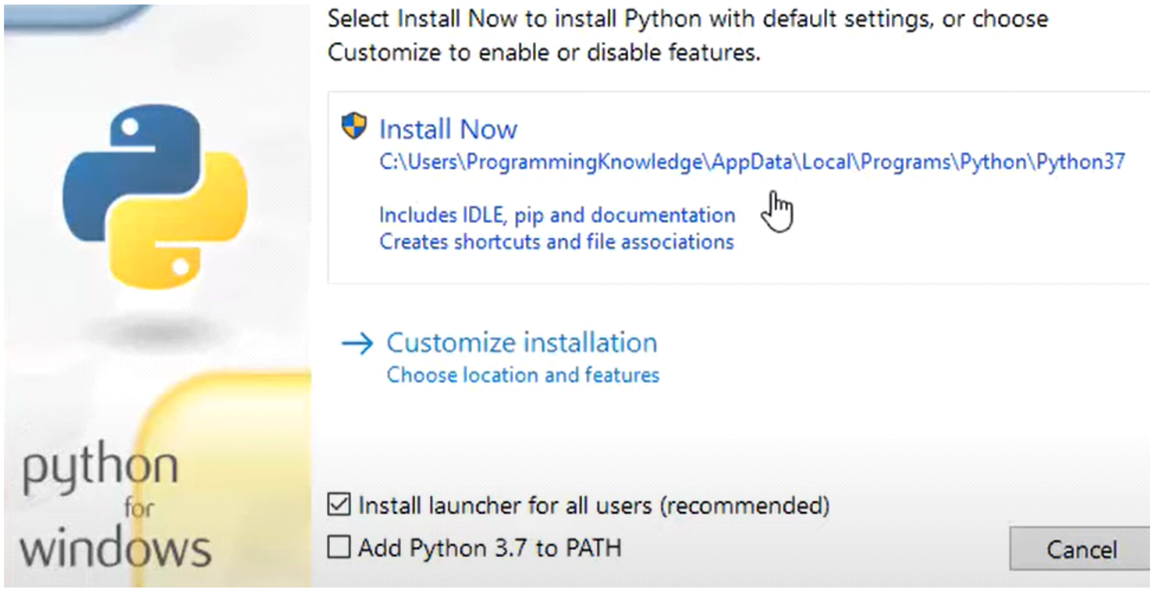
REQUIREMENT:

-PYTHON 3.7.*,

MUST: Checked -- Add Python 3.7 to PATH



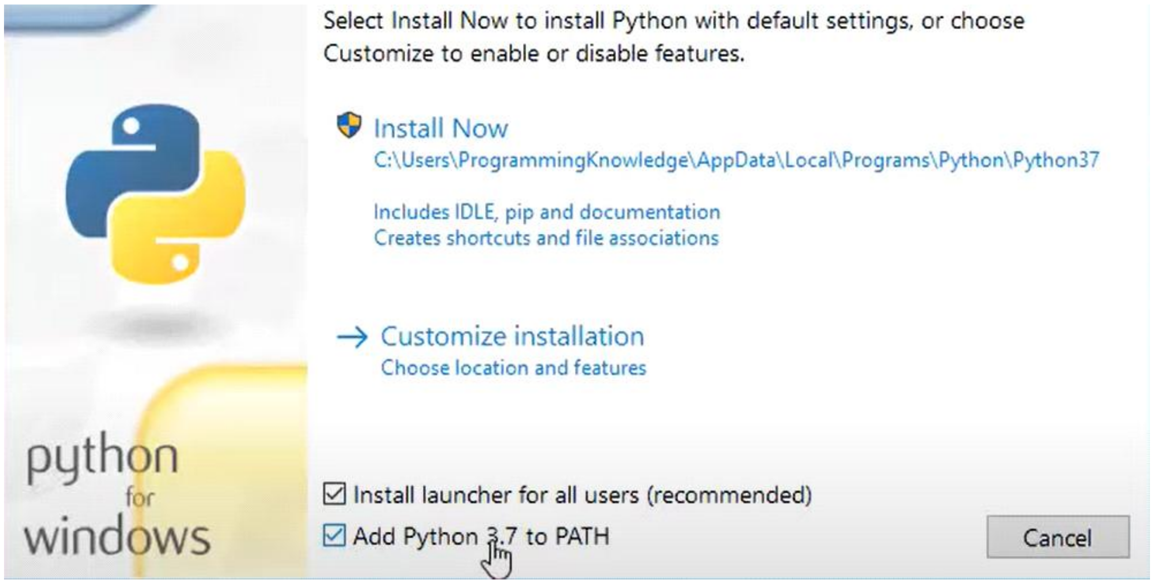
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\ProgrammingKnowledge\AppData\Local\Programs\Python\Python37

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.7 to PATH



Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\ProgrammingKnowledge\AppData\Local\Programs\Python\Python37

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

-Microsoft Visual C++ 2015-2019 Redistributable

https://github.com/IamRupamGanguly/Keras-Automatic-Number-Plate-Recognition/tree/master/ANPR/Softwares

CREATING ENVIRONMENT:

```
python -m venv EnvName
```

ACTIVATING the ENVNAME:

```
cd EnvName\Scripts
activate
```

INSTALLATION:

MATPLOTLIB: pip install matplotlib

PANDAS: pip install pandas

OPENCV: pip install opencv-python

TENSOEFLOW: pip install tensorflow

KERAS: pip install keras

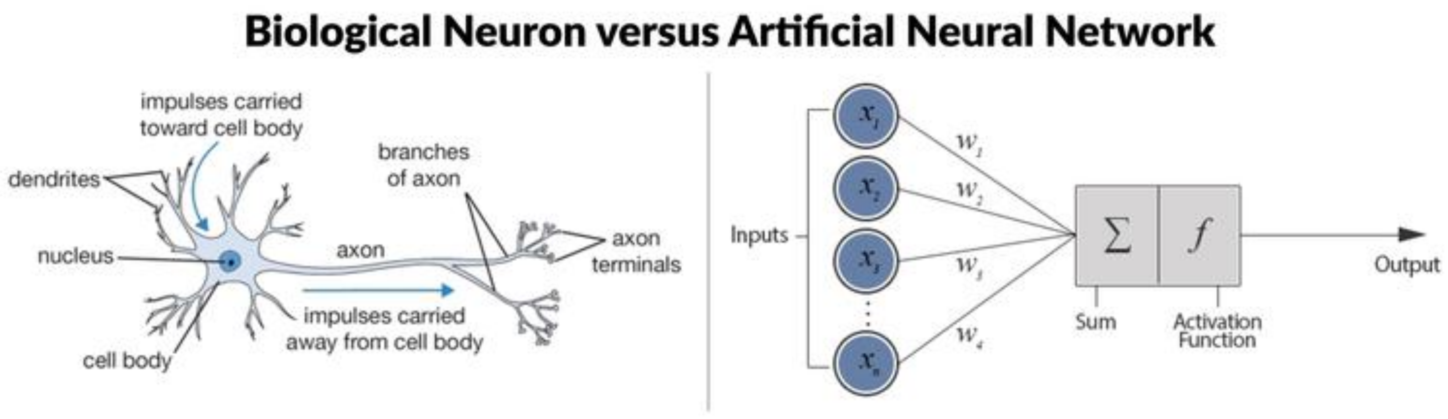
PILLOW: pip install pillow

JUPYTER NOTEBOOK: pip install jupyter

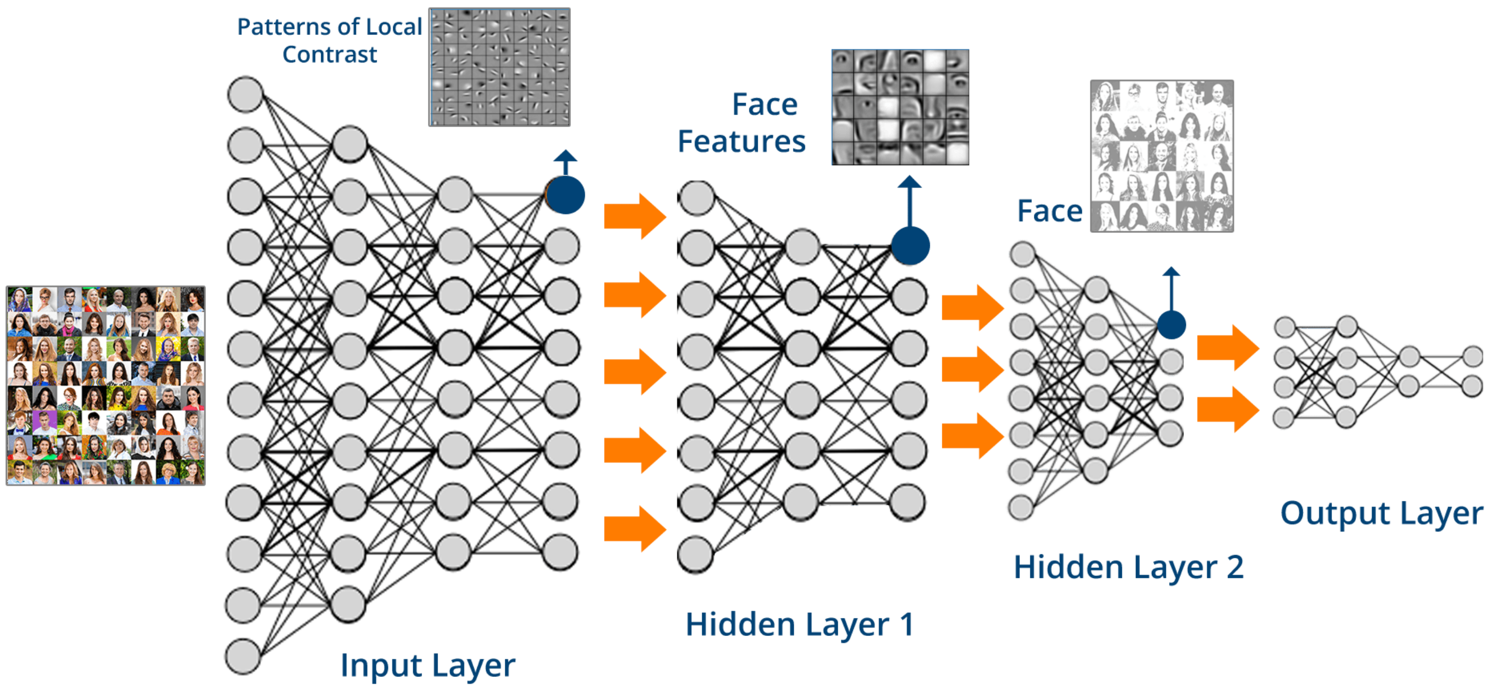
WHAT IS DEEP LEARNING?

DEEP LEARNING is a subfield of machine learning that is a set of algorithms that is inspired by the structure and function of the brain.

These algorithms are usually called Artificial Neural Networks, The human brain is then an example of such a neural network, which is composed of a number of neurons.

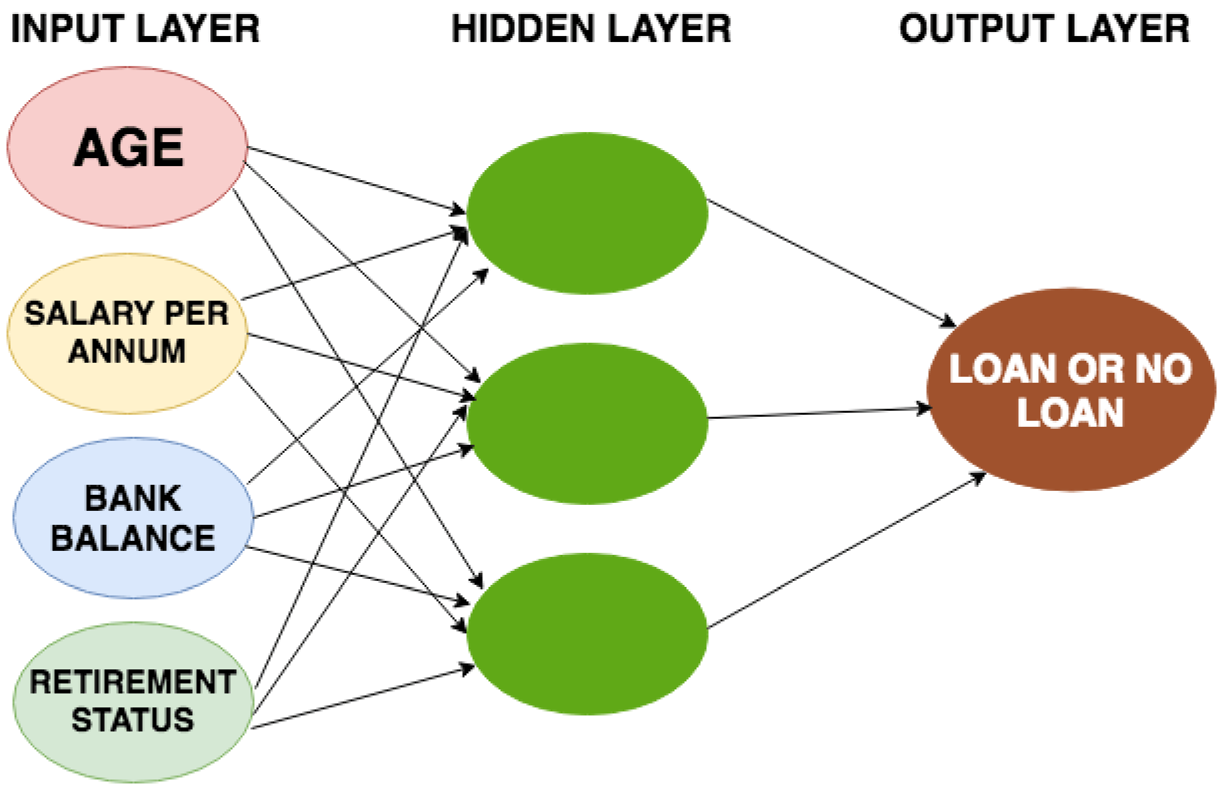


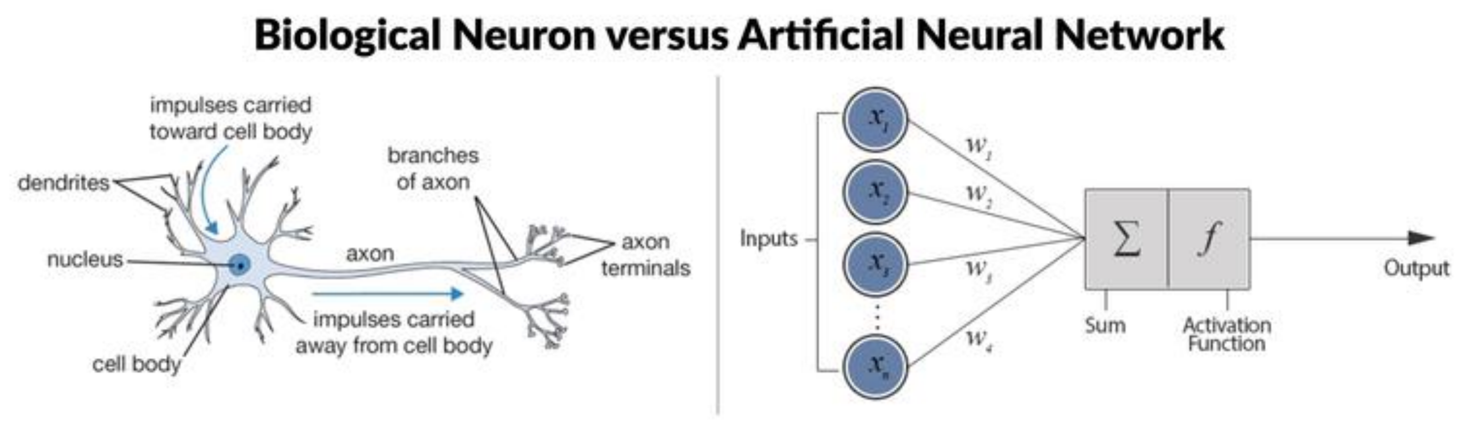
However, just like a biological neuron only fires when a certain threshold is exceeded, the artificial neuron will also only fire when the sum of the inputs exceeds a threshold.



