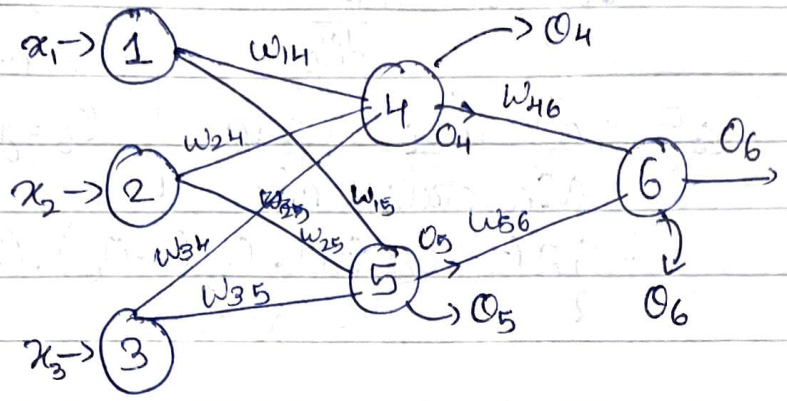
DL

* Feed - forward Networks (Vanilla networks)



→ Inputs - $\{x_1, x_2, x_3\}$

Target - T_6

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

Forward propagation

→ $net_4 = x_1 . w_{14} + x_2 . w_{24} + x_3 w_{34} + O_4$

$net_5 = x_1 . w_{15} + x_2 . w_{25} + x_3 . w_{35} + O_5$

→ $O_4 = f(net_4) = 1/(1+e^{-net_4})$

$O_5 = f(net_5) = 1/(1+e^{-net_5})$

→ $net_6 = O_4 . w_{46} + O_5 . w_{56} + O_6$

→ $O_6 = f(net_6) = 1/(1+e^{-net_6})$

Backward propagation

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

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

If $f(x) = 1/1+e^{-x}$
$f'(x) = f(x)(1-f(x))$
Here ~~$net_6 = 1/1+e$~~
$O_6 = f(net_6) = 1/1+e^{-net_6}$

→ For O_6:

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

$$\frac{dE}{dO_6} = \frac{dE}{dO_6} \cdot \frac{dO_6}{dnet_6} \cdot \frac{dnet_6}{dO_6}$$

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

→ For w_{14}:

$$w_{14} = w_{14} - \eta \frac{dE}{dw_{14}}$$

$$\frac{dE}{dw_{14}} = \frac{dE}{dO_6} \cdot \frac{dO_6}{dnet_6} \cdot \frac{dnet_6}{dO_4} \cdot \frac{dO_4}{dnet_4} \cdot \frac{dnet_4}{dw_{14}}$$

$$= \frac{1}{2} 2(T_6 - O_6)(-1) \cdot O_6(1-O_6) \cdot (1) \cdot w_{46} \cdot O_4(1-O_4) \cdot x_1$$

- **Vanishing Gradient Problem**

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

∴ Range of $f'(x) \in [0, 0.25]$

Hence for O_6, $\frac{dO_6}{dnet_6} \in [0, 0.25]$

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

For w_{14}, $\frac{dO_6}{dnet_6} \in [0, 0.25]$, $\frac{dO_4}{dnet_4} \in [0, 0.25)$

~~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 $k \in (0, 1)$

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

DL

* Convolutional Neural Networks

- Convolutional layers

→ Filter/Kernel/Feature Detector/Set of weights are applied onto images or mxn 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	1
0	0	1	1	0
0	1	1	0	0

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

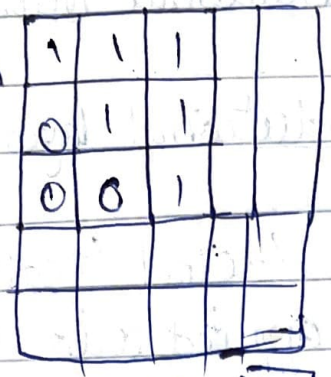
4	3	4
2	4	2
2	3	4

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

→ Working:

1	0	1
0	1	0
1	0	1

Filter



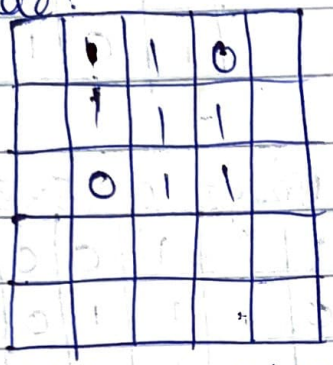
1x1	1x0	1x1
0x0	1x1	1x0
0x1	0x0	1x1

∴ 1+0+1+0+1+0+0+0+1 = 4 →

output

4		

Next stride:



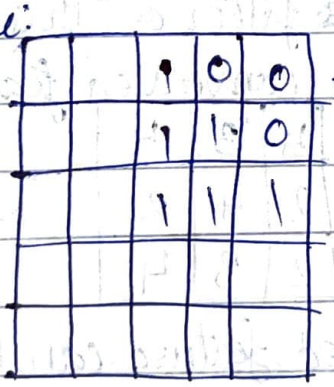
1x1	1x0	0x1
1x0	1x1	1x0
0x0	1x0	1x1

∴ 1+0+0+0+1+0+0+0+1 = 3 →

output

4	3	

Next stride:



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

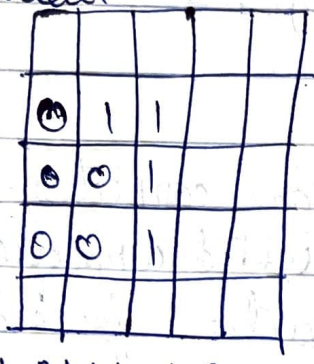
∴ 1+0+0+0+1+0+1+0+1 = 4 →

output

4	3	4

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

∴ Next stride:



⇒

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

$0+0+1+0+0+0+0+0+1=2$ ⇒

output

4	3	4
2		

and so on

→ 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, $W=5, F=3, P=0, S=1$

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

$= 3$

∴ 3x3 output

→ Fully connected layer - A layer 'i' is said to be fully connected if its all neurons are connected to the output of 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
∴ (25×9) weighted connections
9 bias connections for hidden layer
∴ (25×9)+9 = 234 parameters

→ For Conv: with a filter of 3×3
1 neuron of activation map is connected to (3×3=)9 neurons (filter on image) + its own bias
∴ For 1 neuron ⇒ (3×3)+1 = 10 connections
∴ For (3×3) 9 neurons of activation map ⇒ 10×9 = 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}

image

+

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

filter

→

n_1	n_2	n_3
n_4	n_5	n_6
n_7	n_8	n_9

output

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

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

Here the filter weights are shared, hence parameter sharing

→ When filter size < image size then there is said to 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 5x5x3. Therefore, we use a filter of depth 3 i.e. of dimensions fxfx3.

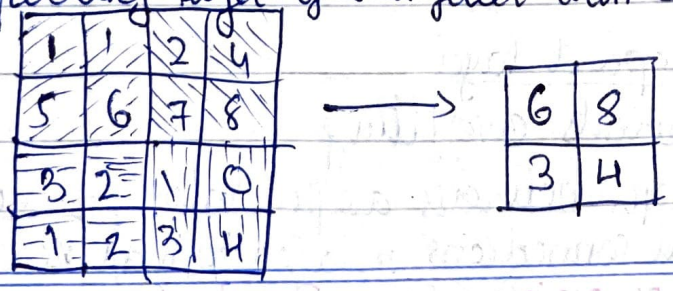
∴ for R channel fxfx0
G channel fxfx1
B channel fxfx2

Each depth inputs are added and 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 2x2 filter with stride 2



CNN can work on variable size images

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

→ 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 too ~~much~~ many neurons / cells.

