# Image Classification Using CNN

Sadaf Iftikhar (317492) MS Data Science-1

***Abstract*—Deep learning and transfer learning techniques have achieved the state of the art results in image classification tasks. Classifying images is one of the major/crucial problems in the field of computer vision and has a large variety of useful applications. In this project I've used three pre-trained models and with the help of transfer learning, images were classified. The dataset I've used for this project consists of nature images. The dataset is divided into 3 categories i.e. training set, validation set, and test set. The models I've used for this project are VGG-19, VGG-16, ResNet-50, and InceptionResNet. I changed several parameters while training, to see which model will give the best results. VGG-19 with optimizer RMSprop outperforms and generates better results than that of the others. Vgg-19 pre-trained model is an updated version of vgg-16 and has more deep layers.**

## I. Introduction

THIS work is regarding the classification of dataset natural-scenes which is a challenging problem in computer vision as such data is a collection of entities i.e objects, and these objects are arranged in an extremely varying layout. Visual representations for such problems have became very popular due to a great level of unpredictability. Lots of recent studies represent scene images in order of fewer collections i.e SIFT/Hog. It's known as BOF (bag of features) depiction. In order to do classification such features gets pooled to an image (invariant in nature) which gets used for discriminant based learning. In new research, FV attained good quality results for the classification of images. Recently on tasks of object-classification, CNN attained high accuracy. Due to this reason many researchers have extended CNN for the problems they encounter. The current multi-layer CNN consists of convolutional layers, activation functions, pooling layers, fully connected layers, and the final classification layer. Convolutional layers are used for extracting the visual features of an image throughout the fully connected layer. Fully-connected layer help in mapping the features into the resultant vector which can be used for the classification of images. In this task, I've used highly accurate models that won the competitions on ImageNet. These trained CNN models achieved high accuracy on image classification tasks. For the sake of experiments, I've used ResNet-50 that has 50 deep layers. ResNet-50 was the winner of the ImageNet challenge in 2015. I've used another pre-trained model that is Inception ResNet model. Inception ResNet(v2) is used in building the Inception based architectures. After continuously getting the model overfitted, I used VGG-19 with optimizer RMSprop. I set the learning rate to 0.001 and batch size to 80. After doing experiments with these 4 models on the scene classification dataset, Vgg-19 was the one which gave best results among the rest.

SEECS, National University of Science and Technology, Islamabad 46000

## II. Methodology

The dataset I used for the respective task is Intel-Image-classification. This dataset consists of 25k images of size 150 x 150. It is distributed under 6 categories/classes (Buildings, Forest, Glacier, Mountain, Sea, Street). In figure 1 overall deep learning network diagram can be seen. In the network diagram we have convolutional layers, max pooling layers and fully connected layers.

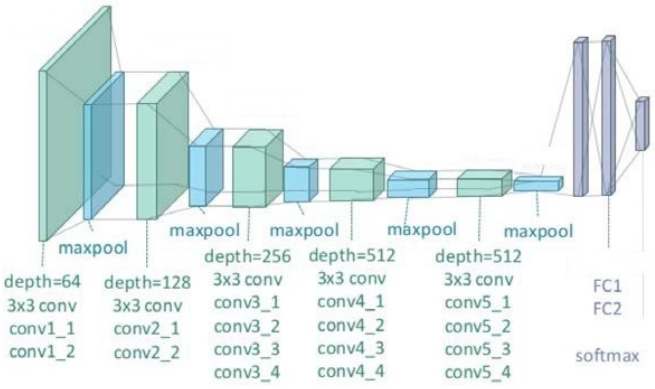


Fig. 1. Achitecure of Network

We have 3 directories which are training, validation, and testing. The training directory has a total of 14034 images. The validation directory has 3000 images, and the testing directory has 7316 images. When we have sparse or low dimensional data, it gets very sensitive to overfitting.

To get rid of overfitting in model I used data augmentation technique. I used generators and at the run time I augmented the data so we can have enough visual representation for the model.

