Course-3

→ Orthogonalization :
↳ "Things specific to a certain problem or a fun?"

↳ Chains of assumptions in ML =
1) Fit training set well on J (cost fun?).
↓
2) Fit dev set well on J. [Regularisation]
↓
3) Fit test set well on J
↳ Performs well in real world.

→ F_1 score → "Average" of Precision P & recall R
↓
$\left(\frac{2}{\frac{1}{P}+\frac{1}{R}}\right)$ → harmonic mean :)!!

Dev set + Single number evaluation matrix
↳ better ≈
(speeds up cycle)

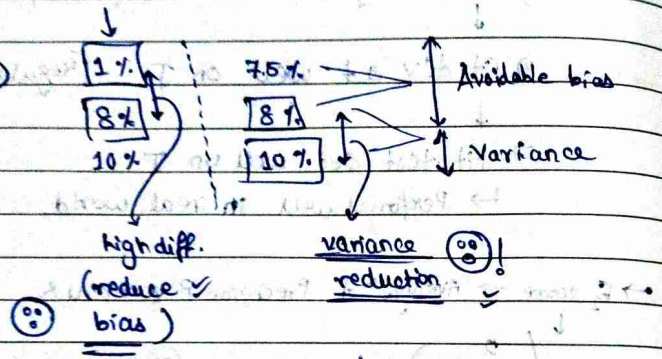
→ If there are multiple classifiers, average matrix is to be generated having lower avg. (error).
→ Dev set and test set must be from distribution.
→ If we have a million examples, → 98% → training
→ 1% → dev
→ 1% → test

→ Set your test set to be big enough to given high confidence in the overall performance of ur system.
↳ Sometimes train + Dev (no test) is fine ✓ [test set becomes dev set]

→ If doing well on your metric + dev/test set doesn't correspond to doing well on your application, change your metric and/or dev/test set.

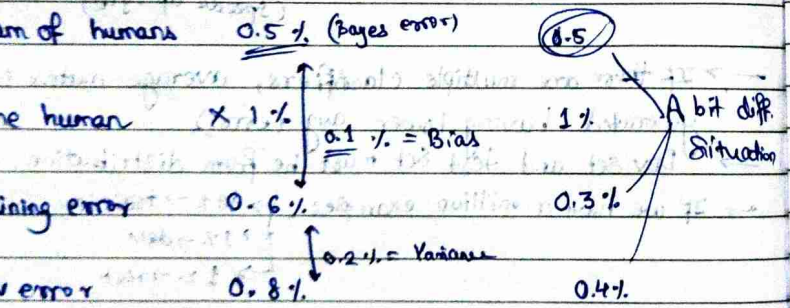
→ If your machine learning algorithm performs worse than a human, you can do:
- get labeled data from humans. (x,y)
- Gain insight from manual error analysis:
  Why did a person get this ryt?
- Better analysis of bias/variance.

Example:



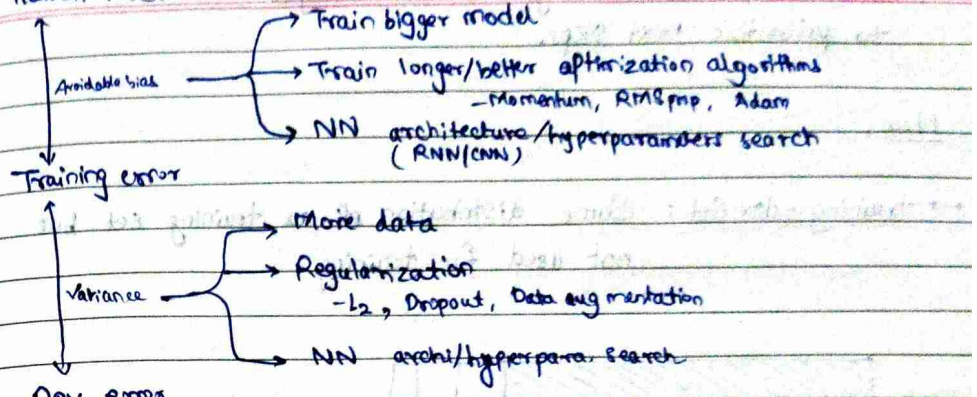
→ Human-level performance gives an estimate for Bayes error

→ Surpassing human-level performance =



→ Two fundamental assumptions of supervised learning =
1. You can fit the training set pretty well. [Low Avoidable bias]
2. The training set performance generalizes pretty well to dev/test set [Variance not too bad]

* Reducing (avoidable) bias and variance →

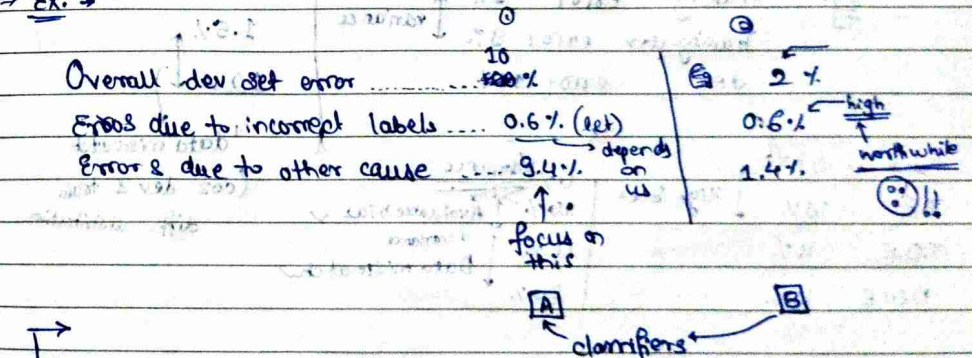


→ Incorrectly labelled examples →

↳ Deep learning algos are quite robust (insensitive) to random errors in training set.

