IE7860: Intelligent Analytics

Assignment 6 : Auto-Encoders & Convolution Neural Networks

FASHION IMAGES AND CATEGORY CLASSIFICATION

Overview

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms.

Data Description

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above) and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

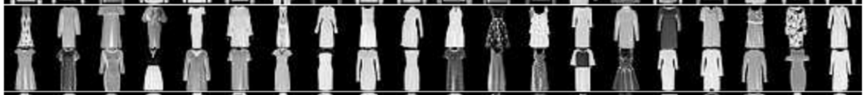
To locate a pixel on the image, suppose that we have decomposed x as $x = i * 28 + j$, where i and j are integers between 0 and 27. The pixel is located on row i and column j of a 28 x 28 matrix. For example, pixel 31 indicates the pixel that is in the fourth column from the left, and the second row from the top.

Data File

Each row is a separate image. Column 1 is the class label. Remaining columns are pixel numbers (784 total). Each value is the darkness of the pixel (1 to 255)

Class Labels: Each training and test example is assigned to one of the following labels: 0 for T-shirt/top, 1 for Trouser, 2 for Pullover, 3 for Dress, 4 for Coat, 5 for Sandal, 6 for Shirt, 7 for Sneaker, 8 for Bag, and 9 for Ankle boot.

Name	Description	# Examples	Size
train-images-idx3-ubyte.gz	Training set images	60, 000	25 MBytes
train-labels-idx1-ubyte.gz	Training set labels	60, 000	140 Bytes
t10k-images-idx3-ubyte.gz	Test set images	10, 000	4.2 MBytes
t10k-labels-idx1-ubyte.gz	Test set labels	10, 000	92 Bytes

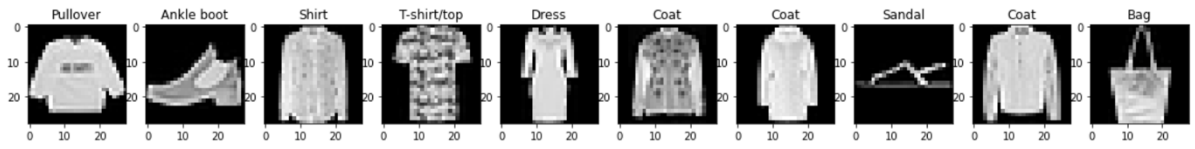
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Data Preprocessing

Then dataset is clean but require certain level of pre-processing. We Start by loading the data from the text file into dataframes. After which we shape the data based on the input and output parameters. This is done so as to shape the data for the model to process.

The validation set is used during the model fitting to evaluate the loss and metrics. However, the model is not fit with this data. The test set is completely unused during the training phase and is only used at the end to evaluate how well the model generalizes to new data. This is especially important with imbalanced datasets where overfitting is a significant concern from the lack of training data.

