

Introduction

Researchers at Zalando developed a new image classification dataset called Fashion MNIST in hopes of replacing MNIST. The original MNIST dataset contains a lot of handwritten digits. Members of the AI/ML/Data Science community love this dataset and use it as a benchmark to validate their algorithms. In fact, MNIST is often the first dataset researchers try. This new dataset from Zalando contains images of various articles of clothing and accessories—such as shirts, bags, shoes, and other fashion items. Since its introduction, machine learning practitioners have set out to train models to predict clothing items using several methods—some of which use the same deep learning libraries and algorithms used to analyze MNIST dataset.

About the data:

The Fashion MNIST training set contains 60,000 examples, and the test set contains 10,000 examples. 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. Fashion MNIST also shares the same train-test-split structure as MNIST, for ease of use.

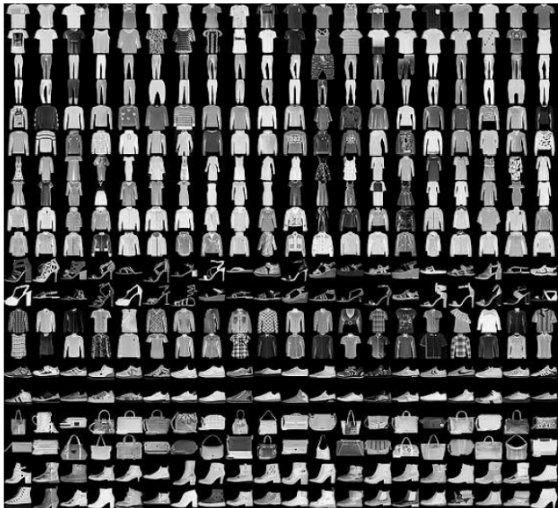
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Table 1: Fashion MNIST dataset

The datasets in csv format were downloaded from Kaggle:
<https://www.kaggle.com/yufengg/fashion-mnist/data>:

Objectives

- 1- Implement the following two methods to classify clothing items:
 - i. Keras Convolutional Neural Network
 - ii. Support Vector Machines using Scikit-Learn
- 2- Answering the following questions:
 - i. What is the accuracy of each method?
 - ii. What are the trade-offs of each approach?
 - iii. What is the compute performance of each approach?

Data Exploration

Dataset break-down:

- train - 60,000 rows and 785 columns
- test - 10,000 rows and 785 columns

Class distribution for train and test datasets:

Ankle Boot	:	6000	Sneaker	:	1000
Bag	:	6000	Shirt	:	1000
Sneaker	:	6000	Sandal	:	1000
Shirt	:	6000	Coat	:	1000
Sandal	:	6000	Dress	:	1000
Coat	:	6000	Pullover	:	1000
Dress	:	6000	Ankle Boot	:	1000
Pullover	:	6000	Trouser	:	1000
Trouser	:	6000	Bag	:	1000
T-shirt/top	:	6000	T-shirt/top	:	1000

All classes are perfectly balanced.

Displaying some the images to make sure the data correspond to actual clothing items:



Fig. 2. Fashion-MNIST items from train dataset (sample of 20)



Fig. 3: Fashion-MNIST items from test dataset (sample of 20)

Data preprocessing

- Reshaped the columns from (784) to (28, 28, 1).
- Data is normalized by dividing the pixel values by 255 which results in values between 0 and 1.
- Saved label (target variable) feature as a separate vector.
- Split the train set into train and validation sets.
- The validation set will be 20% from the original train set.
- New datasets:
 - Train data: 48,000 records
 - Validation: 12,000 records
 - Test: 10,000 records

Data Modeling:

Convolutional Neural Network (CNN) using Keras:

A convolution layer multiplies a matrix of pixels with a filter matrix or 'kernel' and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered.

A model using Keras python library is developed to predict the labels with the following criteria:

Sequential Model:

- **Conv2D: 2D Convolutional layer:**
 - filters = 32
 - kernel_size = (3 x 3)
 - activation = RELU
 - input_shape = 28 x 28
 - The input and output of the Conv2D is a 4D tensor
- **MaxPooling2D is a Max pooling operation for spatial data:**
 - pool_size = (2,2)
 - Conv2D with the following parameters:

- filters: 64
 - kernel_size : (3 x 3)
 - activation : RELU
- **MaxPooling2D:**
 - pool_size : (2,2)
- **Conv2D:**
 - filters: 128
 - kernel_size : (3 x 3)
 - activation : RELU
- **Flatten:**
- **Dense, fully-connected NN layer:**
 - units = 128;
 - activation: RELU
- **Dense, output layer (fully connected):**
 - units = 10 Classes;
 - activation = Softmax (standard for multiclass classification)

The model is compiled using categorical cross-entropy and the Adam optimizer. The number of epochs used to fit the model is 50.

