Assignment 2

Juliana De La Vega Fernandez - MScAC

November 14, 2016

1 EM FOR MIXTURE OF GAUSSIANS

1.1 DERIVE THE EM EQUATIONS FOR MAXIMIZING THE LIKELIHOOD FUNCTION UNDER THE MODEL WITH THE SHARED COVARIANCE MATRIX

Step 1. **Calculate the E-step**, using the conditional probability of z given x:
$\gamma_k = p(z=k|x) = \frac{p(z=k)p(x|z=k)}{p(x)}$

Where $p(x)$ can be replaced as:
$\gamma_k = p(z=k|x) = \frac{p(z=k)p(x|z=k)}{\sum_{j=1}^{K} p(z=j)p(x|z=j)}$

And using:
$p(x|z) = N(x|\mu_k, \Sigma)$

and:
$p(z=k) = \pi_k$

The responsibilities can be described as the following equation after substituting the las terms:

$$\boxed{\gamma_k = p(z=k|x) = \frac{\pi_k N(x|\mu_k, \Sigma)}{\sum_{j=1}^{K} \pi_j N(x|\mu_j, \Sigma)}}$$

Step 2. **Calculate the M-step** using the gaussian distribution:
$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu_k)^T \Sigma^{-1}(x-\mu_k)\right\}$

The log-likelihood function will be described by:
$l(z) = \sum_{n=1}^{N} \log p(x^{(n)}|z) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k N(x^n|\mu_k, \Sigma)$

Deriving with respect to μ:

$$\frac{\partial l}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot |\Sigma|^{-1/2} \cdot \frac{\partial}{\partial \mu_k} exp\left\{-\frac{1}{2}\left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\}}{\sum_{j=1}^{K} \pi_j N(x^n|\mu_j,\Sigma)}$$

Using the rule $\frac{\partial X^T AX}{\partial X} = \left(A + A^T\right)X$:

$$\frac{\partial l}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot |\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2}\left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\} \cdot \left(-\frac{1}{2}\right) \cdot \left(\Sigma^{-1}\left(\Sigma^{-1}\right)^T\right) \cdot \left(x^{(n)}-\mu_k\right) \cdot (-1)}{\sum_{j=1}^{K} \pi_j N(x^n|\mu_j,\Sigma)}$$

$$\frac{\partial l}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot |\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2}\left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\} \cdot \left(\frac{1}{2}\right) \cdot (2\Sigma^{-1}) \cdot \left(x^{(n)}-\mu_k\right)}{\sum_{j=1}^{K} \pi_j N(x^{(n)}|\mu_j,\Sigma)}$$

$$\frac{\partial l}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot |\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2}\left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\} \cdot \left(\Sigma^{-1}\right) \cdot \left(x^{(n)}-\mu_k\right)}{\sum_{j=1}^{K} \pi_j N(x^{(n)}|\mu_j,\Sigma)}$$

Regrouping the gaussian distribution equation in the numerator, and reordering Σ outside of the summation as it does not depend on N: $\frac{\partial l}{\partial \mu_k} = \left(\Sigma^{-1}\right) \cdot \sum_{n=1}^{N} \frac{\pi_k \cdot N(x^{(n)}|\mu_k,\Sigma)}{\sum_{j=1}^{K} \pi_j N(x^{(n)}|\mu_j,\Sigma)} \left(x^{(n)} - \mu_k\right)$

Using the responsibilities from the E-step as calculated before: $\gamma\left(z_k^{(n)}\right) = \frac{\pi_k \cdot N\left(x^{(n)}|\mu_k,\Sigma\right)}{\sum_{j=1}^{K} \pi_j N(x^{(n)}|\mu_j,\Sigma)}$

And substituting into the equation:

$$\frac{\partial l}{\partial \mu_k} = \left(\Sigma^{-1}\right) \cdot \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(x^{(n)} - \mu_k\right)$$

Setting $\frac{\partial l}{\partial \mu_k} = 0$, and solving for μ

$$0 = \left(\Sigma^{-1}\right) \cdot \left(\sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(x^{(n)}\right) - \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(\mu_k\right)\right)$$

The Σ term is removed from the equation by dividing by it on both sides, and the subtraction is expanded:

$$0 = \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(x^{(n)}\right) - \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(\mu_k\right)$$

μ is taken out of the summation as it does not depend on N:

$$\mu_k \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right) = \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(x^{(n)}\right)$$

And dividing by the summation, μ is solved:

$$\boxed{\mu_k = \frac{\sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)\left(x^{(n)}\right)}{\sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)}}$$

Calculating the **derivative of the log-likelihood with respect to** Σ:

$$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot \frac{\partial}{\partial \Sigma}\left(|\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2} \cdot \left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\}\right)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)}$$

Expanding the derivative:

$$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot \left(\frac{\partial}{\partial \Sigma}\left(|\Sigma|^{-1/2}\right) \cdot exp\left\{-\frac{1}{2} \cdot \left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\} + |\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2} \cdot \left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right\} \cdot \left(-\frac{1}{2}\right) \cdot \frac{\partial}{\partial \Sigma}\left(\left(x^{(n)}-\mu_k\right)^T \Sigma^{-1}\left(x^{(n)}-\mu_k\right)\right)\right)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)}$$

Using the following tis rule for the derivation of the determinant:

$\frac{\partial |X|}{\partial X} = |X| \cdot \left(X^{-1}\right)^T$

And this rule for the derivation of the term inside the exponential function:
$\frac{\partial a^T X^{-1} b}{\partial X} = \quad -X^{-1} a b^T X^{-T}$

The derivative would be:
$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot \left(\left(-\frac{1}{2}\right) \cdot |\Sigma|^{-1/2} \cdot |\Sigma|^{-1} \cdot |\Sigma| \cdot \left(\Sigma^{-1}\right)^T \cdot exp\left\{-\frac{1}{2} \cdot \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1} \left(x^{(n)} - \mu_k\right)\right\} + |\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2} \cdot \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1} \left(x^{(n)} - \mu_k\right)\right\} \cdot \left(-\frac{1}{2}\right) \cdot \left(-\Sigma^{-1} \left(x^{(n)} - \mu_k\right) \left(x^{(n)} \right.\right.}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)}$

Taking the common terms out, the equation will be:
$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot \frac{1}{(2\pi)^{D/2}} \cdot |\Sigma|^{-1/2} \cdot exp\left\{-\frac{1}{2} \cdot \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1} \left(x^{(n)} - \mu_k\right)\right\} \cdot \left(-\frac{1}{2}\right) \cdot \left(|\Sigma|^{-1} \cdot |\Sigma| \cdot \left(\Sigma^{-1}\right)^T + \left(-\Sigma^{-1} \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1}\right)\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)}$

The common terms can now be grouped as the gaussian distribution:
$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right) \cdot \left(-\frac{1}{2}\right) \cdot \left(\Sigma^{-1} + \left(-\Sigma^{-1} \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1}\right)\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)}$

Multiplying by Σ on the left side, and using $\Sigma\Sigma^{-1} = I$, the equation is updated to:

$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right) \cdot \left(-\frac{1}{2}\right) \cdot \left(\Sigma\Sigma^{-1} - \Sigma\Sigma^{-1} \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1}\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)}$

Applying $\Sigma^{-1}\Sigma = I$ on the right side, the equation would now be:

$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right) \cdot \left(-\frac{1}{2}\right) \cdot \left(I\Sigma - I \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T \Sigma^{-1}\Sigma\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)}$

$\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right) \cdot \left(-\frac{1}{2}\right) \cdot \left(\Sigma - \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)}$

$\frac{\partial l}{\partial \Sigma} = \left(-\frac{1}{2}\right) \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)} \cdot \left(\Sigma - \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T\right)$

Expanding the summation for both terms in the subtraction:

$\frac{\partial l}{\partial \Sigma} = \left(-\frac{1}{2}\right) \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)} \cdot \Sigma - \frac{\sum_{k=1}^{K} \pi_k \cdot N\left(x^{(n)} | \mu_k, \Sigma\right)}{\sum_{j=1}^{K} \pi_j \cdot N\left(x^{(n)} | \mu_j, \Sigma\right)} \cdot \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T$
Âă
Setting the derivative to zero, $\frac{\partial l}{\partial \Sigma} = 0$, and solving for Σ:

$0 = \sum_{n=1}^{N} \Sigma \quad - \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma\left(z_k^{(n)}\right) \cdot \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T$

$N \cdot \Sigma = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma\left(z_k^{(n)}\right) \cdot \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T$

$$\boxed{\Sigma = \frac{\sum_{n=1}^{N} \sum_{k=1}^{K} \gamma\left(z_k^{(n)}\right) \cdot \left(x^{(n)} - \mu_k\right) \left(x^{(n)} - \mu_k\right)^T}{N}}$$

Calculating the **derivative of the log-likelihood with respect to** π_k. A Lagrange multiplier is needed in order to proceed, so the log likelihood would now be:
$\log\left(p\left(X | \pi_k, \mu_k, \Sigma\right)\right) + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right)$

Deriving with respect to π_k
$\frac{\partial l}{\partial \pi_k} = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} N(x^{(n)}|\mu_k,\Sigma)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)} + \lambda$

We multiply by π_k to obtain:

$\frac{\partial l}{\partial \pi_k} = \sum_{n=1}^{N} \frac{N(x^{(n)}|\mu_k,\Sigma)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)} \cdot \pi_k + \lambda \pi_k$

We would have K of these terms. Using the following rule:
$\sum_{k=1}^{K} \pi_k = 1$

And setting the derivative to zero, $\frac{\partial l}{\partial \pi_k} = 0$, we obtain:

$0 = \sum_{n=1}^{N} \frac{\sum_{k=1}^{K} \pi_k \cdot N(x^{(n)}|\mu_k,\Sigma)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)} + \lambda \sum_{k=1}^{K} \pi_k$

Simplifying:
$0 = \sum_{n=1}^{N} 1 + \lambda$

$0 = N + \lambda$

$\lambda = -N$

With λ we can now calculate π_k, remembering this equation:
$\frac{\partial l}{\partial \pi_k} = \sum_{n=1}^{N} \frac{N(x^{(n)}|\mu_k,\Sigma)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)} \cdot \pi_k + \lambda \pi_k$

and dividing by π_k:

$0 = \sum_{n=1}^{N} \frac{N(x^{(n)}|\mu_k,\Sigma)}{\sum_{j=1}^{K} \pi_j \cdot N(x^{(n)}|\mu_j,\Sigma)} \cdot \frac{\pi_k}{\pi_k} + \lambda$

The term on the left corresponds to the responsibilities, replacing:
$0 = \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right) \cdot \frac{1}{\pi_k} - N$

And solving for π_k:

$N = \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right) \cdot \frac{1}{\pi_k}$

$N\pi_k = \sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)$

$$\boxed{\pi_k = \frac{\sum_{n=1}^{N} \gamma\left(z_k^{(n)}\right)}{N}}$$

2 Convolutional Neural Networks

2.1 Show the update equations for the filters and activities given some loss function E

Using the convolution operator definition:
$y_{n,h',w',k} = x * f = \sum_{i=1}^{I} \sum_{j=1}^{H} \sum_{c=1}^{C} x_{n,h'+i-1,w'+j-1,c} \cdot f_{i,j,c,k}$

The filter transpose definition:
$f^T_{i,j,k,c} = f_{I-i+1,J-j+1,c,k}$

The forward propagation equation:
$y = x^{(I-1,\quad J-1)} * f$

Assuming that $n = 1$, $c = 1$ and $k = 1$, the convolution operator can be simplified as:
$y_{h',w'} = x * f = \sum_{i=1}^{I} \sum_{j=1}^{H} x_{h'+i-1,w'+j-1} \cdot f_{i,j}$

Calculating the **derivative with respect to the filter**:
$\frac{\partial E}{\partial f_{i,j}} = \sum_{h'}^{H} \sum_{w'}^{W} \frac{\partial E}{\partial y_{h',w'}} \frac{\partial y_{h',w'}}{\partial f_{i,j}}$

First we focus on the partial derivative of the forward propagation with respect to the filter:

$\frac{\partial y_{h',w'}}{\partial f_{i,j}} = \frac{\partial}{\partial f_{i,j}} \left(\sum_{i=1}^{I} \sum_{j=1}^{J} x_{h'+i-1,w'+j-1} \cdot f_{i,j} \right)$

Expanding the summation:
$\frac{\partial y_{h',w'}}{\partial f_{i,j}} = \frac{\partial}{\partial f_{i,j}} \left(x_{h',w'} \cdot f_{1,1} + ... + x_{h'+I-1,w'+J-1} \cdot f_{I,J} \right)$

Taking partial derivatives for all the components of the summation results in zero values for all the components, except for those where $i = I$ and $j = J$ in f:
$\frac{\partial y_{h',w'}}{\partial f_{i,j}} = \frac{\partial}{\partial f_{i,j}} \left(x_{h'+I-1,w'+J-1} \cdot f_{I,J} \right)$

$\frac{\partial y_{h',w'}}{\partial f_{i,j}} = x_{h'+I-1,w'+J-1}$

Replacing this partial derivative into the derivative of the loss function with respect to the filters:

$\frac{\partial E}{\partial f_{i,j}} = \sum_{h'}^{H} \sum_{w'}^{W} \frac{\partial E}{\partial y_{h',w'}} x_{h'+I-1,w'+J-1}$

Changing from the correlation to the convolution:
$\frac{\partial E}{\partial f_{i,j}} = \frac{\partial E}{\partial y_{h',w'}} * x_{h,w}^{(I-1,\quad J-1)}$

Using the commutative property:

$$\boxed{\frac{\partial E}{\partial f_{i,j}} = x_{h,w}^{(I-1,\quad J-1)} * \frac{\partial E}{\partial y_{h',w'}}}$$

Calculating the **derivative of the loss function with respect to the activities**:

$\frac{\partial E}{\partial x_{h,w}} = \sum_{h'}^{H} \sum_{w'}^{W} \frac{\partial E}{\partial y_{h',w'}} \frac{\partial y_{h',w'}}{\partial x_{h,w}}$

First we calculate the partial derivative of the forward propagation equation with respect to the activities:
$\frac{\partial y_{h',w'}}{\partial x_{h,w}} = \frac{\partial}{\partial x_{h,w}} \left(\sum_{i=1}^{I} \sum_{j=1}^{J} x_{h'+i-1,w'+j-1} \cdot f_{i,j} \right)$

Expanding the summation:
$\frac{\partial y_{h',w'}}{\partial x_{h,w}} = \frac{\partial}{\partial x_{h,w}} \left(x_{h',w'} \cdot f_{1,1} + ... + x_{h'+I-1,w'+J-1} \cdot f_{I,J} \right)$

The same follows for the partial derivative of each term in the summation:
$\frac{\partial y_{h',w'}}{\partial x_{h,w}} = \frac{\partial}{\partial x_{h,w}} \left(x_{h'+I-1,w'+J-1} \cdot f_{I,J} \right)$

Using the following relationships, $h' + i - 1 \quad = \quad h$, and $w' + j - 1 \quad = \quad w$, we obtain:

$\frac{\partial y_{h',w'}}{\partial x_{h,w}} = f_{I,J} \quad = \quad f_{h-h'+1,w-w'+1}$

Substituting the result of the partial derivative in the derivative of the loss function with respect to the activities:
$\frac{\partial E}{\partial x_{h,w}} = \sum_{h'}^{H} \sum_{w'}^{W} \frac{\partial E}{\partial y_{h',w'}} f_{h-h'+1,w-w'+1}$

Re-structuring in order to get the filters in the form expected by the convolution:
$f_{h-h'+1,w-w'+1} \quad = \quad rot_{180} \left(f_{h-h'+1,w-w'+1} \right) \quad = \quad f_{h+h'-1,w+w'-1}^{T}$

Replacing in the derivation:
$\frac{\partial E}{\partial x_{h,w}} = \frac{\partial E}{\partial y_{h,w}} * rot_{180} \left(f_{h-h'+1,w-w'+1} \right)$

$$\boxed{\frac{\partial E}{\partial x_{h,w}} = \frac{\partial E}{\partial y_{h,w}} * f_{h+h'-1,w+w'-1}^{T}}$$

3 NEURAL NETWORKS

3.1 BASIC GENERALIZATION

A default NN and CNN were trained using the pre-established set of hyperparameters. For the NN these corresponded to a `num_hiddens = [16, 32]`, `eps = 0.01`, `momentum = 0.0`, `num_epochs = 1000`, and `batch_size = 100`. For the CNN these corresponded to `eps = 0.1`, `momentum = 0.0`, `num_epochs = 30`, `filter_size = 5`, `num_filters_1 = 8`, `num_filters_2 = 16`, and `batch_size = 100`.

