6.867: HW3

I. INTRODUCTION

In this report, we study the classification problem (i.e. given training data $D(x_i, y_i)$, how well can we perform classification on unseen data) using simple Neural Networks (NN) and Convolutional Neural Networks (CNN). In Section 1, we report the results of using a homebrew NN to classify low-dimensional data into a small number of classes. In Section 2, we report the results of using a CNN (using the TensorFlow library) to classify art.

II. FULLY CONNECTED NEURAL NETWORKS

For the NN, the architecture is as follows. Input dimension is the dimension of the training data (either $d = 2$ or for the MNIST data $d = N \times N$ where N is the number of pixels in one dimension.) All layers are fully connected. The activation function for any given node in any given hidden layer is the ReLu, $f(z)_{ReLu} = \max(0, z)$. The activation function for an output node is the softmax layer defined as

$$f(z_i)_{SM} = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{1}$$

and the loss function is the cross entropy, which for a given training point (x, y) is

$$L(x, y) = \sum_{i=1}^{k} y_i \log(f(z_i)_{SM}). \tag{2}$$

The backpropagation algorithm is used to train the NN. The gradient of the loss is calculated as

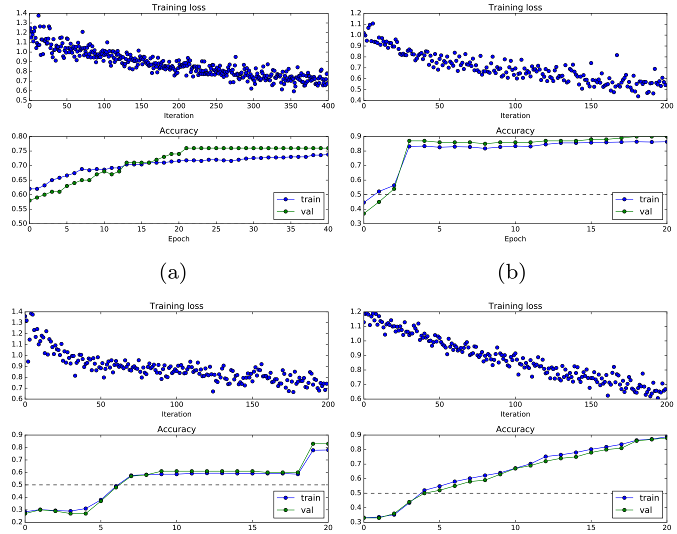
$$\frac{\partial L_j}{\partial z_i} = f(z_i)_{SM} - \mathbb{1}(y_j = i) \tag{3}$$

The gradients of weights (W) and biases (b) for a given hidden layer l are calculated by propagating this loss backwards through the network as

$$\frac{\partial L}{\partial W^l} = a^{l-1}\delta^l \qquad \frac{\partial L}{\partial b^l} = \delta^l \tag{4}$$

where a^{l-1} is the standard affine transformation for layer $l-1$ and δ^l is defined recursively

$$\delta^l = \mathrm{Diag}[\frac{d(f(z^l)_{ReLu})}{dz^l}]W^{l+1}\delta^{l+1}. \tag{5}$$

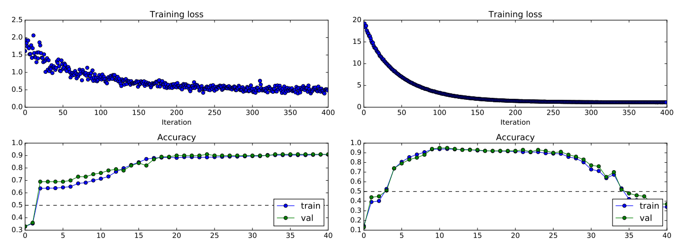


(a) (b)

(c) (d)

FIG. 1: Training on 3-class dataset (a) 1-layer, 2 hidden nodes, (b) 1-layer, 5 hidden nodes, (c) 2-layer, 2 hidden nodes , (d) 2-layer, 5 hidden nodes.