Using data augmentation in generators is very efficient because we can augment our data on the run time. We don't have to save the augmented data in separate folders and load them again repetitively. Instead, we can easily zoom, flip, and sheer images via this technique. Hyper-parameter tuning was done using several models. With Vgg-16 the model gave satisfactory results. But the rest of them except for Vgg-19 did not perform well. Vgg-19 performed very efficiently attaining a high accuracy. Google Collab was used for this task. ResNet50 didn't perform well on the dataset.

## III. Result

At first, I used VGG-16 with the basic configuration. The optimizer was set to Adam with the default learning rate. I

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              8389632
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 1542
=================================================================
Total params: 28,677,958
Trainable params: 8,653,574
Non-trainable params: 20,024,384
_________________________________________________________________
```

Fig. 2. Summary of Hyperparameter tuning

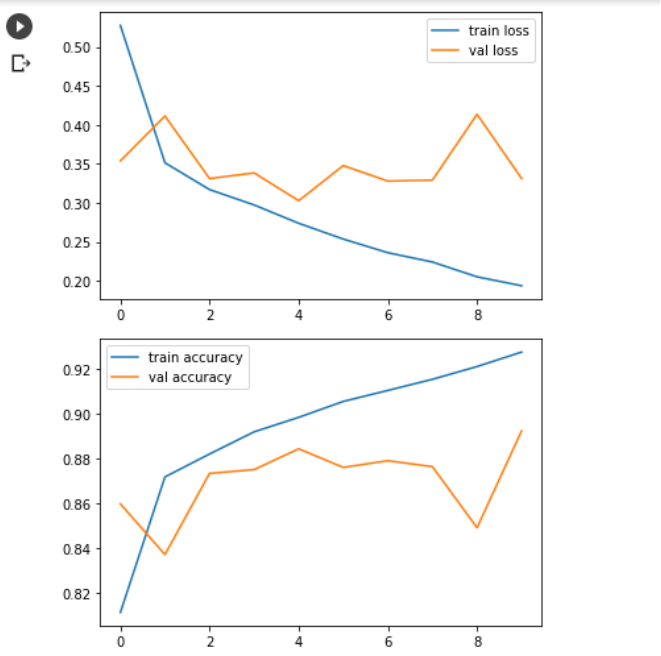


Fig. 3. Accuracy and Loss graphs of Vgg-16 with default settings

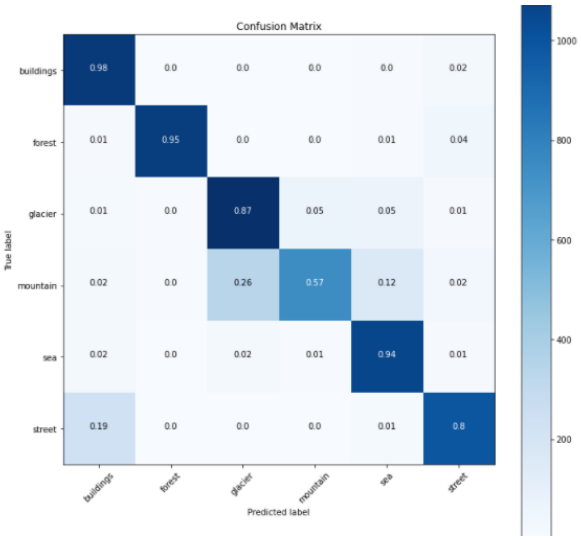


Fig. 4. Vgg-16 confusion matrix with default settings

