Artificial Intelligence

Convolutional neural network for image processing and feature detection

SUBMITTED BY

Usama Naseer 307-BsCs-17

SUBMITTED TO

Sir Ali Raza

Roll NO: 307-BSCS-17

Introduction and overview:

My project is basically a image processing and feature detection which is based on deep learning .so we use algorithm and technique with the help of artificial intelligence ,Machine learning and computer vision they based on these languages due to they are interconnected to each other so there are three main type used in deep leaning supervised ,unsupervised and reinforcement learning ,generative algo also so in supervised leaning we also predict or and detect due to labeled data and unsupervised they used technique clustering and gain knowledge with the help of previous knowledge so in supervised they fit the model in logistic regression and unsupervised we use activation function and support vector machines the fitting data is also used in neural network in hidden layer we also use activation function and optimization technique,some layers are common in ML ,NN and CNNS so these techniques are interlinked eavch other sp applied machine learning is a highly iterative process .in my own pointv of views deep learning are based on algorithms,layers,optimization technique so they implemented in new way manner u build new things such as in CNNS we used 16 type of layer and efficient working they are called res Net so u have some time use pooling layers and applied maximum pooling and on other hand if efficiency is not good u changed its technique use strides and average polling and they robust the process and fit for data set not such hard and fast rule in deep learning and some time we used different type of optimization sometime some optimizer fails to minimum the error,so applied Machine learning ,artificial intelligence and deep learning is a iterative process

#layers

#hidden units

#feed forward

#back propagation
#activation function

#optimization

#support vector machine

#gradient Decent

#Regularization

#validation process or dev test

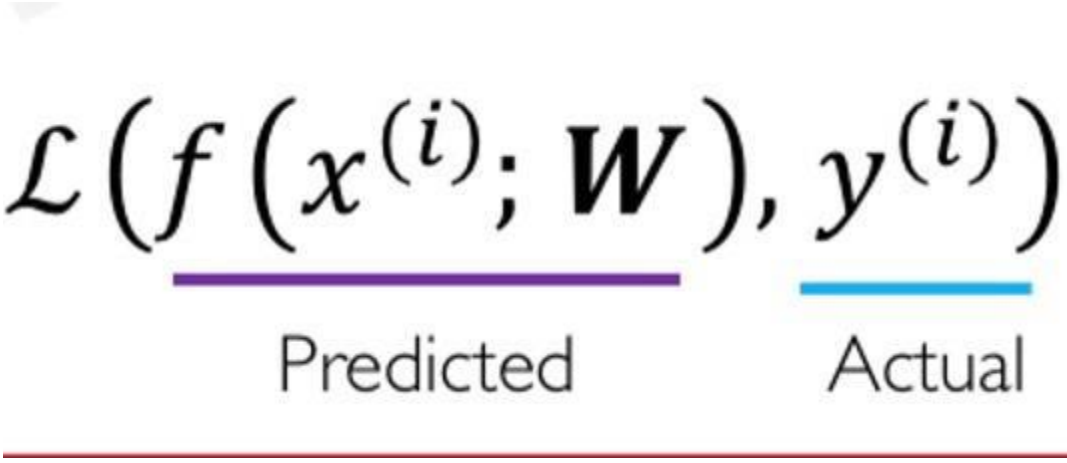
#test and train the data

#loss function

I have discussed some common technique which are necessary to discuss if I am not eleobrate they have not meaning at alone they are interconnected to each other and used in AI,ML,CV and deep learning so

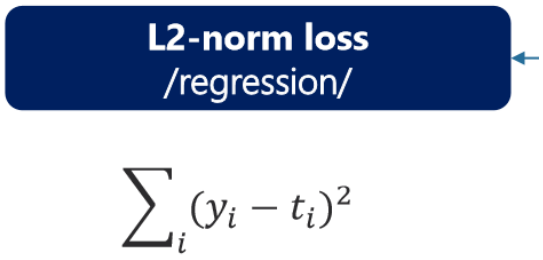
Loss function :

L1-norm



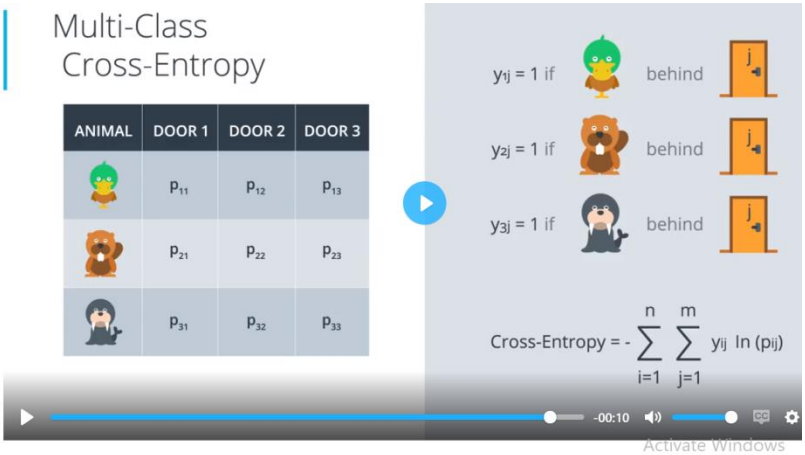
L2-norm or Means Squared error : The L2-norm of a vector, a, (Euclidean length) is given bySo, the closer to the origin is the difference of the outputs and the targets, the lower

the loss, and the better the prediction.



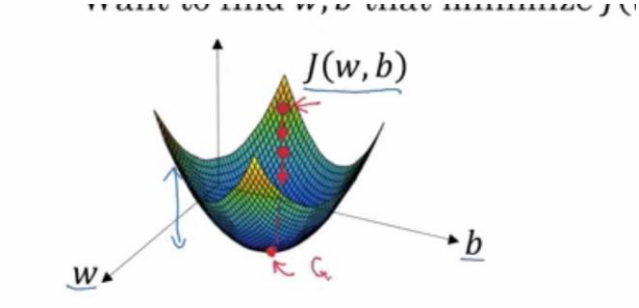
cross entropy :

The cross-entropy loss is mainly used for classification.basically its mechine learning technique to check the model output for its probabilistic nature if the higher probability classification they assighn to 1 and the either discard they gave out put if it is correct is 1 by definition). If the cross-entropy is 0, then we are not missing any information and have a perfect model. And for classification we also use hot encoding technique to slove



Gradient decent:

Gradient decent is used for optimize the error in give function for example if our estimate value is not match our prediction value so we want to get accurate result then it is used gradient decent model the famous example is mount Everest Example if I am in top of the mountain then I am back to ground this technique is helped to back in to ground at shortest path .the gradient decent is apply on the back propagation to minimize the error and its help to to predict new random weights to accomplish our goals .the implementation is explain its picture how to applied this algo in neural network the coast function help to predict which type of error occur in our function which is positive or negative if the derivate is positive they minimize if the first derivative is positive, then we are on the right side of the parabola. Subtracting a positive number from xi will result in a lower number, so our next trial will be to the left (again closer to the minimum). if the first derivative is negative , then we are on the left side of the parabola. Add a positive number from xi will result in a higher number, so our next trial will be to the right (again closer to the minimum).



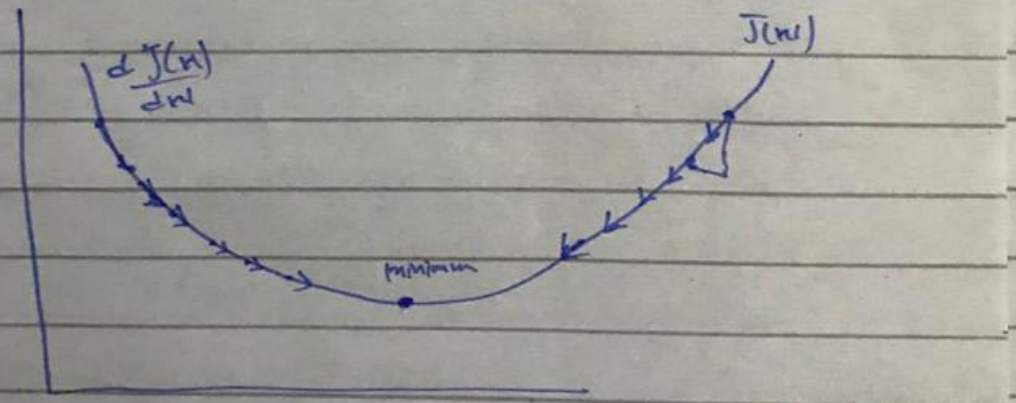
Gradient decent

$$\hat{y} = \sigma(w^T x + b)$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(\hat{y})^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$



$J(w,b)$

$$w : w - \alpha \frac{dJ(w,b)}{dw} \quad \therefore \frac{dJ(w,b)}{d(w)} = d(w)$$

$$b : b - \alpha \frac{dJ(w,b)}{d(b)} \quad \therefore \frac{dJ(w,b)}{d(b)} = d(b)$$

Gradient decent. Multivariate derivation:

$y = xw + b$

Loss = $L = \frac{1}{2} \sum_i (y_i - t_i)^2$ → mean squared error. → L2-norm.

Optimization algorithm

$w_{i+1} = w_i - \alpha \, dw L(w_i)$

$b_{i+1} = b_i - \alpha \, db L(b_i)$

$$dw L = dw \frac{1}{2} \sum (y_i - t_i)^2$$

$$= dw \frac{1}{2} \sum (x_i w^T + b) - t_i)^2$$

$$= \sum_i dw \frac{1}{2} (x_i w^T + b - t_i)^2 =$$

$$= \sum_i x (x_i w^T + b - t_i)$$

$$= \sum x_i (y_i - t_i) =$$

$$= \sum_i x_i \delta_i$$

Gradient check for neural network

Take a weight and bias is initialize to w[1] b[1]……..w[l]b[l] and reshape in to a big vector theta

Take dW[1]dB[1]…..dW[l]dB[l] and reshape in to big vector theta

Find the loss l2 is mean squared loss such as actual value –output value and then squared to it

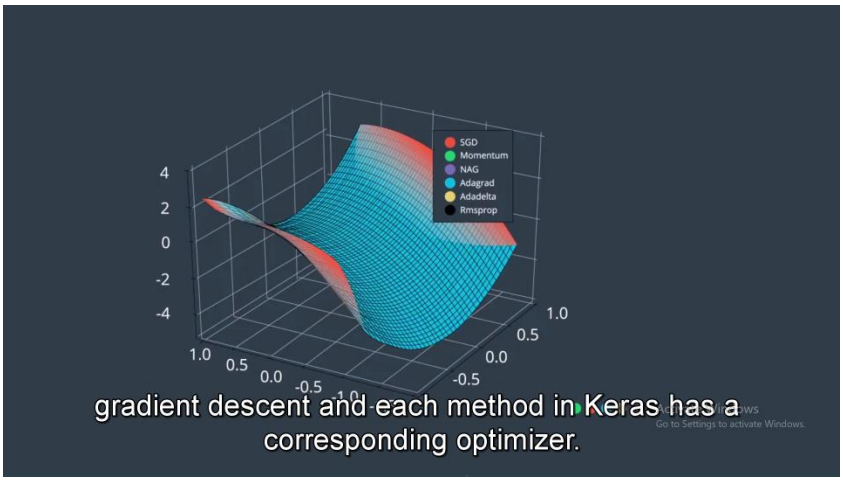
Gradient is not use for training its only for debug

If algorithm fails for gradient checking look at component to try to identify bug

Remember Regularization

Doesn't work with dropout

Repeat and update weight again in training example until the not reach at its goal.

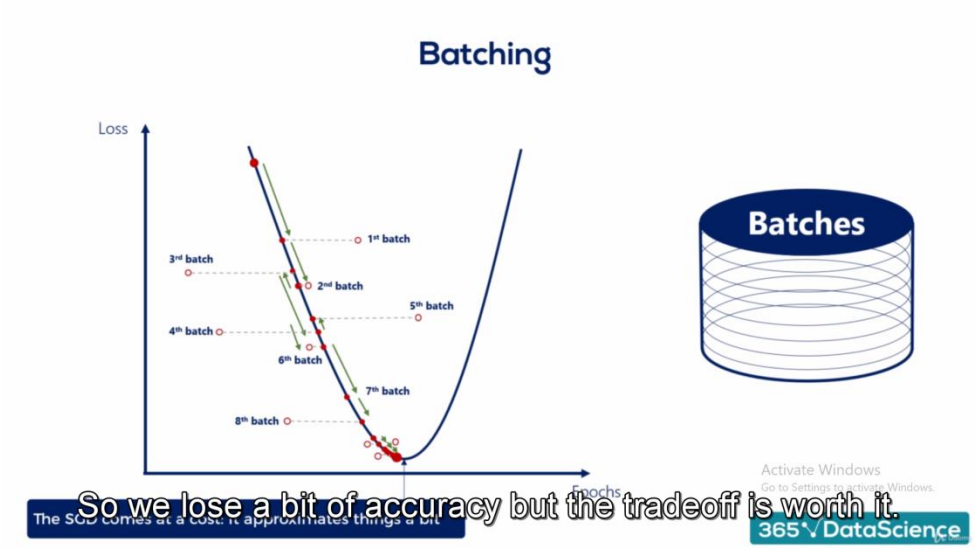


Batch/ Mini- batch Stochastic Gradient decent:

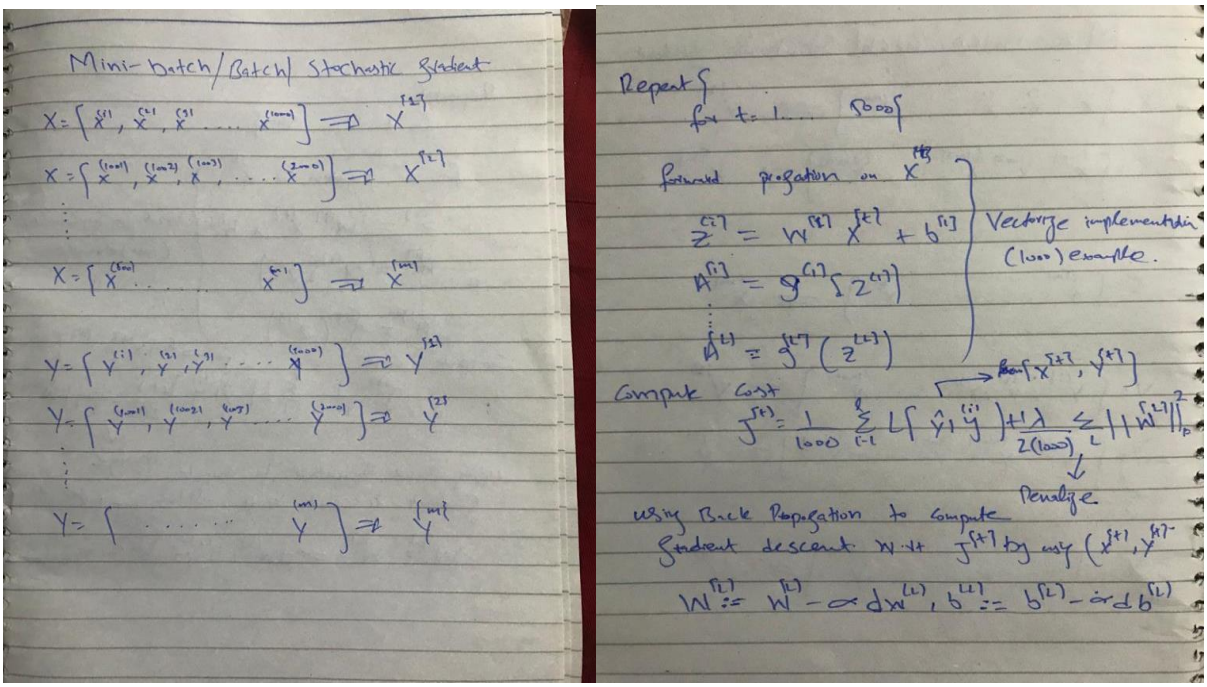
Mini batch gradient descent is used to divide the subset in to chunks form then they divide in to batches

Why I'm used this mini batch technique to improve the system speed and efficiency and less time to training large no of data set contains more time.Vectorization allows you to efficiently compute on m example . and thy also help full also to implement gradient or loss function in batch wise some time its helpful to minimize the error and sometime converge the minimum position and the most power full

technique is avoid the regularization means over fit or under fit the data and stochastic gradient is used for 1 batch and full batch it is used for gradient decent algo stochastic gradient work at same for gradient but its apply for one batch and update the weights in many times .in bottom mean mini batch is not too big its work firstly accurate and not fits the data and reaches gradient descent fasten and its vectorization is nearly 1000 per batch in stochastic gradient its lose speed for vectorization and in batch gradient we use vectorization in million data set its work slowly .

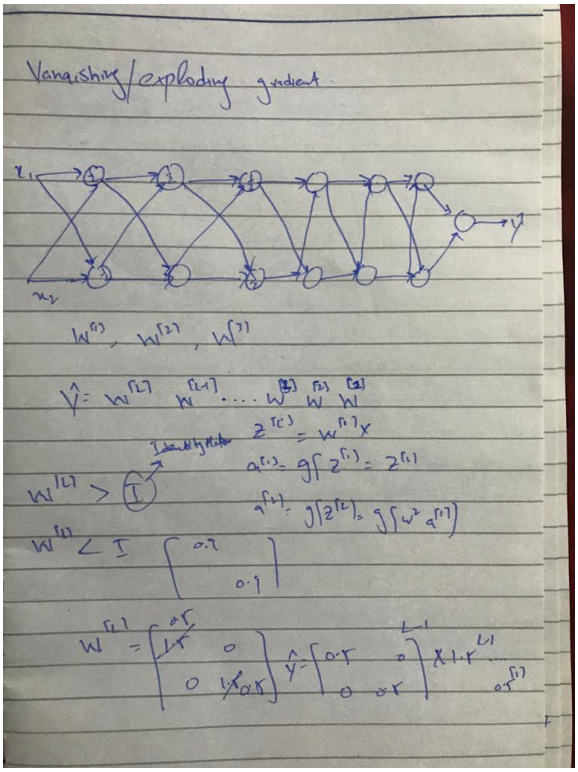


Mini Bach algorithm working :



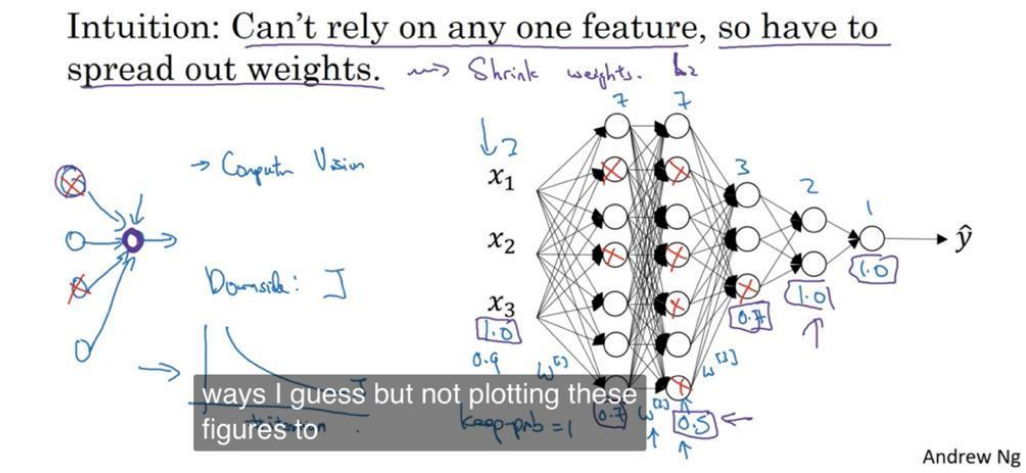
Vanishing /exploding gradient:

This is also a same technique in which previously implemented in gradient decent by pack propagation take the partial derivate and applied chain rule to update the weights and minimize error in this case when the partial derivate value is gradually increase due to exponential rate so this type of training is exploding and in the same case in training the partial derivate is exponentially decreases gradually so they show vanishing



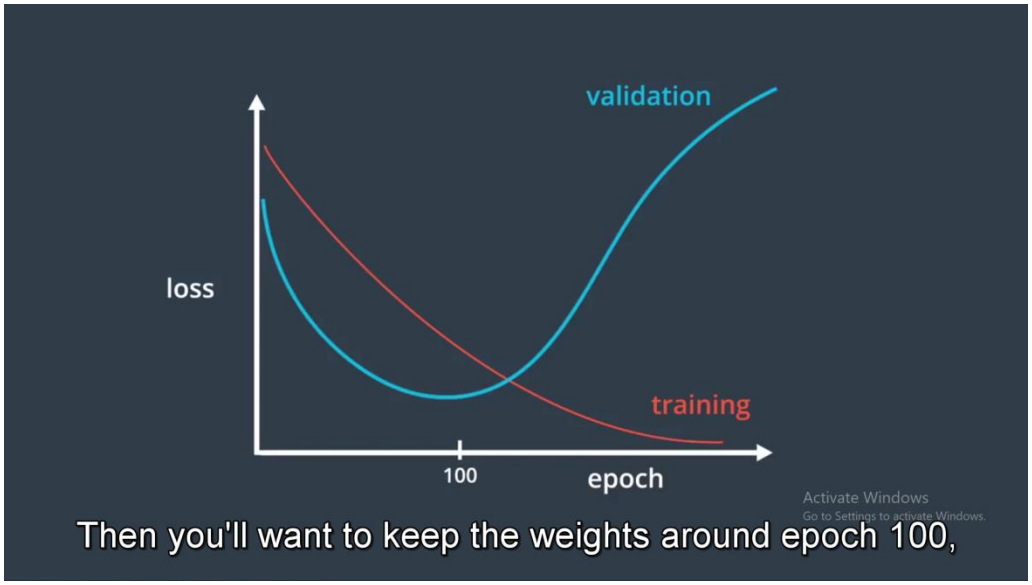
Dropout layer:

Drop out layer is used to enhance and speed the training data set the drop out process is applied when they have less contribute in our neural network how to check what neuron is taking part or not we check our probability and assign if the probability level is less than the assigning value they drop out in training process and its important thing is that they not implemented in final layer output



Early stopping algo:

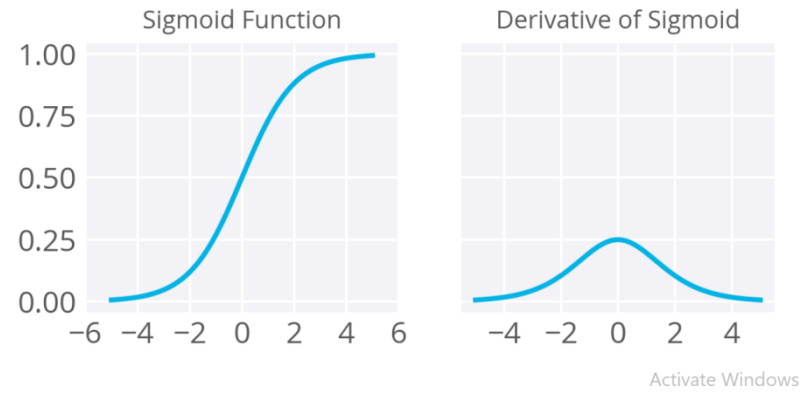
Early stopping algorithm is used to prevent the over fitting the data over fitting is produced due to high variance update the weights many time and they fit the data very well in training data set so they overcome these fitting problem we also use validation test or deployment test or dev. set they recognize and tell what point the start fitting so if the validation test generally increase and test test decrease at that point they use early stopping algo



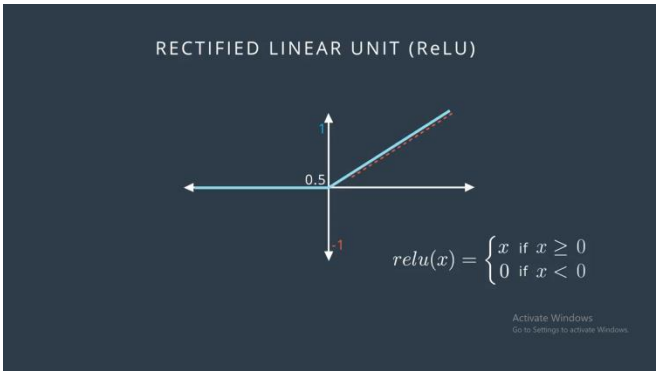
Activations Functions:

Is used to find the non-linearity in our model so there are many activation function used in deep neural network

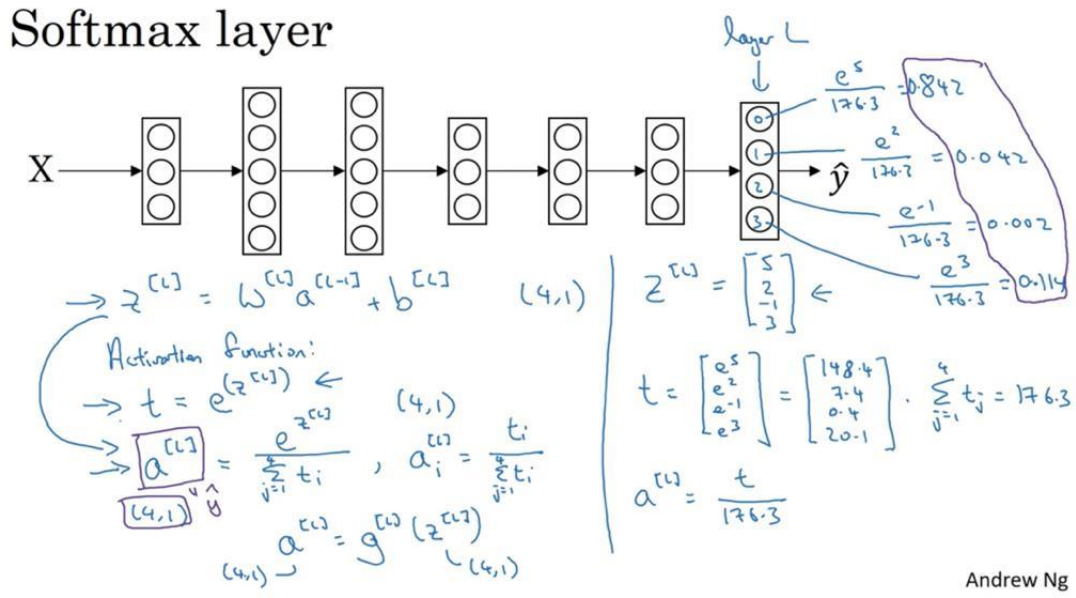
Sigmoid activation: is used to describe the data in linear form. They remove all non-linearity in our model its help to change the value of range from 0 to 1



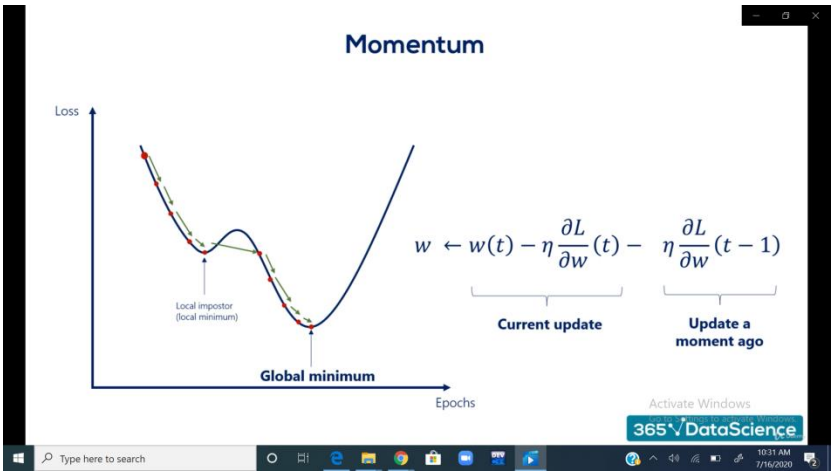
Relu activation: Rectified linear activation function Is basically used for find max value of our function and its also used to represent the positive values only if the value is positive they assign the value to 1 if the value is negative they assighn the value zero.



Softmax function: is basically a exponentially value they are basically for classification with the help of probabilistic model .so they classified the objects its high activation value.

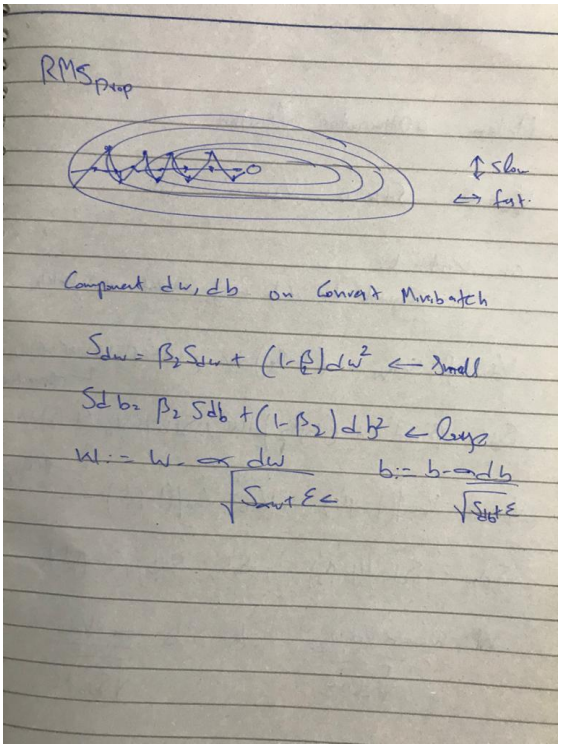


Momentum: is not a activation function but it is used to pull and enforce the activation function to speedy the process basically its is used for when the function is stopped in local minima and they do not reached the global minima l so they stuck at this location so we use momentum and use another technique is also random initialize the position and start function some time its help full and some time its not help full .so momentum is work they update its weight by average of pervious weight

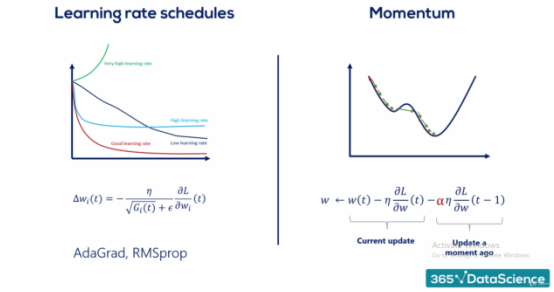


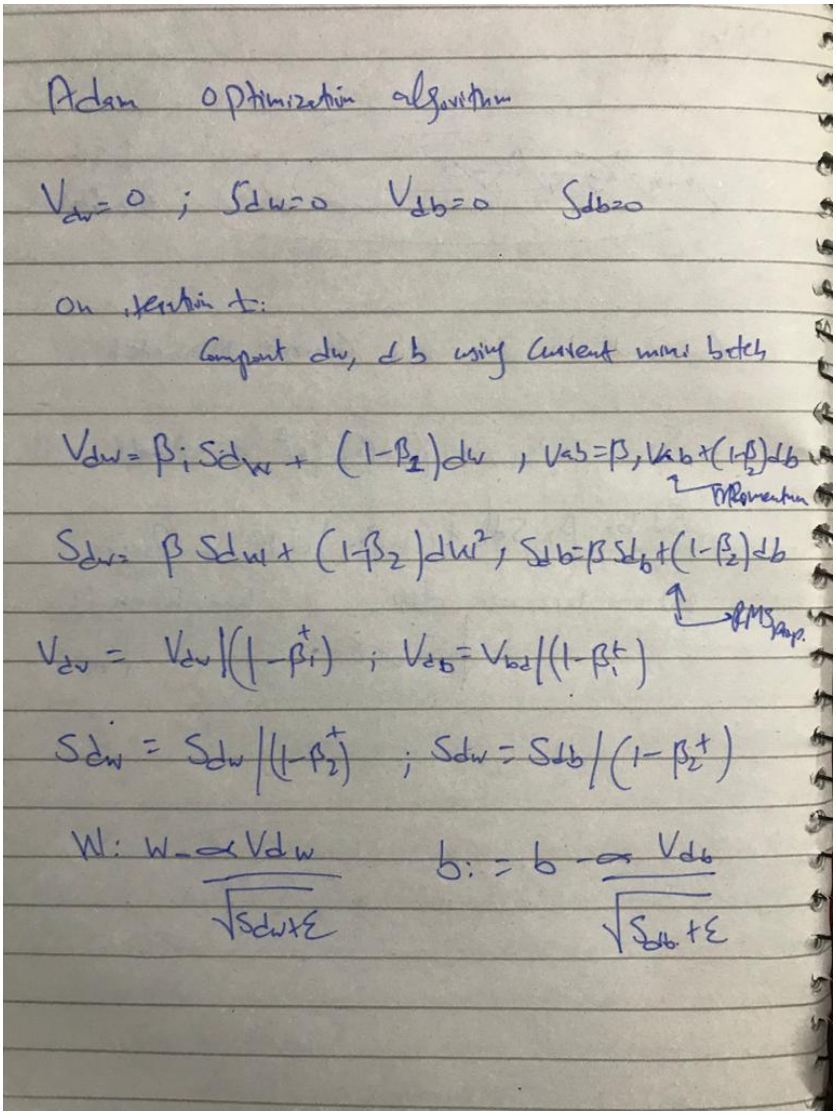
RMSprop optimizer:

Its work same as other optimation such as gradient decent but its add one functionality they add momentum basically they update his weight by its previous step:



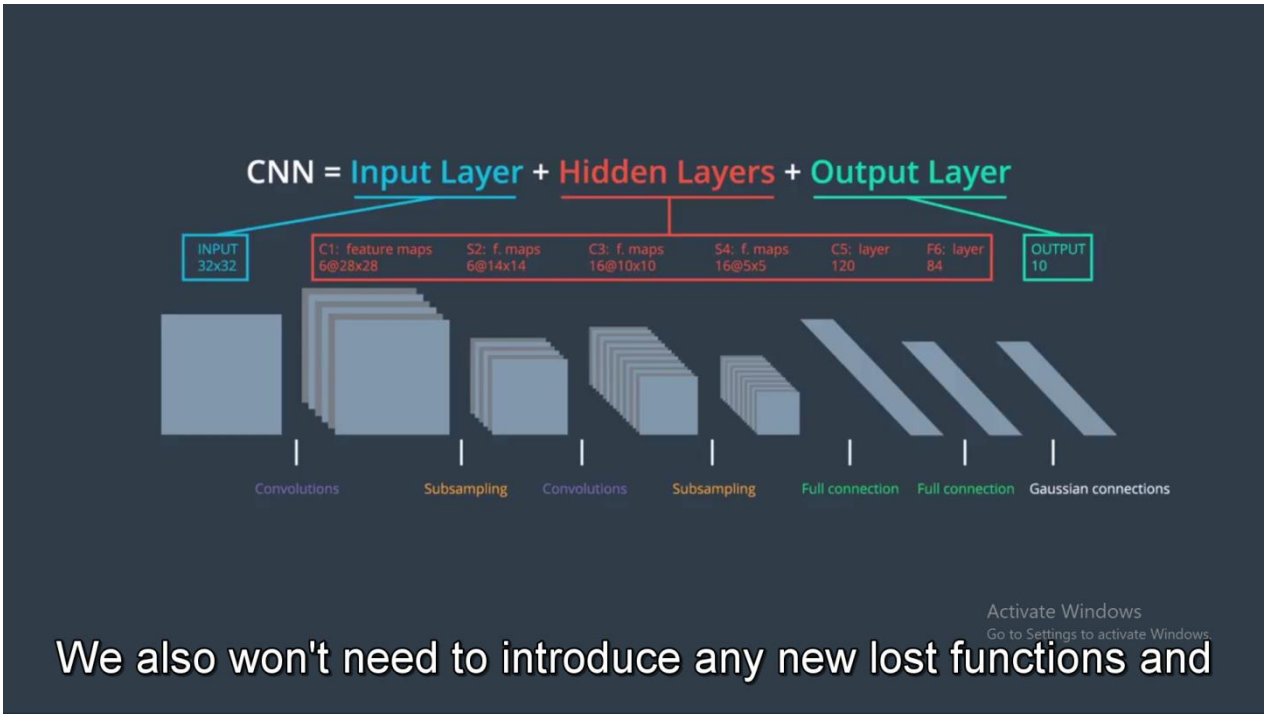
Adam optimizer: the best optimizer for find the minimum point its add all previous functionality its work as average weight of momentum +average weight of rms and find minimal nest result and most programmer this technique for find the fattest result and accurate.





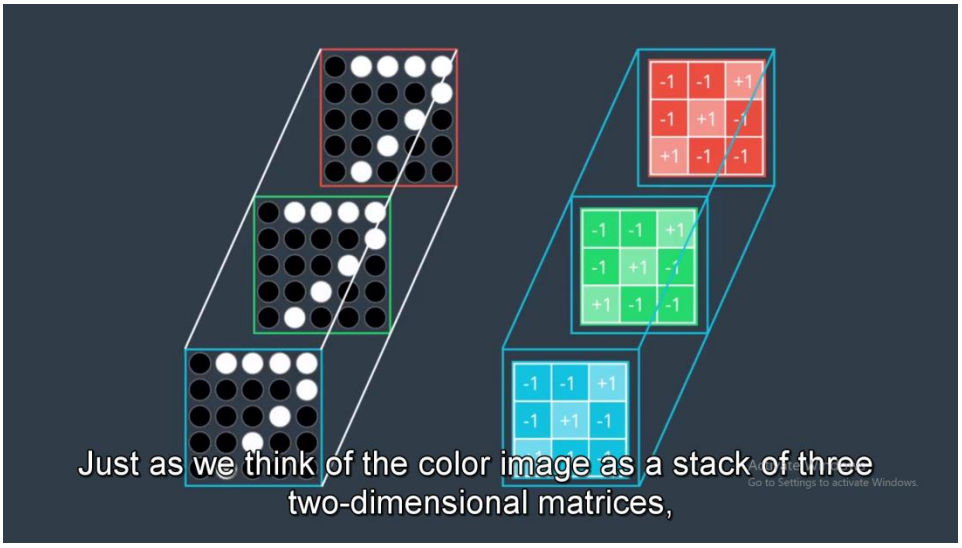
Convolution neural network :

In machine learning and CNNS have not much more differences to detect the feature and image processing technique they have used same type of activation function and loss function they also used to fit the data very well to support vector machine and use hidden Marko models and bays network and basin model these are the common things which are used in machine learning and convolution neural network basically in machine learning we have use vector in the form of input and hidden input and output in machine learning we use vectorization such as flatten layer to and dense layer for training but in CNN we also used matrices such as in the form of filter for input they are the major difference in machine learning and CNN we also used grey scale images but in CNNS we also used RGB images and GRB and also uses many type of channel input .CNNS is also based on previous terminology of input neurons and m no of hidden neurons means filter in addition these things are not used in machine learning as padding or steroid window basically they play important rule in finding feature and recognize the pic and get some prediction. In CNN we calculate millions of parameters by this technique and recognize the objects .the main layers which work in CNN is input layer ,hidden layer or filter, padding layer to detect the feature and activation layer ,flatten layer also used dense connected layer and the final is out put layer to predict the images

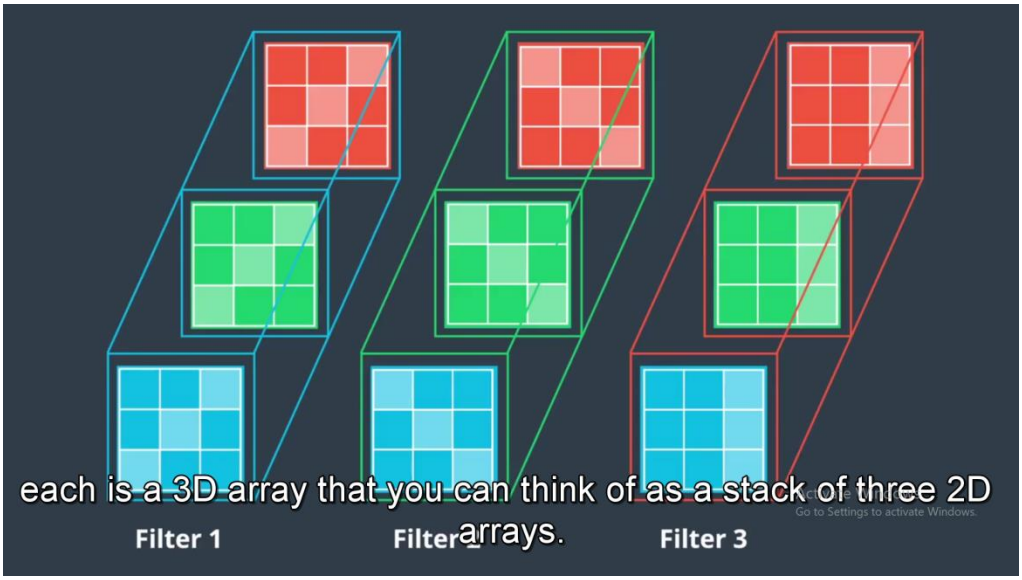


One by one I will discuss al the layers and algorithm how its work so lets start:

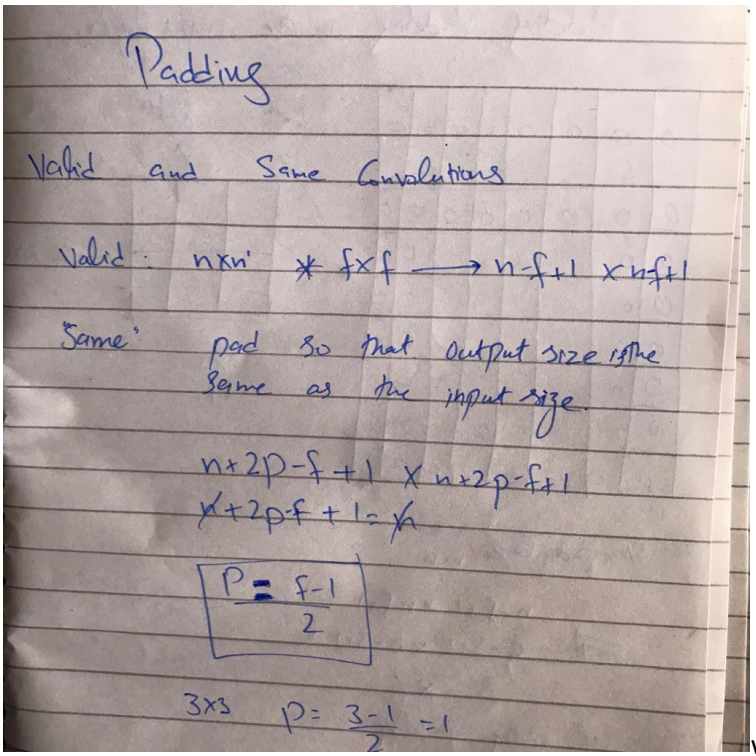
How convolution layers work:



If we use any data set example and contain input of 32by 32 means the height of input is 32 and width of input is also 32 so they means 1024 inputs are used in the form of flatten layer so we also use filter of 4 by 4 for hidden layer in the form of matrix and also contain 3 no of channels RGB no of channel which is used in convolution is same as input layer and filter layer so first 32 is represented the height and the second 32 is represent the weight of the input and 3 show the no of channels so they show three dimension due to 3 no of channel is used in this example and they contain RGB color schema the no of filter is initialize randomly if the number is zero they show black color and number is 255 the show white color or the number between 175 they show grey color



Also use many multi type of filter in the neural network basically filter is used to detect the edges extraction the feature work in the form of slides bcz they have matrix and they work in the form of slides and the input layer is multiply with filter and get the output the things which are usesd for sliding



we used padding and stride to maintain the output values.if padding is same they do not effect the final output if padding is add they increade the output matrixes

Stride Convolution

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7×7

$n \times n$

$*$

3	4	4
1	0	2
-1	0	3

3×3

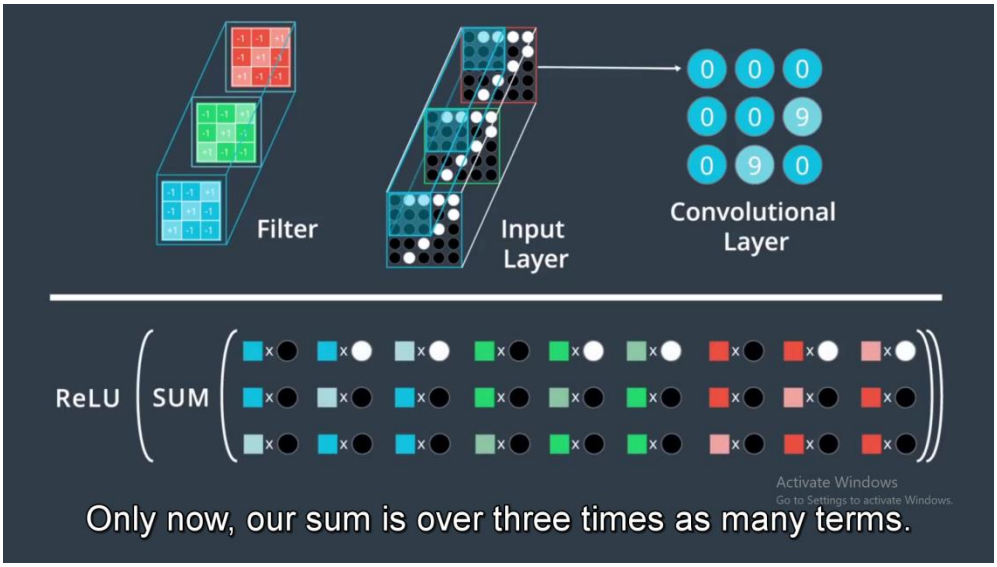
$f \times f$

Stride = 2

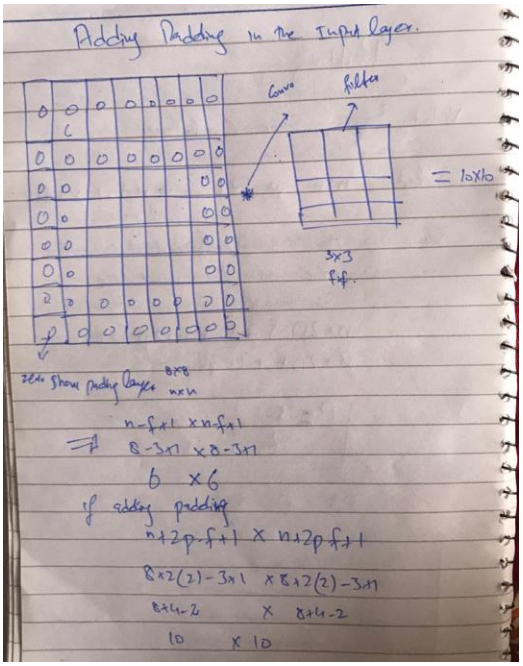
$=$

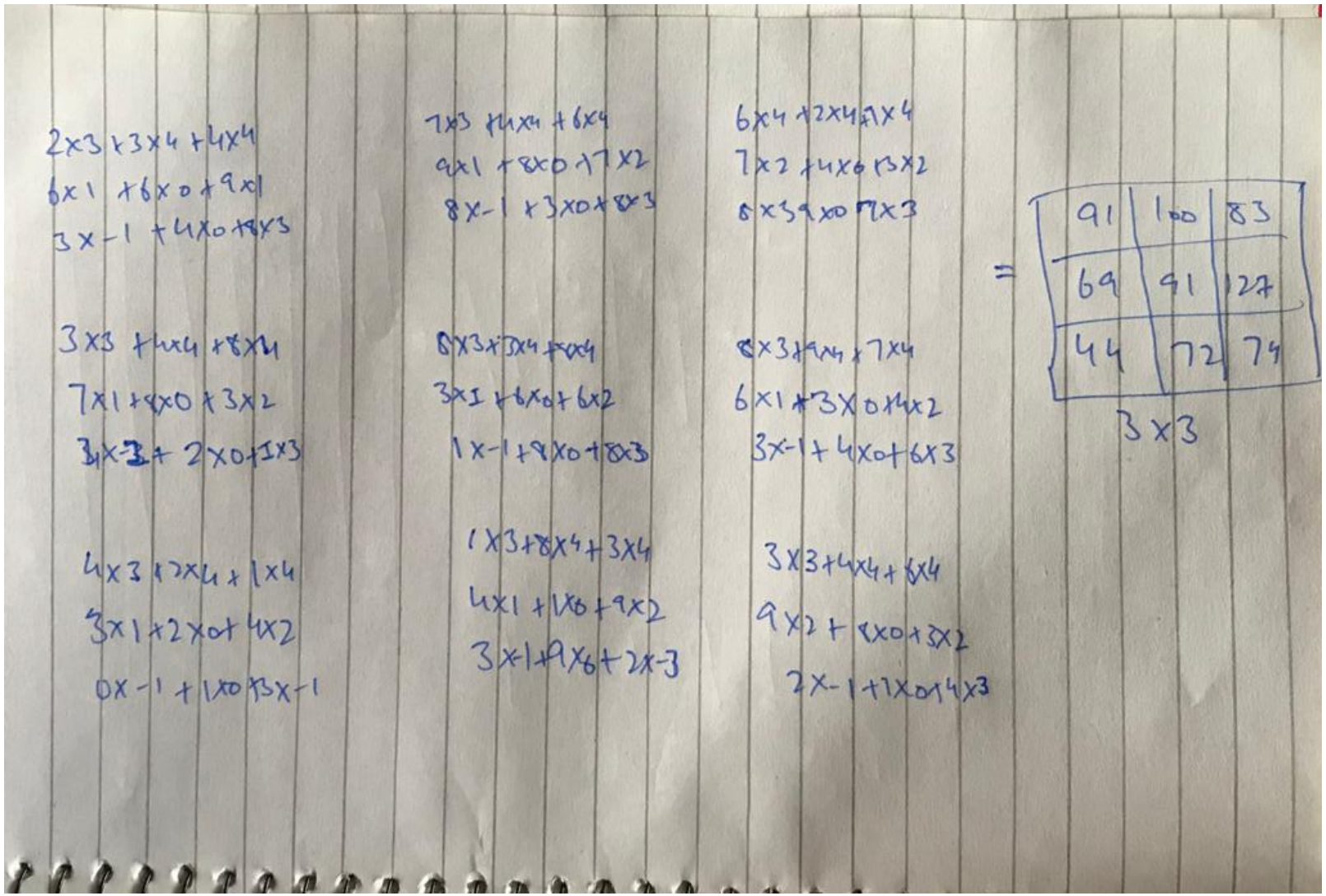
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

$$\frac{7+0-3}{2} + 1 = \frac{4}{2} + 1 = 3$$

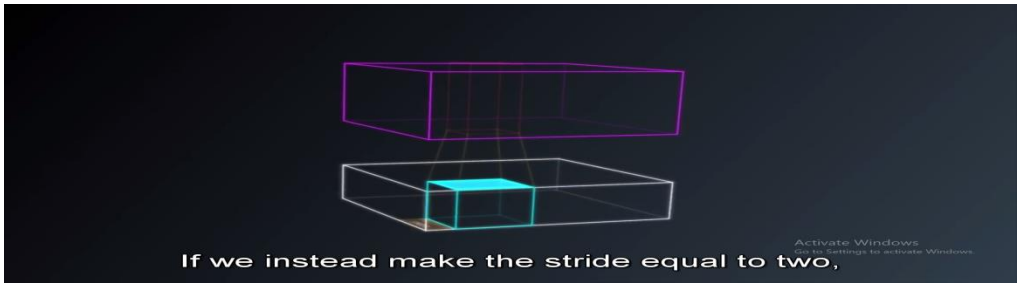


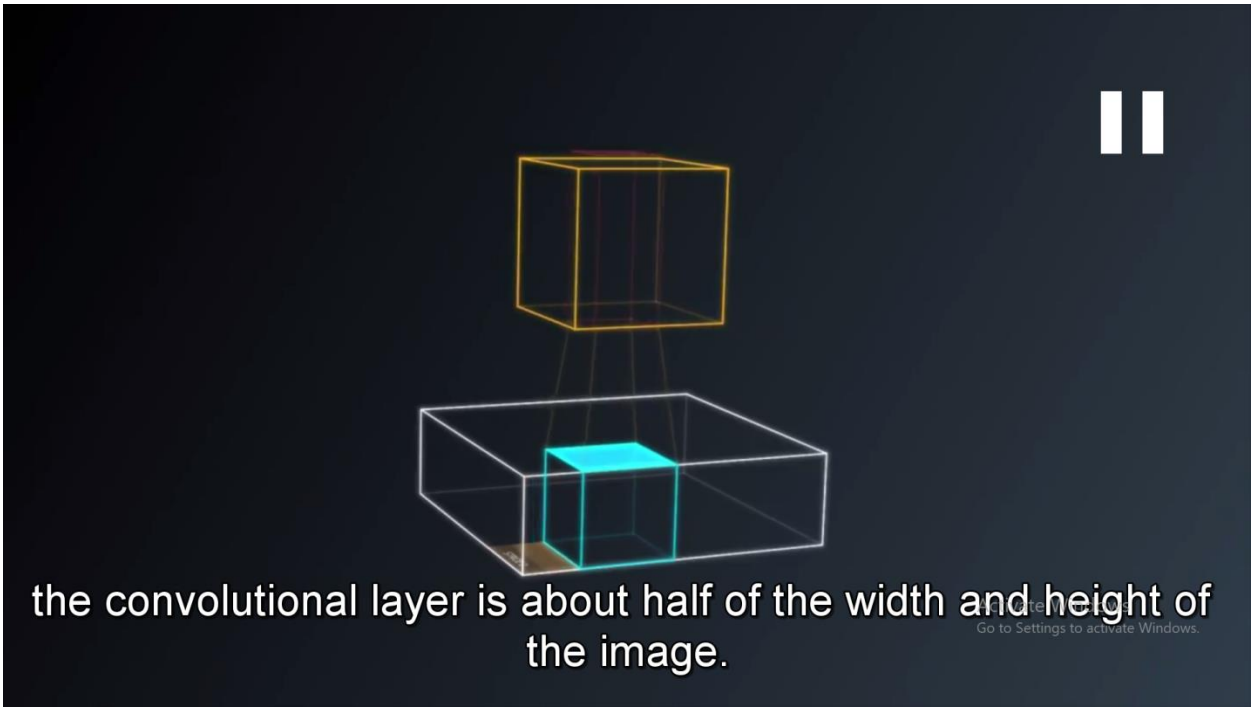
Example to implement all the things in one frame:

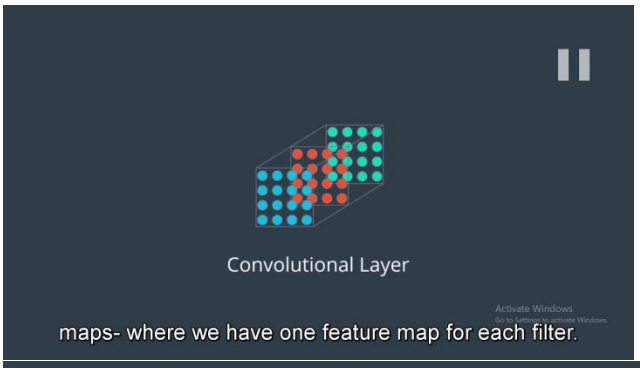


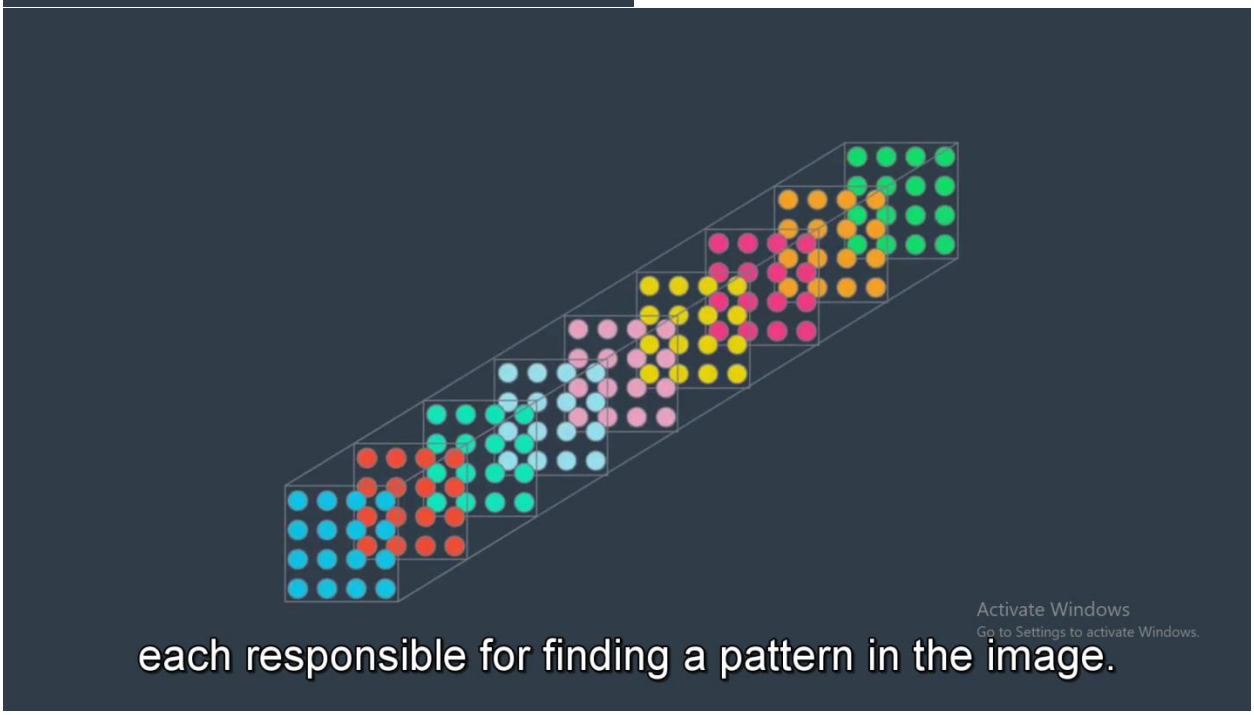

2x3 + 3x4 + 4x4
6x1 + 6x0 + 9x1
3x-1 + 4x0 + 8x3
7x3 + 4x4 + 6x4
9x1 + 8x0 + 7x2
8x-1 + 3x0 + 8x3
6x4 + 2x4 + 1x4
7x2 + 4x6 + 3x2
6x3 + 9x0 + 7x3
=
91 100 83
69 91 127
44 72 74
3x3
3x3 + 4x4 + 8x4
7x1 + 8x0 + 3x2
3x-3 + 2x0 + 1x3
8x3 + 3x4 + 4x4
3x1 + 6x0 + 6x2
1x-1 + 8x0 + 8x3
8x3 + 9x4 + 7x4
6x1 + 3x0 + 4x2
3x-1 + 4x0 + 6x3
4x3 + 2x4 + 1x4
3x1 + 2x0 + 4x2
0x-1 + 1x0 + 3x-1
1x3 + 8x4 + 3x4
4x1 + 1x0 + 9x2
3x1 + 9x6 + 2x-3
3x3 + 4x4 + 8x4
9x2 + 8x0 + 3x2
2x-1 + 7x0 + 4x3

Some time have problem due to last layer have some exception the last layer is not available due to stride method then they overcome this problem is due to padding is used to add one extra layer in the edges of our input layer they manages the layer is padding is same they do not add nay extra layer is padding is valid we use an extra layer to manage the output precisely and efficiently in Convolution neural network we also use loss function and update the filter into minimize the error and predict the output if we use padding and stride they show a bottle neck shape to read the input in the form of matrix shape.









Pooling layer: polling layer is used to find the maximum feature and average feature

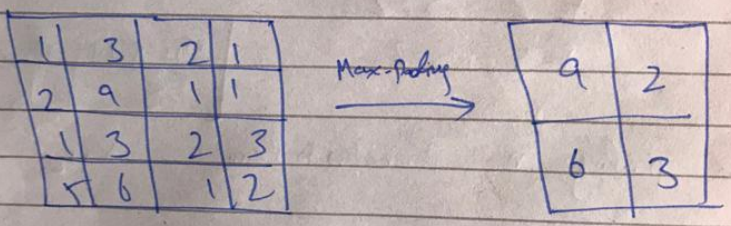
algorithm used to detect with the help of polling layers if we add large no of filter in in hidden layer they effect the depth the number of volume of the classification increase due to large no of filter and length and width decrease if we use padding and stride they get decrease the size of output dimension the convolution is about half of the width and height of the image

Implement algorithm:.

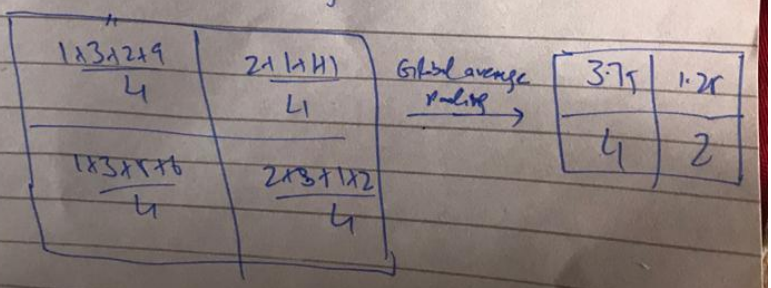
Pooling Layer

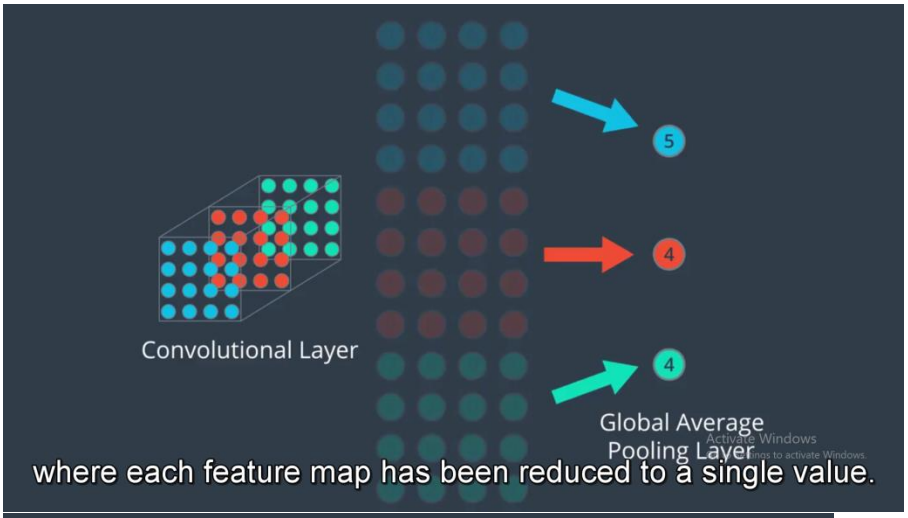
There are two type of Pooling layer

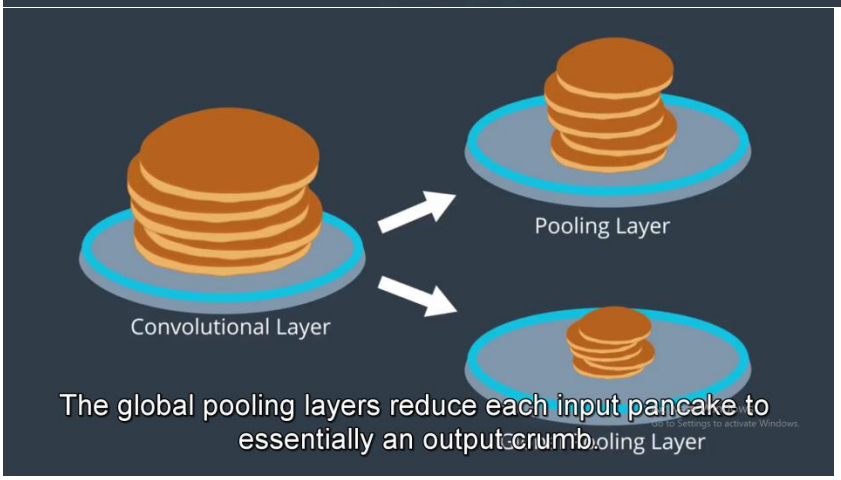
1) Max Pooling
2) Global average Pooling

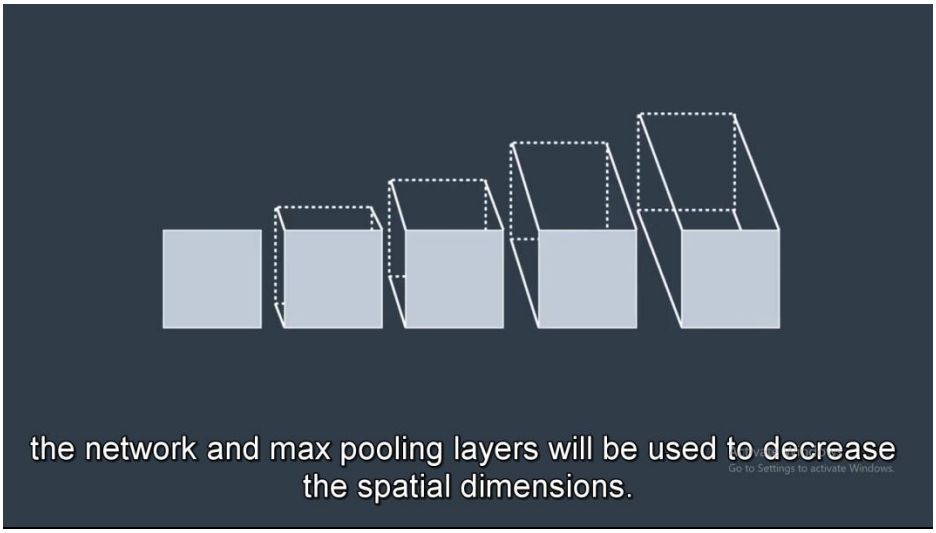


Find Average Pooling

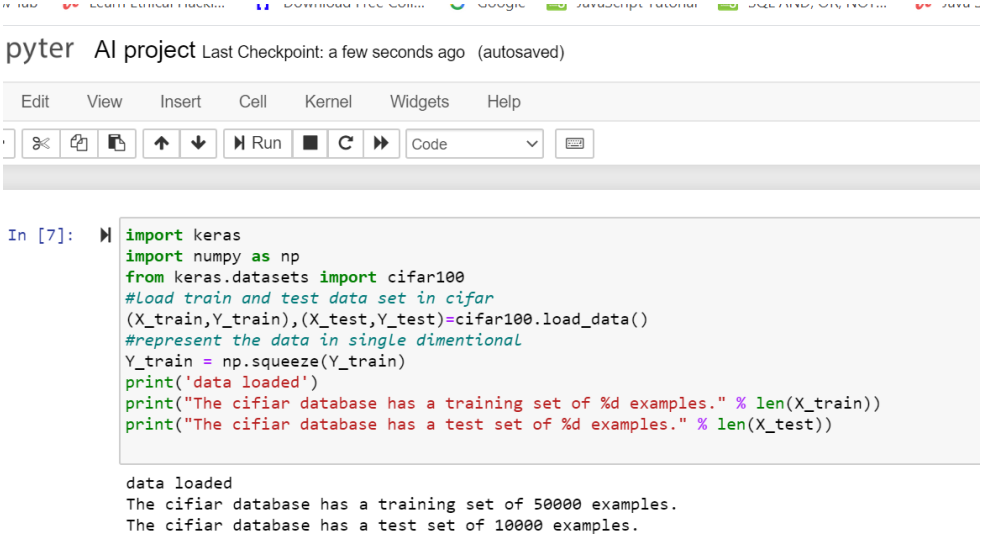


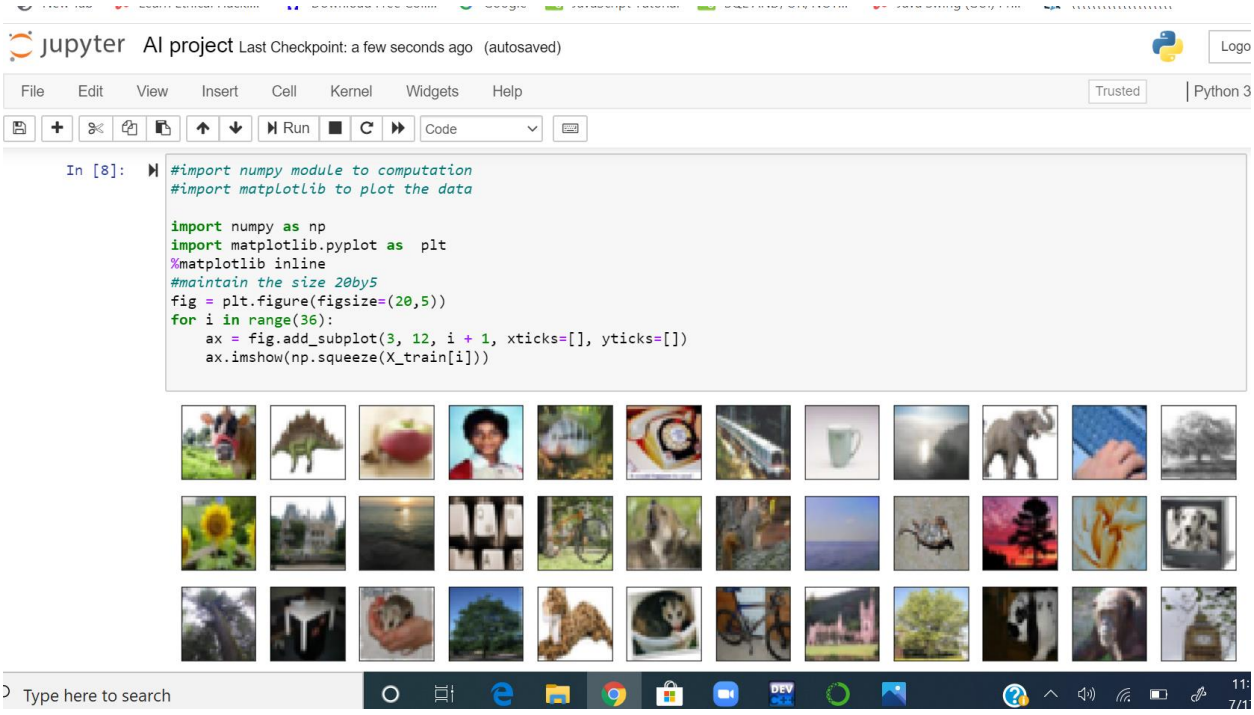

5
4
4
Convolutional Layer
Global Average
Pooling Layer
where each feature map has been reduced to a single value.


Pooling Layer
Convolutional Layer
The global pooling layers reduce each input pancake to
essentially an output crumb.


the network and max pooling layers will be used to decrease
the spatial dimensions.

This times for dirt our hand to use these technique in our coding:





```
In [9]: X_train=X_train.astype('float64')/255
        X_test =X_test.astype('float64')/255

In [47]: from keras.utils import np_utils   #import all training module

         # one-hot encode the labels
```

```
]: #impor keras model layers and libraries to implement the algo
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
#create the model
model = Sequential()
#16 inputs and 32by 32 input value and 3color channel
#max pooling layer 2by 2 matrix to find the maximum no.
#same padding
#no stride by defealt
#activation function
model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
                 input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=128, kernel_size=2, padding='same', activation='tanh'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.summary()
```

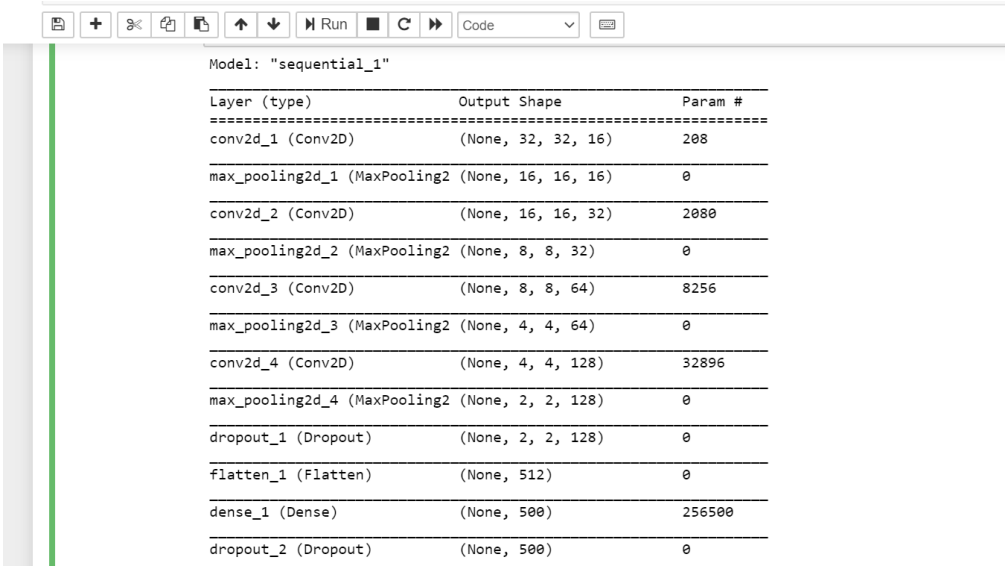


```
Model: "sequential_1"
```

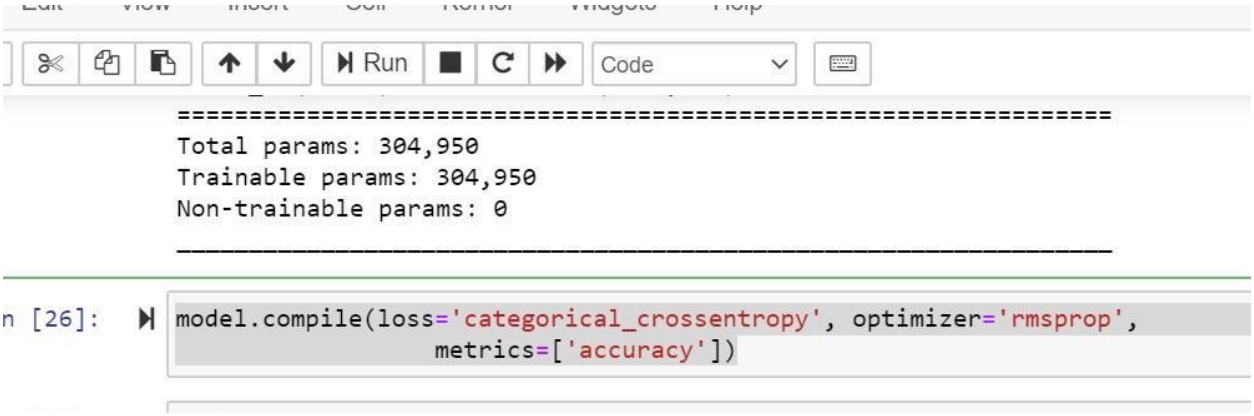

Edit View Insert Cell Kernel Widgets Help

```
In [10]: from keras.utils import np_utils   #import all training module
         # one-hot encode the labels
         num_classes = len(np.unique(Y_train)) #means use the unique values to asssighn to 1 and the others to b ze
         Y_train = keras.utils.to_categorical(Y_train, num_classes)#converted the clasees in the form of classifica
         Y_test = keras.utils.to_categorical(Y_test, num_classes)#converted the clasees in the form of classificati

         # break training set into training and validation sets
         #first the training data starts from 5000 to 50000
         #and the validation data starts from zero to 5000
         #as same  these process for Xand y
         (X_train, X_valid) = X_train[5000:], X_train[:5000]
         (Y_train, Y_valid) = Y_train[5000:], Y_train[:5000]

         # print shape of training set
         #in the form of array
         print('x_train shape:', X_train.shape)

         # print number of training, validation, and test images
         print(X_train.shape[0], 'train samples')
         print(X_test.shape[0], 'test samples')
         print(X_valid.shape[0], 'validation samples')
```

```
x_train shape: (45000, 32, 32, 3)
45000 train samples
10000 test samples
```



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 128)         32896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               256500
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
```



```
=================================================================
Total params: 304,950
Trainable params: 304,950
Non-trainable params: 0
_________________________________________________________________
```

n [26]:

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
                    metrics=['accuracy'])
```

```
import keras
from keras.datasets import cifar100
(X_train, Y_train), (X_test, Y_test) = cifar100.load_data()
print("The cifar database has a training set of %d examples." % len(X_train))
print("The cifar database has a test set of %d examples." % len(X_test))
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


fig = plt.figure(figsize=(30,10))
for i in range(36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(X_train[i]))
# rescale [0,255] --> [0,1]
X_train = X_train.astype('float32')/255
Y_test = X_test.astype('float32')/255
from keras.utils import np_utils   #import all training module


# one-hot encode the labels
```

num_classes = len(np.unique(Y_train)) #means use the unique values to asssighn to 1 and the others to b zero

Y_train = keras.utils.to_categorical(Y_train, num_classes)#converted the clasees in the form of classification for traimg the data

Y_test = keras.utils.to_categorical(Y_test, num_classes)#converted the clasees in the form of classification for traimg the data

break training set into training and validation sets

#first the training data starts from 5000 to 50000

#and the validation data starts from zero to 5000

#as same these process for Xand y

(X_train, X_valid) = X_train[5000:], X_train[:5000]

(Y_train, Y_valid) = Y_train[5000:], Y_train[:5000]

print shape of training set

#in the form of array

print('x_train shape:', X_train.shape)

print number of training, validation, and test images

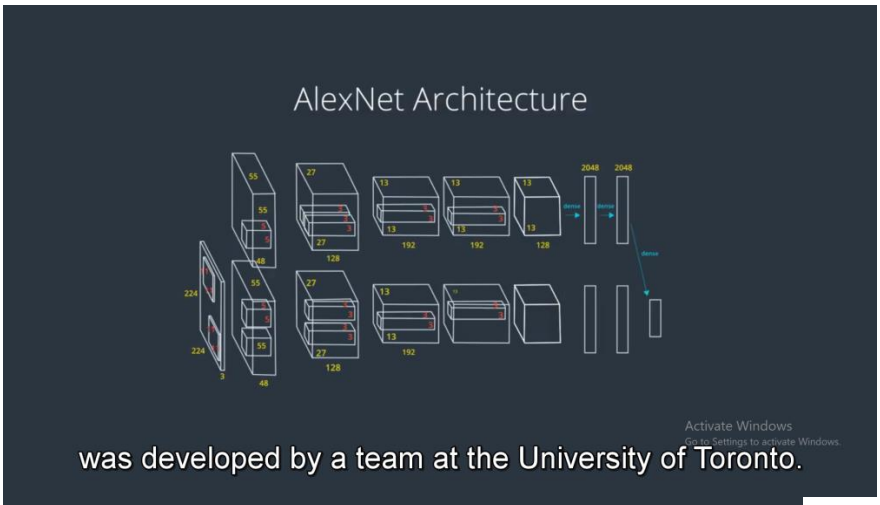
print(X_train.shape[0], 'train samples')

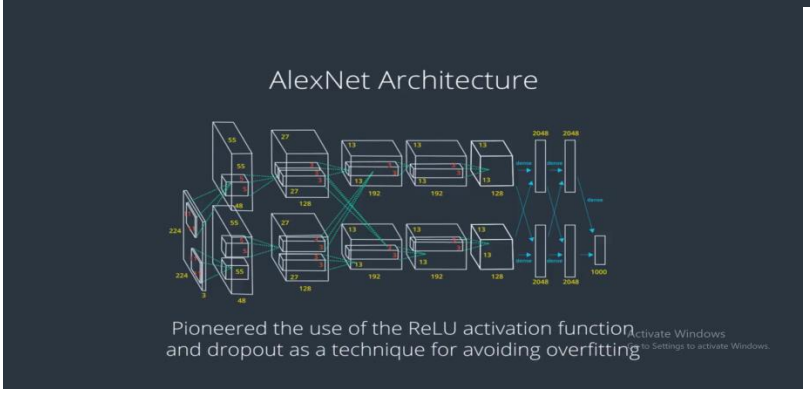
print(X_test.shape[0], 'test samples')

```
print(X_valid.shape[0], 'validation samples')

#impor keras model layers and libraries to implement the algo

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

#create the model

model = Sequential()

#16 inputs and 32by 32 input value and 3color channel

#max pooling layer 2by 2 matrix to find the maximum no.

#same padding

#no stride by defealt

#activation function

model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',

                 input_shape=(32, 32, 3)))

model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=128, kernel_size=2, padding='same', activation='tanh'))

model.add(MaxPooling2D(pool_size=2))
```

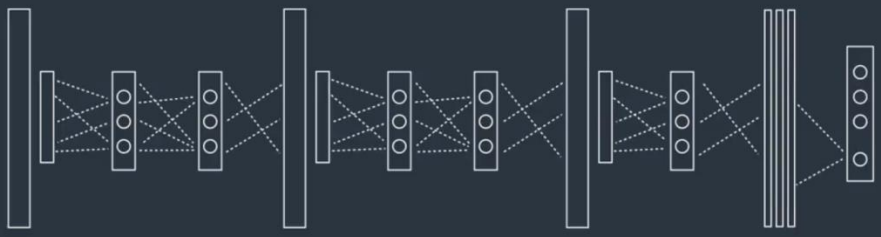
```
model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(500, activation='relu'))

model.add(Dropout(0.4))

model.add(Dense(10, activation='softmax'))



model.summary()

model.compile(loss='categorical_crossentropy', optimizer='rmsprop',

              metrics=['accuracy'])

from keras.callbacks import ModelCheckpoint



# train the model

checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=1,

                       save_best_only=True)

hist = model.fit(X_train, Y_train, batch_size=2, epochs=10,

        validation_data=(X_valid, Y_valid), callbacks=[checkpointer],

         verbose=2,shuffle=True)
```

important CNNS model:





VGG Architecture

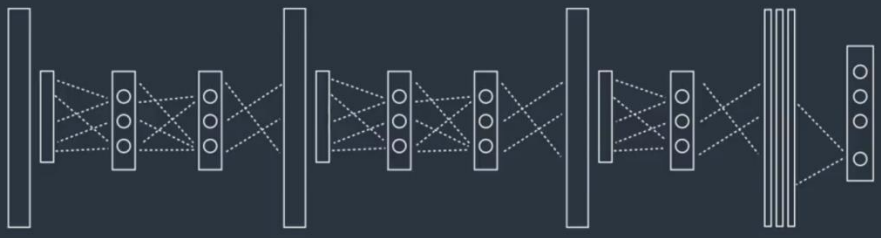


One of those networks was called VGG Net,

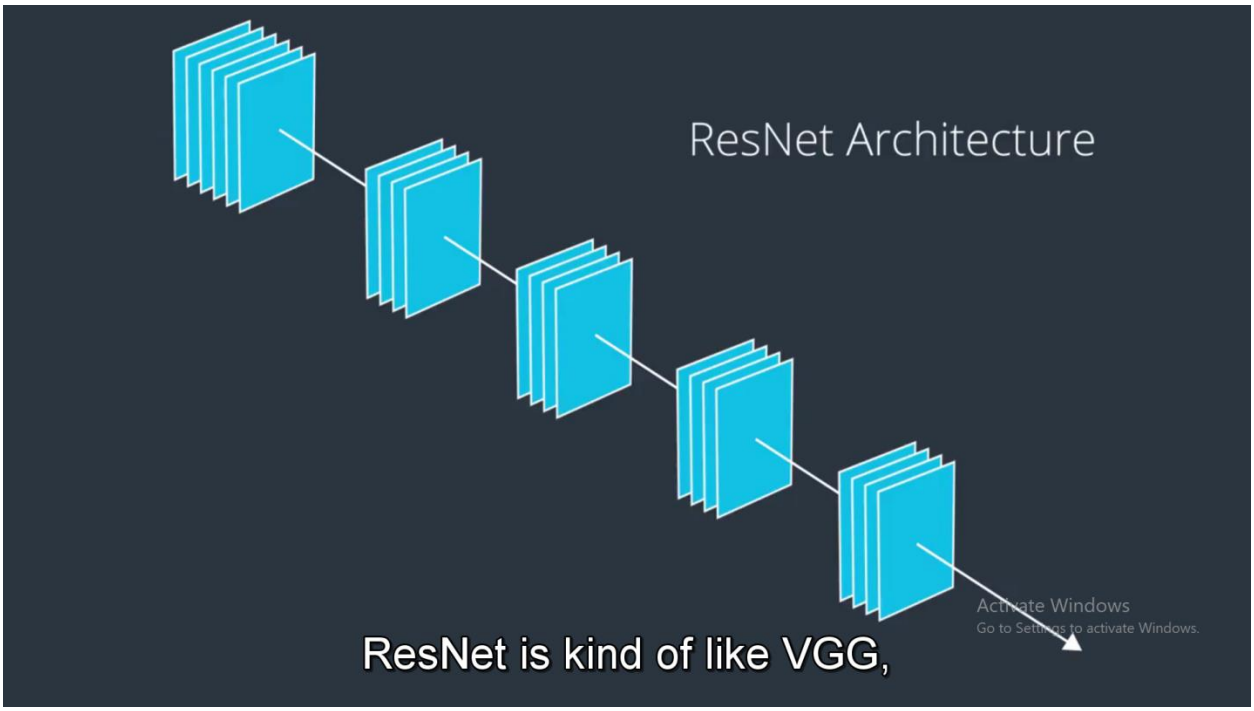
VGG Architecture

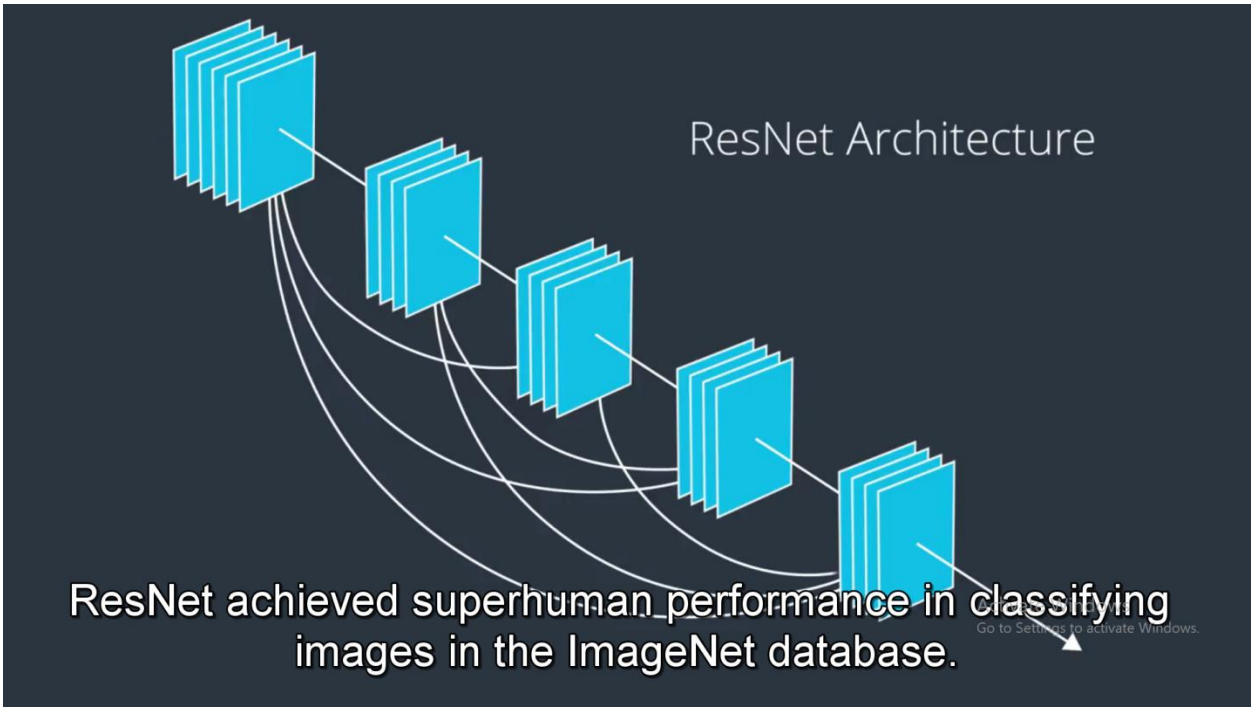
VGG 16

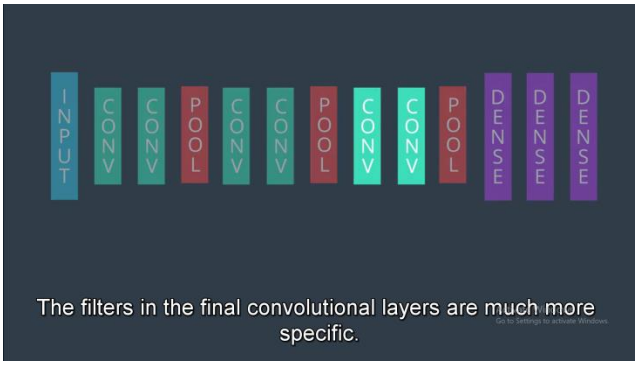
VGG 19

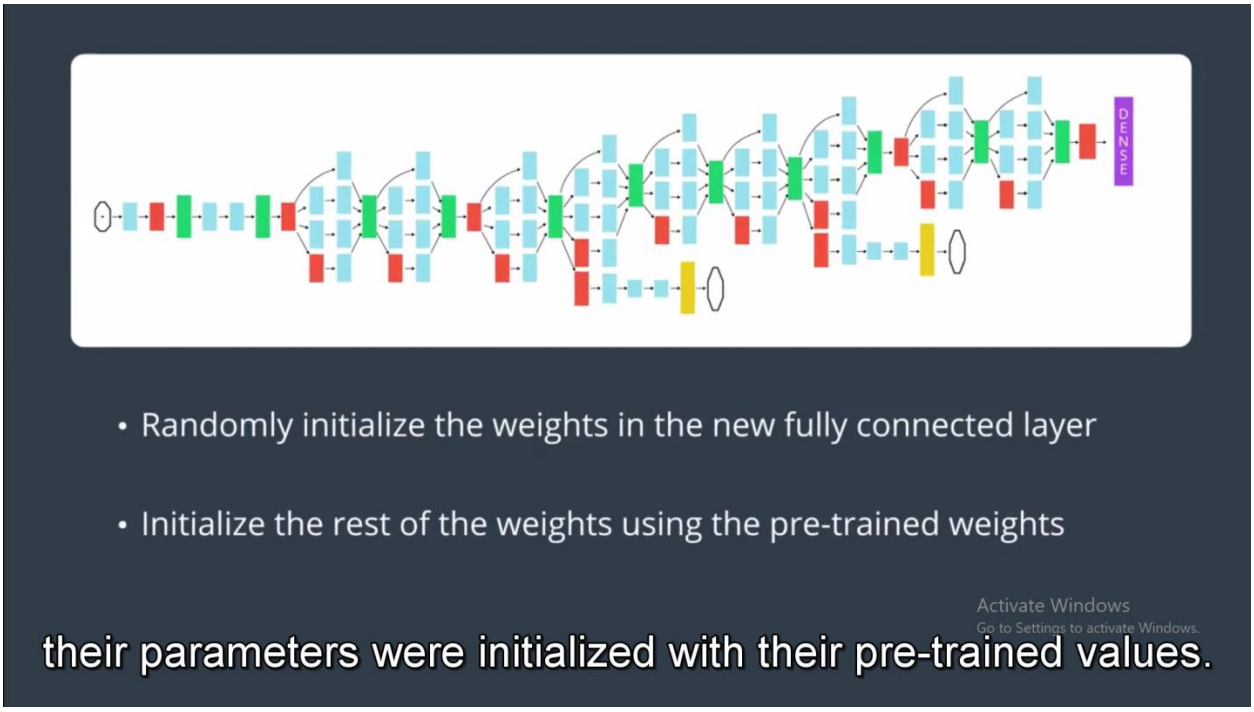


with 16 and 19 total layers, respectively.


ResNet Architecture
ResNet is kind of like VGG,
Activate Windows
Go to Settings to activate Windows.


ResNet Architecture
ResNet achieved superhuman performance in classifying
images in the ImageNet database.
Go to Settings to activate Windows.

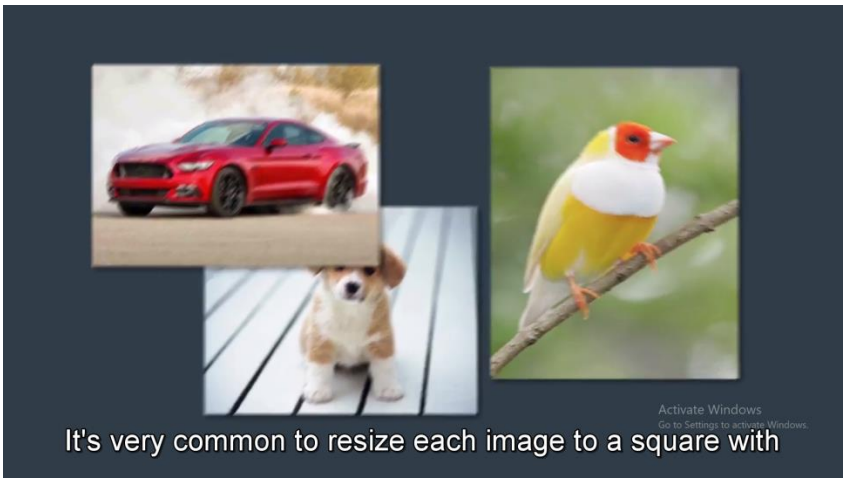




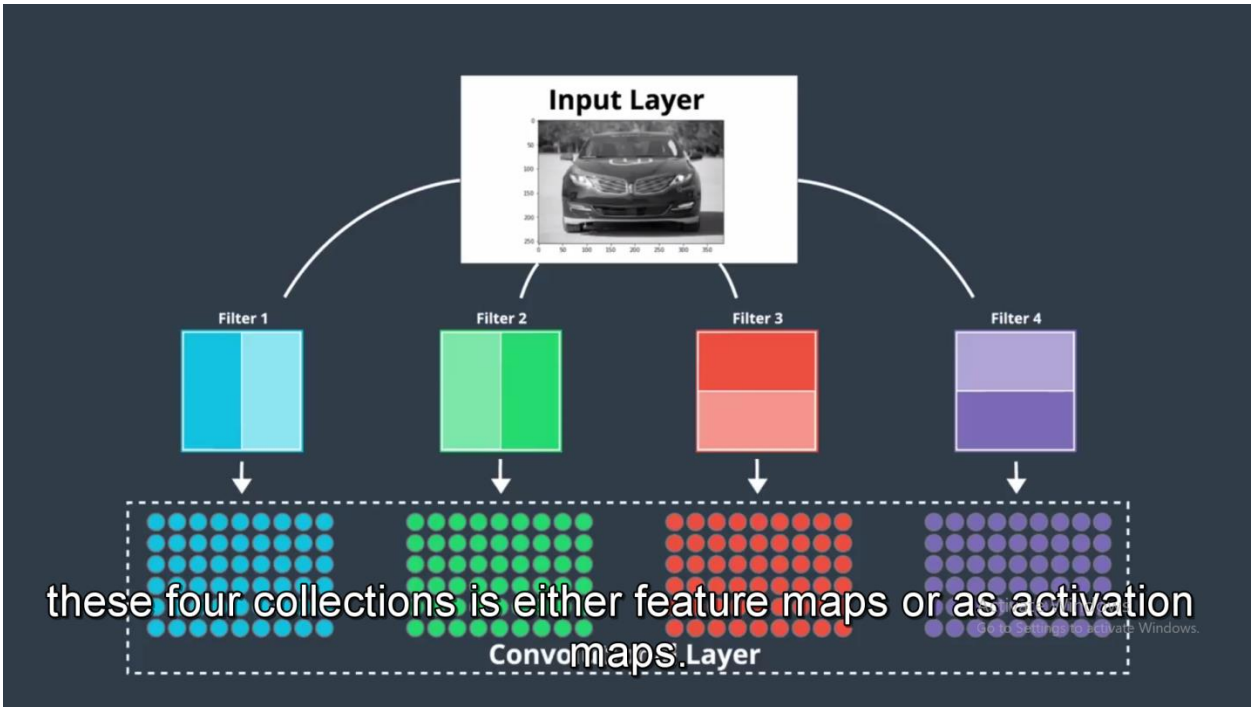
Feature detection:

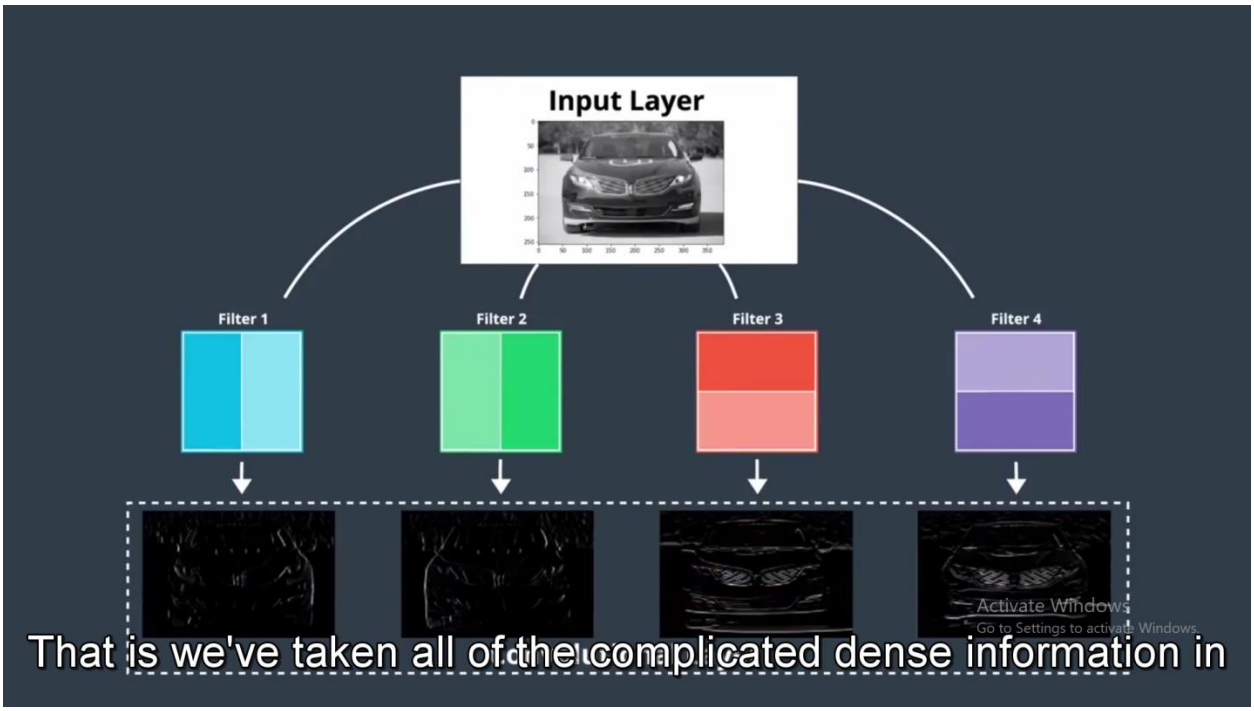
1)by vertical edges

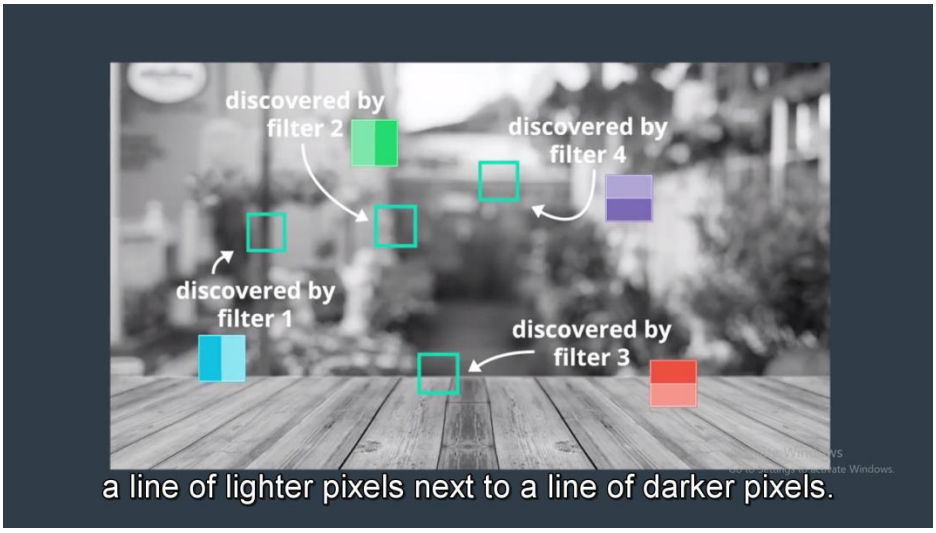
2)by horizontal edges



Convert these images in to gray scale images to detect the edges and feature of the images the high intensity show more clearly images so they changed RGB images in to gray scale images and used the filter layer and set the weight to read the vertical or horizontal images we updated the weight some time to get more clear edges and vice versa and these vector store in to sparse type array connected in to flatten layer and then converted in to dense layer .







