

Abstract geometric lines in the top-left corner of the slide, consisting of several overlapping, tilted rectangles and polygons drawn with thin black lines.

# CLOTHING TYPE CLASSIFIER

Mohamed Kotb

# OUTLINES

**Project Description - Business Impact**

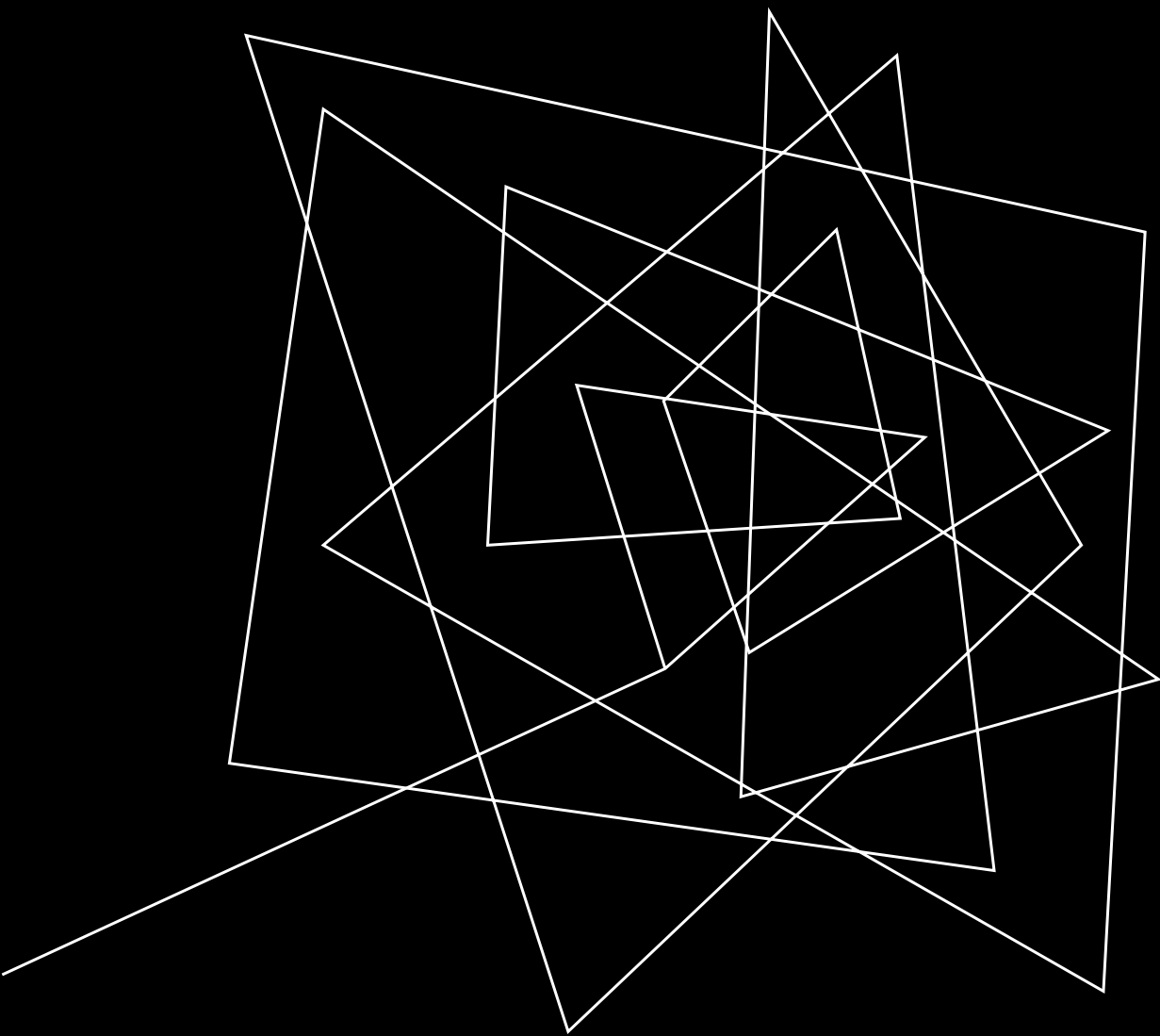
**Exploring the Dataset**

**Visualizing the Dataset**

**Data Pre-Processing**

**Training Models**

**Evaluating the Models**



# PROJECT DESCRIPTION - BUSINESS IMPACT

# DESCRIPTION

**Objective:** classify images of different pieces of clothing.

**Client & Data-Set:** 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.

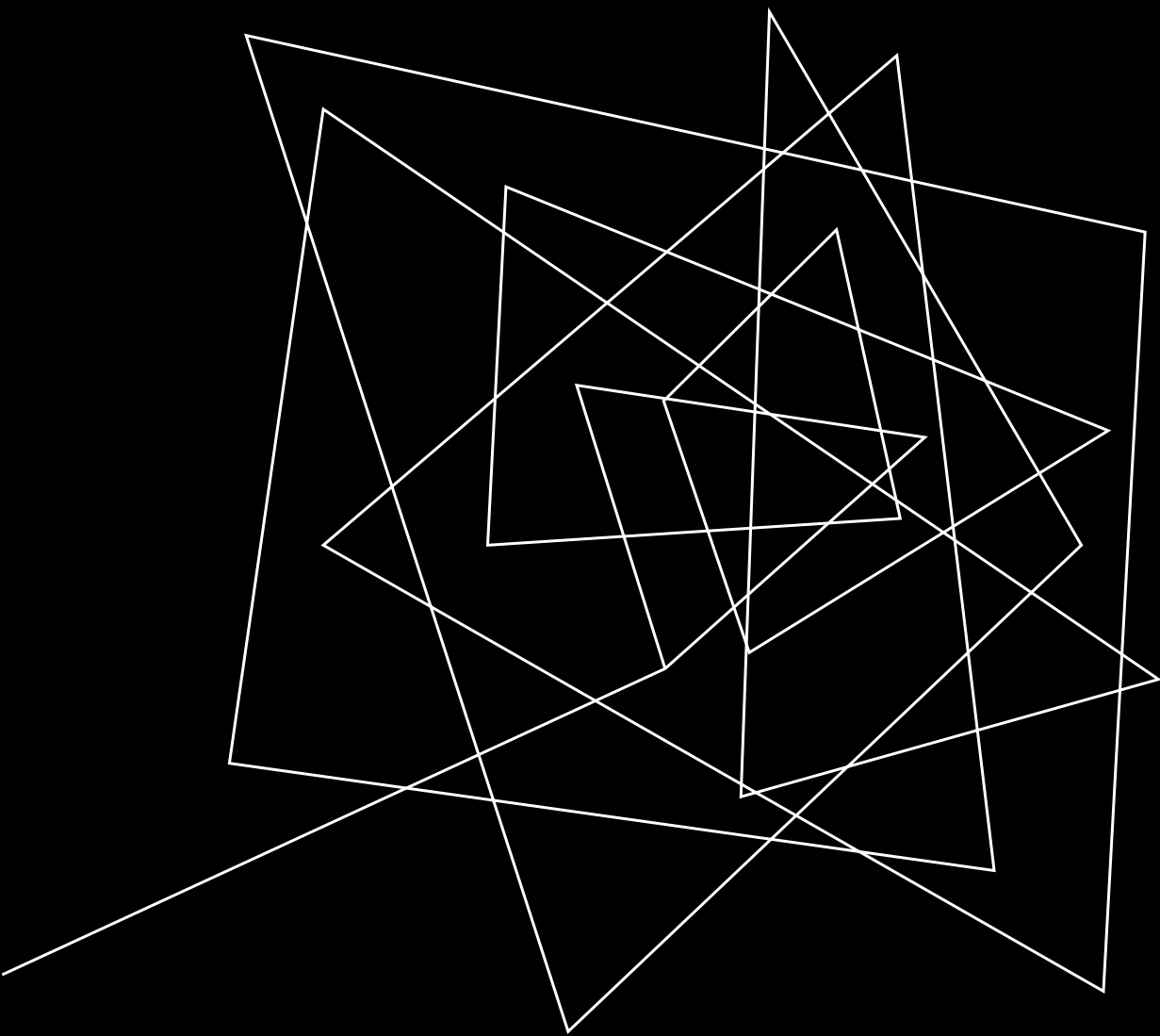
Data-set is publicly available on kaggle and Zalando Fashion MNIST repository on Github.

Fashion-MNIST is intended as direct drop-in replacement for the original MNIST dataset. It shares

the same image size and structure of training and testing splits.

# BUSINESS IMPACT

E-commerce companies have parts of things for deal online which needs parts of pictures to be shown on their websites, applications and on social media. And it takes part of human control and time to isolated these pictures into individual bunches. This classifier which we are progressing to build makes a difference businesses to classify pictures into