DEEP LEARNING PROJECT 2022
Assignment Group 12
CNN Image Classification Task
Ela Güven - u846576

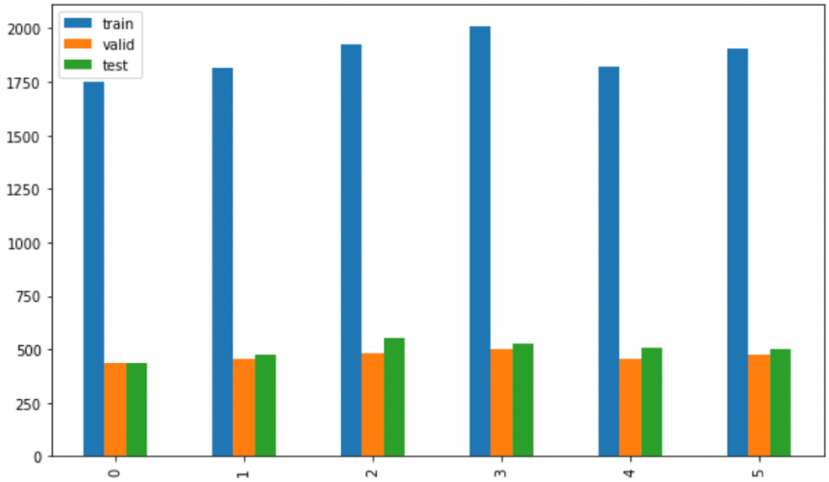
A Convolutional Neural Network (CNN or ConvNet) is one type of neural networks that is used for computer vision tasks such as image classification and segmentation or object detection. It processes the data by considering it in grid-like topology because this type of neural network uses the information that comes from the locality.

Since spatial information is crucial in image classification problems, CNN works ideally for this type of dataset to detect labels of images. In this assignment, we chose to implement CNN algorithm for the Intel Image Classification dataset [1].

First, we imported required packages from the libraries and loaded the data with Keras ImageDataGenerator which helps pre-processing the data. ImageDataGenerator gets the input data and makes data augmentation to increase the generalization of model with some transformations like rescaling, zoom range, horizontal flip, width shift range or rotation range. In this way, the model will make most of the training data and generalize better. This also helps to prevent overfitting. Although data augmentation is one way to prevent overfitting, the augmented samples can be correlated [2]. In this step, the provided code is used to read and separate the dataset. In common agreement, the entire training set was split into training and validation subsets with a proportion of 0.8 and 0.2 respectively. In addition, shuffling was put on for training dataset and it enables to mix up the data in different orders. This helps to prevent overfitting. Later, shuffle was turned off for validation and test set because images can be accessed in same order. In this way, prediction of the images with their unique class will be easier and evaluation metrics both in validation and test set will have similar scores.

The dataset consists of 11230 images in training, 2804 images in validation and 3000 images in the test set with 6 different classes. Proportions of each class regarding training, validation and test set were plotted with their class indexes.

Figure 1: Class distributions over different datasets



Since we have multi classes for this type of classification problem, we designed the model and selected the activation function considering this criterion.

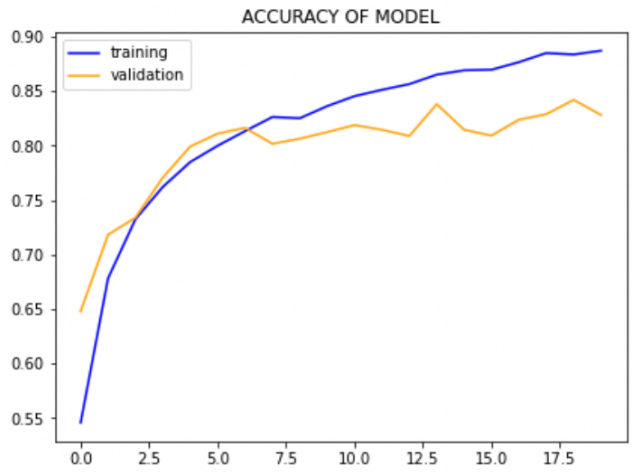
Secondly, a baseline CNN algorithm which is defined in the assignment was implemented. For building network, the Keras sequential model was used and this network was composed of convolutional layers, max pooling layers, dense layers, and an output layer. We built the convolutional layer with 64 filters with the kernel size of 3x3 with ReLu activation functions. In convolutional layers, features will be extracted from the images. Then, structure followed by max pooling layers with the size of 2x2 with three times. In this step, the size of the images was decreased while preserving the information about images. Therefore, the size of the output will be decreased as well. Padding was also added to baseline model in case input size and stride size doesn't fit perfectly. Following step, flattening the layers was used to transform image formats from two dimension of pooled feature to one dimension. After flattening the layers, two fully connected dense layers of size of 64 and 32 with ReLu activation was added. Finally, output layer size of 6 with SoftMax activation was added to the model. As it is mentioned above, there is a multi-class classification problem with 6 different classes in this dataset so same number of nodes was used here. Also, SoftMax activation function was selected because it is most used activation function for last layer in multi class classification tasks. It assigns probabilities of one image where belongs to one of the classes. Adam as optimizer and accuracy as metric were selected for this model. Cross entropy was used for the loss function while optimizing the classification models. This baseline model was structured within the scope of assignment and the changes in the structure will be made after the evaluation of the performance. The baseline model was trained with 20 epochs and batch size as 32.

