02206447

Data Science for Manufacturing and Logistics

CONVOLUTIONAL NEURAL NETWORK

DR. PRAPASSORN TANTIPHANWADI

INDUSTRIAL ENGINEERING, FACULTY OF ENGINEERING AT KHAMPAENGSAEN

KASETSART UNIVERSITY, NAKORN PATHOM

CONTENT

- What is Deep Learning?
- Convolution Neural Network
- VGGNet & ResNet
- Examples

SOME BASIC DEFINITION

Artificial Intelligence

The ability of a digital computer or robot to perform tasks commonly associated with intelligent beings.

- Vision
- Robotics

Machine Learning

The practice of learning a task from data, such as algorithms, models.

- Support vector machines
- KNN
- Bayesian learning

Deep Learning

A type of machine learning in which a model learns to perform tasks directly from images, text, or sound.

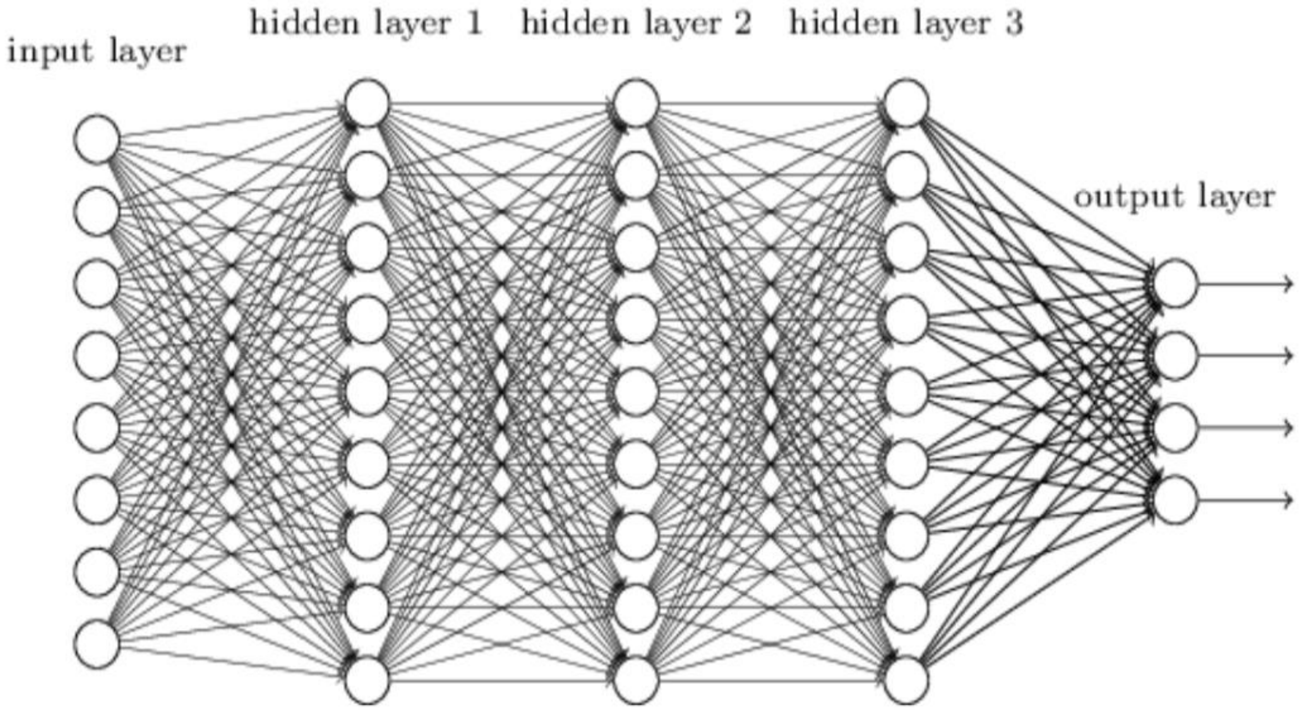
WHAT IS DEEP LEARNING?

- A TYPE OF MACHINE LEARNING IN WHICH A MODEL LEARNS TO PERFORM CLASSIFICATION TASKS DIRECTLY FROM IMAGES, TEXT, OR SOUND.
- IT IS USUALLY IMPLEMENTED USING A NEURAL NETWORK ARCHITECTURE.
- THE TERM "DEEP" REFERS TO THE NUMBER OF LAYERS IN THE NETWORK—THE MORE LAYERS, THE DEEPER THE NETWORK.
- TRADITIONAL NEURAL NETWORKS CONTAIN ONLY 2 OR 3 LAYERS, WHILE DEEP NETWORKS CAN HAVE HUNDREDS.

DEEP NEURAL NETWORK

- Combines multiple nonlinear processing layers, using simple elements operating in parallel and inspired by biological nervous systems.
- Consists of an input layer, several hidden layers, and an output layer.
- The layers are interconnected via nodes, or neurons, with each hidden layer using the output of the previous layer as its input.

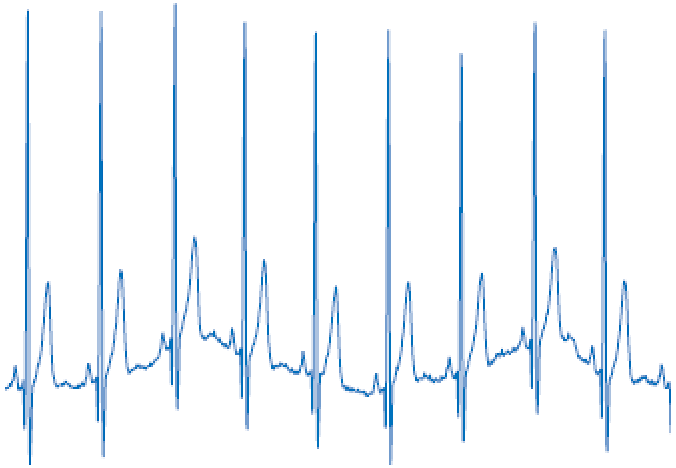
multilayered perceptron (MLP)
The neural units are arranged layer after layer, and adjacent network layers are fully connected to one another. (image source: American journal publishing, 2015)



DEEP LEARNING DATA TYPES

IMAGE

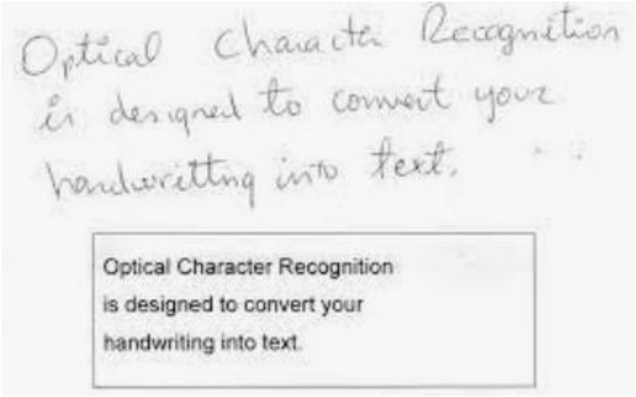
SIGNAL



NUMERIC

AgeCat	WeightQ	GroupCount	mean_BloodPressure	
Under 30	Q1	6	123.17	79.667
Under 30	Q2	3	120.33	79.667
Under 30	Q3	2	127.5	86.5
Under 30	Q4	4	122	78
30-39	Q1	12	121.75	81.75
30-39	Q2	9	119.56	82.556
30-39	Q3	9	121	83.222
30-39	Q4	11	125.55	87.273
Over 40	Q1	7	122.14	84.714
Over 40	Q2	13	123.38	79.385

