SAN JOSÉ STATE UNIVERSITY

EE258: Neural Networks

PROJECT #1

Under guidance of

Prof. Birsen Sirkeci

Fall – 2020

CLASSIFICATION OF IMAGES – INTEL DATASET

By

Pradeepa Rathi Thiagarasu (015015104)

Pooja Narasimmarao Padmanaban (014637805)

Abstract:

The goal of this project is to implement and compare the performance of different fully connected feedforward neural networks on the intel dataset classification. The report discusses in detail about the data used for training and testing, preprocessing techniques, neural network models utilized, performance metrics and conclusion of the project.

Dataset Description:

Intel dataset is a collection of natural scenes. The training set (seg_train) contains 14034 images and test set (seg_test) contains 3000 images. In total, there are 17034 images of size 150x150 pixels. The training and test dataset fall into the following 6 categories: buildings, street, forest, glacier, mountain and sea. Since the dataset contains labels this is a supervised learning.

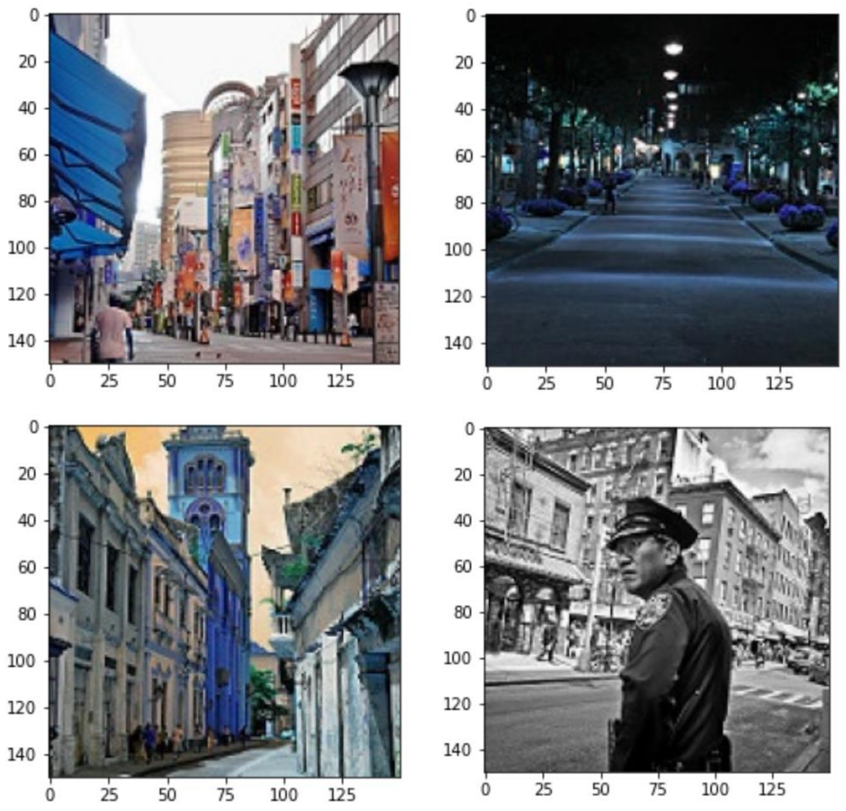


Fig 1 Visualized data from Training set that belongs to Buildings and Street Categories

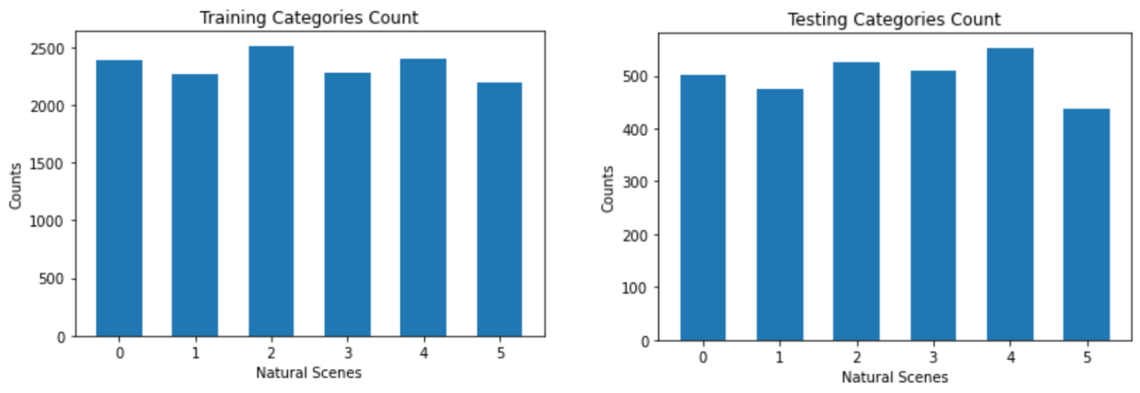


Fig 2 Bar Graph – Category Counts

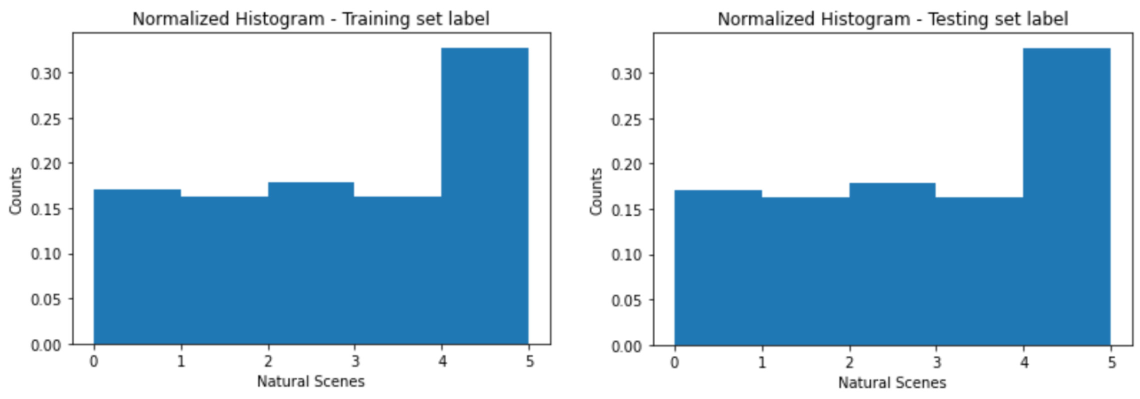


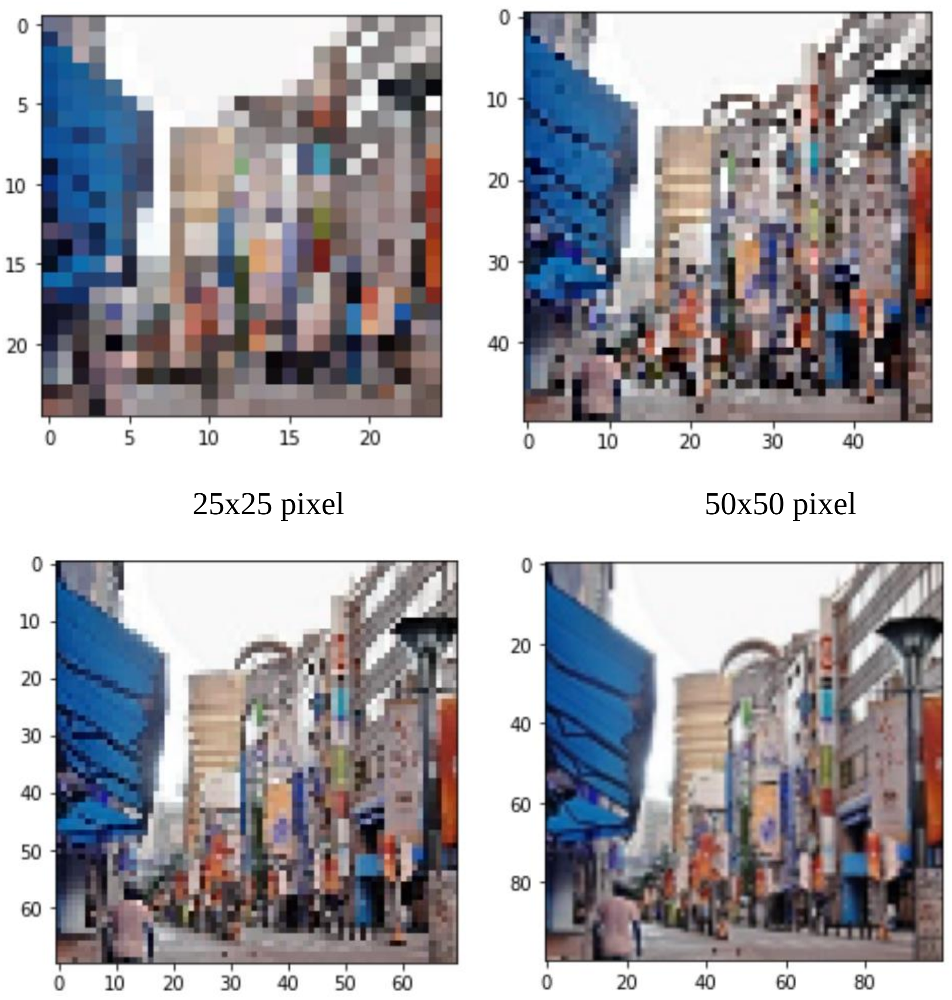
Fig 3 Normalized Histogram – Label

Fig 2 represents the count of images belonging to each category in the training and test sets.
Fig 3 represents the normalized histogram for the same set of values. The above figures depict that the no of data at each category is considerably on similar proportion though there is not an actual balance.

