שניר קוסקא – 208144477
דור ארביב - 206556318

למידה עמוקה – תרגיל בית 3

שאלה 1:

שאלה 3 - חלק תיאורטי

1. Q - נתונות שתי פונ' בצפיפות המרבדות כדלקמן:

1. $\forall (x,y) \in P$, $x = 0$ & $y \sim U(0,1)$
2. $\forall (x,y) \in Q$, $x = \theta$ & $y \sim U(0,1)$
   $\theta \in [0,1]$

אם נתבונן $\theta \neq 0$ נקבל:

1. $D_{KL}(P \| Q) = \int_{y=0}^{1} 1 \cdot \log\left(\frac{1}{0}\right) dy = +\infty$
   $x=0$

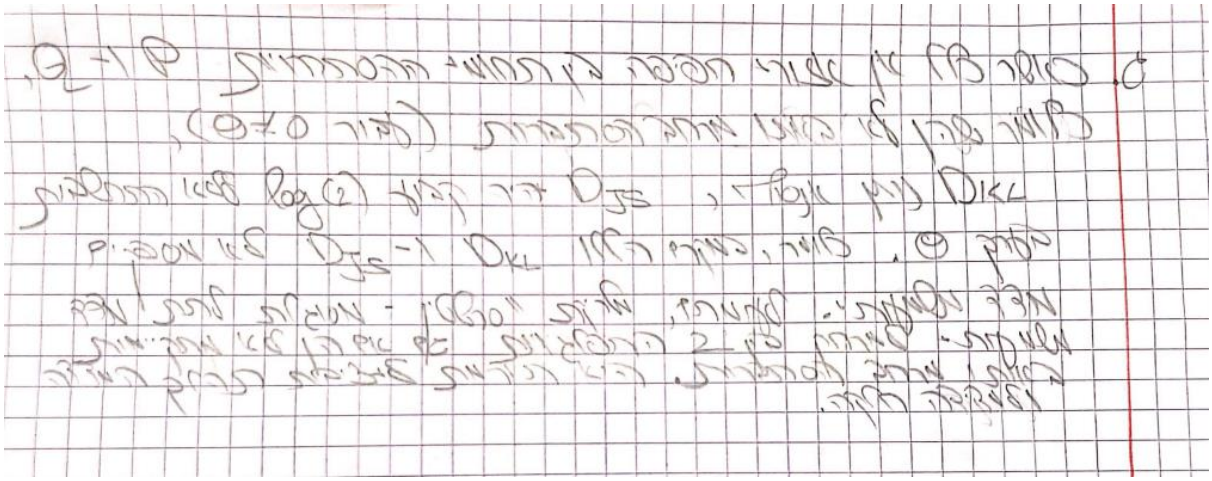
2. $D_{JS}(P \| Q) = \frac{1}{2} \cdot \delta(x) \cdot \int_{y=0}^{1} 1 \cdot \log\left(\frac{1}{\frac{1}{2}}\right) +$
   $+ \frac{1}{2} \cdot \delta(x-\theta) \cdot \int_{y=0}^{1} 1 \cdot \log\left(\frac{1}{\frac{1}{2}}\right) = \log(2)$

3. $W(P,Q) = |\theta|$

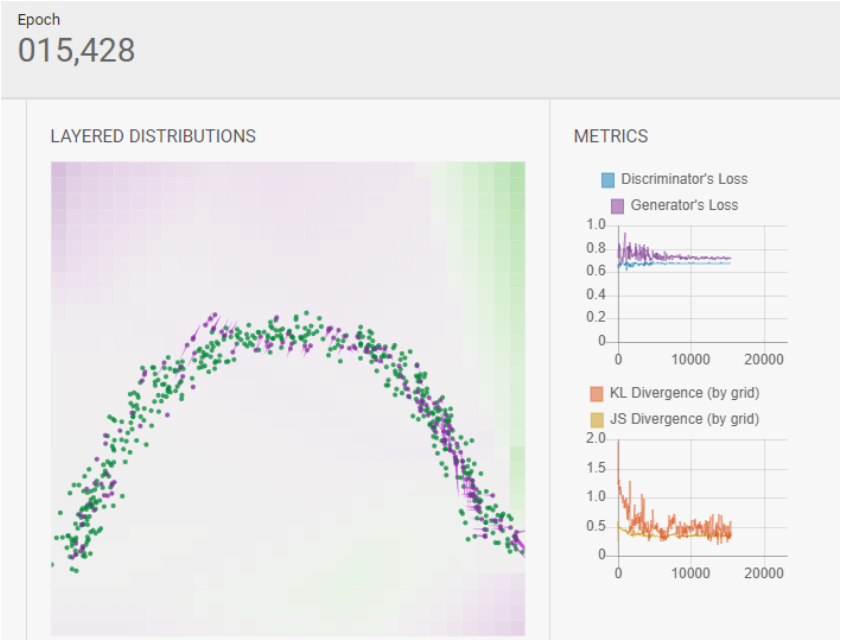
מ. $\theta = 0$: מעתה גם ההתפלגות - $P = Q$ ולכן:

1. $D_{KL}(P \| Q) = D_{KL}(P \| P) = 0$.
2. $D_{JS}(P \| Q) = D_{JS}(P \| P) = \frac{1}{2} D_{KL}(P \| P) + \frac{1}{2} D_{KL}(P \| P) \Leftrightarrow$
   $\Leftrightarrow D_{KL}(P \| P) = 0$.
3. $W(P,Q) = |\theta| = 0$

שניר קוסקא – 208144477
דור ארביב - 206556318



שאלה 2:



בוויזואליזציה הזו, דוגמאות מההתפלגות (בצורה פרבולית) שצוירה מופיעות בנקודות הירוקות ואילו דוגמאות שהגנרטור מייצר מופיעות בנקודות הסגולות.

הדיסקרימינטור מנסה להבחין בין דוגמאות אמיתיות מהפילוג המקורי שצויר לבין דוגמאות מזויפות מהפילוג שנוצר ע״י הגנרטור (שמטרתו לקחת רעש בכניסה וליצור דוגמאות מזויפות שייראו כמה שיותר קרובות לפילוג המקורי). ניתן להבחין כי לאחר תקופת האימון המתוארת, שני הפילוגים בקירוב טוב חופפים, הלוס של הגנרטור קטן משמעותית והתייצב אך עדיין ישנם קווי גרדיאנט (לא אפסי) על חלק מהנקודות בפילוג המזויף (מה שמצביע על כך שעוד אין התכנסות/ התייצבות ״מיטבית״ של תהליך האימון). כן ניתן להבחין בצבע הרקע (מסווג אזורים לפי החלטת הדיסקרימינטור) בגוון בקירוב אפור אחיד, מה שמצביע על כך שהדיסקרימינטור לא מבדיל בין דוגמאות אמיתיות למזויפות, כלומר שהגנרטור הצליח ללמוד ולחכות בקירוב טוב את הפילוג המקורי.

שניר קוסקא – 208144477
דור ארביב - 206556318

Question 3 (results + readme):

*This code was written in google colab and is saved in file ex3_208144477_206556318.ipynb in the shared folder.

*This readme file provides an overview of the code implementation for semi-supervised learning using a Variational Autoencoder (VAE) for feature extraction and a (transductive) Support Vector Machine (SVM) for classification. The code includes training and testing the VAE and SVM models, and analyzes the results on FashionMNIST and MNIST.

*The parts of the code are the following:
creating database, defining encoder-decoder architecture (VAE), training VAE mode, testing the VAE, generating of latent representation, performing SVM predictions, displaying and evaluating the results for the different test scenarios.

* The code supports two datasets: FashionMNIST and MNIST. In database creation part the datasets were automatically downloaded and preprocessed using the torchvision library, for training and testing processes. The number of labels determines the size of the training set.
The architecture is applied on the two datasets with different number of labeled examples in the training set (n_labels = 100, 600, 1000, 3000).

*Models definition: 'Model: encoder', 'Model: decoder' and 'Model: VAE' parts, the network architecture definition is performed. The classification model: LinearSVM is defined in 'Classification model: SVM' part.
Also, The weights of the linear layers in the VAE are initialized using Xavier uniform initialization. This initialization technique helps in maintaining the variance of activations and gradients during training, leading to more stable and efficient learning.

