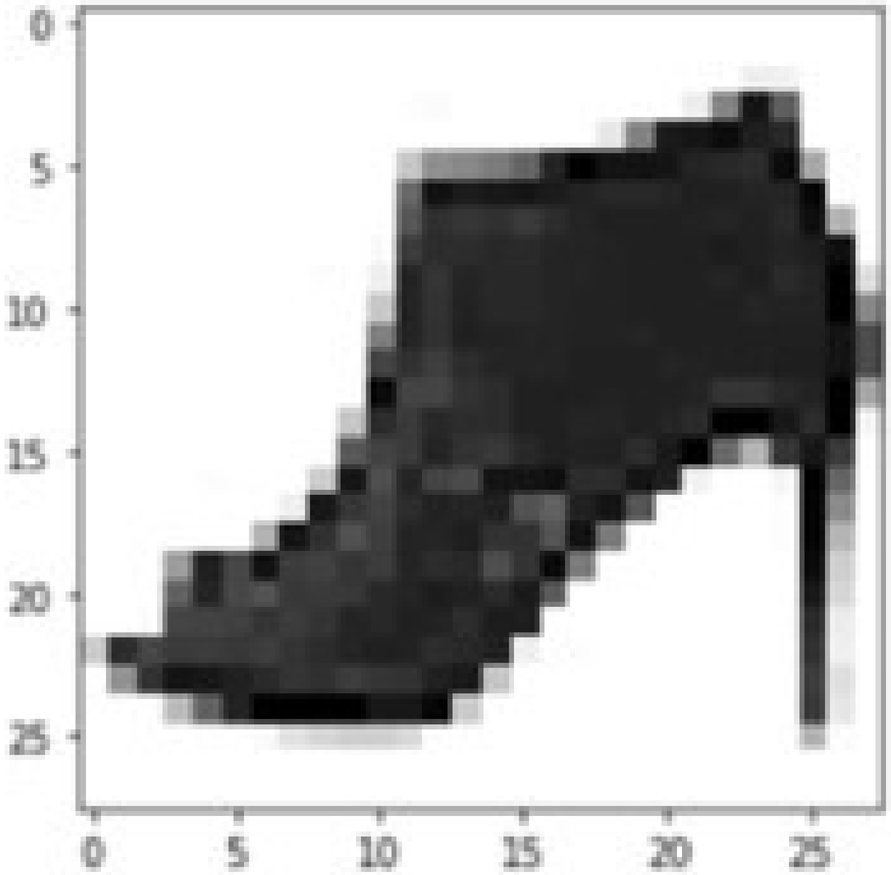
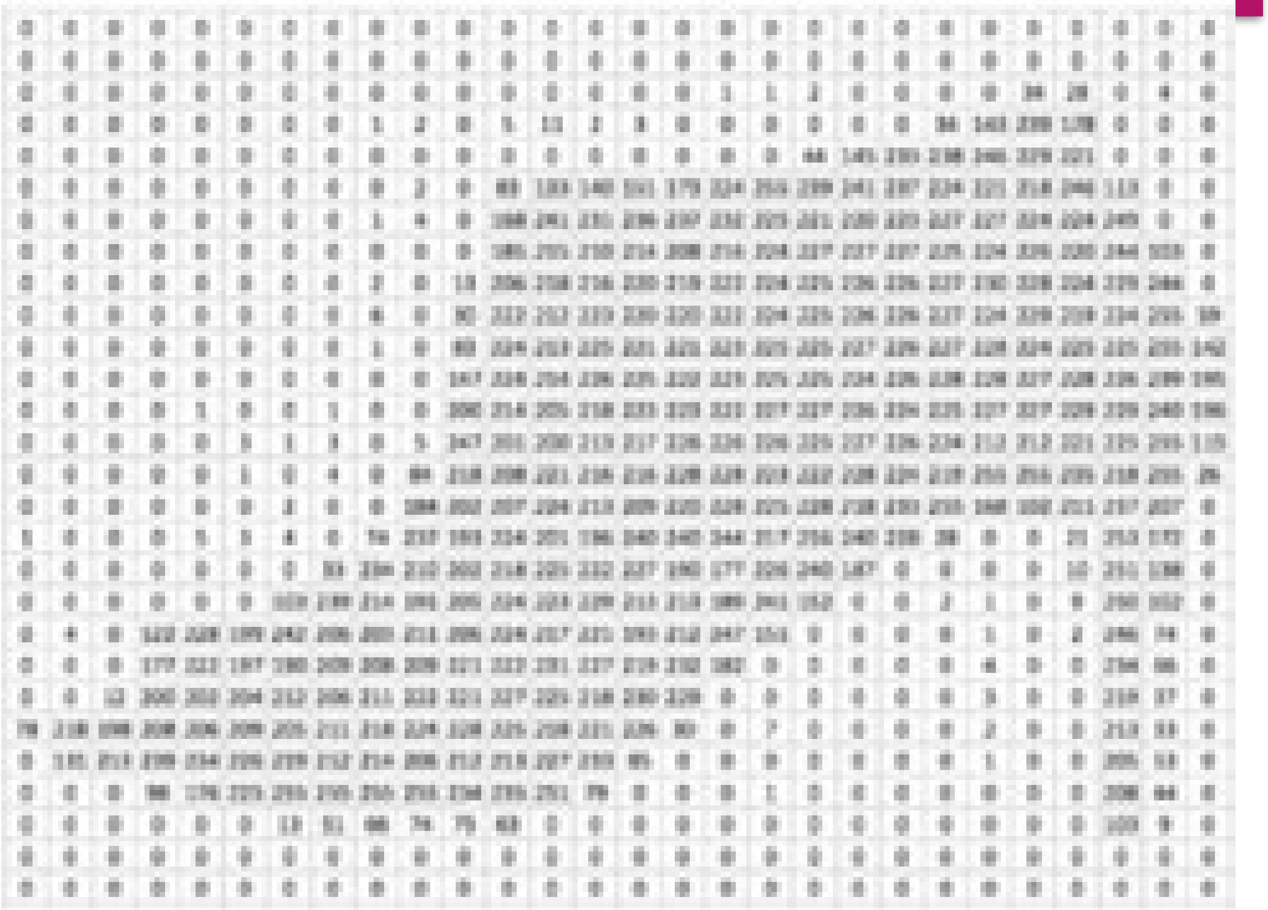
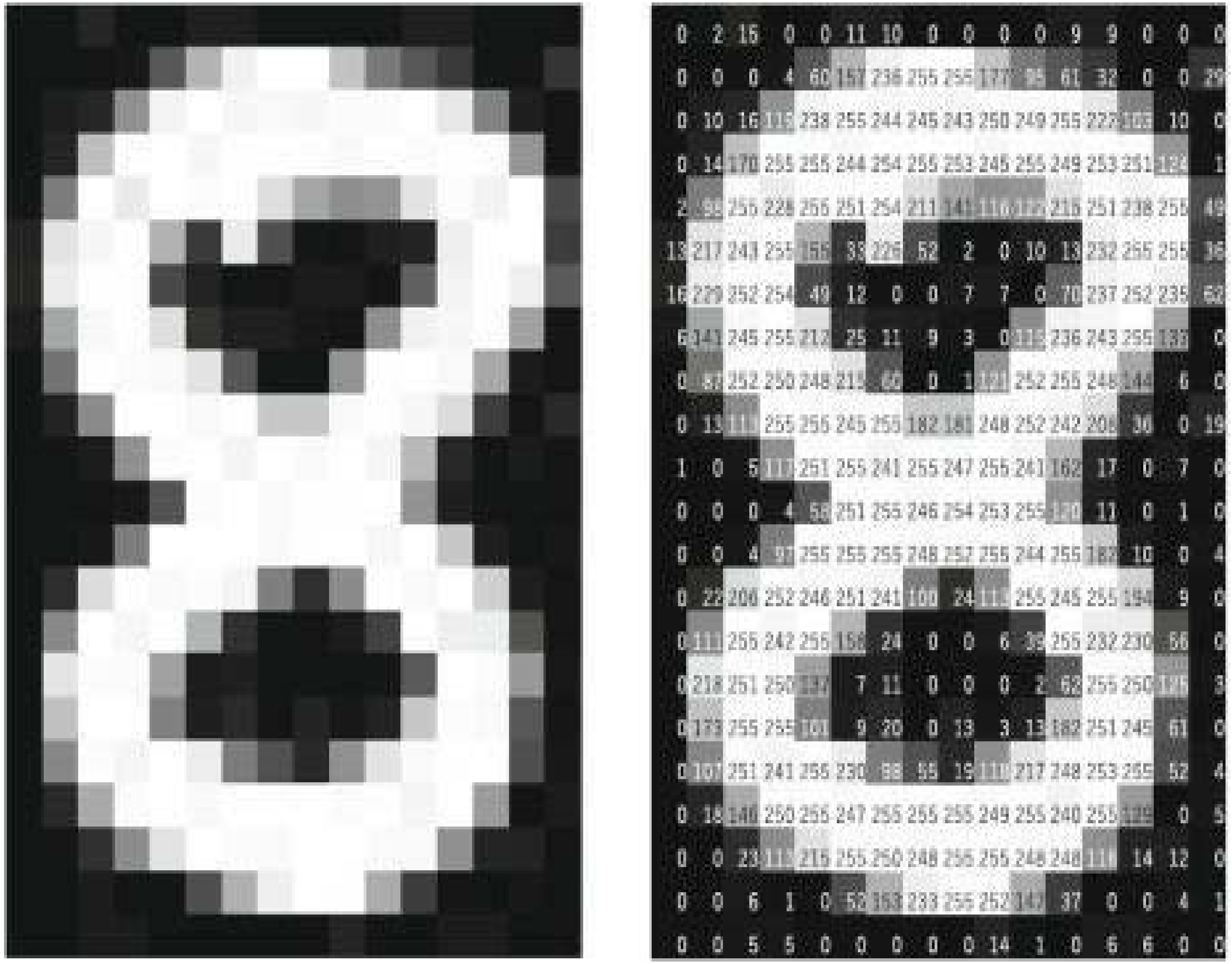
Convolutional Neural Network for FASHION MNIST DATASET