Pre-Processing:

a) Image Resizing

The original input image size is 150x150 pixels. Though the input size can be used as such, the images were resized to 100x100 pixels below which it was difficult to perceive the image category. Resizing will ultimately deform the image features and patterns, there is always a tradeoff between computational efficiency with less memory occupancy and accuracy.

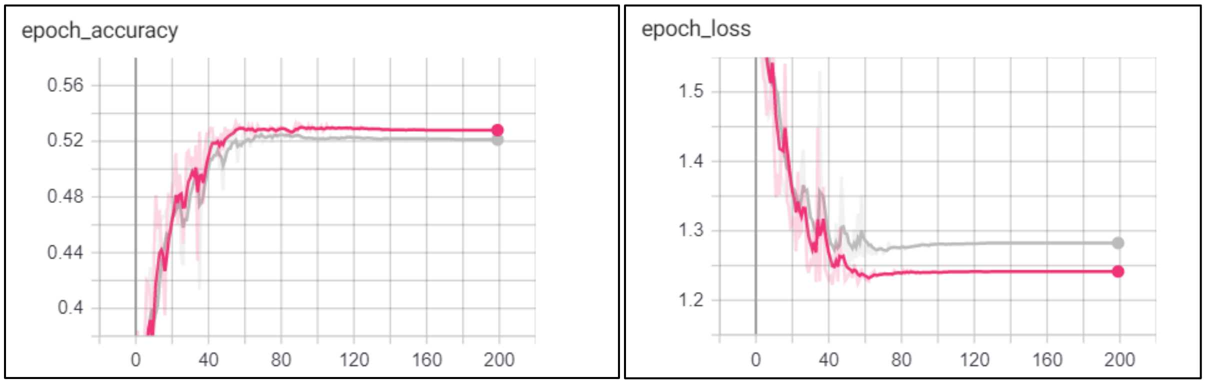


25x25 pixel 50x50 pixel

70x70 pixel 100x100 pixel

Fig 4 Resized Input Image Visualization

As the above figure represents, 25x25 pixel image is greatly deformed and does not give any information about the image content. 50x50 pixel image is better and can be identified to some form of building but with a very low probability.70x70 and 100x100 appears to be the proper resizing that can attain better accuracy with less memory occupancy. We initially trained our networks with the raw image size without any resizing however we then decided to resize in a thought, if the network can properly identify the deformed image, then it is more likely to classify in the unseen data a well identified version of the same image.



● *Model (10000-3000-2000-2000-6) trained with image size 100x100*

● *Model (22500-3000-2000-2000-6) trained with image size 150x150*

Fig 5 Validation accuracy and loss for Image size 150x150 and 100x100

Fig 5 shows that the model with same architecture when trained with image size of 150x150 and 100x100 gave the final validation accuracy of 0.521 and 0.528 respectively. Both yielded a similar accuracy and since resizing the image to 100x100 will require lesser memory we preferred to resize the training and testing images to 100x100.

b) Grayscale Conversion

Input images were imported in grayscale manner as the RGB component did not help in improving the model's performance. Also, one of the significant advantages of using grayscale images is that it takes less memory and eases the computation. The shape of the colored image is 150x150x3 and the shape of a gray scaled image is 150x150x1.

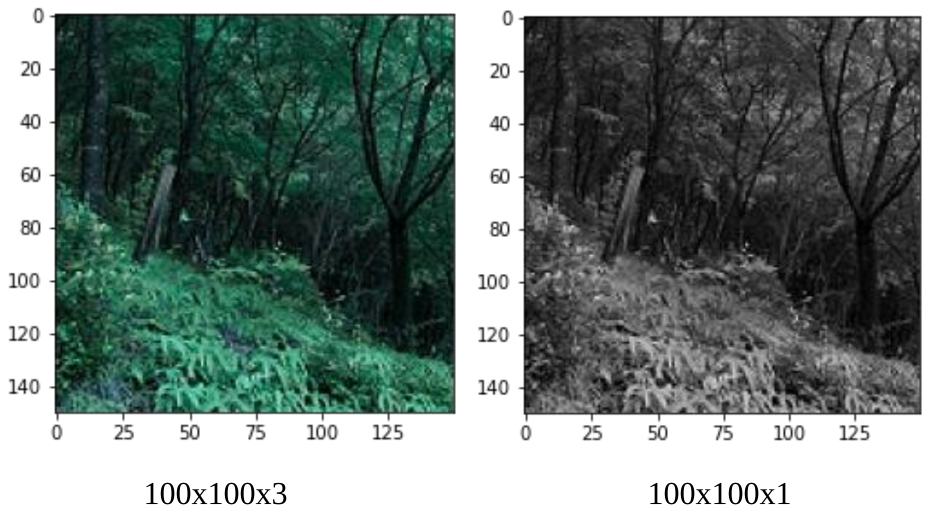
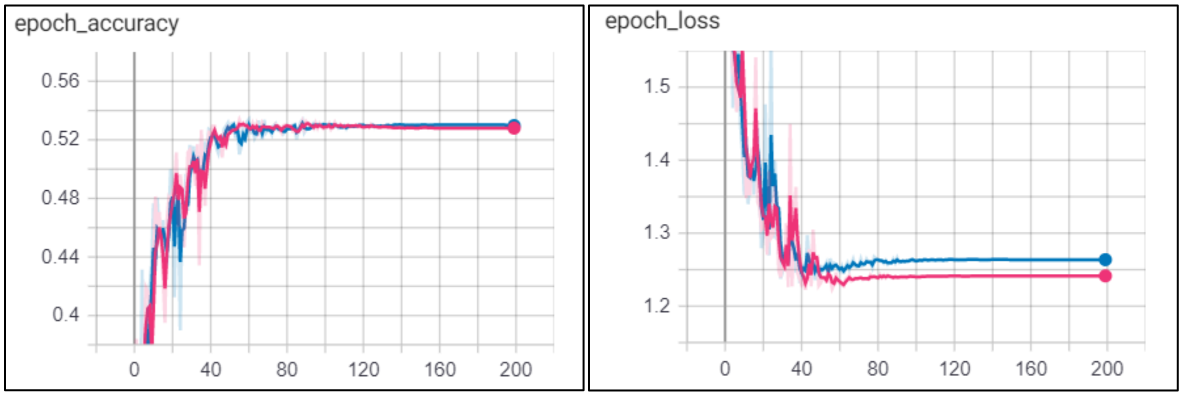


Fig 6 Grayscale Input Image



● *Model (10000-3000-2000-2000-6) trained with grayscale image*

● *Model (10000-3000-2000-2000-6) trained with colored image*

Fig 7 Validation accuracy and loss for colored ● and grayscale images ●

Fig 7 shows that the model trained with colored and grayscale images yielded a validation accuracy of 0.5297 and 0.528 respectively. Since they are similar, we decide to use grayscale mode of input images which again requires less memory and easy for computations.

c) Train Validation Split

We are using Validation Set Approach to split the training data into training and validation set with test size =0.2 and random state 42. Test size =0.2 will split 20% of the training data for validation and the rest 80% for training. We have also tried other cross validation techniques which are explained in detail in model improvements.

d) Shuffling

Random shuffling of data is done as training the algorithm with sorted data will reduce its predictive power. Specifically, for gradient descent algorithms the training data must be independent and identically distributed for it to converge to a global minimum. If the data is sorted with respect to class, then SGD will start to converge based one specific class. To avoid the above problem the data is shuffled before it is fed to the algorithm.

e) Data Scaling

After resizing, the input is a 100x100 images with each input carrying the pixel value ranging from 0-255. The input data is scaled by diving the pixel values by 255 thus scaled between 0-1.

Models Utilized:

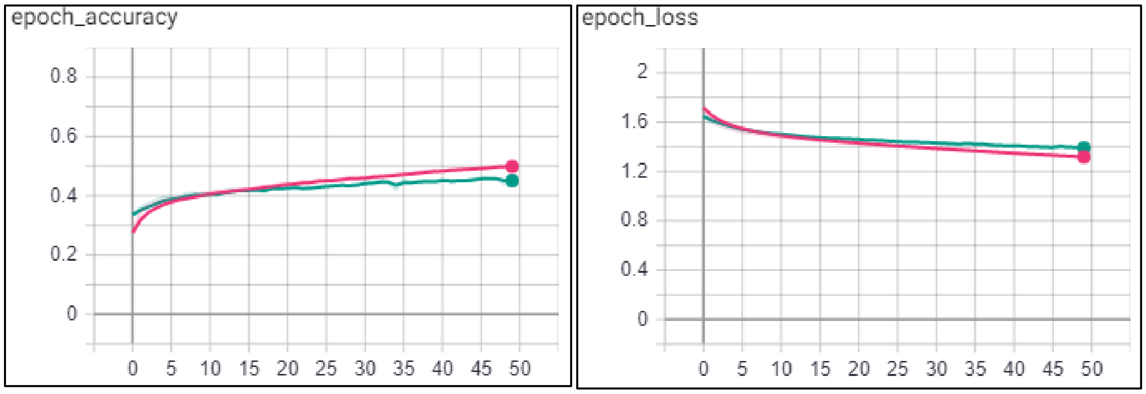
MLP (Multi-Layer Perceptron), a feed forward artificial neural network is used for the classification process as the problem involves categorizing images into 6 different classes. Neural network architecture is implemented using keras, an open source library which runs on top of the tensorflow library. We have utilized stochastic gradient descent optimization algorithm for the models with initial learning rate 0.001. Relu activation function is our first choice for the hidden layers and SoftMax for the output layer.

Model No	Network Architecture	Optimization Algorithm	Batch size	Epochs	Activation Function	Accuracy			Test Classification Error	No of Misclassified Test Datapoints
						Training	Validation set	Test		
1	10000-3000-2000-6	SGD - Mini Batch	128	50	relu-relu-SoftMax	0.4997	0.4515	0.443	0.557	1651
2	10000-3000-2000-6	SGD- Full Batch	11695	200	relu-relu-SoftMax	0.3667	0.3591	0.366	0.634	1901
3	10000-3000-2000-6	SGD	1	25	relu-relu-SoftMax	0.4893	0.3844	0.391	0.609	1672
4	10000-8700-8700-6	SGD - Mini Batch	128	25	relu-relu-SoftMax	0.5264	0.4718	0.468	0.532	1379

Table 1: Basic Models Utilized for Image Classification

a) Model 1(MLP with SGD Batch size =128(minibatch))

This predictive model was attained by performing all the preprocessing steps explained in the previous section with batch size as 128 and epochs 15. The SGD learning rate was kept constant as 0.001. The model gave a training and validation accuracy of 0.4721 and 0.4385, test accuracy as 0.44. Though 0.44 is low, it’s better than our initials trials with architectures such as 22500-600-600-600-600-6, 22500-8700-8700-6, 22500-3000-3000-2000-6. Minibatch gave us better results and took less time compared to other two algorithms with GPU. This motivated us to adopt minibatch and try many model improvisations.

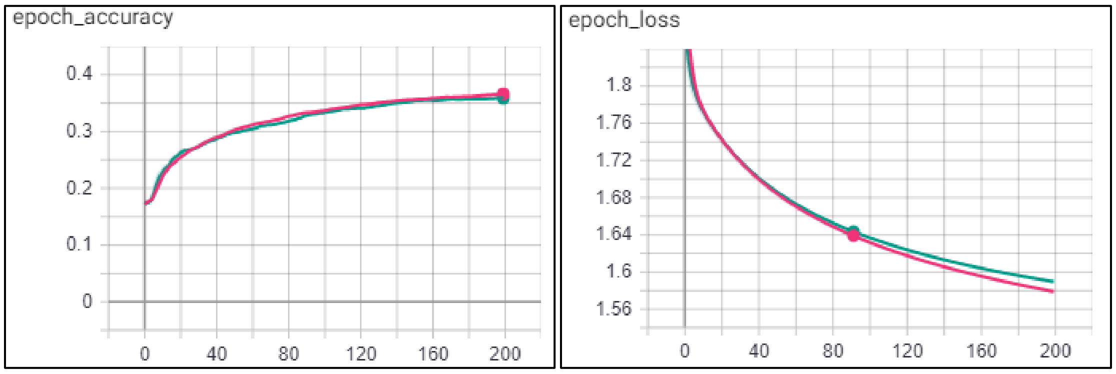


● Training metrics ● validation metrics

Fig 8 Performance of Model 1(10000-3000-2000-6 / SGD - Mini Batch/Epoch 50)

b) Model 2 (MLP with SGD Batch size = 11695(full batch))

For this model, the algorithm was trained with batch size 11695(training data size after validation split) and 100 epochs. SGD learning rate is 0.001. Even with 100 epochs we were able to get a training accuracy and validation accuracy of 0.3181, testing accuracy 0.328. This implies that the algorithm would have converged at a local minimum with a constant learning rate.



● Training metrics ● validation metrics

Fig 9 Performance of Model 2(10000-3000-2000-6 / SGD Full batch/Epoch 128)

c) Model 3(MLP with SGD Batch size =1)

The algorithm was trained with a batch size of 1 and 25 epochs with constant learning rate 0.001. The model gave a test accuracy of 0.391 which is higher compared to mini and full batch, but it took a long time for training. Though SGD's is computationally easy at each iteration, overall, it takes time as the algorithm updates the weights at every iteration.

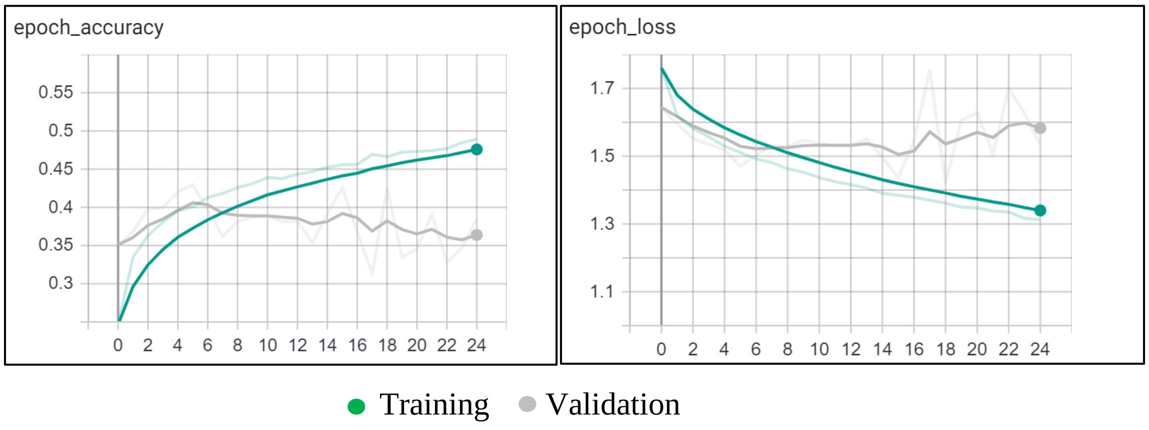


Fig 10 Performance of Model 3(10000-3000-2000-6 / SGD /Epoch 25)

d) Model 4(MLP with SGD Minibatch =128**)**

When training the model with lesser no of neurons and layers, they didn't give an expected performance and training accuracy, so we decided to increase the complexity of the model by increasing the no. of neurons in hidden layer to 8700 and trained with SGD minibatch 25 epochs. The performance was better than all the previously tried models with training and test accuracy as 0.5264 and 0.468. The model correctly classified 1621 out of 3000 test images.

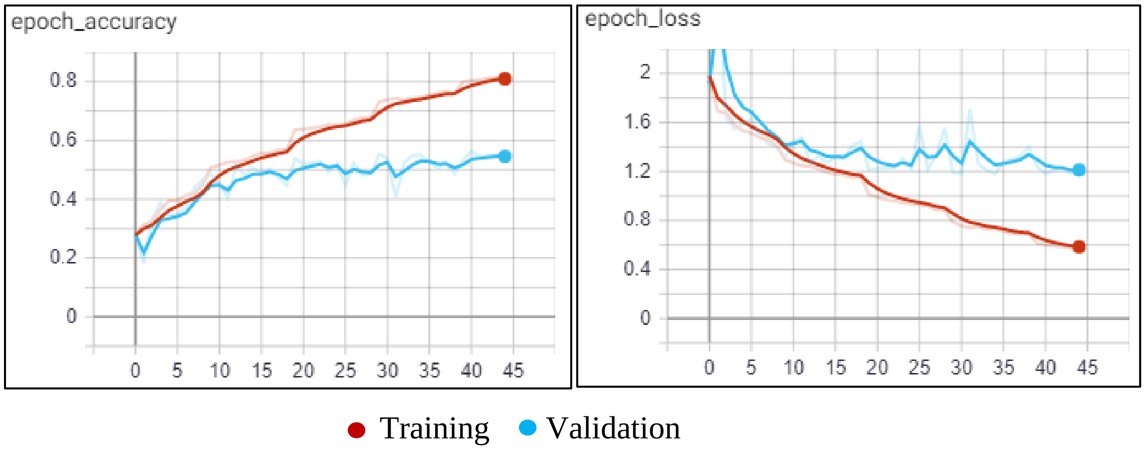


Fig 11 Performance of Model 4(10000-8700-8700-6 / SGD Minibatch /Epoch 25)

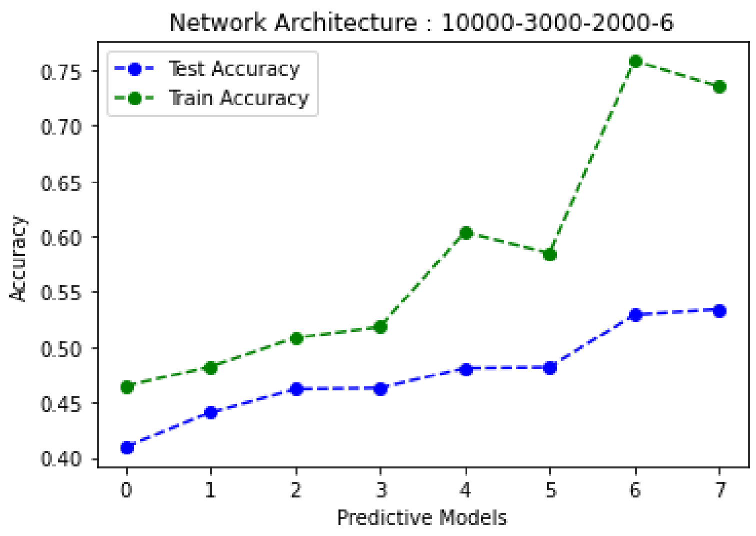
Model Improvements:

Initially we tried a simple model with lesser number of neurons adopting minibatch and constant learning rate. Upon experimenting with different network layer architectures mentioned in Table 1 and their hyperparameters like epochs, batch size, activation functions, validation sets we choose some of the better performing models among them to apply the regularization techniques. 10000-8700-8700-6, 10000-3000-2000-6 and 10000-2000-300-300-128-6

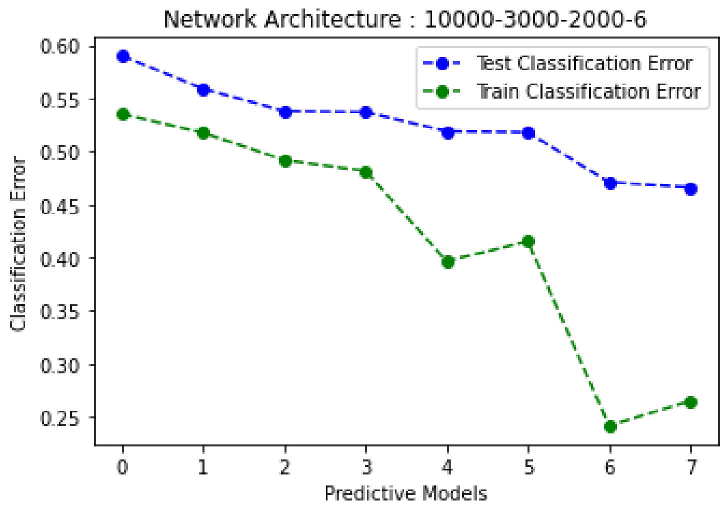
S. No	MODEL HYPERPARAMETERS				MODEL IMPROVEMENTS	PERFORMANCE		Classification Error
	Network Architecture	Epochs	Batch size	Activation Functions		Training /Validation Accuracy	Test Accuracy	
1	10000-600-600-600-600-6	15	128	Hidden - Relu/Out-Softmax	Initial Model	0.3850/0.3848	0.382	0.618
2	10000-8700-8700-6	25	128	Hidden - Relu/Out-Softmax	Increased the no of Hidden layer neurons	0.5264/0.4718	0.468	0.532
3		100	128	Hidden - Relu/Out-Softmax	Introduced Adaptive Learning Rate and Early Stopping	0.8139/0.5451	0.545	0.4549
4	10000-3000-3000-2000-6	15	128	Hidden - Relu/Out-Softmax	Increased the Hidden layers	0.4621/0.4521	0.445	0.555
5	10000-3000-2000-6	15	128	Hidden - Relu/Out-Softmax	Reduced the hidden layer numbers	0.472/0.4385	0.441	0.559
6		15	128	Hidden - Relu/Out-Softmax	Network Trained with RGB images	0.4648/0.4179	0.410	0.59
7		25	128	Hidden - Relu/Out-Softmax	Increased the Epochs	0.5082/0.4641	0.462	0.538
8		200	128	Hidden - Relu/Out-Softmax	Increased the Epochs	0.5848/0.4758	0.482	0.518
9		200	128	Hidden - Relu/Out-Softmax	Introduced **Adaptive Learning Rate (ALR)**	0.7301/0.5490	0.534	0.466
10		200	128	Hidden - Relu/Out-Softmax	ALR with **dropout** (0.5) for last hidden layer	0.7582/0.5361	0.529	0.471
11		50	128	Hidden - Relu/Out-Softmax	Maintained 50 Epochs to avoid overfitting	0.4824/0.4385	0.441	0.559
12		5	1	Hidden - Relu/Out-Softmax	Changed the optimization algorithm to SGD	0.5183/0.4578	0.463	0.537
13		20	1	Hidden - Relu/Out-Softmax	SGD with 20 Epochs	0.5494/ 0.5187 (in the seventh epoch) 0.6035/0.4877	0.481	0.519

14		25	128	Hidden - sigmoid/Out-Softmax	Changed the Activation function for the hidden layer	0.3815	0.3751	0.378
15		25	128	Hidden - Relu/Out-sigmoid	Changed the Activation function for the output layer	0.4364	0.4090	0.410
16	10000-3000-2000-2000-6	200	128	Hidden - Relu/Out-Softmax	ALR and Introduced **Early Stopping** with patience =10 and increased hidden layer with dropout	0.6888/0.5442	0.529	0.471
17		200	128	Hidden - Relu/Out-Softmax	Removed Early Stopping	0.7213/0.5280	0.530	0.47
18	10000-2000-300-300-128-6	100	128	Hidden - Relu/Out-Softmax	Changed neural architecture and Epochs	0.4441/0.4524	0.44	0.56
19		200	128	Hidden - Relu/Out-Softmax	Increased the Epochs	0.5450/0.5009	0.481	0.519
20		300	128	Hidden - Relu/Out-Softmax	Increased the Epochs	0.5738/0.5030	0.497	0.503
21		300	128	Hidden - Relu/Out-Softmax	Dropout at every layer and early stopping with patience = 5	0.3246/0.2876	0.300	0.7
22		300	128	Hidden - Relu/Out-Softmax	Remove early stopping from previous trial	0.5501/0.3909	0.406	0.594
23		300	128	Hidden - Relu/Out-Softmax	Removing dropout from 1st hidden layer and early stopping patience = 10	0.5850/0.5002	0.505	0.495
24	10000-3000-2000-128-6	25	1	Hidden - Relu/Out-Softmax	Changed the neural architecture and Batch size	0.4893/0.3844	0.391	0.609

