Boyuan Du, Jiaoyue Sun, Saurav Mainali
Machine Learning II, Fall 2020
Group 6 – Final Report

Scene Classification

Introduction

Computer Vision, a field in Computer Science, which aims to replicate the complex human vision system can today be found surpassing the humans. Thanks to the incessant advancement in Deep Learning and Neural Networks. This field has created wide range of applications, one of which is – classification of Scenes around the world. Its application could range from organization of photos in Smartphones; assist in growth of country's economy through Tourism Planning and so on.

We, members of Group 6 worked on the problem of classifying Scenes around the globe into one of six possible classes using Deep Neural Networks. We trained CNN networks from scratch and, also experimented with multiple pre-trained networks. The performance of all these models were ultimately compared and analyzed.

The following section of this report consists of Dataset Description, Description of Deep Learning Networks Used, Experimental Setup, Results, Summary and Conclusion.

Dataset Description

Dataset for this project is obtained from Kaggle. It can be downloaded through the link - https://www.kaggle.com/puneet6060/intel-image-classification/download

There are total of 25,000 images captured by Jan Bottinger. Out of them only 17,000 were labelled. The six possible categories are - Buildings, Forest, Glacier, Mountain, Sea or Street. The figures below show the proportion of each category in the train dataset and test dataset.

Figure.1 Train dataset

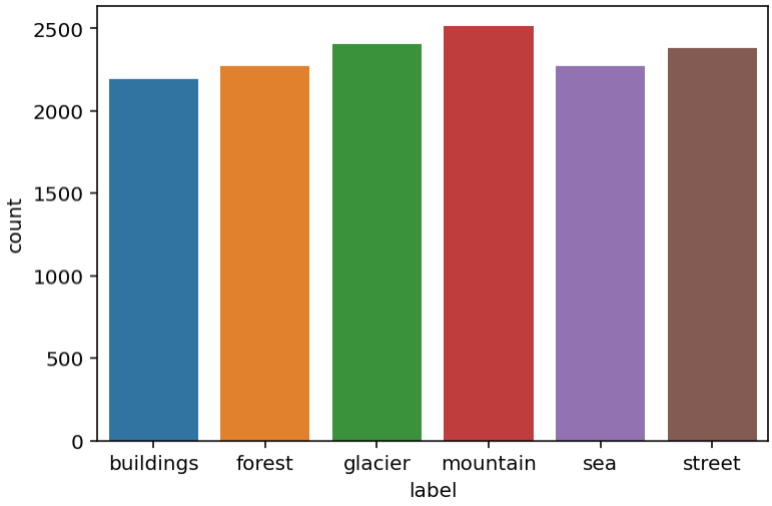
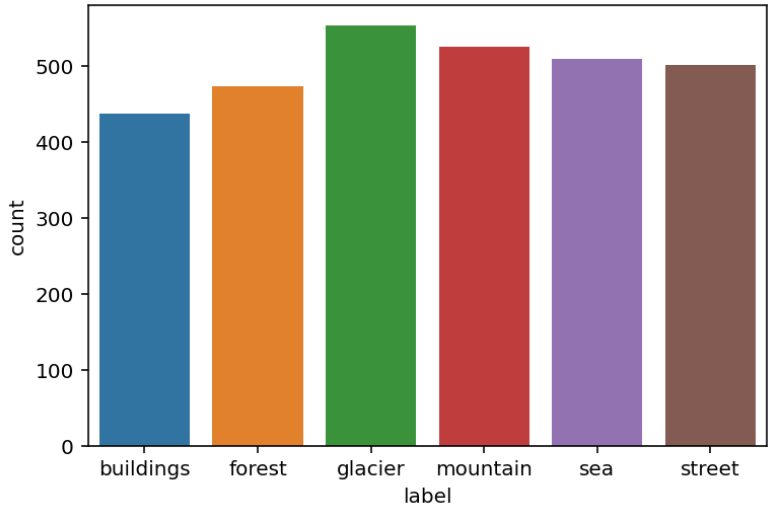


Figure.2 Test dataset



As we can see from the bar chart, the dataset is balanced.

The following section describes various Deep Learning Networks we trained using the dataset.

