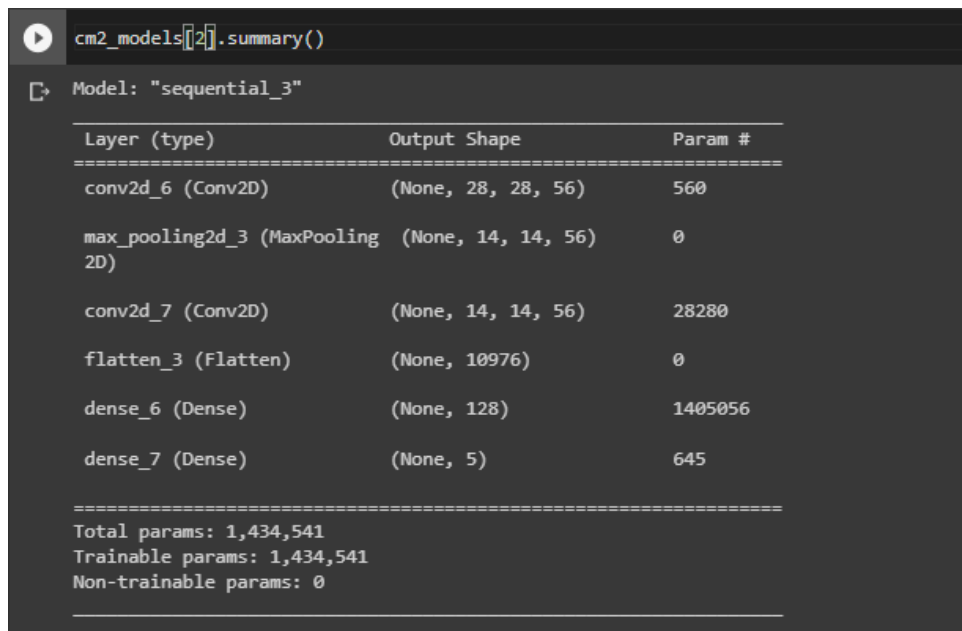


CM4: Encoding

PCA

Since we are going to apply PCA after the final dense layer of the neural network, we need to collect the output of the final layer and feed it into the PCA algorithm. This can be done by first identifying the layer by using the summary as shown in Figure 19.



```
cm2_models[2].summary()
```

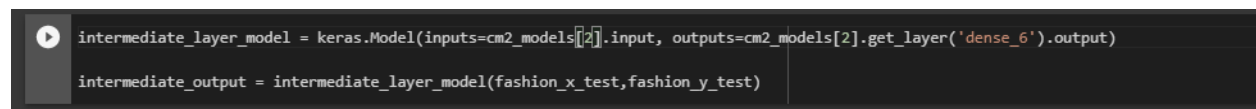
Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 28, 28, 56)	560
max_pooling2d_3 (MaxPooling 2D)	(None, 14, 14, 56)	0
conv2d_7 (Conv2D)	(None, 14, 14, 56)	28280
flatten_3 (Flatten)	(None, 10976)	0
dense_6 (Dense)	(None, 128)	1405056
dense_7 (Dense)	(None, 5)	645

=====
Total params: 1,434,541
Trainable params: 1,434,541
Non-trainable params: 0
=====

Figure 19: Network structure of model number 2

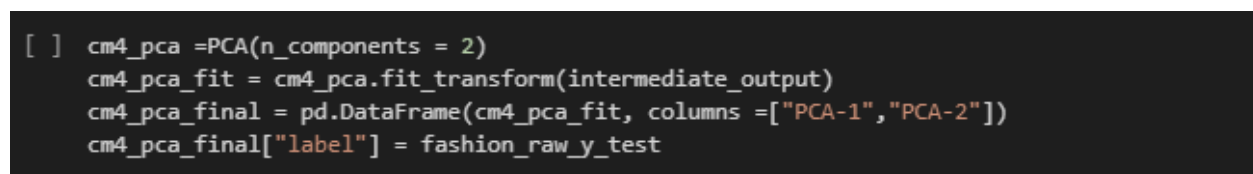
Dense 6 is the last dense layer and dense 7 is the softmax layer. Therefore, we will record the output of the dense_6 when we provide the test data as input. This is shown in the screenshot of the code in Figure 20:



```
intermediate_layer_model = keras.Model(inputs=cm2_models[2].input, outputs=cm2_models[2].get_layer('dense_6').output)
intermediate_output = intermediate_layer_model(fashion_x_test, fashion_y_test)
```

