COMPUTER VISION WITH OPENCV

# Course Structure

|  |  |
| --- | --- |
| **TITLE** | WEEK |
| Introduction | 1 |
| Image processing with opencv | 1 and 2 |
| Optical Character Recognition using Tesseract (introduction) | 2 |
| Convolutional neural networks using Tensorflow | 3 |
| Transfer Learning | 4 |
| YOLOv3 | 5 |
| Generative Adversarial Networks | 6 |

# Introduction

## What is computer vision?

Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. "Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding.

## What is image processing?

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image.

## Difference between computer vision and image processing

Image processing is a subset of computer vision. A computer vision system uses the image processing algorithms to try and perform emulation of vision at human scale. For example, if the goal is to enhance the image for later use, then this may be called image processing. And if the goal is to recognise objects, defect for automatic driving, then it can be called computer vision.

# Image processing with Opencv

## What is Opencv?

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

## Getting started with images and videos

Open 1.py

## Drawing in images and adding track bar

Open 2.py

## Image Addition and bitwise operations

Bitwise operations are AND, OR, NOT and XOR.

Open 3.py

## Changing colorspaces and tracking of specific color

Many color spaces available. Most common are BGR, HSV, Gray.

The syntax is cv2.cvtColor(input image, flag) where flag can be like cv2.COLOR\_BGR2GRAY.

Object tracking via color is generally done in HSV because it easier to represent a color in HSV than RGB. Use cv2.inRange(image, lower value, higher value). This returns a mask which has true for all the values that lay in that range and using cv2.bitwise\_and with mask that color can be extracted.

For more detail <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces>

WAP to make a track bar with HSV lower and higher values to see different color combinations obtained in a range using cv2.inRange – 4.py

Tip – While reading image if we write cv2.imread(‘image\_name’, 0) then it reads the image directly in grayscale.

## Geometric transformation of images

Geometrical transformations include

* scaling (resize)
* translation - shifting objects location
* rotation
* affine transform - used to correct for geometric distortions or deformations that occur with non-ideal camera angles
* perspective transform – It deals with the conversion of 3d world into 2d image.

Open 5.py

Here we saw that while rotation the image was cut. We will write a program to rotate an image such that it is not cut after learning about contours.

## Thresholding

In **simple thresholding** if a pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black) i.e. it converts the image to a binary image. Pixel values go from 0 to 255. For thresholding an **image must be in grayscale.**

ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

Where img is image in grayscale, second value is the threshold value, third is the pixel values assigned to pixel that are greater than threshold value and last is the style of thresholding.

In simple thresholding, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for **adaptive thresholding**. In this, the algorithm calculates the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

thresh = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, \

cv2.THRESH\_BINARY, 11, 2)

Where img is image in grayscale, second argument is pixel values assigned to values greater than threshold, then it is adaptive thresholding method followed by style of thresholding and lastly the block size.

If an image is bimodal (bimodal image is an image whose histogram has two peaks) **Otsu’s thresholding** is useful. It automatically calculates the threshold value from the image histogram.

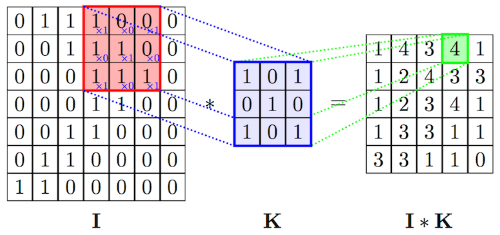
ret, thresh = cv2.threshold(img, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

For more details - <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding>

Open 6.py to see all the thresholding techniques and their results.

## Smoothing Images (Blurring)

We can blur the image using low pass filters and even apply custom made filters on our images. Its working can be understood as a kernel which is convoluted over each pixel.



