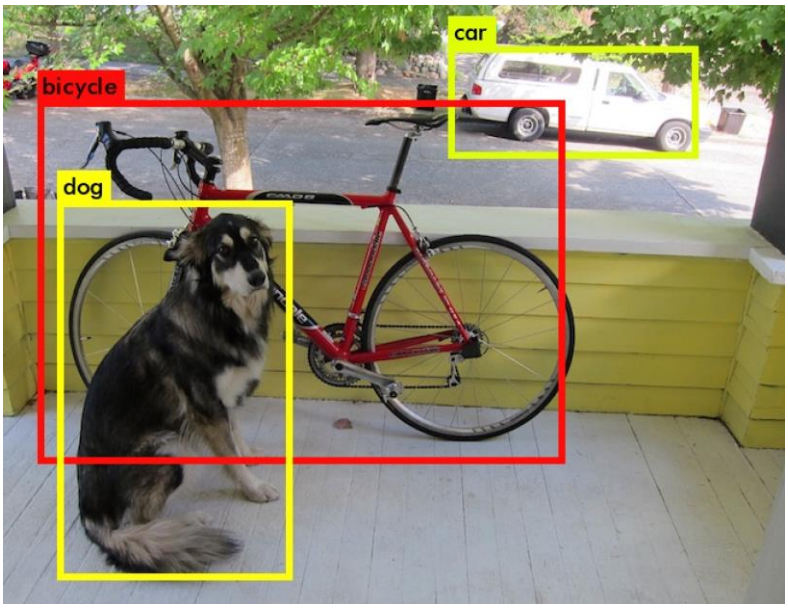
Object detection

Given an image we want to detect all the object in the image that belong to a specific classes and give their location. An image can contain more than one object with different classes.



Method 1-Semantic Segmentation:

- We want to Label each pixel in the image with a category label. It detects no objects just pixels.
- If there are two objects of the same class is intersected, we won't be able to separate them.

Fig 4. Semantic Segmentation

Method 2-Instance Segmentation

We want to know which pixel label but also distinguish them.

Fig 5. Instance Segmentation

To make classification with localization we use a Conv Net with a softmax attached to the end of it and a four numbers bx, by, bh, and bw to tell you the location of the class in the image. The dataset should contain this four numbers with the class too.

Defining the target label Y in classification with localization problem:

$$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ . \\ . \\ c_n \end{bmatrix} = \begin{bmatrix} object\ is\ presented? \\ x - \ center\ of\ the\ object \\ y - center\ of\ the\ object \\ b_h - box\ height \\ b_w - box\ width \\ c_1 - is\ it\ object\ c1? \\ c_2 - is\ it\ object\ c2? \\ . \\ . \\ c_n - is\ it\ object\ cn? \end{bmatrix}$$

No object example: $\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ . \\ . \\ ? \end{bmatrix}$

The loss function for the Y we have created (Example of the square error):

```
L(y',y) = (y1'-y1)^2 + (y2'-y2)^2 + ...          if y1 = 1
L(y',y) = (y1'-y1)^2                             if y1 = 0
```

In practice we use **logistic regression for** `pc`, log **likely hood loss for classes**, and **squared error for the bounding box**.

Landmark Detection

In some of the computer vision problems you will need to output some points. That is called **landmark detection**.

For example, if you are working in a face recognition problem you might want some points on the face like corners of the eyes, corners of the mouth, and corners of the nose and so on. This can help in a lot of application like detecting the pose of the face.

$$Y = \begin{bmatrix} p_c \\ point_1 \\ . \\ . \\ . \\ point_n \end{bmatrix}$$

How to find the box shape and size and location in Object Detection:

Sliding windows detection algorithm:

Train a network to identify cars.

i. Decide a rectangle size.
ii. Split your image into rectangles of the size you picked. Each region should be covered. You can use some strides.
iii. For each rectangle feed the image into the Conv net and decide if its a car or not.
iv. Pick larger/smaller rectangles and repeat the process from 2 to 3.
v. Store the rectangles that contains the cars.
vi. If two or more rectangles intersects choose the rectangle with the best accuracy.

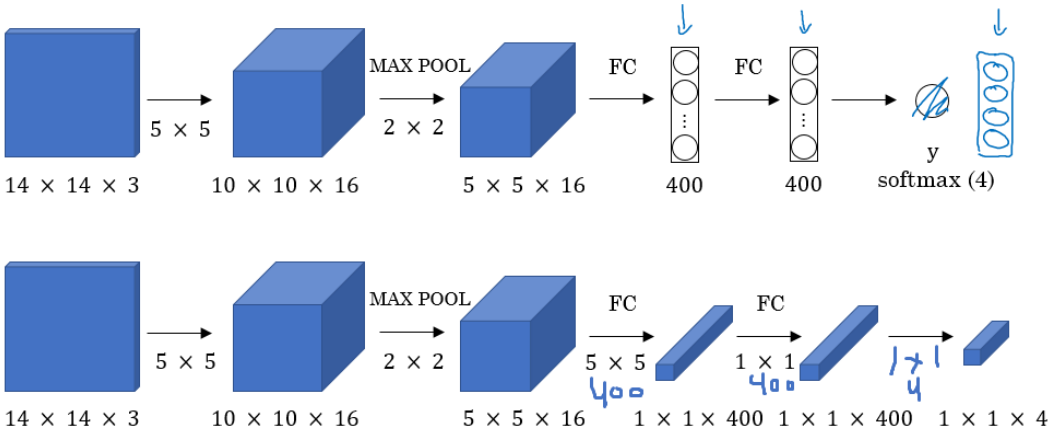
Disadvantage: computation time.

Solution: To solve this problem, we can implement the sliding windows with a ***Convolutional approach***.

Convolutional Implementation of Sliding Windows

Turning FC layer into convolutional layers (predict image class from four classes).

Remainder: 1X1 filter work similar to FC layer on the input depth.