Model Results

Once fitted, the test dataset was used to measure the accuracy:

- Test loss: 0.655
- Test accuracy: 0.912

Keras model train history is used to plot loss and accuracy (train/validation data):

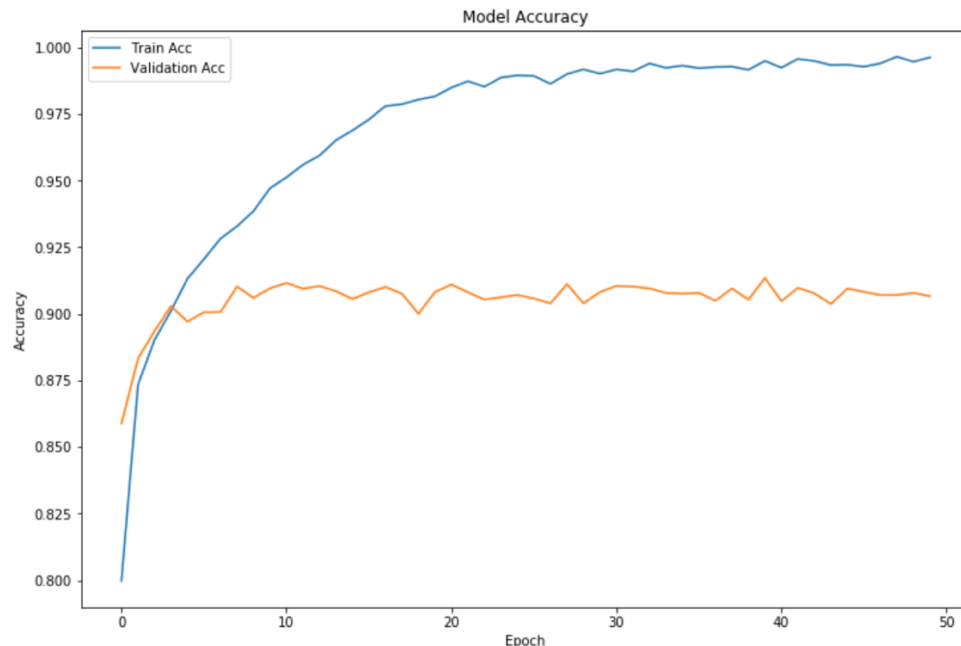


Fig.4: train/validation accuracy by epochs

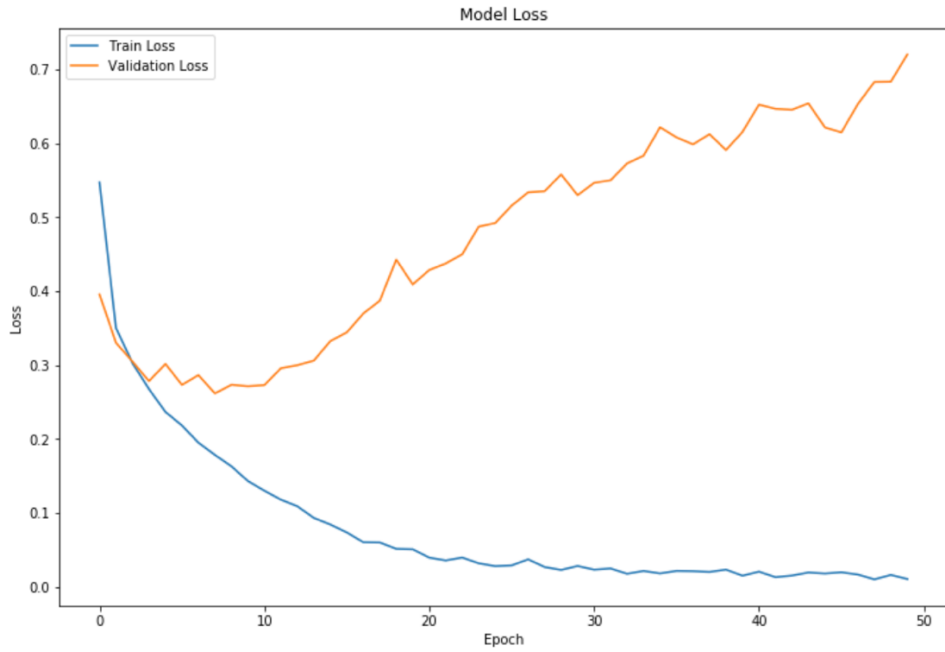


Fig. 5: train/validation loss

Observations:

- Accuracy does not increase after a few epochs.
- Loss increases steadily which indicates the model is over fitting.

