Belief Networks

Using a Belief Network to assign certainties for Character Recognition Using MNIST Dataset

Creating Experts:

To start this research, 5 'expert systems' are created. These are 5 Convolutional Neural Networks, each with different architectures, and different training regiments. The result is 5 models/experts that can identify handwritten characters. The details of training these networks is not included in this document.

Expert Predictions/Basic Assignment:

Figure 1: An example of an image that the models/experts haven't been trained on:

When an expert sees an image that it hasn't been trained on, it makes a prediction of what that character is being represented in that image. Its output looks like this:

Class 0: 0.05
Class 1: 0.03
Class 2: 0.07
Class 3: 0.02
Class 4: 0.08
Class 5: 0.33
Class 6: 0.12
Class 7: 0.05
Class 8: 0.18
Class 9: 0.07

This output is considered as $V^* = V^*(\Phi, L) = P[\text{photo } \Phi \text{ possesses Label L}]$

This V^* is considered the self-assessment that that model's certitude. Note that the sum of the probabilities of the self-assessments add up to 1. This is because the output of each function is placed through a SoftMax function. The SoftMax function also guarantees that $0 <= V^* <= 1$.

Expert's Quality

To rate the quality of an expert, the expert is shown an image with it hasn't been trained on. The expert will make a prediction of what that image is. Since the experts return probabilities for each class, the class with the largest probability is used as that's expert best guess.
Doing so allows us to test the expert like a Bernoulli Trial (see remark 2 of section 1.4) If the best guess is equal to the ground truth, it is scored as P. If the best guess is different from the ground truth, it is scored as F.
Each expert/model is shown M=10,000 images it hasn't been shown before. The number of passes divided by the total number or trials gives us a value for p̃.

```
1. * model1 = 8272/10000
2. * model2 = 9832/10000
3. * model3 = 7772/10000
4. * model4 = 9735/10000
5. * model5 = 9349/10000
```

Modelled as a Bernoulli trial, the std dev for each expert can be found by calculating $(\tilde{p}\ (1-\tilde{p}))^{0.5}$

Shown Below:

	Number of Trials	Number of Passes	Accuracy (p~)	Standard Deviation
Expert1	10,000	8272	0.8272	0.003780743
Expert2	10,000	9832	0.9832	0.001285214
Expert3	10,000	7772	0.7772	0.004161252
Expert4	10,000	9735	0.9735	0.001606168
Expert5	10,000	9349	0.9349	0.002467022