↳ If mislabelling in dev/test set → Perform error analysis with a column of Incorrectly labelled.

⇒ Ex. →



↳1) Apply same process to your dev and test sets to make sure they continue to come from the same distribution

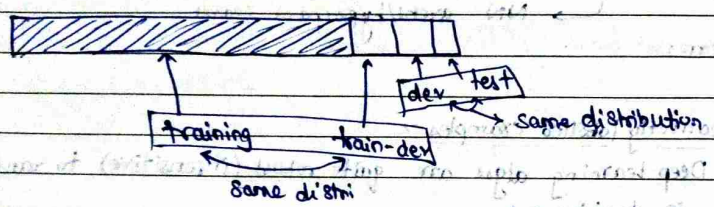
↳2) Consider examining examples your algorithm got right as well as ones it got wrong.

↳3) Train & dev/test | data may now come from slightly diff distribution..

→ 1) Set up dev/& test set and metric.
2) Build initial system quickly.
3) Use Bias/Variance analysis & Error analysis to prioritize next steps.

Now,

→ training-dev Set : Same distribution as training set, but not used for training.



→ No back propn on train-dev only on training.

eg. Training error 1%
Training-dev error 9%
dev error 10%

variance

Human	0%	
T.E.	10%	High bias
T.D.E	11%	
Dev E	12%	

0%	Variance	
10%	Avoidable bias	✓
11%	Variance	
20%	Data mismatch	✓

1%		
1.5%		
10%		
	data mismatch (coz dev & train diff. distribution)	

→ Ex:-

	General speech recogni.	Rearview mirror speech data
Human level	"Human level" 4%	6%
Error on examples trained on	"Training error" 7%	6%
Error on examples not trained on	"Training-dev" 10%	"Dev/Test error" 6%
		avoidable bias
		variance
		data mismatched

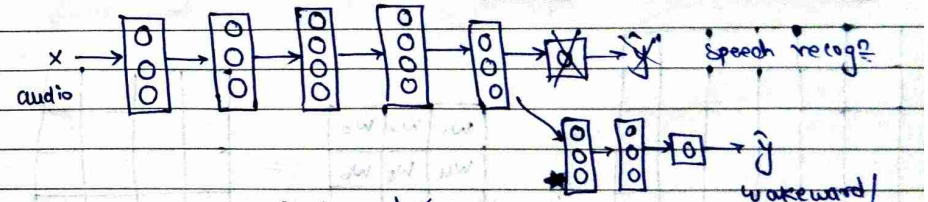
Addressing Data mismatch →

- Carry out manual error analysis to try to understand difference blw training & dev/test sets.
- Make training data more similar; or collect more data similar to dev/test sets.
  - How to?
  - Artificial data synthesis →

eg.

"The quick brown fox jumps over the lazy dog" + Car noise = Synthesized in-car audio

Transfer Learning →



Oftenly used ✓

When does transfer learning makes sense = (transfer from A→B)

- Task A and B have the same input x.
- You have a lot more data for Task A & Task B.
- Low level features from A could be helpful for learning B.

When does multi-task learning makes sense = [Better] ✓ 😊!!

- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.
- Can train a big enough neural network to do well on all tasks.

→ end-to-end learning.

↳ Pros:

1) Let the data speak $x \rightarrow y$ → "phonemes" cat
2) Less hand-designing of components needed.

↳ Cons:

1) May need large amount of data.
2) Excludes potentially useful hand-designed compon.

↔ Key Question: Do you have sufficient data to learn a func? of the complexity need to map x to y?

Course-4 → Week 1

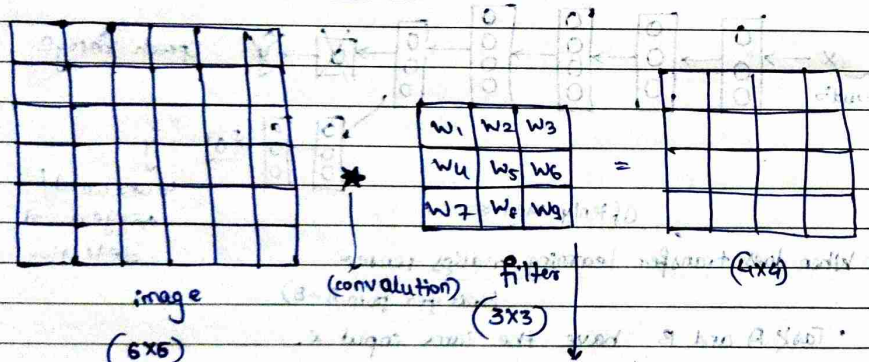


image (6×6) * (convolution) filter (3×3) = (4×4)

can find $w_1, w_2, \ldots, w_9$ by back-traking & NN properties like contours/edges.

eg. of filters

1	0	-1
1	0	-1
1	0	-1

1	0	-1
2	0	-2
1	0	-1

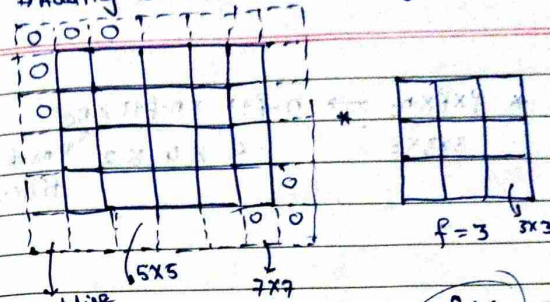
Sobel filter

3	0	-3
10	0	-10
3	0	-3

Schass filter

⊙ Padding → Problems: 1) Shrinking output 2) Throwing away info. from edges.

