Report

The report should be a short paper detailing what you did, experiments you ran, and the results.

It’s suggested you follow the traditional format of:

* Abstract
* Introduction – motivate the problem
* Related work – compare and contrast to other publications
* Contribution / Method – detail your method
* Results – Show experimental results and evidence supporting your claims

Be sure to document things that you struggled with, strange results, and output your own images / data. It is strongly recommended that your write your report in LaTeX.

The report will be graded based on:

* Quality – lacks types, spelling errors, etc.
* Readability – it is easy to understand the contribution and results.
* Completeness – it is clear from the report how well the project achieves the requirements. (I recommend making a header for each requirement).

**Abstract**

In the following project I worked on the MNIST classification problem. The tensorflow and keras libraries are used to create custom CNN architecture. The custom CNN model was trained using two optimizers and three learning rate configurations. I also fine-tuned the pre-trained VGG16 architecture available in the tensorflow library and compared the results with custom CNN model.

**Introduction – motivate the problem**

In image processing, classification is the problem of finding category of object in the image, e.g. person, car, bus, cat, etc. The images consist of intensity values usually ranging from 0 to 255. Humans have inherent ability to understand images and can easily classify things without any difficulty. But for computers, images are just a matrix of numbers stored in storage media. In order for computers to understand images we need to extract features from images. Features can be thought of as edges, shapes, objects, eyes, nose, faces, hands, wheels etc. With the advent of machine learning and deep learning various image classification techniques are created in the past. Image classification can be divided into two steps: 1) Feature Extraction and 2) Image Classification. In conventional machine learning and image processing, the first step required domain expertise and was very tedious because various techniques needed to be created for efficient feature extraction and in the second stage numerous classification algorithms were available, e.g. SVM, Logistic Regression, Random Forest, etc. With the advent of deep learning and efficient algorithms to train neural networks, both stages are taken care of automatically. Neural networks are used as feature extractor as well as classifiers depending on the type of layer. In deep learning we can create neural networks of arbitrary number of hidden layers. The deeper the NN, the better is its ability to extract features in hierarchical fashion.

**Related work – compare and contrast to other publications**

Images are 2d grid of numbers and each pixel has relation in both directions. The neural networks are not meant to learning 2d relations. So, in deep learning Convolutional Neural Networks (CNNs) are type of neural networks that can learn 2d relationship between pixels in the images. The first CNN was created by LeCunn as LeNet [1] in around 1998, but due to technological limitations the idea didn’t survive much. In 2012, with devices like GPU, CNN gained much popularity when AlexNet [2] significantly improved the performance of machines on classification task. Numerous research works on CNNs exploited the effect of depth, width, filter properties, layer types etc on the performance and various state-of-the-art models were proposed from time to time. The VGG16 and VGG19 [3] models highlighted the significance of small 3x3 receptive fields. The InceptionNet [4] highlighted the importance of using 1x1, 3x3, 5x5 receptive fields in parallel to extract diverse set of features and then fusing them to obtain better discrimination ability of CNNs. The ResNet [5] family introduced the idea of residual connection that solved the vanishing gradient problem and models of much large depths (ResNet-101, ResNet-151) were made possible. Using the idea of residual connection, DenseNet [6] transferred signal from all layers to the current layer to improve the learning ability of CNNs.

**Contribution / Method – detail your method**

**Dataset**

In this project I used the MNIST database [7]. MNIST stands for Modified National Institute of Standards and Technology database. It is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. The dataset was created by "re-mixing" the samples from NIST's original datasets. The black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

Tensorflow library [8] provides access to MNIST dataset through Keras interface. Specifically MNIST is located in *tensorflow.keras.datasets.mnist* module.

**Custom CNN**

The CNN model consists of two parts: 1) Feature extraction part and 2) Classification part. The feature extraction part was created using sequence of Conv2D, BatchNormalization, ReLU, and MaxPool2D layers. The dropout was also used to avoid overfitting. The classification part was created using Dense layers along with BatchNormalization and dropout layers. The figure1 shows the architecture of Custom CNN model. The total number parameters are 753,418. The total trainable parameters are 751,690 and total non-trainable parameters are 1,728 in custom CNN architecture.

