1 EM for Mixture of Gaussians

We use the following model for $p(\vec{x})$

$$p(\vec{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}|\vec{\mu}_k, \Sigma_k) \tag{1}$$

where

$$\sum_{k=1}^{K} \pi_k = 1 \tag{2}$$

The loss function is defined as

$$l(\boldsymbol{X}|\vec{\pi}; \mu; \Sigma) = \sum_{n=1}^{N} ln(\sum_{k=1}^{K} \pi_k \mathcal{N}(\vec{x}^n|\vec{\mu}_k, \Sigma_k)) \tag{3}$$

A useful quantity for the following derivations is

$$\gamma_k^n = \frac{\pi_k \mathcal{N}(\vec{x}^n|\vec{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k)} \tag{4}$$

1.1 $\vec{\mu}$ update

$$\frac{\partial l}{\partial \vec{\mu}_k} = \sum_{n=1}^{N} \frac{\frac{\partial}{\partial \vec{\mu}_k}(\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k))}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k)} = \sum_{n=1}^{N} \frac{\pi_k \frac{\partial}{\partial \vec{\mu}_k}(\mathcal{N}(\vec{x}^n|\vec{\mu}_k, \Sigma_k))}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k)} \tag{5}$$

$$\begin{aligned} \frac{\partial \mathcal{N}(\vec{x}|\vec{\mu}_j, \Sigma_k)}{\partial \vec{\mu}_k} = \frac{\partial}{\partial \vec{\mu}_k} \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} e^{\frac{-1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} = \\ \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} \frac{\partial}{\partial \vec{\mu}_k}(-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)) = \\ \mathcal{N}(\vec{x}|\vec{\mu}_j, \Sigma_k)\Sigma_k^{-1}(\vec{x}-\vec{\mu}_k) \end{aligned} \tag{6}$$

Plugging equation 6 into equation 5 we get

$$\frac{\partial l}{\partial \vec{\mu}_k} = \sum_{n=1}^{N} \gamma_k^n \Sigma_k^{-1}(\vec{x}^n - \vec{\mu}_k) \tag{7}$$

Setting equation 7 to $\vec{0}$ and solving for $\vec{\mu}_k$ gives the update rule

$$\vec{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^n \vec{x}^n \tag{8}$$

where N_k is defined as $N_k = \sum_{n=1}^{N} \gamma_k^n$

1.2 Σ update

$$\frac{\partial l}{\partial \Sigma_k} = \sum_{n=1}^{N} \frac{\frac{\partial}{\partial \Sigma_k}(\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k))}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k)} = \sum_{n=1}^{N} \frac{\pi_k \frac{\partial}{\partial \Sigma_k}(\mathcal{N}(\vec{x}^n|\vec{\mu}_k, \Sigma_k))}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k)} \tag{9}$$

$$\begin{aligned}
\frac{\partial \mathcal{N}(\vec{x}|\vec{\mu}_j, \Sigma_k)}{\partial \Sigma_k} &= \frac{\partial}{\partial \Sigma_k} \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} e^{\frac{-1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} = \\
&\frac{\partial}{\partial \Sigma_k}(\frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}}) e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} + \\
\frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} &\frac{\partial}{\partial \Sigma_k}(-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)) = \\
&-\frac{1}{2\sqrt{(2\pi)^d |\Sigma_k|^{\frac{3}{2}}}} |\Sigma_k| \Sigma_k^{-1} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} + \\
\frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1}(\vec{x}-\vec{\mu}_k)} &(\frac{1}{2}(\vec{x}-\vec{\mu}_k)(\vec{x}-\vec{\mu}_k)^T (\Sigma_k^{-1})^2) = \\
&\frac{1}{2}\mathcal{N}(\vec{x}|\vec{\mu}_j, \Sigma_k)((\vec{x}-\vec{\mu}_k)(\vec{x}-\vec{\mu}_k)^T \Sigma_k^{-1} - I)\Sigma_k^{-1}
\end{aligned} \tag{10}$$

Plugging equation 10 into equation 9 yields

$$\frac{\partial l}{\partial \Sigma_k} = \sum_{n=1}^{N} \frac{\pi_k \frac{\partial}{\partial \Sigma_k}(\mathcal{N}(\vec{x}^n|\vec{\mu}_k, \Sigma_k))}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma_k)} = \frac{1}{2}\sum_{n=1}^{N} \gamma_k^n((\vec{x}^n - \vec{\mu}_k)(\vec{x}^n - \vec{\mu}_k)^T\Sigma_k^{-1} - I)\Sigma_k^{-1} \tag{11}$$

Setting equation 11 equal to 0 and solving for Σ_k we arrive at the following update rule

$$\Sigma_k = \frac{1}{N_k}\sum_{n=1}^{N} \gamma_k^n(\vec{x}^n - \vec{\mu}_k)(\vec{x}^n - \vec{\mu}_k)^T \tag{12}$$

If instead we work under the assumption that $\Sigma_j = \Sigma$ for all $1 \leq j \leq K$ then the we have

$$\begin{aligned}\frac{\partial l}{\partial \Sigma} = \sum_{n=1}^{N} \frac{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma)((\vec{x}^n - \vec{\mu}_j)(\vec{x}^n - \vec{\mu}_j)^T\Sigma^{-1} - I)\Sigma^{-1}}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\vec{x}^n|\vec{\mu}_j, \Sigma)} = \\ \sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j^n((\vec{x}^n - \vec{\mu}_j)(\vec{x}^n - \vec{\mu}_j)^T\Sigma^{-1} - I)\Sigma^{-1}\end{aligned} \tag{13}$$

Setting this equation equal to 0 leads to

$$\begin{aligned}&\sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j^n((\vec{x}^n - \vec{\mu}_j)(\vec{x}^n - \vec{\mu}_j)^T - \Sigma) = 0 \\ &\Rightarrow \Sigma = \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j^n(\vec{x}^n - \vec{\mu}_j)(\vec{x}^n - \vec{\mu}_j)^T\end{aligned} \tag{14}$$

1.3 $\vec{\pi}$ update

To compute the update rule for $\vec{\pi}$ we will need to introduce a latent variable $\vec{z}^n$ which is a 1 in K encoding of the knowledge of the actual Gaussian used to generate $\vec{x}^n$. Utilizing this new variable our loss function becomes

$$l(\boldsymbol{X}|Z;\vec{\pi};\mu;\Sigma) = \sum_{n=1}^{N}\sum_{j=1}^{K} z_j^n ln(\pi_j) + ln(\mathcal{N}(\vec{x}^n|\vec{\mu_j};\Sigma)) \tag{15}$$

Since we must optimize this equation under the constraint imposed by equation 2 we will proceed by utilizing a Lagrange multiplier λ

$$\mathcal{L}(\vec{\pi}) = l(\vec{\pi}) - \lambda\sum_{j=1}^{K}\pi_j - 1 \tag{16}$$

$$\nabla\mathcal{L} = [\sum_{n=1}^{N}\frac{z_1^n}{\pi_1} - \lambda, ..., \sum_{n=1}^{N}\frac{z_K^n}{\pi_K} - \lambda, \sum_{j=1}^{K}\pi_j] \tag{17}$$

Setting equation 15 to $\vec{0}$ requires $\forall_j\ \pi_j\lambda = \sum_{n=1}^{N} z_j^n$. From this fact we can write $\lambda = \sum_{j=1}^{K}\pi_j\lambda = \sum_{j=1}^{K}\sum_{n=1}^{N} z_j^n = \sum_{j=1}^{K} N_j = N$. Hence our update rule is

$$\pi_k = \frac{N_k}{N} \tag{18}$$

2 Convolutional Neural Networks

The definition of the convolution operation $*$ in this context is

$$y_{n,h',w',k} = x_{n,h,w,k} * f_{i,j,c,k} = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{c=1}^{C} x_{n,h+i-1,w+i-1,k} f_{i,j,c,k} \tag{19}$$

For the rest of this section we assume $n = k = 1$. From this we have

$$\begin{aligned}\frac{\partial E}{\partial f_{i,j}} = \\ \sum_{h'=1}^{H'}\sum_{w'=1}^{W'}\frac{\partial y_{h',w'}}{\partial f_{i,j}}\frac{\partial E}{\partial y_{h',w'}} = \\ \sum_{h'=1}^{H'}\sum_{w'=1}^{W'}x_{h+i-1,w+j-1}\frac{\partial E}{\partial y_{h',w'}} = \\ x^{(I-1,J-1)} * (\frac{\partial E}{\partial y})\end{aligned} \tag{20}$$

I could not figure out how to derive the second equation :(

3 Neural Networks

3.1 Notation

For ease of reference the following notation scheme as been adopted throughout this section.