The predefined blurring techniques are

* Averaging
* Gaussian blurring
* Median blurring
* Bilateral filtering

For more details <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering>

Open 7.py

It is not necessary to use a square kernel always. Different size kernels are also useful as you can see this in my answer <https://stackoverflow.com/questions/56249667/how-to-refine-image-with-rough-horizontal-lines-using-opencv/56252251#56252251>

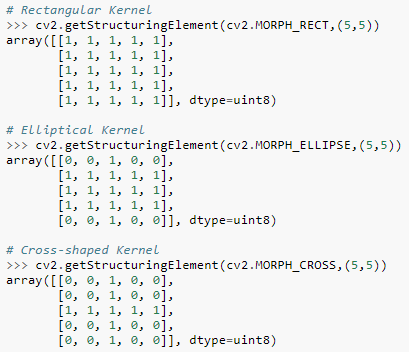
## Morphological operations

Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. Morphological operators — dilate, erode, open, and close — can be applied through image filtering to grow or shrink image regions, as well as to remove or fill-in image region boundary pixels. Additional morphology filters include top-hat transforms. morphological gradient, and morphological Laplace. Best way is to have an image in grayscale or binary using thresholding. Different types explained:

* Erosion – It erodes away the boundaries of foreground object (Always try to keep foreground in white).
* Dilation – It dilates away the boundaries of foreground object (Always try to keep foreground in white).
* Opening – It is erosion followed by dilation. It is useful in removing noise as erosion removes the white noises and also shrinks our object. So we dilate it. Since noise is gone, they won’t come back, but our object area increases.
* Closing – It is dilation followed by erosion. It is useful for filling small holes in our object.
* Morphological Gradient – It is the difference between dilation and erosion of an image. The result is like the outline of the object.
* Top Hat – It is the difference between input image and Opening of the image
* Black Hat – It is the difference between the closing of the input image and input image.

Open 8.py

We need kernels as well for morphological operations. Generally, we use rectangle shaped which can be easily obtained using numpy. If we need different shaped ones we need to use cv2.getStructuringElement().



## Edge detection

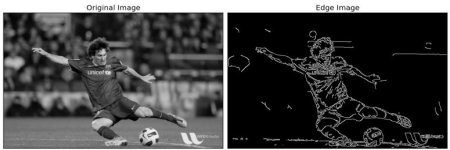
For edge detection we use canny edge detection. It consists of:

* Noise reduction
* Finding Intensity Gradient of Image
* Non-maximum suppression
* Hysteresis thresholding

To read about each step in detail go to <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_canny/py_canny.html#canny>

edges = cv2.Canny(img, 100, 200)

Here img is grayscale image and second and third arguments are minVal and maxVal.



You can build a track bar that controls the second and third arguments of canny edge detector to see what happens with different values.

## Contours

### Introduction

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

* For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
* In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

im, cnts, hierarchy = cv2.findContours(thresh, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE)

In input there are three arguments, first one is source image, second is contour retrieval mode, third is contour approximation method. cnts is a list of contours. Hierarchy will be discussed later.

Contours can be drawn using

cv2.drawContours(img, cnts, -1, (0,255,0), 3)

Where img is image, cnts are the contours obtained. The third argument denotes the contour to be drawn. If we have to draw say only the second contour, then we would pass 1. Passing -1 means drawing all the contours.

There are two contour approximation methods generally used -

* cv2.CHAIN\_APPROX\_NONE – It stores all the boundary points.
* cv2.CHAIN\_APPROX\_SIMPLE – Suppose a contour is found on a straight line. Then it not required to store all points. That’s what this does. It just stores the two endpoints thereby saving memory.

### Contour Features

Contours have several useful features like:

* Moments – An image moment is a certain particular weighted average (moment) of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property or interpretation.

cnt = cnts[0]

M = cv2.moments(cnt)

This can now be used to calculate centroid:

cx = int(M['m10']/M['m00'])

cy = int(M['m01']/M['m00'])

* Contour Area –

area = cv2.contourArea(cnt)

* Contour Perimeter – It is also called arc length.

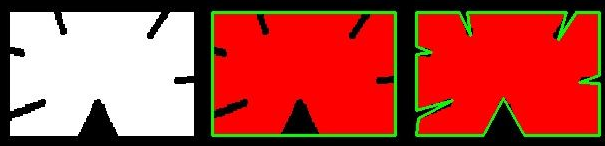
perimeter = cv2.arcLength(cnt, True)

If it is a closed contour second argument is passed as True else pass it as False.

* Contour approximation – It approximates a contour shape to another shape with less number of vertices depending upon the precision we specify.

epsilon = 0.1\*cv2.arcLength(cnt, True)

approx = cv2.approxPolyDP(cnt, epsilon, True)



Value of epsilon as 10% and 1%.

