COMP 551 Assignment 3 Write-up

Faith Ruetas, Joey Marten, Gary Zhang

December 5, 2022

Abstract

We implemented a Multi-Layer Perceptron (MLP) neural network, then ran our model on the Fashion-MNIST (Fashion) dataset. After preprocessing the data and training the model, we found that our MLP best classified the Fashion image data at a rate of 88.31% with 2 hidden layers of 128 units each, a learning rate of 0.32, a batch size of 125, and 38 epochs. Though we performed multiple experiments to determine these optimal hyperparameter values, our model was outperformed by a Convolutional Neural Network (CNN) and Residual Neural Network (RNN) implemented using TensorFlow. The latter models reached maximum accuracies of 90.21% and 90.62% respectively.

1 Introduction

In this project, our task was to implement the Multi-Layer Perceptron (MLP) model from scratch. To evaluate the performance of our MLP, we used the Fashion-MNIST (Fashion) dataset, which comprises article images of clothing items.

Overall, our MLP reached its maximum testing accuracy of 88.31% with 2 hidden layers, 128 units each, 0.32 learning rate, 125 batch size, and 38 epochs. Though we performed multiple experiments to find the optimal values for these hyperparameters, we could not surpass our Convolutional Neural Network (CNN) or our Residual Neural Network (RNN), which performed at 90.21% and 90.62%, respectively.

Our work follows pre-existing scholarship on neural networks classifying image-based data. To categorize tree species, for instance, Sumsion et al. found that MLP performed better than vector machines and random forests did[2]. Furthermore, Foody used Multi-Layer Perceptron and radial basis function models to predict land and crop classifications [1].

2 Dataset

The Fashion dataset comprises images of 10 types of clothing items from the online retailer, Zalando. There are 60,000 training images and 10,000 testing images. Each instance is a 28 x 28 greyscale image with an associated integer label from 0 - 9 representing its clothing type: "T-shirt/top," "Trouser," "Pullover," "Dress," "Coat," "Sandal," "Shirt," "Sneaker," "Bag," "Ankle boot."

2.1 Preprocessing

After loading the data into dataframes, we did preprocessing in two ways. For one copy of the training and testing dataframes, we centered the data by performing mean subtraction, then normalized the data using the formula $(x - x_{min})/(x_{max} - x_{min})$, for each column x. For another copy of the dataframes, we normalized the data using the formula $(x - 0)/(255 - 0) = x/255$. The first method uses the column-specific minimum and maximum values; the second uses the fact that the minimum and maximum greyscale values of each pixel are 0 and 255.

2.2 Analysis

To better understand the dataset, we plotted its label distributions. Both the training and testing data had an equal number of instances across each of the 10 classes (Fig 1). We also visualized one instance from each class by plotting the greyscale pixel values alongside the integer and string-value class labels (Fig 2).

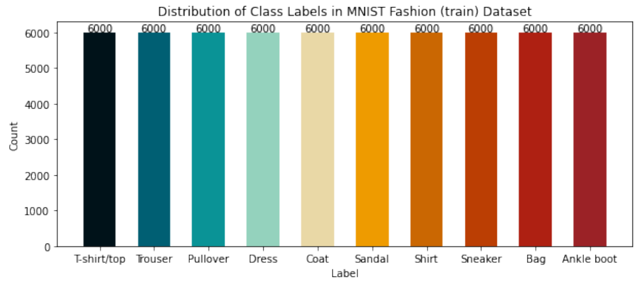

Figure 1: The data was split evenly among the classes

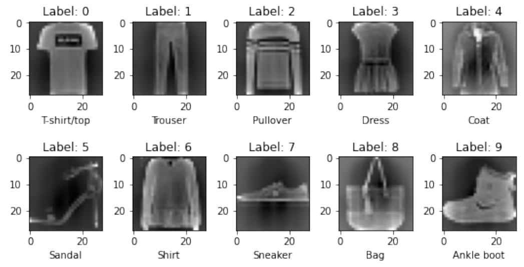

Figure 2: Visualizations of each clothing type

3 Results

3.1 MLP

Our MLP implementation uses linear input layers, non-linear hidden layers, and a softmax output layer to perform categorical predictions. By using mini-batch gradient descent, we cut the maximum memory used when fitting our model to the training data.

3.1.1 Baseline accuracy tests

We performed three baseline tests with varying numbers of hidden layers: one with 0 hidden layers, one with 1 hidden layer of 128 units, and one with 2 hidden layers of 128 units each. We set the other hyperparameters to ReLU activation, 0.1 learning rate, 1000 batch size, and 20 epochs.

