1 Introduction

In this assignment, we have a classification problem with convolutional neural networks. We have 3 different dataset which are validation, train and test. Additionally, we have 10 separate class in each dataset. We have 3 parts in this assignment. In each part we get different experiences.

In part I, we will use one of the pretrained model of convolutional neural network, which is VGG-16, as feature extractor without trained the model. VGG16 is one of the strongest and deep model. To extract features we delete some layers so we take 1x4096 vectors for each image both for train and test from FC-7 layer after ReLU. After we extract features, we will use one-vs-the-rest multi-class linear support vector machine to classify.

In part II, we will just use the VGG-16 model and fine tune it that means we just train some of the layers not all. We compare the top 1 and top 5 accuracy for different situations like different batch size, learning rates, epoch size etc.

In part III, we repeat the same thing like in part I but this time we use the model after train it.

The rest of the papers as follows, The method I followed and details of the solution in Section 2, Experimental results for different parameters and comments on them in Section 3, the weakness of my implementation in Section 4 and references in Section 4.

2 Implementation Details

In part I, after I load the dataset, I call the VGG-16 model which is pretrained on imageNet dataset and remove the last 2 layers to extract features whose are 1x4096 vectors. Because of we should not train the model again in this part, I freeze the all of parameters. After that, I send the multiclass images in train and test dataset to the model one by one and I store the extracted features and labels in arrays. Then I send these arrays to one-vs-rest multiclass linear support vector machine as parameteres and I plot the confusion matrix of the predictions on test set. During implementation of this part, I face with some out of memory errors because the extracted features are so big and we have too much data. To solve this problem, after the feature extracted for one image, I immediately change its type to the numpy array instead of tensor.

In part II, we fine tune the pretrained VGG-16 model on imageNet dataset. Fine tuning is made to increase the accuracy. To do that, we take weights of a model that is already trained on a dataset and retrain the model. It is possible to retrain all of the model again or retrain just few of the layers. This method increase the accuracy because instead of starting the weights of the model randomly, we start the model with meaningful weights. Additionally, to decrease the running time, we can freeze some of the layers and just retrain the rest especially fully connected layers. Convolutional Neural Network layers are more generic in convolutional layers which extract the features and more original dataset specific in fully connected layers which make the classification so we usually just retrain the fully connected

layers. After we retrain the fully connected layers, we test the model and plot the train and validation loss/accuracy plots and we will observe the effect of the hyperparameters.(see section 3). In addition these, because of imageNet has 1000 classes and we have just 10 classes, we modify the last layer of the VGG-16 model.

In part III, we follow the same method with part I but this time we retain the VGG-16 model. After that, we delete the last 2 layers, we extract the features that are 1x4096 vectors. We again use one-vs-rest multiclass linear support vector machine to classify images. In this part, I again face with out of memory error that says your ram is full and also after I trained model GPU, I cannot extract features. So I try to train model on CPU but it takes more time.

3 Experimental Results

3.1 Part-I Experiences

To experiment effects of different situations, I run the code over and over again.

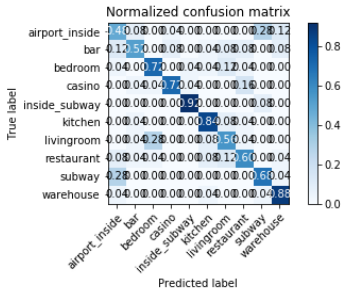
Name	Batch Size = 16	Batch Size = 32
Airport Inside	4.8	6.4
Bar	5.2	5.2
Bedroom	7.2	7.2
Casino	7.2	7.6
Inside Subway	9.2	8.4
Kitchen	8.4	8.8
Living room	5.6	6.8
Restaurant	6	7.2
Subway	6.8	7.2
Warehouse	8.8	9.2
General Accuracy	69.2	74

Table 1: The average and class based accuracy according to different batch sizes for one-to-rest multiclass linear SVM

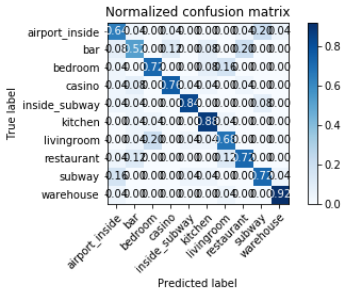
Batch size has several effects in many ways. The batch size determine the number of samples to process so the batch size must be between 1 and total number of samples. When I take the batch size large, I come up with memory error. That means resources is not enough to process that much sample at the same time. On the other hand if the batch size is so small the running time increase. So to find the optimum batch size I try several batch sizes 4, 16 and 32. But for batch size 4, the running time is too long and for batch size 16 and 32 we see the results on 1. We see that when batch size increase, we take a better accuracy result and when we

compare the class based accuracy, just accuracy of "Inside Subway" class decrease a bit when batch size increase.