This is also useful in finding different shapes! A good example can be seen on this blog post <https://www.pyimagesearch.com/2016/02/08/opencv-shape-detection/>. Pyimagesearch is a great blog for opencv python.

* Convex Hull – cv2.convexHull() function checks a curve for convexity defects and corrects it. Generally speaking, convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects.

hull = cv2.convexHull(cnt)

* Checking convexity – There is a function to check if a curve is convex or not, cv2.isContourConvex(). It just returns whether True or False.

k = cv2.isContourConvex(cnt)

* Bounding rectangle – We draw a bounding rectangle over our contour. It is of two types:

1. Straight bounding rectangle – It does not consider rotation of an object. Therefore, it is not a minimum area rectangle.

x, y, w, h = cv2.boundingRect(cnt)

cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

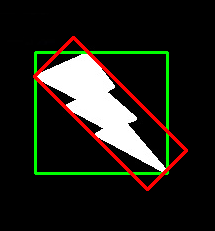
1. Rotated bounding rectangle – It is drawn for minimum area therefore; it considers rotation as well.

rect = cv2.minAreaRect(cnt)

box = cv2.boxPoints(rect)

box = np.int0(box)

cv2.drawContours(img, [box],0, (0, 0, 255), 2)



The green one is simple and red one is rotated bounded rectangle.

* Minimum Enclosing Circle – It is a circle which completely encloses the object with minimum area.

(x, y), radius = cv2.minEnclosingCircle(cnt)

center = (int(x), int(y))

radius = int(radius)

cv2.circle(img, center, radius, (0, 255, 0), 2)

* Fitting an ellipse – It fits an ellipse to an object.

ellipse = cv2.fitEllipse(cnt)

cv2.ellipse(img, ellipse, (0, 255, 0), 2)

* Fitting a line – Similarly a line can be fit.

rows, cols = img.shape[:2]

[vx, vy, x, y] = cv2.fitLine(cnt, cv2.DIST\_L2,0,0.01,0.01)

lefty = int((-x\*vy/vx) + y)

righty = int(((cols-x)\*vy/vx)+y)

cv2.line(img, (cols-1, righty), (0, lefty), (0, 255, 0), 2)

### Contour Properties

For contour properties refer to <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_contours/py_contour_properties/py_contour_properties.html#contour-properties>.

There are many like Aspect ratio, extent, Solidity, etc. I will explain directly from documentation.

### Contour Hierarchy

im, cnts, hierarchy = cv2.findContours(thresh, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE)

Now what is hierarchy in this? And what about the different contour retrieval modes.

Hierarchy denotes the relation between contours. If there are nested objects it tells which is the parent and which is the child. Each contour has an id (number) which denotes which one is it. Hierarchy is an array consisting of [Next, Previous, First\_Child, Parent], where Next denotes next contour in the same hierarchical level, previous denotes previous contour at the same hierarchical level, first child denotes its first child contour and parent denotes index of its parent contour.

Different contour retrieval modes are:

* RETR\_LIST – It is used when we don’t care about hierarchy. So the third and fourth columns of hierarchy are always -1.
* RETR\_EXTERNAL – As its name suggests it does not return any child contours. So in this case also the third and fourth column will have -1.
* RETR\_CCOMP – This flag retrieves all the contours and arranges them to a 2-level hierarchy. i.e. external contours of the object (i.e. its boundary) are placed in hierarchy-1. And the contours of holes inside object (if any) is placed in hierarchy-2. If any object inside it, its contour is placed again in hierarchy-1 only. And its hole in hierarchy-2 and so on.
* RETR\_TREE – This flag retrieves all the contours and creates a full family hierarchy of it.

Open 9.py

## Some Programs

* Perform corrected rotation – As stated above opencv does not perform proper rotation and the image gets cut. However, we can perform it such that the whole image is present with a black mask around the empty parts. Open 10.py
* Document Scanner – A document scanner is made using image processing techniques. We will build a document scanner using opencv. Open 11.py
* Find size of rectangular objects using a euro – How cool would it be to click a photo of an object and find its size. We will make a program which does exactly that using a euro as a reference and we can find the size of other objects using that. Open 12.py
* Using automatic Sudoku solver show perspective transform – To round up opencv we will make an automatic Sudoku solver. We will give the program an input in the form of an image of a Sudoku puzzle and it will give us its output by filling the image with the solution. For this we would also be using optical character recognition using tesseract. For their details refer below. The program is subdivided into 13.py and 14.py. 13.py is image processing of the Sudoku image and 14.py is ocr and Sudoku solving code. Run 14.py for result.