Single Photograph, 1 expert at work, Creating Certainty functions

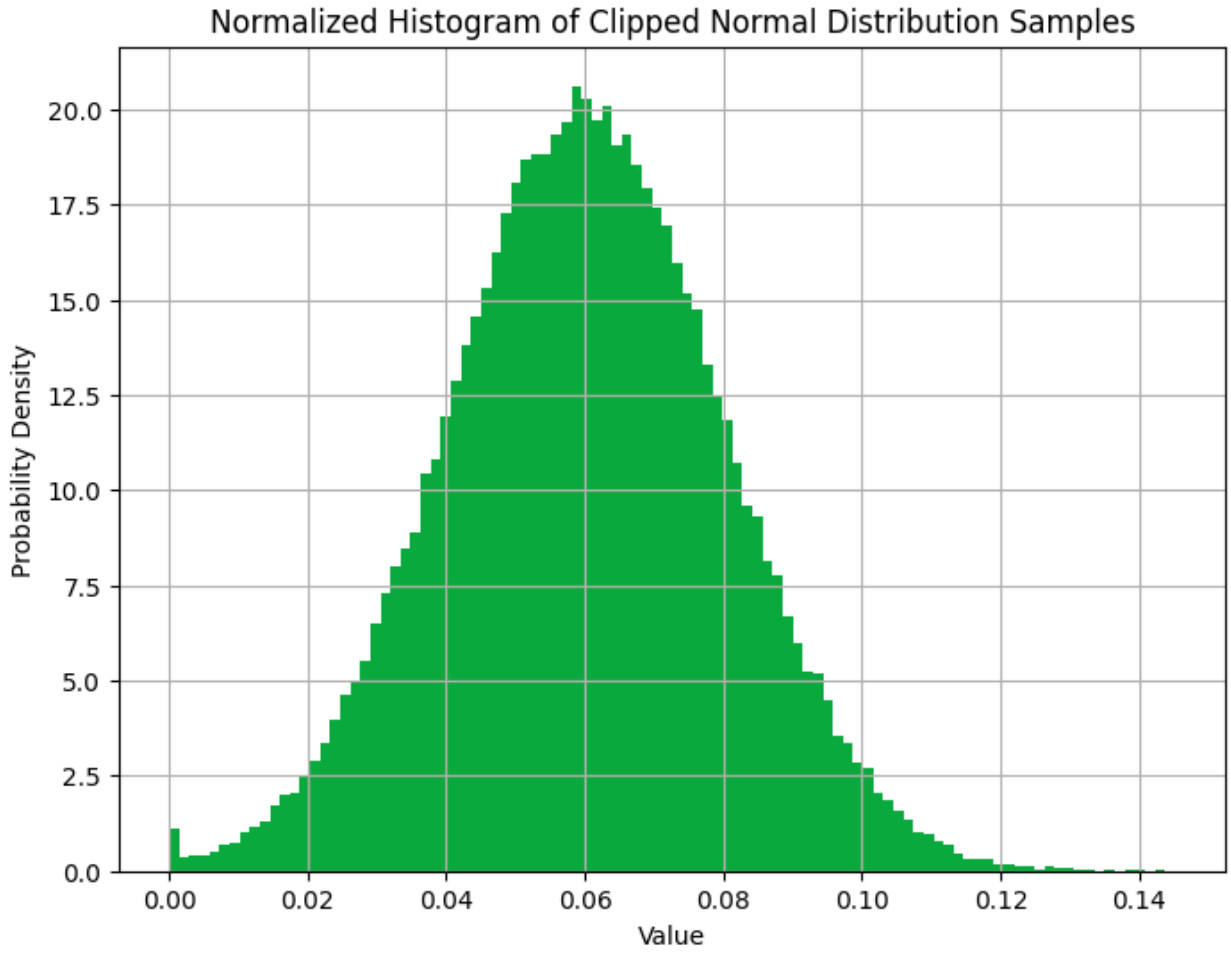
Using the Central Limit theorem, we can model the certainty functions as a normal distribution with a mean value of $\frac{\tilde{p}}{M}$ and a standard deviation of $\left(\frac{\tilde{p}(1-\tilde{p})}{M}\right)^{0.5}$.
This normal is centered at p̃V* for each label, with the standard deviation of $\left(\frac{\tilde{p}(1-\tilde{p})}{M}\right)^{0.5}$.

