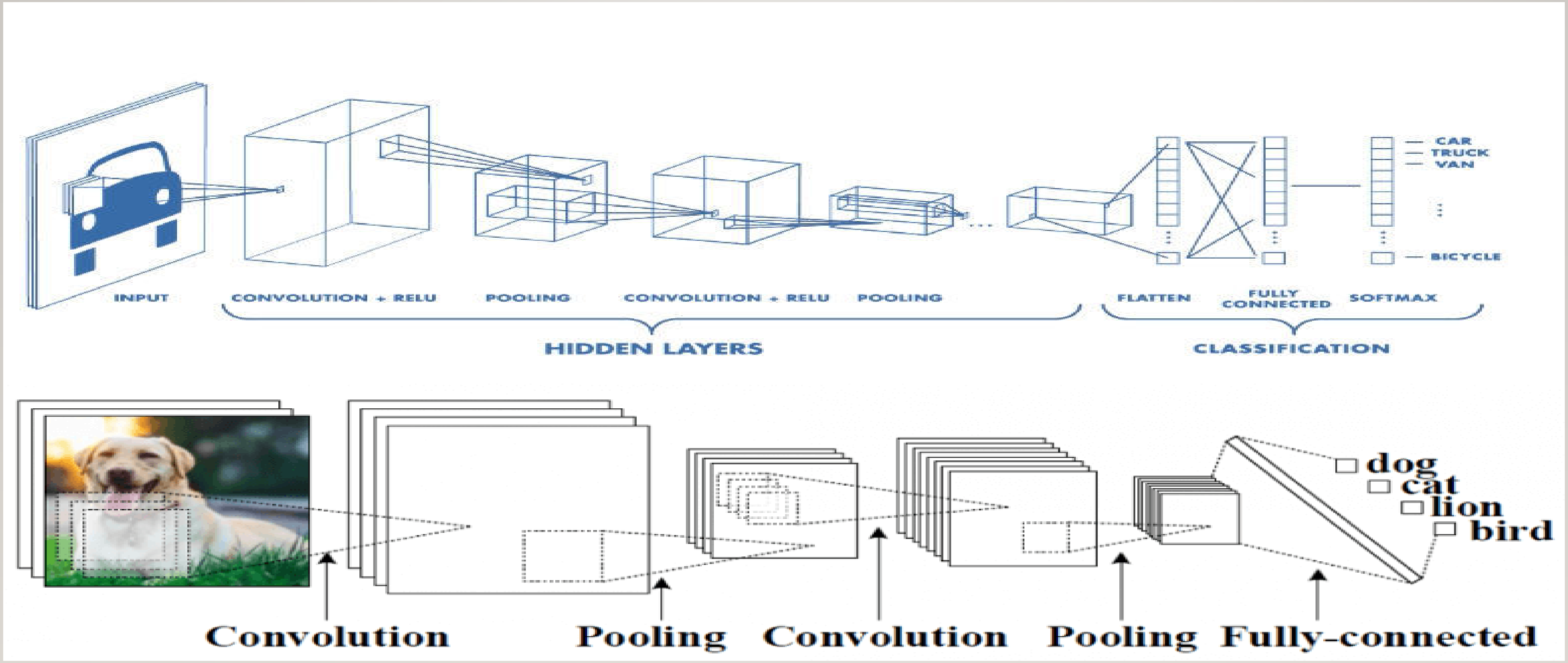
IMAGE RECOGNITION MODEL BASED ON CONVOLUTIONAL NEURAL NETWORKS



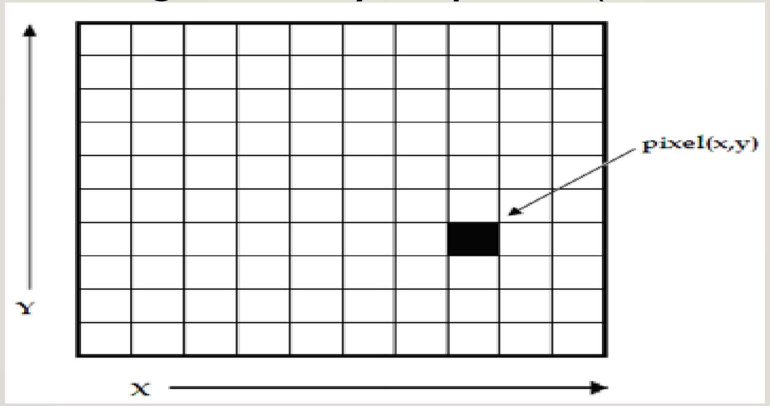
PREPARED & PRESENTED BY: AMIT TIWARI

TABLE OF INDEX

Code link: https://github.com/smilyamit/Machine-Learning/tree/master/Convolutional_Neural_Networks

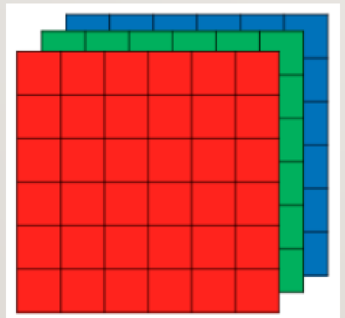
WHY CNN ?

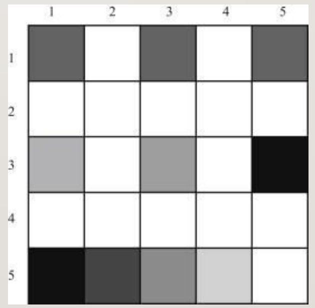
- In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the model which are mainly involved in images classifications, Objects detections & recognition of faces.
- CNN image classifications takes an input image, process it and classify it under certain categories (e.g. Bear or Dear, Lion or Leopard)
- Computers sees an input image as array of pixels (A combination of rows and column)



INPUT TO CNN

- CNN will see h x w x d as (h = Height, w = Width, d = Dimension).
- For e.g. An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB dim.) and an image of 5x 5 x 1 array of matrix of grayscale image (1 refers to grayscale dim.)


6 x 6 x 3



5x5x1

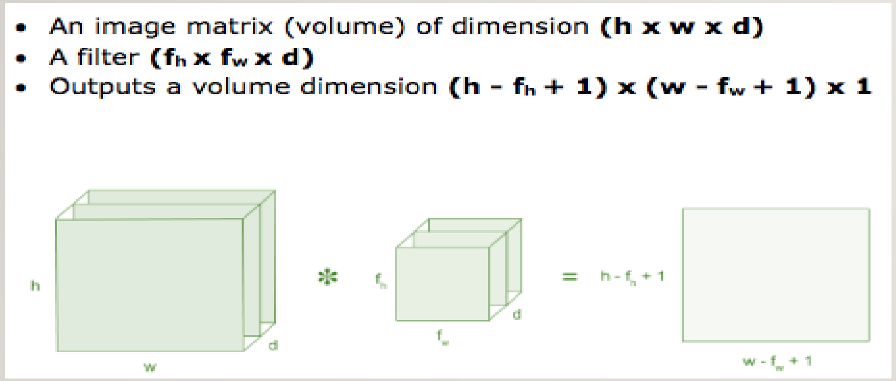
CORE WORKING PRINCIPLE OF CNN

- Each input image will pass it through a series of convolution layers with filters (Kernals) than subjected to RELU function, Pooling, Flattening and after that fully connected layers is formed and finally by applying Softmax or Sigmoid function classification of an object with probabilistic values between 0 and 1 is obtained.
- key terms related to CNN are Convolutional Layer (Image Matrix+Filter with Strides), Padding, Non-Linearity Function (e.g. ReLU), Pooling Layer (Synergy of Parent Matrix, Strides & Pooling tricks), Flattening, Fully Connected Layer & Activation Function (e.g. Softmax or Sigmoid)

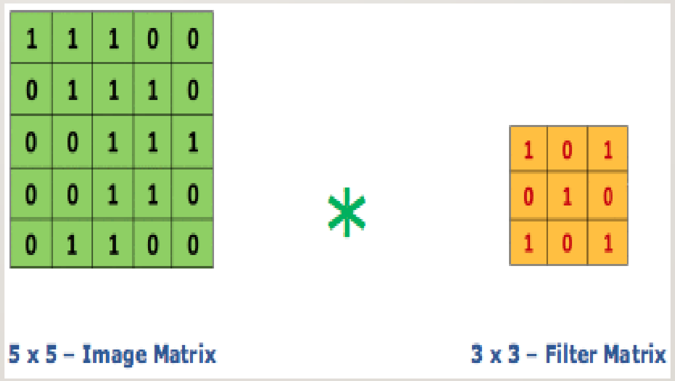
MAJOR KEY TERMS

❖ Convolutional Layer:

➢ Convolution is the first layer to extract features from an input image. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.



- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



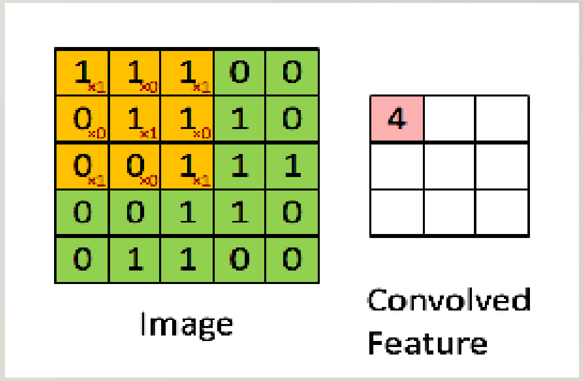


Fig. Image matrix multiplies kernel or filter matrix

MAJOR KEY TERMS

- Strides:

- Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on.

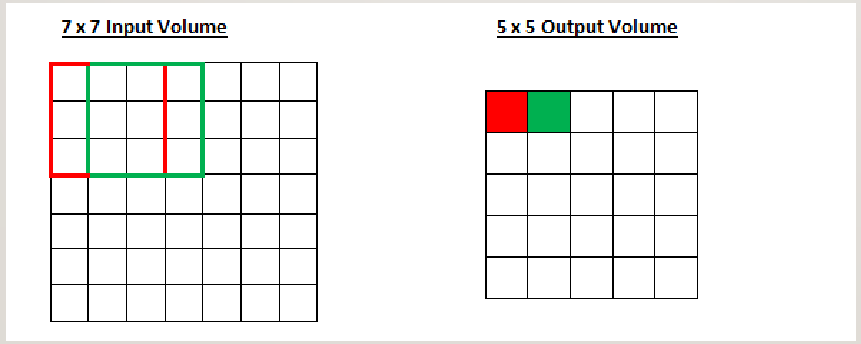


Fig. Example of Stride of one

MAJOR KEY TERMS

❖ Padding:

➢ Sometimes filter does not fit perfectly to the input image matrix. We have two options:

1. Pad the picture with zeros (zero-padding) so that it fits
2. Drop the part of the image where the filter did not fit

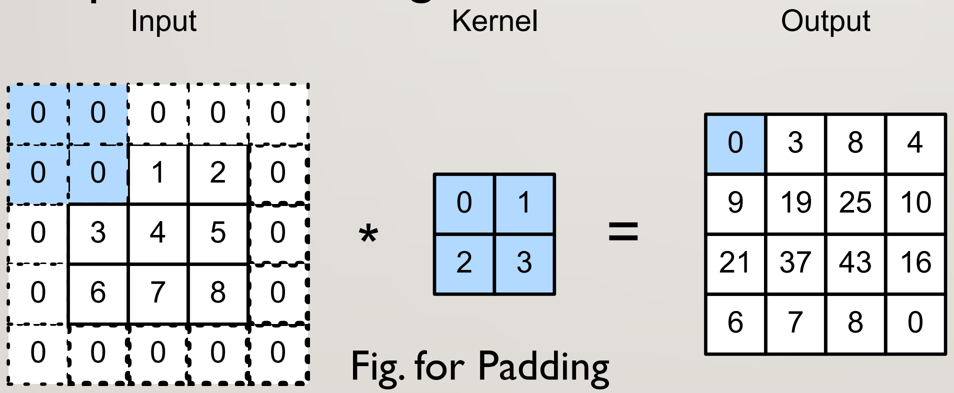


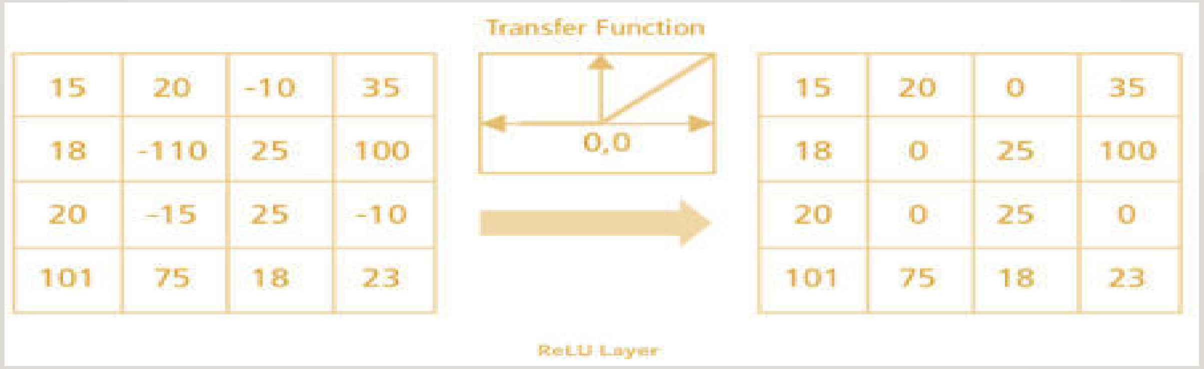
Fig. for Padding

MAJOR KEY TERMS

❖ **Non Linearity (ReLU)**

➢ ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ***f(x) = max(0,x)***

➢ Why ReLU is important : 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



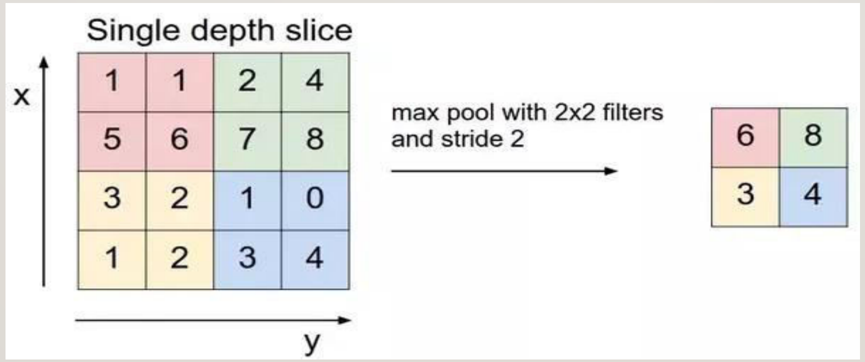
15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23

15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23

MAJOR KEY TERMS

❖ **Pooling Layer:**

➢ Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types: e.g. Max Pooling, Average Pooling, Sum Pooling



MAJOR KEY TERMS

❖ **Fully Connected Layer:**

➢ The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.

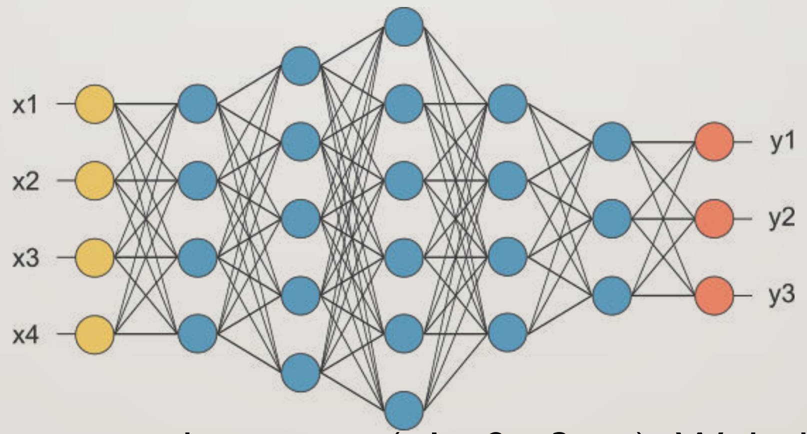


Fig. After pooling layer, flattened as FC layer

In the above diagram, feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as Bear, dear, cat, dog etc.

Algorithm of CNN For Real World Application

```python
# Installing Tensorflow & Keras at once:
# conda install -c conda-forge keras

# Part 1 - Building the CNN

# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 64,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 64,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         steps_per_epoch = 8000/32,
                         epochs = 4,
                         validation_data = test_set,
                         validation_steps = 2000/32)

# Part 3 - Making new predictions

import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)       # to make 3 dim image
test_image = np.expand_dims(test_image, axis = 0) # to make extra 1 dim for batch, while inputting total dim = 4
                                                  # axis = 0 as first index is equal to 0
result = classifier.predict(test_image)
training_set.class_indices    # class_indices attribute used to show their associated numl value 0 or 1
if result[0][0] == 1:         # that is mapping of cat n dog
    prediction = 'dog'
else:
    prediction = 'cat'
```

Mathematical Intuition

Error Calculation:

In NN1

Dog	cat	Dog	Cat
0.9	0.1	1	0
0.1	0.9	0	1
0.4	0.6	1	0

Mean Square Error: $1/n \Sigma (y-y')^2$

MSE for Dog = 0.12667

MSE for Cat = 0.12667

MSE = MSE for Dog + MSE for Cat = 0.25

Cross Entropy Error: $-\Sigma p(x). \text{Log}(q(x)) = 1.126$

Where,

P(x): Calculated outcomes based on weights(W) in CNN

q(x): Real value