Table 1: Baseline CNN model for image classification

```
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 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
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
_________________________________________________________________
```

Figure 2: Accuracy and loss plots for Intel Image Classification

a. Training and validation loss

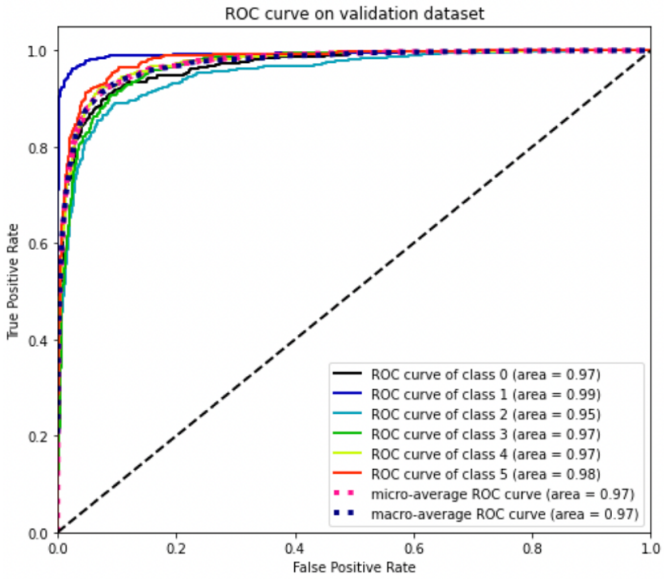


b. Training and validation accuracy

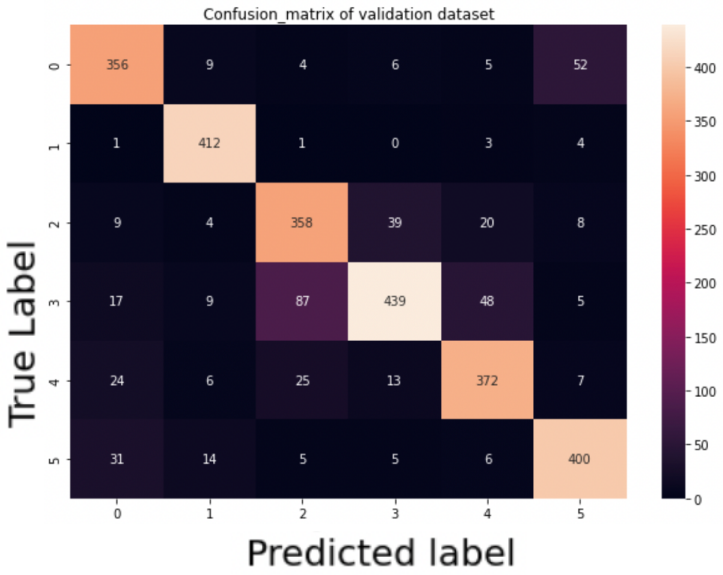
Loss of the model was analyzed to measure the performance of the baseline model because it specifies the errors which were produced by the model. Whereas training loss is the error of the model on training set, validation loss is the errors are measured on validation dataset. Training loss is calculated after each batch and explains if the model fits the training data. Validation loss is used to evaluate the performance of the model on the validation set. It is computed after each epoch and if the model requires any tuning.

Looking at the plots above, it can be said that training accuracy is getting higher than validation accuracy because the model has better performance on the training dataset than validation dataset. If training would be kept for more epochs, then the gap between them will increase. The moment when validation loss stays without improvement, then the performance of model will not increase anymore. Since too many epochs may lead to overfitting of training dataset, early stopping can be implemented for better performance. This method allows to stop training when the performance of the model stops improving on the validation dataset. Call-back functions can be handy to use in this sort of situations to preserve the best out of the training process. Using a larger training dataset is one another solid approach to avoid from overfitting.

