67750 | Advanced Practical Course in Machine Learning | Exercise 2

Guy Lutsker 207029448

Part I
Theoretical Part

Question 1

1.1 MLE in the EM algorithm

In class we saw that the expected log likelihood function for the Gaussian mixture model can be written as:

$$\mathbb{E}[\ell(S,\theta)] = \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} log\big(\pi_y \mathcal{N}(x_i;\mu_y,\Sigma_y)\big)$$

Show that the MLE of the multinomial distribution of our mixture weights is indeed:

$$\pi_y = \frac{1}{N}\sum_{i=1}^{N} c_{i,y}$$

Hint: don't forget to the constraint on π_y...

$$Solution:$$

$\forall y \in [i,k]$ We would like to maximize $\mathbb{E}[\ell(S,\theta)]$ with respect to π_y:

$$\mathbb{E}[\ell(S,\theta)] = \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot ln(\pi_y \cdot \mathcal{N}(x_i;\mu_y,\Sigma_y))$$

According to the restrain: $\sum_{y=1}^{k} \pi_y = 1$.

And so, using Lagrange multiplier we get that we want maximize:

$$f = \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot ln(\pi_y \cdot \mathcal{N}(x_i;\mu_y.,\Sigma)) - \lambda((\sum_{y=1}^{k} \pi_y) - 1)$$

Let us derive with respect to λ and find its roots:

$$0 = \frac{\partial f}{\partial \lambda} = \sum_{y=1}^{k} \pi_y - 1$$

$$(a) \rightarrow \sum_{y=1}^{k} \pi_y = 1$$

Lets us also derive with respect to π_y $\forall y \in [k]$ and find its roots:

$$0 = \frac{\partial f}{\partial \pi_y} = \sum_{i=1}^{N} \frac{c_{i,y} \cdot \mathcal{N}(x_i; \mu_y, \Sigma_y)}{\pi_y \mathcal{N}(x_i; \mu_y, \Sigma_y)} - \lambda = \sum_{i=1}^{N} \frac{c_{i,y}}{\pi_y} - \lambda$$

$$(b) \rightarrow \sum_{i=1}^{N} \frac{c_{i,y}}{\lambda} = \pi_y$$

Applying equation (b) onto equation (a) gets us:

$$\sum_{i=1}^{N} \sum_{y=1}^{k} \frac{c_{i,y}}{\lambda} = 1$$

Recall the definition of $c_{i,y} = \mathbb{P}(y|x_i)$and from Law of total probability we get that:

$$\sum_{y=1}^{k} c_{i,y} = \sum_{y=1}^{k} \mathbb{P}(y|x_i) = 1$$

$$\rightarrow \sum_{i=1}^{N} \sum_{y=1}^{k} c_{i,y} = \lambda \rightarrow \sum_{i=1}^{N} 1 = \lambda$$

$$\rightarrow N = \lambda$$

And so $\forall y \in [k]$ we get $\pi_y = \frac{1}{N} \sum_{i=1}^{N} c_{i,y}$ as required.

And so we win :)

Question 2

1.2 MLE in the GSM Model

The GSM model assumes that image patches are samples from a mixture of k Gaussians with the distribution $\mathcal{N}(0, r_y^2 \Sigma)$ for a shared Σ between the Gaussians. Show that the maximization update for r_y^2 is:

$$r_y^2 = \frac{\sum_{i=1}^{n} c_{i,y} x_i^T \Sigma^{-1} x_i}{d \sum_{i=1}^{n} c_{i,y}}$$

Here are some linear algebra identities and the Gaussian distribution to help get you started (A is an $n \times n$ matrix, c is a scalar):

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

$$det(cA) = c^n det(A)$$

$$(cA)^{-1} = c^{-1} A^{-1}$$

Hint: simplify the expression as much as you can before taking the derivative.

$Solution:$

For elegance sake let us set $2\pi = \tau$ (cause tau is cooler than pi).