Description of Deep Learning Network Used

Self-trained CNN Network

Firstly, we created a custom CNN Network using the Keras Framework. The structure of the model can be seen below:

Figure.3 Self-trained CNN Network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 200)     5600
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 146, 146, 180)     324180
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 29, 29, 180)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 27, 180)       291780
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 140)       226940
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 100)       126100
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 21, 21, 50)        45050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 50)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 180)               144180
_________________________________________________________________
dense_1 (Dense)              (None, 100)               18100
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout (Dropout)            (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 1,187,286
Trainable params: 1,187,286
```

This Self-Trained model has six Conv2D layers, all of which had kernel size of (3, 3) and stride of 1 were connected end by end. Each had Relu activation unit, and second Conv2D layer and fifth Conv2D layer are followed by Max Pooling with kernel of (2, 2). The output was then flattened and passed into three Dense layers, each with 180 filters, 100 filters and 50 filters. Finally, with the dropout rate of 0.5, they were all Normalized again and passed into SoftMax Unit having filters equal to number of class labels.

Learning Rate was set to 0.001 and with the batch size of 512 the model was trained for 40 epochs with Callback monitoring loss of validation set.

Pre-trained CNN Network

We made use of five different pretrained CNN networks namely -VGG16, VGG19, ResNet152, InceptionV3, InceptionResV2, which were trained using the ImageNet dataset. These models were available inside keras.applications library.
Performance for each of these models were compared with each other and with that of custom trained CNN model too. A code snippet of an InceptionV3 model can be seen below:

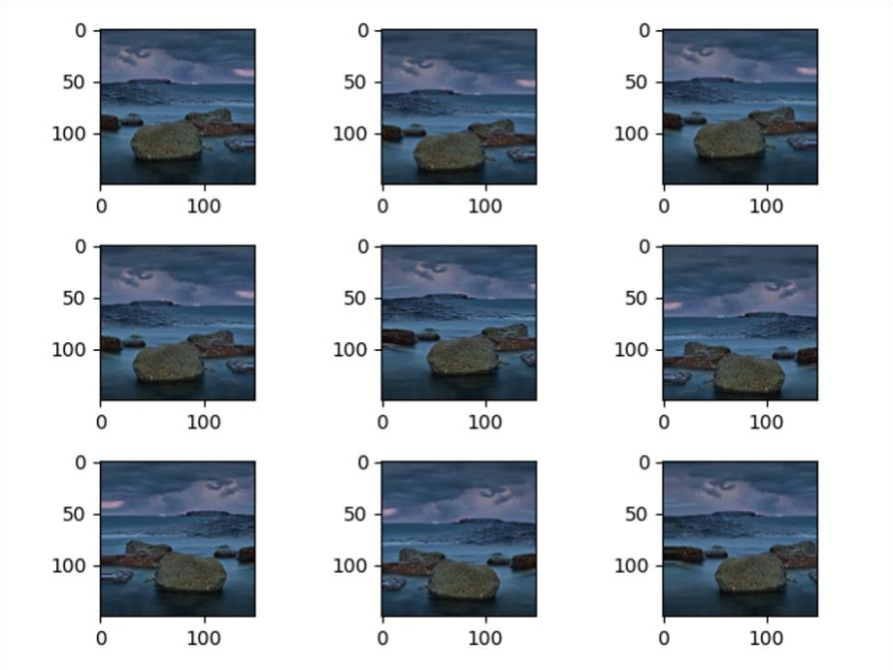
```
res = applications.InceptionV3(input_shape=(150,150,3), weights='imagenet', include_top=False)
res.trainable = False
print('inception pre trained model is loaded ....')

