**TECHNICAL REPORT**

|  |
| --- |
| **Date#40**  27/03/2018 |
| **Title#15**  **Models for Machine Vision: A How-To Guide – Aditya Prabhakaron** | | | | |
|  | | | | |
| **Summary#60**  This report contains investigations and provides guidelines for developing a model using Convolutional Neural Network architecture for machine vision application. | | | | |
|  | | | | |
| **Additional keywords#90**  Machine Vision, CNN, Deep Learning | | | |  |
|  | |  |  | |
|  | |  |  | |

Contents

[List of Tables 3](#_Toc492904759)

[List of Figures 3](#_Toc492904760)

[Change History 4](#_Toc492904761)

[Definitions 4](#_Toc492904762)

[1 Introduction and Background 5](#_Toc492904764)

[1.1 Scope 5](#_Toc492904765)

[2 Fashion MNIST Dataset 5](#_Toc492904766)

[3 Baseline Model Architecture 7](#_Toc492904767)

[4 Parametric Study of Baseline Model 8](#_Toc492904770)

[4.1 Effect of batch size 8](#_Toc492904771)

[4.2 Effect of number of epochs 9](#_Toc492904772)

[4.3 Effect of drop out 10](#_Toc492904773)

[4.4 Effect of padding 13](#_Toc492904773)

[4.5 Effect of learning rate 15](#_Toc492904773)

[4.6 Effect of convolution filter size 16](#_Toc492904773)

[4.7 Effect of type of pooling 18](#_Toc492904773)

[4.8 Effect of pooling filter size 18](#_Toc492904773)

[4.9 Effect of number of convolution filters 19](#_Toc492904773)

[4.10 Effect of number of hidden layers 19](#_Toc492904773)

[5 Benchmarking baseline model on Fashion MNIST Dataset 22](#_Toc492904774)

[6 NEU Surface Defect Dataset 23](#_Toc492904779)

[7 Benchmarking baseline model on NEU Dataset 23](#_Toc492904780)

[8 Conclusions 24](#_Toc492904781)

[9 References 25](#_Toc492904782)

# List of Tables

[Table 1 Class names and example images in Fashion-MNIST dataset 6](#_Toc492904787)

[Table 2 Baseline model hyperparameter values for benchmarking performance on Fashion MNIST dataset 22](#_Toc492904788)

[Table 3 Benchmarking performance against models quoted in literature on Fashion MNIST dataset 22](#_Toc492904789)

[Table 4 Baseline model hyperparameter values for benchmarking performance on NEU dataset 24](#_Toc492904790)

[Table 5 Benchmarking performance against models quoted in literature on NEU dataset 24](#_Toc492904791)

# List of Figures

[Figure 1: Examples of the Fashion MNIST dataset 7](#_Toc492904797)

[Figure 2: Typical example of CNN architecture as applied to image classification task 8](#_Toc492904798)

[Figure 3: Baseline CNN model architecture 8](#_Toc492904799)

[Figure 4: Variation of training and validation (a) accuracy and (b) loss versus epochs for different batch sizes 9](#_Toc492904800)

[Figure 5: Variation of training and validation (a) accuracy and (b) loss versus epochs 10](#_Toc492904801)

[Figure 6: Variation of training and validation (a) accuracy and (b) loss versus epochs for different values of drop out fractions 12](#_Toc492904802)

[Figure 7: Variation of test (a) accuracy and (b) loss versus drop out fractions 13](#_Toc492904803)

[Figure 8: Variation of training and validation (a) accuracy and (b) loss versus epochs for different padding 14](#_Toc492904804)

[Figure 9: Variation of training and validation (a) accuracy and (b) loss versus epochs for different learning rates 16](#_Toc492904805)

[Figure 10: Variation of training and validation (a) accuracy and (b) loss versus epochs for different convolutional filter sizes 17](#_Toc492904806)

[Figure 11: Variation of training and validation (a) accuracy and (b) loss versus epochs for different pooling filter sizes 19](#_Toc492904807)

[Figure 12: Variation of training and validation (a) accuracy and (b) loss versus epochs for different number of hidden layers 20](#_Toc492904808)

[Figure 13: Variation of test (a) accuracy and (b) loss versus number of hidden layers 21](#_Toc492904809)

[Figure 14: Example images of the six classes of defects in the NEU dataset 23](#_Toc492904810)

# 

# Change History

| Issue | Author | Date | Comment |
| --- | --- | --- | --- |
| 1 | Aditya Prabhakaron | 28/03/2018 | First issue |
|  |  |  |  |

# Definitions

|  |  |
| --- | --- |
| Term/Acronym | Definition |
| CNN | Convolutional Neural Network |
| FCN | Fully connected Layer |
| ILSVRC | ImageNet Large Scale Visual Recognition Challenge |

# Introduction and Background

The Society of Manufacturing Engineers defines Machine Vision as “the use of devices for optical non-contact sensing to automatically receive and interpret an image of a real scene in order to obtain information and/or control machines or processes”. Computer vision refers in broad terms to the capture and automation of image analysis with an emphasis on the image analysis function across a wide range of theoretical and practical applications. Machine vision, on the other hand, refers to the use of computer vision in an industrial or practical application or process where it is necessary to execute a certain function or outcome based on the image analysis done by the vision system.

Machine Vision has many potential applications in manufacturing such as automated inspection during and after manufacture, inspection during after sales service, process monitoring, surveillance of shop floors etc.

Machine Vision is a rapidly evolving field and recent advances have made it promising and worthwhile to invest in this technology. The recent advances in deep learning over the last decade, like AlexNet[1] model which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 and other deep convolutional networks have made great strides in increasing accuracy in tasks like object recognition, feature extraction etc. It is possible to use pre-trained models with weights updated by training on large datasets such as ImageNet to be used for different applications with little modification or tuning required [2][3]. The advent of powerful GPU chips (NVidia Titan or Tesla series for example), with NVidia taking the lead, has made training these models possible in a reasonable amount of time.

The literature broadly consists of two types of approaches for performing object detection tasks: traditional machine learning and deep learning. The traditional machine learning techniques for image recognition typically employed Support Vector Machines or Nearest Neighbour classifiers.

Among deep learning approaches, Convolutional Neural Networks (CNN) is popular for image recognition and related activities. But lack of required computational resources was an obstacle for progress in CNN. However, after AlexNet model won the ILSVRC in 2012 using a deep CNN architecture, there has been renewed interest and research in CNN.

In this report, we explore the adoption of CNN models for the purpose of machine vision.

## Scope

Scope of this report is to provide a How-To Guide for developing Machine Vision capability within RR. A detailed literature survey on Machine Vision capabilities, applications and recent advances is out of the scope of this report. A detailed description of the machine learning algorithms, specifically Convolutional Neural Networks, is also out of the scope of this report.

# Fashion MNIST Dataset

.

Deep Learning models typically have many hyperparameters which require careful tuning to achieve optimal results. For this purpose, it is important to take a sample public dataset to perform the parametric study to develop intuition about these hyperparameters and their influence on the accuracy and loss function of the model.

To this end, the Fashion MNIST dataset of Zalondo [4] has been chosen for the initial study. Despite the handwritten digits dataset of MNIST [5] being very popular, it suffers from the disadvantage that Convolutional Nets can achieve 99.5% on MNIST[5]. Classic machine learning algorithms can also achieve 97% easily [4]. This makes it harder to explore the effects of different parameters on accuracy and is also not representative of the feature detection tasks that will be required on RR datasets.

So as an initial dataset, the Fashion MNIST dataset is used. This dataset contains grayscale images of ten different classes of clothing. Each image is 28X28 pixels. The dataset totally contains 70000 images (7000 in each class).

For this study, 60,000 images have been taken for training and 10,000 images are taken for testing. Among the 60,000 training images, 12,000 images are randomly kept aside as the validation set.

**Table 1: Class names and example images in Fashion-MNIST dataset [4]**

|  |  |
| --- | --- |
| **Label** | **Description** |
| 0 | T-Shirt/Top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandals |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boots |

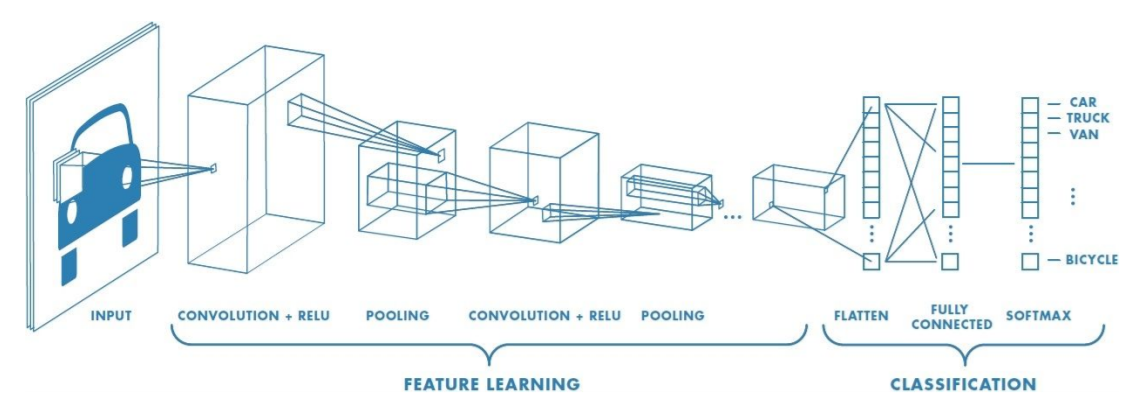


Figure 1: Examples of the Fashion MNIST dataset (each class takes three rows)

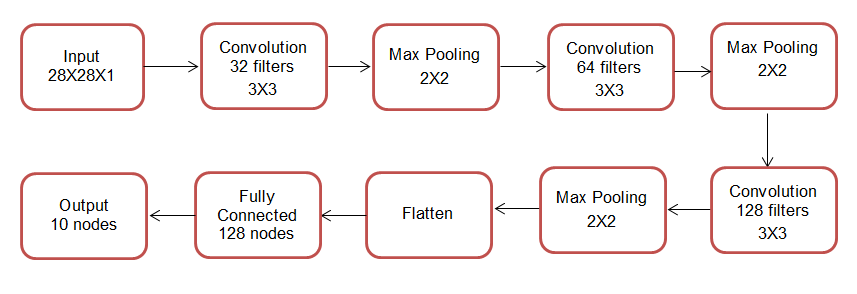
# Baseline Model Architecture

The baseline model adopted for the parametric study uses the CNN architecture. The baseline model has three sets of convolution and pooling layers followed by a fully connected layer and finally the output layer. The first convolution layer has 32 filters with kernel size of (3X3), stride equal to one (since the image size is 28X28, stride greater than 1 was not explored), padding ‘same’ followed by a max pooling layer with kernel size of (2X2) and padding ‘same’. This is followed by two more sets of convolution and max pooling layers with only difference being the number of filters in the convolution layer being double that of the previous convolution layer (i.e. 64 filters in the second convolution layer and 128 filters in the third convolution layer). All the convolution and pooling layers have ReLU as the activation function. The first fully connected layer (FCN) has 128 nodes and ReLU activation function followed by the output FCN with 10 nodes (corresponding to the ten output classes) and softmax activation function. The optimizer used is Adam.

A graphical visualization of a typical CNN architecture is shown in Figure 2[6]. The baseline model architecture is depicted in Figure 3.



