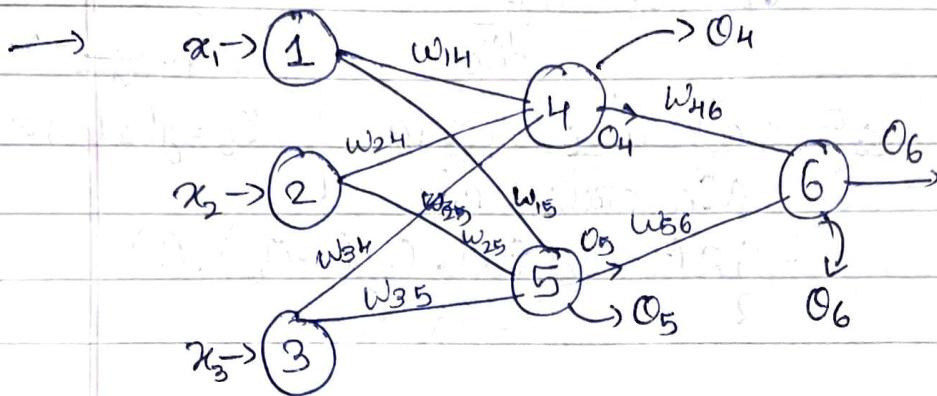


* Feed-forward Networks (Vanilla networks)



→ Inputs - $\{x_1, x_2, x_3\}$

Target - T_6

$$f(x) = \frac{1}{1+e^{-x}}$$

Forward propagation

$$\text{net}_4 = x_1 \cdot w_{14} + x_2 \cdot w_{24} + x_3 \cdot w_{34} + O_4$$

$$\text{net}_5 = x_1 \cdot w_{15} + x_2 \cdot w_{25} + x_3 \cdot w_{35} + O_5$$

$$\rightarrow O_4 = f(\text{net}_4) = \frac{1}{1+e^{-\text{net}_4}}$$

$$O_5 = f(\text{net}_5) = \frac{1}{1+e^{-\text{net}_5}}$$

$$\rightarrow \text{net}_6 = O_4 \cdot w_{46} + O_5 \cdot w_{56} + O_6$$

$$\rightarrow O_6 = f(\text{net}_6) = \frac{1}{1+e^{-\text{net}_6}}$$

Backward propagation

$$\rightarrow \text{Error } E = \frac{1}{2} (T_6 - O_6)^2$$

$$\rightarrow \text{Gradient descent: } \theta := \theta - \eta \frac{dE}{d\theta}$$

→ For O_6 :

$$O_6 = O_6 - n \frac{dE}{dO_6}$$

$$\frac{dE}{dO_6} = \frac{dE}{dO_6} \cdot \frac{dO_6}{dnets} \cdot \frac{dnets}{dO_6}$$

$$= \frac{1}{2} (T_6 - O_6) (-1) \cdot O_6 (1 - O_6) \cdot (1)$$

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$

Here $n = 4$

$$O_6 = f(nets_6) = \frac{1}{1 + e^{-nets_6}}$$

→ For W_{14} :

$$W_{14} = W_{14} - n \frac{dE}{dW_{14}}$$

$$\frac{dE}{dW_{14}} = \frac{dE}{dO_6} \cdot \frac{dO_6}{dnets_6} \cdot \frac{dnets_6}{dO_4} \cdot \frac{dO_4}{dnets_4} \cdot \frac{dnets_4}{dW_{14}}$$

$$= \frac{1}{2} (T_6 - O_6) (-1) \cdot O_6 (1 - O_6) \cdot (1) \cdot W_{14} \cdot O_4 (1 - O_4) \cdot 2_4$$

- Vanishing Gradient Problem

→ Range of $f(x) = \frac{1}{(1 + e^{-x})} \in [0, 1]$

∴ Range of $f'(x) \in [0, 0.25]$

Hence for O_6 , $\frac{dO_6}{dnets_6} \in [0, 0.25]$

∴ $\frac{dE}{dO_6}$ becomes $\frac{1}{4}$ atleast (could be lesser)

For W_{14} , $\frac{dO_6}{dnets_6} \in [0, 0.25]$, $\frac{dO_4}{dnets_4} \in [0, 0.25]$

Hence $\frac{dE}{dW_{14}}$ becomes $\frac{1}{16}$ atleast (could be lesser)

- Therefore while using log sigmoid, the updation value becomes $\frac{1}{4}$ times with every layer
- In case of large number of layers this can lead to very small values and hence no updation in weights. This is called vanishing gradient problem.

- To overcome this problem we can ReLU activation function:

$$f(x) = \text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Here $f'(x) \in [0, 1]$, ∵ vanishing gradient is delayed significantly

- However, if $x < 0$ then $f(x) = 0$. This will result in the multiplication product of updation being 0 for all layers attached to that neuron. This is called dead neuron problem and is more serious than vanishing gradient but is easier to avoid.

- To overcome dead neuron problem we can use leaky ReLU or parameterized ReLU

$$f(x) = \text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ kx & \text{otherwise} \end{cases}$$

where $k \in (0, 1)$

$$f'(x) \in [k, 1]$$

* Convolutional Neural Networks

• Convolutional layers

→ Filter / Kernel / Feature Detector / Set of weights are applied onto images or $m \times n$ matrix to extract features. For eg:

1	0	1
0	1	0
1	0	1

→ Consider an image of 5×5 , binary

1	1	1	0	0
0	1	1	1	0
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

→ Applying the filter on this image we get the following convolved feature / activation map:

4	3	4
2	4	2
2	3	4

→ This convolved feature can be considered analogous to hidden layer. This method helps us work on images without changing their dimensions which can result in loss of spatial information.

→ Working:

1	0	1
0	1	0
1	0	1

Filter

1	1	1			
0	1	1			
0	0	1			

1x1	1x0	1x1
0x0	1x1	1x0
0x1	0x0	1x1

Output

$$\therefore 1+0+1+0+1+0+0+1 = 4 \Rightarrow \boxed{4}$$

4		

Next stride:

1	1	0
1	1	1
0	1	1

1x1	1x0	0x1
1x0	1x1	1x0
0x0	1x0	1x1

$$\therefore 1+0+0+0+1+0+0+0+1 = 3 \Rightarrow \boxed{3}$$

4	3	

Next stride:

1	0	0
1	1	0
1	1	1

1x1	0x0	0x1
1x0	1x1	0x0
1x1	1x0	1x1

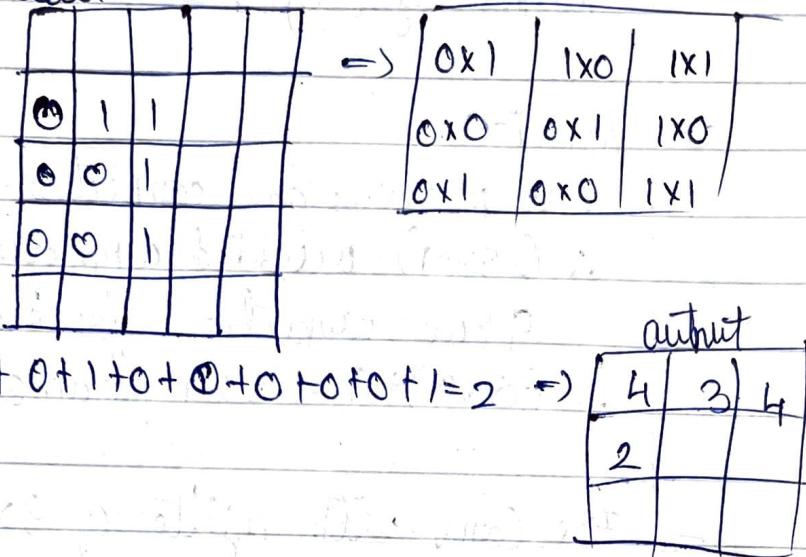
$$\therefore 1+0+0+0+1+0+1+0+1 = 4 \Rightarrow \boxed{4}$$

