Computer Vision : Assignment 3

Question 1: Stereo Vision **[20]**

Click some pictures of different scenes and show their disparity maps. Also calculate their depth maps. Explain the working of the inbuilt function you have used in detail in the report.

Stereo vision is an area within the field of computer vision that addresses the reconstruction of the three-dimensional coordinates of points for depth estimation. A stereo vision system consists of a stereo camera, namely, two cameras placed horizontally (i.e., one on the left and the other on the right). The two images captured simultaneously by these cameras are then processed for the recovery of visual depth information. The challenge is to determine the best method of approximating the differences between the views shown in the two images to map (i.e., plot) the correspondence (i.e., disparity) of the environment. Intuitively, *a disparity map represents corresponding pixels that are horizontally shifted between the left image and right image*.

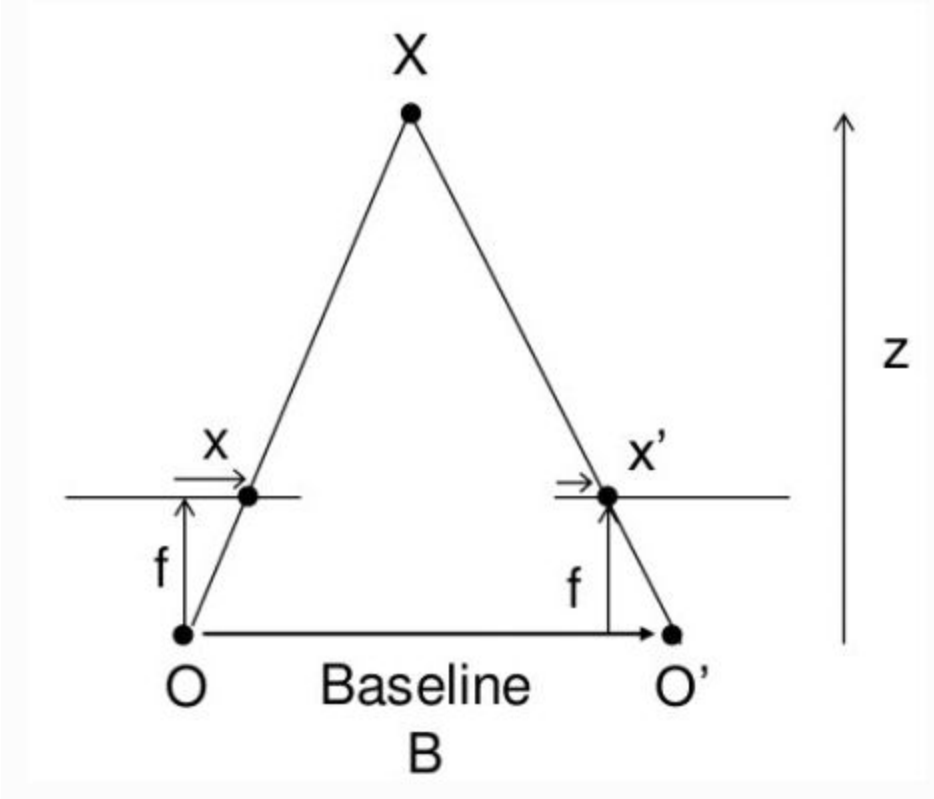
Calculating Disparity map:

The images are padded with a frame of zero pixels to facilitate the window operation (SSD/SAD) at the border.

SSD - Sum of Squared Differences and SAD - Sum of Absolute Differences

First, squared difference or absolute difference is calculated for each pixel and then all the values are summed over a window W.

For each shift value of the right image, there is an SSD/SAD map equal to the size of the image. The disparity map is a 2D map reduced from 3D space. The disparity of a pixel is equal to the shift value that leads to minimum SSD/SAD for that pixel.



So, disparity is calculated using the following formula:

Disparity = x - x' = Bf/Z

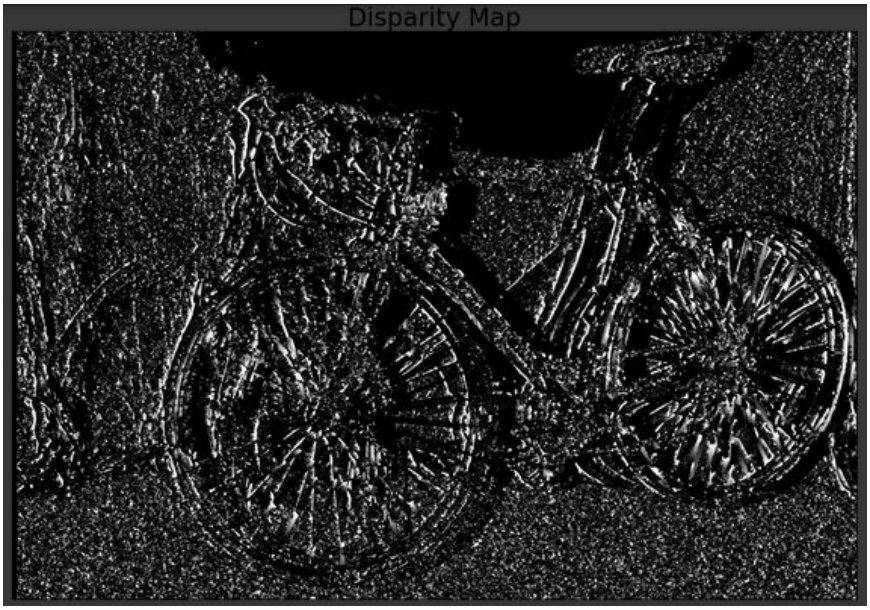
x and x' are the distance between points in the image plane corresponding to the scene point 3D and their camera center. B is the distance between two cameras(already known) and f is the focal length of the camera (already

known). So in short, the above equation says that the depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points and their camera centers. So with this information, we can derive the depth of all pixels in an image.
I have used **cv2.StereoBM_create()** function for calculating disparity map. The parameters used for this function are - **numDisparity = 16 and blockSize = 9.**
The disparity of each pixel is encoded by a gray value. Light gray values represent high disparities and dark gray values small disparities. The resulting image is called a disparity map.
In this program, I have used 3 pairs of images for showing the disparity and depth map. For disparity calculation, we need the focal length and the distance between two cameras.
.

Cycle: Left Image Right Image

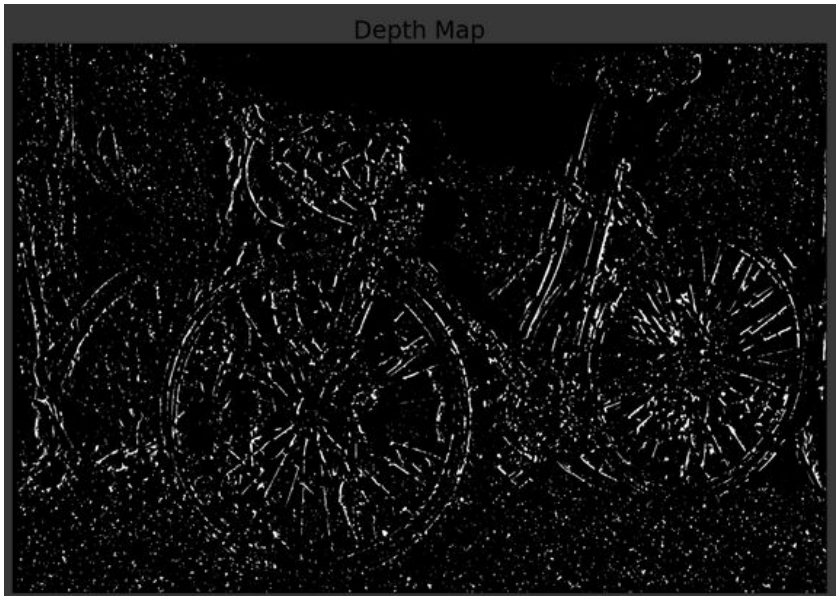
Focal Length used : 5299.313 and Base distance: 177.288

Disparity Map:



Disparity map is calculated by matching the similarities in the **image epipolar geometry** by **sliding a window** along the right scanline and comparing contents of that window with the reference window. The minimum cost point would be the same pixel value. Smaller window gives more detail with added noise and a larger window gives a smoother disparity map with less detail.

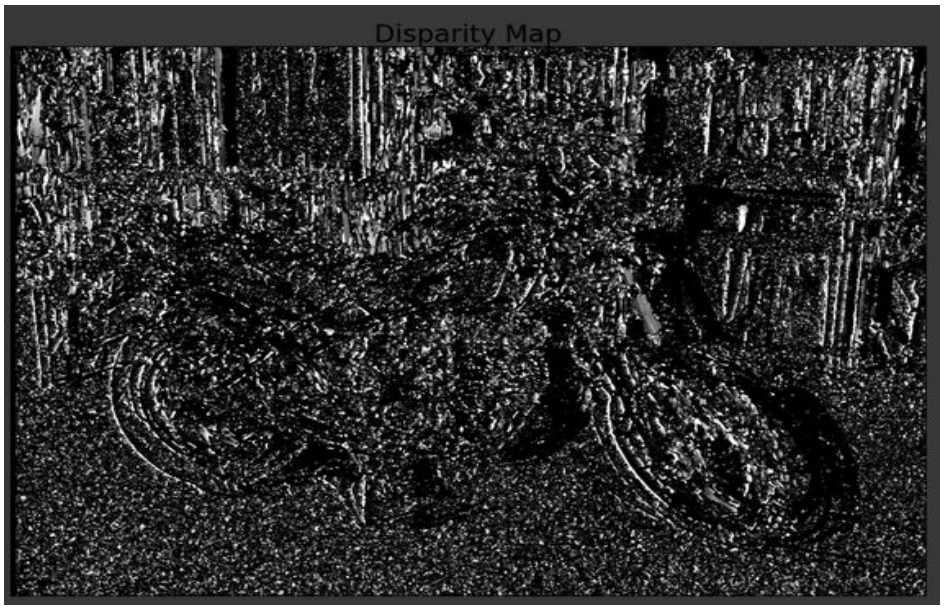
Depth Map: Depth is calculated by dividing the product of distance and focal length with disparity map.



Motorcycle: Left Image Right Image

Focal Length used : 3979.911 and Base distance: 193.001

Disparity Map:



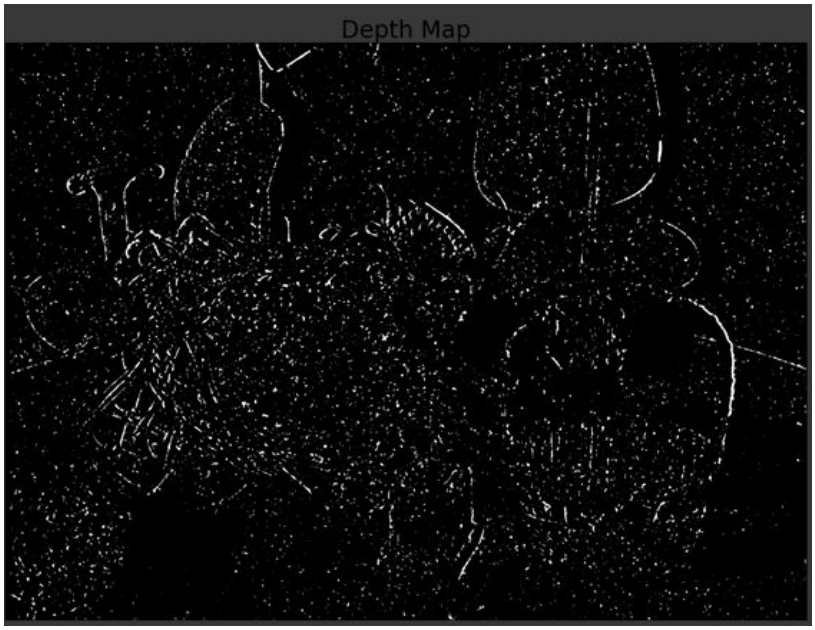
Depth Map:

Mask: Left Image Right Image

Focal Length used : 4844.97 and Base distance: 170.458

Disparity Map:

Depth Map:

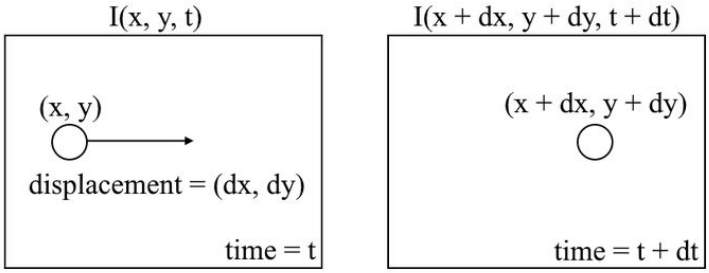


Question 3: Face detection and Tracking **[30]**

Capture a video of a person whose face is visible most of the time. Use a pre-trained Viola-Jones face detector to detect the face in the first frame and after that use Optical Flow algorithm to continue tracking it in the rest of the video frames. Report the intermediate results and explain the working of the algorithm. You can test the algorithm with more videos.

In this problem, I have used a face video of a person for face detection and tracking. I have used a pre-trained Viola-Jones face detector from OpenCV library to detect a face in the first frame of the video then I have used the Optical Flow algorithm to track the features.

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of camera or objects. The 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second. The problem of optical flow is shown below.



The image intensity (I) is a function of space (x,y) and time (t) and if we take the first image I(x,y,t) and move its pixels by (dx,dy) over t time, we obtain the new image I(x+dx,y+dy,t+dt). After Taylor's series expansion, we get the optical flow equation as -

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0$$

where u=dx/dt and v=dy/dt

dI/dx ,dI/dy and dI/dt are the image gradients along the horizontal axis, the vertical axis, and time. The problem of optical flow is solving u(dx/dt) and v(dy/dt) to determine movement over time. We cannot directly solve the optical flow equation for u and v since there is only one equation for two unknown variables.

We identify some interesting features to track and iteratively compute the optical flow vectors of these points. However, adopting the Lucas-Kanade method only works for small movements (from our initial assumption) and fails when there is large motion. Therefore, the OpenCV implementation of the Lucas-Kanade method adopts pyramids.

Program explanation:

Celebration video of Cristiano Ronaldo is taken for the face recognition and tracking. The video has the following details. Also the first frame is shown below.

```
FPS: 30
Frame Size: (1280, 720)
Frame count: 105
```

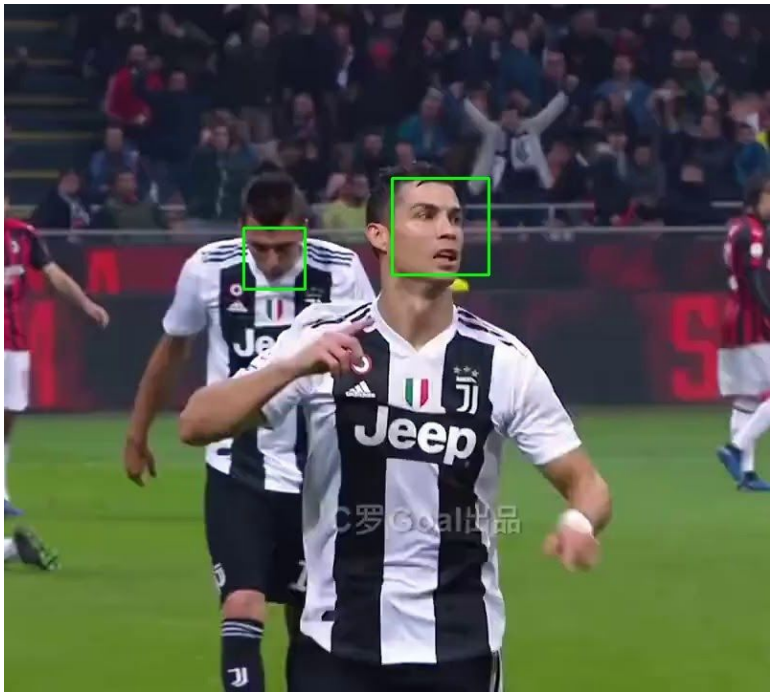
cv2.VideoCapture() function is reading the video frame by frame.

Face Detection:

For the face detection task, I have used cv2.CascadeClassifier() with *haarcascade_frontalface_default.xml* as the pre-trained viola-jones classifier. **cv2.CascadeClassifier.detectMultiScale()** method gives a list of rectangles for the detected objects. I have used minNeighbours(Parameter specifying how many neighbors each candidate rectangle should have to retain) = 4 and minSize(Minimum possible object size) = (50,50). This has given us the following output.

```
Detected Face rectangles:
 [[604 181 103 103]
 [452 232  64  64]]
```

Now if we plot the rectangle in the first frame, we can see the bounding boxes over the faces.



Now I have chosen only a single face(Ronaldo's) rectangle for the rest of the code.

Tracking:

For tracking the face, we need to find the 4 points (three random in the rectangle and one in the centre). These are the *good features that we would like to track using the optical flow algorithm*. The sample tracking points are shown below.

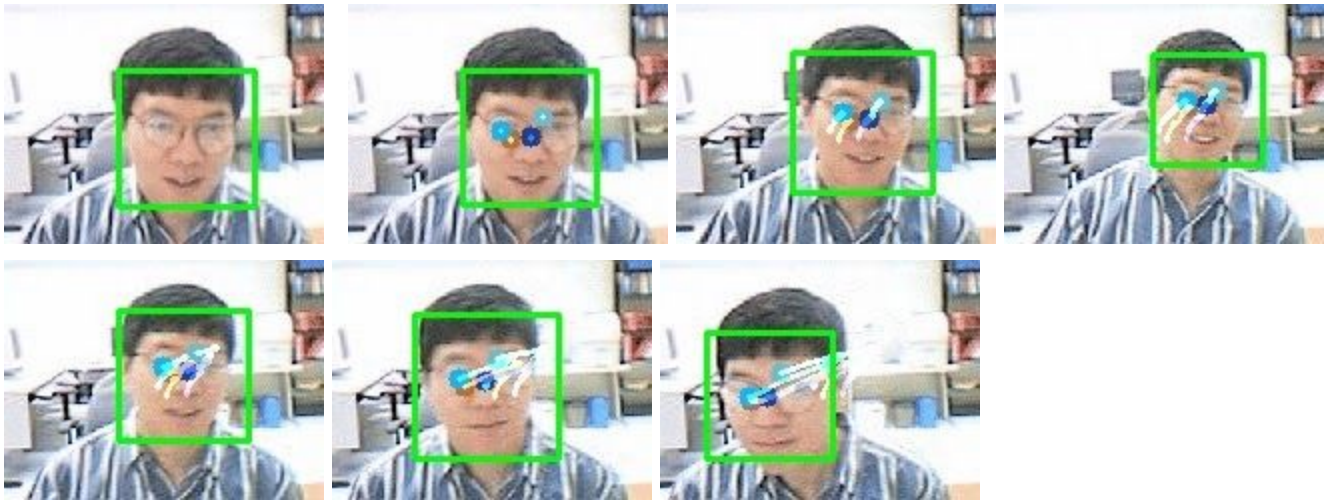
```
Tracking Points:
 [[[461.63898 210.05482]]

 [[458.24667 248.03429]]

 [[474.39648 246.88142]]

 [[455.5     232.5    ]]]
Rectangle centre point:
 (0, 455.5, 232.5)
```

For Lucas Kanade optical flow, we have defined the following parameters with **calcOpticalFlowPyrLK()**.

```
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (30,30),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

It calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.
It uses 3 levels of pyramids with window size (30,30).
This gives estimated points in the next frame and also the status and error if needed. Now we track the features using color circles and color lines from old frame to new frame. This gives us the sparse optical flow of the good features.
We have 11 unique rectangles for our 105 frame video. These will give new points to track inside a face rectangle.
The few outputs are shown below -

I have also done this procedure for another video. The few frames are as follows.



I have added the frames and made the videos which are attached to the folder for this question.

Question 3: Object Recognition [40]

Use the CIFAR10 dataset for this question.

Extract the following features from the images and train a 2-layer neural network for classification.

a. Local Binary Patterns (LBP)

b. Scale-Invariant feature Transform (SIFT)

c. Deep features (Use AlexNet pre-trained on Imagenet to extract the 4096 dimensional feature vector)

d. Deep features (Use ResNet18 pre-trained on Imagenet to extract feature vector)

Report the classification results and do some comparative analysis between them.

Answer:

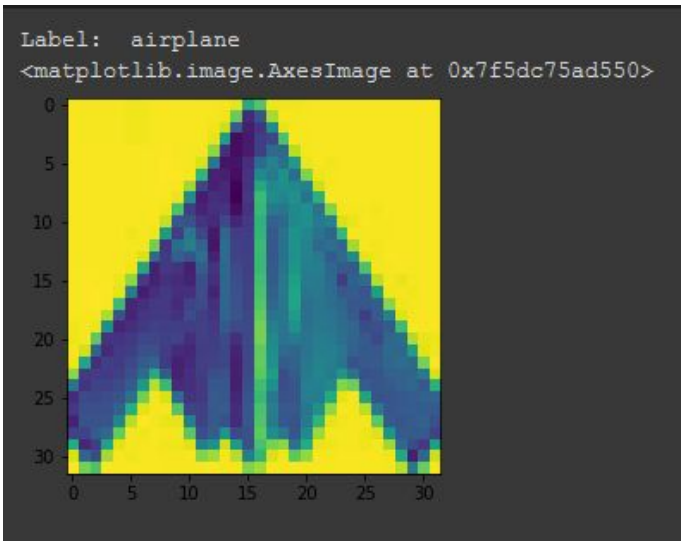
Dataset Description : In this question, we have the task of object recognition on CIFAR10 dataset. In this dataset, there are a total 60000 images available in 10 categories. The image resolution is 32*32. The dataset is divided into train and test dataset of 50000 and 10000 respectively. The categories are as follows -

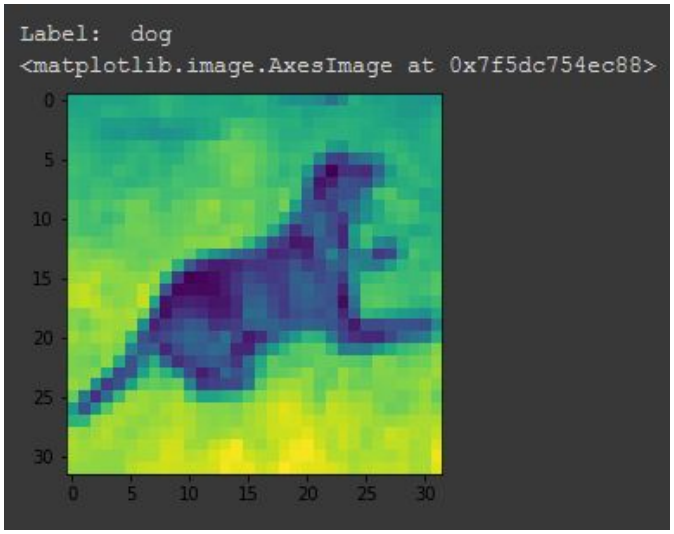
```
['labels.txt', 'train', 'test']
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Frameworks/Libraries used:

1. Pytorch
2. Cv2
3. Numpy
4. Matplotlib
5. Torch
6. Torchvision
7. Skimage
8. copy

Data pre-processing: The image set has been divided into 3 parts - train, test and validation. Train, validation and test set contains 40000, 10000 and 10000 images respectively. All the images have been converted into grayscale and *torch tensor* before creating the corresponding *torch dataloader*. Training and validation set have a batch size of 128 and the test has a batch size of 256. The sample images are shown below.





A. <u>Object detection with Local Binary Pattern with 2 layer Neural Network:</u>

So we will first find the local binary patterns(LBPs) of each image and then we will train a 2 layer neural network using the features. T

Local Binary Patterns(LBPs):

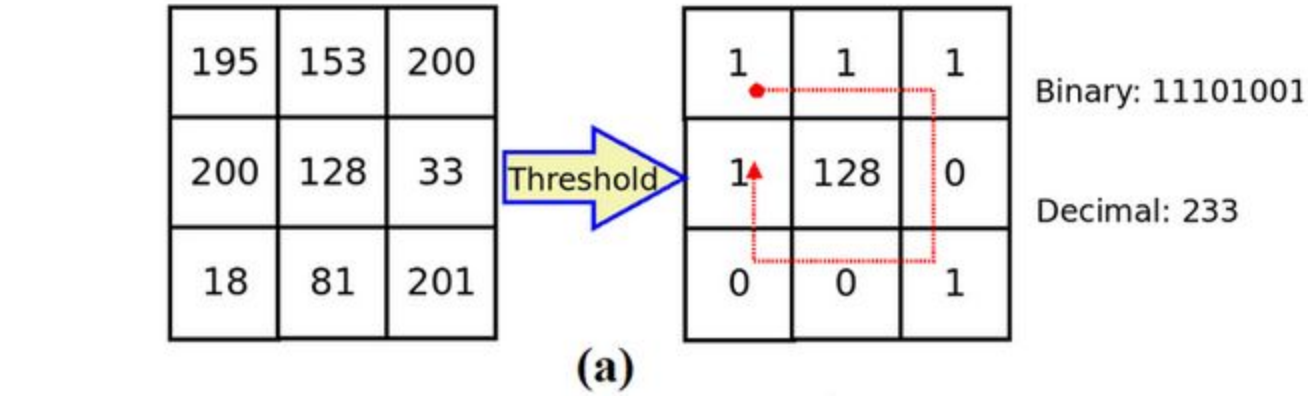
Unlike several global feature extractor like HOG etc., LBP is a local feature extraction technique. This local representation is constructed by comparing each pixel with its surrounding neighborhood of pixels. The algorithm is described in detail.

There are mainly 4 parameters that are needed for LBPs.

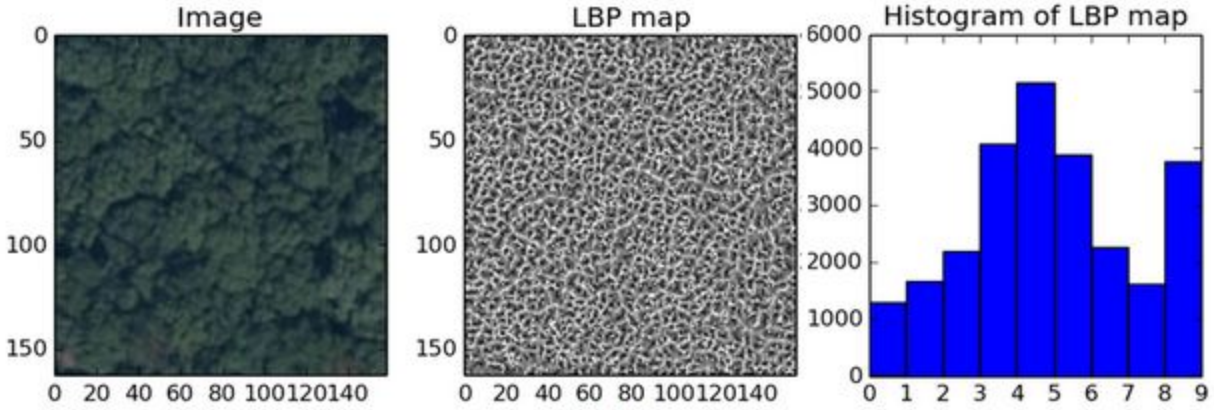
1. **Radius**: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. **I have taken the radius value 1.**
2. **Neighbors**: the number of sample points to build the circular local binary pattern. **I have taken the neighbour value 24.**
3. **Grid X:** The number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
4. **Grid Y**: The number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

Finding the LBPs:

To find the LBP histogram, we need to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters radius and neighbors.



(a)



(b)

Based on the image above, let's break it into several small steps so we can understand it easily:

a. A grayscale image is represented as a 3x3 matrix containing the intensity of each pixel (0~255).
b. Then, we need to take the central value of the matrix to be used as the threshold.
c. This value will be used to define the new values from the 8 neighbors.
d. For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
e. Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g. 10001101).
f. Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is actually a pixel from the original image.
g. At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.

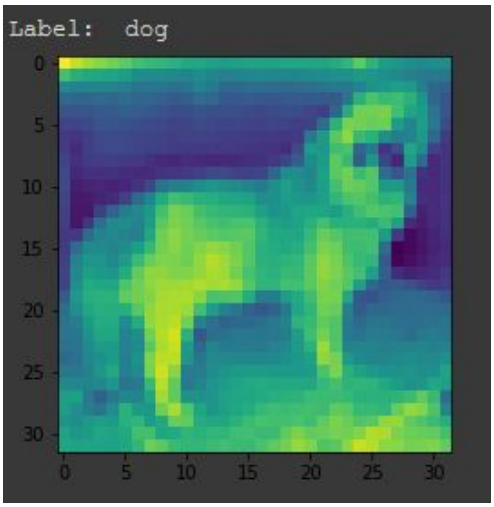
We can expand this procedure with different numbers of radius and neighbors, it is called **Circular LBP**.

Here, I have taken **uniform LBP**. A LBP is considered to be uniform if it has at most two 0-1 or 1-0 transitions. For example, the pattern 00001000 (2 transitions) and 10000000 (1 transition) are both considered to be uniform patterns since they contain at most two 0-1 and 1-0 transitions. The pattern 01010010) on the other hand is not considered a uniform pattern since it has six 0-1 or 1-0 transitions.

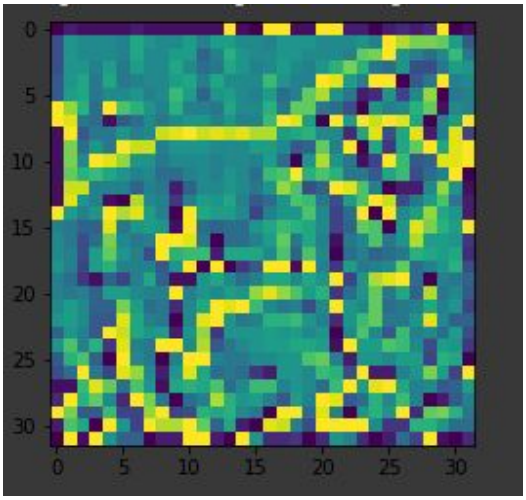
Experiments:

The number of uniform prototypes in a Local Binary Pattern is completely dependent on the number of points p. As the value of p increases, so will the dimensionality of your resulting histogram. The number of points p in the LBP there are p + 1 uniform patterns. The final dimensionality of the histogram is thus p + 2 (**In our case, it is 24+2=26**) , where the added entry tabulates all patterns that are not uniform.

The LBPs on a dog image is shown below with radius = 1, point =24 and histogram dim = 26.



Dog Image



LBP Histogram

```
Hist: [0.03808594 0.03613281 0.02636719 0.01464844 0.01367187 0.01757812
 0.02148437 0.02441406 0.02734375 0.04296875 0.05761719 0.09667969
 0.10644531 0.09863281 0.0625     0.04882812 0.03417969 0.0234375
 0.02148437 0.02050781 0.01269531 0.01074219 0.01367187 0.01074219
 0.03320312 0.0859375 ]
```

Histogram values

I have stacked the histogram of a batch to create the feature vector of a batch. So the train feature vector has dimension = 128 * 26

Training the 2 layer neural network with LBP features:

I have created a 2 layer neural network with input size = 26, hidden size = 16 and output is equal to number of image categories (10). ***ReLU*** *is chosen as the activation function. Since it is a classification task, I have used* ***cross entropy loss*** *for cost function.* **Adam optimizer** has been used for gradient calculation. The model is shown below.

```
CifarModel(
  (linear1): Linear(in_features=26, out_features=16, bias=True)
  (linear2): Linear(in_features=16, out_features=10, bias=True)
)
```

The number of parameters in the model is 4.

The initial learning rate is chosen as 0.005 and I have used a learning rate scheduler with step size 15 and gamma 0.1. So it will decrease the learning rate by 10 times after each 15 epochs. Total number of epochs used 40.

Results:

After training for 40 epochs, the 2 layer neural network with LBPH features has achieved the below result.

```
epoch: 4, training_loss: 2.0435, training_acc: 0.2552, val_loss: 2.0311, val_acc: 0.2606
epoch: 8, training_loss: 1.9902, training_acc: 0.2711, val_loss: 1.9773, val_acc: 0.2763
epoch: 12, training_loss: 1.9686, training_acc: 0.2818, val_loss: 1.9622, val_acc: 0.2773
epoch: 16, training_loss: 1.9543, training_acc: 0.2904, val_loss: 1.9433, val_acc: 0.2988
epoch: 20, training_loss: 1.9455, training_acc: 0.2947, val_loss: 1.9349, val_acc: 0.2979
epoch: 24, training_loss: 1.9413, training_acc: 0.2976, val_loss: 1.9309, val_acc: 0.3006
epoch: 28, training_loss: 1.9392, training_acc: 0.2971, val_loss: 1.9295, val_acc: 0.3000
epoch: 32, training_loss: 1.9379, training_acc: 0.2994, val_loss: 1.9297, val_acc: 0.2980
epoch: 36, training_loss: 1.9360, training_acc: 0.2977, val_loss: 1.9279, val_acc: 0.3007
epoch: 40, training_loss: 1.9353, training_acc: 0.2994, val_loss: 1.9275, val_acc: 0.2999
```

Best training Loss : 1.9269
Best training accuracy: 0.3038
Best Validation Loss: 1.9275
Best Validation accuracy: 0.3069

I have chosen the model parameters which have given the best validation result. The final model is tested using a test dataset which has given me the below results for test data.

```
test_loss: 1.9264, test_acc: 0.3029
```

Conclusion:

Since LBPH is only giving us the histogram of local features and the model is trained only with a single hidden layer, it is only giving us 30% accuracy in object detection.

B. **<u>Object detection with Scale Invariant Feature Transform(SIFT) with 2 layer Neural Network:</u>**

Scale Invariant Feature Transform (SIFT) is a feature extraction technique that learns image features which remain scale invariant.

The steps are as follows -

1. **Scale-space extrema detection** - To extract these features, a window with a single shape is not efficient. Detecting larger corners requires larger windows. For this, scale-space filtering is used.
   We need to find the Laplacian of Gaussian for different σ values. Gaussian kernel with low σ gives high value for small corner while gaussian kernel with high σ fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x,y,σ) values which means there is a potential keypoint at (x,y) at σ scale.

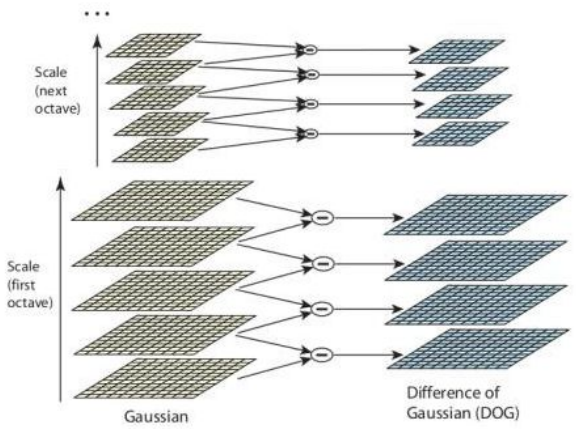
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Blurred image

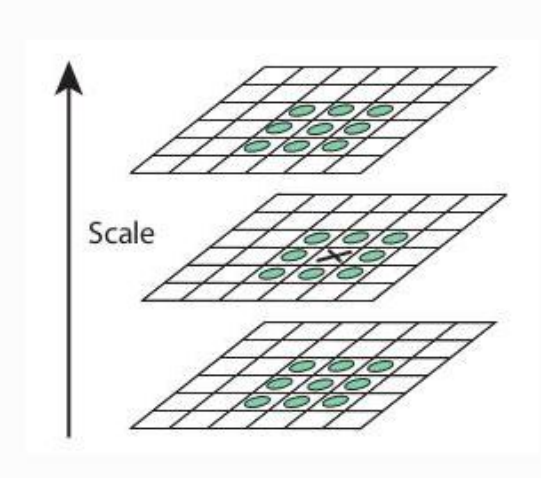
$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Gaussian Blur operator

But this LoG is a little costly, so the SIFT algorithm uses Difference of Gaussians(DoG) which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and kσ. This process is done for different octaves of the image in the Gaussian Pyramid.



Once these DoGs are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that the keypoint is best represented in that scale.



2. **Keypoint Localization:**

As we get keypoint locations, they are refined using Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrastThreshold in OpenCV. So it only retains the strong interest points.

3. **Orientation Assignment:**

An orientation is assigned to each keypoint to achieve invariance to image rotation. A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with the same location and scale, but different directions. It contributes to stability of matching.

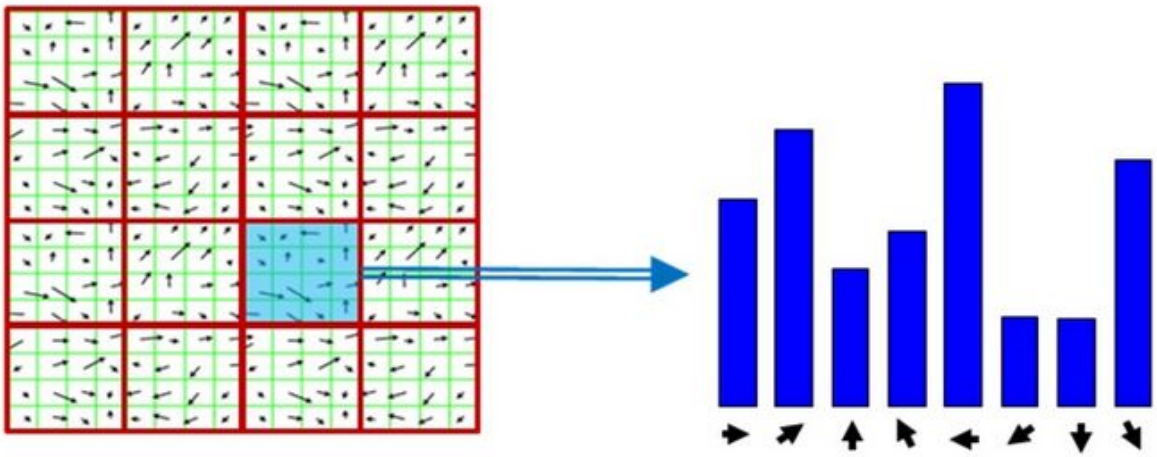
4. **Keypoint descriptor:**

Now a 16x16 neighbourhood around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size. For each sub-block, an 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form a keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

5. **Keypoint Matching:**

Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first and then a ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected.

Divide the feature region into 16 square subregions. Bin each region into an 8-direction histogram



via Gil's CV Blog

Concatenate the 16 histograms together to get the final 128-element SIFT descriptor!

via Gil's CV Blog

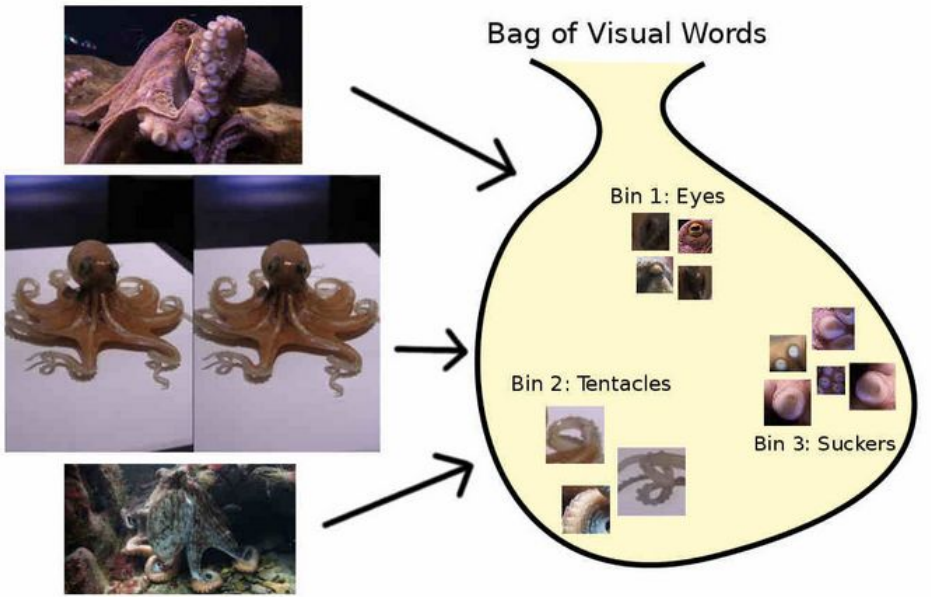
Experiments:

I have used **the Bag of Words** model for extracting the histogram for a batch of images. This feature will be fed into the NN for detection.

Creating fixed number of descriptors: Since SIFT generates different number of feature descriptors for different images, I have created a 4*4 grid for each image so that every grid has one SIFT descriptor. So each image of size 32*32 has 8 SIFT features.

Clustering the descriptor for a batch: Now I have stacked all the descriptors for a batch into a single feature vector. This feature array has size of 128* 8. Now I have used the K Means clustering algorithm to cluster the points into specific k = 100 clusters. 10 Iterations with epsilon 1.0 have been used for clustering criteria.

Creating the Bag of Visual Words: SIFT features are not so literal as a specific word, they tend to represent something more like “bright blob which gradually blends into a darkish blob with a diagonal oblong midtone blob”. We can’t bin them together based on some human-meaningful definition. But we can bin them together mathematically. SIFT descriptors are 128-dimensional vectors, so we can simply make a matrix with every SIFT descriptor in our training set as its own row, and 128 columns for each of the dimensions of the SIFT features.The K means clustering has separated the features into k=100 clusters. Now I have created a histogram based on the frequency of each feature which belongs to the K clusters. This has generated a bag of words model for the batch.



Creating the NN model: The NN model is created with input size = 100 , hidden layer size = 64 and output size of 10(number of categories). ***ReLU*** *is chosen as the activation function. Since it is a classification task, I have used* ***cross entropy loss*** *for cost function.* **Stochastic Gradient Descent (SGD)** is used for gradient calculation.

The initial learning rate is chosen as 0.0005 and total number of epochs used 30.