↳ Adding a border to the image



$p \rightarrow$ padding $\Rightarrow 1$

$n-f+1$: $7-3+1$; $5-3+1$

after padding: $n+2p-f+1 \times n+2p-f+1$

⊙ Valid Convolution: (No padding) $n \times n * f \times f \rightarrow n-f+1 * n-f+1$

⊙ Same Convolu?: Pad so that output size is the same as the input size. (padding) →

$n+2p-f+1 * n+2p-f+1$

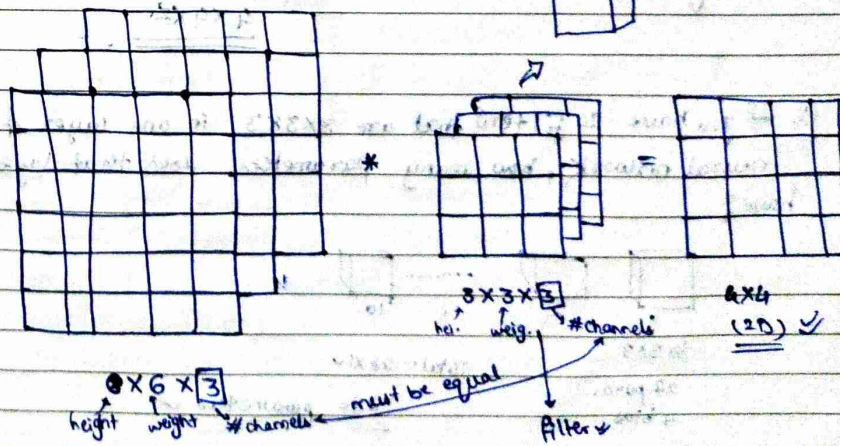
$p = \frac{f-1}{2}$

f is usually odd

→ if stride 's':

↳ $\left[\frac{n+2p-f}{s}+1\right] * \left[\frac{n+2p-f}{s}+1\right]$

* On 3-D images:

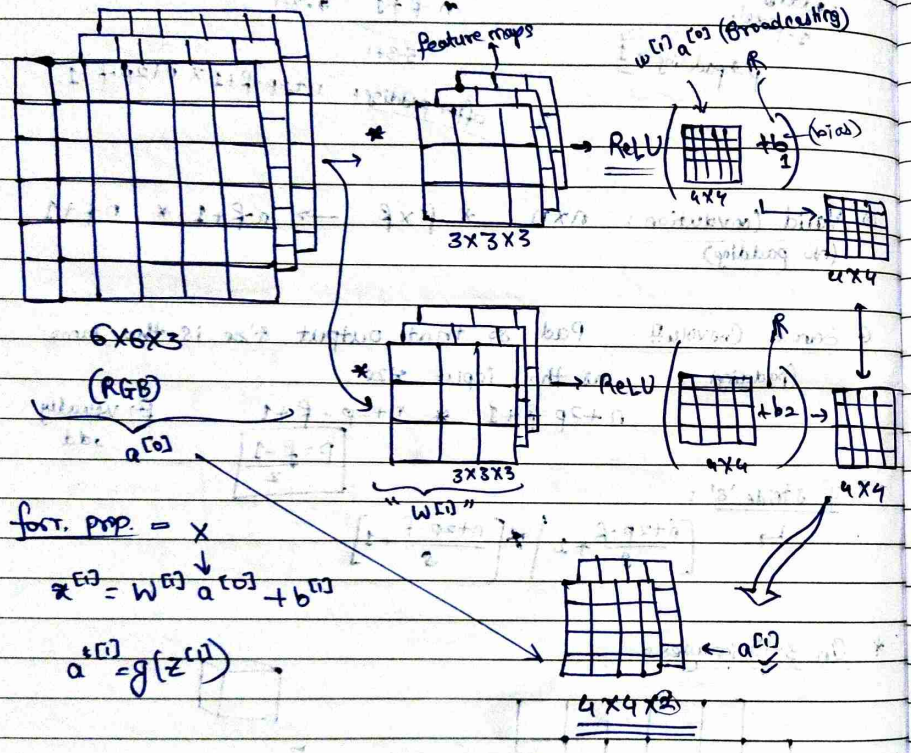


→ We can apply multiple filters to get multiple output images, we can stack them together for a combined effect.

Summary: $n \times n \times \boxed{n_c}$ * $f \times f \times n_c \rightarrow n-f+1 \times n-f+1 \times n_c'$

$6 \times 6 \times 3$ $3 \times 3 \times 3$ $4 \times 4 \times 2$ ← # no. of filters

One Layer of CNN →



for. prop. = X

$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g(z^{[1]})$

Q. If you have 10 filters that are 3×3×3 in one layer of a neural network, how many parameters does that layer have?

27 para. + bias

Total = 28×10

= 280 parameters ✓

Notation →

→ If layer l is a convolutional layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = no. of filters

Input : $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

Output : $n_H^{[l]} \times n_w^{[l]} \times \boxed{n_c^{[l]}}$

$$n_{H/w}^{[l]} = \left[\frac{n_{H/w}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right]$$

Each filter's size : $f^{[l]} \times f^{[l]} \times \boxed{n_c^{[l-1]}}$

$n_c^{[l]}$ ⇒ no. of filters applied.