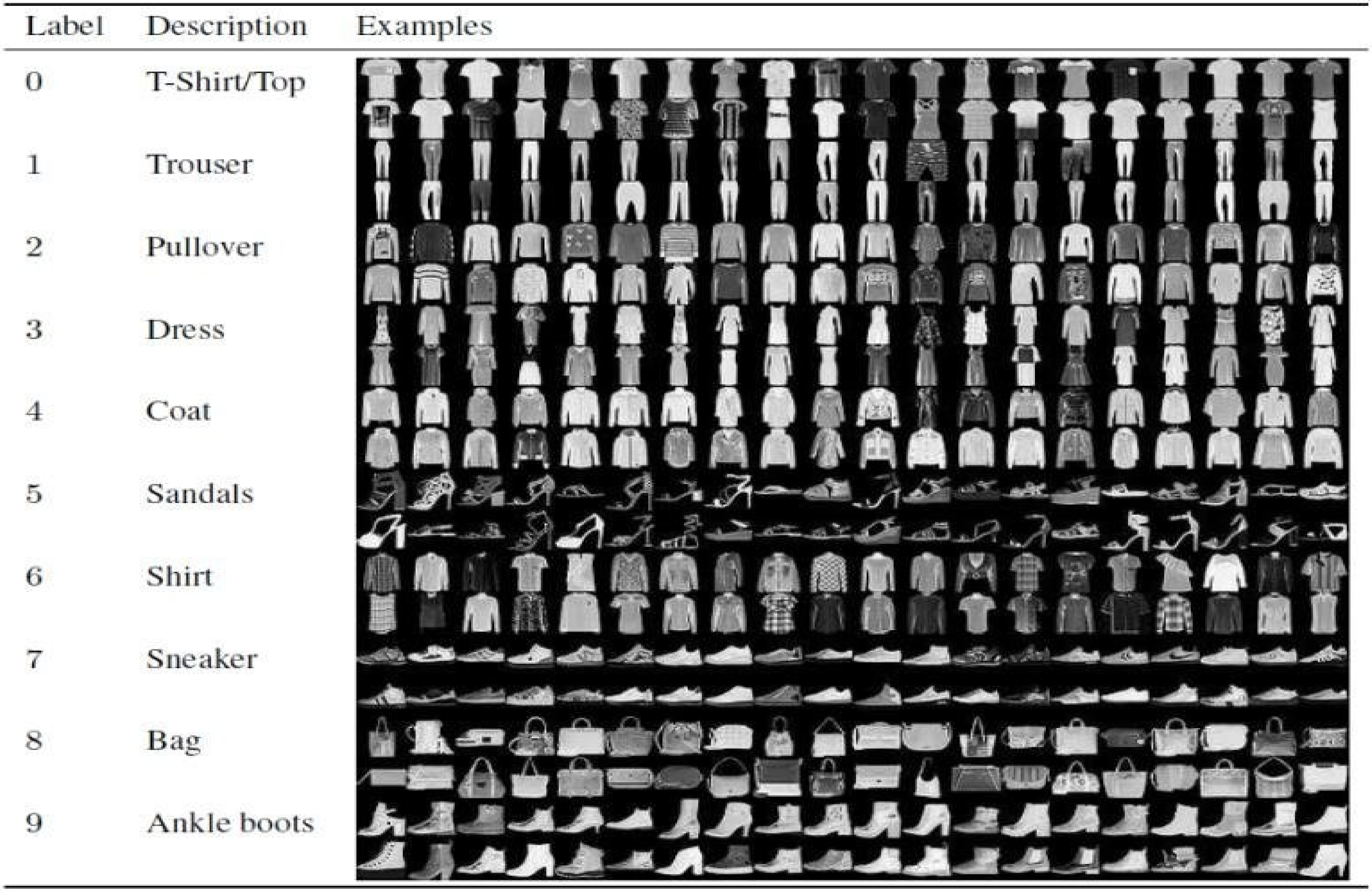




0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	0	0	0	0	34	28	0	4	0
0	0	0	0	0	0	0	0	1	2	0	5	11	2	3	0	0	0	0	0	0	34	143	209	178	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	145	193	238	246	229	221	0	0	0
0	0	0	0	0	0	0	0	0	2	0	83	183	140	151	179	224	255	239	241	237	224	221	218	246	113	0	0
0	0	0	0	0	0	0	0	1	4	0	198	241	231	236	237	232	225	221	220	223	227	227	224	224	249	0	0
0	0	0	0	0	0	0	0	0	0	0	185	215	210	214	208	214	224	227	227	227	225	224	226	220	244	103	0
0	0	0	0	0	0	0	0	2	0	13	206	218	216	220	219	222	224	225	226	226	227	230	228	224	229	246	0
0	0	0	0	0	0	0	0	6	0	90	222	212	223	220	220	222	224	225	226	226	227	224	229	219	234	255	18
0	0	0	0	0	0	0	0	1	0	89	224	213	225	221	221	223	223	225	227	229	227	228	224	220	231	255	142
0	0	0	0	0	0	0	0	0	0	167	226	214	226	225	222	223	225	225	224	226	228	228	227	228	226	239	195
0	0	0	0	1	0	0	1	0	0	200	214	205	218	233	225	222	227	227	226	224	225	227	227	228	229	240	196
0	0	0	0	0	3	1	3	0	5	247	201	200	213	217	226	226	226	225	227	226	224	212	212	221	225	255	115
0	0	0	0	0	1	0	4	0	84	218	208	221	226	218	228	228	223	222	228	224	219	255	255	235	218	255	26
0	0	0	0	0	0	2	0	0	188	202	207	224	213	209	223	226	225	228	218	230	233	168	102	211	237	207	0
1	0	0	0	5	3	4	0	74	237	193	224	201	196	240	240	244	217	236	240	239	38	0	0	21	253	172	0