TEXT



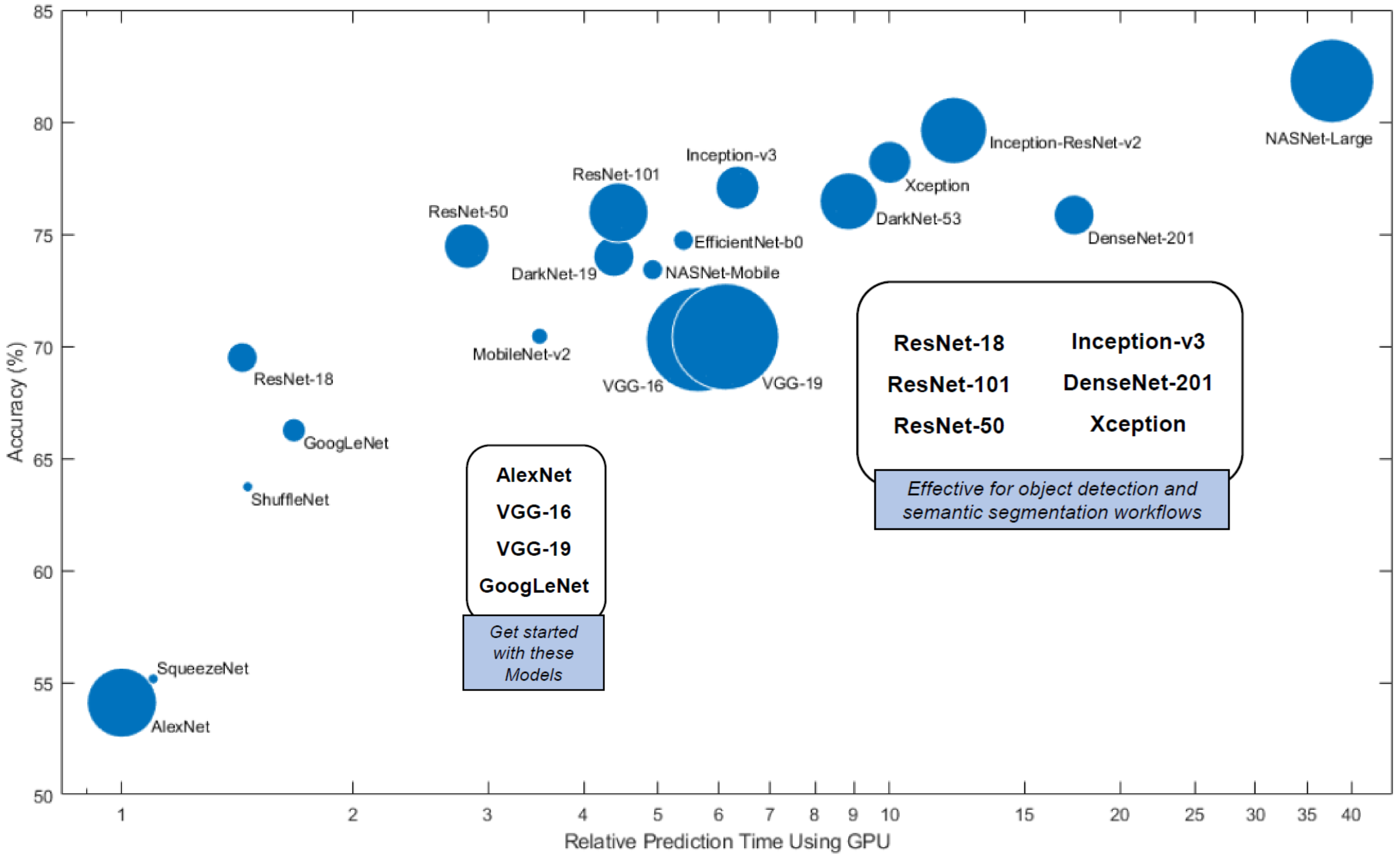
(Image sources: Internet search)

VARIOUS NEURAL NETWORK ARCHITECTURES

(https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html)

A NETWORK CAN BE BUILT

- FROM SCRATCH
- OR PRE-TRAIN MODEL



CONVOLUTION NEURAL NETWORK

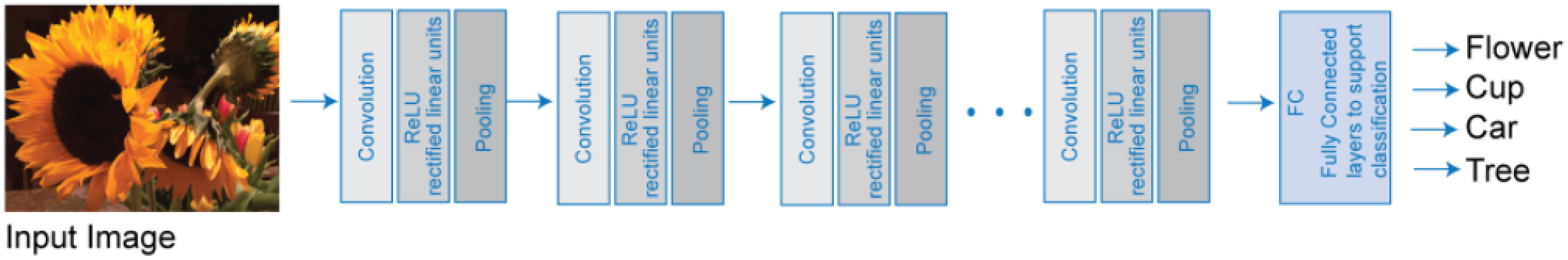
CNNs are multilayered neural networks that

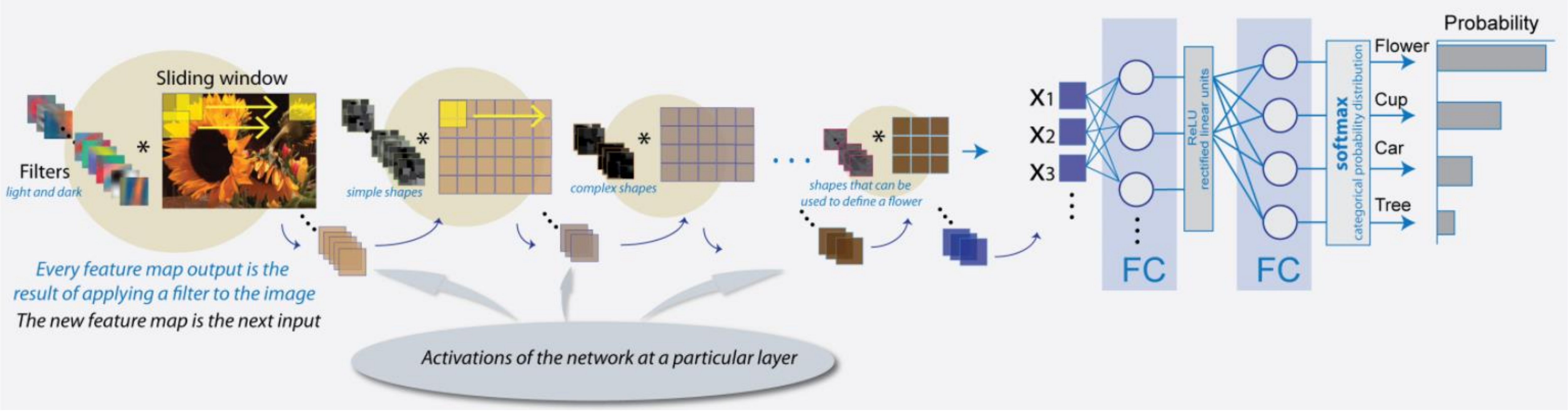
- ✓ are known as a high degree of invariance to translation, scaling, and rotation in two-dimensional image data neural network.
- ✓ based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps.
- ✓ Need to be trained in a supervised way. Typically, a labeled set of object classes, such as MNIST or else, is provided as a training set.