**Figure 2[6]**: Typical example of CNN architecture as applied to image classification task.



**Figure 3**: Baseline CNN model architecture

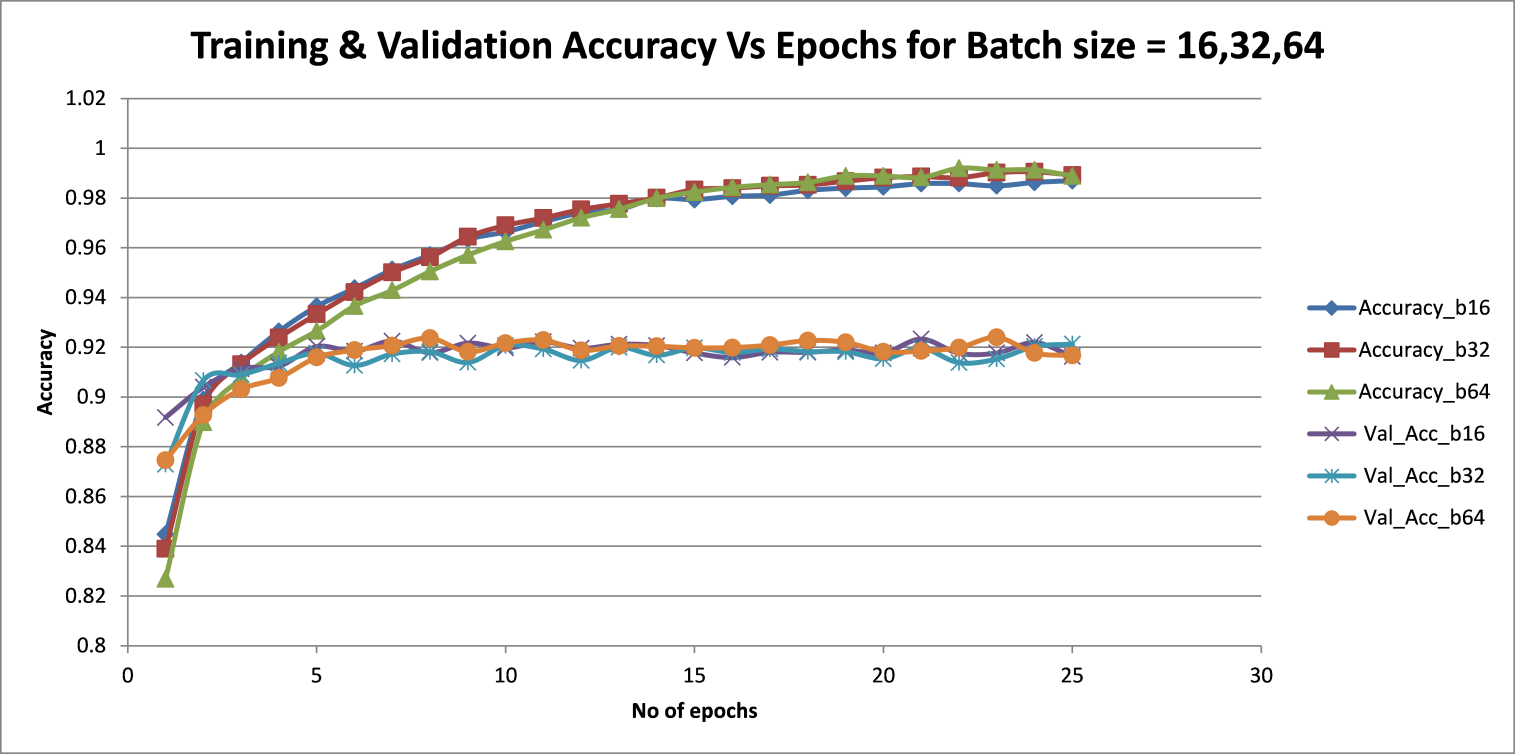
# Parametric Study of Baseline Model

This section details the parametric study of the hyperparameters of the baseline model on the Fashion MNIST dataset. Hyperparameters are parameters of the neural network which require user input as opposed to weights of the neural network which are updated with every iteration of the backpropagation algorithm.

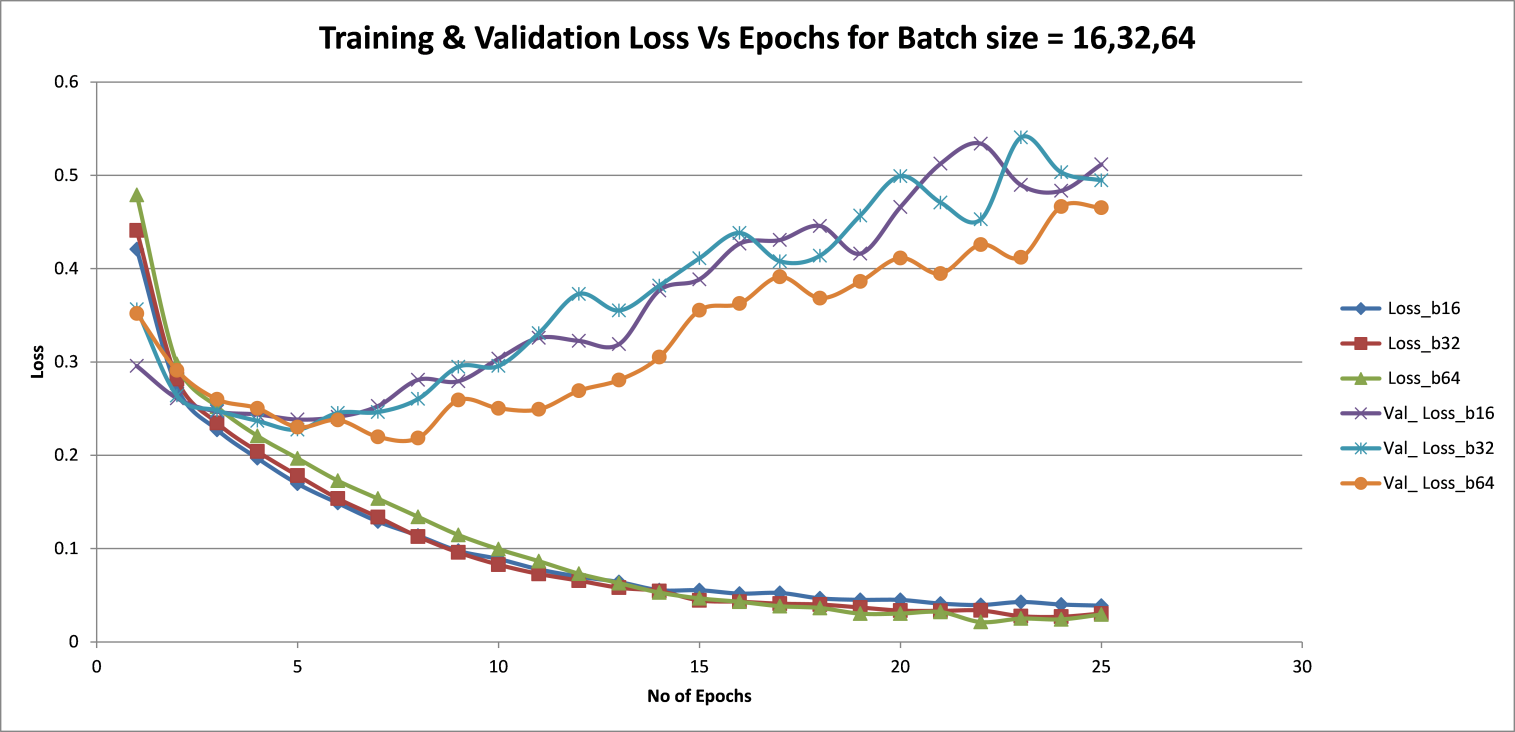
All the model runs were performed on Intel i5-6300 CPU processor. In the interest of resources and practicality, the design space has been chosen from general thumb rules that are commonly adopted by practitioners and serve as a guide for choosing the hyperparameter values.

## Effect of batch size

Batch size refers to the number of images taken at one instance for each calculation of the derivatives and gradient descent of the backpropagation algorithm. Here, the batch size is varied from 16, 32 and 64. The effect of batch size on training, validation and test accuracy and loss is shown in Figure 4. It can be seen that batch size does not have a significant effect on either training and validation accuracy or loss. For subsequent parametric runs, the batch size is taken as 32.



