NORTH CAROLINA STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING

ECE 763: COMPUTER VISION

Spring 2020 **Project 3** Dr. Tianfu Wu

ECE 763: Computer Vision: Models, Learning, and Inference

Evaluation of LeNet 5 for Facial Recognition

John McDonald

April 29th, 2020

Abstract

In this paper we evaluate the use of Deep Neural Networks in computer vision through the baby-sitting procedure. The neural network is known to be an effective tool for classification problems in facial detection. The final project for this course will evaluate the DNN to retrospectively compare this method with other applications in computer vision such as Viola-Jones object detection framework, and various fundamental mathematical models for classification. In this paper, we specifically implement the LeNet 5, a classic convolutional neural network. Data from Project 1 is preprocessed to fit as an input to the standard LeNet 5 network architecture. The hyperparameters for the model are initialized randomly. The baby-sitting process is used for hyperparameter selection, data augmentation, regularization, and to evaluate various steps in data preprocessing and model performance. The finalized model is effective at classifying testing set with over 97% testing accuracy. The accuracy is significantly improved from initial build with default hyperparameters.

Objective

The objective of the project is to evaluate a LeNet 5 DNN model for 2 class face detection. This includes:

- Data Preparation, using 2 classes of positive and negative faces.
- Development of LeNet 5 Neural Network, utilizing TensorFlow library
- Model training and hyperparameter selection, for choosing an optimized model build
- Analysis of Regularization implementation, such as Ridge and Lasso Regression
- Experimentation with data augmentation
- Best model selection based on applying results of regularization and hyperparameter seleciton

Data Preparation

The dataset and initial preparation used for this model are like that of the implementation in project 1.

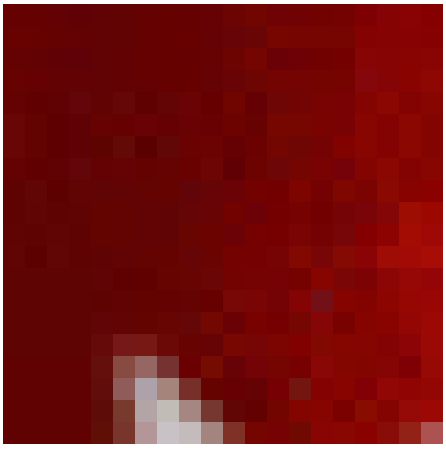
The dataset used for this projected as pulled from github page for face resources, https://github.com/betars/Face-Resources [1]. The 7th dataset listed contains a Face Detection and Data Set Benchmark, of 5k images which is

sufficient for our project, found at http://vis-www.cs.umass.edu/fddb/ [2]. This dataset was chosen is it contains the annotations for images, and a helpful Readme with instructions, to get started in cropping and creating the needed face and non-face image sets.

Once the dataset was selected, it was used to create two folders each of 1100 images, one for faces and another for non-faces, both at 20 x 20 resolution, as RGB images. All training face images, and test images were separated within the python program to quickly and easily change training testing split.

Example:

a)  b) 

c)  d) 

1. a) Original image from FDDB dataset b) used annotation from FDDB dataset to crop face image
   c) Original image from FDDB dataset d) randomly selected negative image that doesn't overlap face

The LeNet 5 standard network architecture accepts a data set of 28x28 sized images. Therefore, the images are resized using cv2 and a ratio of 1.4 to match the network requirements. Colors are also read in BGR, explaining coloration of images below.

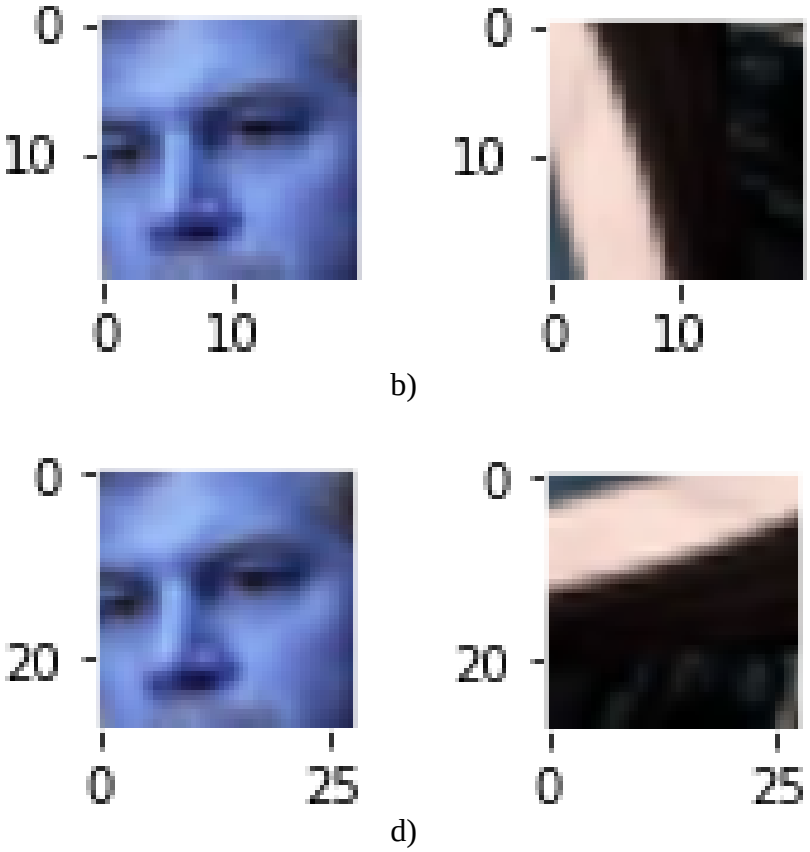


Figure 2. a) Original positive image for 20x20 scale b) Original negative image for 20x20 scale
c) Resized positive image for 28x28 scale d) Resized negative image for 28x28 scale

A padding of 2 pixels is added to the image x and y dimensions. This is done to help with increasing accuracy for images, as the convolutional network uses a kernel as a filter scans an image and converts it to a different array format within the network.

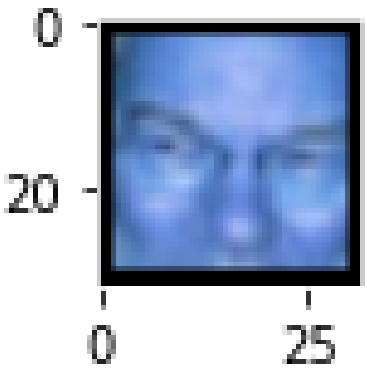


Figure 3. Positive image after adding a padding of 2 pixel to each side of the image.

Finally, the image dataset is split between a training, validation, and testing array. Other associated arrays are created to contain the labels for each image in the splits. This split ratio can be specified in the code structure for the data_prep() method, however, for this demonstration we chose a 80% validation split and a 90% testing split. For hyperparameter testing, the model used 220 testing images, 296 validation images, and 1584 training images. Data augmentation techniques affect the sample splits; this is evaluated further in the data augmentation section.