Figure 20: Capturing features from the last dense layer

The PCA algorithm was initialized using scikit learn as shown in Figure 21:



```
[ ] cm4_pca = PCA(n_components = 2)
cm4_pca_fit = cm4_pca.fit_transform(intermediate_output)
cm4_pca_final = pd.DataFrame(cm4_pca_fit, columns = ["PCA-1", "PCA-2"])
cm4_pca_final["label"] = fashion_raw_y_test
```

Figure 21: PCA algorithm

After this, the first two components of PCA were plotted using the original labels of the data as shown in Figure 22:

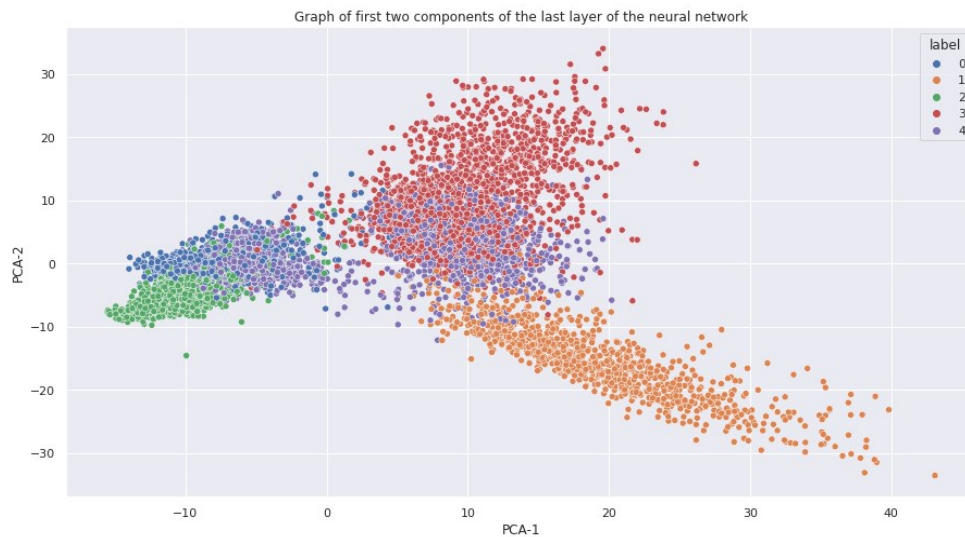


Figure 22: PCA plot with original labels

Using K-means

Now the K-means algorithm, from Scikit Learn, was used to form clusters. The initialization of the K-means is shown in Figure 23:

```
[ ] # initialize the k-means clustering algorithm
cm4_kmeans = KMeans(n_clusters=5)

# fit the data from the output of the last dense layer
cm4_kmeans.fit(cm4_pca_fit)

# check the number of labels
np.unique(cm4_kmeans.labels_)

array([0, 1, 2, 3, 4], dtype=int32)
```

Figure 23: Initialize K-means to be used with PCA

The graph generated by using the k-means clustering is shown in Figure 24:

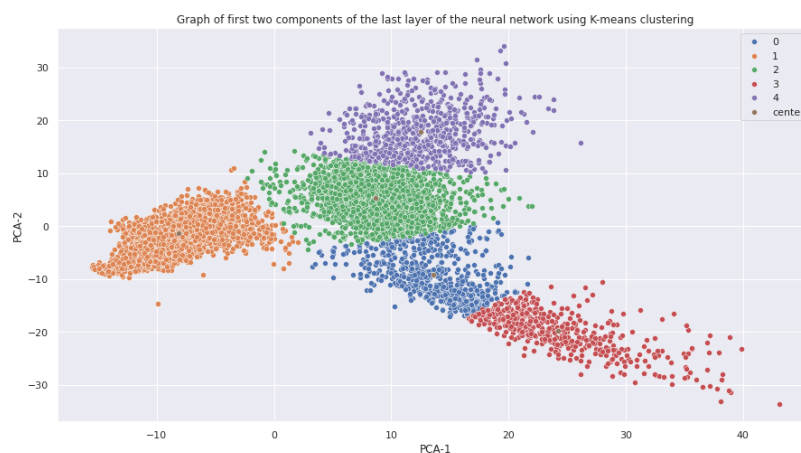


Figure 24: PCA using labels generated from K-means

Using DBSCAN

Now the same procedure that was implemented with K-means will be implemented with DBSCAN. First will initialize the DBSCAN algorithm from Scikit Learn which is shown in Figure 25:

```
[447] # initialize DBSCAN from scikit learn
      cm4_dbscan = DBSCAN(eps= 11, min_samples=13)

      # input the data from the last dense layer
      cm4_dbscan_final = cm4_dbscan.fit_predict(intermediate_output)

      # check the number of labels
      np.unique(cm4_dbscan.labels_)

      array([-1,  0,  1,  2,  3,  4])
```

Figure 25: Initialize DBSCAN

Eps of 11 and min_samples of 13 were determined via trial and error. It is possible to determine a better combination to achieve a better fit using Silhouette Scores.

DBSCAN generated 6 labels. -1 represents the noise in the data and the rest are the labels of the data. The graph generated is shown in Figure 26:

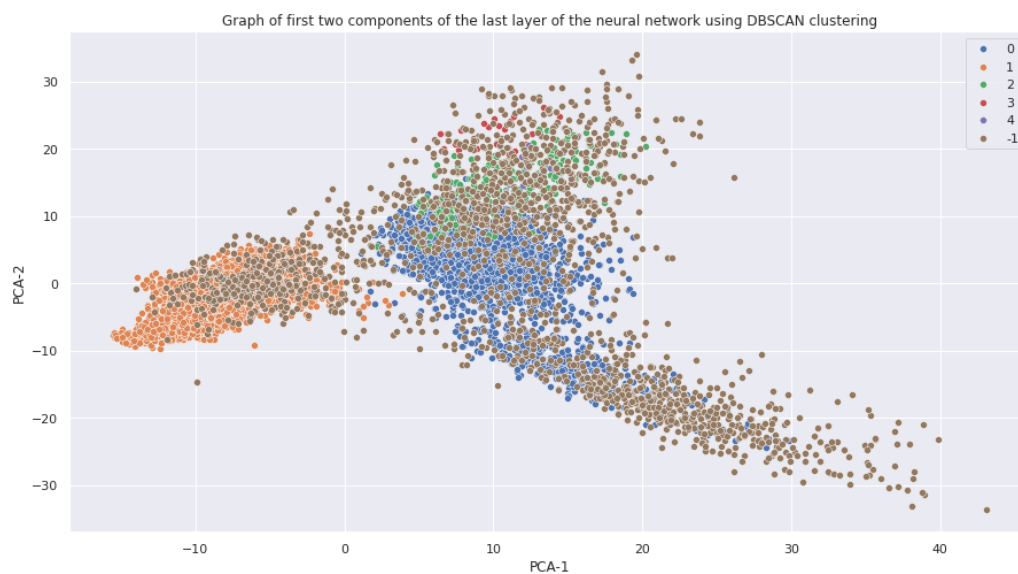


Figure 26: PCA using labels generated from DBSCAN

t-SNE

Afterwards, the t-SNE algorithm from Scikit Learn was initialized as shown in Figure 27 :

```
▼ t-SNE

[123] cm4_tsne = TSNE(n_components=2, verbose=1, random_state=27).fit_transform(intermediate_output, fashion_raw_y_test)
      cm4_tsne_final = pd.DataFrame(cm4_tsne, columns=["Comp-1", "Comp-2"])
```