Phrase	Notation
Neural Network	NN
Convolutional Neural Network	CNN
learning rate	ϵ
momentum	m
batch size	b

Table 1: Notation

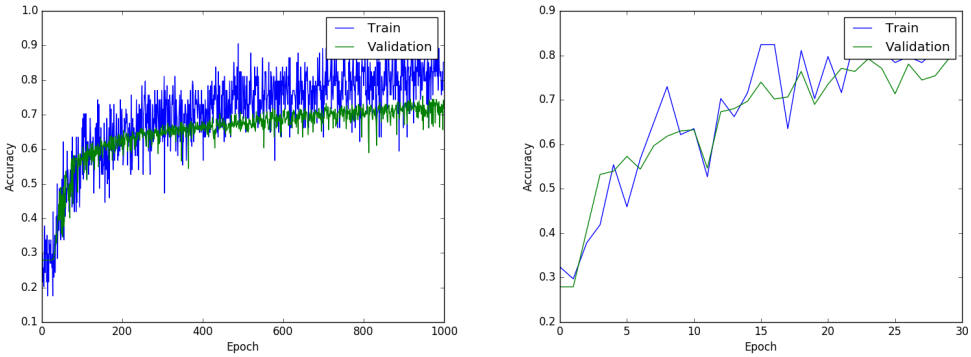
3.2 Basic generalization

Figure 1 shows a comparison of the accuracy of a NN against the accuracy of a CNN for a default set of hyperparameters. The exact values of those hyperparameters are shown in table 2. Both charts show the training set accuracy and validation set accuracy as a function of the current epoch. The overall trend in both is that the training set accuracy leads the

validation set accuracy, however the difference between them is not great. This is the ideal case where training our network for a long period of time has not resulted in overfitting, as the network generalizes to the validation set well. A notable difference between the two however is that the CNN reaches high levels of accuracy in fewer epochs then the NN does. Also of note is the noise in both figures. This is a function of the batch size used. The closer we get to using a batch size of 1 the closer we are to performing stochasitc optimization, which in general is more noisy and not guaranteed to move towards the optimum at each step

	ϵ	m	b
NN	0.01	0.0	100
CNN	0.1	0.0	100

Table 2: Default hyperparameters for NN and CNN



(a) NN Accuracy

(b) CNN accuracy

Figure 1: Comparison of accuracy between a NN and a CNN for the default set of hyperparameters

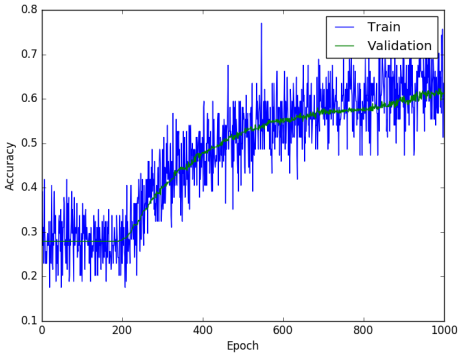
3.3 Optimization

In this section we will try various settings for our 3 hyperparameters in an attempt to tune both our NN and CNN to the ideal fit. The ϵ is set to $\{0.001, 0.01, 0.1, 0.5, 1.0\}$, the m to $\{0.0, 0.450.9\}$, and the b to $\{1, 10, 100, 500, 1000\}$. For both the NN and the CNN high ϵ values resulted

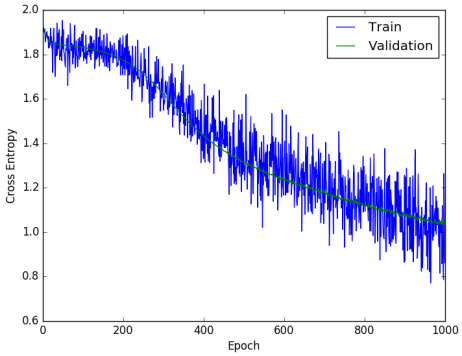
in poor performance in terms of both training accuracy and validation accuracy. $\epsilon = 0.1$ being a clear cutoff point. Both networks performed better with $m = 0.0$, although perhaps values between 0.0 and 0.45 would have outperformed the $m = 0.0$ case. Lastly both experienced increasingly erratic behavior as b decreased

3.3.1 NN

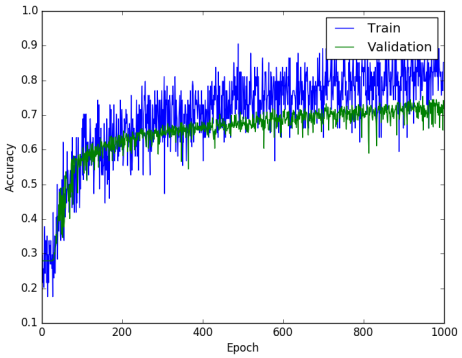
Figure 2 shows both the accuracy and cross entropy for training the NN with a given set of hyperparameters. By looking at these plots the best set of hyperperameters (in terms of validation accuracy) is either $\epsilon = 0.01$, $m = 0.0$, and $b = 100$ or $\epsilon = 0.01$, $m = 0.45$, and $b = 100$. Looking at the statistics for both selections yeilds no significant difference between the validation accuracy of one or the other. Therefore it makes most sense to choose $m = 0.0$ as it reduces the model complexity by effectively removing a hyperparameter.
The NN stomached lower values of ϵ than the CNN, performing okay with $\epsilon = 0.01$. This is in line with it's optimal ϵ being lower than the ϵ for the CNN



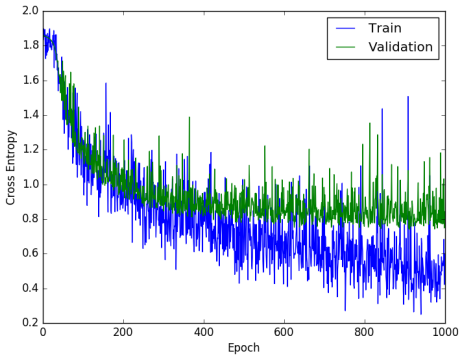
(a) $\epsilon = 0.001$, $m = 0.0$, $b = 100$



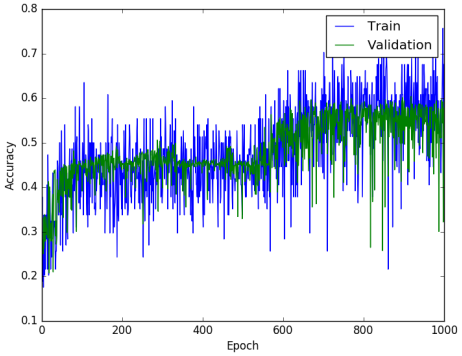
(b) $\epsilon = 0.001$, $m = 0.0$, $b = 100$



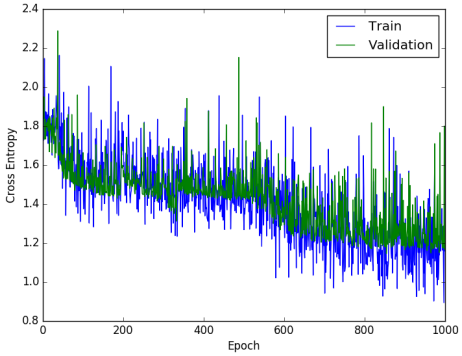
(c) $\epsilon = 0.01$, $m = 0.0$, $b = 100$



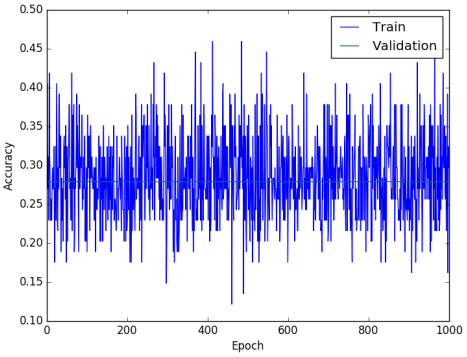
(d) $\epsilon = 0.01$, $m = 0.0$, $b = 100$



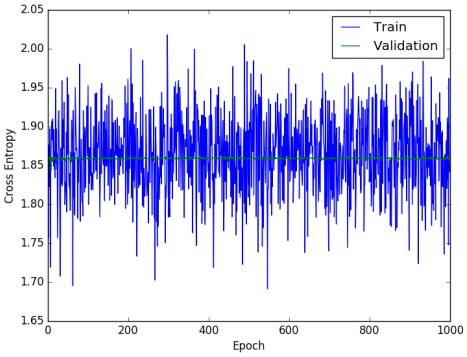
(e) $\epsilon = 0.1$, $m = 0.0$, $b = 100$



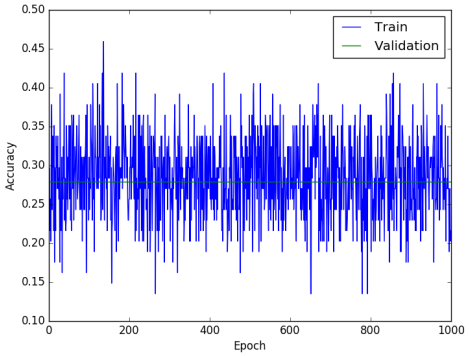
(f) $\epsilon = 0.1$, $m = 0.0$, $b = 100$



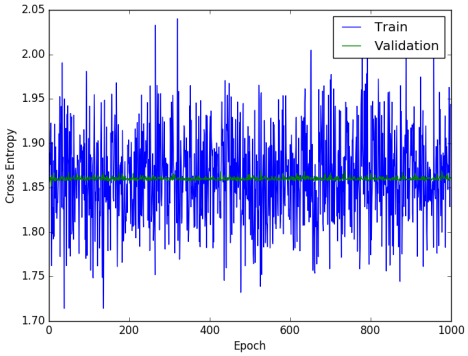
(g) $\epsilon = 0.5$, $m = 0.0$, $b = 100$



