CS 615 - Deep Learning

Assignment 3 - Convolutional Neural Networks
John Obuch

1 Theory

1. (2pts) Apply kernel K to data X. In other words, what is $X * K$?:

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} K = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Given the the above matrices, we will compute the convolution of$X * K$. Recall the following formulation:

$$F_{ab} = (X_{ab} * K) = \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{M}{2}}^{\frac{M}{2}} X_{(a-i,b-j)} K_{ij}$$

To simplify our computation, we will transform our matrix K by first flipping our matrix horizontally and then vertically prior to performing the convolution computation. Once the transformation is complete, we will perform element-wise matrix multiplication (i.e. take the Hadamard product) and summing the results. The following steps illustrate this process.

1. Flip K horizontally.

$$\tilde{K} = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix}$$

2. Flip $\tilde{K}$ vertically.

$$\hat{K} = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

Now that we have K in the correct form (notated here as $\hat{K}$) we can perform the convolution computation $X * \hat{K}$. It is important to observe that there is only one "valid" convolution in the data provided which will yield in a scalar result as shown as follows:

$$\begin{aligned} X * \hat{K} &= 1(9) + 2(8) + 3(7) + 4(6) + 5(5) + 6(4) + 7(3) + 8(2) + 9(1) \\ &= 9 + 16 + 21 + 24 + 25 + 24 + 21 + 16 + 9 \\ &= 165 \end{aligned}$$

■

2. Given the feature map filter output, $F = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 3 \\ 4 & 5 & 6 & 0 & 0 & 12 \\ 7 & 8 & 9 & 1 & 0 & 4 \\ -100 & -100 & -100 & -100 & -100 & -100 \end{bmatrix}$, what is the output from a pooling layer with width of 2 and stride of 2 if we are using:

   - (3pts) Max-Pooling?

     Given the Filtered response output provided above, we will compute the resulting matrix after computing Max-Pooling. Given that our pooling layer is of width 2 with a stride of 2, performing Max-pooling yields the following resulting Max-pooled matrix which I will notate as M_p.

     $$M_p = \begin{bmatrix} 5 & 6 & 12 \\ 8 & 9 & 4 \end{bmatrix}$$

   - (3pts) Mean-Pooling?

     Leveraging the padded matrix we defined above and performing Mean-Pooling, we achieve the following resulting matrix which I will notate as $\tilde{M}_p$.

     $$\tilde{M}_p = \begin{bmatrix} 3 & 3.25 & 4 \\ -46.25 & -47.5 & -49 \end{bmatrix}$$

   ■

3. (2pts) Given an image X, what would the kernel K be that can reproduce the image when convoluted with it? That is, what is K such that $X * K = X$?

   The only kernel K that can reproduce the original image X is known as the "Do Nothing" kernel. Formally called the identity kernel. It should be noted that the identity kernel K is an $M_o \times M_o$ matrix such that $M \geq 1$ and M is odd (notated here as M_o) where all elements of the matrix are zero except for the center value which is one. However, it should be noted that for our case, the kernel K of interest to us is $K = \begin{bmatrix} 1 \end{bmatrix}$ assuming a "valid" convolution. Although, higher dimensional identity matrices will work if we implement padding prior to performing convolution. What follows is an example 3×3 identity kernel which can be expanded to higher dimensional space given that the dimensions of the kernel satisfy the criteria mentioned above and padding is performed prior to convolving the image X with the kernel K.

   $$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

   ■

2 CNN for LSE Classification

Let's start attempting to differentiate between your two synthetic images using a simple CNN architecture. The CNN architecture is as follows. All hyperparameter choice are up to you:

1. A **single** 40×40 convolutional kernel.
2. A max-pool layer with $width = 1$ and $stride = 1$.
3. A fully-connected layer
4. A linear activation function
5. A squared error objective function.