Figure 3: ROC curve and confusion matrix of validation dataset



a. ROC Curve



b. Confusion Matrix

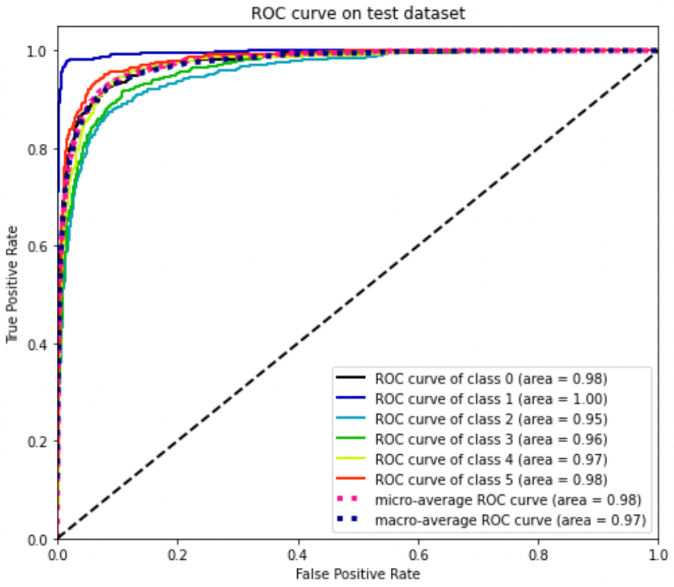
ROC (receiver operating characteristic) curve shows the performance of different models via true positive rate (TPR) and false positive rates (FPR). In the ROC curve, if classifier gives curves close to the top left corner, then the model has a better performance.

Looking the confusion matrix for validation dataset, the classification of classes is not harmonic to each other. Whereas the model can successfully classify the class 1, there are many wrong classifications for class 3. Also, the model classified class 0 as class 5 in many cases.

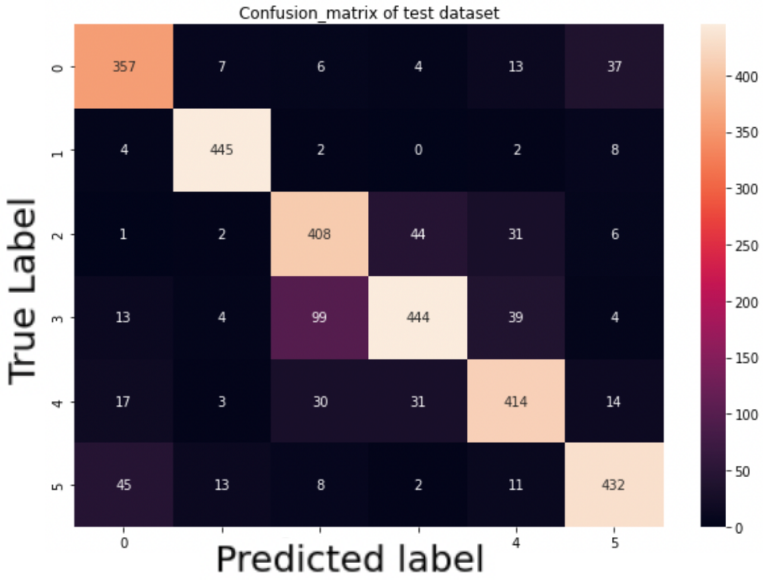
Table 2: Performance measures of validation dataset

```
classification_report of validation dataset :

              precision    recall  f1-score   support

           0       0.82      0.81      0.82       438
           1       0.98      0.91      0.94       454
           2       0.82      0.75      0.78       480
           3       0.73      0.87      0.79       502
           4       0.83      0.82      0.83       454
           5       0.87      0.84      0.85       476

    accuracy                           0.83      2804
   macro avg       0.84      0.83      0.84      2804
weighted avg       0.84      0.83      0.83      2804

############################################################
validation_accuracy: 0.833
auc score of validation dataset = 0.9725285971566359
############################################################
```

Figure 4: ROC curve and confusion matrix of test dataset



a. ROC Curve



b. Confusion Matrix

Table 3: Performance measures of test dataset

classification_report of test dataset:

	precision	recall	f1-score	support
0	0.84	0.82	0.83	437
1	0.97	0.94	0.95	474
2	0.83	0.74	0.78	553
3	0.74	0.85	0.79	525
4	0.81	0.81	0.81	510
5	0.85	0.86	0.85	501
accuracy			0.83	3000
macro avg	0.84	0.84	0.84	3000
weighted avg	0.84	0.83	0.83	3000

```
############################################################
test_accuracy: 0.833
auc score of test dataset = 0.9743244355051994
############################################################
```

Thirdly, an improved model was created based on baseline model and the performance of the model on the test set was increased with different adjustments. Hyperparameter tuning is used to have an impact on the performance of the model by controlling the learning process. Some hyperparameters in the model were changed to find the best combinations of the parameters.