model = Sequential([res,
                    Flatten(),
                    Dense(400,activation='tanh'),
                    Dropout(0.5),
                    BatchNormalization(),
                    Dense(6,activation='softmax')
                    ])
```

The weights of pre-trained networks were fixed. The output from pretrained networks were flattened and passed to Dense layer with 400 filters and Tanh activation unit. Then with dropout rate of 0.5 the outputs were normalized and passed into a final Dense layer with 6 filters and SoftMax as activation unit.

Data augmentation

We applied data augmentation to the five selected pre-trained networks mentioned above. In this case, we use the 0.1 as the percentage of the height and 0.1 as the percentage of the width to shift the image. Both horizontal and vertical positive and negative shifts probably make sense for the chosen photograph, but in some cases, the replicated pixels at the edge of the image may not make sense to a model. The flip augmentation is specified by a boolean horizontal_flip or vertical_flip argument to the ImageDataGenerator class constructor. Only horizontal flips are considered in this model. Random rotations via the rotation_range argument is considered, with rotations to the image between 0 and 180 degrees. Other augmentations like zoom and shear are not used in this project. We will compare the results of the pretrained networks before and after data augmentation to see if this would improve the model.

Figure.4 Data augmentation example



The following section describes how we finalized on the above discussed Networks and its corresponding parameters.

Experimental Setup

We used 70% of the labelled images for Training, 15% for Validation and remaining 15% for Testing. Images from each class label were evenly distributed while dividing the data.

Training parameters were decided by using the trial and error approach. We experimented with various layers of Networks, sizes of filters and kernels, learning rate, minibatch size, dropout rates, epochs by monitoring models performance. Also, to prevent overfitting we implemented the concept of Early Callback. Besides we were mindful on selecting parameters that closed gap between training and validation accuracy. Parameters and networks discussed in the above section gave us the best result.