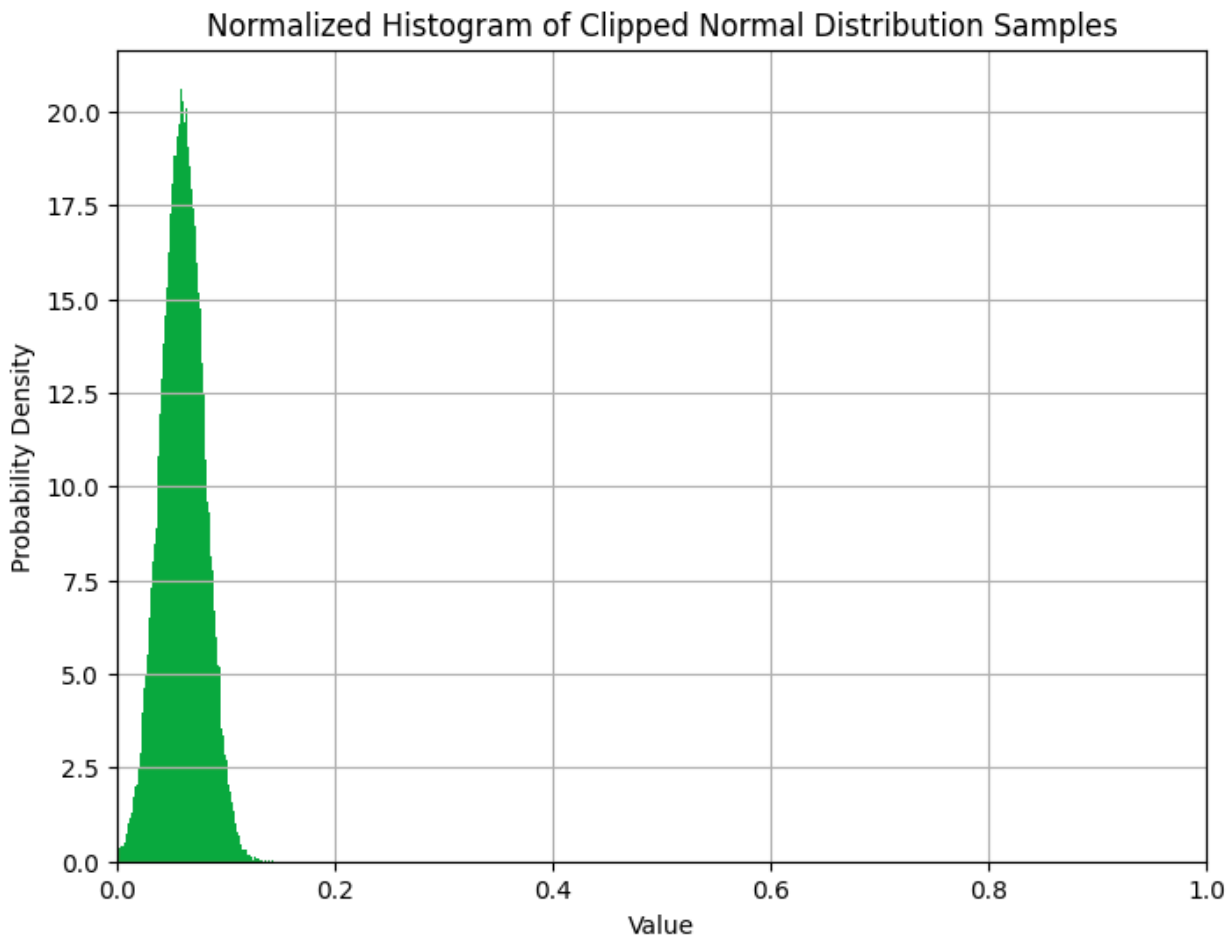
For Example, using Expert 1, we have the following Certitude Functions for image 1 presented above.

	V*	p~ V*	std dev	p~ V* - 1.96stddev	p~ V* + 1.96stddev
Class0	0.05	0.0414	0.003781	0.033950	0.048770
Class1	0.03	0.0248	0.003781	0.017406	0.032226
Class2	0.07	0.0579	0.003781	0.050494	0.065314
Class3	0.02	0.0165	0.003781	0.009134	0.023954

Class4	0.08	0.0662	0.003781	0.058766	0.073586
Class5	0.33	0.2730	0.003781	0.265566	0.280386
Class6	0.12	0.0993	0.003781	0.091854	0.106674
Class7	0.05	0.0414	0.003781	0.033950	0.048770
Class8	0.18	0.1489	0.003781	0.141486	0.156306
Class9	0.07	0.0579	0.003781	0.050494	0.065314

Showing where the +/- 1.96 confidence intervals here is simply for illustrative purposes. By extension, the probability of sampling from any distribution from any class with normal (p~V, stddev) and obtaining a value outside the range [0,1] is exceedingly slim. In these instances, if the number is less than 0, it would clipped and assigned a value of 0. Likewise, if the number is greater than 1, it is clipped and a value of 1 is returned. This ensures that the conditions of a probability distribution are upheld.
For example, drawing 100,000 samples from a normal distribution centered at 0.06 with standard deviation of 0.02, and clipping the results presents the following histogram.