Hyperparameters:

1. Iteration threshold $= 1000$
2. Learning rate $\alpha = 0.01$
3. Regularization $\lambda = 0.00001$
4. θ was initialized randomly between $(-1, 1)$
5. Kernel K was initialized randomly between $(-0.0001, 0.0001)$

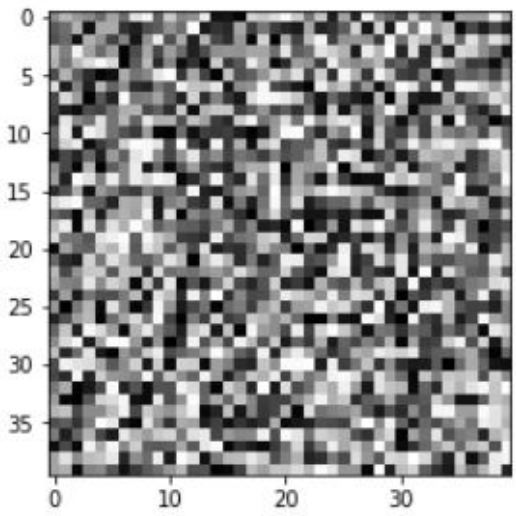


Figure 1: Initial Kernel

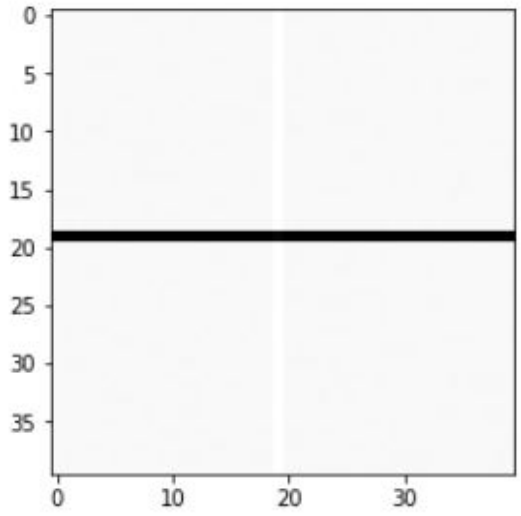


Figure 2: Final Kernel

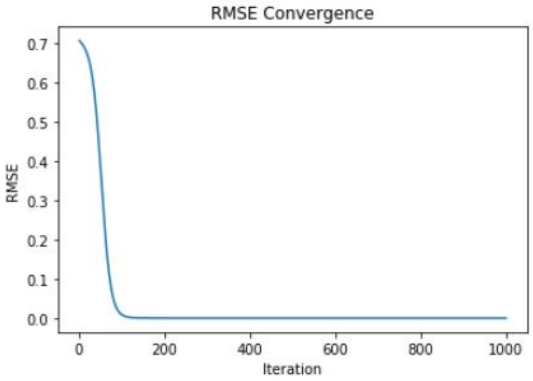


Figure 3: Cost VS Iteration

3 CNN for MLE Classification

Next we'll change our CNN architecture to use a sigmoid activation function and log likelihood objective function. Therefore, the CNN architecture is as follows. Once again, all hyperparameter choice are up to you:

1. A single 40×40 convolutional kernel.
2. A max-pool layer with $width = 1$ and $stride = 1$.
3. A fully-connected layer
4. A sigmoid activation function
5. A log likelihood objective function.

Hyperparameters:

1. Iteration threshold = 1000
2. Learning rate $\alpha = 0.01$
3. Regularization $\lambda = 0.00001$
4. θ was initialized randomly between $(-0.0001, 0.0001)$
5. Kernel K was initialized randomly between $(-0.0001, 0.0001)$