To increase the accuracy on the test set, adding more layers for network structure was tried to decrease the learning process with many epochs. Adding two more layers was assumed to help us for extracting more features. Since the baseline model had an overfitting issue, adding more layers increased the model complexity and the model tended to overfit more.

Batch normalization technique was applied for the network to set the activations and inputs for a layer in each mini batch. In this way, learning process became more stabilized and the number of training epochs was decreased. It also has a regularization impact which helps to reduce the error.

Another regularization techniques to prevent the model from overfitting were applied such as different network size, the weight regularization and dropout method. In regularization, additional parameters were added to the model to decrease the complexity. For this classification task, the aim is trying to figure out important features rather than removing the features of the image. Thus, L2 regularization was used to forcing the weight to take small values.

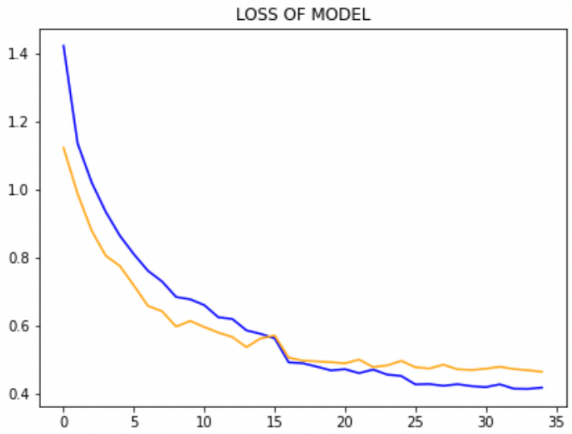
Dropout was applied to set zero for randomly chosen output features of the layer during training. In this way, randomly removing a different subset of neurons on the layer can bring noise in the output values of a layer and yields the robustness to variations of images [3].

After flattening the layers, number of neurons in dense layers were changed from 64 and 32 to 32 and 16. It has been discovered that classifying with a denser layer improved our accuracy result because of reducing the network size.

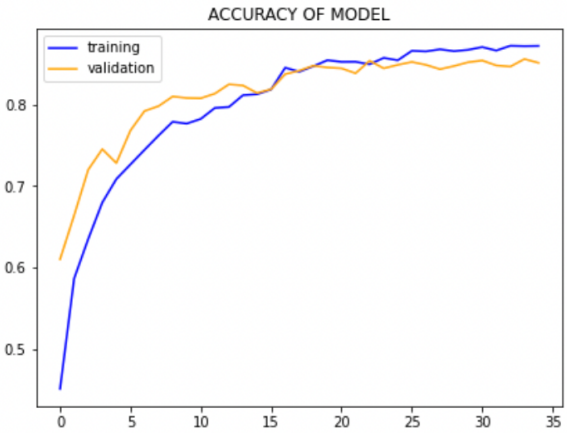
Different optimizers such as SGD and RMSprop were tried. It is observed that Adam optimizer was faster than gradient descent optimizer to reach the best results regarding the number of epochs. Adam optimizer was kept same with respect to time and its efficiency compared to other optimizers [4].

In the improved model, the number of epochs as 35 was implemented but at the same time, learning rate was reduced by a factor of 0.2 in the case of no improvements on validation loss. ReduceLROnPlateau tracked the performance of the model and reduced the learning rate. In this way, it allowed to take smaller learning steps while finding the optimal solution. Decreasing the learning can improve generalization accuracy, especially on large, complex problems [5].