Firstly let us recall the definition of the normal distribution:

$$(a): \mathcal{N}(x;\mu,\Sigma) = \frac{1}{\sqrt{|\tau\Sigma|}} \cdot exp(-\frac{1}{2}(x-\mu)^\intercal \Sigma^{-1}(x-\mu))$$

Secondly recall that $\forall i\ \mathbb{P}(x_i) = \sum_{y=1}^{k} \pi_y \cdot \mathcal{N}(x_i; 0, r_y^2 \cdot \Sigma)$

And so, our log likelihood is: $\sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot ln(\pi_y \cdot \mathcal{N}(x_i; 0, r_y^2 \cdot \Sigma))$

Applying equation (a) gets us: $\sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot ln(\pi_y \cdot \frac{1}{\sqrt{|\tau r_y^2 \Sigma|}} \cdot exp(-\frac{1}{2}(x_i-0)^\intercal (r_y^2\cdot\Sigma)^{-1}(x_i-0)))$

$$\overset{\text{Tending to the exp expression in numerator}}{=} \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot ln(\pi_y \cdot \frac{exp(-\frac{x_i^\intercal \Sigma^{-1} x_i}{2r_y^2})}{\sqrt{|\tau r_y^2 \Sigma|}})$$

$$\overset{\text{Tending to the square root}}{=} \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot ln(\pi_y \cdot \frac{exp(-\frac{x_i^\intercal \Sigma^{-1} x_i}{2r_y^2})}{\sqrt{(\tau)^d \cdot r_y^{2d} \cdot |\Sigma|}})$$

$$\overset{\text{Tending to the } ln}{=} \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot (-\frac{x_i^\intercal \Sigma^{-1} x_i}{2r_y^2} + ln(\pi_y) - ln(((\tau)^d \cdot r_y^{2d} \cdot |\Sigma|)^{-\frac{1}{2}}))$$

$$\overset{\text{Tending to the } ln(\surd)}{=} \sum_{i=1}^{N}\sum_{y=1}^{k} c_{i,y} \cdot (-\frac{x_i^\intercal \Sigma^{-1} x_i}{2r_y^2} + ln(\pi_y) - \frac{1}{2}(d\cdot ln(\tau) + d\cdot ln(r_y^2) + ln(|\Sigma|))$$

Now, let is derive this monster(lets call it f) with respect to r_y^2:

$$\frac{\partial f}{\partial r_y^2} = \sum_{i=1}^{N}(\frac{c_{i,y}\cdot x_i^\intercal \Sigma^{-1}\cdot x_i}{2\cdot r_y^4}) - \frac{c_{i,y}\cdot d}{2r_y^2} = \frac{1}{2\cdot r_y^4}\cdot\sum_{i=1}^{N}(c_{i,y}\cdot x_i^\intercal\Sigma^{-1}\cdot x_i) - \frac{d}{2\cdot r_y^2}\sum_{i=1}^{N} c_{i,y}$$

Now, lets find its roots $\forall y \in [k]$:

$$\frac{1}{2\cdot r_y^4}\cdot\sum_{i=1}^{N}(c_{i,y}\cdot x_i^\intercal\Sigma^{-1}\cdot x_i) - \frac{d}{2\cdot r_y^2}\sum_{i=1}^{N} c_{i,y} = 0$$

$$\rightarrow \frac{1}{r_y^2}\cdot\sum_{i=1}^{N}(c_{i,y}\cdot x_i^\intercal\Sigma^{-1}\cdot x_i) = d\sum_{i=1}^{N} c_{i,y}$$

$$\rightarrow r_y^2 = \frac{\sum_{i=1}^{N}(c_{i,y}\cdot x_i^\intercal\Sigma^{-1}\cdot x_i)}{d\cdot\sum_{i=1}^{N} c_{i,y}}$$

As required.

And so we win :)

Question 3

1.3 EM Initialization

Yossi didn't listen in class and decided to initialize all of the Gaussians in his GMM to have the same parameters:

$$\forall y:\ \pi_y = \frac{1}{k},\ \mu_y = \mu,\ \Sigma_y = \Sigma$$

Write down the first two iterations of the EM algorithm for this initialization. What's the problem?

$$Solution:$$

Let us initialize the Gaussians to have the same parameters s.t

$$\forall y\ \pi_y = \frac{1}{k}, \mu_y = \mu.\Sigma_y = \Sigma$$

Notice that $\forall y, \mathcal{N}(x_i;\mu_y,\Sigma_y) = q_i$, where $i \in [N], q \in \mathbb{R}^N$ is some constant vector.
And lets look at the first two iterations of the EM algorithm:
Iteration number 1:

1.
$$c_{i,y} = \frac{\pi_y \mathcal{N}(x_i;\mu_y,\Sigma_y)}{\sum_{l=1}^{k}\pi_l \mathcal{N}(x_i;\mu_l,\Sigma_l)} \overset{\mathcal{N}(x_i;\mu_y,\Sigma_y)=q_i}{=} \frac{\pi_y q_i}{\sum_{l=1}^{k}\pi_l q_i} = \frac{q_i}{k\cdot q_i} = \frac{1}{k}$$

2.
$$\pi_y = \frac{1}{N}\sum_{i=1}^{N} c_{i,y} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{k} = \frac{1}{k}$$

3.
$$\mu_y = \frac{\sum_{i=1}^{N} c_{i,y}\cdot x_i}{\sum_{i=1}^{N} c_{i,y}} = \frac{\frac{1}{k}\cdot\sum_{i=1}^{N} x_i}{\frac{N}{k}} = \frac{1}{N}\cdot\sum_{i=1}^{N} x_i \overset{\text{Surprisingly so}}{=} \mu_y \overset{\text{Def.}}{=} \mu$$

4.
$$\Sigma_y = \frac{\sum_{i=1}^{N} c_{i,y}(x_i-\mu_y)(x_i-\mu_y)^\intercal}{\sum_{i=1}^{N} c_{i,y}} = \frac{\frac{1}{k}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^\intercal}{\frac{N}{k}} = \frac{\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^\intercal}{N} = \Sigma_y$$

And iteration number 2 :

1.
$$c_{i,y} = \frac{\pi_y \mathcal{N}(x_i;\mu_y,\Sigma_y)}{\sum_{l=1}^{k}\pi_l \mathcal{N}(x_i;\mu_l,\Sigma_l)} \overset{\mathcal{N}(x_i;\mu_y,\Sigma_y)=q_i}{=} \frac{\pi_y q_i}{\sum_{l=1}^{k}\pi_l q_i} = \frac{q_i}{k\cdot q_i} = \frac{1}{k}$$

2.
$$\pi_y = \frac{1}{N}\sum_{i=1}^{N} c_{i,y} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{k} = \frac{1}{k}$$

3.
$$\mu_y = \frac{\sum_{i=1}^{N} c_{i,y}\cdot x_i}{\sum_{i=1}^{N} c_{i,y}} = \frac{\frac{1}{k}\cdot\sum_{i=1}^{N} x_i}{\frac{N}{k}} = \frac{1}{N}\cdot\sum_{i=1}^{N} x_i \overset{\text{Surprisingly so}}{=} \mu_y \overset{\text{Def.}}{=} \mu$$

4.
$$\Sigma_y = \frac{\sum_{i=1}^{N} c_{i,y}(x_i-\mu_y)(x_i-\mu_y)^\intercal}{\sum_{i=1}^{N} c_{i,y}} = \frac{\frac{1}{k}\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^\intercal}{\frac{N}{k}} = \frac{\sum_{i=1}^{N}(x_i-\mu)(x_i-\mu)^\intercal}{N} \overset{\text{Def.}}{=} \Sigma_y$$

As we can see, the algorithm will assign the values to the variables - $c_{i,y} = \frac{1}{k}, \pi_y = \frac{1}{k}, \mu_y = \mu, \Sigma_y = \Sigma$ .And so, since the termination of the algorithm depends of the convergence of $c_{i,y}$ (and $c_{i,y}$ converges immediately to $\frac{1}{k}$), all of the other variables(π_y, μ_y, Σ_y) stay as they were initialized, And the algorithm is useless.

Part II
Practical Part

1 Preface

In this section we were asked to code 3 models for denoising images. The models were are MVN,GSM,ICA. I will explain how I implemented each one.

