**#CNN project on MNIST Fashion:**

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. Preprocessing is simple since tensorflow objects are not used, and numpy arrays are being used.
4. Image can be manipulated using a variable.

Primary classification: Glasses/Sunglasses, Trousers/Jeans, Shoes

1. These classes have additional subclass, i.e., whether a photo shows a pair of optical glasses or a pair of sunglasses.
2. Trousers and jeans are complicated, a part from trousers and jeans, these can also be differentiated between male and female. (4 variations – male trousers, female trousers, male jeans, female jeans.)
3. Shoes have 11 subclasses in total – Men(boots, trainers, sneakers, sandals, flipflops, and slippers, formal shoes), Women(boots, ballerina shoes, trainers/sneakers, high heels, sandals/flipflops/slippers)
4. Different networks need to be created.
5. An ideal kernel should have an odd number as the dimensions. So an essential quality of a kernel, is that it should be associated with a certain main pixel.

Datasets needed:

1. Primary categories, primary categories-train.npz, primary categories-test.npz, primary categories-validation.npz

For any Beginner in the domain of Neural Network or Machine Learning, the most suitable data-set to get his/her hands dirty, is the **MNIST Dataset**.

But the first challenge that anyone would face before using the data in the images of the hand-written digits, in the data-set, is that the data-set is available in **IDX** format. Hence, the data needs to be converted to suitable format before we can use it in our code.

Now, let’s get to know what this file format

**IDX** file format is a binary file format.

Now, why store in this format when we have other text file formats?

The answer is **performance** and **memory requirements**.

If we look in terms of performance, **binary** file formats are far better than text file formats like **CSV**. **CSV** file formats are used to store tabular data, where for reading a particular value on a certain row or column, the software has to iterate over all the previous values. Whereas in **binary** file format you can literally store anything, provided you also write the proper information for parsing it, in the header of the file. It then makes accessing a **binary** file simpler and faster.

Also, storing data in **binary** format takes less memory, which is really an added advantage when a large volume of data needs to be stored.

->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.

-> train\_images\_array = train\_images\_array/255.0

test\_images\_array = test\_images\_array/255.0

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

**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.

**#Building layers:**

1. Conv2D -> MaxPool2D -> Conv2D ->MaxPool ->Flatten(to unpack the tensor) ->Dense to output the layers representing the classes. (CNN architecture)
2. In general, batch size may affect the speed of the training, but not the accuracy.
3. When the BATCH\_SIZE is parameter is set, tensorflow will automatically SHUFFLE and BATCH the numpy arrays
4. L2 regularization can be done to prevent overfitting.
5. First layer will be Conv2D not Dense, since that would unpack the images, into 1D vectors.
6. Kernel size set to (5,5) since images, since the images are 28X28 -> 7x7 should be the max kernel size.
7. Deciding the kernel no. is not always obvious what the correct no. should be
8. Activation fn will be relu, since it behaves well in most situations.
9. Conv2D is followed by a maxpool with kernel size 2,2 and stride = 2
10. Since the first conv layer has 5, 5 dimensions, so the image will get scaled from 28x28 to 24x24.
11. Since the maxpool layer only cuts the spacial dimensions, it will output a layer of 24x24x50 to 12x12x50.
12. Having a kernel size of 3x3, reduces the number of parameters.
13. The output from the 4th layer is multidimensional, and it is to be made 1D and this can be done with Flatten layer,
14. 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).
15. With adam optimizer, accuracy on the test\_set came out to be 90.27% , and on training\_set it was 92.29% (also L2 regularization was applied)
16. Not much improvement with SGD.
17. After removing the regularizer, it performed much better.
18. Not predicting correctly, some changes needed.
19. 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
20. 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.
21. 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.
22. 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**.
23. 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**.
24. 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**.
25. 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
3. Prediction not that much accurate, 92.250% maximum achieved, on the test\_set
4. Wasn’t giving accurate predictions on all samples.

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. Let’s say you 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 me, 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.