To test that our NN implementation, we use a 3-class, $d = 2$ dataset. Figure 1 depicts the training loss and the training and validation accuracy for several architectures.



(a) (b)

FIG. 2: Training on 3-class dataset (a) 1-layer, 10 hidden nodes $\lambda = 0.01$, (b)1-layer, 10 hidden nodes $\lambda = 1.0$.

The initial guesses for the weights where chosen from a normal distribution centered at zero with a variance of $\delta = 1/\sqrt{d}$. Regularization was implemented using the L_2 norm

$$J(W) = L(W) + \lambda \sum_{l}^{L} ||W^l||_F^2 \tag{6}$$

which in pseudocode looks something like

NN	Current Work Test error	LeCunn 1998 Test error
2 layer, 300 hidden	8.0	4.7
2 layer, 1000 hidden	8.6	4.5
3 layer, 300, 100 hidden	8.8	3.05
3 layer, 500, 300 hidden	7.6	3.05

TABLE I: Comparison between LeCunn's work [1] and current work.

for l = 1, ..., L **do**
  $Loss \leftarrow Loss + \lambda\sqrt{\mathrm{tr}((W^l)^T W^l)}$
end for

The effect of regularization is shown in Figure 2. Setting $\lambda = 1$ was found to over penalize weights, and $\lambda = 0.01$ was found to be suitable based on validation performance. With the functionality of the NN implementation validated, we moved on to the classification problem from the previous assignment (HW2, where we used LR and SVM to perform the classification). In this case 4 datasets with input dimension $d = 2$ where to be classified into $y_i \in \{-1, 1\}$.

When compared with the SVM approach, the NN (at least for the 1 to 3 layer systems studied here) does not perform as well as the RBF kernel, which can achieve classification error below 10% for the second dataset (comparable performance was obtained for the other datasets, see Figures 3 and 4). We also perform digit classification on the MNIST dataset. The training set consisted of 500 samples, and the validation and testing set contained 50 samples.

Some examples of training are shown in Figure 5. The performance on the MNIST dataset is reported in Table I. When compared with LeCunn's website, our implementation performs rather poorly, but this is expected as the all the hyperparameters (learning rate, regulartization, etc.) were not completely optimized.