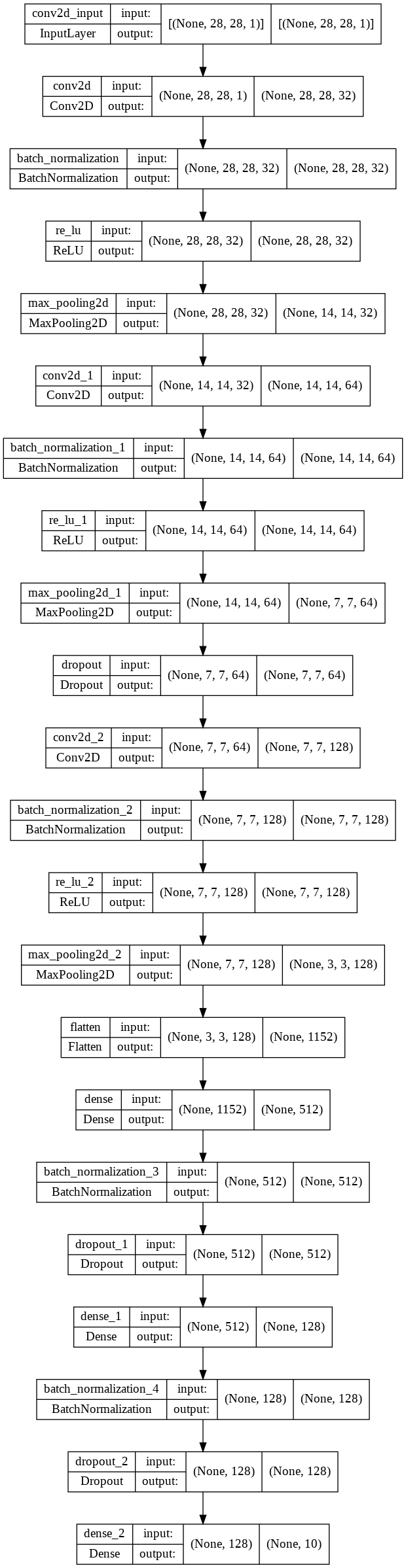
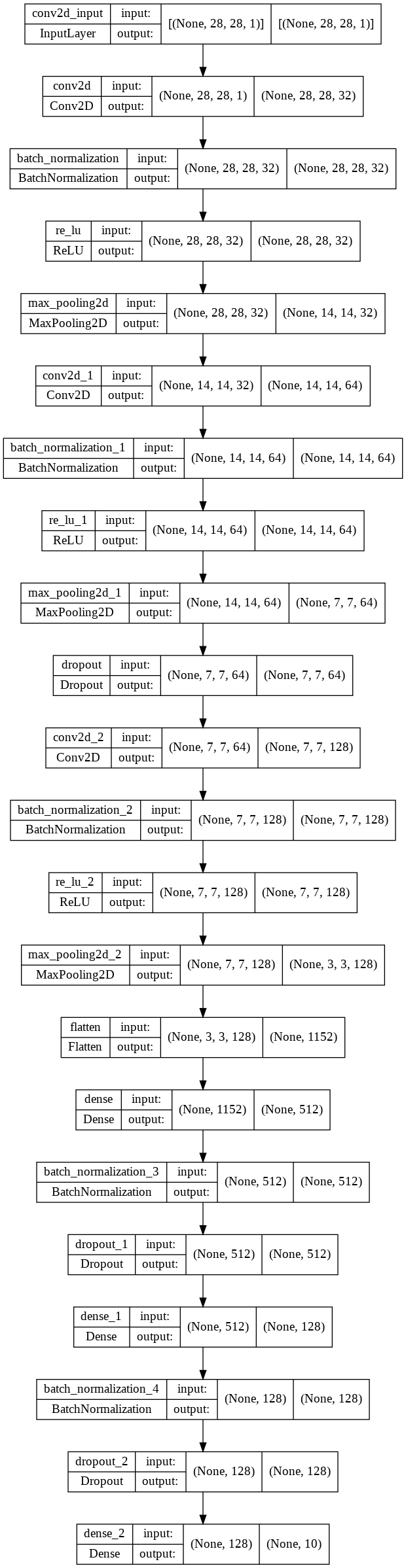


Figure 1: Custom CNN architecture. The output of left part is input to the right part.