0	0	0	0	0	0	0	33	234	212	202	218	220	222	227	190	177	220	240	187	0	0	0	0	10	251	138	0
0	0	0	0	0	0	103	239	214	191	205	224	223	229	213	213	180	241	152	0	0	2	1	0	8	250	102	0
0	4	0	122	228	195	242	206	203	211	206	224	217	221	193	212	247	151	0	0	0	0	1	0	2	246	74	0
0	0	0	177	222	197	190	209	208	209	221	222	231	227	219	232	182	0	0	0	0	0	4	0	0	234	66	0
0	0	12	200	202	204	212	208	211	222	221	227	225	218	230	228	0	0	0	0	0	0	3	0	0	218	37	0
78	218	198	208	206	209	205	211	218	224	228	225	228	221	226	80	0	7	0	0	0	0	2	0	0	213	33	0
0	131	213	239	234	226	229	212	214	206	212	216	227	233	85	0	0	0	0	0	0	0	1	0	0	205	53	0
0	0	0	98	176	225	255	255	250	255	234	255	251	79	0	0	0	1	0	0	0	0	0	0	0	208	44	0
0	0	0	0	0	0	13	51	66	74	75	63	0	0	0	0	0	0	0	0	0	0	0	0	0	103	9	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



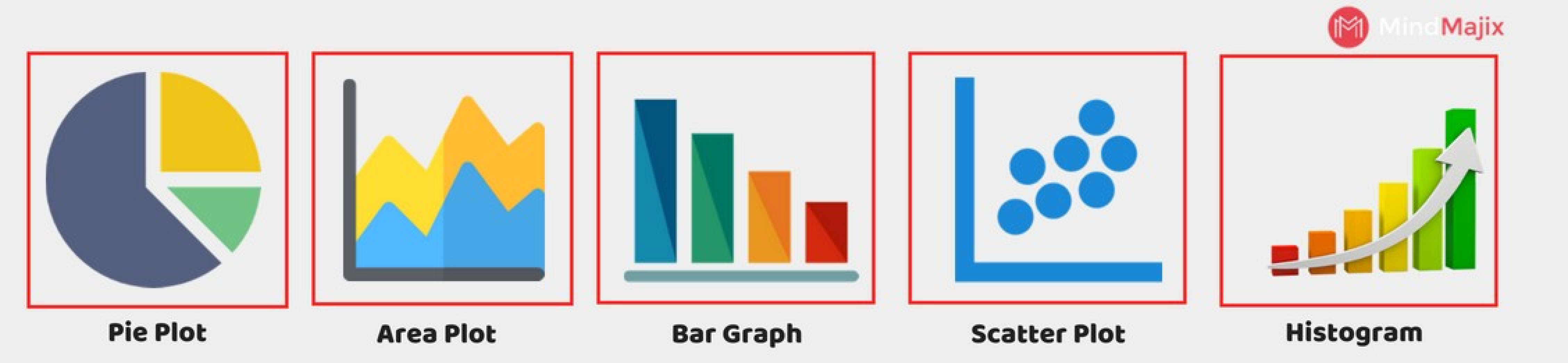
0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0



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	

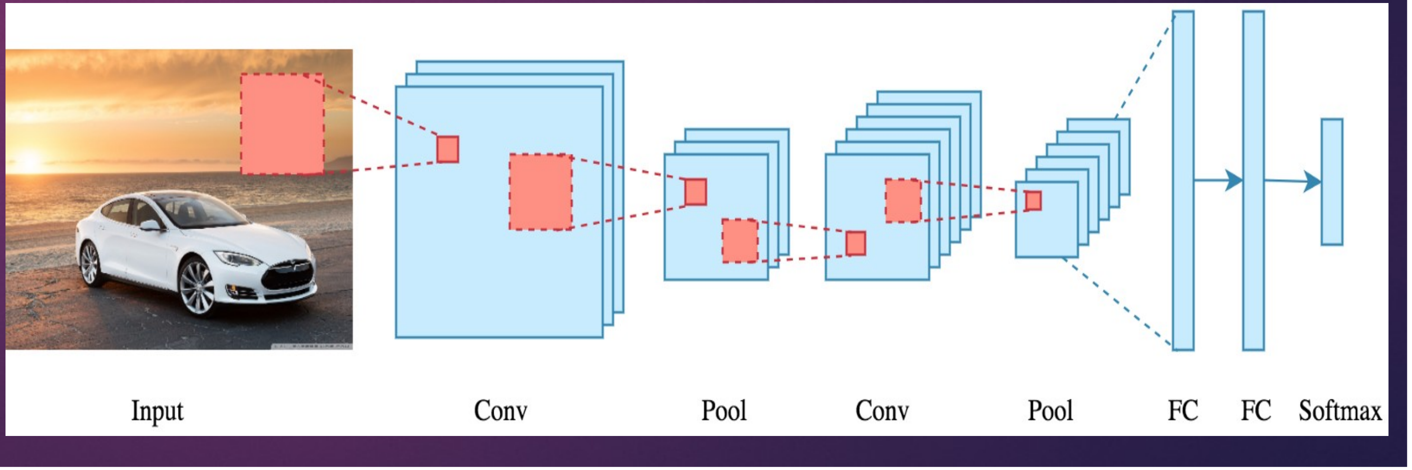
Matplotlib

Matplotlib is a Python 2D plotting library that produces high-quality charts and figures, which helps us visualize extensive data to understand better. Pandas is a handy and useful data-structure tool for analyzing large and complex data.And **pyplot** function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.



