2AMM10 2021-2022 Assignment 4: Group 47

Ahmet Ayrancioglu (1678620) Ricardo Andrade (1725882) Ruben Wolters (1342355)

June 20, 2022

1 Problem formulation

Solving the problem of learning the distribution of a dataset opens the door to other interesting tasks in the machine learning field such as the detection of out-of-distribution data points (1), learning relevant features of the data (2), and performing classification when only very few labels are present (3).

To perform the three tasks mentioned above, 4 datasets built from FashionMNIST will be used. The first dataset contains 26000 in-distribution data points but does not have any labels. This dataset will be used for (2) to achieve tasks (1 and 3). The second dataset has 2000 labeled, in-distribution data points and is used to train the model for classification. The remaining two datasets will be used to evaluate the performance of created models since they contain 1052 fully labeled data points, but these are slightly (5%) out-of-distribution.

To solve this problem, first, a model that encodes the first dataset and learns its distribution is developed. The reconstruction error, given by the mean squared error (MSE) is used for training the autoencoder. Using the trained autoencoder model and the reconstruction errors of the training set, a threshold value is calculated. Using this threshold value, outlier detection can be performed on the last two datasets that contain 5% noise. Finally, a classification head on top of the encoder is developed and trained on the second dataset using the embeddings learned by the autoencoder. This classification head is evaluated on the last 2 datasets.

2 Model formulation

To achieve the described goals, two interconnected models are developed. As mentioned previously, the first part of the network, the encoder, is trained in combination with a decoder, which combined is called an autoencoder. This autoencoder learns the underlying distribution of the data and learns relevant features of the data. This is achieved by creating a bottleneck in the network at the output layer of the encoder. By forcing input to go through a bottleneck in the network, irrelevant features are removed from the input only leaving what is crucially necessary to reconstruct the input. Thus, the autoencoder approach can learn the underlying distribution of the dataset. Additionally, using the autoencoder outlier detection can be performed. This is achieved by putting the inputs into the model and observing how well the model can reconstruct the input. If the reconstruction error is too much, it can be assumed that the input is an outlier and not part of the distribution that the model learned.

The second part of the model, the classifier, is trained after the training of the encoder is finished. Taking advantage of the relevant features that are learned by the encoder, the classification head can achieve high performance even when it is trained with a small number of labeled examples.

To train the autoencoder, Mean Squared Error (MSE) Loss is used, defined as follows:

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where N is the batch size, y is the true value, and $\hat{y}$ is the predicted value by the model.

The classifier is trained using Cross Entropy Loss, defined as follows:

$$L = -\sum_{i=1}^{N} t_i \log(p_i),$$

where t_i it the truth label for class i, p_i is the Softmax probability for the class i and N is the total number of classes.

3 Implementation and training

3.1 Autoencoder

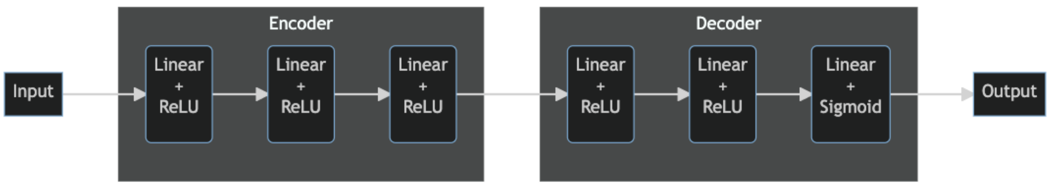

Figure 1: High-level overview of the model of the autoencoder

The autoencoder consists of an encoder and a decoder. The encoder consists of 3 dense layers, with decreasing number of nodes in all layers. All of the layers use ReLU activation. The decoder consists of 3 dense layers, with an increasing number of nodes in all layers, The first two layers use ReLU activation, and the last one uses a Sigmoid. A high-level representation of the model can be seen in Figure 1. The exact description of the layers can be found in Table 1.

For the training of the autoencoder, a reconstruction loss is needed to gauge the performance of the model. For this task, mean squared error (MSE) was chosen. MSE is a straightforward metric to determine how close exactly two numeric inputs are to each other, which is suitable for the task at hand.

To perform the outlier detection, a threshold is computed using the reconstruction loss. After training the autoencoder, the training set is fed back to the model, and reconstruction losses for all samples are recorded. Then the mean of all reconstruction losses was calculated as well as the standard deviation. Finally, a threshold was determined using the following formula:

$$\mathit{threshold} = \mu + 3 \cdot \sigma$$

where μ is the mean of the losses and σ is the standard deviation of the losses. Any input with a reconstruction error larger than the threshold was deemed to be an outlier.

Table 1: Table showing the layers used in the autoencoder model

	Layer	Type
Encoder	1	Linear: input=1024, output=32
	2	ReLU
	3	Linear: input=32, output=16
	4	ReLU
	5	Linear: input=16, output=16
	6	ReLU
Decoder	7	Linear: input=16, output=16
	8	ReLU
	9	Linear: input=16, output=32
	10	ReLU
	11	Linear: input=32, output=1024
	12	Sigmoid

The model uses the AdamW optimizer for training. Adam optimizer is one of the most popular and efficient optimizers. It gives good results and converges to a local minimum relatively quickly. AdamW is a variation of that optimizer that incorporates weight decay. Weight decay helps models to be more generalizable as it is a regularization technique. The parameters for the AdamW optimizer are as follows: betas=(0.9, 0.999), eps=1e-08, lr=0.01, weight decay=0.01.

For the training of the autoencoder, the first dataset is used. As this dataset does not contain any labels but has sufficiently large samples, it is perfect for training an autoencoder. Autoencoder does not need labels during training and it is an unsupervised model. With a large number of samples, the chances of the model learning the underlying distribution is high.

A batch size of 64 is used for the autoencoder. This value is large enough to it giving solid results, while still allowing for the usage of mini-batch gradient descent. The model is trained for 20 epochs. The training was done in Google Colab using a GPU.

3.2 Classifier

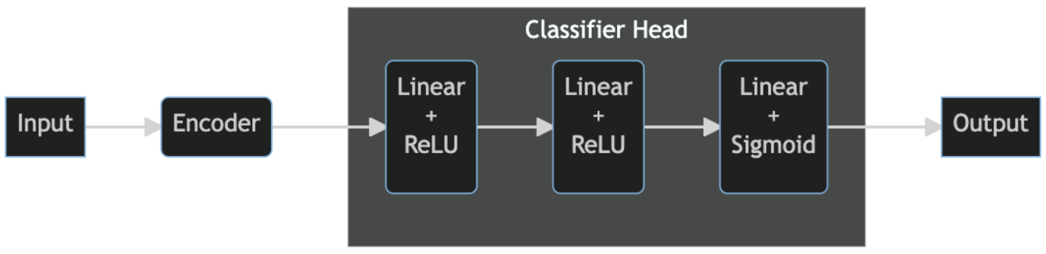

Figure 2: High-level overview of the classifier

The classifier consists of the same encoder which was used and trained in the autoencoder with 3 dense layers on top. A high-level representation of the model can be seen in Figure 2. All linear layers have the same parameters except, the first and last layers which have different input sizes and output sizes, respectively. The exact description of these layers can be seen in Table 2.

For training the classifier, Cross-Entropy Loss is used, as the dataset consists of more than

Table 2: Table showing the layers used in the classifier model

Layer	**Type**
1	Pre-trained Encoder
2	Linear: input=16, output=32
3	ReLU
4	Linear: input=32, output=32
5	ReLU
6	Linear: input=32, output=6
7	Softmax

2 classes and the labels are one-hot encoded.

For the optimizer, AdamW is used for the classifier as well. The parameters used are: betas=$(0.9, 0.999)$, eps=1e-08, lr=0.001, weight decay=0.01.

For the training of the classifier, the second dataset is used. As this dataset does not contain any outliers and all the samples are labeled. Additionally, the fact that this dataset is quite small is overcome by using the pre-trained encoder to extract the embeddings from the inputs.

To train the classifier, a batch size of 32 is used. Using the same batch size as was used for training the autoencoder could work, except that the classifier is trained on smaller datasets. As such, using a smaller batch size seems more appropriate. The classifier is trained for 30 epochs, with an evaluation step every 5 epochs. The entire training was done in Google Colab using a GPU.

4 Experiments and results

4.1 Evaluation Metrics

The pairwise metrics that are used for evaluating both the outlier detection and the classification task are as follows:

- Precision = $\frac{TruePositives}{TruePositives+FalsePositives}$, measures how many of the inputs that are labeled as one class are actually members of that class. Outputs values between 0 and 1.
- Recall = $\frac{TruePositives}{TruePositives+FalseNegatives}$, measures how many of the images that are members of one class are actually labeled as that class. Outputs values between 0 and 1.
- F1-Score = $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$, is the weighted harmonic mean of precision and recall and measures the balance between precision and recall. Outputs values between 0 and 1.
- Accuracy = $\frac{TruePositives+TrueNegatives}{TruePositives+TrueNegatives+FalsePositives+FalseNegatives}$, measures how well the model correctly classifies the samples provided. Outputs values between 0 and 1.

For all metrics, a value closer to 1 means that the model is performing well, while a value closer to 0 means the opposite.

4.2 Outlier Detection

As previously stated, the autoencoder is trained using the MSE loss as the reconstruction loss. From Figure 3, it can be seen that training loss continues to decrease until around epoch 20. The training loss drops rapidly until epoch 2 and then slows down. Towards the end of the training, there is little to no improvement in the training loss. Finally, the MSE ended up at around 0.015.

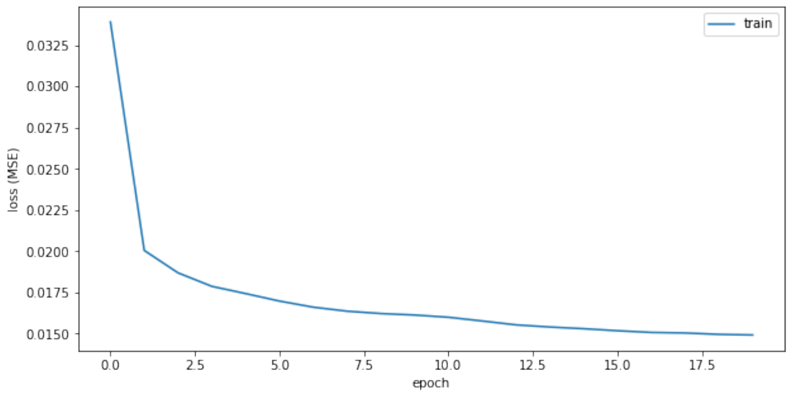

Figure 3: Plot showing MSE Loss against number of epochs.

Table 3: Evaluation results of outlier detection. 0 is in-distribution samples, 1 is out of distribution samples.

Dataset		precision	recall	f1-score	support
	0	0.98	0.98	0.98	1000
	1	0.60	0.56	0.58	52
1	accuracy			0.96	1052
	macro avg	0.79	0.77	0.78	1052
	micro avg	0.96	0.96	0.96	1052
	0	0.98	0.98	0.98	1000
	1	0.60	0.52	0.56	52
2	accuracy			0.96	1052
	macro avg	0.79	0.75	0.77	1052
	micro avg	0.96	0.96	0.96	1052

After training the autoencoder, the mean error and the standard deviation of error were calculated. Then the threshold was calculated as mean + 3 standard deviations. The final threshold value was 0.04. Any input which had a reconstruction error larger than 0.04 was deemed as an outlier. The evaluation results of this approach can be seen in Table 3. As it can be observed this approach works decently well and it achieves an accuracy of 0.96 for both test tests.

4.3 Classification

The labels for every image in the test set are predicted using the classification model explained above. One thing to note is that, because the outlier detection method is not 100% accurate, there are still samples left in the dataset that are outliers. Thus, to correctly calculate the classification metrics, a classification head with 6 outputs is used. The first 5 outputs represent the labels in the dataset and the last output represents the outliers in the dataset. However, as the classifier was trained on a dataset with no outliers, it will never output a sample as the 6th class. This 6th class is only there to use the evaluation metrics correctly.

In Figure 4 the Cross-Entropy Loss for the classifier is plotted against the epochs. In Figure 5 the accuracy score is plotted against the epochs. While the loss is not fantastic, there is only a small difference between the value achieved in the validation set and the training set. For the accuracy, both score very well. For the training set, the MSE ended at 1.1, while the accuracy score ended at 0.95. For the validation set, the MSE ended at 1.18, while the accuracy score ended at 0.92.

From Table 4, it can be observed that this approach achieves decent performance in the classification task. The overall accuracy of the results is good with 0.9 and 0.93 for the

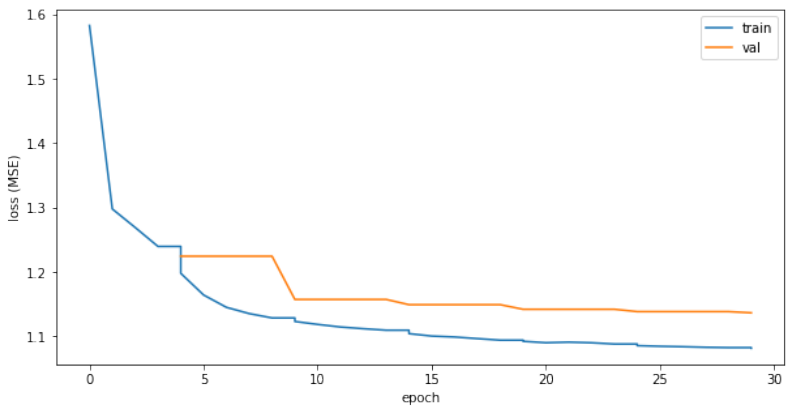

Figure 4: Plot showing the Cross Entropy Loss against the epochs for the classifier.