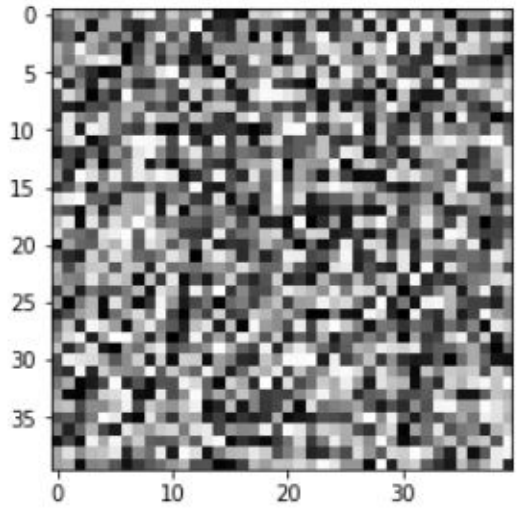


Figure 4: Initial Kernel

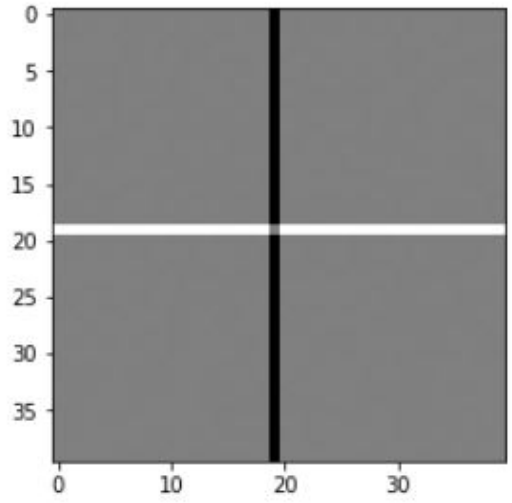


Figure 5: Final Kernel

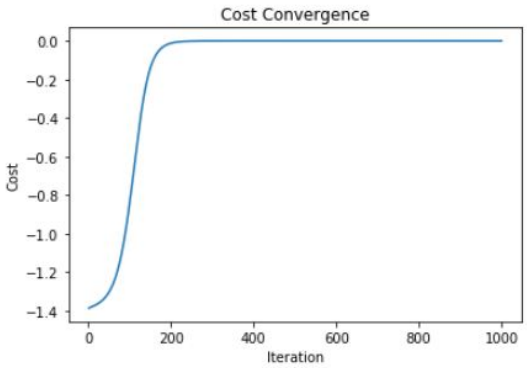


Figure 6: Cost VS Iteration

4 CNN for LCE Classification

One more basic architecture.... Change our CNN architecture to use a softmax activation function and cross-entropy objective function. Therefore, the CNN architecture is as follows. Once again, all hyperparameter choice are up to you:

1. A single 40×40 convolutional kernel.
2. A max-pool layer with $width = 1$ and $stride = 1$.
3. A fully-connected layer
4. A softmax activation function.
5. A cross-entropy objective function.

Hyperparameters:

1. Iteration threshold $= 1000$
2. Learning rate $\alpha = 0.01$
3. Regularization $\lambda = 0.00001$
4. θ was initialized randomly between $(-1, 1)$
5. Kernel K was initialized randomly between $(-0.0001, 0.0001)$