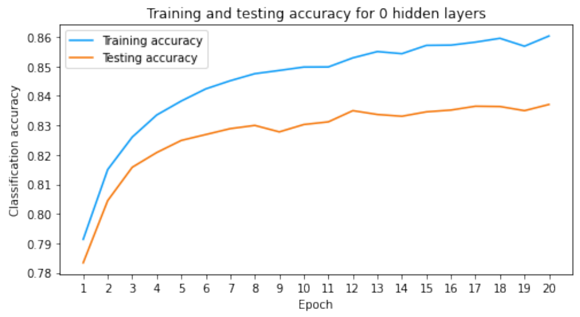

Figure 3: 0 hidden layers: 83.71% testing accuracy

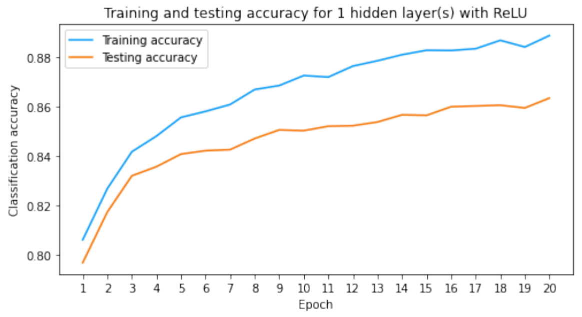

Figure 4: 1 hidden layer: 86.35% testing accuracy

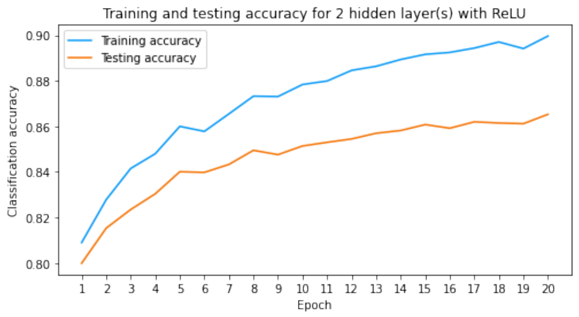

Figure 5: 2 hidden layers: 86.53% testing accuracy

The 2-layer model had the highest accuracy of 86.53%, followed by the 1-layer model and the 0-layer model with 86.35% and 83.71%, respectively (Figs 3, 4, 5). We expected the 2-layer model to perform best due to its larger number of weights and biases, which helps better predict variance in the data. That being said, we did not expect it to only perform 0.18% better than the 1-layer model. While this may suggest that network depth has less effect on performance, this similarity can also be explained by the combination of the relatively standard learning rate and low epoch number. Nevertheless, the non-linearity of these models clearly distinguishes them from the linear 0-layer model, which performed at least 2.64% lower.

3.1.2 Checking gradient

We attempted to perform gradient checks through small perturbations to the linear layer parameters. We believe our layer code is correct as our MLP and back-propagation implementations are based on those from the class Colab. However, we could not find a viable solution as our gradient check attempts produced a significant difference among the layers at a scale of 10^4.

3.1.3 Activation, regularization, and unnormalization tests

We performed three additional tests with models adapted from our baseline 2-layer model: one with Tanh activation, one with Leaky-ReLU activation, one with L2 regularization, and one trained on unnormalized data.

	Tanh	Leaky-ReLU	L2 regularization	Unnormalized training data
Training accuracy	88.76%	89.96%	90.00%	76.58%
Testing accuracy	85.48%	86.51%	86.52%	74.45%

Between the two activation function-based experiments, the Leaky-ReLU model performed 1.03% better than the one with Tanh activation, though both performed at least 0.02% worse than their ReLU counterpart. This is not to say that any given activation function is better than the rest. While ReLU and Leaky-ReLU help solve the vanishing gradient problem, one might choose to use TanH activation if they desire a more consistent decrease in cross-entropy loss, which TanH produced on the Fashion data (Fig 9-12 in appendix).

After adding L2 regularization to our model, its training accuracy increased by 0.03% and its testing accuracy increased by 0.01%, yielding a 90% training and 86.52% testing accuracy (Figs 13 and 14 in the appendix).

To train the model on unnormalized data and still yield meaningful results, we changed the learning rate for this fourth experiment from 0.1 to 0.0001. Even so, the testing accuracy dropped significantly with the model performing 12.1% worse than our baseline model (Figs 15 and 16 in the appendix).