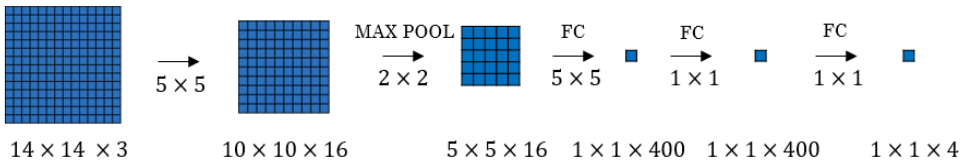


Convolution implementation of sliding windows:

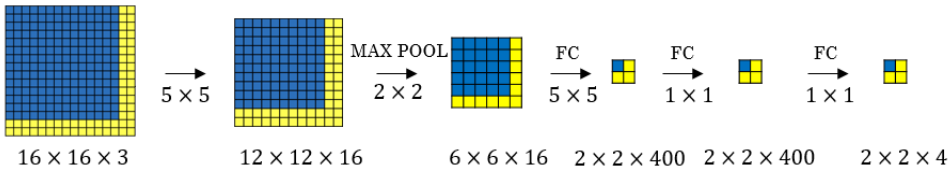
[Sermanet et al., 2014, OverFeat: Integrated recognition, localization and detection using convolutional networks]

Say now we have a 16 x 16 x 3 image that we need to apply the sliding windows in. By the normal implementation that have been mentioned in the section before this, we would run this Conv net four times each rectangle size will be 16 x 16. The convolution implementation will run once.

Instead of running it 4 times:



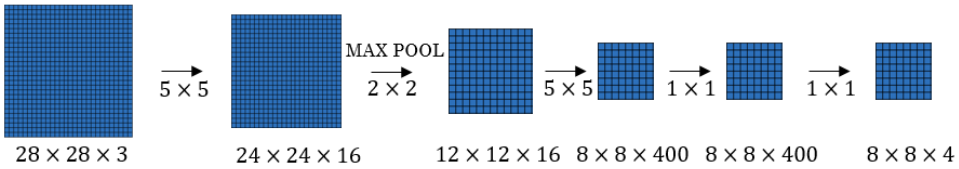
Use the convolution implementation and run it once:



- Simply we have feed the image into the same Conv net we have trained.
- The left cell of the result "The blue one" will represent the the first sliding window of the normal implementation. The other cells will represent the others.
- Its more efficient because it now shares the computations of the four times needed.

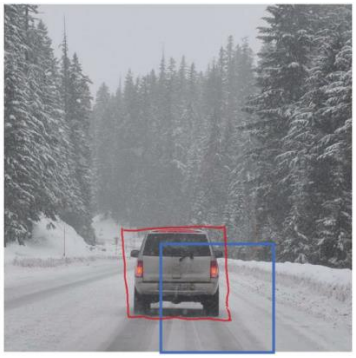
Another example would be:

This example has a total of 16 sliding windows that shares the computation together.



Algorithm weaknes:

The weakness of the algorithm is that the position of the rectangle wont be so accurate. Maybe none of the rectangles is exactly on the object you want to recognize.



Yulo Algorithm:

YOLO algorithm

Lets say we have an image of 100 X 100

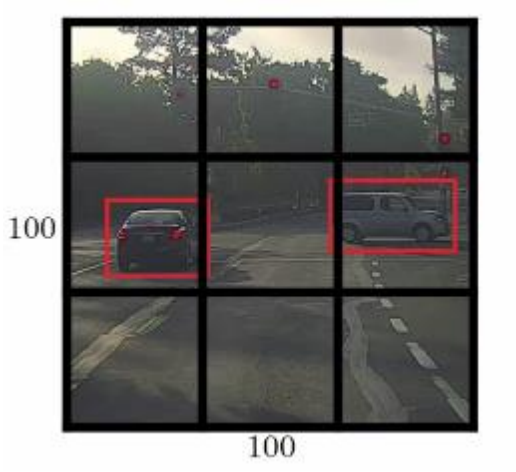
i. Place a 3 x 3 grid on the image. For more smother results you should use 19 x 19 for the 100 x 100
ii. Apply the classification and localization algorithm we discussed in a previous section to each section of the grid. $b_x$and b_y will represent the center point of the object in each grid and will be relative to the box so the range is between 0 and 1 while b_h and b_w will represent the height and width of the object which can be greater than 1.0 but still a floating point value.
iii. Do everything at once with the convolution sliding window. If Y shape is 1 x 8 as we discussed before then the output of the 100 x 100 image should be 3 x 3 x 8 which corresponds to 9 cell results.
iv. Merging the results using predicted localization mid point.

Yolo different then Object detectors:

- YOLO uses a single CNN network for both classification and localizing the object using bounding boxes.

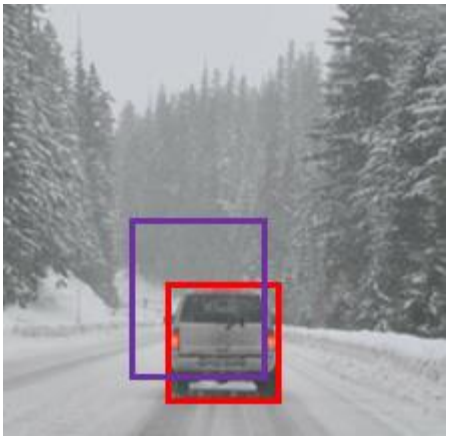
Disadvantage:

- We have a problem if we have found more than one object in one grid box.



Intersection Over Union

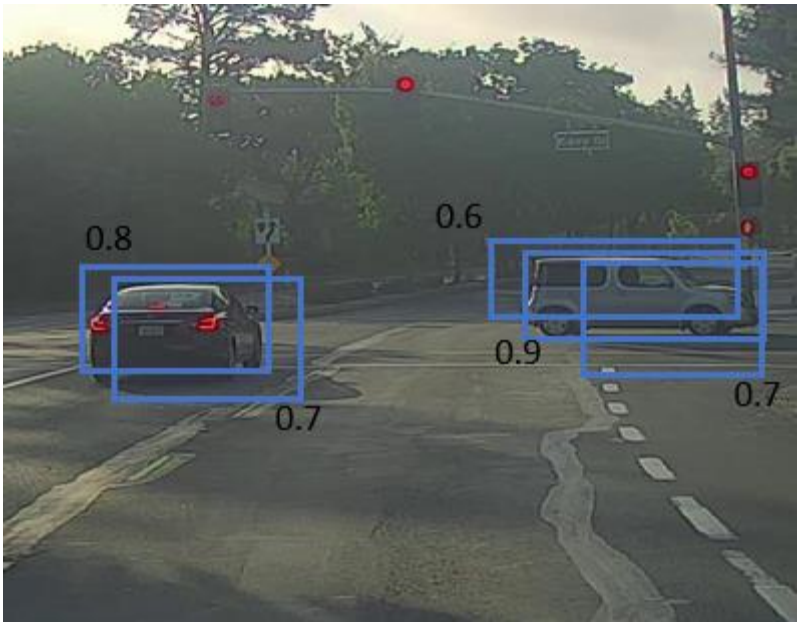
We want to differ the red box over the purple.