particular groups. Methodology: For this venture we'll be attending to utilize profound learning concepts like manufactured neural networks and convolutional neural systems to construct an picture classification demonstrate which can learn to recognize 10 diverse thing pictures into their particular categories.



# EXPLORING THE DATASET

# LOAD DATA

- Use simple pandas Library :
- `df = pd.read_csv("")`
- `df.head()`

**Number of training sentences: (60000, 785)**

**Number of testing sentences: (10000, 785)**

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	0	0	0	0	0
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	1	0	0	0	0
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

# FEATURES








\*Label: \* The Target variable.

\*Pixels: \* The smallest unit of a Digital Image or Graphic that can be displayed on Digital Display Device.

Where humans can see the objects due to the Light Receptors in their Eyes which send Signals via the Optic Nerve to the Primary Visual Cortex, where the input is processed ,

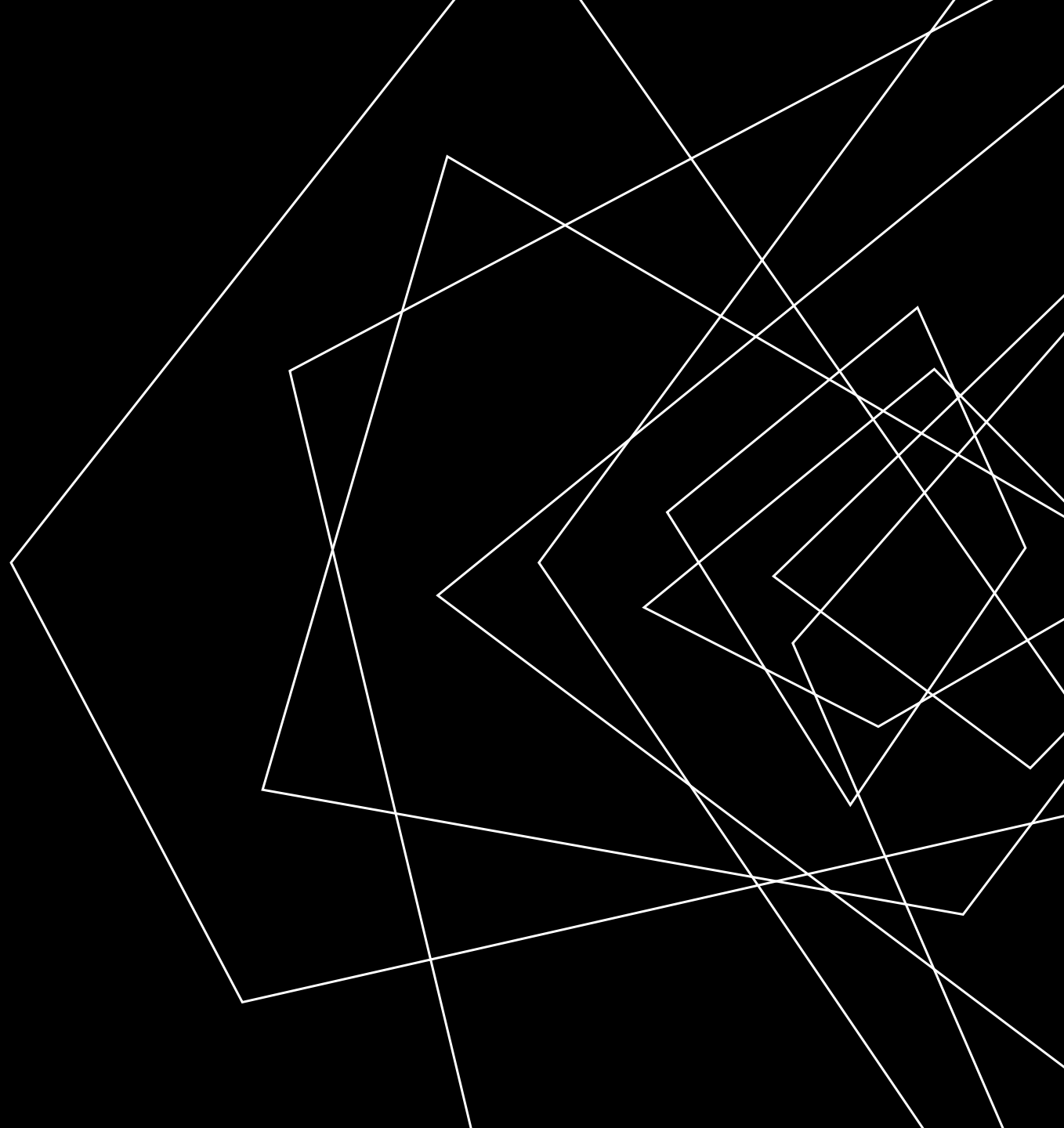
Computers on the other hand, see the Image as 2-dimensional arrays of numbers, known as pixels. They Classify Images based on Boundaries and Curvatures of the Object (Represented by pixel values, either RGB or GrayScale) .