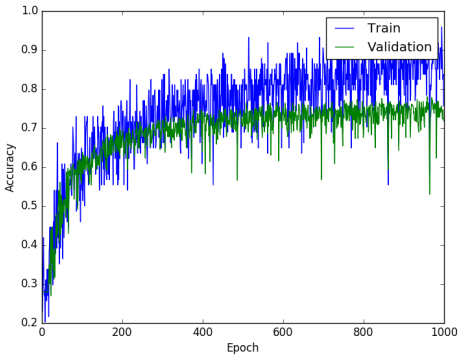
(h) $\epsilon = 0.5$, $m = 0.0$, $b = 100$



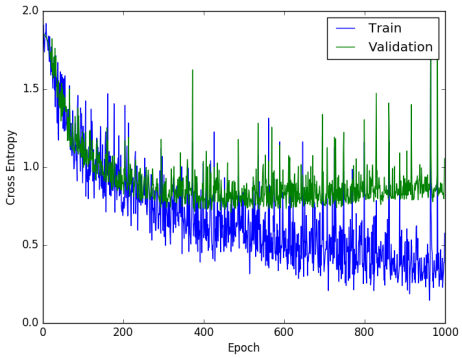
(i) $\epsilon = 1.0$, $m = 0.0$, $b = 100$



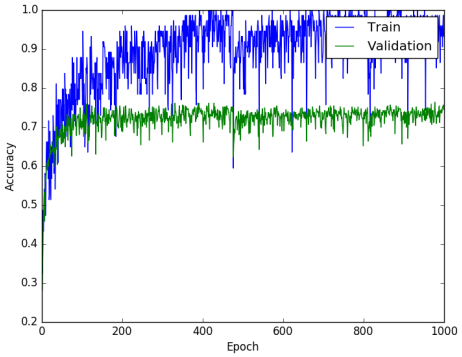
(j) $\epsilon = 1.0$, $m = 0.0$, $b = 100$



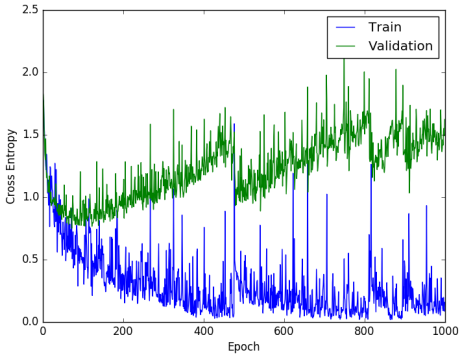
(k) $\epsilon = 0.01$, $m = 0.45$, $b = 100$



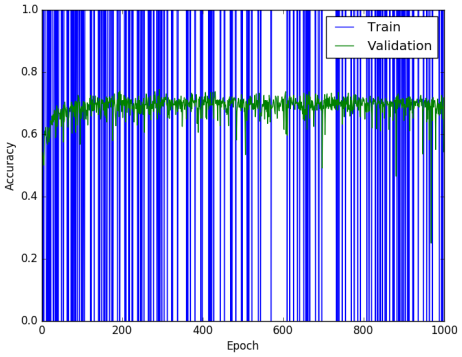
(l) $\epsilon = 0.01$, $m = 0.45$, $b = 100$



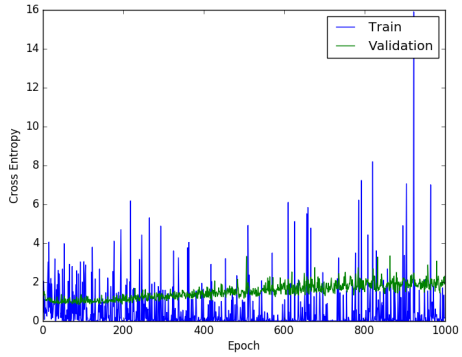
(m) $\epsilon = 0.01$, $m = 0.9$, $b = 100$



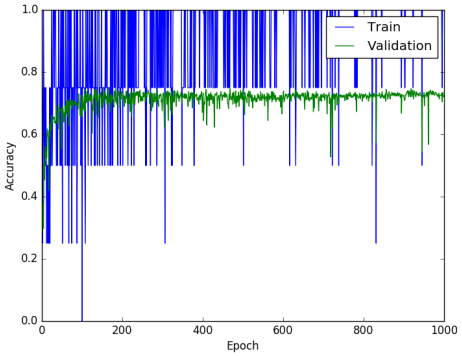
(n) $\epsilon = 0.01$, $m = 0.9$, $b = 100$



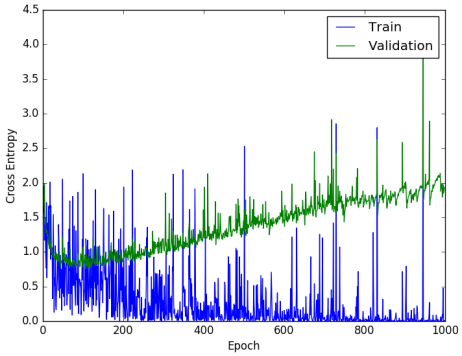
(o) $\epsilon = 0.01$, $m = 0.0$, $b = 1$



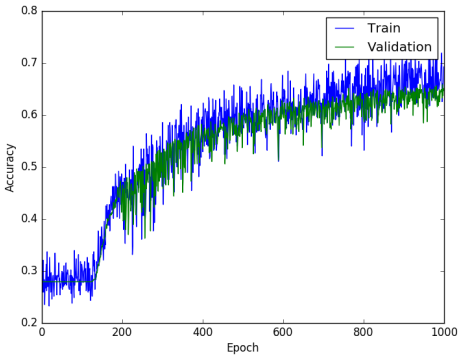
(p) $\epsilon = 0.01$, $m = 0.0$, $b = 1$



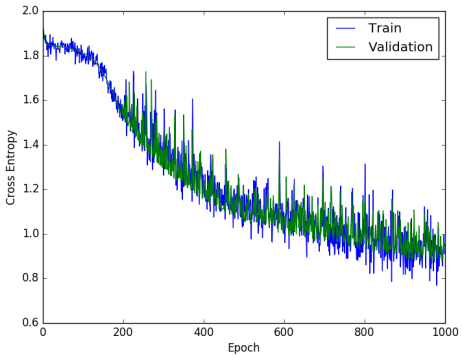
(q) $\epsilon = 0.01$, $m = 0.0$, $b = 10$



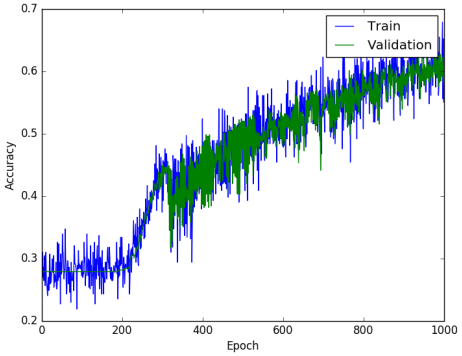
(r) $\epsilon = 0.01$, $m = 0.0$, $b = 10$



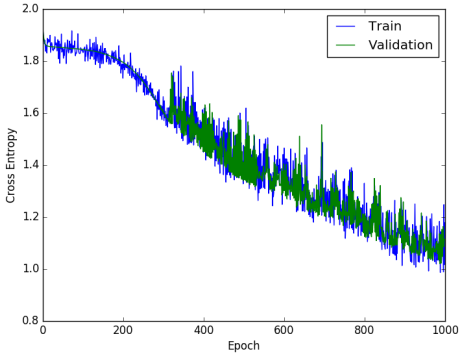
(s) $\epsilon = 0.01$, $m = 0.0$, $b = 500$



(t) $\epsilon = 0.01$, $m = 0.0$, $b = 500$



(u) $\epsilon = 0.01$, $m = 0.0$, $b = 1000$

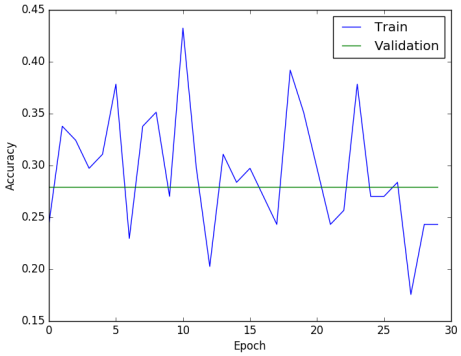


(v) $\epsilon = 0.01$, $m = 0.0$, $b = 1000$

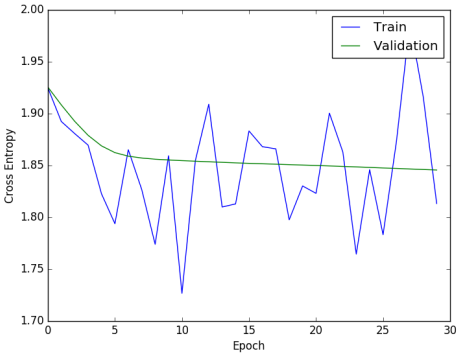
Figure 2: Accuracy and cross entropy in the NN for a given set of hyperparameters