activations : $a^{[l]}$

↳ $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

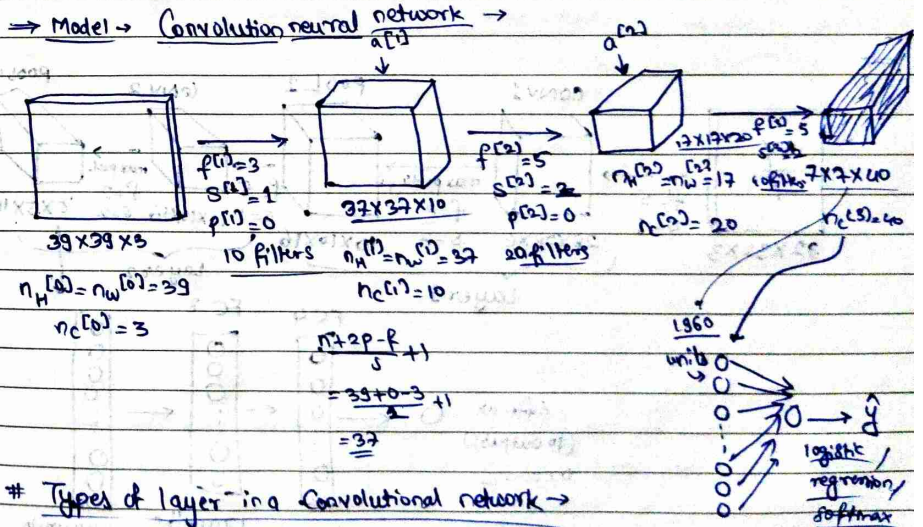
$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$ → Batch / Gradi. Descent

(set of m activ^n)

Weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ← no. of filters in layer l

bias : $n_c^{[l]}$ → $(1,1,1,n_c^{[l]})$

⇒ Model → Convolution neural network →



$n_H^{[0]} = n_w^{[0]} = 39$

$n_c^{[0]} = 3$

$n_H^{[1]} = n_w^{[1]} = 37$

$n_c^{[1]} = 10$

$\frac{n+2p-f}{s} + 1$

$= \frac{39+0-3}{1} + 1$

$= 37$

$n_H^{[2]} = n_w^{[2]} = 17$

$n_c^{[2]} = 20$

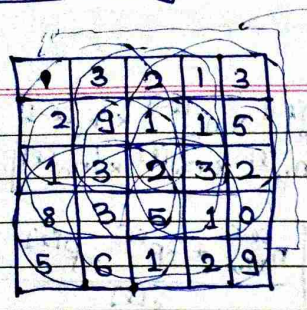
$n_H^{[3]} = n_w^{[3]} = 7$

$n_c^{[3]} = 40$

Types of layer in a Convolutional network →

- Convolution (CONV)
- Pooling (POOL)
- Fully connected (FC)

⊙ **Max Pooling** : Used often ⩔



5X5 (2D)

f=3
s=1

3X3 (2D)

f=2
s=2
used often

f=3
s=2

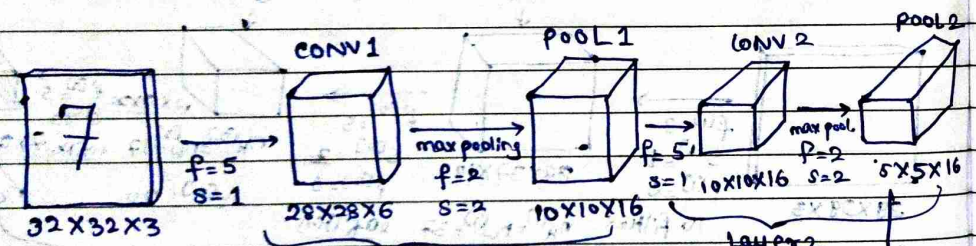
(we apply the 3x3 filter & take the maximum of all number in the desired mapping matrix)

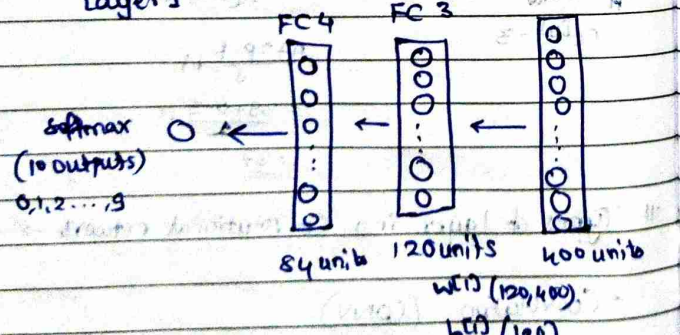
No parameters to learn!

If (3D) 5X5X2 ⟶ 3X3X2

⊙ **Average pooling** : Instead of maximum, we take avg. of all nos.

⟹





⟹

	Activation shape	Activation size	# parameters
Input	(32,32,3)	3072	0
CONV1 (f=5, s=1)	(28,28,8)	6272	208
POOL 1	(14,14,8)	1568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1600	416
POOL 2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48001
FC4	(84,1)	84	10801
Softmax	(10,1)	10	841

decreasing

↦ **Why Convolutions ?**

↳ Relatively small no. of parameters (parameter sharing)

↓

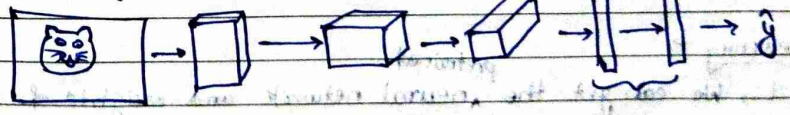
A feature detector (such as vertical edge detector) that's useful in one part of the image is probably useful in another part of image.

• **Sparsity of connections** : In each layer, each output value depends only on a small no. of inputs.

Putting it together →

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$

(w,b)



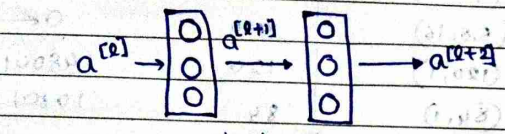
→ cost $J = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

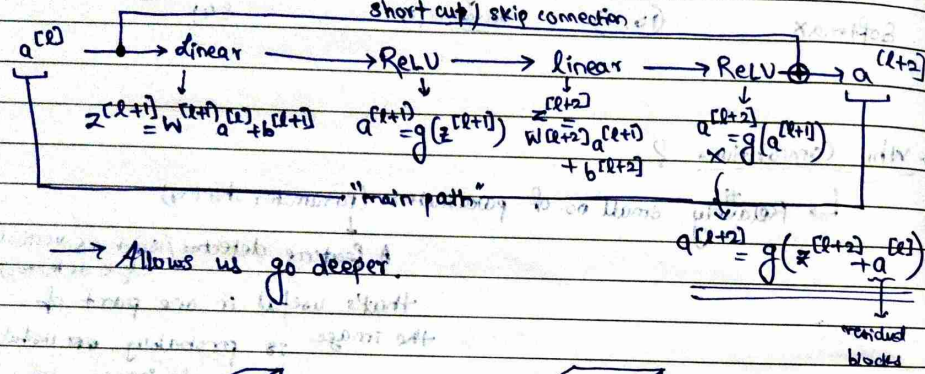
→ Use gradient descent to optimize parameters to reduce J (or Adam)

→ Case Studies (Architectures of Neural Networks) =

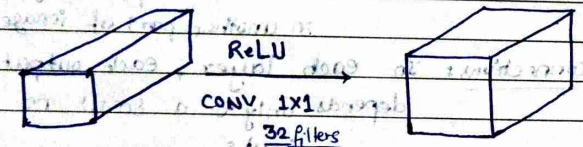
1) LeNet-5 ✓

ResNets : Made up of Residual blocks.





→ Allows us go deeper.



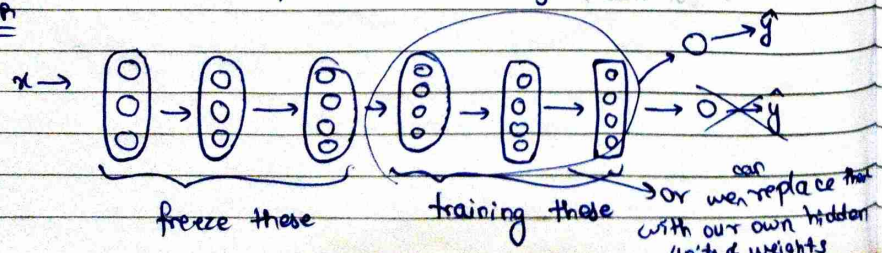
Transfer Learning →

↳ We can get the pretrained neural network and weights of an open source model & change its (or replace it) softmax fun^n with our for our desired output.

↓

We can freeze the hidden layers.

OR



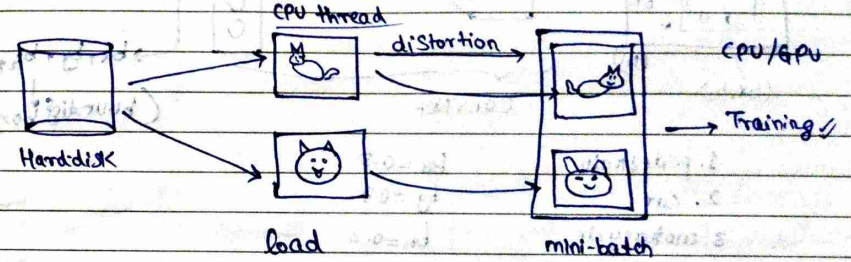
* Data Augmentation =

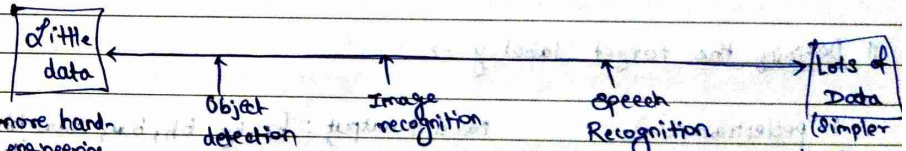
- → Mirroring of image
- → random cropping
- → Shearing
- → Local warping

* Color shifting →

↳ Changing the RGB values of image such as adding more R, G than B (small changes).

↳ Implementing distortions during training =





(Transfer learning)

→ Two sources of knowledge =

- Labelled data (x, y)
- Hand engineered features/network architecture/ other components.

Tips for doing well on benchmarks →

↳ Train several networks independently & average their outputs (Ensembling) (easy)

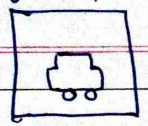
↳ Multi-crop at same time

↳ Run classifier on multiple versions of test images & avg. results. (10-crop technique)

→ Use pretrained models & fine-tune on your dataset

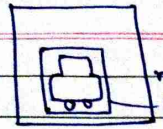
Week-3

Image Classification



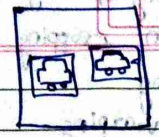
"Car"

Classification & localisation



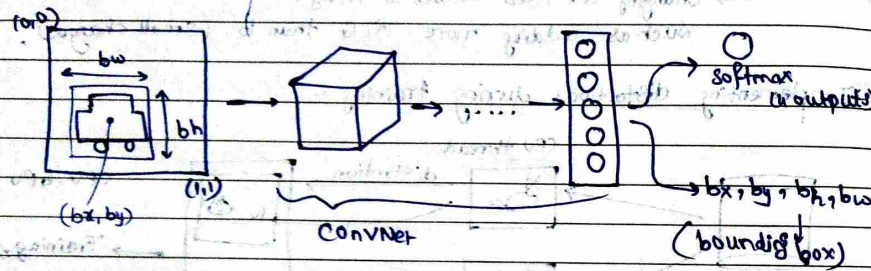
"Car"

Detection



multiple objects

1 Object



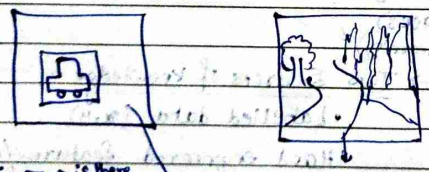
1. pedestrain
2. car
3. motorcycle
4. background