**(a)**

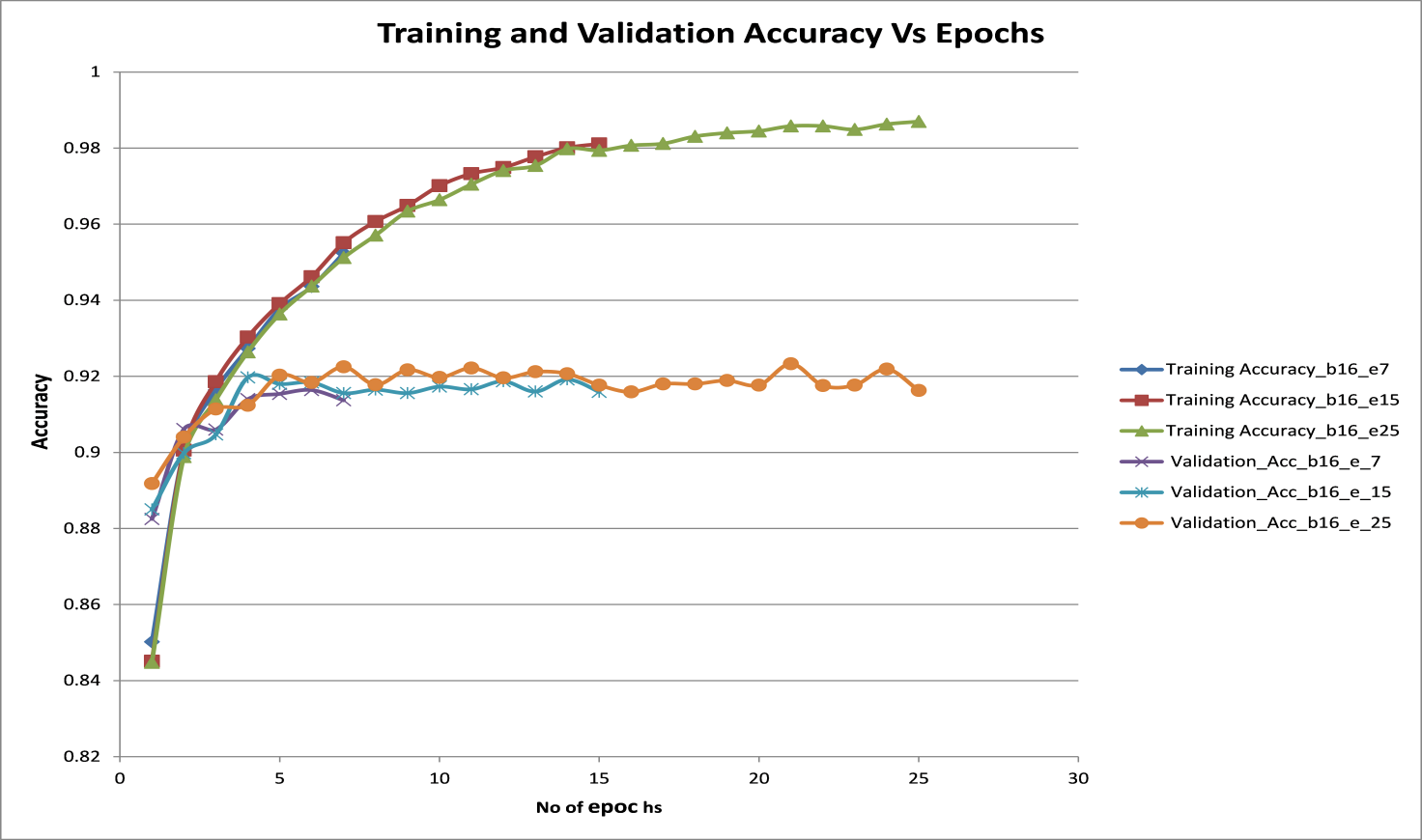


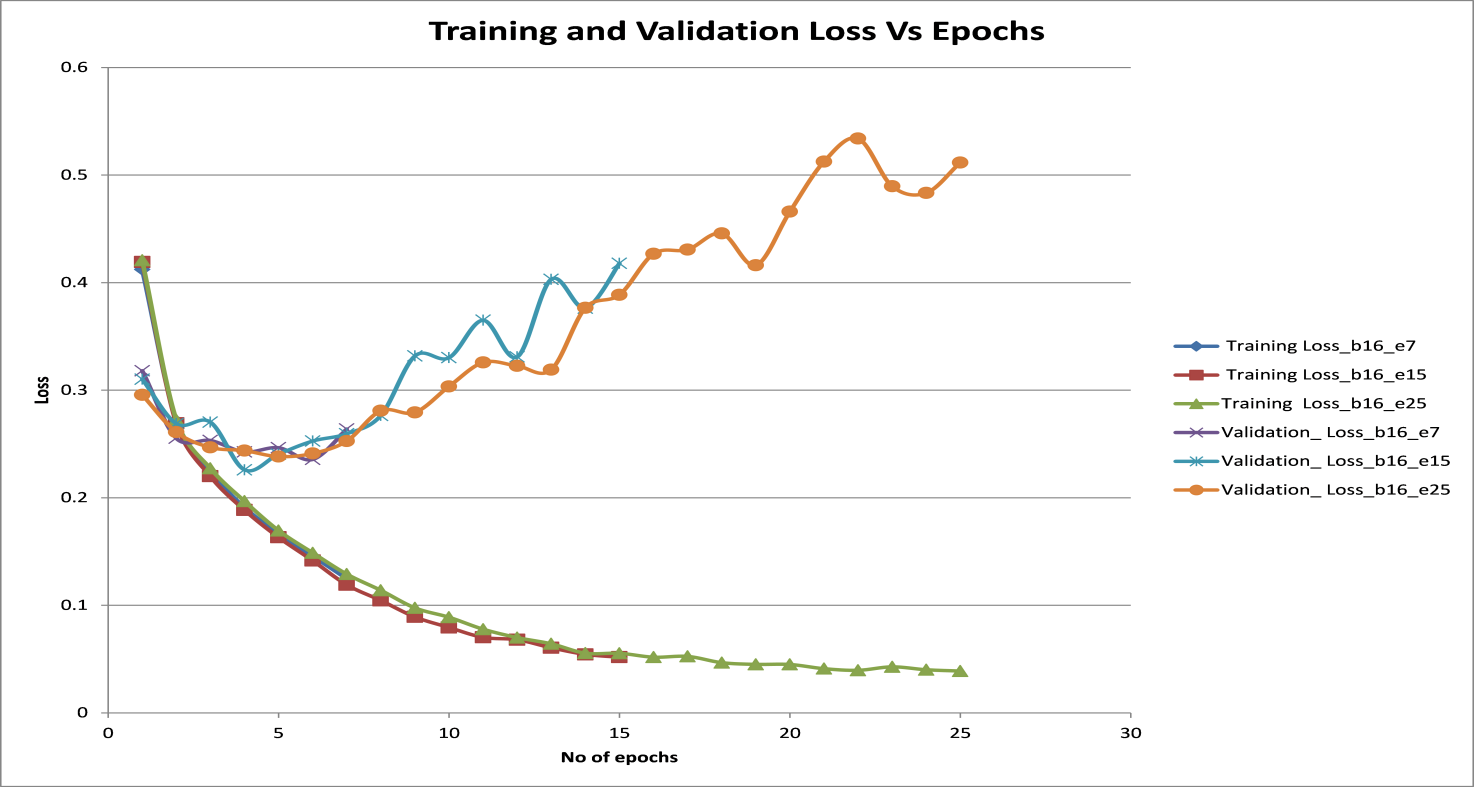
**(b)**

**Figure 4**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different batch sizes

## Effect of number of epochs

Epochs refers to the number of iterations that the model goes through the entire training set. Typically, accuracy increases and loss decreases with number of epochs until they flatten out. From Figure 5, it can be seen that training and validation accuracy continue to increase beyond 7 epochs and begin to flatten out at around 20 epochs. So, 25 epochs were used for parametric runs henceforth. It is also seen that training loss decreases with number of epochs as expected. However, validation loss decreases initially and then increases monotonically with number of epochs. This means that the model is training well but is unable to generalize. It is a symptom of the model suffering from overfitting. To address this, drop out layers are introduced in the model as detailed in section 4.3.





**Figure 5**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs

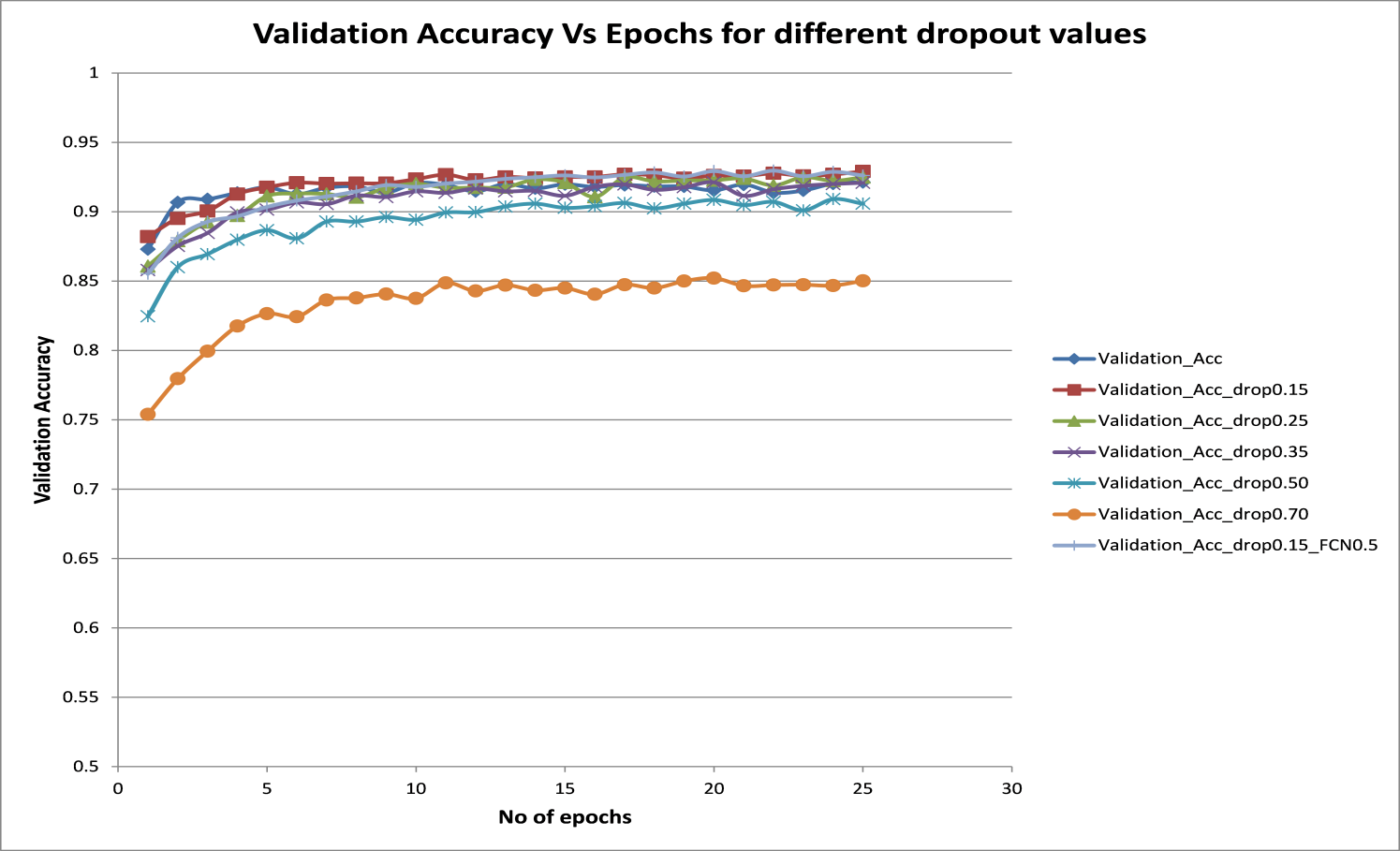
## Effect of drop out

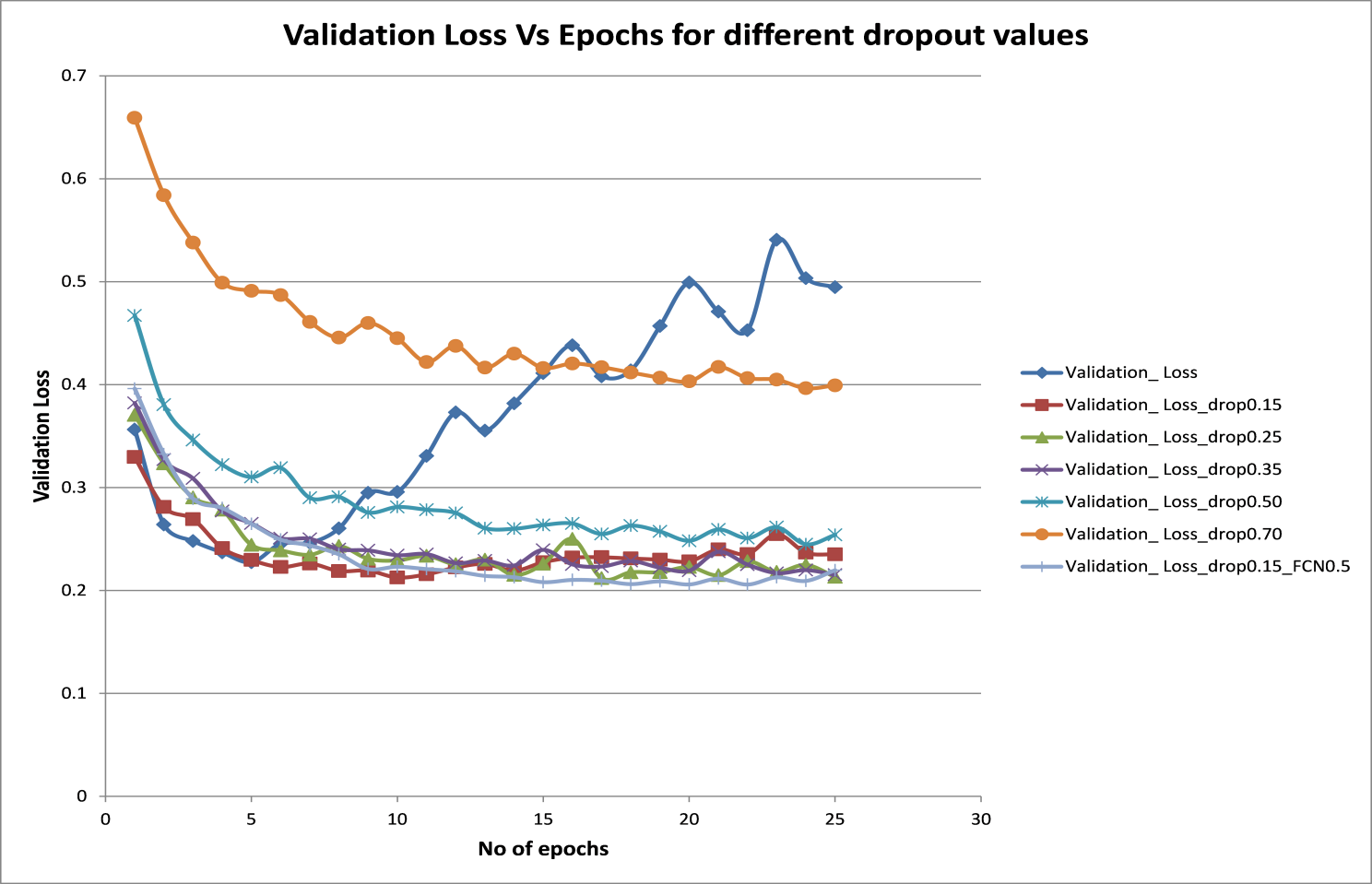
Drop out is a common technique in deep learning to achieve regularization and reduce overfitting [7]. This technique randomly drops neurons and their connections from the neural network during training to prevent them from co-adapting too much [7].

Drop out is generally introduced after the FCN layer since it has high number of neurons and has the maximum amount of co-adaption. It is argued that convolutional layers do not suffer from overfitting because the number of parameters for the convolutional layers is small relative to the number of activations. Nevertheless, dropout in convolutional layer is proven to improve generalization performance in some extent by adding noise to the activation.

Drop out fractions of 0.15, 0.25, 0.35, 0.5 and 0.7 were introduced after each pooling layer and also after the FCN layer. Drop out fractions of 0.15 after every pooling layer and 0.5 after the FCN layer, was also tested.