# Optical Character Recognition

## Introduction

Optical character recognition or optical character reader (OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image.

## Tesseract

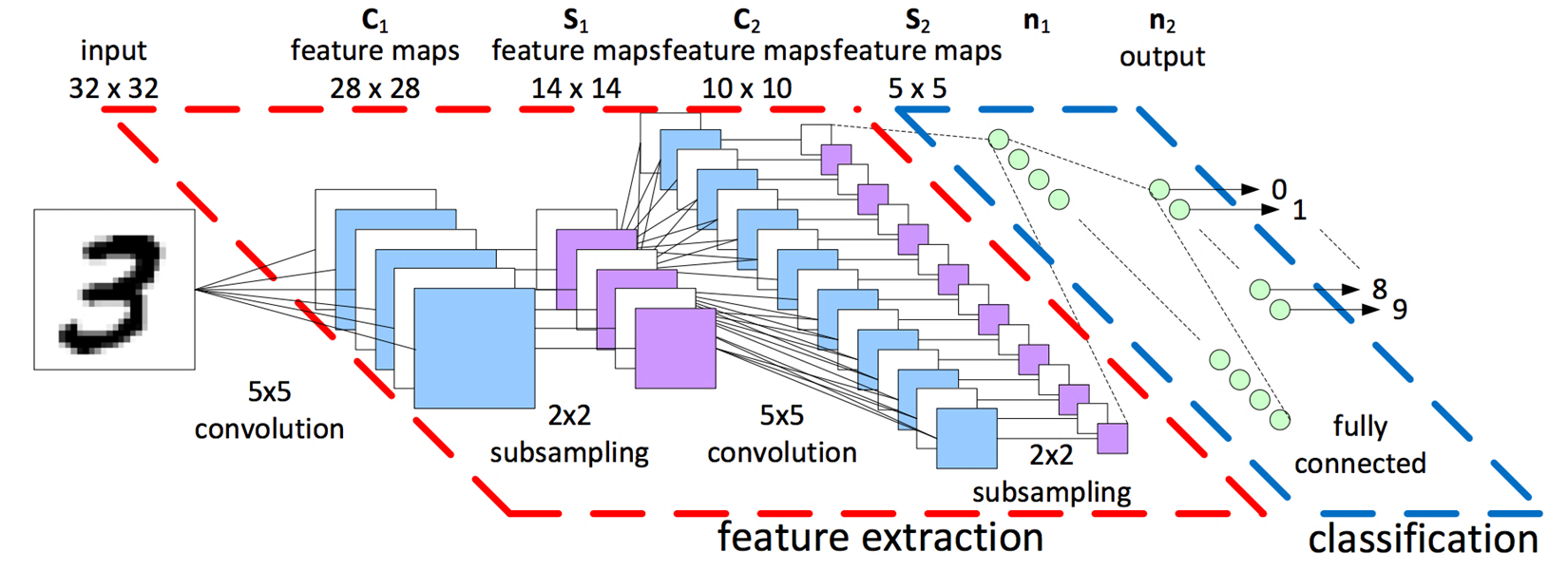
Tesseract is an OCR engine with support for unicode and the ability to recognize more than 100 languages out of the box. It can be trained to recognize other languages. Tesseract is used for text detection on mobile devices, in video, and in Gmail image spam detection.

# Convolutional Neural Networks Using Tensorflow

## What is a convolutional neural network?

A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data. The role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product.



## Different layers of CNN

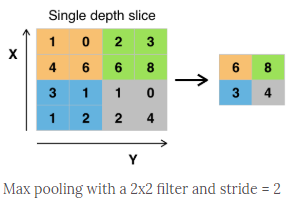
A CNN generally consists of

* Convolutional layer – It is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Three hyperparameters control the size of the output volume of the convolutional layer: the depth, stride and zero-padding.

1. The **depth** of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input. For example, if the first convolutional layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of color.
2. **Stride** controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1 then we move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. When the stride is 2 then the filters jump 2 pixels at a time as they slide around.
3. Sometimes it is convenient to pad the input with zeros on the border of the input volume. The size of this **padding** is a third hyperparameter. Padding provides control of the output volume spatial size.