➢ The crux of any CNN model are

- ✓ **Convolution Layer**
- ✓ **Pooling Layer**
- ✓ **ReLU Layer**

CONVOLUTION NEURAL NETWORK





Ref source: Mathworks.com/help center

CONVOLUTION OPERATOR

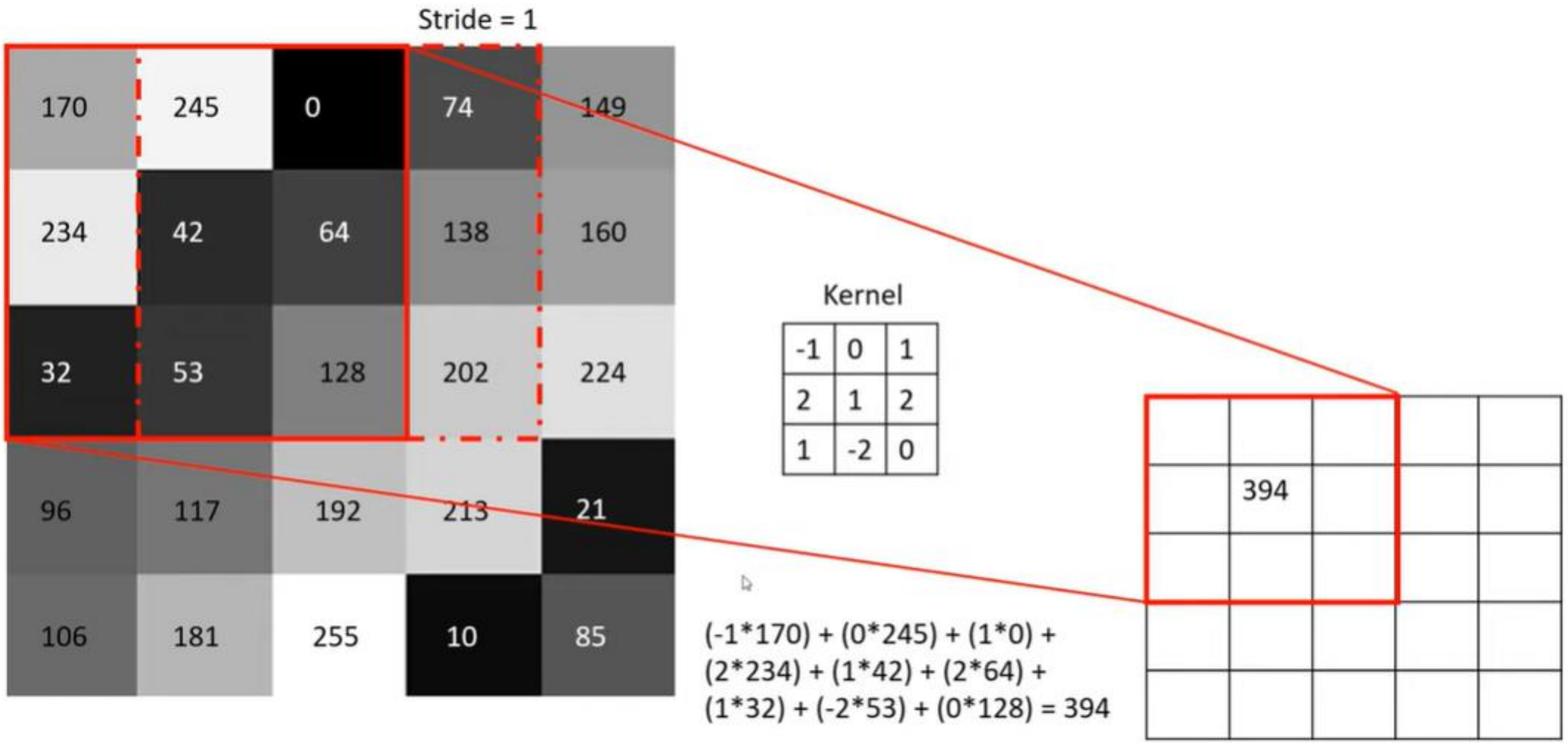
- mathematical operation of convolution, which is a specialized kind of linear operation.
- Example: A two-dimensional image $\boldsymbol{I}(\boldsymbol{m},\boldsymbol{n})$
- A two-dimensional kernel of the convolution, $\boldsymbol{K}$ (or weight function)
- The convoluted image, $\boldsymbol{S}(\boldsymbol{i},\boldsymbol{j})$, can be calculated as:

$$\boldsymbol{S}(\boldsymbol{i},\boldsymbol{j}) = \sum_{\boldsymbol{m}} \sum_{\boldsymbol{n}} \boldsymbol{K}(\boldsymbol{m},\boldsymbol{n}) \boldsymbol{I}(\boldsymbol{i}-\boldsymbol{m},\boldsymbol{j}-\boldsymbol{n})$$

Ref: Mathworks.com/help center

CONVOLUTION OPERATOR

The figure explains how with a kernel of size 3x3, with stride 1, the convolution layer output is calculated.

