Exploring Denoising Autoencoders and Analyzing Their Efficacy for Data Clustering on the MNIST Fashion Dataset

Anthony Wong

CS6220 Big Data Systems & Analytics

1. Introduction

For this project, I aimed to find out how effective autoencoders trained for denoising could be in producing embeddings in their latent representation layer such that the data could properly be clustered with high accuracy to the number of target classes in the dataset. Autoencoders are a class of neural networks used in unsupervised machine learning, whether that be for dimensionality reduction, feature learning, or denoising. Generally speaking, they have two components: an encoder and a decoder. The encoder maps the input data into a lower-dimensional representation, called a latent space. This latent space ideally captures important features of the data. The decoder then takes this encoding from the latent space and reconstructs the original data.

The dataset I used was the infamous MNIST Fashion dataset from Tensorflow, which consists of 60000 train and 10000 test samples of greyscale 28x28 images that fall into the following categories:

Index	Category
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

With this dataset, I ran multiple experiments:

1. Added noise to the data, trained a denoising autoencoder (the denoising autoencoder was from Tensorflow and can be found here) on the noisy data, encoded the noisy test samples, used KMeans to cluster the embeddings
2. The same as Experiment 1 except with different hyperparameters
3. The same as Experiment 1 except with a different model altogether (This model focused on using Dense layers as opposed to Conv2D layers. The inspiration can be found here)

4. The same as Experiment 1 except that noise was not added to the data and a different autoencoder was used (this model can also be found on Tensorflow's website)

These are the model summaries (Split into Encoder and Decoder):

- Experiment 1 & 2:

Encoder Summary:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 16)        160

 conv2d_1 (Conv2D)           (None, 7, 7, 8)           1160

=================================================================
Total params: 1320 (5.16 KB)
Trainable params: 1320 (5.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Decoder Summary:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_transpose (Conv2DTr  (None, 14, 14, 8)         584
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 28, 28, 16)        1168
 Transpose)

 conv2d_2 (Conv2D)           (None, 28, 28, 1)         145

=================================================================
Total params: 1897 (7.41 KB)
Trainable params: 1897 (7.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- Experiment 3:

Encoder Summary:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1500)              1177500

 dense_1 (Dense)             (None, 1000)              1501000
```

```
 dense_2 (Dense)             (None, 500)               500500

 dense_3 (Dense)             (None, 10)                5010

=================================================================
Total params: 3184010 (12.15 MB)
Trainable params: 3184010 (12.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Decoder Summary:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 500)               5500

 dense_5 (Dense)             (None, 1000)              501000

 dense_6 (Dense)             (None, 1500)              1501500

 dense_7 (Dense)             (None, 784)               1176784

=================================================================
Total params: 3184784 (12.15 MB)
Trainable params: 3184784 (12.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- Experiment 4:

Encoder Summary:

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_8 (Dense)             (None, 64)                50240

=================================================================
Total params: 50240 (196.25 KB)
Trainable params: 50240 (196.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Decoder Summary:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 784)               50960
```

```
 reshape (Reshape)           (None, 28, 28)            0

=================================================================
Total params: 50960 (199.06 KB)
Trainable params: 50960 (199.06 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

As previously mentioned, KMeans was used to cluster the embeddings from the latent space. The number of clusters was set to 10, just as the number of target classes in the MNIST Fashion dataset shown in the table above. I also used the Silhouette Coefficient and Davies-Bouldin Index metrics as part of the cluster analysis. The Silhouette Coefficient measures how similar each data point in one cluster is to the data points in the same cluster compared to the nearest neighboring cluster, quantifying the separation between clusters and the cohesion within a cluster. Its value ranges from -1 to 1, where a higher number in that range represents good clustering and good cluster separation. The Davies-Bouldin Index measures similarity between each cluster and its most similar cluster, quantifying the separation and cohesion of clusters. The lower this score is, the better the data is clustered.