2 Coding the Models

2.1 MVN

The MVN model assumes the data comes from a multivariate Normal distribution, and as we learned in class if the data comes from thisa kind of distribution we can simply use the Weiner filter to find the optimal solutionto the denoising problem. In order to learn the parameters for this model i simply needed to calculate the mean and covariance of the data. While the denoising part, as i have said before simply uses the weiner filter:

$$Weiner(y) = (\Sigma^{-1} + \frac{1}{\sigma^2} \cdot I)^{-1} \cdot (\Sigma^{-1}\mu + \frac{1}{\sigma^2} \cdot y)$$

Where x was our original picture, and $y = x + \sigma$ (σ is some normal noise) , Σ represents our covariance matrix and μ represents the mean. Thi model was pretty much straight forward.

2.2 GSM

The GSM model assumes our data comes from a Gaussian mixture of some sort, meaning our data samples are generated from a mixture of a finite number(k) of Gaussian distributions of unknown parameters. In order to learn the parameters of this model we will use the EM algorithm:

Algorithm 1 The EM Algorithm for a Gaussian Mixture Model

1: Initialize $\{\pi_y, \mu_y, \Sigma_y\}_{y=1}^k$
2: **while** not converged **do**
3: $\quad c_{i,y} = \frac{\pi_y \mathcal{N}(x_i;\mu_y,\Sigma_y)}{\sum_{l=1}^k \pi_l \mathcal{N}(x_i;\mu_l,\Sigma_l)}$
4: $\quad \pi_y = \frac{1}{N}\sum_{i=1}^N c_{i,y}$

Figure 1: EM Algorithm

We will only need to learn the π_y and $c_{i,y}$ since we assume a shared co-variance matrix and a 0 mean. In addition we will have to calculate the "weights" of the Gaussian- r_y which are calculated i the following way: $r_y^2 = \frac{\sum_{i=1}^n c_{i,y} \cdot x_i^\intercal \Sigma^{-1} x_i}{d \cdot \sum_{i=1}^n c_{i,y}}$. Also we would like to work in log space, and so the this iterative process will be logged. In my code the learning process was done with using as efficiently as I could - I tried to calculate everything i could before the while loop, and saw that I could use the np.tile function to calculate the $x_i^\intercal \Sigma^{-1} x_i$ term which repeated all the time with different scalars beforehand. And when looking at the $c_{i,y}$ formula I tried to simplify it:

$$ln(\mathcal{N}(x;0,\Sigma)) = ln(\frac{1}{\sqrt{det(2\pi\Sigma)}} e^{-\frac{1}{2}r^2 x^\intercal \Sigma^{-1} x}) = ln(\frac{1}{\sqrt{(2\pi)^d det(r^2\Sigma)}}) - \frac{1}{2r^2} x_i^\intercal \Sigma^{-1} x_i$$

$$\overset{\text{Simple Algebra}}{=} -\frac{d}{2}ln(2\pi) - \frac{d}{2}ln(r^2) - \frac{1}{2}ln(|\Sigma|) - \frac{1}{2r^2} x_i^\intercal \Sigma^{-1} x_i$$

My code reflects this simplification, and the iteration process stops once the $c'_{i,y}s$ converge, which in my test usually took around ~70 iteration. The GSM denoising process, much like the MVN model, uses the Weiner filter, with one big difference - it uses a weighed model of the Wiener formula. Here we calculate weights c_i for each k of our Gaussians - where $c_i = \frac{\pi_i \mathcal{N}(y;0,\Sigma_i+\sigma^2 I)}{\sum_{j=1}^k \pi_j \mathcal{N}(y;0,\Sigma_j+\sigma^2 I)}$ (We will once again work in log space, but no significant simplifications here:() . And in the end we will calculate our result using a weighted Weiner formula: $x_{denoised} = \sum_{i=1}^k c_i Weiner_i(y)$.

2.3 ICA

The ICA model is general model in which we assume each of our variables are sampled from a fixed linear combination of non Gaussians. Specifically we assume that for our d dimensional sample vector(picture), each coordinate is a random variable from a Gaussian mixture distribution. And so in the model I relied heavily of my GSM implementation. For learning the ICA model I first calculated the co-variance matrix, diagonalized it, and multiplied my data by the resulting eigenvector matrix. Next I took each coordinate from my sample data, and trained my learn_GSM function to get d models i could then combine into the ICA_model object. And in order to denoise the images I then again used the denoise_GSM function for each of the test samples(multiplied by the eigenvector matrix).