After training over the Toronto Faces dataset, the NN displays the following results:

```
CE: Train 0.49304 Validation 0.88757 Test 0.96508
Acc:  Train 0.82691 Validation 0.75179 Test 0.72987
```

These results reflect the expected decrease from training to validation and test set. The accuracy in the training set is higher because this was the class used in order to "teach" the neurons how to distinguish faces. What they "learned" from the training set they apply on the validation set, obtaining a 75.179% accuracy, which is lower than the one portrayed in the training set. These are faces the model had not seen before. For the test set, accuracy is similar to that of the validation set because it has the same characteristics. The test set contains faces that the model had not seen. It's important to highlight that the validation set and the test set had similar performance. This indicates that the distribution of faces is similar in both sets, and across the whole data set.

Figure 3.1.a. displays the accuracy achieved by the NN, how ever it is possible to notice that the curve could still go higher if the number of epochs were increased. In figure 3.1.b we may see the cross entropy cost for the NN, as is seen in the results, its final value es 49.304%. We know that if a neurons output is close to the desired output, the cross-entropy will be close to 0, This means that there is a lot more training to do for this model in order to get neurons to perform better during validation and test time.

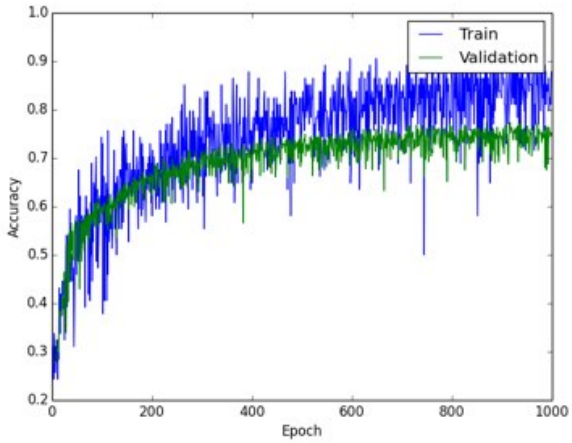
During the validation, which is represented by the green curve in both of these graphs, we are able to see that the model is behaving very similarly to the training set. The shape it is taking mirrors the one of the training set, at a lower level, which is expected as was explained above.

The results for the CNN after training over the Toronto Faces dataset using the default parameters is:

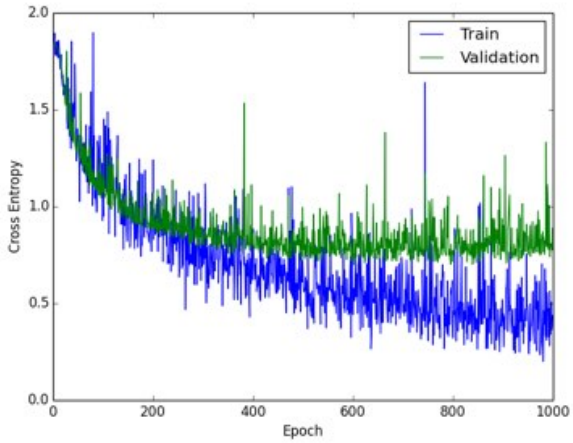
```
CE: Train 0.79223 Validation 0.92690 Test 0.91298
Acc:  Train 0.71458 Validation 0.66826 Test 0.65714
```

As in NN, the CNN achieves higher accuracy in the training set than in the validation and test set. However, the results for validation and test set are much more alike with themselves and closer to that achieved by the training set. It is possible to see in the cross-entropy that the model has still much more training to do, for it is not close to 0, but it has some good results.

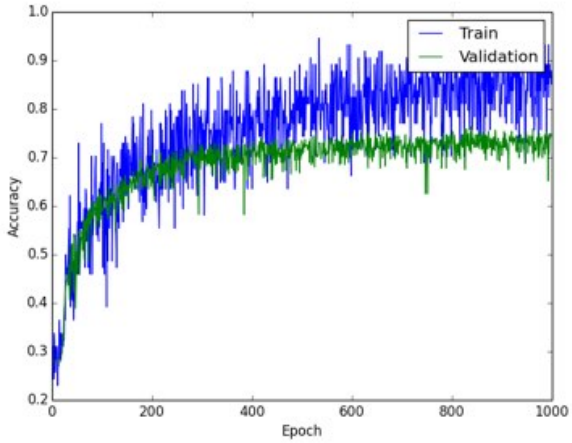
Figure 3.1.c shows the accuracy for the CNN, we can se that with respect to figure 3.1.a the validation and the training curves seem more distant in CNN. Despite that, the validation and training curves for CNN are much closer together than for NN's. It is the dispersion of data that has reduced in CNN's, making the breach between the training and the validation seem larger.For the NN the difference between the training set accuracy and the validation set accuracy was around 7.5%, while for the CNN it was around 4.6%.



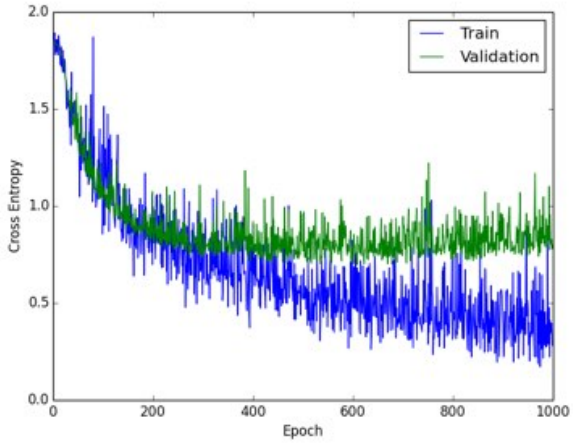
(a) Accuracy for the Default NN



(b) Cross Entropy for the Default NN



(c) Accuracy for the Default CNN



(d) Cross Entropy for the Default CNN

Figure 3.1: Accuracy and error curves for default hyperparameters for NN and CNN.

3.2 Optimization

3.2.1 Modification of the learning rate ϵ hyperparameter

The modification over the ϵ parameter for NN is viewed in figure 3.2 and 3.3, and in figure 3.4 and 3.5 for the CNN. For the NN, the highest accuracy in the training set is reached using $\epsilon = 0.05$, reporting a 85.95%. The highest accuracy in the testing set is reached using the same ϵ, with 74.026%.

The lowest accuracy in all sets is reached using the $\epsilon = 0.001$, which is too low in order for the model to learn. Although the $\epsilon = 0.01$ has a low accuracy, it is higher than that achieved with the lowest epsilon. The model seems to converge faster with an $\epsilon = 0.05$ or $\epsilon = 0.01$, if the learning rate is higher or lower than this, the accuracy of the model decreases.

For the CNN, using an $\epsilon = 0.05$ reaches the highest accuracy over all the sets. This learning rate is followed by $\epsilon = 0.1$ which is close to the last learning rate. Rates lower than this demonstrate a lower accuracy over all the sets. Still, more epochs are needed in order to determine with certainty because the cross entropy is still high for all models trained.

3.2.2 Modification of the momentum hyperparameter

The modification of the momentum hyperparameter can be viewed in figure 3.6 for the NN, and figure 3.7 for the CNN. Three momentum values were tested: 0.1, 0.5, 0.9. A high convergence for the NN makes the model reach a lower accuracy. Although a momentum of 0.1 reaches a good accuracy, the best accuracy for all sets is reached using a momentum of 0.5.

For the CNN the convergence changes a bit. The best model is trained using a momentum of 0.1. As the momentum increases, the accuracy decreases. In this case a lower momentum reaches a higher accuracy, it makes the model converge faster.

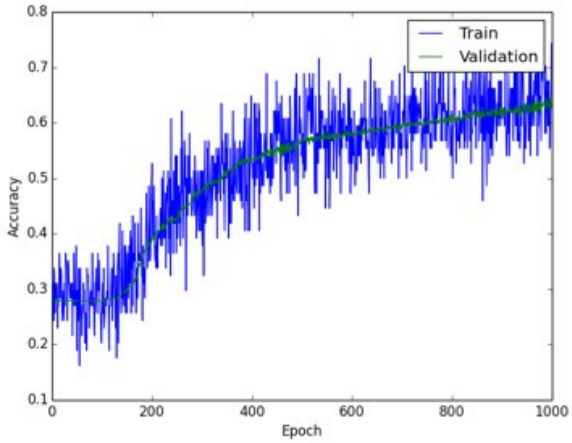
3.2.3 Modification of the mini-batch hyperparameter

The modification of the mini-batch hyperparameter can be seen in figures 3.8 and 3.9 for NN, and figures 3.10 and 3.11 for CNN. For the NN, a model using a batch of 1 was trained, which trained for over 1 hour. Using a batch size of 100 makes the convergence be faster. As the batch size increases, the convergence is slower, and the accuracy of the model decreases. The highest accuracy is achieved using a batch of 100.

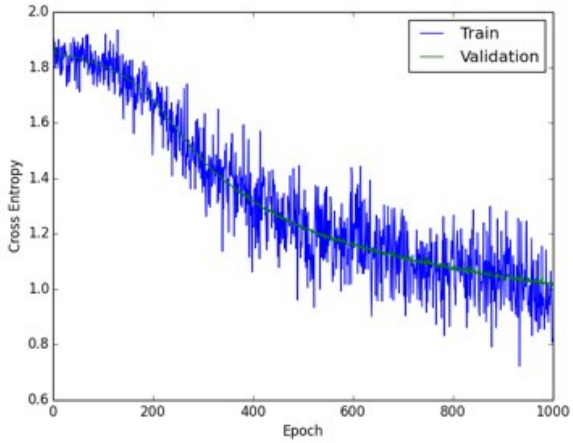
For the CNN similar results are observed. The fastest convergence is reached using a batch of 100, where the accuracy over all the sets is highest. As the batch size is increased, the convergence rate decreases, achieving lower accuracy in these models.

3.2.4 Choosing the best value of the parameters

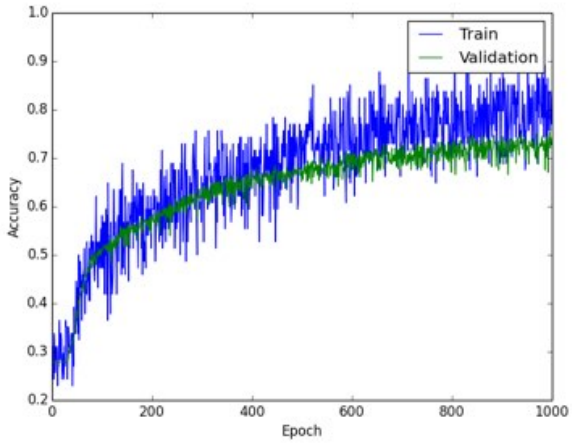
In order to choose the best model for the parameters, new models need to be trained. Probably the best choice would be to update one parameter to its best, for example, the learning rate, and afterwards perform new tests holding constant all parameters except for momentum. After that, the best momentum is selected and the model is updated, all parameters will be held constant except for the mini-batch size. Here, tests using multiple batch sizes will be performed, and the best one will be selected. If the accuracy of the model is higher than all the other trained before, this model may be selected as the best model.



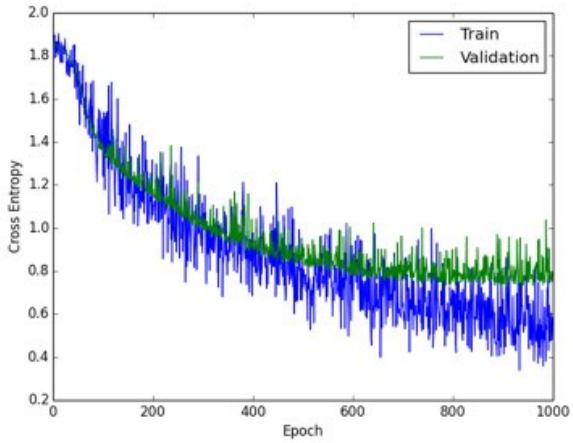
(a) Accuracy for NN using learning rate of $\epsilon = 0.001$



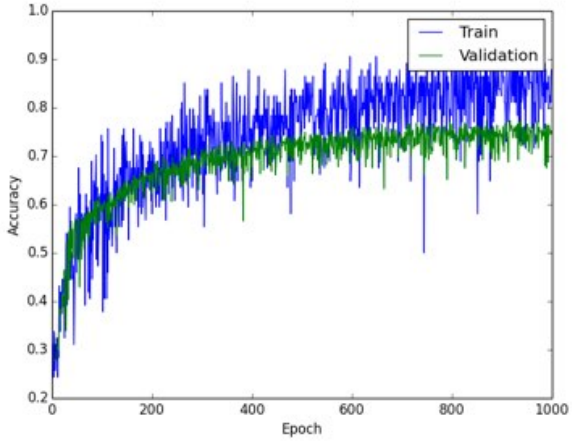
(b) Cross Entropy of NN using learning rate of $\epsilon = 0.001$



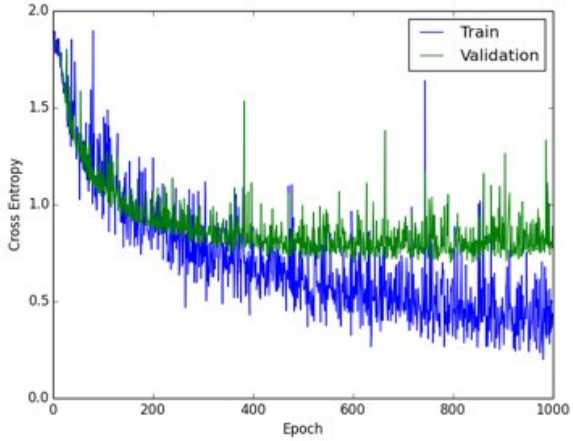
(c) Accuracy for NN using learning rate of $\epsilon = 0.005$



(d) Cross entropy of NN using learning rate of $\epsilon = 0.005$

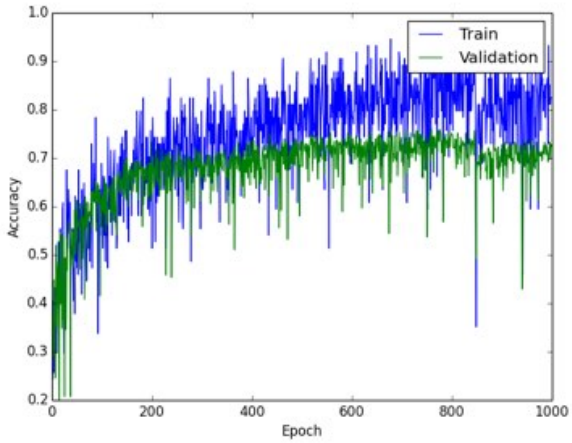


(e) Accuracy for NN using learning rate of $\epsilon = 0.01$

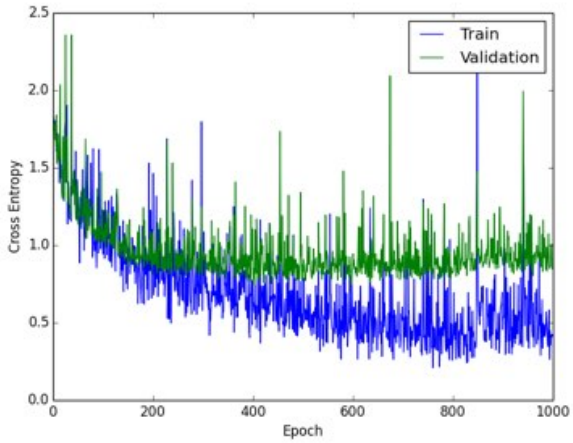


(f) Cross entropy of NN using learning rate of $\epsilon = 0.01$

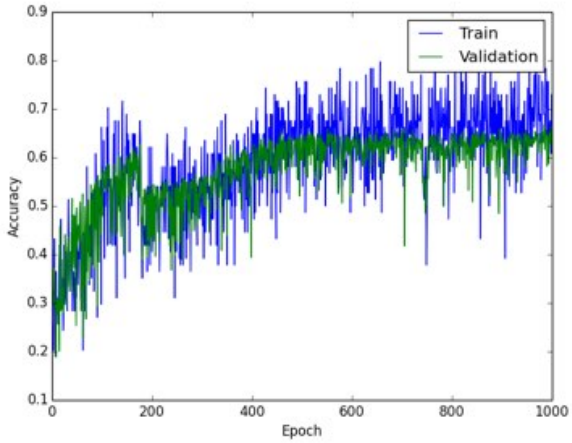
Figure 3.2: Accuracy and error curves for NN trained at $\epsilon = 0.001, 0.005, 0.01$.