Figure 5: Accuracy and loss plots for improved model



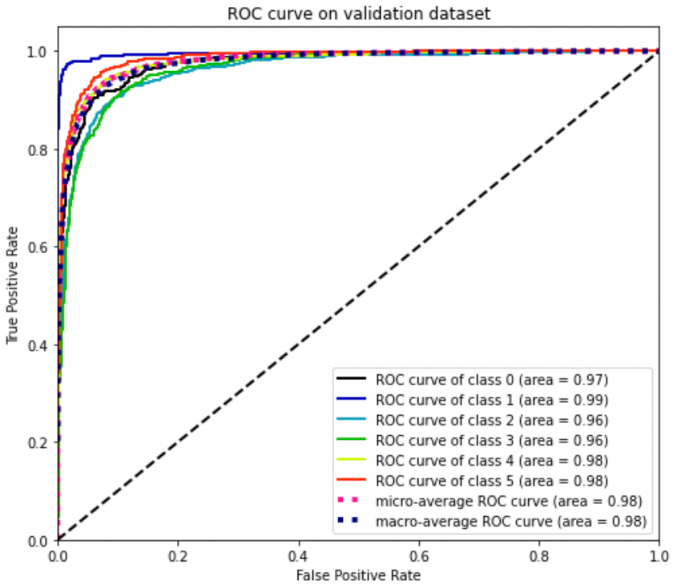
a. Training and validation loss



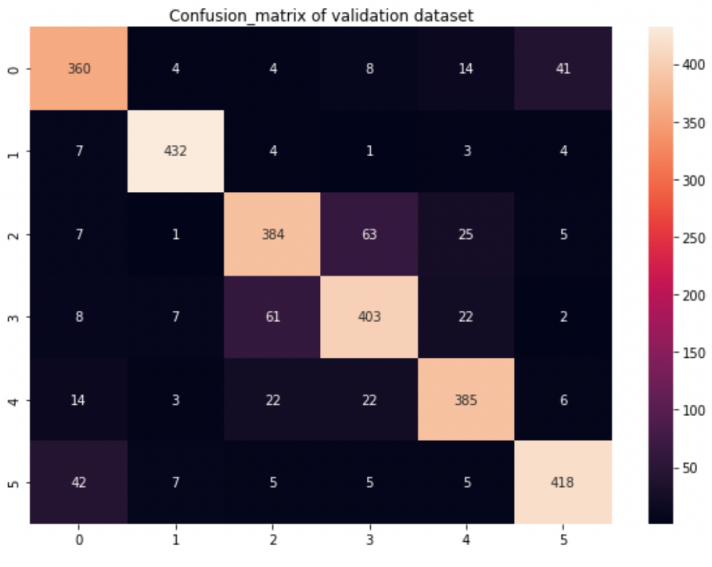
b. Training and validation accuracy

Training process performance has been saturated, reached to the accuracy floor. Overfitting issues have been resolved in small amount. Even though learning rate has been reduced, initial cross entropy loss successfully recovered. However, interclass confusion still remains same, i.e., classifier unable to differ 0 and 5 classes for the most of time like it did not for the first model. There are some improvements but some deteriorations as well.

Figure 6: ROC curve and confusion matrix of validation dataset on improved model



a. ROC Curve



b. Confusion Matrix

Table 4: Performance measures of validation dataset

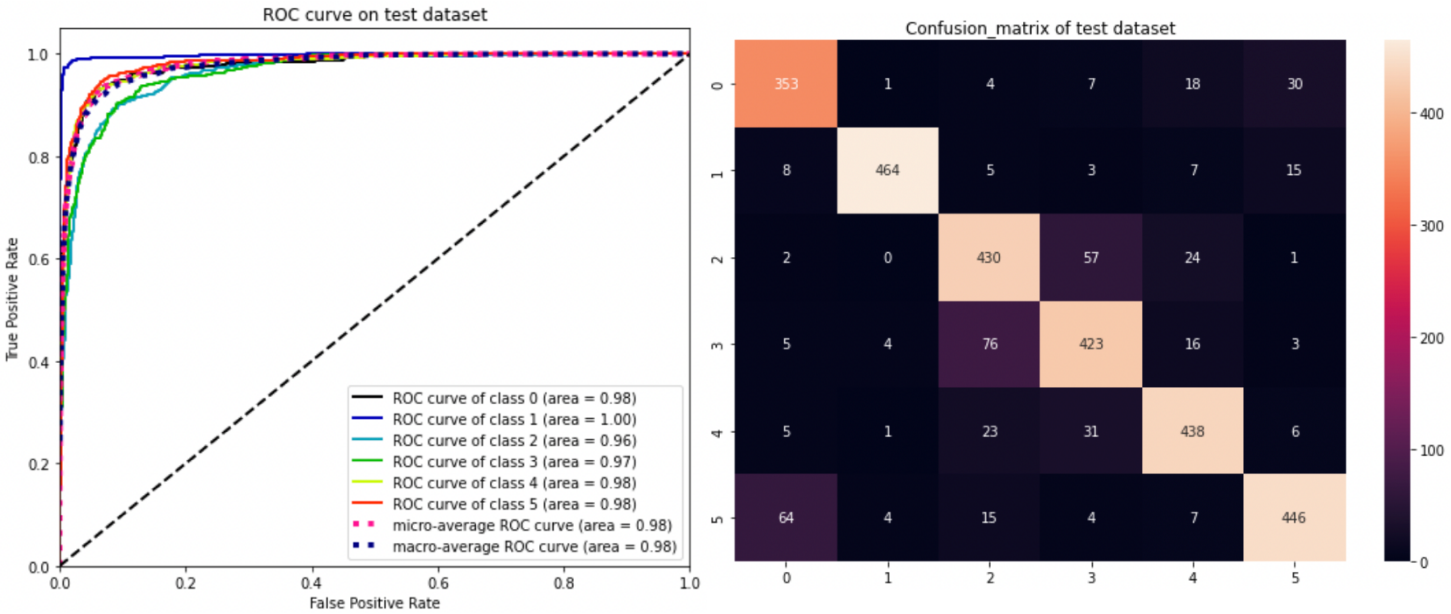
classification_report of validation dataset :