Solution:

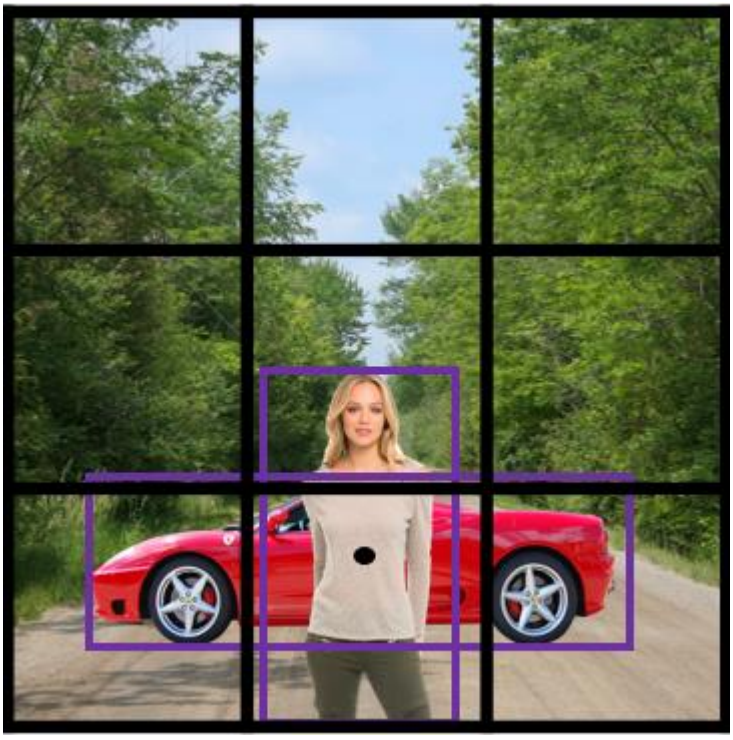
1. Comute the IAU=$\frac{A \cap B}{A \cup B}$
2. If IAU>a we will save only the rectangle Pc(the prediction if there is an object in the picture) is larger. In our case the red box would have a bigger Pc keeping only him.

Example: the numbers are Pc.



Anchor boxes -overlapping objects detection

In YOLO, a grid only detects one object. What if a grid cell wants to detect multiple object when the center of the object is in the same pixel?



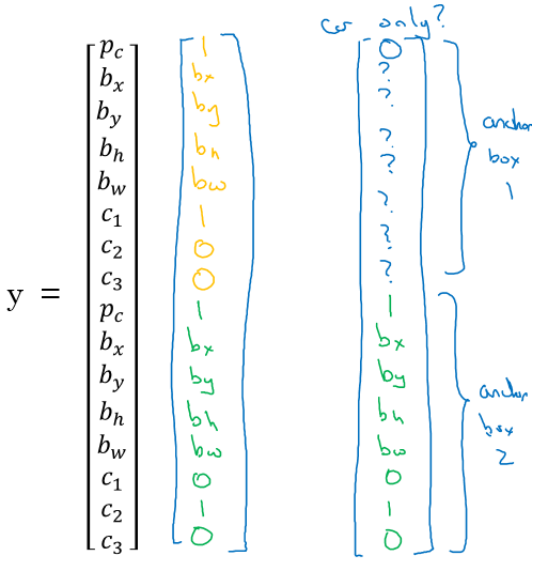
Solution:

$$\text{Before : Y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ . \\ . \\ c_n \end{bmatrix} = \begin{bmatrix} object\ is\ presented? \\ x - \ center\ of\ the\ object \\ y - center\ of\ the\ object \\ b_h - box\ height \\ b_w - box\ width \\ c_1 - is\ it\ object\ c1? \\ c_2 - is\ it\ object\ c2? \\ . \\ . \\ c_n - is\ it\ object\ cn? \end{bmatrix}$$

now : Y_1, Y_2 .... The number of object we want to detect in each pixel.

Anchor box 1: Anchor box 2:

Example of data:



Where the car was near the anchor 2 than anchor 1?

In order to be able to check Y-Y_hat we need some convention on the anchor box order. One way is to choose some kinds of anchor boxes like long, wide, rectangle and check every object which iau is bigger for those types.

YOLO Algorithm-summary

YOLO is a state-of-the-art object detection model that is fast and accurate

Example:

Suppose we need to do object detection for our autonomous driver system. It needs to identify three classes:

i. Pedestrian (Walks on ground).

ii. Car.
iii. Motorcycle.

We decided to choose two anchor boxes, a taller one and a wide one.

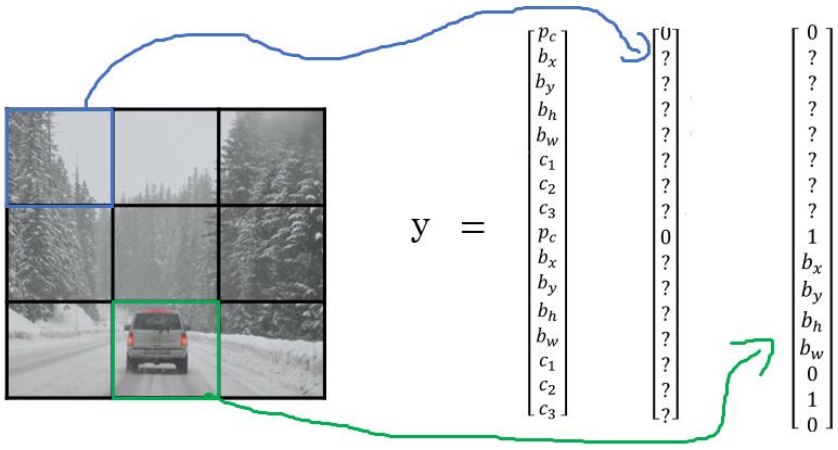
- Like we said in practice they use five or more anchor boxes hand made or generated using k-means.

