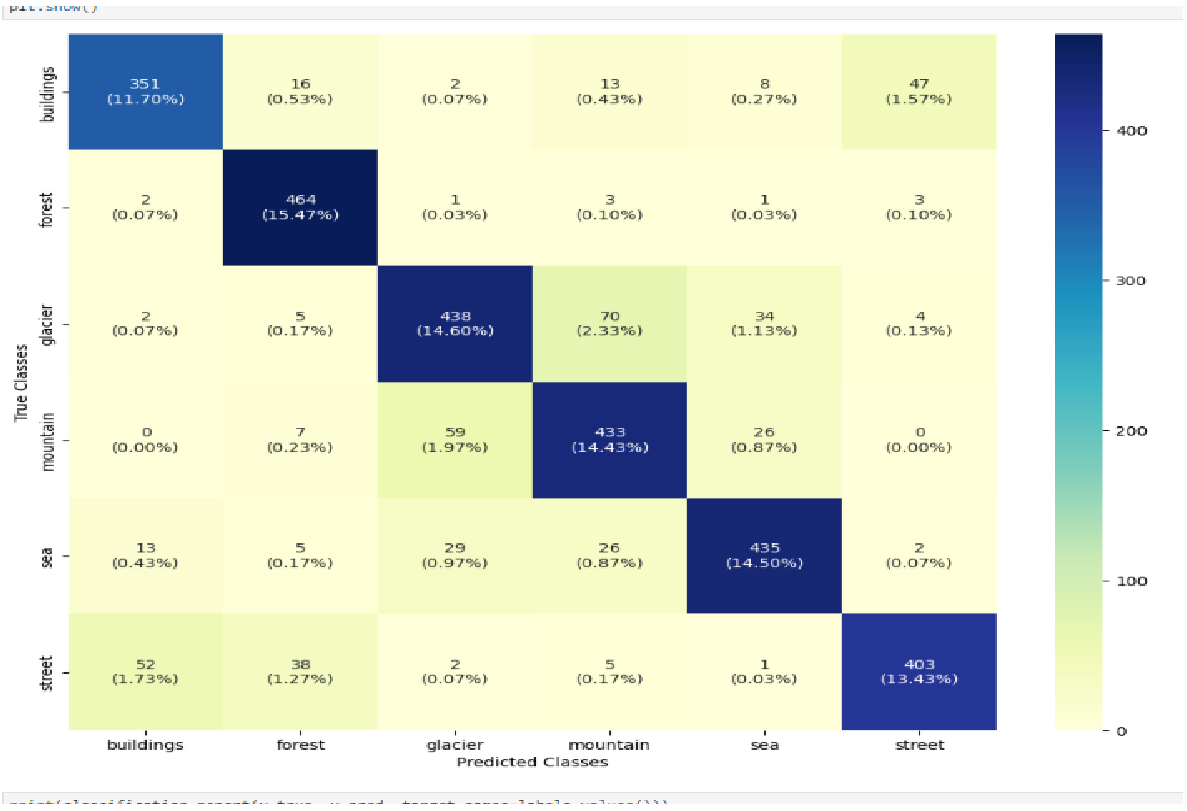
After training, the model gave total accuracy of 88.58% and loss of 33%. The training accuracy rose up from 49% to 88.58 % and training loss dropped from 204% to 33%. Similarly the validation loss dropped from 165% to 32% and validation accuracy rose up from 42% to 89%. Finally the test loss detected was 32% and test accuracy was 89%.





Confusion Matrix:

Confusion matrix drastically explains the loss and accuracy of each class The model has significantly improved during training, as indicated by the rise in training accuracy from 49% to 88.58%.Training loss has dropped from 204% to 33%, reflecting the model's better ability to minimize errors during training.



	buildings	forest	glacier	mountain	sea	street
buildings	351 (11.70%)	16 (0.53%)	2 (0.07%)	13 (0.43%)	8 (0.27%)	47 (1.57%)
forest	2 (0.07%)	464 (15.47%)	1 (0.03%)	3 (0.10%)	1 (0.03%)	3 (0.10%)
glacier	2 (0.07%)	5 (0.17%)	438 (14.60%)	70 (2.33%)	34 (1.13%)	4 (0.13%)
mountain	0 (0.00%)	7 (0.23%)	59 (1.97%)	433 (14.43%)	26 (0.87%)	0 (0.00%)
sea	13 (0.43%)	5 (0.17%)	29 (0.97%)	26 (0.87%)	435 (14.50%)	2 (0.07%)
street	52 (1.73%)	38 (1.27%)	2 (0.07%)	5 (0.17%)	1 (0.03%)	403 (13.43%)

```
              precision    recall  f1-score   support

   buildings       0.84      0.80      0.82       437
      forest       0.87      0.98      0.92       474
     glacier       0.82      0.79      0.81       553
    mountain       0.79      0.82      0.81       525
         sea       0.86      0.85      0.86       510
      street       0.88      0.80      0.84       501

    accuracy                           0.84      3000
   macro avg       0.84      0.84      0.84      3000
weighted avg       0.84      0.84      0.84      3000
```

Ended here