	precision	recall	f1-score	support
0	0.84	0.82	0.83	438
1	0.96	0.95	0.95	454
2	0.79	0.80	0.80	480
3	0.80	0.80	0.80	502
4	0.85	0.85	0.85	454
5	0.87	0.88	0.87	476
accuracy			0.85	2804
macro avg	0.85	0.85	0.85	2804
weighted avg	0.85	0.85	0.85	2804

```
############################################################
validation_accuracy: 0.850
auc score of validation dataset = 0.9768595788124692
############################################################
```

It has been observed that tuning hyperparameters has limited capability to improve a network by itself. Accuracy performance shows similar results yet, acceleration in learning process has been regularized.

Figure 7: ROC curve and confusion matrix of test dataset on improved model



a. ROC Curve

b. Confusion Matrix

Table 5: Performance measures of validation dataset

```
classification_report of test dataset:

              precision    recall  f1-score   support

           0       0.85      0.81      0.83       437
           1       0.92      0.98      0.95       474
           2       0.84      0.78      0.81       553
           3       0.80      0.81      0.80       525
           4       0.87      0.86      0.86       510
           5       0.83      0.89      0.86       501

    accuracy                           0.85      3000
   macro avg       0.85      0.85      0.85      3000
weighted avg       0.85      0.85      0.85      3000

  ############################################################
  test_accuracy: 0.851
  auc score of test dataset = 0.9777765681523414
  ############################################################
```

Compared to baseline model, the improved model worked little bit better on test set and the accuracy was increased. It is safe to assume that two models' accuracy for each classes varies partially. For instance, Class 2 has better performance on model 1 by precision and f1 score while Class 4 performance has been enhanced. Transition between two models gives user to a flexibility to assign a priority to different classes.

Lastly, a new model was trained using transfer learning method where patterns in the other network can be used to identify completely different datasets. There are many best performing models which can be reused for image classification tasks. One of the best models is VGG16 model and Keras enables to access this pre-trained model easily. VGG16 can classify 1000 classes and this network was trained on ImageNet dataset which contains more than 14 million high resolution images. In this task, VGG16 achieved 92.7% accuracy with training millions of images [7].

The dataset was loaded and some pre-processing steps were applied to make it ready as inputs of transfer learning. The image input shape was changed from 64x64x3 to 150x150x150. Also, other transformations were done in the input data such as rotation rate, sheer range, brightness range and width shift range to make more benefit from training data. This will enable to make better generalizations on unseen data.

To utilize VGG16 architecture for feature extraction in this classification task, the final layers of VGG16 was removed. A new flatten layer was added after the last pooling layer in the VGG16 model. Then two dense layers were added. First dense layer contained 128 units with ReLu and final dense layer still has 6 units with SoftMax activation. For hyperparameter tuning, dense layers were tried with 64, 128 and 512 units during the training. Also, batch normalization technique was done after flatten layer. The best performance was received with 128 units.

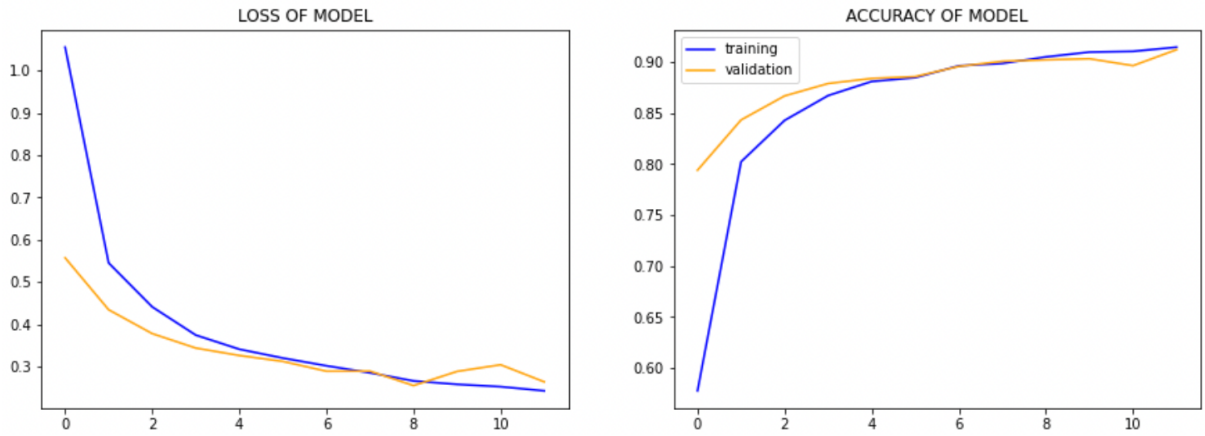
The number of the epoch as 40 with different batch sizes such as 32, 64 and 80. Increased batch size and epochs didn't help more after some points so therefore, batch size as 32 and epoch numbers as 20 were kept same as in the beginning.