3 Evaluating & Comparing Models

Lets look at the results of the 3 models trained on 20,000 samples with noise levels: 0.01, 0.05, 0.1, 0.2(top is noised, bottom is denoised):

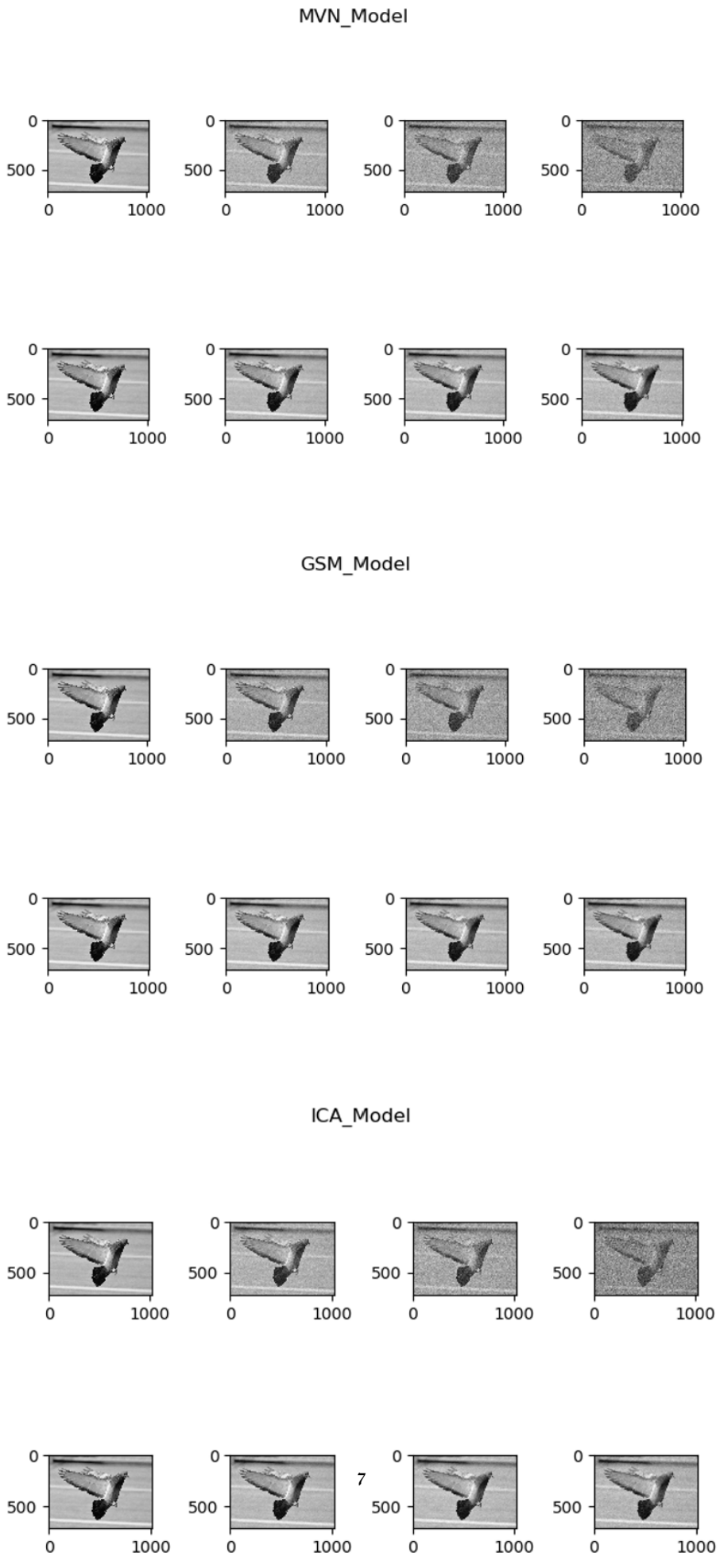

Figure 2：1. MVN Model Results 2. GSM(k=5) Results 3. ICA Model(k=5) Results.