Multi-layer are also known as “feed-forward neural networks”, they consist of multiple neurons that are organized in layers. The number of layers is usually limited to two or three, but theoretically, there is no limit!

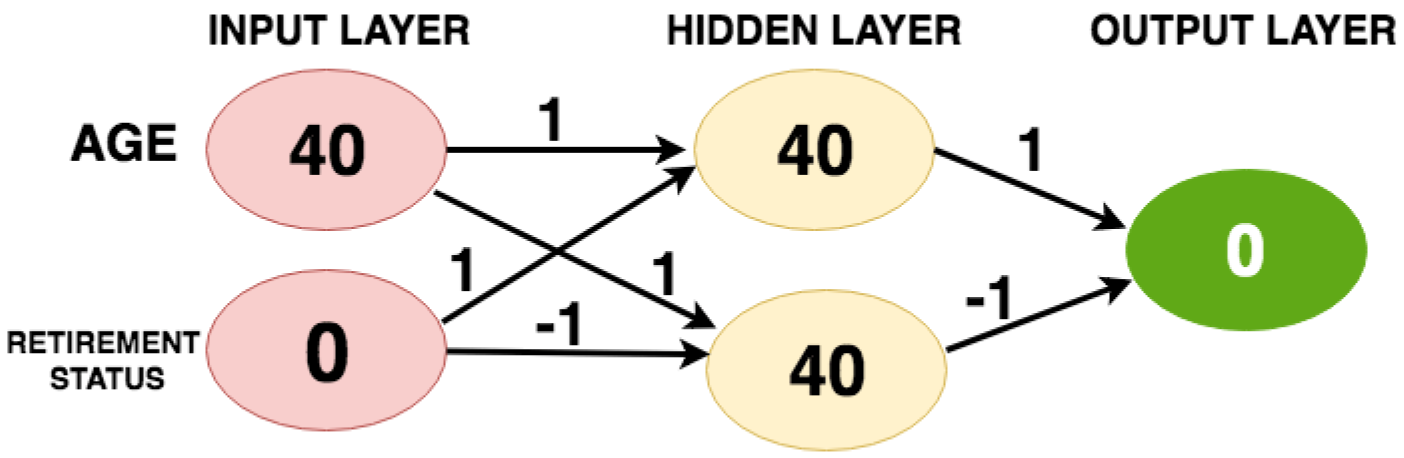
The layers act very much like the biological neurons that you have read about above: the outputs of one layer serve as the inputs for the next layer.





Let's start by seeing how neural networks use data to make predictions which is taken care by the forward propagation algorithm.

Let's consider only two features as an input namely age and retirement status, the retirement status being a binary (0 - not retired and 1 - retired) number based on which you will make predictions.



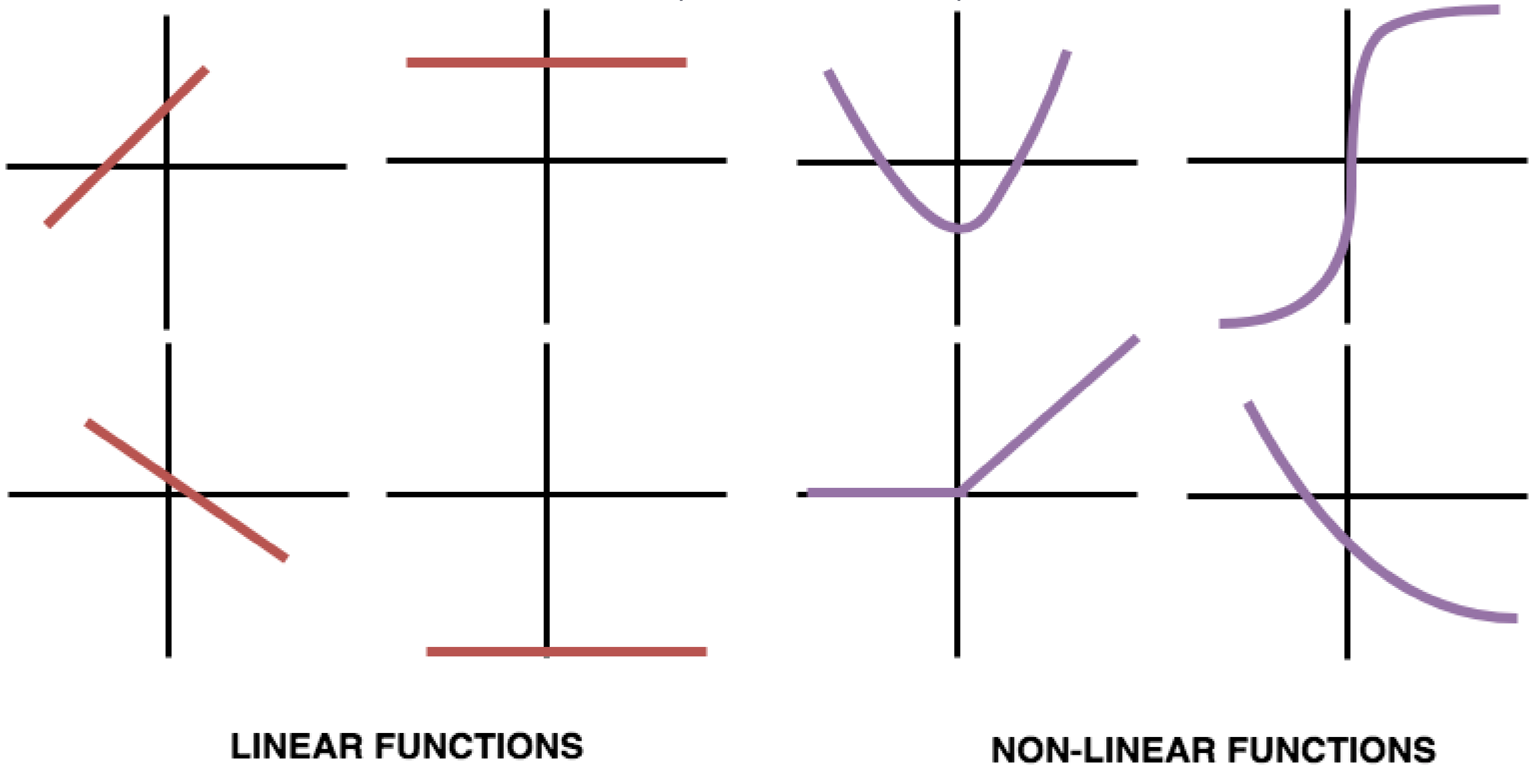
The above figure shows a customer with age 40 and is not retired. The forward propagation algorithm will pass this information through the network/model to predict the output layer. The lines connect each node of the input to every other node of the hidden layer. Each line has a weight associated with it which indicates how strongly that feature affects the hidden node connected to that specific line.

Remember these weights are the key in deep learning which you train or update when you fit a neural network to the data. These weights are commonly known as **parameters**.

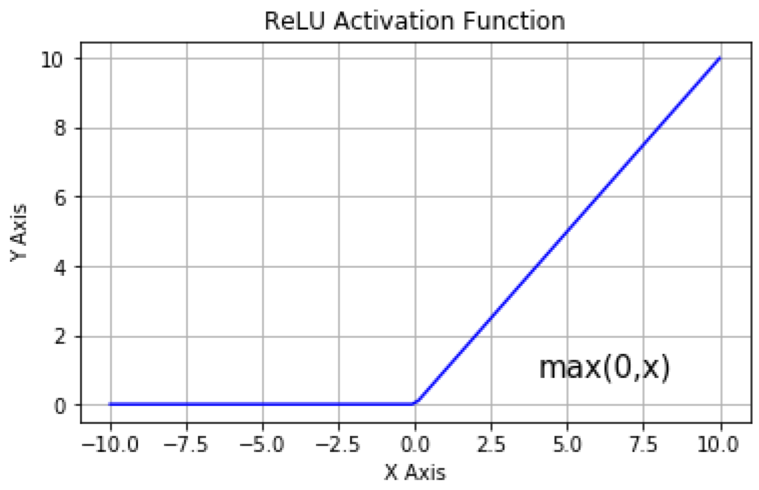
To make a prediction for the top node of the hidden layer, you consider each node in the input layer multiply it by the weights connected to that top node and finally sum up all the values resulting in a value 40 (40 * 1 + 0 * 1 = 40) as shown in above figure. You repeat the same process for the bottom node of the hidden layer resulting in a value 40. Finally, for the output layer you follow the same process and obtain a value 0 (40 * 1 + 40 * (-1) = 0). This output layer predicts a value zero.