$b_x = 0.5$
$b_y = 0.7$
$b_h = 0.3$
$b_w = 0.4$

Defining the target label y →

1. pedestrain (c_1)
2. Car (c_2)
3. motorcycle (c_3)
4. background

Need to output : b_x, b_y, b_h, b_w, class label (1-4)

$X =$ 

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \leftarrow \text{is there any obj?}$$

↑ 8 components

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \leftarrow \text{"don't care"}$$

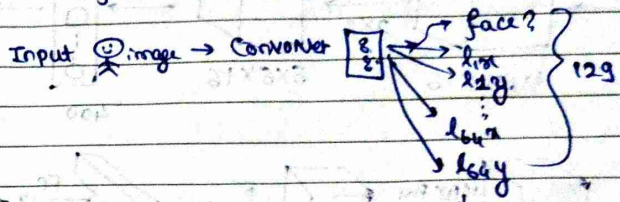
$$\mathcal{L}(\hat{y}, y) = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2$$

if $y_1 = 1$ → $(p_c = 1)$

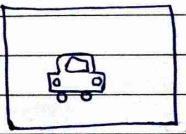
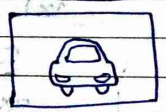
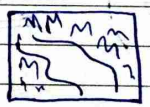
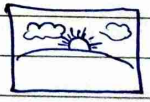
if $y_1 = 0$ $(p_c = 0)$

→ Landmark detection →

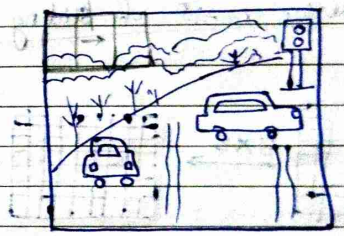
eg. Key features of face like the jawline points, eye points, etc.



* Car Detection Example :-

Training set : x	Output y
	1
	1
	1
	0
	0

Sliding Window Algorithm →



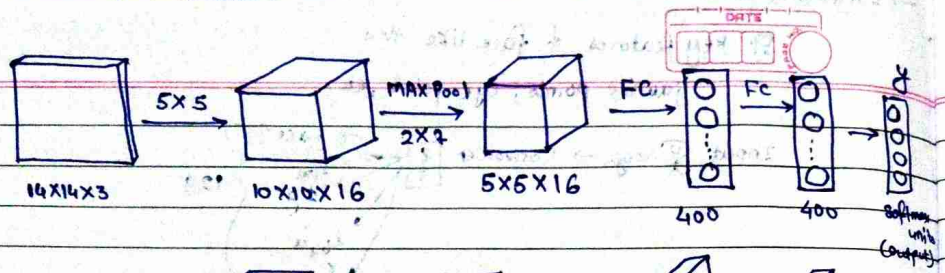
we make this window & put it into ConvNet

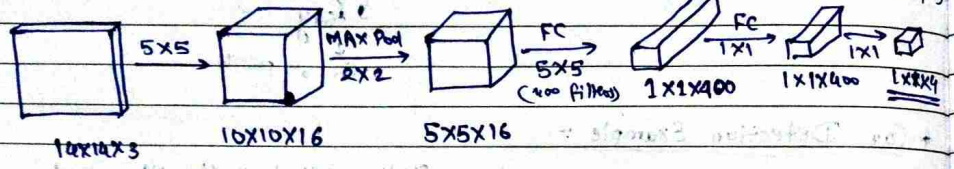
→ we keep going (sliding) until we detect the car or all over image.

→ we can now take a larger window & feed that to ConvNet

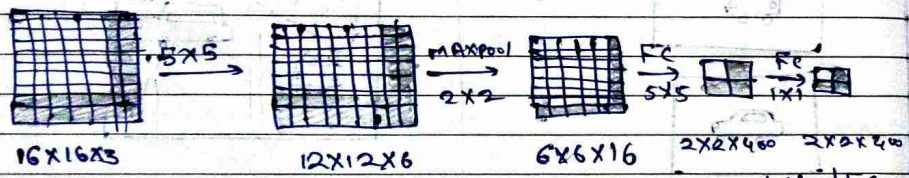
Disadv. : Very high computational cost.

→ Turning FC layer into Convolutional layer →





↝ Convolution implⁿ of sliding windows :-

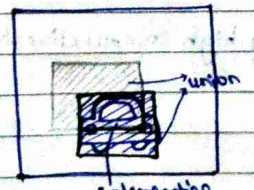


Less Computation ✓

Make all predictions at the same time ✓

Instead of doing all the time

* Evaluating localization of objects →



Intersection Over Union (IoU)

= Size of intersection / Size of union

if IoU ≥ 0.5 → "Correct"

predicted bounded box

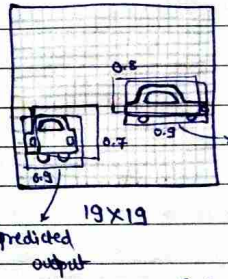
→ Our algorithm can detect the object multiple times which is not good for us

Therefore, Non-maximum suppression algorithm detects each object only one.

→ Non-max Suppression algorithm →

→ First we look at the probability associated with each detection.

→ Maximum probability detection box suppresses other boxes with lower probability.