4	3	4

प्रत्युने प्रार्थना एवं शक्तिशाली हथीयार से।

No more right stride possible, so go to extreme left and one row down

∴ Next stride:



and so on

$$\rightarrow \text{Size of output} = \frac{w - F + 2P}{S} + 1$$

where w is size of input

F is size of filter

P is amount of padding

S is stride

For above case, $w=5$, $F=3$, $P=0$, $S=1$

$$\therefore O = \frac{5 - 3 + 2(0)}{1} + 1 \\ = 3$$

∴ 3×3 output

→ Fully connected layer - A layer i is said to be fully connected if its neurons are connected to all neurons of the layer $i-1$

• → No of connections in FC vs Conv:

→ Consider image of 5×5 and activation map/ hidden layer of 3×3 .

→ For FC:

All neurons are connected

$\therefore (25 \times 9)$ weighted connections

9 bias connections for hidden layer

$$\therefore (25 \times 9) + 9 = 234 \text{ parameters}$$

→ For Conv: with a filter of 3×3

1 neuron of activation map is connected to $(3 \times 3 = 9)$ neurons (filter on image) + its own bias

\therefore For 1 neuron $\Rightarrow (3 \times 3) + 1 = 10$ connections

\therefore For $(3 \times 3) \cdot 9$ neurons of activation map $\Rightarrow 10 \times 9$

= 90 connections

→

x_1	x_2	x_3	x_4	x_5
x_6	x_7	x_8	x_9	x_{10}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{16}	x_{20}	
x_{21}	x_{25}

F_{11}	F_{12}	F_{13}
F_{21}	F_{22}	F_{23}
F_{31}	F_{32}	F_{33}

n_1	n_2	n_3
n_4	n_5	n_6
n_7	n_8	n_9

filter output

image

$$n_1 = x_1 \cdot F_{11} + x_2 \cdot F_{12} + x_3 \cdot F_{13} + x_6 \cdot F_{21} + x_7 \cdot F_{22} + x_8 \cdot F_{23} \\ + x_{11} \cdot F_{31} + x_{12} \cdot F_{32} + x_{13} \cdot F_{33} + b_1$$

$$n_2 = x_2 \cdot F_{11} + x_3 \cdot F_{12} + x_4 \cdot F_{13} + x_7 \cdot F_{21} + x_8 \cdot F_{22} + x_9 \cdot F_{23} \\ + x_{12} \cdot F_{31} + x_{13} \cdot F_{32} + x_{14} \cdot F_{33} + b_2$$

Here the filter weights are shared, hence parameter sharing

प्रत्युने प्रार्थना से शक्तिशाली हो जाएगी।

Receptive fields are defined portion of space or spatial construct containing units that provide input to a set of units within a corresponding layer. The receptive field is defined by the filter size of a layer within a convolution neural network.

- When filter size < image size then there is said to be local connectivity as only a certain subset of neurons will be connected to the next layer.
- The area in the input layer ^{or} of the previous layer which is connected to the next layer is said to be its receptive field.
- For RGB image, we have 3 channels. Therefore, input image has dimensions $5 \times 5 \times 3$. Therefore, we use a filter of depth 3 i.e. of dimensions $f \times f \times 3$.
 - ∴ for R channel $f \times f \times 1$
 - Or channel $f \times f \times 1$
 - B channel $f \times f \times 2$
- Each depth inputs are added on the whole filter has a single bias.

- Pooling layer
- Used to extract certain values from input images
- Different types: max, min, mean, etc

- Max pooling layer of 2×2 filter with stride 2

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→

6	8
3	4

CNN can work on variable size images

श्री रामनारायण इवार्षिन भिश्वास

Dt.: / / Pg.no.:

$$\rightarrow \text{Size of output vol } O = \frac{W-F+1}{S}$$

\rightarrow Only two common Max pooling layers are seen:

i) $f=3 \quad S=2$ (overlapping pooling)

ii) $f=2 \quad S=2$

Pooling sizes with larger receptive fields
are destructive as they can discard ^{many} neurons/cells.

- Normalization layer

\rightarrow Normalization layer is used to normalize

\rightarrow It can be used in two ways:

i) normalizing output of a hidden layer before
sending it to next layer, mostly done after
activation function and in the same layer wrt values

ii) normalizing inputs of preceding layer before
working on it in the current layer, done just
after receiving the input and in the next layer wrt val

\rightarrow In practice, second method is found to be better than
the first

\rightarrow Can be done in batch or mini-batch method

- Dropout layer

\rightarrow Prevents overfitting

\rightarrow Drops neurons as per ratio $\approx (0.2, 0.3, \text{etc})$

\rightarrow All connections related to that neuron are dropped

प्रत्युत्तम आदेश से शक्तिशाली होती है।

* Softmax Activation function

→ Used as a probabilistic measure, generally for multiclass classification problems

$$\rightarrow S(y_i) = \frac{e^{y_i}}{\sum_{j=0}^n e^{y_j}} \quad n \text{ is no of } \cancel{\text{class}} \text{ inputs}$$

→ Consider $y = \{2.0, 1.0, 0.1\}$

$$S(y_1) = \frac{e^{y_1}}{e^{y_1} + e^{y_2} + e^{y_3}} \quad S(y_2) = \frac{e^{y_2}}{e^{y_1} + e^{y_2} + e^{y_3}} \quad S(y_3) = \frac{e^{y_3}}{e^{y_1} + e^{y_2} + e^{y_3}}$$

$$p(y_1) = 0.7 \quad p(y_2) = 0.2 \quad p(y_3) = 0.1$$

∴ class 1 has max probability.

