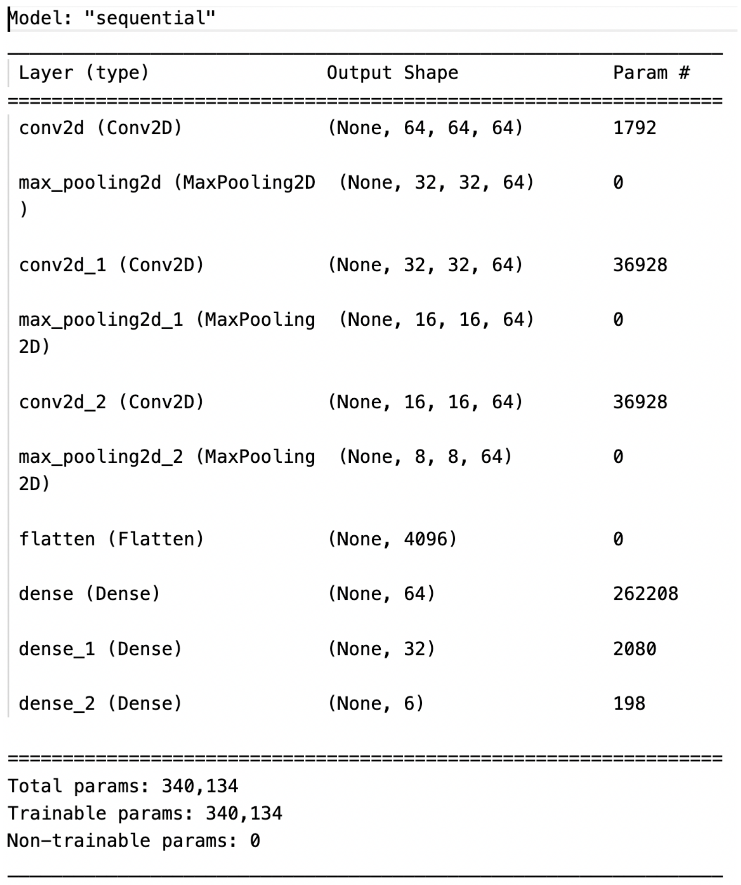
Multiclass image classification using CNN

Group: Group 6

Name: Sunse kwon

Student number: 2076460

Model 1: the best model with data augmentation



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 32, 32, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 64)                262208