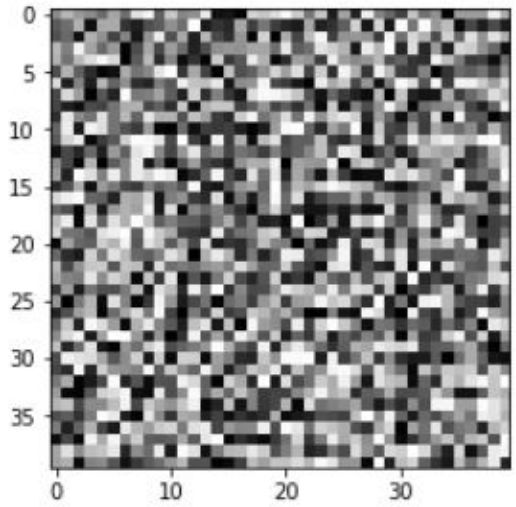


Figure 7: Initial Kernel

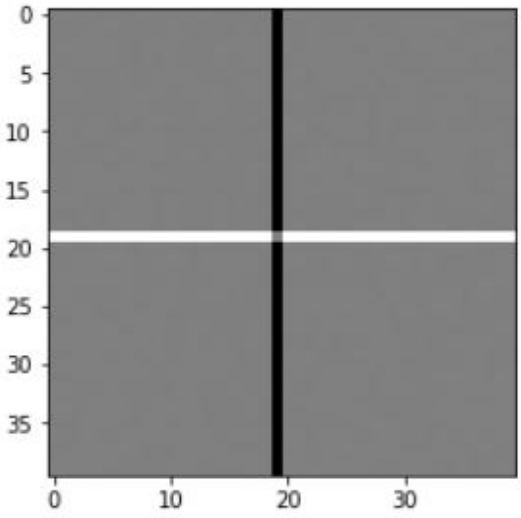


Figure 8: Final Kernel

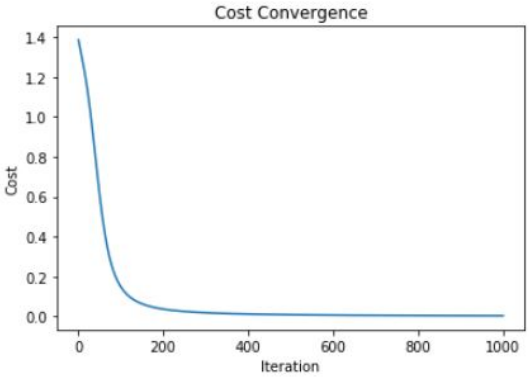


Figure 9: Cost VS Iteration

5 CNN With Multiple Kernels

Let's play around with the convolutional and pooling layers now. Here's the architecture:

1. Four 5×5 convolutional kernels.
2. A max-pool layer with $width = 2$ and $stride = 2$.
3. A fully-connected layer
4. Your choice of activation and objective function!

Hyperparameters:

1. Iteration threshold = 1000
2. Learning rate $\alpha = 0.01$
3. Regularization $\lambda = 0.00001$
4. θ was initialized randomly between $(-1, 1)$
5. Kernel K was initialized randomly between $(-0.0001, 0.0001)$