CONVOLUTIONAL NEURAL NETWORK

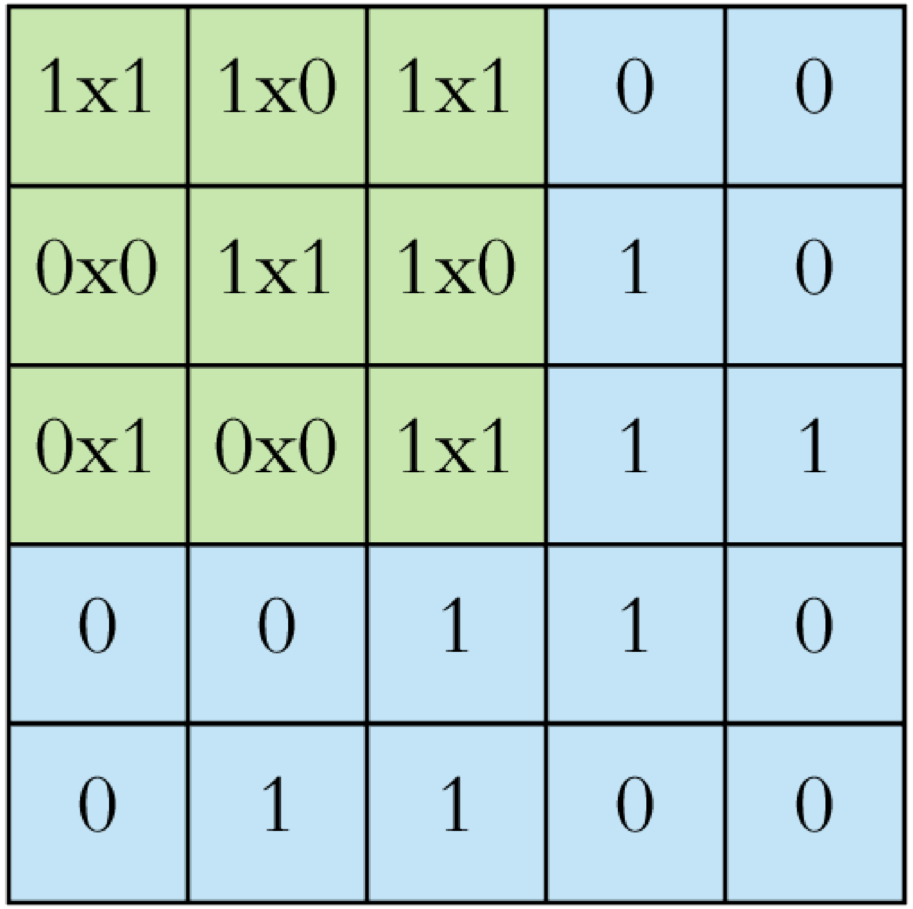
- Convolutional Neural Networks or CNN is a type of deep neural networks that are efficient at extracting meaningful information from visual imagery.
- The role of the CNN is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.



Parameters in CNN

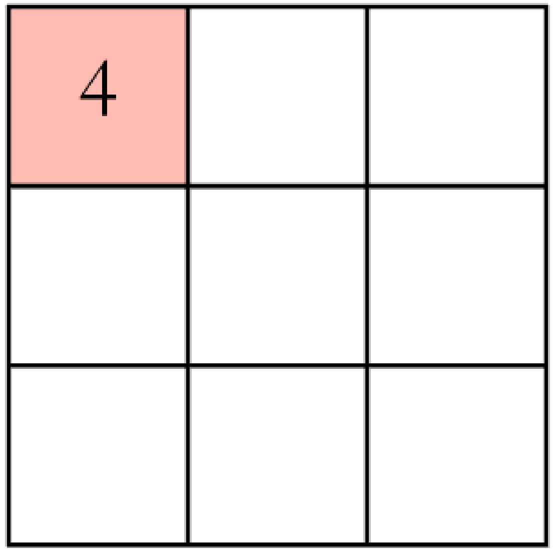
- **filters:** Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides:** The amount by which the filter shifts is the stride
- **padding:** one of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding with zeros evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input. Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems
- **activation**: Activation function to use. If you don't specify anything, no activation is applied

CNN layer working



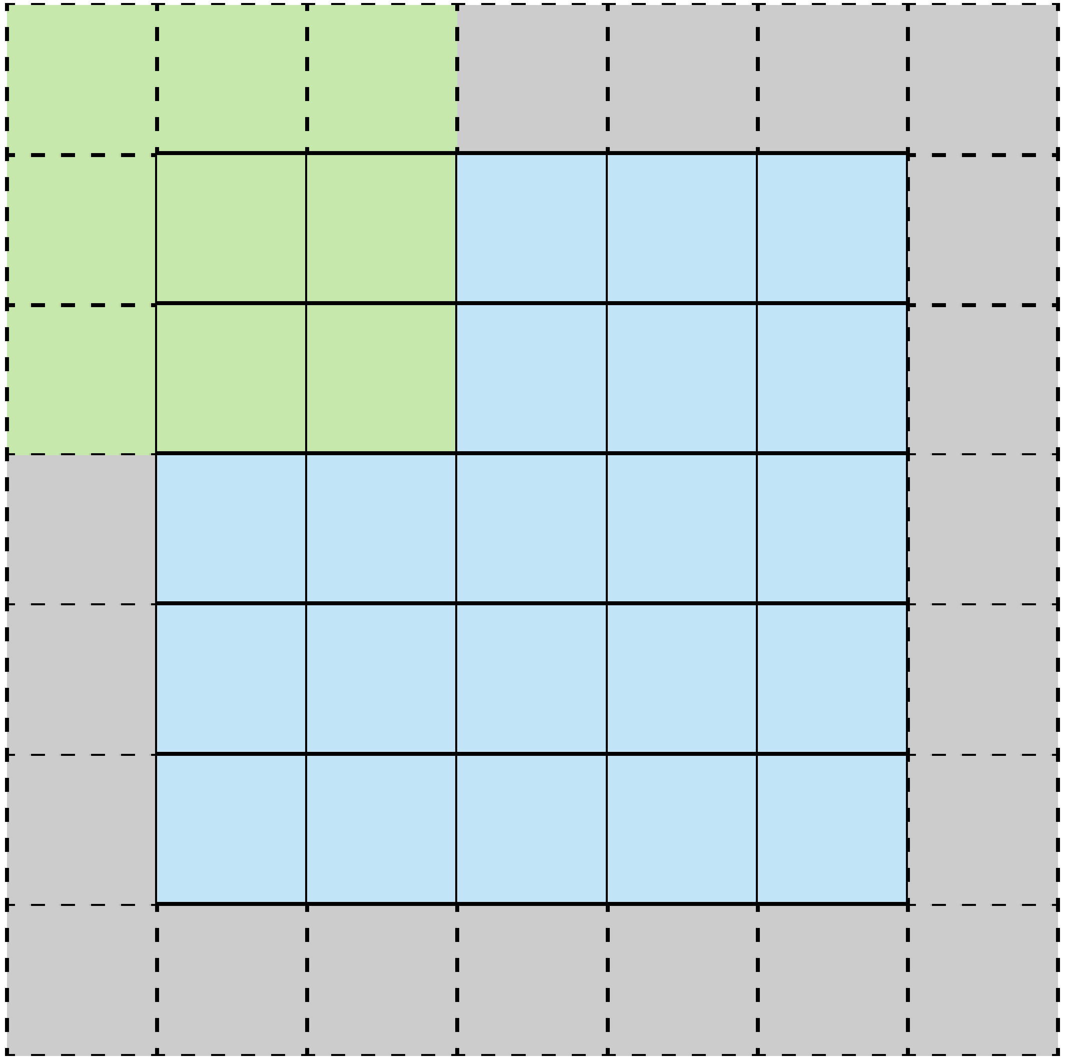
1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

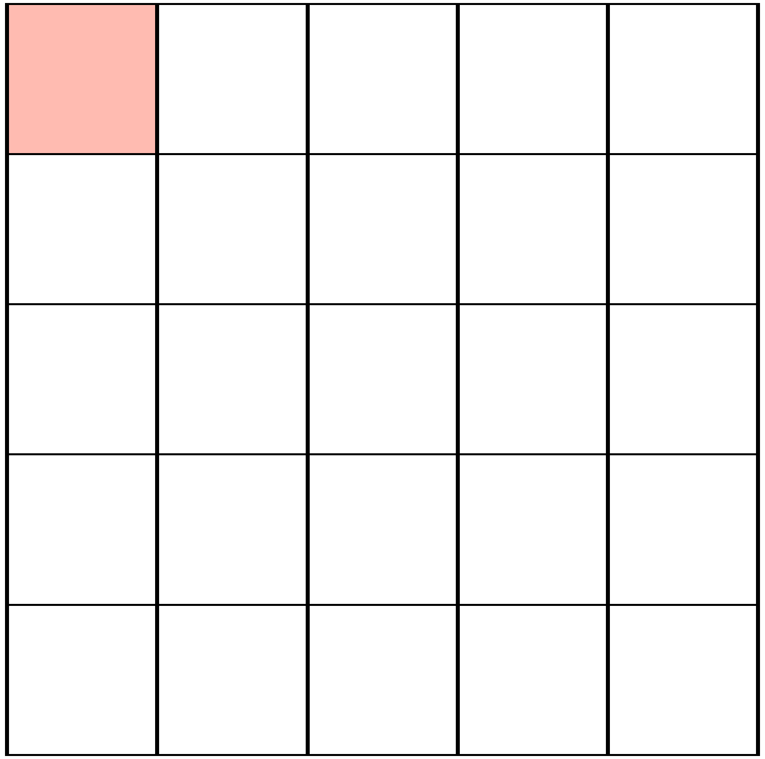


4		

Feature Map



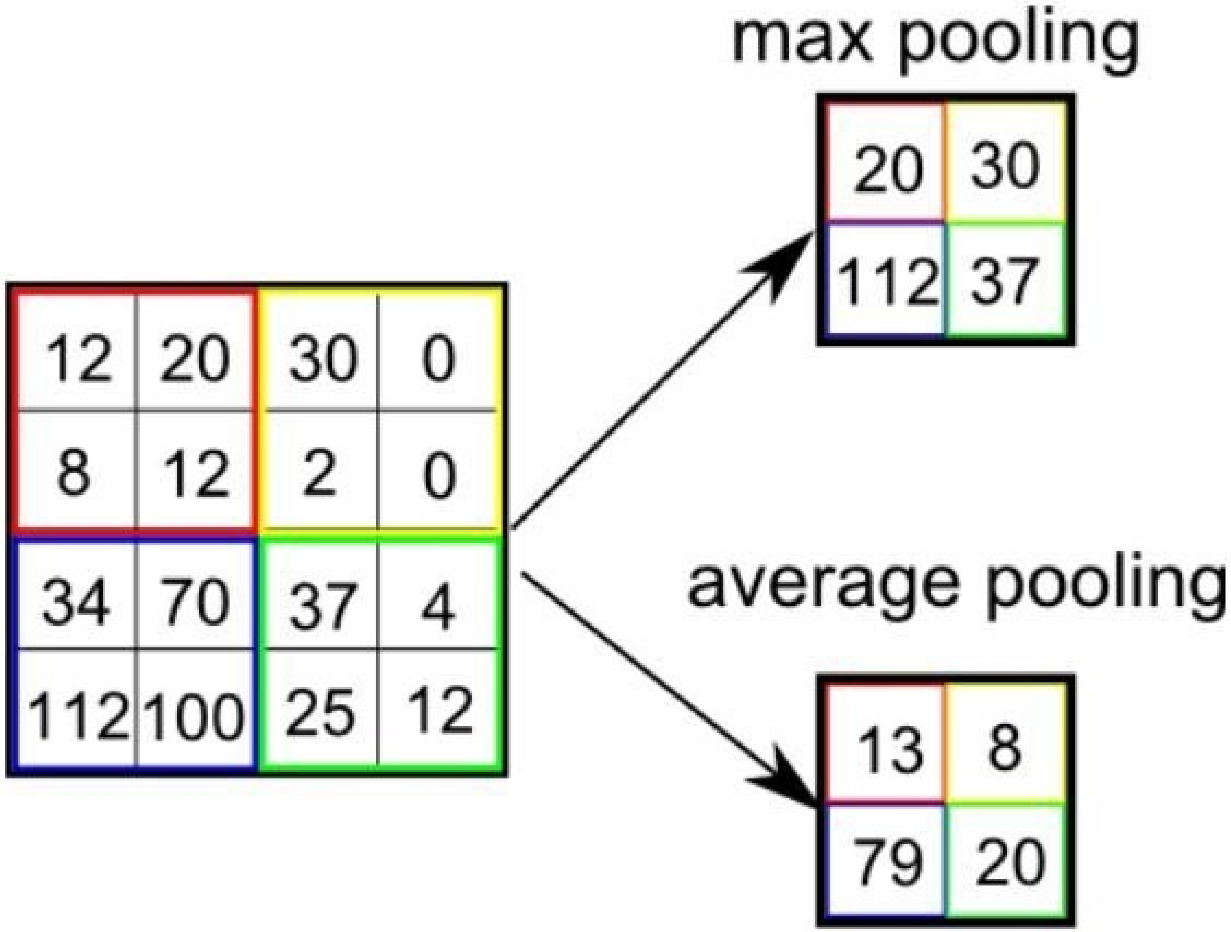
Stride 1 with Padding



Feature Map

POOLING LAYER:

- 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** by reducing the dimensions.
- There are two types of pooling average pooling and max pooling.



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

max pooling

20	30
112	37

average pooling

13	8
79	20

DENSE LAYER

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

Keras Dense Layer Parameters

Let us see main parameters of dense layer function of Keras below –

1. Units

The **most basic parameter** of all the parameters, it uses positive integer as it value and represents the **output size** of the layer.

It is the unit parameter itself that plays a major role in the **size of the weight matrix** along with the **bias vector**.

2. Activation

The activation parameter is helpful in applying the element-wise activation function in a dense layer. By default, Linear Activation is used but we can alter and switch to any one of many options that Keras provides for this.