This is the Partial View of the Labels and the Dataset.

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	

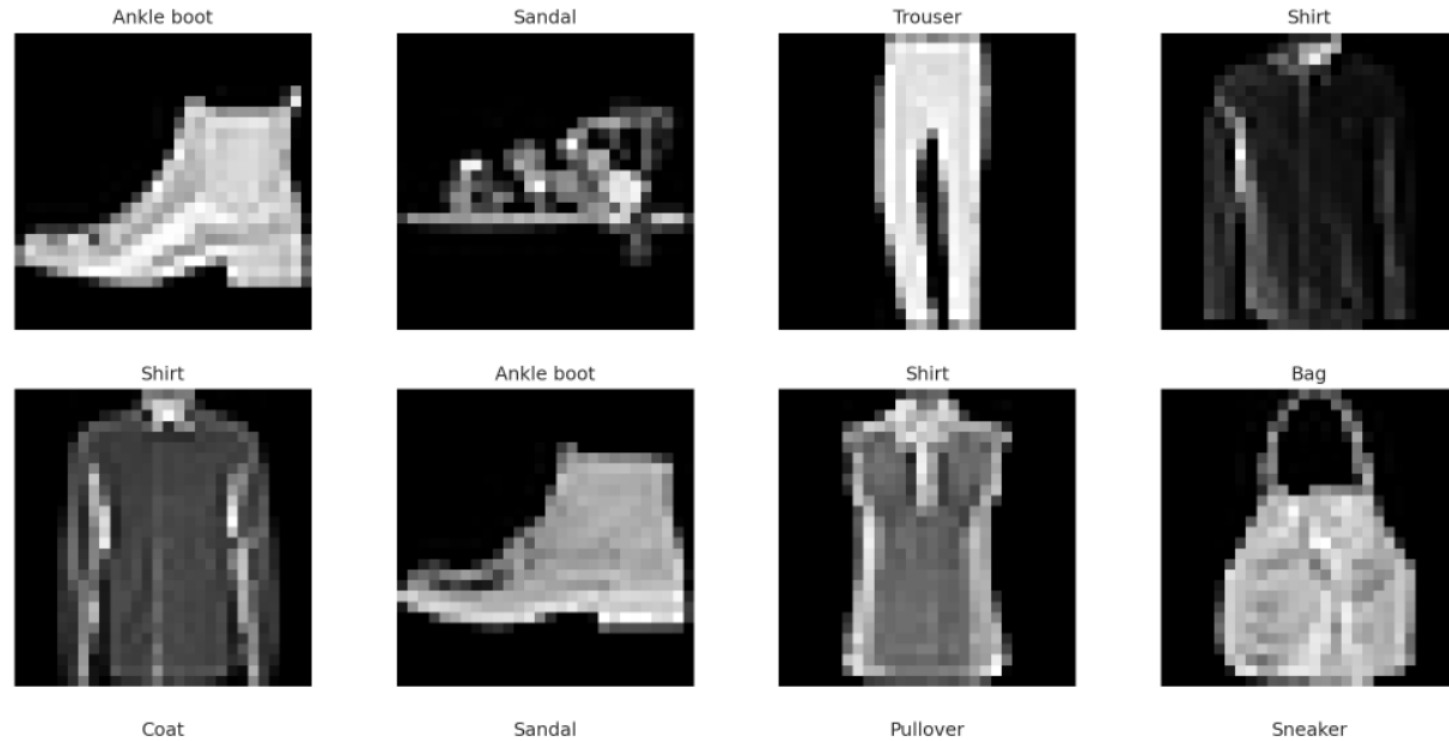


# VISUALIZING THE DATASET



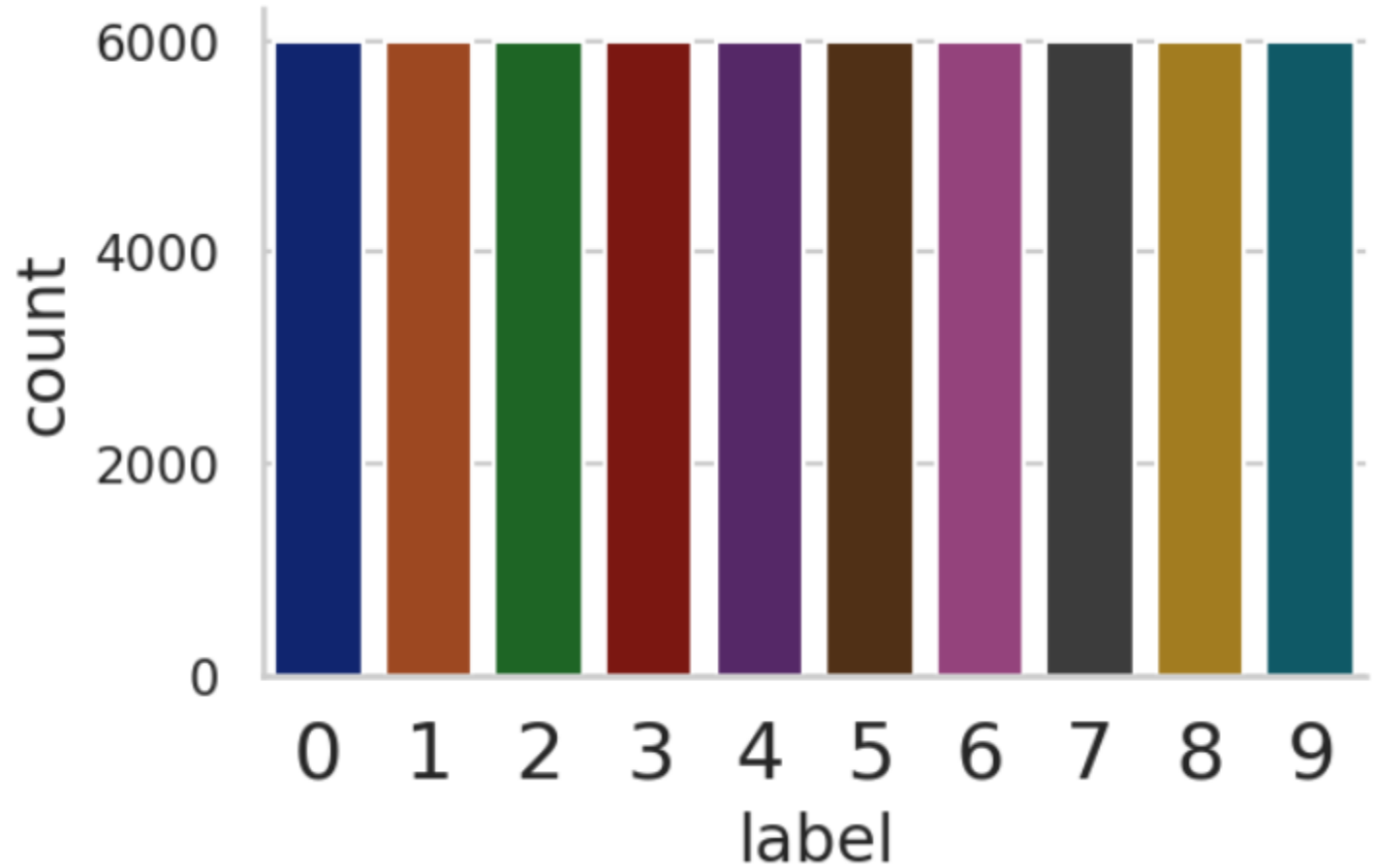
# PLOTTING RANDOM IMAGES

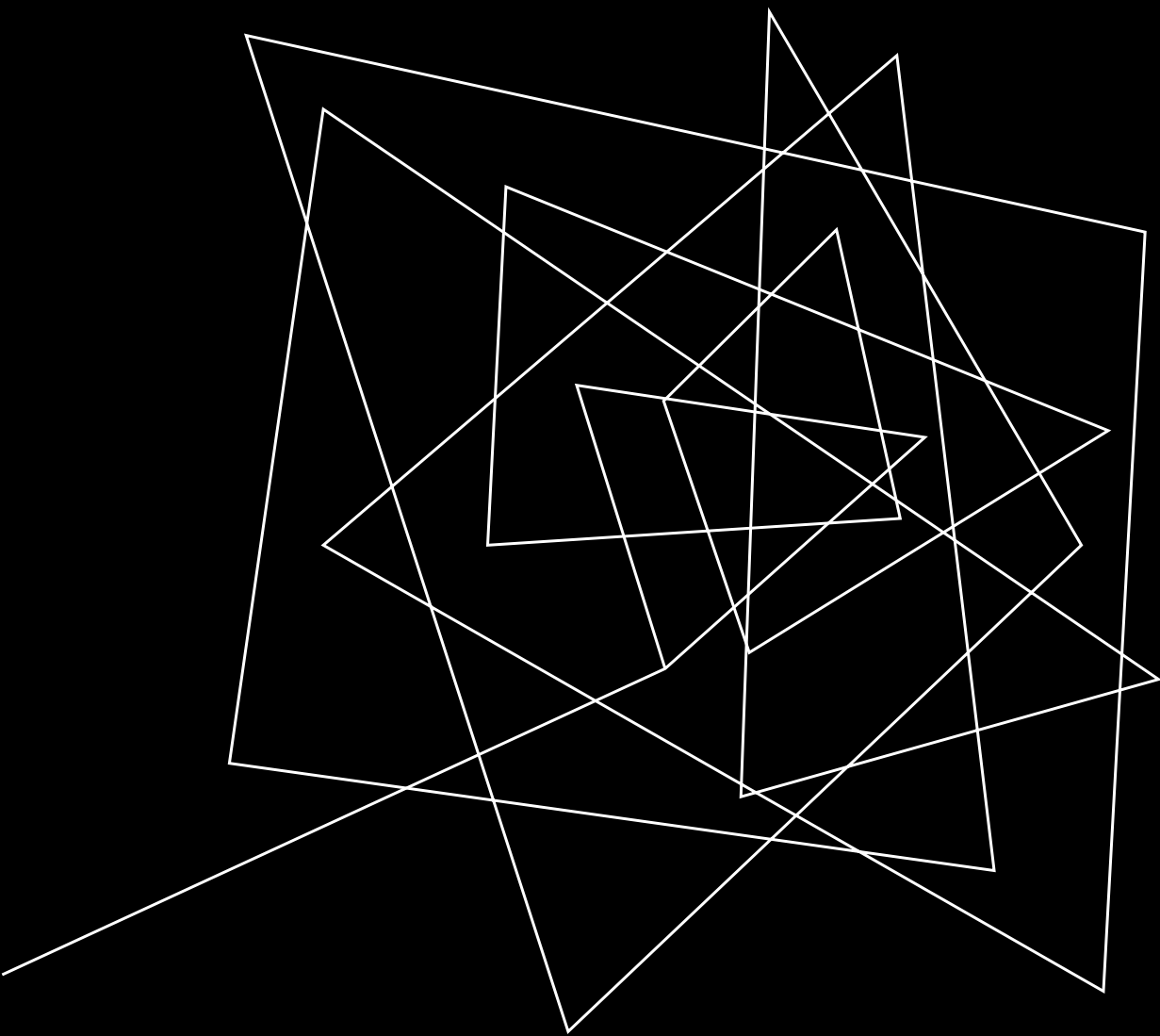
```
fig, axes = plt.subplots(4, 4, figsize = (15,15))
for row in axes:
    for axe in row:
        index = np.random.randint(60000)
        img = df.drop('label',
axis=1).values[index].reshape(28,28)
        cloths = df['label'][index]
        axe.imshow(img, cmap='gray')
        axe.set_title(clothing[cloths])
        axe.set_axis_off()
```