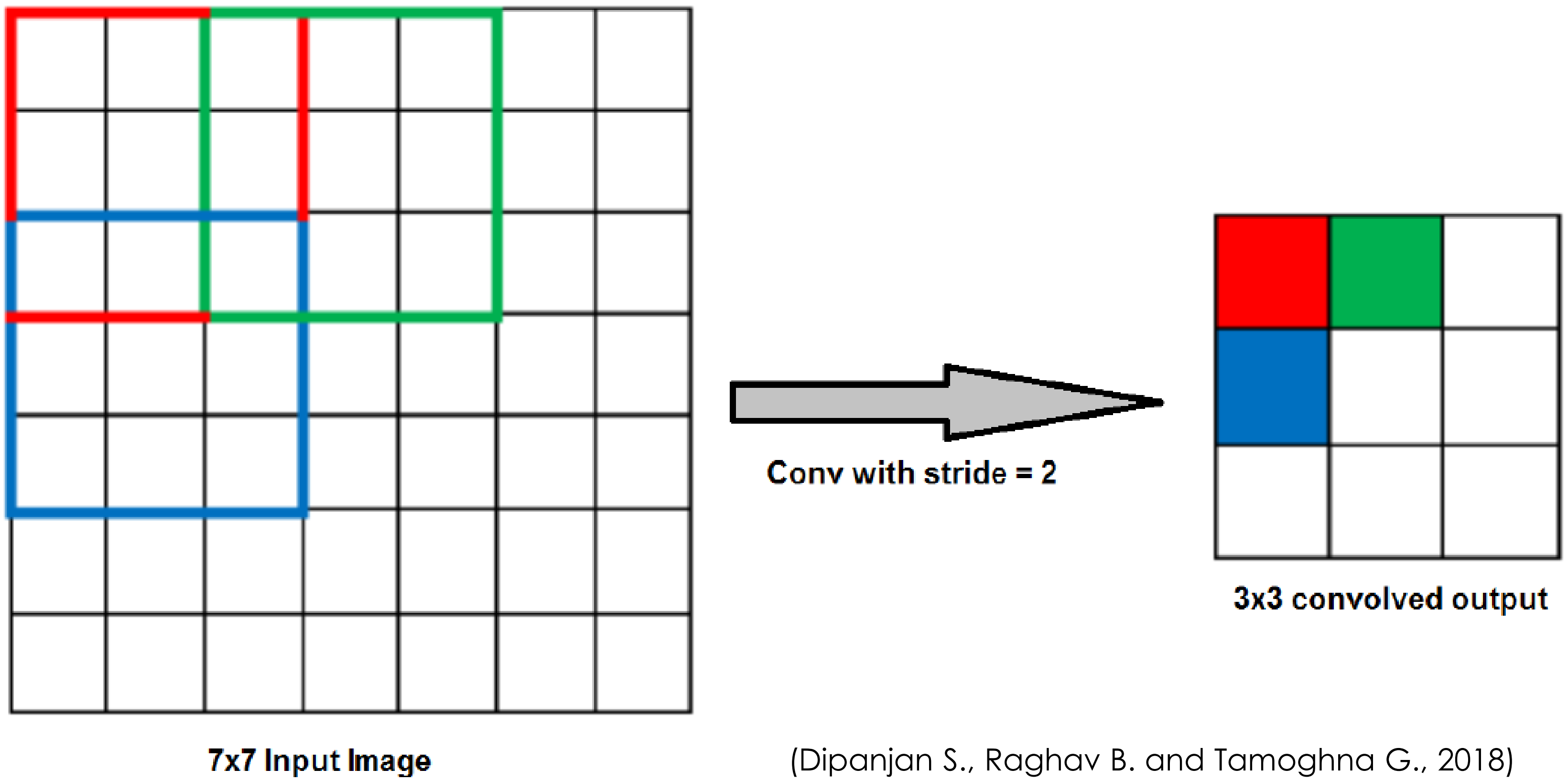


Source: https://www.youtube.com/watch?v=1GUgD2SBl9A , DigitalSreni, 2020

STRIDE & PADDING OPERATORS

- The convolution kernel convolves around the input volume by shifting one column/row at a time.
- the number of output units shrinks.

- The amount by which the filter shifts is called the stride.
- To preserve the size of the input, we need to pad zeros evenly around the input.



(Dipanjan S., Raghav B. and Tamoghna G., 2018)

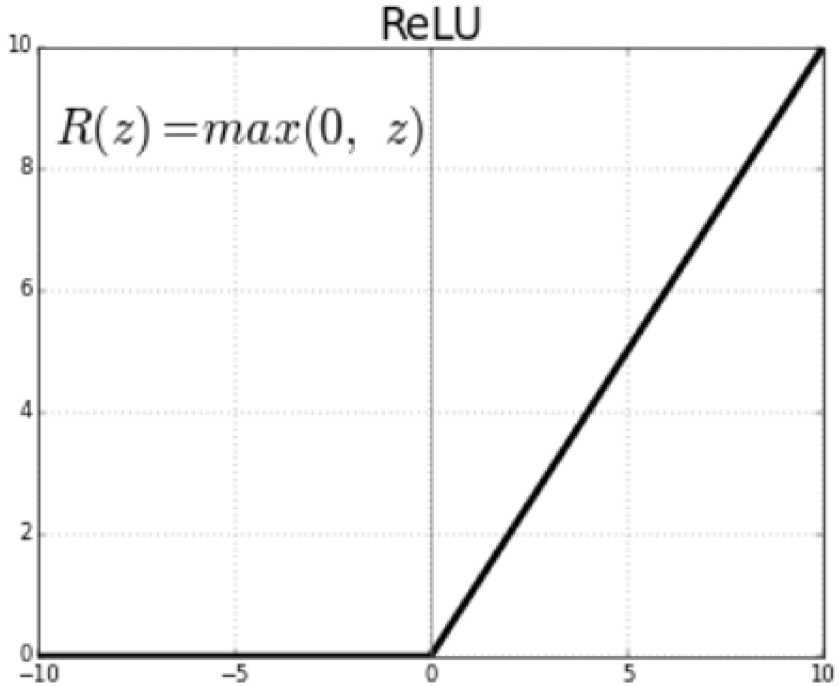
CONVOLUTION LAYER

consists of three major stages:

1. Feature extraction

- Each unit makes connections from a locally receptive field in the previous layer, thus forcing the network to extract **local features**.
- The weights associated with the receptive field of a hidden neuron is the kernel of the convolution.
- The output of this local linear activation is run through a nonlinear activation function, such as **ReLU** (Rectified Linear Units).

$$R(z) = \max(0, z)$$

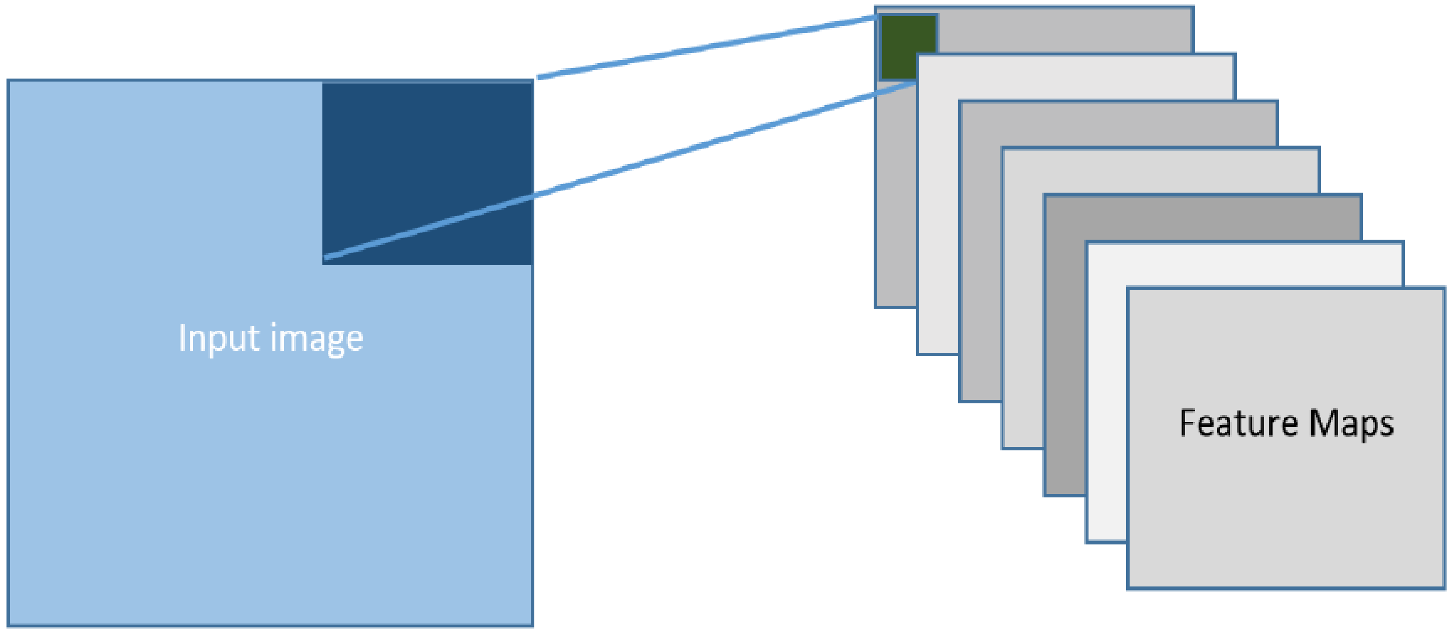


(Dipanjan S., Raghav B. and Tamoghna G., 2018)

CONVOLUTION LAYER

2. Feature mapping

- The feature detector creates a feature map that is in the form of a plane.
- To extract the different types of local features and have a richer representation of the data, several convolutions are performed in parallel to produce several feature maps.