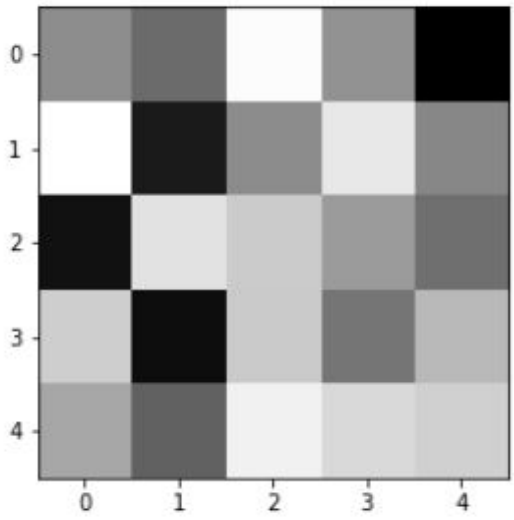


Figure 10: Initial Kernel 1

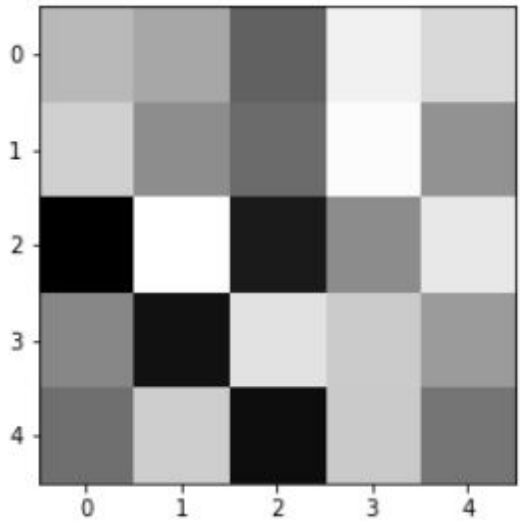


Figure 11: Initial Kernel 2

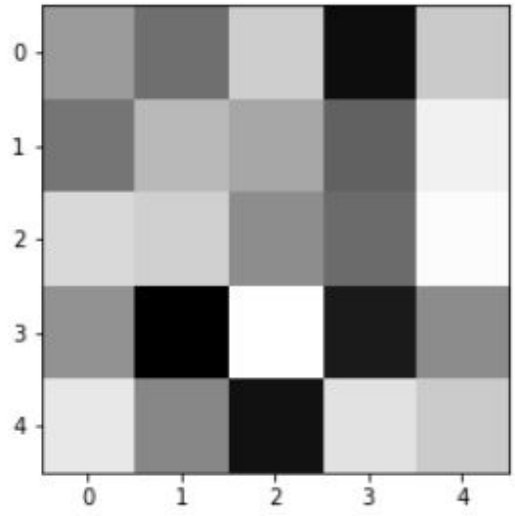


Figure 12: Initial Kernel 3

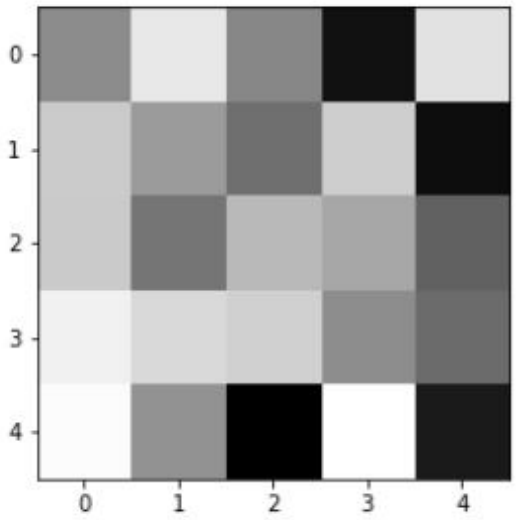


Figure 13: Initial Kernel 4

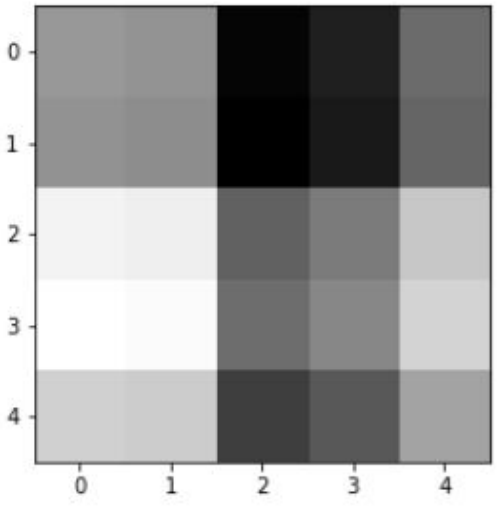


Figure 14: Final Kernel 1

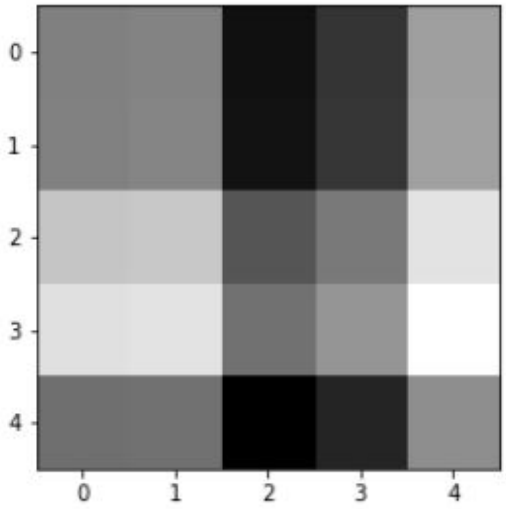


Figure 15: Final Kernel 2

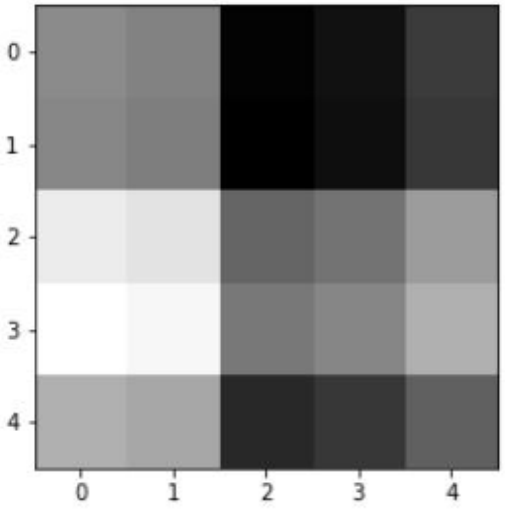


Figure 16: Final Kernel 3

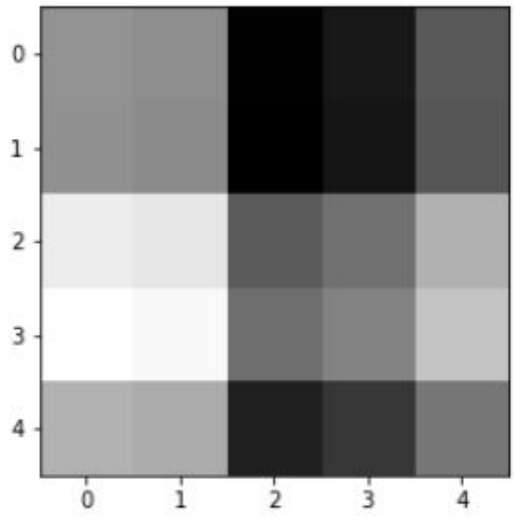


Figure 17: Final Kernel 4

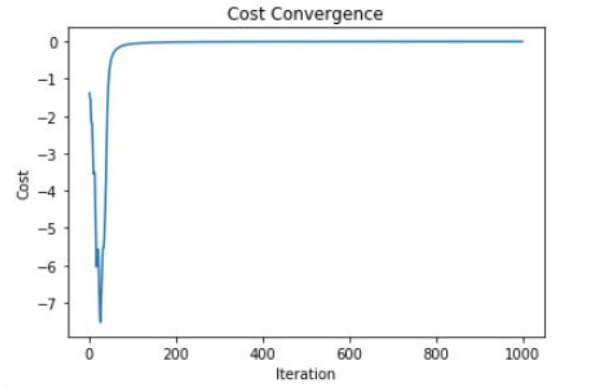


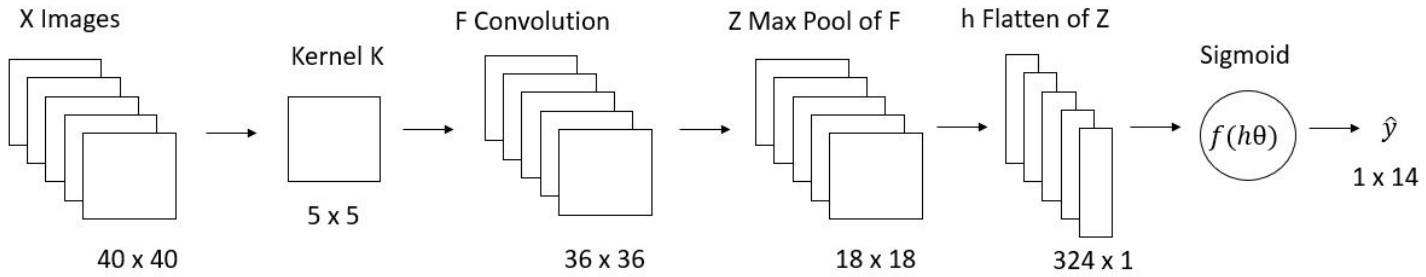
Figure 18: Cost VS Iteration

6 Multi-Kernel CNN For Image Classification

Finally let's open this up to our Yale Faces problem.

At this point *all design decisions are up to you*. The only constraints are that you must have at least one convolutional layer and that you try at least two different architectures.

Architecture Configuration 1:



Note: Theta was initialized randomly with dimensions of (324 x 14). Data was trained for each image iteratively and then compiled after training. For the pooling layer, a window of 2 and stride of 2 were used. The objective function used was log likelihood.

Figure 19: Model Architecture

Hyperparameters:

1. Iteration threshold = 1000
2. Learning rate $\alpha = 0.02$
3. Regularization $\lambda = 0.00001$
4. θ was initialized randomly between $(-0.1, 0.1)$
5. Kernel K was initialized randomly between $(-0.0001, 0.0001)$