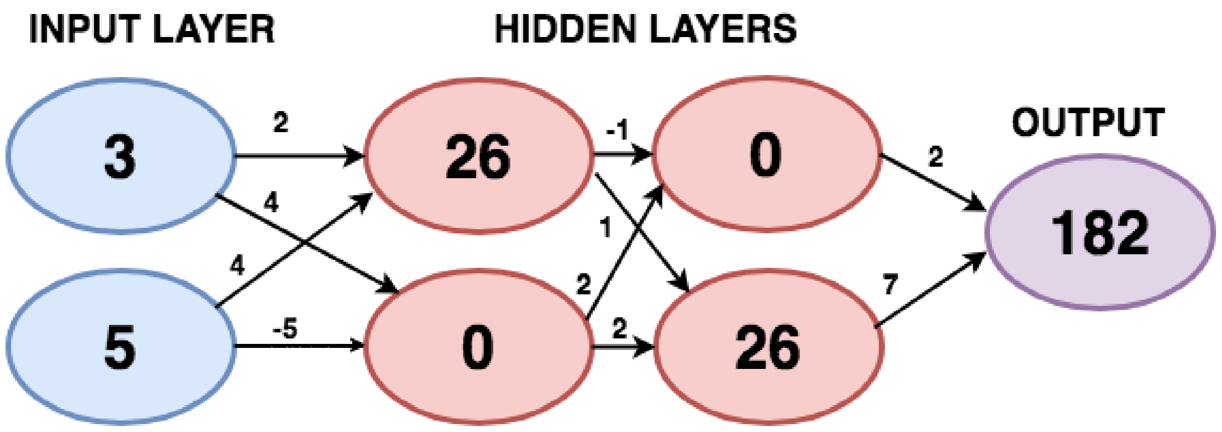
Output of zero indicates a loan sanction and an output of one indicates a loan prohibition.

To utilize the maximum predictive power, a neural network uses an activation function in the hidden layers. An activation function allows the neural network to capture non-linearities present in the data.



In today's time, an activation function called Rectifier Linear Unit (ReLU) is widely used in both industry and research. Even though it has two linear pieces, it's very powerful when combined through multiple hidden layers. ReLU is half rectified from the bottom as shown in the figure below.



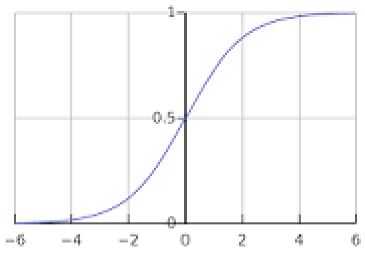


Research shows that increasing the number of hidden layers massively improves the performance making the network capable of more and more interactions.

The working in a network with just a single hidden layer and with multiple hidden layers remain the same. You forward propagate through these successive hidden layers as you did in the previous example with one hidden layer.

Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape.



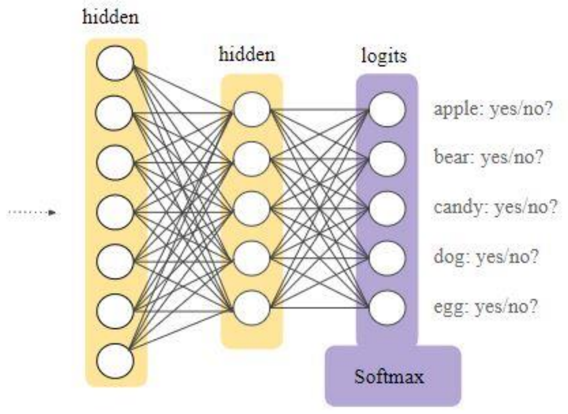
The Sigmoid function takes a value as input and outputs another value between 0 and 1

The main reason why we use sigmoid function is because it exists between **(0 to 1).** Therefore, it is especially used for models where we have to **predict the probability** as an output. Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.

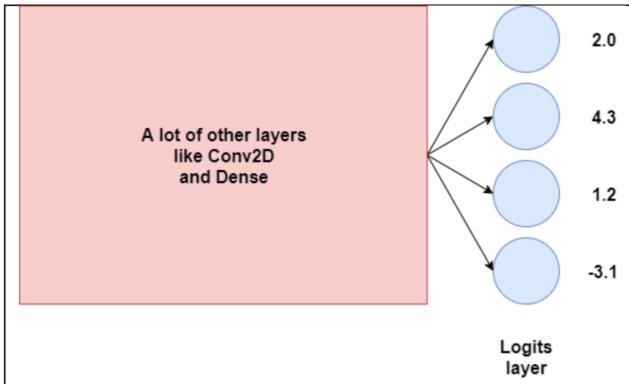
A problem where you classify an example as belonging to one of two classes. The problem is framed as predicting the likelihood of an example belonging to class one, e.g. the class that you assign the integer value 1, whereas the other class is assigned the value 0.

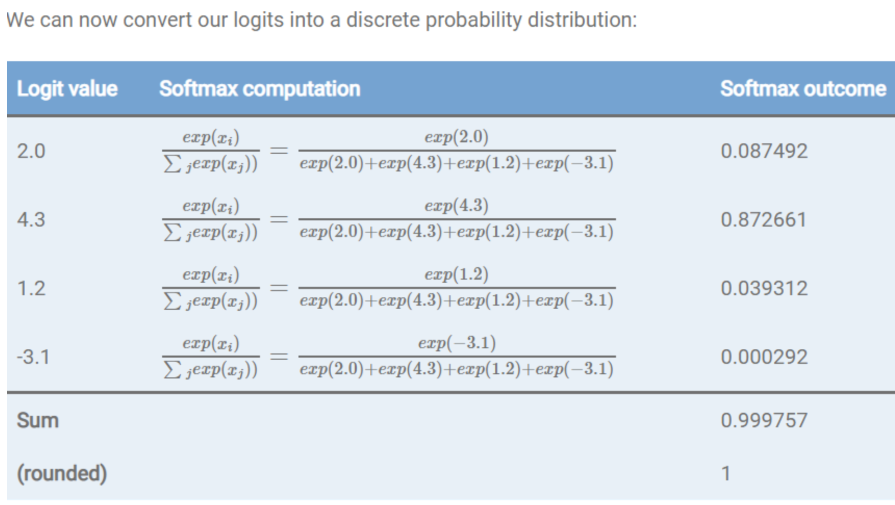
SoftMax function: When you're creating a neural network for classification, you're likely trying to solve either a binary or a multiclass classification problem. it's very likely that the activation function for your final layer is the so-called **SoftMax activation function**, which results in a multiclass probability distribution over your target classes.

It always "returns a probability distribution over the target classes in a multiclass classification problem". The Softmax layer must have the same number of nodes as the output layer."