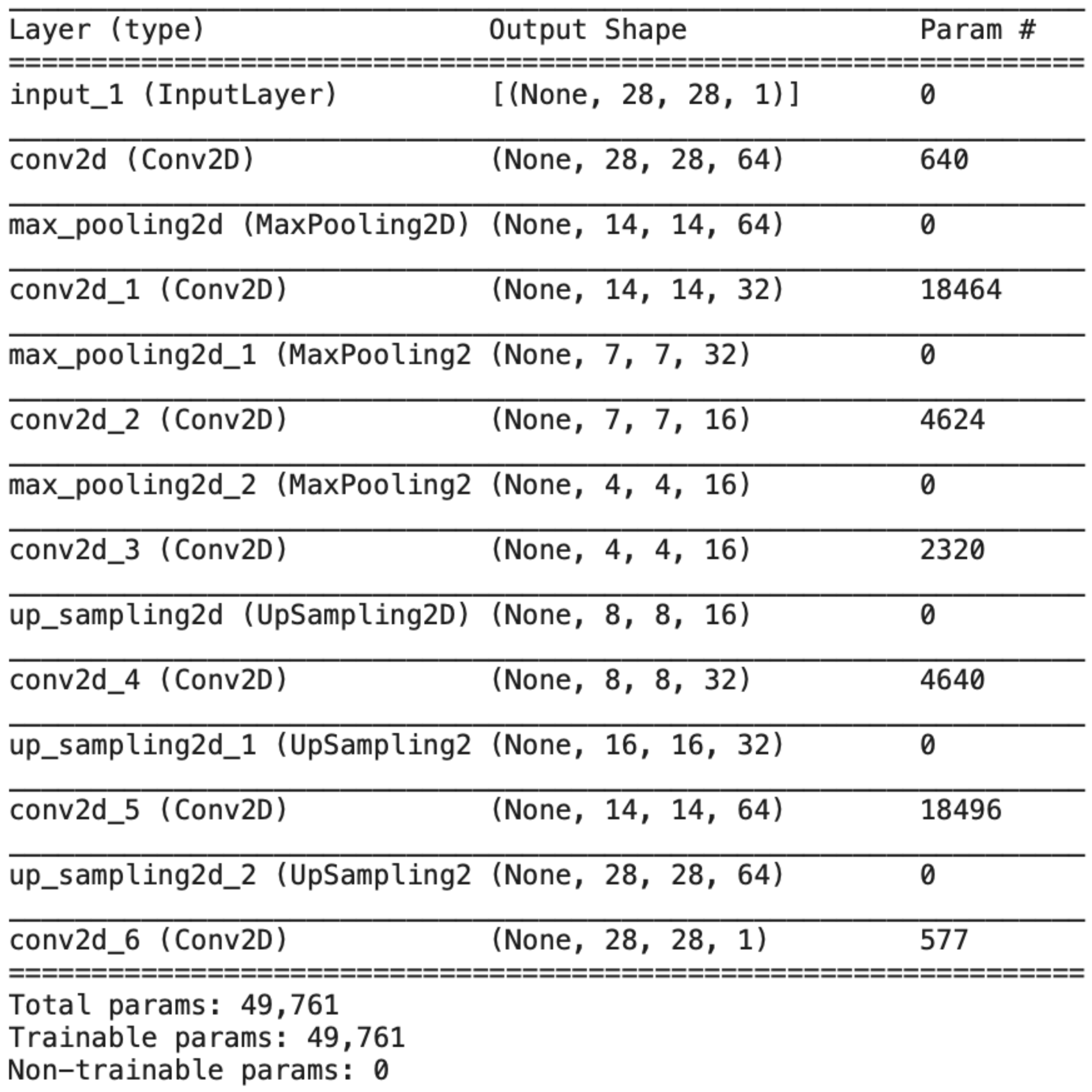
The data is then reduced to grayscale. This reduces dimension at the base level. Its also reshaped into (-1,28,28,-1) to be implemented into a CNN and Convolutional Neural Network. Since the data is originally well defined, there is no further need of augmentation.



Creating the Neural Network and Visualising High Dimensional Data

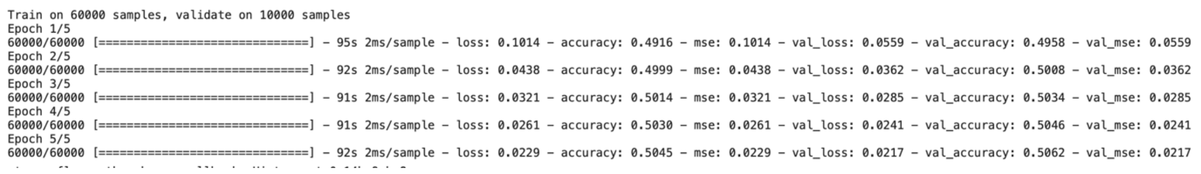
The model definition is just a few lines of code but has all the parameters required. The backend and all the complications are handled by Keras and Tensorflow. The algorithm uses randomness in order to find a good enough set of weights for the specific mapping function from inputs to outputs in your data that is being learned. It means that your specific network on your specific training data will fit a different network with a different model skill each time the training algorithm is run.

We create the model using conv2d, maxpooling2d and upsampling2d layers from the keras library built into TensorFlow 2.0. The first two pairs of conv2d and maxpool2d are the encoders the third maxpool layer is the latent layer in between the encoder and the decoder. The following layers are part of the decoder. Here the final conv2d layer is the output layer which outputs the image in the dimensions (28,28,1). The following model was achieved after multiple additions and deletions to the structure to achieve a very low loss.



