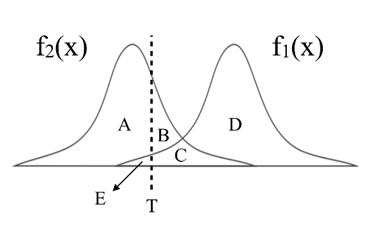
**Up Computer Vision: from Recognition to Geometry**

**HW2**

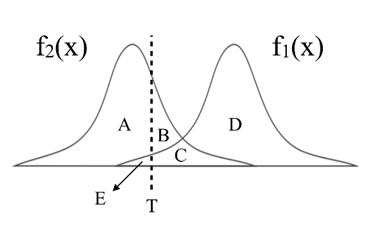
**Name:** 楊仲萱 **Department:** 電子所 ICS組 博一 **Student ID:** F07943023

Problem 1

1. Assume 𝑋 is a continuous random variable that denotes the estimated probability of a binary classifier. The instance is classified as positive if 𝑋 > 𝑇 and negative otherwise. When the instance is positive, 𝑋 follows a PDF 𝑓1(𝑥). When the instance is negative, 𝑋 follows a PDF 𝑓2(𝑥). Please specify which regions (A ~ E) represent the cases of *False Positive* and *False Negative*, respectively. Clearly explain why. (6%)



Answer:



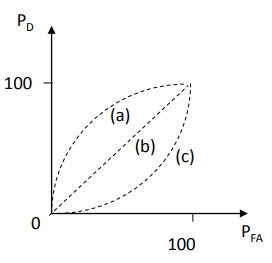
**False Positive**

**False Negative**

The region B and region C represent the case of *False Positive* since they are under the PDF 𝑓2(𝑥) which are negative in reality but are classified as positive. Similarly, the region E follows the PDF 𝑓1(𝑥) meaning it’s positive in reality but is classified as negative, therefore the region E represent the case of *False Negative.*

It makes sense that when we lower the threshold 𝑇 from the middle, there are more instances being classified as positive. As the result, the area of *False Positive* will increase and the area of *False Negative* will decrease.

1. There are three ROC curves in the plot below. Please specify which ROC curves are considered to have reasonable discriminating ability, and which are not. Also, please answer that under what circumstances will the ROC curve fall on curve (b)? (6%)