2. Experiments/Results

As previously mentioned, for each experiment, data was preprocessed according to what was needed for the experiment and used to train the corresponding autoencoder. Embeddings from the latent space were taken by passing testing data through the trained autoencoder and clustered using KMeans. For each experiment, we have the clusters returned from KMeans, the top three classes within those clusters, and the corresponding percentages that represent the proportion of each cluster they comprise.

Experiment 1:

After training the denoising autoencoder and clustering the embeddings, I had the following clusterings:

Cluster	Clothing Item	Percentage
0	Sandal	61.49%
	Bag	8.36%
	Shirt	7.46%
1	Coat	37.0%
	Pullover	35.87%
	Shirt	19.81%
2	Bag	96.02%
	T-shirt/Top	1.41%
	Coat	1.17%
3	T-shirt/Top	46.07%
	Dress	19.96%

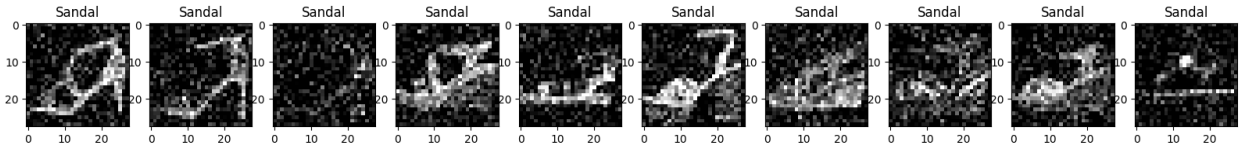
	Shirt	15.33%
4	Trouser	59.39%
	Dress	35.7%
	Coat	1.66%
5	Sneaker	60.26%
	Sandal	35.47%
	Bag	2.43%
6	Ankle Boot	90.83%
	Sandal	8.72%
	Sneaker	0.45%
7	Ankle Boot	69.74%
	Sneaker	21.69%
	Sandal	8.31%
8	Shirt	27.79%
	Pullover	25.81%
	T-shirt/Top	19.28%
9	Bag	91.99%
	Shirt	3.88%
	Pullover	1.21%

Table 1: Experiment 1 Clusters

This clustering had a Silhouette coefficient of 0.153 and a Davies-Bouldin index of 1.93. This means that our clustering is not great, with clusters having overlap and many points being wrongly assigned. This is evident as not every class is represented when looking at the most common class per cluster. Here we analyze examples of good and bad clustering:

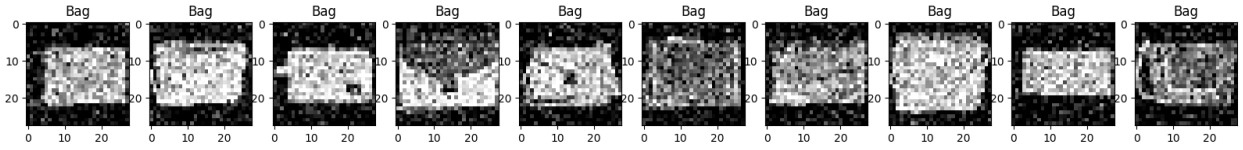
Good:

Though there are issues with the clustering, there are some examples of reasonable clustering. One example of this is cluster 0 with a reasonable percentage going to Sandal:



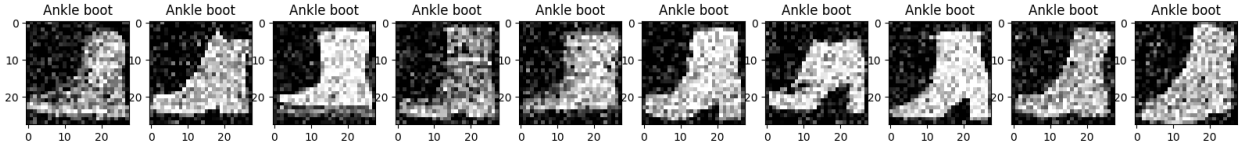
With this, we can see that this cluster has captured this high heel structure and has properly put these samples together. It is also evident that these samples are darker in color, which has likely helped the clustering. The downside to this is likely that other darker samples of other classes are more likely to slip in.