3.1.4 Extra experiments

We performed two extra MLP experiments.

One of these entailed comparing the model's accuracy when trained on 10^k images. As expected, performance increases with the number of available images for training (Fig 17 in appendix). This trend occurs with the exception of a dip in performance when $k = 1$, which can be explained by the unequal proportion of data from each class within those 10 instances. This shows the importance of having a relatively equal data distribution from each class within the training set.

Our second extra experiment used training and testing data normalized by the *possible* minimum and maximum greyscale values of each pixel (0 and 255) instead of the actual minimum and maximum values for each pixel (feature), as discussed in preprocessing. When plotted, the model's loss and accuracy per epoch appear smoother than those of our baseline one (Fig 18 in appendix). However, testing accuracy dropped by 3.96% to 82.57%.

3.2 Finding optimal MLP hyperparameters

One part of this assignment was to come up with an MLP architecture that performs as well as possible. To find the optimal hyperparameter values, we tested our model on different learning rates, hidden layer/unit configurations, epochs, and batch sizes.

3.2.1 Learning rate

We experimented with learning rates ranging from 0.001 - 0.33. As shown below, the highest accuracy occurred when the learning rate was 0.32 for a model with 2 hidden layers (where each layer in this test has 128 units).

	0.001	0.01	0.1	0.2	0.3	0.31	0.32	0.33
0	24.67%	64.2%	78.34%	78.67%	76.87%	79.16%	76.96%	76.46%
1	31.45%	66.17%	79.68%	81.68%	81.97%	82.23%	82.35%	82.33%
2	36.82%	66.67%	79.99%	81.11%	82.25%	81.66%	82.74%	82.6%

3.2.2 Hidden layer/unit configurations

We experimented with different configurations of hidden layers and hidden units. The best-performing combination was 2 hidden layers of 128 units each, as expected.

	[32]	[64]	[128]	[32, 32]	[32, 64]	[64, 64]	[64, 128]	[128,128]
Training	81.62%	81.72%	83.16%	78.09%	81.58%	82.38%	82.55%	83.72%
Testing	80.75%	80.12%	82.35%	77.45%	80.67%	81.26%	81.34%	82.74%

3.2.3 Epochs

To test for the optimal number of epochs, we ran our baseline 0-layer model for 50 epochs and identified the point at which training accuracy peaked. Although we could have attained higher accuracy with an even higher epoch number, we factored training time into our consideration of model performance. As a result, we noted a maximum of 84.29% accuracy after 46 epochs (Fig 6).

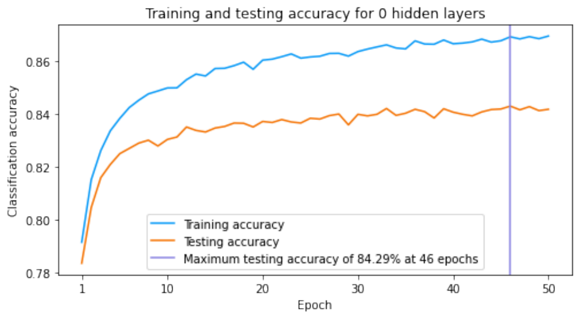

Figure 6

3.2.4 Batch size

We experimented with batch sizes ranging from 125 - 1000. As shown below, the highest accuracy occurred at batch size 125 for a model with 2 hidden layers (where each layer in this test has 128 units).

	125	150	200	500	1000
0	83.11%	83.07%	82.1%	81.73%	80.45%
1	85.29%	85.08%	84.81%	82.83%	81.73%
2	85.33%	85.24%	84.89%	82.71%	81.53%

3.2.5 Optimal MLP architecture

By combining the optimal hyperparameters found in the experiments above (0.32 learning rate, 2 hidden layers, 128 hidden units, 46 epochs, 125 batch size), we were able to improve the performance of our baseline, 2-layer model by 4.6% to 88.31% after 38 epochs (Fig 7). Again, we might have been able to reach a higher testing accuracy with more epochs, but we factored training time into our definition of performance.