Single Photograph, Multiple Experts at work

If multiple experts look at the same photograph, each has their own certitude functions for each possible label. These certitude functions can be combined by defining a new varliable such as μ

$$Z = \frac{X + Y}{2}$$

$$\mu z = \frac{\mu X + \mu Y}{2}$$

$$\sigma z = \frac{(\sigma x)^2 + (\sigma y)^2}{4}$$

	Number of Trials	Number of Passes	Accuracy (p~)	Standard Deviation
Expert1	10,000	8272	0.8272	0.003780743
Expert2	10,000	9832	0.9832	0.001285214

This can be extended to 3 experts, or more as:

$$\mu z = \frac{\mu 1 + \mu 2 + \cdots + \mu N}{N}$$

$$\sigma z = \frac{(\sigma 1)^2 + (\sigma 2)^2 + \cdots + (\sigma N)^2}{N^2}$$

	Expert 1			Expert 2			Combined	
	V*	p~ V*	std dev	V*	p~ V*	std dev	p~ V*	std dev
Class0	0.05	0.04136	0.00378	0.04	0.03933	0.00129	0.0403	0.0019966
Class1	0.03	0.02482	0.00378	0.10	0.09832	0.00129	0.0616	0.0019966
Class2	0.07	0.05790	0.00378	0.03	0.02950	0.00129	0.0437	0.0019966
Class3	0.02	0.01654	0.00378	0.05	0.04916	0.00129	0.0329	0.0019966
Class4	0.08	0.06618	0.00378	0.12	0.11798	0.00129	0.0921	0.0019966
Class5	0.33	0.27298	0.00378	0.48	0.47194	0.00129	0.3725	0.0019966
Class6	0.12	0.09926	0.00378	0.03	0.02950	0.00129	0.0644	0.0019966
Class7	0.05	0.04136	0.00378	0.02	0.01966	0.00129	0.0305	0.0019966
Class8	0.18	0.14890	0.00378	0.08	0.07866	0.00129	0.1138	0.0019966
Class9	0.07	0.05790	0.00378	0.05	0.04916	0.00129	0.0535	0.0019966

Remark1. The convolution of 2 Normal Distributions, one with mu1, and var1, the other with mu2 and var 2, results in a normal distribution with mean (mu1+mu2) and variance of (var1 +var 2). This approach is problematic. For example, if the values were mu1 = 0.8, and mu2 = 0.6, then the resulting distribution from the convolution has a a mean of 1.4. This mean would be outside of the range [0,1].

Remark 2: Scribbled in an old textbook, I found this suggestion. To capture the total variance when averaging two normal, use the formulas below. I think it is motivated by covariance, or trying to capture 'the variance of the means'. After lots of research, I can't find any mathematical basis for it.

$$\mu z = \frac{\mu 1 + \mu 2 + \cdots + \mu N}{N}$$

$$(\sigma z)^2 = (\sigma 1)^2 + (\sigma 2)^2 + \cdots + (\sigma N)^2 + var(\mu 1, \mu 2, \mu 3 \ldots \mu N)$$

Multiple photographs, Multiple experts at work

The process for single photograph, multiple experts at work can be done for multiple images.
The set of all certitude functions for all images is what we call the 'Belief Network'.
If the set of images run from image 0 through image I,
And the set of possible lables are from Label0 through LabelL

It contains the following the information:

P[Label0|image0], P[Label2|image0],P [Label3|image0],… P[LabelL|image0]
P[Label0|image1], P[Label2|image1],P [Label3|image1],… P[LabelL|image1]
P[Label0|image2], P[Label2|image2],P [Label3|image2],… P[LabelL|image2]
…
P[Label0|imageI], P[Label2|imageI],P [Label3|imageI],… P[LabelL|image1]

Where each P[label|image] is a certitude function modelled as a normal distribution with a mean and standard deviation calculated as described in the previous sections.