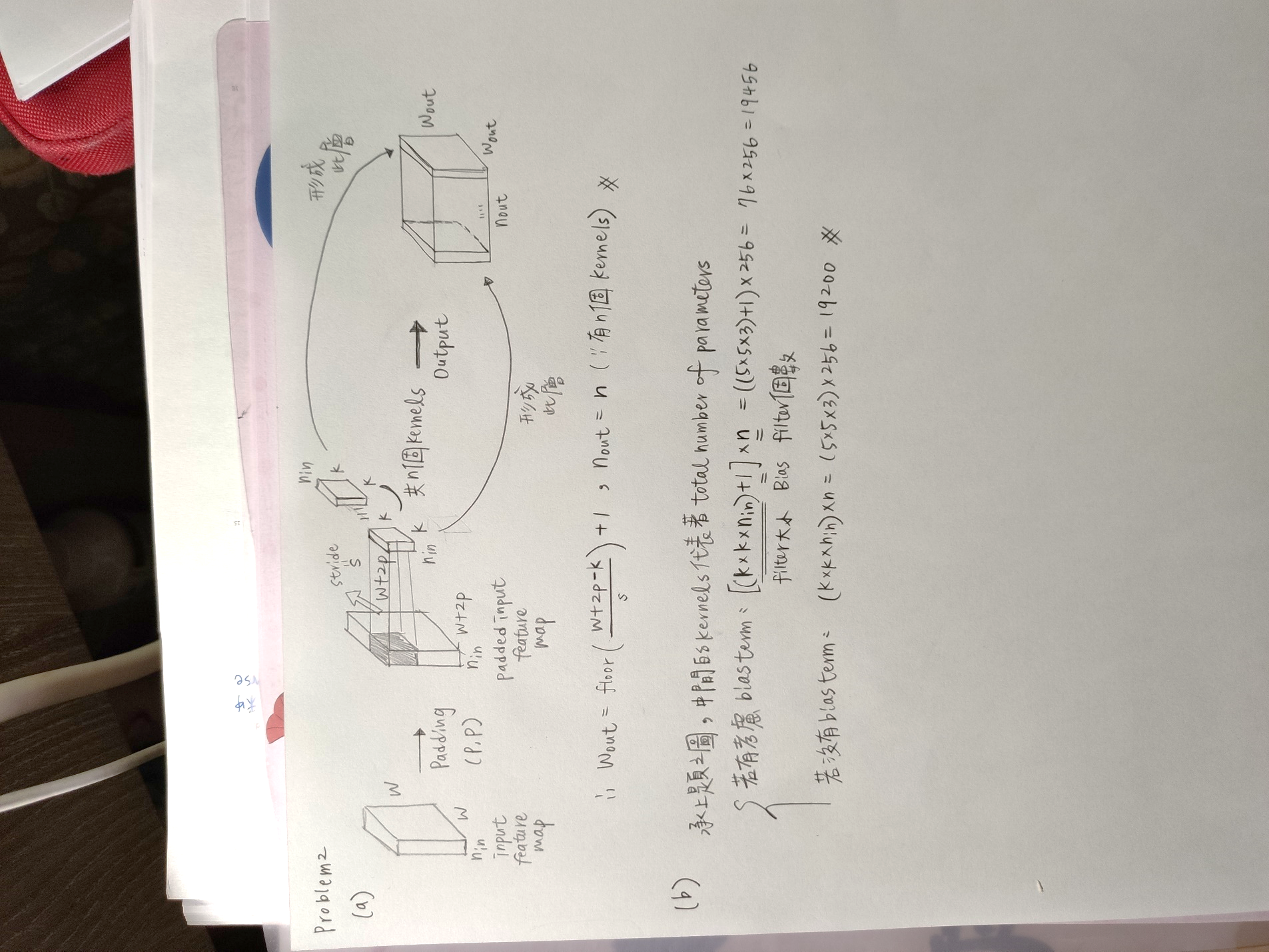


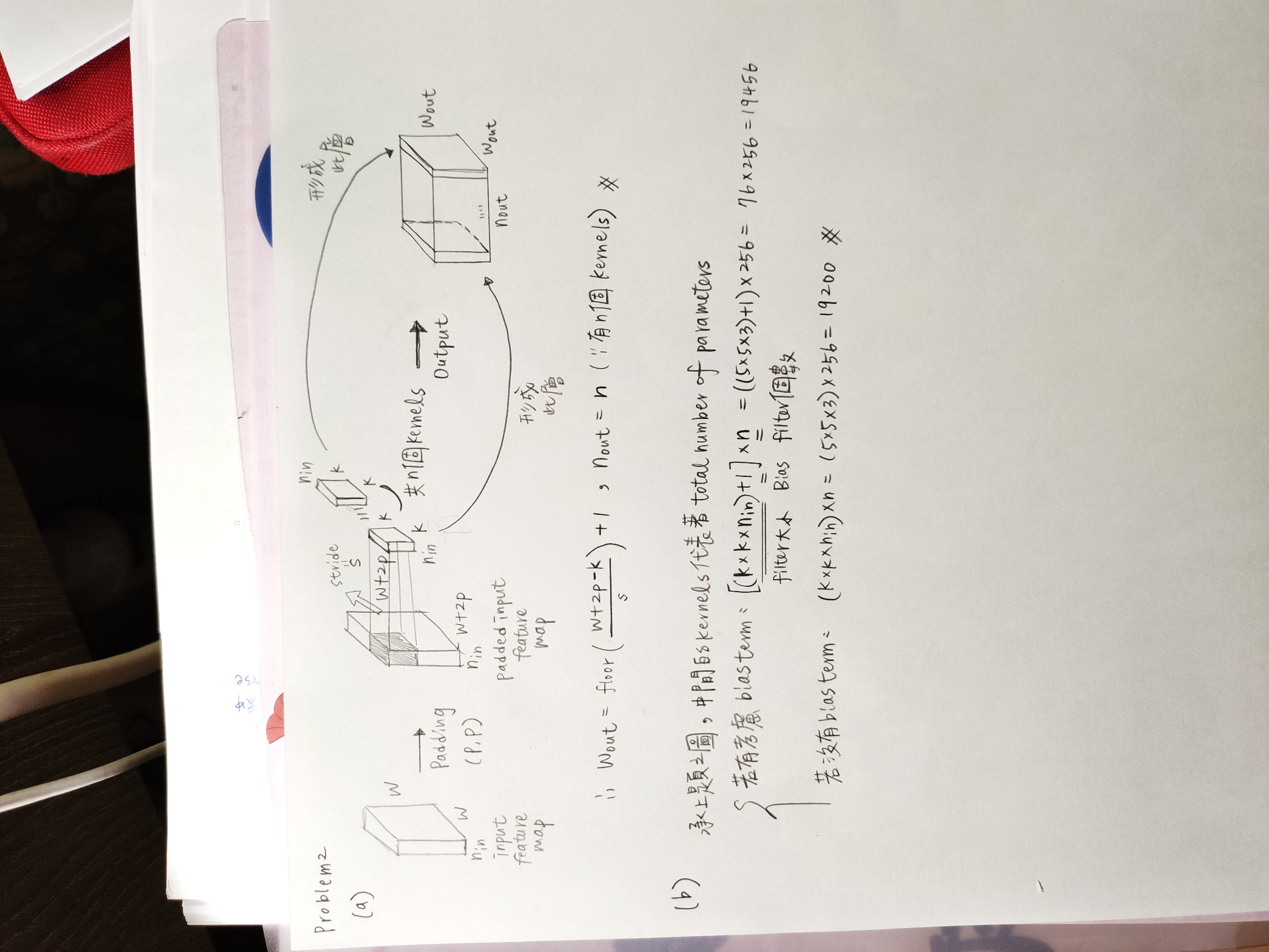
Answer:

The ROC curve (a) are considered to have reasonable discriminating ability, since when we move the threshold value away from negative infinity, the value of drops faster than which is reasonable. On the other hand, ROC curves (b) and (c) are not considered to have reasonable discriminating ability since curve (b) is just a random guess classifier and curve (c) is a classifier that is worse than random classifier.