Not: In table 1, just batch size are different from each other and the random state parameter of the SVM is 0.



(a) Confusion matrix
$(batchsize = 16; random\_state = 0)$



(b) Confusion matrix
$(batchsize = 32; random\_state = 0)$

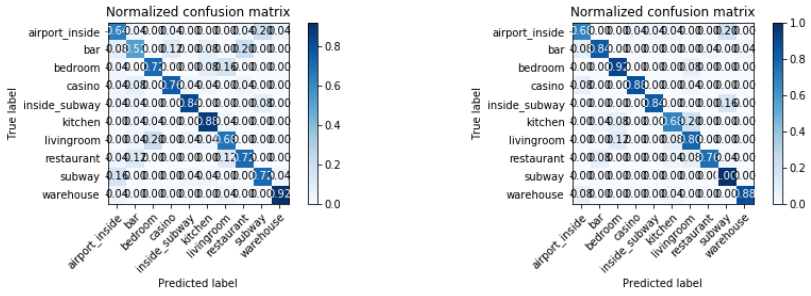
Figure 1: Confusion Matrices

In figure 1, we see the confusion matrices of the experiments(see table 1). Although we have some bad results because we do not train the model again and the imageNet dataset and our own dataset are completely different from each other and also there may be some images that are so similar to each other but belong the different classes, mostly model can make true predictions.

Name	Random State = 0	Random State = 1
Airport Inside	6.4	6
Bar	5.2	6.8
Bedroom	7.2	7.2
Casino	7.6	7.6
Inside Subway	8.4	8.8
Kitchen	8.8	8.8
Living room	6.8	4.8
Restaurant	7.2	5.6
Subway	7.2	7.2
Warehouse	9.2	8.8
General Accuracy	74	71.6

Table 2: The average and class based accuracy according to random state for one-to-rest multiclass linear SVM

The random state is a parameter of the support vector machine and determine the random seed passed to the pseudo-random number generator. As we see in table 2, when we make the Random State 1, the accuracy decrease. In addition, when Random State is any integer except 0, it produces the same class based and general accuracy but when it is 0, the result changes every time.



(a) Confusion matrix
$(batchsize = 32; random\_state = 0)$

(b) Confusion matrix
$(batchsize = 32; random\_state = 1)$

Figure 2: Confusion Matrices

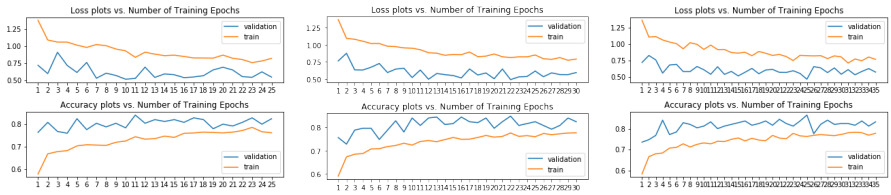
The figure 2 shows the confusion matrices of the table 2. Both have the same batch size which is 32 but different random state values. When we look at the confusion matrices we see that the first confusion matrix, whose random state is 0, makes more accurate prediction than the other.

3.2 Part-II Experiences

To get more accuracy, we should determine the optimum hyper-parameters for our model but this cannot be determined at the first time so we should try different values. To get the best accuracy, I try different situations.

Accuracy	# of epoch = 25	# of epoch = 30	# of epoch = 35
Top-1	80	85	83
Top-5	98	99	99

Table 3: Top-1 and Top-5 accuracy for different number of epochs
$(batchsize = 32; lr = 0.001)$



(a) Epoch $= 25$ $(batchsize = 32; lr = 0.001)$

(b) Epoch $= 30$ $(batchsize = 32; lr = 0.001)$

(c) Epoch $= 35$ $(batchsize = 32; lr = 0.001)$

Figure 3: Validation and Train loss/accuracy plots for different epochs

In table 3, we see the top-1 and top-5 accuracy and in figure 3 we see the validation and train loss/accuracy plots for different number of epochs. To find the top-1 accuracy, we just look at predicted class for the test image and for top-5 accuracy we look 5 class that has the most probabilities. If the true label in these 5 class we assume "true predicted", otherwise "wrong predicted". Because of we have 10 different class, we give the model %50 chance. When we look at the figure 3, we see that there is no overfitting or underfitting problem and the model can learn and in table 3, we see that we get the most top-1 accuracy when epoch number is equal to 30 and we cannot generalize the model according to epoch number. There may be different reasons such that:

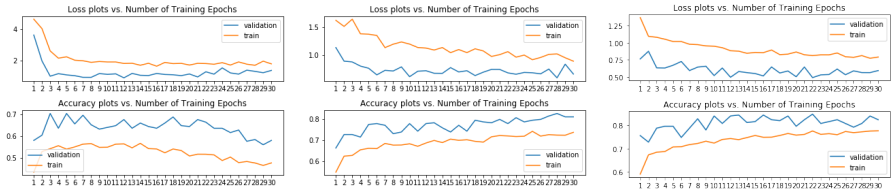
- We take the batches from shuffled training set so average is taken during back-propagation. In each run, we come up with different combination of the samples so it may affect the accuracy.
- We modify the last layer of the model according to our number of class. The weights and biases are set randomly in each run so it also affects the accuracy.

Not: Because of we get the best accuracy when epoch is equal to 30, we use it as base score and compare all the other experiments with this score. Briefly our base hyperparameters and top 1 - top 5 accuracy ;

- $epoch = 30$
- $lr = 0.001$
- $batch\_size = 32$
- $Top\_1 = \%85$
- $Top\_5 = \%99$

Accuracy	Batch Size = 4	Batch Size = 16	Batch Size = 32
Top-1	71	83	85
Top-5	96	98	99

Table 4: Top-1 and Top-5 accuracy for different batch sizes
$(epoch = 30; lr = 0.001)$



(a) Batch Size = 4
$(epoch = 30; lr = 0.001)$

(b) Batch Size = 16
$(epoch = 30; lr = 0.001)$

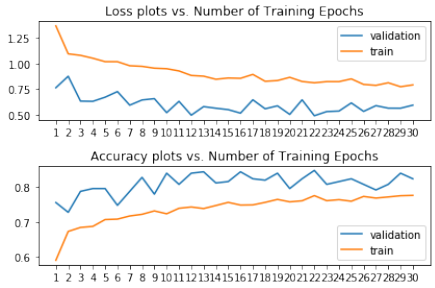
(c) Batch Size = 32
$(epoch = 30; lr = 0.001)$

Figure 4: Validation and Train loss/accuracy plots for different batch sizes

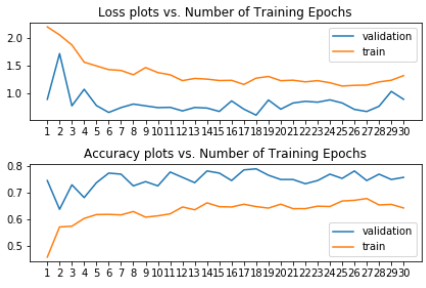
Batch size determine the number of samples that processed in the same time. When we take it too large, we come up with an out of memory error but the running time decrease. On the other hand, when we take it too small, the running time increase but it requires less memory. So we have a trade-off between the memory and running time. To select the best batch size, we try different batch sizes which are 4, 16 and 32. When we look at the table 4, we see that we get the best top-1 and top-5 accuracy when batch size equal to 32. Additionally, we can say that the model can learn the weights and generalize for all situations from figure 4.

Accuracy	Learning Rate = 0.001	Learning Rate = 0.002	Learning Rate = 0.003	Learning Rate = 0.1
Top-1	85	83	73	13
Top-5	99	98	96	55

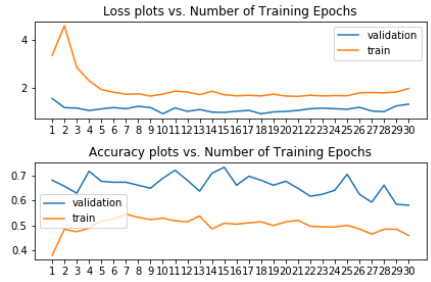
Table 5: Top-1 and Top-5 accuracy for different learning rates
$(epoch = 30; batchsize : 32)$



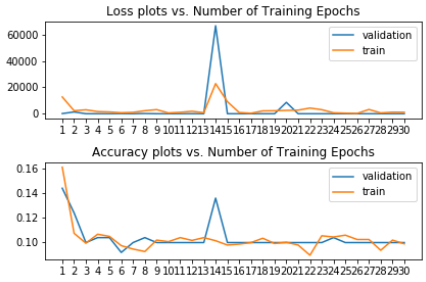
(a) Learning Rate = 0.001
$(epoch = 30; batch\_size = 32)$



(b) Learning Rate = 0.002
$(epoch = 30; batch\_size = 32)$



(c) Learning Rate = 0.003
$(epoch = 30; batch\_size = 32)$



(d) Learning Rate = 0.1
$(epoch = 30; batch\_size = 32)$

Figure 5: Validation and Train loss/accuracy plots for different learning rates

Learning rate is one of the most important hyperparamaters that shows the learning speed.

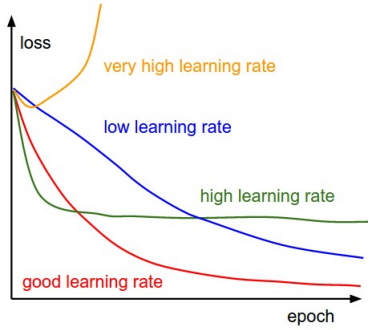


Figure 6: Effect of learning rates in general (image : cs231n)

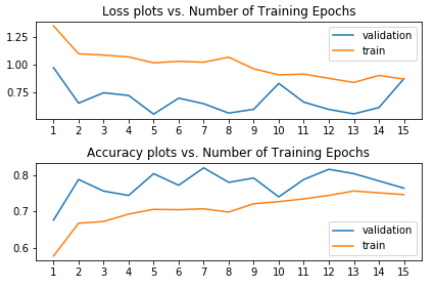
As we see in figure 6, when learning rate is large or low, the model cannot learn very well. Additionally, if the learning rate is high, we may miss the optimum point and if the learning rate is small, the training process will be long so we should decide

an optimum learning rate.

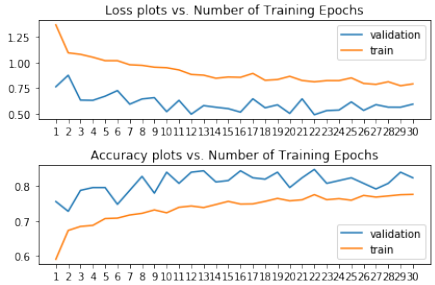
Table 5 and figure 5 shows us the behavior of our model according to different learning rates. When the learning rate is increased, the accuracy both for top-1 and top-5 decrease. We can say that when the learning rate is too high, the model cannot learn properly (lr $= 0.1$) so it cannot predict truly when a new image is given to the model and the best learning rate is 0.001 for our model.

Accuracy	With Early Stopping	Without Early Stopping
Top-1	84	85
Top-5	100	99

Table 6: Top-1 and Top-5 accuracy with/without early stopping($epoch = 30; batchsize : 32; lr = 0.001$)



(a) With Early Stopping ($epoch = 30; batch\_size = 32; lr = 0.001; n\_epochs\_stop = 10$)



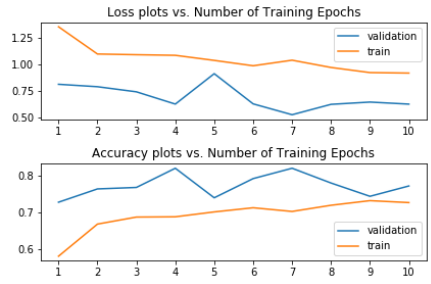
(b) Without Early Stopping ($epoch = 30; batch\_size = 32; lr = 0.001$)

Figure 7: Validation and Train loss/accuracy plots with/without early stopping

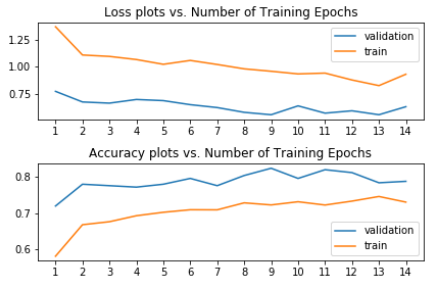
Choosing the number of epoch is critical because if the number of epoch is small, we may come up with an underfitting problem that means the model have not learned yet. On the other hand, if the number of epoch is large, we may come up with an overfitting problem that means the model memorize the train images and cannot generalize so when a new image is given, the model cannot predict the class of the image truly. So to prevent this kind of situations we use early stopping technique. In table 6 and figure 7, we have the results of models with/without early stopping. Although the accuracy of the early stopping should be higher than the other so to see the effect of the n_epochs_stop which is a parameter is used to stop the model if the validation loss does not decrease n_epochs_stop number of epochs.

Accuracy	n_epochs_stop = 3	n_epochs_stop = 5	n_epochs_stop = 6	n_epochs_stop = 10
Top-1	82	83	83	84
Top-5	98	98	99	100

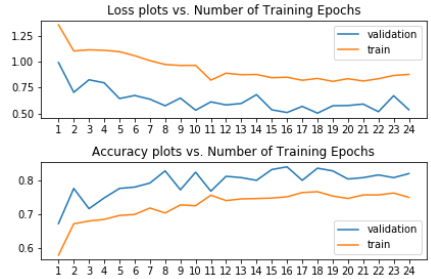
Table 7: Top-1 and Top-5 accuracy for different n_epoch_stop during early stopping $(epoch = 30; batchsize : 32; lr = 0.001)$



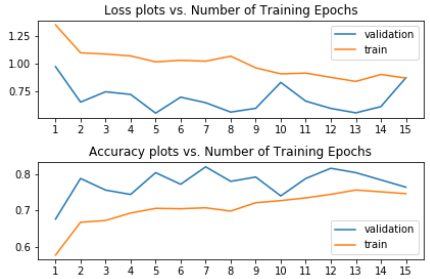
(a) n_epoch_stop = 3
$(epoch = 30; batch\_size = 32; lr = 0.001)$



(b) n_epoch_stop = 5
$(epoch = 30; batch\_size = 32; lr = 0.001)$



(c) n_epoch_stop = 6
$(epoch = 30; batch\_size = 32; lr = 0.001)$



(d) n_epoch_stop = 10
$(epoch = 30; batch\_size = 32; lr = 0.001)$

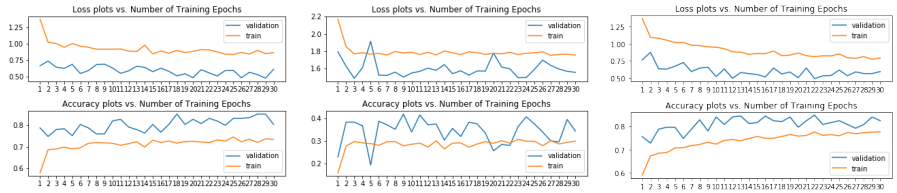
Figure 8: Validation and Train loss/accuracy plots for different n_epoch_stop values during early stopping

When we look at the table 7 and figure 8, we cannot generalize anything. Because in each run:

- We take the batches from shuffled training set so average is taken during back-propagation. In each run, we come up with different combination of the samples so it may affect the accuracy.
- We modify the last layer of the model according to our number of class. The weights and biases are set randomly in each run so it also affects the accuracy.

Accuracy	weight decay = 0.001	weight decay = 0.5	weight decay = 1 (base)
Top-1	82	41	85
Top-5	99	97	99

Table 8: Top-1 and Top-5 accuracy for different weight decays
$(epoch = 30; batchsize : 32; lr = 0.001)$



(a) Weight Decay = 0.001 $(epoch = 30; batch\_size = 32; lr = 0.001)$

(b) Weight Decay = 0.5 $(epoch = 30; batch\_size = 32; lr = 0.001)$

(c) Weight Decay = 1 $(epoch = 30; batch\_size = 32; lr = 0.001)$

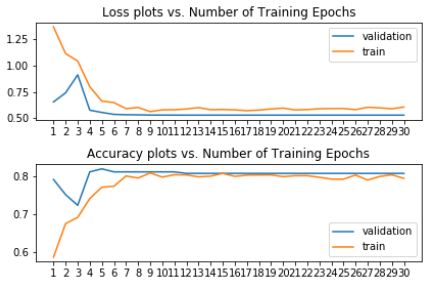
Figure 9: Validation and Train loss/accuracy plots for different weight decays

Weight decay is a regularization term that make the weights multiply by a factor that is less than 1 after each update. The other name of the weight decay is L2 regularization and the main idea behind it, adding an extra term to the cost function to reduce the loss.

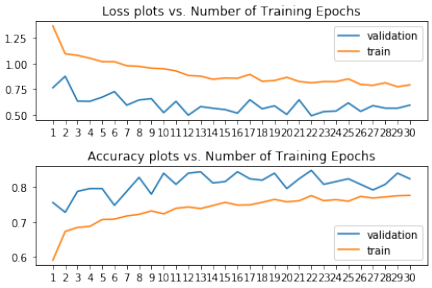
To see the effect of the weight decay, we try different values for the weight decay.(see table 8 and fig. 9.) Although we should take the best accuracy with weight decay, we get the best score in our base model whose weight decay is 1. Because as we say before, randomness affect the results. Without any randomness, we may get a better result with weight decay less than 1. Additionally, the other hyperparameters may not be convenient so we should play with them when we add a regularization term.

Accuracy	lr scheduler	constant lr
Top-1	85	85
Top-5	100	99

Table 9: Top-1 and Top-5 accuracy for lr scheduler
$(epoch = 30; batchsize : 32; lr = 0.001)$



(a) With lr scheduler
$(epoch = 30; batch\_size = 32)$



(b) With a constant lr
$(epoch = 30; batch\_size = 32; lr = 0.001)$

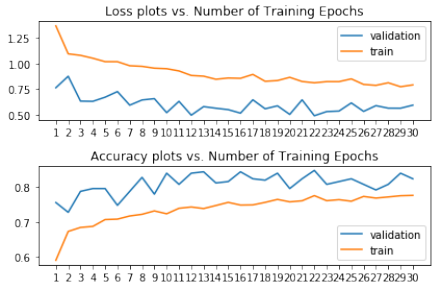
Figure 10: Validation and Train loss/accuracy plots for lr scheduler and constant lr

Adaptive learning rate can increase the performance and decrease the training time. We usually use the adaptive learning rate to reduce the learning rate over time and this provide the model learn quicker.

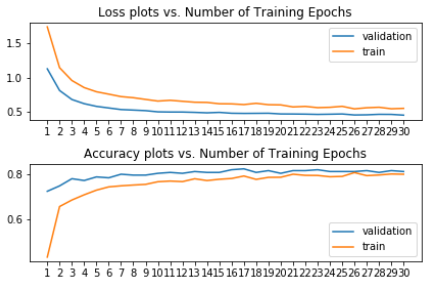
In table 9 and figure 10, we examine the effect of the learning rate scheduler. Although the top-1 accuracy coming the same for both situation, the top-5 accuracy better when we use adaptive learning rate. (see table 9). And in the figure 10, although the top-1 accuracy is same for both, the train and loss plots are more proper when we used scheduler. In constant learning, we have some oscillations for validation.

Accuracy	Adam optimizer	SGD optimizer
Top-1	85	84
Top-5	99	98

Table 10: Top-1 and Top-5 accuracy for different optimizers
$(epoch = 30; batchsize : 32; lr = 0.001)$



(a) Adam optimizer
$(epoch = 30; batch\_size = 32; lr = 0.001)$



(b) SGD optimizer
$(epoch = 30; batch\_size = 32; lr = 0.001)$

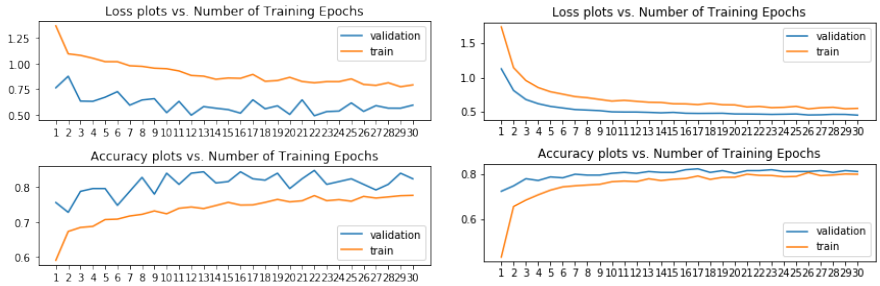
Figure 11: Validation and Train loss/accuracy plots for different optimizers

Optimizer algorithms are used to minimize or maximize the loss. SGD is a variant of gradient descent. It just perform on a small samples instead of all dataset. On the other hand, Adam is a gradient-based optimization algorithm and combines the 2 SGD extensions: RMSProp and AdaGrad.

In table 10 and figure 11, we see the effects of the Adam and SGD. They perform similar accuracy based performance (see table 10) but there are more oscillations in Adam especially for validation set. (see fig. 11). Of course that other hyperparameters may cause the oscillation.