We also see that for cluster 9, Bag comprises over 90% of the cluster:



These bags are generally quite rectangular and therefore likely to be very distinct from the other fashion items. This is likely why they are able to be distinct from them and as a result have a high percentage.

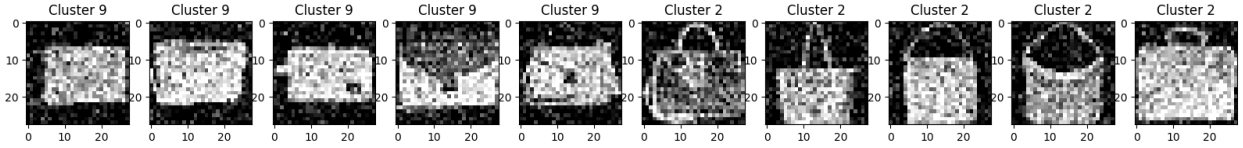
Lastly, we have cluster 6 where Ankle Boot comprises over 90% of the cluster as well:



This likely clusters well since all these samples are clear despite the noise and have a unique shape and height that would not exist for other footwear. The result is a high cluster percentage.

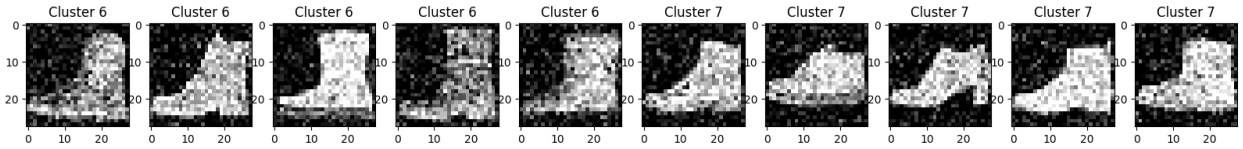
Bad:

When looking at examples of poor clustering we can see that Bag has the highest percentage in two clusters. When visualizing them, we can see why:



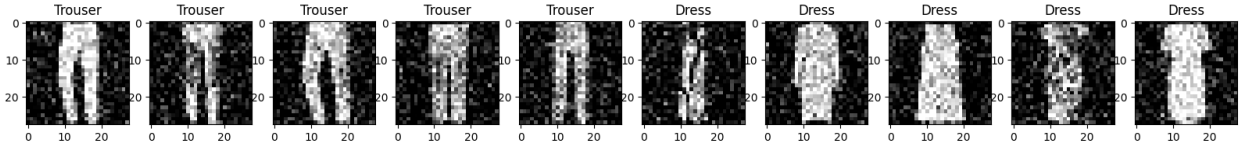
Visually, we see that the bags in cluster 2 are distinct from bags in cluster 9 because the former has visible handles. It is likely that our model learned those differences, and since they were distinct enough, resulted in them clustering differently.

We also see that Ankle Boot is also the highest percentage in two clusters, 6 and 7:



Here, we see slight differences between ankle boots of cluster 6 and cluster 7, with cluster 6 having boots that go up slightly higher on the ankle/shin. I was surprised to find that this small feature was noticeable enough to turn them into separate clusters. The lower height of the boot is likely why there is a fair percentage of Sneaker in the cluster as well.

Lastly, I wanted to see why in cluster 4, Trousers and Dresses had significant percentages of that cluster:



Given the results from different types of bags, I would have thought that the model would be able to distinguish the dark space between the pant legs. However, this was not the case and may require a better trained model or a different model architecture to recognize.

Experiment 2:

Following the results from Experiment 1, I wanted to tune the hyperparameters such that I could ensure the model converges more precisely. I lowered the learning rate from .001 to .0005, lowered the batch size from 32 to 16, and raised the epochs from 10 to 20. The goal of this was to create a more stable model that would have more time to learn. These were the results:

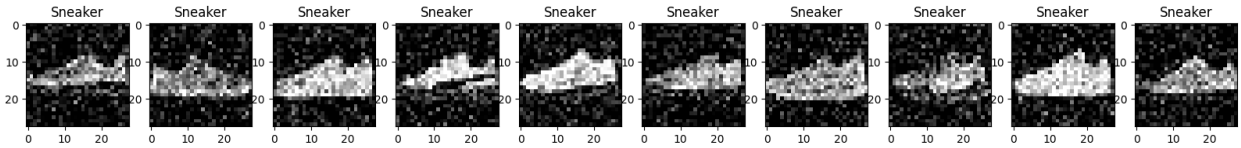
Cluster	Clothing Item	Percentage
0	Coat	38.98%
	Pullover	35.26%
	Shirt	18.7%
1	Sneaker	71.22%
	Sandal	22.69%
	Bag	3.51%
2	T-shirt/Top	46.0%
	Dress	22.41%
	Shirt	15.04%
3	Bag	95.15%
	Pullover	1.21%
	Coat	1.21%
4	Bag	92.08%
	Shirt	3.96%
	Pullover	1.32%
5	Trouser	60.91%
	Dress	34.13%
	Coat	1.89%
6	Shirt	27.37%
	Pullover	26.15%
	T-shirt/Top	19.11%
7	Ankle Boot	89.78%
	Sandal	9.41%
	Sneaker	0.41%
8	Sandal	45.72%
	Shirt	10.57%
	T-shirt/Top	8.41%
9	Ankle Boot	68.04%
	Sneaker	21.54%
	Sandal	9.33%

Table 2: Experiment 2 Clusters

This clustering had a Silhouette coefficient of 0.141 and a Davies-Bouldin index of 1.92. By comparison, this autoencoder with new hyperparameters performed generally the same. We will again analyze examples of good and bad clustering.

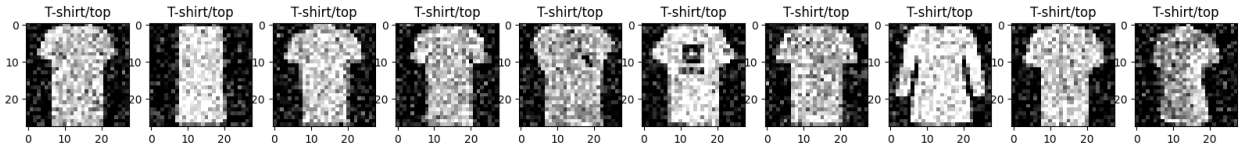
Good:

This time, we see a reasonable Sneaker percentage in cluster 1:



The low profile of the shoes along with how the samples are a solid color throughout is likely why the percentage was reasonable. The solid color is where they likely differ from sandals though evidently this isn't perfect, as Sandal has the next highest percentage.

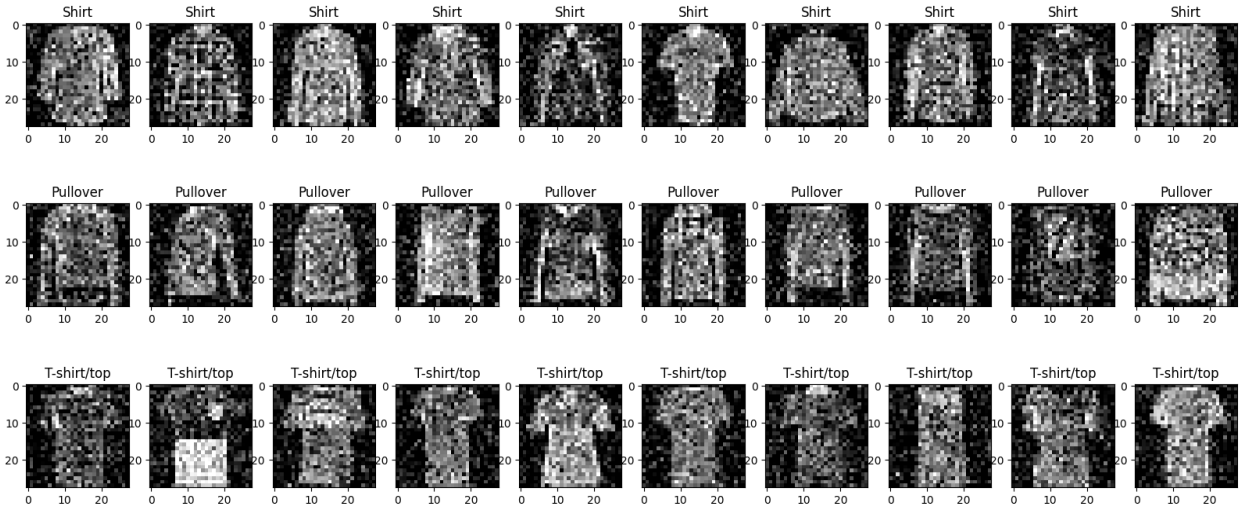
Cluster 2 where the top percentage is T-shirt/Top is more of a mixed bag but has some okay clustering:



It seems that our autoencoder + KMeans system can distinguish sleeves that are not full length to a reasonable extent. The fashion item that is most likely to have this feature are dresses, which is likely why Dress has the next highest percentage.

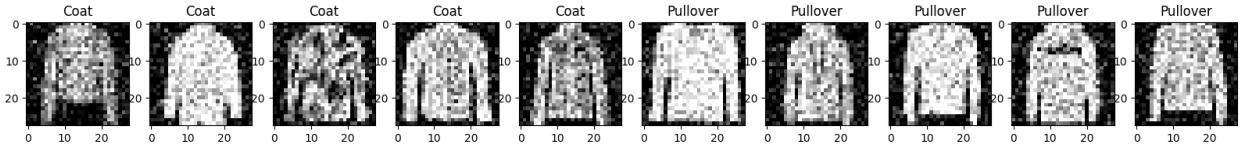
Bad:

A clear poor example of clustering is with cluster 6 where all of the top three have reasonable percentages:



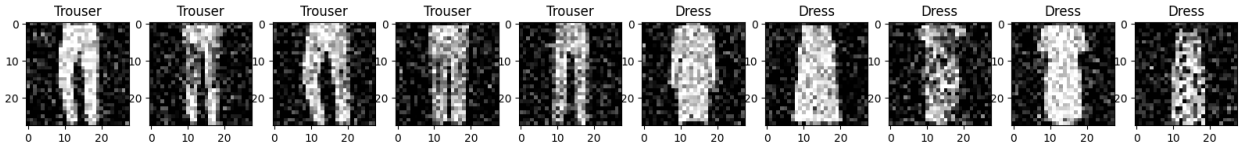
This is likely as bad as it is mainly due to the darker nature of these pictures along with the added noise. Pullovers and shirts already are pretty difficult to distinguish at this pixel granularity, to the point where I'm not sure if the human eye could either. For T-shirts/tops the darkness makes it so that it's likely harder for the lack of long sleeves to be recognized.

Next, looking at cluster 0 with pullovers and coats, we can see they have very similar percentages:



Though these samples are not dark and are well defined, it is similar to the previous analysis that at this pixel granularity, they look very similar and thus are hard to distinguish.

Part of my goal here was to see if I could further optimize hyperparameters such that we could distinguish trousers from dresses:



It would seem that this did not work. We will see if a different architecture for the autoencoder could produce anything different.

Experiment 3:

Inspired by existing architectures I found on Kaggle that utilized Dense layers instead of Conv2D layers, I wanted to see how if this type of autoencoder would perform any differently.

Cluster	Clothing Item	Percentage
0	Ankle Boot	73.11%
	Sandal	15.8%
	Sneaker	10.99%
1	Coat	46.8%
	Pullover	26.48%
	Shirt	22.58%
2	Sandal	27.18%
	Ankle Boot	17.56%
	Shirt	16.59%
3	T-shirt/Top	75.42%
	Shirt	19.81%
	Dress	2.33%
4	Pullover	51.71%
	Coat	23.29%
	Shirt	18.38%