Results:

After training for 30 epochs, the 2 layer neural network with **SIFT features with Bag of Visual Word vectors** has achieved the below result.

```
Epoch [1], val_loss: 1.9386, val_acc: 0.2843
Epoch [2], val_loss: 1.8459, val_acc: 0.3213
Epoch [3], val_loss: 1.8030, val_acc: 0.3411
Epoch [4], val_loss: 1.7752, val_acc: 0.3577
Epoch [5], val_loss: 1.7689, val_acc: 0.3695
Epoch [6], val_loss: 1.7174, val_acc: 0.3826
Epoch [7], val_loss: 1.6994, val_acc: 0.3792
Epoch [8], val_loss: 1.6440, val_acc: 0.4033
Epoch [9], val_loss: 1.6480, val_acc: 0.4055
Epoch [10], val_loss: 1.6771, val_acc: 0.3992
Epoch [11], val_loss: 1.7375, val_acc: 0.3945
Epoch [12], val_loss: 1.5890, val_acc: 0.4401
Epoch [13], val_loss: 1.6046, val_acc: 0.4299
Epoch [14], val_loss: 1.5731, val_acc: 0.4380
Epoch [15], val_loss: 1.6136, val_acc: 0.4289
Epoch [16], val_loss: 1.6008, val_acc: 0.4341
Epoch [17], val_loss: 1.5581, val_acc: 0.4450
Epoch [18], val_loss: 1.5913, val_acc: 0.4359
Epoch [19], val_loss: 1.5793, val_acc: 0.4379
Epoch [20], val_loss: 1.5526, val_acc: 0.4485
Epoch [21], val_loss: 1.5452, val_acc: 0.4563
Epoch [22], val_loss: 1.5373, val_acc: 0.4533
Epoch [23], val_loss: 1.5718, val_acc: 0.4490
Epoch [24], val_loss: 1.5366, val_acc: 0.4640
Epoch [25], val_loss: 1.5788, val_acc: 0.4456
Epoch [26], val_loss: 1.5205, val_acc: 0.4645
Epoch [27], val_loss: 1.5925, val_acc: 0.4411
Epoch [28], val_loss: 1.5119, val_acc: 0.4677
Epoch [29], val_loss: 1.5300, val_acc: 0.4660
Epoch [30], val_loss: 1.5247, val_acc: 0.4642
```

Best Validation Loss: 1.5119
Best Validation accuracy: 46.77%

I have chosen the model parameters which have given the best validation result. The final model is tested using a test dataset which has given me the below results for test data.

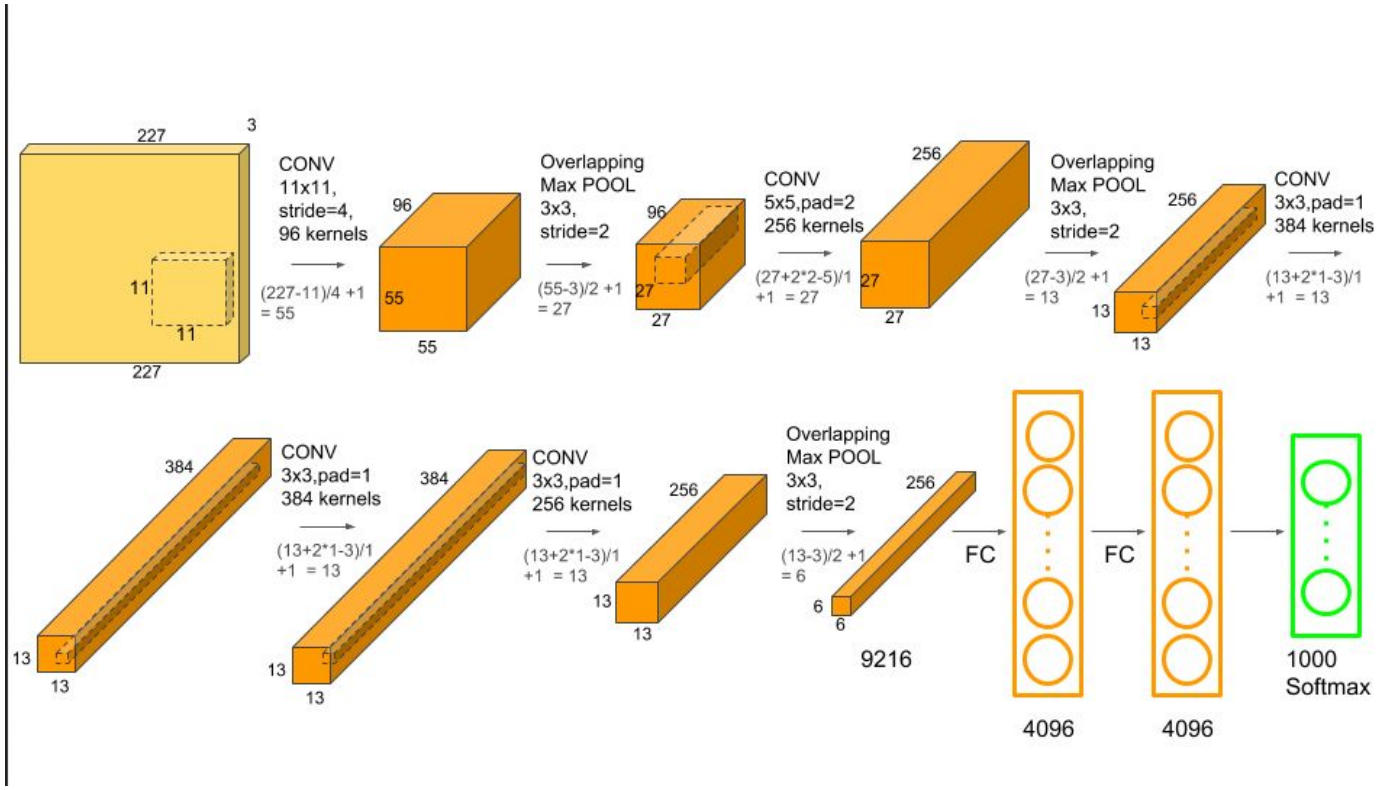
```
test_loss: 1.5153, test_acc: 0.4752
```

Conclusion:

Since SIFT is giving us the histogram of features using the Bag of word model and the model is trained only with a single hidden layer, it is only giving us **47.5% test accuracy** in object detection.

C. **<u>Object detection with Deep Features(AlexNet pretrained on ImageNet with 4096 dimension) with 2 layer Neural Network:</u>**

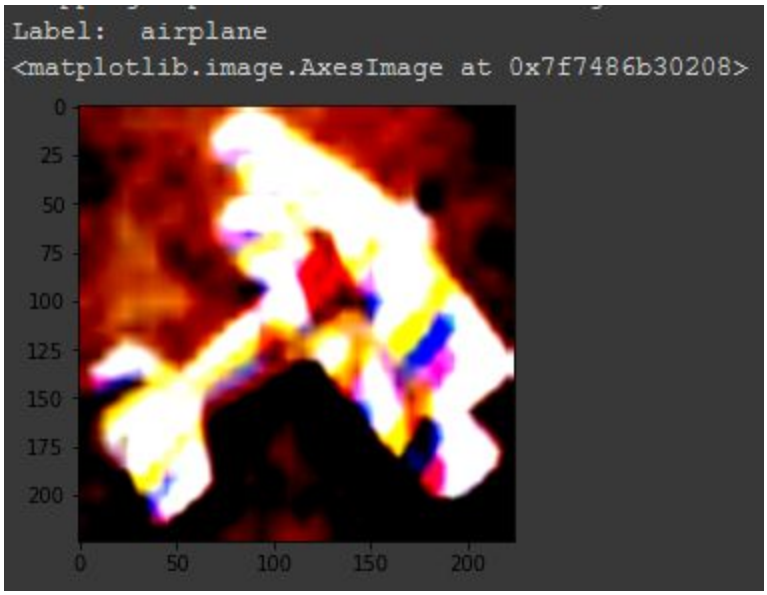
AlexNet was introduced as opposing the standard CNN networks. The architecture consists of eight layers: five convolutional layers and three fully-connected layers. In ALexNet, ReLU was first introduced for CNN based object recognition models instead of tanh functions. It has also introduced Data augmentation and Dropout to avoid overfitting.