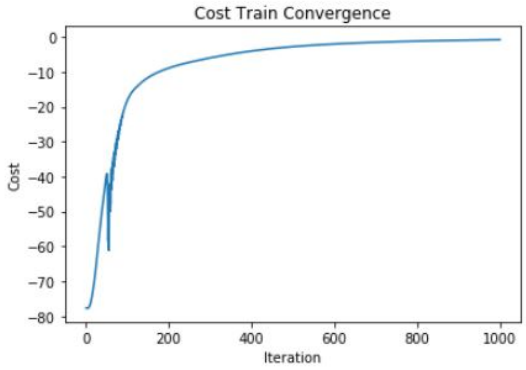


Figure 20: Train Cost VS Iteration

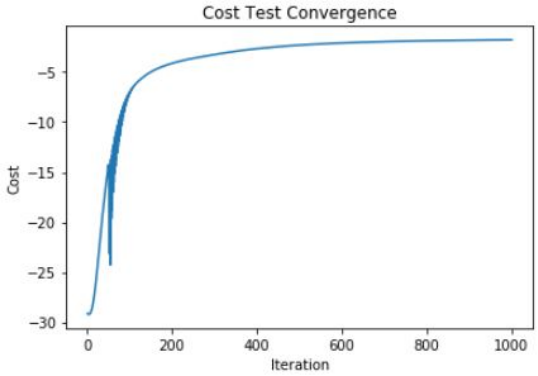


Figure 21: Test Cost VS Iteration

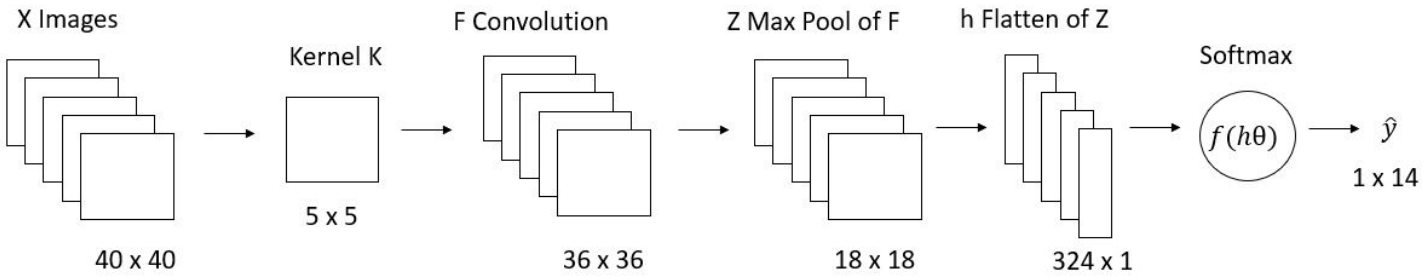
Tux Accuracy Metrics:
Train Accuracy = 100%

Test Accuracy = 85.71%

Architecture Configuration 2:

Hyperparameters:

1. Iteration threshold = 1000
2. Learning rate $\alpha = 0.02$
3. Regularization $\lambda = 0.00001$
4. θ was initialized randomly between $(-0.1, 0.1)$
5. Kernel K was initialized randomly between $(-0.0001, 0.0001)$



Note: Theta was initialized randomly with dimensions of (324 x 14). Data was trained for each image iteratively and then compiled after training. For the pooling layer, a window of 2 and stride of 2 were used. The objective function used was cross entropy.

Figure 22: Model Architecture

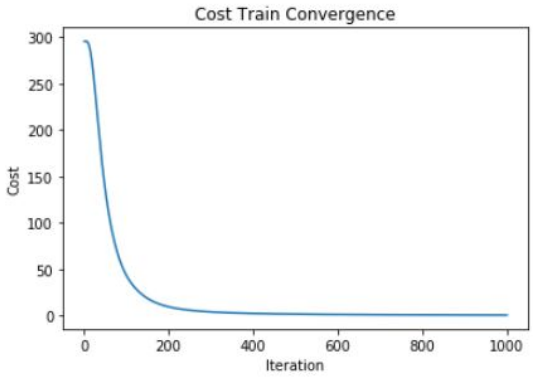


Figure 23: Train Cost VS Iteration

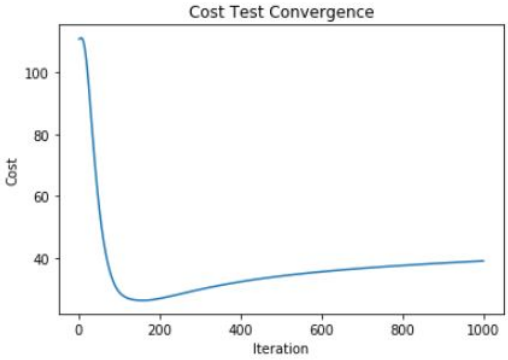


Figure 24: Test Cost VS Iteration

Tux Accuracy Metrics:
Train Accuracy = 100%

Test Accuracy = 83.33%