## DISTRIBUTION OF LABELS

- We can see that all classes are equally Distributed.
- So, there is no need for OverSampling or UnderSampling.





# DATA PRE-PROCESSING

# DATA PRE-PROCESSING

## Splitting Data into Train and Validation Set:

- Now we are gonna split the training data into Train and Validation Set. Train set is used for Training the model and Validation set is used for Evaluating our Model's Performance on the Dataset.
- This is achieved using the `train_test_split` method of scikit learn library.

## Reshaping the Images:

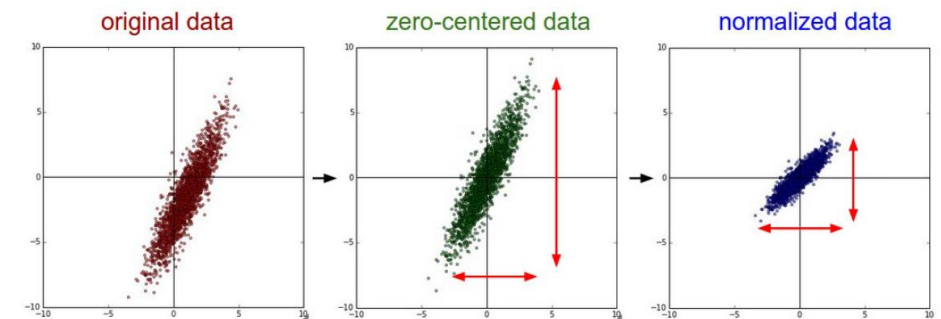
- Note that we have Images as 1D vector each containing 784 pixels. Before we feed the data to the CNN we must reshape the data into (28x28x1) 3D matrices.
- This is because Keras wants an Extra Dimension in the end, for channels. If this had been RGB images, there would have been 3 channels, but as MNIST is gray scale it only uses one.

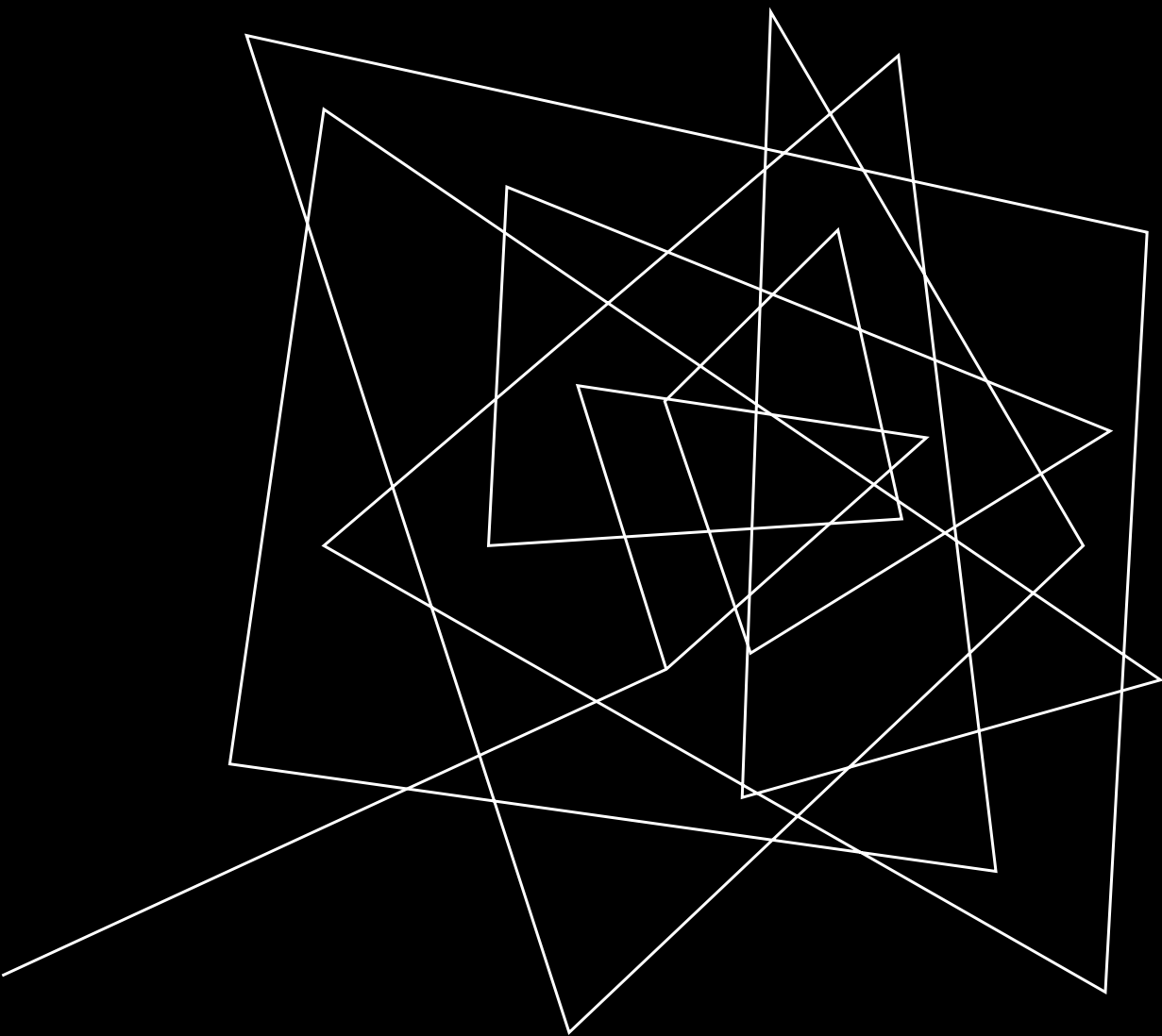
## Normalization:

The Pixel Values are often stored as **Integer** Numbers in the range 0 to 255, the range that a single 8-bit byte can offer. They need to be scaled down to [0,1] in order for Optimization Algorithms to work much faster. Here, we achieve Zero Mean and Unit Variance.

## One Hot Encoding

The labels are given as integers between 0-9. We need to one hot encode them, Eg 8 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0].





MODELS

# CONVNET

1) At First, we use Sequential Keras API which is just a linear stack of layers. We add one layer at a time starting from input.

2) Next We add Convolutional Layers, which are the Building blocks of ConvNets. Convolutional Layers has set of Independent Filters whose depth is equal to Input and other dimensions can be set manually. These Filters when convolved over the Input Image produce Feature Maps.

It includes some HyperParameters such as The number of filters, Dimensions of Filter (F), Stride (S), Padding(P) , Activation Function etc. which we input manually. Let the Input Volume Size be denoted by (W) ,

Then, the Output will have Dimensions given by -->

$$(\text{Height, Width}) = ((W - F + 2P) / S) + 1$$

And the Depth will be equal to Number of Filters Specified.

3) Next We add Pooling Layers, which are used for Dimensionality Reduction or DownSampling the Input. These are used where we have lot of Input Features. It reduces the amount of Parameters and Computational power required drastically, thus reducing Overfitting. These along with Convolutional layers are able to learn more Complex features of the Image.

4) We add Batch Normalization where we achieve Zero mean and Variance one. It scales down outliers and forces the network to learn features in a distributed way, not relying too much on a Particular Weight and makes the model better Generalize the Images.

5) To avoid Overfitting We add Dropout. This randomly drops some percentage of neurons, and thus the weights gets Re-Aligned. The remaining Neurons learn more features and this reduces the dependency on any one Neuron. Dropout is a Regularization Technique, which Penalizes the Parameters. Generally we set the DropOutRate between 0.2-0.5 .

6) Finally we add Flatten layer to map the input to a 1D vector. We then add Fully connected Layers after some convolutional/pooling layers. It combines all the Features of the Previous Layers.

7) Lastly, we add the Output Layer. It has units equal to the number of classes to be identified. Here, we use 'sigmoid' function if it is Binary Classification otherwise 'softmax' activation function in case of Multi-Class Classification.



# COMPILING THE MODEL

1) We need to compile the model. We have to specify the optimizer used by the model We have many choices like Adam, RMSprop etc.. Refer to Keras doc for a comprehensive list of the optimizers available.

2) Next we need to specify the loss function for the neural network which we want to minimize.

For Binary Classification we use "binary\_crossentropy" and for Multi-class Classification we use "categorical\_crossentropy".

3) Finally, We need to specify the metric to evaluate our models performance. Here I have used accuracy.



# MODEL SUMMARY

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
batch_normalization (Batch Normalization)	(None, 28, 28, 32)	128
conv2d_1 (Conv2D)	(None, 28, 28, 32)	9248
batch_normalization_1 (Batch Normalization)	(None, 28, 28, 32)	128
dropout (Dropout)	(None, 28, 28, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 14, 14, 128)	73856
batch_normalization_2 (Batch Normalization)	(None, 14, 14, 128)	512