- **Normalization layer**

→ Normalization layer is used to normalize
→ 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

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

→ Can be done in batch or mini-batch method

- **Dropout layer**

→ Prevents overfitting
→ Drops neurons as per ratio r (0.2, 0.3, etc)
→ All connections related to that neuron are dropped

* Softmax Activation function

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

→ $S(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{n} e^{y_j}}$ n is no of ~~classes~~ inputs

→ Consider $y = \{2.0, 1.0, 0.1\}$

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

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

∴ class 1 has max probability.

* Binary / Categorical cross entropy

→ Cross Entropy :

$$CE = -\sum_{i=1}^{|C|} t_i \log_2 (f(s)_i)$$

→ For binary problem: C_1 C_2

In → Target : 0 1

t_{0_1} t_2

→ NN → 0.3 $f(s)_1$

0.7 $f(s)_2$

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

$= -(0)\log_2(0.3) - (1)(\log_2(0.7))$

$= -\log_2(0.7)$

* 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 pedestrian 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 in ~~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

~~It is important~~

ii) Fine-tuning the ConvNet

→ Replace and retrain the classifier on top of the ConvNet on the new dataset, ~~test~~ 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 OG 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 OG dataset
→ Pretrained Net can be fine tuned (as large dataset)

iii) Small and different from OG dataset
→ Train linear SVM or shallow NN classifer
→ If pretrained Net is to be used then take features from earlier layers as they capture generic features. Layers closer to output capture specifics

iv) Large and different from OH dataset
- → 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 dissimilars
→ tasks are not well related

DL

* 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

DL

* Autoencoders

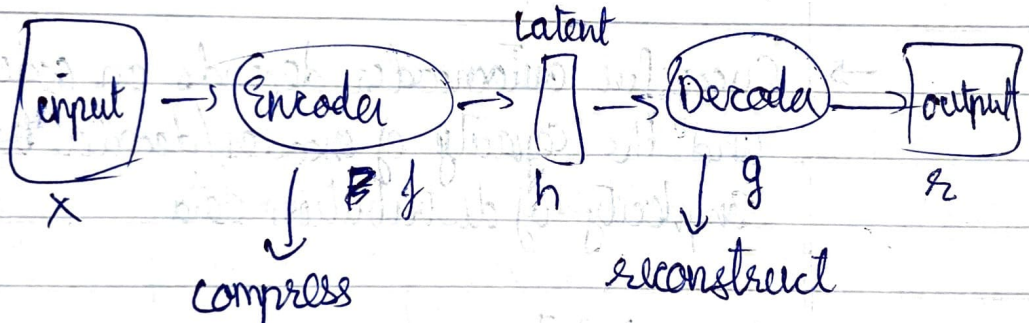
→ Unsupervised Neural nets

→ 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

→ 

$h = f(x)$ ⇒ encoder ⇒ input to latent repr.

$r = g(h)$ → decoder ⇒ latent repr. to reconstruct

∴ 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 'h' to have small 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 also 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

→ Succesful autoencoder 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.