* Binary / Categorical cross entropy

$$\rightarrow \text{Cross Entropy} : CE = - \sum_{i=1}^n t_i \log_2 (f(s)_i)$$

→ For binary problem

In → Target

$$\begin{array}{c|c} t_1 & t_2 \\ \hline \end{array} \rightarrow [NN] \rightarrow \begin{array}{c|c} 0.3 & f(s)_1 \\ 0.7 & f(s)_2 \end{array}$$

$$\text{Then } CE = -t_1 \log_2 (f(s)_1) - t_2 \log_2 (f(s)_2)$$

$$\text{Then } CE = -(0) \log_2 (0.3) - (1) \log_2 (0.7)$$

$$= \log_2 (0.7)$$

DL

* Transfer learning

- Ability of a system to recognize and apply knowledge and skills learned in previous tasks to novel tasks is called transfer learning
- Traditional supervised learning paradigm fails in the absence of sufficient labelled data for the task/domain to train a reliable model
- Applying similar domain models eg day-time images for a domain without ground truth eg - night time images pedestrian detection can often result in performance deterioration due to inherited bias which cannot be generalized to the new domain.
- If we want to detect cyclists, then we cannot reuse the model due to different labels.
- In such cases, transfer learning can be used by using existing labelled data of a similar task/domain. The knowledge gained is stored solving the source task is stored and then applied to the custom model
- Two major transfer learning scenarios are:
 - 1) ConvNet as a fixed feature extractor
Take a ConvNet pretrained on ImageNet, remove the last fully connected layer, then treat rest of the ConvNet as a fixed feature extractor for new dataset

प्रारंभिक भावना से शक्तिशाली होती रहती है।

~~It is important to have been spent (i.e.)~~

ii) Fine-tuning the ConvNet

- Replace and retrain the classifier on top of the ConvNet on the new dataset, ~~but~~ ^{and} also fine tune the weights of the pretrained network by continuing the backpropagation
- The mode of transfer learning to use depends on several factors, out of which the most important two factors are size of new dataset, and its similarity to the original dataset
- Based on this we have 4 scenarios:

(i) Small and similar to OGI dataset

- Pretrained Net as fixed feature extractor
- Extract CNN codes
- Using CNN codes and labels, use ^{linear} SVM or shallow classifier (as small dataset)

ii) Large and similar to OGI dataset

- Pretrained Net ~~can~~ can be fine tuned (~~as~~ large dataset)

iii) Small and different from OGI dataset

- Train linear SVM or shallow NN classifier
- If pretrained Net is to be used then take features from earlier layers as they capture generic features. Layers closer to output capture specific

iv) large and different from O/w datasets

- → Learn CNN from scratch

OR

→ Take pretrained net and backpropagate till earlier layers

- Negative transfer

- Negative transfer happens when source domain data and task contribute to reduced performance of learning in target domain
- This can happen if:
 - domains are too dissimilar
 - tasks are not well related

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

→ joint task of learning from D1 and D2 (to

reduces negative transfer)

प्रभुने प्रार्थना एवं शक्तिशाली हृदीयारं हैं।

* Sequence Models

- Used when sequence data is involved as input, output or both.
 - In sequence data, order within the data is important. For eg - speech, text data, DNA sequence

↑ participant ↑ witness
the effect is 10%

~~all samples contain significant amounts of~~

July 15, 1991 010505.6

~~2018-07-13~~

birds are still with the old and appear to be pushing off in the south.

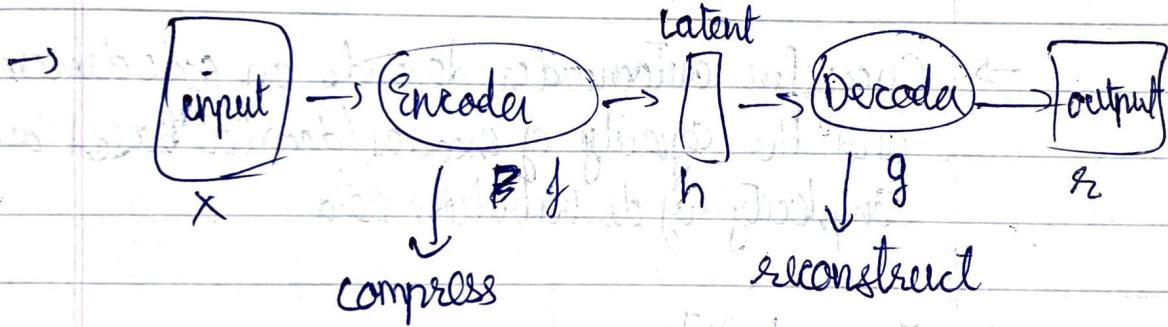
13. ~~guitar~~ guitar solo: guitars only.

~~Group of 2000+ fingerlings~~ 2000+ fingerlings

analysis of *Streptomyces* -
turkestanicus

* Autoencoders

- Unsupervised Neural nets
- Aim to copy their inputs to outputs
- Work by compressing the input into a latent space representation, and then reconstructing the output from this representation
- Reconstruction loss = $\frac{\text{Reconstruction} - \text{Original}}{\text{No. of inputs}}$
(mean squared average or other errors)
- Aim is to minimize reconstruction loss



$$h = f(x) \Rightarrow \text{encoder} \Rightarrow \text{input to latent rep.}$$

$$r = g(h) \rightarrow \text{decoder} \Rightarrow \text{latent rep. to reconstruct}$$

\therefore autoencoder is $g(f(x)) = r$

- We train the autoencoder to copy input to output, so that latent representation h takes up useful properties. This can be achieved by placing constraints on the copy task.

- One way is to constrain z_h to have smaller dimensions than x . This is called undercomplete autoencoder.
- This forces it to learn most salient features.
- If too much capacity is given, it can learn to perform copying without extracting any useful information.
- This can happen when latent size is more than input. This is known as overcomplete autoencoder.
- In such cases, linear encoder/decoder can also copy the input without learning useful information.
- Successful autoencoders depends on code dimension and the capacity of encoder/decoder based on complexity of distribution data.

* Types of Autoencoders

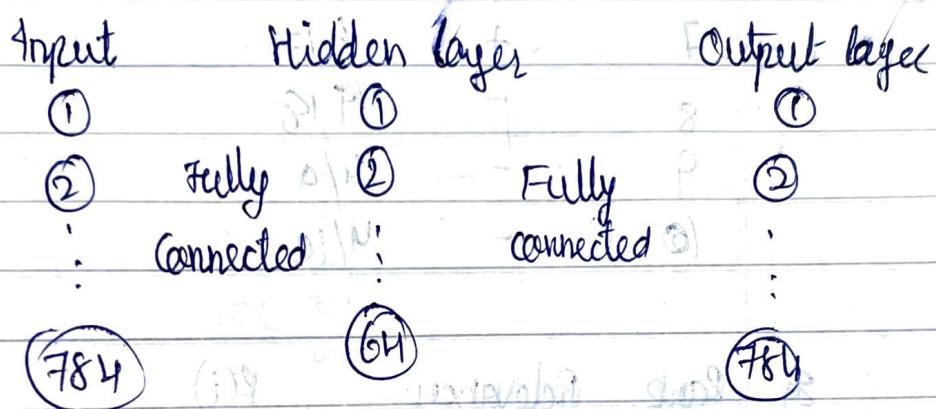
- Vanilla
- Multilayer
- Convolutional
- Regularized

P.T.O.