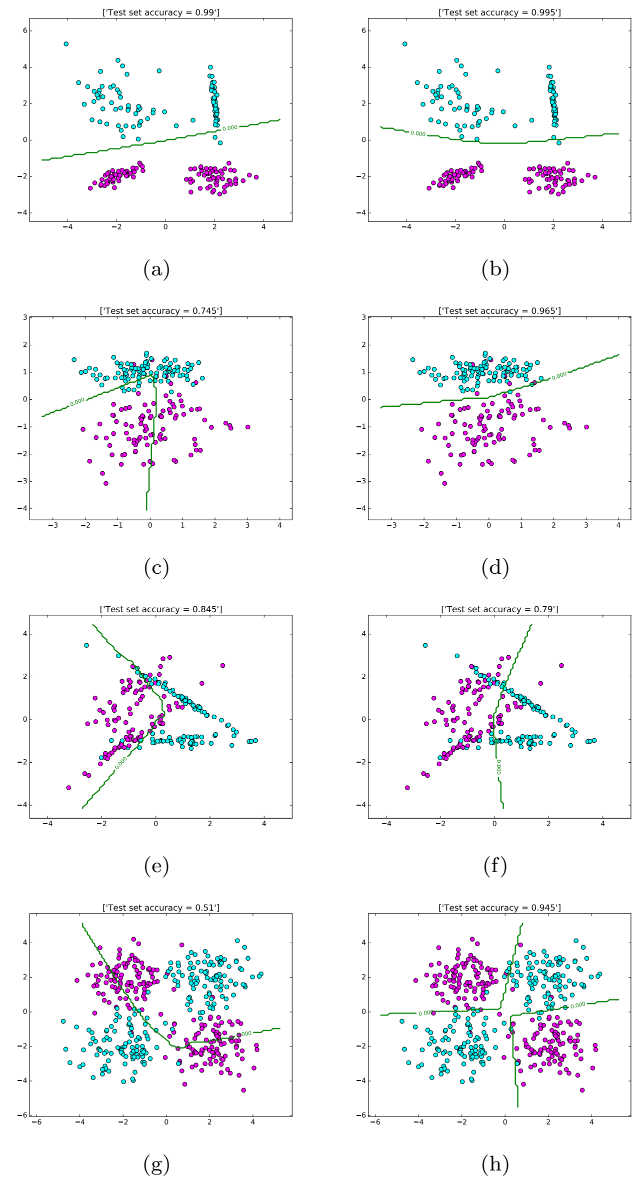


FIG. 3: Example decision boundaries on HW2 test sets (a) dataset 1, 1-layer, 2 hidden nodes, (b) dataset 1, 1-layer, 40 hidden nodes, (c) dataset 3, 1-layer, 2 hidden nodes, (d) dataset 3, 1-layer, 40 hidden nodes, (e) dataset 2, 2-layer, 1000 hidden nodes, (f) dataset 2, 1-layer, 100 hidden nodes, (g) dataset 4, 2-layer, 10 hidden nodes (h) dataset 4, 2-layer, 100 hidden nodes.

III. CONVOLUTIONAL NEURAL NETWORKS

Our next task was to classify 451 works of art according to their respective painter (11 different painters in this case). To do so, we use the TensorFlow library (i.e we have access to objects like `tf.nn` and functions like `tf.nn.softmax`). The first part of this section is concerned with understanding the impact of various hyperparameters on validation accuracy, the results of which are presented in Table II. Quickly, to understand the effect of filter size and stride, we take the following example. Given a 5 by 5 filter with stride 1 that maps input to Z_1 followed by a 3 by 3 filter with stride 1 that maps Z_1 to Z_2, the dimension of Z_2 is determined using $(W - F + 2P)/S + 1$ (W is input size, F is filter size, P is padding, S is stride). So for a 50 by 50 input with zero padding, the dimension of Z_2 will be 43 by 43. The receptive field of Z_2 will be 15 by 15. Thus, as we add more convolutional layers, we creating a hierarchical relationship between the original pixels, thereby allowing us to capture the (spatial) scales of different features.

The result of the original CNN is the first row in the oddly long Table II which will be used as a reference to mark the effect of a given hyperparameter as well as a benchmark for performance. The hyper parameters