The following section describes performance of each model described in previous Section.

Results

Model accuracy

To measure the performance of model Accuracy, Cohen Kappa Score and Macro F1 Score were used. Since our dataset is balanced, Accuracy Score was considered as the prime performance indicator. It can give us idea regarding how many predictions were correct among all the predictions made. Similarly, Macro F1 Score can hint us on performance of model by giving equal importance to each of the class label. Result obtained is shown below:

Model	Accuracy	Cohen Kappa Score	Macro F1 Score
VGG16	89.70 %	0.87	0.89
VGG19	88.49 %	0.86	0.88
ResNet152V2	90.63 %	0.88	0.90
InceptionV3	87.23 %	0.84	0.87
InceptionResV2	88.13 %	0.85	0.88
Self-Trained Model	86.09 %	0.77	0.83

ResNet152V2 performed the best with Accuracy of 90.63%, Cohen Kappa Score of 0.88 and Macro F1 Score of 0.90. Performance of remaining pre-trained models were close to the best model. The Self-trained model performance was worst with accuracy of 86.09%.

The result of the networks with data augmentation is shown below:

Model	Accuracy	Cohen Kappa Score	Macro F1 Score
VGG16	89.43 %	0.87	0.90
VGG19	88.73 %	0.86	0.89
ResNet152V2	87.27 %	0.84	0.88
InceptionV3	84.20 %	0.81	0.85
InceptionResV2	83.99 %	0.81	0.85

We could see that the models do not appear to perform better after data augmentation. Balanced data may not be improved as much as unbalanced data by using data augmentation. So, we would consider using models without data augmentation to do our further analysis.

Predictions

In order to find out the best model for our project, we decided to compare the prediction ability about the top3 models, vgg16, vgg19, ResNet and our self-trained model. The prediction results are shown below:

Figure.5 Self-trained prediction

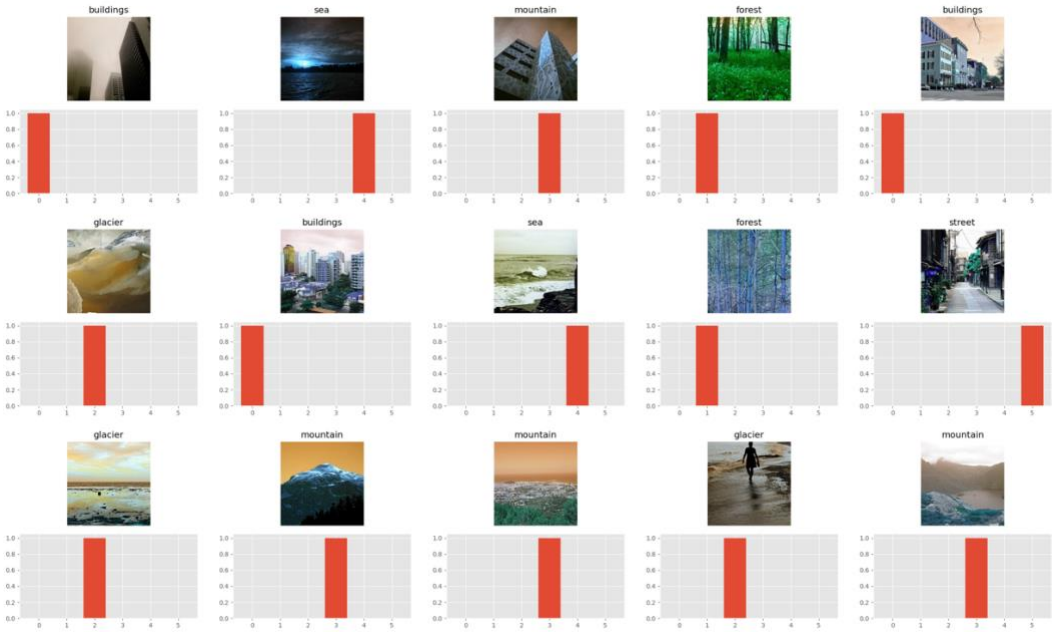


Figure.6 ResNet prediction

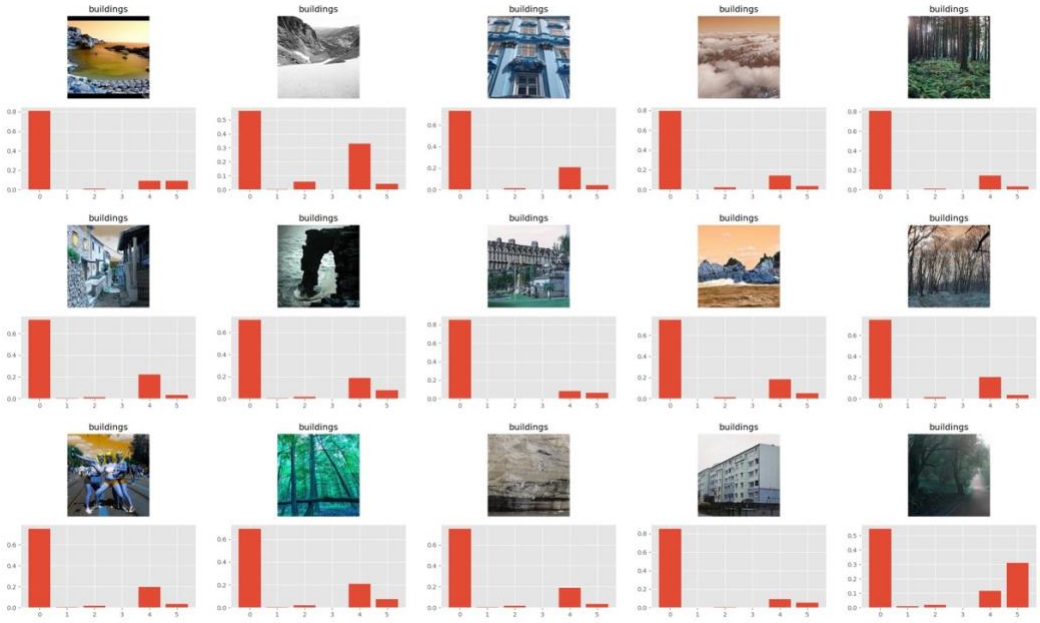
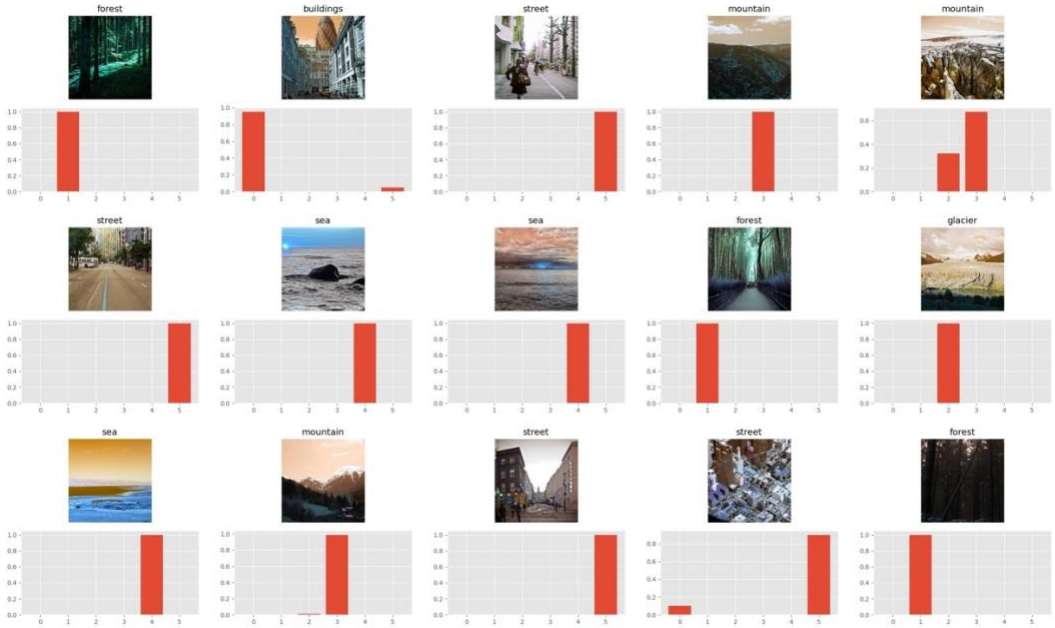


Figure.7 Vgg16 prediction

Figure.8 Vgg19 prediction



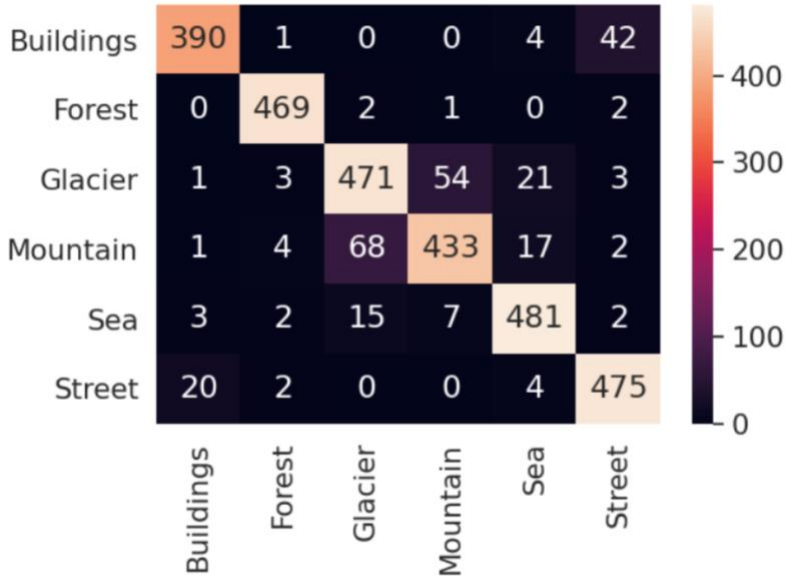