When a predictor makes its decision by random guesses, the ROC curve will fall on curve (b) which is the line of no-discrimination. Because of the randomness, the fraction of cases in true positive is the same as it in false positive, meaning that the true positive rate and the false positive rate are the same (diagonal line).

Problem 2

1. Given a convolutional layer which contains:   
    𝑛 kernels with size   
    padding size (𝑝, 𝑝)   
    stride size (𝑠, 𝑠).   
   With input feature size , calculate the size of output feature .
   1. 𝑊\_𝑜𝑢𝑡=? (2%)
   2. 𝑛\_𝑜𝑢𝑡 =? (2%)
2. = 3, 𝑘 = 5, 𝑠 = 2, 𝑝 = 1, 𝑛 = 256, 𝑊 = 64, calculate the number of parameters in the convolutional layer (4%)



Problem 3 (report 35% + code 5%)

1. PCA (15%)

In this task, you need to implement PCA from scratch, which means you cannot call PCA function directly from existing packages.

1. Perform PCA on the training data. Plot the mean face and the first five eigenfaces and show them in the report. (5%)

The mean face of all 280 training data is shown below.



The first five eigenfaces are shown in the following table.

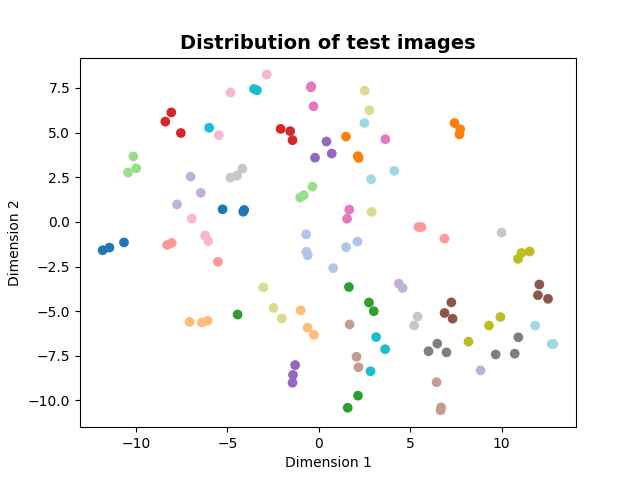
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Eigenface 1** | **Eigenface 2** | **Eigenface 3** | **Eigenface 4** | **Eigenface 5** |
| Result  Image |  |  |  |  |  |

1. Take 𝒑𝒆𝒓𝒔𝒐𝒏8\_ 𝒊𝒎𝒂𝒈𝒆6, and project it onto the above PCA eigenspace. Reconstruct this image using the first n = { 5, 50, 150, all } eigenfaces. For each n, compute the mean square error (MSE) between the reconstructed face image and the original 𝒑𝒆𝒓𝒔𝒐𝒏8\_ 𝒊𝒎𝒂𝒈𝒆6. Plot these reconstructed images with the corresponding MSE values in the report. (5%)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Original\_8\_6** | **n = 5** | **n = 50** | **n = 150** | **n = all (280)** |
| MSE |  | 694 (693.702) | 119 (119.200) | 40 (40.397) | 0 (9.5x10-25) |
| Result  Image |  |  |  |  |  |

1. Reduce the dimension of the image in testing set to dim = 100. Use t-SNE to visualize the distribution of test images. (5%)

After reducing the dimension of test images and visualizing with t-SNE, the distribution is shown below.