If a total of 10,000 images were used, and each has 1 of 10 possible categories, this creates a 10000x10 matrix.

Remark: When implementing, the Belief network can be implemented as a Matrix, with each element in the matrix as a tuple, with entry 1 of the tuple being the mean, and entry 2 of the tuple being the standard deviation.

A Priori Evaluations

I have an expert/model/algorithm that can look at an image, and determine the new images' similarity to all images used in creating of the belief network matrix. It does this by computing the SSIM score (structural similarity Index Measure) between the new image and each target image. (See https://en.wikipedia.org/wiki/Structural_similarity_index_measure)

The SSIM index is calculated on various windows of an image. The measure between two windows x and y of common size $N \times N$ is:[4]

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

with:

- μ_x the pixel sample mean of x;
- μ_y the pixel sample mean of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel} - 1$);
- $k_1 = 0.01$ and $k_2 = 0.03$ by default.

The result is a vector, Φ , with a size of [10,000 X 1] where each entry has a value between -1 and 1. This vector is passed through a SoftMax function, and the resultant vector has properties of a probability distribution (sum of all entries =1, all entries lie between 0 and 1, entries are independent)

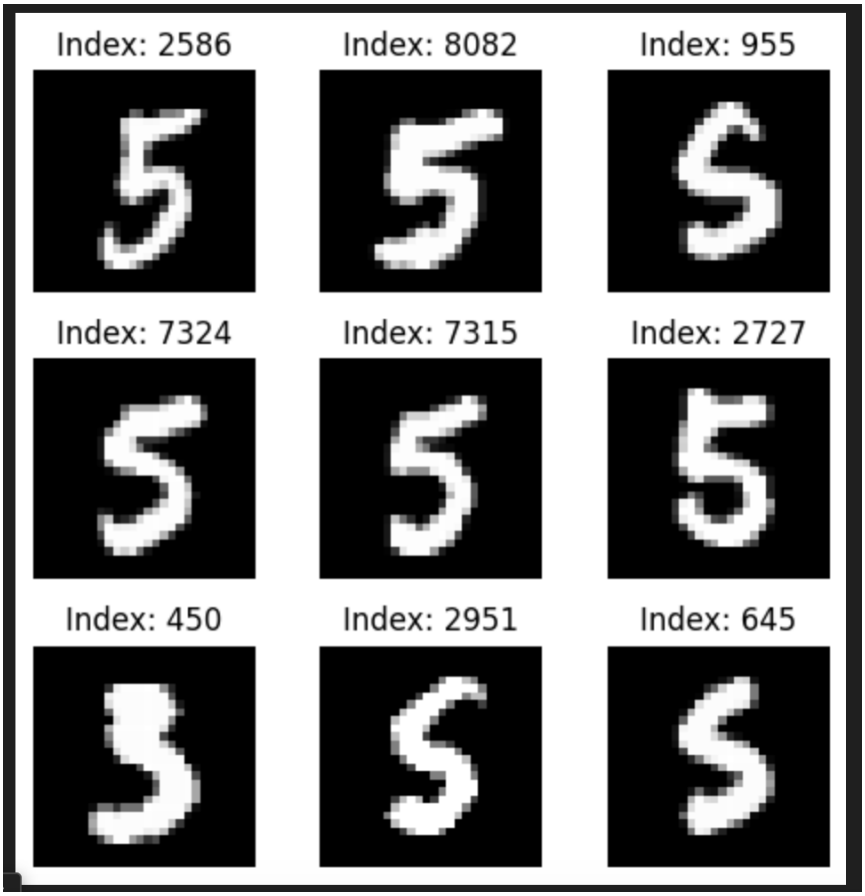


Figure 2: Images used in creating the Belief Network with the highest SSIM score to figure 1

Remark 1: There are other techniques for comparing the similarity of two images, such as Peak Signal to Noise Ratio (PSNR).