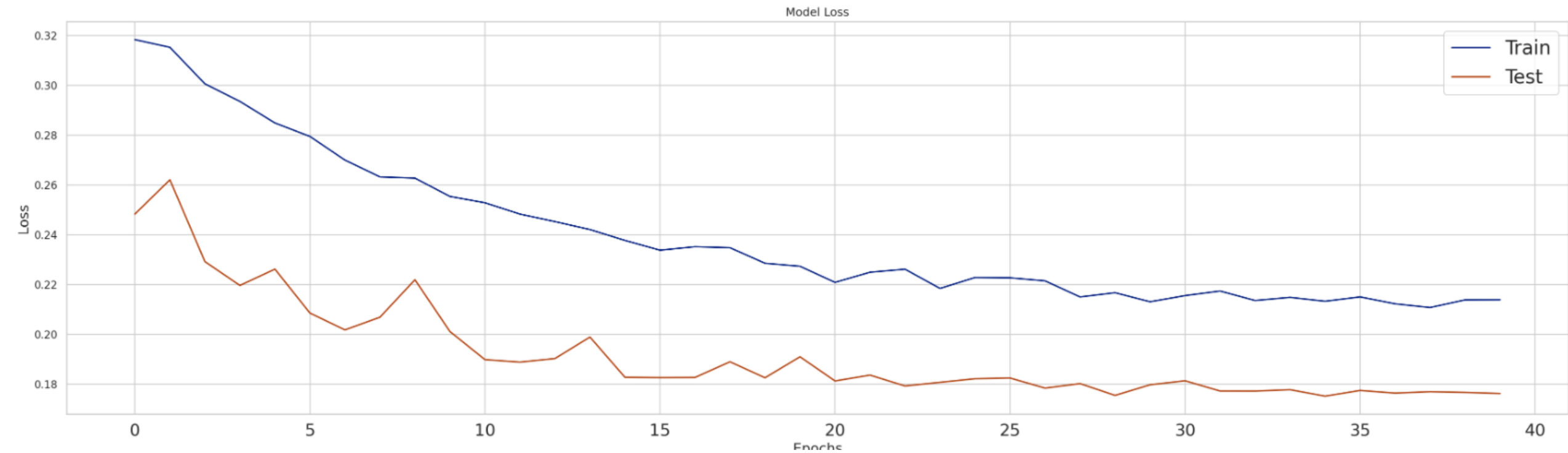
dropout_2 (Dropout)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12845568
batch_normalization_3 (Batch Normalization)	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
batch_normalization_4 (Batch Normalization)	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

=====  
Total params: 13,017,770  
Trainable params: 13,016,106  
Non-trainable params: 1,664  
=====

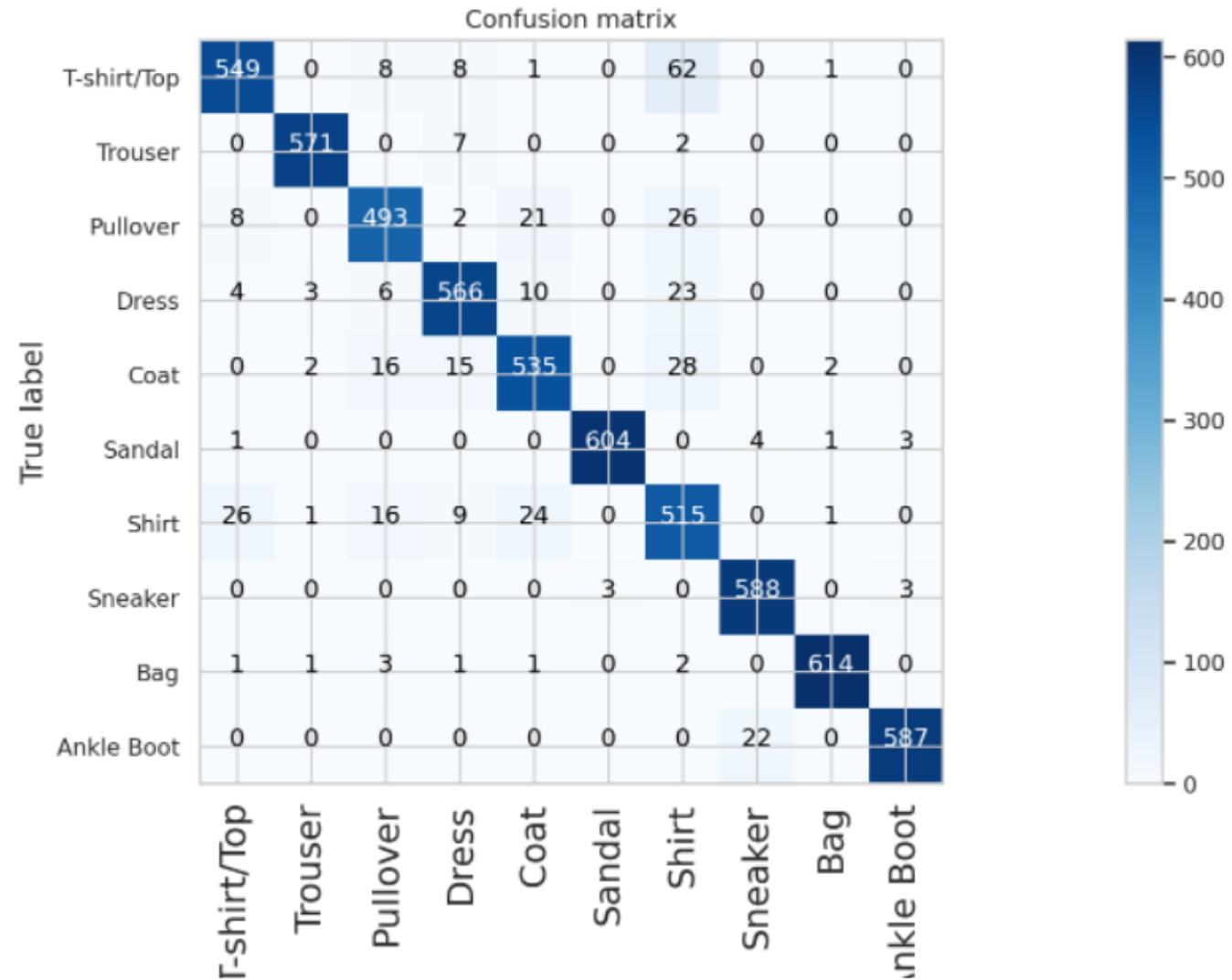
# DATA AUGMENTATION

```
datagen = ImageDataGenerator(  
    rotation_range = 8, # randomly rotate images in the  
range (degrees, 0 to 180)  
    zoom_range = 0.1, # Randomly zoom image  
    shear_range = 0.3, # shear angle in counter-  
clockwise direction in degrees  
    width_shift_range=0.08, # randomly shift images hori-  
zontally (fraction of total width)  
    height_shift_range=0.08, # randomly shift images ver-  
tically (fraction of total height)  
    vertical_flip=True) # randomly flip images
```