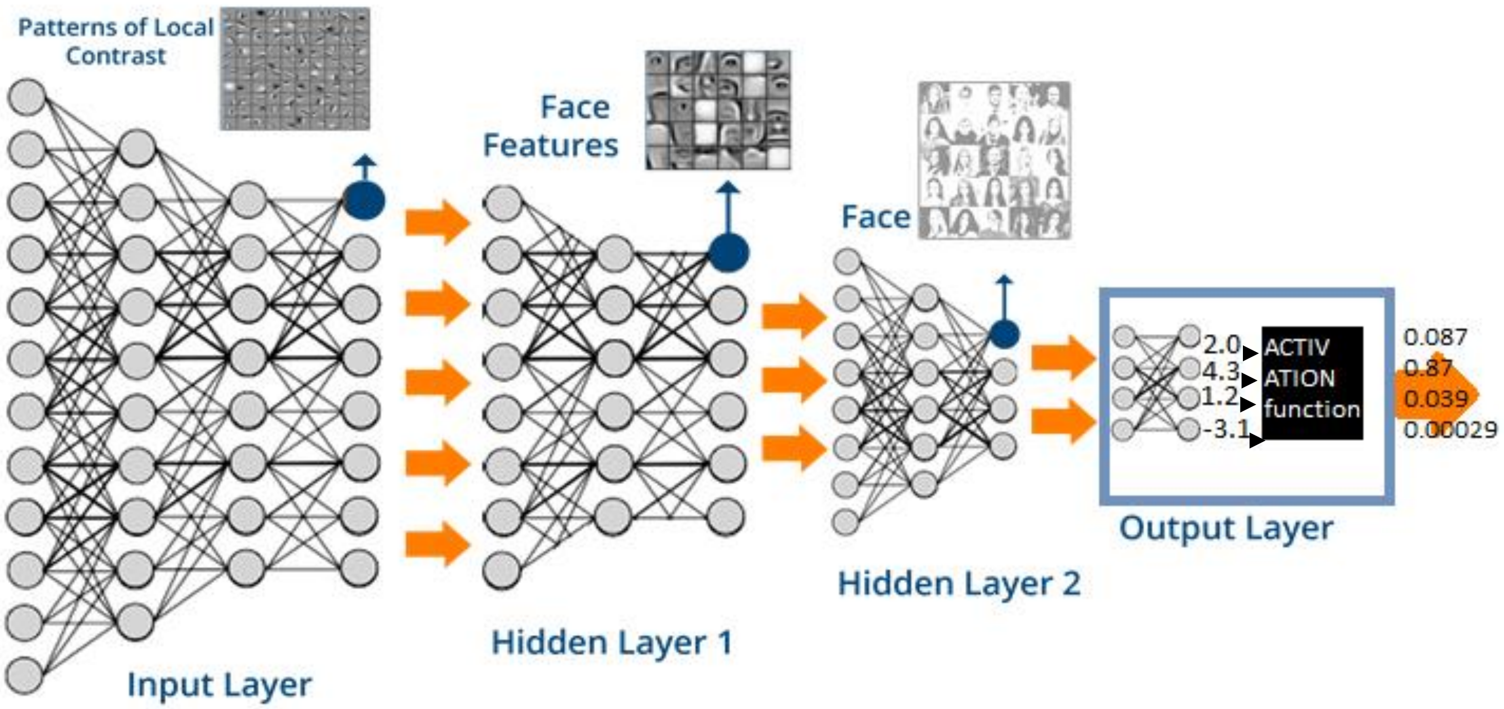
The final layer of the neural network, *without the activation function*, is what we call the **"logits layer"**





We can now convert our logits into a discrete probability distribution:

Logit value	Softmax computation	Softmax outcome
2.0	$\frac{exp(x_i)}{\sum_j exp(x_j))} = \frac{exp(2.0)}{exp(2.0)+exp(4.3)+exp(1.2)+exp(-3.1)}$	0.087492
4.3	$\frac{exp(x_i)}{\sum_j exp(x_j))} = \frac{exp(4.3)}{exp(2.0)+exp(4.3)+exp(1.2)+exp(-3.1)}$	0.872661
1.2	$\frac{exp(x_i)}{\sum_j exp(x_j))} = \frac{exp(1.2)}{exp(2.0)+exp(4.3)+exp(1.2)+exp(-3.1)}$	0.039312
-3.1	$\frac{exp(x_i)}{\sum_j exp(x_j))} = \frac{exp(-3.1)}{exp(2.0)+exp(4.3)+exp(1.2)+exp(-3.1)}$	0.000292
Sum		0.999757
(rounded)		1

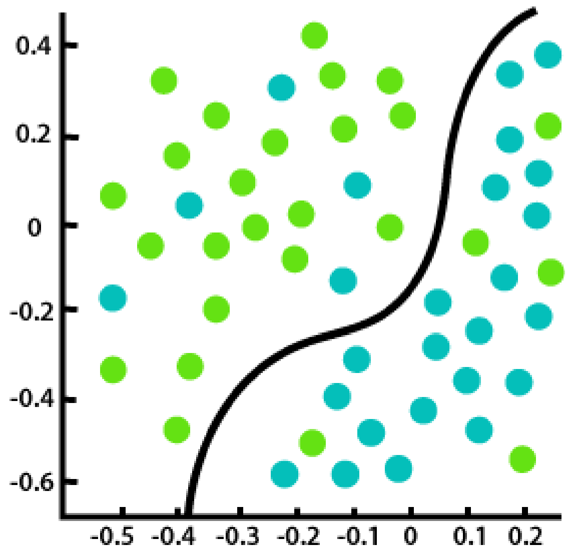


Error and Loss Function: At its core, a loss function is a measure of how good your prediction model does in terms of being able to predict the expected outcome (or value).
In most learning networks, error is calculated as the difference between the actual output and the predicted output.

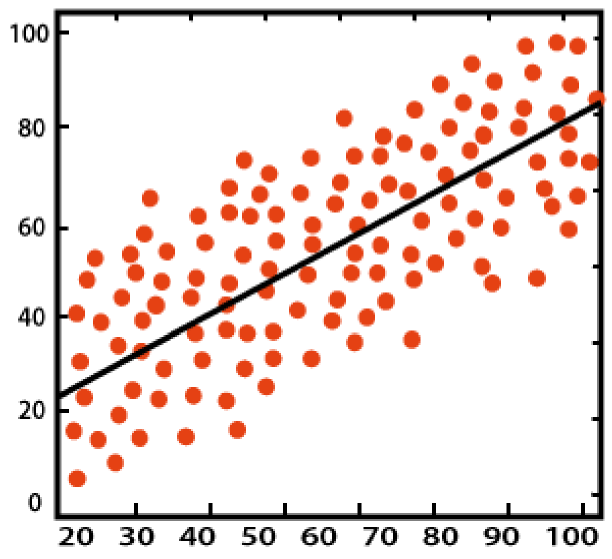
The function that is used to compute this error is known as Loss Function. Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model. One of the most widely used loss function is mean square error, which calculates the square of difference between actual value and predicted value. Different loss functions are used to deal with different type of tasks, i.e. regression and classification.

A loss function is for a single training example. It is also sometimes called an error function. A cost function, on the other hand, is the average loss over the entire training dataset. The optimization strategies aim at minimizing the cost function.

ML problems and corresponding Loss functions



Classification



Regression

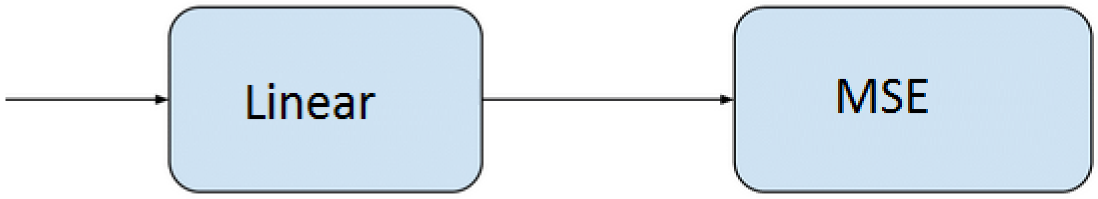
Classification is a process of finding a function which helps in dividing the dataset into classes based on different parameters. In Classification, a computer program is trained on the training dataset and based on that training, it categorizes the data into different classes. **Example:** The best example to understand the Classification problem is Email Spam Detection. The model is trained on the basis of millions of emails on different parameters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.	Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of **Market Trends**, prediction of House prices, etc. **Example:** Suppose we want to do weather forecasting, so for this, we will use the Regression algorithm. In weather prediction, the model is trained on the past data, and once the training is completed, it can easily predict the weather for future days.

Let us see what are commonly used output layers and loss function in machine learning models:

Regression Problem

A problem where you predict a real-value quantity.