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 6)                 198

=================================================================
Total params: 340,134
Trainable params: 340,134
Non-trainable params: 0
_________________________________________________________________
```

1. Input to classifier

Dataset used is from Kaggle Intel Image Classification containing around 25k images of size 150x150 distributed under six categories.

{'buildings' -> 0,'forest' -> 1,'glacier' -> 2,'mountain' -> 3,'sea' -> 4,'street' -> 5}

Train data spitted 80% of train data and 20% of validation data. The total number of samples in our train, validation and test set is separated as 11230 rows in the training set, 2804 rows in the validation set, and 3000 rows in the test set. In order to feed into the CNN model, the Shape of X needs to be (batch_size, height, width, depth). The default shape of each sample in X consists of a 4-dimensional tensor (64,64,64,3) from the image data generator. The corresponding Y shape consists of (64,6), which is 6, meaning that we have six classes (building, forest, glacier, mountain, sea, and street).

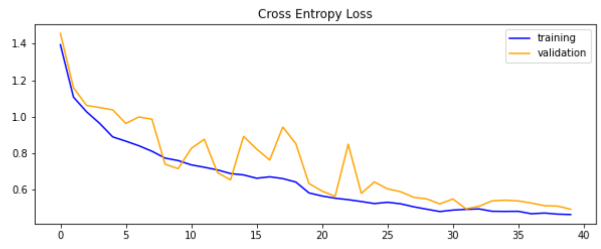
2. Preprocessing

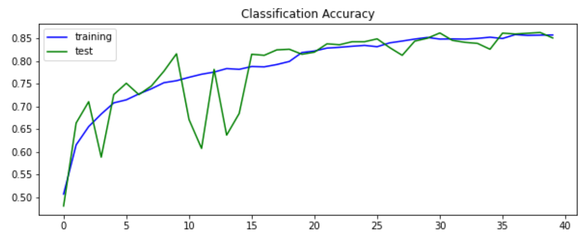
The image data generator has an argument called rescale. We multiplied data by the value provided before fed into the model. As our original model consists of RGB coefficient in the 0 to 255, these values are too high to process by our model. Therefore, we normalize values between 0 and 1 by scaling the 1/255 factor.

In the image data generator, we have several arguments to perform data augmentation. In our best model, rotation_range, zoom_range, width_shift_range, height_shift_range, and fill_mode were selected as we experimented with several combinations of other data augmentation techniques. As we empirically investigated the source image, we found that we thought rotation, width_shift, height_shift, and zoom range need to be minor, around 20,0.2,0.2,0.2 respectively, to capture lines in a diagonal. For fill_mode, we tried several other modes, but the default 'nearest' showed the best performance. Other augmentation methods, such as horizontal_flip and vertical_flip, were ignored as they made the performance of the model drop.

3. Model Training

First, a sequential model from the Keras package was selected to build CNN architecture, as spatial dependency is considered essential to train image data. For the forward propagation process, we fed a 3-dimensional tensor with the shape of (64,64,3) into the input layer. After the input layer, Convolutional layers (Conv2d) are only connected to spatially closed units in the next layer. Then the output of the convolutional layer is fed into the batchnormalization layer. It normalizes the output of previous layers. After that, the maxpooling layer located, which is the maximum value will be taken from each partition of output of previous layers. Maxpooling layers help to capture more abstract objects when data go into deeper layers. We have three repeated structures of the sequence of conv2d, batchnormalization, maxpooling layers. After that, we flatten all the outputs using flatten layers to feed a fully connected (dense) layer. Then output layers with six units as our output have six classes. Optimizer Adam was selected instead of RMSProp to save computation costs. The loss function chose as categorical cross entropy as our nature of the task is multiclass classification. Once we find the cost function derivative, our model will continuously look for the weight and bias of each node of the previous layers and update accordingly. As backpropagation process goes during the iteration set by epochs, our Adam will find local minima, which minimizes our cost function.





As can be seen clearly, training and validation loss gaps are minimized during training and do not show an overfitting tendency. From the right figure, accuracy on both training and test sets increased and converged around 0.85, meaning we can generalize our model into unseen data.

4. Hyperparameter tuning

For hyper parameter choice, we selected following

first, epochs = 40. As our baseline epochs were configured as 20, we chose Epochs as 40. The reason is that by plotting loss over the epochs, we found that increasing trend until 30 epochs, and in when 35th iteration, our training, and validation error starts to converge. Second, batch size =32. From baseline, the batch size was set as 32, but in our dataset, the original batch size was 64. after comparison of the two batch sizes, we identified model with a batch size of 32 showed an improvement in performance than the counterpart as the model with a batch size of 64. therefore, we stick to 32. The third hyperparameter is reduce learning rate on plateau. It basically helps to overcome the risk of our optimizer being stuck at a plateau. With patience 5, we can identify the meaning of unchaining validation loss, whether it is in the plateau or in local minima and reduce being stuck in the plateau. Fourth, learning rate = 0.001. as conventionally recommended, it was chosen for optimizer Adam. Adam will automatically find different learning rates over the training process. Fifth, padding = same. When applying the kernel, it remained a baseline to preserve all the information, including edges in the input layer. Sixth, the size of the unit in conv2d layers = 64,64,32 respectively. as our baseline show overfitting, we tried to reduce the number of units in the last conv2d layers.