Figure 27: Initialization of t-SNE

Then the graph of the two components of t-SNE were plotted using the original labels as shown in Figure 28:

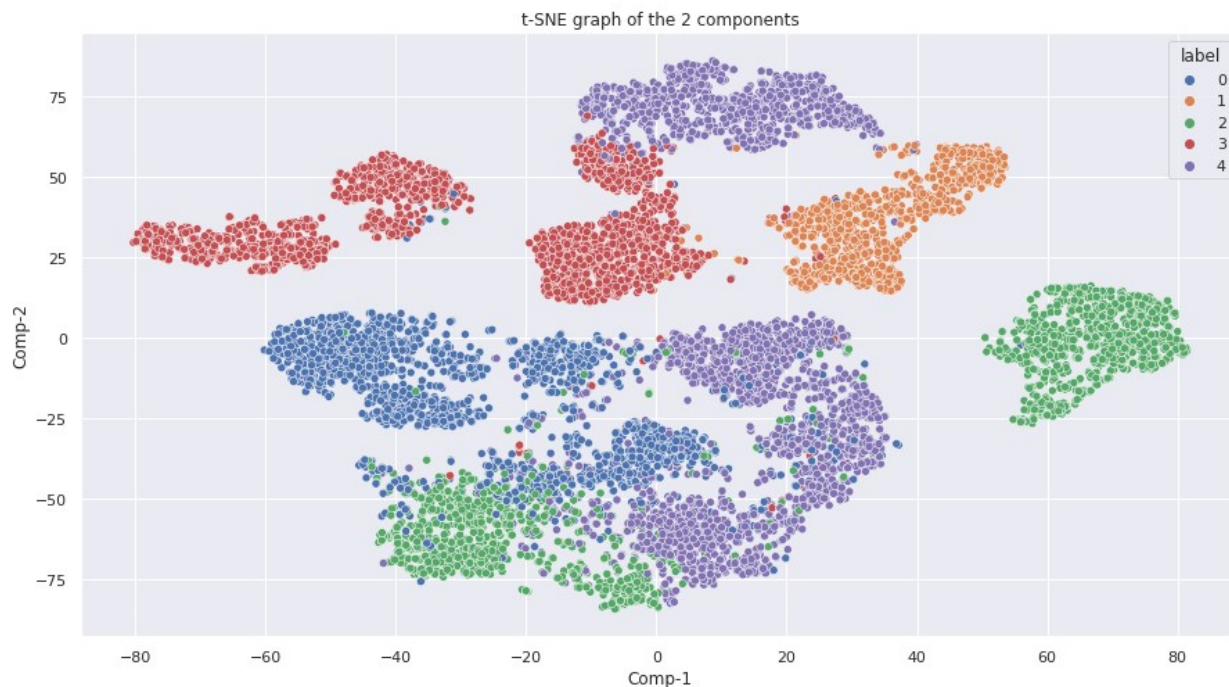


Figure 28: t-SNE plot of 2 components using original labels

Using K-means

The K-means algorithm had already generated the labels for the test data. Now the labels are applied on the t-SNE graph by using the code shown in Figure 30:

```
[477] kmeans_tsne_filtered_label0 = cm4_tsne[cm4_kmeans.labels_ == 0]
kmeans_tsne_filtered_label1 = cm4_tsne[cm4_kmeans.labels_ == 1]
kmeans_tsne_filtered_label2 = cm4_tsne[cm4_kmeans.labels_ == 2]
kmeans_tsne_filtered_label3 = cm4_tsne[cm4_kmeans.labels_ == 3]
kmeans_tsne_filtered_label4 = cm4_tsne[cm4_kmeans.labels_ == 4]
```

Figure 29: applying k-means on t-sne dataset

Afterwards, the graph was generated using the labels generated by k-means which is shown in Figure 30:



Figure 30: t-SNE plot of 2 components using labels generated by k-means

Using DBSCAN

The DBSCAN algorithm had already generated the labels for the test data. Now the labels are applied on the t-SNE graph by using the code in Figure 31:

```
dbscan_tsne_filtered_label0 = cm4_tsne[cm4_dbscan.labels_ == 0]
dbscan_tsne_filtered_label1 = cm4_tsne[cm4_dbscan.labels_ == 1]
dbscan_tsne_filtered_label2 = cm4_tsne[cm4_dbscan.labels_ == 2]
dbscan_tsne_filtered_label3 = cm4_tsne[cm4_dbscan.labels_ == 3]
dbscan_tsne_filtered_label4 = cm4_tsne[cm4_dbscan.labels_ == 4]
dbscan_tsne_filtered_label_1 = cm4_tsne[cm4_dbscan.labels_ == -1]
```

Figure 31: implementing labels in t-sne dataset

Afterwards, the graph was created using the labels generated by DBSCAN which is shown in Figure 32:

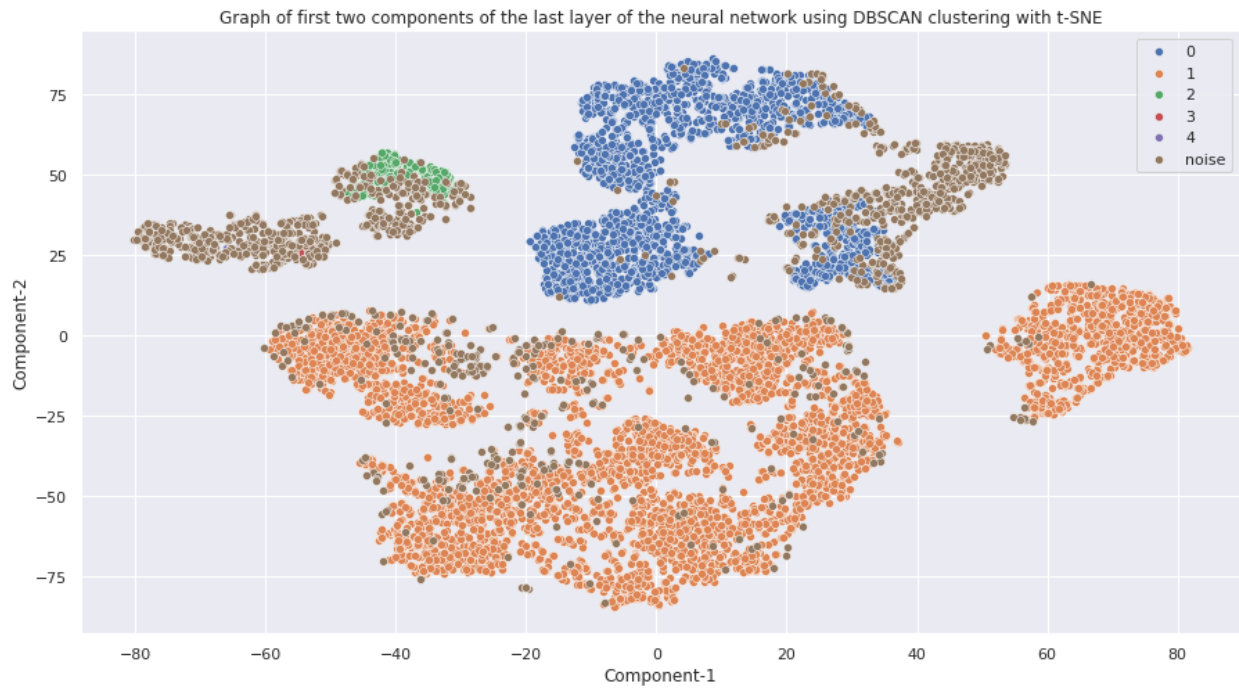


Figure 32: t-sne graph of 2 components with labels generated from DBSCAN

Results

Original Labels

Based on the original labels, the graphs generated by the PCA did not have clear separation among the clusters, but the t-SNE plot had relatively better separation. This can be since the data may be of non-linear dimensions and t-SNE is better suited for the dataset.