3.3.2 CNN

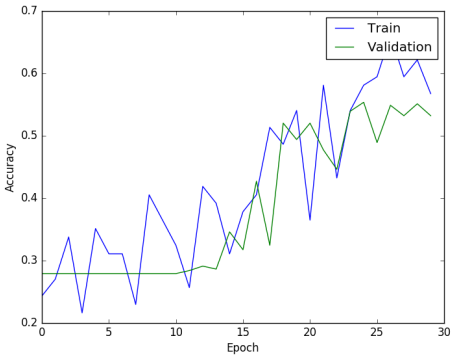
Figure 3 shows the same plots as in the NN case but for the CNN. Here the best set of hyperparameters in terms of validation accuracy is $\epsilon = 0.1$, $m = 0.0$, and $b = 100$. Again $\epsilon = 0.01$, $m = 0.45$, and $b = 100$ has comparable results but by setting $m = 0.0$ we are effectively removing a model parameter.
The CNN had a sweet spot for it's ϵ, experiencing sharp falloffs in validation accuracy for values on either side of it's optimal parameter $\epsilon = 0.1$



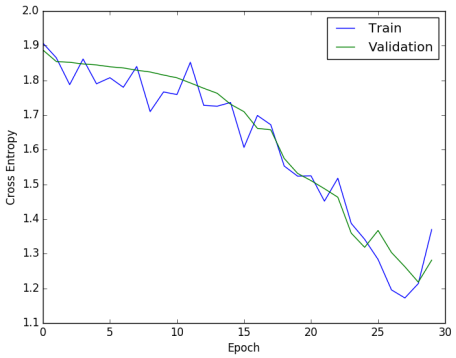
(a) $\epsilon = 0.001$, $m = 0.0$, $b = 100$



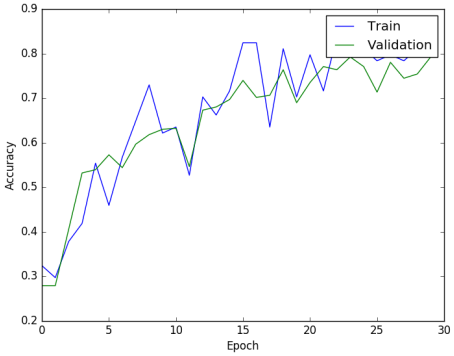
(b) $\epsilon = 0.001$, $m = 0.0$, $b = 100$



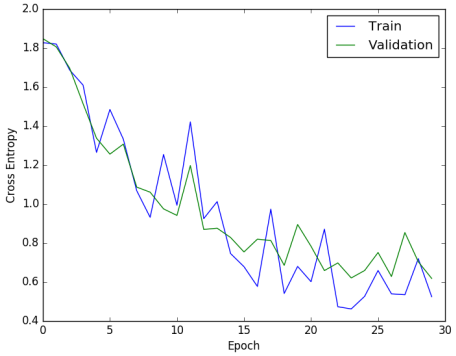
(c) $\epsilon = 0.01$, $m = 0.0$, $b = 100$



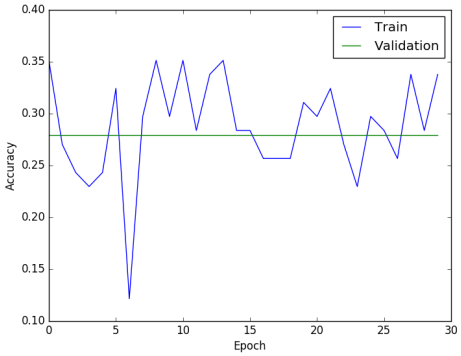
(d) $\epsilon = 0.01$, $m = 0.0$, $b = 100$



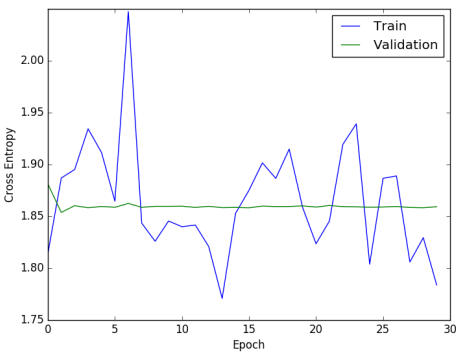
(e) $\epsilon = 0.1$, $m = 0.0$, $b = 100$



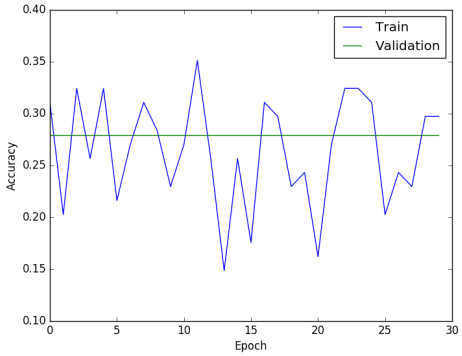
(f) $\epsilon = 0.1$, $m = 0.0$, $b = 100$



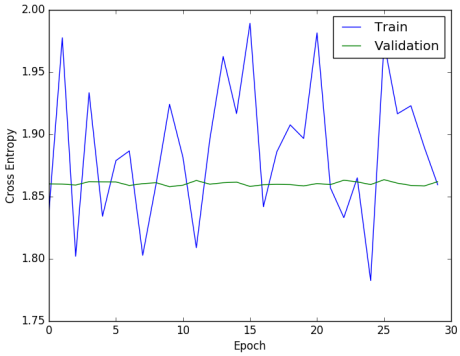
(g) $\epsilon = 0.5$, $m = 0.0$, $b = 100$



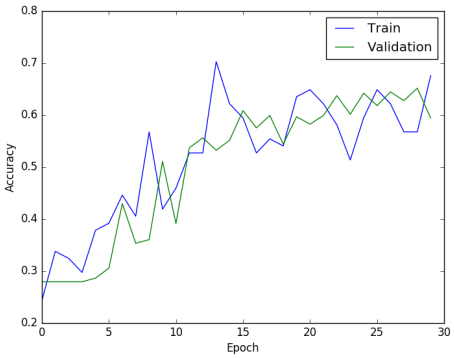
(h) $\epsilon = 0.5$, $m = 0.0$, $b = 100$



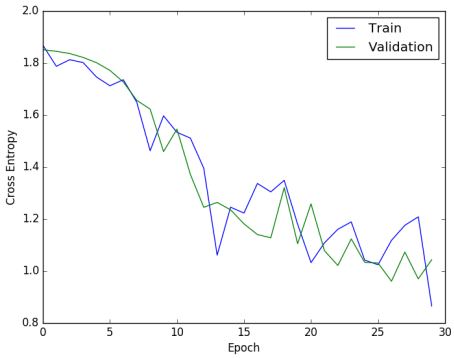
(i) $\epsilon = 1.0$, $m = 0.0$, $b = 100$



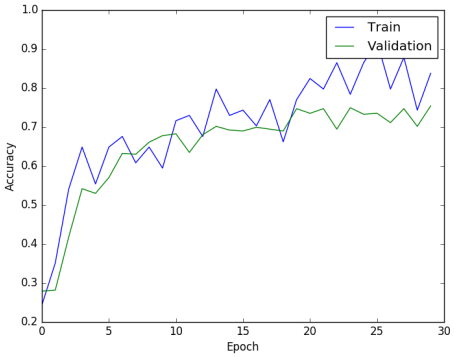
(j) $\epsilon = 1.0$, $m = 0.0$, $b = 100$



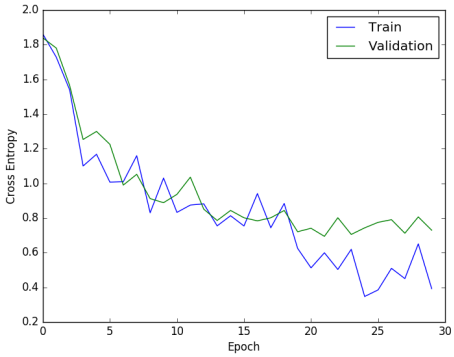
(k) $\epsilon = 0.01$, $m = 0.45$, $b = 100$



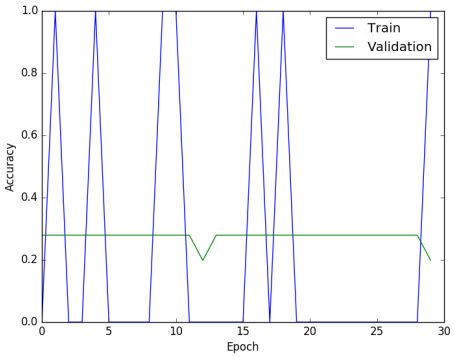
(l) $\epsilon = 0.01$, $m = 0.45$, $b = 100$



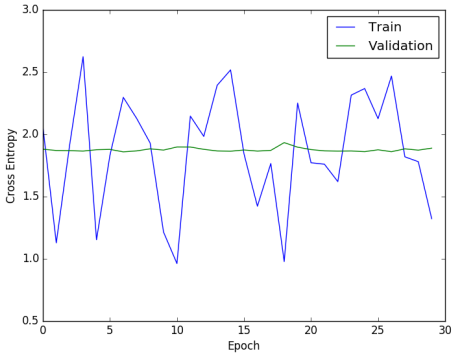
(m) $\epsilon = 0.01$, $m = 0.9$, $b = 100$



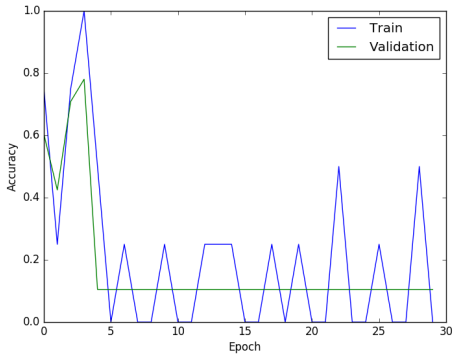
(n) $\epsilon = 0.01$, $m = 0.9$, $b = 100$