Output Layer Configuration: One node with a linear activation unit. We start off with the simplest function; the linear function. The value of *f(z)* increases proportionally with the value of *z*.



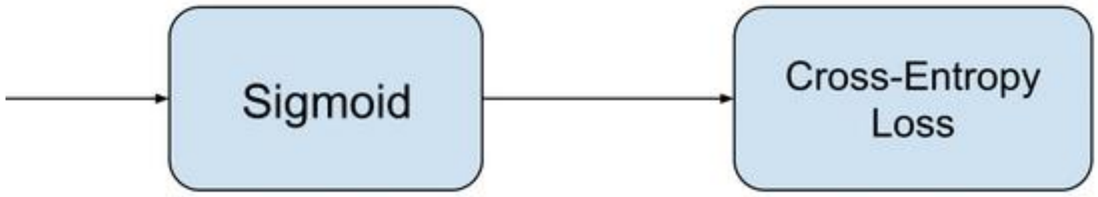
Mean Square Error (MSE) is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicted values.

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

You want to predict future house prices, and your dataset includes homes that are orders of magnitude different in price. The price is a continuous value, and therefore, we want to do regression. MSLE can here be used as the loss function.

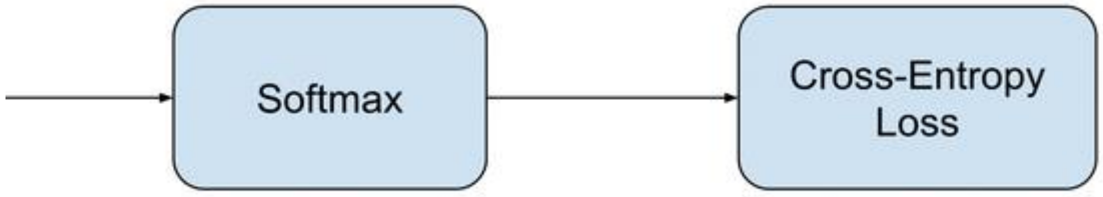
Binary Classification Problem

For example, classifying an email as *spam* or *not spam* based on, say its subject line, is binary classification.



Multi-Class Classification Problem

Emails are not just classified as spam or not spam (this isn't the 90s anymore!). They are classified into various other categories – Work, Home, Social, Promotions, etc.

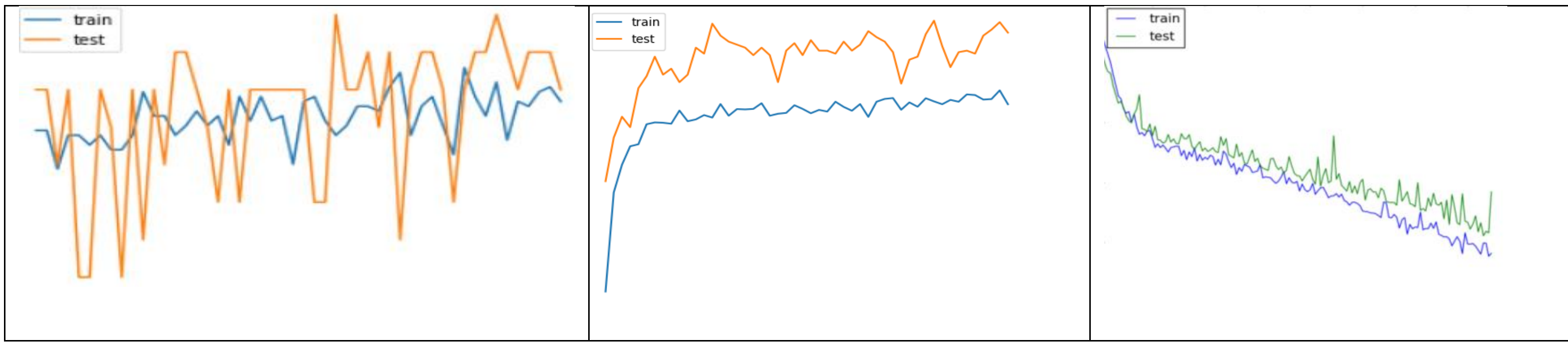


The loss function binary cross entropy is used on yes/no decisions, e.g., multi-label classification. The loss tells you how wrong your model's predictions are. For instance, in multi-label problems, where an example can belong to multiple classes at the same time, the model tries to decide for each class whether the example belongs to that class or not.

Optimizers and Learning rate:

Optimizers are algorithms or methods used to change the attributes of your **neural network** such as weights and **learning** rate in order to reduce the losses. The amount that the weights are updated during training is referred to as the step size or the "*learning rate*."

Momentum can smooth the progression of the learning algorithm that, in turn, can accelerate the training process.



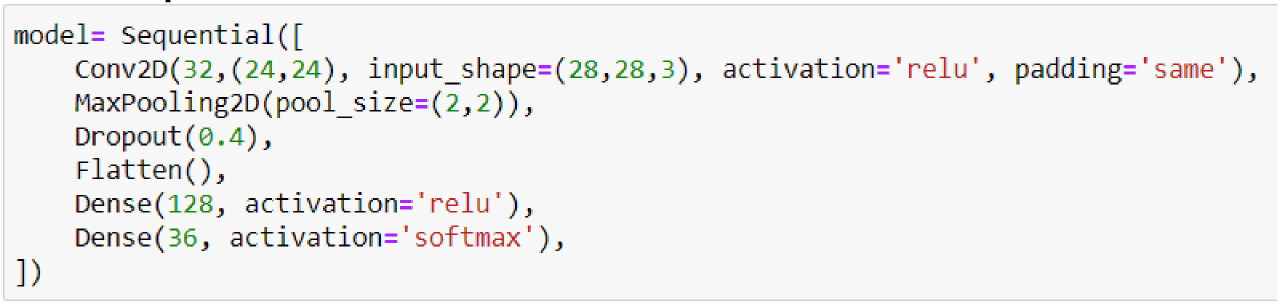
This Graph shows very Up-Down learning, but with momentum the model can learn in Smother manner.

Some of the most popular optimization algorithms used are the Stochastic Gradient Descent (SGD), ADAM and RMSprop.
Depending on whichever algorithm you choose, you'll need to tune certain parameters, such as learning rate or momentum.

Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum.
Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters

There are two ways to build Keras models: *sequential* and *functional*.
The **sequential** API allows you to create models layer-by-layer for most problems.It just tell the machine that, execute first layer specified then Second layer then third layer and so on.. Means it execute sequentially

This is **sequential** API:



```
model= Sequential([
    Conv2D(32,(24,24), input_shape=(28,28,3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.4),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(36, activation='softmax'),
])
```

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

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.

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see

h x w x d (h = Height, w = Width, d = Dimension). E.g., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

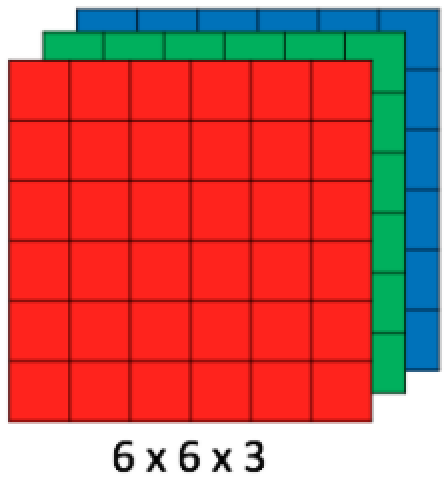
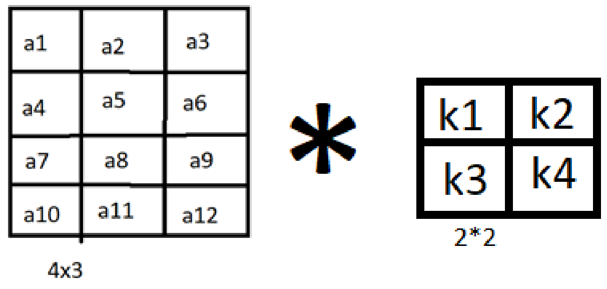
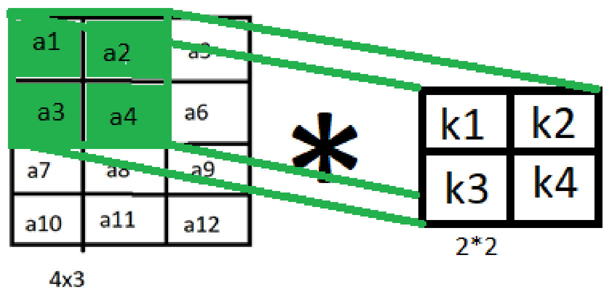
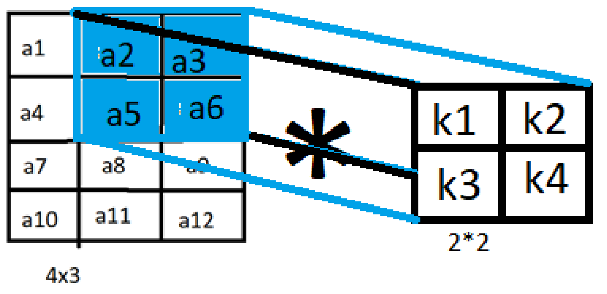


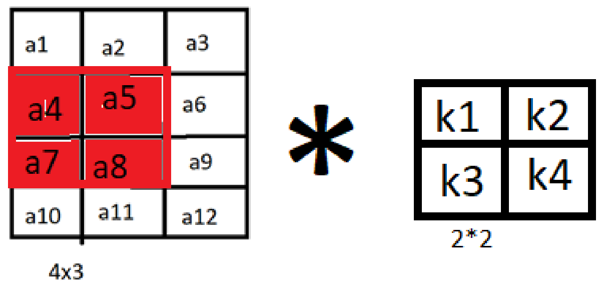
Fig 2.4.2

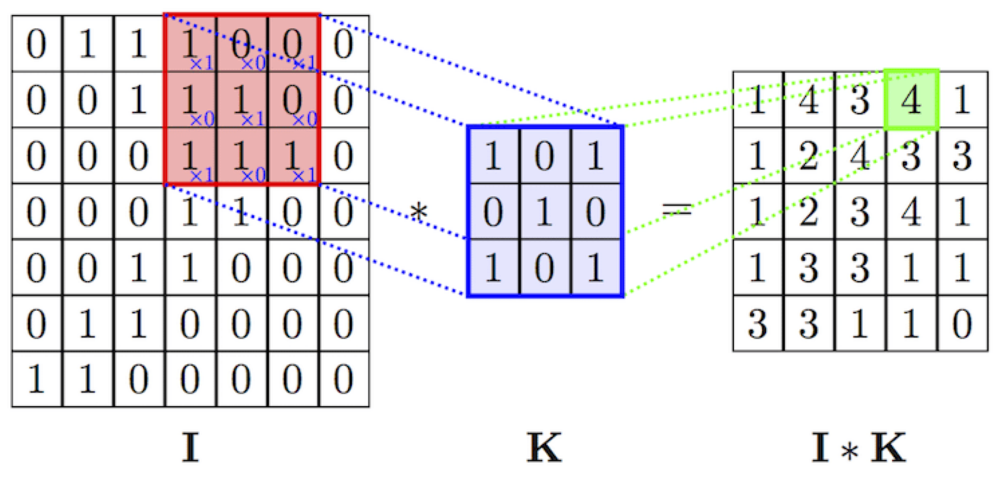




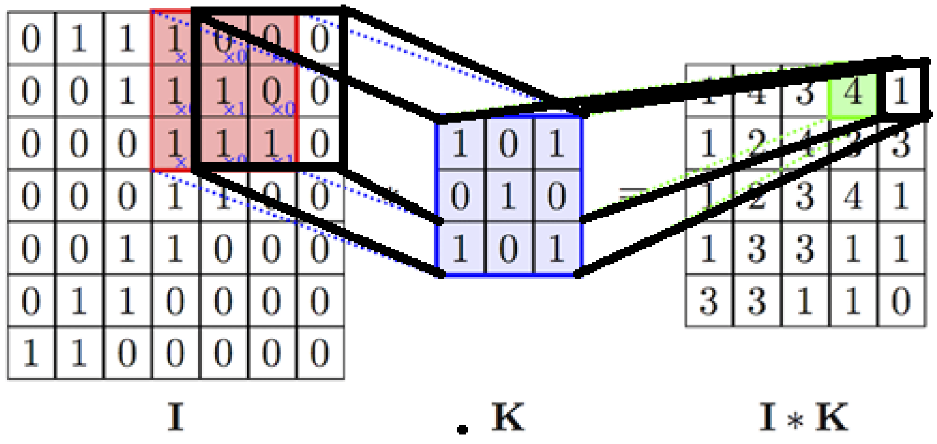
a1*k1 + a3*k3 + a2*k2 + k4







(1*1 + 1*0 +1*1) + (0*0 + 1*1 + 1*0) + (0*1 + 0*0 +1*1) = 4



(0*1 + 1*0 + 1*1) + (0*0 + 0*1 + 1*0) + (0*1 + 0*0 +0*1) = 1

Padding: can take two values 'valid' or 'same'.

Setting the value to 'valid' parameter means that the input volume is not zero-padded and the spatial dimensions are allowed to reduce via the natural application of convolution. When the value is 'same' output volume size matches the input volume size.

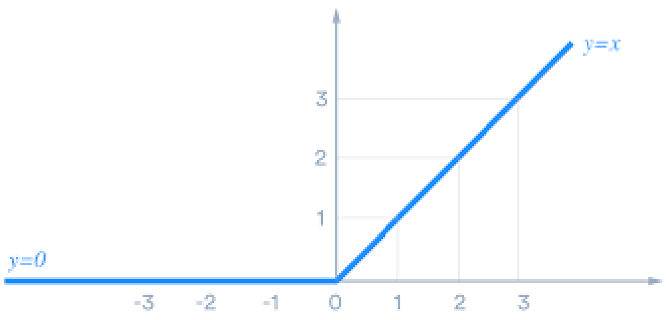
deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1.

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

In our ANPR, the first layer will be a convolutional layer with 32 output filters, a convolution window of size (24,24), and '**Relu' as activation function.**

ReLU stands for Rectified Linear Unit for a non-linear operation.

ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real-world data would want our ConvNet to learn would be non-negative linear values.



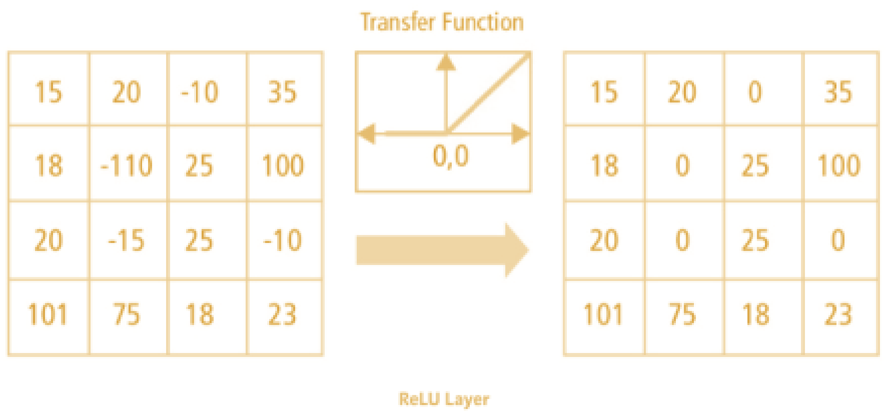


Fig 2.4.4

Next, we'll be adding a max-pooling layer with a window size of (2,2).