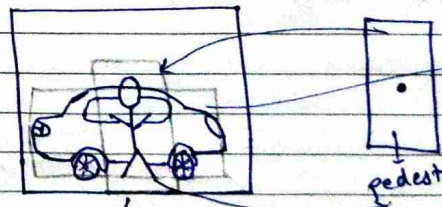
Output (prediction) for each pixel : $\begin{bmatrix} P_c \\ b_x \\ b_w \\ b_y \\ b_w \end{bmatrix}$

Discard all boxes with $P_c \leq 0.6$

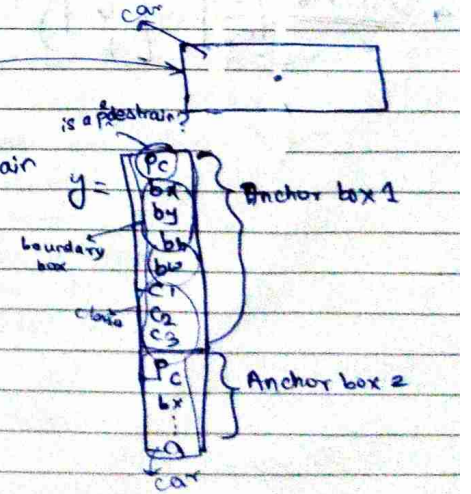
→ While there are any remaining boxes:

- Pick the box with the largest P_c, Output that as prediction.
- Discary any remaining box with IOU ≥ 0.5 with the box output in previous step.

→ Anchor Boxes :-



Anchor box 1: Anchor box 2:



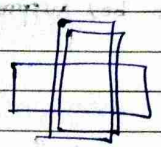
$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

→ Previously: Each object in training image is assigned to grid cell that contains that object's midpoint.

→ With 2 anchor boxes: Each object in training image is assigned to grid cell that contains object's centre (midpt.) and anchor box for the grid cell with IoU → highest.

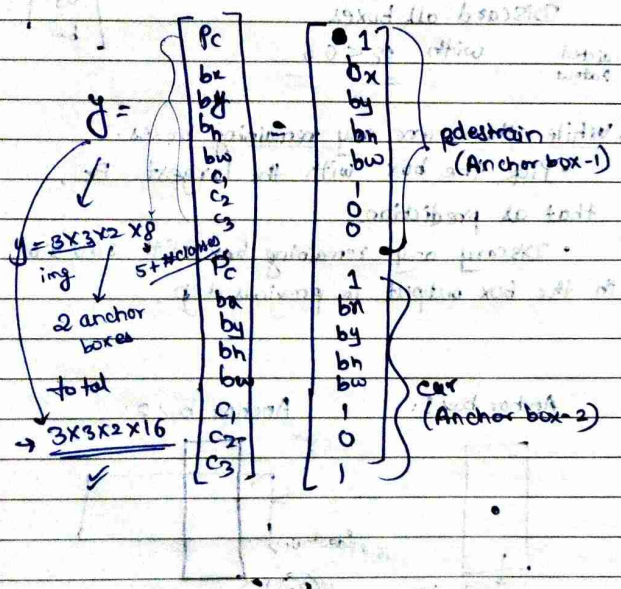
(grid cell, anchor box) pair

Output y:
3×3×8

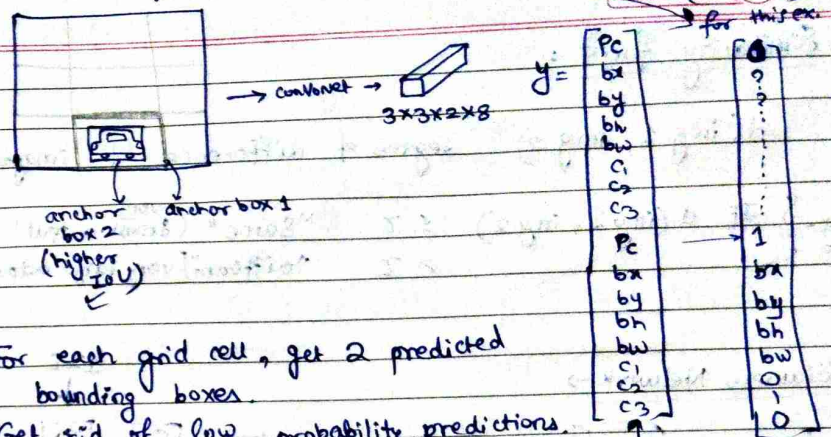


Output:
3×3×16

previous ex. output →



YOLO Algorithm → Object Detection



- For each grid cell, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrain, car, motorcycle) use non-max suppression to generate final predictions.

⊙ R-CNN → Region → The way we run the region proposal is called a segmentation algorithm. (Taking meaningful regions into consideration) blobs.

→ R-CNN: (Find of slow) Propose regions. Classify proposed regions one at a time. Output label + bounding box.

→ Fast R-CNN: Propose regions. Use convolution implementation of sliding windox to classify all the proposed regions.

→ Faster-RCNN: Use convolutional network to propose regions.

Week-4 →

↳ Face Verification (99.9%): 1:1 problem
- Input image, name/ID
- Output whether the input image is that of the claimed person.

↳ Recognition (98%): 1:K problem
- Has a database of K persons.
- Get an input image.
- Output ID if the image is any of the K-persons (or "not recognised").

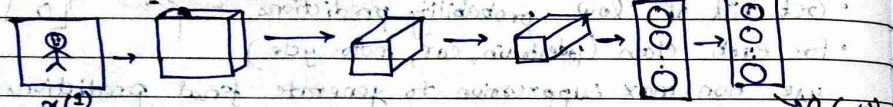
→ One-Shot learning → Learning from one example to recognize the person again.