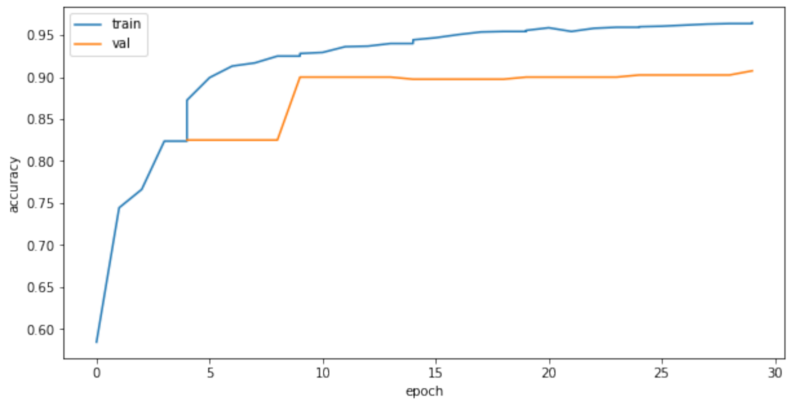

Figure 5: Plot showing the accuracy score against the epochs for the classifier.

datasets. In addition to accuracy, F1 scores are also high for all classes with almost all above 0.9, which means that the model has a balanced performance for all classes. Even though these results are not perfect, with the little data that was provided it is still impressive.

5 Conclusion

In summary, an autoencoder is trained on a large, unlabeled dataset to learn the underlying distribution of the dataset and to extract the relevant features. This autoencoder is then used to remove noise from the noisy datasets. Finally, a classifier is trained on top of the pre-trained encoder from the autoencoder and it is used to classify the data from the noise-free datasets.

For the outlier detection task, the autoencoder achieves an accuracy of 0.96 for both test sets. For the classification tasks, after the removal of outliers detected by the autoencoder, our model achieves an accuracy of 0.9 and 0.93 for test set 1 and 2 respectively.

Even though these results are not close to state-of-the-art results that are achieved in similar tasks, our approach manages to get great performance given a small number of samples to train.

Table 4: Evaluation results of the classification task.

Dataset	Class	Precision	Recall	F1-Score	Accuracy
1	0	0.92	1.00	0.96	0.90
	1	0.90	0.84	0.87	
	2	0.85	0.89	0.87	
	3	0.94	0.96	0.95	
	4	0.89	0.91	0.90	
2	0	0.93	0.99	0.96	0.93
	1	0.92	0.91	0.92	
	2	0.92	0.92	0.92	
	3	0.94	0.98	0.96	
	4	0.94	0.97	0.96	

6 Discussion

One improvement that can be made to the outlier detection method would be to use the 3rd dataset as a training set to determine whether an input is an outlier or not. As it stands now, we are using a static threshold value to determine outliers. However, training another model could help us determine a threshold that is suitable for the dataset at hand. Another further improvement could be made by using CNNs instead of regular feed-forward neural networks as the inputs are images and a CNN could extract more information about the underlying distribution.

Another big performance improvement can be achieved using transfer learning. Transfer learning works best for well-studied tasks that have been tackled by bigger teams who have access to larger datasets and models. These pre-trained models that are trained on larger datasets are typically more generalizable compared to specialized models. Thus, by using these pre-trained models to bootstrap the training of new models, the performance of new models can be improved. Since image classification is already a very widely explored topic within machine learning, especially using the given example dataset, one could apply transfer learning to do the classification part of this assignment.

A large pre-trained model that is trained on a similar image classification task can be used as the classification model in our task. By replacing the classification head of the pre-trained model with a suitable one for our dataset, we can fine-tune it to our specific dataset to improve the performance. Given the autoencoder's functionality of removing the noise from the slightly out-of-distribution datasets, using a pre-trained model to classify the images would work quite well.

Since the assignment specified that the model should be able to work for any image dataset, transfer learning is clearly not a valid strategy, as a classifier that is trained on a particular dataset will not be able to accurately classify a completely different dataset. Another situation where using transfer learning would be detrimental to performance is if the pre-trained model was trained on a dataset that is not relevant to the dataset fine-tuning would be done. The pre-trained weights could make it harder for the model to converge compared to starting with randomly initialized weights. The last remark about pre-trained models is that they are big and complex models and avoiding over-fitting is a hard task, especially with small datasets used for fine-tuning.