# Image Classification

## 1. Load the data set:

The dataset consists of six classes with 14034 training and 3000 testing examples. The split ratio of each classes shown below:

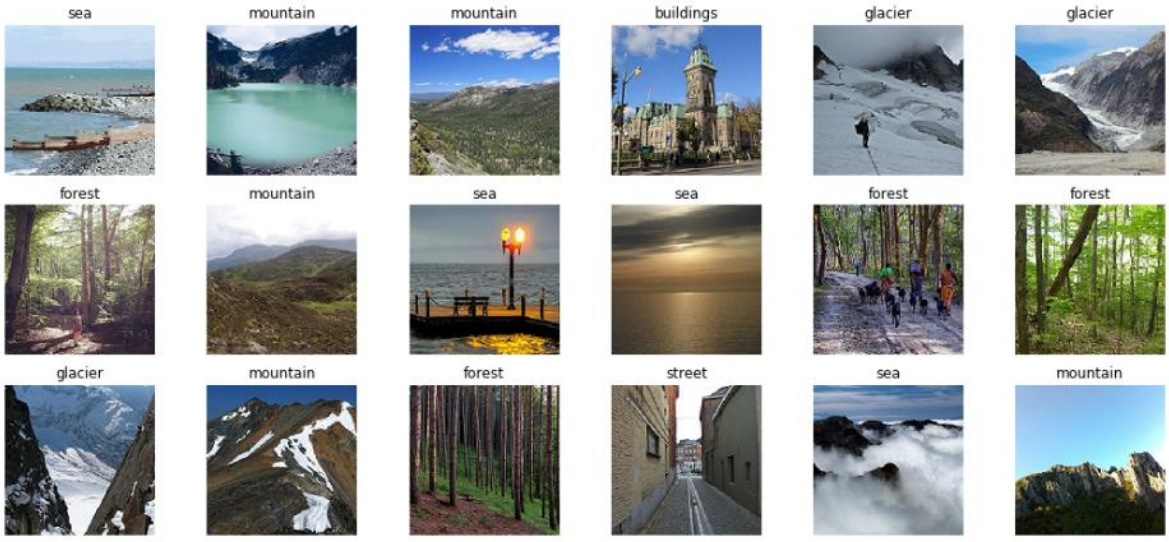
| Category | Train | Test |
|---|---|---|
| buildings | 2191 | 437 |
| forest | 2271 | 474 |
| glacier | 2404 | 553 |
| mountain | 2512 | 525 |
| sea | 2274 | 510 |
| street | 2382 | 501 |
| Total | 14034 | 3000 |

I have read the images using opencv and resize them 224*224*3, so that we can also use them in pretrained imagenet model. Final loaded Dataset is saved in numpy array that could be use able in the training time.

```
#function for loading dataset.
def data_reading(Images_path,classes,img_size=(224,224)):
    images=[]
    labels=[]
    for i in range(len(Images_path)):
        img=cv2.imread(Images_path[i])#reading the images
        img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)#converting the bgr image read by cv2 to rgb
        img=cv2.resize(img,img_size,interpolation=cv2.INTER_CUBIC)# resizing the image to 224*224*3
        images.append(img)
        label=classes[Images_path[i].split('\\')[1]]#finding the class index
        labels.append(label)
    images=np.array(images)
    labels=np.array(labels)
    return images,labels
```

We have visualized first 18 images to check the different categories images in the dataset.

```
# Showing first 18 images. and their labels using pyplot.
plt.figure(figsize=(18,8))
for i in range (0,18):
    plt.subplot(3,6,i+1)
    img1=train_images[i]
    plt.imshow(img1)
    plt.title(classes_names[train_labels[i]])
    plt.axis('off')
plt.show()
```



## 2. Perform Image Augmentation:

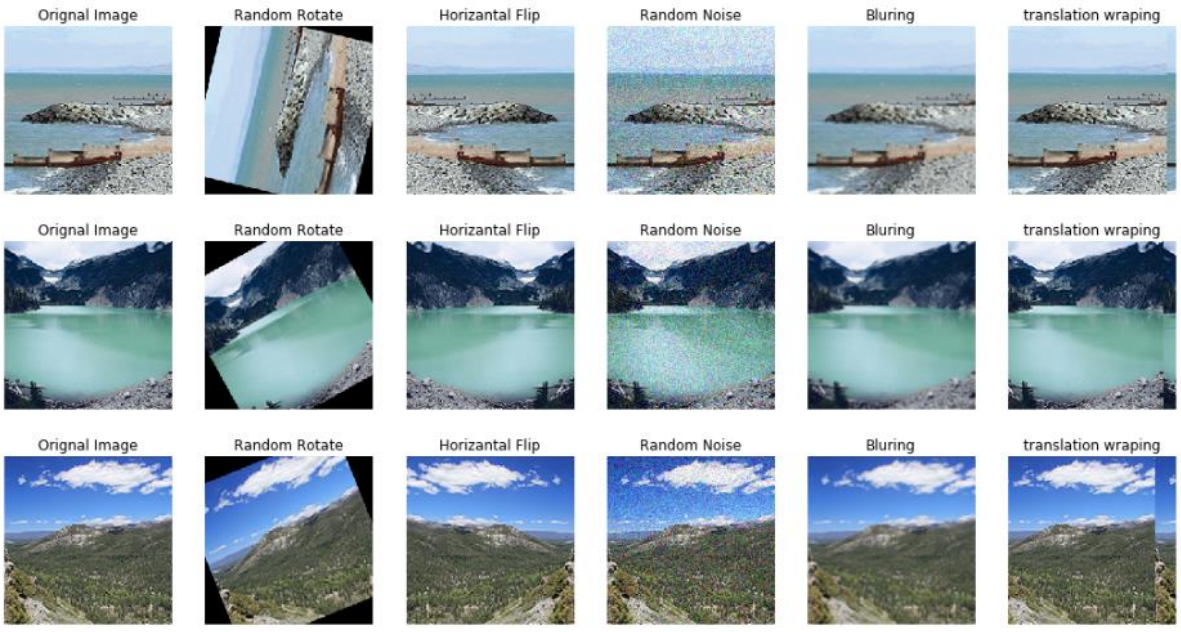
I have performed Randomly five type of data augmentation techniques on images which are describes below:

- **Random Rotate:** Randomly rotating image in clock and anti-clockwise direction
- **Horizontal Flip:** Flipping the image in Horizontal Direction
- **Random Noise:** Adding Random Noise in the image
- **Blurring:** Adding blurring through gaussian filter.
- **Translation Wrapping:** Translating specific part of image and placing it in the other part.

Below function read the image and then apply one of these data augmentation technique Randomly.

```
#function for loading dataset.
def data_reading_with_augmentation(Images_path,classes,img_size=(224,224)):
    images=[]
    labels=[]
    for i in range(len(Images_path)):
        img=cv2.imread(Images_path[i])#reading the images
        img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)#converting the bgr image read by cv2 to rgb
        img=cv2.resize(img,img_size,interpolation=cv2.INTER_CUBIC)# resizing the image to 224*224*3
        images.append(img)
        label=classes[Images_path[i].split('\\')[1]]#finding the class index
        labels.append(label)

        random_number=random.randint(0,4)
        if random_number==0:
            #-negative for clockwise and positive for anticlockwise
            images.append(rotate(img, random.randint(-60,60)))
            labels.append(label)
        if random_number==1:
            # image fliping horizantal
            images.append(np.fliplr(img))
            labels.append(label)
        if random_number==2:
            # random image noise
            images.append(random_noise(img))
            labels.append(label)
        if random_number==3:
            # image blurring
            images.append(cv2.GaussianBlur(img, (5,5),0))
            labels.append(label)
        if random_number==4:
            # image translation wraping
            images.append(warp(img, AffineTransform(translation=(random.randint(-30,30),0)), mode="wrap"))
            labels.append(label)

    images=np.array(images)
    labels=np.array(labels)
    return images,labels
```

3 images examples are shown below that how these augmentation effect on image.



## 3. Build your Convolutional Neural Network:

I have tried developing multiple models using different cnn layers features and different hyper parameters. Some of model architecture are explained below:

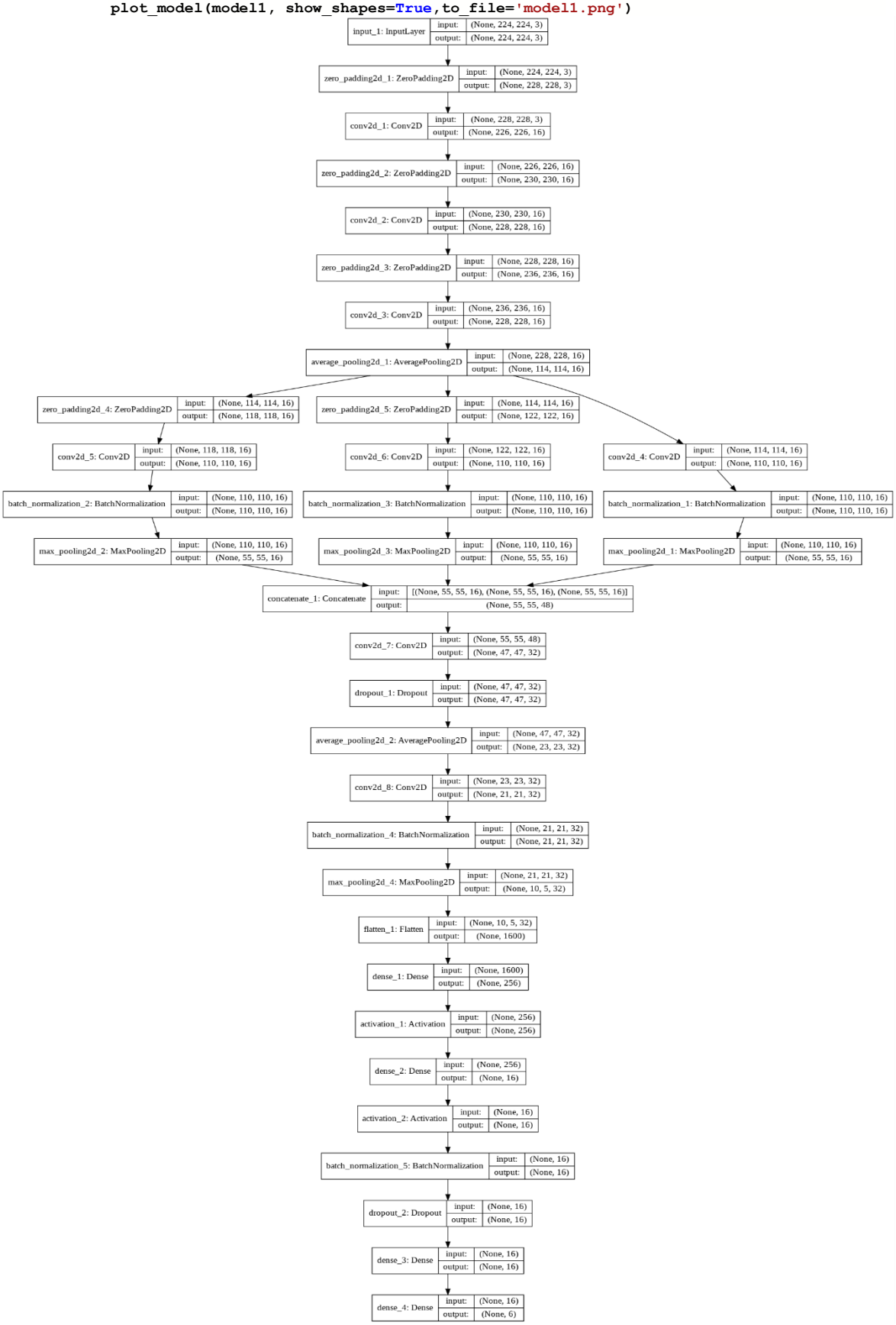
### 3.1 Model1

The first model that I have tried has 8 layers of cnn with different combination of max pooling, batch normalization and Dropout. At the we flatten the cnn layer and use 2 dense layers. We have used inception model style of layers as well to further capture features with different kernel sizes. We have use relu activation function in Dense layers and softmax function in last Dense layers. Softmax help us to generate probabilites of each class at the end, in it we try to maximize the probabilities. We have used categorical_crossentropy loss function because we have 10 number of classes at the end, in which all entries are 0 except the correspondene class is 1.

```
class CNNNet1:
    def build(width, height, depth, classes):
        # initialize the model
        input1 = Input(shape=(height, width,depth))

        x1=layers.ZeroPadding2D(padding=(2, 2))(input1)## Zero padding on the sides
        x1=layers.Conv2D(16, (3, 3),padding='valid',activation='relu',use_bias=False)(x1) ##CNN LAYER1
        x1=layers.ZeroPadding2D(padding=(2, 2))(x1)#Zero padding so information not loss at the edges.
        x1=layers.Conv2D(16, (3, 3),padding='valid',activation='relu',use_bias=False)(x1)##CNN LAYER2
        x1=layers.ZeroPadding2D(padding=(4, 4))(x1)
        x1=layers.Conv2D(16, (5, 5),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(x1)##CNN LAYER3
        x1=layers.AveragePooling2D(pool_size=(2, 2))(x1)# average pooling layers. Reducing the size of layer, by replacing the av

        x1_1=layers.Conv2D(16, (3, 3),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(x1)##CNN LAYER4
        x1_1=layers.BatchNormalization()(x1_1)# batch nomalization layer
        x1_1=layers.MaxPooling2D(pool_size=(2, 2))(x1_1)# Max pooling layer.

        x1_2=layers.ZeroPadding2D(padding=(2, 2))(x1)
        x1_2=layers.Conv2D(16, (5, 5),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(x1_2)##CNN LAYER5
        x1_2=layers.BatchNormalization()(x1_2)# batch nomalization layer
        x1_2=layers.MaxPooling2D(pool_size=(2, 2))(x1_2)# Max pooling layer.

        x1_3=layers.ZeroPadding2D(padding=(4, 4))(x1)
        x1_3=layers.Conv2D(16, (7, 7),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(x1_3)##CNN LAYER6
        x1_3=layers.BatchNormalization()(x1_3)# batch nomalization layer
        x1_3=layers.MaxPooling2D(pool_size=(2, 2))(x1_3)# Max pooling layer.

        concatenated0 = layers.concatenate([x1_1, x1_2,x1_3], axis=-1)
        concatenated0=layers.Conv2D(32, (5, 5),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(concatenated
        concatenated0=layers.Dropout(0.1)(concatenated0) #dropput at the probability of 10%.
        concatenated0=layers.AveragePooling2D(pool_size=(2, 2))(concatenated0)
#         # second set of CONV => RELU => POOL layers
        concatenated0=layers.Conv2D(32, (3, 3),activation='relu')(concatenated0)##CNN LAYER8
        concatenated0=layers.BatchNormalization()(concatenated0)# batch nomalization layer
        concatenated0=layers.MaxPooling2D(pool_size=(2, 4))(concatenated0)# Max pooling layer.

        # first (and only) set of FC => RELU layers
        x3=layers.Flatten()(concatenated0)
        x3=layers.Dense(256)(x3)##FC LAYER1
        x3=layers.Activation("relu")(x3)

        x3=layers.Dense(16)(x3)##FC LAYER2
        x3=layers.Activation("relu")(x3)
        x3=layers.BatchNormalization()(x3)# batch nomalization layer
        x3 =layers.Dropout(0.1)(x3)#dropput at the probability of 10%.
        x3 =layers.Dense(16, activation="relu")(x3)##FC LAYER3
        predictions = Dense(classes, activation='softmax')(x3)## OUTPUT LAYER
        model = Model(input1, predictions)
        model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.1),metrics=["acc"])
        return model

model1 = CNNNet1.build(224,224,3,len(classes_names))
```

## a. Model1 Architecture visualization:

```
from keras.utils import plot_model

plot_model(model1, show_shapes=True,to_file='model1.png')
```



# 4. Model1 Compiling and Results:

## *a. Model Compiling:*

We have use Adam Optimizer and Train the model for 80 epochs in total dataset. Note that we are loading the dataset in chunks because it takes a lot of Ram in system memory.

```
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.1),metrics=["acc"])
```

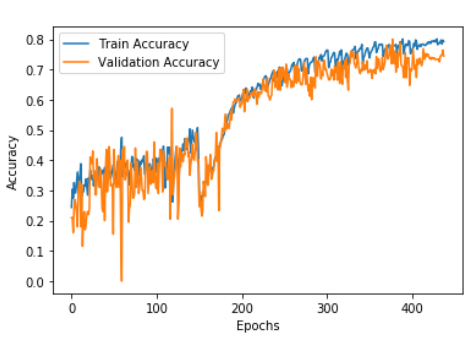
```
acc=[]
val_acc=[]
loss=[]
val_loss=[]

for g in range(10):
    random.shuffle(all_train_images_path)
    for i in range(0,len(all_train_images_path),4000):#len(all_train_images_path)
        train_images_aug=[]
        train_images_aug,train_labels_aug=data_reading_with_augmentation(all_train_images_path[i:i+4000],classes,img_size=(224,
        train_images_aug=train_images_aug/255.
        train_labels_aug=np_utils.to_categorical(train_labels_aug)

        batch_size=64
        hist=model1.fit(train_images_aug,train_labels_aug,batch_size=batch_size,epochs=8,validation_split=0.1)

        acc=acc+hist.history['acc']
        val_acc=val_acc+hist.history['val_acc']
        loss=loss+hist.history['loss']
        val_loss=val_loss+hist.history['val_loss']
```

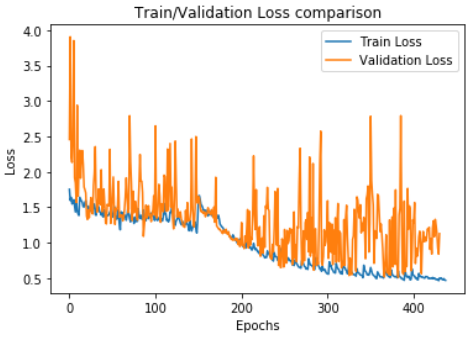
```
Train on 7200 samples, validate on 800 samples
Epoch 1/8
7200/7200 [==============================] - 79s 11ms/step - loss: 0.5676 - acc: 0.7665 - val_loss: 0.7067 - val_acc: 0.7337
Epoch 2/8
7200/7200 [==============================] - 79s 11ms/step - loss: 0.5299 - acc: 0.7764 - val_loss: 0.9248 - val_acc: 0.7175
Epoch 3/8
7200/7200 [==============================] - 79s 11ms/step - loss: 0.5275 - acc: 0.7783 - val_loss: 0.8089 - val_acc: 0.7200
Epoch 4/8
7200/7200 [==============================] - 79s 11ms/step - loss: 0.5131 - acc: 0.7824 - val_loss: 0.8777 - val_acc: 0.7375
Epoch 5/8
7200/7200 [==============================] - 79s 11ms/step - loss: 0.5176 - acc: 0.7757 - val_loss: 1.0852 - val_acc: 0.7237
```

## *b. Train Validation Accuracy and Train Validation Loss comparison:*

The results show that model is not overfitting on training data, it reach maximum 80 accuracy on training data. But validation loss is quite more than training loss. We use matplotlib library to plot the accuracy and loss graph.

```
import matplotlib.pyplot as plt
plt.plot(acc,label='Train Accuracy')
plt.plot(val_acc,label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('Train_Validation_Accuracy_comparison1.png')
plt.show()

plt.plot(loss,label='Train Loss')
plt.plot(val_loss,label='Validation Loss')
plt.title('Train/Validation Loss comparison')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('Train_Validation_Loss_comparison1.png')
plt.show()
```




### *c. Accuracy on Test Data:*

We have received 80% Accuracy on Test images of 3000. We are using sklearn library to measure the accuracy from model prediction.

```
from sklearn.metrics import accuracy_score
predicted_labels = model1.predict(test_images/255) #predicting on test images
predicted_labels=np.argmax(predicted_labels,axis=1) # Converting Returned categorical to numbers.
accuracy = accuracy_score(test_labels, predicted_labels) * 100
accuracy
```

```
80.83333333333333
```

### *d. Confusion Matric on Test Data:*

In the confusion matric we can see that most street case are classified as buildings as there are some similarity between them. And glacier is some classifies as mountain because of some similarity.

Confusion matrix

| | buildings | forest | glacier | mountain | sea | street |
|---|---|---|---|---|---|---|
| **buildings** | 339 | 3 | 2 | 6 | 27 | 60 |
| **forest** | 5 | 452 | 2 | 3 | 0 | 12 |
| **glacier** | 3 | 4 | 393 | 102 | 48 | 3 |
| **mountain** | 3 | 3 | 70 | 413 | 33 | 3 |
| **sea** | 12 | 2 | 58 | 16 | 415 | 7 |
| **street** | 56 | 11 | 5 | 6 | 10 | 413 |

```
from sklearn.metrics import confusion_matrix
import pandas as pd
print ("Confusion matrix")
#confusion matrix between predicted and actual truths
pd.DataFrame(confusion_matrix(test_labels,predicted_labels),columns=classes_names,index=classes_names)
```

## 5. Redo steps 3 and 4 with different structures and different values of the hyperparameters:

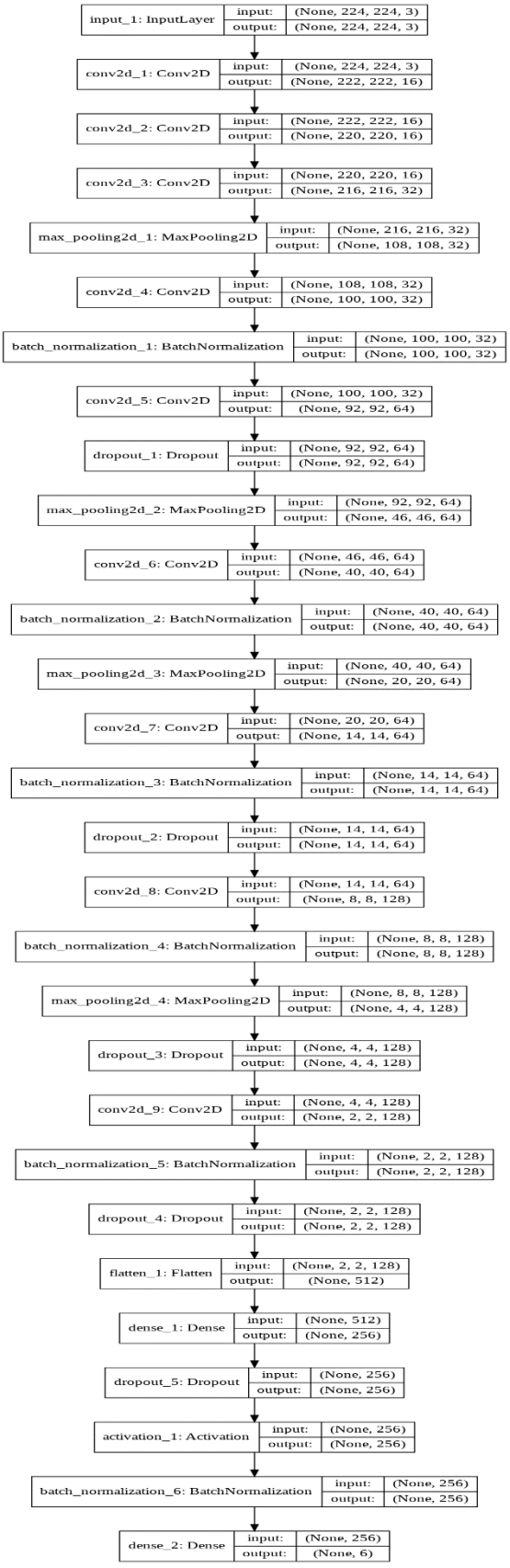
### 5.1 Model2

Our First model has less number of training parameters, so the next model I have created so that it has specific number of parameters that can learns more features. The 2nd model that I have tried has 9 layers of cnn with different combination of max pooling, batch normalization and Dropout. At the we flatten the cnn layer and use 2 dense layers. We have use relu activation function in Dense layers and softmax function in last Dense layers.

```
class CNNNet2:
    def build(width, height, depth, classes):
        # initialize the model
        input1 = Input(shape=(height, width,depth))

        x1=layers.Conv2D(16, (3, 3),padding='valid',activation='relu',use_bias=False)(input1) #cnn layer1
        x1=layers.Conv2D(16, (3, 3),padding='valid',activation='relu',use_bias=False)(x1) #cnn layer2
        x1=layers.Conv2D(32, (5, 5),padding='valid',activation='relu',use_bias=False)(x1) #cnn layer3
        x1=layers.MaxPooling2D(pool_size=(2, 2))(x1) # max pooling layer 1

        x1=layers.Conv2D(32, (5, 5),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(x1)#cnn layer4
        x1=layers.BatchNormalization()(x1) #batch normalization layer 1

        x1=layers.Conv2D(64, (5, 5),dilation_rate=(2,2),padding='valid',activation='relu',use_bias=False)(x1)#cnn layer5
        x1=layers.Dropout(0.1)(x1)
        x1=layers.MaxPooling2D(pool_size=(2, 2))(x1) # max pooling layer 2

#         # second set of CONV => RELU => POOL layers
        x1=layers.Conv2D(64, (7, 7),activation='relu')(x1)#cnn layer6
        x1=layers.BatchNormalization()(x1)#batch normalization layer 2
        x1=layers.MaxPooling2D(pool_size=(2, 2))(x1) # max pooling layer 3

        x1=layers.Conv2D(64, (7, 7),activation='relu')(x1)#cnn layer7
        x1=layers.BatchNormalization()(x1)#batch normalization layer 3
        x1=layers.Dropout(0.1)(x1)


        x1=layers.Conv2D(128, (7, 7),activation='relu')(x1)#cnn layer8
        x1=layers.BatchNormalization()(x1)#batch normalization layer 4
        x1=layers.MaxPooling2D(pool_size=(2, 2))(x1) # max pooling layer 4
        x1=layers.Dropout(0.1)(x1)

        x1=layers.Conv2D(128, (3, 3),activation='relu')(x1)#cnn layer9
        x1=layers.BatchNormalization()(x1)#batch normalization layer 1
        x1=layers.Dropout(0.1)(x1)

        x1=layers.Flatten()(x1)
        x1=layers.Dense(256)(x1)
        x1=layers.Dropout(0.3)(x1)
        x1=layers.Activation("relu")(x1)
        x1=layers.BatchNormalization()(x1)

        # output layer
        predictions = Dense(classes, activation='softmax')(x1)s
        # At model instantiation, we specify the two inputs and the output:
        model = Model(input1, predictions)
        model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.01),metrics=["acc"])
        # return the constructed network architecture
        return model

model2 = CNNNet2.build(224,224,3,len(classes_names))
```

## a. Model2 Architecture visualization:

```
from keras.utils import plot_model

plot_model(model2, show_shapes=True,to_file='model2.png')
```



## b. Model2 Results:

### *i. Model Compiling:*

We have use Adam Optimizer and Train the model for 80 epochs in total dataset. Note that we are loading the dataset in chunks because it takes a lot of Ram in system memory.

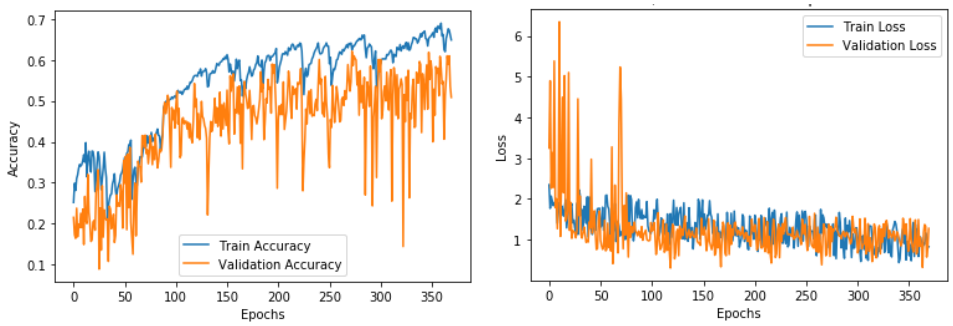
```
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.001),metrics=["acc"])
```

```
acc=[]
val_acc=[]
loss=[]
val_loss=[]

for g in range(10):
    random.shuffle(all_train_images_path)
    for i in range(0,len(all_train_images_path),4000):#len(all_train_images_path)
        train_images_aug=[]
        train_images_aug,train_labels_aug=data_reading_with_augmentation(all_train_images_path[i:i+4000],classes,img_size=(224,22
        train_images_aug=train_images_aug/255. #Normalization of features
        train_labels_aug=np_utils.to_categorical(train_labels_aug) #converting number based classes to categorical classes.

        batch_size=64
        hist=model2.fit(train_images_aug,train_labels_aug,batch_size=batch_size,epochs=8,validation_split=0.1)

        acc=acc+hist.history['acc']
        val_acc=val_acc+hist.history['val_acc']
        loss=loss+hist.history['loss']
        val_loss=val_loss+hist.history['val_loss']
```

### *ii. Train Validation Accuracy and Train Validation Loss comparison:*

The results show that model is not overfitting on training data, it reaches maximum 80 accuracy on training data. But validation loss is quite more than training loss. We use matplotlib library to plot the accuracy and loss graph.

```
import matplotlib.pyplot as plt
plt.plot(acc,label='Train Accuracy')
plt.plot(val_acc,label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('Train_Validation_Accuracy_comparison1.png')
plt.show()

plt.plot(loss,label='Train Loss')
plt.plot(val_loss,label='Validation Loss')
plt.title('Train/Validation Loss comparison')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('Train_Validation_Loss_comparison1.png')
plt.show()
```




### *iii. Accuracy on Test Data:*

We have received 78.2% Accuracy on Test images of 3000. We are using sklearn library to measure the accuracy from model prediction.

```
from sklearn.metrics import accuracy_score
predicted_labels = model2.predict(test_images/255)#predicting on test images
predicted_labels=np.argmax(predicted_labels,axis=1)# Converting Returned categorical to numbers.
accuracy = accuracy_score(test_labels, predicted_labels) * 100
accuracy
```

```
78.2
```

### *iv. Confusion Matric on Test Data:*

In the confusion matric we can see that most street case are classified as buildings as there are some similarity between them. And glacier is some classifies as mountain because of some similarity.

```
Confusion matrix
```

| | buildings | forest | glacier | mountain | sea | street |
|---|---|---|---|---|---|---|
| **buildings** | 325 | 15 | 10 | 16 | 18 | 53 |
| **forest** | 0 | 461 | 2 | 6 | 2 | 3 |
| **glacier** | 7 | 14 | 367 | 98 | 65 | 2 |
| **mountain** | 1 | 22 | 55 | 409 | 37 | 1 |
| **sea** | 5 | 3 | 37 | 81 | 379 | 5 |
| **street** | 40 | 33 | 6 | 9 | 8 | 405 |

```
from sklearn.metrics import confusion_matrix
import pandas as pd
print ("Confusion matrix")
#confusion matrix between predicted and actual truths
pd.DataFrame(confusion_matrix(test_labels,predicted_labels),columns=classes_names,index=classes_names)
```

## 6. Save your Final Model:

Model1 performs better than the model2 so we have saved it weights.

| Classifier/Model | Accuracy |
|---|---|
| Model1 | 80.83 |
| Model2 | 78.2 |

```
model1.save('model/model1.h5')
```

## 7. Use your model with your own photos:

Testing it using Model1 as it has given good accuracy on testing data. We have download 6 images of different classes from internet and we use those images to test the model. Note that those images are never seen by model.

```
from keras.models import load_model
model1 = load_model('model/model1.h5')# loading model1
```

```
import glob
images_path=glob.glob('Testing_images/*')# loading own testing images.
```

```
img_size=224,224
All_images=[]
for i in range(len(images_path)):
    img=cv2.imread(images_path[i])#reading the images
    img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)#converting the bgr image read by cv2 to rgb
    img=cv2.resize(img,img_size,interpolation=cv2.INTER_CUBIC)# resizing the image to 224*224*3
    All_images.append(img)
All_images=np.array(All_images)
All_images.shape
```

```
(6, 224, 224, 3)
```

```
predicted_labels=np.argmax(model1.predict(All_images/255),axis=1)# predicting using the model and converting it to genera
predicted_labels
```

```
array([0, 1, 2, 3, 4, 5], dtype=int64)
```

**Results:**

```
# Showing first tested images. and their predicted labels using pyplot.
plt.figure(figsize=(15,10))
for i in range (0,len(images_path)):
    plt.subplot(int(len(images_path)/6),len(images_path),i+1)
    img1=All_images[i]
    plt.imshow(img1)
    plt.title(classes_names[predicted_labels[i]])
    plt.axis('off')
plt.show()
```


## 8. Bonus: use a pre-trained model like ResNet to improve your results.:

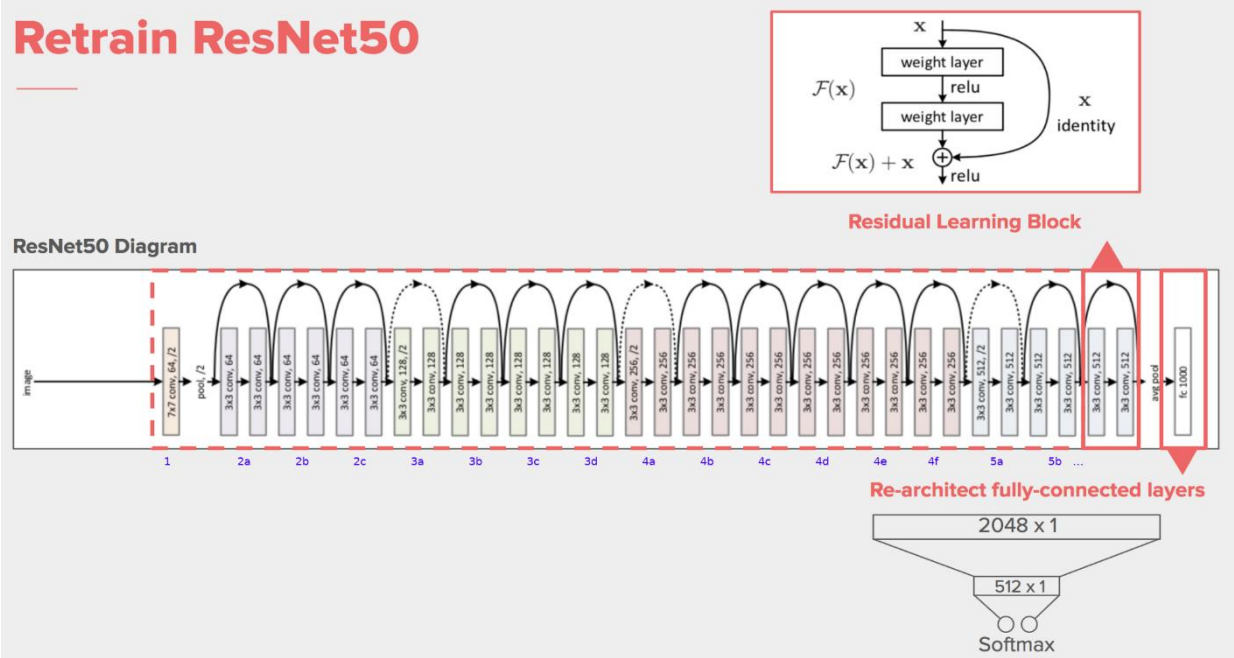
We have tried pretrained Resnet50 model that has trained on imagenet dataset. The model output 1000 neurons at the output, we have connected 2 more dense layers at the end to narrow down the output to 6 classes. We have used softmax at the last layer and cateogircal_crossentropy loss at the end.