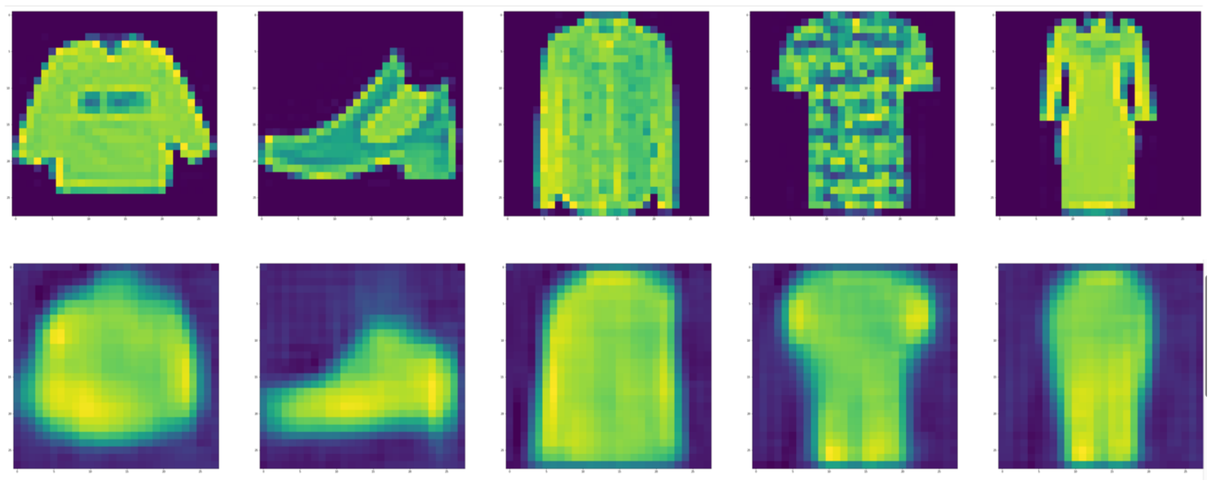
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0         
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 64)        640       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        18464     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 16)          4624      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 16)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 16)          2320      
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 16)          0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 32)          4640      
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        18496     
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 64)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         577       
=================================================================
Total params: 49,761
Trainable params: 49,761
Non-trainable params: 0
```

After Compiling the Neural Network, we begin the training process. Here we observe that the final validation loss is 0.0217 with only 5 epochs. Initially with shallower models I was getting a validation loss of 0.673 over 50 epochs. Which in itself took 4 hours to run due to the size of the data.

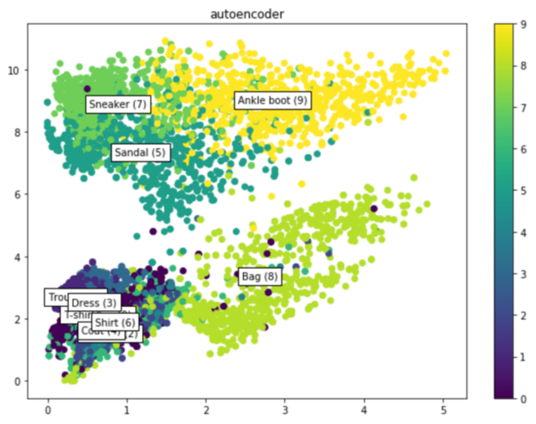


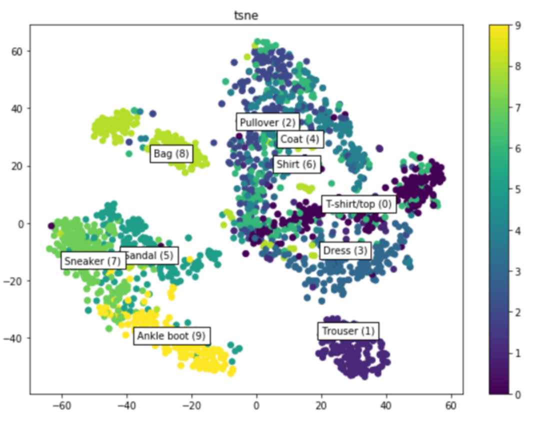
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 95s 2ms/sample - loss: 0.1014 - accuracy: 0.4916 - mse: 0.1014 - val_loss: 0.0559 - val_accuracy: 0.4958 - val_mse: 0.0559
Epoch 2/5
60000/60000 [==============================] - 92s 2ms/sample - loss: 0.0438 - accuracy: 0.4999 - mse: 0.0438 - val_loss: 0.0362 - val_accuracy: 0.5008 - val_mse: 0.0362
Epoch 3/5
60000/60000 [==============================] - 91s 2ms/sample - loss: 0.0321 - accuracy: 0.5014 - mse: 0.0321 - val_loss: 0.0285 - val_accuracy: 0.5034 - val_mse: 0.0285
Epoch 4/5
60000/60000 [==============================] - 91s 2ms/sample - loss: 0.0261 - accuracy: 0.5030 - mse: 0.0261 - val_loss: 0.0241 - val_accuracy: 0.5046 - val_mse: 0.0241
Epoch 5/5
60000/60000 [==============================] - 92s 2ms/sample - loss: 0.0229 - accuracy: 0.5045 - mse: 0.0229 - val_loss: 0.0217 - val_accuracy: 0.5062 - val_mse: 0.0217
```

The output now encoded and can easily resolve the image and is very effective at simplifying the learning process.



The next step of the process was to visualise the input data with methods such as UMAP, t-SNE. These are methods that are used to reduce the dimensionality of the data. In cases such as this where visualising images as clusters is not practical at all. Methods mentioned above reduce the dimensions of an image of 28x28=784 dimensions to 2. It is relatively easy to implement UMAP and t-SNE since very robust libraries are available and can be easily used. We are also going to compare how it defers from auto encoders and which results are relatively better.

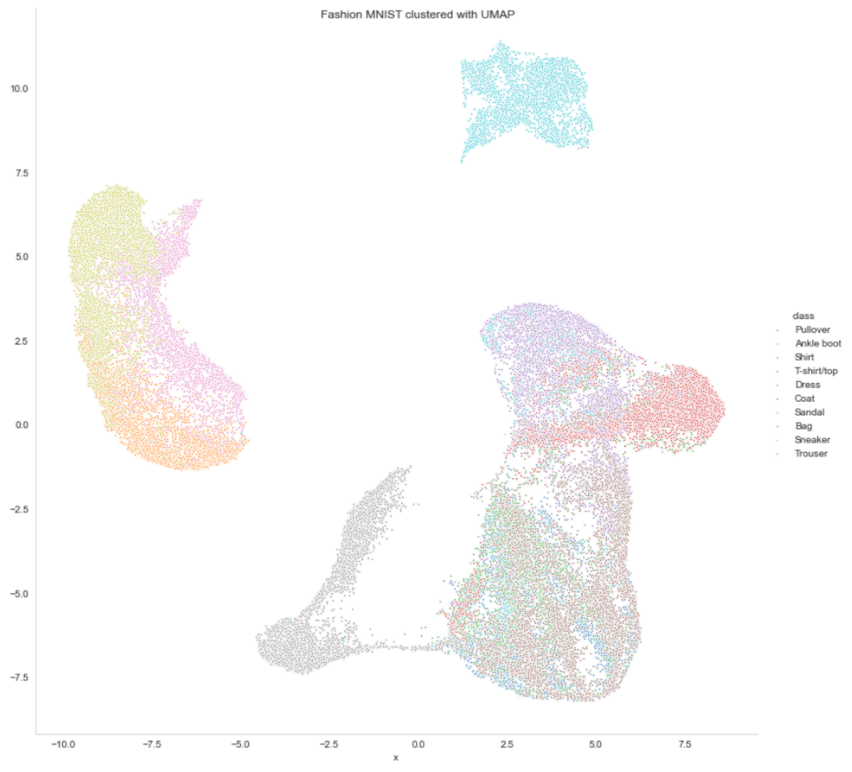




In the above images, the first one is for AUTOENCODERS and the second one is for t-SNE. We can clearly observe that autoencoders fails to encode clothes properly. It clusters them all together. Whereas t-SNE is clearly the better method mong the two, the dimensionality reduction algorithm trumps autoencoders. There is a relatively better separation between the individual clusters.

Now looking at another dimensionality reduction method to visulaise the input data in two dimentions. UMAP, at its core, works very similarly to t-SNE - both use graph layout algorithms to arrange data in low-dimensional space. In the simplest sense, UMAP constructs a high dimensional graph representation of the data then optimizes a low-dimensional graph to be as structurally similar as possible.

The visualisation below is for UMAP. Even though UMAP is supposed to be better in most cases. But for this data I would conclude that t-SNE did a better job. Since UMAP also has a similar problem as autoencoders where it is unable to between classes that are similar. Such as T-shirts from Dresses. The main difference between t-SNE and UMAP is the interpretation of the distance between clusters. I use the quotation marks since both algorithms are not meant for clustering. They are meant for visualization, mostly t-SNE preserves local structure in the data whereas UMAP claims to preserve both local and global structure in the data. This means with t-SNE you can't interpret the distance between two clusters at different ends of your plot. You cannot infer that these clusters are more dissimilar two other clusters, where the two new ones might be closer. But within a cluster, you can say that points close to each other are more similar objects than points at different ends of the cluster image. With UMAP, you should be able to interpret both the distances between points and clusters. Both algorithms are highly stochastic and very much dependent on choice of hyperparameters (t-SNE even more than UMAP) and can yield very different results in different runs, so one plot might obfuscate an insight in the data that a subsequent run might reveal.



References

D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classifi- cation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical im- age database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document

recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML- 13)*, pages 1058–1066, 2013.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms.

Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.