* Pooling Layer – Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting.



* ReLU layer – ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function f(x)=max (0, x). It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent f(x)=tanh(x), f(x)=|tanh(x)|, and the sigmoid function σ(x)=(1+e^{-x})^{-1)). ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

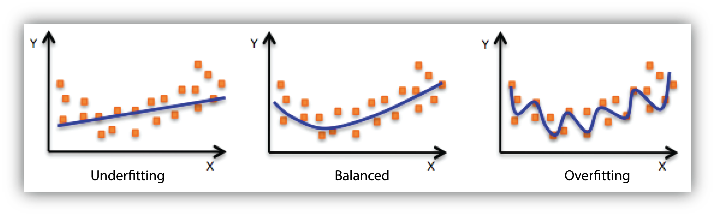
* Flattening - Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer.
* Fully Connected Layer – Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers.
* Loss Layer – The "loss layer" specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions appropriate for different tasks may be used. Softmax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1].

**To remember –** There is no fixed architecture for CNN. You need to read about it a lot to understand more about it and different ways to improve accuracy of your model.

## Overfitting and Underfitting

The accuracy of our training model peaks after a certain number of epochs and then starts decreasing. This is called overfitting. Your model is overfitting your training data when you see that the model performs well on the training data but does not perform well on the evaluation data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

Underfitting is the opposite of overfitting. Your model is underfitting the training data when the model performs poorly on the training data. This is because the model is unable to capture the relationship between the input examples (often called X) and the target values (often called Y).

Solving underfitting is easy. We just need to increase the number of epochs and train the model for a longer time. For solving overfitting, we need to have more training data. But that it generally not possible so we use dropout and regularization.

## Image Augmentation

Image augmentation is a technique that is used to artificially expand the data-set. This is helpful when we are given a data-set with very few data samples. In case of Deep Learning, this situation is bad as the model tends to over-fit when we train it on limited number of data samples. It is recommended that we perform image augmentation on our dataset as it increases the size of training data as well as helps us diversifies our data. Keras has a built in method for data augmentation. Image augmentation parameters that are generally used to increase the data sample count are zoom, shear, rotation, preprocessing\_function and so on. An example is given below:

from keras.preprocessing.image import ImageDataGenerator

image\_gen = ImageDataGenerator(rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.01,

zoom\_range= [0.9, 1.25],

horizontal\_flip=True,

vertical\_flip=False,

fill\_mode='reflect',

brightness\_range= [0.5, 1.5])

Open 15.py and 16.py. For deep learning we can use google colab which provides free GPU. 16.py is on colab and can be viewed from <https://colab.research.google.com/drive/16O7FteO6bBO7aORwP49L7X6CuCzA3cbS>

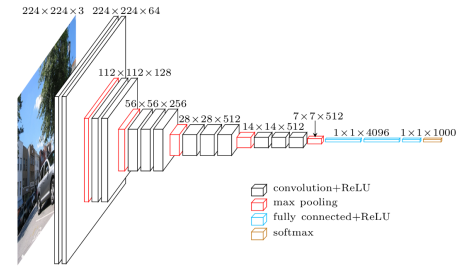
# Transfer Learning

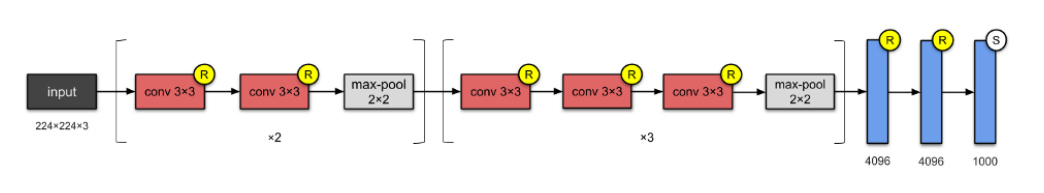
## Introduction

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems. They are trained on large datasets like imagenet.

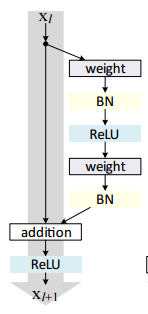
## Different Models

* VGG-16 and VGG-19 – The 16 and 19 stand for the number of weight layers in the network. It uses only 3x3 convolutional layers stacked on top of each other. Max pooling is used to reduce size. Finally, two fully connected layers with 4096 nodes are followed by a Softmax classifier. The input image size is 224x224x3. VGGNet is very slow to train and their size is also very large i.e. 533 MB for VGG-16 and 573 MB for VGG-19.