Furthermore, based on the original dataset and labels, if instances are chosen at random and plotted for each label and the images generated are shown in Figure 33:



Figure 33: Classification based on the original labels

Based on the aforementioned classification, the following conclusion can be reached:

S. No	Label	Classification
1	0	Shirts
2	1	Sandals
3	2	Pullover & trouser
4	3	Sneakers & bags
5	4	Coats, Dresses & Boots

Table 5: Labels according to the original labels of the dataset

The original labels have clear categories that are understandable by humans

K-means

In the case of PCA, there were some overlaps among the clusters. However, the overall separation was clear and apparent in the graph.

In the case of t-SNE, there was not much overlap among the clusters and the separation was evidently clear.

The label frequency generated by K-means is the following:

Label	Frequency	Percentage terms
0	5086	0.5
1	2086	0.2
2	950	0.095
3	933	0.093
4	945	0.094

Table 6: Frequency of labels that were generated by K-means

If the labels are used and the relation to the images are seen, the following plot is generated:



Figure 34: plot of the instances of the data using k-means

Based on Figure 34, K-means has found characteristics in the data that are different to the original labels of the dataset. For example, with the original labels, label 1 represented sandals, but in this case, it's a mixture of coats, trouser, boots and hand bags. Label 4 is mostly upper wear with some trousers.

DBSCAN

For both PCA and t-SNE, the labels generated by the DBSCAN algorithm were not able to generate clear separation. Furthermore, there is considerable noise generated by DBSCAN. Furthermore, there is considerable overlap among the clusters and the frequency of each cluster has greatly changed when compared to the original labels. The frequency of the individual clusters is the following:

Label	Frequency	Percentage terms
-1 (Noise)	1763	0.17
0	2348	0.23
1	5612	0.56
2	241	0.024
3	21	0.002
4	15	0.001

Table 7: Frequency of labels generated by DBSCAN

Compared to K-means, DBSCAN did not perform as well in creating good separation in clusters. This could be due to the fact the DBSCAN requires hyperparameter tuning and K-means does not.

If the labels are applied on the images, a plot is generated which is shown in Figure 35:



Figure 35: images with labels generated from DBSCAN

It can be observed, that DBSCAN was able to find a feature in the data that is different to the original labels. Although label 3 and 4 don't have many instances, they basically contain every item according to

the original labels. Label 2 mostly contain shoes which is a combination of boots, sneakers, and sandals. 0 & 1 also have mixture of item. As most of the instances are classified as noise, we need to adjust the hyperparameters so that noise could be reduced and more instances can be part of the labels.

Other Approaches on the Encoding

Mini batch K-means algorithm is a variant of the K-means algorithm. It uses mini-batches to reduce the computational time required to do the clustering. Although mini-batch k-means should result in the same output as the k-means algorithm, it typically results in different clustering. I wanted to see the difference on the fashion MNIST dataset. Mini batch K-means was applied on the output of the last dense layer as shown in Figure 36:

```
[516] cm4_minik = MiniBatchKMeans(n_clusters=5)
      cm4_minik.fit(intermediate_output)
      cm4_minik.labels_

      array([3, 0, 4, ..., 1, 4, 1], dtype=int32)

[517] minik_filtered_label0 = cm4_tsne[cm4_minik.labels_ == 0]
      minik_filtered_label1 = cm4_tsne[cm4_minik.labels_ == 1]
      minik_filtered_label2 = cm4_tsne[cm4_minik.labels_ == 2]
      minik_filtered_label3 = cm4_tsne[cm4_minik.labels_ == 3]
      minik_filtered_label4 = cm4_tsne[cm4_minik.labels_ == 4]
```

Figure 36: Mini batch k-mean labels applied to the t-sne data

Afterwards, a graph was generated using the t-SNE data with the labels generated by mini batch K-mean as shown in Figure 37:

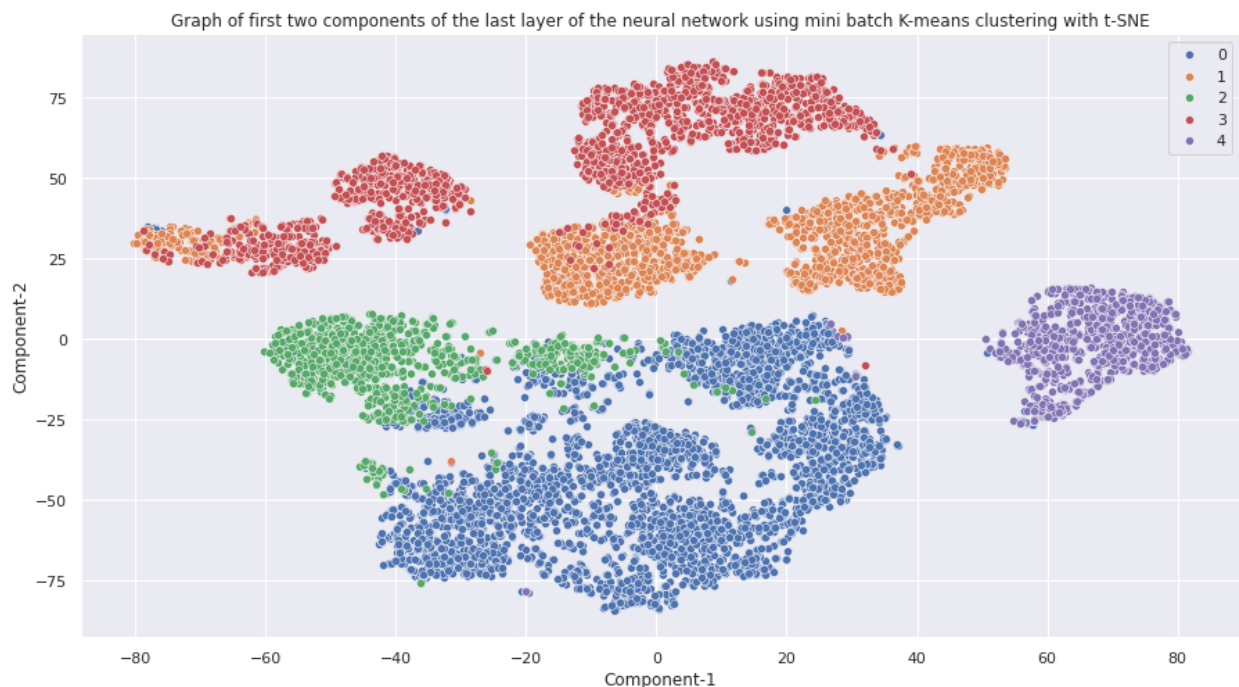


Figure 37: t-SNE plot with labels generated from mini batch k-means

The clusters generated by the mini batch k-means clustering algorithm are different to that approached by k-means.

Bibliography

Arvai, Kevin. K-Means Clustering in Python: A Practical Guide. [Online] <https://realpython.com/k-means-clustering-python/>.

Brownlee, Jason. 2019. A Gentle Introduction to Padding and Stride for Convolutional Neural Networks. *Machine Learning Mastery*. [Online] 2019. <https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/>.

Kumar, Veer. 2021. Tutorial for DBSCAN Clustering in Python Sklearn. [Online] 2021. <https://machinelearningknowledge.ai/tutorial-for-dbscan-clustering-in-python-sklearn/>.

Preda, Gabriel. 2022. CNN with Tensorflow | Keras for Fashion MNIST. *Kaggle*. [Online] 2022. <https://www.kaggle.com/code/gpreda/cnn-with-tensorflow-keras-for-fashion-mnist/notebook#Read-the-data>.

Sharma, Aditya. 2020. Principal Component Analysis (PCA) in Python Tutorial. [Online] 2020. <https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python>.

Violante, Andre. 2020. An Introduction to t-SNE with Python Example. [Online] 2020. <https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1>.