The result in our self-trained model is good, but it seems to mix tall buildings and mountains to some extent, and the same situation happened in class sea and glacier. It may because these pictures have some similar features. Then in ResNet results, our previous best model did not show a good prediction as we were expected, as in the examples we show here, it mispredict lot of them. That's not what we want

from it. In the end, the vgg16 and vgg19 show very good prediction. And given Vgg16 has better accuracy, we decided to choose vgg16 as our best model.

Class Confusion

Below is Confusion Matrix of the best performing model, which gives deeper insight on predictions made and the training images:

Figure.9 Confusion Matrix



The matrix indicates there is confusion between ‘Glacier & Mountain’ and ‘Buildings & Street’. So, on investigating the dataset further, we found this problem was due to the nature of the training images.

Images which were labelled as Glacier had Mountains in the background making the model prone to misclassification. Similarly, error between Buildings and Street was also due to the similar reason. Images that were classified as Street had Buildings in it and vice versa. The images can be seen below:

Summary and Conclusions

Thus, the pretrained models performed much better than the self-trained CNN model. This could be because pretrained models were trained using huge dataset with very large number of trainable parameters. This project taught us that by using deep networks we can create extremely powerful models, given we have large number of quality training data with proper labels.

To make further improvement on this project we would recommend approaching this problem as Multi-Labelled Classification problem by making necessary amendments in the class labels.

References

Scene Classification

- MathWorks. “Train Deep Learning Network to Classify New Images”. https://www.mathworks.com/help/deeplearning/examples/train-deep-learning-network-to-classify-new-images.html

- MathWorks. “Scene Classification using Deep Learning”. https://medium.com/mathworks/scene-classification-using-deep-learning-853c64318f6b

Codes

- https://github.com/amir-jafari/Deep-Learning

- https://github.com/krishnaik06/Transfer-Learning
- https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
- https://www.jianshu.com/p/502b4c637c0e