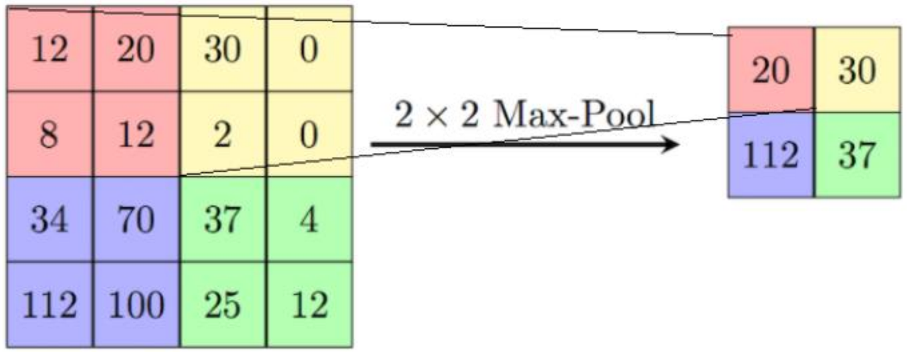


(Dipanjan S., Raghav B. and Tamoghna G., 2018)

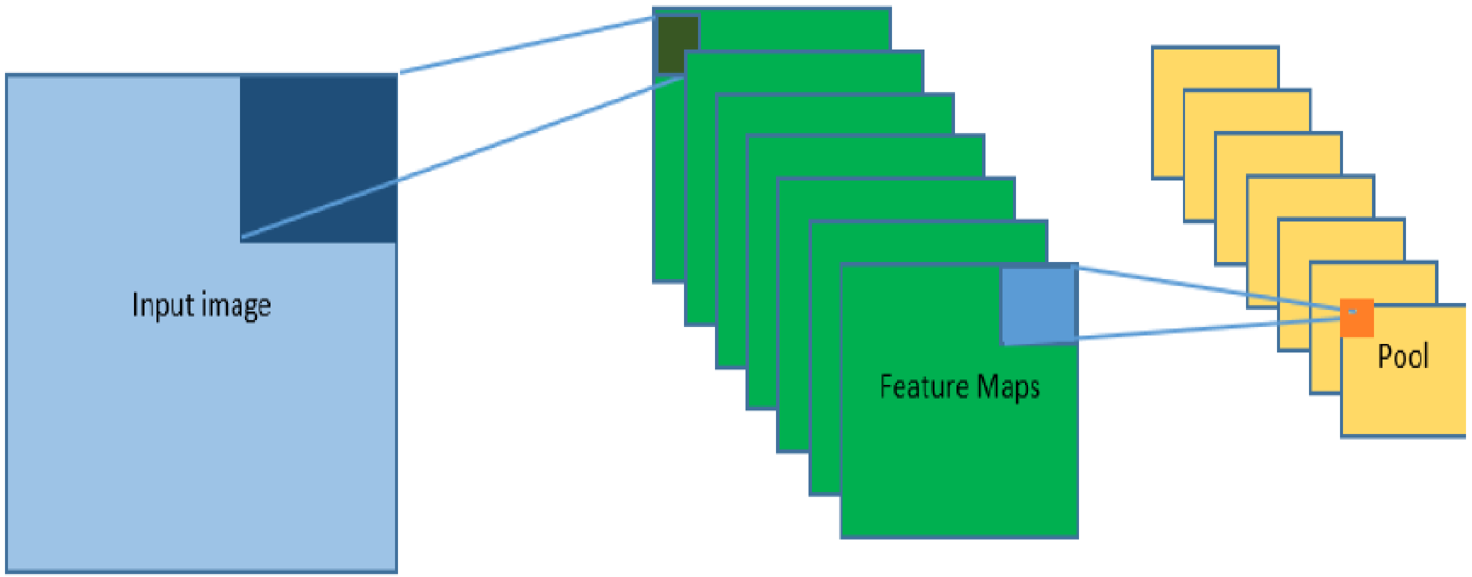
CONVOLUTION LAYER

3. Sub-sampling by pooling

- This is done by a computational layer that subsamples the output of the feature detector by replacing the feature detector units at certain locations with summary statistics of the nearby units.
- The summary statistics can be maximums or averages.



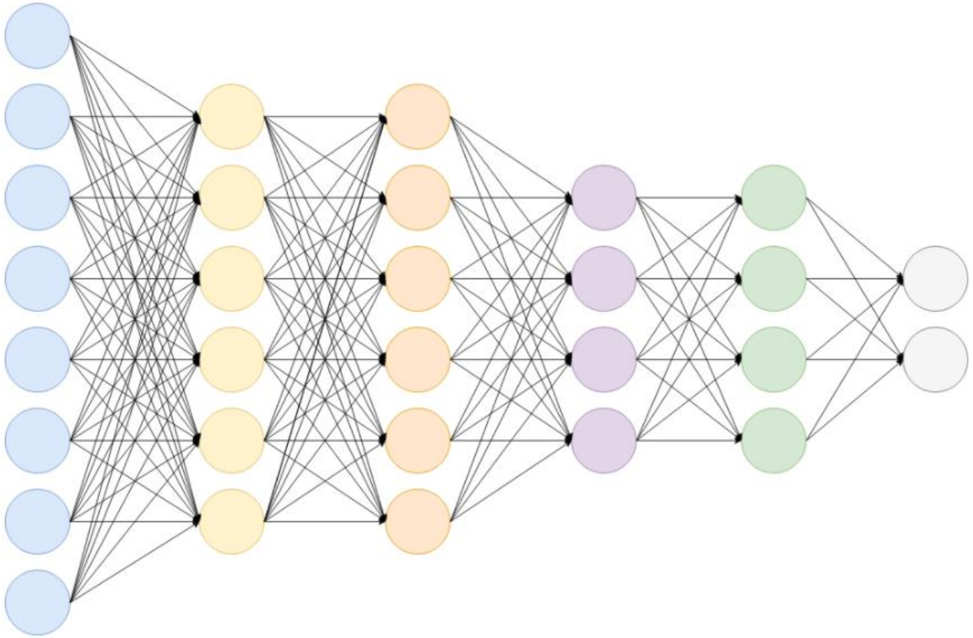
(Dipanjan S., Raghav B. and Tamoghna G., 2018)