To avoid overfitting, multiple drop-out layers are added to the model and the training is executed again. The new model summary is shown below:

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 128)	73856
dropout_2 (Dropout)	(None, 3, 3, 128)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_2 (Dense)	(None, 128)	147584
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1290
Total params: 241,546		
Trainable params: 241,546		
Non-trainable params: 0		

The new model was fitted, and the history plot shows how accuracy increases for validation data and loss decreases as epochs are completed:

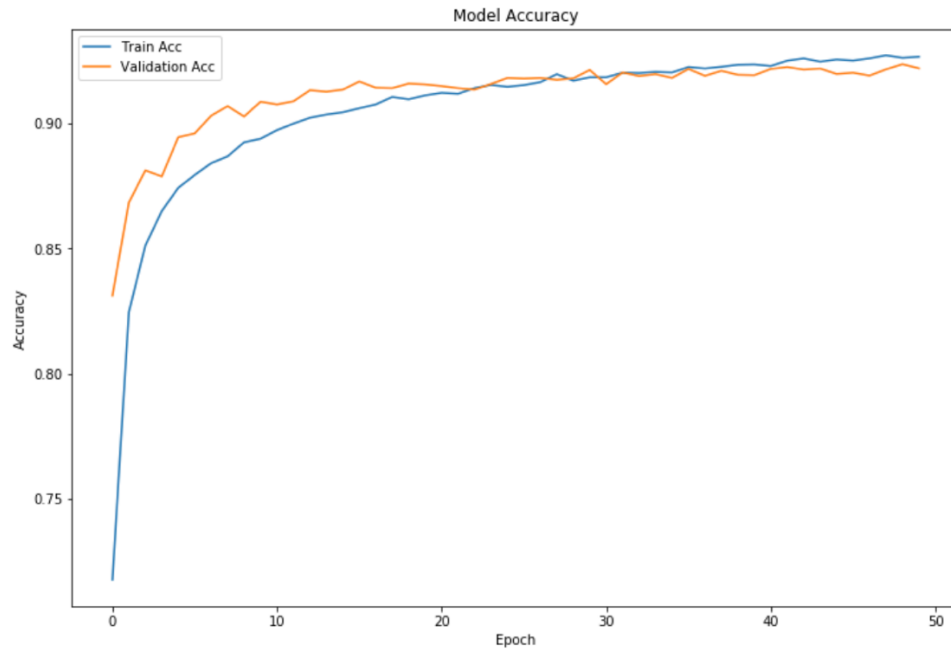


Fig.6: train/validation accuracy by epochs of new model

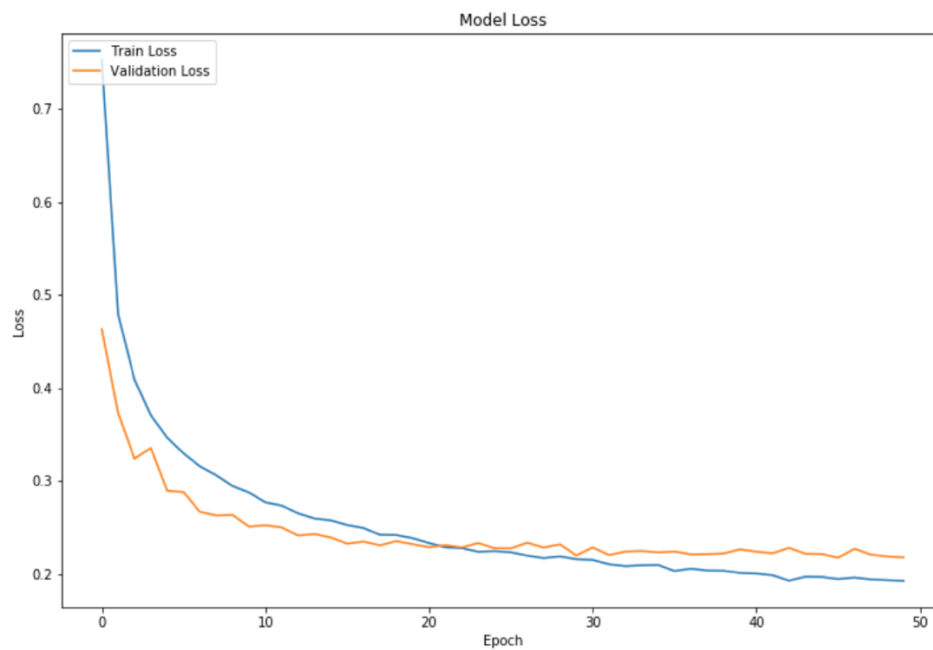


Fig.7: train/validation loss by epochs of new model

Results are better overall:

- Test loss: 0.196
- Test accuracy: 0.932
- Model does well with train and validation datasets

- Correct predicted classes: 9,245
- Incorrect predicted classes: 755

Displaying precision, recall and F1-score metrics by class:

	precision	recall	f1-score	support
Class 0 (T-shirt/top) :	0.89	0.85	0.87	1000
Class 1 (Trouser) :	0.99	0.99	0.99	1000
Class 2 (Pullover) :	0.89	0.88	0.88	1000
Class 3 (Dress) :	0.93	0.94	0.93	1000