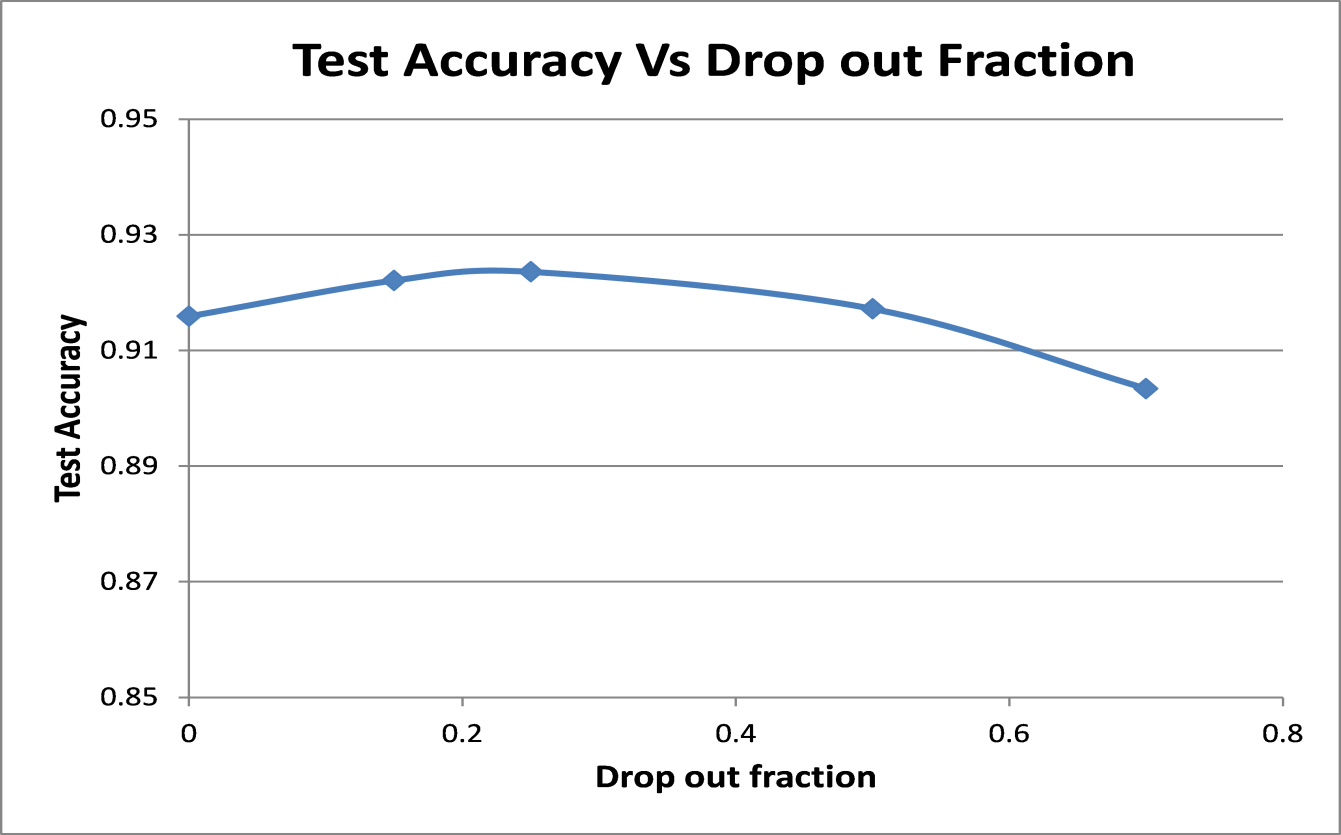
Figure 6 shows that though drop out causes a small decrease in validation accuracy, it successfully causes the validation loss to decrease like the training loss thus reducing overfitting. But higher drop out fractions of 0.5 and 0.7 cause significant drop in validation as well as test accuracy which is undesirable. This is because large drop out fractions result in large number of neurons becoming deactivated or ‘dead’ and hence, result in loss of information. Hence, there is a trade-off between reducing overfitting and loss of information. The plots of test accuracy and test loss versus epochs for different drop out fractions (Figure 7) show that there is an optimal amount of drop out fraction which maximizes accuracy. For subsequent parametric test runs, drop out fraction of 0.25 was taken.

**(a)**

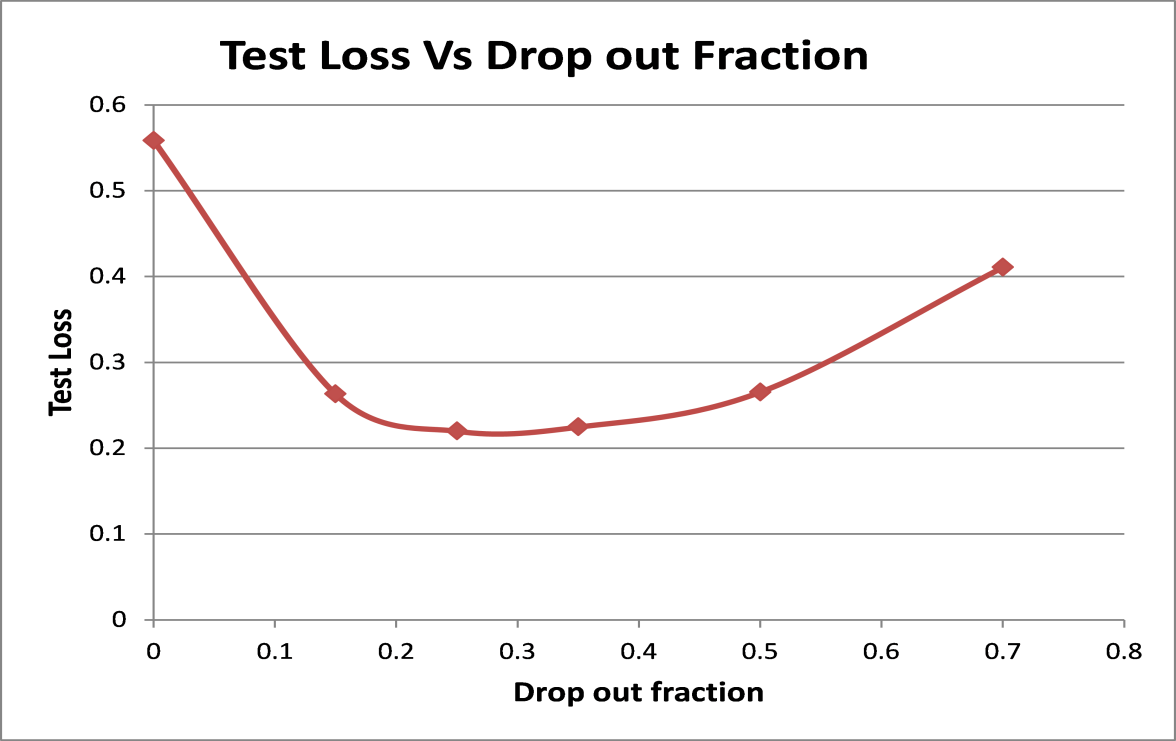


**(b)**

**Figure 6**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different values of drop out fractions



**(a)**

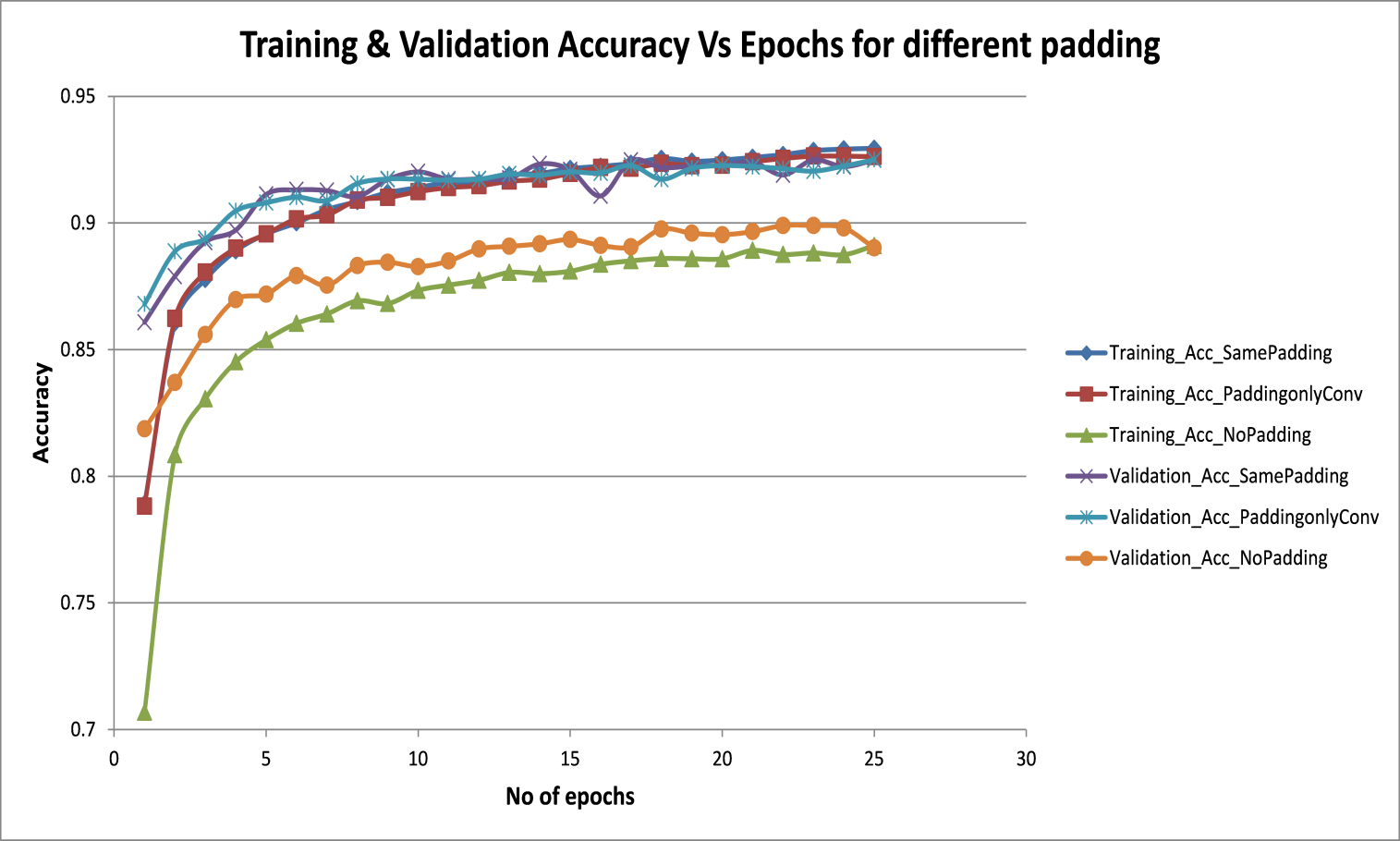


**(b)**

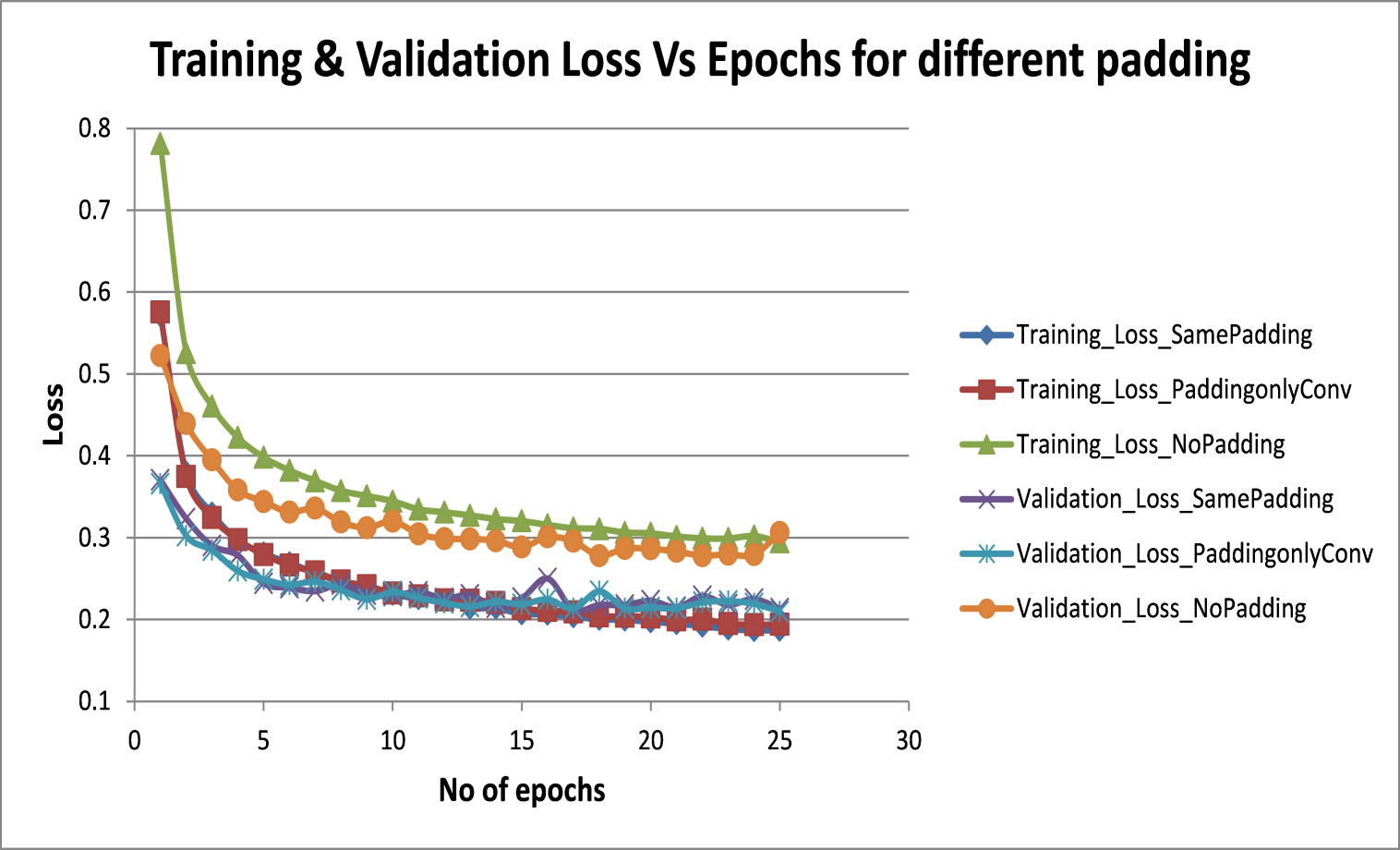
**Figure 7**: Variation of test **(a)** accuracy and **(b)** loss versus drop out fractions

## Effect of padding

In the convolution layer, as the filter convolves over the input volume at that layer, the output volume decreases in size and some information in the corners of the input volume (and hence the image) is not used as much as the other regions of the input. To avoid this, the input volume is padded with zeroes all around to retain the size of the volume whilst not distorting the information contained in the volume during the convolution. Figure 8 shows the effect of padding. ‘valid’ padding refers to no padding and ‘same’ padding refers to zero padding to ensure output volume size is same as input volume size. It can be seen that with no padding the training and validation accuracy decreases and the loss increases. The test accuracy values for ‘same’ padding, ‘same’ padding only for convolution layers (and not for pooling layers) and for no padding (‘valid’) are 0.924, 0.922 and 0.88 respectively. Hence, padding is important to prevent loss of information. For subsequent parametric runs, padding was taken as ‘same’ for all the convolution and pooling layers.