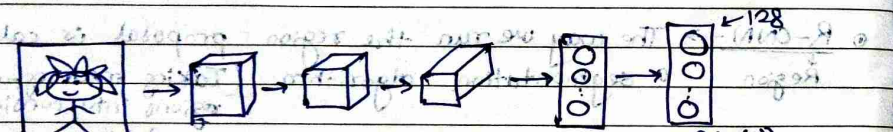
→ Similarity fun^n :

d(img 1, img 2) = degree of difference b/w images

verification { If d(img 1, img 2) ≤ τ "same" (very small output)
> τ "different" (very large output)

Siamese Network →





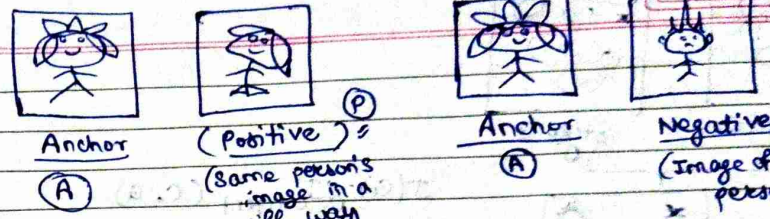
$$\rightarrow \boxed{d(x^{(i)}, x^{(j)}) = \|f(x^{(i)}) - f(x^{(j)})\|_2^2}$$

↳ Parameters of NN define an encoding $f(x^{(i)})$ → 128

↳ Learn parameters so that:

- If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|_2^2$ is small 😊!
- If $x^{(i)}, x^{(j)}$ are the diff. person, $\|f(x^{(i)}) - f(x^{(j)})\|_2^2$ is large. 😟!

→ Triplet Loss fun^n →



Want/Objective: $\underbrace{\|f(A) - f(P)\|^2}_{d(A,P)} \leq \underbrace{\|f(A) - f(N)\|^2}_{d(A,N)}$

$$\Rightarrow \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 \leq 0 \quad (-\alpha)$$

$+\alpha$ → another hyperparameter

→ Given 3 images A, P, N (A, P: same; N: diff.)

$$\mathcal{L}(A,P,N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0\right)$$

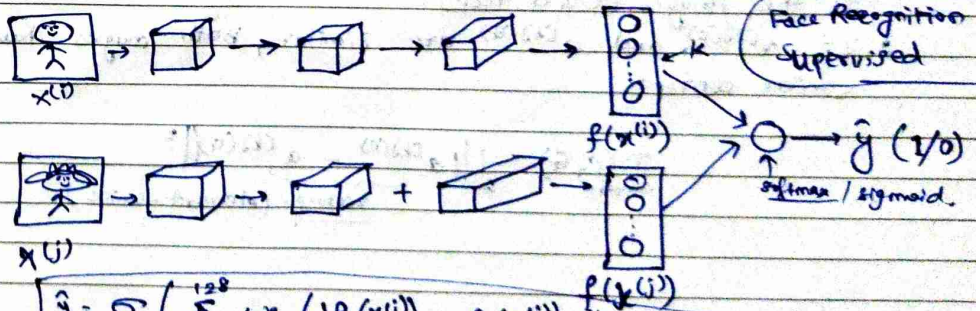
↳ single triplet

$$J = \sum_{i=1}^{m} \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

- Chose triplets that're hard to train on.

During training, if A, P, N are chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied.

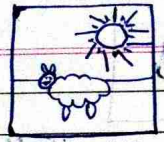
→ Use gradient descent to minimize the cost fun^n.



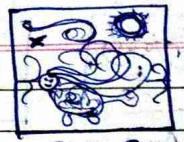
$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \underbrace{\left|f(x^{(i)})_k - f(x^{(j)})_k\right|}_{\text{or}} + b\right)$$

$$\text{or} \rightarrow \frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k} \quad \chi^2 \text{ (chi-square similarity)}$$

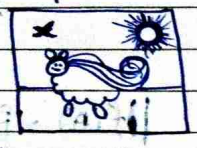
→ Neural Style Transfer Cost fun?:



Content C



Style S



Generated Imge G

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

$\alpha, \beta \rightarrow$ hyperparameters.

↳ Find the generated image G →

1. Initiate G randomly
   $G: 100 \times 100 \times 3$ → generates white noise imge
2. Use gradient descent to minimize $J(G)$
   $$G := G - \frac{\partial J(G)}{\partial G}$$

↳ Content cost fun?:

$\alpha J_{content}(C, G)$

- Say you use hidden layer l (not too deep not too shallow) to compute content cost.
- Use pre-trained ConvNet. (Eg. VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images. (C & G resp.)
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

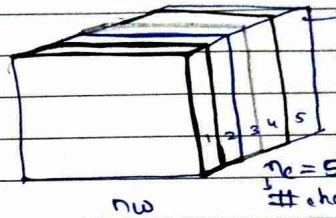
$$J_{content}(C, G) = \frac{1}{2} || a^{[l](C)} - a^{[l](G)} ||^2$$

(vectors. (element-wise))

↳ Style cost fun?:

→ Say you are using layer 'l''s activation to measure "Style".

Define style as correlation b/w activations across channels.



How Correlated are the activations across diff. channels?

→ Using the degree of correlation between channels as a measure of style, which measures the degree to which the first channel is correlated with the second channel in the generated image.

↳ Style Matrix:

Let $a^{[l]}_{i,j,k}$ = activation at (i, j, k). (i: height, j: width, k: #channels)

$G^{[l](S)}$ is $n_c^{[l]} \times n_c^{[l]}$

$$G^{[l](S)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l](S)}_{ijk} \cdot a^{[l](S)}_{ijk'} \leftarrow \text{Style image}$$

$$G^{[l](G)}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l](G)}_{ijk} \cdot a^{[l](G)}_{ijk'} \leftarrow \text{Generated image}$$

$k/k' = 1, 2, \ldots, n_c^{[l]}$

!! ☺

$$J^{[l]}_{style}(S, G) = \frac{1}{(2 n_H^{[l]} n_w^{[l]} n_c^{[l]})^2} \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)$$

$$\Rightarrow J_{style}(S, G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S, G)$$

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$