Table 2 Model Improvement Approaches



a) Predictive Models vs Accuracy



b) Predictive Models vs Classification Error

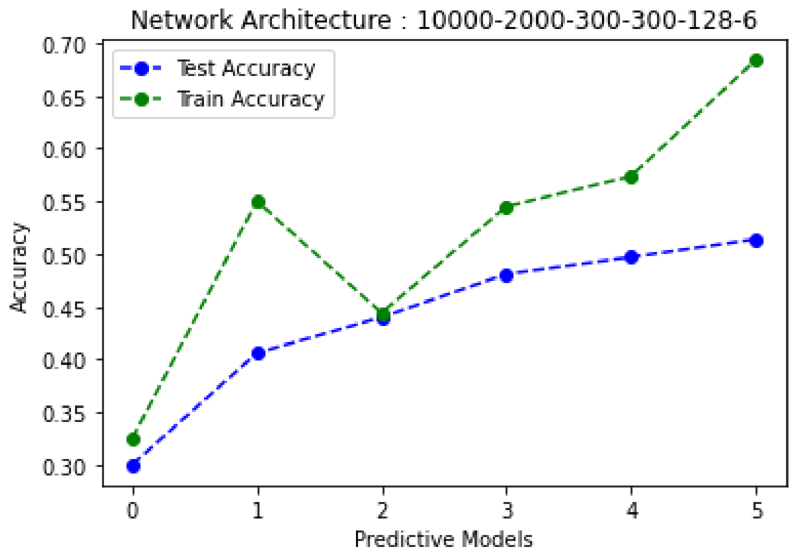
Model 0 : Network Trained With RGB Image
Model 1 : Reduced the No.of Hidden layers
Model 2 : Increased the Epochs from 15 to 25
Model 3 : Changed the optimization algorithm to SGD(Batchsize=1)
Model 4 : Increased the Epochs of SGD to 20
Model 5 : Increased the Epochs to 200 for SGD MiniBatch
Model 6 : Adaptive Learning Rate(ALR) with dropout(0.5) for last hidden layer
Model 7 : Introduced Adaptive Learning Rate

Fig 12 Comparison of Model (10000-3000-2000-6) performance by varying hyper parameters and adopting ALR, Regularization techniques

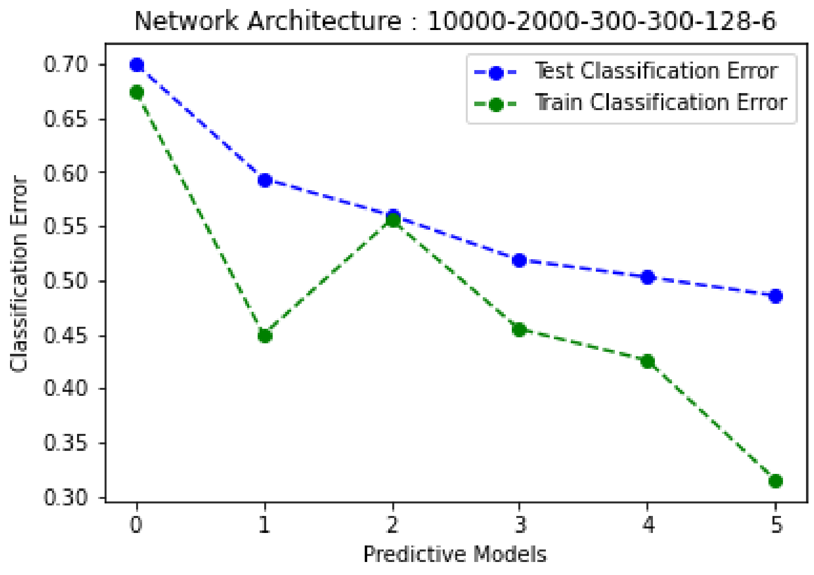
The performance of model with architecture 10000-3000-2000-6 upon trying various improvements is represented in Fig 12. Model 0 is obtained by training the algorithm with colored images in contrast to previous models trained using grayscale images. But there is no significant improvement in performance, test accuracy was 0.441. Model 2 is obtained by increasing the no. of epochs from 15 to 25, the training and test accuracy increased to 0.5082 and 0.468. At this point we decided to increase the no. of epochs to a higher number to attain better accuracy rates. So, the algorithm was trained with 200 epochs to get Model 5 a little increase in performance than the previous models with test accuracy 0.482. With the same no of epochs (200) adaptive learning rate step decay technique was introduced which decreases the learning rate of the gradient descent algorithm when it is close to its convergence. Model 7 was obtained with better test accuracy of 0.5349. This motivated us to create another model 6 by training the algorithm with 200 epochs, ALR and drop out regularization of rate 0.5 for the last hidden layer. This model yielded a little lesser result like Model 7, but the dropout addition should have omitted certain valued links to the output layer.

Fig 13 shows the performance comparison of improvements done on model 10000-2000-300-300-128-6. Initially the algorithm was trained with 100 epochs and then increased to 200 epochs to get Model 3 with training and test accuracy as 0.5450 and 0.481. Model 4 was also obtained in a similar fashion by increasing the epochs to 300 with 0.5738 and 0.497 accuracies. Since we have four hidden layers in this model, dropout was added to every hidden layer for the Model 0. It was taking a long time to run large number of epochs, so the regularization technique Early stopping was used with patience 5 to stop the model if there is no change in the performance for continuous 5 epochs.

Model 0 performance drastically reduced to a test accuracy of 0.300. On analyzing the reason, we observed that adding the dropouts in the first hidden layer is increasing the chance loosing significant links to its neighboring hidden layer thereby missing out features. So, in the next Model 5, the dropout at the first hidden layer got removed and early stopping with patience 10 was used to get the better performance of 0.514 test accuracy for this neural architecture



a) Predictive Model obtained through model improvement techniques Vs Accuracy

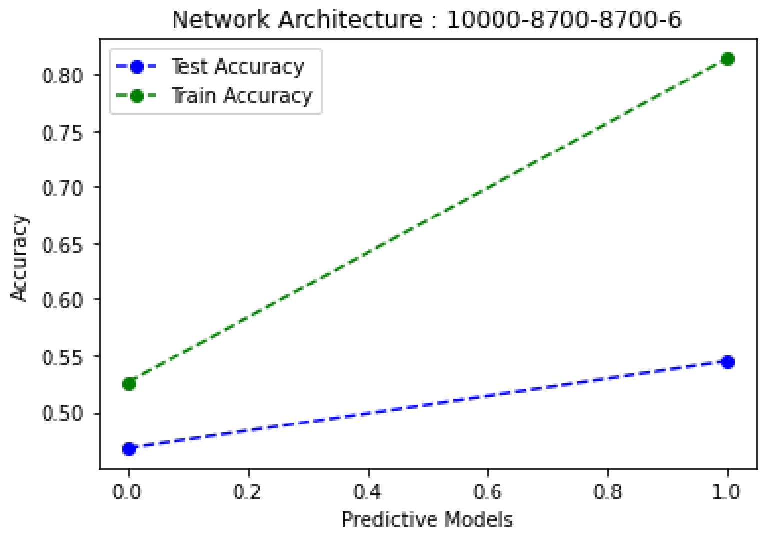


b) Predictive Model Vs Classification Error

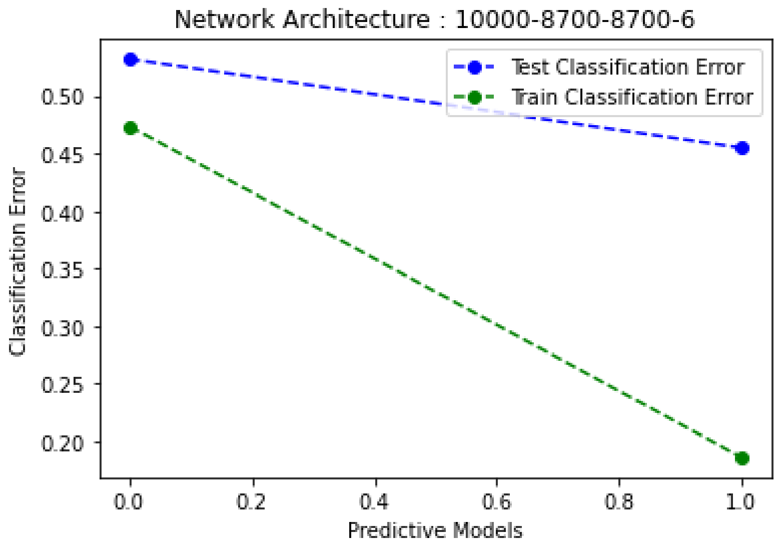
Model 0 : Dropout at every layer and early stopping with patience = 5,Epochs=300
Model 1 : Remove early stopping from previous trial,Epochs=300
Model 2 : Epochs=100
Model 3 : Epochs=200
Model 4 : Epochs=300
Model 5 : Removing dropout from 1st hidden layer and early stopping patience = 10