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

Max Pooling

Average Pooling

Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

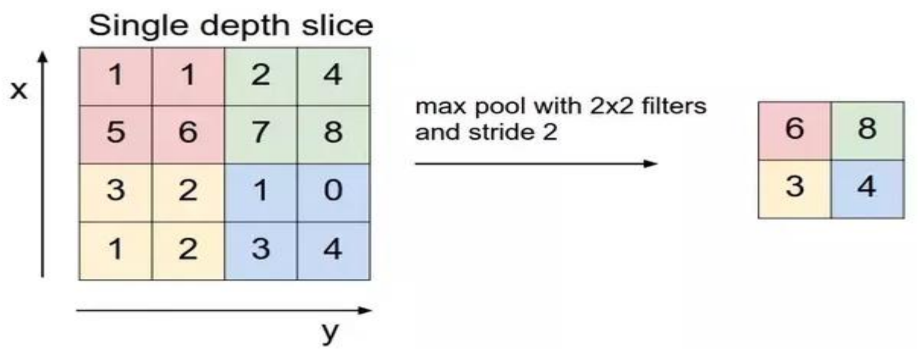


Fig 2.4.3

Now, we will be adding some dropout rate to take care of overfitting. Large neural nets trained on relatively small datasets can overfit the training data.

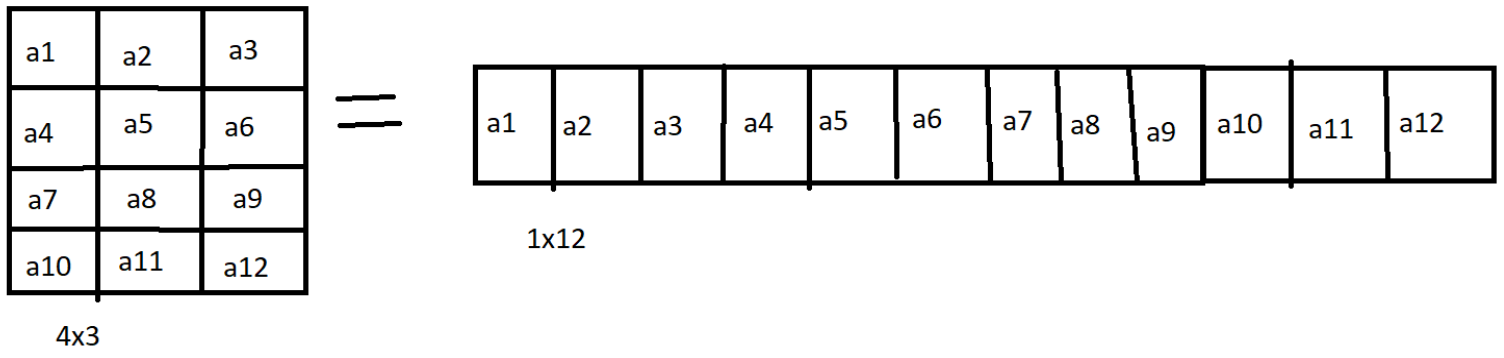
This has the effect of the model learning the statistical noise in the training data, which results in poor performance when the model is evaluated on new data, e.g. a test dataset.

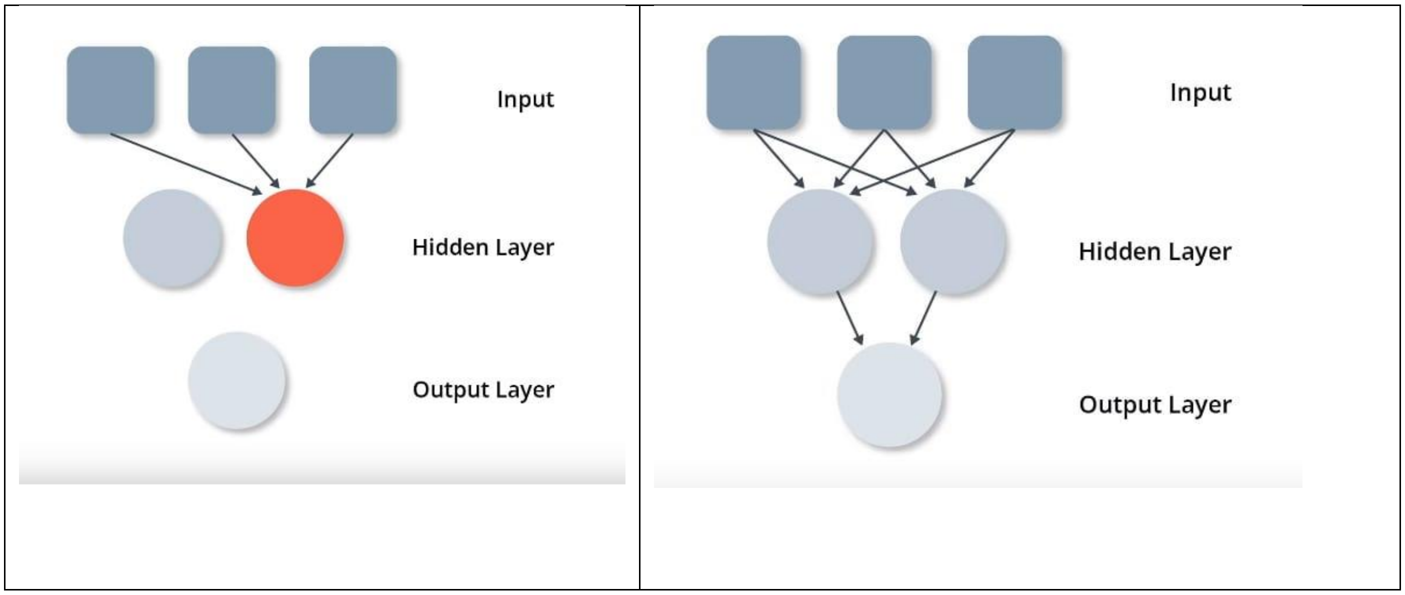
Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.

During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different “view” of the configured layer.

Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.

Now it’s time to flatten the node data so we add a flatten layer for that**. Flatten** layer takes data from the previous layer and represents it in a single dimension. 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. In other words, we put all the pixel data in one line and make connections with the final layer.





Not Dense Layer   Dense Layer

Finally, we will be adding 2 dense layers, one with the dimensionality of the output space as 128, activation function=’relu’ and other, our final layer with 36 outputs for categorizing the 26 alphabets (A-Z) + 10 digits (0–9) and activation function=’ softmax’.

We use softmax as the output function of the last layer in neural networks (if the network has n layers, the n-th layer is the softmax function). This fact is important because the purpose of the last layer is to turn the score produced by the neural network into values that can be interpreted by humans.

Alternatively, the **functional** API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers. In fact, you can connect layers to (literally) any other layer.

This is **functional** API:
Ex1:

```
# Multilayer Perceptron
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense

visible = Input(shape=(10,))
hidden1 = Dense(10, activation='relu')(visible)
hidden2 = Dense(20, activation='relu')(hidden1)
hidden3 = Dense(10, activation='relu')(hidden2)
output = Dense(1, activation='sigmoid')(hidden3)
model = Model(inputs=visible, outputs=output)
```

The model has 10 **inputs**, 3 hidden layers with 10, 20, and 10 neurons, and an output layer with 1 output. Rectified linear activation functions are used in each hidden layer and a sigmoid activation function is used in the output layer, for binary classification.

Ex2:

```
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
visible = Input(shape=(64,64,1))
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(16, kernel_size=4, activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat = Flatten()(pool2)
hidden1 = Dense(10, activation='relu')(flat)
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```

Training and evaluating our convolutional neural network

```
model= Sequential([
    Conv2D(32,(24,24), input_shape=(28,28,3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.4)
    Flatte 128: int
    Dense(128, activation='relu'),
    Dense(36, activation='softmax'),
])
```

M↓

```
model.compile(Adam(lr=.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

M↓

```
history = model.fit_generator(train_batches, steps_per_epoch=1679, validation_data=valid_batches,
validation_steps=216, epochs=50)
```

```
Epoch 1/50
1679/1679 [==============================] - 51s 31ms/step - loss: 0.1610 - accuracy: 0.9857 - val_loss: 0.0
000e+00 - val_accuracy: 0.9907
Epoch 2/50
1679/1679 [==============================] - 54s 32ms/step - loss: 0.1993 - accuracy: 0.9857 - val_loss: 0.0
000e+00 - val_accuracy: 0.9907cy: 0.98
Epoch 3/50
1679/1679 [==============================] - 52s 31ms/step - loss: 0.3320 - accuracy: 0.9809 - val_loss: 0.0
000e+00 - val_accuracy: 0.9815
```