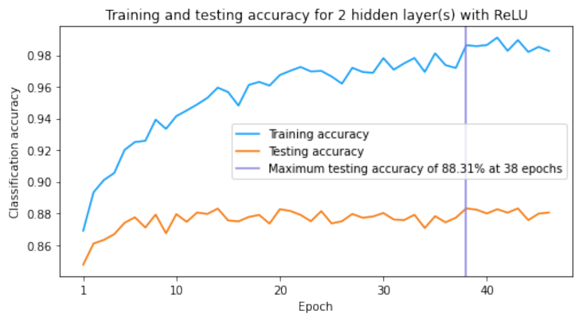

Figure 7

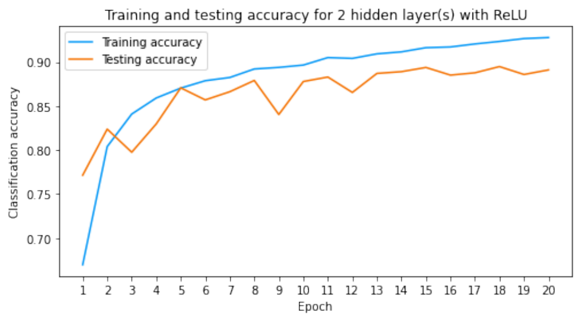

Figure 8

3.3 CNN

We chose the following hyperparameters for each convolution layer: 8 filters, a filter size of 3 x 3, no stride, no padding, and no dilation. The convolution layers were followed by 2 dense layers with 128 units each. Similarly to our MLP model, CNN has a final softmax output layer to perform categorical predictions.

3.3.1 Baseline accuracy tests

The baseline CNN outperformed our baseline 2-layer MLP model by 2.58% with a testing accuracy of 89.11% (Fig 8). It's possible that the filters better captured the characteristics of individual clothing item types, allowing for greater tolerance of different styles or variations of the same class.

3.3.2 Batch normalization

As a baseline for our next test, we chose to train a model using batch normalization. For each convolution layer, we set the activation to 'none' and subsequently added a batch normalization layer and a ReLU activation layer. The dense layer remained the same. With this architecture, the model reached 90.1% testing accuracy.

3.3.3 Residual Neural Network (ResNet)

As an extra experiment, we implemented a Residual Neural Network (ResNet) using TensorFlow's functional API and the Residual Block structure seen in class (Fig 21 in appendix) [3]. After training the model for 20 epochs with a learning rate of 0.1 and a batch size of 1000, we obtained a testing accuracy of 89.4%. This was 2.87% higher than our baseline 2-layer model (which also used 20 epochs, 0.1 learning rate, and 1000 batch size) (Figs 19 and 20 in appendix). With a deeper network and a more complex dataset, this figure is likely to increase.

4 Discussion and conclusion

In summation, our optimal 2-layer MLP model outperformed our initial 2-layer MLP by 1.78%. However, it performed worse than our batch-normalized CNN, which trained for fewer epochs.

In the future, possible directions of exploration include creating deeper models that might benefit from ResNet blocks. Additionally, it would be interesting to see the effect of maximum and average pooling on both the accuracy and training time due to pooling's reduction of parameters. Lastly, we could train a semi-supervised model by taking a subset of the training feature data to use as unlabeled data and comparing its results to our supervised model.

4.1 Statement of contributions

Faith preprocessed the data, performed MLP experiments, and merged work on Github; Joey performed CNN and RNN experiments; Gary made functions for CNN and visualized the data as images.

References

[1] G. M. Foody. “Supervised image classification by MLP and RBF neural networks with and without an exhaustively defined set of classes”. In: *International Journal of Remote Sensing* 25.15 (2004).

[2] Kimball T. Hill G. Rex Sumsion Michael S. Bradshaw. “Remote sensing tree classification with a multilayer perceptron”. In: *PeerJ* 7.e6101 (2019).

[3] Yue Li. “Lecture 20 21 - Module 5.3 Convolutional neural network”. In: *COMP 551 Applied machine learning* (2022), p. 36.