Fig 13 Comparison of Model (10000-2000-300-300-128-6) performance by varying hyper parameters and adopting ALR, Regularization techniques

Fig 14 represents the performance of neural model 10000-8700-8700-6 where at first the model gave a training and test accuracy of 0.5264 and 0.468. Since the training error is very high, we decided to increase the model's complexity. The architecture already had neural complexity with 8700 neurons in the hidden layer, so no. of epochs was increased along with Adaptive Learning Rate (ALR) step decay and early stopping techniques. The improved model gave a training accuracy of 0.8139 and 0.545 test accuracy. Though 0.545 seems low this was out highest accuracy attained by using MLP.



a) Accuracy of the predictive models with and without ALR and early stopping



Model 0 : Without ALR and Early Stopping
Model 1 : With ALR and Early Stopping

b) Classification Error of the predictive models with and without ALR and early stopping

Fig 14 Comparison of Model (10000-8700-8700-6) Performance by varying hyper parameters and regularization technique

Below is the confusion matrix of the highest performing model 1 (10000-8700-8700-6) among its fellow models after applying model improvements (ALR and early stopping). The model has correctly classified 1604 out of 3000 test datapoints.

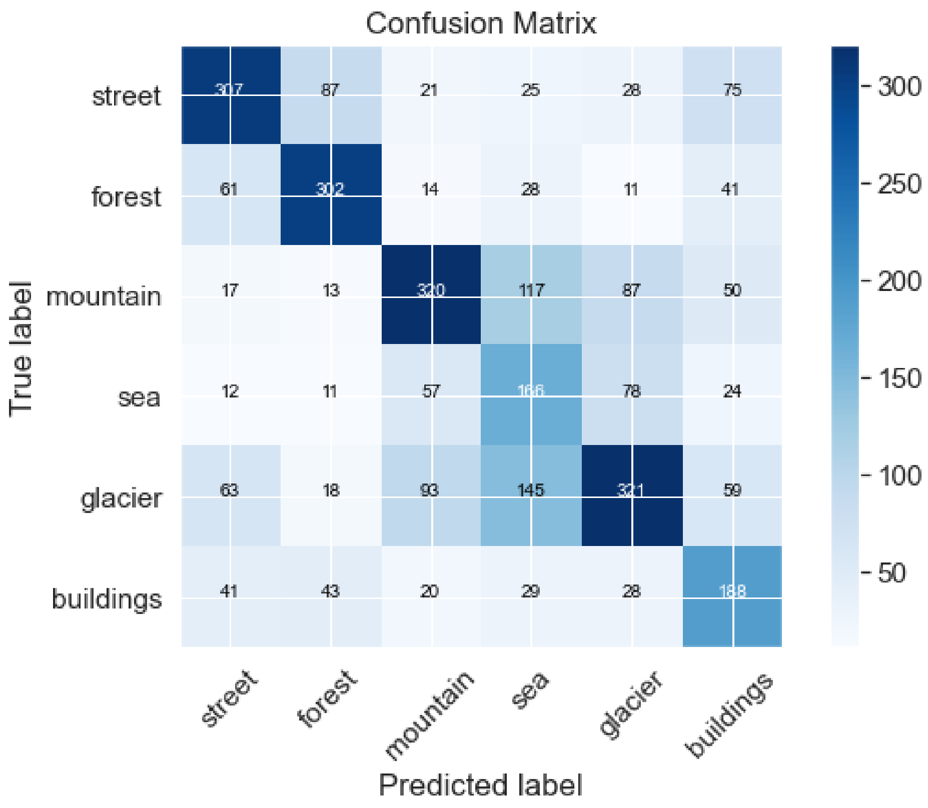


Fig 15 Confusion Matrix Model (10000-8700-8700-6)

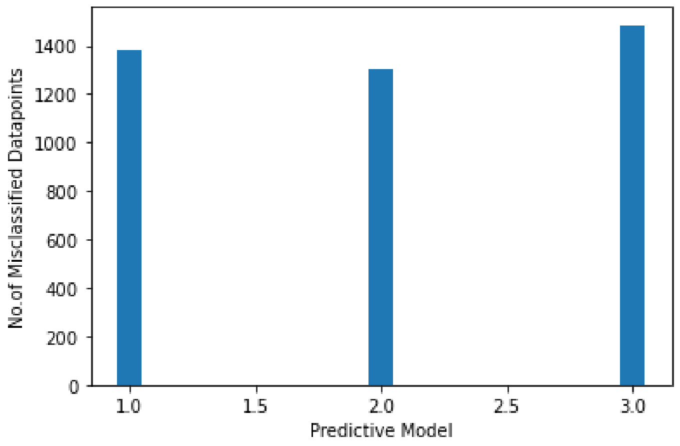
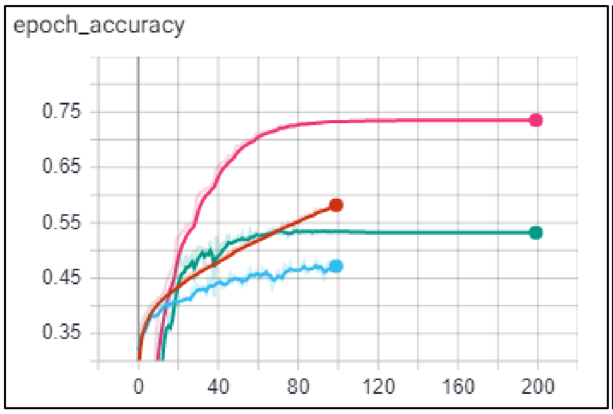


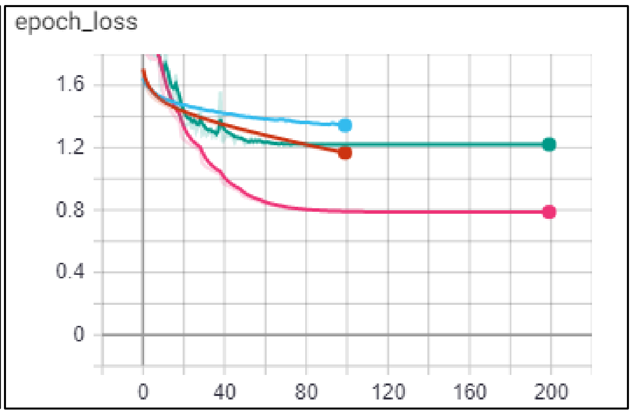
Fig 16 Predictive Model Vs No. Of Misclassified Samples

Adaptive Learning Rates:

Apart from trying out model improvements by varying the hyperparameter, Adaptive learning rate is one of the efficient techniques incase if we are using gradient descent optimization techniques. The basic models utilized in the start were using a constant learning rate (0.001). Gradient descent algorithms calculate the cost function, takes gradient of cost function and moves in the opposite direction of gradient in steps. How large the steps should be, is decided by learning rate. The best practice is to have large learning rates at start to prevent the algorithm from getting trapped in the local minimum and as the algorithm converges to the global minimum the learning rate steps should be smaller.

Learning Rate Scheduler performs the step decay of learning rate by setting the rate to an initial value 0.01 and reducing the rate to half after every 10 epochs. Adopting ALR drastically improved the model’s Test accuracy from 0.482 to 0.5349 as shown in Fig 17 below.





● Model with ALR Validation metrics ● Model without ALR Validation metrics

● Model with ALR Training metrics ● Model without ALR Training metrics

Fig 17 Model (10000-3000-2000-6/Epochs 200/Batch size128)Improvement with Adaptive Learning Rate

Comparing the validation accuracy in the above figure the blue and green lines, adopting a higher learning rate initially and decaying or decreasing when the algorithm converges improves the performance.

Cross Validation Techniques

Cross validation techniques are used to find the ability of model to predict the unseen data when the training is in progress by retaining a portion of the training data for validation. The following validation techniques were tested in the model (MLP, SGD, batch size=128, constant learning rate = 0.001, epochs =20)

1. Validation set Approach – splits the data into two sets based on the test size. Here testsize=0.2 was used which splits 20% data for validation and 80% of data for training, random state =42, Shuffling of the data is done automatically on using test_split_data as the default value for shuffling is set to True.
2. LOOCV (Leave Out One Cross Validation) splits the data by leaving one datapoint and keeping the rest of the datapoints for training. Though it is good for small datasets, it took lot of time for splitting and computation, it also required more memory. So, it was not computed.
3. K-fold – splits the data into n splits and forms k models. Here n_split=3 was used with shuffle and random state = 42. Training data is split into 3 clusters.
4. Stratified K-fold – again splits the data into n folds but with one significance to K-fold is that it has balanced distribution of data from each class in the validation set. n_split=6 with shuffle and random state.