trained my model with 10 epochs. In figure 3 we can see the accuracy and loss graphs of the mentioned model. The model overfitted and couldn't generate better results. When the number of epochs were increased, model started learning on the data but again overfitting occurred. Overfitting occurs when the model is good at training stage but can't perform efficiently for the testing data. 94% accuracy, 88% validation accuracy, 0.2 loss and 0.34 validation loss were attained. Overfitting occurred due to default configuration. In figure 4 we can see the confusion matrix of the Vgg-16 model with default setting.

In order to avoid overfitting, parameters of the model were tuned. RMSprop was used as an optimizer in which the batch size was set to value 64. Data was fed to model after passing it through generators data augmentation. 86% accuracy, 0.79% validation accuracy, 0.2 loss and 12 validation loss were attained. With mentioned parameters, model again went through overfitting. Figure 5 shows the confusion matrix of the described tuned model. Whereas in figure 6 we can see the accuracy graph of the model.

In figure 7  figure 8 we can see the accuracy and loss graphs of ResNet50 model results. The validation accuracy is very low, and training accuracy is very high which is not optimal. From the loss graph we can see that the training loss is near to 0 whereas validation loss is more than 50. This means that the model did not start to learn from the training data thus it failed on validation data.

The model ResNet50 is completely overfitted and in figure 10 we can see the confusion matrix of the ResNet50 model. ResNet50 is considered as a well-known trained model but on the given data-set it did not perform well.

Data-set was applied to InceptionResNet version 2 model. InceptionResNet version 2 is the combination of both Inception model and ResNet model which receives very high results at classification tasks didn't perform well when it was applied to the dataset

After several experiments, I chose Vgg-19 model in order to resolve the issue. I used image size of 150 x 150. Via transfer learning, and Vgg-19 model, I added 2 more dense layers with neurons of 1024 and 256. By setting the optimizer to RMSprop and learning rate to 0.001 model was compiled. Batch size was set to value 80. In figure 11 and figure 12,

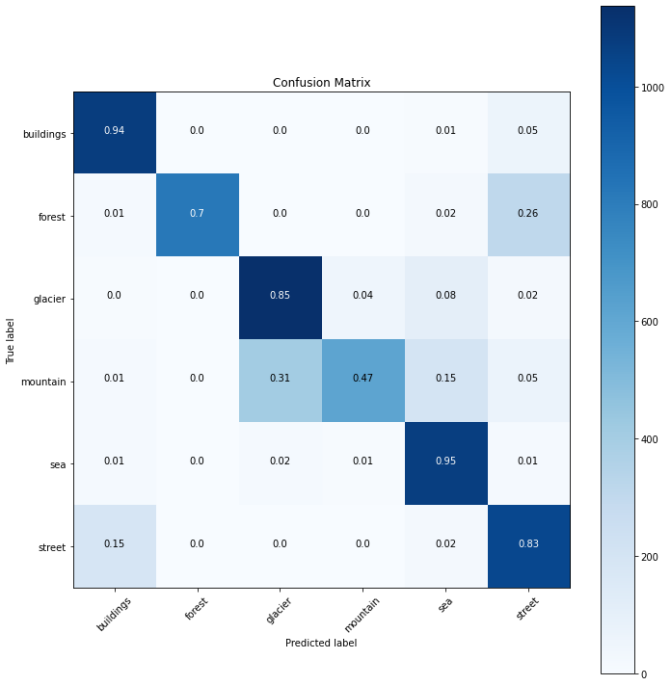


Fig. 5. Vgg-16 confusion matrix tuned

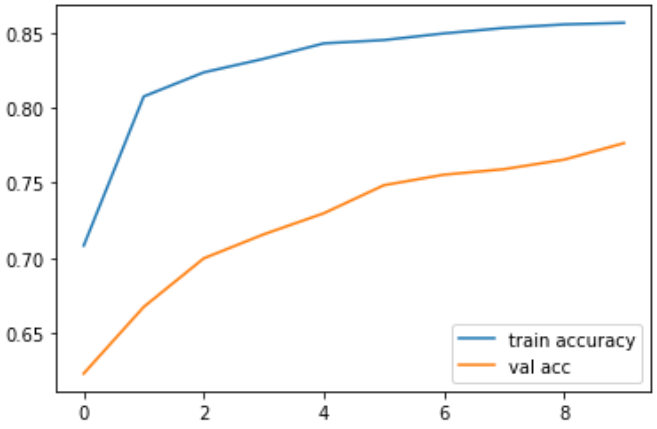


Fig. 6. Vgg-16 accuracy graph with RMSprop

accuracy and loss of the Inception ResNet model have been shown.