Our labeled Y shape will be `[Ny, HeightOfGrid, WidthOfGrid, 16]`, where Ny is number of instances and each row (of size 16) is as follows:

```
Ny =[Pc, bx, by, bh, bw, c1, c2, c3, Pc, bx, by, bh, bw, c1, c2, c3]
```

- Your dataset could be an image with a multiple labels and a rectangle for each label, we should go to your dataset and make the shape and values of Y like we agreed.

An example:

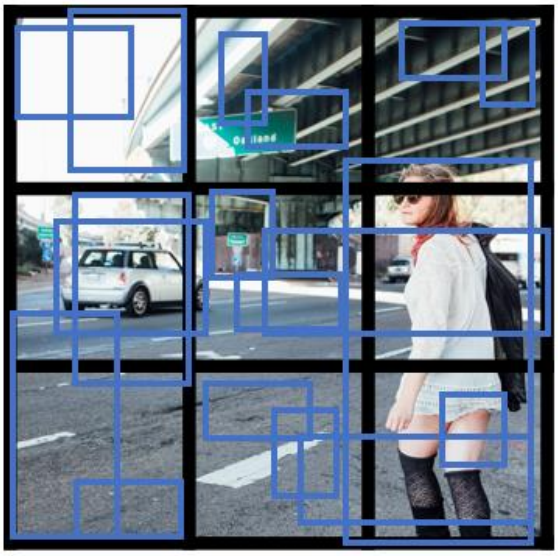


- We first initialize all of them to zeros and ?, then for each label and rectangle choose its closest grid point then the shape to fill it and then the best anchor point based on the IOU. so that the shape of Y for one image should be `[HeightOfGrid, WidthOfGrid,16]`

- Train the labeled images on a Conv net. you should receive an output of `[HeightOfGrid, WidthOfGrid,16]` for our case.

- To make predictions, run the Conv net on an image and run Non-max suppression algorithm for each class you have in our case there are 3 classes.

You could get something like that:



Total number of generated boxes are grid_width * grid_height * no_of_anchors = 3 x 3 x 2

By removing the low probability predictions, you should have:

Then get the best probability followed by the IOU filtering:

- 

YOLO problem: not good at detecting smaller object.

YOLO implementations

- https://github.com/allanzelener/YAD2K
- https://github.com/thtrieu/darkflow
- https://pjreddie.com/darknet/yolo/

Better then Yolo:

YOLO9000 Better, faster, stronger

Summary:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 608, 608, 3)	0	
conv2d_1 (Conv2D)	(None, 608, 608, 32)	864	input_1[0][0]
batch_normalization_1 (BatchNorm	(None, 608, 608, 32)	128	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 608, 608, 32)	0	batch_normalization_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 304, 304, 32)	0	leaky_re_lu_1[0][0]
conv2d_2 (Conv2D)	(None, 304, 304, 64)	18432	max_pooling2d_1[0][0]
batch_normalization_2 (BatchNorm	(None, 304, 304, 64)	256	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 304, 304, 64)	0	batch_normalization_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 152, 152, 64)	0	leaky_re_lu_2[0][0]
conv2d_3 (Conv2D)	(None, 152, 152, 128)	73728	max_pooling2d_2[0][0]

```
batch_normalization_3 (BatchNorm (None, 152, 152, 128) 512         conv2d_3[0][0]
leaky_re_lu_3 (LeakyReLU)        (None, 152, 152, 128) 0     batch_normalization_3[0][0]
conv2d_4 (Conv2D)                (None, 152, 152, 64)  8192        leaky_re_lu_3[0][0]
batch_normalization_4 (BatchNorm (None, 152, 152, 64)  256         conv2d_4[0][0]
leaky_re_lu_4 (LeakyReLU)        (None, 152, 152, 64)  0     batch_normalization_4[0][0]
conv2d_5 (Conv2D)                (None, 152, 152, 128) 73728       leaky_re_lu_4[0][0]
batch_normalization_5 (BatchNorm (None, 152, 152, 128) 512         conv2d_5[0][0]
leaky_re_lu_5 (LeakyReLU)        (None, 152, 152, 128) 0     batch_normalization_5[0][0]
max_pooling2d_3 (MaxPooling2D)   (None, 76, 76, 128)   0           leaky_re_lu_5[0][0]
conv2d_6 (Conv2D)                (None, 76, 76, 256)   294912      max_pooling2d_3[0][0]
batch_normalization_6 (BatchNorm (None, 76, 76, 256)   1024        conv2d_6[0][0]
leaky_re_lu_6 (LeakyReLU)        (None, 76, 76, 256)   0     batch_normalization_6[0][0]
conv2d_7 (Conv2D)                (None, 76, 76, 128)   32768       leaky_re_lu_6[0][0]
batch_normalization_7 (BatchNorm (None, 76, 76, 128)   512         conv2d_7[0][0]
leaky_re_lu_7 (LeakyReLU)        (None, 76, 76, 128)   0     batch_normalization_7[0][0]
conv2d_8 (Conv2D)                (None, 76, 76, 256)   294912      leaky_re_lu_7[0][0]
batch_normalization_8 (BatchNorm (None, 76, 76, 256)   1024        conv2d_8[0][0]
leaky_re_lu_8 (LeakyReLU)        (None, 76, 76, 256)   0     batch_normalization_8[0][0]
max_pooling2d_4 (MaxPooling2D)   (None, 38, 38, 256)   0           leaky_re_lu_8[0][0]
conv2d_9 (Conv2D)                (None, 38, 38, 512)   1179648     max_pooling2d_4[0][0]
batch_normalization_9 (BatchNorm (None, 38, 38, 512)   2048        conv2d_9[0][0]
leaky_re_lu_9 (LeakyReLU)        (None, 38, 38, 512)   0     batch_normalization_9[0][0]
conv2d_10 (Conv2D)               (None, 38, 38, 256)   131072      leaky_re_lu_9[0][0]
batch_normalization_10 (BatchNor (None, 38, 38, 256)   1024        conv2d_10[0][0]
leaky_re_lu_10 (LeakyReLU)       (None, 38, 38, 256)   0    batch_normalization_10[0][0]
conv2d_11 (Conv2D)               (None, 38, 38, 512)   1179648    leaky_re_lu_10[0][0]
batch_normalization_11 (BatchNor (None, 38, 38, 512)   2048        conv2d_11[0][0]
leaky_re_lu_11 (LeakyReLU)       (None, 38, 38, 512)   0    batch_normalization_11[0][0]
conv2d_12 (Conv2D)               (None, 38, 38, 256)   131072      leaky_re_lu_11[0][0]
batch_normalization_12 (BatchNor (None, 38, 38, 256)   1024        conv2d_12[0][0]
leaky_re_lu_12 (LeakyReLU)       (None, 38, 38, 256)   0   batch_normalization_12[0][0]
conv2d_13 (Conv2D)               (None, 38, 38, 512)   1179648     leaky_re_lu_12[0][0]
batch_normalization_13 (BatchNor (None, 38, 38, 512)   2048        conv2d_13[0][0]
leaky_re_lu_13 (LeakyReLU)       (None, 38, 38, 512)   0    batch_normalization_13[0][0]
max_pooling2d_5 (MaxPooling2D)   (None, 19, 19, 512)   0           leaky_re_lu_13[0][0]
conv2d_14 (Conv2D)               (None, 19, 19, 1024)  4718592     max_pooling2d_5[0][0]
batch_normalization_14 (BatchNor (None, 19, 19, 1024)  4096        conv2d_14[0][0]
leaky_re_lu_14 (LeakyReLU)       (None, 19, 19, 1024)  0    batch_normalization_14[0][0]
conv2d_15 (Conv2D)               (None, 19, 19, 512)   524288      leaky_re_lu_14[0][0]
batch_normalization_15 (BatchNor (None, 19, 19, 512)   2048        conv2d_15[0][0]
leaky_re_lu_15 (LeakyReLU)       (None, 19, 19, 512)   0    batch_normalization_15[0][0]
conv2d_16 (Conv2D)               (None, 19, 19, 1024)  4718592     leaky_re_lu_15[0][0]
batch_normalization_16 (BatchNor (None, 19, 19, 1024)  4096        conv2d_16[0][0]
leaky_re_lu_16 (LeakyReLU)       (None, 19, 19, 1024)  0    batch_normalization_16[0][0]
conv2d_17 (Conv2D)               (None, 19, 19, 512)   524288      leaky_re_lu_16[0][0]
batch_normalization_17 (BatchNor (None, 19, 19, 512)   2048        conv2d_17[0][0]
leaky_re_lu_17 (LeakyReLU)       (None, 19, 19, 512)   0    batch_normalization_17[0][0]
conv2d_18 (Conv2D)               (None, 19, 19, 1024)  4718592     leaky_re_lu_17[0][0]
batch_normalization_18 (BatchNor (None, 19, 19, 1024)  4096        conv2d_18[0][0]
leaky_re_lu_18 (LeakyReLU)       (None, 19, 19, 1024)  0    batch_normalization_18[0][0]
conv2d_19 (Conv2D)               (None, 19, 19, 1024)  9437184     leaky_re_lu_18[0][0]
batch_normalization_19 (BatchNor (None, 19, 19, 1024)  4096        conv2d_19[0][0]
conv2d_21 (Conv2D)               (None, 38, 38, 64)    32768       leaky_re_lu_13[0][0]
leaky_re_lu_19 (LeakyReLU)       (None, 19, 19, 1024)  0    batch_normalization_19[0][0]
batch_normalization_21 (BatchNor (None, 38, 38, 64)    256         conv2d_21[0][0]
```

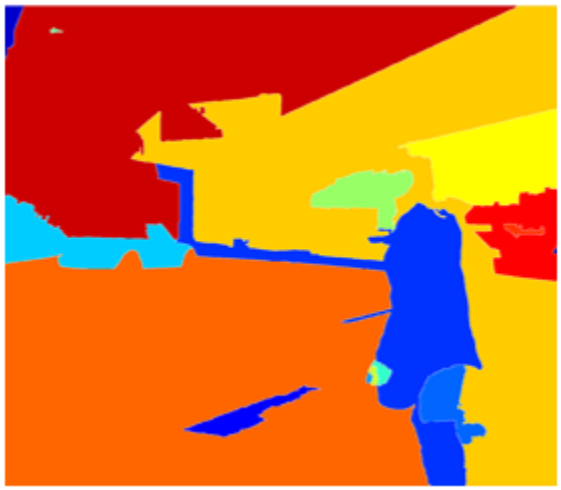
```
conv2d_20 (Conv2D)              (None, 19, 19, 1024)  9437184     leaky_re_lu_19[0][0]
leaky_re_lu_21 (LeakyReLU)      (None, 38, 38, 64)    0     batch_normalization_21[0][0]
batch_normalization_20 (BatchNor (None, 19, 19, 1024)  4096        conv2d_20[0][0]
space_to_depth_x2 (Lambda)      (None, 19, 19, 256)   0           leaky_re_lu_21[0][0]
leaky_re_lu_20 (LeakyReLU)      (None, 19, 19, 1024)  0     batch_normalization_20[0][0]
concatenate_1 (Concatenate)     (None, 19, 19, 1280)  0         space_to_depth_x2[0][0]
                                                               leaky_re_lu_20[0][0]
conv2d_22 (Conv2D)              (None, 19, 19, 1024)  11796480    concatenate_1[0][0]
batch_normalization_22 (BatchNor (None, 19, 19, 1024)  4096        conv2d_22[0][0]
leaky_re_lu_22 (LeakyReLU)      (None, 19, 19, 1024)  0     batch_normalization_22[0][0]
conv2d_23 (Conv2D)              (None, 19, 19, 425)   435625      leaky_re_lu_22[0][0]
==================================================================================================
Total params: 50,983,561
Trainable params: 50,962,889
Non-trainable params: 20,672
__________________________________________________________________________________________________
```

Other objects detection Algorithm (R-CNN ...)

Downsides of YOLO process a lot of areas where no objects are present. The author of YOLO claims that this algorithm is 1000x faster than R-CNN and 100x faster than Fast R-CNN. See our paper for more details on the full system.

R-CNN stands for regions with Conv Nets.

- R-CNN tries to pick a few windows and run a Conv net (your confident classifier) on top of them.
- The algorithm R-CNN uses to pick windows is called a segmentation algorithm. Outputs something like this:



- If for example the segmentation algorithm produces 2000 blob then we should run our classifier/CNN on top of these blobs.

There has been a lot of work regarding R-CNN tries to make it faster:

    - R-CNN:
        - Propose regions. Classify proposed regions one at a time. Output label + bounding box.
        - Downside is that its slow.
        - [Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]
    - Fast R-CNN:
        - Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions.
        - [Girshik, 2015. Fast R-CNN]
    - Faster R-CNN:
        - Use convolutional network to propose regions.
        - [Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks]
    - Mask R-CNN:
        - https://arxiv.org/abs/1703.06870

- Most of the implementation of faster R-CNN are still slower than YOLO.
- Andew Ng thinks that the idea behind YOLO is better than R-CNN because you are able to do all the things in just one time instead of two times.
- Other algorithms that uses one shot to get the output includes **SSD** and **MultiBox**.
    - [Wei Liu, et. al 2015 SSD: Single Shot MultiBox Detector]
- **R-FCN** is similar to Faster R-CNN but more efficient.
    - [Jifeng Dai, et. al 2016 R-FCN: Object Detection via Region-based Fully Convolutional Networks]

Face recognition & Neural style transfer

Face Recognition

What is face recognition?

Face recognition system identifies a person's face. It can work on both images or videos.

Face verification vs. face recognition:

Verification (is this the claimed person?):

- Input: image, name/ID.
- Output: is this the claimed person? Yes/NO

Example:

A party you need show your face and write your ID. The system find the picture from a database using the ID and compare it to the face.

Recognition (who is this person?)

- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons (or not recognized)

Example:

Entering gate to a secret facility. The camera takes a picture of you and check if this picture is similar to any person in the database images.

We can use a face verification system to make a face recognition system. The accuracy of the verification system has to be high (around 99.9% or more) to be use accurately within a recognition system because the recognition system accuracy will be less than the verification system given K persons.

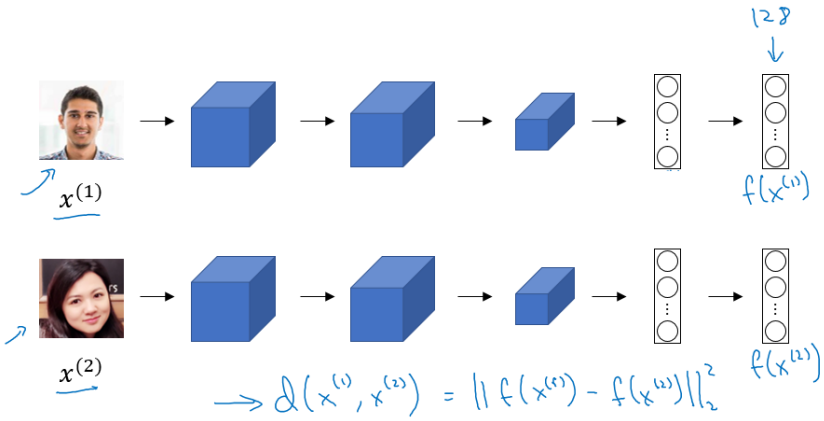
face recognition -One Shot Learning problem

- One Shot Learning: A recognition system is able to recognize a person, learning from one image. This is one of the face recognition challenges.
- Instead to make this work, we will learn a **similarity function**:
  - d(**img1**, **img2**) = degree of difference between images.
  - We want d result to be low in case of the same faces.
  - If d(**img1**, **img2**) <= T(T some number) Then the faces are the same.
- Similarity function helps us solving the one shot learning. Also its robust to new inputs.

Siamese Network

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

- We will implement the similarity function using a type of NNs called Siamease Network in which we can pass multiple inputs to the two or more networks with the same architecture and parameters.
- Siamese network architecture are as the following:



  - We make 2 identical conv nets which encodes an input image into a vector. In the above image the vector shape is (128,)
  - The loss function will be `d(x1, x2) = || f(x1) - f(x2) ||^2`
  - If x1, x2 are the same person, we want d to be low. If they are different persons, we want d to be high.

Triplet Loss

[Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering]

We need to define new Loss function that will have high loss for positive matching and low loss negative matching.

Our learning objective in the triplet loss function is to get the distance between an **Anchor** image and a **positive** or a **negative** image.

Formally we want:

Positive distance to be less than negative distance:

- `||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 <= 0`

To make sure the NN won't get an output of zeros easily:

- `||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + alpha <= 0`

Then

Final Loss function Given 3 images (A, P, N):

- `L(A, P, N) = max (||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + alpha , 0)`
- `J = Sum(L(A[i], P[i], N[i]) , i)` for all triplets of images.

Notes:

- You need multiple images of the same person in your dataset. Then get some triplets out of your dataset. Dataset should be big enough.
- Choosing the triplets A, P, N:
  - During training if A, P, N are chosen randomly (Subjet to A and P are the same and A and N aren't the same) then one of the problems this constrain is easily satisfied
    - `d(A, P) + alpha <= d (A, N)`
    - So the NN wont learn much
  - What we want to do is choose triplets that are **hard** to train on.
    - So for all the triplets we want this to be satisfied:
    - `d(A, P) + alpha <= d (A, N)`

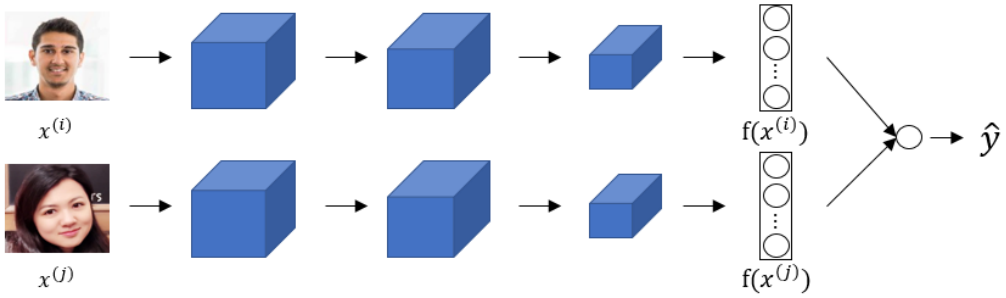
        - This can be achieved by for example same poses!
        - Find more at the paper.
- Commercial recognition systems are trained on a large datasets like 10/100 million images.
- There are a lot of pretrained models and parameters online for face recognition.

Face Verification with Binary Classification:

[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Triplet loss is one way to learn the parameters of a conv net for face recognition there's another way to learn these parameters as a straight binary classification problem.

Learning the similarity function another way:



Notes:

- The final layer is a sigmoid layer.
- `Y' = wi * Sigmoid ( f(x(i)) - f(x(j)) ) + b` where the subtraction is the Manhattan distance between f(x(i)) and f(x(j))
- Some other similarities can be Euclidean and Ki square similarity.
- The NN here is Siamese means the top and bottom convs has the same parameters.

A good performance/deployment trick:

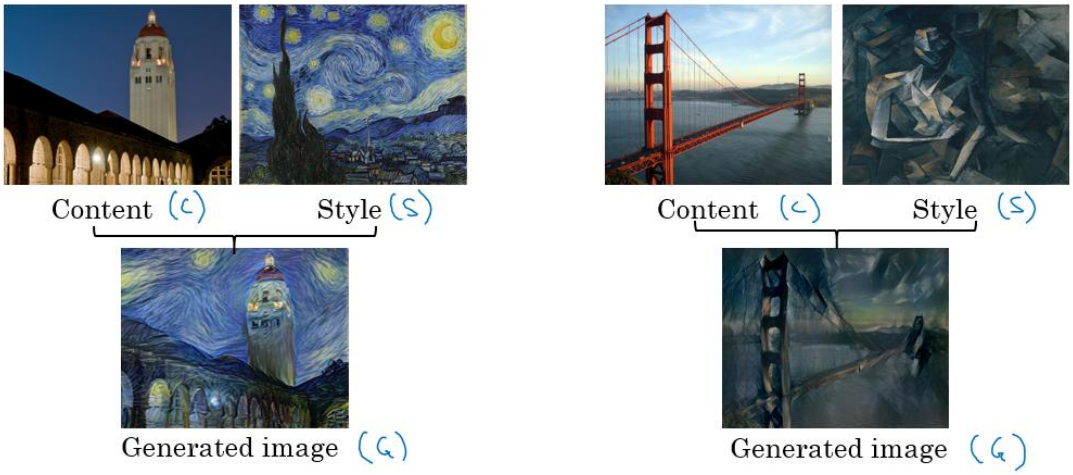
    - Pre-compute all the images that you are using as a comparison to the vector f(x(j))

    - When a new image that needs to be compared, get its vector f(x(i)) then put it with all the pre computed vectors and pass it to the sigmoid function.
- This version works quite as well as the triplet loss function.
- Available implementations for face recognition using deep learning includes:
    - Openface
    - FaceNet
    - DeepFace

Neural Style Transfer

What is neural style transfer?

Neural style transfer takes a content image C and a style image S and generates the content image G with the style of style image.



In order to implement this you need to look at the features extracted by the Conv net at the shallower (the style) and deeper layers(the content).

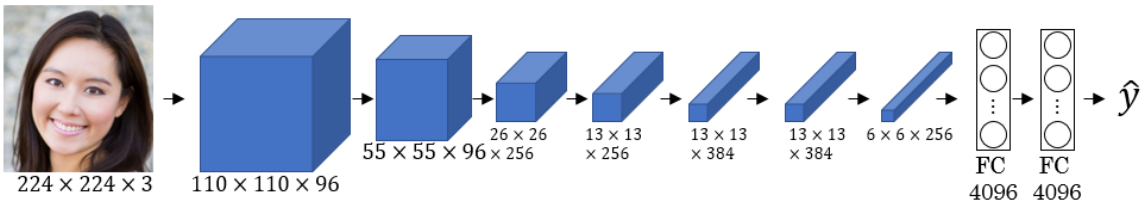
It uses a previously trained convolutional network like VGG, and builds on top of that.

What are deep ConvNets learning?

[Zeiler and Fergus., 2013, Visualizing and understanding convolutional networks]

Visualizing what a deep network is learning:

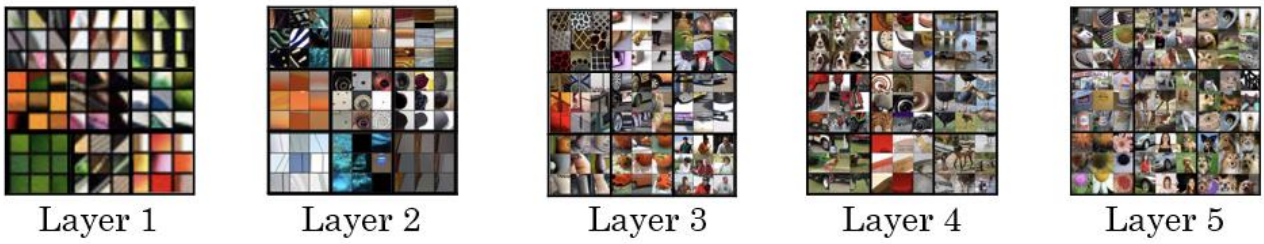
Given this AlexNet like Conv net:



How do you visualize the Net?

- Pick a unit in layer l. Find the images that maximize the unit's activation.
  - Notice that a hidden unit in layer one will see relatively small portion of NN, so if you plotted it will match a small image in the shallower layers while it will get larger image in deeper layers.
- Repeat for other units and layers.

It turns out that layer 1 are learning the low level representations like colors and edges. You will find out that each layer are learning more complex representations.



- The first layer was created using the weights of the first layer. Other images are generated using the receptive field in the image that triggered the neuron to be max.