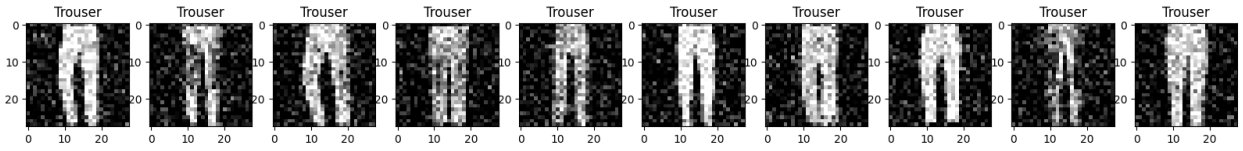
5	Sneaker	59.08%
	Sandal	35.28%
	Ankle Boot	3.34%
6	Dress	61.77%
	T-shirt/Top	10.35%
	Coat	7.89%
7	Bag	96.77%
	T-shirt/Top	1.08%
	Coat	0.86%
8	Trouser	96.15%
	Coat	1.1%
	Dress	0.99%
9	Bag	84.82%
	Shirt	7.69%
	T-shirt/Top	3.04%

Table 3: Experiment 3 Clusters

This clustering had a Silhouette coefficient of 0.205 and a Davies-Bouldin index of 1.54. Though these numbers are still not representative of amazing clustering, they are somewhat better than both the previous experiments.

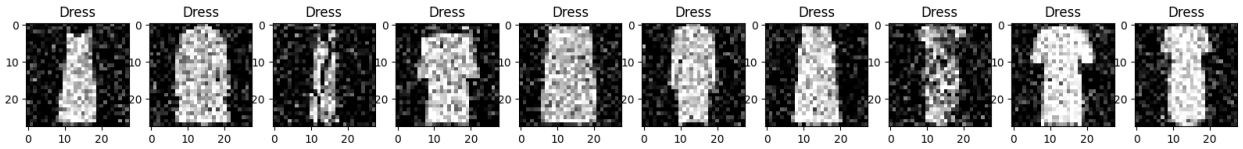
Good:

One new astounding result is cluster 8 with Trouser being over 96%:



Though Conv2D should be better for images, since it is able to recognize spatial patterns, it seems for this specific case that Dense works better which is incredibly surprising. It could potentially due simply to the massive number of parameters we trained in comparison.

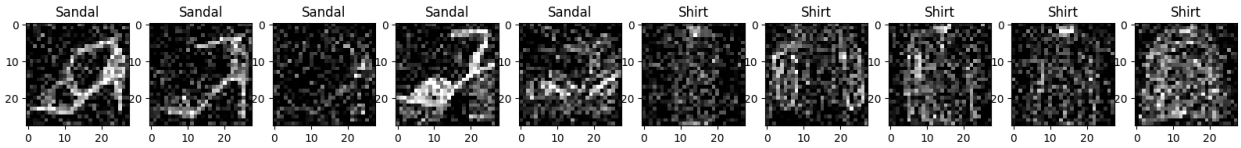
Additionally, for the first time we see Dress at the top of a cluster:



Though I'm not sure, like above, it may be the large amount of parameters that allowed the dress feature (widening of the clothing as you go downwards) to be properly recognized.

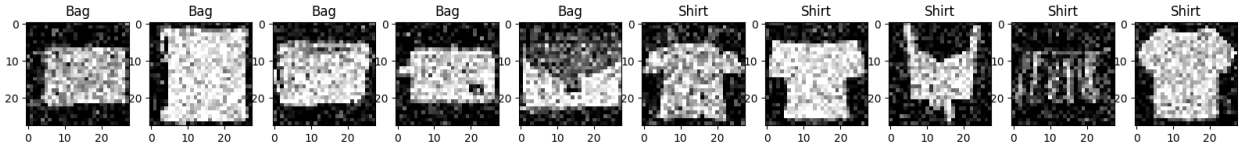
Bad:

With the Dense layers, we found that clustering Sandal performed worse. These are the samples for Sandal and Shirt:



I feel this is where the lack of spatial understanding of Dense layers shows. These same darkness level images are much better clustered in Experiment 1. When it is dark and there is spatial understanding, the understanding of what may seem to be disjoint structures in dark space may have meaning due to their locations in space. However, Dense layers do not have this, which is likely why these shirts ended up in the same cluster. The extra noise only makes this worse.