The custom CNN takes as input a grayscale image of size 28x28. The image passes through feature extractor and dense layers at the end of CNN are used for classification of MNIST digits.

**Fine-Tuning VGG16**

The VGG16 model remained state-of-the-art classification model in 2014. The idea of using two 3x3 kernels instead of 5x5 and three 3x3 kernels instead of 7x7 is shown to be effective in this paper. It is due to the fact that non-linearity after each 3x3 layer provides more control in the feature space and the model is able to separate between features of different classes. The VGG16 available in Tensorflow library is trained on ImageNet dataset. Normal input image size of VGG16 is 224x224 and minimum image size is 32x32. The output of VGG16 is 1000 numbers because ImageNet contains 1000 classes. For the purpose of comparison, the dataset of MNIST digits were resized from 28x28 images to 32x32 images and VGG16 architecture was modified at the final layer to support only 10 MNIST digit classes. The VGG16 architecture is provided in figure2 and figure 3 provides the modified VGG16 architecture for MNIST classification.

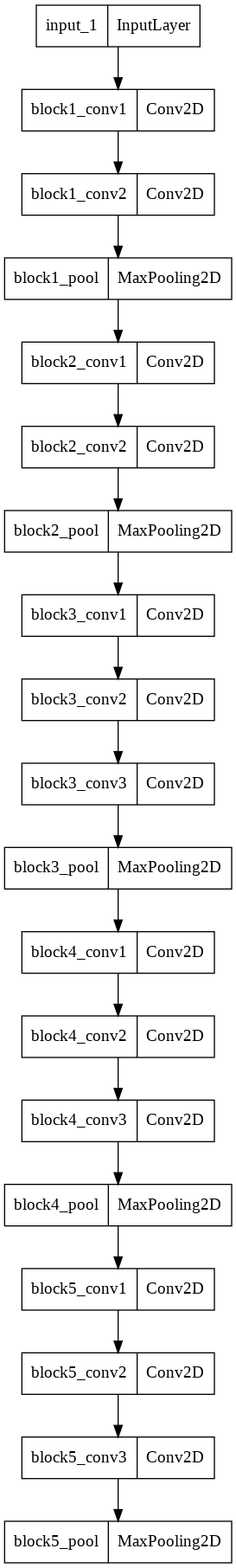
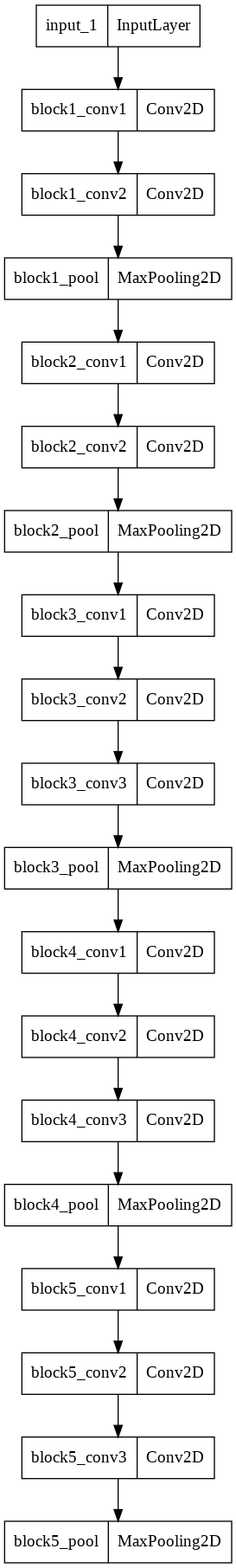


