Project 3: Hyperparameter Optimization

Param Vipulkumar Chokshi
University of South Florida
Department of Computer Science and Engineering
pvc@usf.edu

Abstract

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network, most commonly applied to analyze visual imagery.They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps. To efficiently use CNN, tuning hyper parameters that builds the CNN architecture of the model, is an important aspect. Hyper-parameter tuning is solely done by logical experimentation processes. The following report will expand upon one such experimentation process used for tuning hyper-parameters.

1. Introduction

The project aims at practising the experimentation process to find the best set of hyper-parameters for a particular model. Here we use LeNet, which was among the first published CNNs to capture wide attention for its performance on computer vision tasks. The model was introduced by (and named for) Yann LeCun, then a researcher at ATT Bell Labs, to recognize handwritten digits in images. This work represented the culmination of a decade of research developing the technology. In 1989, LeCun published the first study to train CNNs via backpropagation successfully.

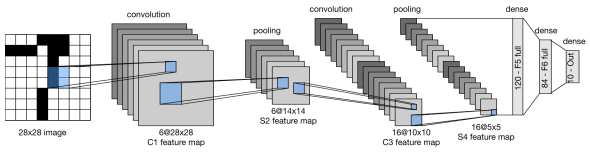

Figure 1: LeNet Architecture

Here we use LeNet as a classifier, which is used to predict 10 classes of the Fashion-MNIST Dataset. Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

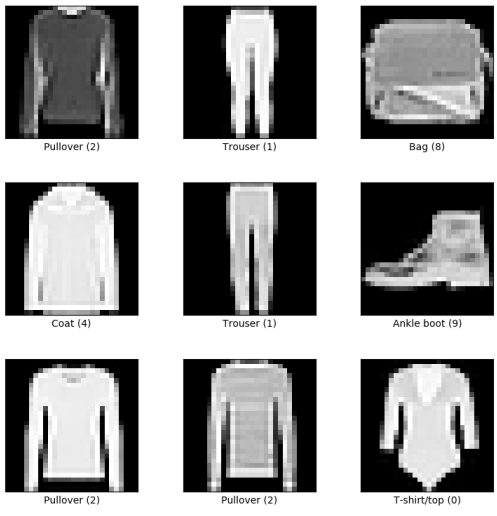

Figure 2: Example images from Fashion-MNIST

2. Experimental Setup

In the project 10 Hyper-parameters were considered. The experiment was divided into 10 tasks, in which each one focused at one hyper-parameter. In each task vacations were applied to the corresponding parameter, while keeping other hyper-parameters fixed. Based on the test accuracy of

the trained model, the best option was decided.

The environment used to implement these project comprised of python along with some essential libraries.

Image Processing: imageio, skimage
Linear algebra: numpy
Plotting: matplotlib
Machine Learning: pytorch , torchvision

General function for training the model for each tasks.

Algorithm 1 SampleTask()

$variations \leftarrow$ list of variations to test
$net \leftarrow$ initialize LeNet
$train\_accuracy \leftarrow []$
$test\_accuracy \leftarrow []$
$train\_loss \leftarrow []$
for i in variations **do**
  $net.train()$
  $train\_accuracy \leftarrow net.train\_accuracy$
  $test\_accuracy \leftarrow net.test\_accuracy$
  $train\_loss \leftarrow net.train\_loss$
end for

$plot(train\_accuracy, test\_accuracy, train\_loss)$
return variation info with best test accuracy

All task can be run together, and the best variation from each task can be plotted simultaneously.

Algorithm 2 runAllTasks(); batch of all tasks together

$best\_params \leftarrow []$
$net \leftarrow$ initialize LeNet
$train\_accuracy \leftarrow []$
$test\_accuracy \leftarrow []$
$train\_loss \leftarrow []$
for i in 1 to 10 **do**
  $best\_var \leftarrow SampleTask()$
  $best\_params \leftarrow best\_params \cup best\_var$
end for

$plot(best\_params)$

3. Results

3.1. Optimization Functions

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.
Variations used are:

1. SGD: Stochastic Gradient Descent

2. Adam

3. RMSprop

Best Variation: SGD

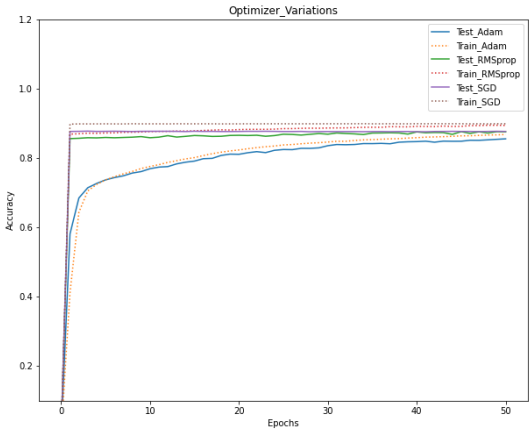

Figure 3: Plot visualizing the learning process of optimizer variations

3.2. Batch Size

Batch size is a term used in machine learning and refers to the number of training examples utilized in one iteration.
Variations Used:

1. 16

2. 32

3. 64

4. 256

Best Variation: 256

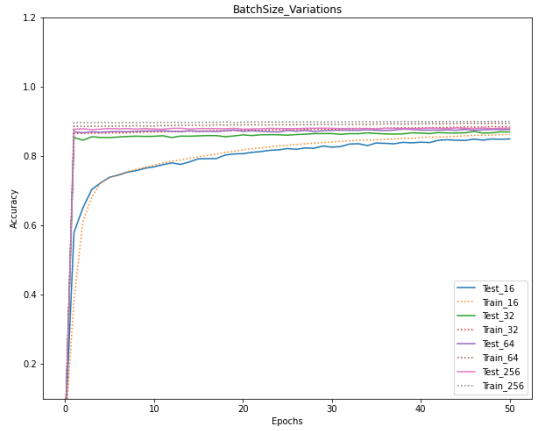

Figure 4: Plot visualizing the learning process of batch size variations

3.3. Convolution Kernel Sizes

Convolution Kernel size corresponds to the size of filter masks used to perform convolutions at each convolutional layer in the model. Convolution helps in getting a compact representation of the input images. Variations used:

1. 3*3
2. 5*5
3. 7*7

Best Variation: 5*5

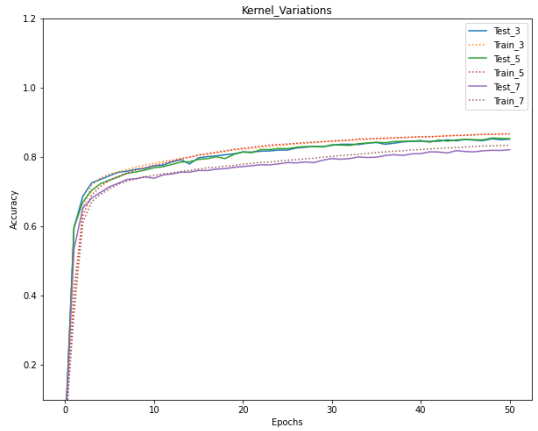

Figure 5: Plot visualizing the learning process of kernel variations

3.4. Number of Output Channels

The input and the output channel are split into multiple channels in CNN. The kernel is applied to each of the channels, which can potentially result into better feature extraction. Variations used:

1. (4,12)
2. (6,16)
3. (8,20)

Best Variation: [8,20]

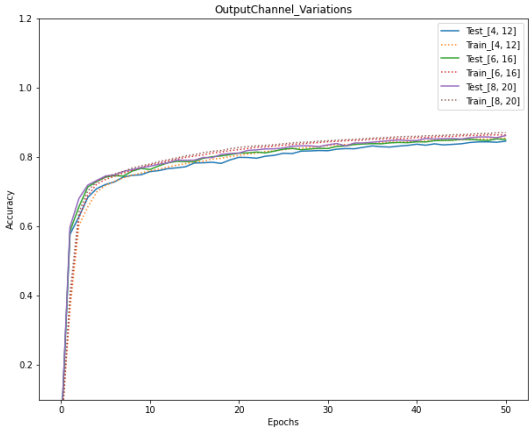

Figure 6: Plot visualizing the learning process of Output Channel variations

3.5. Pooling

Pooling layers serve the dual purposes of mitigating the sensitivity of convolutional layers to location and of spatially down-sampling representations. Variations used:

1. Average Pooling
2. Maximum Pooling

Best Variation: Maximum Pooling

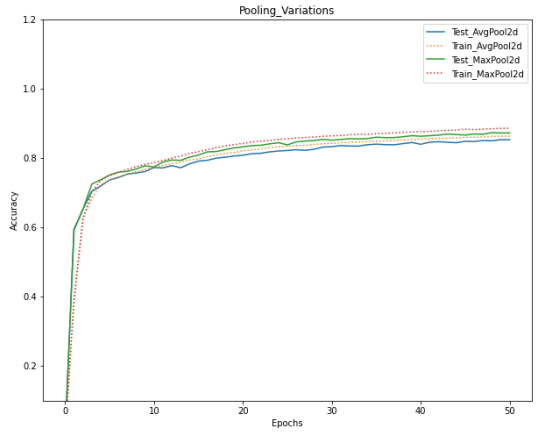

Figure 7: Plot visualizing the learning process of pooling variations

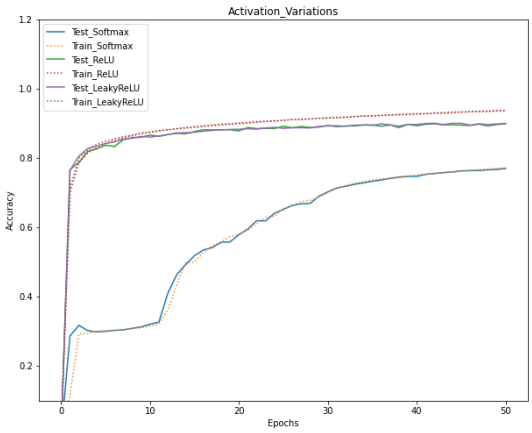

Figure 8: Plot visualizing the learning process of Activation function variations

3.6. Activation Function

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.Sometimes the activation function is called a "transfer function." If the output range of the activation function is limited, then it may be called a "squashing function." Many activation functions are nonlinear and may be referred to as the "nonlinearity" in the layer or the network design.The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model. Variations used:

1. Softmax
2. ReLU
3. LeakyReLU

Best Variation: LeakyReLU

3.7. Number of Convolution Layers

Modifying the number of convolutional layers changes the size of the parameter space which our network is trying to optimize. Based on the size of our input and the complexity of our features, adding more layers might help in improving our model accuracy, but at the same time can also result into over-fitting if we have more than necessary. Variations used:

1. 1 layer
2. 2 layers
3. 3 layers

Best Variation: 1 layer

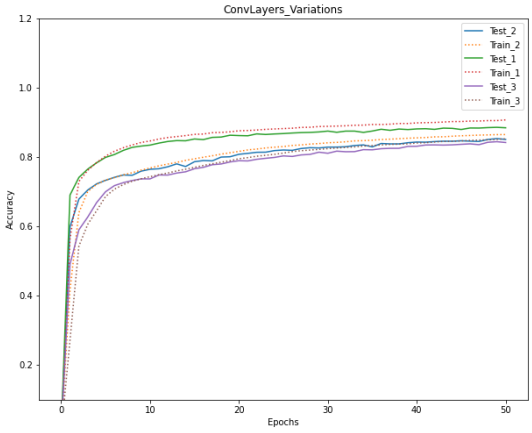

Figure 9: Plot visualizing the learning process of Convolution layers variations

3.8. Number of Fully Connected Layers

In naive terms, adding convolution layers is refereed to as adding depth too the network , while adding fully-connected layers is refereed to as adding width. Summarily to the convolution layer discussion above, adding more FC layers might help in extracting more specific features, but could also result into the case where the network over fits by storing highly specific features corresponding to the training data. Variations used:

1. 1 layer
2. 2 layers
3. 3 layers

Best Variation: 3 Layers

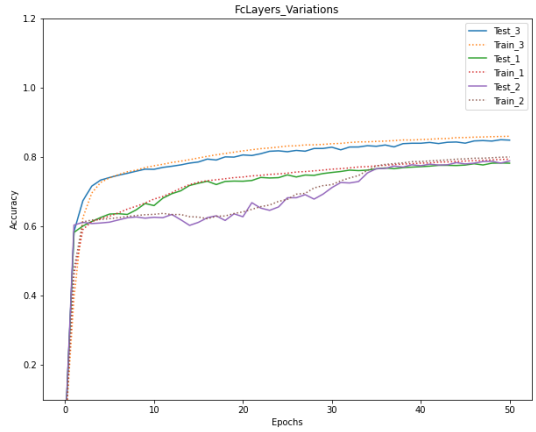

Figure 10: Plot visualizing the learning process of Fully-connected layers variations

3.9. Learning rate Epochs

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.An epoch means training the neural network with all the training data for one cycle. In an epoch, we use all of the data exactly once. Thus learning rate and the number of epoch are related to each other. For slower learning rates, we will need more iterations of training to converge and reach the best accuracy. Conversely, we need faster learning rates to complete our training process with fewer epochs. Variations used (Learning rate, of epochs):