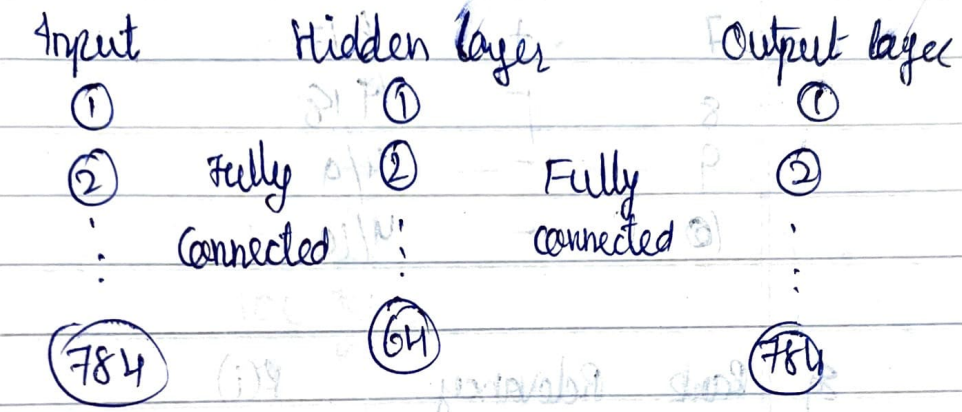
* Vanilla Autoencoder

→ Simplest form — three layer net → input, 1 hidden layer, output

→ Fully connected dense layers

→ Adam optimizer and MSE loss are used

Eg:

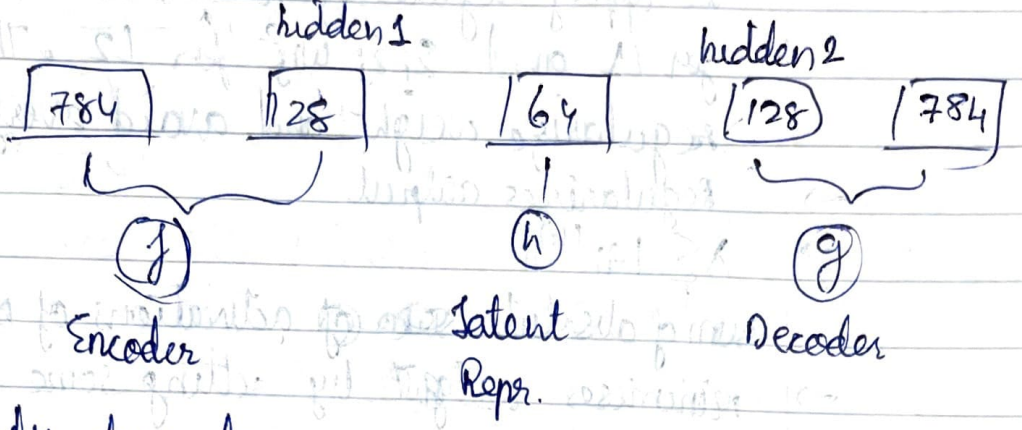


→ Once quality ~~and~~ of output is as per desire, decoder can be removed. Now latent layer acts like a feature extractor for input

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

* 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:

i) Sparse AE.

ii) Denoising A.E.

615

P.T.O.

- Sparse Autoencoder term
  → we apply regularization like $\Sigma_i \Sigma_j |w_{ij}|$ for L1 and $\Sigma_i \Sigma_j w_{ij}^2$ for L2. This helps regularize weight and avoid overfitting
  → Regularizes output
  → $\lambda \sum_i |a_i^{(h)}|$
  sum of absolute activations of neurons
  → minimises by setting some neuron output to zero

- Denoising autoencoders

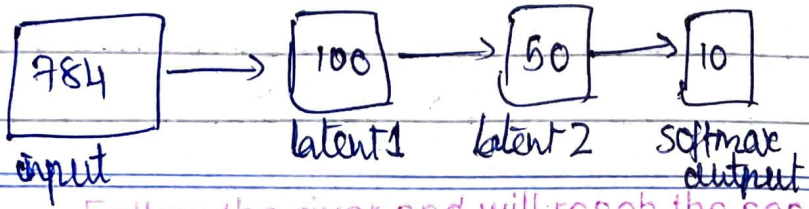
→ Another way to force AE to learn useful features, is to add random noise to its inputs and making it recover original noise-free data
→ AE cant simply copy the input to output due to presence of random noise
→ Subtract noise and reconstruct underlying input
→ The architecture is same, only inputs differ

* 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:



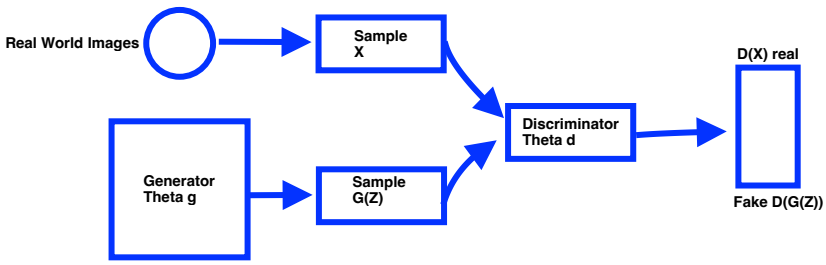
DL

* 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 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 fail discriminator
→ Whichever component fails, loss is used to improve performance

→ 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 ~~upsampler~~ upsampler
Discriminator is downsampler

→ 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 mini batch size 50. Draw 50 sample of original images and sample 50 from generated images
→ Input to discriminator is 100 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)

P.T.O.

- 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 back~~ is ~~used~~ backprop to generator. Weights are updated for generator only

- Equations

→ x is sampled from real data
P_{data} → real data
z is sampled from fake data
$P(z)$ → fake data.

→ $G_{og}(z)$
~~$G_{og}(z)$~~ → generating
$D_{od}(G_{og}(z))$ → identify fake as real
$1 - D_{od}(G_{og}(z))$ → identify fake as fake

→ Alternate between:

i) Gradient ascent on discriminator (∵ maximize)

$$\max_{D_d} \left[E_{x \sim P_{data}} \log D_{od}(x) + E_{z \sim P(z)} \log \left(1 - D_{od}(G_{og}(z))\right) \right]$$

log likelihood of real identified as real

log likelihood of fake identified as fake

→ maximize identifying fake as fake and real as real

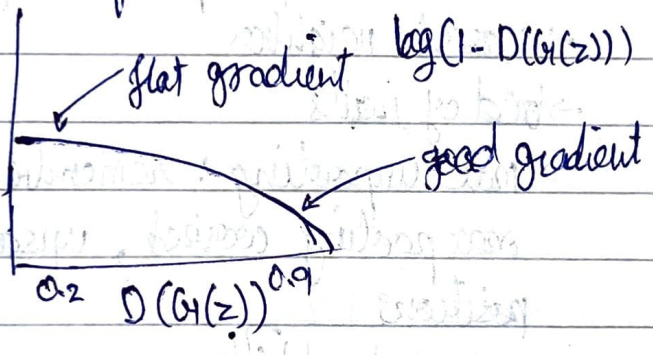
ii) Gradient descent on generator: (∵ minimize)

$$\min_{\theta_g} \left[E_{z \sim p(z)} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)\right]$$

log likelihood of fake identified as fake

→ ~~from~~ minimize identifying fake as fake

→ Problem with this loss function is that when generator is not performing well, gradient information (error) is ~~not~~ poorly available. When generator perform well, gradient information is available which is not required



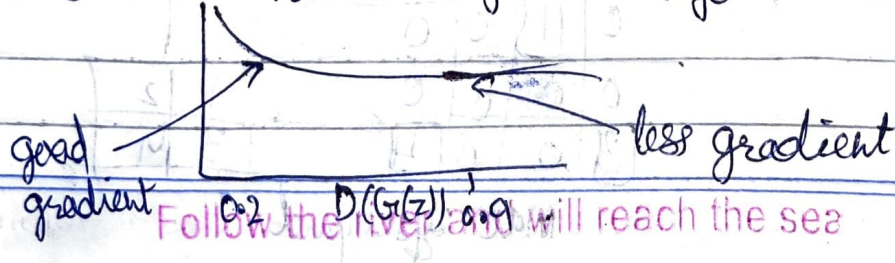
→ Therefore we change the loss function of generator to:

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

→ Works better and better gradient info available

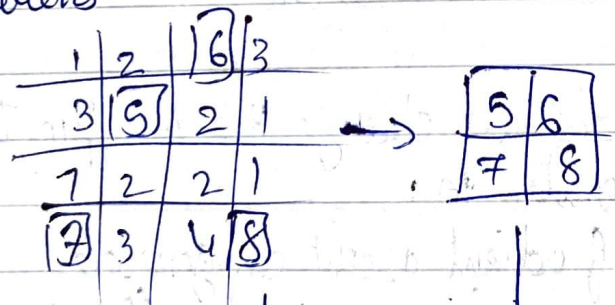


→ GAN training algorithm: refer ppt

* DCGAN
→ Deep Convolutional GAN
→ Generator is upsampler with fractionally-strided convolutions
→ Discriminator is a convolutional network

→ For architecture, refer ppt.

→ In-network upsampling / unpooling:
- nearest-neighbor
- bed of nails
- max unpooling: remember positions on which max pooling worked, upsample to those same positions



DC

→ If asymmetric padding (2 dims only) then $p = 1/2$

→ ~~Learnable Upsampling: Transpose convolution~~

→ Stride : ratio of movement in input and output

• Learnable Upsampling : transpose convolution

→ fractionally strided conv., Backward strided conv., upconvolution, deconvolution (bad name)

→ Here stride = p/q (fraction) ∴ fractionally strided. Eg $s = 1/2$, 1 in input, 2 in output

→ Transpose conv with 3x3 filter on 2x2 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 x filter
∴ 4 filters of 3x3, with 1, 2, 3, 4 in all 3x3 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 inp ⇒ 2 in out

1	1	1+2	2	2	
1	1	1+2	2	2	
1+3	1+3	1+2+3+4	2+4	2+4	
3	3	3+4	4	4	
3	3	3+4	4	4	

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

→ GANS/DCGANS 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 → King vector
  V_m → Man vector
  V_w → Woman vector

Interpretable vector math can be considered as:

$$V_k - V_m + V_w = \hat{V}_q \text{ (Queen vector)}$$

We use the word embedding layer on queen and find V_q
Difference between V_q and $\hat{V}_q$ gives us similarity between desired result and actual result

DL

* 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 ~~binary~~ 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 ~~objects~~ 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.

→ Complexity hierarchy:

Image Classification
↓
Classification + Localization
↓
Object Detection
↓
Semantic Segmentation
↓
Instance segmentation