**(a)**



**(b)**

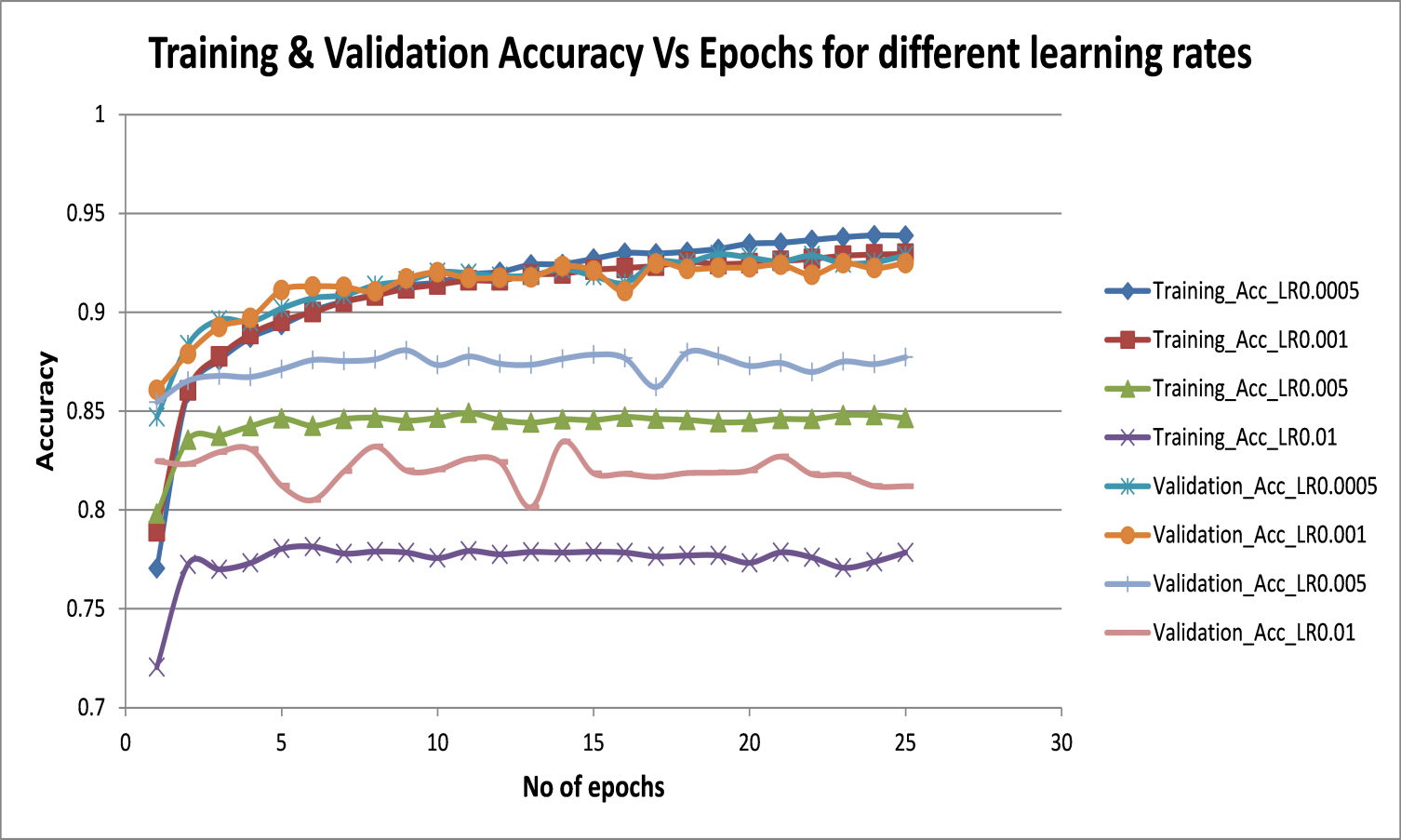
**Figure 8**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different padding

## Effect of learning rate

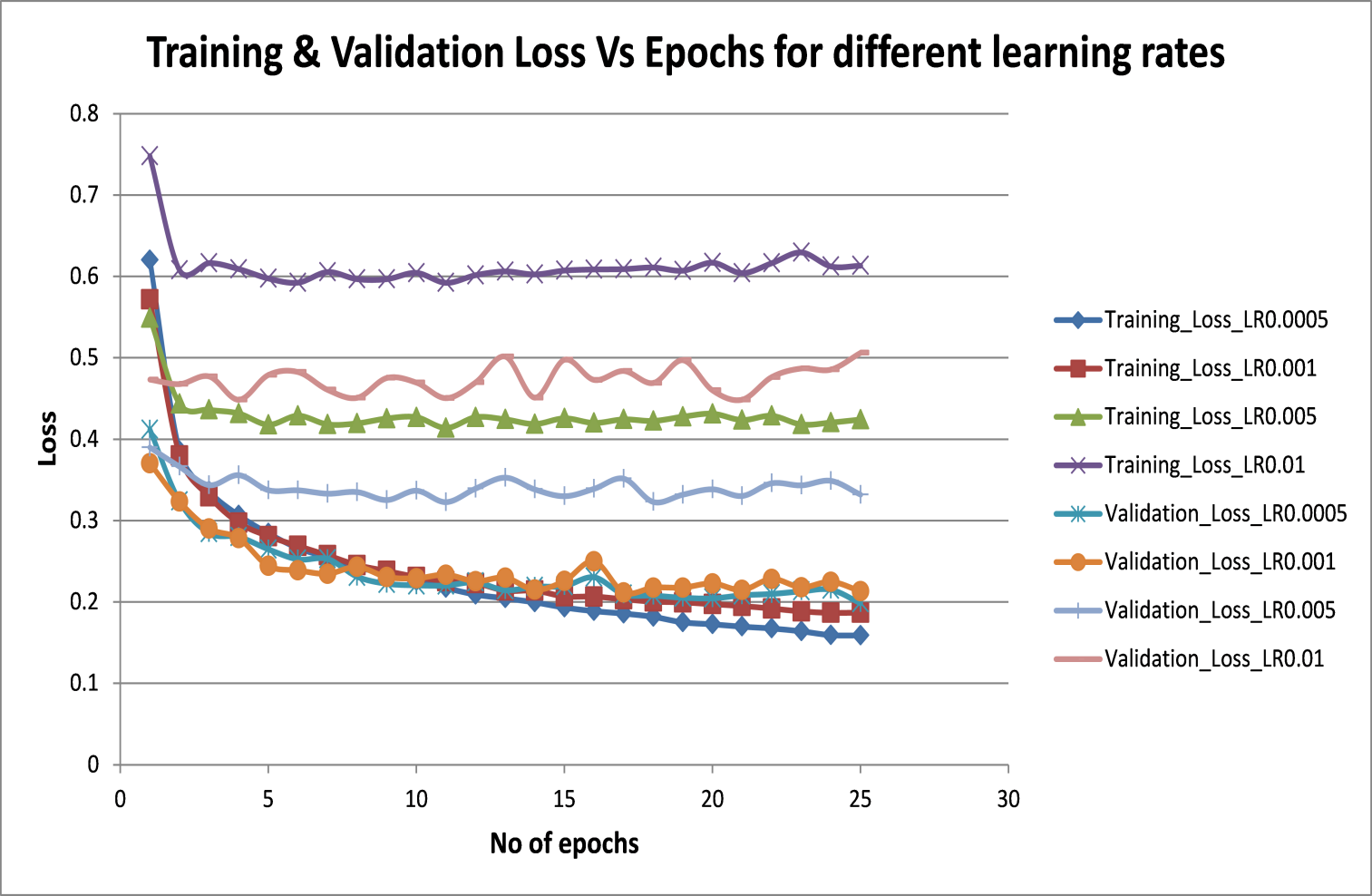
Learning rate decides the amount of change in weights and bias of the neurons during each update of the backpropagation algorithm. Learning rate defines for the optimizer quantum of the jump in weights in the direction of the decreasing gradient for each batch.

If the learning rate is low, then the model is more stable and reliable, however, it takes a long time since the steps towards the minimum loss function are small. If the learning rate is high, then model may not converge itself and may instead diverge during training since the weights may overshoot the minimum point.

Figure 9 shows that higher learning rate of 0.01 and 0.005 decreases the training and validation accuracy and increase the loss function values when compared to learning rate values of 0.001 and 0.0005. This is also reflected in the test accuracy values. Test accuracy for learning rate of 0.0005 and 0.001 are 0.925 and 0.924 respectively. However, they decrease to 0.872 and 0.813 for learning rate of 0.005 and 0.01 respectively.