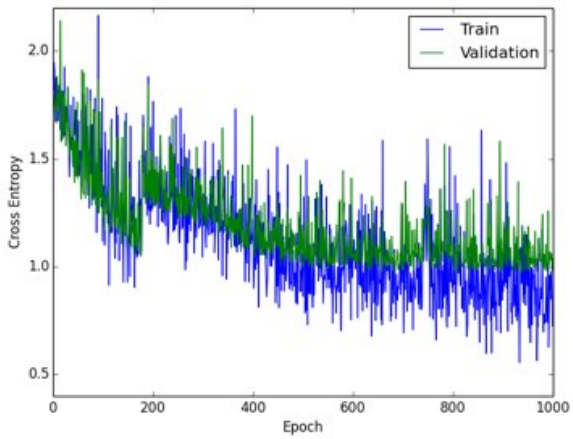
(a) Accuracy for NN using learning rate of $\epsilon = 0.05$



(b) Cross entropy of NN using learning rate of $\epsilon = 0.05$

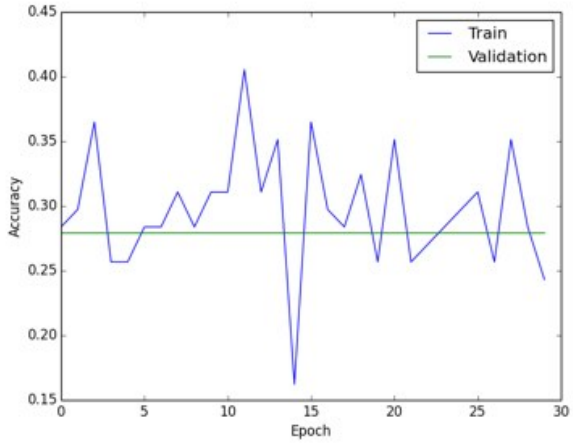


(c) Accuracy for NN using learning rate of $\epsilon = 0.1$

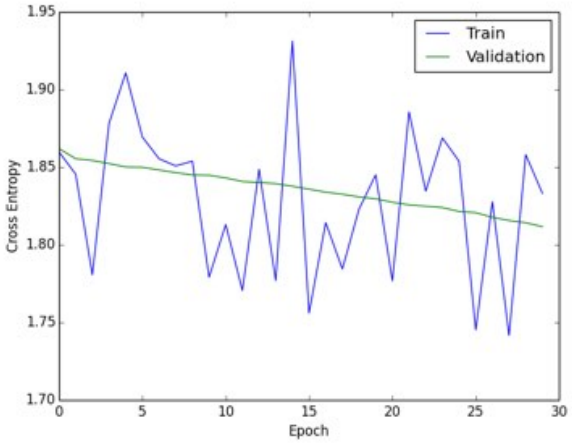


(d) Cross entropy of NN using learning rate of $\epsilon = 0.1$

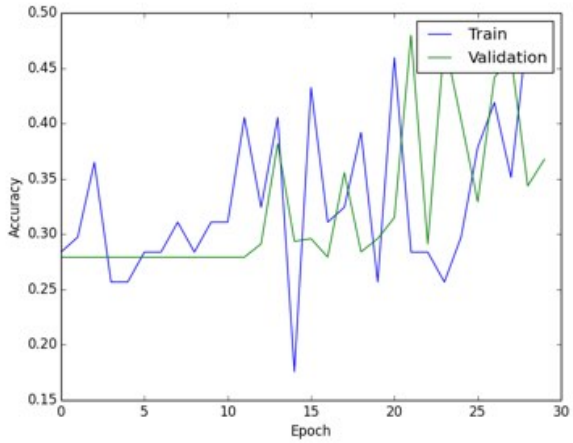
Figure 3.3: Accuracy and error curves for NN trained at $\epsilon = 0.05, 0.1$.



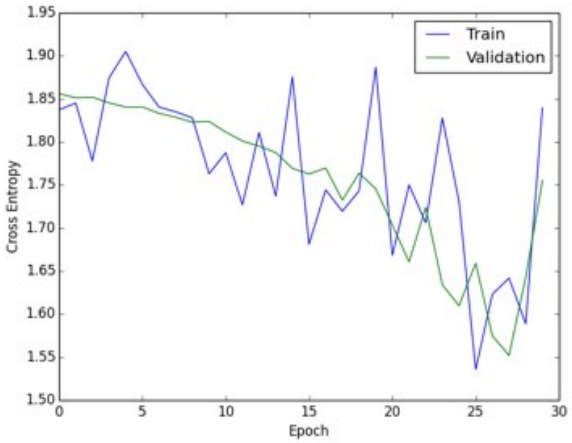
(a) Accuracy for CNN using learning rate of $\epsilon = 0.001$



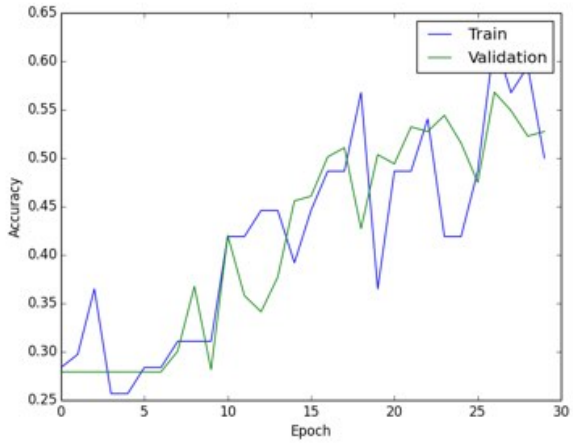
(b) Cross Entropy of CNN using learning rate of $\epsilon = 0.001$



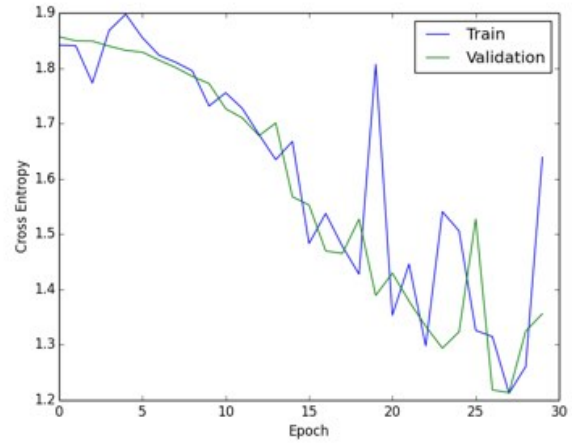
(c) Accuracy for CNN using learning rate of $\epsilon = 0.005$



(d) Cross entropy of CNN using learning rate of $\epsilon = 0.005$

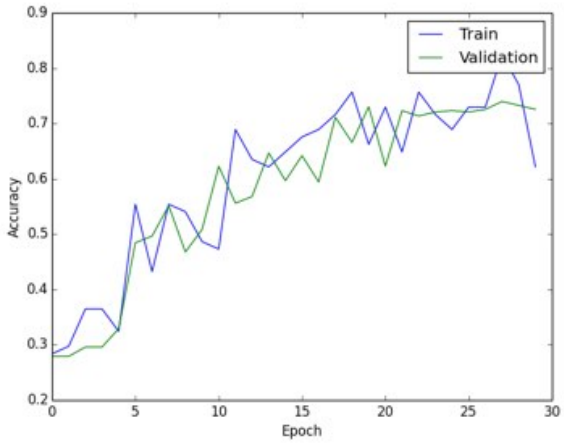


(e) Accuracy for CNN using learning rate of $\epsilon = 0.01$

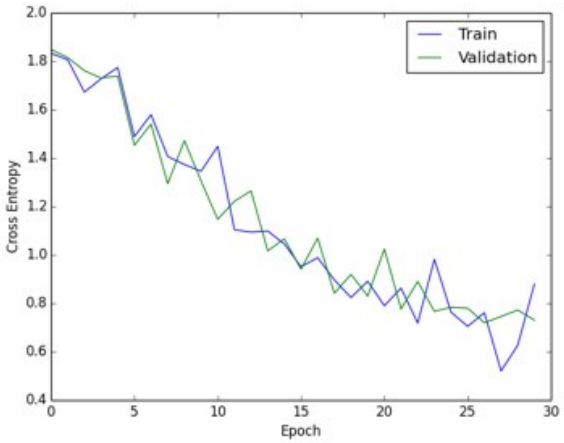


(f) Cross entropy of CNN using learning rate of $\epsilon = 0.01$

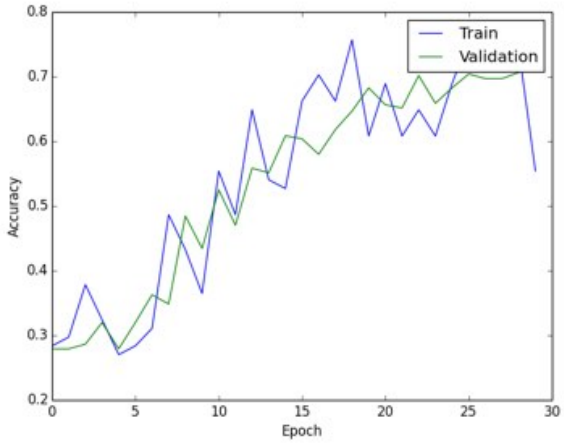
Figure 3.4: Accuracy and error curves for CNN trained at $\epsilon = 0.001, 0.005, 0.01$.



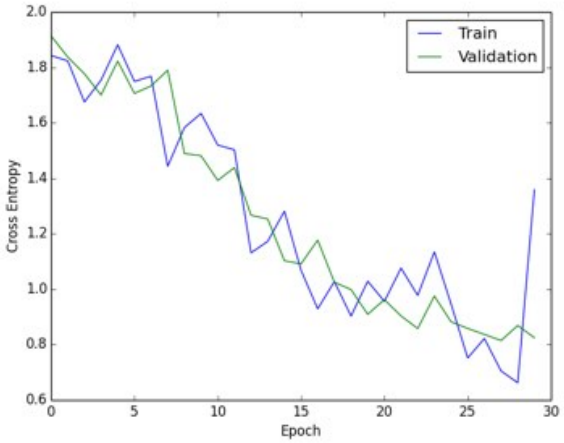
(a) Accuracy for CNN using learning rate of $\epsilon = 0.05$



(b) Cross entropy of CNN using learning rate of $\epsilon = 0.05$

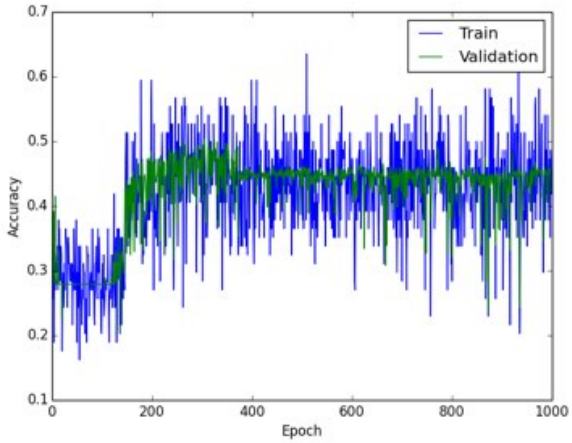


(c) Accuracy for CNN using learning rate of $\epsilon = 0.1$

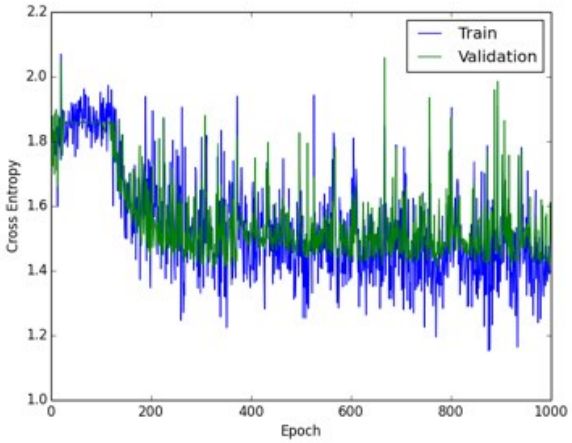


(d) Cross entropy of CNN using learning rate of $\epsilon = 0.1$

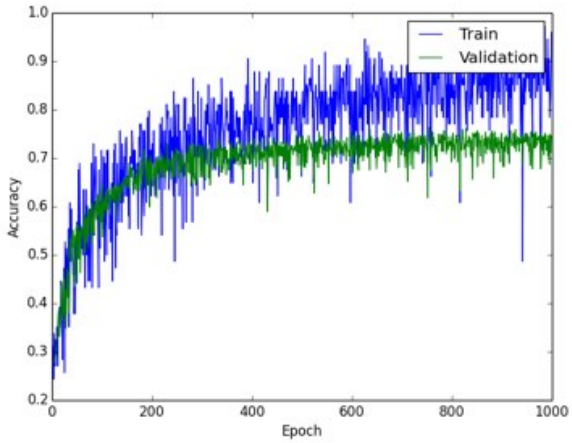
Figure 3.5: Accuracy and error curves for CNN trained at $\epsilon = 0.05, 0.1$.



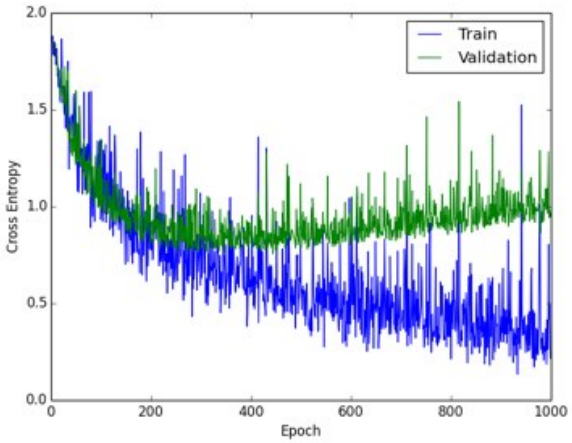
(a) Accuracy for NN using momentum of 0.9



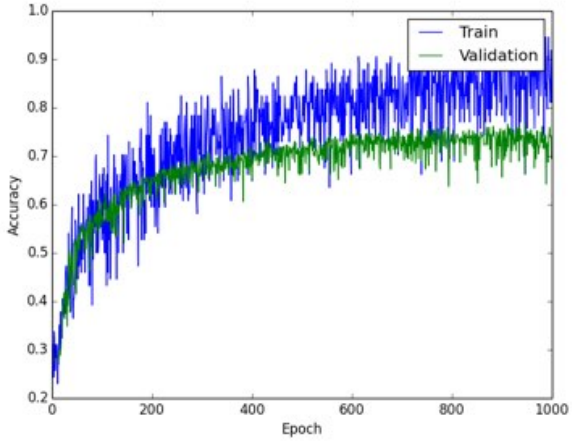
(b) Cross Entropy of NN using momentum of 0.9



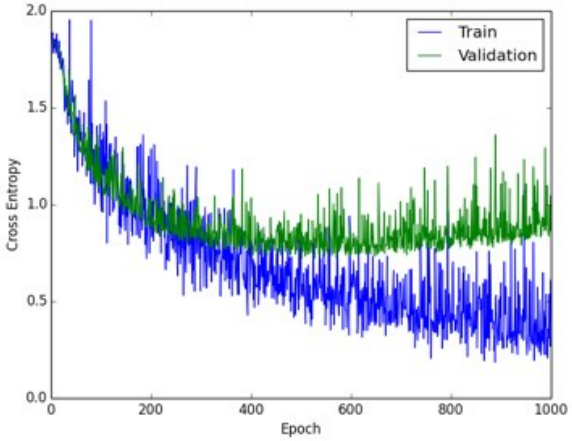
(c) Accuracy for NN using momentum of 0.5



(d) Cross entropy of NN using momentum of 0.5

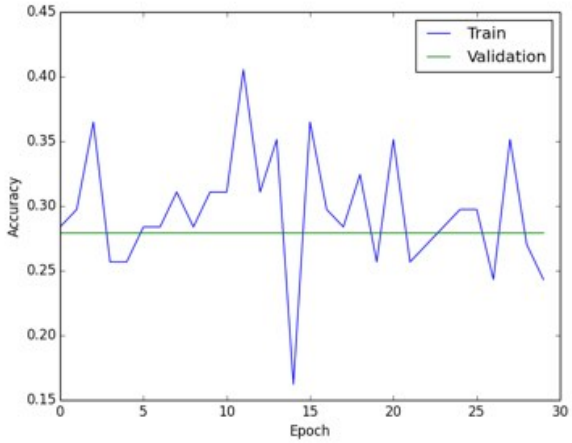


(e) Accuracy for NN using momentum of 0.1

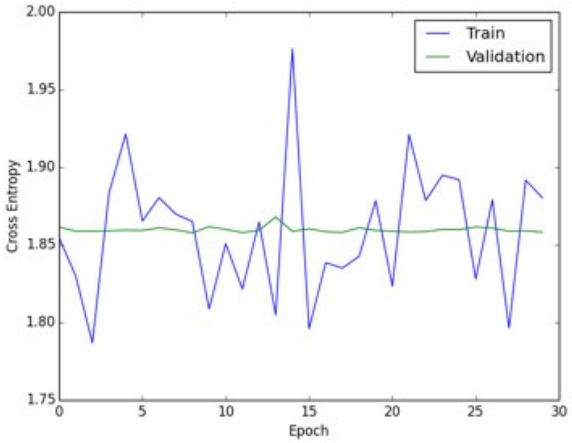


(f) Cross entropy of NN using momentum of 0.1

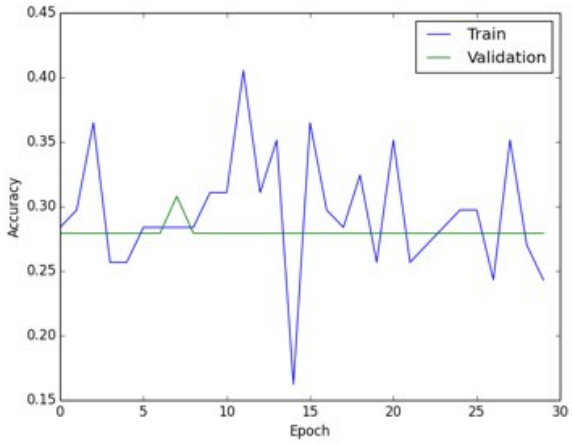
Figure 3.6: Accuracy and error curves for NN trained at $momentum = 0.9, 0.5, 0.1$.



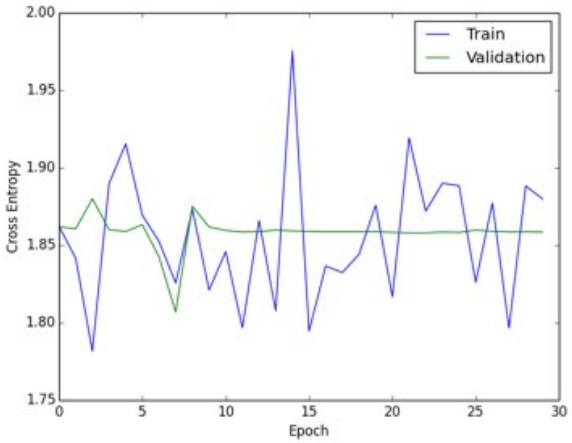
(a) Accuracy for CNN using momentum of 0.9



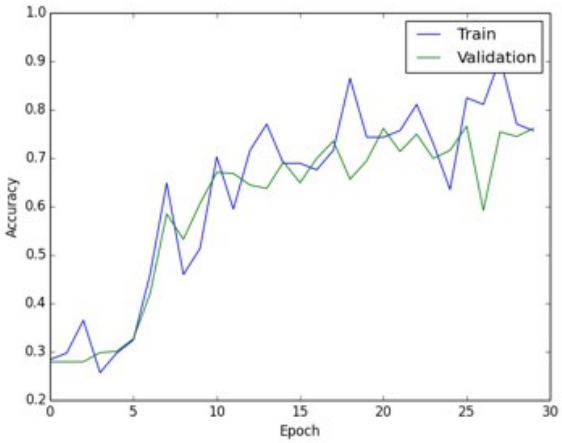
(b) Cross Entropy of CNN using momentum of 0.9



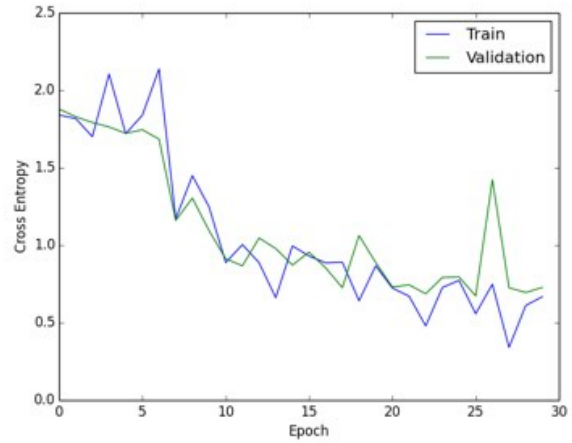
(c) Accuracy for CNN using momentum of 0.5



(d) Cross entropy of CNN using momentum of 0.5

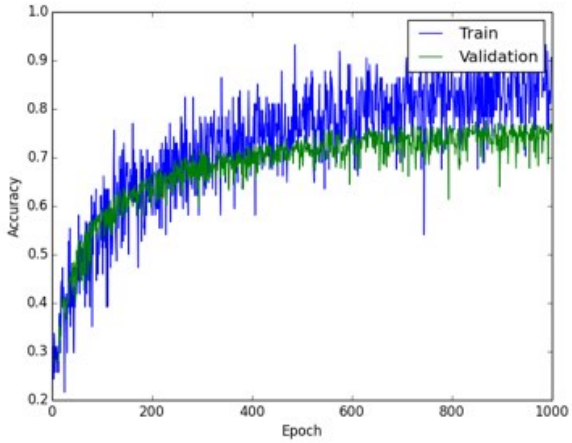


(e) Accuracy for CNN using momentum of 0.1

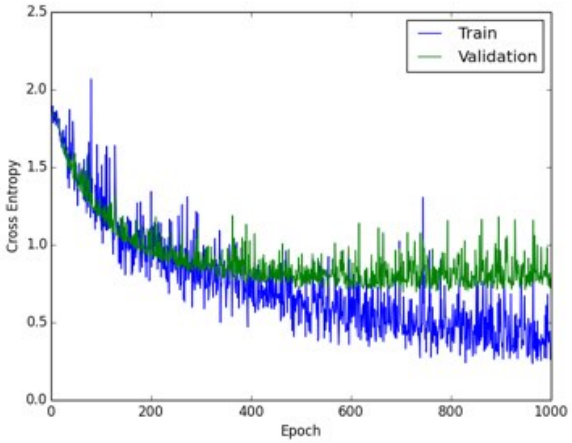


(f) Cross entropy of CNN using momentum of 0.1

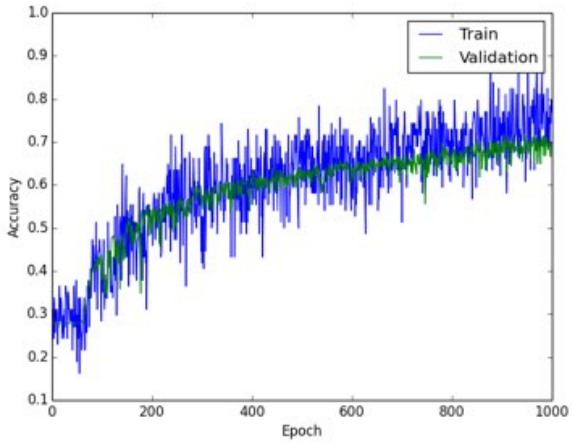
Figure 3.7: Accuracy and error curves for CNN trained at $momentum = 0.9, 0.5, 0.1$.



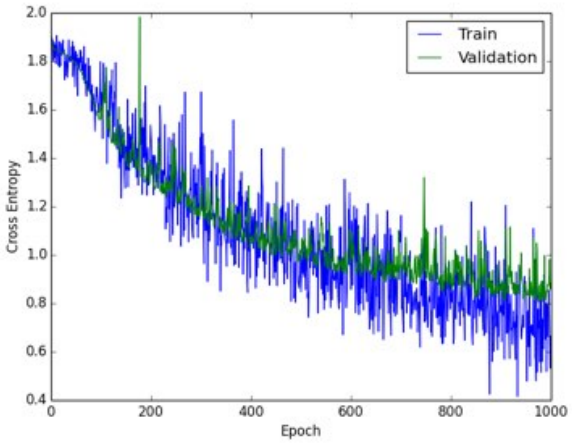
(a) Accuracy for NN using mini-batch size of 100



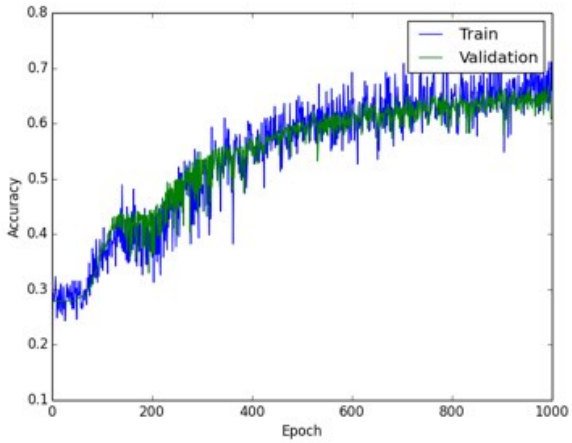
(b) Cross Entropy of NN using mini-batch size of 100



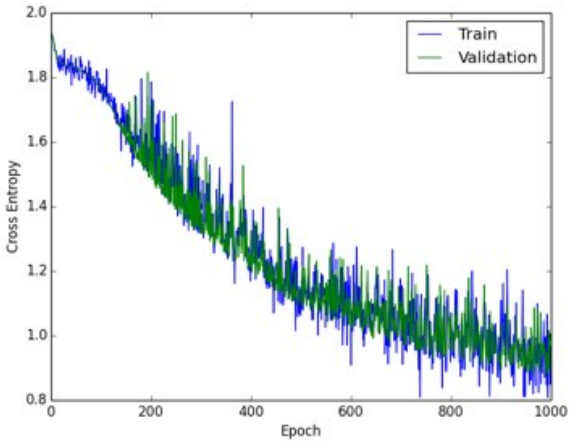
(c) Accuracy for NN using mini-batch size of 300



(d) Cross entropy of NN using mini-batch size of 300

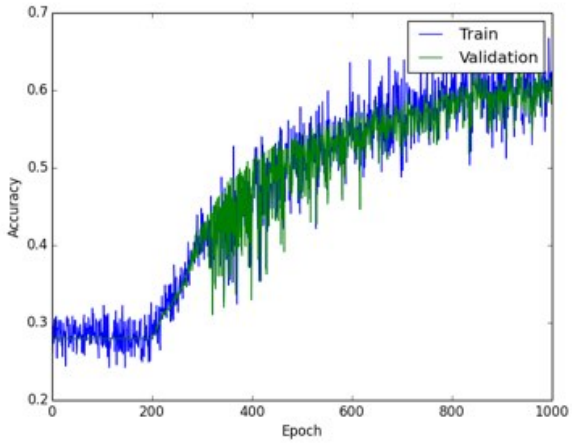


(e) Accuracy for NN using mini-batch size of 500

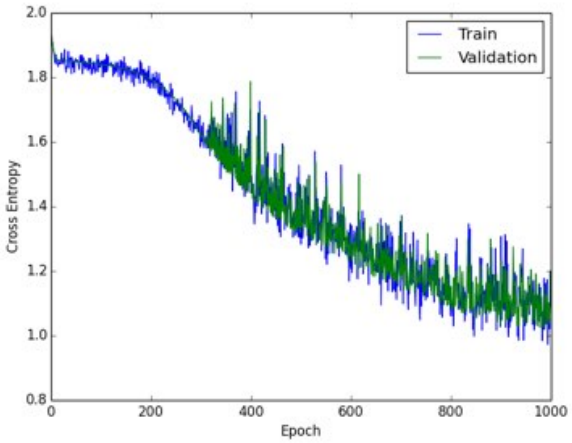


(f) Cross entropy of NN using mini-batch size of 500

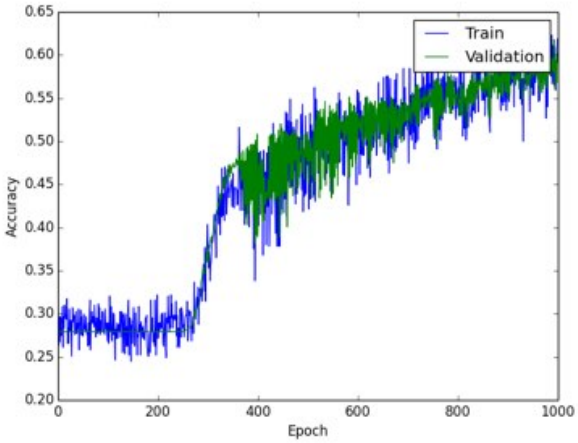
Figure 3.8: Accuracy and error curves for NN trained at mini-batch = 100, 300, 500.