LeNet 5 Neural Architecture

This project uses a LeNet 5 architecture instead of a 3 layered Fully Connected network. This is because this project build allowed for usage of TensorFlow library which is popular framework in industry, and it is a good one to learn and gain experience with. The LeNet 5 also is a classic Convolutional Neural Network that is specialized for computer vision applications, and the accuracy would most likely be better than a simple FC network.

The Tensorflow tutorial provides many template builds for networks such as the LeNet 5. Tutorial code and descriptions of all their tensorflow layer functions was used as a reference from their online material [3]. Tutorials are used as a reference for the following build, while hyperparameter testing and customizations are personalized.

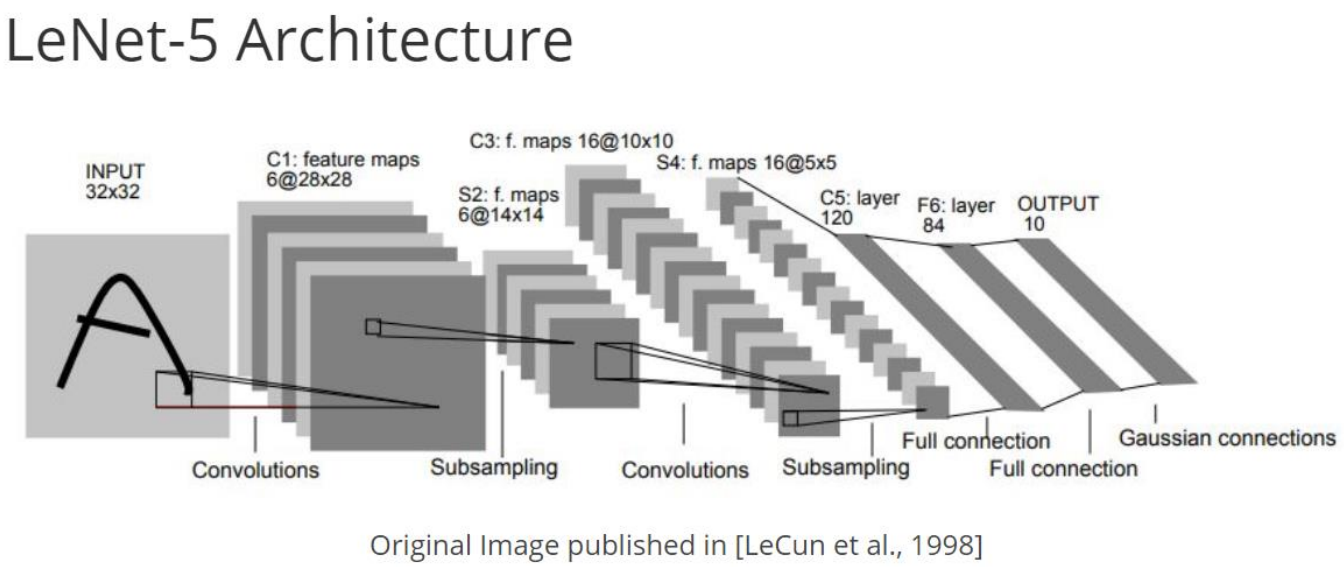


Figure 4. Reference of template LeNet-5 Architecture

While TensorFlow has a useful visualization toolkit named TensorBoard that allows users to view automated model structure and array sizes of build models, the functionality of this toolkit is mainly specific to Keras builds with Tensorflow 2.0. I decided to create the model using lower level TensorFlow functions without the Keras wrapper to maintain integrity of this project and make customizations for babysitting method. Therefore, without this visualization toolkit I will display the LeNet 5 model summary in Table 1.

Table 1. Summary of Customized Model from LeNet 5 Architecture

Layer		Feature Map	Size	Kernal Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	Relu/Relu6/Selu
2	Avg. pooling	6	14x14	2x2	2	-
3	Convolution	16	10x10	5x5	1	Relu/Relu6/Selu
4	Avg. pooling	16	5x5	2x2	2	-
5	FC0 (Flatten)	-	400	-	-	-
6	FC1	-	120	-	-	Relu/Relu6/Selu
7	Dropout	-	.1/.2/.3/4	-	-	-
8	FC2	-	84	-	-	Relu/Relu6/Selu
9	Dropout	-	.1/.2/.3/4	-	-	-
Output	FC3	-	2	-	-	Sigmoid

Note that the final activation function is chosen as a sigmoid to handle binary classification as opposed to a multiple classification problem. Furthermore, the Dropout layers are added as a means of testing hyperparameter selection. Furthermore, the activation function is variable as it is tested as a hyperparameter in model training.

Model Training and Hyperparameter Selection

Multiple hyper-parameters were tuned in this process. The ones chosen to look at are (A) number of batch size, (B) learning rate, (C) dropout rate, and (D) activation function. These are manually selected by changing one hyper-parameter at a time while keeping the others fixed to the baseline hyper-parameters. The baseline hyper-parameters are 100 epochs, batch size of 64, a learning rate of 0.001, 0.3 dropout rate, and ReLU as the activation function. Using the baseline parameters, the initial model is created by overfitting a small sample. Hyperparameters are then evaluated based on how they improve accuracy and handle overfitting.

A. Batch Size

As batch size increases, the accuracy decreases slightly. The higher batch size does provide a more consistent testing accuracy than that of a lower batch size. The choice of this hyperparameter is a tradeoff that likely depends mutually on other hyperparameter selections.

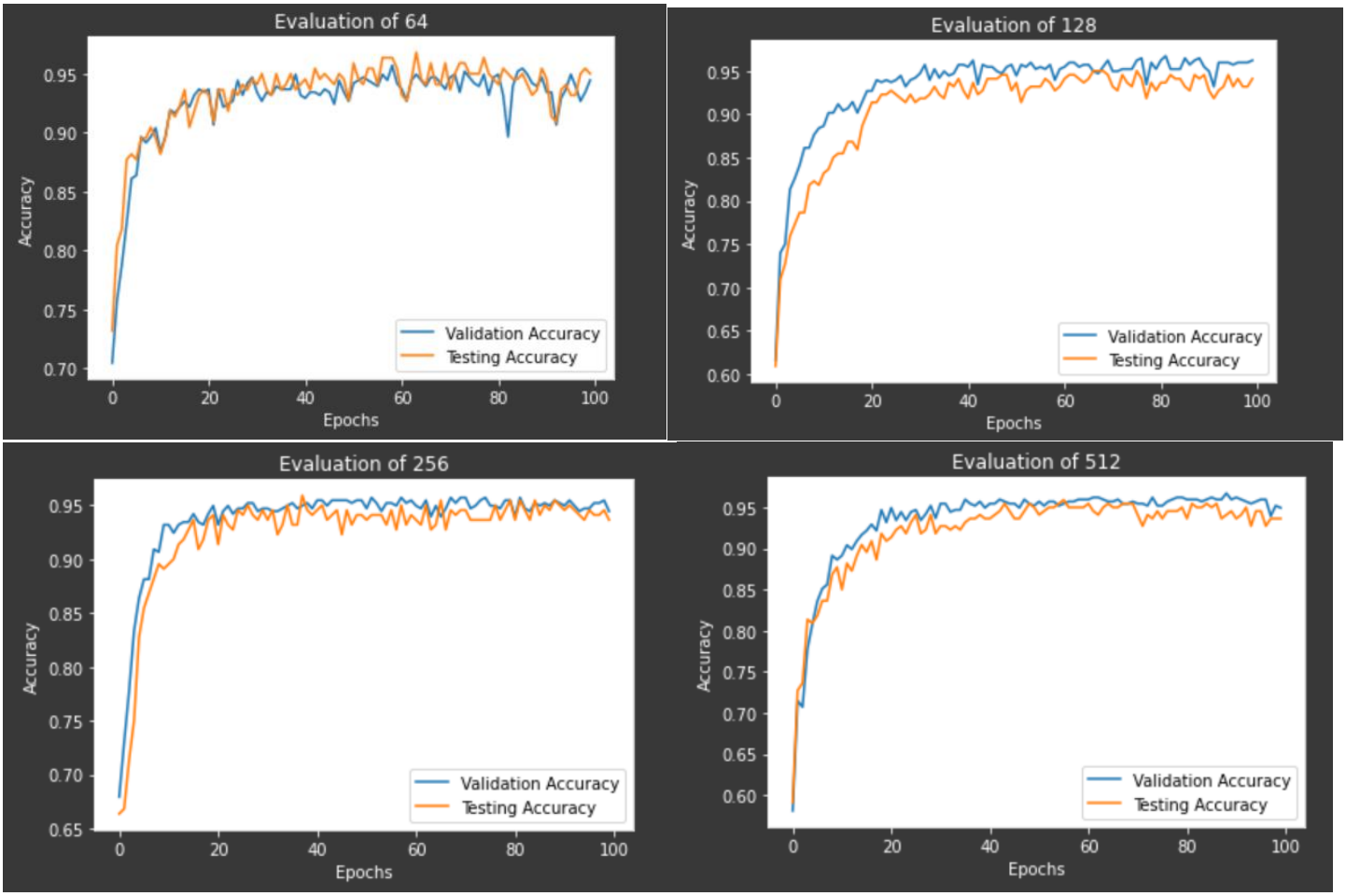


Figure 5. Evaluation of validation and testing accuracy for Batch Sizes of 64, 128, 256, and 512.

B. Learning Rate

When it comes to learning rate, it seems that the lower end of our testing range performed better. This did take longer computational time for training, however, it is worth it for the stability in testing and validation accuracy, as higher learning rates tend to create inconsistency in model performance.

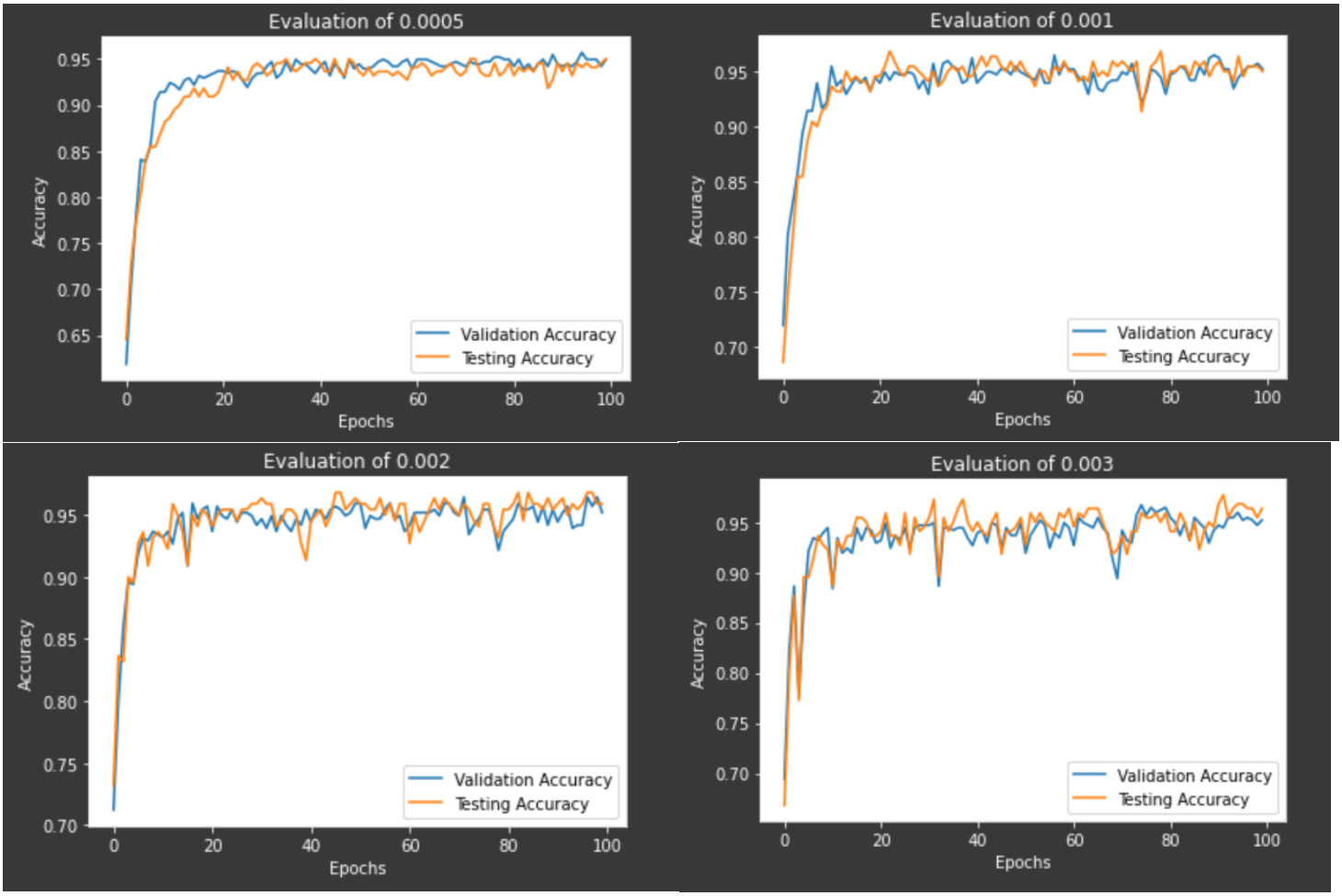


Figure 6. Evaluation of validation and testing accuracy for Learning Rates of .0005, .001, .002, and .003.

C. Dropout rate

Dropout rate primarily had an impact on the stability of accuracy in our model performance, and its ability to handle overfitting in training epochs. Trends in dropout rate performance were not so clear, as it likely depends mutually on other hyperparameter selections that affect overfitting, i.e. learning rate and batch size. We notice that the model training is more stable for a dropout rate of .1 and alternatively .3, which was our default value.

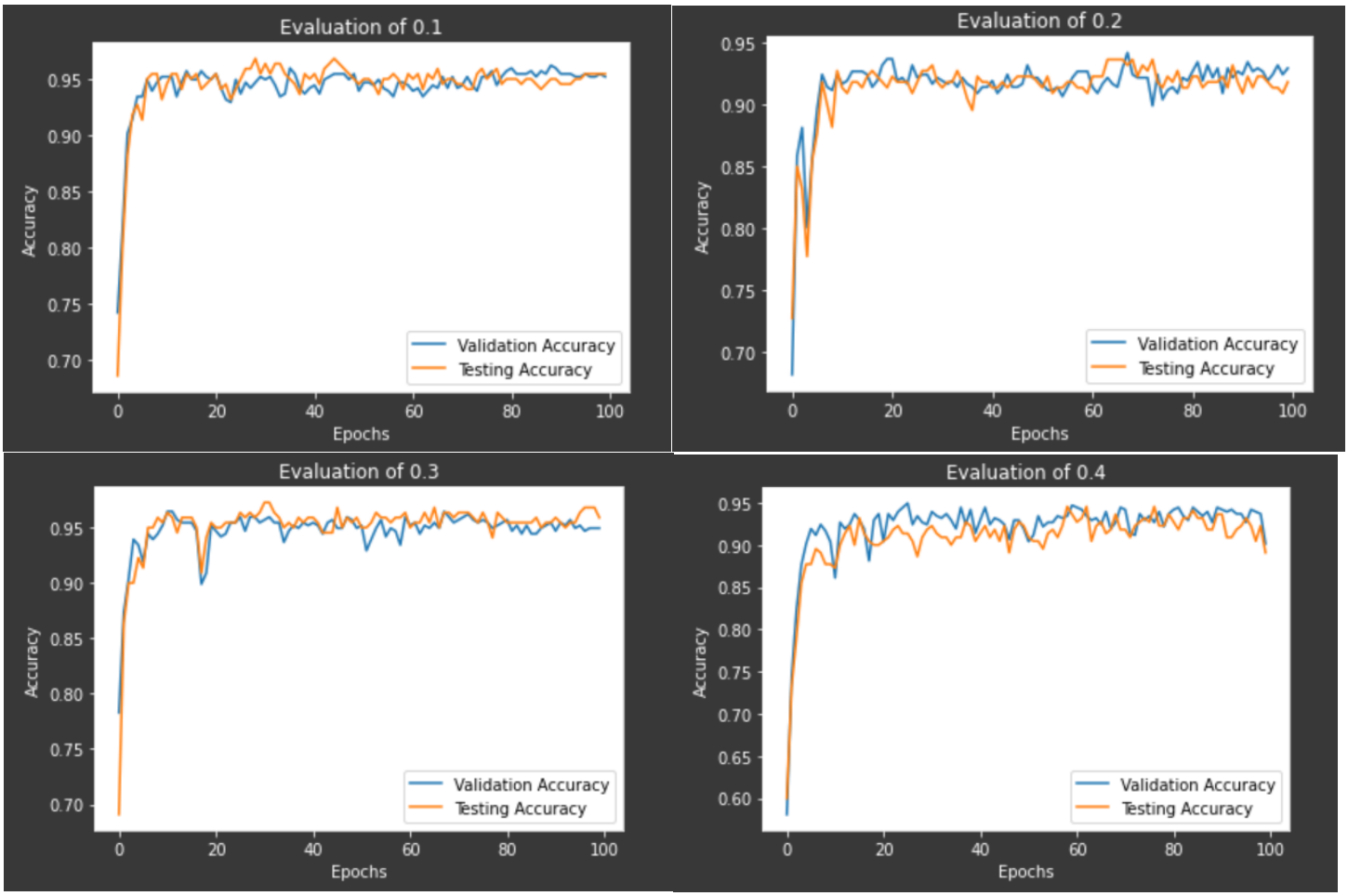


Figure 7. Evaluation of validation and testing accuracy for Dropout rates of .1, .2, .3, and .4.

D. Activation function

The choice of activation function also has a large impact on the model and is specific to the network structure. ReLU is standard for the LeNet 5, however, we decided to test also the SeLU and ReLU6, which were common ReLU alternatives provided in the tensorflow package.

The benefits of SeLU is that there is no issue with vanishing gradients, and that they are known to work fast and accurately [4]. In this test we see that the SeLU can provide significant boosts in testing accuracy, compared to the standard ReLU.

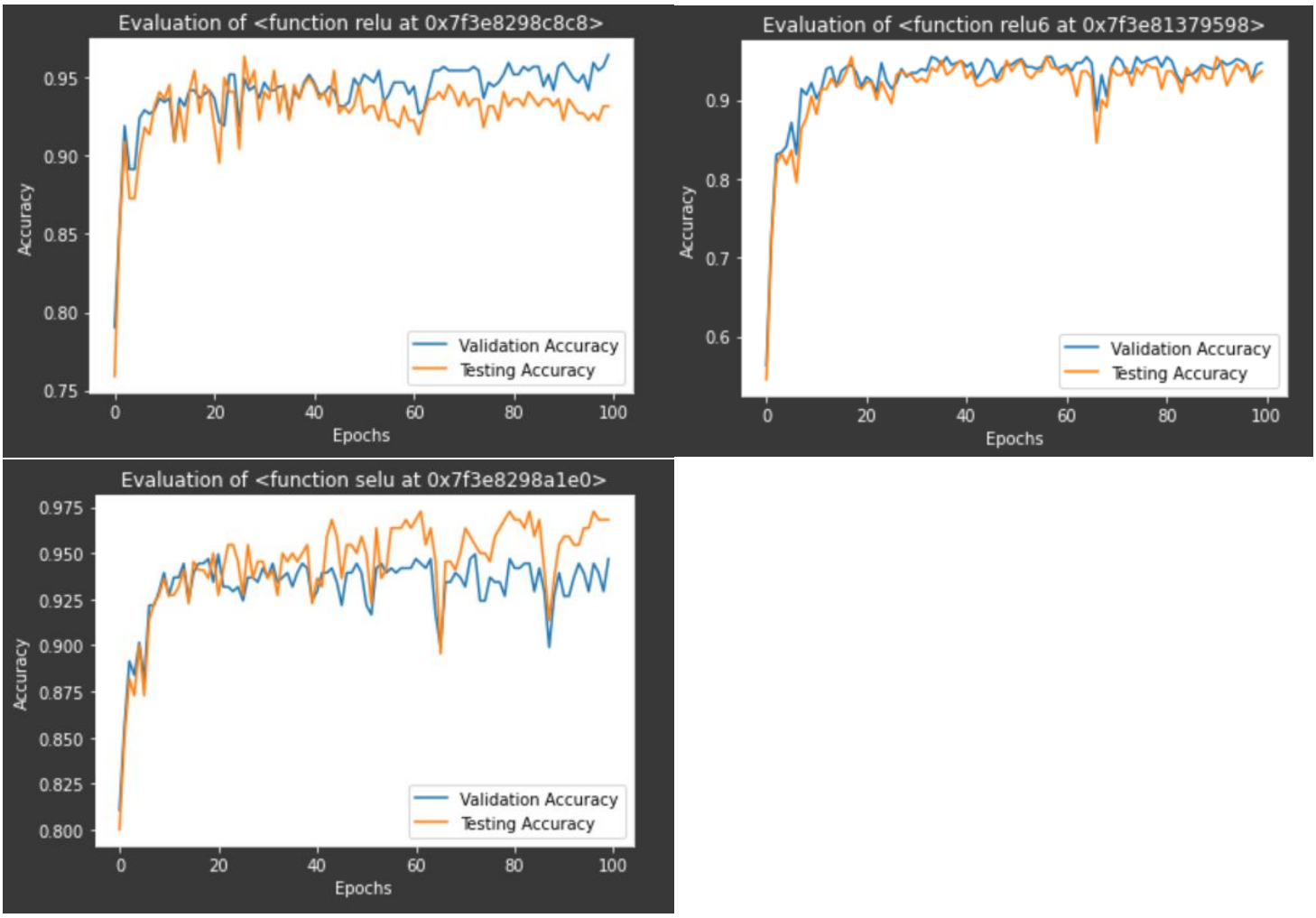


Figure 8. Evaluation of validation and testing accuracy for Activation Functions of ReLU, ReLU6, and SeLU.

Further Analysis of Regularization

Methods for regularization include modifying the cost function to penalize overfitting when performing regression tasks. This regression is done using the common ridge or lasso regression, as they are simple to implement and effective. Implementing these regression terms in the TensorFlow are done as follows.

A. Lasso Regularization

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

```
lasso_param = tf.constant(0.9)
heavyside_step = tf.truediv(1., tf.add(1., tf.exp(tf.multiply(50., tf.subtract(A, lasso_param)))))
regularization_param = tf.multiply(heavyside_step, 99.)
loss_operation = tf.add(tf.reduce_mean(cross_entropy), regularization_param)
```

B. Ridge Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad \text{(Weight decay)}$$

```
    ridge_param = tf.constant(1.)
    ridge_loss = tf.reduce_mean(tf.square(A))
    loss_operation = tf.expand_dims(tf.add(tf.reduce_mean(cross_entropy), tf.multiply(ridge_param, ridge_loss)), 0)
```

As shown, the regularization adds the cost term to the existing cross entropy loss function to implement a regression type. In this project we test the use of Lasso, Ridge, and no regularization terms by spiking learning rate to create an apparent issue of overfitting in training. We notice that Ridge has the best affect in handling overfitting issues with the model, by helping stabilize model validation and test accuracy across epochs.

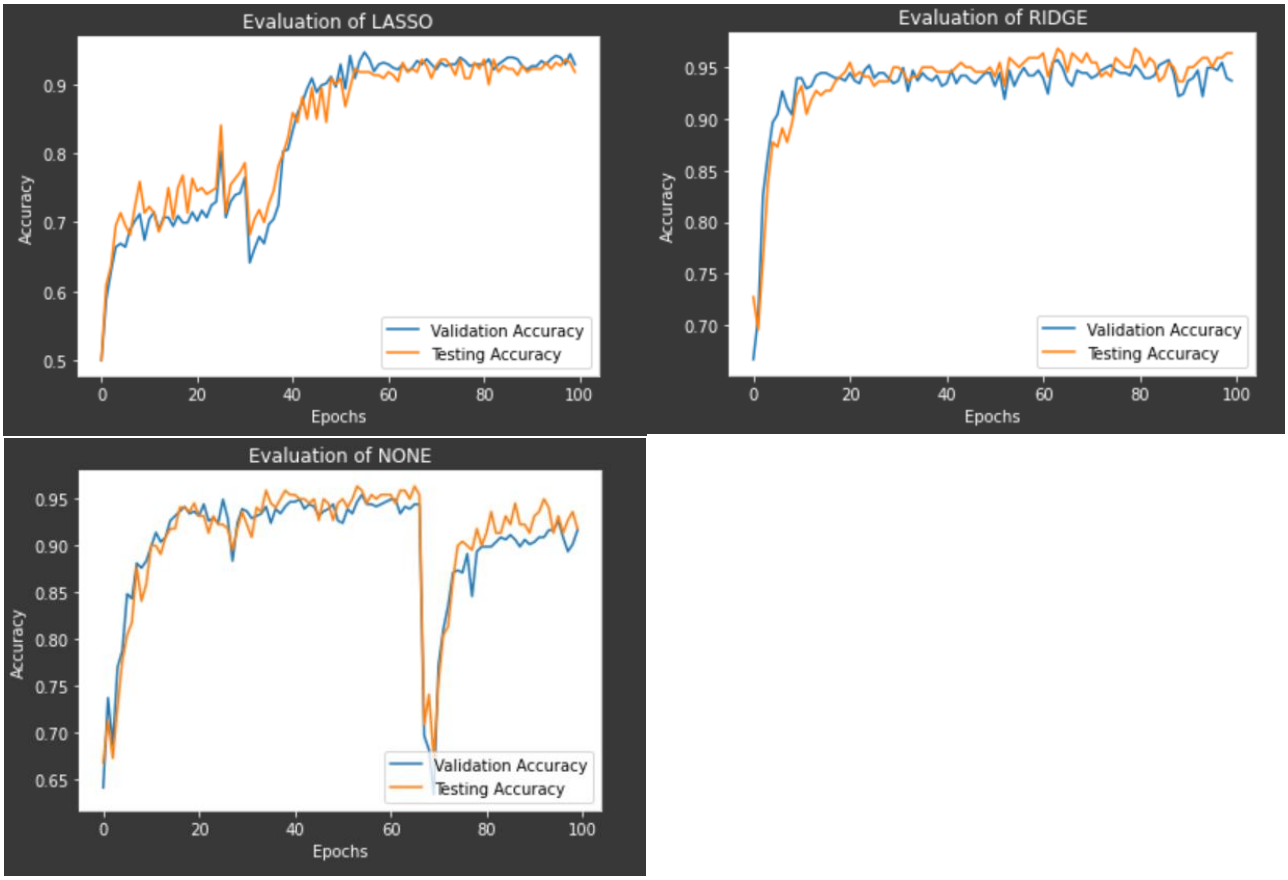


Figure 9. Evaluation of validation and testing accuracy for regularization types of Lasso, Ridge, and None.

C. Data Normalization

The data normalization is almost always a good idea for classification problems. Classification loss can be sensitive to changes in weight matrix and hard to optimize. The process of normalization makes data less sensitive to small changes in weights and easy to optimize. The purpose is to center the data on a mean of zero and equal variance. For our image data, we use a term of 128 to approximately normalize our data.

```
X_train = (X_train - 128)/128
X_valid = (X_valid - 128)/128
X_test = (X_test - 128)/128
```

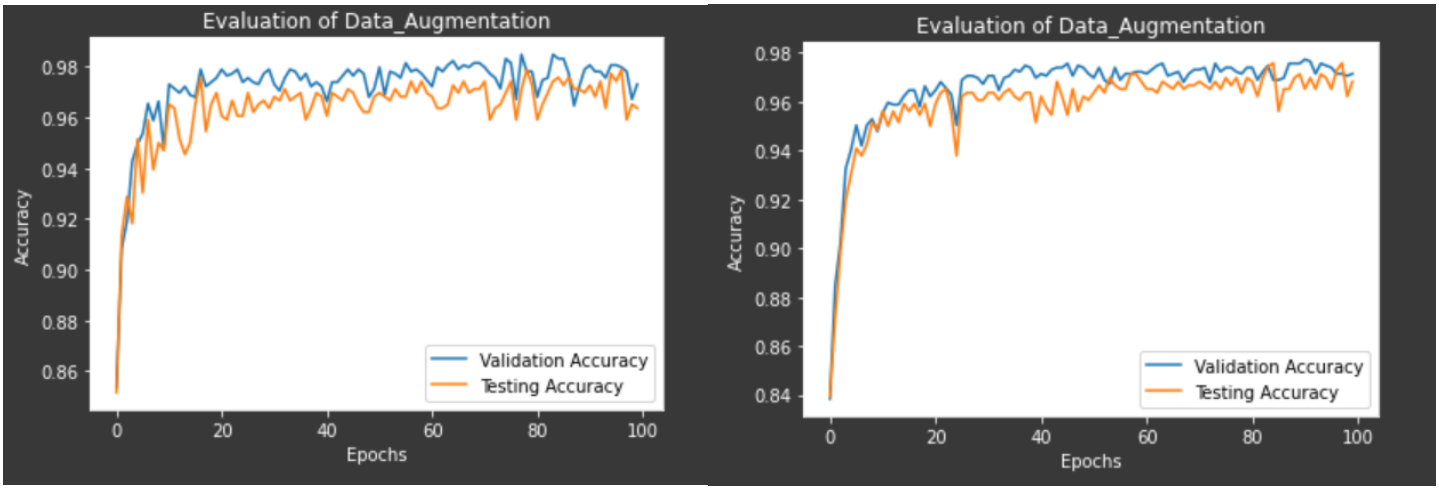


Figure 10. Evaluation of validation and testing accuracy for Data Normalization effects. The left graph shows model accuracy without normalization compared to the right graph with normalization. Note that the accuracy does improve marginally. For applying normalization to non-optimized models, the effects are more visible.

E. Data Augmentation

Data augmentation is another technique that is almost always beneficial to test accuracy. For this project it is especially effective because our data set is small. Also, augmentation techniques are easy provided the nature of our classification problem. Before augmentation, our project had 1100 positive and 1100 negative samples. For positive images, we decided to employ a horizontal flip for all images, doubling the size of positive samples to 2200. This is an effective augmentation trick because classification needs to work for faces that have turned their head to the left or right – the use of a vertical flip or rotation would not have been effective. For negative samples, our augmentation technique is more versatile. This is because the negative samples are essentially random crops of 20x20 images that do not include a face. To increase negative sample size, we rotate all negative samples by 90, 180, and 270 degrees, effectively quadrupling the negative sample size. Summary of the augmented data set are shown in Figure 11.

```
Total samples before Data Augmentation:
Positives: 1100
Negatives: 1100
Total samples after Data Augmentation:
Positives: 2200
Negatives: 4400
```

Figure 11. Summary of total samples before and after augmentation techniques.

```
Number of Classes: 2
Training Set:   4752 samples
Validation Set: 1188 samples
Test Set:       660 samples
```

Figure 12: Summary of training, validation, and testing split after shuffling all 6600 augmented samples.

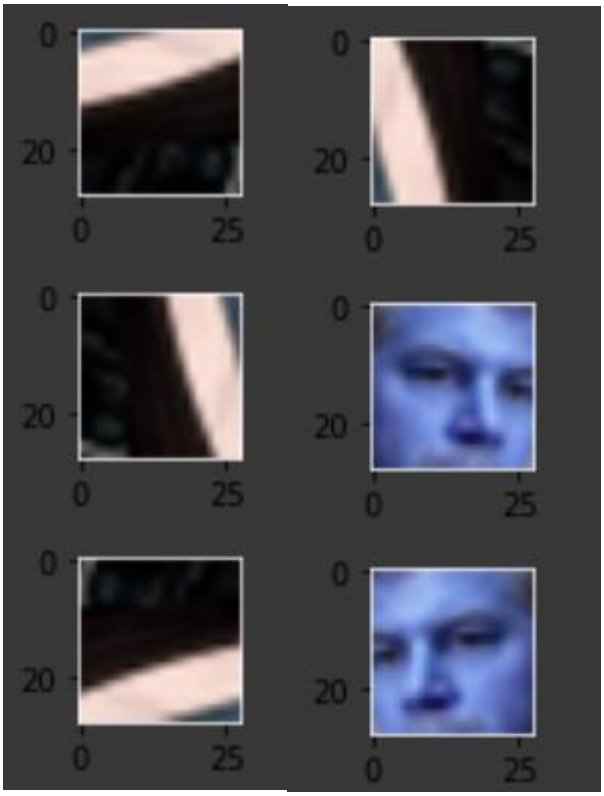


Figure 13: Display of negative sample being rotated multiple times to increase negative sample data set by factor of 4. Display of positive sample being horizontally flipped once to increase positive sample data set by factor of 2.

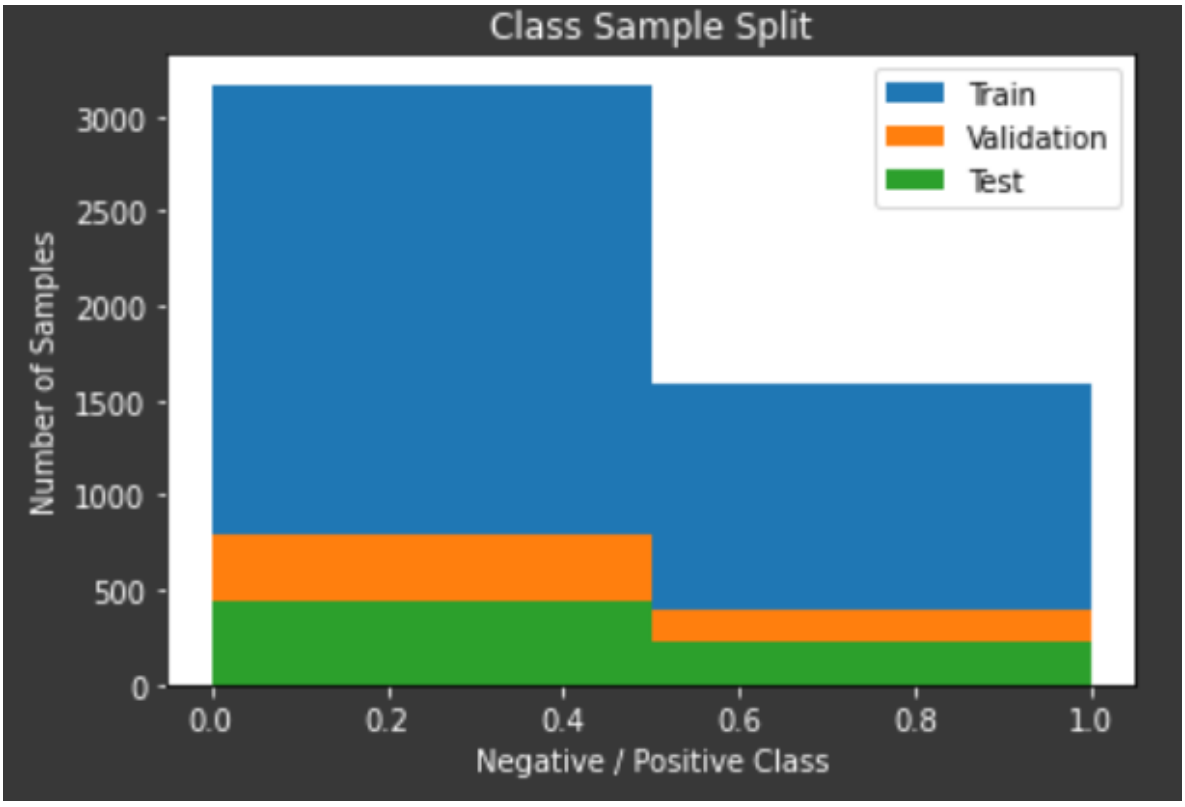


Figure 14: Visualization of data skew in positive and negative samples introduced after augmentation. Large skews can create bias in classification problems. In this case, the need of additional data samples outweighs the issues introduced by the skew, as we still see an increase in model accuracy.

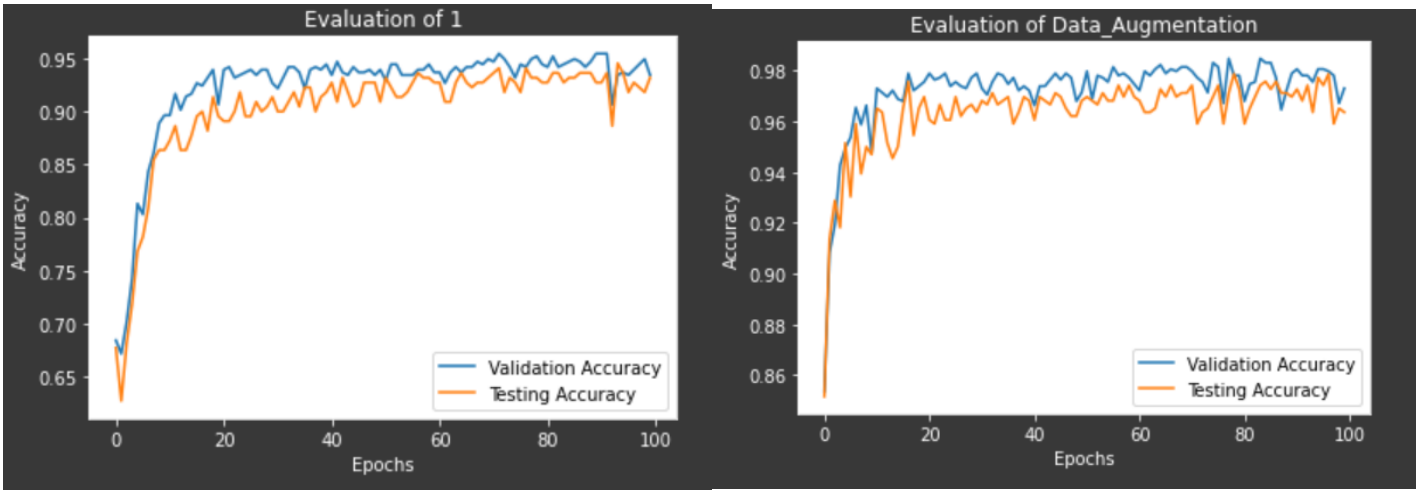


Figure 15: Evaluation of model performance before data augmentation (left) and after data augmentation (right).

Notice that the addition of augmented data samples effectively improves the test accuracy by about 4%. This augmentation included methods that were easy to implement, such as image rotations and flips. More technical approaches would include using auto encoders or generative models to create holistically new samples from the dataset provided. Overall, we note that usage of data augmentation is very effective in increasing performance of models for small data sets.

F: Dropout Layer

The purpose of this layer is to prevent the co-adaptation of features by introducing a forced randomness in training The dropout rate was measured from .1 to .4 in the section of hyperparameter tuning to evaluate the effects of the parameter in the model. Here, the importance of the dropout layer is further expanded upon, by displaying an image subject to strong overfitting without any dropout layers, and then introducing a dropout of .3. We notice that the testing accuracy of the model is increased by an additional .7% for an already optimized model.

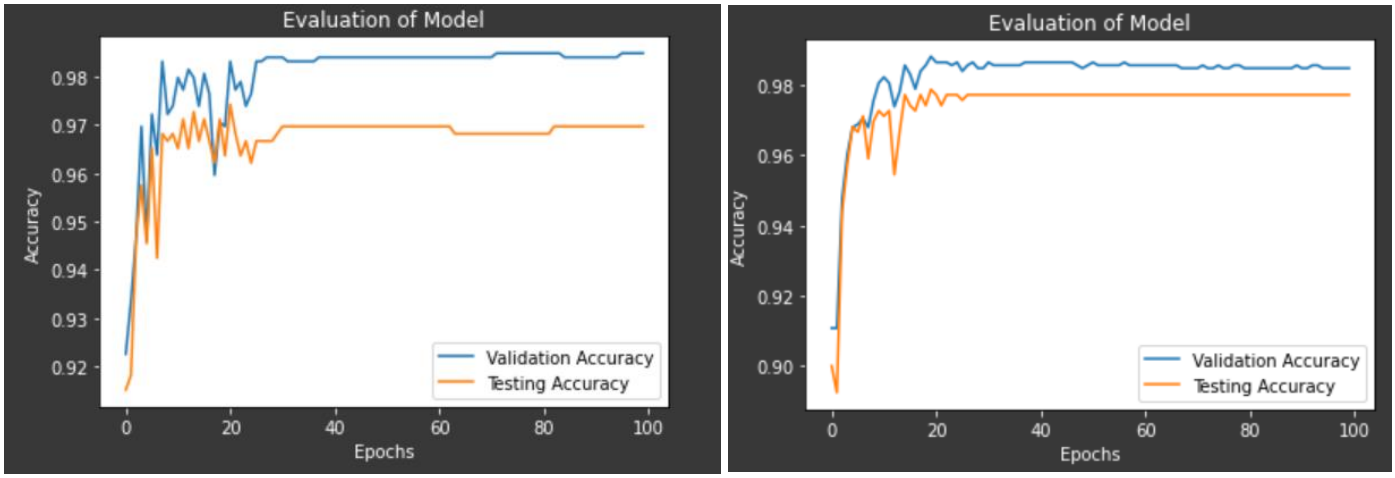


Figure 16. Evaluation of dropout layer, with no dropout on the left, and a dropout with a .3 rate on the right.

Best Model Selection

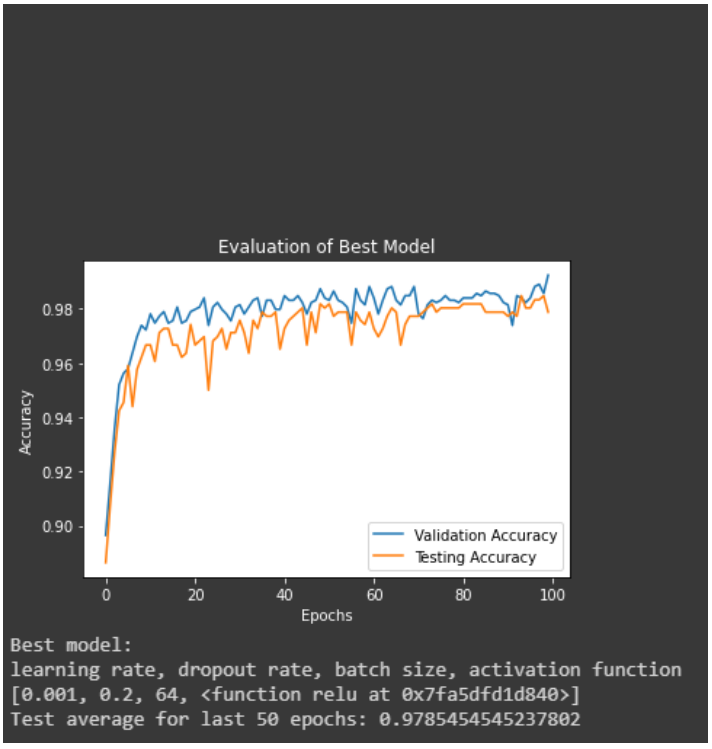
After performing our further analysis of regularization, we aim to optimize the LeNet 5 accuracy. Our final model evaluation is based on the following assumptions from our baby-sitting process:

- A Ridge Regression is selected, as this proved to be most effective in handling overfitting.
- Data normalization is applied in data preparation
- data augmentation is applied in data preparation

The hyperparameters are then tested in a nested For-Loop, evaluating 4 effects at 3 different intervals for a total of 81 models. The optimal model is saved based on the mean of the last 50 epochs, to ensure that best model selection is based on consistency of high accuracies. Our final model is shown in Figure 16 (a).

Reviewing the evaluation of the other 81 models, another model shown in Figure (b) had a much lower variance in test accuracy, while sacrificing only .2% in test accuracy average. Therefore, this proves to be another optimal model for selection.

a)



b)

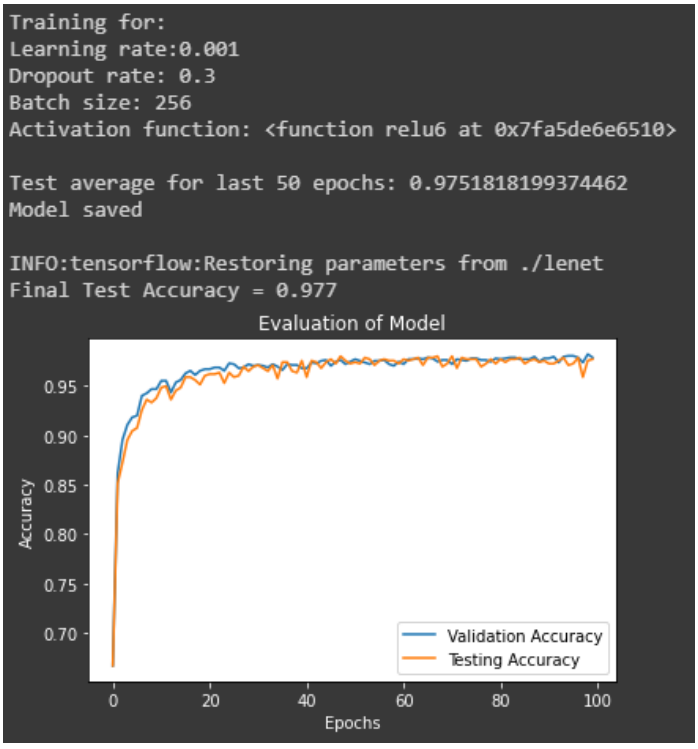


Figure 16. A) Best model selection based on highest average of last 50 epochs, of 97.8% test accuracy.
B) Best model selection based on both low variance and high average accuracy, of 97.5% test accuracy.

Conclusion

The tutorials and libraries provided by frameworks such as Tensorflow are great starting point for developing deep neural networks in computer vision classification problems. While the model structure may stay the same as a tutorial build, almost always there will need to be some data preparation to feed the template network a modified pipeline of the specific dataset in use. In this respect a knowledge of data structures, arrays, Numpys library, and Python come in use. Furthermore, to create a competitive model in industry, these templates will always require fine tuning. Practices discussed in ECE 763 and the baby-sitting procedure are essential to producing a highly accurate model. Hyperparameter selection is often dependent on the dataset, and thus parameter values will vary. We note that some procedures are almost always useful in increasing model accuracy, such as data augmentation, data normalization, and regularization. By applying these methods and hyperparameter tuning, we reach a test accuracy of about 98%, a strong 10% improvement from the previous project utilizing the Viola-Jones object detection framework. Thus, the simplicity and effectiveness of the Deep Neural Network in comparison to previous mathematical models and object detection frameworks discussed throughout the course of ECE 763 make the DNN a strong and simple tool for Computer Vision applications.

References

[1] Betars, "betars/Face-Resources," *GitHub*, 27-Apr-2017. [Online]. Available: https://github.com/betars/Face-Resources. [Accessed: 27-Feb-2020].

[2] "Face Detection Data Set and Benchmark Home," *FDDB : Main*. [Online]. Available: http://viswww.cs.umass.edu/fddb/. [Accessed: 27-Feb-2020].

[3] "Module:tf:TensorFlow Core v2.1.0," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/. [Accessed: 15-Apr-2020].

[4] T. Bohm, "An Introduction to SELUs and why you should start using them as your Activation Functions," *Towards Data Science*, 28-Aug-2018. [Online]. Available: https://towardsdatascience.com/gentle-introduction-to-selus-b19943068cd9. [Accessed: 15-Apr-2020].