*SVM Classifier and Kernel Selection:
The SVM classifier is chosen as the transductive classifier for this semi-supervised learning task. In the code, the linear SVM classifier (svm.LinearSVC()) is used with a random state of 0 and a tolerance of 1e-5.
We chose linear kernel for the SVM classifier (which utilizes the library 'liblinear'), due to its simplicity and good performance for many classification tasks.
The linear SVM offers improved scalability when dealing with large datasets ,provides more flexibility in selecting loss functions, making it adaptable to different scenarios. Also, it handles problems with multiple classes effectively, and supports both dense and sparse input formats, allowing it to cope with a wide range of data representations efficiently.

*The training process is mainly done in 'train_model' and 'fit' functions, using the Adam optimizer, and follows these steps:
Encoding input data into a distribution in the latent space.
Then, from the learned distribution in the latent space, sampling a point.
Decoding the sampled point, and calculating the reconstruction error, which measures the dissimilarity between the original input and the decoded output, accordingly.
The reconstruction error is then used to update the parameters of the network through backpropagation, allowing the model to learn and improve its ability to reconstruct the input data.
And finally, when the model is trained, a lower-dimensional feature space in the latent representation is obtained, where we can apply a SVM to classify between different classes.
The SVM classifier will be trained using the scikit-learn library(fit method) and the latent representation extracted by the VAE(of the training data).

For evaluating the models, the test accuracy, precision, recall, and F-score will be displayed for the chosen dataset and number of labeled examples.

* after the training process, the models are saved with:

```
torch.save(model.state_dict(), f'model_VAE_{n_labels}_labels.pt')
torch.save(svm_model, f'model_SVM_{n_labels}_labels.pt')
```

loading the model is done with:

```
model.load_state_dict(torch.load(f'model_VAE_{n_labels}_labels.pt'))
svm_model_trained = torch.load(f'model_SVM_{n_labels}_labels.pt')
```

*eventually, there are 8 saved models in the shared file (2 for each number of labels):
- model_VAE_{n_labels}_labels.pt
- model_SVM_{n_labels}_labels.pt

*Before running the whole code, please modify the 'mode' variable in the code to choose between training (mode = 'train') to train and then test the models, and testing (mode = 'load') to load the last saved models (in the shared file) and avoid training proccess. our best models are already saved in the shared file (are also backed up in 'models' file). Picking 'train' mode will overwrite the existing models. By default, the code is set to the training mode.

*Results:

In 'results' & 'results table' parts, the accuracy values of each of the models on the test set are displayed.

Results Summary - different datasets and number of labels in training

Dataset	# of labels in train set	Test Accuracy
FashionMNIST	100	66.2
FashionMNIST	600	70.76
FashionMNIST	1000	72.12
FashionMNIST	3000	75.73
MNIST	100	65.77
MNIST	600	79.81
MNIST	1000	81.99
MNIST	3000	84.86

The code evaluates the performance of the model on two datasets: FashionMNIST and also MNIST. The results show that as the number of labeled examples in the training set increases, the test accuracy improves. This phenomenon occurs because the availability of more labeled samples(more information for the models is provided) allows the model to better generalize and learn discriminative features for classification, in order to make accurate predictions.
Also, the results on the MNIST dataset tend to be better than those on the FashionMNIST dataset. This difference can be attributed to the inherent characteristics of the datasets. MNIST consists of handwritten digit images, which are relatively easier to classify compared to FashionMNIST, which contains images of fashion items. It seems that the simpler and more distinct nature of the MNIST dataset probably makes it easier for the model to learn meaningful representations and achieve higher accuracy and better performances also on other assessment metrics(such as- precision, recall and f1 score).