Accuracy	Base VGG-16 model	One more dropout layer
Top-1	85	86
Top-5	99	98

Table 11: Effect of adding one more dropout layer)
$(epoch = 30; batchsize : 32; lr = 0.001)$



(a) Original VGG-16 model $(epoch = 30; batch\_size = 32; lr = 0.001)$

(b) VGG-16 model with one more layer $(epoch = 30; batch\_size = 32; lr = 0.001)$

Figure 12: Validation and Train loss/accuracy plots to see the effect of adding one more dropout layer

Dropout is a regularization technique and used to prevent the network from overfitting. Dropout makes the randomly selected neurons ignore during training.

When we add a one more dropout layer to the beginning of the fully connected layer, we see that top-1 accuracy increase but top-5 accuracy decrease.(see table 11) Additionally, the loss and accuracy plots are better when we add one more layer and losses decrease both for validation and train set. (see fig. 12). Briefly, we can say that adding one more dropout layer affects the model better and we get better performance. To increase accuracy more, we can add more dropout layers and play with other hyperparameters.

Accuracy	Adding Dropout & weight decay
Top-1	84
Top-5	99

Table 12: Effect of weight decay and adding one more dropout layer at the same time)
$(epoch = 30; batchsize : 32; lr = 0.001)$

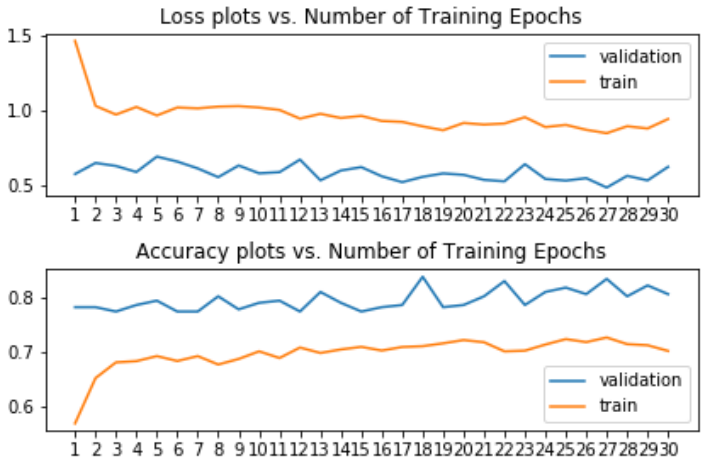
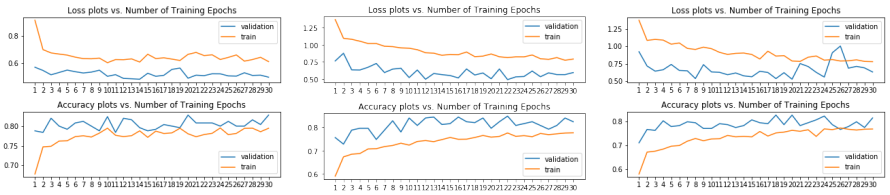


Figure 13: Validation and Train loss/accuracy plots when we apply weight decay and add one more dropout layer at the same time
$(epoch = 30; batch\_size = 32; lr = 0.001)$

When we apply the weight decay and add one more dropout layer at the same time we get a better result from the experiment that we just apply weight decay (see table 8 and figure 9) but worse than the experiment that we add dropout layer.(see table 11 and figure 12) and there are some oscillations on the plots. (see fig. 13)

Accuracy	Situation 1	Situation 2	Situation 3
Top-1	82	85	88
Top-5	100	99	98

Table 13: Top-1 and Top-5 accuracy for different situations.
(Situation 1: Retrain the last layer of FC; Situation 2: Retrain all FC layers; Situation 3: Retrain the all FC layers & last 3 conv layers)
$(epoch = 30; batchsize : 32; lr = 0.001)$



(a) Situation 1 ($epoch = 30; batch\_size = 32; lr = 0.001$)

(b) Situation 2 ($epoch = 30; batch\_size = 32; lr = 0.001$)

(c) Situation 3 ($epoch = 30; batch\_size = 32; lr = 0.001$)

Figure 14: Validation and Train loss/accuracy plots for different number of layers freezing *(Situation 1: Retrain the last layer of FC; Situation 2: Retrain all FC layers; Situation 3: Retrain the all FC layers & last 3 conv layers)*

From beginning to now, we just retrain the fully connected layers of the model. To see the effect of the number of retrained layers, we retrain different number of layers. Looking at the table 13, we can say that the more layers we retrain, the accuracy increase because we use a pretrained model on imageNet dataset and classes of the imageNet are different from our own dataset. So to increase the accuracy and to make the weights of the model more convenient to our classes, we should retrain the model. Increasing the number of training layers means that make the weights of the model more convenient and take make the classification more correctly.