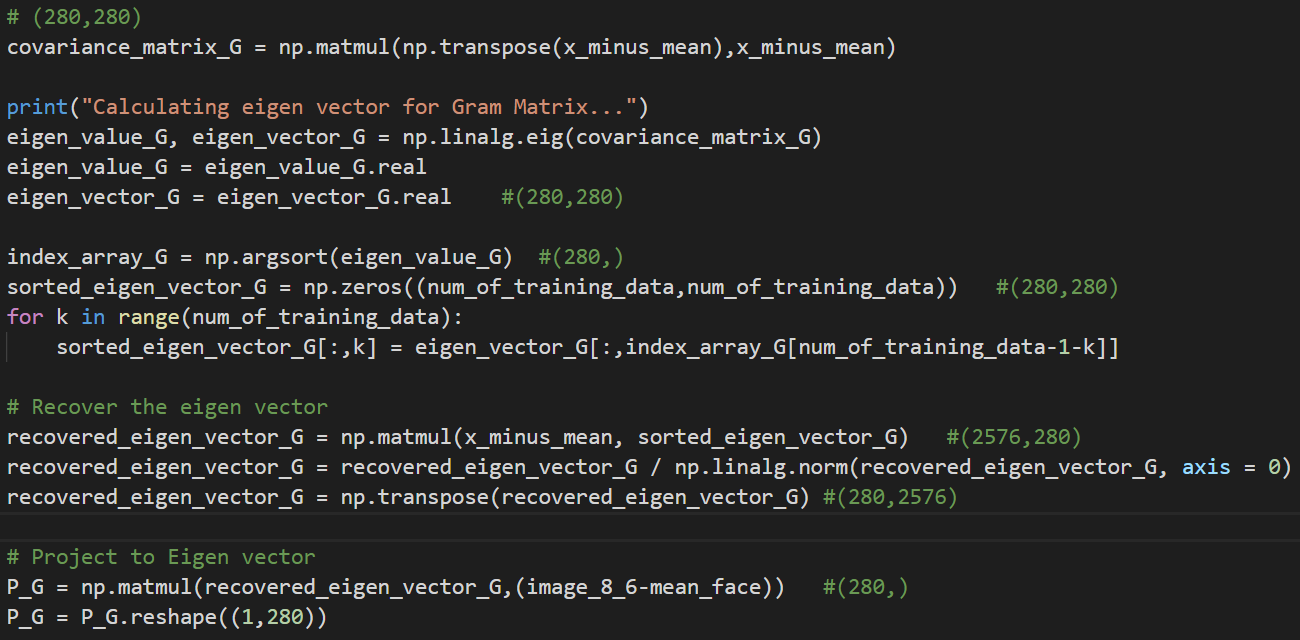


1. (bonus 5%) Implement the Gram Matrix trick for PCA. Compare the two reconstruction images from standard PCA/Gram Matrix process. If the results are different, please explain the reason. Paste the main code fragment(screenshot) on the report with discussion.

The results from standard PCA and Gram Matrix are the same. The reconstructed output images are listed in the table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Original** | **n = 5** | **n = 50** | **n = 150** | **n = all (280)** |
| Reconstruct images from standard PCA | | | | | |
| MSE | 0 | 694  (693.702) | 119  (119.200) | 40  (40.397) | 0  (9.5x10-25) |
| Result  Image |  |  |  |  |  |
| Reconstruct images from Gram Matrix | | | | | |
| MSE |  | 694  (693.702) | 119  (119.200) | 40  (40.397) | 0  (0.283) |
|  |  |  |  |  |  |

The main code fragment for Gram Matrix is in the following picture.



The new covariance matrix is now only in the size of (280,280) which is much smaller than the original one (2576,2576). As the result, the process of finding eigen vectors and eigen values can be faster. In order to find the real eigen vector, it can be achieved with multiplying the input (X-mean) matrix and then normalizing the result. After these steps, the true eigen vector can be recovered, therefore the results from two methods are the same.

1. k-NN (10%)

To apply the k-nearest neighbors (k-NN) classifier to recognize the testing set images, please determine the best k and n values by 3-fold cross-validation.

For simplicity, the choices for such hyper-parameters are:

k = {1, 3, 5} and n = {3, 10, 39}.

Please show the cross-validation results and explain your choice for (k, n). Also, show the recognition rate on the testing set using your hyper-parameter choice.

|  |  |  |  |
| --- | --- | --- | --- |
| **The accuracy after using 3-fold cross-validation** | | | |
|  | k = 1 | k = 3 | k = 5 |
| n = 3 | 0.68333  (0.7 + 0.6625 + 0.6875) / 3 | 0.59306  (0.5417 + 0.625 + 0.6125) / 3 | 0.51528  (0.4333 + 0.5125 + 0.6) / 3 |
| n = 10 | 0.85139  (0.8667 + 0.8625 + 0.825) / 3 | 0.74583  (0.725 + 0.7875 + 0.725) / 3 | 0.67778  (0.6833 + 0.6875 + 0.6625) / 3 |
| n = 39 | **0.92361**  (0.9333 + 0.9375 + 0.9) / 3 | 0.84861  (0.8083 + 0.8875 + 0.85) / 3 | 0.77639  (0.7417 + 0.8 + 0.7875) / 3 |

With the help of cross\_val\_score function from sklearn.model\_selection, I take the average of accuracy from 3-fold cross-validation as its final accuracy. Since the accuracy when using k as 1 and n as 39 is the highest, I chose (k, n) = (1, 39) as the hyper-parameter.

The recognition rate on the testing set (120 images) using (k, n) = (1, 39) is **0.95833**

1. K-means (10%)

Reduce the dimension of the images in the first 10 class of training set to dim = 10 using PCA. Implement the k-means clustering method to classify these images.