Also, we removed one fully connected layer, which is a size of 32 units, to reduce the complexity of the model. Seventh, kernel size = (3,3). We remain this as the baseline. After applying the (5,5) kernel, the performance drop. Therefore, we stick to (3,3) as a baseline.

Regarding activation function, ReLu activation functions are selected across hidden layers rather than tanh activation as computation cost is cheaper. Also, softmax was chosen for output layers due to softmax predicts a probability distribution of six labels, and the output of softmax will be fed into our cost function, which is categorical cross entropy, as our task is multi-label classification.

Also, two optimization methods were used. First, kernel_initializer= he_uniforms. It helps to solve the vanishing and exploding weight problem partially. By choosing the appropriate scaling of weight, we can prevent the weights from exploding or vanishing quickly. meaning it prevented biased weight initialization. Second, the batch normalization layer helps to speed up training speed by subtracting mean and normalizing variance by applying a square root. When the output of conv2d results in different scales, if we do not apply normalization, the ratio of features is different. So, the cost function will not be symmetric. Gradient descent will be more oscillating and takes more time to converge to local minima. Therefore, batch normalization will help the cost function find local minima easier and faster.

Lastly, two regularizations are used to prevent overfitting. First, an l2 regularizer is used. When we have a high variance problem, we add an l2 regularizer in the second and third conv2d layers to prevent overfitting. Rate= 0.0005 chosen as we do not want majority of weights to be zeros. Second, the dropout layer is located in front of the output layer. As dropout less perform than l2 regularizer in smaller network (Phaisangittisagul, 2016), we set rate = 0.5. It masks 50% of neurons from the previous layer to the next layer.

5. Performance result with graph

To see the performance of the model for each class, we analyzed the confusion matrix and roc curve below. Each diagonal in the confusion matrix for the best model has a higher number than our baseline. We can see all the values of diagonal in the best model increased in comparison to the baseline.

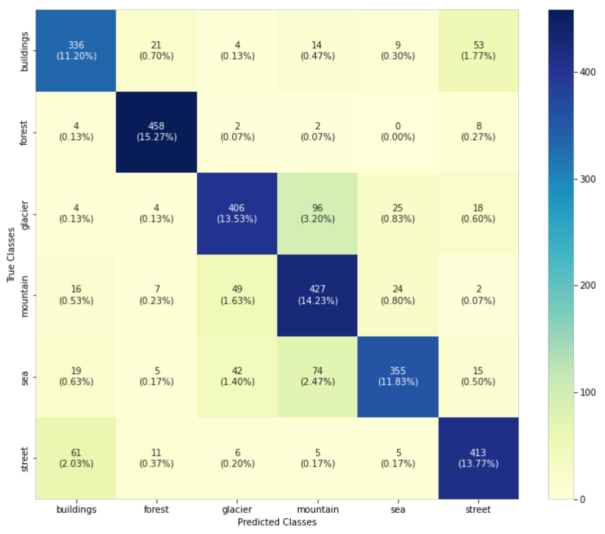


Figure 1 confusion matrix for baseline

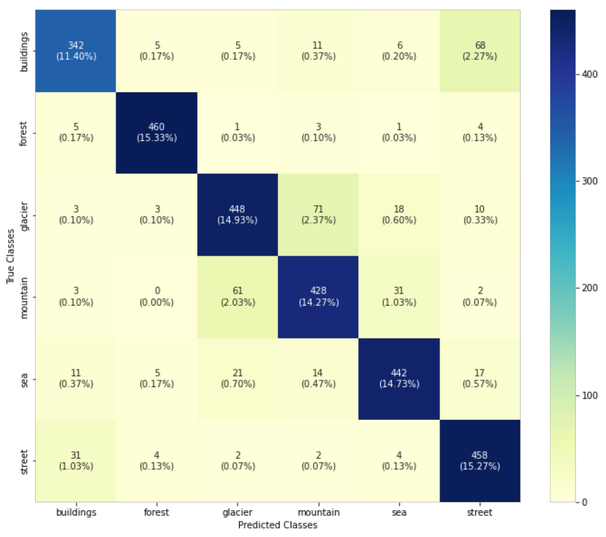


Figure 2 confusion matrix for the best model

Also, we can see that our best model shows that each class has at least a 0.97 area under the curve scores, which shows an increase in performance in each class in comparison of baseline model.

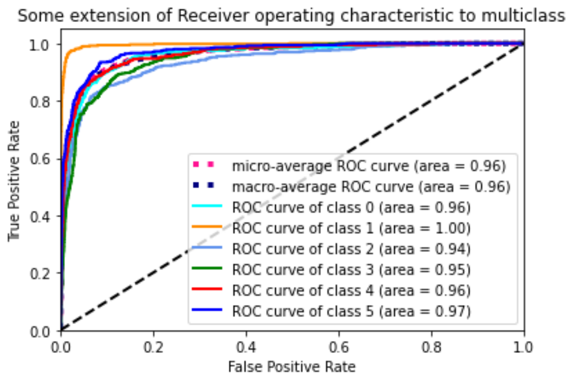


Figure 3 roc curve for baseline.

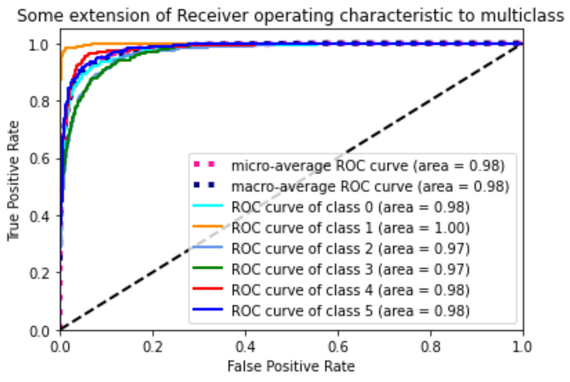


Figure 4 roc curve for the best model

Finally, we can compare the scores baseline with the best model. Accuracy on the test set is 0.85, which is a 5% increase in performance from the baseline which scores of 0.80

Compared to the baseline, the forest is most correctly classified, with an F1 score of 0.97, sea (0.87), and street (0.86), respectively. However, building (0.82), glacier (0.82), and mountain (0.81) showed only a slight increase in performance from the baseline.

Regarding specificity, it showed how our model correctly predicted the actual negative class as negative. Most of them are high, meaning they are correctly specified. Moreover, it can also be seen in the baseline as well. It seems like class imbalance which is one class is much more than the other class. However, it is expected as the other 5 class has not belonged to 1 class.

For sensitivity (recall), we can see how the true class can be correctly predicted. We can see all our classes in the best model show increased value from baseline.

	Sensitivity	Specificity	F1 score
Building	0.77	0.96	0.78
Forest	0.96	0.97	0.92
glacier	0.68	0.96	0.74
mountain	0.85	0.90	0.75
sea	0.75	0.97	0.80
street	0.76	0.97	0.81

Figure 5 performance measure for baseline

	Sensitivity	Specificity	F1 score
Building	0.78	0.97	0.82
Forest	0.97	0.99	0.97
glacier	0.81	0.96	0.82
mountain	0.81	0.95	0.81
sea	0.86	0.97	0.87
street	0.91	0.95	0.86

Figure 6 performance measure of the best model

Model 2: transfer learning

In addition to building the CNN model, we used transfer learning for our problem.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 64, 64, 3)]       0

 sequential (Sequential)     (None, 64, 64, 3)         0

 tf.__operators__.getitem (S  (None, 64, 64, 3)        0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 64, 64, 3)        0

 vgg16 (Functional)          (None, 2, 2, 512)         14714688

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 64)                131136

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 6)                 390

=================================================================
Total params: 14,850,374
Trainable params: 135,686
Non-trainable params: 14,714,688
```