Remark 2: There are other methods to make the to make Φ have properties of a probability distribution (sum of all entries =1, all entries lie between 0 and 1, entries are independent). For example, a constant, greater in magnitude than the smallest argument in Φ, can be added to each argument, making them all positive. Then the vector can be normalized.

Remark 3: Another technique could involve choosing the largest n entries in the Φ vector, and setting all other values to 0. Then this new Φ vector can be normalized using Softmax or any other technique. This may have some computational efficiencies later.

With a Φ vector identified, We can solve the equation

L = F Φ

where F is the matrix of the belief network after sampling each certitude function F_{ij}.

Algorithm1

Initialize $\hat{L}$= []
Repeat for several thousand iterations (iter):
  For each certitude function F_{ij} in the Belief Network:
    Sample from the normal distribution network.
    Clip values between 0 and 1 if necessary
  Solve L = F Φ
  Append L to $\hat{L}$

Once algorithm 1 is complete, $\hat{L}$ will be a matrix with 10 rows and iter columns. The values of each row can be plotted in a histogram. This histogram will approximate a normal distribution (proof needed? Or can we just make an assumption?)

For each row in $\hat{L}$, we can find the mean and standard deviation of the (iter) entries. These can be used to represent the P[label|image] with a given confidence interval. For example, it would be the Mean +-1.96 standard deviation for the 95% confidence interval.

For example, using this approach, we obtained the following results for figure 1. It is most likely a 5, and the 95% confidence intervals are included.

```
5

Class 0:  0.006613669499057572 0.010624803717968125
Class 1:  0.0036834363084226737 0.008588765933691272
Class 2:  0.017628046796708017 0.023288696572799476
Class 3:  0.3022275599470232 0.31378980409705526
Class 4:  0.0032582016308709827 0.008611669169728912
Class 5:  0.3173174643802176 0.33163205662405293
Class 6:  0.09068700592338338 0.09826666673732953
Class 7:  0.0015710197258132325 0.00467363888599744
Class 8:  0.161687465783088 0.17060855946618525
Class 9:  0.022551639777054443 0.02943170505013931
```

A Quick Aside on Optimization Problems

A Toy Example

Solve

$$0.8 * x1 + 0.9 * x2 + 0.7 * x3 <= 0.85$$
$$0.2 * x1 + 0.1 * x2 + 0.3 * x3 <= 0.15$$

Subject to

x1,x2,x2<=1

x1,x2,x2=>0

x1+x2+x3<=1

Using a least squares approach, we obtain:

X1 = 0.3333, X2 = 0.58333, X3 = 0.08333

Notice this solution solves the inequality as an equality.

$$0.8 * x1 + 0.9 * x2 + 0.7 * x3 = 0.85$$
$$0.2 * x1 + 0.1 * x2 + 0.3 * x3 = 0.15$$

The least squares approach is algorithmic, so it always obtains the same answer.

However, using a MonteCarlo approach leads to an interesting result. By repeatedly guessing different values of x1,x2,x3 that meet the constraints and checking the one that best minimizes the cost function:

$$||(0.8 * x1 + 0.9 * x2 + 0.7 * x3 - 0.85)^2 + (0.2 * x1 + 0.1 * x2 + 0.3 * x3 - 0.15)^2 ||$$

We obtain multiple solutions to the problem. This can be anticipated as we have 3 free variables, but only 2 equations. Some examples of solutions are:

X1 = 0.1511, X2 = 0.67447, X3 = 0.1744

Or

X1 = 0.41292708, X2 = 0.54353048, X3 = 0.04354243

Or

X1 = 0.41292708, X2 = 0.54353048, X3 = 0.04354243

Remark: This cost function uses the 2 norm. But it can be modified to further penalize values that go over the constraint, but not under, for example:

Cost function = sup norm + np.argmin(0,(0.8 * x1 + 0.9 * x2 + 0.7 * x3 - 0.85))

A second toy example:

Solve

$$0.8 * x1 + 0.9 * x2 + 0.7 * x3 <= 0.15$$
$$0.2 * x1 + 0.1 * x2 + 0.3 * x3 <= 0.85$$

Subject to

$$x1,x2,x2<=1$$
$$x1,x2,x2=>0$$
$$x1+x2+x3<=1$$

Using a sequential least square programming optimizer, we obtain:

X1 = 0.00
X2 = 0.00
X3 = 0.62068

w/ a 2 norm error of = 0.7221853807375879

Using a MonteCarlo Approach, we obtain

X1 = 0.0043
X2 = 0.0037
X3 = 0.6234

w/ a 2 norm error of = 0.7228862753461998

This toy problem has 2 noticeable outcomes. First, the Montecarlo solution comes very close to the actual solution. Running the MC simulation for longer may even improve accuracy further.

Another interesting outcome is that we cannot find values for x1,x2,x3 that solve the equations exactly. Even though we have 3 variables and 2 equations, getting an exact match is not possible.

$$0.8 * x1 + 0.9 * x2 + 0.7 * x3 = 0.15$$
$$0.2 * x1 + 0.1 * x2 + 0.3 * x3 = 0.85$$

The only way to make this match exact is if x1,x2,or x3 violate a condition or two For example [0.333, -2.917, 3.5833] solves these 2 equations exactly.

A Posteriori Problem

An Expert/Model 6 who was not considered as part of the belief network is asked to look at figure 1 and make an educated guess as to what figure 1.
It returns the following guess:

Class0:	0.00
Class1:	0.04
Class2:	0.02
Class3:	0.16
Class4:	0.16
Class5:	0.39
Class6:	0.02
Class7:	0.01
Class8:	0.15
Class9:	0.05

We do not know the accuracy of model 6. Therefore, we would like to use the belief network to add some confidence bounds to model6's guess.

To do this, we need to first solve for a viable vector Φ such

$$L <= F\ \Phi$$

$$\text{Sum}(\Phi)<=0$$
$$0<= \Phi_i <=1$$

Once this vector Φ is determined, we can then use algorithm 1.

To do this, we can use a least squares approach, but this seems to crash my computer. Motivated by the toy problems, a MonteCarlo Approach was used for Algorithm 2.

Algorithm 2 – MonteCarlo Approach

Create an error function such as(L-F Φ)^2 + min(0, F Φ -L) which will be minimized

Initialize $\Phi_{composite}$ as zeroes
Initialize $\Phi_{Best} = 0$
Initialize min_error = inf

Repeat for some large number.
    Sample each certitude function Fij to create F
      Repeat for some large number of iterations:
        Create a random Vector Φ_{rand} that meets the constaints.
        Compute the error using function of Φ_{rand}
        If error < min_error:
          Update $\Phi_{Best} = \Phi_{rand}$
          Update min_error =error
    Update $\Phi_{composite}$ += Φ_{Best}
Normalize Φ_{Best}
Perform Algorithm 1 with Φ_{Best}

Algorithm 1 shold be run several thousand times. Once algorithm 1 is complete, $\hat{L}$ will be a matrix with 10 rows andx columns. The average and standard deviation can be used to add uncertainty to Model6's output

Using this approach, we obtain the following 95% confidence intervals to model6's outputs

Class 0: 0.009808, 0.01743
Class 1: 0.0209, 0.02287
Class 2: 0.01817, 0.02223
Class 3: 0.1461, 0.164
Class 4: 0.1518, 0.1593
Class 5: 0.399, 0.4218
Class 6: 0.01763, 0.02135
Class 7: 0.0137, 0.01787
Class 8: 0.1132, 0.1244
Class 9: 0.02472, 0.0342

For reference, here are the 25 images with the highest values in $\Phi_{Best.}$