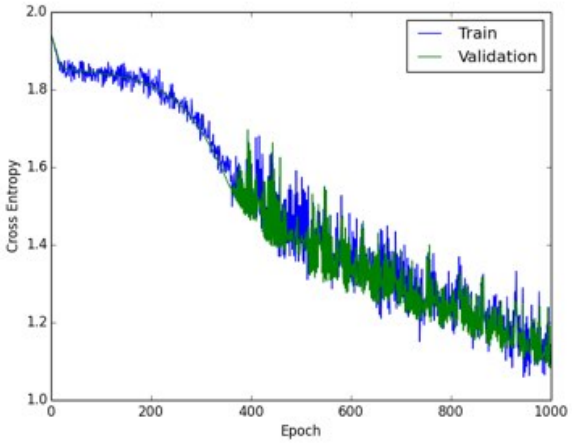
(a) Accuracy for NN using mini-batch size of 700



(b) Cross entropy of NN using mini-batch size of 700

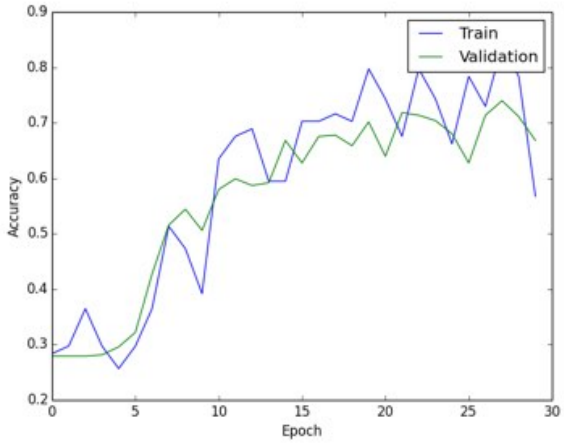


(c) Accuracy for NN using mini-batch size of 900

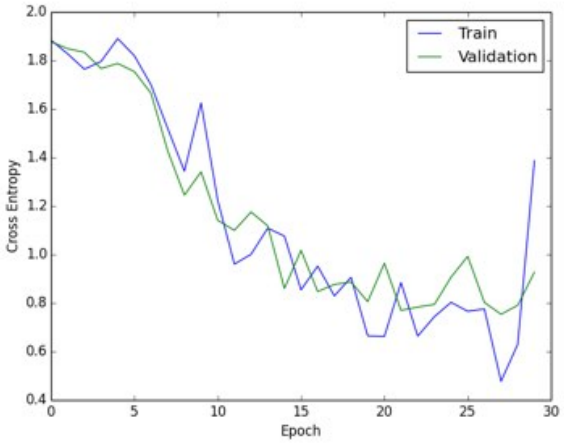


(d) Cross entropy of NN using mini-batch size of 900

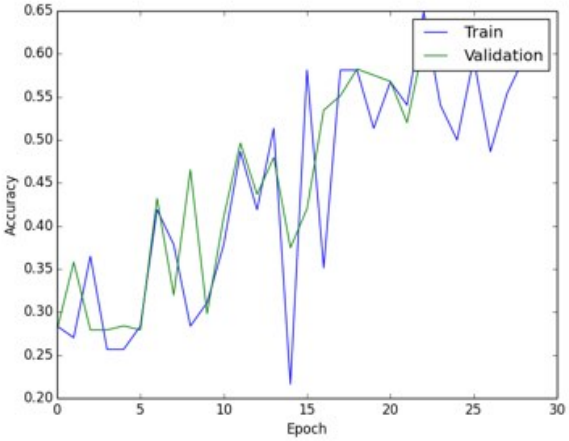
Figure 3.9: Accuracy and error curves for NN trained at mini-batch = 700, 900.



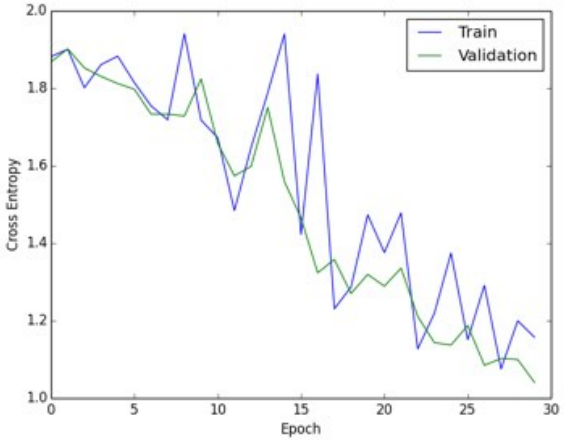
(a) Accuracy for CNN using mini-batch size of 100



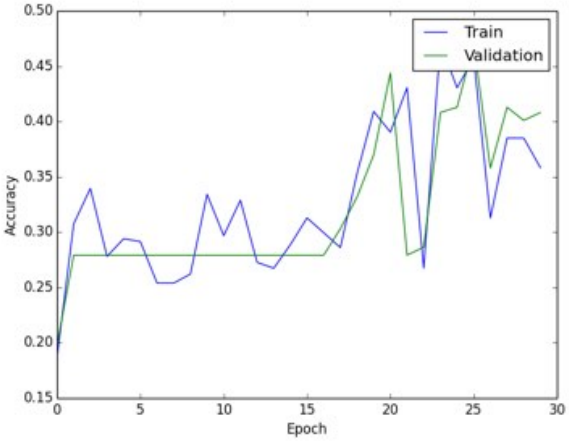
(b) Cross Entropy of CNN using mini-batch size of 100



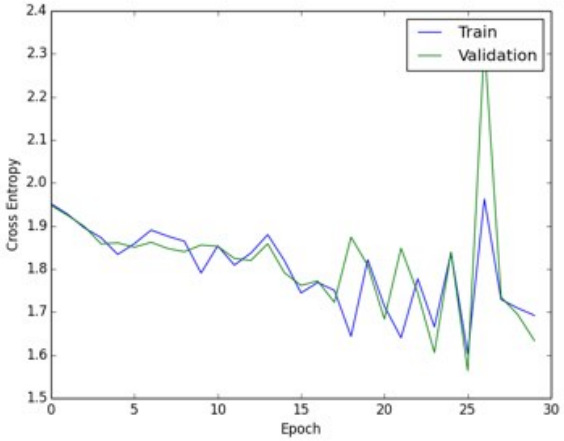
(c) Accuracy for CNN using mini-batch size of 300



(d) Cross entropy of CNN using mini-batch size of 300

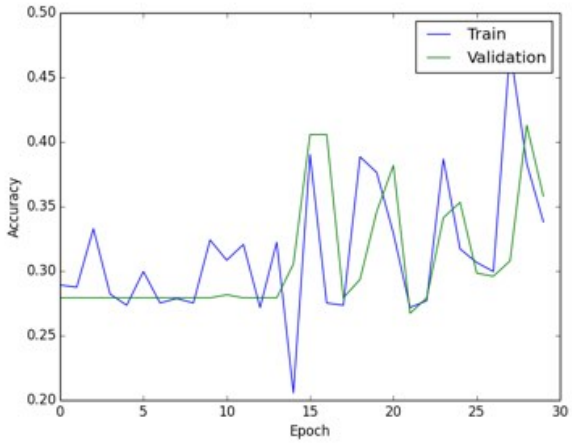


(e) Accuracy for CNN using mini-batch size of 500

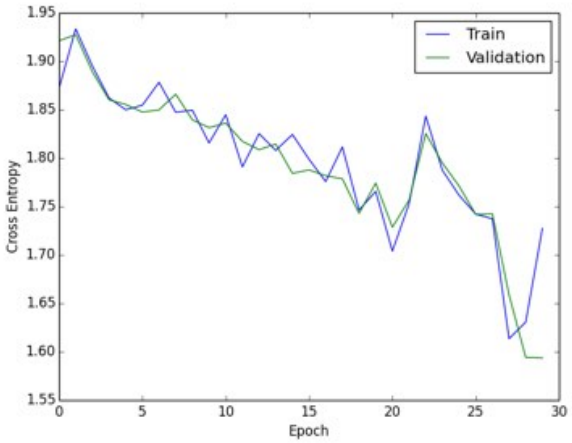


(f) Cross entropy of CNN using mini-batch size of 500

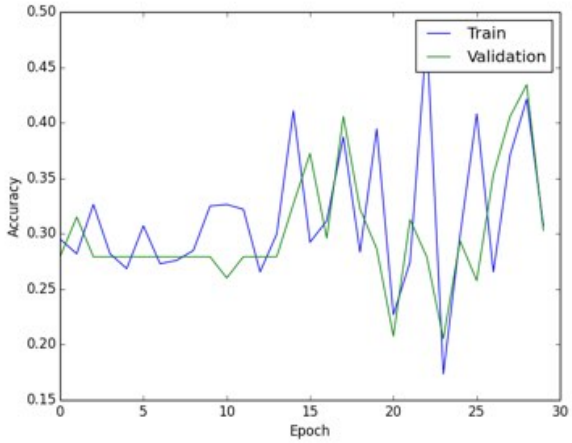
Figure 3.10: Accuracy and error curves for CNN trained at mini-batch = 100, 300, 500.



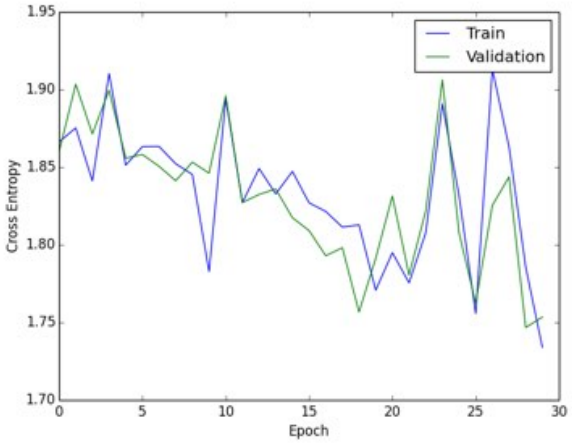
(a) Accuracy for CNN using mini-batch size of 700



(b) Cross entropy of CNN using mini-batch size of 700



(c) Accuracy for CNN using mini-batch size of 900



(d) Cross entropy of CNN using mini-batch size of 900

Figure 3.11: Accuracy and error curves for NN trained at mini-batch = 700, 900.

This way, the hyperparameters wont affect one another, and the best model will be reached, optimizing each time by modifying only one variable.

3.3 Model Architecture

The number of hidden units for the NN was modified over different models, the results of these models is showed in figures 3.12 and 3.13. Figure 3.12 portrays the results for when the first layer of hidden units is maintained at 16, and the second layer's size of filter changes. Three sizes for the second layer were examined,

$$16, 24, 40$$

. The highest accuracy for this model was reached when using the filters of size 40 in the second layer, although it's important to point out that for all models the result were very similar. In general, increasing the size of the second layer filter improves the result.

Figure 3.13 displays the results obtained for the NN models trained using a constant second hidden layer of size 32, while the first layes size varied over

$$8, 16, 24$$

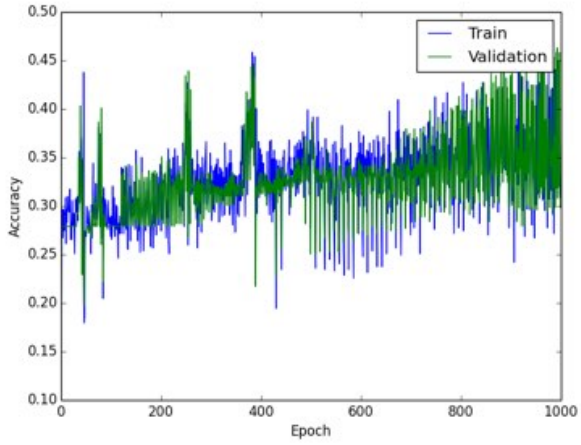
. As was obtained in the previous experiment, increasing the filter size leads to a higher accuracy. And, between the six models trained, the best accuracy is achieved for that one with 16x40 filter sizes. For future tests, a model using a larger filter size for both layers can be tried out, assuming the accuracy will be higher.

For the CNN, different models were trained using different values for the number of filters in each layer. The results is shown in the figures 3.14, 3.15, and 3.16. Figure 3.14 holds the first layer of the net at a size of 8 and modifies the values of the second layer between [4, 8, 24]. This model, as may be seen in the figures, is not performing any learning, and the accuracy for all of the models is the same. This is due to the fact that the epochs and learning rate were used as default. Figure 3.15 displays the result of modifying the CNN using a fixed second layer size of 16, and varying over the first layer size using [8, 16, 24]. The same problem as before is encountered.

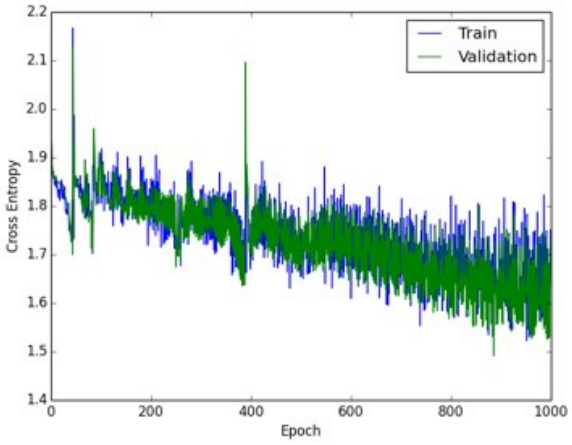
Figure 3.16 shows two CNN models trained using different learning rate and epoch values. 3.16.a shows the accuracy for the CNN model trained using eps equal to 0.01, and filters 32x50, for 30 epochs and a 100 batch size. This model achieved a 51.067% accuracy over the training set, 55.609% over the validation set, and 49.610% over the test set. All of the accuracies obtained were close to each other, however the cross entropy in 3.16.b shows that more epochs are needed in order to make this model learn what it needs to learn. Its results were still better than those portrayed in figures 3.14 and 3.15.

Figure 3.16.c shows the accuracy for a CNN model trained using eps equal to 0.01, 60 epochs, filters 24x40, and a batch size of 100. This model achieved higher accuracy than the past model, 90.041% in the training set, 75.418% in the validation set, and 79.481% in the testing set. The cross entropy in this model is closer to 0, which means that it is closer to learning correctly the parameters that are introduced into the model. This might be due to the amount of epochs. However, it is also possible that this filter size is better for capturing the features needed from the faces in order to determine their emotions.

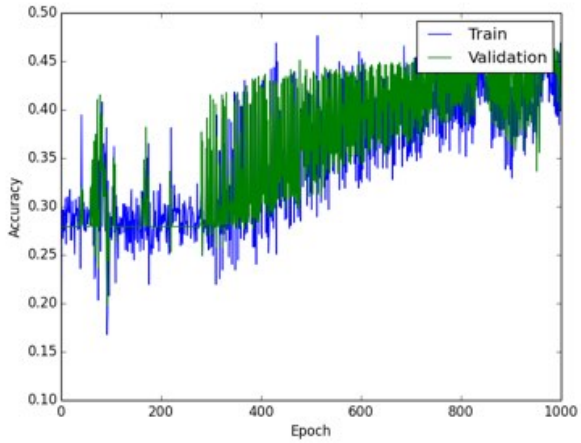
The general effect of the filters on the convergence for both models is that a larger filter will be able to detect the features in a more effective way. Nonetheless, a filter that is too large will possible miss the feature. The network may be generalized using larger filters, between 24 and 40 in size, and a larger number of epochs, around 80 epochs. If too many epochs are used, the model might be overfitted to the training set.