Please use weighted Euclidean distance to implement the k-means clustering. The weight of the best 10 eigenfaces is [ 0.6, 0.4, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1 ]

1. Please visualize the features and the centroids to 2D space with the first two eigenfaces for each iteration in k-means clustering (up to 5 images). (5%)

The blue “X” represents the centroid of each cluster.

|  |  |  |
| --- | --- | --- |
| **Initialization (Iteration 0)** | **Iteration 1** | **Iteration 2** |
|  |  |  |
| **Iteration 3** | **Iteration 4** |
|  |  |

1. Compare the results of K-means and ground truth. (5%)

The results of K-means clustering is down below.

[0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 **4 4 3 3** 3 **2** 3 3 **2** 3 3 4 4 4 4 4 4 4 5 5 5 5 5 5 5 6 6 6 6 6 6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 9 9 9 9 9 9 9]

The ground truth in green and red part is 2, and the ground truth in blue part is 3.

After input images being projected to dim = 10, they are more similar to resulted means.

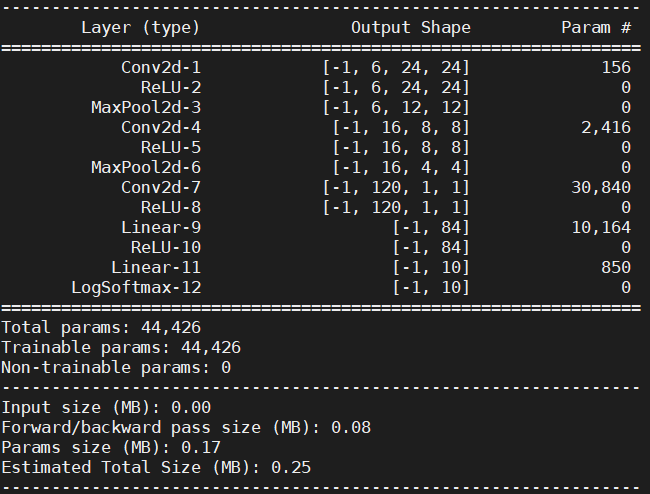
|  |  |  |  |
| --- | --- | --- | --- |
| Ground Truth | Results of K-means | The projected input image | |
| Projected mean\_2 | Projected mean\_4 | 2\_3 | 2\_4 |
| Projected mean\_2 | Projected mean\_3 | 2\_5 | 2\_6 |
| Projected mean\_3 | Projected mean\_2 | 3\_1 | 3\_4 |

Problem 4 (report 30% + baseline 10%)

1. MNIST Classification (20%)
2. Build a CNN model and a Fully-Connected Network and train them on the MNIST dataset. Show the architecture of your models and the correspond parameters amount in the report. (5%)

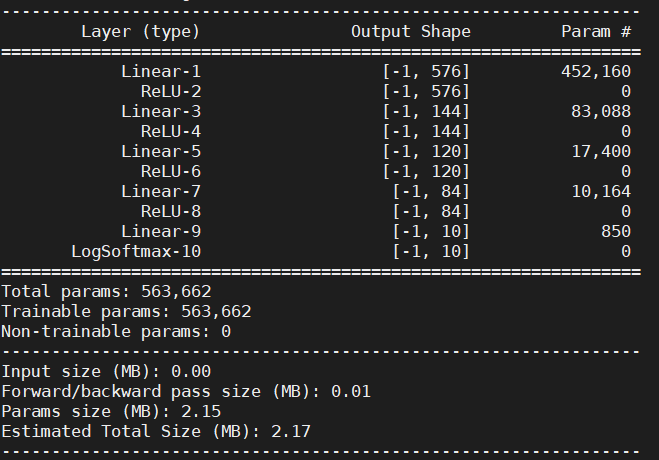
For CNN model, the architecture is shown below.

The amount of parameters is 44,426.



For Fully-Connected Network model, the architecture is shown below.

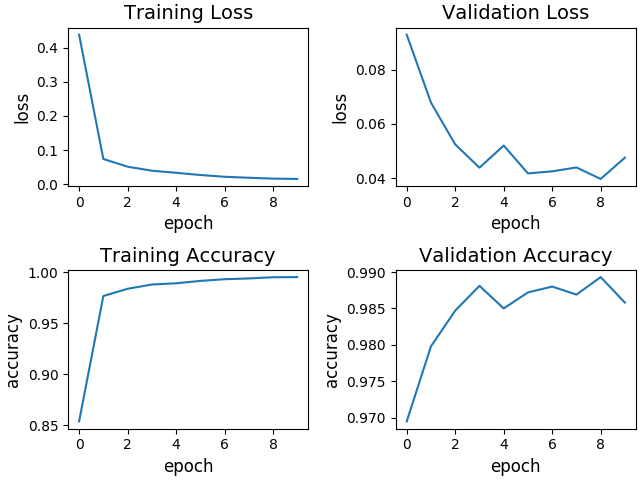
The amount of parameters is 563,662.



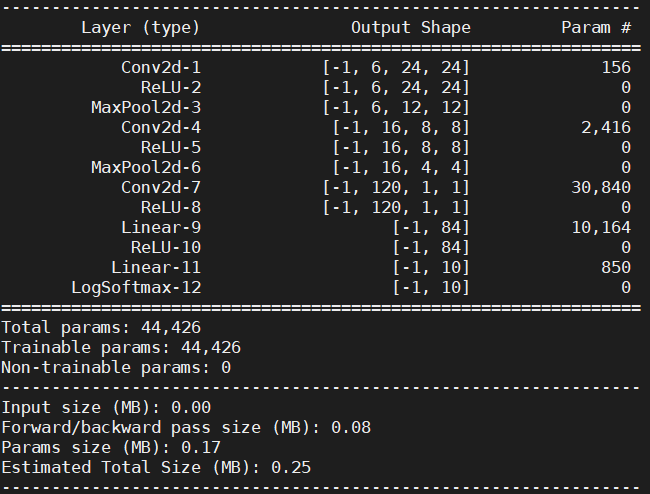
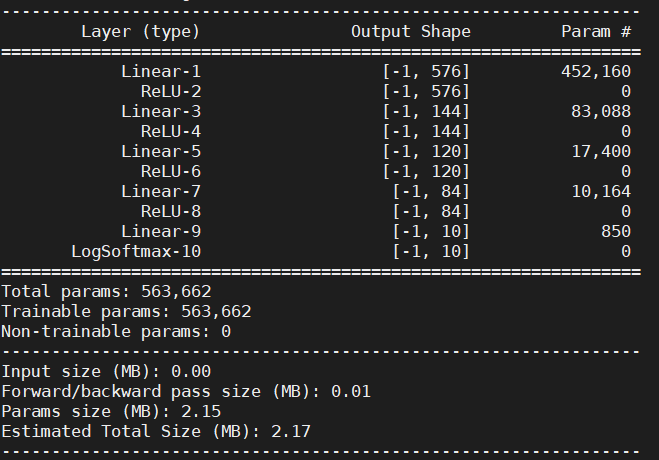
1. Report your training / validation accuracy, and plot the learning curve (loss, accuracy) of the training process. (figure 5%+ baseline 5%)

(2-1) CNN model: Training accuracy: **99.5%** ; Validation accuracy: **98.6%**.

(2-2) Fully-Connected Network: Training accuracy: **98.6%** ; Validation accuracy: **97.5%**.

 **CNN model Fully-Connected Network**

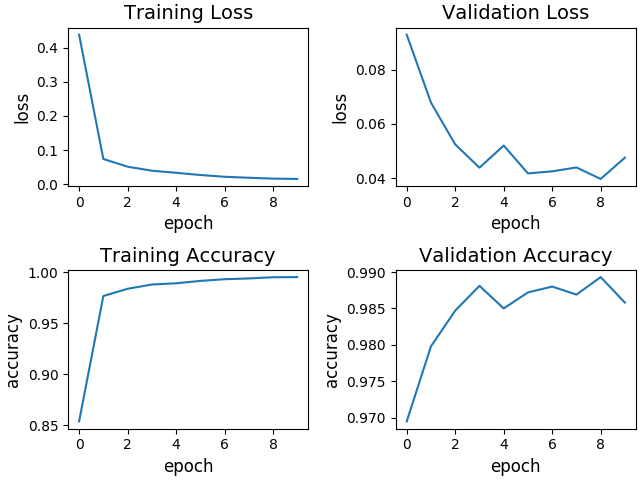
1. Compare the results of both models and explain the difference. (5%)

CNN Model Fully-Connected Network

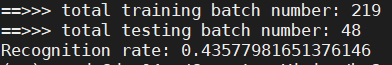
The number of layers in CNN model is more than which in Fully-Connected network, but the total amount of parameters in CNN model is less than in Fully-Connected Network. Therefore, it shows that using CNN model is more suitable and more efficient for MNIST classification.

Furthermore, the final validation accuracy when using CNN model is higher than using Fully-Connected network which can be seen in the following figure.

**CNN** **Fully-Connected Network**



1. Face Recognition (20%)
2. Extract image feature using pytorch pretrained alexnet and train a KNN classifier to perform human face recognition on the given dataset. Report the validation accuracy. (5%)

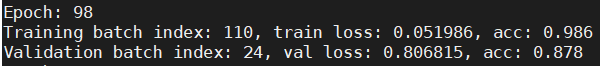
The validation accuracy is 43.578%

1. Build your own model and train it on the given dataset to surpass the accuracy of previous stage. Show and explain the architecture of your model and report the validation accuracy. Paste the main code fragment(screenshot). (5%)



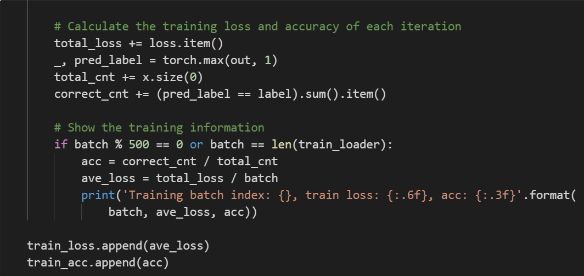
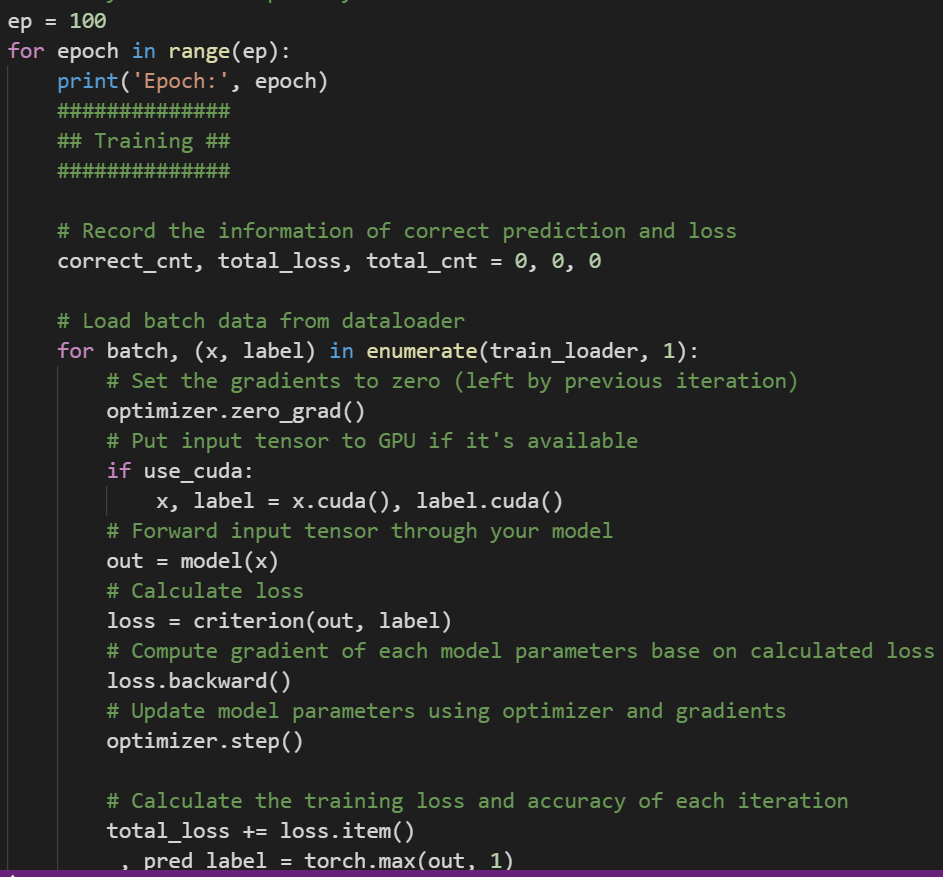
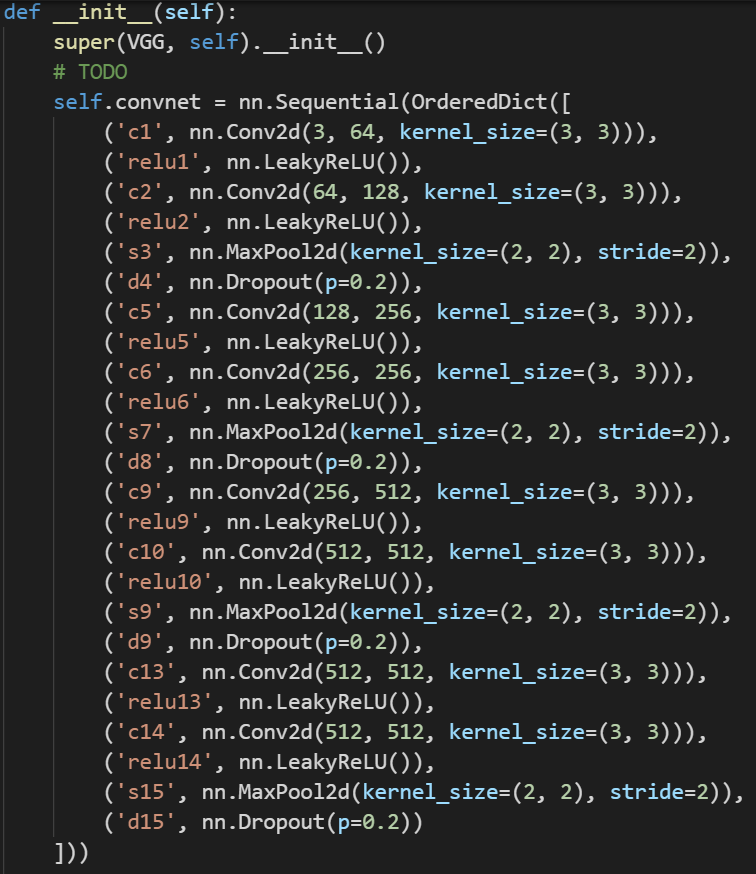
The architecture of my model is shown in the right figure. Since the task is face recognition, I decided to use VGG architecture as my reference to build my model and add some dropout layers to make its validation accuracy higher.

The final validation accuracy is shown in the below figure, which is 87.8%

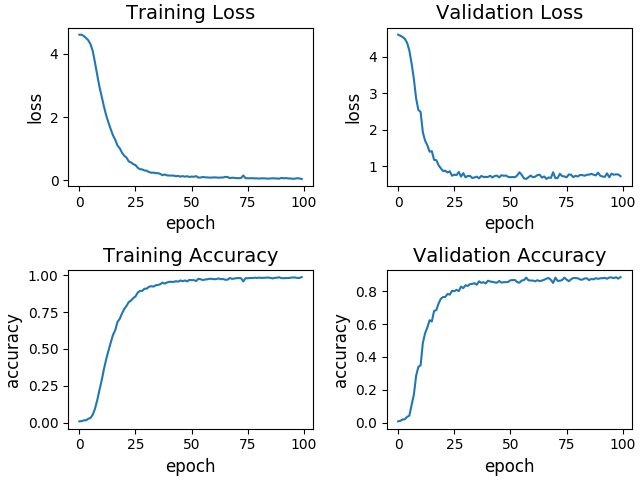


The main code fragment (screenshot) is in the following figures including the training part and the model building part.

**Training Part**  **Model Building Part**



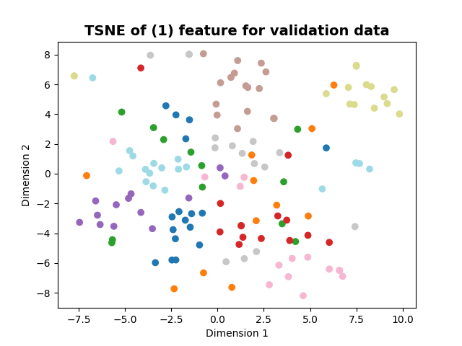
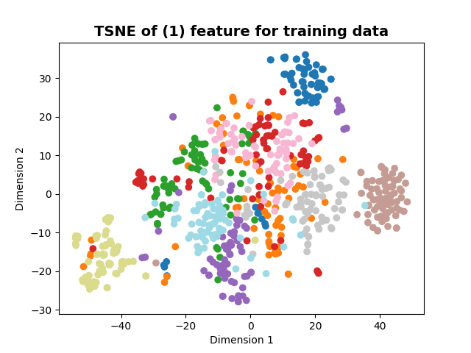
1. Plot the learning curve of your training process(training/validation loss/accuracy) in stage 2, explain the result you observed. (5%)



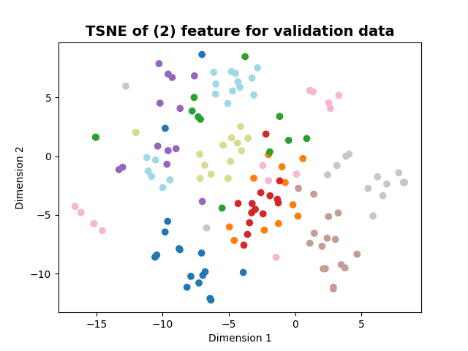
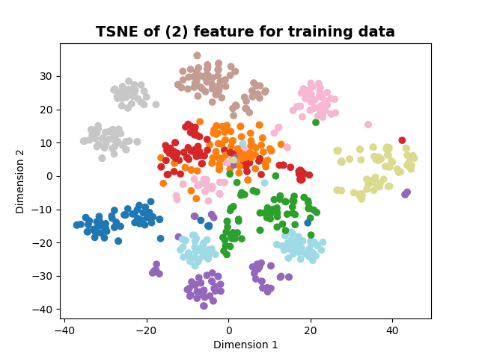
The validation accuracy is higher compared to the baseline accuracy. Moreover, it needs about 25 epochs to achieve the point that the following accuracy won’t increase dramatically, but the baseline model only took about 4 epochs.

1. Pick the first 10 identities from dataset. Visualize the features of training/validation data you extract from pretrained alexnet and your own model to 2D space with t-distributed stochastic neighbor embedding (t-SNE), compare the results and explain the difference. (5%)

It’s obvious that the features of training and validation data extracted from my model are more separate meaning that the accuracy for classification will be higher which is the same as the final result. The reason is that the VGG architecture is more suitable and the data processed by more convolutional layers will get high-level features.



**Pretrained Alexnet**



**My own model**