ACTIVATION FUCNTIONS – RELU, SOFTMAX

RELU :

The **rectified linear activation function** or **ReLU** for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

SOFTMAX:

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1 (check it on the figure above).
The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

STEPS INVOLVED

- Import all the libraries ,packages
- Load the data from tensorflow or keras
- Split the data
- Pre-process the data into the appropriate form
  a) normalize
- Start creating model
- Add all the layers sequentially
- Compile, train and validate
- Predict and print the predicted labels with respective objects

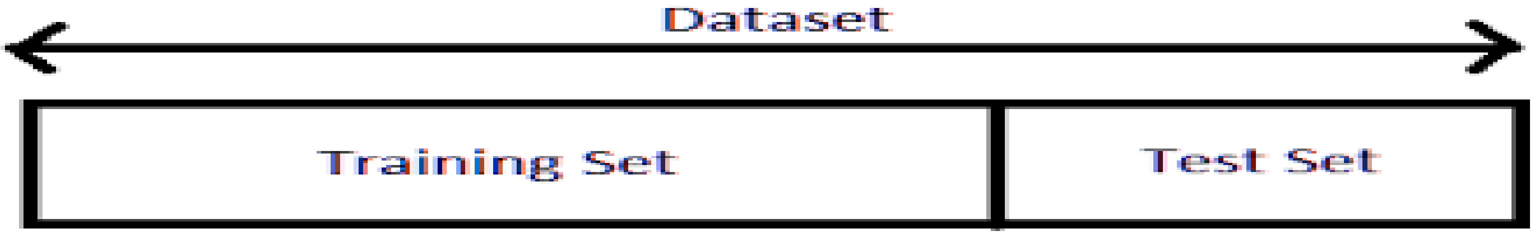
Importing dataset and splitting

LOADING DATASET

```
tf.keras.datasets.cifar10.load_data()
 Or
keras.datasets.cifar10.load_data()
```

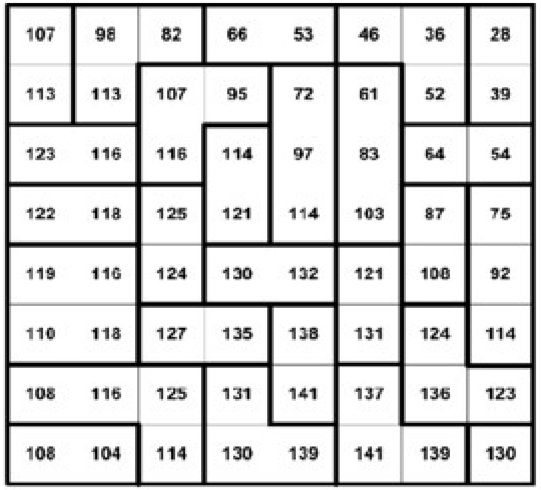
SPLITTING OF DATASET

Further the data is splitted into training and testing set based on size of the dataset .



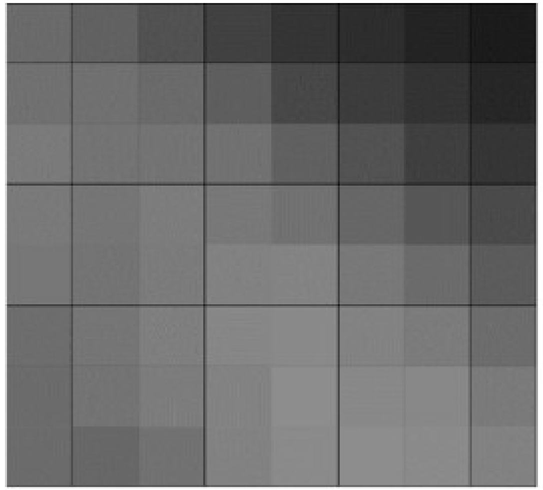
PRE-PROCESSING - NORMALIZATION

- In image processing, **normalization** is a process that changes the range of pixel intensity values.
- **Normalization** is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network
- Most probably the images are converted to gray scale as the pixels range will be 0-255 only



107	98	82	66	53	46	36	28
113	113	107	95	72	61	52	39
123	116	116	114	97	83	64	54
122	118	125	121	114	103	87	75
119	116	124	130	132	121	108	92
110	118	127	135	138	131	124	114
108	116	125	131	141	137	136	123
108	104	114	130	139	141	139	130

a



b

IMPORTING LAYERS and ADDING LAYERS

FROM KERAS

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten
```

From tensorflow

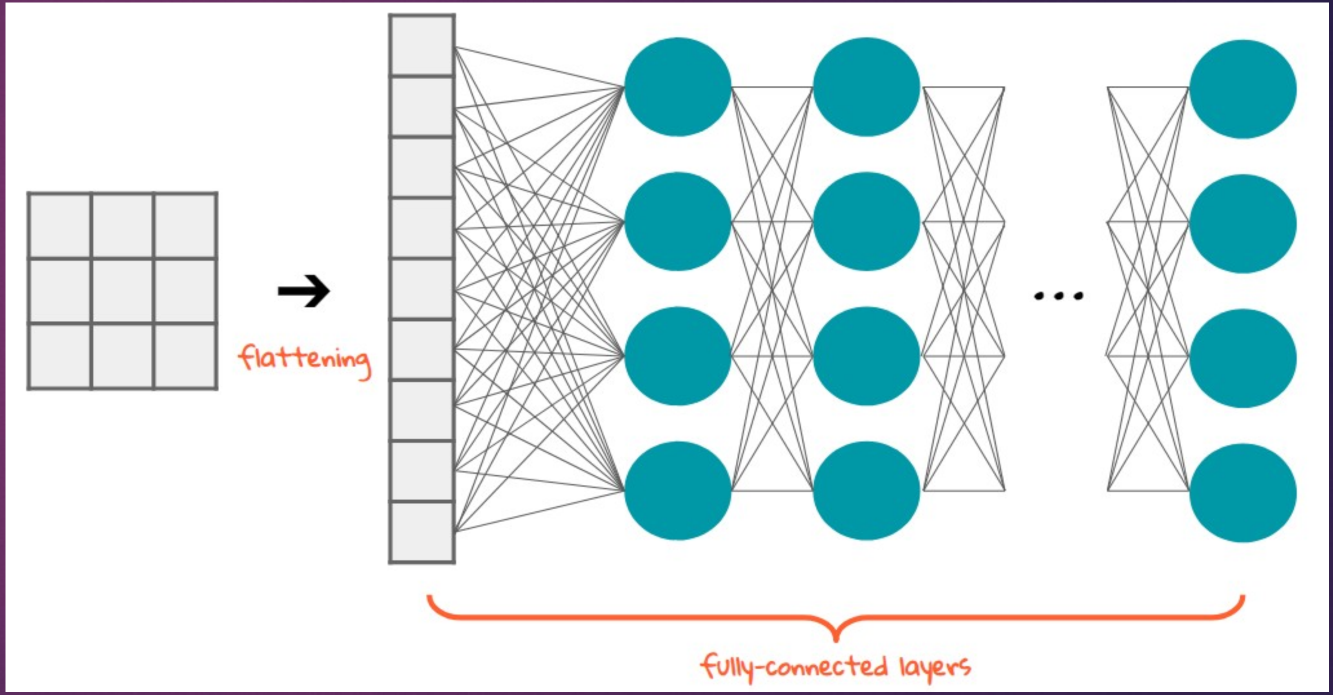
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten,
 MaxPooling2D
```

We add layer my using add () function in sequential order as we chose sequential model

FLATTEN LAYER

- **Flattening** is converting the data into a 1-dimensional array for inputting it to the next **layer**. We **flatten** the output of the convolutional **layers** to create a single long feature vector.