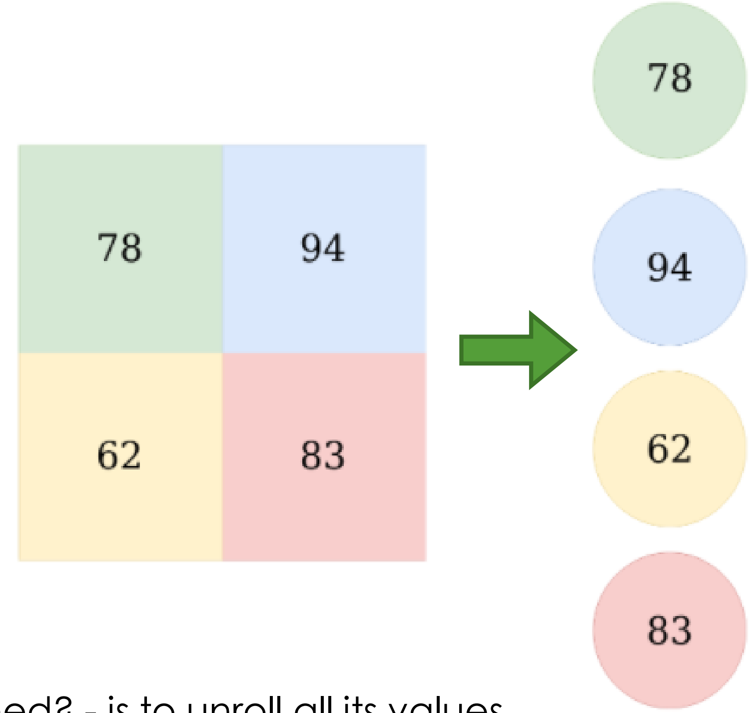
FULLY CONNECTED LAYER

- Fully Connected Layers are simply, feed forward neural network. They form the last few layers in the network.
- The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.
- Flattened? - is to unroll all its values into a vector.

(Dipanjan S., Raghav B. and Tamoghna G., 2018)



Fully connected layer architecture



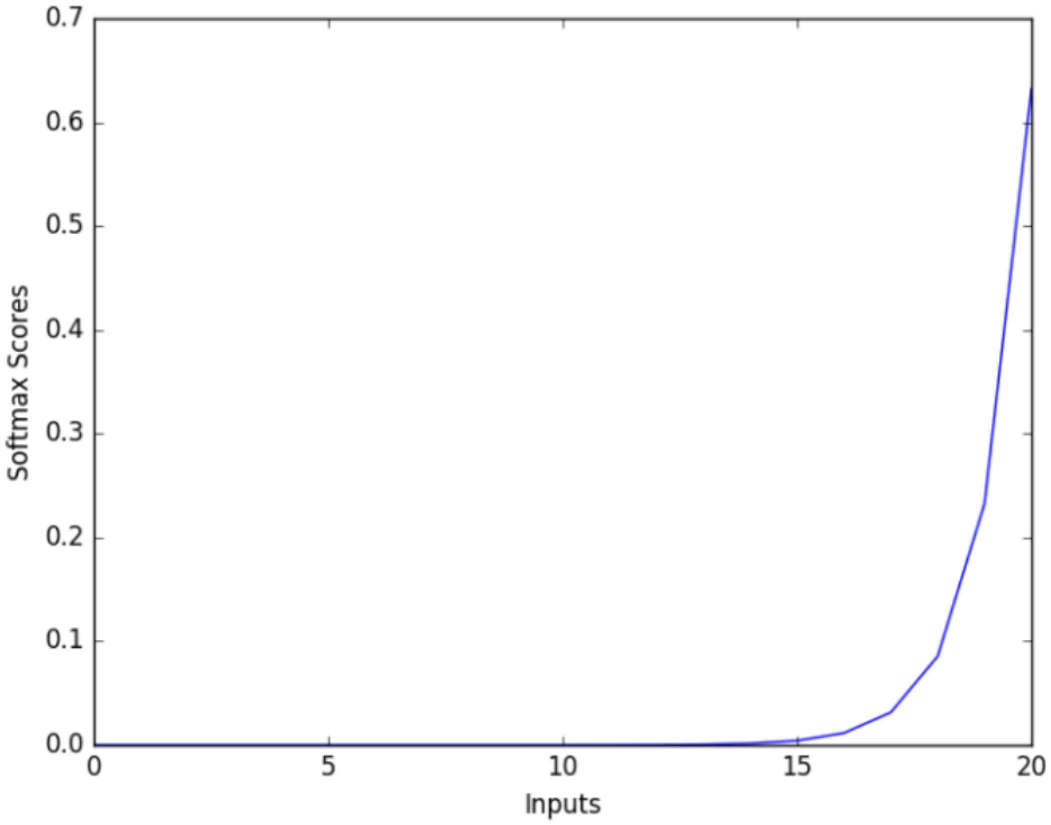
Flattened? - is to unroll all its values into a vector.

SOFTMAX FUNCTION

- **Softmax function** maps our output not only to a [0,1] range but also maps each output in such a way that the total sum is 1. The output of Softmax function is a probability distribution.
- Where z is vector of inputs to output layer and ***j*** indexes of the output units from *1,2, 3 …. k*

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

$$for \;\; j = 1, \cdots, K$$



(Dipanjan S., Raghav B. and Tamoghna G., 2018)