Figure 2: VGG16 Architecture

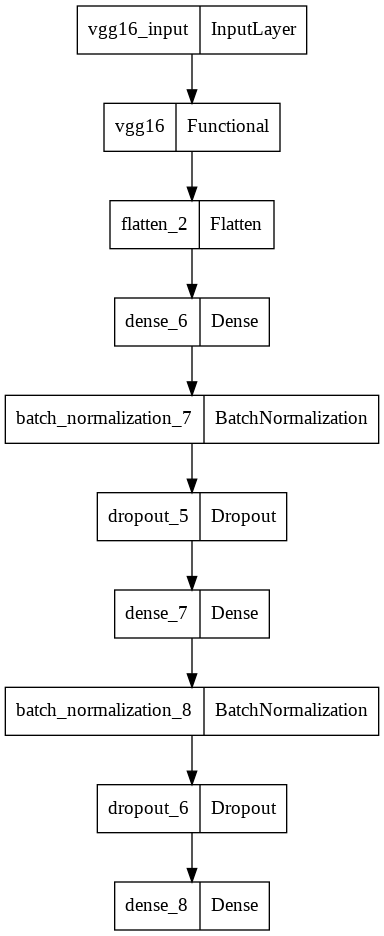


Figure 3: VGG16 Architecture Modified

**Results – Show experimental results and evidence supporting your claims**

The custom CNN was trained on two different optimizers, i.e. Adam Optimizer and SGD Optimizer. Both the optimizers were trained with three different learning rate i.e. 0.1, 0.01, 0.001. So, in total six custom CNNs were trained each with different optimizer and learning rate (LR). The *sparse\_categorical\_crossentropy* loss function was used to train all the different versions of custom CNN. The figure 4 shows the learning curves (loss vs epoch and accuracy vs epoch) for train dataset (top row) and validation dataset (bottom row). In classification problem, the accuracy is defined as percentage of correct predictions that model performed correctly. The curves show that Adam optimizer with around small learning rate is good.

The figure 5 shows comparison between fine-tuned VGG16 and custom CNN architecture. The curves show that the custom CNN and fine-tuned VGG16 performed better with Adam optimizer and small learning rate.