A Appendix

A.1 Testing TanH activation

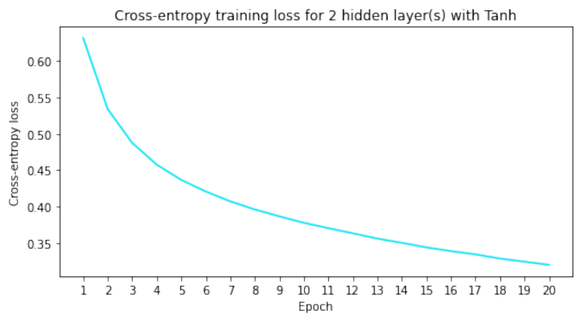

Figure 9

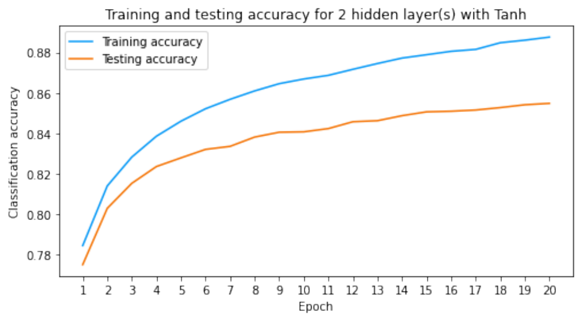

Figure 10

A.2 Testing Leaky Re-LU activation

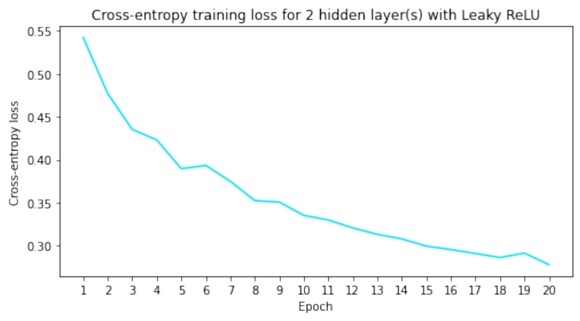

Figure 11

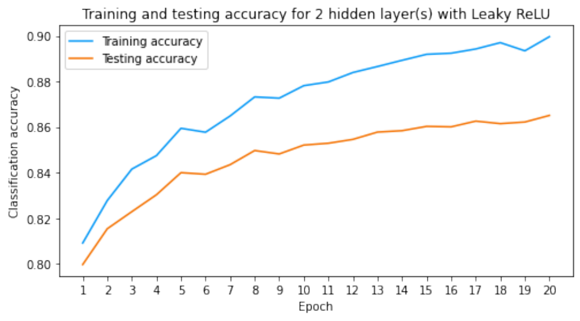

Figure 12

A.3 Adding L2 regularization

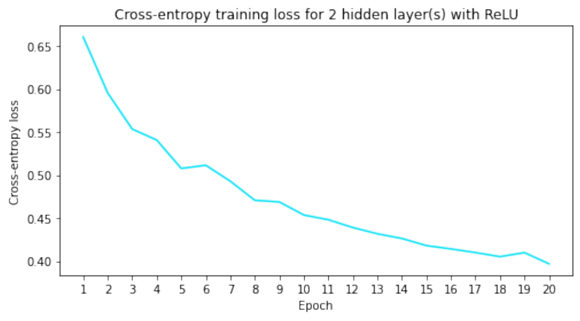

Figure 13

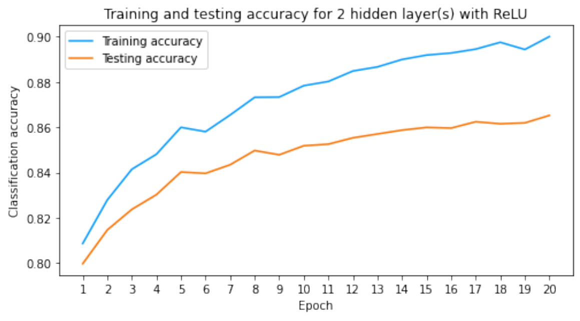

Figure 14

A.4 Using a model trained on unnormalized images

