The dataset was taken from Kaggle.com 🡪 <https://www.kaggle.com/datasets/zalando-research/fashionmnist>

Kaggle is an online community for data scientists and machine learning practioners.

Problem Statement:

Imagine you’re a fashion designer, and your model has created 1000s of different shoes, shirts, accessories etc. Now I’ve decided to open an online website for my products, and so photos need to be taken for these products, and label them. Let’s say you have these products labelled but do you have these products labelled with the specifics the online platform demands? And if I suppose want to add more and more photos then it would be nice if the labelling can be automated and stream lined. If I have a website, where people can sell their clothes, but any person can join and upload what they want to sell, but these people don’t take their time to label their products.

This can prove to be a problem for the owner, since the consumers may struggle to find the items, and the platform losses ground to competition. So, I decide to create a model that can assign these products their correct labels. This illustrates the significance of the image classification problem.

**Dataset Info:**

Context

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. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

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. "If it doesn't work on MNIST, it won't work at all", they said. "Well, if it does work on MNIST, it may still fail on others."

Zalando seeks to replace the original MNIST dataset

Content

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, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Labels

Each training and test example is assigned to one of the following labels:

* 0 T-shirt/top
* 1 Trouser
* 2 Pullover
* 3 Dress
* 4 Coat
* 5 Sandal
* 6 Shirt
* 7 Sneaker
* 8 Bag
* 9 Ankle boot

TL;DR

* 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)

There’s no validation set, so it needs to be created for the early stopping mechanism to get called, hence to avoid overfitting.

**# KERNELS:**

Basic building block of filters and CNNs. Kernels are used to structurally transform the underlying structures and properties of the image. Kernels are the matrices that describe how a certain transformation is to be applied to a certain image. Kernels are used for feature detection. It is very computationally intensive.

**# AIM OF THE PROJECT:**

The aim of this project is to assign the products (in this case, the images of clothing accessories) to their correct labels, and this illustrates the significance of the image classification problem.

**# PROCESS OF THE PROJECT, AND THE STEPS TAKEN WHILE BUILDING THE MODEL:**

1. **Data Pre-processing –**

->Since, the dataset was in a csv file format, the csv files were first imported through the pandas library by creating a data frame.

->Then the data frame was split into the features (independent variables for the model), and the labels – the target value which needs to be predicted (dependent variable).

->The matrix of features of the training\_set, and the test\_set was then converted to a numpy array, since the TensorFlow model inputs only numpy array as it’s input.

->Further, the training and the test set were reshaped in-order to plot out the image files which are to be fed into the model

->Since the images are in grayscale, and for any grayscale image, the pixel value is a single number which represents the brightness of that pixel. The most common format is the byte image, where the pixel value is stored between 0 to 255, (0->black, 255->white)

->Once, this process has taken place, normalization is performed to improve the accuracy, and the pixel values are brought down in the range [0,1].

->Now, the training\_set is further split into the training and the validation\_set to validate the model simultaneously while the model is training through backpropagation in an epoch.

->Working with numpy arrays have numerous benefits, like they can be easily rescaled.

In [image processing](https://en.wikipedia.org/wiki/Image_processing), **normalization** is a process that changes the range of [pixel](https://en.wikipedia.org/wiki/Pixel) intensity values.

The purpose of dynamic range expansion in the various applications is usually to bring the image, or other type of signal, into a range that is more familiar or normal to the senses, hence the term normalization.

-> X\_train = X\_train/255.0

X\_test = X\_test/255.0

->When working with numpy arrays, tensorflow can easily batch and shuffle the dataset. This is done in the .fit() method. So, here the data pre-processing part gets over.

**2. General information about Convolutional Neural Network --**

-> A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

->Further the images are reshaped into 28,28,1 – which represents 28,28 grayscale images.

-> The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

->The kernel shifts every time, while performing matrix multiplication to each of the pixels in the image, with the corresponding values in the feature detector thus forming a convolved feature (feature map).

->Convolution operation is performed so as to extract high level features from the images

**3. Why CNNs over ANNs?**

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? Uh.. not really.

In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Using ANNs would make the process more computationally intensive once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

1. Converting images to NumPy arrays.
2. Neural networks don’t work with images files, they work with tensors, so all these have to be converted into NumPy arrays.
3. Pre-processing is simple since TensorFlow objects are not used, and NumPy arrays are being used.
4. Image can be manipulated using a variable.
   1. **Conv2D layer and Pooling:**

The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

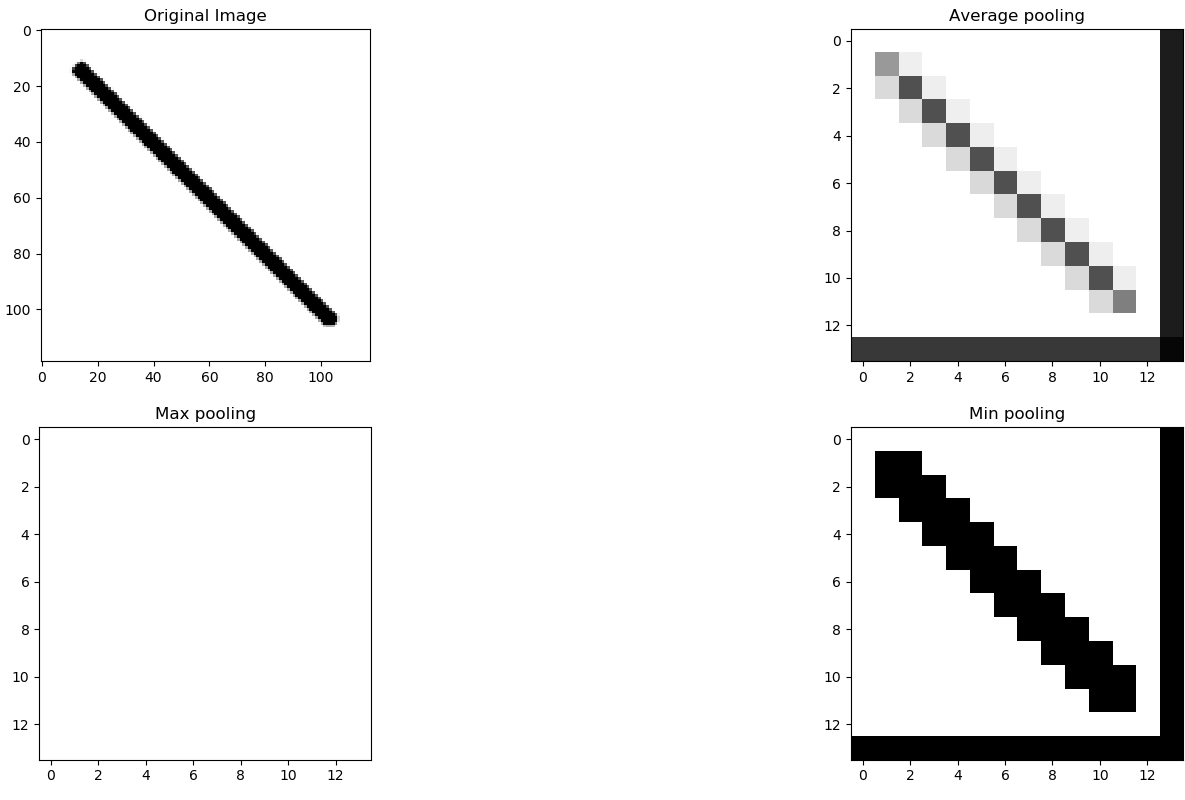
Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model. Pooling is performed in neural networks to reduce variance and computation complexity

Removal of information to prevent overfitting

Pooling is used right after a convolutional layer, to only preserve the most significant features and to become less dependent on their precise positioning

Max Pooling also performs as a**Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.

Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image.



**# Building the layers:**

In general, batch size may affect the speed of the training, but not the accuracy.

1. When the BATCH\_SIZE is parameter is set, tensorflow will automatically SHUFFLE and BATCH the numpy arrays
2. First layer will be Conv2D not Dense, since that would unpack the images, into 1D vectors.
3. Kernel size set to 3 since images, since the images are 28X28 -> 7x7 should be the max kernel size.
4. Deciding the kernel no. is not always obvious what the correct no. should be
5. Activation fn will be relu, since it behaves well in most situations.
6. Conv2D is followed by a maxpool with kernel size 2,2
7. Since the first conv layer has 5, 5 dimensions, so the image will get scaled from 28x28 to 24x24.
8. Since the maxpool layer only cuts the spacial dimensions, it will output a layer of 24x24x50 to 12x12x50.
9. Having a kernel size of 3x3, reduces the number of parameters.
10. The output from the 4th layer is multidimensional, and it is to be made 1D and this can be done with Flatten layer,
11. Then Dense layer is put, which classifies into 10 numbers. A dense layer is a transformation, in which every output is a linear combination of the inputs. (Fully connected layer).
12. Reducing max\_pool to 1, removing the strides, filters early were set to 50, changed to 32,62, added some dropout layers and an additional dense layer after flattening
13. No, when having two consecutive convolution layers can't be combined into one. The subsequent filter's inputs are the features extracted from the previous one. This results in the second layer's features are of **higher-level** than the previous.
14. This is the basis of the whole CNN. Having multiple convolutional layers stacked along the depth of the network, allows the network to extract high-level features (not just edges and corners) from the input images.
15. The first convolutional layer of a CNN is essentially a standard image filter (+ a ReLU). Its goal is to take a **raw image** and extract **basic features** from it (e.g. edges, corners). These are referred to as **low-level features**.
16. The second convolutional layer, instead of the raw image, accepts the features extracted by the first as its input. This allows it to combine these basic shapes into **more complex features**.
17. The features extracted become more and more complex as we go further down the network. Layers near the middle of the network extract the so called **mid-level features**, while the final layers extract **high-level features**.
18. CNNs are powerful tools because it is trained to extract the **best** features for each task. This results in the network extracting **different** features for different tasks.

## Adding a kernel initializer, Usage of initializers Initializers define the way to set the initial random weights of Keras layers.

The keyword arguments used for passing initializers to layers depends on the layer. Usually, it is simply kernel\_initializer and bias\_initializer:

1. And putting the filters as 32 and 64 respectively.
2. While pre-processing the image, the np.expand\_dims() is used to expand the dimension of dimension of images since the image is fed into batches, so this pre-processed image also needs to be converted into a batch, hence this method is applied. Also, the batch dimension needs to be the first dimension, hence the axis=0

**# Building 3 Deep CNNs so as to increase the number of tuneable parameters to increase accuracy:**

**Model 1:**

1. Added one Conv2D layer, initially set the output filters to 32 and the size of the feature size to 3,3. Activation is set to ‘relu’ since it behaves well in most situations.
2. Then MaxPooling is done, and some dropout layers have been added to reduce the complexity hence to avoid overfitting.
3. Then the image vector is simply flattened out to a 1-D vector with the   
   Flatten class. And the output layer has SoftMax as it’s activation function in order to output a probability distribution of values.

**Model 2:**

1. Initially same Conv2D layer, and a MaxPool layer have been added.
2. Next, another Conv2D layer with an output filter set to 64 is added. This is done so that the network is able to extract high-level features from the images and so that it has a wholesome understanding of the images.
3. Finally the images is Flattened out, and the final output layer has incorporated ‘SoftMax’ to output the distributions.

**Model 3:**

1. The initial Conv2D, and the MaxPool layer is the same as the model1 and ‘kernel\_initializer’ is used to initialize the random weights and biases of the model.
2. Rest of the layers are similar as the other models.

The models are then compiled using the ‘sparse-categorical cross-entropy function’, since the output layer need to output 10 different labels. Could have used ‘categorical-crossentropy’ as the loss function if the labels would have been encoded into dummy variables.

The optimizer is used Adam, Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the ‘gradient descent with momentum’ algorithm and the ‘RMSP’ algorithm.

**How Adam works?**

Adam optimizer involves a combination of two gradient descent methodologies:

**Momentum:**

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the ‘exponentially weighted average’ of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

**Root Mean Square Propagation (RMSP):**

Root mean square prop or RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the ‘exponential moving average’.

The models are trained to 50 epochs, to generalize the outcomes from all the models, and provide an overview that more tuneable parameters in the model leads to better accuracy, provided the model is trained long enough.

Therefore, the early stopping callbacks are not added, even if the validation\_loss is seen increasing.

The models are finally evaluated on the test\_set, and the model\_accuracy and loss are plotted out

**# METRICS:**

The metrics are evaluated, and the classification report and the confusion matrix are plotted out.

**# TESTING:**

Finally, the model is tested on the uploaded images. Since the model is trained on grayscale images, hence the testing images have to be converted to grayscale before the testing is done.

**# CONCLUSION:**

The histories and the losses, accuracy are generalized for each of the models, hence overcoming

the obstacle faced by the network.