Class 4 (Coat) :	0.90	0.87	0.89	1000
Class 5 (Sandal) :	1.00	0.98	0.99	1000
Class 6 (Shirt) :	0.76	0.81	0.78	1000
Class 7 (Sneaker) :	0.95	0.97	0.96	1000
Class 8 (Bag) :	0.99	0.98	0.99	1000
Class 9 (Ankle Boot) :	0.97	0.97	0.97	1000
micro avg	0.92	0.92	0.92	10000
macro avg	0.93	0.92	0.92	10000
weighted avg	0.93	0.92	0.92	10000

Predictions for Shirts class are the lowest with 76% precision and 81% recall. The following are some examples of incorrect predictions:



FIG. 8: Sample of CNN incorrect predictions

Support vector Machines Model (SVM):

A SVM classification model was built using Scikit-Learn library. SVM's are highly effective in high dimensional spaces such as the case of fashion-MNIST dataset. Because different kernel functions can be specified for the decision function, SVM's are versatile. The disadvantage of SVM is that it does not provide a probability estimate for classifications.

For this model we used the default kernel ‘RGF’ function with a gamma coefficient of 0.05 and a penalty parameter C=5.

SVM Results:

The model trained in 23.5 minutes which is a big improvement over the CNN model which took 2.5 hours to train. Below graphs show the confusion metrics with normalization and without normalization:

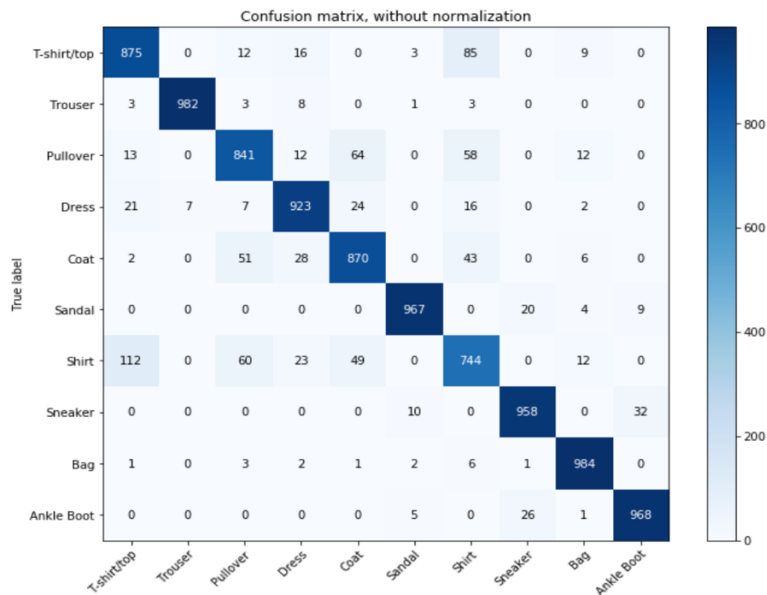


Fig. 9: SVM confusion matrix without normalization

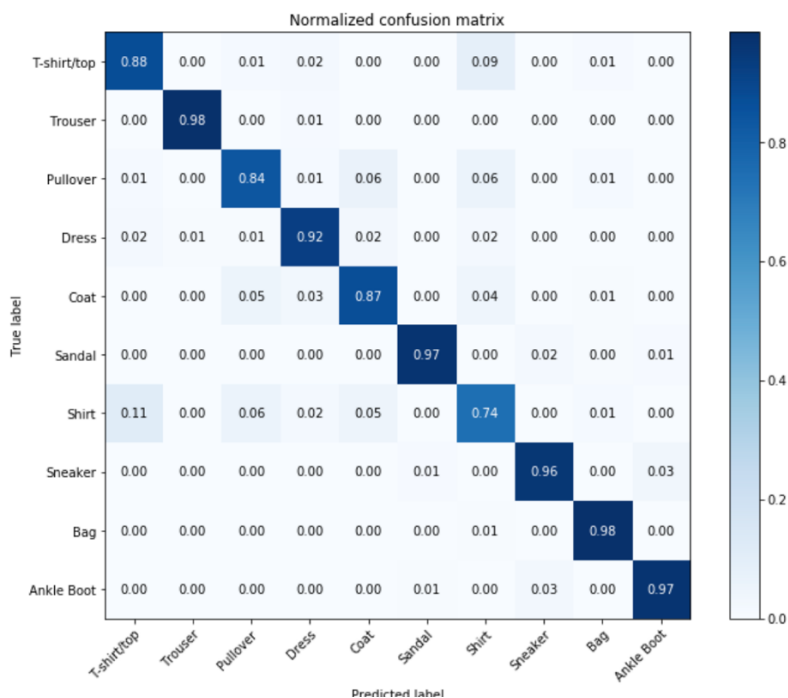


Fig. 10: SVM confusion matrix with normalization

The SVM models also struggles predicting shirts for which the accuracy is 74%. The overall precision and recall metrics for the model is 91% which is shy of the CNN model.

	precision	recall	f1-score	support
T-shirt/top	0.85	0.88	0.86	1000
Trouser	0.99	0.98	0.99	1000
Pullover	0.86	0.84	0.85	1000
Dress	0.91	0.92	0.92	1000
Coat	0.86	0.87	0.87	1000
Sandal	0.98	0.97	0.97	1000
Shirt	0.78	0.74	0.76	1000
Sneaker	0.95	0.96	0.96	1000
Bag	0.96	0.98	0.97	1000
Ankle Boot	0.96	0.97	0.96	1000
micro avg	0.91	0.91	0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Conclusions (answers to questions):

The Convolutional Neural Network or CNN model developed with Keras (python API to TensorFlow) delivered a higher accuracy of 93% compared to the Support Vector Machine classifier using Scikit-Learn which reached a 91% accuracy. With both models struggling to predict “Shirts” class. Although the only adjustment made to the models was adding dropout layers to the CNN which eliminated overfitting, the performance of the SVM was surprisingly very good because it took significantly less time to train for the same dataset. The SVM took less than 24 minutes while the CNN took 2.5 hours to train. Fine-tuning both models is highly recommended in order to achieve better results. In general, SVM’s are known to deliver very good results for image classification (See MNIST benchmarks) when using gaussian and polynomial kernels. Large CNN models deliver state-of-the-art results—often better than human—however, these networks are resource-intensive and require GPU hardware for training. Overall, this analysis has shown that both, CNN and SVM algorithms are useful to identify clothing items.

Resources

- Kaggle.com*. (n.d.). Retrieved from Fashion MNIST An MNIST-like dataset: <https://www.kaggle.com/zalando-research/fashionmnist>
- Tensorflow Hub Authors. (2018). *Google Colab*. Retrieved from Fashion MNIST with Keras and TPUs: https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/fashion_mnist.ipynb#scrollTo=edfbxDDh2AEs
- Tensorflow, Google Inc. (n.d.). *Train your first neural network: basic classification*. Retrieved from Tensorflow: https://www.tensorflow.org/tutorials/keras/basic_classification
- Yann LeCun, C. C. (n.d.). *yann.lecun.com*. Retrieved from THE MNIST DATABASE of handwritten digits: <http://yann.lecun.com/exdb/mnist/>