This model performed well, and issue of overfitting got resolved as well. The prediction using this model is also accurate which can be seen in confusion matrix shown in figure 16. It can be seen in confusion matrix figure that vgg-19 has accurate prediction results. The diagonal elements have high values which shows that it predicted about 90% accurate images. Vgg-19 accurately categorized images than that of the other models.

Figure 17 shows correctly predicted image while doing its training.

## IV. Discussion Conclusion

In this project, pre-trained Vgg-19 model was used. Data was fed to training model with the help of transfer learning

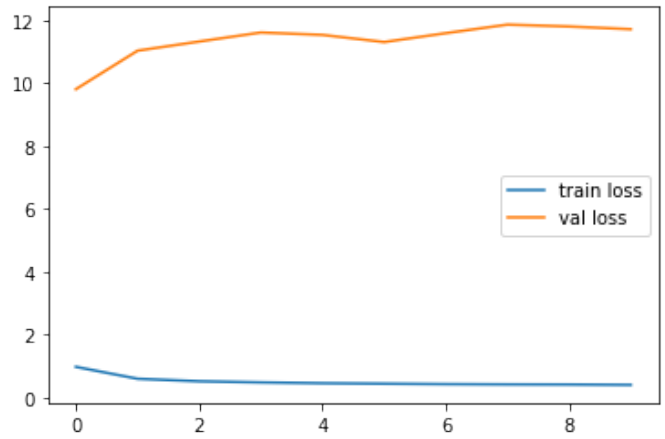


Fig. 7. Vgg-16 Loss graph with RMSprop

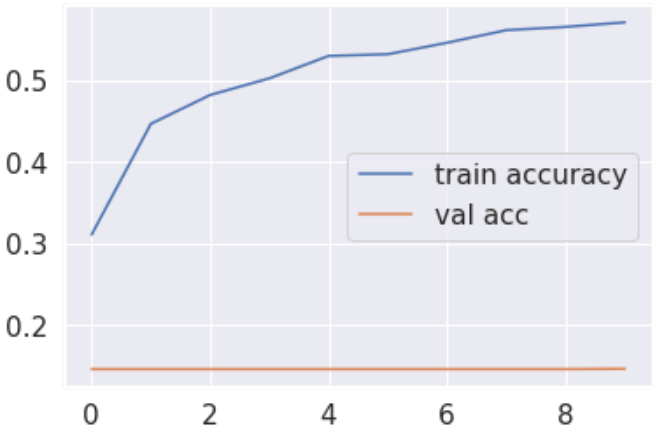


Fig. 8. Accuracy graph of ResNet50

technique. Validation accuracy of 88% and Validation loss of 0.4% were achieved. Experiments with different models were done in order to achieve better results. Hyperparameters of the models were also tuned. Vgg-19 gave better results than rest of the models. The optimizer RMSprop showed a better learning capability on the respective dataset. We can improve our existing project by using auto-keras which will return better results for the given problem. 2 additional dense layers were added to the model so it can learn the visual features from the data more efficiently. Datasets having less dimensions images are very sensitive to overfitting. Thus, if we have a good dataset with detailed visual representations, we can make state-of-the-art models and improve the learning capabilities of the existing models. The augmentations techniques also vary according to the datasets. For example, if we have x-ray images datasets, we cannot do augmentations because in x-ray images we have minute details. But in images like natural scenes, we can do extensive augmentation to train the model more accurately.

## V. Github Repository Link

https://github.com/sadafiftikhar/Assignment-3


Fig. 9. Loss graph of ResNet50

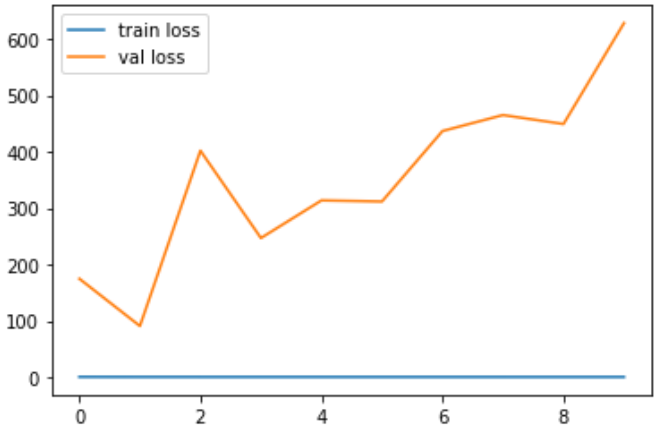


Fig. 12. Inception ResNet version 2 loss graph

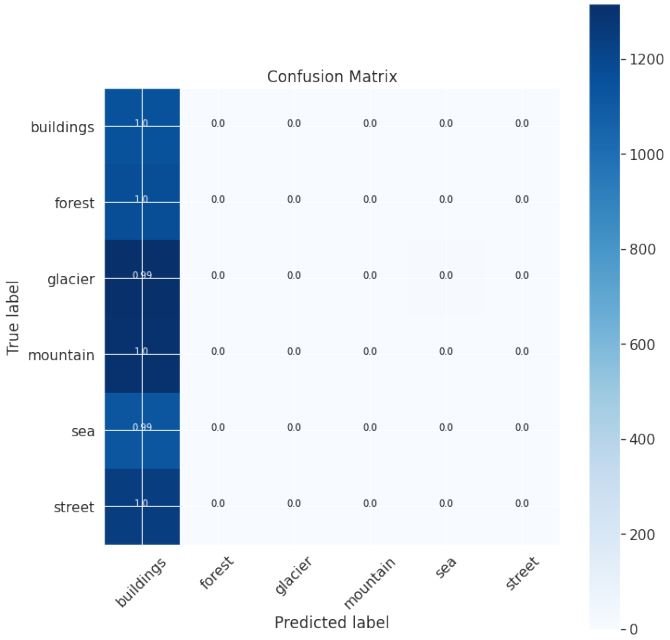


Fig. 10. Confusion Matrix Resnet

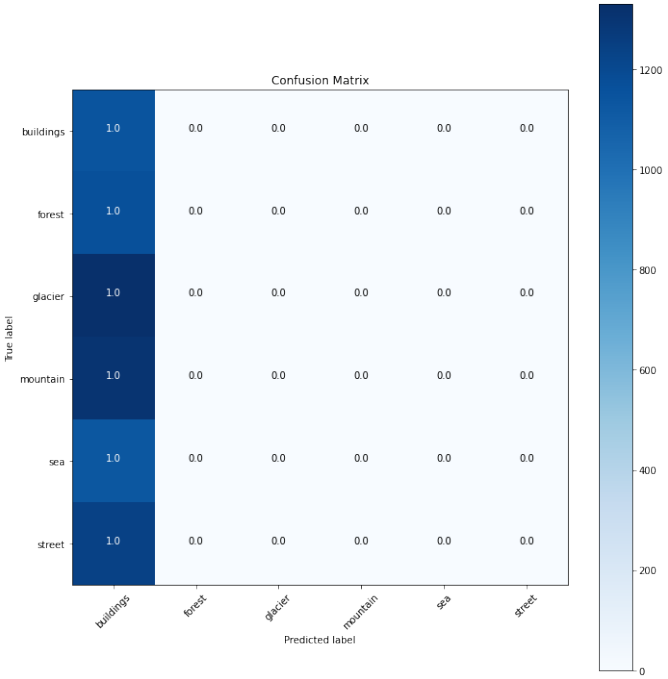


Fig. 13. Inception RestNet Confusion Matrix

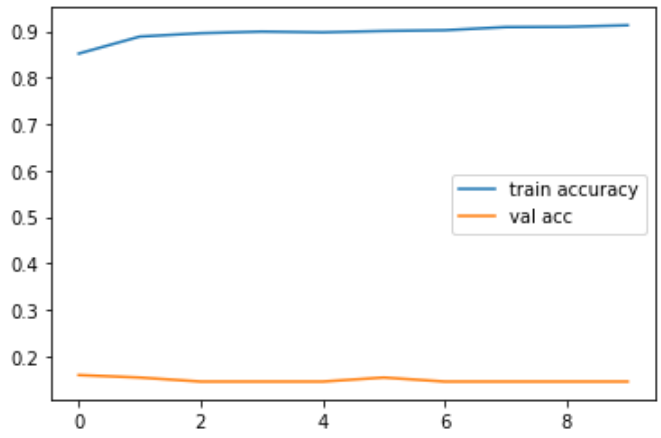


Fig. 11. Inception ResNet version 2 accuracy graph

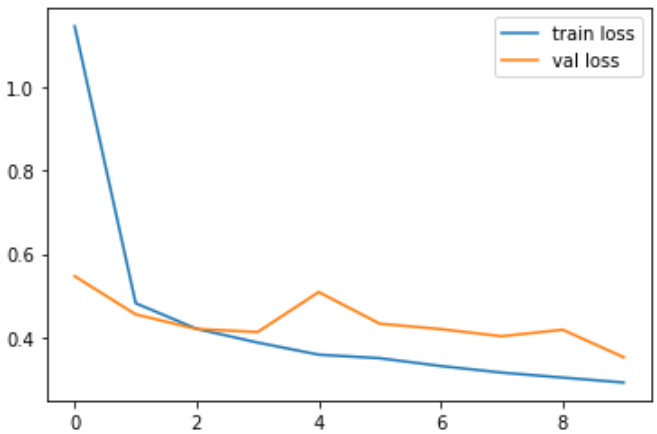


Fig. 14. Vgg-19 model with RMSprop loss graph

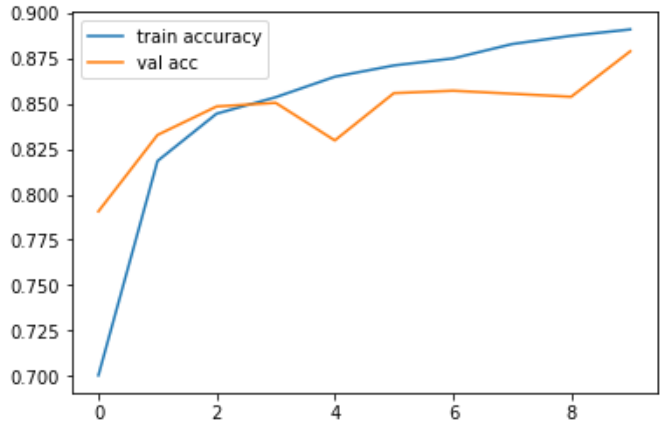


Fig. 15. Vgg-19 model with RMSprop Accuracy graph

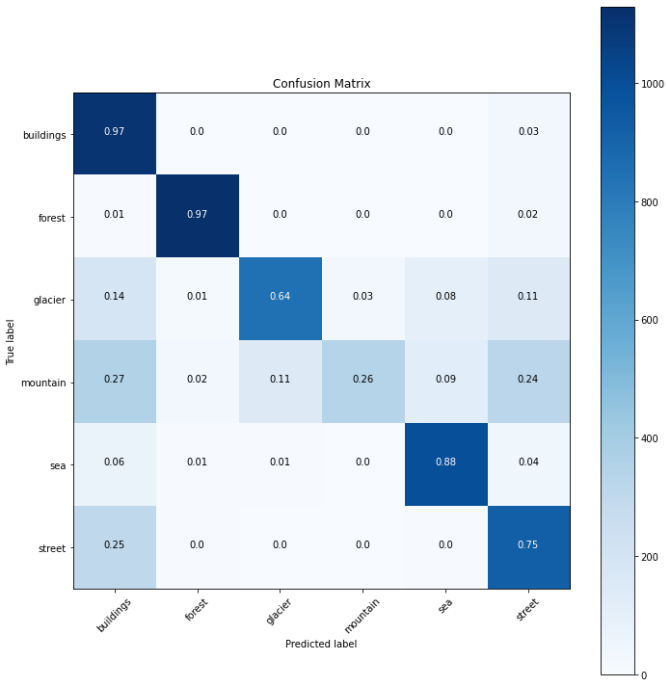


Fig. 16. Confusion matrix of Vgg-19 model


Fig. 17. Correctly classified Image