CNN ARCHITECTURES

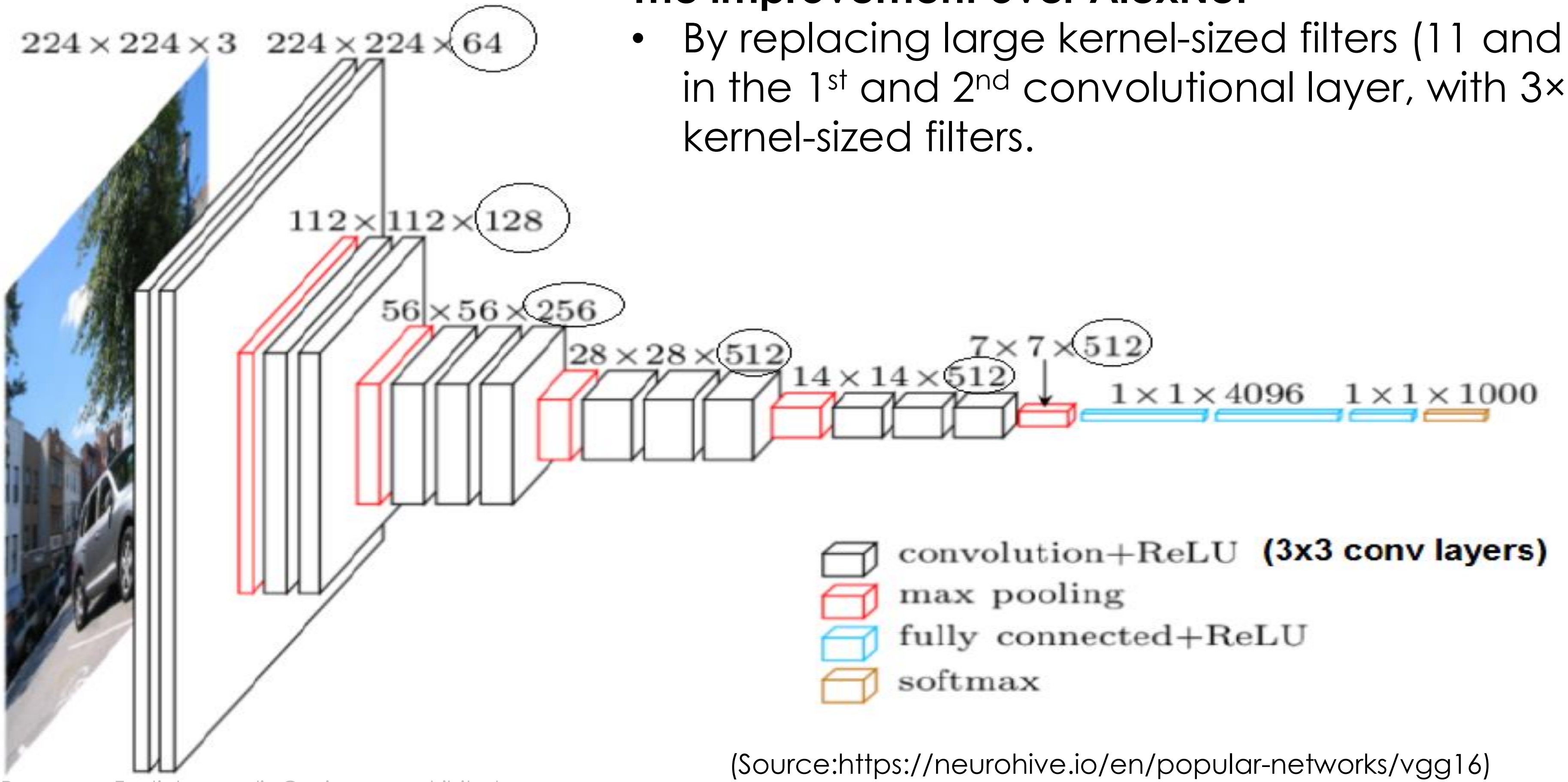
Architectures	Description
LeNet	A pioneering **seven-level** convolutional network, designed by LeCun and their co-authors in 1998.
ImageNet	Algorithms for object detection and **image classification** on a large scale (of LeNet)
AlexNet	A very similar architecture to that of LeNet, but has **more filters** per layer and is **deeper**. Applications are for Computer Vision.
XFNet	Improved on AlexNet by utilizing a **smaller filter size** in the first convolution layer helps to retain a lot of the original pixel information.
GoogleNet	Introduced a new architectural component using a CNN called the **inception layer**, that use larger convolutions, but also keep a fine resolution for smaller information on the images.
VGG	Developed by the Oxford Visual Geometry Group, it confirms the importance of **depth** in image representations
ResNet	Architecture with **skip connections** and batch normalization was introduced by Kaiming He and their co-authors from Microsoft Research Asia

VGG ARCHITECHTURE

- The 16-layered architecture VGG-16 is shown in the following diagram.
- Very Deep Convolutional Networks for Large-Scale Image Recognition.
- achieves 92.7% top-5 test accuracy.

The improvement over AlexNet

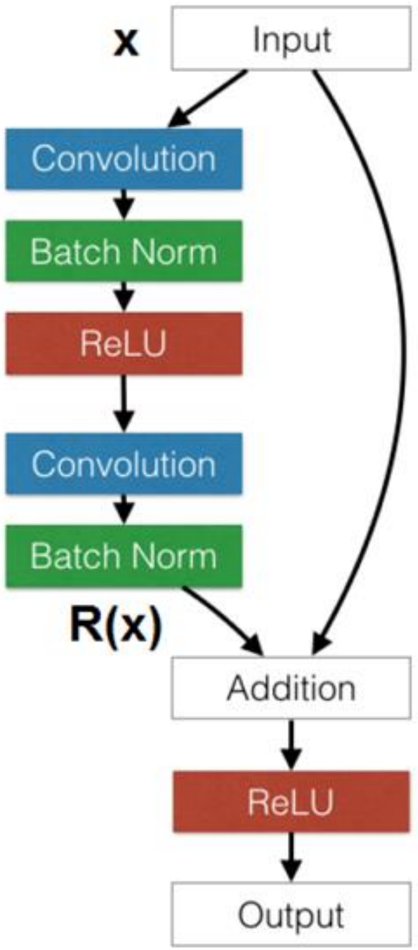
- By replacing large kernel-sized filters (11 and 5) in the 1st and 2nd convolutional layer, with 3×3 kernel-sized filters.



(Source:https://neurohive.io/en/popular-networks/vgg16)

RESNET (RESIDUAL NEURAL NETWORK) ARCHITECHTURE

- RESNETs have achieved the human level image classification result. They can extract low, middle and high-level features and classifiers in an end-to-end multi-layers.
- Achieve higher accuracy than AlexNet and VGGNet.

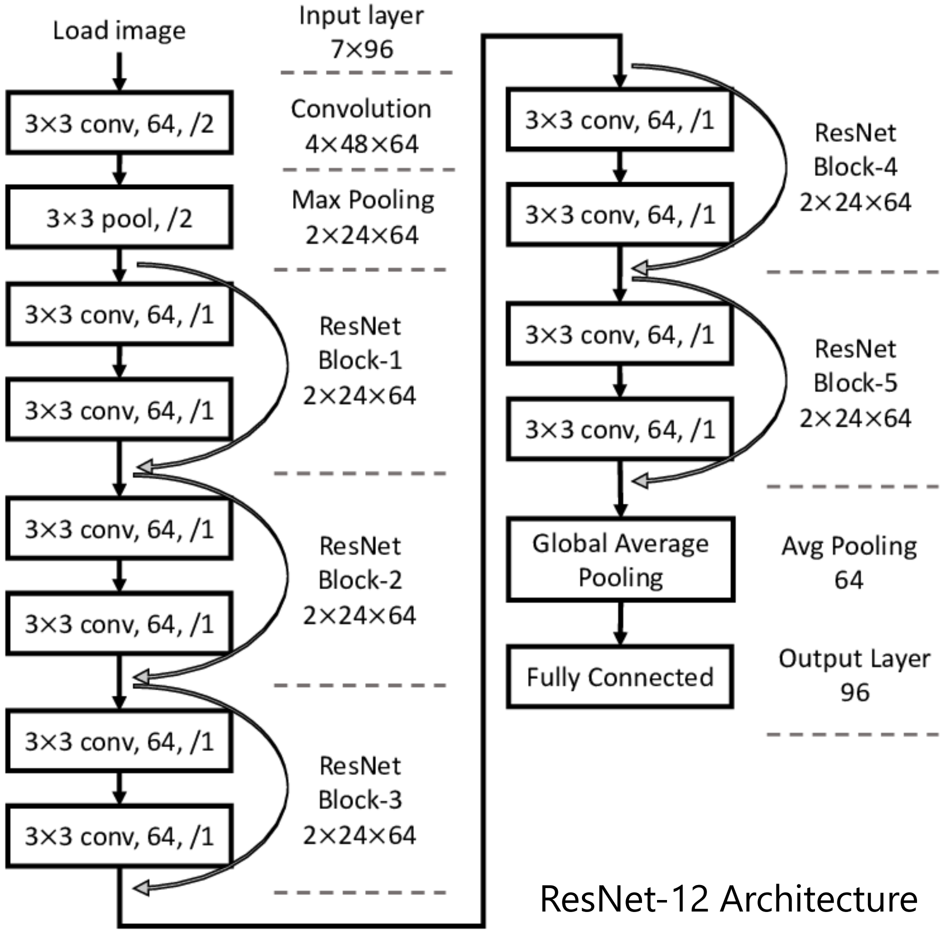


$$H(x) = R(x) + x$$

The formulation of R(x)+x can be realized with shortcut connections that work in two ways.