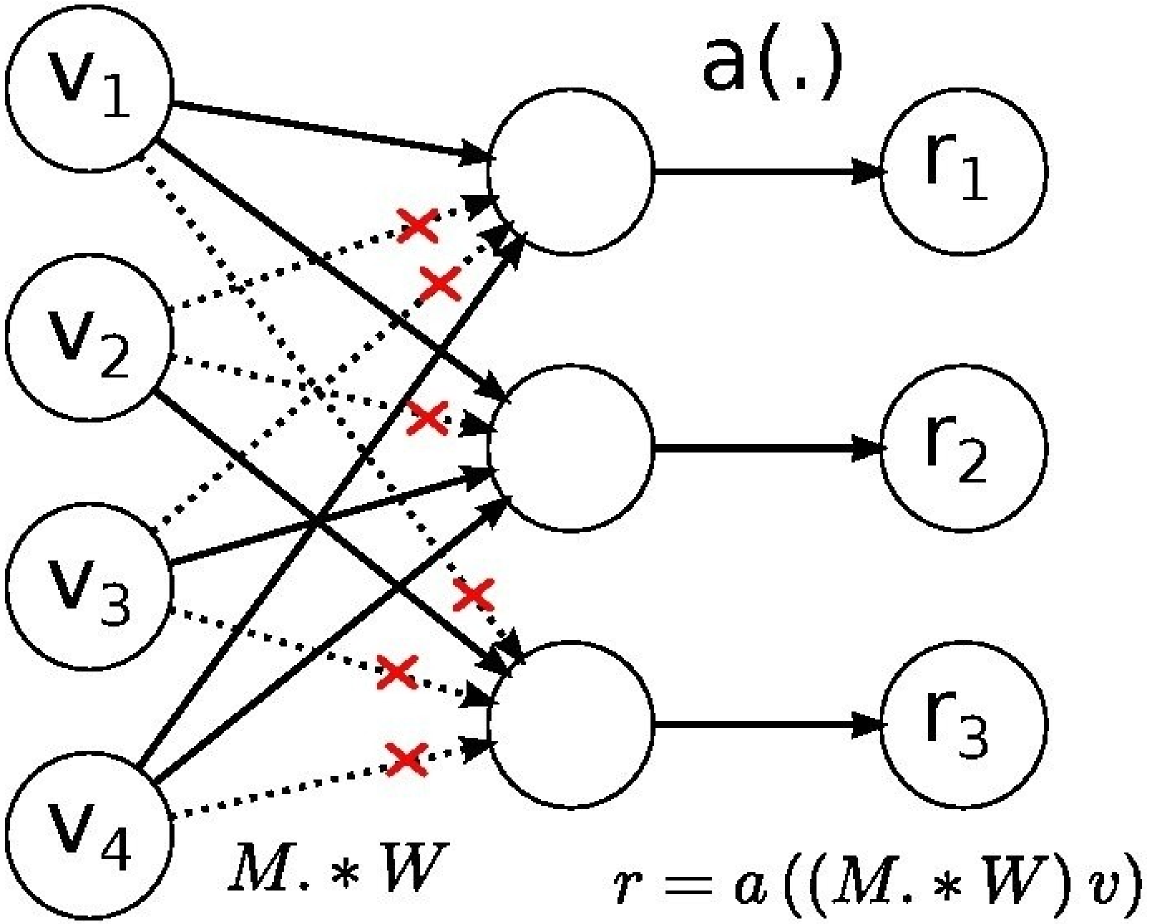
- And it is connected to the final classification model, which is called a fully-connected **layer**

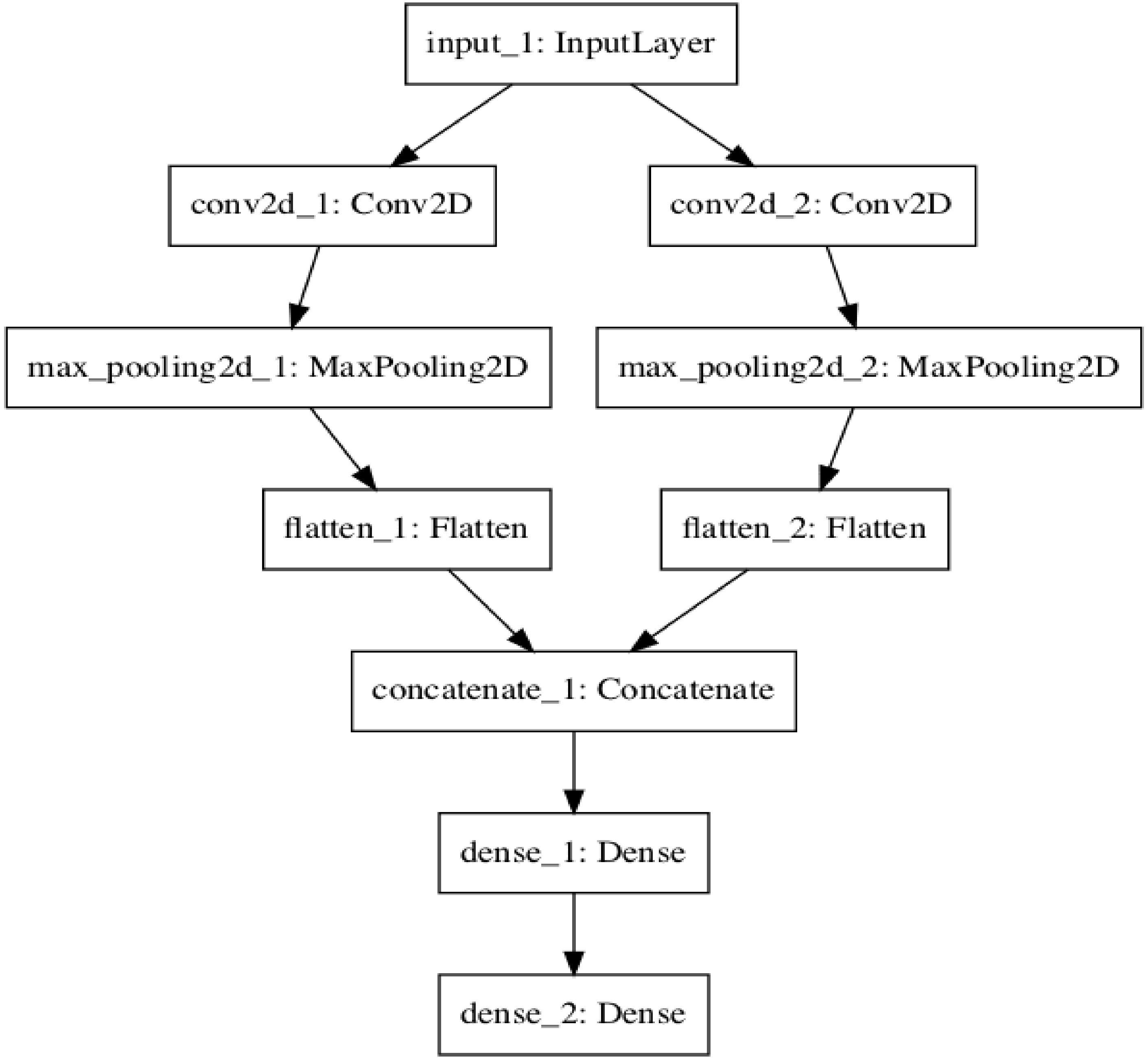


Dropout

Dropout is a way to regularize the neural network. During training, it may happen that neurons of a particular layer may always become influenced only by the output of a particular neuron in the previous layer. In that case, the neural network would overfit.