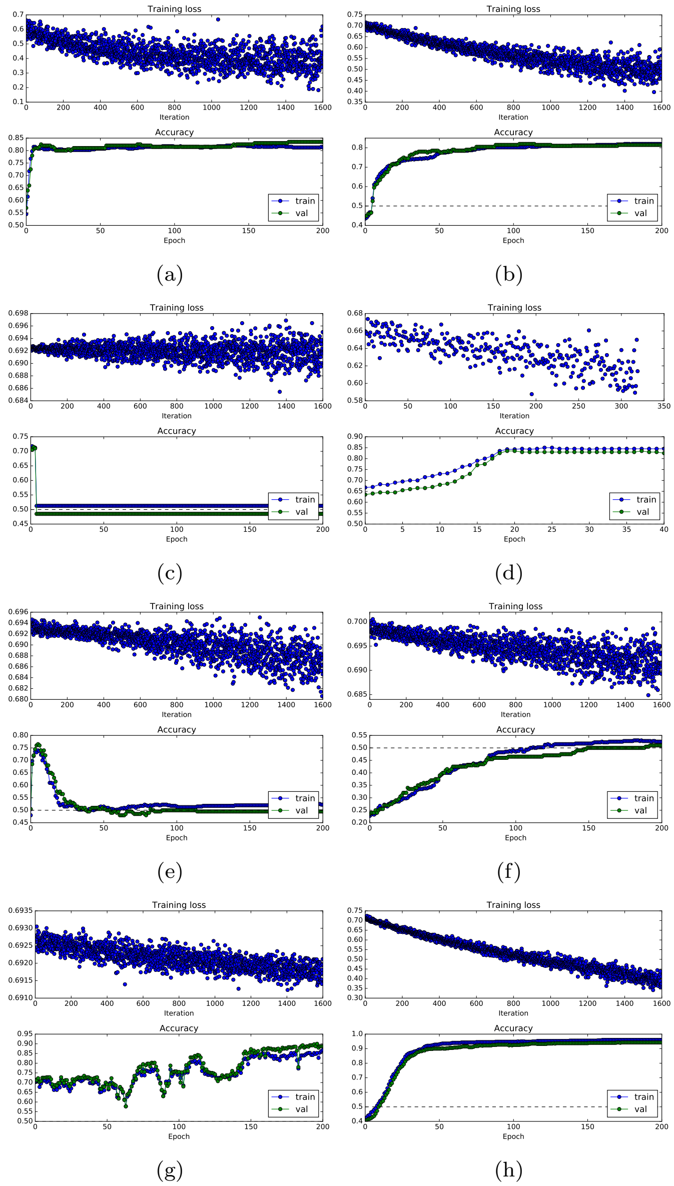


FIG. 4: Training and validation on HW2 datasets (a) dataset 2, 1-layer, 2 hidden nodes, (b) dataset 2, 1-layer, 100 hidden nodes, (c) dataset 2, 2-layer, 2 hidden nodes, (d) dataset 2, 2-layer, 10 hidden nodes, (e) dataset 3, 1-layer, 2 hidden nodes, (f) dataset 3, 1-layer, 5 hidden nodes, (g) dataset 4, 3-layer, 10 hidden nodes (h) dataset 4, 3-layer, 50 hidden nodes.