**(a)**

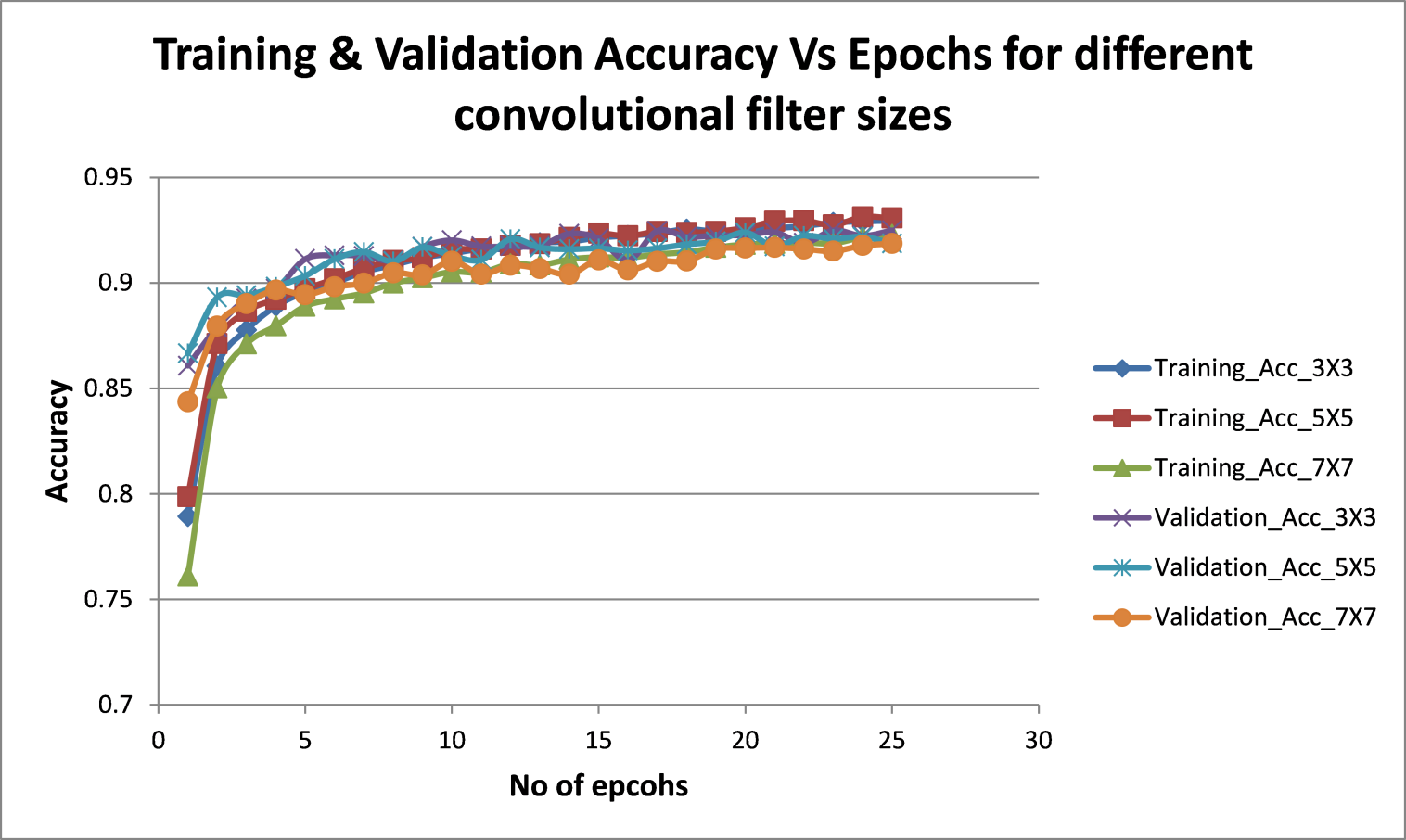


**(b)**

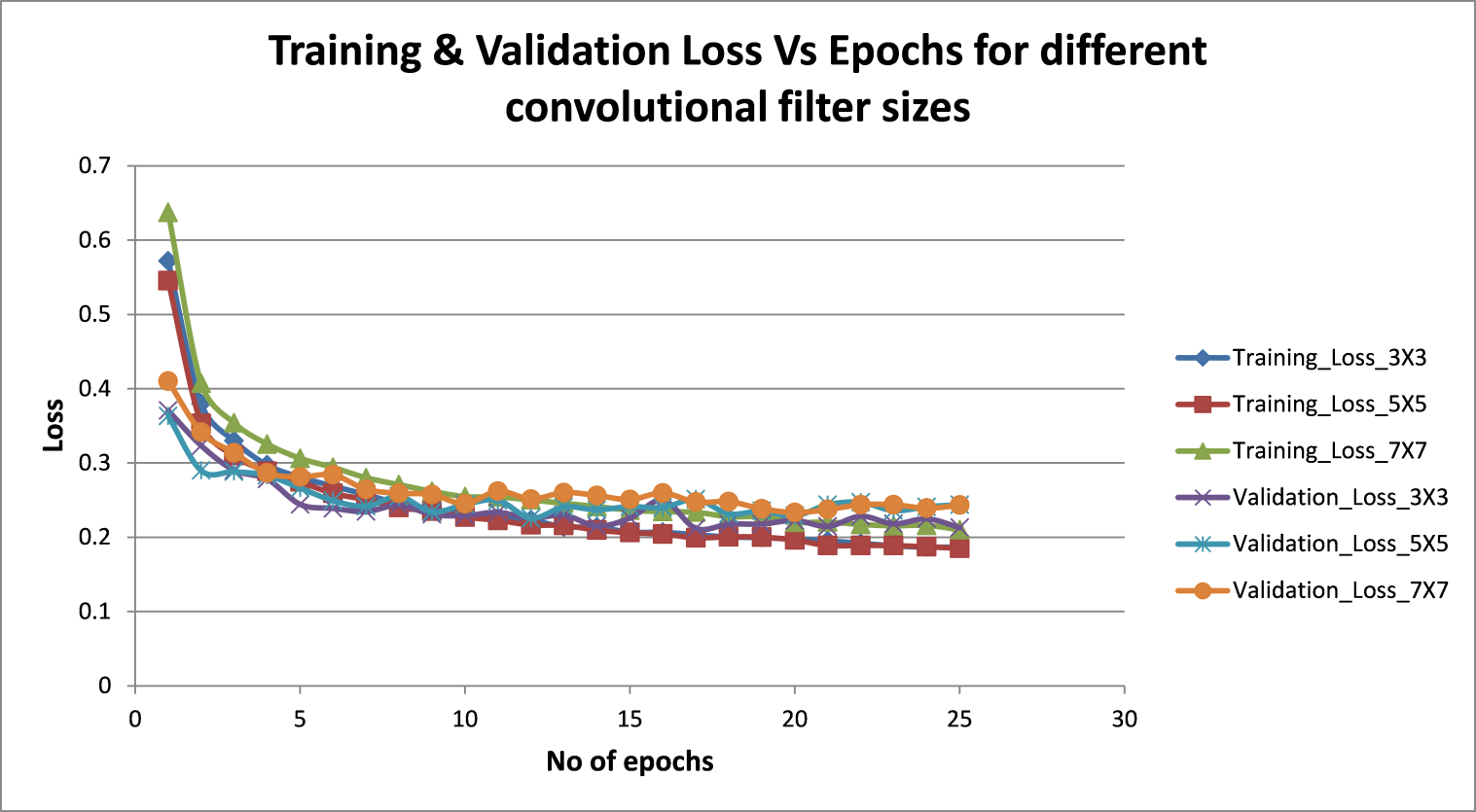
**Figure 9**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different learning rates

## Effect of convolution filter size

The filter size of all the convolution layers were varied with (3X3), (5X5) and (7X7) filters. From Figure 10, it can be seen that as the filter size is increased to 7X7, there is a decrease in training and validation accuracy and increase in loss. Test accuracy also is marginally lower at 0.914 for 7X7 filter as opposed to 0.924 for 3X3 filter. This is expected as for small image sizes (28X28), increasing the convolutional filter size will lead to some loss of information contained in the input. For subsequent parametric runs, the convolutional filter size is taken as 3X3.



**(a)**



**(b)**

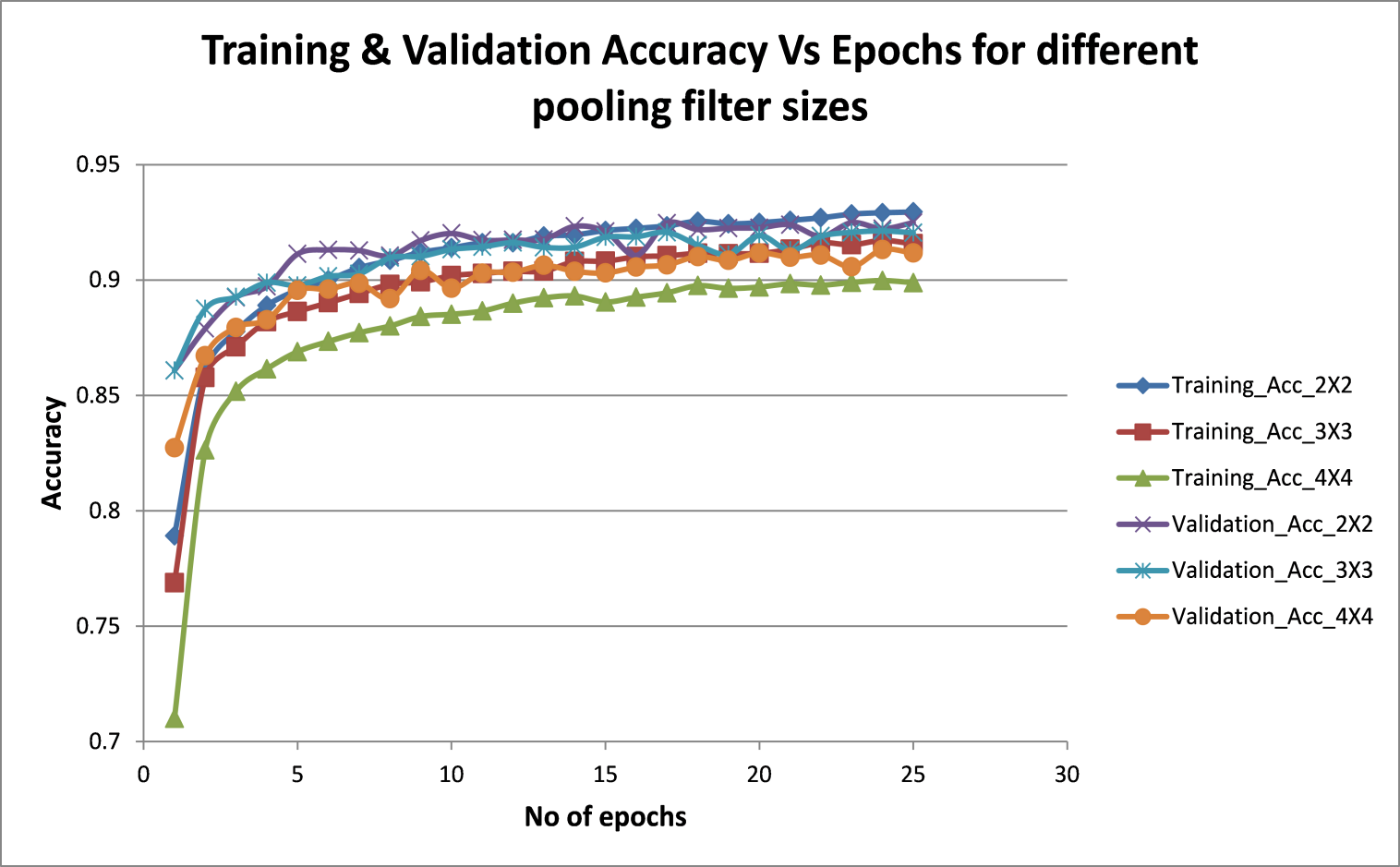
**Figure 10**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different convolutional filter sizes

## Effect of type of pooling

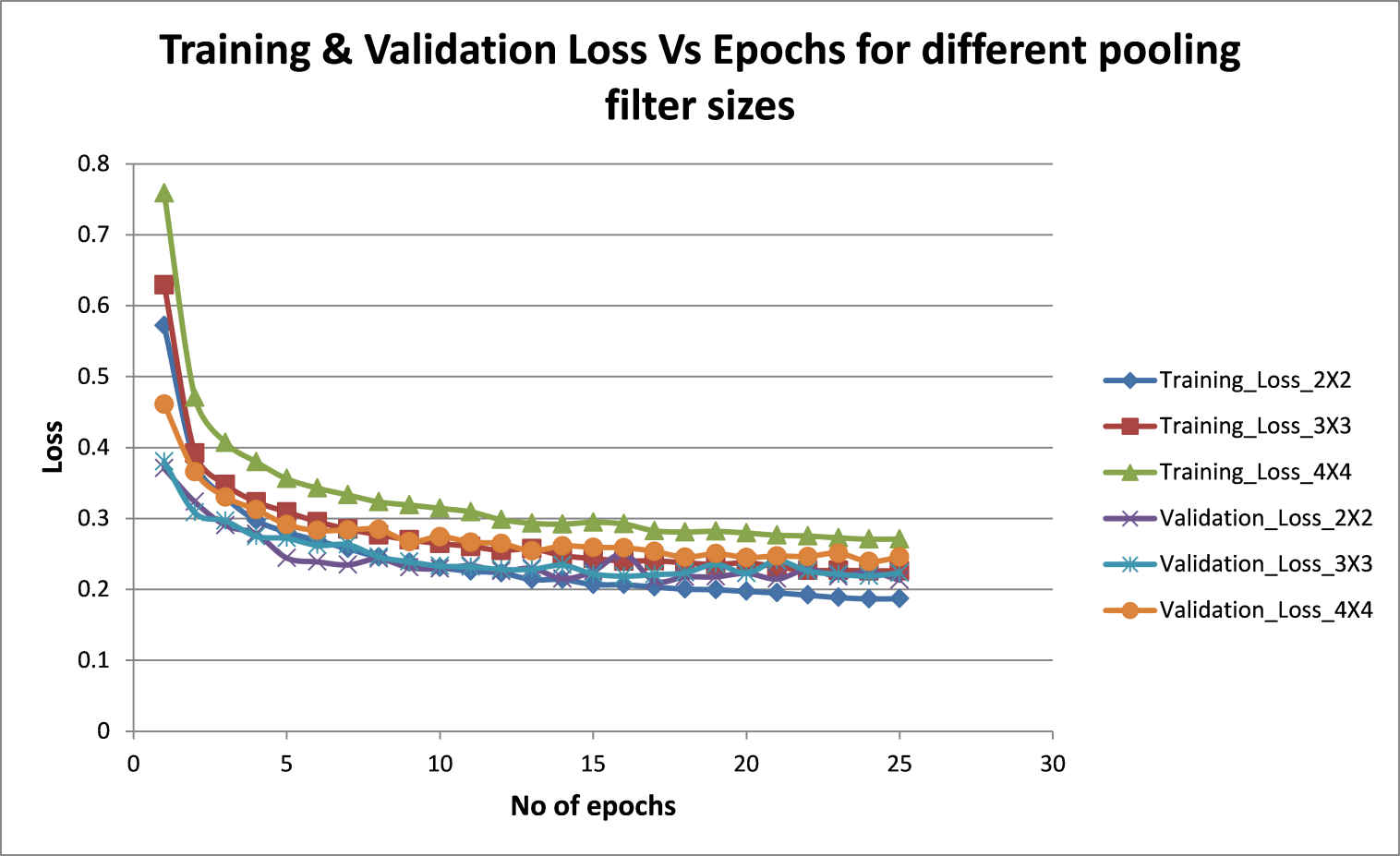
Pooling is a subsampling technique which gets a representation of the input by reducing its dimensions. This is done to prevent overfitting. The most common types of pooling are max pooling and average pooling. In max pooling, the highest pixel intensity value is taken from the region of the image covered currently by the filter. In average pooling, the average of all pixel intensity values under the filter is instead taken. After 25 epochs, the training accuracy for max pooling and average pooling are 0.93 and 0.934 respectively. The validation accuracy for max pooling and average pooling are 0.925 and 0.926 respectively. The test accuracy for max pooling and average pooling are 0.924 and 0.918 respectively. From these values it is observed that for this dataset, the difference between max pooling and average pooling is negligible. For subsequent parametric runs, the pooling was taken as max pooling for all pooling layers.