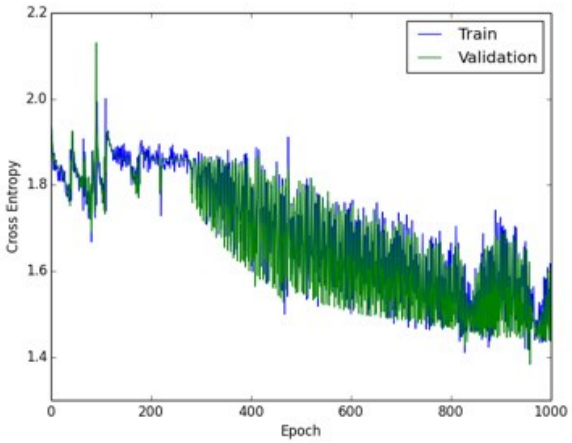
(a) Accuracy for NN using 16x16



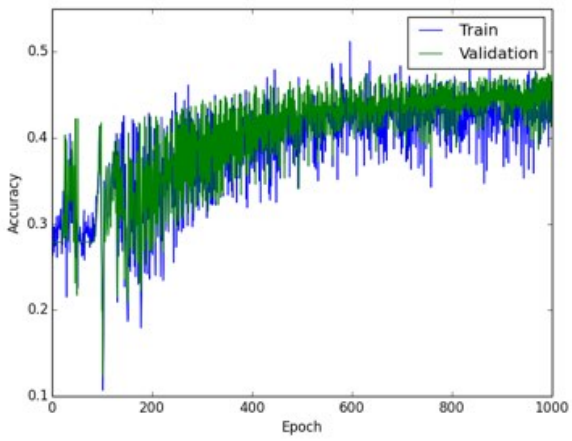
(b) Cross Entropy of NN using 16x16



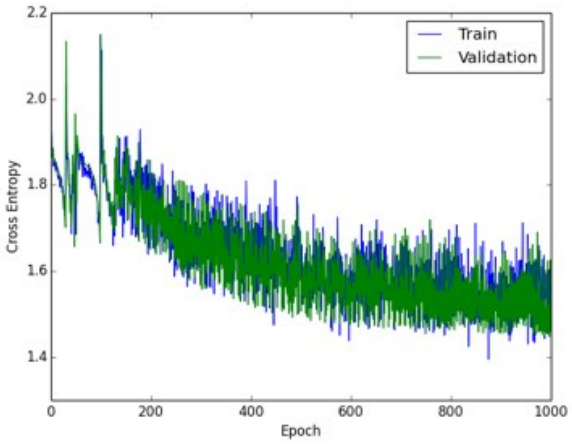
(c) Accuracy for NN using 16x24



(d) Cross entropy of NN using 16x24

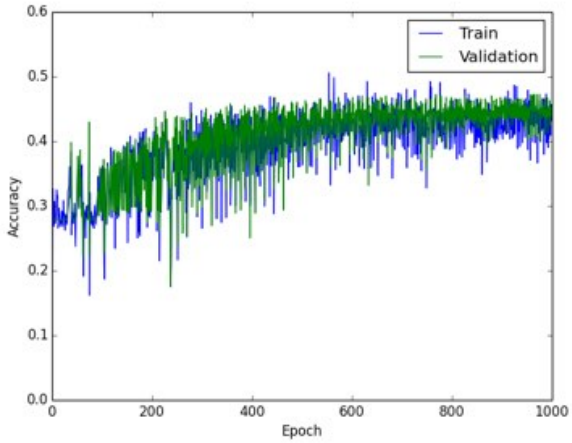


(e) Accuracy for NN using 16x40

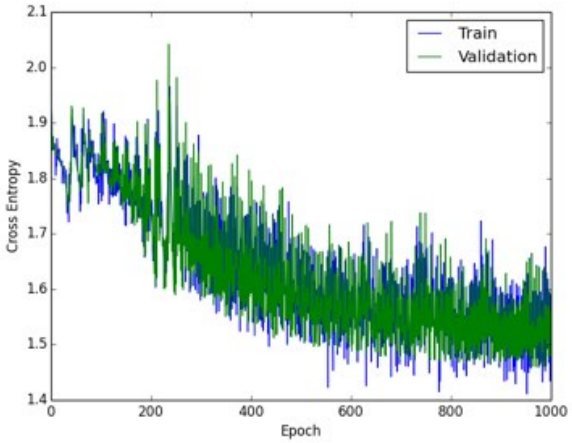


(f) Cross entropy of NN using 16x40

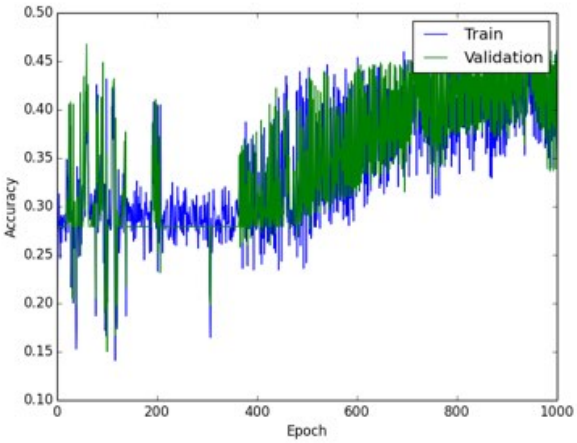
Figure 3.12: Accuracy and error curves for NN trained using first hidden layer at 16 and modifying the second hidden layers using values of 16, 24, 40.



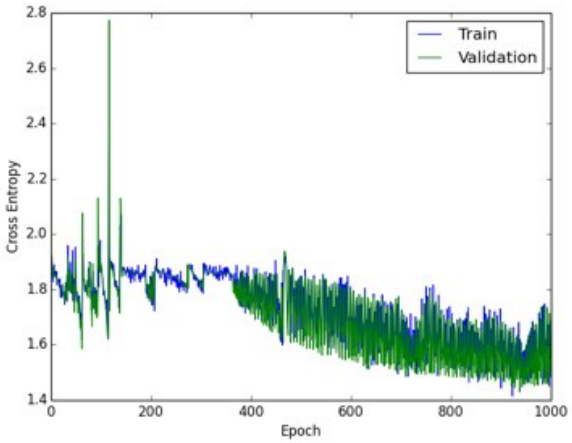
(a) Accuracy for NN using 8x32



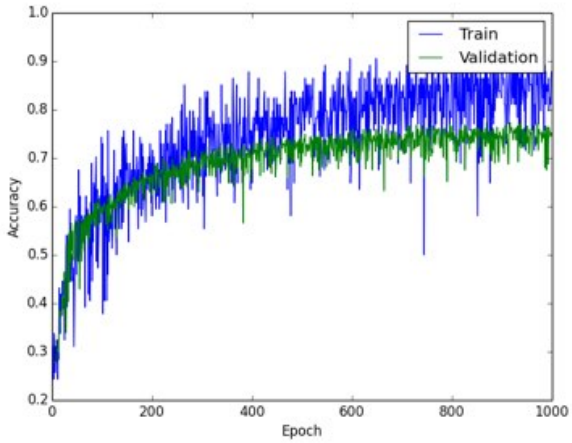
(b) Cross Entropy of NN using 8x32



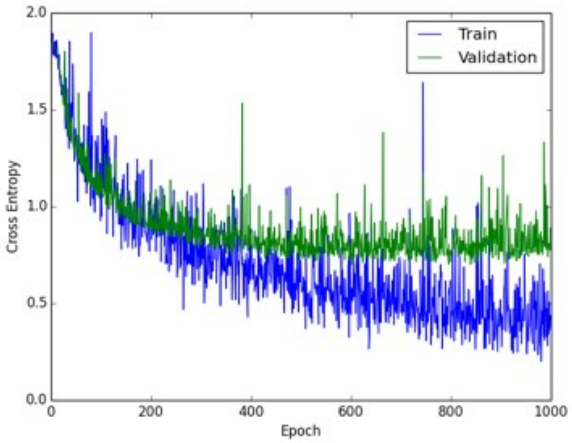
(c) Accuracy for NN using 24x32



(d) Cross entropy of NN using 24x32

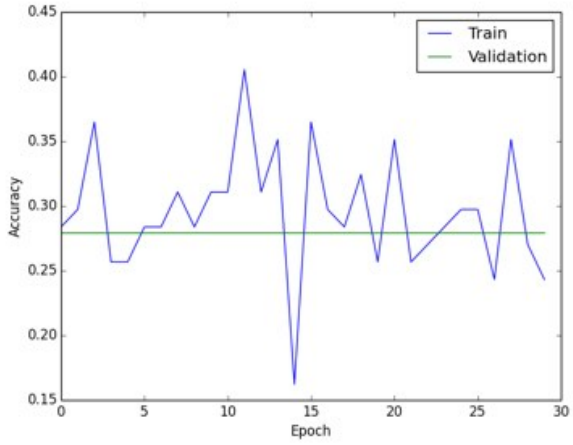


(e) Accuracy for NN using 16x32

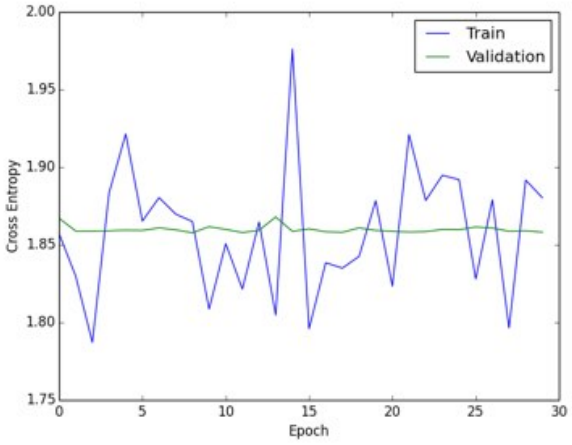


(f) Cross entropy of NN using 16x32

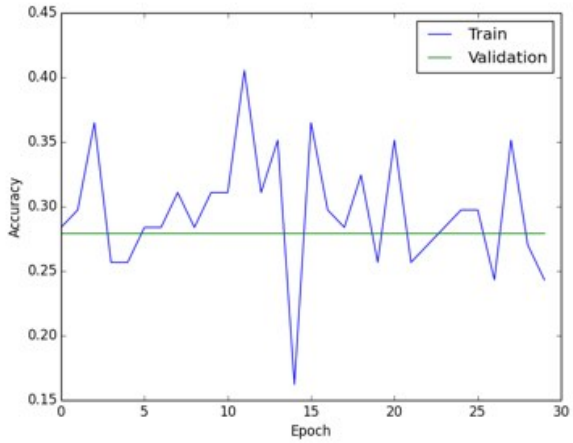
Figure 3.13: Accuracy and error curves for NN trained using second hidden layer at 32 and modifying the first hidden layers using values of 8, 16, 24.



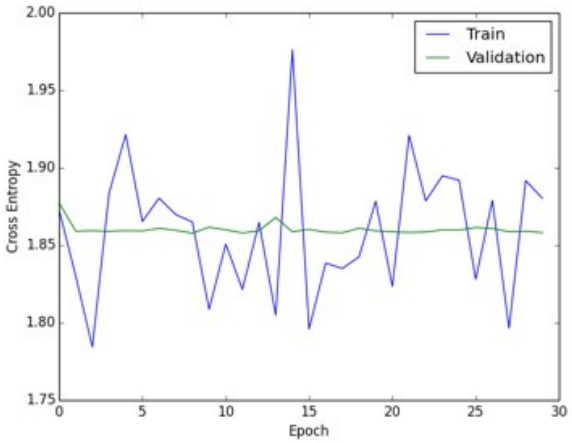
(a) Accuracy for CNN using 8x4



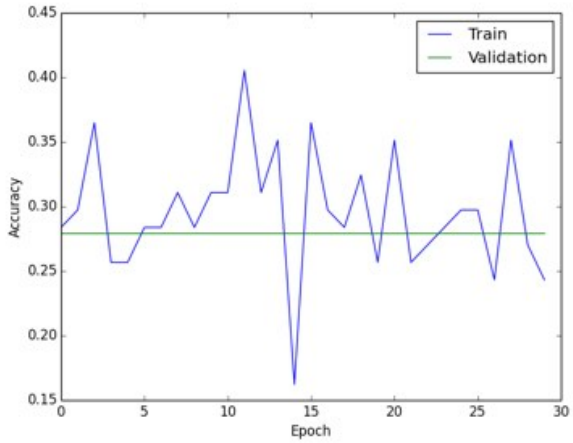
(b) Cross Entropy of CNN using 8x4



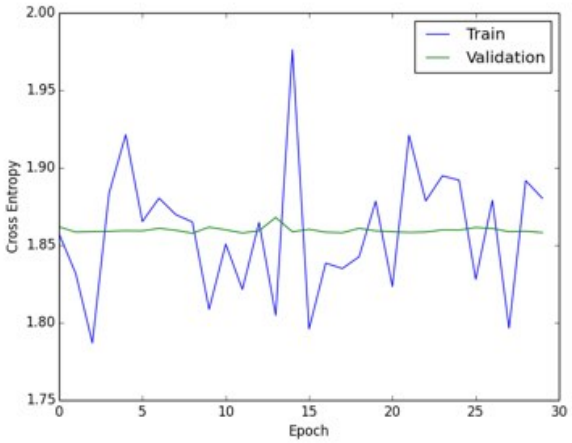
(c) Accuracy for CNN using 8x24



(d) Cross entropy of CNN using 8x24

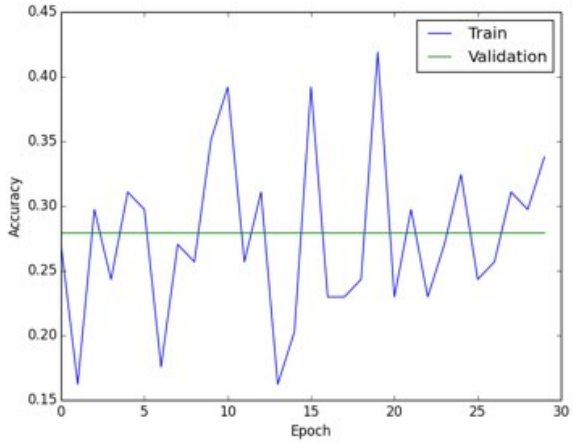


(e) Accuracy for CNN using 8x8

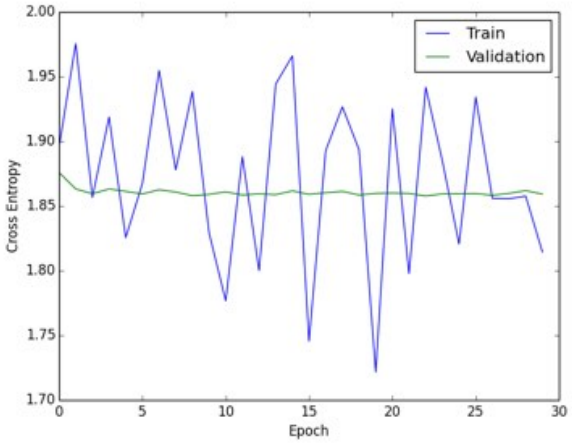


(f) Cross entropy of CNN using 8x8

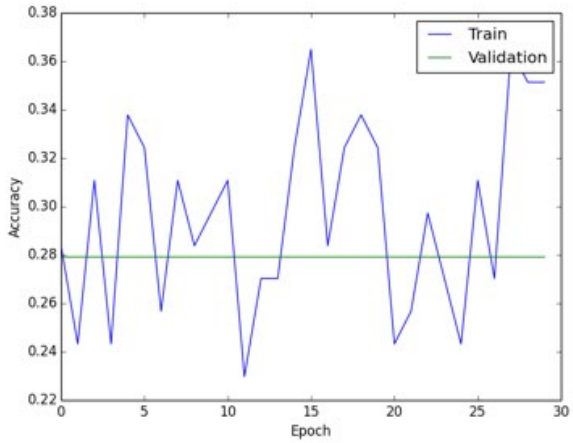
Figure 3.14: Accuracy and error curves for CNN trained using first hidden layer at 8 and modifying the second hidden layers using values of 4,24,8.