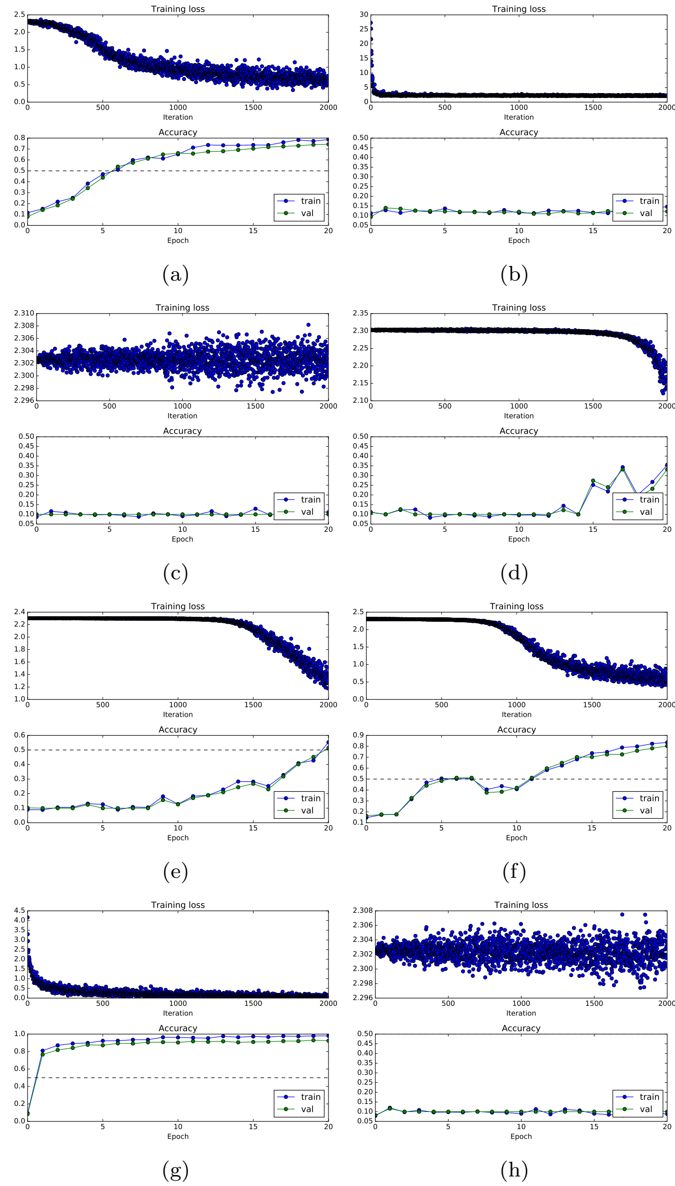


FIG. 5: Training and validation on MNIST datasets (a) 1-layer, 5 hidden nodes, $\delta = 0.1$, (b) 1-layer, 5 hidden nodes, $\delta = 1.0$, (c) 2-layer, 10 hidden nodes, $\delta = 0.01$, (d) 2-layer, 50 hidden nodes, $\delta = 0.01$, (e) 2-layer, 100 hidden nodes, $\delta = 0.01$, (f) 2-layer, 200 hidden nodes, $\delta = 0.01$, (g) 3-layer, 200 hidden nodes, $\delta = 0.1$, (h) 3-layer, 200 hidden nodes, $\delta = 0.01$.

which were varied were the following (indicated by the bold font in the table): the filter size, the depth, the stride length, the addition of pooling layers, the batch size for the SDG, the effect of dropout, data augmentation, early stopping and weight penalization. The first entry in the validation column corresponds to the training accuracy after 1500 iterations (for the augmented data, 6000 iterations were used), while the second line corresponds to the maximum validation accuracy obtained and could be interpreted as a form of early stopping. This itself might be phrased as a machine learning problem, where the inputs are CNNs and hyperparameters are optimized via gradient descent.

Pooling (specifically, pool filter size and pool stride) was found to have improve the validation accuracy, but decrease the training accuracy. Having a filter size too large (10) or too small (1) in the first convolutional layer was found to decrease validation accuracy. Having a larger depth size in the first layer (32 vs 16) was found to increase validation accuracy, however a depth size of 48 did not improve validation accuracy. The opposite scenario of greater depth in the second layer did not improve validation accuracy. Decreasing the stride length of the first layer was found to have negligible effect, but was computationally more costly. Increasing batch size improved validation accuracy. A weight penalty of 0.0005 and training on augmented data did not improve validation accuracy, while a dropout factor of 0.9 and early stopping were generally found to be beneficial. Given that CNNs are (generally) non-linear, non-convex, we have no reason to expect that these hyperparameters are uncoupled and so, while they may not improve the performance independently, together, the proper selection of hyperparameters *can* improve the performance as will be shown in Table III.