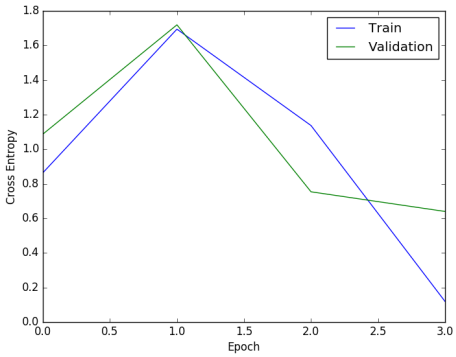
(o) $\epsilon = 0.1$, $m = 0.0$, $b = 1$



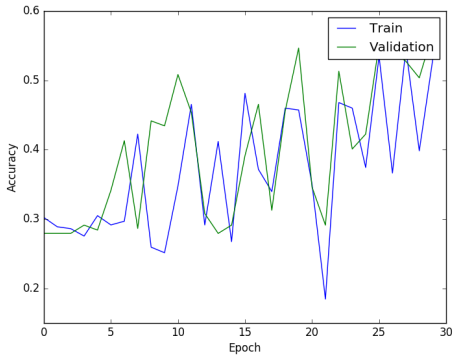
(p) $\epsilon = 0.1$, $m = 0.0$, $b = 1$



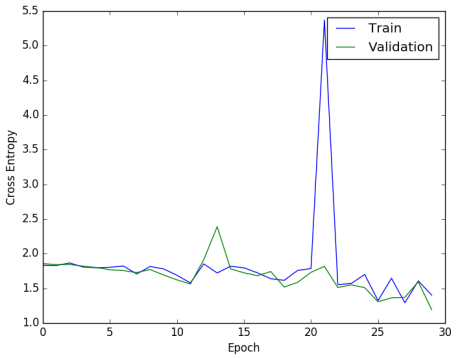
(q) $\epsilon = 0.1$, $m = 0.0$, $b = 10$



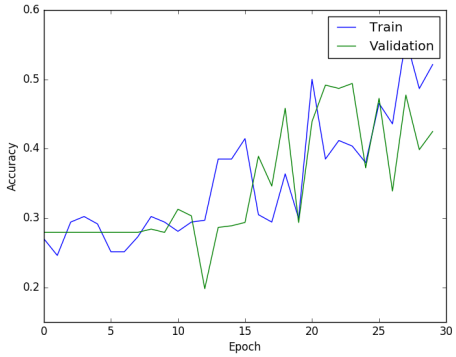
(r) $\epsilon = 0.1$, $m = 0.0$, $b = 10$



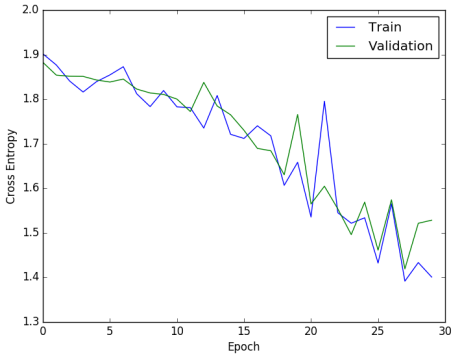
(s) $\epsilon = 0.1$, $m = 0.0$, $b = 500$



(t) $\epsilon = 0.1$, $m = 0.0$, $b = 500$



(u) $\epsilon = 0.1$, $m = 0.0$, $b = 1000$



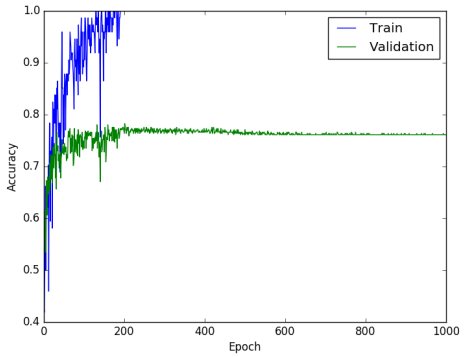
(v) $\epsilon = 0.1$, $m = 0.0$, $b = 1000$

Figure 3: Accuracy and cross entropy in the CNN for a given set of hyper-parameters

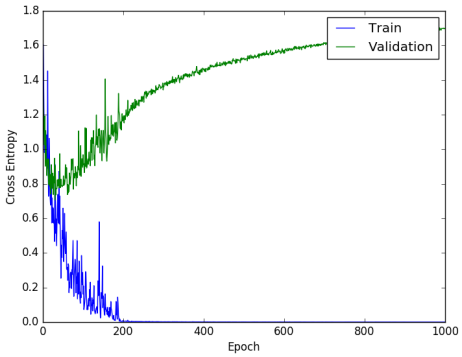
3.4 Model architecture

3.4.1 NN

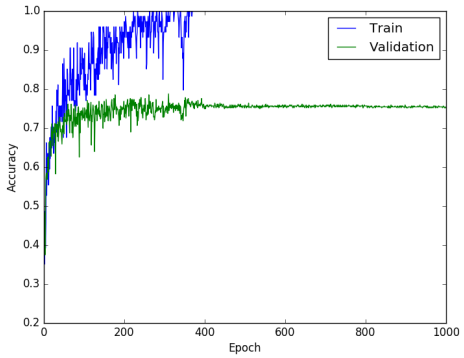
In order to explore the affect of the number of hidden units on accuracy and generalization the number of units in each layer was varied while keeping the hyperparameters set at $\epsilon = 0.01$,$m = 0.9$,$b = 100$. The specific values chosen where $(200, 140)$,$(150, 20)$,$(6, 2)$. The aim in choosing these three pairs was to explore what happens when both layers have a large number of units, just one layer has a large number of units, and neither layer has a large number of units. The results are displayed in figure 4. For cases where at least one hidden layer has many units the NN overfits the training data. This is apparent when looking at the cross entropy charts; the training curve decreases while the validation curve increases. When both layers have few hidden units the danger of overfitting disappears but the validation accuracy is not as high as the cases explored in section 3.3.



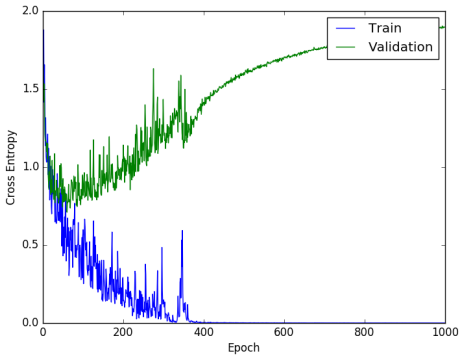
(a) $(200, 140)$



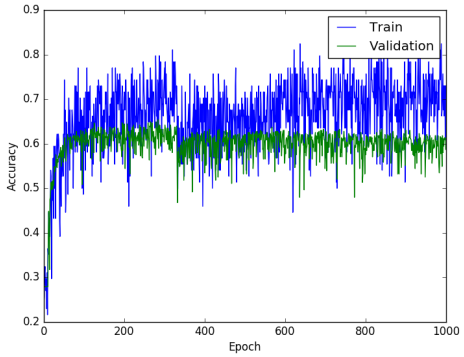
(b) $(200, 140)$



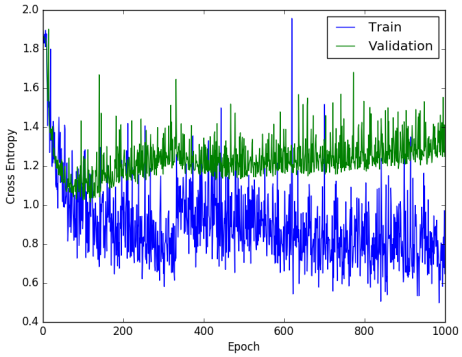
(c) $(150, 20)$



(d) $(150, 20)$



(e) $(6, 2)$

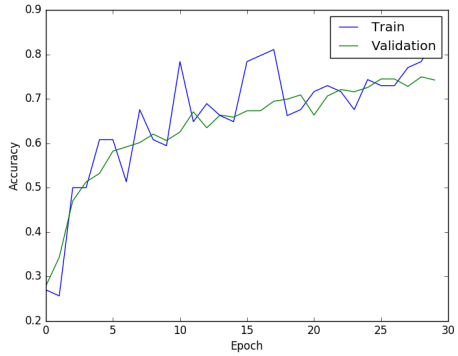


(f) $(6, 2)$

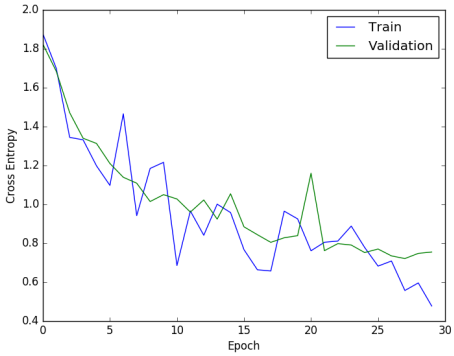
Figure 4: Accuracy and cross entropy in the NN for a given number of units per hidden layer. $\epsilon = 0.01$,$m = 0.9$,$b = 100$

3.4.2 CNN

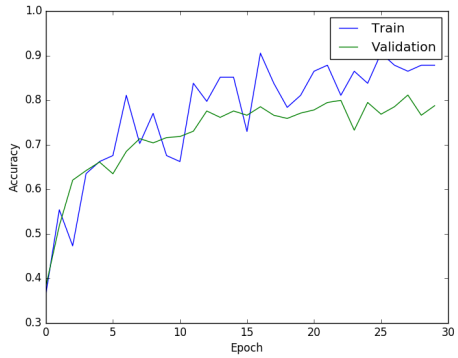
In order to explore the affect of the number of filters on accuracy and generalization the number of filters was varied while keeping the hyperparameters set at $\epsilon = 0.01$,$m = 0.9$,$b = 100$. The result is that there is little difference between many filters and few filters in terms of end results. However the CNNs with more filters reached higher levels of validation set accuracy slightly faster.