Strategies like early stopping, reducing the learning rate and dropout methods are used to prevent overfitting. Since the best accuracy performance was received at 12 epochs before any drop on validation accuracy, early stopping feature stopped the training automatically. Different dropout rate and learning rates were selected for hyperparameter tuning. Finally, training the model was completed.

Table 4: Transfer Learning Model with VGG16

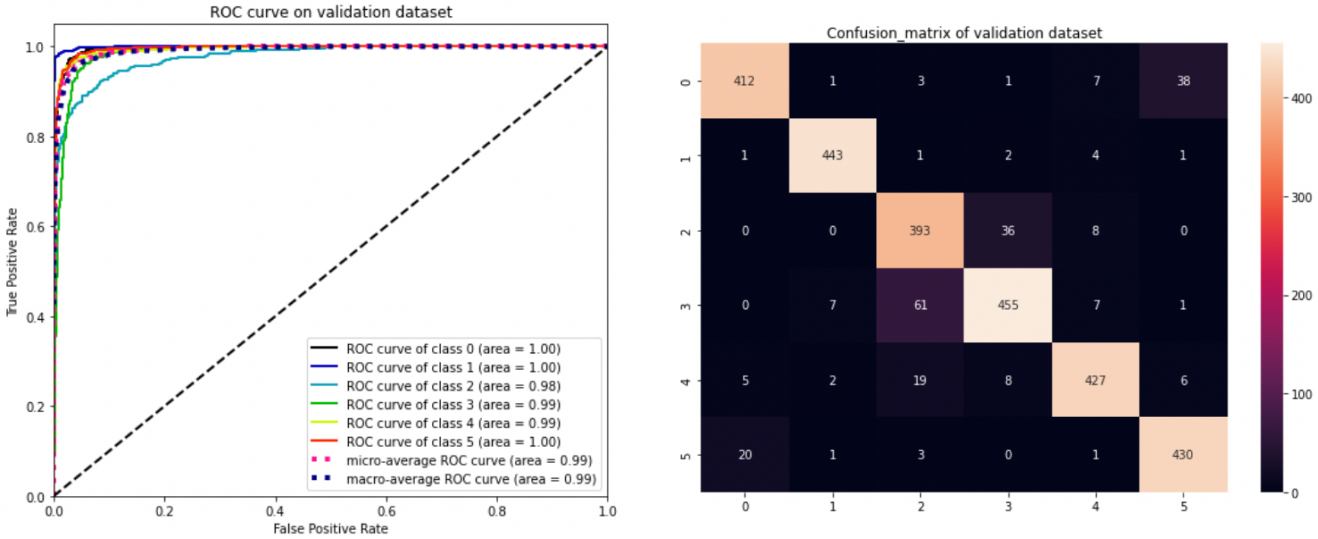
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
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_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_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_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8192)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               1048704
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 774
=================================================================
Total params: 15,764,166
Trainable params: 15,764,166
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Accuracy and loss plots for model with transfer learning



This pretrained model with transfer learning provided a significant decrease on the loss of the model with a small number of epochs.

