
KLE Technological University

Creating Value
Leveraging Knowledge

**School**

**of**

**Computer Science and Engineering**

**Machine Learning Course Poject Report**

**on**

# INTEL IMAGE CLASSIFICATION

**By:**

1. **Nitin Verma** USN: 01FE17BCS125
2. **Puneet Gupta** USN: 01FE17BCS144
3. **Rahetul Asquin** USN: 01FE17BCS148
4. **Ritbik Bharti** USN: 01FE17BCS158

**Semester: V, 2019-2020**

# Contents

# Chapter 1

# Introduction

This was a project hosted on analyticsvidhya.com and data was provided was Intel. This is image data of Natural Scenes around the world.

This Data contains around 25k images of size 150x150 distributed under 6 categories. 'buildings' - 0, 'forest' - 1, 'glacier' - 2, 'mountain' - 3, 'sea' - 4, 'street' - 5 The Train, Test and Prediction data is separated in each zip files. There are around 14k images in Train, 3k in Test and 7k in Prediction.

The goal of this project is to build powerful Neural network that can classify these images with more accuracy.

# Chapter 2

# Literature survey

The CNN is very powerful and used in most of the cases for image classification.

CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos.
Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. The computer is assigned a value from 0 to 255 to each of these numbers. his value describes the intensity of the pixel at each point.

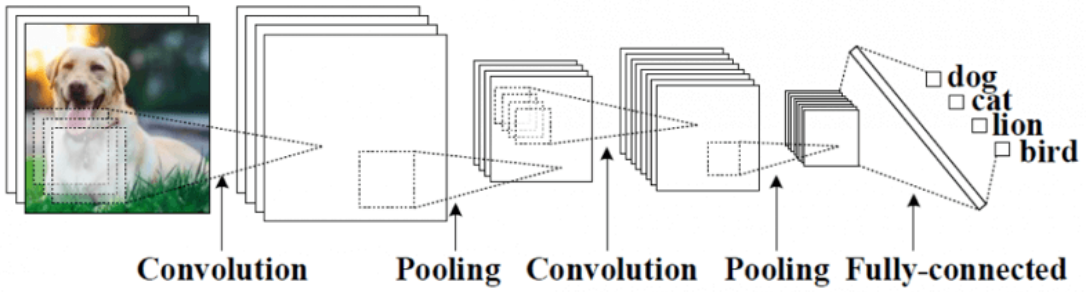
The image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.

The Convolution layer is always the first. he image (matrix with pixel values) is entered into it. Imagine that the reading of the input matrix begins at the top left of image. Next the software selects a smaller matrix there, which is called a filter (or neuron, or core). Then the filter produces convolution, i.e. moves along the input image. The filter's task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the filter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the filter across all positions, a matrix is obtained, but smaller then a input matrix.

The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a downsampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, than a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures.

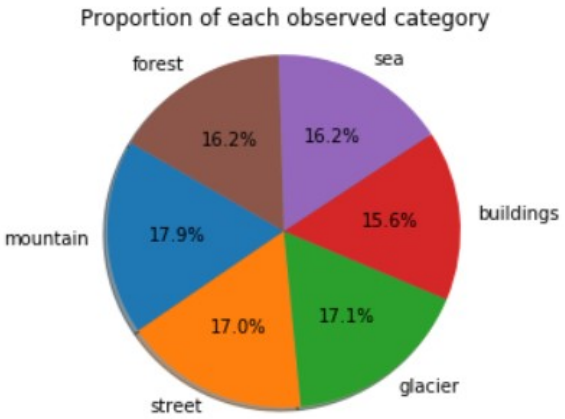
After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class.