Cross Validation	**Validation Accuracy**	**Test Accuracy**
Validation Test Set	0.3922	0.404
LOOCV	Could not compute	Could not compute
K-fold(n_split=3)	0.4008	0.394
StratifiedKfold(n_split=6)	0.4147	0.416

Table 3 Validation and Test Accuracy for different CV techniques

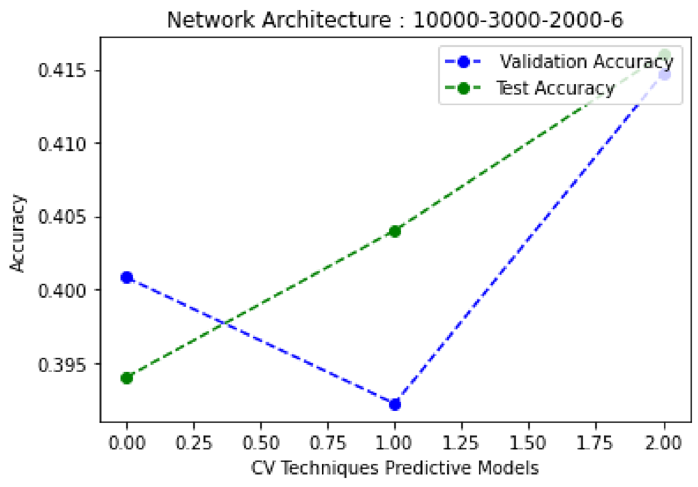


Fig 18 Predictive Models (CV Techniques) Vs Accuracy

Model 0: K-fold(n_split=3)

Model 1: Validation Test set

Model 2: Stratified K-fold(n_split=6)

Comparing the Validation and Test Accuracy value of the specified model on using the mentioned CV techniques, apart from LOOCV all the others produced considerable results. K fold, Stratified Kfold, LOOCV very taking time to split the datasets. So, we adopted validation set approach for all the models which was computationally easy and produced good results.

Activation Function

We also observed the model's performance by changing the activation functions of the hidden and output layers. Sigmoid activation function is used in both hidden and output layers, but there is no significant improvement in performance. For Multiclass classification problem SoftMax at output layer is more appropriate than sigmoid. Sigmoid can serve better in Binary classification problems.

S. No	Model	Activation Function	Accuracy		
			Training	Validation	Test
1	10000-3000-2000-6	Hidden layer -Relu Output layer - Softmax	0.5082	0.4641	0.462
2	10000-3000-2000-6	Hidden layer -Sigmoid Output layer - Softmax	0.3815	0.3751	0.378
2	10000-3000-2000-6	Hidden layer -relu Output layer - Sigmoid	0.4364	0.4090	0.410

Table 4 Performance of Models with Sigmoid Activation Function

Regularization:

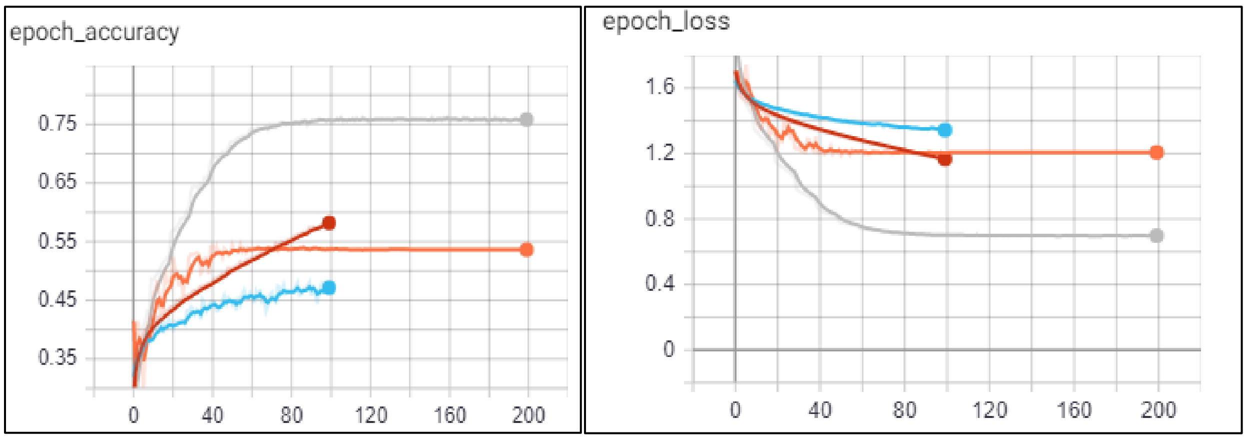
a) Early Stopping

Initially the models were producing less training accuracy. To improve the algorithms performance, the hyperparameter no. of epochs was increased to 200,300. Training with large epoch values took long time and we need to wait till the model is trained even if the performance is not improving. Early Stopping callback function will keep a track on the algorithm's performance while training and takes decision to stop the training if there is no improvement in the performance. We can set the patience value to the early stopping say 3, then the function will monitor for similar performance up to 3 epochs and halts the training. In general, if the model accuracy is not improving in a consecutive 3 epochs, then it is very unlikely that the accuracy will not improve after that. So, the patience level was set to 3 in this case. This also helps to capture the model near to its optimal performance by eliminating the unwanted epochs thereby saving time.

b) Dropout

Dropout is another regularization parameter that is used to track the active connections between the layers of neurons. It sets a threshold probability value for a connection, below which the connection will be removed (connection weight is set to zero). The models were regularized with a dropout rate of 0.5. We also observed that putting dropouts at the first hidden layer reduced the model's performance. For example, for the model 10000-2000-300-300-128-6 mentioned in the improvements section dropped in accuracy from 0.497 to 0.300 after adding the dropout layer in the first hidden layer.

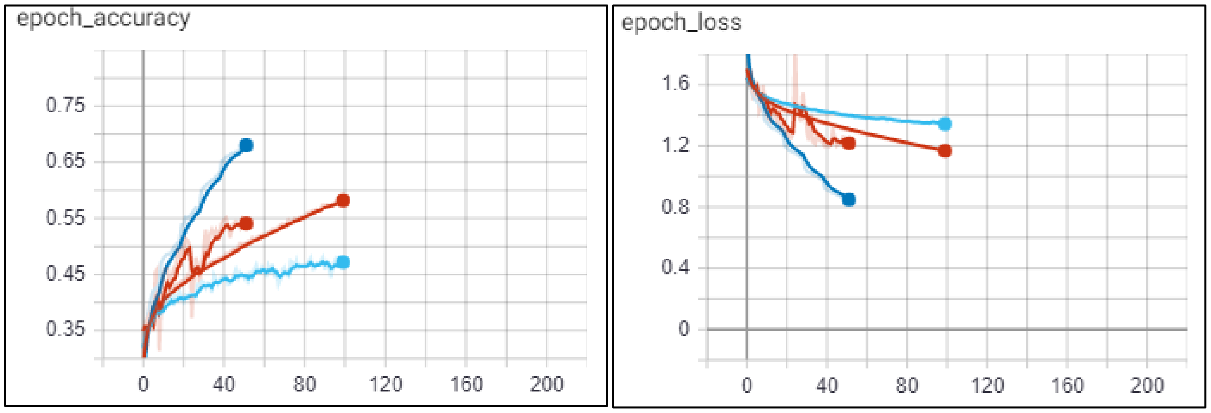
Fig 19 20 21 represents the performance of models with and without the regularization techniques. In most of the cases regularization helped to achieve a better performing model



● Model without Dropout Validation metrics ● Model with Dropout Validation metrics