Layers	Filter Size	Depth	Stride	Pooling	Aug. Data	Batch Size	Learning Rate	Dropout	Weight Penalty	Training accuracy	Validation accuracy
Conv2d, ReLu	5	16	2	None	None	10	0.01	0.0	0.0	98.6	66.7
Conv2d, ReLu	5	16	2	None	None		GD			98.6	66.7
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	**2,2**	None	10	0.01	0.0	0.0	83.8	69.0
Conv2d, ReLu	5	16	2	**2,2**	None		GD			83.8	69.0
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	**10**	16	2	None	None	10	0.01	**0.9**	0.0	97.5	57.5
Conv2d, ReLu	5	16	2	None	None		GD			91.2	63.2
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	**1**	16	2	None	None	10	0.01	**0.9**	0.0	66.8	47.1
Conv2d, ReLu	5	16	2	None	None		GD			58.8	49.4
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	None	None	10	0.01	0.9	0.0	92.6	70.1
Conv2d, ReLu	5	16	2	None	None		GD			86.8	72.4
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	None	None	10	0.01	**0.9**	0.0	92.9	74.7
Conv2d, ReLu	**4**	16	2	None	None		GD			92.9	74.7
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**48**	2	None	None	10	0.01	**0.9**	0.0	93.7	69.0
Conv2d, ReLu	**4**	16	2	None	None		GD			93.7	69.0
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	None	None	10	0.01	**0.9**	0.0	94.5	67.8
Conv2d, ReLu	**4**	**24**	2	None	None		GD			93.1	70.1
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	None	None	10	0.01	**0.9**	0.0	97.3	62.1
Conv2d, ReLu	**4**	16	2	None	None		GD			93.4	65.5
Fully Connected	**128**										
Fully Connected	**128**										
Conv2d, ReLu	5	**32**	**1**	None	None	10	0.01	**0.9**	0.0	97.0	66.7
Conv2d, ReLu	**4**	16	2	None	None		GD			96.7	70.1
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	**1**	None	None	10	0.01	0.0	0.0	85.7	67.8
Conv2d, ReLu	5	16	2	None	None		GD			85.7	67.8
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	**3**	None	None	10	0.01	**0.9**	0.0	88.5	69.0
Conv2d, ReLu	**4**	16	2	None	None		GD			85.7	71.3
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	**3**	None	None	10	0.01	**0.9**	0.0	95.3	64.4
Conv2d, ReLu	4	16	**1**	None	None		GD			89.8	67.8
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	**7**	**32**	**3**	None	None	10	0.01	**0.9**	0.0	97.3	62.1
Conv2d, ReLu	**4**	16	2	None	None		GD			83.0	66.7
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	None	None	**20**	0.01	0.0	0.0	87.4	67.8
Conv2d, ReLu	5	16	2	None	None		GD			87.4	73.6
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	None	None	**40**	0.01	0.0	0.0	92.6	62.1
Conv2d, ReLu	5	16	2	None	None		GD			90.4	69.0

Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	32	2	**2,2**	None	10	0.01	**0.9**	0.0	74.2	63.2
Conv2d, ReLu	**4**	16	2	**2,2**	None		GD			72.5	65.5
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	**2,1**	None	10	0.01	**0.9**	0.0	92.3	66.7
Conv2d, ReLu	**4**	16	2	**2,1**	None		GD			90.4	69.0
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	**1,1**	None	10	0.01	**0.9**	0.0	92.6	67.8
Conv2d, ReLu	**4**	16	2	**1,1**	None		GD			90.4	69.0
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	**1,1**	None	10	0.01	**0.9**	**0.0005**	91.5	65.5
Conv2d, ReLu	**4**	16	2	**1,1**	None		GD			85.4	66.7
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	None	None	10	0.01	**0.9**	0.0	90.4	63.2
Conv2d, ReLu	5	**32**	2	None	None		GD			93.4	67.8
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	**1,1**	None	10	0.01	**0.9**	0.0	87.9	66.7
Conv2d, ReLu	5	**32**	2	**1,1**	None		GD			87.9	66.7
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	None	None	10	0.01	**0.9**	0.0	87.9	66.7
Conv2d, ReLu	**3**	**32**	2	None	None		GD			87.9	66.7
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	None	None	10	0.01	**0.9**	0.0	84.3	55.2
Conv2d, ReLu	**3**	**32**	**1**	None	None		GD			73.1	60.9
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	None	None	10	0.01	**0.9**	0.0	90.4	63.2
Conv2d, ReLu	**3**	16	**1**	None	None		GD			86.0	65.5
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	2	**1,1**	None	10	0.01	**0.9**	0.0	87.8	67.8
Conv2d, ReLu	**3**	16	**1**	**1,1**	None		GD			87.8	67.8
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	**32**	1	**1,1**	None	10	0.01	**0.9**	0.0	87.1	55.2
Conv2d, ReLu	**3**	16	1	**1,1**	None		GD			83.0	59.8
Fully Connected	64										
Fully Connected	64										
Conv2d, ReLu	5	16	2	None	**Yes**	10	0.01	0.0	0.0	95.1	66.7
Conv2d, ReLu	5	16	2	None			GD			83.6	71.3
Fully Connected	64										
Fully Connected	64										

TABLE II: Various CNNs and their hyperparameters.