1. (0.00001, 1000)
2. (0.0001,500)
3. (0.001,250)
4. (0.01, 100)
5. (0.1,50)

Best Variation: [0.0001, 500]

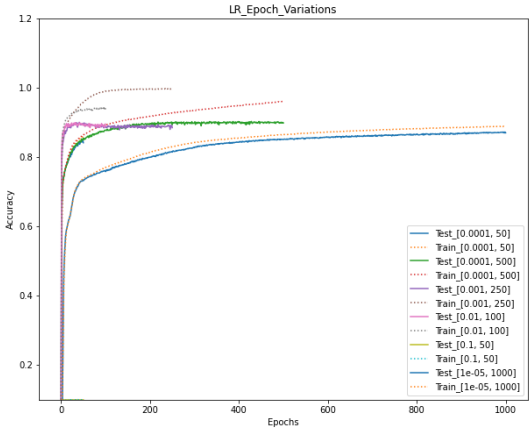

Figure 11: Plot visualizing the learning process of Learning rate Epochs variations

3.10. Initialization

Weight initialization is a procedure to set the weights of a neural network to small random values that define the starting point for the optimization (learning or training) of the neural network model. Variations used:

1. Default/ none
2. Xavier
3. Kaiming

Best Variation: Xavier

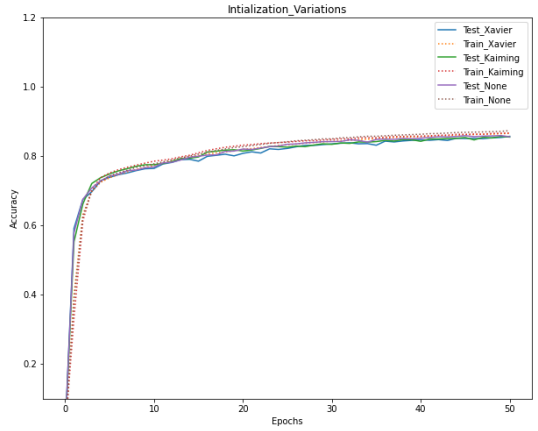

Figure 12: Plot visualizing the learning process of Initialization variations

3.11. Best Parameters

To find the best parameters, the metric used for comparison is test accuracy. Here we don't consider training accuracy as they are not always a good indicator of the classifier's ability to predict unseen data. While training CNN models, there is a possibility of over fitting the training data. Over-fitting is easy to identify by looking at the plot of training and test accuracy. Big differences in final test and training accuracy indicates over-fitting.

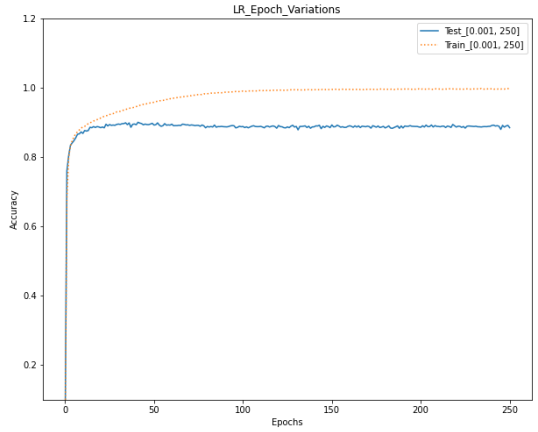

Figure 13: Over-fitting example

The following figure shows the best performing hyper-parameters. Note, in the image the graph for the model trained for 500 epoch is cut short just to improve the comparative analysis of different models.

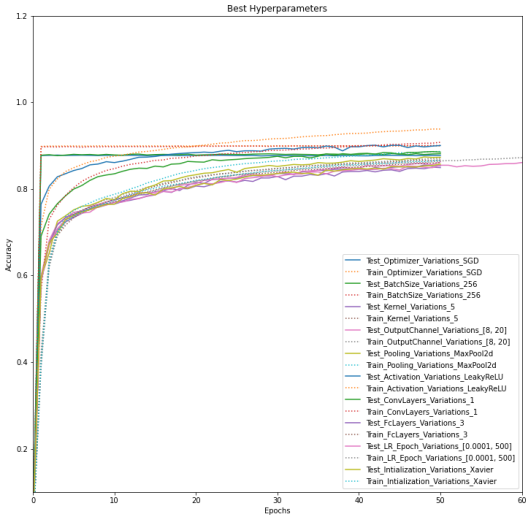

Figure 14: Plot visualizing the learning process of best hyper parameter variations

To help visualizing the model, we can display the activation (outputs) of the first and the second layer of the model for various input classes.