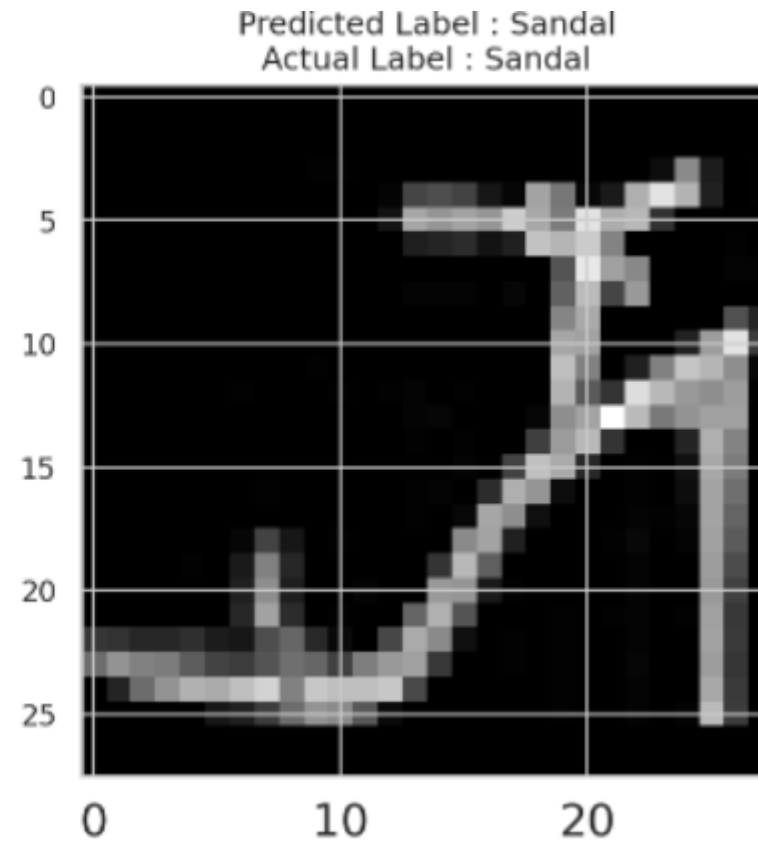
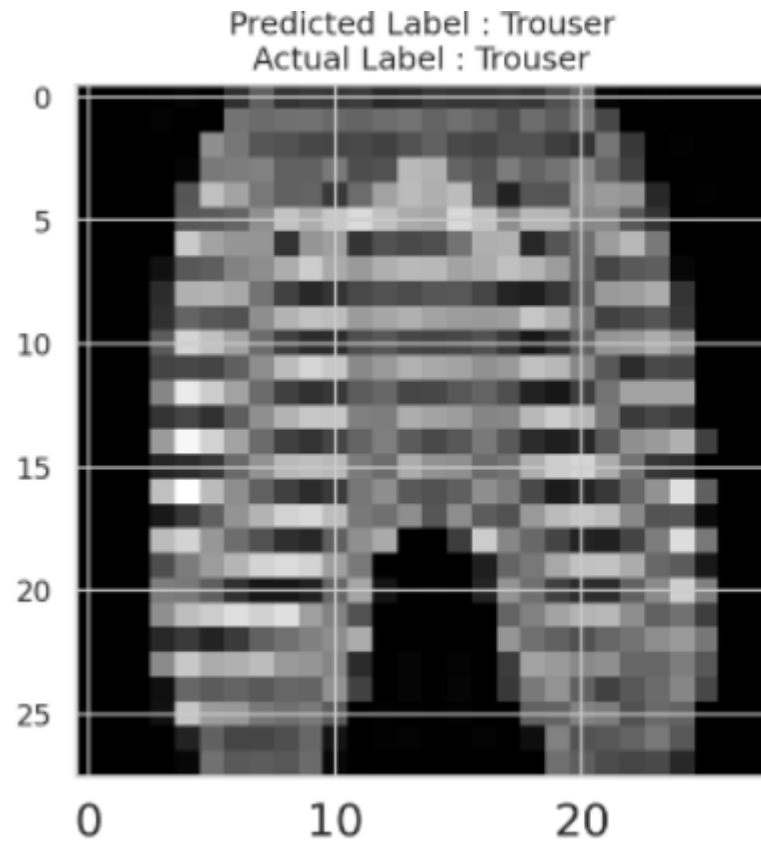
# PLOTTING THE TRAINING AND VALIDATION CURVES



# CONFUSION MATRIX



# VISUALIZATION OF PREDICTED CLASSES



# CLASSIFICATION REPORT

The classification report visualizer displays the precision, recall, F1, and support scores for the model.

•**Precision:**

Precision is the ability of a classifier not to label an instance positive that is actually negative. Basically, it is defined as the ratio of true positives to the sum of true and false positives. “For all instances classified positive, what percent was correct?”

•**Recall:**

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. “For all instances that were actually positive, what percent was classified correctly?”

•**F1 Score:**

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0 . Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation.

•**Support:**

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing.

# CLASSIFICATION REPORT

	precision	recall	f1-score	support
T-shirt/Top	0.93	0.87	0.90	629
Trouser	0.99	0.98	0.99	580
Pullover	0.91	0.90	0.90	550
Dress	0.93	0.92	0.93	612
Coat	0.90	0.89	0.90	598
Sandal	1.00	0.99	0.99	613
Shirt	0.78	0.87	0.82	592
Sneaker	0.96	0.99	0.97	594
Bag	0.99	0.99	0.99	623
Ankle Boot	0.99	0.96	0.98	609
accuracy			0.94	6000
macro avg	0.94	0.94	0.94	6000
weighted avg	0.94	0.94	0.94	6000

A series of white, thin, overlapping geometric lines on a black background, forming a complex, abstract shape on the left side of the slide.

THANK YOU