# Chapter 3

# Implementation

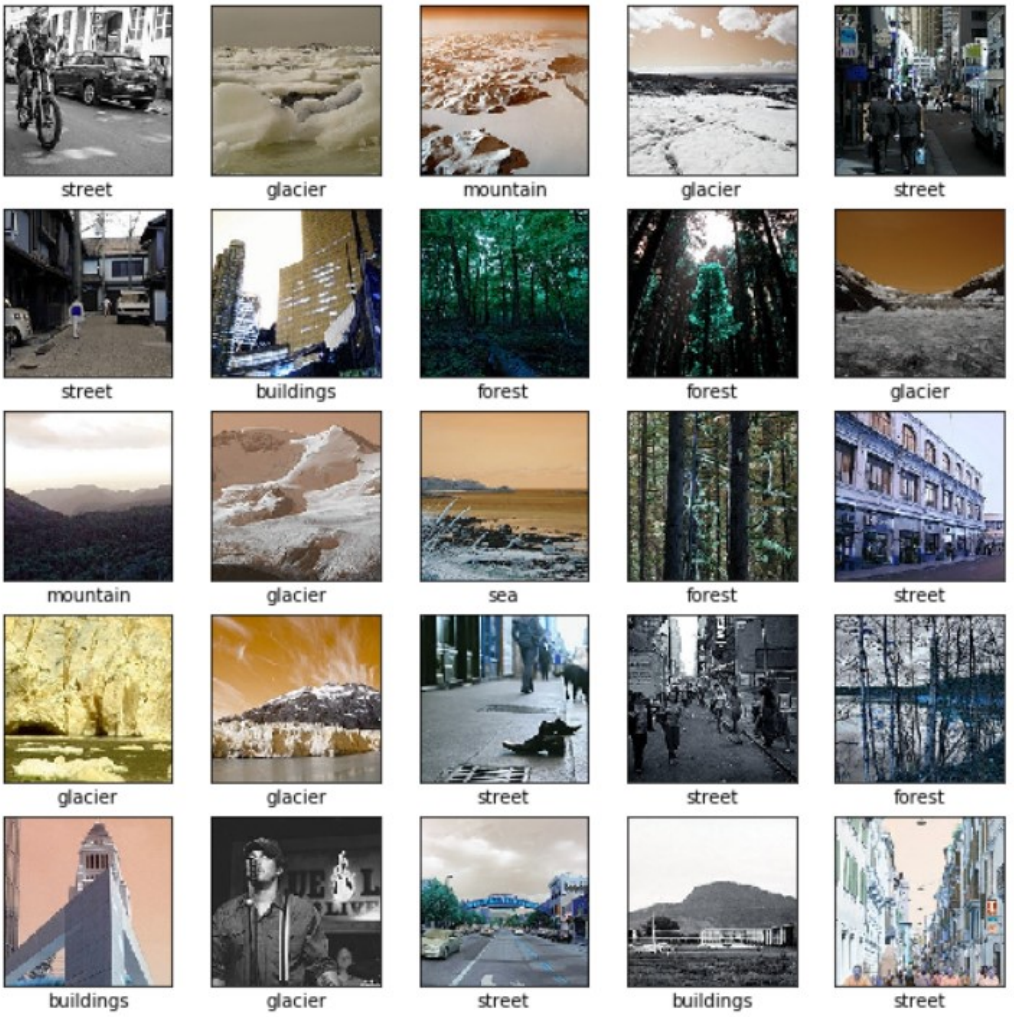
1. Loading Dataset




Number of training examples: 14034
Number of testing examples: 3000
Each image is of size: (150, 150, 3)

We loaded the dataset and then scale them so that training becomes fast.

2. We then displayed some sample images from the dataset belonging to all classes.




3. We then had to train the dataset and create a neural network for classification of the images.

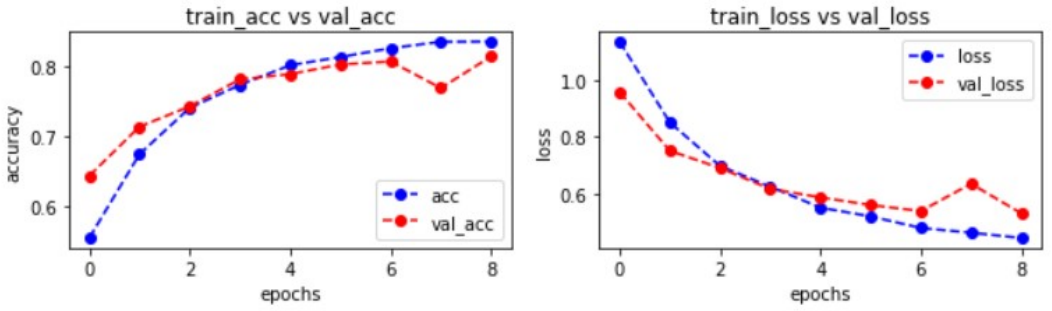
We built an easy model composed of different layers such as:
Conv2D: (32 filters of size 3 by 3) The features will be "extracted" from the image.
MaxPooling2D: The images get half sized.
Flatten: Transforms the format of the images from a 2d-array to a 1d-array of 150 150 3 pixel values.
Relu : given a value x, returns max(x, 0).
Softmax: 6 neurons, probability that the image belongs to one of the classes.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)), # the nn will learn the good filter to use
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(42, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(42, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(42, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
```

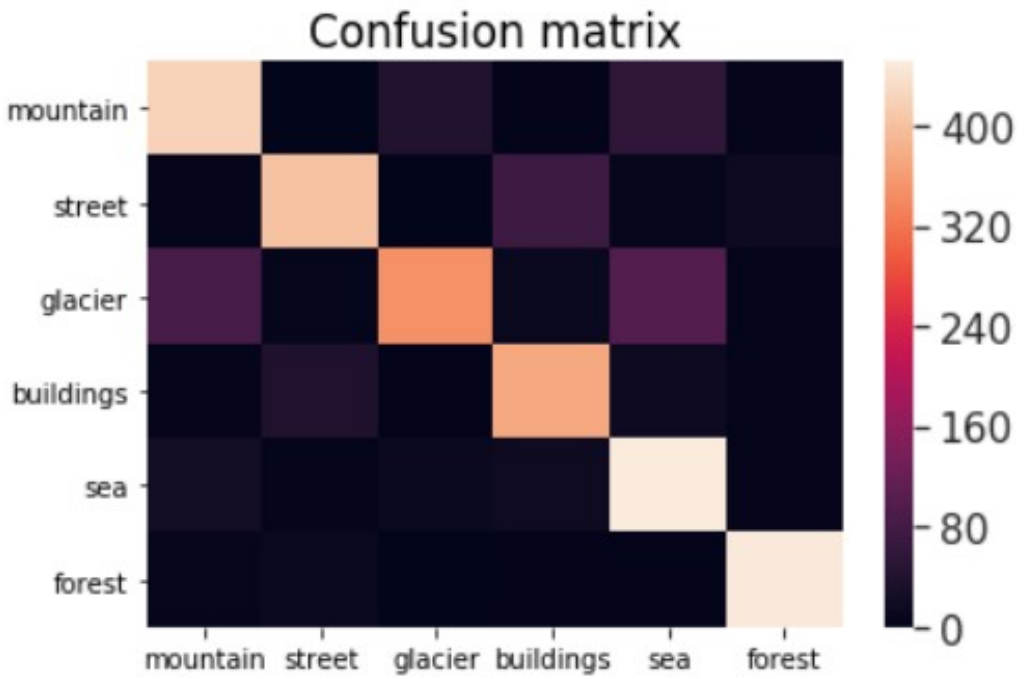
```
history = model.fit(train_images, train_labels, batch_size=128, epochs=9, validation_split = 0.2)
```

```
Train on 11227 samples, validate on 2807 samples
Epoch 1/9
11227/11227 [==============================] - 10s 888us/sample - loss: 1.1382 - accuracy: 0.5541 - val_loss: 0.9592 - val_accuracy: 0.6434
Epoch 2/9
11227/11227 [==============================] - 5s 452us/sample - loss: 0.8515 - accuracy: 0.6744 - val_loss: 0.7508 - val_accuracy: 0.7139
Epoch 3/9
11227/11227 [==============================] - 5s 480us/sample - loss: 0.6970 - accuracy: 0.7410 - val_loss: 0.6898 - val_accuracy: 0.7431
Epoch 4/9
11227/11227 [==============================] - 5s 467us/sample - loss: 0.6236 - accuracy: 0.7740 - val_loss: 0.6157 - val_accuracy: 0.7820
Epoch 5/9
11227/11227 [==============================] - 5s 458us/sample - loss: 0.5491 - accuracy: 0.8023 - val_loss: 0.5856 - val_accuracy: 0.7895
Epoch 6/9
11227/11227 [==============================] - 5s 472us/sample - loss: 0.5182 - accuracy: 0.8140 - val_loss: 0.5584 - val_accuracy: 0.8033
Epoch 7/9
11227/11227 [==============================] - 5s 464us/sample - loss: 0.4770 - accuracy: 0.8267 - val_loss: 0.5372 - val_accuracy: 0.8076
Epoch 8/9
11227/11227 [==============================] - 5s 463us/sample - loss: 0.4603 - accuracy: 0.8359 - val_loss: 0.6339 - val_accuracy: 0.7695
Epoch 9/9
11227/11227 [==============================] - 5s 456us/sample - loss: 0.4416 - accuracy: 0.8363 - val_loss: 0.5290 - val_accuracy: 0.8147
```

4.We then compared training and testing loss and accuracy for our initial model




We obtained an accuracy of 0.82 and loss of 0.5455 for test data.
We observed that the model is correctly fitting the dataset as there is no sign of under fit and over fit.

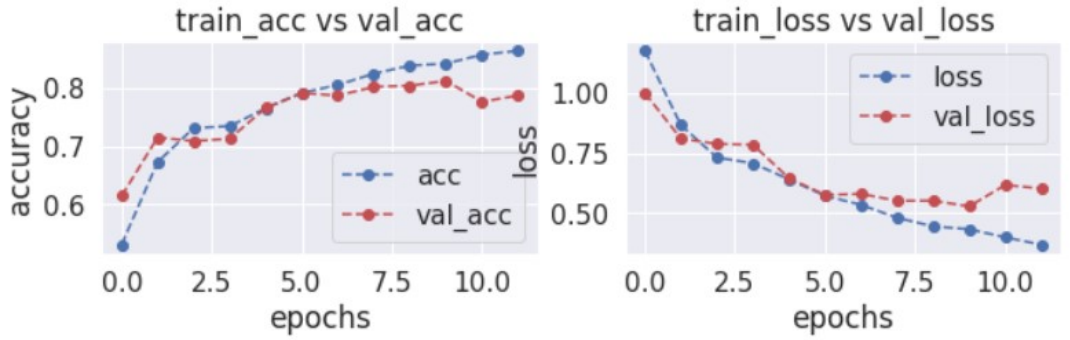



The above figure shows the confusion matrix for our initial model. It indicates how many images of each classes were correctly and incorrectly classified.

5. We then applied Dropout to reduce overfitting in our initial model.
It is a form of regularization that forces the weights in the network to take only small values, which makes the distribution of weight values more regular and the network can reduce over-fitting on small training examples.

```
model_new = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)), # the nn will learn the good filter to use
    tf.keras.layers.MaxPooling2D(2,2),
    Dropout(0.2),
    tf.keras.layers.Conv2D(42, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(42, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(42, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
```

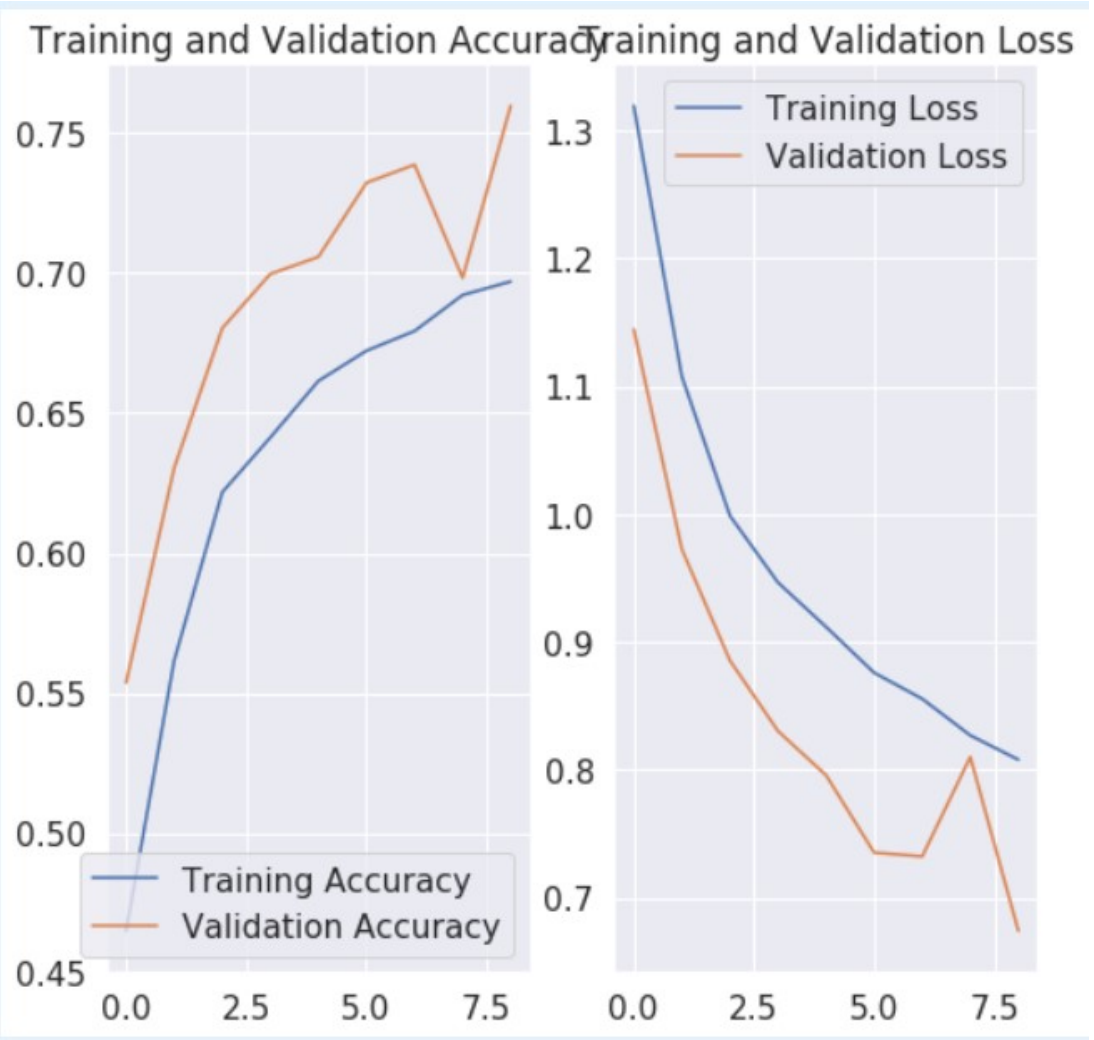
When applying 0.2 dropout to a certain layer, it randomly kills 20 percentage of the output units in each training epoch.




We obtained an accuracy of 0.8273 after Dropouts.

6. Data Augmentation : Overfitting generally occurs when there are a small number of training examples. One way to fix this problem is to augment the data set so that it has a sufficient number of training examples. Data augmentation takes the approach of generating more training data from existing training samples by augmenting the samples using random transformations that yield believable-looking images.
We then applied rescale, 45 degree rotation, width shift, height shift, horizontal flip and zoom augmentation to the training images.

```
image_gen_train = ImageDataGenerator(
                    rescale=1./255,
                    rotation_range=45,
                    width_shift_range=.15,
                    height_shift_range=.15,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )
```



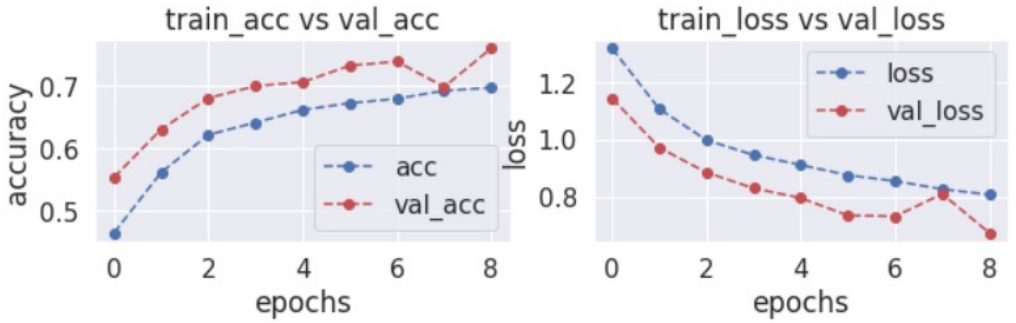

The accuracy didn't increase after doing augmentation.

7. We then added more layers to the neural network and also changed the adam's learning rate to 0.0001 and validation split to 0.30

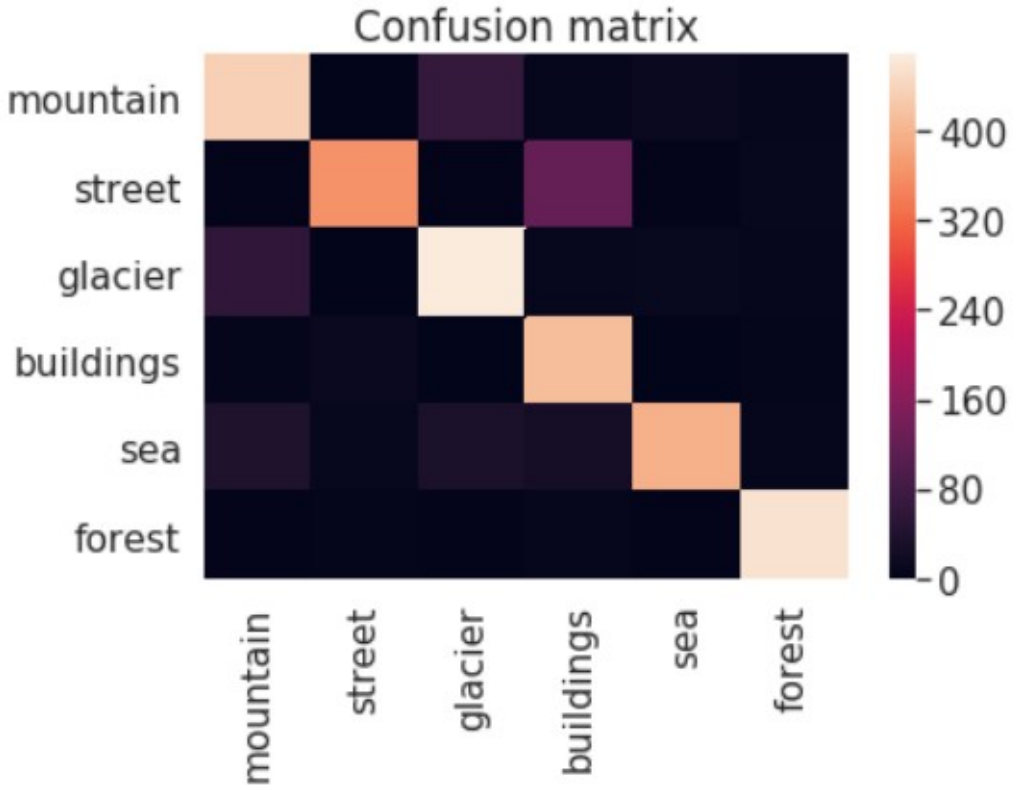
```
model = Models.Sequential()