1. Input to the classifier

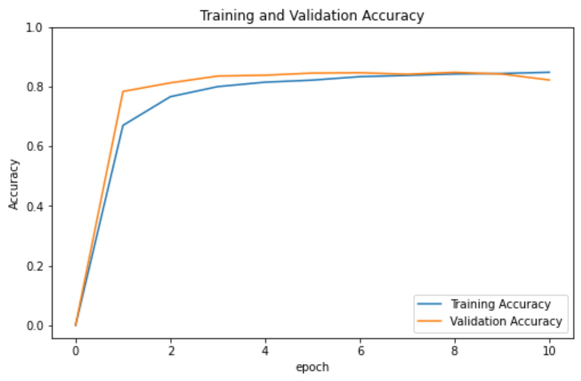
Input configuration is the same as the baseline except rescale. Also, we turned shuffle = False on the validation and test set. Input is also a 4-dimensional tensor with a shape of (64,64,64,3)

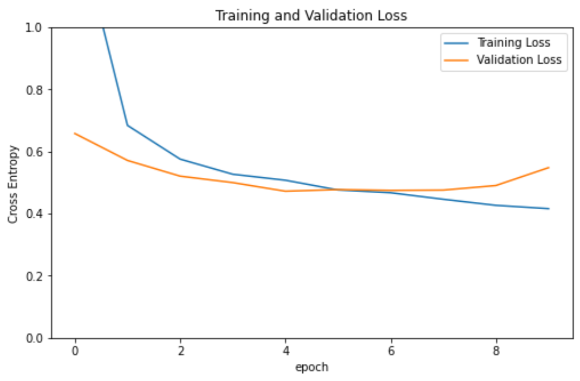
2. Preprocessing

Instead of specifying data augmentation in the image data generator, we built a function that was first based on a sequential model. We add the first layer that performs random horizontal flip and add second layer that performs random rotation with 0.2 rates.

3. Model Training

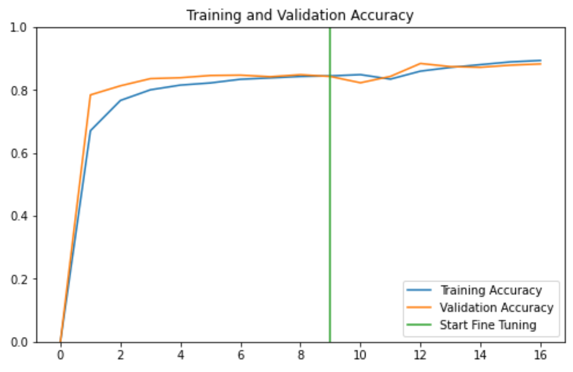
We used a pre-trained vgg16 model from the Keras package for our training. Imported convolutional layers only by setting include_top= False. Then freeze this convolutional part that is not trainable. After the input layer, the input went through the data augmentation function specified in the previous section. Then it went through the convolutional layer of vgg16 models. After that, we added flattened layers and two fully connected layers with 64 units. Lastly output layer with six units. Optimizer Adam was selected with a categorical cross-entropy loss function. As seen below graph, after epoch 8, it starts to overfit. Our accuracy starts to diverge epochs around 8.5. at this point, the accuracy is 0.84.