Lastly, I was interested why in cluster 9, bags and shirts were both there together:



We see this is because these shirts are generally not standard and have more boxy of a shape, and therefore we see them together here.

Experiment 4:

With all of this investigation done on MNIST Fashion with added noise, I wanted to see the extent of the impact noise has on clustering. Here, we use another autoencoder given by Tensorflow and do not add noise to input data.

Cluster	Clothing Item	Percentage
0	Sandal	36.42%
	Shirt	11.61%
	T-shirt/Top	10.33%
1	Sneaker	73.58%
	Sandal	23.05%
	Ankle Boot	1.89%
2	Dress	49.32%
	Coat	21.49%
	T-shirt/Top	9.95%
3	Trouser	79.55%
	Dress	19.51%
	Coat	0.38%
4	Coat	42.83%
	Pullover	35.54%
	Shirt	19.06%
5	T-shirt/Top	78.34%
	Shirt	15.43%
	Pullover	4.3%
6	Ankle Boot	64.89%

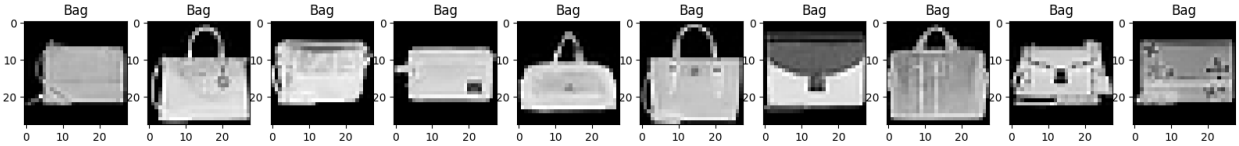
	Sneaker	23.98%
	Sandal	10.8%
7	Ankle Boot	78.34%
	Sandal	21.34%
	Sneaker	0.44%
8	Shirt	22.7%
	Pullover	20.13%
	Dress	15.25%
9	Bag	84.1%
	Shirt	5.71%
	Pullover	4.21%

Table 4: Experiment 4 Clusters

This clustering had a Silhouette coefficient of 0.133 and a Davies-Bouldin index of 2.11. We see that even without noise, this methodology does not improve clustering performance by these metrics and may be slightly worse.

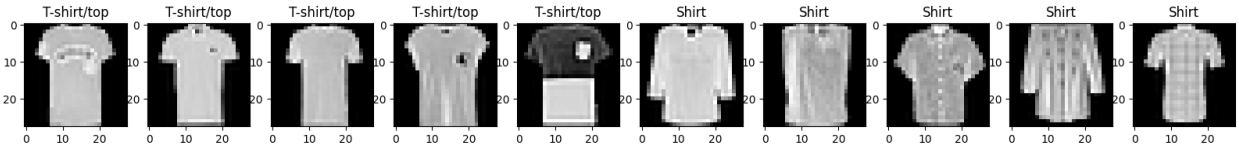
Good:

Interestingly, Bag is only top of this list in one cluster:



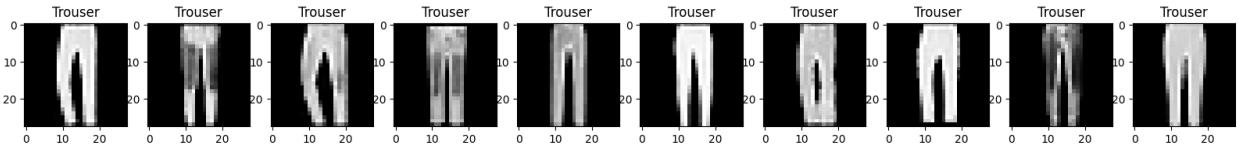
Here we see a mix of bags with visible handles and those without. Curiously, our model must have learned that this feature is not integral to classification. With noise, maybe this non-relationship was harder to determine.

Additionally, we see that we were able to cluster T-shirt/Top well:



Without noise, the length of the sleeves is easier to decipher, making this classification easier. The shirts in this cluster also have sleeves that do not run all the way down.