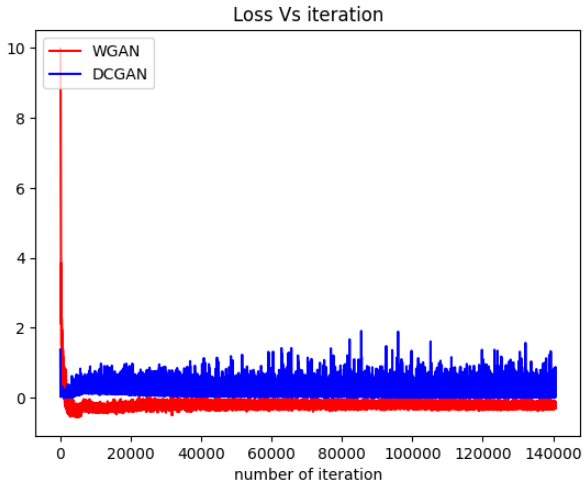
שניר קוסקא – 208144477
דור ארביב - 206556318

שאלה 4: (קובץ readme לחלק זה מופיע בנפרד בתיקיה)

בהתבסס על המאמר הנתון בחלק זה אומנו 2 מודלים WGAN ו- DCGAN. המודל של WGAN יושם ע"י שימוש ב- gradient penalty כפי שתואר במאמר (עם משקל השווה ל -10 לביטוי הנוסף ללוס שמטרתו לשמור את הגרדיאנט של הדיסקרימינטור להיות בקירוב בעל נורמה השווה ל 1).

מודל DCGAN נלקח ממקור [22] במאמר (אליו הופנה במאמר כאשר דובר על מודל זה).

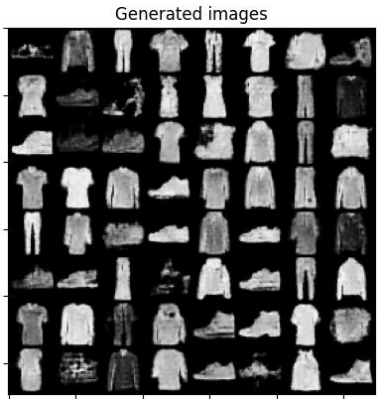
לאחר אימון 2 המודלים, מתקבל גרף הלוס הבא:



ניתן לראות כי האימון בשיטת WGAN בעזרת gradient penalty אכן הביא להקטנה מהירה יותר של הלוס. נראה כעת את תוצאות 2 המודלים.

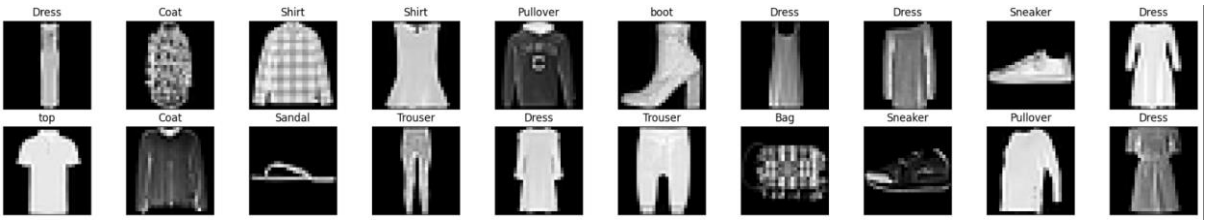
סט התמונות המתקבל לאחר אימון מודל DCGAN:

סט התמונות המתקבל לאחר אימון מודל WGAN:

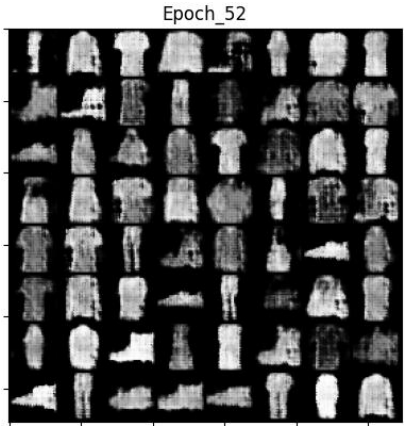


שניר קוסקא – 208144477
דור ארביב - 206556318

סט תמונות אמיתיות:



עבור מודל WGAN היו כ-23 epochs בהם פלט המודל נכשל ולא נראה טוב, הפלט בepoch מספר 52 כבר נראה טוב למדי:



עבור מודל DCGAN היו כ-27 epochs בהם פלט המודל נכשל ולא נראה טוב, הפלט בepoch מספר 65 כבר נראה טוב למדי: