|  |
| --- |
| Defining the target label (y): |
| **Landmark detection**  To find the bounding box around the object in object detection problem we need to get the values for b\_x,b\_y,b\_h,b\_w  Those values are landmark in the image.  Like if we want to detect the facial expression along with detecting face we can mark important areas in the face and get the points in the image of the face.  Similar kind of approach is seen for posture detection systems.  The landmark should be consistent across the example labels. Means if l1\_x,l1\_y represent the corner of the left eye of the face it should be same for all labels. |
| **Object Detection**  We have seen object localization and landmark. Now let see the problem of object detection.  Let say we want to design a Car detection model. Then we start with a convnet and train it with some closely cropped images of car. Where for label 1 there are mostly the car image in the picture as given below. |
| Next, we use the sliding window method to pass a big image with several objects. Which will identify the objects and the locations of corresponding objects.  We typically start with a small window and pass the regions of the image sliding through it’s region. Then increase the window size and slide till end. Then increase again and do the same.  The idea of doing this is if we follow this approach then the model will detect the object in one window which will o/p probability as 1 and the location will be the window location.  But there is a huge downside of this method.   * Computationally poor as we need to crop the image several times and pass it to conv net * If we use small stride, then it will be too costly and if the stride is big there will be chance of bigger error   Why this algorithm was in use?  In earlier phase of computer vision when people use logistic regression or simpler machine learning algorithm which computes simpler function this algorithm was still useful and computationally not that poor.  But in case of convnet we are calculating huge number of parameters and deriving a complex function so that demand lots of computation and makes this algorithm very slow. |
| **How to solve sliding window issues in convnet?**  Instead of using sliding window technique, we can use the convolution to the entire image and get the same result. To do this we need to transform FC layers into convolutional layers which we can do as follows.  **Why we need to convert this into convolutional layer not clear**  **Let say,** we trained convnet with 14x14x3 image which gives final o/p as 1x1x400 conv.  Next in test set we have an image of size 16x16x3. So if we follow the sliding window approach then we can create 4 subset / cropped version of the picture and pass it through convnet and get 4 different o/p. But it comes out that this is not the best way to do. Instead of cropping the image we can pass the entire image which will finally give 2x2x400 o/p volume. Where each corner segment is ideally gives same o/p if we have chosen the separate subset of the image.  **There is one issue with this approach the bounding box might not be very accurate** |
| **Bounding box window problem with convolution implementation:**  It appears that with convolution implementation of the object detection sliding window algorithm is fast. However, it fails to predict the bounding box very accurately. One of the ways of getting accurate bounding box is using YOLO algorithm (You only look once)  In this algorithm we divide the image into grid and get the label o/p for each grid.  Where the o/p will generate the probability of the object if there is object centre in this grid and the bounding boxes.  Let us consider the below image of shape 100x100x3 as our input. In YOLO algorithm it will divide the image into 9 regions and gives o/p of size 8.  So the final o/p will be of size 3x3x8   * This is a convolution implementation, so it is very fast * 19x19 segmentation reduces the probability of having multiple objects in one segment hence improves accuracy further      * Bounding boxes are specified relative to grids. |
| **Intersection over union** |
| **Non max suppression** |
| **Anchor Object:**  So far, we see from each grid it is possible to predict one object. And with 19x19 grid the chance of having centre of 2 objects in same grid is very less.  But what if one grid contains more than one objects. In that case we can use there is one concept called Anchor boxes.  Where we define different Anchor boxes of different shape and size. The output shape is changed accordingly. Like in the below example we have chosen two anchor boxes.    Each object in training image is assigned to grid cell that contains object’s midpoint and anchor box for the grid cell with highest IoU.   * **This algorithm does not handle well if there is 3 objects in same grid cell** * **Also if the IOU of 2 objects same for 2 anchor objects in same cell** |
| **YOLO Algorithm**  **Training set**-    **Making Predictions**    Outputting the non-max suppressed outputs:   * For each grid call, get 2 predicted bounding boxes. * Get rid of low probability predictions. * For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions. |
| **Region proposal**  The motivation behind this algorithm is to do prediction for the regions of the image which make sense. In order to do so it runs segmentation algorithm to find some regions and then run the detection algorithm surrounding those regions instead of running on entire image. |
| **Face recognition**  Face Verification Vs. Face Recognition  Face Verification   * Given an input image , nameID * O/p is that the given image is of the given person or not 1:1 mapping problem   Face recognition   * Has a database of K person * Get an input image * o/p is one of the K person or unrecognized, 1:K mapping problem   Face recognition system implementation is harder than face verification. One of the reason is that it is a 1:K mapping instead of 1:1 mapping. So, if we achieve 99% accuracy in face verification system and want to use the same for face recognition. Then the performance would not be as expected because 1% error for K person will be huge for recognition. May be if we have 99.9999% accuracy in face verification that we can try to implement into face recognition. |