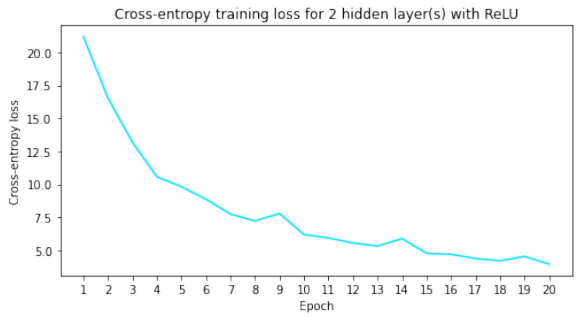

Figure 15

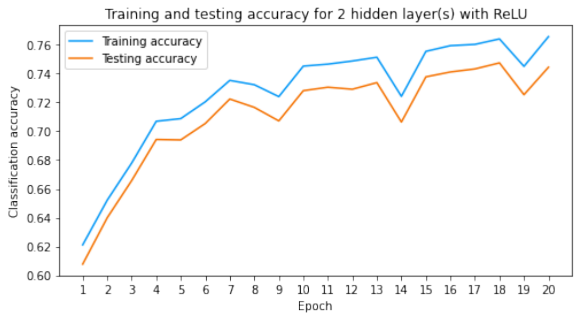

Figure 16

A.5 Extra MLP experiments

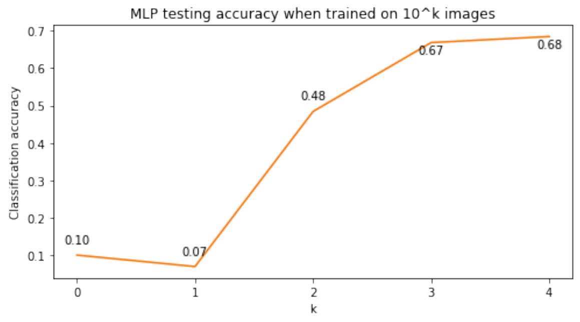

Figure 17

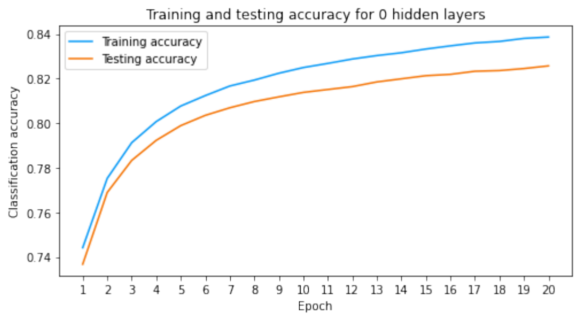

Figure 18

A.6 ResNet

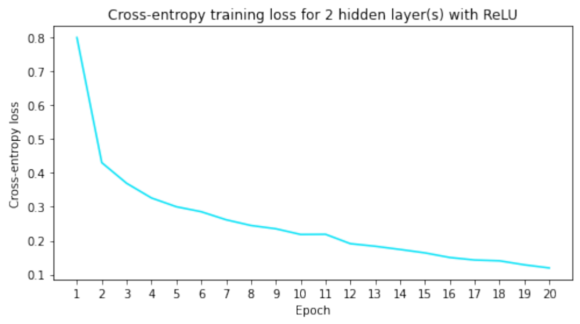

Figure 19

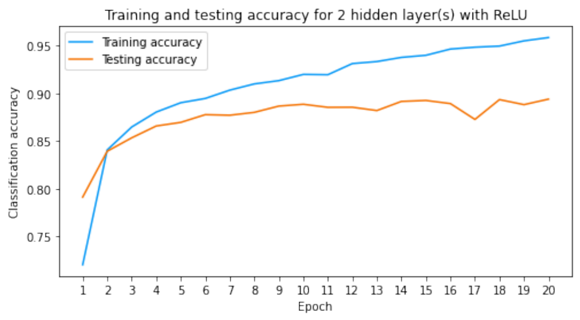

Figure 20

A.7 ResNet Block

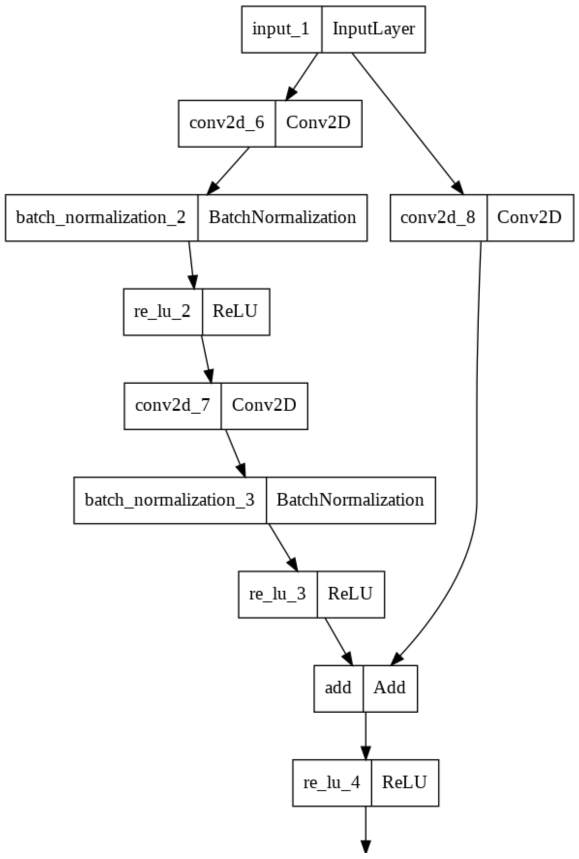

Figure 21: Architecture for our Res Block