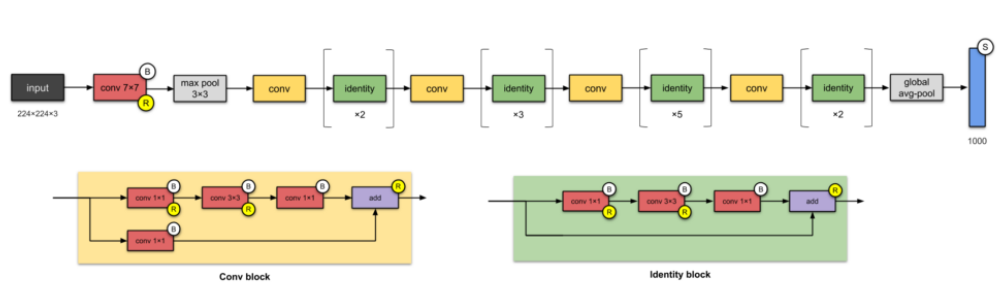




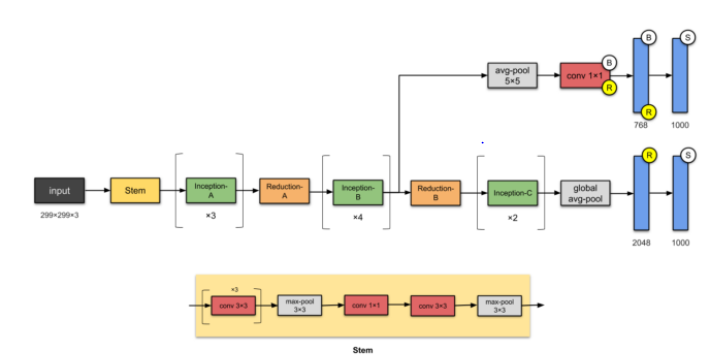
* ResNet – It follows network-in-network architecture that is micro-architecture modules. The ResNet is deeper than VGG still it is smaller and has a size of about 102 MB as it uses global average pooling rather than fully connected layers.

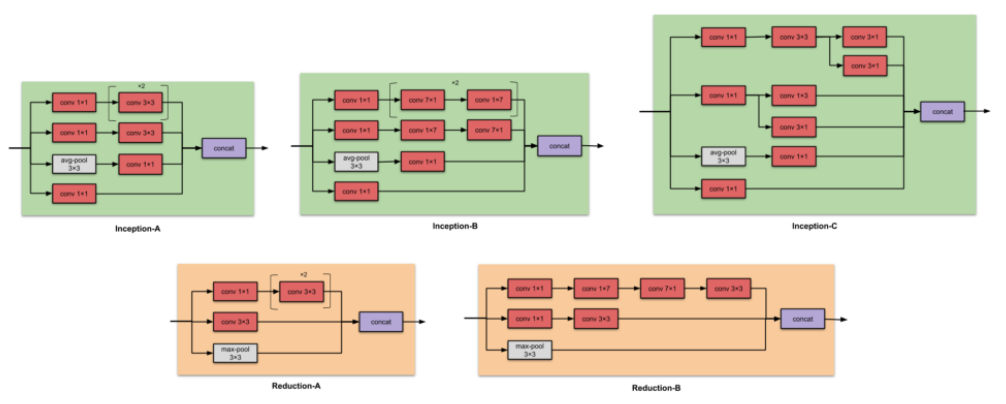


(BN stands for bottleneck block)



* Inception V3 – The goal of the inception module is to act as a “multi-level feature extractor” by computing 1×1, 3×3, and 5×5 convolutions within the same module of the network — the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network. Weights for Inception V3 is smaller than both VGG and ResNet and has a size of 96 MB. The size of input image should be 299×299.





There are many more like Xception, AlexNet, SqueezeNet, etc.

The models can be loaded from tf.keras.applications. We can use the model as a feature extractor or add dense layers in the end according to our dataset.

References

Opencv-python documentation

Pyimagesearch

Machinelearningmastry

Towards data science medium articles