* Vanilla Autoencoder

- Simplest form - Three layer net \rightarrow input, 1 hidden layer, output
- Fully connected dense layers
- Adam optimizer and MSE loss are used

Eg:

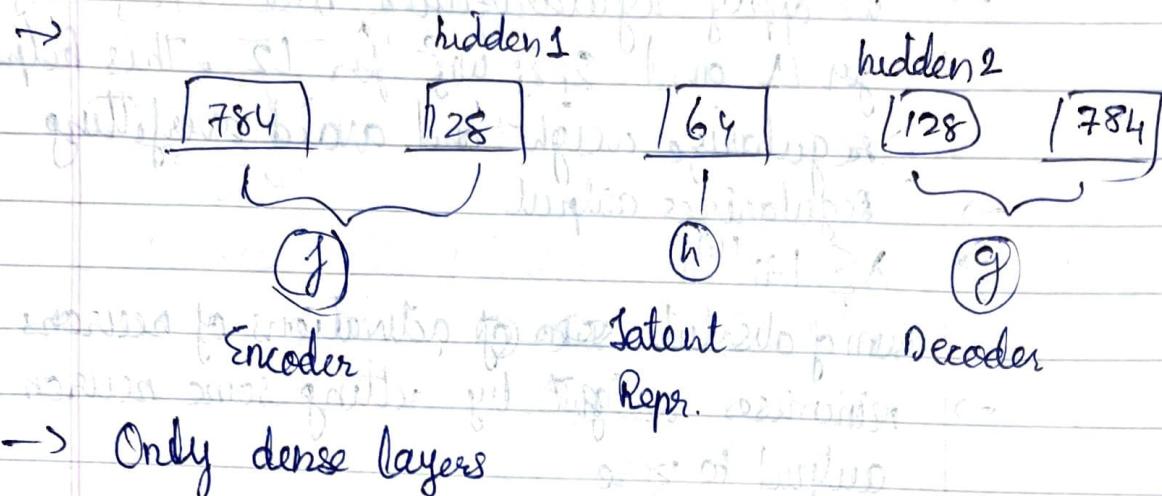


→ Once quality of output is as per desire, decoder can be removed. Now latent layer acts like a feature extractor for input

→ If input data is completely random without internal correlations, then undercomplete autoencoders will not be able to reconstruct perfectly.

DL

* Multilayer Autoencoder



→ Only dense layers

* Convolutional Autoencoder

→ Can use convolutional layers. & no need to flatten data

→ Check ppt for architecture

* Regularized Autoencoders

→ Rather than limiting model capacity by keeping encoder/decoder shallow and code size small, we regularize the loss

→ Two types:

- Sparse AE.
- Denoising A.E.

F.T.O.

- Sparse Autoencoder

- we apply regularization like $\sum_i \sum_j |w_{ij}|$ for L1 and $\sum_i \sum_j w_{ij}^2$ for L2. This helps regularize weight and avoid overfitting.
- regularizes output
- $\lambda \sum_i |g_i^{(h)}|$
- sum of absolute ~~sum of~~ activations of neurons
- minimises ~~weight~~ by setting some neuron output to zero

- Denoising autoencoders
- Another way to force AE to learn useful features, is to add random noise to its inputs and making it recover original noise-free data
- AE can't simply copy the input to output due to presence of random noise
- Subtract noise and reconstruct underlying input
- The architecture is same, only inputs differ

* Stacked Autoencoders

→ AE:1 (784 - 100 - 784)

Train

Get satisfactory reconstruction

Drop decoder

Call AE1.predict (784-100) latent becomes output

AE:2 (100 - 50 - 100) latent is new input

Train

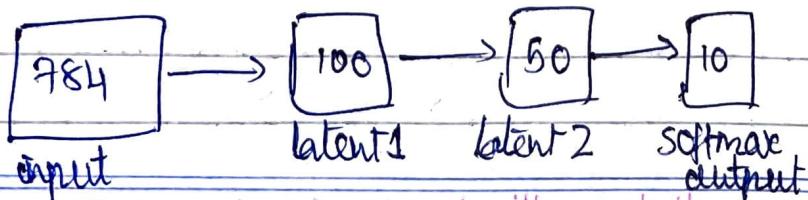
Get satisfactory reconstruction

Drop decoder

Call AE2.predict (100 - 50) final latent is output

Final softmax layer for 10 class classification

Final architecture :

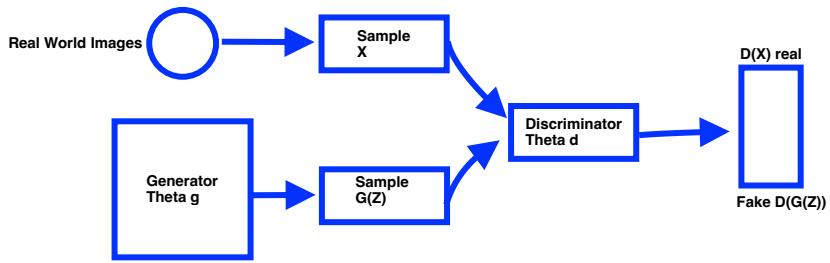


Follow the river and will reach the sea

DL

* Generative Adversarial Networks

- Discriminative networks help differentiate
- Generative networks help generate
- Estimate joint probability (generative)
- Estimate probability of y given x (discriminative)
- GANs consist of generator NN and discriminative NN both.
- Θ_g refers to all parameters of generator
- ~~Θ_d~~ Θ_d refers to all parameters of discriminator
-



- x is real
- $D(x)$ is likelihood of x being real
- $G(z)$ is synthetic. (z is noise vector)
 $D(G(z))$ is likelihood of $G(z)$ being real
- Discriminator tries to differentiate real and fake
Generator tries to fool discriminator
- Whenever component fails, loss is used to improve performance

Recitation is the best mode of worship.

- This process is known as alternative training
- Once satisfactory generation quality is achieved, discriminator can be removed. After that, input of 'z' will generate new sample
- Generator is ~~use~~ upsampler
Discriminator is down sampler

$(\mu, \sigma) + (x, y)$ & $\text{latent} - (E, D)$

latent space
latent variable
latent representation
latent dimension
latent feature
latent variable

meaning of noise in latent dimension of latent
variable

noise in latent space
noise in latent variable
noise in latent representation
noise in latent dimension
noise in latent feature
noise in latent variable

noise

- Strict alternative training:
Train discriminator for 1 epoch followed by generator for 1 epoch and repeat
- Not strictly alternative Training:
Train discriminator for $K (> 1)$ followed by generator for k epochs and repeat. K is variable
- Training Discriminator (MNIST eg. (28×28))
- Weights of generator are fixed, no backprop in generator
- Take batch size $\frac{50}{\text{mini}}$. Draw 50 original images and sample 50 from generated images
- Input to discriminator is 100×1 images. Original images are labelled 1 and generated are labelled 0
- Assuming discriminator is fully connected, input vector becomes (100×784)
- Train discriminator, get loss and use backprop on discriminator only. Weights are updated in discriminator only
- x mini batches are used in 1 epoch. Training is complete after 1 epoch (strictly alternative)

- Training Generator (MNIST eg (28×28))
 - Trained using fake images only
 - Discriminator weights are fixed
 - 50 images generated and passed to discriminator.
 - Generator wants that for 50 fake images, discriminator outputs them as real i.e. 1
 - Input vector (50×784) , output vector (50×1) with all target values expected 1
 - Discriminator output is used to calculate loss (number of images classified as fake). This loss is backprop to generator. Weights are updated for generator only

• Equations

- x is sampled from real data
 $P_{\text{data}} \rightarrow$ real data
- z is sampled from fake data
 $P(z) \rightarrow$ fake data.