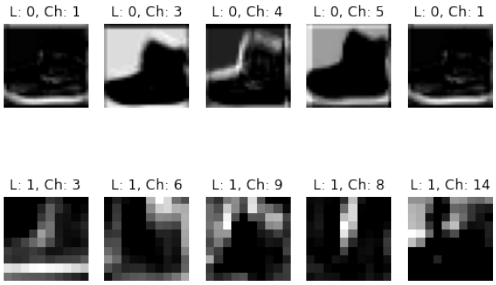

Figure 15: Input Class: Ankle boot

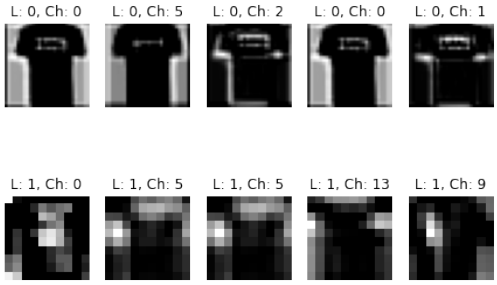

Figure 16: Input Class: T-Shirt/top

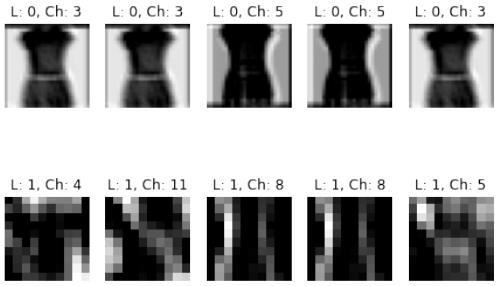

Figure 17: Input Class: Dress

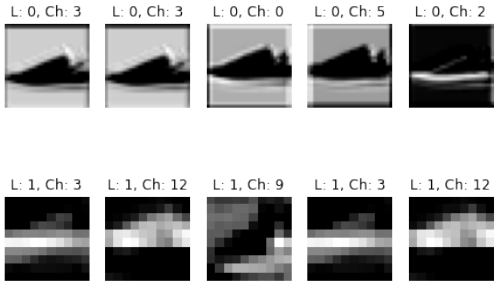

Figure 18: Input Class: Sneaker

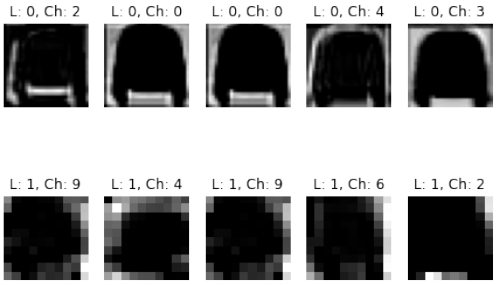

Figure 19: Input Class: Coat

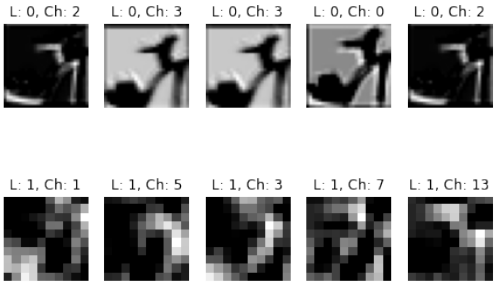

Figure 20: Input Class: Sandal

4. Conclusions

Hyper-parameter optimization is a time consuming and tedious process. CNN networks have long training time. Availability of resources like GPU and TPU can boost training times. Thus to achieve results in a timely manner, the experiment should start by evaluating factors which might affect the time the most. For example, tuning the number epochs in the beginning can help in getting an estimate on the number epochs needed to achieve good performance and at the same time not waste time in over fitting the classifier. Apart from that, using good initialization can help our model converge faster. Also experimenting with the number of convolutional and fully-connected layers can help reduce the parameters space which we are trying to optimize. Thus this ensures that we look for the most compact representation of our classes. Reducing layers also help in getting faster inference times which might be critical for applications like mobile biometrics. Optimizing Kernel sizes, pooling values, and activation functions can lead into finding better weights that results into higher accuracy.

5. References

The report and the code uses elements from Dr. Sudeep Sarkar's (Department of Computer Science and Engineering, University of South Florida) Colab Notebooks, from the class curriculum of CAP 4410 Computer Vision, Fall 2021:

https://github.com/SudeepSarkar/Undergraduate-Computer-Vision/blob/main/CAP_4410_Lecture_16_Convolutional_Neural_Networks.ipynb