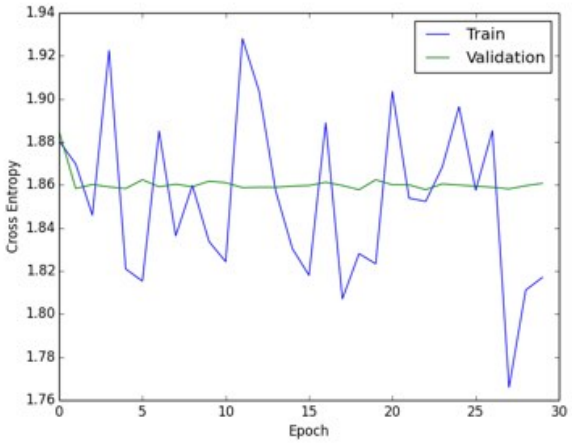
(a) Accuracy for CNN using 24x16



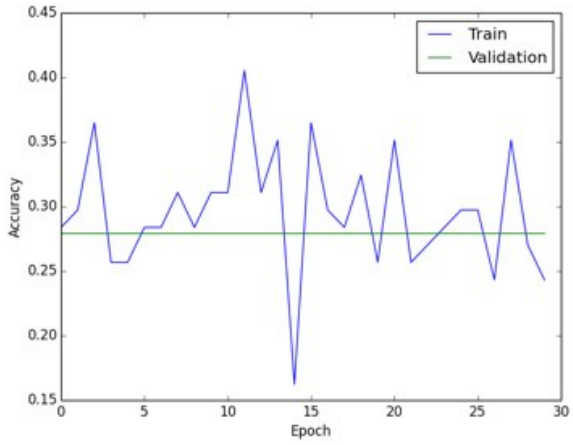
(b) Cross Entropy of CNN using 24x16



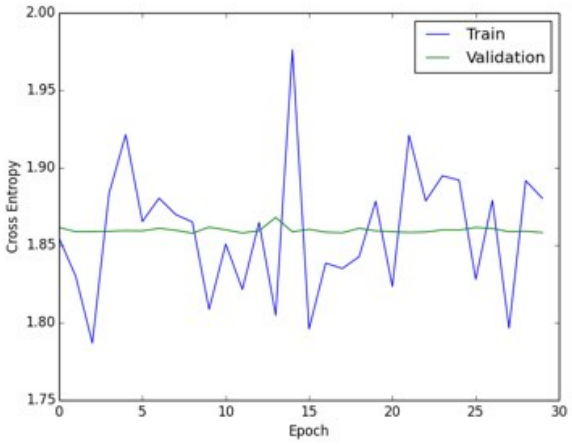
(c) Accuracy for CNN using 16x16



(d) Cross entropy of CNN using 16x16

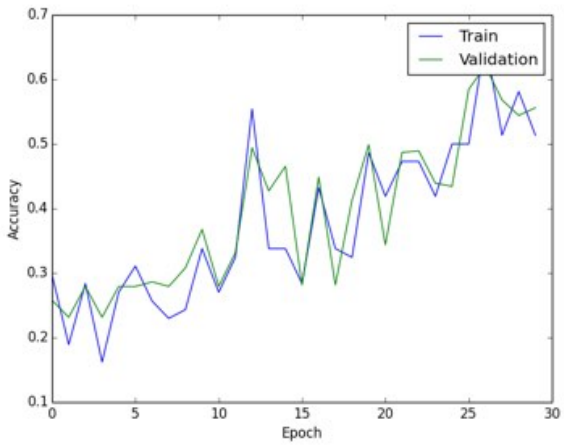


(e) Accuracy for CNN using 8x16

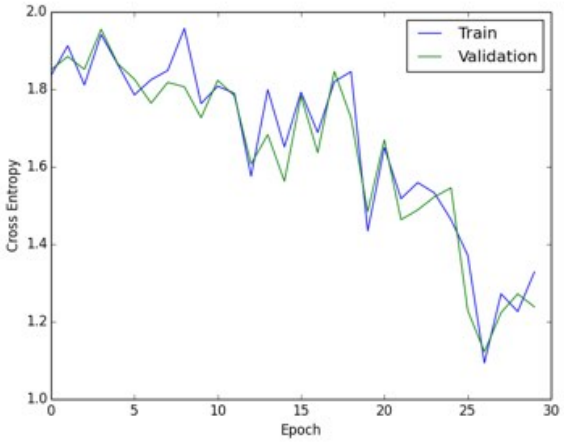


(f) Cross entropy of CNN using 8x16

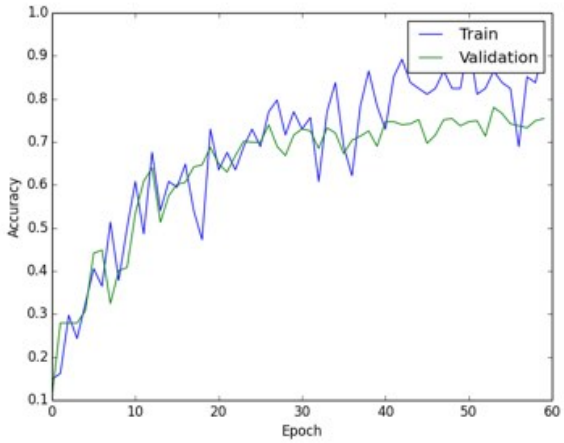
Figure 3.15: Accuracy and error curves for CNN trained using second hidden layer at 16 and modifying the first hidden layers using values of 8, 16, 24.



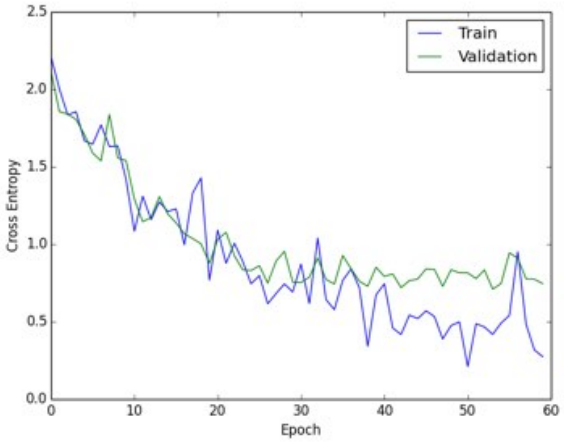
(a) Accuracy for CNN using 32x50, eps=0.01, epochs=30.



(b) Cross Entropy of CNN using 32x50, eps=0.01, epochs=30.



(c) Accuracy for CNN using 24x40, eps=0.01, epochs=60



(d) Cross entropy of CNN using 24x40, eps=0.01, epochs=60

Figure 3.16: Accuracy and error curves for CNN trained using different filter sizes, and different eps and epochs than the default, with momentum 0.9

3.4 COMPARE CNNS AND FULLY CONNECTED NETWORKS

Using the CNN model described in figure 3.16.c and 3.16.d, and the NN model described in figure 3.3.a and 3.3.b as the similar models. The CNN model that uses an eps of 0.01, momentum of 0.9, 60 epochs, and a 24x40 filters, 42591 parameters are obtained by summing over the filters and biases. For the NN model using the default parameters, and an eps set o 0.05, 37655 parameters are obtained by summing over the weights and biases. The CNN reaches 85% of accuracy in the training set, 75% in the validation set and 74% in the testing set. On the other hand, the NN reaches 86% on the training set, 71% on the validation set, and 76% on the testing set.

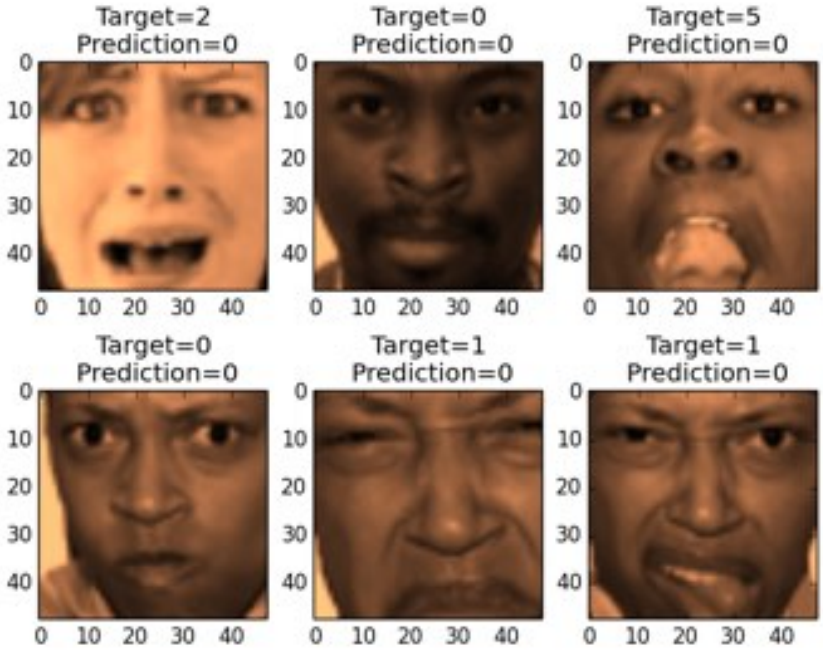
(a) First layer of weights for the fully connected neural network

(b) First layer of filters for the CNN

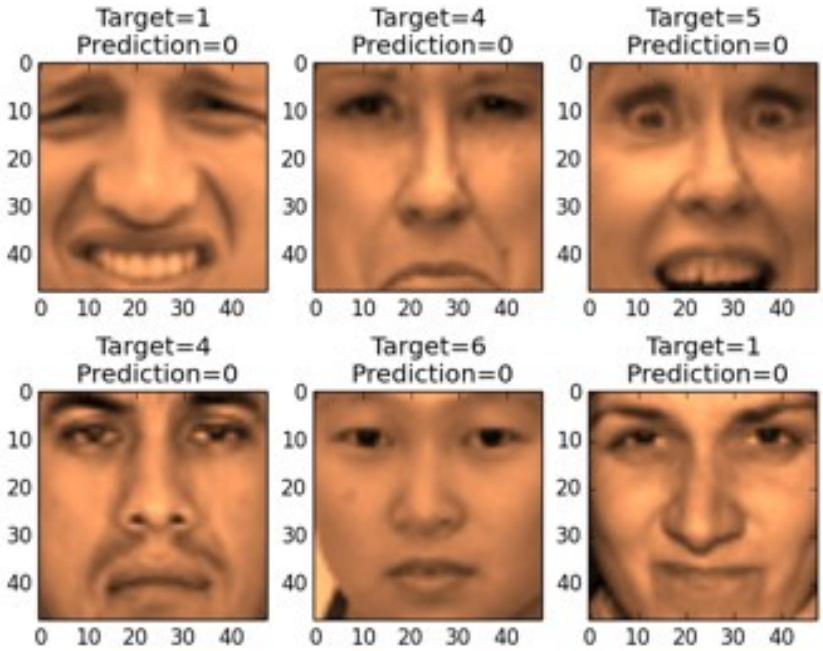
Figure 3.17: Comparison on the first layer of filters and weights for the CNN and NN respectively

Figure 3.17 shows how the first layer of weights for the NN displays features of faces, like eyes, mouth, eyebrows, nose, amongst other features. In contrast, the CNN first layer of filters only shows pixels which highlight certain aspects within a picture. However, it would be very hard to determine what it is the machine is trying to do with these. Probably the number of epochs has a lot to do with it. The NN trains for 900 epochs, which would allow it to train a better weights layer, while the CNN has only trained for 60, leading to the pixelated response.

The better generalization is lead by the NN model, for it trains faster than CNN and provides a weights layer that extracts features that play an important role when determining emotions from facial expressions. It also achieves a little better results using less parameters.



(a) Examples where the CNN is not confident using threshold at 0.5



(b) Examples where the NN is not confident using threshold at 0.5

Figure 3.18: Comparison of the network uncertainty in CNN and NN

In order to determine the uncertainty of the classifiers, a threshold of 0.5 was used as the top scoring value from which a classification would be taken as certain. For this question, the validation set was used. For the CNN, the classifier has the most confusion identifying anger, all the cases examined in figure 3.18.a show the classifier falsely identifying anger in 4/6 cases, and being correct but uncertain in 2/6 of the cases. The classifier mostly confuses anger with disgust. If the classifier were to choose the output on the top scoring class, it woud be right around 33%, assuming the sample is representative of the whole set.

For the NN, in figure 3.18.b, the target for anger is the most confused as well. Though the performance is worsened, for this time 5/6 classifications are wrong, and only 1/6 is correct but uncertain. It is mostly confusing anger with fear, and disgust. If the classifier were to chose the output on the top scoring class, it would be wrong. However these examples are hard to classify, for the expression are not very clear.