- Firstly, they alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through.
- Secondly, they enable the model to learn an identity function.

One building block of residual network



ResNet-12 Architecture

(Source:https://neurohive.io/en/popular-networks/vgg16)

EXAMPLE 1 IMAGE CLASSIFICATION USING A PRE-TRAINED MODEL

THE STRUCTURE

1. Loads the pre-trained model into Jupyter notebook.
2. Loads an image into the notebook.
3. Prepares the image for the model.
4. Predicts the probability across all output classes.
5. It converts the probabilities into labels.
6. Identifies the highest probability.
7. Displays the results

THE PROGRAM

Step 1: Open a new Jupyter notebook
Step 2: Import TensorFlow and Keras utilities into notebook
Step 3: Load the model into the notebook → VGG16(), ResNet50()
Step 4: Load an image into the notebook
Step 5: Prepare the image for the model
Step 6: Make the prediction
Step 7: Display the classification

(Silaparasetty N., 2020)

EXAMPLE 1 Image Classification Using a Pre-Trained Model Result

Program 1: Image Classification Using Pre-Training Model

In [1]:
```
#Import TensorFlow and Keras utilities

import tensorflow as tf
import tensorflow.keras

from tensorflow.keras.preprocessing.image import load_img,img_to_array

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input,decode_predictions

from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_input
```

In [2]:
```
# Load pre-trained model into the notebook

model = VGG16()
```

VGG16

In [3]:
```
#Load an image into the notebook

image = load_img('D:\DataScience\Image\Program1\Kitten1.jpg', target_size=(224,224))
```

In [4]:
```
image
```

Out[4]:

In [5]:
```
image = img_to_array(image)  #convert image's pixel into a Numpy array
```

In [7]:
```
import numpy as np
```

In [23]:
```
image1 = image.reshape(1, image.shape[0], image.shape[1], image.shape[2])
```

In [28]:
```
image1.shape
```

Out[28]: (1, 224, 224, 3)

In [26]:
```
image2 = preprocess_input(image)
```

In [29]:
```
image2.shape
```

Out[29]: (224, 224, 3)

In [38]:
```
label = decode_predictions(result)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 1us/step
49152/35363 [==========================================] - 0s 1us/step
```

In [49]:
```
label
```

Out[49]:
```
[('n02124075', 'Egyptian_cat', 0.8688644),
 ('n02123045', 'tabby', 0.05178757),
 ('n02127052', 'lynx', 0.038320195),
 ('n02123159', 'tiger_cat', 0.014145465),
 ('n02123597', 'Siamese_cat', 0.0032178778)]
```

EXAMPLE 2 Clothing (Fashon MNIST dataset) Classification Using CNN

The Structure

1. Loads the given dataset of images
2. Divides the data into the training set and the test set.
3. Creates the CNN with as many layers as we specify, along with the activation functions.
4. Creates the last dense layer and the output layer with the parameters.
5. Begins training with the training data to recognize the similarities and differences between each image and segregate them accordingly.
6. Finally, it tests its accuracy with the test data to see how well it has learned the differences between each image.

The Program

Step 1: Import TensorFlow library and Keras utilities.
Step 2: Load the Fashion MNIST dataset.
Step 3: Check the shape of the images.
Step 4: Reshape the input values.
Step 5: Prepare the data – normalize data by divided with 255
Step 6: Build the CNN
Step 7: Add the final dense layer and output layer.
Step 8: Compile the model.
Step 9: View the model.
Step 10: Train the model.
Step 11: Test the model.

(Silaparasetty N., 2020)

EXAMPLE 2

Clothing (Fashion MNIST dataset) Classification Using CNN

Result

In [209...
```
#Import TensorFlow and Keras utilities

import tensorflow as tf
import tensorflow.keras
from tensorflow import keras
from tensorflow.keras import datasets, layers, models

from tensorflow.keras.preprocessing.image import load_img,img_to_array
from tensorflow.keras.preprocessing import image

import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np
```

In [213...
```
#Step2 Load the Fashion MNIST dataset
```

In [214...
```
data = datasets.fashion_mnist
```

In [215...
```
(ip_train,op_train), (ip_test,op_test) = data.load_data()
```

In [216...
```
#STep3 Check the shape of the images
```

In [217...
```
print(ip_train.shape, ip_test.shape)
```

(60000, 28, 28) (10000, 28, 28)

In [218...
```
print(op_train.shape, op_test.shape)
```

(60000,) (10000,)

In [219...
```
#Step4 Reshape the input values
```

In [220...
```
ip_train = ip_train.reshape((60000,28,28,1))
ip_test = ip_test.reshape((10000,28,28,1))
```

In [221...
```
print(ip_train.shape, ip_test.shape)
```

(60000, 28, 28, 1) (10000, 28, 28, 1)

In [222...
```
#STep5 Prepare the data
```

In [223...
```
ip_train, ip_test = ip_train/255.0, ip_test/255.0
```

In [224...
```
#STep6 Build CNN - require convolutional layers and pooling layers - No need to flatten the images befo
# two-dimensional data. Con2D
```

In [225...
```
model = models.Sequential()
```

In [226...
```
model.add(layers.Conv2D(32,(3,3),activation="relu",input_shape=(28,28,1)))  #32 filters or kernels, ea
```

In [227...
```
model.add(layers.MaxPooling2D((2,2))) # the previous layer output pass through with pooling filer of siz
```

In [228...
```
model.add(layers.Conv2D(64,(3,3),activation="relu"))
```

EXAMPLE 2 CLOTHING (FASHON MNIST DATASET) CLASSIFICATION USING CNN RESULT

In [229…
```
model.add(layers.MaxPooling2D((2,2)))
```

In [230…
```
model.add(layers.Conv2D(64,(3,3),activation="relu"))
```

In [231…
```
#STep7 Add the final dense layer (full connected layer) and followed by the output layer.  Here we need
# to the dense layer.
```

In [232…
```
model.add(layers.Flatten())  #flatten teh input before feeding
```

In [233…
```
model.add(layers.Dense(64,activation='relu')) #full connected layer with 64 neurons
```

In [234…
```
model.add(layers.Dense(10,activation='softmax')) #output layer or the classification layer.
```

In [235…
```
#Step 8 Compile the model
```

In [236…
```
model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
```

In [237…
```
#Step9 View the model
```

In [238…
```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_5 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                36928
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

EXAMPLE 2 CLOTHING (FASHON MNIST DATASET) CLASSIFICATION USING CNN RESULT

In [239…

```
#STep 10 Train the model
```

In [241…

```
model.fit(ip_train,op_train,epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.3228 - accuracy: 0.8813
Epoch 2/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.2772 - accuracy: 0.8986
Epoch 3/5
1875/1875 [==============================] - 16s 9ms/step - loss: 0.2464 - accuracy: 0.9088
Epoch 4/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.2237 - accuracy: 0.9180
Epoch 5/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.2027 - accuracy: 0.9253
```

Out[241…

```
<keras.callbacks.History at 0x16bc8de5ac0>
```

In [242…

```
#Step11 Test the model
```

In [243…

```
model.evaluate(ip_test,op_test,verbose=1)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2690 - accuracy: 0.9026
```

Out[243…

```
[0.26898252964019775, 0.9025999903678894]
```

THE END