## Effect of pooling filter size

The filter sizes of all the pooling layers were varied with 2X2, 3X3 and 4X4 filters. From Figure 11, it can be seen that the training and validation accuracy decrease and loss increase as the pooling filter size is increased. This is because pooling is a subsampling technique and a higher filter size causes lesser information being relayed to the output of the pooling layer. This is also seen in the test accuracy value which decreases from 0.924 for 2X2 to 0.917 for 3X3 and to 0.906 for 4X4 filter. For subsequent parametric runs, the pooling filter size was taken as 2X2 for all pooling layers.



**(a)**

****

**(b)**

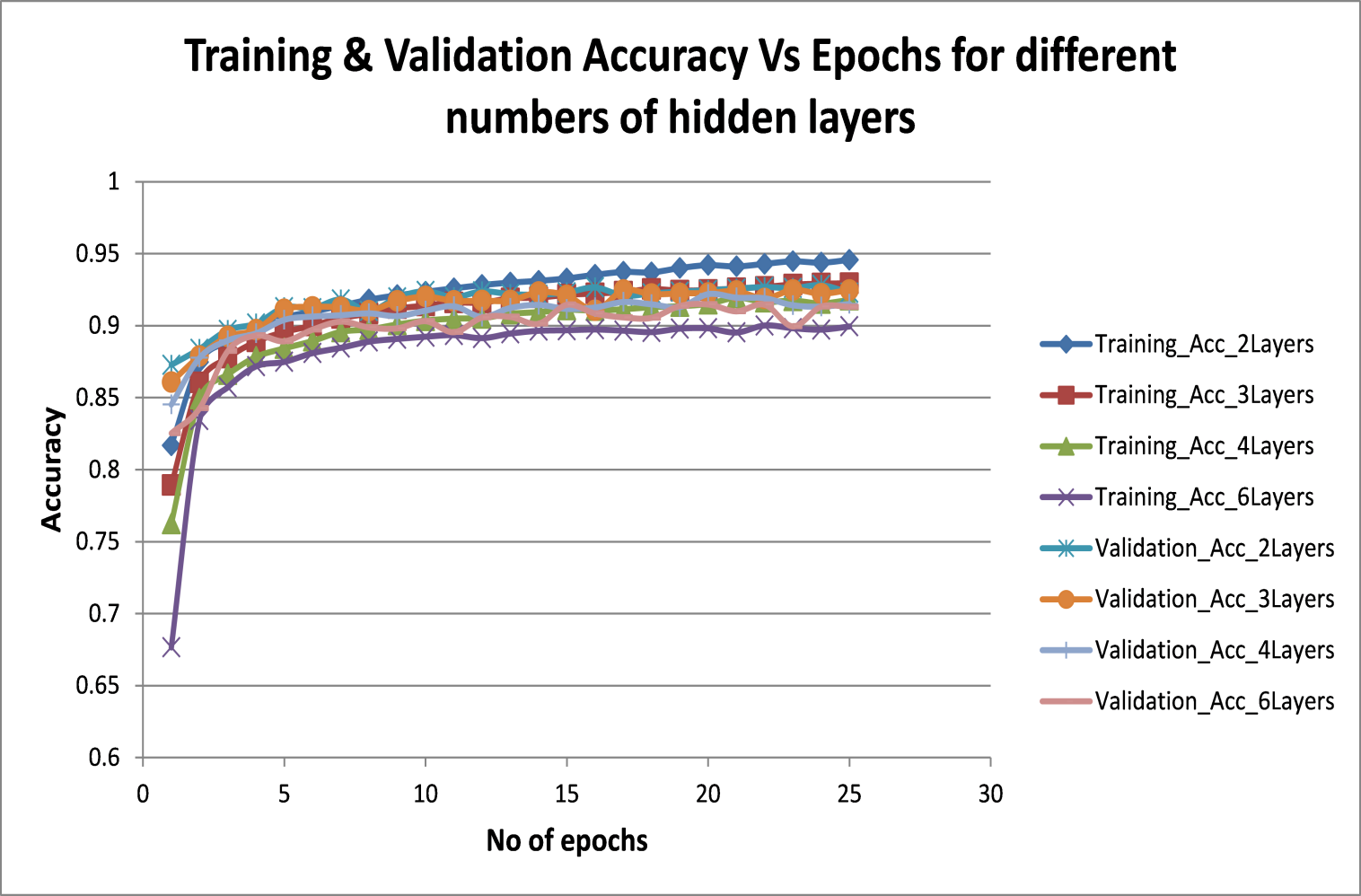
**Figure 11**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different pooling filter sizes

## Effect of number of convolution filters

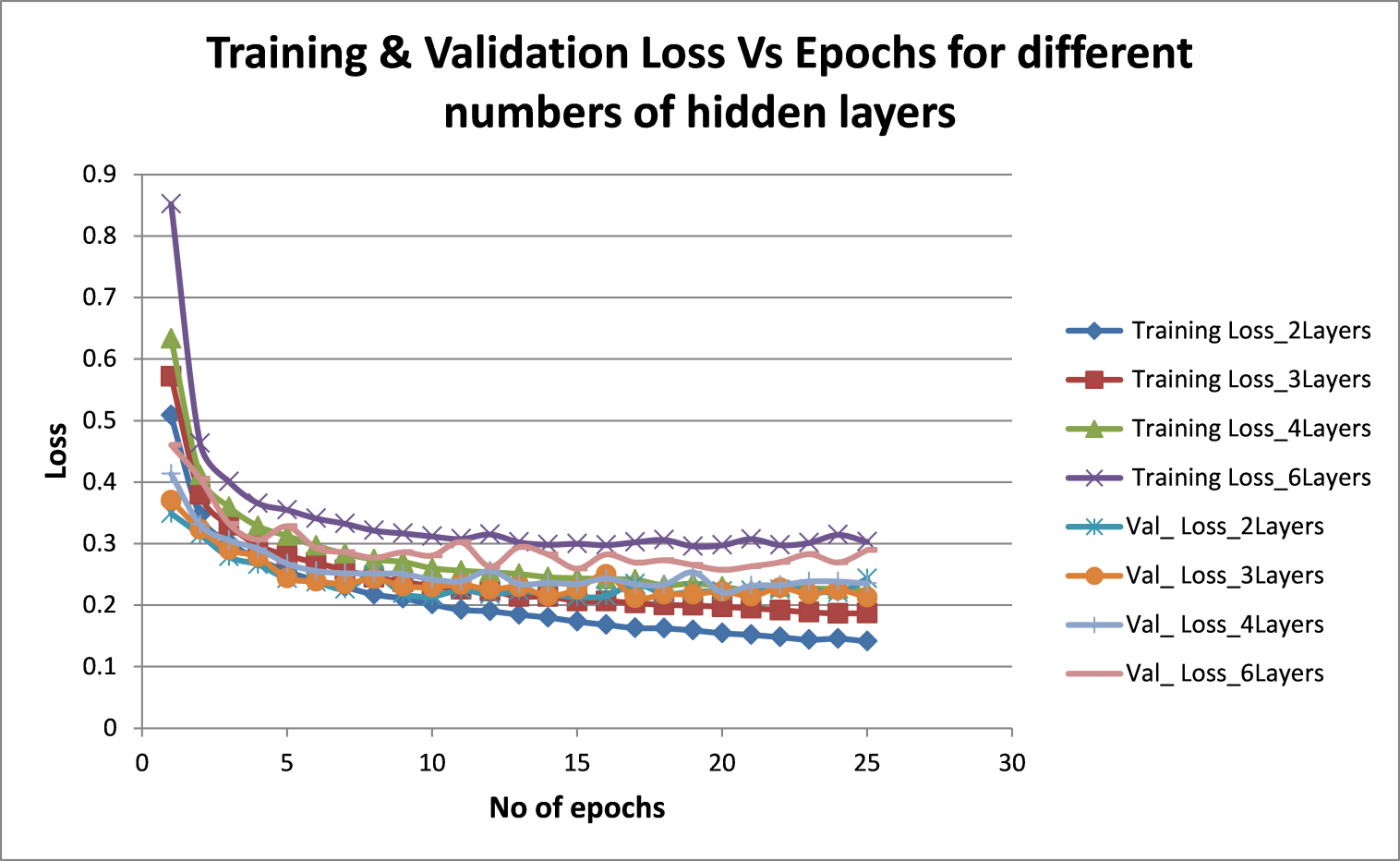
The number of filters in each layer was doubled, that is, 64 filters in the first, 128 filters in the second and 256 filters in the third convolutional layer and the results were compared. After 25 epochs, the training accuracy (0.922) and validation accuracy (0.923) did not change much form the original number of filters (0.93 and 0.925 respectively). The test accuracy also changed from 0.924 to 0.922. So, in this case, the change in number of filters in each convolutional layer has not had a significant effect on the model’s accuracy.

## Effect of number of hidden layers

The number of convolutional and pooling layers is varied as two layers (32 and 64 convolutional 3X3 filters respectively and max pooling 2X2 filters), three layers (baseline architecture described in section 3), four layers (32, 64, 128 and 256 convolutional 3X3 filters respectively and max pooling 2X2 filters) and six layers (32, 64, 128, 256, 256 and 512 convolutional 3X3 filters respectively and max pooling 2X2 filters). The results on training and validation accuracy and loss can be seen in Figure10. It is seen that training accuracy and loss varies little with model depth. But, validation loss for six layers begins to diverge after a few epochs. This is because the model begins to overfit the data and is unable to generalize well. Increase in model depth also comes at the price of increased computational time and power required for training. Finally, from the test accuracy and test loss graphs (Figure 13) it is observed that there exists an optimum model depth at which accuracy is highest with reasonable consumption of computational time. So, as a thumb rule, it is important to begin with 2 or 3 hidden layers and add hidden layers one by one until no reasonable improvement in accuracy is achieved.