● Model without Dropout Training metrics ● Model with Dropout Training metrics

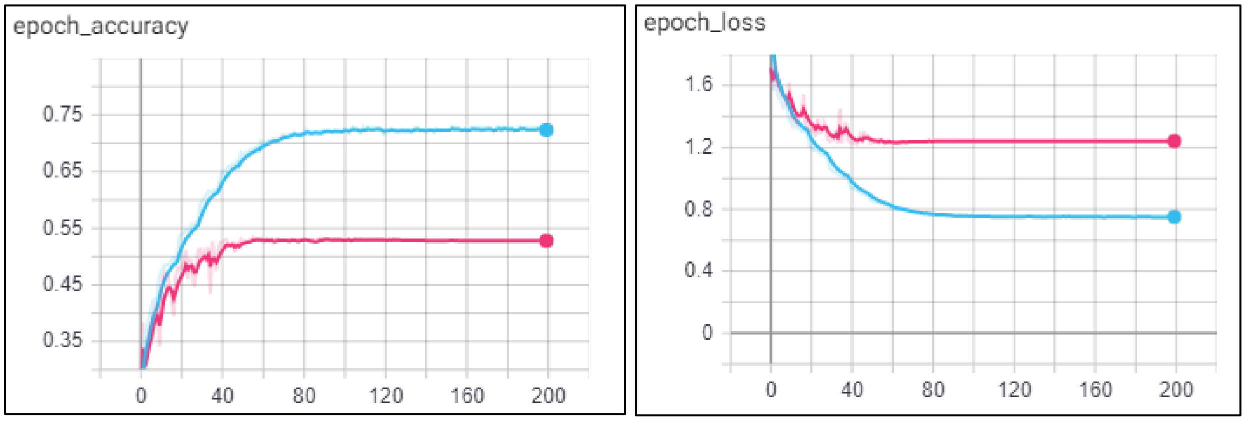
Fig 19 Drop out Regularization



● Model without regularization Validation metrics ● Model with regularization Validation metrics

● Model without regularization Training metrics ● Model with regularization Training metrics

Fig 20 Drop out and Early Stopping (ES) Regularization



● Model without regularization Training metrics ● Model with regularization Validation metrics

Fig 21 Drop out Regularization for 3-Hidden Layers

Discussions & Conclusion:

Intel Image Classification was done using Multi-Layer Perceptron (MLP) fully connected feedforward algorithm. Many models were formed by changing the hyperparameters of the algorithm and using by improvement and regularization technique and their performance was discussed. Model with neural architecture 10000-8700-8700-6, 10000-2000-300-300-128-6, 10000-3000-2000-6 performed well with Adaptive learning rate, early stopping and dropout regularization techniques. We were able to achieve a maximum accuracy of 0.545 where the model was able to correctly classify 1604 unseen data out of 3000.

Color of an image is not a significant feature for image classification. The raw images belonging to 6 categories are colored images of size 150x150. We converted the images into grayscale and got better performance with less computational time rather than training with colored images.

Decreasing learning rate closer to algorithms convergence yields better model performance. Adaptive Learning rate proves to be an efficient technique in improving the performance of the models using gradient descent optimization. Learning rate determines the rate of convergence of gradient descent algorithms. In general, choosing a higher learning rate initially and reducing it while the algorithm converges if efficient.

Performing dropout regularization after the first hidden layer increases the chance of missing significant features for classification. In case of multi-layer algorithms with large no of neurons dropouts are helpful retaining valuable connection however it also omits features.

Future Suggestions

Convolutional Neural Networks can be used for this image classification to obtain accuracy above 0.9 as they extract features effectively compared to MLPs. First few layers for the CNN are exclusively used for feature extraction.

Time Constraints

Remove the confusing images to improve the performance of algorithm. We wanted to notice the most misclassified images at every model and remove them. For example, certain images contain both street and buildings and some has forest in the background. When two or more classes are present in one image then it become difficult to classify. Those kinds of images can be removed based on probability of misclassification.

We wanted to use pretrained weight values from ImageNet datasets classification models and observe the performance of models. Though the inputs datasets are different the goal of the algorithms are same to extract features and classify images into proper categories.

We also wanted to try Back Propagation Algorithm and see the model’s performance, computation time and complexity.

Issues

Data Augmentation is a regularization technique which is used to increase the training data points by doing some edits on the original image. The function basically does the cropping, flipping, zooming, rotating image at specific angles of a training images and adds them to the training set. This is an efficient way of increasing datapoints. We were able to do the augmentation on the training data, but we were not able to train the network due to memory constrains. The code was not able to allocate memory to perform the training with the available hardware. Below are some of the images after augmentation.

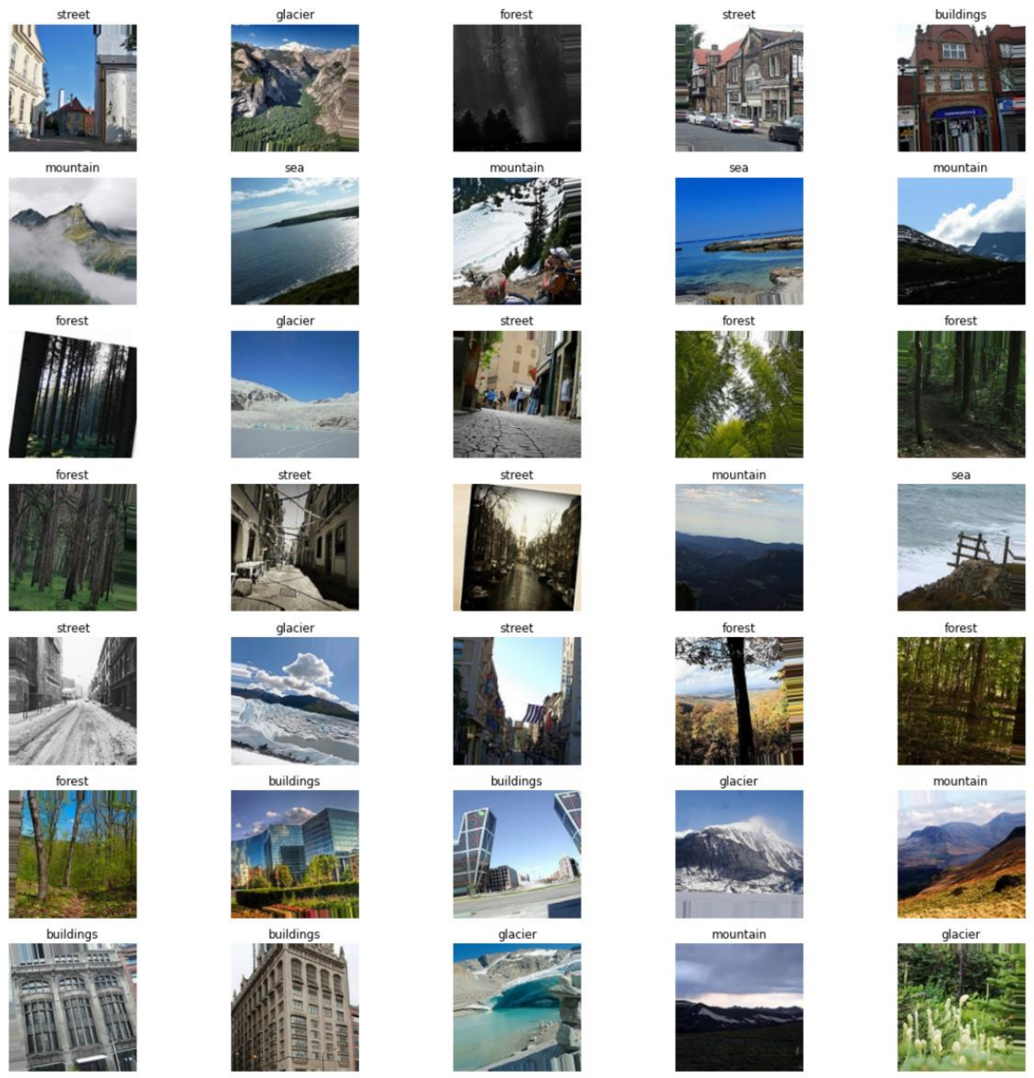


Fig 22 Training Images after Data Augmentation

Code References

We referred the following sites for writing the code.

[1] Data loading - https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/

This site explains in detail to load the data from local directory as previously we were using only the inbuilt data frames or csv files.

[2] Cross Validation functions - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

We referred the scikit learn to adopt the cross-validation techniques. All the functions are explained in detail with reference to the input arguments to be used and their types.

[3] Adaptive Learning Rate: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

TowardsDataScience always gives a valuable search result for deep leaning related queries. Learning rate scheduler with step decay was referred from this site.

[4]Confusion Matrix - https://www.kaggle.com/grfiv4/plot-a-confusion-matrix We took the code for creating the confusion matrix from this site.

[5] Data Augmentation: https://www.kaggle.com/vchauhanusf/90-intel-image-classification We referred to Kaggle for the data augmentation implementation.