The filter in which the value is grater then zero they show white type spacing in the gray scale images and the black images they have low intensity that why they not show any detection and the Sobel filter is very commonly used in edge detection and in finding patterns in intensity in an image. Applying a Sobel filter to an image is a way of **taking (an approximation) of the derivative of the image** in the xx or yy direction. The operators for Sobel_x$Sobel_x$ and Sobel_y$Sobel_y$, respectively, look like this:

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Noise speckle or discoloration in an images by pass low pass filter and blur or smooth the images by saturation and hue effects

Canny Edge detection: filter out using a Gaussian blur images the find the strength and direction of edge using sobel filter, applies non-maximum suppression to isolate the strongest edge and then to one pixel wide line use hysteresis thrash holding value to isolate the best edges,

By using these technique apply we use computer vision libraries to find the edges, intensity of light for different effects they are basically prominent the sharp edges and feature of the detection:

Find edges detection with different filters:

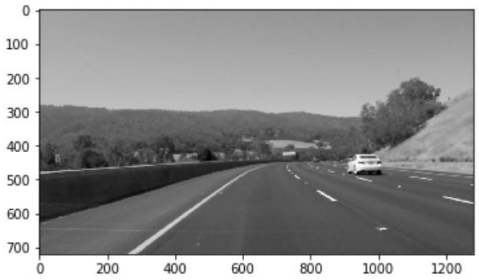
```
]:  import matplotlib.pyplot as plt
    import matplotlib.image as mpimg

    import cv2
    import numpy as np

    %matplotlib inline

    # Read in the image
    image = mpimg.imread('images/curved_lane.jpg')
```

Out[31]: <matplotlib.image.AxesImage at 0x213803e52c8>

Run Code

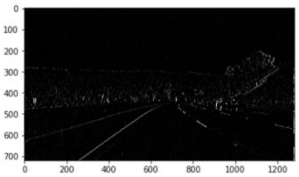
In [32]:
```
#Convert to grayscale for filtering
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)#convert the colored channel ir

plt.imshow(gray, cmap='gray')
```

Out[32]: <matplotlib.image.AxesImage at 0x213806fe108>



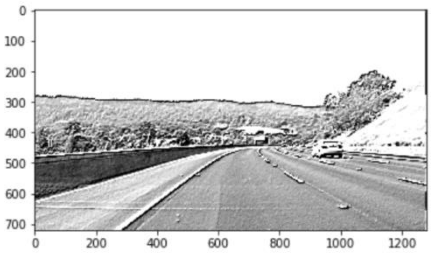
```
In [33]:
# Create a custom kernel

# 3x3 array for edge detection
sobel_x = np.array([[ -1, 0, 1],
                    [ -2, 0, 2],     #appl sobel x filter to find detection of vertically
                    [ -1, 0, 1]])


# Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel)
filtered_image = cv2.filter2D(gray, -12, sobel_y)
plt.imshow(filtered_image, cmap='gray')
```

```
Out[33]: <matplotlib.figure.AxesImage at 0x2138075fb48>
```




```
# 5x5 array for edge detection
sobel_y = np.array([[ -1, -2, -1,0,-3],
                    [ 0, 0, 0,0,1],
                    [ 1, 2, 1,0,0],
                    [-3,-1,0,-1,3],
                    [-1,0,1,4,2]])

# Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel)
filtered_image = cv2.filter2D(gray, -1, sobel_y)

plt.imshow(filtered_image, cmap='gray')
```

```
Out[34]: <matplotlib.figure.AxesImage at 0x213807caac8>
```



One older (from around 2001), but still popular scheme for face detection is a Haar cascade classifi based classification cascades that learn to isolate and detect faces in an image. You can read the c

Let's see how face detection works on an exampe in this notebook.

```
In [35]:
# import required libraries for this section
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
In [43]:
# load in color image for face detection
image = cv2.imread('images/usama.jpeg')

# convert to RBG
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)#convert the brg chanel in to

plt.figure(figsize=(10,10)) #the height and weight is 10by10
plt.imshow(image)
```

```
Out[43]: <matplotlib.figure.AxesImage at 0x261003ff5c8>
```

```
In [51]:
# Load in color image for face detection
image = cv2.imread('images/m.jfif')

# convert to RBG
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)#convert the brg chanel in to rgb

plt.figure(figsize=(10,10)) #the height and weight is 10by10
plt.imshow(image)
```

```
Out[51]: <matplotlib.image.AxesImage at 0x261001a5788>
```

Type here to search

```
In [52]:
# convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  #convert the color channel in to

plt.figure(figsize=(20,10))
plt.imshow(gray, cmap='gray')#map the image in to gray
```

```
Out[52]: <matplotlib.image.AxesImage at 0x261009f5cc8>
```

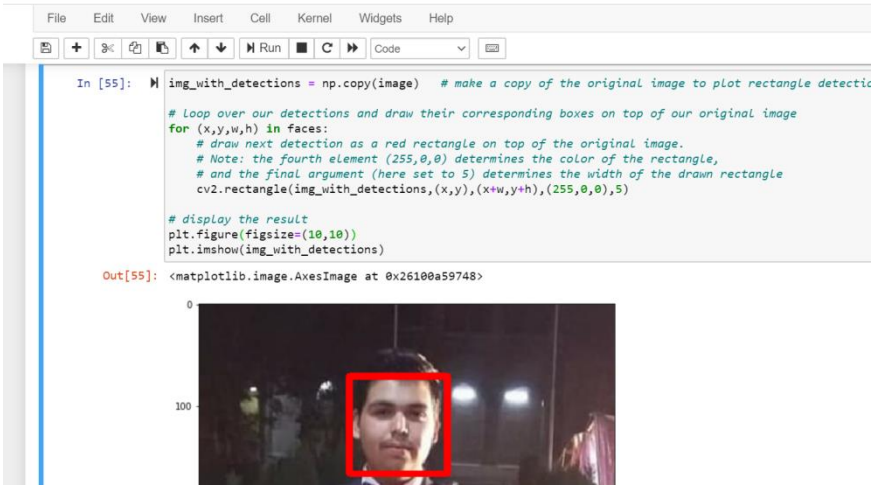
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

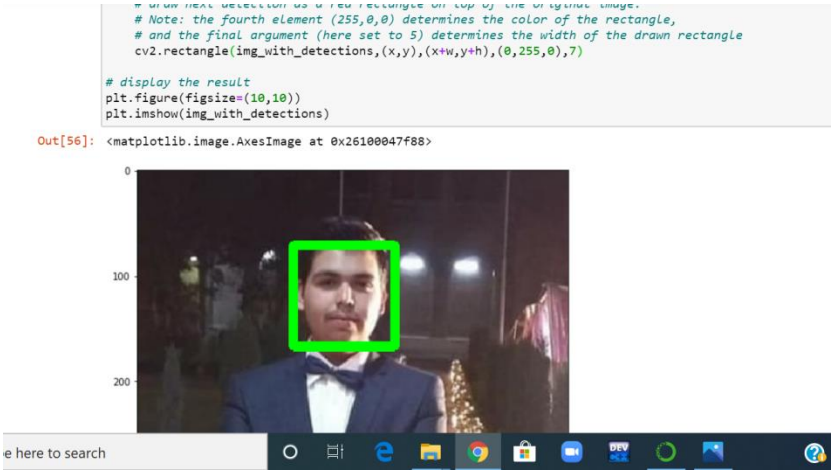
```
In [53]:
# Load in cascade classifier
face_cascade = cv2.CascadeClassifier('detector_architectures/haarcascade_frontalface_default.xml')

# run the detector on the grayscale image
faces = face_cascade.detectMultiScale(gray, 4, 6)#detectmultiscale is function to detect the images with the help of PCAs val
```

```
4]:  # print out the detections found
     print ('the algorithm has to detect  ' + str(le
     print ("Their coordinates and lengths/widths ar
     print ('==============================')
     print (faces)
```

```
the algorithm has to detect  1 faces in this im
Their coordinates and lengths/widths are as fol
==============================
[[148  71  96  96]]
```

Let's plot the corresponding detection boxes on our original ima

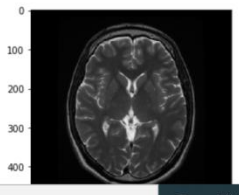




By Gaussian blur detection:

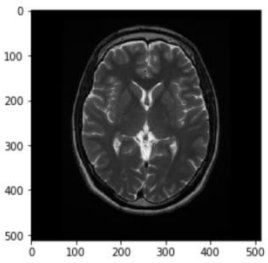
```
In [1]:
import numpy as np
import matplotlib.pyplot as plt
import cv2

%matplotlib inline

# Read in the image
image = cv2.imread('images/brain_MR.jpg')

# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
```

```
Out[1]: <matplotlib.figure.AxesImage at 0x20318124bc8>
```




```
In [3]:
# Convert the image to grayscale for processing
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

plt.imshow(gray, cmap='gray')
```

```
Out[3]: <matplotlib.figure.AxesImage at 0x2031a6949c8>
```



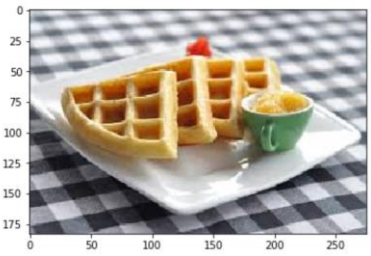
Harris corner Detection:

1]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2

%matplotlib inline

# Read in the image
image = cv2.imread('images/waffle.jpg')

# Make a copy of the image
image_copy = np.copy(image)

# Change color to RGB (from BGR)
image_copy = cv2.cvtColor(image_copy, cv2.COLOR_BGR2RGB)

plt.imshow(image_copy)
```

Out[1]: <matplotlib.image.AxesImage at 0x21d2bf9e788>

Run Code

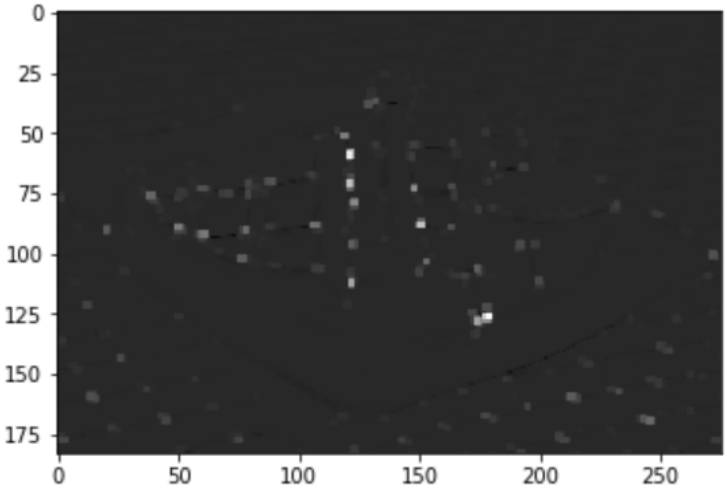
Out[1]: <matplotlib.image.AxesImage at 0x21d2bf9e788>



In [2]:
```
# Convert to grayscale
gray = cv2.cvtColor(image_copy, cv2.COLOR_RGB2GRAY)
gray = np.float32(gray)

# Detect corners
dst = cv2.cornerHarris(gray, 2, 3, 0.04)

# Dilate corner image to enhance corner points
dst = cv2.dilate(dst,None)

plt.imshow(dst, cmap='gray')
```

Out[2]: <matplotlib.image.AxesImage at 0x21d3f255808>

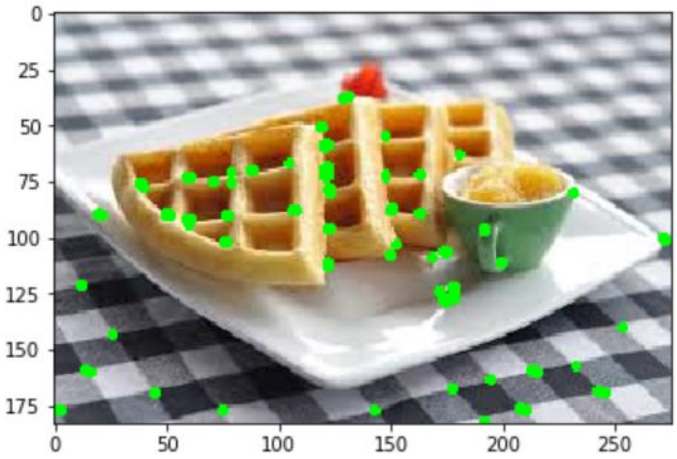
Out[2]: <matplotlib.image.AxesImage at 0x21d3f255808>



```
## TODO: Define a threshold for extracting strong corners
# This value vary depending on the image and how many corners you want to de
# Try changing this free parameter, 0.1, to be larger or smaller ans see wha
thresh = 0.1*dst.max()

# Create an image copy to draw corners on
corner_image = np.copy(image_copy)

# Iterate through all the corners and draw them on the image (if they pass t
for j in range(0, dst.shape[0]):
    for i in range(0, dst.shape[1]):
        if(dst[j,i] > thresh):
            # image, center pt, radius, color, thickness
            cv2.circle( corner_image, (i, j), 1, (0,255,0), 1)

plt.imshow(corner_image)
```

3]: <matplotlib.image.AxesImage at 0x21d3f2c8cc8>

Out[3]: <matplotlib.image.AxesImage at 0x21d3f2c8cc8>

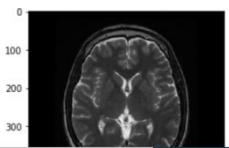


In [ ]:

Canny Edge Detection:

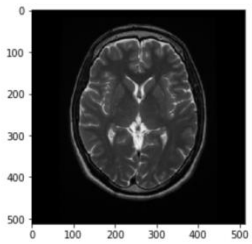
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

%matplotlib inline

# Read in the image
image = cv2.imread('images/brain_MR.jpg')

# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
```

Out[1]: <matplotlib.image.AxesImage at 0x20318124bc8>



Type here to search

```
In [3]: # Convert the image to grayscale for processing
        gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

        plt.imshow(gray, cmap='gray')

Out[3]: <matplotlib.image.AxesImage at 0x2031a6949c8>
```



jupyter 5. Canny Edge Detection Last Checkpoint: 12 hours ago (autosaved)

Edit View Insert Cell Kernel Widgets Help

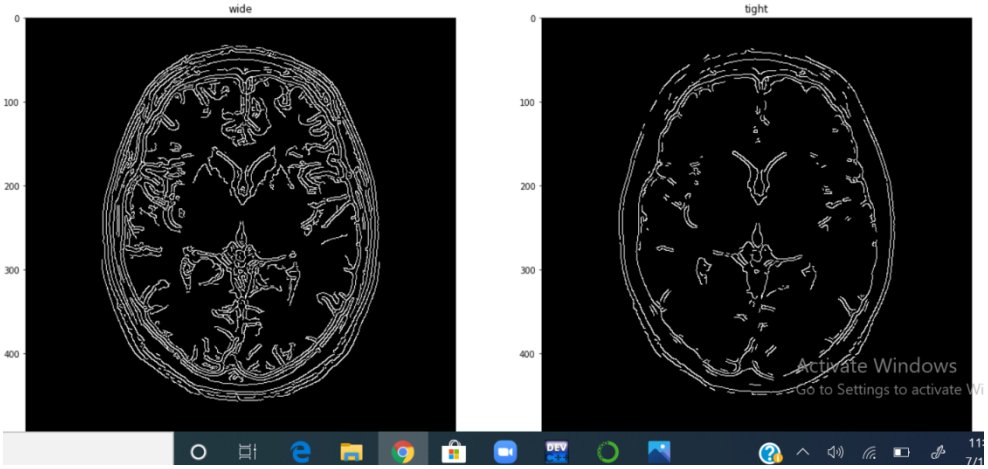
Run Code

Implement Canny edge detection

```
In [4]: # Try Canny using "wide" and "tight" thresholds

        wide = cv2.Canny(gray, 30, 100)
        tight = cv2.Canny(gray, 200, 240)


        # Display the images
        f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))

        ax1.set_title('wide')
        ax1.imshow(wide, cmap='gray')

        ax2.set_title('tight')
        ax2.imshow(tight, cmap='gray')
```

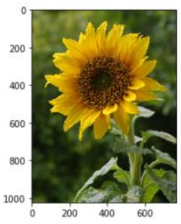
Run Code

matplotlib.image.AxesImage at 0x2031a6e3b88>



In [5]:
```
# Read in the image
image = cv2.imread('images/sunflower.jpg')

# Change color to RGB (from BGR)""
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
```

Out[5]: <matplotlib.figure.AxesImage at 0x2031bba2c48>



here to search

Out[6]: <matplotlib.figure.AxesImage at 0x2031aa40748>