model.add(Layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(150,150,3)))
model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
model.add(Layers.MaxPool2D(5,5))
model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(140,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(100,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(50,kernel_size=(3,3),activation='relu'))
model.add(Layers.MaxPool2D(5,5))
model.add(Layers.Flatten())
model.add(Layers.Dense(180,activation='relu'))
model.add(Layers.Dense(100,activation='relu'))
model.add(Layers.Dense(50,activation='relu'))
model.add(Layers.Dropout(rate=0.5))
model.add(Layers.Dense(6,activation='softmax'))
```

We also tested our new model to check training and validation loss and accuracy, their differences so that degree of underfiiting or overfitting can be known.




As we can see from above graphs that there is very less difference between training and validation error , hence our model is not overfitting.




The above figure shows the confusion matrix for our initial model. It indicates how many images of each classes were correctly and incorrectly classified.

# Chapter 4

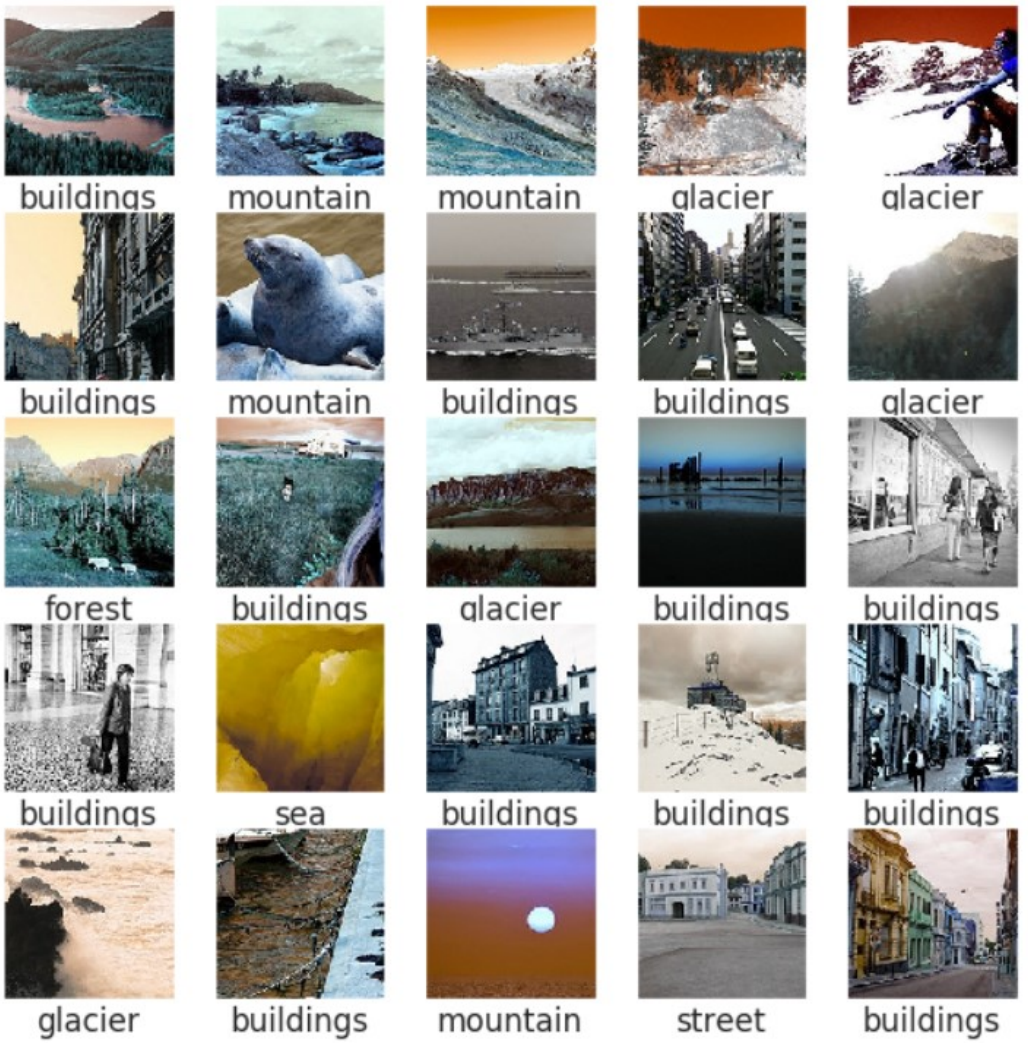
# Results

1. CNN with 3 hidden layers and learning rate 0.001 have an accuracy of 0.82

2. CNN with Dropouts of 0.2 gave an accuracy of 0.83

3. CNN with Augmentation of dataset didn't improvised the accuracy.

4. CNN with more layers and validation split=0.30 and learning rate = 0.0001 gave an accuracy of 0.86

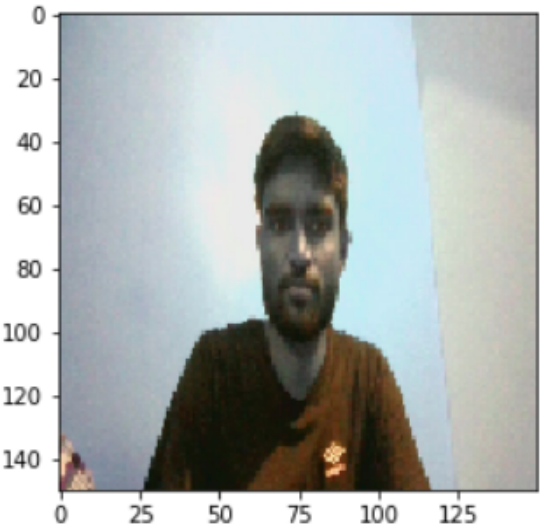



The given figure shows some of the misclassified images in each class.

We then tested our model on our custom images.
We put one image of a our group member and one forest image

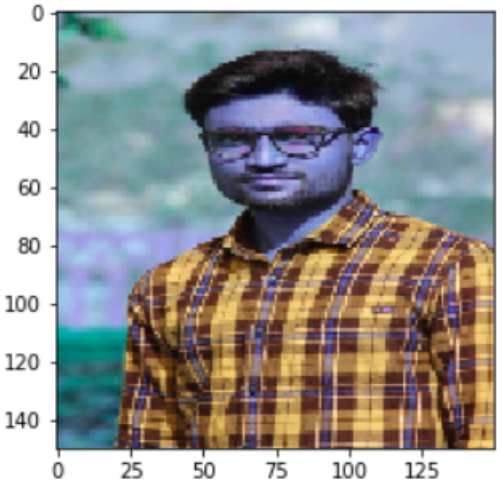
```
In [11]: plt.imshow(img/256.)
```