↗ $G_{\text{gen}}(z)$
 ~~$G_{\text{gen}}(z)$~~ → generating
 $D_{\text{ad}}(G_{\text{gen}}(z)) \rightarrow$ identify fake as real
 $1 - D_{\text{ad}}(G_{\text{gen}}(z)) \rightarrow$ identify fake as fake

→ Alternate between:

- i) Gradient ascent on discriminator (\therefore maximize)

$$\max_{D_d} [E_{x \sim P_{\text{data}}} \log D_{\text{ad}}(x) + E_{z \sim P(z)} \log (1 - D_{\text{ad}}(G_{\text{gen}}(z)))]$$

log likelihood of
real identified as real

log likelihood of
fake identified as
fake

- maximize identifying fake as fake and real as real

information is the best mode of worship.

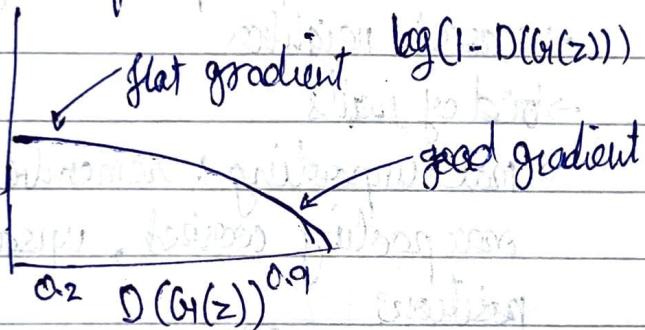
ii) Gradient descent on generator: (\because minimize)

$$\min_{\theta_g} \left[E_{z \sim p(z)} \log \underbrace{\left(1 - D_{\theta_d}(G_{\theta_g}(z)) \right)}_{\text{log likelihood of fake identified as fake}} \right]$$

log likelihood of
fake identified as fake

\rightarrow from minimize identifying fake as fake

\rightarrow Problem with this loss function is that when generator is not performing well, gradient information (error) is ^{poorly} ~~not~~ available. When generator performs well, gradient information is available which is not required



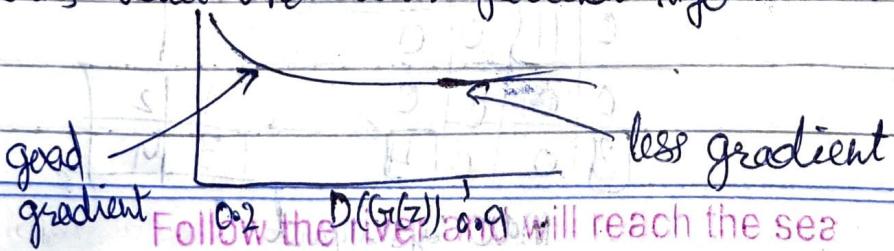
\rightarrow Therefore we change the loss function of generator to:

gradient ascent on generator

$$\max_{\theta_g} \left[E_{z \sim p(z)} \log \left(D_{\theta_d}(G_{\theta_g}(z)) \right) \right]$$

maximize identifying fake as real

\rightarrow works better and better gradient info available



→ GAN Training algorithm: refer ppt

* DCGAN

- Deep Convolutional GAN
- Generator is unsampler with fractionally strided convolutions
- Discriminator is a convolutional network

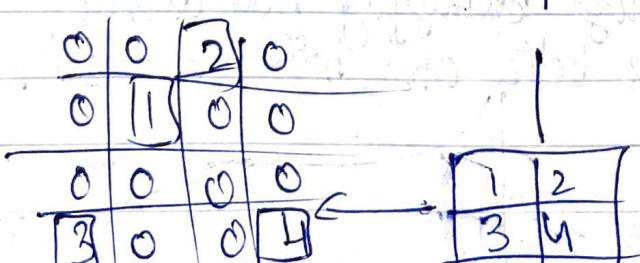
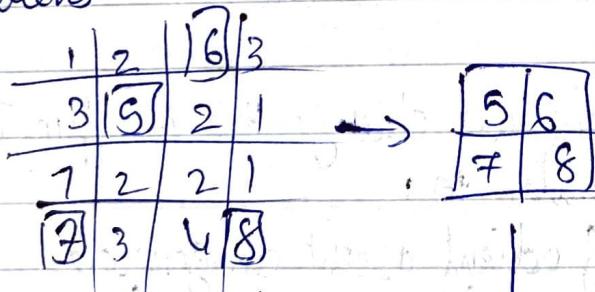
→ For architecture, refer ppt.

→ In-network upsampling / upscaling:

→ nearest-neighbor

→ bed of nails

→ max unpooling: remember positions on which max pooling worked, upsample to those same positions



Max unpool

- If asymmetric padding (2 dims. only) then $p = \frac{1}{2}$
- Searchable Upsampling: transpose convolution
- Stride : ratio of movement in input and output
- \Rightarrow Searchable Upsampling : transpose convolution
- fractionally strided conv., backward strided conv., upconvolution, deconvolution (bad name)
- Here stride = P/q (fraction) \therefore fractionally strided. Eg $s = \frac{1}{2}$, 1 in input, 2 in output
- Transpose Conv with 3×3 filter on 2×2 image
 $P = \frac{1}{2}$ (bottom and right) $s = \frac{1}{2}$

Img:

1	2
3	4

filter:

1	1	1
1	1	1
1	1	1

Filter maps formed by input \times filter

\therefore 4 filters of 3×3 , with 1, 2, 3, 4 in all 3×3 of each respectively

P.T.O.

→ Filter 1



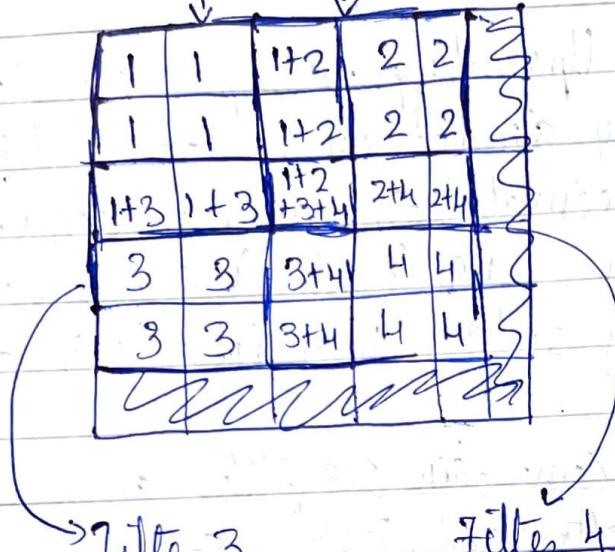
1	1	1
1	1	1
1	1	1

Filter 2

2	2	2
2	2	2
2	2	2

Since $s=1/2$

1 in step \Rightarrow 2 in output



→ Filter 3

3	3	3
3	3	3
3	3	3

Filter 4

4	4	4
4	4	4
4	4	4

→ Result

1	1	3	2	2
1	1	3	2	2
4	4	10	6	6
3	3	7	4	4
3	3	7	4	4

P.T.O

→ Removing right & bottom padding

1	1	3	2
1	1	3	2
4	4	10	6
3	3	7	4

- GRANS/DCGRANS support interpretable vector math
- Embedding layer converts one-hot vector into dense layer. They support interpretable vector math.
- One-hot is sparse.
- consider one-hot encoding of King, man, queen, woman.