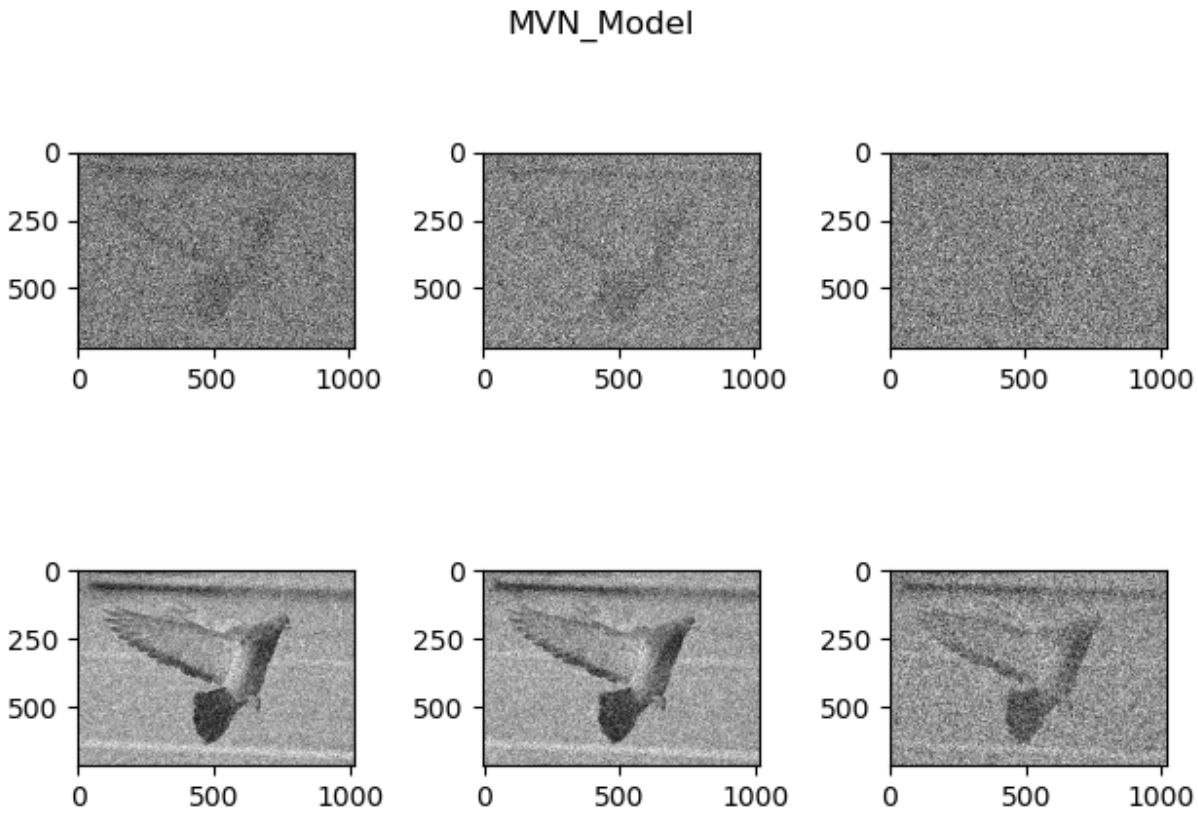

Figure 3: An example of MNV With noise levels: 0.8, 0.9, 1(!)

First of all the results(at least to me) seem pretty impressive! All three model cleaned the images well enough to identify the object in the original images. And at first glance its even hard to distinguish which model did better, since they all preformed really good. And in figure 3. we can see that sometimes even with extreme noise where we can hardly see the image, the model can still denoise it really well. Lets look at some MSE graphs for each model to try and find out who's the best!