```
Out[11]: <matplotlib.image.AxesImage at 0x7f6f6d13aef0>
```

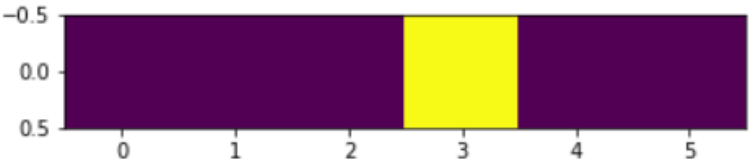



```
In [22]: plt.imshow(i3/256.)
```

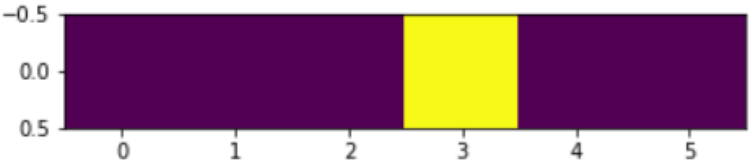
```
Out[22]: <matplotlib.image.AxesImage at 0x7f6f6cc437b8>
```



```
In [23]: plt.imshow(predictions3/256.)
Out[23]: <matplotlib.image.AxesImage at 0x7f6f6cbaa390>
```



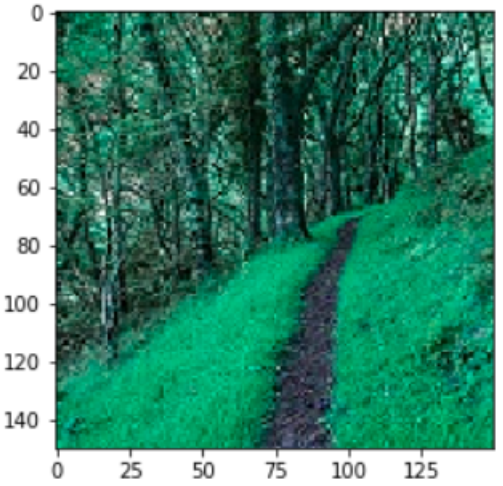

```
In [12]: plt.imshow(predictions/256.)
Out[12]: <matplotlib.image.AxesImage at 0x7f6f6cc67cc0>
```



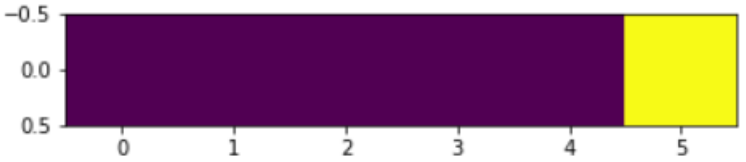

The person's image was incorrectly classified as our model is not trained on people's images

We put one forest image for testing and our model correctly predicted it as forest.

```
In [29]: plt.imshow(i4/256.)
Out[29]: <matplotlib.image.AxesImage at 0x7f6f6cb16390>
```




```
In [30]: plt.imshow(predictions4/256.)
Out[30]: <matplotlib.image.AxesImage at 0x7f6f6ca6df28>
```



# Chapter 5

# References

1. https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

2. https://www.tensorflow.org/tutorials/images/classificationdata$_a ugmentation$