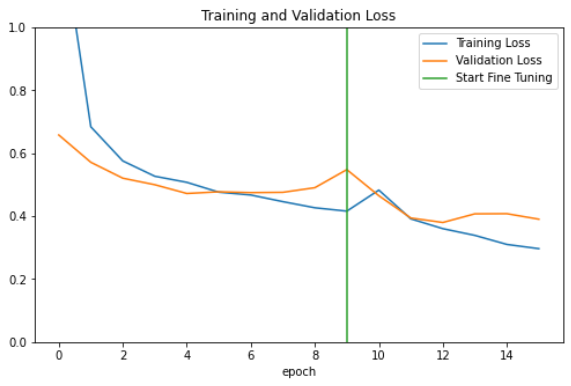




4. Hyperparameter tuning

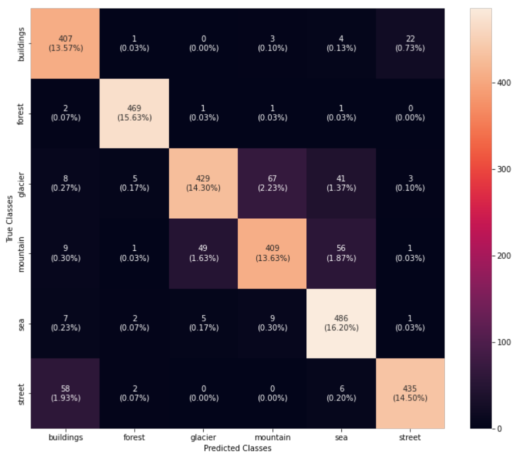
for hyper parameter choice,_Epochs = 15 used, which is initial epoch =10 plus additional epoch for fine-tune =5. And Learning rate = 0.0001 selected. as our Adam optimizer will automatically adapt the learning rate at different derivatives. We set it as a conventionally recommended value. regarding activation function, we used the ReLu activation function for two fully connected layers. for the output layer, we used the SoftMax function. In terms of optimization and regularization techniques, we added fine-tune layers and tried to fine-tune our convolutional layer to overcome overfitting problem. As can be seen below, after 8.5 epochs, the fine tune started, and it increased accuracy slightly and also lowered the loss in both training and validation sets.

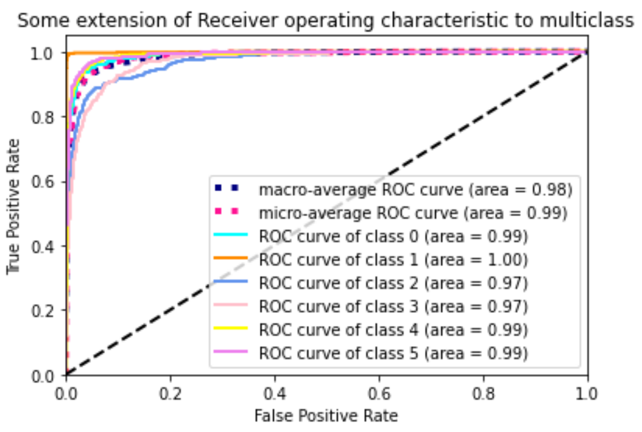




5. Performance result with graph

As can be seen confusion matrix, most classes are correctly classified. Nevertheless, it is not clear how much different our pre-trained model is from our best model. In the ROC curve, we can see that AUC scores are generally slightly higher.





The overall accuracy score is now 0.88, which is a 3% increase from our best model and a 8% increase from our baseline model.

Except for glacier (0.83) and mountain (0.81), the f1 score of All other class scores is over 0.88. therefore, a model with a pre-trained model outperforms our best model.

6. Reference

Dr. Gorkem Saygili, Deep Learning (Fall) (880008-M-6), Tilburg University

Intel Image Classification Dataset: https://www.kaggle.com/datasets/puneet6060/ intel-image-classification, accessed on 11/09/22.

E. Phaisangittisagul, "An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network," 2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), 2016, pp. 174-179, doi: 10.1109/ISMS.2016.14.