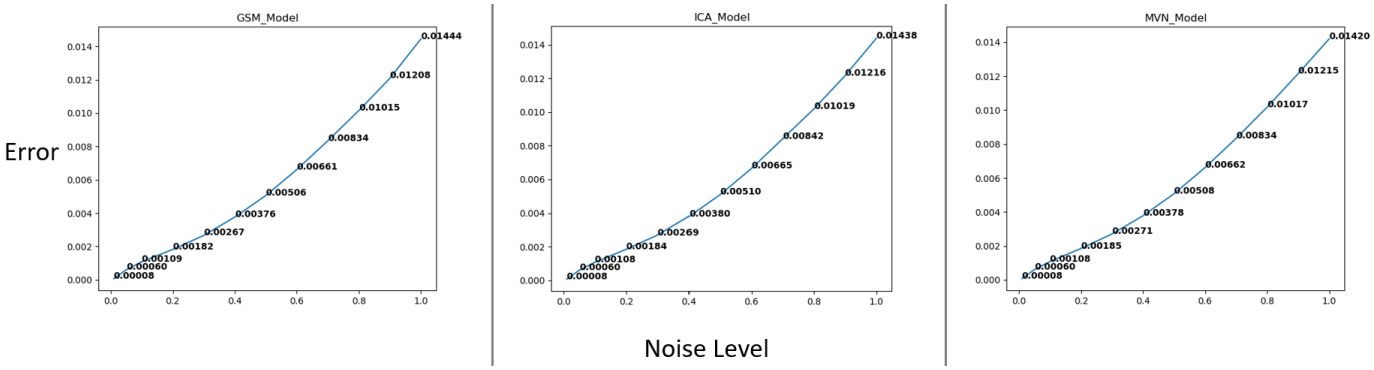

Figure 4: MSE of Three Models(k=5 in last two models)

As we can see there isn't a clear distinction here either, and all 3 models seem to work well. Also, just for fun, since the models worked really well even on really high noise levels, I wanted to see whether it would benefit from running the same algorithm over the denoised pictures, to see if it could denoise them even further:

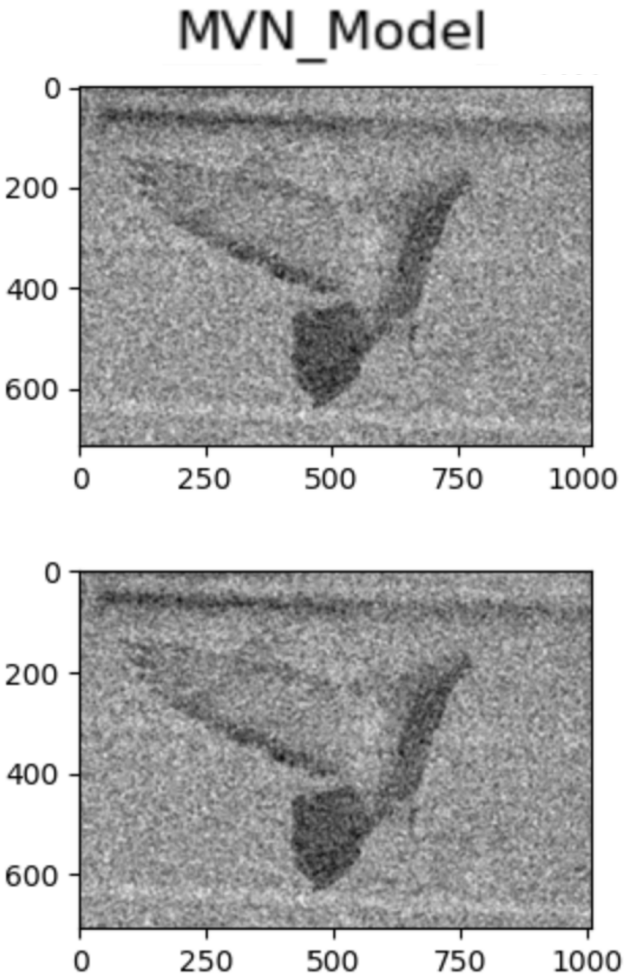

Figure 5: Top: Denoised picture with original noise level 1. Bottom: Denoised once more

As we can see in figure 5. attempting the denoising procedure again did not do anything to the picture. I was initially motivated to do this, because I thought maybe the model would be able to detect the "graininess" in the picture as the noise it originally was, and we would be able to get an even better denoised image. In retrospect this makes all the seance in the world, since our models either use a system in which we search for a minima using the Wiener filter case(of which we wouldst expect to fall out of), or an iterative process - in the EM algorithm approach, and running the same algorithm on an image that already went though the iterative process shouldn't add anything. I decided to add this attempt to the report since I found this result interesting.

In addition, I wanted to see if changing the number of components(k) of the GSM model will have an affect on the result. i tried running GSM and ICA with k=1, 2, 3, 5, 10, 20, 100 and both for GSM and ICA, I couldn't see any significant difference in the generated denoised images. Also, the MSE graphs stayed the same(didn't attach them because they were so similar to Figure 4.).