word embedding will make dense vector of each. Same layer is used for all

$V_k \rightarrow$ King vector

$V_m \rightarrow$ Man vector

$V_w \rightarrow$ Woman vector

④ Interpretable vector math can be considered as:

$$V_k - V_m + V_w = V_q \text{ (Queen vector)}$$

We use the word embedding layer on queen and find V_q

Difference between V_q and V_q' gives us similarity between desired result and actual result

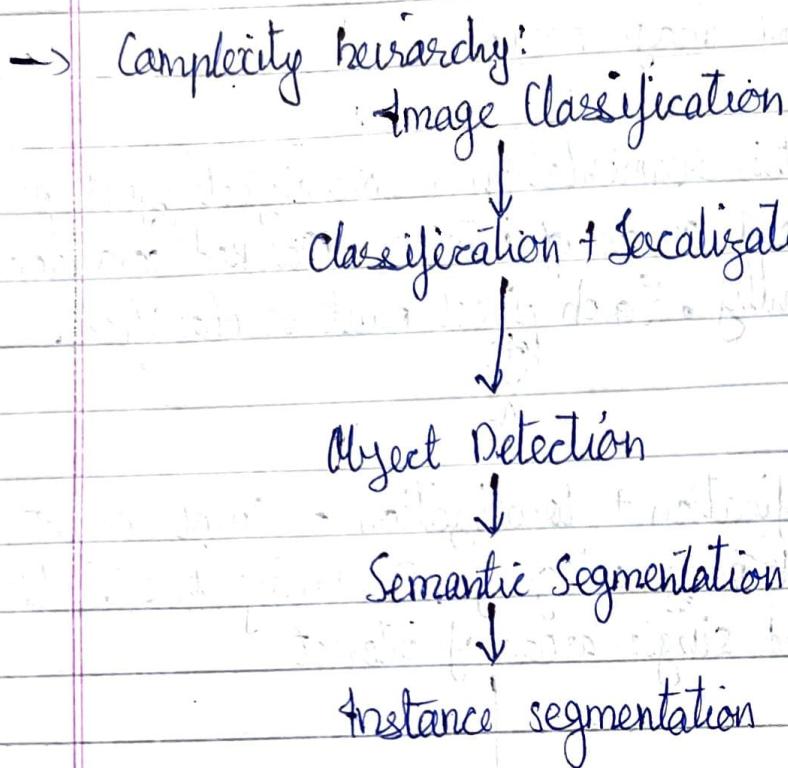
* Different image models

- i) Semantic Segmentation - no objects, just pixels.
Different areas of interest marked/coloured differently. Each pixel must be classified.

- ii) Classification + localization - Single object. Classify into ~~two~~ category. Bounding box involved around single area of interest

- iii) Object Detection - multiple objects. Multiple bounding boxes around multiple areas of interest.
All objects of interest must be categorized into their respective categories.
- iv) Instance segmentation - Each pixel is classified.
Instances of the ~~object~~ pixels are also differentiated.
For eg - Image of 2 dogs and 1 cat, instance segmentation will differentiate Dog 1 and Dog 2 as well whereas semantic segmentation will differentiate dogs from cat but not from each other.

P.T.O.



* Semantic Segmentation

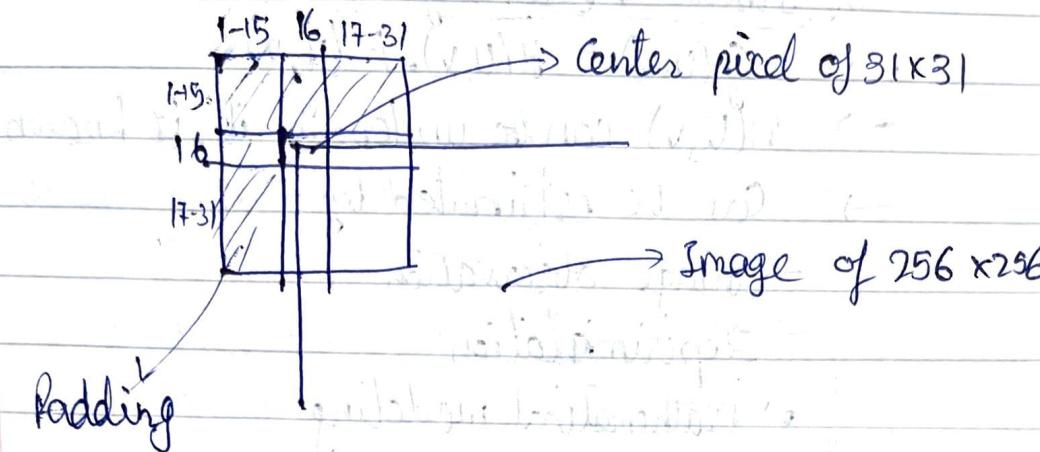
- Differentiate / Classify each pixel
- Instances of same object are not differentiated

• Sliding Window

- Consider img of 256×256 , \$ window of $31 \times 31$$
- Patch is extracted, entire patch is fed into CNN, class of patch is the class for CENTER PIXEL only.
- Computationally expensive as 256×256 : forward propagations required in ~~prediction~~ ^{prediction} stage alone.
- Very inefficient as shared features between overlapping patches
- Stride has to be 1 otherwise, pixel will be skipped

→ How pixel classification starts

Padding used on edges

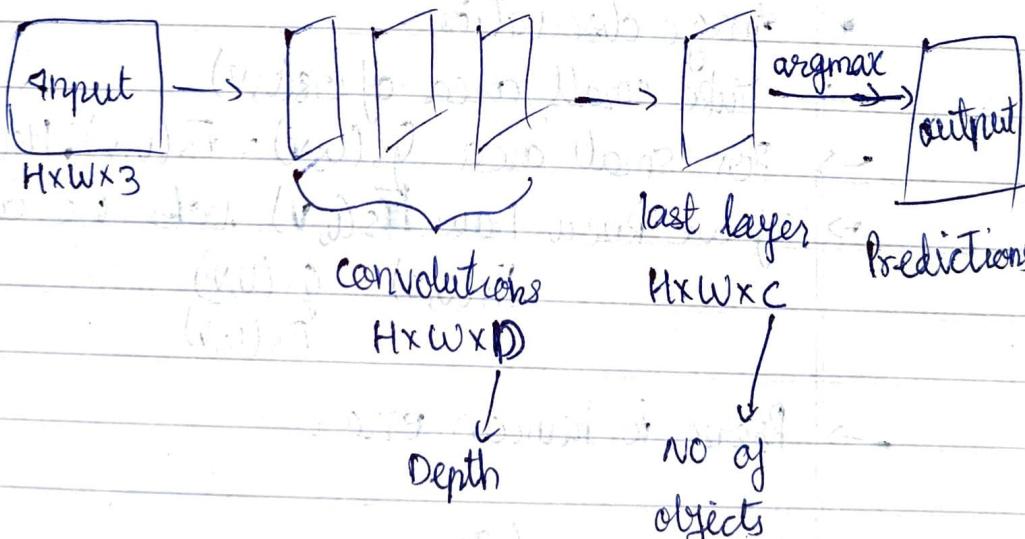


• Fully Convolutional Neural Network

→ All layers are convolutional

→ Spatial extent remains unchanged

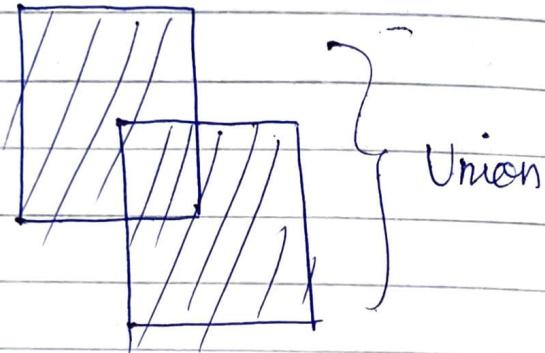
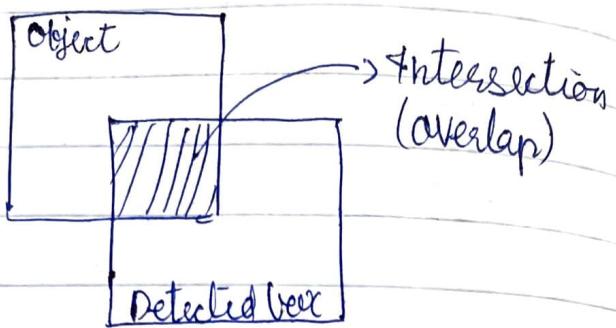
→



→ Convolutional on original ~~layer~~ image layers is costly

* Intersection over Union

$$\rightarrow \text{IOU} = \frac{\text{Area of overlap}}{\text{Area of union}}$$

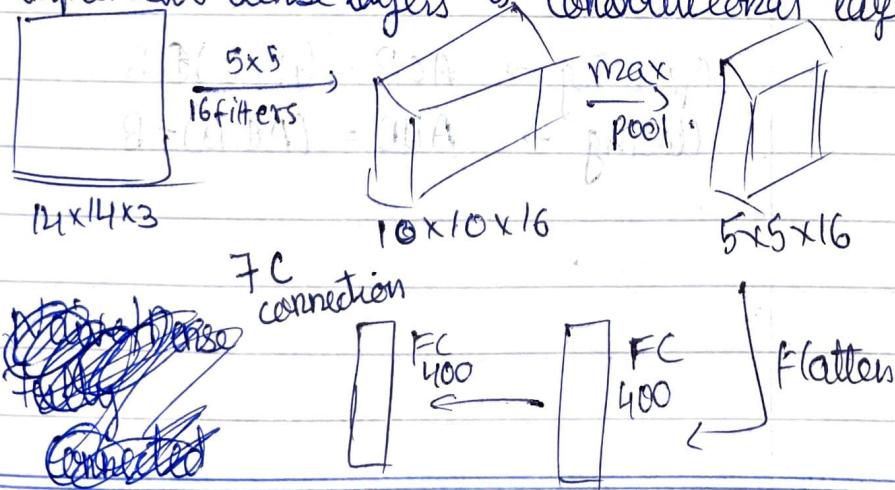


P.T.O.

* Object Detection

- Sliding Window Method
 - CNN trained on images
 - for new image, patches are taken.
 - CNN used on patch, if patch is classified as car, its boundaries are made into bounding box
 - Patch size may not cover entire object
 - Lots of computation as lot of forward passes during prediction — lots of patches taken depending on patch size
 - Not actual bounding box detection

- Convolutional implementation of sliding window
 - As patch sizes change, dense layers may throw error
 - Hence we use convolutional layers only or implement dense layers as convolutional layers

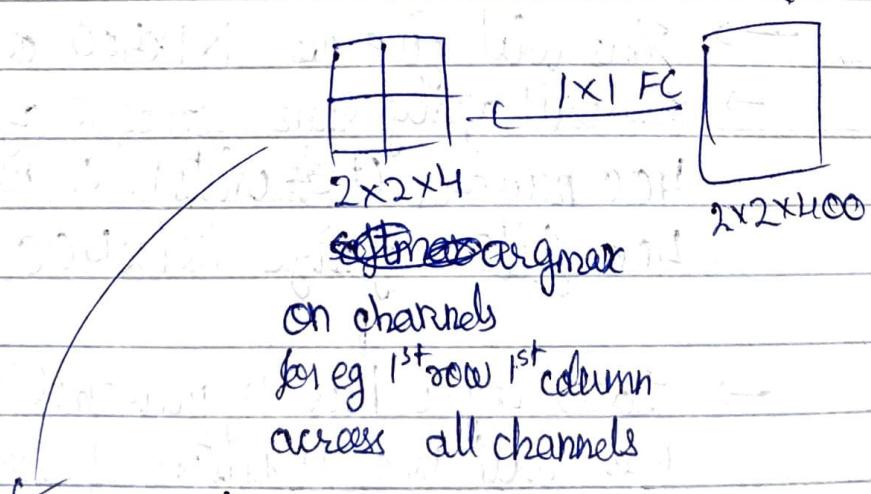
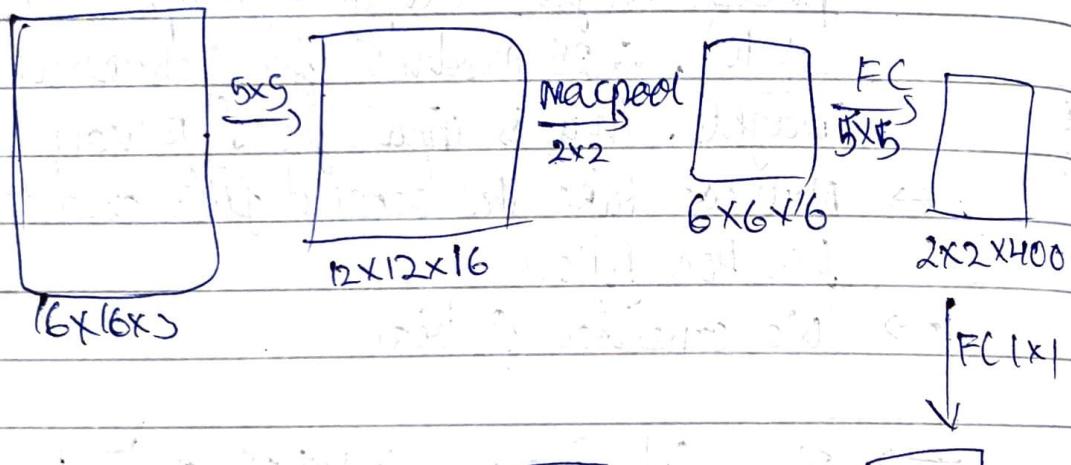
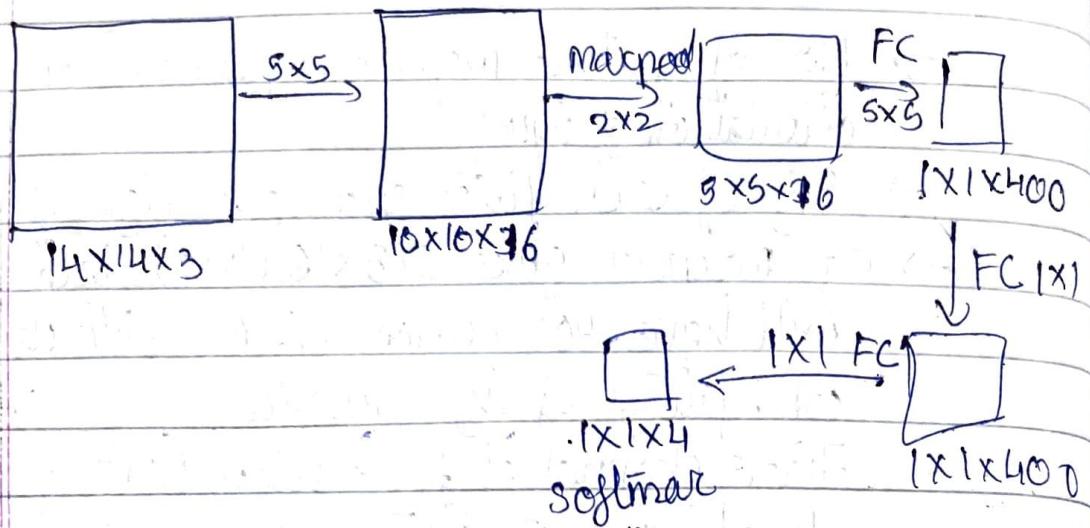


— आपका सम्मुख दृष्टिकोण वीरों को अचूक नहीं जाये।

DL

- conv implementation (continued)
- No of neurons in dense depends on spatial orientation of input
- No of feature maps is dependent on spatial orientation of input
- Size of each filter should be size of the source
- e.g. Maxpool gives $5 \times 5 \times 16$ (400) Then our FC will have 400 neurons. \therefore We will use 400 conv filters of size $5 \times 5 \times 16$
- First channel of feature filters (out of 16) is placed on first output of maxpool channel (out of 16). \therefore 25 products for 16 channels = 400 weights. This is input for 1st neuron
- Now we take the second filter and continue for 400 filters
- We consider 0 bias
- This will give us $1 \times 1 \times 400$ outputs.
- We will apply same concept again. We want 400 neurons (FC units). \therefore We will use 400 filters of size $1 \times 1 \times 400$
- Last layer has 4 neurons (softmax). \therefore We will use 4 filters of size $1 \times 1 \times 400$
- NO of parameters remain same for FC and Conv layers but conv input is flexible as it depends on previous layer size

→ Suppose input is implicit $14 \times 14 \times 3$ for training and $16 \times 16 \times 3$ for prediction:



This can be said as convolution on 4 patches of $14 \times 14 \times 3$

→ This allows input of any size. Then ~~segment~~ 2D matrix gives class of patches.

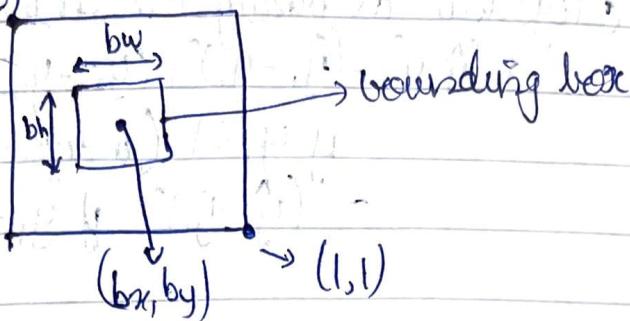
પ્રણુણે પ્રાર્થના એ શક્તિશાળી હૃદીયાર છે.

DL

Classification with localization

→ Regression approach for bounding box

→ : $(0,0)$



$bx, by \Rightarrow$ center of object (co-ordinates)

$bh \Rightarrow$ height of object (number value)

$bw \Rightarrow$ width of object (number value)

→ Consider 4 classes in an image!

1 → pedestrian

2 → car

3 → cycle

4 → background

Hence 4 neurons in final layer softmax.

We need 4 more neurons for bx, by, bh, bw

∴ Total 8 neurons in output layer

1st 4 → softmax

2nd 4 → bx, by, bh, bw in that order

→ This can be changed to $\langle P_c, bx, by, bh, bw, c_1, c_2, c_3 \rangle$ where $P_c = 1$ indicates object present. If $P_c = 0$ then no object is present so the other values do not matter. P_c is basically one hot vector of background

IOU used for eval
correct if $\text{IOU} > 0.5$

- the predictions can be said to be $\langle \hat{y}_1, \hat{y}_2, \dots, \hat{y}_8 \rangle$
- For training we use a different loss than categorical or binary crossentropy:
$$L(\hat{y}, y) = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2$$
 if $y_i = 1$
if $y_i = 1$: (image object is present)
 $= (\hat{y}_1 - y_1)^2$ if $y_1 = 0$ (object not present)
- ↪ A mix of categorical crossentropy and MSE loss can also be used as per the problem

- YOLO (You Only Look Once)
- Input image divided into grids ($19 \times 19 \times 8$ in original YOLO)
- Target is created for each grid cell.
- Here we consider 3×3 grid $\Rightarrow 9$ cells
- ∴ We will have $3 \times 3 \times 8$ (3 object classes, 1 background, 4 object detail values)
- Each grid will have extremes $(0, 0)$ and $(1, 1)$
- If two objects present then we consider center of the grid. The grid is classified as the object to which center grid belongs

P, F, O

P, F, O

Non-max suppression based bounding box

- Non-max suppression based bounding box
- Used when more than 1 bounding box on a single object.
- It retains only the best bounding box
- Each output is given by $\langle P_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3 \rangle$
- First discard bounding boxes with $P_c < \alpha$, where α is some threshold value (for eg=0.6)
- While there are any remaining boxes:
 - Pick the box with largest P_c value
 - Output that as a prediction
 - Discard any remaining box with $IoU \geq 0.5$ with the box output in the previous step

Non-max suppression based bounding box

Non-max suppression

Non-max suppression is a process of discarding overlapping bounding boxes. It is used to reduce redundant detections and improve precision.

Non-max suppression is a process of discarding overlapping bounding boxes. It is used to reduce redundant detections and improve precision.

- Anchor Boxes
 - When a grid has more than one object, we use anchor boxes.
 - Hyperparameter → No of anchor boxes.
 - If \neq types of objects are known, we can decide number of anchor boxes explicitly
 - Anchor box is a bounding box with a fixed orientation. Height, width can be changed
 - Suppose an image divided into 3×3 grids then it has ~~shape~~ shape $3 \times 3 \times 8$ (target value)
If 2 anchor boxes are assigned then the shape becomes $3 \times 3 \times 16$
 - In case of multiple objects in 1 grid, there must be a ^{type of} shape or other distinctive feature. If same objects have their center in same grid, then distinguishing them is not possible.
 - Each object in training image is assigned to grid based on image midpoint and the anchor box which has highest IOU with it is assigned the object

- Evaluation Metrics
 - True Positive (TP): ~~correct detection~~^{detection}. Detection with having $\text{IOU} \geq \text{threshold}$
 - False Positive (FP): wrong detection. Detection having $\text{IOU} < \text{threshold}$ (Ground truth does not have bounding box with sufficient IOU)
 - False Negative (FN): Ground truth not detected

Meditation is the best mode of worship.