4 Mixture of Gaussians

4.1 Code

4.2 Training

Figure 4.1. shows the visualization of the mean vector and the variance vector of a mixture of gaussian model using randConst 1.0. the mixing coefficients are:

$$\begin{bmatrix} 0.06373954 \\ 0.20039651 \\ 0.17219636 \\ 0.07681843 \\ 0.08714358 \\ 0.14575943 \\ 0.25394615 \end{bmatrix}$$

Figure 4.2. shows the visualization of the mean vector and the variance vector of a mixture of gaussian model using randConst 5.0. the mixing coefficients are:

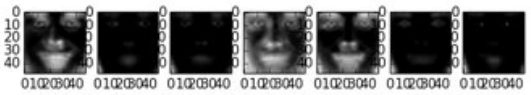
$$\begin{bmatrix} 0.24782643 \\ 0.15357497 \\ 0.07288465 \\ 0.05186755 \\ 0.17579833 \\ 0.1984613 \\ 0.09958677 \end{bmatrix}$$

Figure 4.3. shows the visualization of the mean vector and the variance vector of a mixture of gaussian model using randConst 10.0. the mixing coefficients are:

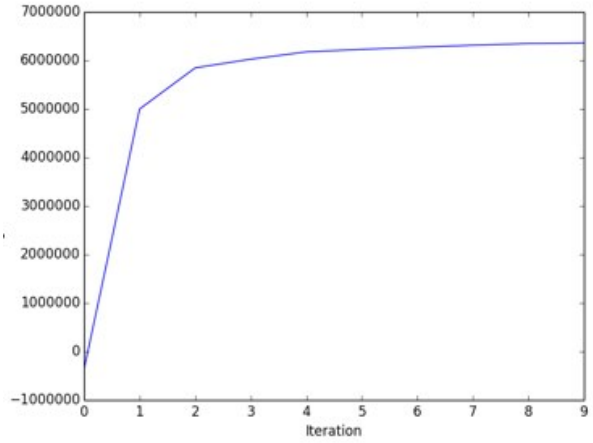
$$\begin{bmatrix} 0.08568436 \\ 0.07858868 \\ 0.17810674 \\ 0.06249704 \\ 0.12237593 \\ 0.26633376 \\ 0.20641349 \end{bmatrix}$$

Figure 4.4. shows the visualization of the mean vector and the variance vector of a mixture of gaussian model using randConst 100.0. the mixing coefficients are:

(a) Mean vector for Mixture of Gaussian model



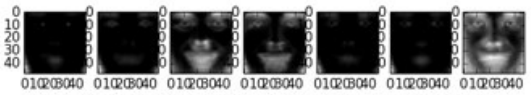
(b) Variance vector for Mixture of Gaussian model



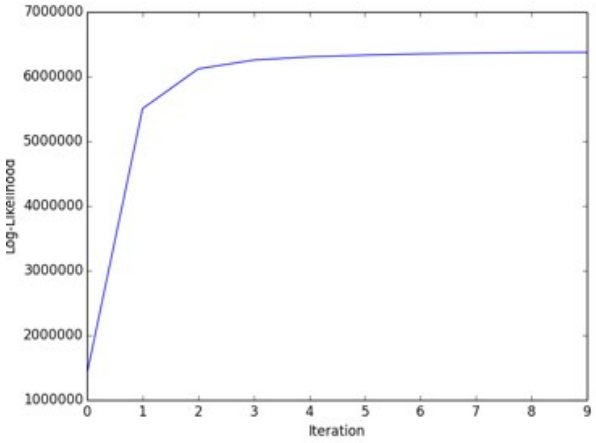
(c) Likelihood mixing proportion of the clusters

Figure 4.1: Results for the Mixture of Gaussians model using randConst=1.0

(a) Mean vector for Mixture of Gaussian model



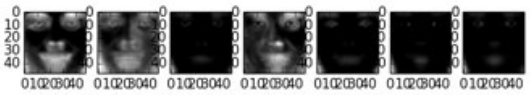
(b) Variance vector for Mixture of Gaussian model



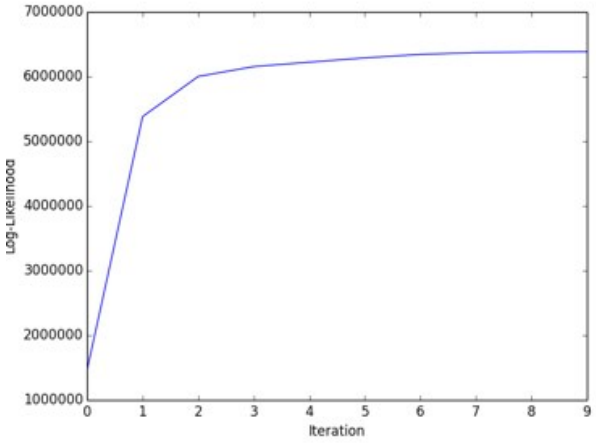
(c) Likelihood mixing proportion of the clusters

Figure 4.2: Results for the Mixture of Gaussians model using randConst=5.0

(a) Mean vector for Mixture of Gaussian model



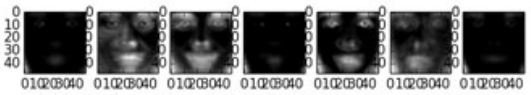
(b) Variance vector for Mixture of Gaussian model



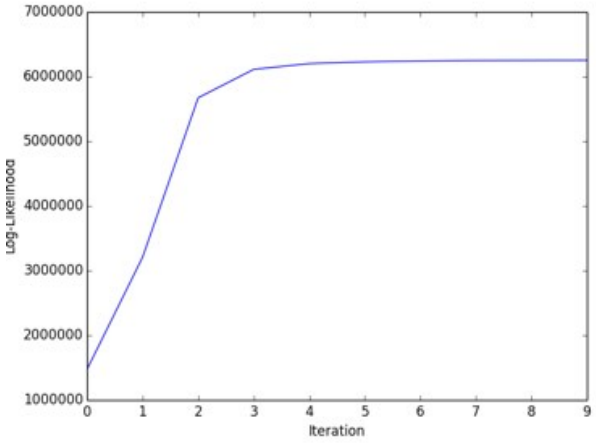
(c) Likelihood mixing proportion of the clusters

Figure 4.3: Results for the Mixture of Gaussians model using randConst=10.0

(a) Mean vector for Mixture of Gaussian model



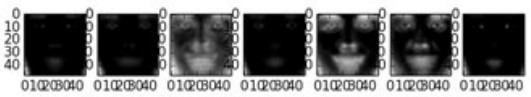
(b) Variance vector for Mixture of Gaussian model



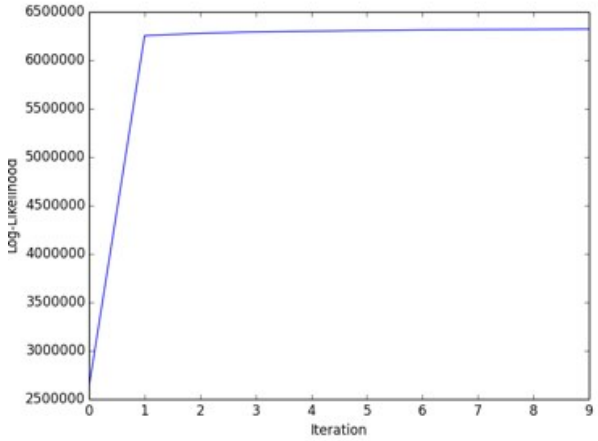
(c) Likelihood mixing proportion of the clusters

Figure 4.4: Results for the Mixture of Gaussians model using randConst=100.0

(a) Mean vector for Mixture of Gaussian model



(b) Variance vector for Mixture of Gaussian model



(c) Likelihood mixing proportion of the clusters

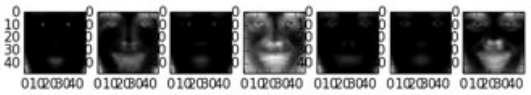
Figure 4.5: Results for the Mixture of Gaussians model using randConst 1.0

$$\begin{bmatrix} 0.21189946 \\ 0.06135153 \\ 0.13603072 \\ 0.28157521 \\ 0.07206022 \\ 0.03845229 \\ 0.19863058 \end{bmatrix}$$

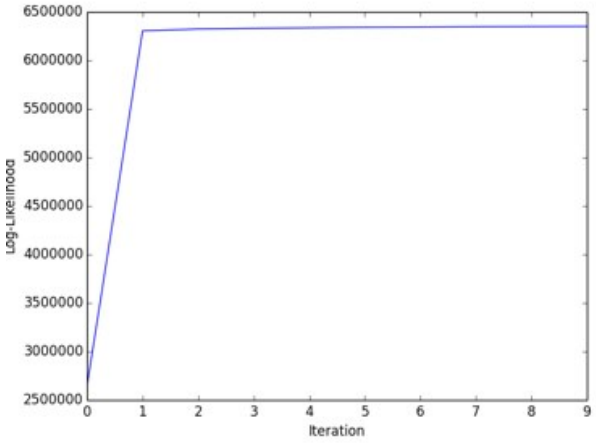
4.3 Initializing a mixture of Gaussians with k-means

Figures 4.5, 4.6, 4.7, 4.8 show the MoG model using an initialization with k-means for randConst of 1.0, 5.0, 10.0 and 100.0. The convergence of the model using K-means, comparing figure 4.1 and figure 4.5 is much higher in figure 4.5, which uses k-means. The log-likelihood for the convergence of MoG with k-means is reached at the second value. The MoG model reaches convergence at the eighth iteration of the log-likelihood. This shows that the speed of convergence for a model that initializes with k-means is higher than that of a model that does not.

(a) Mean vector for Mixture of Gaussian model



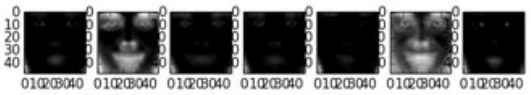
(b) Variance vector for Mixture of Gaussian model



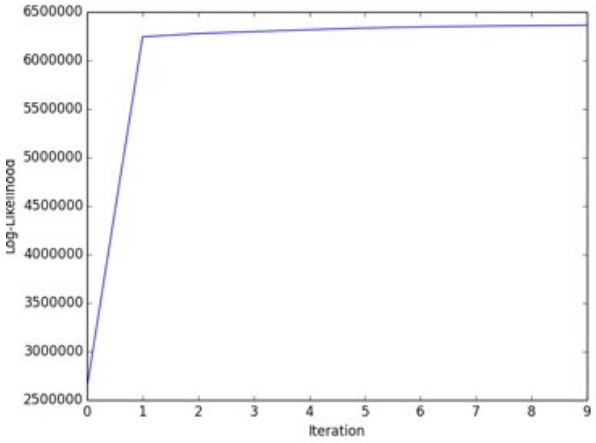
(c) Likelihood mixing proportion of the clusters

Figure 4.6: Results for the Mixture of Gaussians model using randConst 5.0

(a) Mean vector for Mixture of Gaussian model



(b) Variance vector for Mixture of Gaussian model

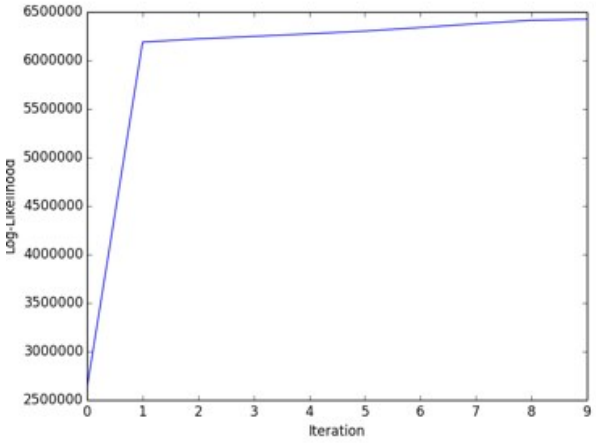


(c) Likelihood mixing proportion of the clusters

Figure 4.7: Results for the Mixture of Gaussians model using randConst 10.0

(a) Mean vector for Mixture of Gaussian model

(b) Variance vector for Mixture of Gaussian model



(c) Likelihood mixing proportion of the clusters

Figure 4.8: Results for the Mixture of Gaussians model using randConst 100.0

For the cases with higher randConst, which converge faster than with values closer to 1 in this variable, the log-likelihood is reached at the second step for the k-means initialized model, while it takes the MoG model around 6 iterations to reach the higher values for convergence.

4.4 Classification using MoGs

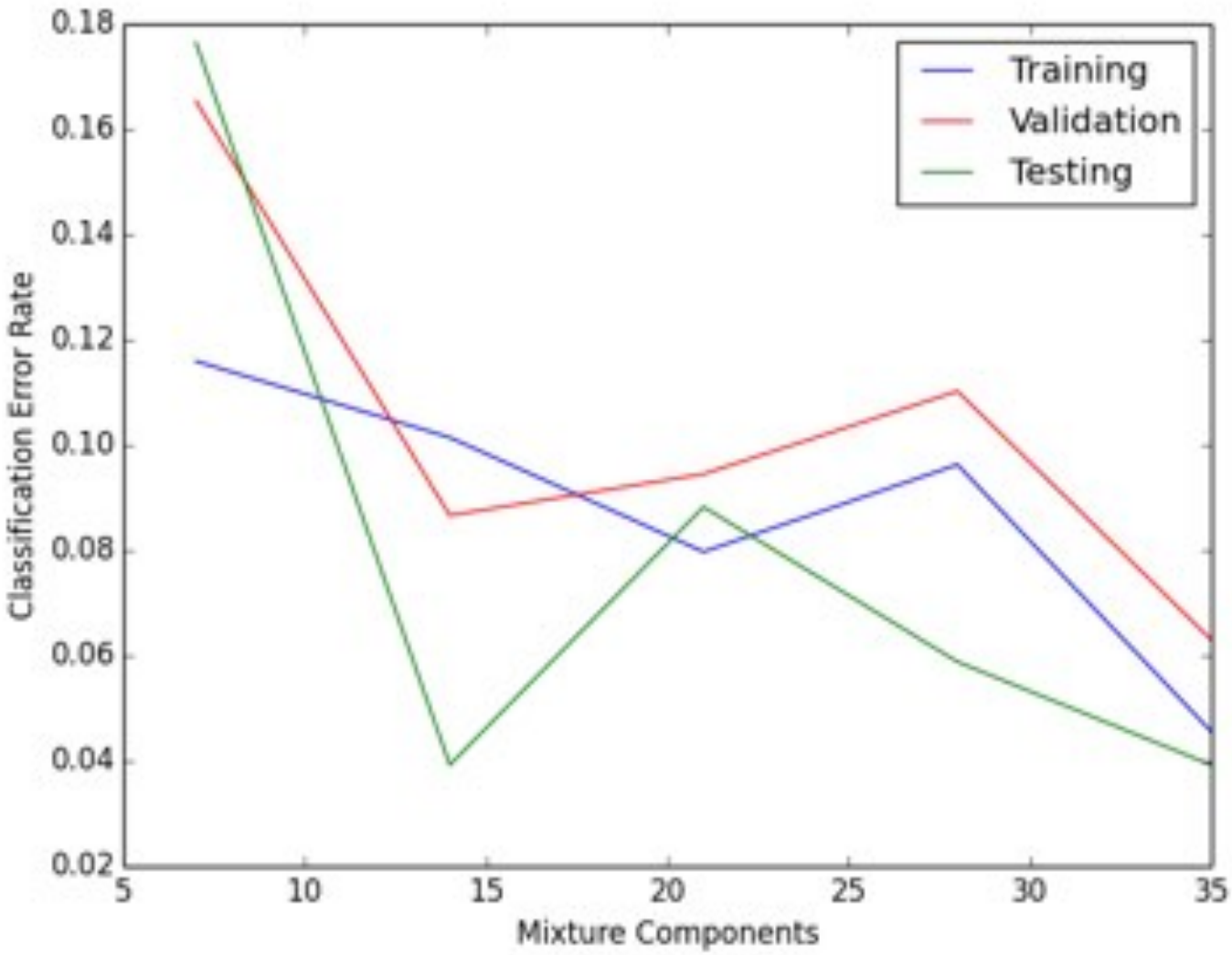


Figure 4.9: Classification error rate on the training set, validation set, test set.

4.4.1 Error rates on the training sets generally decrease as the number of clusters increases

Figure 5.1 shows how the error rate on the training sets decreases in most cases as the number of clusters increases. What this is representing is how the model is becoming more sensible as it clusters the data. As more gaussians are included in the modelling of the class, the probability that data will be produced exactly as observed increases. By mixing the Gaussians in a class, densities are approximated to what the the data is portraying.

4.4.2 Properties and trends of the error rate curve for the test set

The green line in figure 5.1. represents the error within the test set. Every change made in the model that is trained in the training set is felt more hardly in the testing set. Even more so than in the validation set. It seems as if the data for angry and happy faces in the training set may be more similar to the data in the validation set. But, the data in the validation set is more similar to the data in the testing set. As the number of mixture components within each class is increased, the curve on the test set starts to behave more like the training set or the validation set. The testing set also displays a lower classification error rate. So, when the number of mixture components increases, the model becomes more accurate as to training for the determination of angry vs happy in faces it has not yet seen. And it also achieves a lower error than what the model achieves within the training set and the validation set.