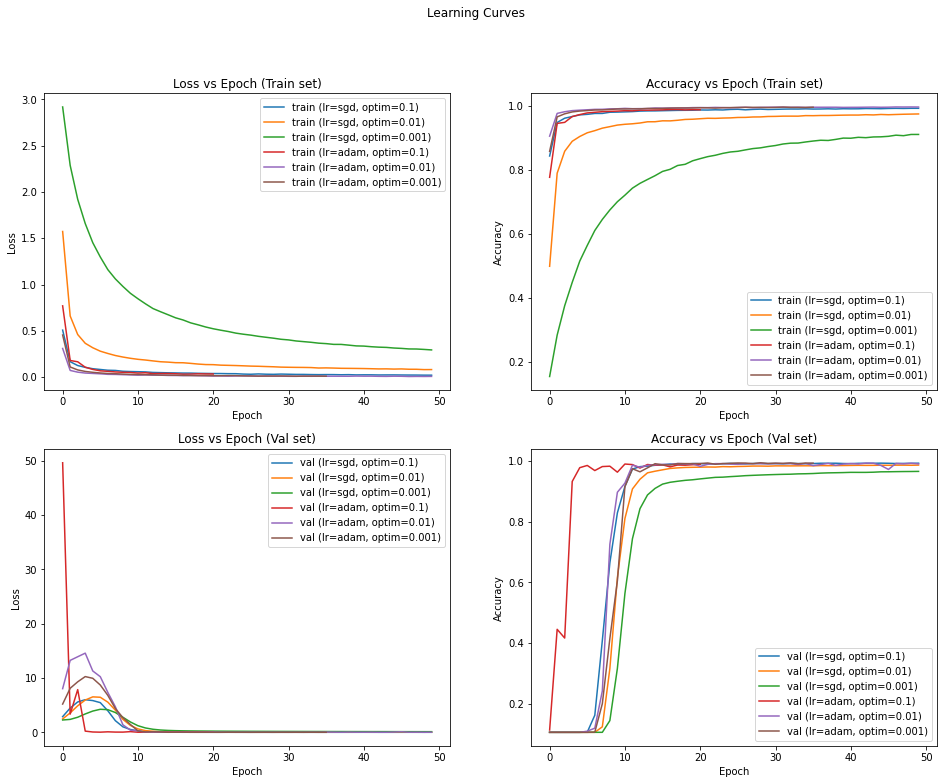


Figure 4: Learning Curves of Custom CNN

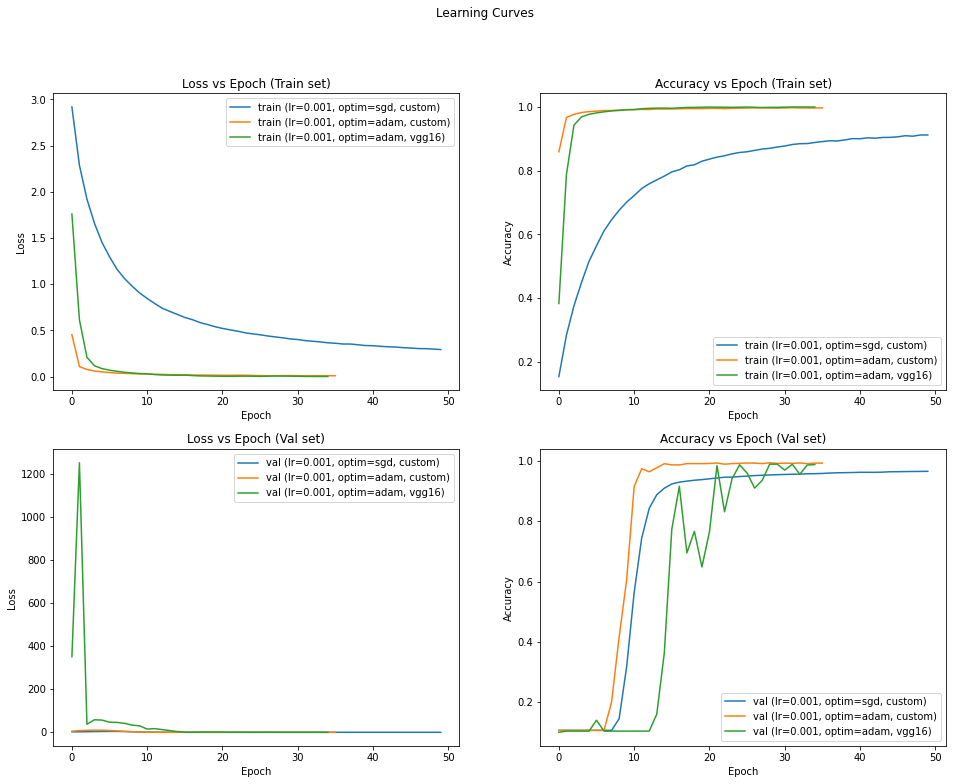


Figure 5: Fine-Tuned VGG16 Comparison for Adam Optimizer and 0.001 LR

# References

|  |  |
| --- | --- |
| [1] | Y. B. L. B. a. P. H. Yann LeCun, "Gradient Based Learning Applied to Document Recognition," 1998. |
| [2] | I. S. G. E. H. Alex Krizhevsky, "ImageNet Classification with Deep Convolutional Neural Networks," 2012. |
| [3] | A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014. |
| [4] | W. L. Y. J. P. S. S. R. D. A. D. E. V. V. A. R. Christian Szegedy, "Going Deeper with Convolutions," 2014. |
| [5] | X. Z. S. R. J. S. Kaiming He, "Deep Residual Learning for Image Recognition," 2015. |
| [6] | Z. L. L. v. d. M. K. Q. W. Gao Huang, "Densely Connected Convolutional Networks," 2016. |
| [7] | C. C. C. J. B. Yann LeCun, "The MNIST Database of Hand Written Digits," [Online]. Available: http://yann.lecun.com/exdb/mnist/. |
| [8] | Google, "Tensorflow," [Online]. Available: https://www.tensorflow.org/. |