Dropout prevents overfitting and regularizes by randomly cutting the connections (also known as dropping the connection) between neurons in successive layers during training




input_1: InputLayer
conv2d_1: Conv2D
conv2d_2: Conv2D
max_pooling2d_1: MaxPooling2D
max_pooling2d_2: MaxPooling2D
flatten_1: Flatten
flatten_2: Flatten
concatenate_1: Concatenate
dense_1: Dense
dense_2: Dense

Compiling model

Compile defines the loss function, the optimizer and the metrics. It has nothing to **do** with the weights and you can **compile** a **model** as many times as you want without causing any problem to pretrained weights. You need a **compiled model** to train (because training uses the loss function and the optimizer

```
compile(
   optimizer,
   loss = None,
   metrics = None,
   loss_weights = None,
   sample_weight_mode = None,
   weighted_metrics = None,
   target_tensors = None
)
```

The important arguments are as follows −loss function,Optimizer,metrics

LOSS :

The error for the current state of the model must be estimated repeatedly. This requires the choice of an error function, conventionally called a **loss function**, that can be used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation.

METRICS :

Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process. Keras provides quite a few metrics as a module, **metrics** and they are as follows

- accuracy
- binary_accuracy
- categorical_accuracy
- sparse_categorical_accuracy

OPTIMIZER :

Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Keras provides quite a few optimizer as a module, *optimizers* **.**

Adam is the **best optimizers**. If one wants to train the **neural network** in less time and more efficiently than Adam is the **optimizer**.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout,
Flatten, MaxPooling2D

model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_sh
ape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

TRAINING MODEL

-
  ```
  model.fit(X_train, Y_train, batch_size=128, epochs=20, validation_data=(X_test, Y_test))
  ```

- the **batch size** is a number of samples processed before the **model** is updated.
- The number of epochs is the number of complete passes through the training dataset.
- The **size** of a **batch** must be more than or equal to one and less than or equal to the number of samples in the training dataset.

EVALUATE , PREDICT

EVALUATE:

Evaluating the model requires that you first choose a holdout dataset used to evaluate the model. This should be data not used in the training process so that we can get an unbiased estimate of the performance of the model when making predictions on new data.

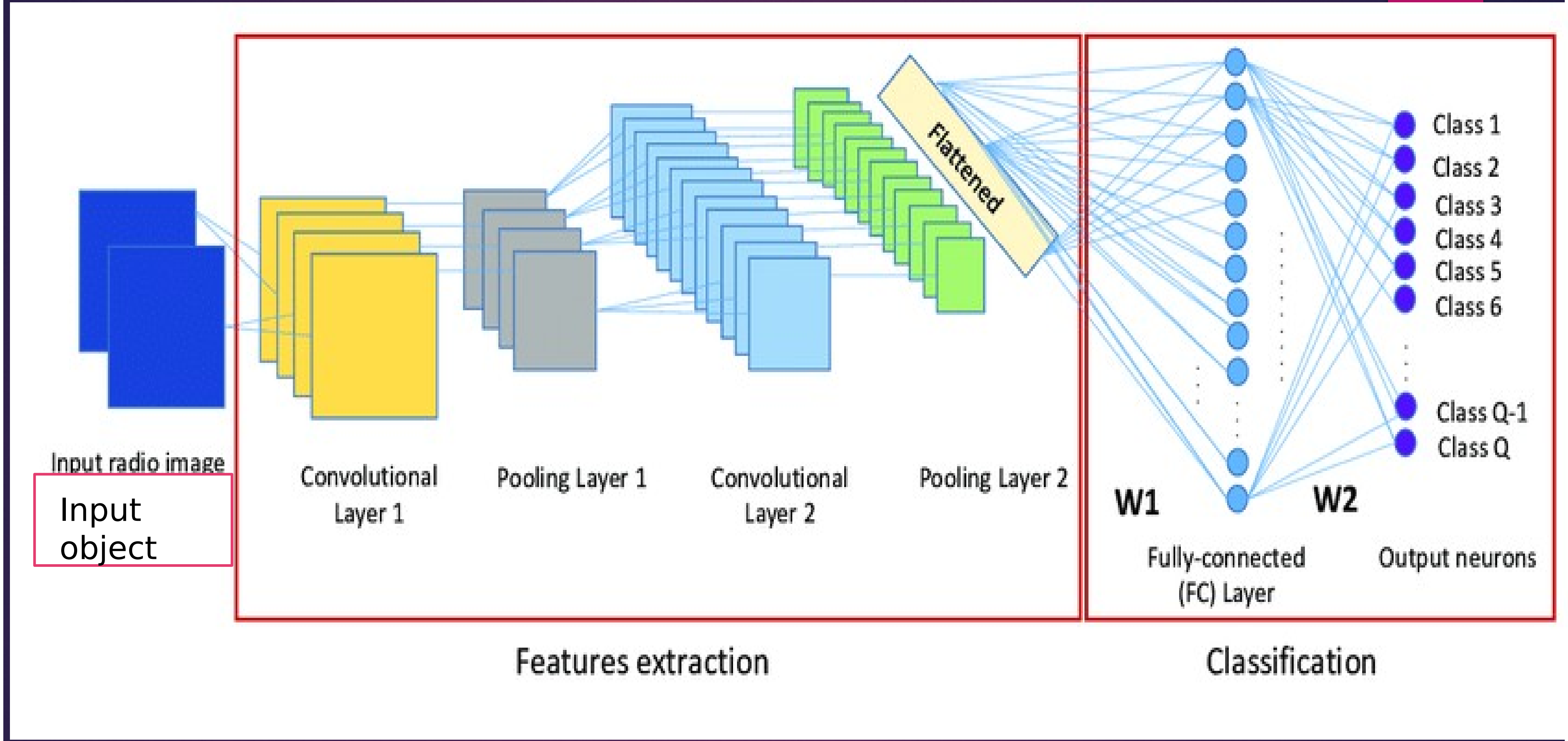
```
model.evaluate(X_test, Y_test)
```

PREDICT

Making a prediction is the final step in the life-cycle. It is why we wanted the model in the first place.

It requires you have new data for which a prediction is required, e.g. where you do not have the target values.

```
model.predict(X_test[100:105])
```



Input object