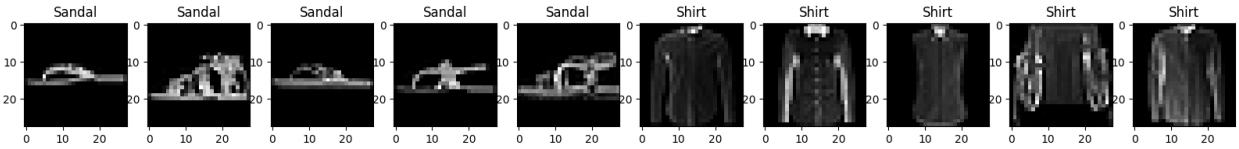
Lastly, we see that we were also able to cluster Trouser well:



Without noise, it is likely easier to decipher specifically that there is a gap in between the pant legs, and therefore we more consistently can classify this.

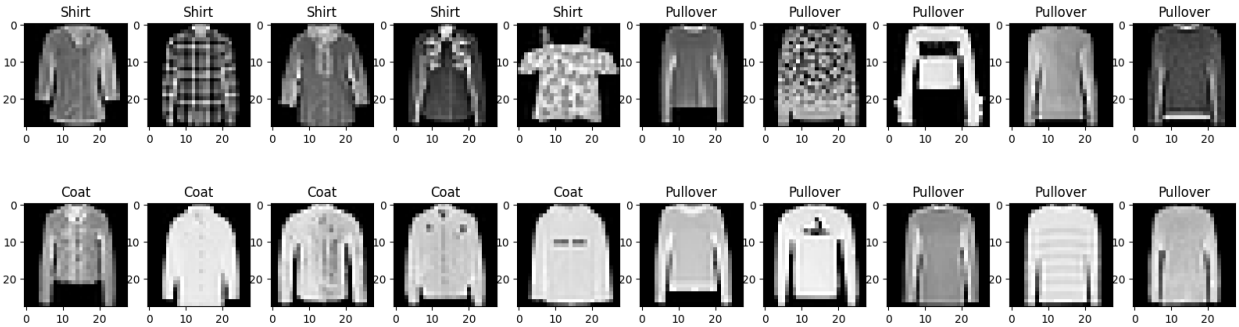
Bad:

Cluster 0 had some odd clustering with the top two being Sandal and Shirt:



This likely represents the limits of this architecture/process and dataset. This also proves my previous statements. The darkness of some of these images often results in bad clustering since there may be "less" to observe. Though I hoped it would perform better than this, these are just figments of the dataset.

Lastly, there are still classes that do not cluster well:



This also potentially proves the limits of this architecture/process and dataset. Generally, the shapes are all similar. As previously stated, with this pixel granularity it is extremely hard to distinguish these. To do this you would likely need to include more detail and elements like color.

3. Conclusion

Overall, this project gave me a really good taste of autoencoders and how they work, especially since I was able to try out different architectures and play with "different" data. For this data clustering task specifically on MNIST Fashion, I felt that I hit some hard limit of how well the clusterings could be due mainly to the lack of pixel granularity from the 28x28 images and lack of color. No matter if noise was added to the images or not, and what architecture was used, there was always some area where there was poor performance. My questions coming out of this are whether there exists a better autoencoder architecture that can perform much better, since the ones I used may have been more simple, and if a different clustering algorithm such as GMM or DBScan could have altered the performance in any way. Regardless, in comparing my resulting statistics with others who had similar processes, it is evident that my results are somewhat comparable to theirs, as seen by this Kaggle page. I believe this shows that if there is performance/accuracy to be gained here, it may minimally require a different autoencoder architecture/deep learning technique and/or clustering algorithm.

Resources Used

For Tensorflow autoencoders: https://www.tensorflow.org/tutorials/generative/autoencoder

For Kaggle autoencoder: https://www.kaggle.com/code/shivamb/how-autoencoders-work-intro-and-usecases

For results comparison: https://www.kaggle.com/code/meshukarayamajhi/k-means-clustering-in-fashion-mnist/notebook