AlexNet Architecture

Data Preprocessing:

AlexNet needs the input image size of 224*224. So I have resized all the images to this resolution. Also I have normalized the images using mean=(0.4914, 0.4822, 0.4465) and std=(0.2023, 0.1994, 0.201), which is standard on ImageNet RGB data . A sample is shown below.



Pretrained AlexNet model: Pretrained AlexNet has 60 million parameters which takes an enormous amount of GPU and time to train on ImageNet dataset. We Need to get the 4096 dimension of the feature vector, so that we could train a 2 layer NN with those features instead of 1000 output final layer of AlexNet.

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Pretrained AlexNet

I have added a sequential layer of a single *hidden layer with 2752 neurons and the output layer of size 10. ReLU has* been used as an activation function.

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Sequential(
      (0): Linear(in_features=4096, out_features=2752, bias=True)
      (1): ReLU(inplace=True)
      (2): Linear(in_features=2752, out_features=10, bias=True)
    )
  )
)
```

Updated AlexNet last layer

I have frozen all the parameters except the last sequential layer. So only 4 parameters need to be updated.

```
Parameters to update for Feature extraction:
classifier.6.0.weight
classifier.6.0.bias
classifier.6.2.weight
classifier.6.2.bias
```

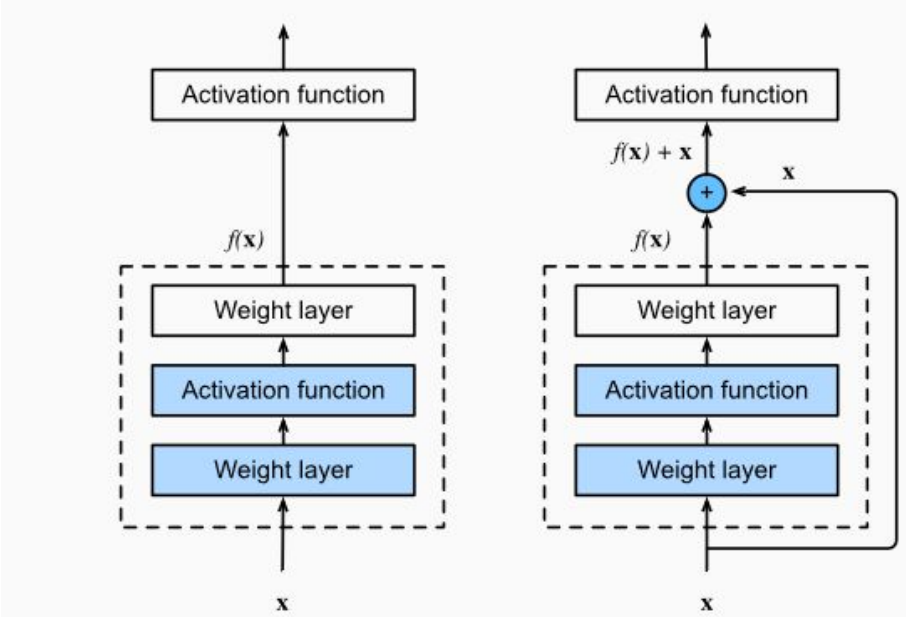
Training: For training the last two layers, I have used Stochastic Gradient Descent (SGD) with momentum for the above four parameters. The *learning rate is 0.005 and momentum is 0.9. Cross Entropy loss is used as the cost function. The model is trained with 40000 images CIFAR dataset and validated with another 10000 images for 30 epochs. Best* ***validation accuracy is obtained 84.66%.***

On the test dataset of 10000 images, this model has achieved *84.03% accuracy* which is much better than LBPH and SIFT models that we have created before.

```
test_loss: 0.4570, test_acc: 0.8403
```

D. **Object detection with Deep Features(ResNet18 pretrained on ImageNet with 512 dimension) with 2 layer Neural Network:**

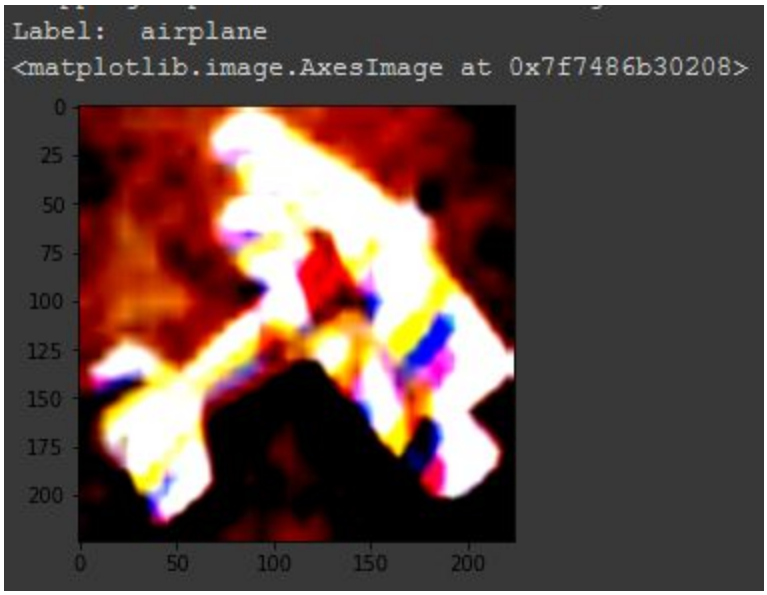
ResNet features special skip connections for removing the vanishing gradient issue in deep CNNs and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network.



The difference between a regular block (left) and a residual block (right). In

Data Preprocessing:

ResNet18 needs the input image size of 224*224. So I have resized all the images to this resolution. Also I have normalized the images using mean=(0.4914, 0.4822, 0.4465) and std=(0.2023, 0.1994, 0.201), which is standard on ImageNet RGB data . A sample is shown below.



Pretrained ResNet18: We are using 18 layer pretrained ResNet. It has lesser parameters and lesser skip connections. The penultimate layer produces a 512-dimension feature vector. This feature vector is then used to train 2 layer FCN for our classification.

I have added a sequential layer of a single *hidden layer with 64 neurons and the output layer of size 10. ReLU has* been used as an activation function.

```
(fc): Sequential(
  (0): Linear(in_features=512, out_features=64, bias=True)
  (1): ReLU(inplace=True)
  (2): Linear(in_features=64, out_features=10, bias=True)
```

I have frozen all the parameters except the last sequential layer. So only 4 parameters need to be updated.

```
Parameters to update for Feature extraction:
fc.0.weight
fc.0.bias
fc.2.weight
fc.2.bias
```

Training: For training the last two layers, I have used Stochastic Gradient Descent (SGD) with momentum for the above four parameters. The *learning rate is 0.005 and momentum is 0.9. Cross Entropy loss is used as the cost function. The model is trained with 40000 images CIFAR dataset and validated with another 10000 images for 30 epochs. Best* ***validation accuracy is obtained 82.12%.***

On the test dataset of 10000 images, this model has achieved *81.5% accuracy* which is much better than LBPH and SIFT models that we have created before.

Comparison:

Model Description	Test Accuracy	Remarks
Linear Binary Pattern with NN	30 %	Local features with histogram
SIFT with NN	47.5 %	Scale invariant features with bag of visual words
AlexNet with FC layer	84.03 %	Deep CNN
ResNet18 with FC layer	81.5 %	Residual Deep CNN

Fine Tuning ResNet18 may have given better results than others. More ResNet layers should have increased the test accuracy over other models.
Training ResNet18 is more computationally expensive than other models.