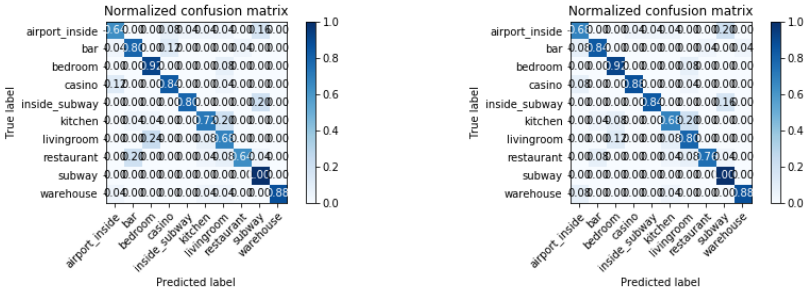
Briefly, we can say that although we get similar results for each experiment, I think that we cannot observe the effects because of the shuffling dataset and the randomness on the last layer (that we modify) of the model in each run and the effects of the other hyperparameters should be observed for each experiment but because of the time restrictions, we cannot do. Additionally, the classes of imageNet dataset are different from our own dataset to get more accuracy we may use MIT Places dataset [1] that is more convenient with out own dataset and we may collect more dataset and class.

3.3 Part-III Experiences

In this part, we will repeat the part-I but this time we retrain the model. I use the model, that gives me the best top-I accuracy in part-II, whose more layers are retrained. (see table 13 and figure 14.)

Name	Part I	Part III
Airport Inside	6.4	6.8
Bar	5.2	8.4
Bedroom	7.2	9.2
Casino	7.6	8.8
Inside Subway	8.4	8.4
Kitchen	8.8	6.8
Living room	6.8	8
Restaurant	7.2	7.6
Subway	7.2	10
Warehouse	9.2	8.8
General Accuracy	74	82.8

Table 14: Comparison of the part-I and part-III



(a) Confusion matrix of the part I $(batchsize = 32; random\_state = 0)$

(b) Confusion matrix of the part-III $(batchsize = 32; random\_state = 0)$

Figure 15: Confusion Matrices

In table 14, we see the comparison of accuracy of the part-I and part-III. We take the best scores in both parts and we can see that after fine-tunning, the SVM generally gives better accuracy both for general and class based. We retrain the model in part-III so the extracted features are more accurate than the part-I. When we give these features to SVM, SVM can make better classification. Additionally, when we look at the confusion matrices (see fig. 15), we see that the faults decrease but for both cases, the model generally predict wrongly the same classes. We can conclude that some images in different classes are similar each other.

4 Conclusion

In each parts, we do several experiments to make the image classification more accurately. When we compare all parts, we get the highest accuracy in part-II so we can say that convolutional neural networks are better than SVM to classify images. (see table 13 and figure 14.) Additionally, there are some weakness that makes the accuracy lesser and comparisons harder :

- We take the batches from shuffled training set so average is taken during back-propagation. In each run, we come up with different combination of the samples so it may affect the accuracy.
- We modify the last layer of the model according to our number of class. The weights and biases are set randomly in each run so it also affects the accuracy.
- We have small dataset.
- The class of the ImageNet is different from our dataset.
- Insufficient resources like memory etc. To get more accuracy we should run the model using more number of epoch, batch size etc but we cannot because of the insufficient resources.

So as a future work:

- We can collect more dataset.
- We get sufficient resources like taking more RAM etc.
- We can play more with hyperparameters
- We can use a deeper neural network from the VGG-16.
- We can use MIT places dataset [1] instead of imageNet.
- We can use different classification algorithms.
- We can use image processing techniques like edge detection etc.

Finally, during this process I faced with some problems. I run the part-I in CPU but part-II and part-III in GPU so part-I takes longer time than the others and additionally making part-III in GPU enforce me because of moving the parameters and all other things to GPU. Additionally, in part-I and part-III, I face with some out of memory error because of the tensor arrays.

References

[1] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.