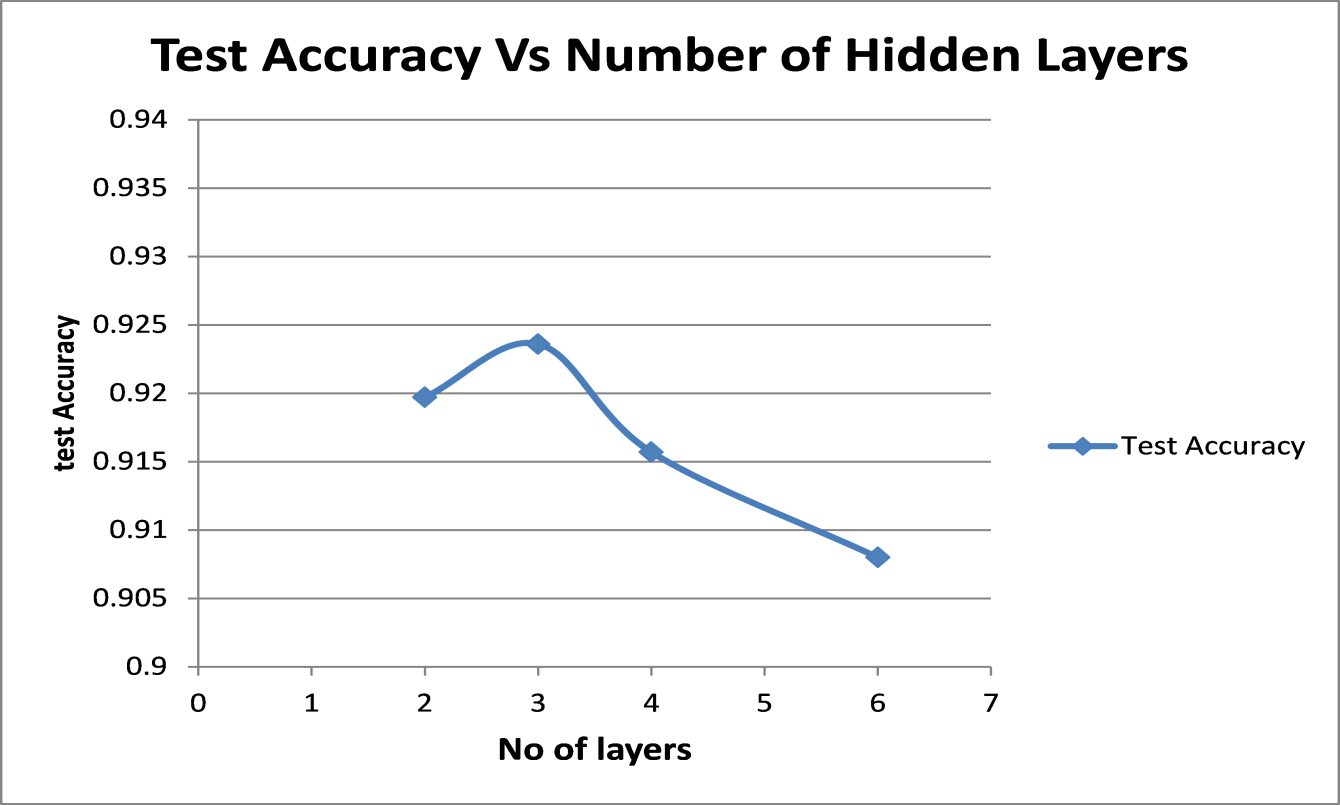


**(a)**

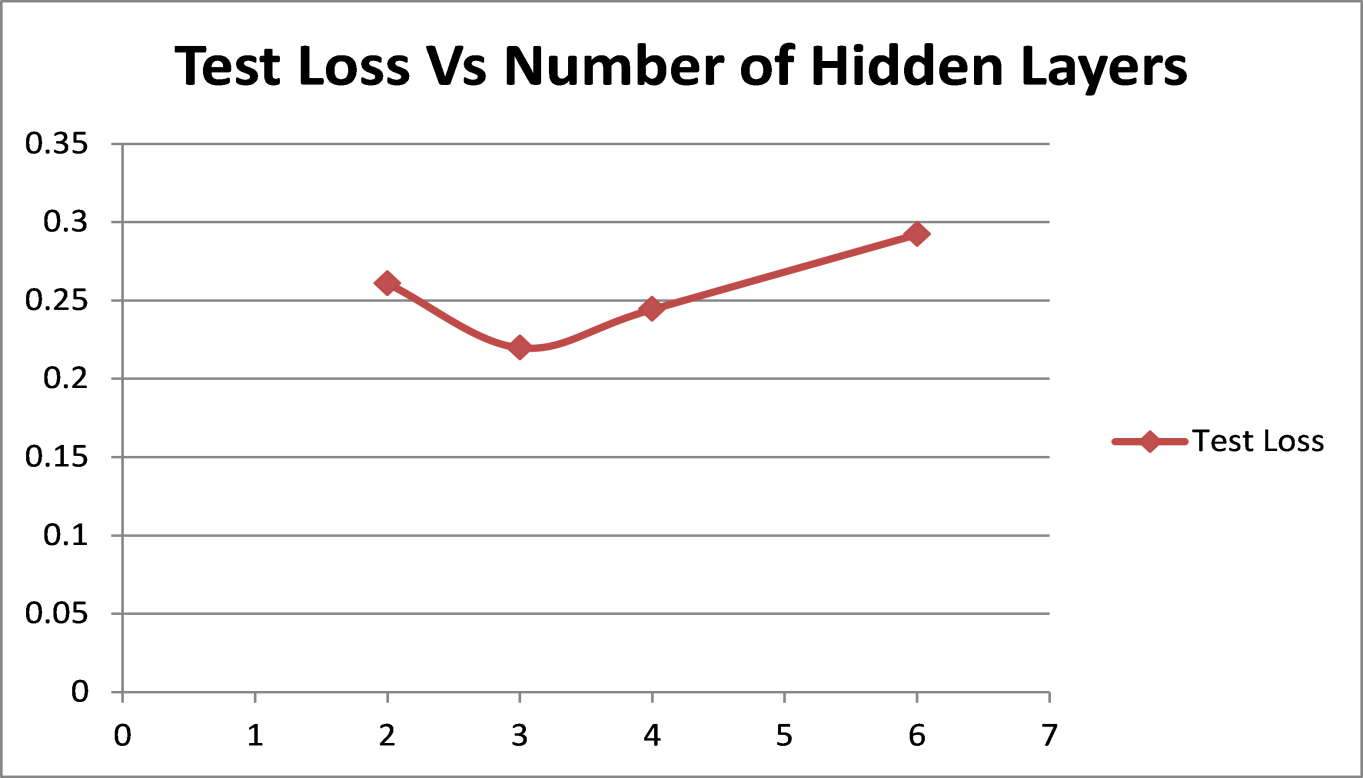
****

**(b)**

**Figure 12**: Variation of training and validation **(a)** accuracy and **(b)** loss versus epochs for different number of hidden layers



**(a)**

****

**(b)**

**Figure 13**: Variation of test **(a)** accuracy and **(b)** loss versus number of hidden layers

# Benchmarking baseline model on Fashion MNIST Dataset

The model used to benchmark against performance quoted in literature has the following hyperparameters:

**Table 2: Baseline model hyperparameter values for benchmarking performance on Fashion MNIST dataset**

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Batch size | 32 |
| Epochs | 25 |
| No of convolutional and pooling layers | 3 |
| No of convolutional filters | 32, 64, 128 respectively |
| Type of pooling | Max |
| Pooling filter size | 2X2 |
| Convolutional filter size | 3X3 |
| Padding | ‘same’ |
| Drop out | 0.25 |
| Learning rate | 0.001 |

The model achieved mean test accuracy of 0.922. The benchmark against traditional algorithms as quoted in literature is shown below in Table 3:

**Table 3: Benchmarking performance against models quoted in literature on Fashion MNIST dataset [4]**

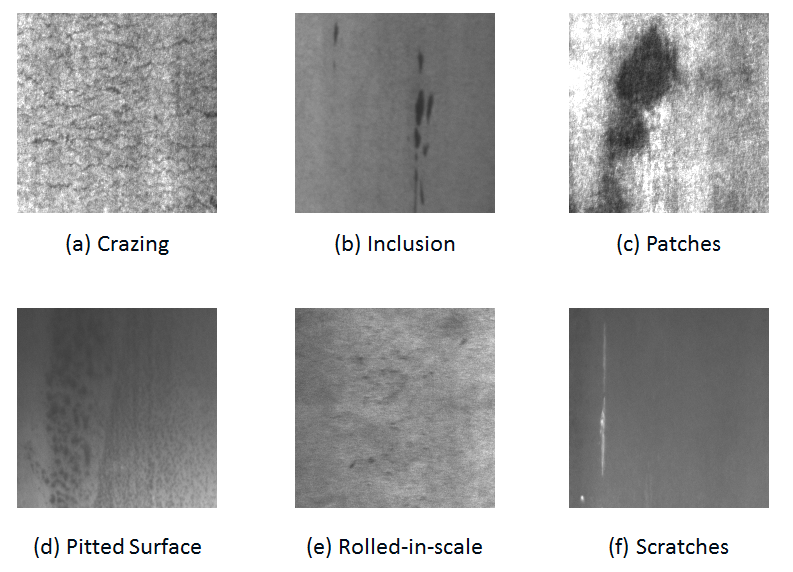
|  |  |  |
| --- | --- | --- |
| **Name** | **Parameter** | **Accuracy (mean)** |
| SVC | {"C":10,"kernel":"poly"} | 0.897 |
| GradientBoostingClassifier | {"loss":"deviance","max\_depth":10,"n\_estimators":100} | 0.888 |
| RandomForestClassifier | {"criterion":"entropy","max\_depth":50,"n\_estimators":100} | 0.879 |
| MLPClassifier | {"activation":"relu","hidden\_layer\_sizes":[100]} | 0.877 |
| KNeighborsClassifier | {"n\_neighbors":5,"p":1,"weights":"distance"} | 0.86 |
| LogisticRegression | {"C":1,"multi\_class":"ovr","penalty":"l1"} | 0.842 |
| LinearSVC | {"C":1,"loss":"hinge","multi\_class":"ovr","penalty":"l2"} | 0.837 |
| SGDClassifier | {"loss":"modified\_huber","penalty":"elasticnet"} | 0.831 |
| DecisionTreeClassifier | {"criterion":"entropy","max\_depth":10,"splitter":"best"} | 0.801 |
| PassiveAggressiveClassifier | {"C":10} | 0.794 |
| Perceptron | {"penalty":"l1"} | 0.771 |
| **RR Baseline CNN** | **Activation: relu, criterion: losses.categorical\_crossentropy, optimizer: Adam** | **0.922** |

# NEU Surface Defect Dataset

After benchmarking the performance of the model against the Fashion MNIST dataset, the model is then benchmarked against an engineering surface defect dataset that is publicly available. The dataset chosen is the NEU surface defect dataset [9]. This dataset contains images of six classes of defects on hot-rolled steel strip. The defects are Crazing (Cr), Inclusion (In), patches (Pa), pitted surface (PS) and scratches (Sc). Each class contains 300 grayscale images of size 200X200 pixels.

Sample images of the six classes of defects are shown in Figure 14. The test set is chosen by randomly picking 150 images in each defect class, as done in literature [10]. Out of the 150 images per class remaining for training, 45 images per class are randomly set aside for validation and 105 images per class are used for training the model.

The NEU dataset serves as a good means to benchmark the performance of the baseline model. In addition, the dataset serves some additional challenges in the form of a small dataset (300 images per class). For deep learning models, this poses a formidable challenge to train the model with limited data points and make it learn to generalize. Also, the intra-class visual differences are at times greater than the inter-class visual differences, as also mentioned in literature [9].



**Figure 14**: Example images of the six classes of defects in the NEU dataset

# Benchmarking baseline model on NEU Dataset

Given the nature and size of the dataset, the model’s hyperparameters were changed slightly to produce optimal results. An important distinction was that the model took more number of epochs for accuracy and loss to flatten out as compared to the Fashion MNIST dataset.

**Table 4: Baseline model hyperparameter values for benchmarking performance on NEU dataset**

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Batch size | 32 |
| Epochs | 100 |
| No of convolutional and pooling layers | 4 |
| No of convolutional filters | 32, 64, 128, 256 respectively |
| Type of pooling | Max |
| Pooling filter size | 2X2 |
| Convolutional filter size | 3X3 |
| Padding | ‘same’ |
| Drop out | 0.15 (0.5 for FCN layer) |
| Learning rate | 0.0005 |

The model achieved mean test accuracy of 0.962. The benchmark against other models as quoted in literature [10] is shown below in Table 5:

**Table 5: Benchmarking performance against models quoted in literature on NEU dataset [10]**

|  |  |  |
| --- | --- | --- |
| **Model** | **Classifier** | **Test Accuracy (mean)** |
| LBP | NNC | 95.07 |
| SVM | 97.93 |
| LTP | NNC | 95.93 |
| SVM | 98.22 |
| CLBP | NNC | 96.91 |
| SVM | 98.28 |
| AECLBP | NNC | 97.93 |
| SVM | 98.93 |
| **RR Baseline CNN** | **Softmax** | **96.23** |

# Conclusions

The art of tuning hyperparameters of a deep learning model requires careful understanding of effect of each hyperparameter on the output and balancing the effects of different hyperparameters for optimal results. This report helps develop the intuition behind setting these hyperparameters. However, it must be noted the optimal value also depends on the properties of the dataset such as the number of training images and size of the input images. The baseline model employing CNN architecture achieved an accuracy of 0.922 on the Fashion MNIST dataset and an accuracy of 0.962 on the NEU Surface Defect dataset. The performance of the model is also benchmarked and compared with the models in literature.

# References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf Process. Syst., 2012, pp. 1097–1105.

[2] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” in Proc. Adv. Neural Inf. Process. Syst., Montreal, QC, Canada, 2014, pp. 3320–3328.

[3] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “CNN features off-the-shelf: An astounding baseline for recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops, Columbus, OH, USA, 2014, pp. 806–813.

[4] Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document

recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[6] http://web.stanford.edu/class/cs231a/lectures/intro\_cnn.pdf

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15 (2014) 1929-1958.

[8] Sungheon Park and Nojun Kwak. Analysis on the Dropout Effect in Convolutional Neural Networks. Computer Vision – ACCV 2016: 13th Asian Conference on Computer Vision, Revised Selected Papers Part II (pp. 189-204)

[9] K. Song and Y. Yan, “A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects,” Appl. Surface Sci., vol. 285, pp. 858–864, Nov. 2013.

[10] K. Song, S. Hu, and Y. Yan, “Automatic recognition of surface defects on hot-rolled steel strip using scattering convolution network,” J. Comput. Inf. Syst., vol. 10, no. 7, pp. 3049–3055, 2014.