The benefit of using pretrained model that its train faster on similar data, and converge to optimal in less time.

```
def Resnet50_model():
    initial_model: Model =ResNet50(weights='imagenet',  input_shape=(224,224,3), classes=1000)

    initial_model.trainable = True
    for i, layer in enumerate(initial_model.layers):
        layer.trainable = True

    # We include MobileNetV2 layers directly in the final model.
    initial_model_output = initial_model.output

    x = layers.Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01))(initial_model_output)
    predictions = layers.Dense(len(classes_names), activation='softmax')(x)

    model = Model(initial_model.input, predictions)
    model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.001),metrics=["acc"])
    return model
```

## a. Resnet50 Architecture visualization:

Resnet is quite deep network structure in which input of one layer before cnn operation added to the output to avoid feature loss. The structure of resnet shown below:

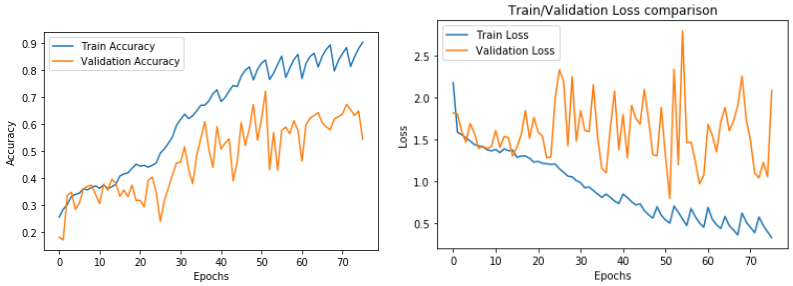



## b. Resnet50 Results:

### i. *Train Validation Accuracy and Train Validation Loss comparison:*

The results show that model is not overfitting on training data, it reaches maximum 80 accuracy on training data. But validation loss is quite more than training loss. We use matplotlib library to plot the accuracy and loss graph.

```
:  import matplotlib.pyplot as plt
   plt.plot(acc,label='Train Accuracy')
   plt.plot(val_acc,label='Validation Accuracy')
   plt.xlabel('Epochs')
   plt.ylabel('Accuracy')
   plt.legend()
   plt.savefig('Train_Validation_Accuracy_comparison1.png')
   plt.show()

   plt.plot(loss,label='Train Loss')
   plt.plot(val_loss,label='Validation Loss')
   plt.title('Train/Validation Loss comparison')
   plt.xlabel('Epochs')
   plt.ylabel('Loss')
   plt.legend()
   plt.savefig('Train_Validation_Loss_comparison1.png')
   plt.show()
```



### *ii. Accuracy on Test Data:*

We have received 87.4% Accuracy on Test images of 3000. We are using sklearn library to measure the accuracy from model prediction.

```
from sklearn.metrics import accuracy_score
predicted_labels = model_resnet.predict(test_images/255)
predicted_labels=np.argmax(predicted_labels,axis=1)
accuracy = accuracy_score(test_labels, predicted_labels) * 100
accuracy
```

```
87.4
```

### *iii. Confusion Matric on Test Data:*

In the confusion matric we can see that most street case are classified as buildings as there are some similarity between them. And glacier is some classifies as mountain because of some similarity.

```
Confusion matrix
```

| | buildings | forest | glacier | mountain | sea | street |
|---|---|---|---|---|---|---|
| **buildings** | 366 | 4 | 1 | 7 | 8 | 51 |
| **forest** | 1 | 462 | 1 | 5 | 3 | 2 |
| **glacier** | 6 | 1 | 418 | 90 | 33 | 5 |
| **mountain** | 4 | 3 | 40 | 455 | 23 | 0 |
| **sea** | 6 | 1 | 11 | 22 | 463 | 7 |
| **street** | 27 | 8 | 0 | 2 | 6 | 458 |

```
from sklearn.metrics import confusion_matrix
import pandas as pd
print ("Confusion matrix")
#confusion matrix between predicted and actual truths
pd.DataFrame(confusion_matrix(test_labels,predicted_labels),columns=classes_names,index=classes_names)
```

## c. Testing on Own Images Using Resnet Model:

### i. Experimental Results

i. Accuracy score comparison of different models is shown below:

| Classifier/Model | Accuracy |
|---|---|
| Model1 | 80.83 |
| Model2 | 78.2 |
| Resnet50 | 87.4 |

So first Resnet50 model is perform quite good. So we tested the results using it. These are not the images that were part of dataset. They were downloaded from google images and I have used them.

### Testing on Own Images:

So we have trained 3 model, the best model that have maximum accuracy is Resnet50 pretrained on imagenet. So checking the results on our ov
uisng Resnet50.

```
from keras.models import load_model
model1 = load_model('model/model_resnet.h5')# loading model1
```

```
import glob
images_path=glob.glob('Testing_images/*')# loading own testing images.
```

```
img_size=224,224
All_images=[]
for i in range(len(images_path)):
    img=cv2.imread(images_path[i])#reading the images
    img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)#converting the bgr image read by cv2 to rgb
    img=cv2.resize(img,img_size,interpolation=cv2.INTER_CUBIC)# resizing the image to 224*224*3
    All_images.append(img)
All_images=np.array(All_images)
All_images.shape
```

```
(6, 224, 224, 3)
```

```
predicted_labels=np.argmax(model1.predict(All_images/255),axis=1)# predicting using the model and converting it to g
predicted_labels
```

```
array([0, 1, 2, 3, 4, 5], dtype=int64)
```

### ii. Results:

```
# Showing first tested images. and their predicted labels using pyplot.
plt.figure(figsize=(15,10))
for i in range (0,len(images_path)):
    plt.subplot(int(len(images_path)/6),len(images_path),i+1)
    img1=All_images[i]
    plt.imshow(img1)
    plt.title(classes_names[predicted_labels[i]])
    plt.axis('off')
plt.show()
```