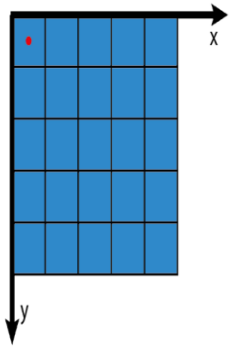
A good explanation on how to get **receptive field** given a layer:

n : number of features
r : receptive field size
j : jump (distance between two consecutive features)
$start$: center coordinate of the first feature

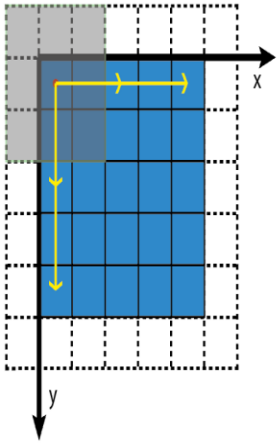
k : convolution kernel size
p : convolution padding size
s : convolution stride size

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$
$$j_{out} = j_{in} * s$$
$$r_{out} = r_{in} + (k-1) * j_{in}$$
$$start_{out} = start_{in} + \left(\frac{k-1}{2} - p\right) * j_{in}$$

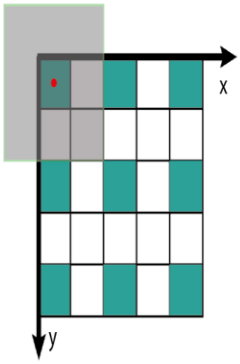
Layer 0: $n_0 = 5; r_0 = 1; j_0 = 1;$ $start_0 = 0.5$



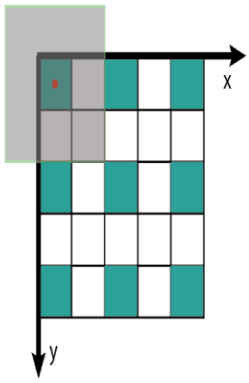
Conv1: $k_1 = 3; p_1 = 1; s_1 = 2$



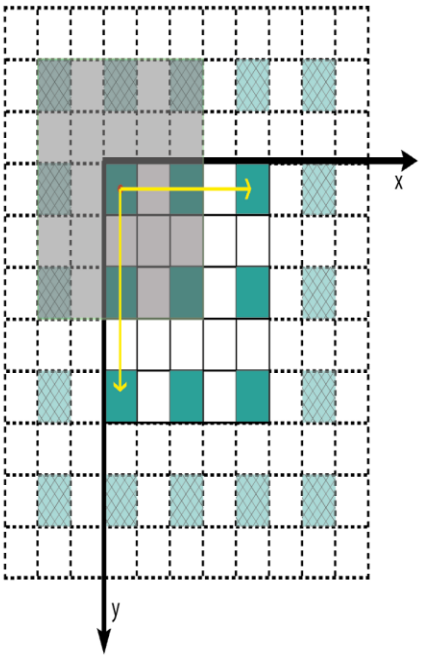
Layer 1: $n_1 = 3; r_1 = 3; j_1 = 2;$ $start_1 = 0.5$



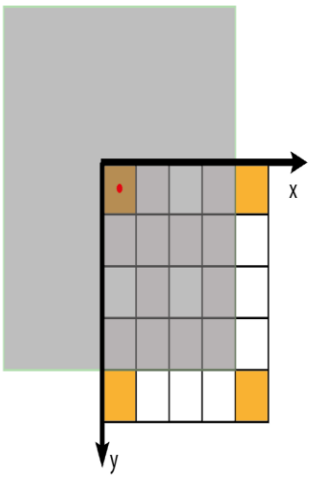
Layer 1: $n_1 = 3; r_1 = 3; j_1 = 2;$ $start_1 = 0.5$



Conv2: $k_2 = 3; p_2 = 1; s_2 = 2$



Layer 2: $n_3 = 2; r_3 = 7; j_3 = 4;$ $start_1 = 0.5$



From [A guide to receptive field arithmetic for Convolutional Neural Networks]

Cost Function for Style Transfer

We will define a cost function for the generated image that measures how good it is

Give a content image C, a style image S, and a generated image G:

- `J(G) = alpha * J(C,G) + beta * J(S,G)`
    - `J(C, G)` measures how similar is the generated image to the Content image.
    - `J(S, G)` measures how similar is the generated image to the Style image.
    - alpha and beta are relative weighting to the similarity and these are hyperparameters.

Find the generated image G:

1. Initiate G randomly For example G: 100 X 100 X 3
2. Use gradient descent to minimize `J(G)`
    - `G = G - dG` We compute the gradient image and use gradient decent to minimize the cost function.
- The iterations might be as following image:

To combine this images :


The back propagation will look like this:




Content Cost Function

We want to use the lower hidden layers which does not contain the information about the style to compare our content image and G. if we choose the layer to be to low we will force the network to get similar output. In practe [l] is not too shallow and not too deep.

Practice:

- Use pre-trained ConvNet. (E.g., VGG network)
- Let `a(c)[1]` and `a(G)[1]` be the activation of layer `1` on the images.
- If `a(c)[1]` and `a(G)[1]` are similar then they will have the same content
    - `J(C, G) at a layer l = 1/2 || a(c)[l] - a(G)[l] ||^2`

Style Cost Function

The Definition of style is the correlation between **activations** across **channels**. The first layers are responsible for lines detections. Each channel detects different lines in the picture. Correlation between channels indicates the style. Correlated means if a value appeared in a specific channel a specific value will appear too.

Practice:

Style matrix (Gram matrix):

$$G_{Kk'} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} a_{ijk}\, a_{ijk'}$$

$G_{Kk'}$ is a-$n_c[l], n_c[l]$ matrix. Each member of the matrix is the correlation between k channel and k' channel.

Next step is to calculate this $G_{Kk'}$ for the style picture(s) and for our generated picture (G).

Finally the cost function will be as following:

- `j(S, G) at layer l = (1/ 2 * H * W * C) || G(l)(s) - G(l)(G) ||`

Usually we use multilayer for styling and average on them:

- `J(S, G) = Sum (lamda[l]*j(S, G)[l], for all layers)`

Steps to be made if you want to create a tensorflow model for neural style transfer:

i. Create an Interactive Session.
ii. Load the content image.
iii. Load the style image
iv. Randomly initialize the image to be generated
v. Load the VGG16 model
vi. Build the TensorFlow graph:
   - Run the content image through the VGG16 model and compute the content cost
   - Run the style image through the VGG16 model and compute the style cost
   - Compute the total cost
   - Define the optimizer and the learning rate
vii. Initialize the TensorFlow graph and run it for a large number of iterations, updating the generated image at every step.