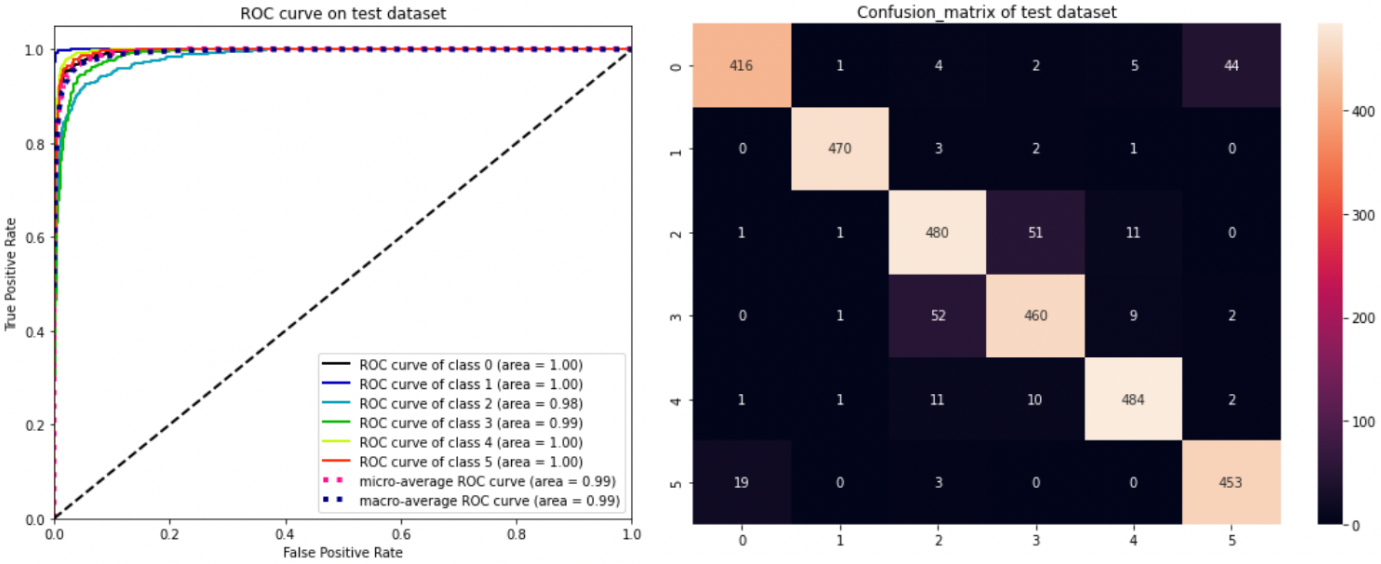
Figure 9: ROC curve and confusion matrix of validation dataset with transfer learning



a. ROC Curve

b. Confusion Matrix

Figure 10: ROC curve and confusion matrix of test dataset with transfer learning



a. ROC Curve

b. Confusion Matrix

Confusion matrix showed that most of the classes are classified correctly with this model.

Table 5: Performance measures of validation dataset

classification_report of validation dataset :

	precision	recall	f1-score	support
0	0.89	0.94	0.92	438
1	0.98	0.98	0.98	454
2	0.90	0.82	0.86	480
3	0.86	0.91	0.88	502
4	0.91	0.94	0.93	454
5	0.95	0.90	0.92	476
accuracy			0.91	2804
macro avg	0.91	0.91	0.91	2804
weighted avg	0.91	0.91	0.91	2804

```
############################################################
validation_accuracy: 0.913
auc score of validation dataset = 0.9915508943426102
############################################################
```

Generalization performance was increased for each class after the implementation of transfer learning.

Table 6: Performance measures of test dataset

classification_report of test dataset:

	precision	recall	f1-score	support
0	0.88	0.95	0.92	437
1	0.99	0.99	0.99	474
2	0.88	0.87	0.88	553
3	0.88	0.88	0.88	525
4	0.95	0.95	0.95	510
5	0.95	0.90	0.93	501
accuracy			0.92	3000
macro avg	0.92	0.92	0.92	3000
weighted avg	0.92	0.92	0.92	3000

##
test_accuracy: 0.921
auc score of test dataset = 0.9937605743055532
##

In conclusion, the model with transfer learning showed better performance compared to other two models. A pretrained model gained rapid progress than other models which were requiring the training from scratch. Designing these two models is computationally expensive and needs a large amount of data to achieve the best performance score. Since VGG16 model has a structure to understand features of images better than other two models, it is computationally efficient and enabled better result for this dataset.

References

1. Intel Image Classification Dataset: https://www.kaggle.com/datasets/puneet6060/ intel-image-classification, accessed on 11/09/22.
2. Taylor L. and Nitschke G. "Improving Deep Learning using Generic Data Augmentation"
3. Park S. and Kwak N. "Analysis on the Dropout Effect in Convolutional Neural Networks"
4. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv*. https://doi.org/10.48550/arXiv.1412.6980
5. Wilson Randal D. and Martinez Tony R. "The Need for Small Learning Rates on Large Problems" International Joint Conference on Neural Networks (IJCNN'01), 115-119.
6. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." Journal of Machine Learning Research 15 (2014) 1929-1958
7. Simonyan K. and Zisserman A. "Very Deep Convolutional Networks for Large-Scale Image Recognition"