1) How cleaning/EDA was performed

There was no requirement of cleaning the data as the data didn't consisted of any missing or null values.

EDA steps that were performed are:

- Seeing the metadata
- Checking presence of missing values
- Applying some feature engineering techniques to make data ready for modelling
- Visualising different categories & sub categories using plots

2) Your independent and dependent feature

independent features are all the pixels of images stored in different columns and our **target or dependent** variable is label which we need to classify using our model.

3) Why and how selection/engineering/scaling were performed

As we're working with CNN which is able to extract the features of the input given hence no such major feature engineering/scaling technique was used. The only thing that was taken into consideration was converting the image into and numpy array for processing in the model.

4) Which activation function was chosen and why?

For CNN we've used **relu** and **softmax** activation function because relu on one hand is a easy way to calculate the mapping as it returns a value only when it receives a positive input else 0. On the other hand Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would.

5) Which optimizer was chosen and why?

Adam as it combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

6) Which neural network and why? Describe how your neural structuring?

As we were working with image data so we decided to go with **CNN**.

The first step in image processing is to understand, how to represent an image so that the machine can read it?

Every image is an cumulative arrangement of dots (a pixel) arranged in a special order. If you change the order or color of a pixel, the image would change as well.

Three basic components to define a basic convolutional neural network.

- The Convolutional Layer
- The Pooling layer
- The Output layer

The Convolutional Layer :

In this layer if we have an image of size 66. We define a weight matrix which extracts certain features from the images. We have initialized the weight as a 33 matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output. The 66 image is now converted into a 44 image. Think of weight matrix like a paint brush painting a wall. The brush first paints the wall horizontally and then comes down and paints the next row horizontally. Pixel values are used again when the weight matrix moves along the image. This basically enables parameter sharing in a convolutional neural network

The Pooling Layer

If images are big in size, we would need to reduce the no. of trainable parameters. For this we need to use pooling layers between convolution layers. Pooling is used for reducing the spatial size of the image and is implemented independently on each depth dimension resulting in no change in image depth. Max pooling is the most popular form of pooling layer.

The Output layer

With no of layers of convolution and padding, we need the output in the form of a class.

To generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need.

Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class.

The Output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete the backpropagation begins to update the weight and biases for error and loss reduction.

Neural network structure:

The first layer in model network, keras.layers.Flatten, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels). This layer unstacks rows of pixels in the image and lining them up and has no parameters to learn; it only reformats the data.

After the pixels are flattened, the network consists of a sequence of two keras.layers.Dense layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 32 nodes (or neurons). The second (and last) layer is a 10-node softmax layer that returns an array of 10 probability scores that sum to 1. Each node contains a score that indicates the probability that the current image belongs to one of the 10 classes.

Before the model is ready for training, it needs a few more settings. These are added during the model's compile step:

Loss function

This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction. Here we will use "sparse_categorical_crossentropy"

Optimizer

This is how the model is updated based on the data it sees and its loss function.

Metrics

Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the images that are correctly classified.