Given some idea of the effect of the CNN's many hyperparameters, we move on to increasing the number of convolutional and fully connected layers as the next step in our attempt to maximize validation accuracy. We take inspiration from Krizhevsky et al. [2] as a starting point for our CNN architecture as the classification task is similar (inputs are images). We add a 3rd convolutional layer, as well as a third fully connected layer. We use a pyramidal structure like [2] with increasing depth as we move deeper into the network. We also adopt the momentum gradient method and the local response normalization (LNR) used in [2]. A dropout factor of 0.9 and weight penalty of 0.0005 was also found to be helpful.

The results obtained using the described architecture is reported in Table III for training on the augmented data with 2000 iterations. When compared with our two-layer CNN in Table II, we clearly see the benefit of the third layer as our validation accuracy improves. We note sufficient batch size is important (given the small number of iterations).

The selected CNN is finally tested on the invariance dataset (last row ofTable III). We find poor performance for the inverted data and translated dataset as we have no reason to expect that these sorts of features are captured the training set. Furthermore, the hierarchical structure of the CNN will likely have to be adapted to perform on these types of features.

Layers	Filter Size	Depth	Stride	Pooling	LNR	Batch Size	Learning Rate	Dropout	Weight Penalty	Training accuracy	Validation accuracy
Conv2d, ReLu	5	48	2	(2,2)	True	80	0.01	0.9	0.0005	99.9	74.7
Conv2d, ReLu	5	64	2	None	None		Momemtum			99.9	74.7
Conv2d, ReLu	5	128	2	(2,2)	True						
Fully Connected	2000										
Fully Connected	256										
Fully Connected	256										
Conv2d, ReLu	5	48	2	(2,2)	True	20	0.01	0.9	0.0005	62.4	55.2
Conv2d, ReLu	5	64	2	None	None		Momemtum			62.4	55.2
Conv2d, ReLu	5	128	2	(2,2)	True						
Fully Connected	2000										
Fully Connected	256										
Fully Connected	256										
Conv2d, ReLu	5	48	2	(2,2)	True	80	0.01	0.9	0.0005	99.9	72.4
Conv2d, ReLu	5	64	2	None	None		Momemtum			99.4	75.9
Conv2d, ReLu	5	128	2	(2,2)	True						
Fully Connected	1000										
Fully Connected	128										
Fully Connected	128										
Conv2d, ReLu	5	48	2	(2,2)	True	80	0.01	0.8	0.0005	99.9	67.8
Conv2d, ReLu	5	64	2	None	None		Momemtum			99.9	71.3
Conv2d, ReLu	5	128	2	(2,2)	True						
Fully Connected	1000										
Fully Connected	128										
Fully Connected	128										
Conv2d, ReLu	5	48	2	(2,2)	True	160	0.01	0.9	0.0005	99.9	75.9
Conv2d, ReLu	5	64	2	None	None		Momemtum			99.9	78.2
Conv2d, ReLu	5	128	2	(2,2)	True						
Fully Connected	1000										
Fully Connected	128										
Fully Connected	128										
Conv2d, ReLu	5	48	2	(2,2)	True	240	0.01	0.9	0.0005	100.0	75.9
Conv2d, ReLu	5	64	2	None	None		Momemtum			100.0	75.9
Conv2d, ReLu	5	128	2	(2,2)	True						
Fully Connected	1000										
Fully Connected	128										
Fully Connected	128										
Testing on invariance dataset with the CNN configuration from row above.								normal validation data		75.9	
								translated data		34.5	
								brightened data		65.5	
								darkened data		72.4	
								high contrast data		65.5	
								low contrast data		72.4	
								flipped data		50.6	
								inverted data		12.6	

TABLE III: Deeper CNNs and their hyperparameters.

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Proceedings of the IEEE **86**, 2278 (1998).

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., 2012) pp. 1097–1105.