| **One Shot Learning**  One of the challenges in face recognition system is that you need to solve One-shot learning problem. Which means you will not get huge number of training images to learn and recognize the given image rather only one image.  Let’s understand this with and example.  Say we have 5 employees in our organization as below whose profile pic is stored in the company database. Next, we want to recognize one of them given one other picture or someone who does not belong to the organization.  one approach we can think of training a Deep CNN with the image we have in database and creating the final softmax layer with no of o/p equals no of employees in the organization.  But there are some big issues in this approach   * with that small dataset it is not possible to build a robust CNN model with high accuracy * what if new employee joins or someone leaves the organization, then we need to update the network architecture and train multiple times.   To solve this problem we can think of an approach of learning a similarity function instead of learning images.  **d(img1,img2) degree of difference between images**  **If d(img1,img2) ; then same person**  **; then diff person** |
| **Siamese network**  To find the similarity function we can compare pixel by pixel 2 images and find the similarity among them. But this is not a very good approach as the result depends on various factor of the image(lighting condition, color, pose etc)  So, instead of finding the similarity of the image itself, if we can represent the image in an encoding format that does not vary so much for one person that would be the good approach.  If we remove the last layer of softmax function of the CNN classification model and the last layer gives o/p 128 activations (encoding). We can use this network to encode 2 images and then compare the encoding of the 2 images to find the similarity.  **Goal of the learning:**   * Parameters of NN define an encoding   Learn parameters so that:   * If are the same person, is small. * If are different persons, is large. |
| Triplet loss  For an image 𝑥x, we denote its encoding 𝑓(𝑥), where 𝑓f is the function computed by the neural network.    Anchor  Negative  Anchorr  Positive  Training will use triplets of images (𝐴,𝑃,𝑁):   * A is an "Anchor" image--a picture of a person. * P is a "Positive" image--a picture of the same person as the Anchor image. * N is a "Negative" image--a picture of a different person than the Anchor image.   These triplets are picked from our training dataset. We will write (𝐴(𝑖),𝑃(𝑖),𝑁(𝑖)) to denote the 𝑖th training example.  You'd like to make sure that an image 𝐴(𝑖) of an individual is closer to the Positive 𝑃(𝑖) than to the Negative image 𝑁(𝑖) by at least a margin 𝛼:  ∣∣𝑓(𝐴(𝑖))−𝑓(𝑃(𝑖))∣∣22+𝛼<∣∣𝑓(𝐴(𝑖))−𝑓(𝑁(𝑖))∣∣22  You would thus like to minimize the following "triplet cost":  i.e Triplet loss = max ((d(A,P) – d(A,N) + alpha),0)  Notes:   * The term (1) is the squared distance between the anchor "A" and the positive "P" for a given triplet; you want this to be small. * The term (2) is the squared distance between the anchor "A" and the negative "N" for a given triplet, you want this to be relatively large, so it thus makes sense to have a minus sign preceding it. * 𝛼 is called the margin. It is a hyperparameter that you should pick manually.   Choosing the triplets A,P,N:  During training, if A,P,N are chosen randomly, is easily satisfied.  Choose triplets that’re “hard” to train on.  The goal of the learning algorithm would be to minimize the triplet cost function by reducing d(A,P) or by increasing d(A,N) |
| **Face verification and Binary classification**  The Siamese network that we have seen before, if we connect the encoding of the 2 CNN with one binary classification o/p. It would o/p as 1 if the 2 images are of same person. Else return 0  We can learn below optimization function to train the network |
| **Neural Style Transfer**  Neural Style Transfer is an area of deep learning where we are giving input 2 images to the model one is content image and another is style image. The model will generate the new image in the given style having similar kind of content as input content image.  Below are the example: |
| **What deep ConvNet is learning**  Let’s consider below convolution network. If we pick one unit in layer 1 and see which of the 9 image patch is activating that unit then we will see that specific unit gets triggered for specific features in the image.  In initial layers of deep CNN we would see that layers are capturing low level features edges/ patterns in a very small region of the source image. Where as in later layers we will see that it would detect high level features.    **Below are the visualization taken from different layers of a Deep ConvNet**    These are the nine patches that cause one hidden unit to be highly activated. And then each grouping, this is a different set of nine image patches that cause one hidden unit to be activated. So this visualization shows nine hidden units in layer 2, and for each of them shows nine image patches that causes that hidden unit to have a very large output. |
| **Cost Function**  In Neural style transfer we are measuring how good is the generated image given a content and a style image.  So, there are two components of the cost function. Cost w.r.t Content and Cost w.r.t style. The final cost is the weighted sum of the both.   * Build the content cost function 𝐽𝑐𝑜𝑛𝑡𝑒𝑛𝑡(𝐶,𝐺) * Build the style cost function 𝐽𝑠𝑡𝑦𝑙𝑒(𝑆,𝐺) * Put it together to get 𝐽(𝐺)=𝛼𝐽𝑐𝑜𝑛𝑡𝑒𝑛𝑡(𝐶,𝐺)+𝛽𝐽𝑠𝑡𝑦𝑙𝑒(𝑆,𝐺)   alpha and beta are the hyper parameters of the model   * We will initialize generated image G as arbitrary value * Then based on the cost function we will run the gradient descent to minimize the cost. * Then update the G based on gradient. In this case we are updating the pixel values to have similar style and content in the generated image. |
| **Content Cost function:**  The first part of the cost function is the content cost function   * **Say you use hidden layer to compute content cost.** * **Use pre-trained ConvNet. (E.g., VGG network)** * **Let and be the activation of layer on the images** * **If and are similar, both images have similar content**     **Style Cost**   * Style is the measure of correlation of different layers of a particular layer in ConvNet     Let say we see the red channels which is responsible of detecting vertical edges and the yellow channel is responsible for detecting orange region. Then the style is the measure of how often they occur or does not occur at same time.  We calculate correlation by gram matrix. |