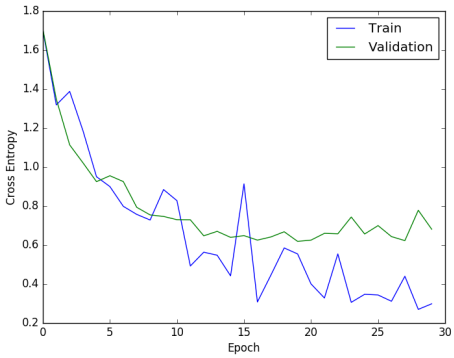
(a) $(50, 3)$



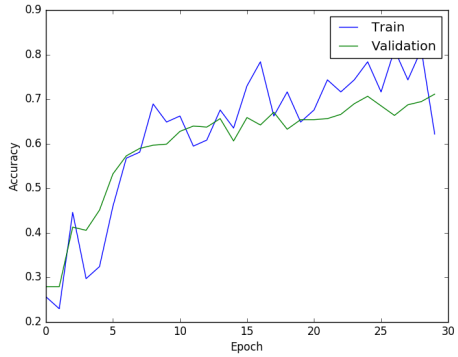
(b) $(50, 3)$



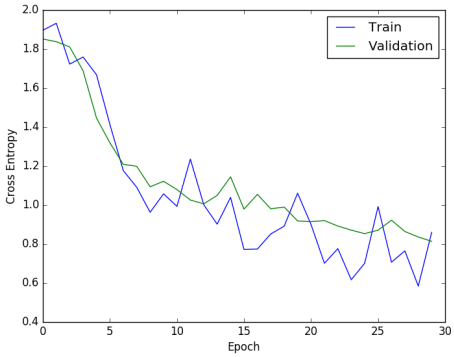
(c) $(35, 30)$



(d) $(35, 30)$



(e) $(5, 2)$



(f) $(5, 2)$

Figure 5: Accuracy and cross entropy in the CNN for a given number of units per hidden layer. $\epsilon = 0.01$,$m = 0.9$,$b = 100$

3.5 Comparison of NN and CNN

3.5.1 Number of parameters in NN

Let u_i be the number of units in layer i of the NN. Let L be the number of layers (including the input layer). Then the number of parameters required to train the NN is $\sum_{i=1}^{L-1} u_i u_{i+1} + \sum_{i=2}^{L} u_i$. The first sum accounts for the weights between layers and the last sum accounts for the bias of each node. The first layer has no bias so the second sum begins at 2. In the case of images with more than one color channel this result would have to be multiplied by the number of channels. In our case the NN has
$(2304 * 16) + (16 * 32) + (32 * 7) + 16 + 32 + 7 = 37655$ parameters.

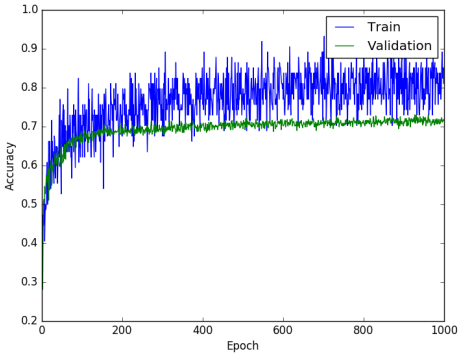
3.5.2 Number of parameters in CNN

Let f_i^n be the i^{th} filter of the n^{th} convolution layer. It will have $c_i^n = I_i^n * J_i^n * D_i^n + 1$ parameters where I_i^n,J_i^n,D_i^n are height, width, and depth of the filter respectively. The +1 comes from the bias term for the filter. Assuming all filters share the same spatial extent, as is the case for this assignment, allows us to drop the subscript on the dimensions. So in our case $c_i^n = c^n = I^n * J^n * D^n + 1$. Let K^n be the number of filters in the n^{th} convolution layer. Then $D^n = K^{n-1}$, where K^0 is defined as the number of color channels of the input image. Calling the total number of convolution layers L the total number parameter count is then $\sum_{n=1}^{L} \sum_{i=1}^{K^n} c_i^n = \sum_{n=1}^{L} K^n c^n$. In our case the CNN has
$8 * (5 * 5 * 1 + 1) + 16 * (5 * 5 * 8 + 1) + 7 * (64 * 16 + 1) = 10,599$ parameters.

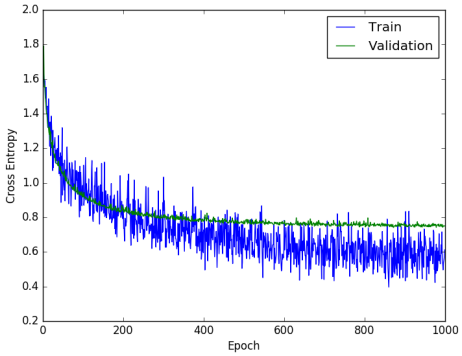
3.5.3 NN and CNN for a fixed number of parameters

In this section we will fix the number of parameters to $16,128$ for the NN and $16,137$ for the CNN. This corresponds to a fully connected NN with no hidden layers and a CNN with 2 filter layers with the first containing 21 filters and the second containing 16 filters all of size 5×5. The aim will be to compare the two methods to see which performs better given the same "model complexity". For the sake of simplicity we used the optimal hyperparemeter values selected from the original networks. Namely

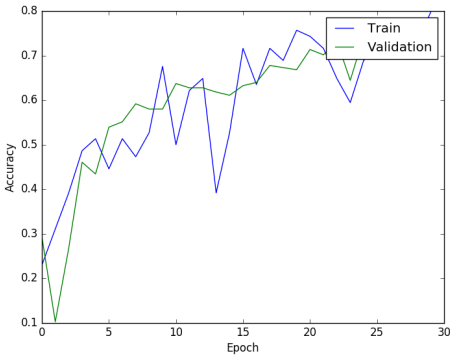
$\epsilon = 0.01$,$m = 0.0$,$b = 100$ for the NN and $\epsilon = 0.1$,$m = 0.0$,$b = 100$ for the CNN.
Figure 6 shows the results for both the NN and the CNN. From looking at these plots it's apparent that both perform approximately the same. This does not come as a surprise as CNNs were developed to address the problem of exponential blow up of the number of required parameters in fully connected NNs. We should recover the results of a NN if the parameter count of the CNN is inflated to the levels required for a fully connected NN. The power of a CNN is in achieving similar levels of accuracy as a given NN with a far fewer number of parameters.
Figure 7 shows a comparison of the weights learned for both the NN and the CNN. Since the NN is fully connected one can still recognize the fact that the dataset is faces. However since the CNN is only connected to regions of the faces each 5×5 block of learned weights is not discernible as a face. As a note we added an extra block of zeros to then end of the NN representation to even out the image, it has no relevance beyond that.



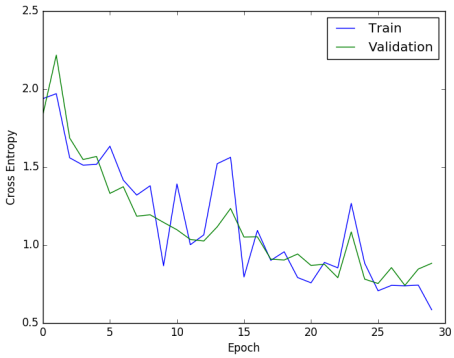
(a) NN accuracy



(b) NN cross entropy



(c) CNN accuracy



(d) CNN cross entropy

Figure 6: Comparison of NN and CNN for similar number of parameters

(a) NN weights layer 1 (b) CNN weights layer 1

Figure 7: Comparison of first layer weights in NN and CNN

3.6 Network uncertainty

Figure 8 shows all inputs in the test set where the highest class probability was $\leq 30\%$ using the NN which performed the best from section 3.3.
The face in the top left is stoic so it makes sense that the NN classifying the emotion. The face in the bottom left is very bright in comparison to the other images, which might cause the NN issue since it's being trained on pixel intensity.
The NN will not always be correct if it outputs the class with the highest probability. That selection criteria only minimizes the expected loss.

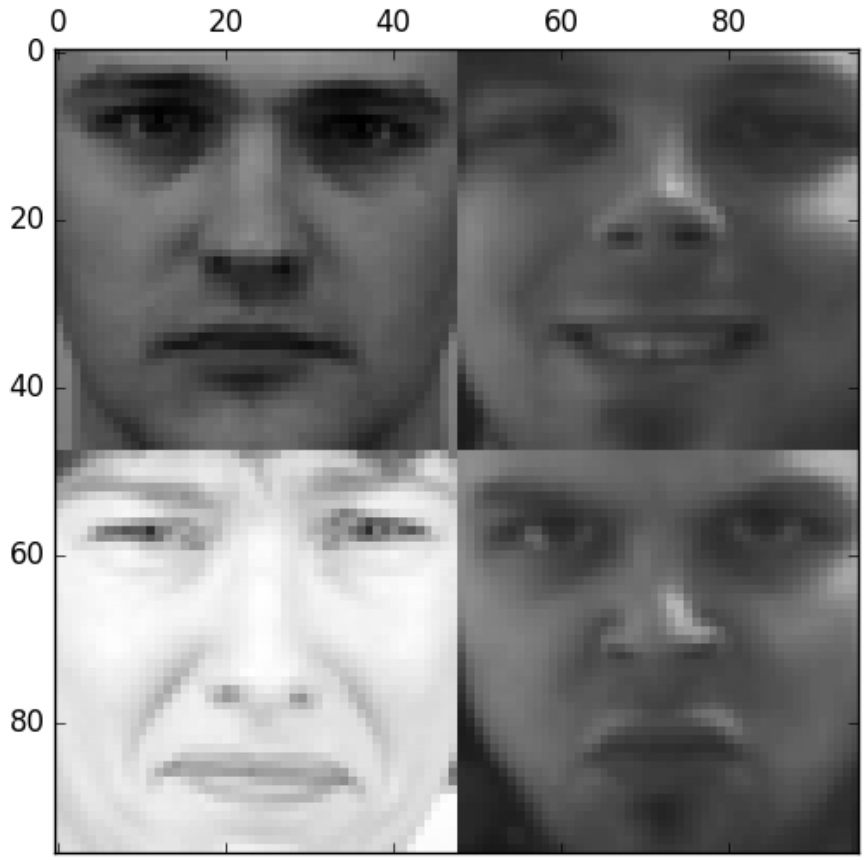


Figure 8: Inputs where highest class probability was less than 30%

4 Mixtures of Gaussians

4.1 Training

Figure 9 shows the learning curve for using 7 clusters, a minimum variance of 0.01, and a randConst of 100. Choosing a high randConst corresponds to total an initial $\vec{\pi}$ where all components are approximately equal. The

resulting μ and Σ values for each Gaussian is shown in figure 10. Finally the mixture rate at 20 iterations is shows in figure 11

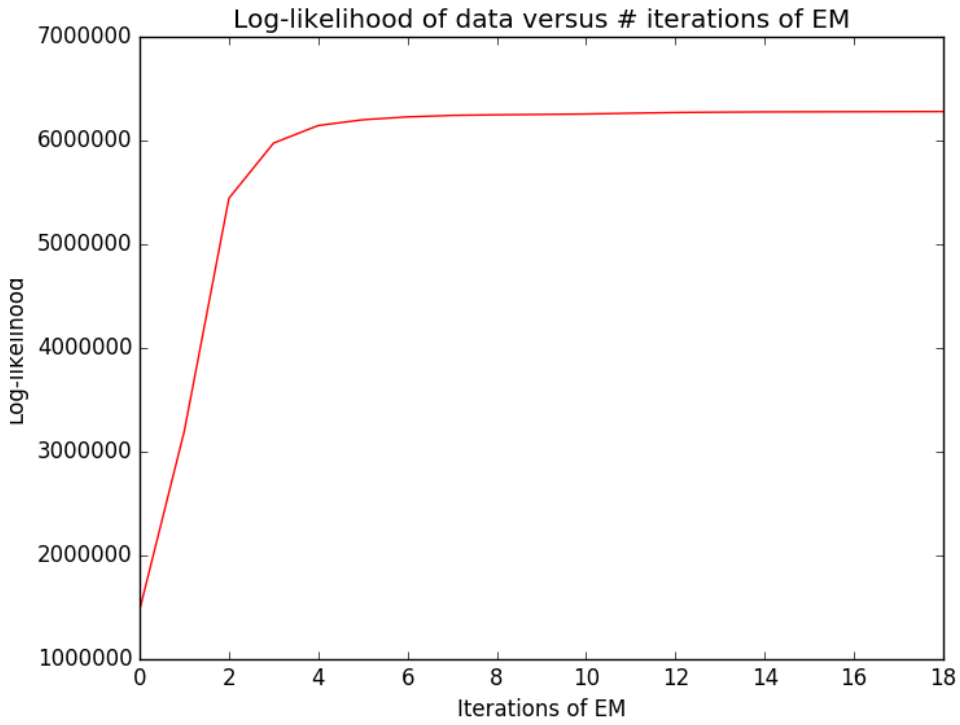


Figure 9: Log likelihood for a 7 cluster MoG

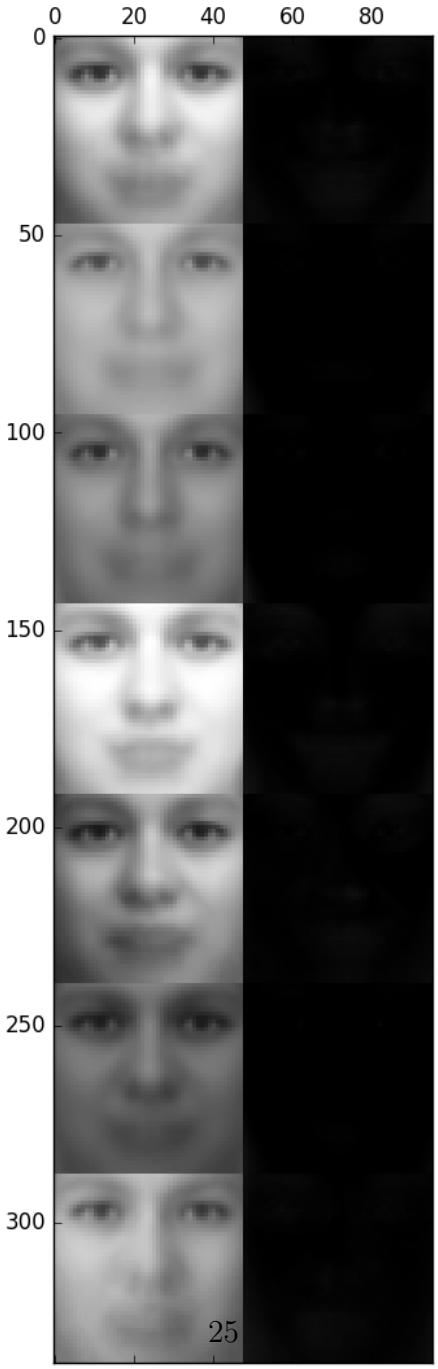


Figure 10: μ and Σ for a 7 cluster MoG

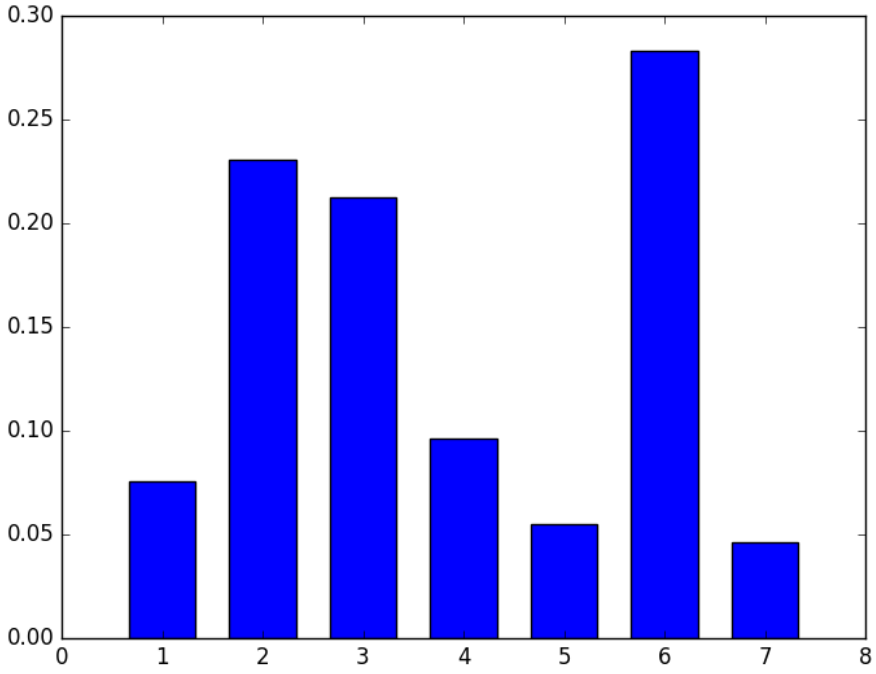


Figure 11: $\vec{\pi}$ for a 7 cluster MoG

4.2 Initializing a Mixture of Gaussians with k-means

Here instead of randomly initializing the parameters as in section 4.1 we use 5 iterations of K-means. The same 3 charts as in section 4.1 are shown (Figures 12,13,14) as a means to compare between both techniques. The most obvious difference is that the learning curve for MoG initialized with k-means converges faster than the learning curve for random initialization.

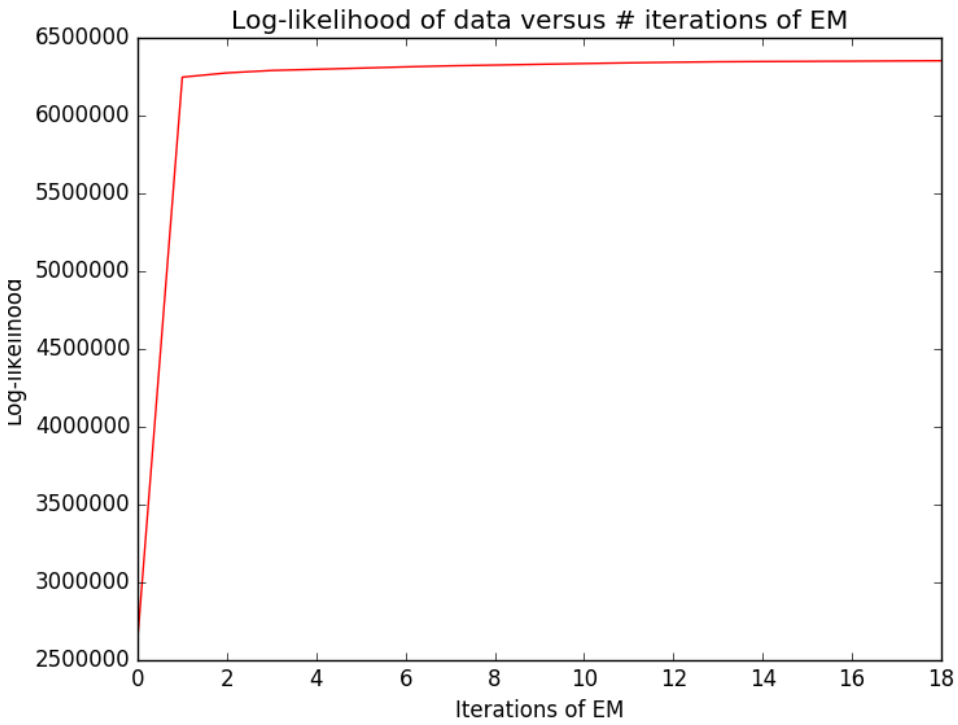


Figure 12: Log likelihood for a 7 cluster MoG initialized with k-means

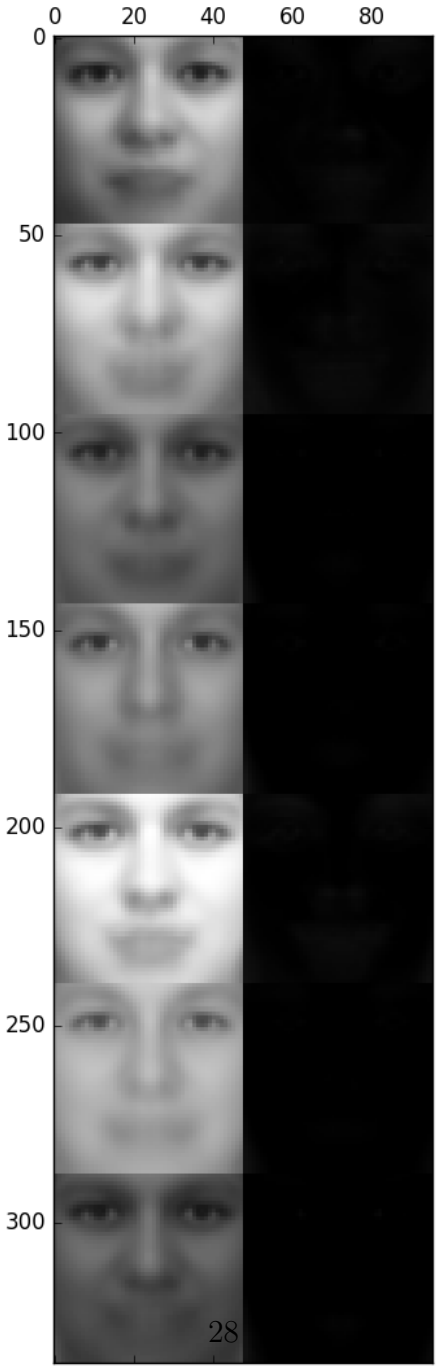


Figure 13: μ and Σ for a 7 cluster MoG initialized with k-means

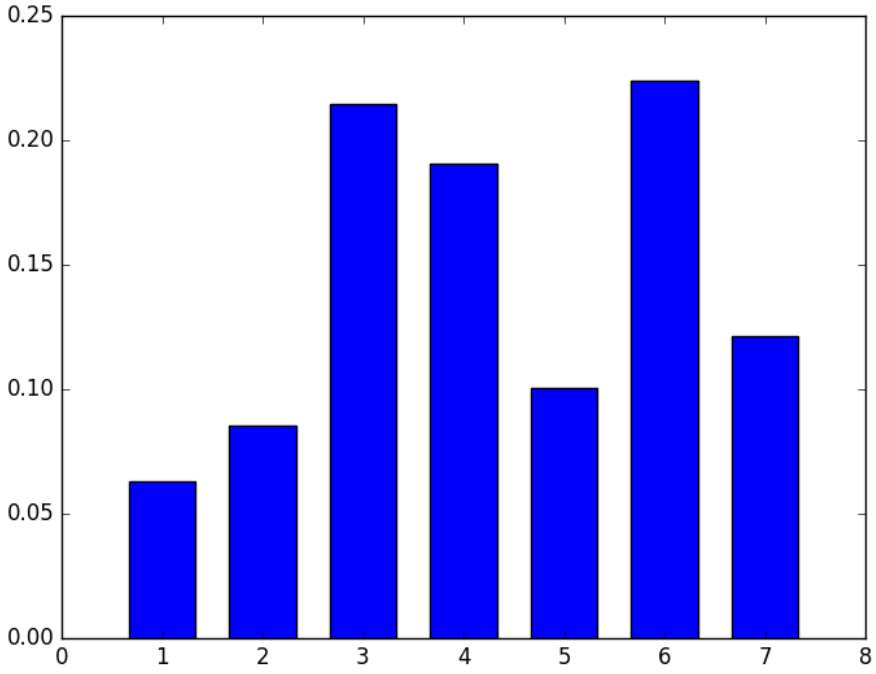


Figure 14: $\vec{\pi}$ for a 7 cluster MoG initialized with k-means

4.3 Classification using MoGs

In this section we classify validation and test data using MoGs. Each class is represented by a MoG with the same number of clusters for each class. The total number of clusters is varied and the resulting error plot can be used to access how well the model performed. Figure 15 has the results of 4 runs. The hyperparameters were kept the same as in section 4.1
The training set error rate in all 4 charts generally decreases. As we increase the number of clusters we are increasing the number of parameters. The more parameters we have the easier it is to fit the training data;the limit being 1 parameter per training point resulting in a perfect fit. So in general the more parameters we add the lower the training error will be, however this increases the chance of overfitting the data.
The test set error rate in all 4 charts seem generally more noisy. Having a minimum around 21 clusters before rising again. This would imply that the

model begins to overfit the training data after that point.

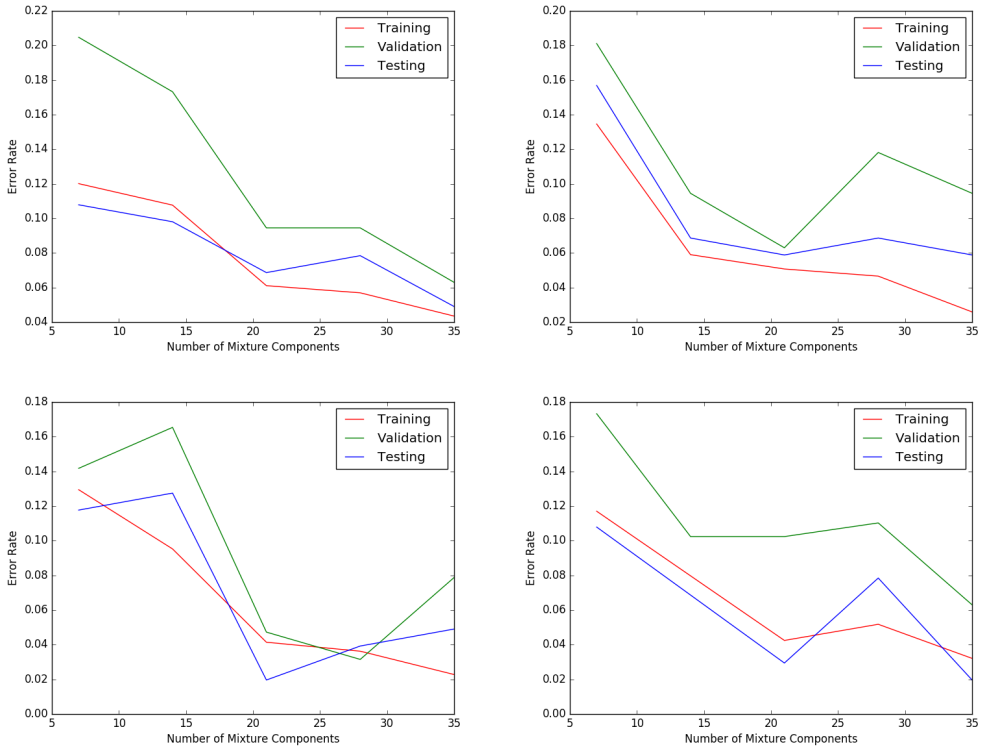


Figure 15: Error rates for classification using MoG