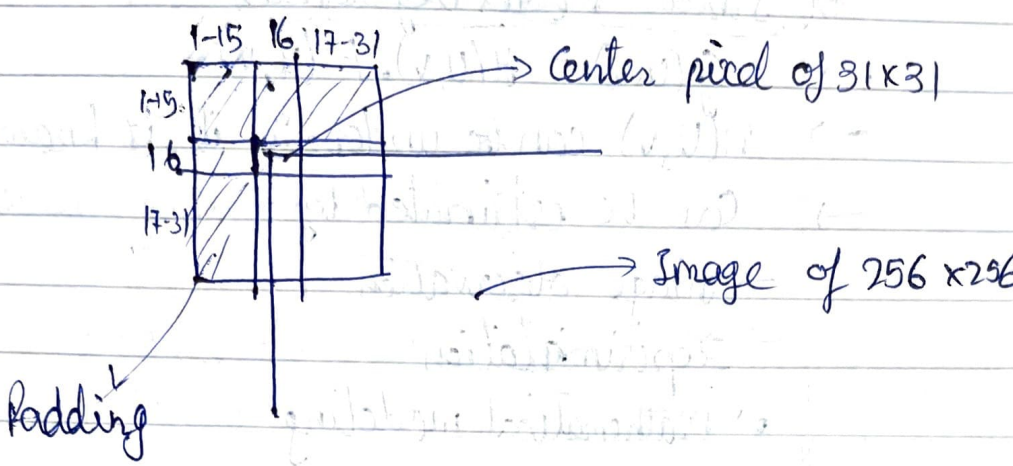
* Semantic Segmentation

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

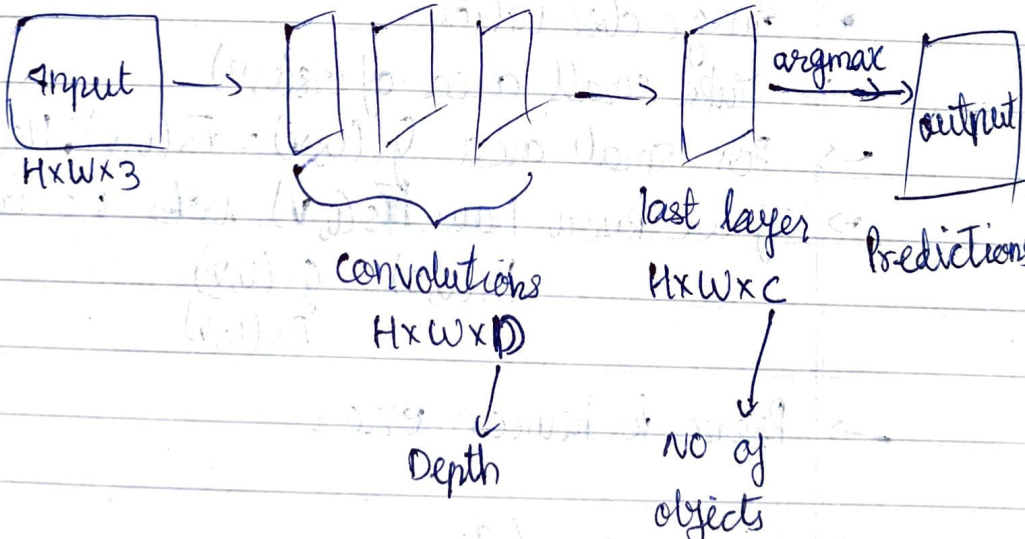
• Sliding Window

→ Considering of 256 x 256, & window of 31 x 31
→ Patch is extracted, entire patch is fed into CNN, class of patch is the class for CENTER PIXEL only.
→ Computationally expensive as 256 x 256 forward propagations required in Training alone. Prediction
→ 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



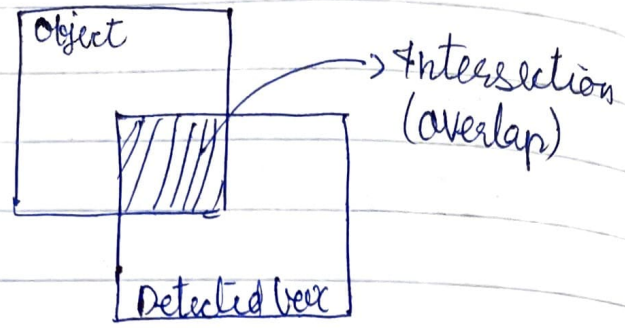
- Fully Convolutional Neural Network
→ All layers are convolutional
→ Spatial extent remains unchanged
→

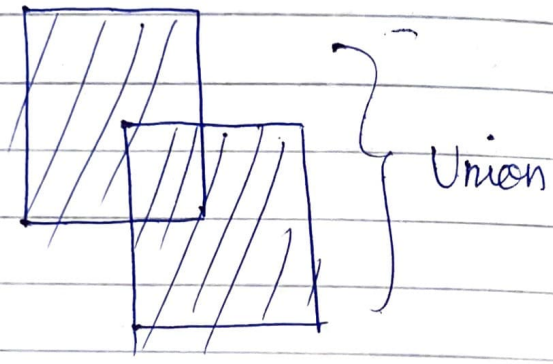


→ Convolutional on original ~~layer~~ image layers is costly

* Intersection over Union

→ IOU = Area of overlap / 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 props during prediction — lots of patches taken depending on patch size

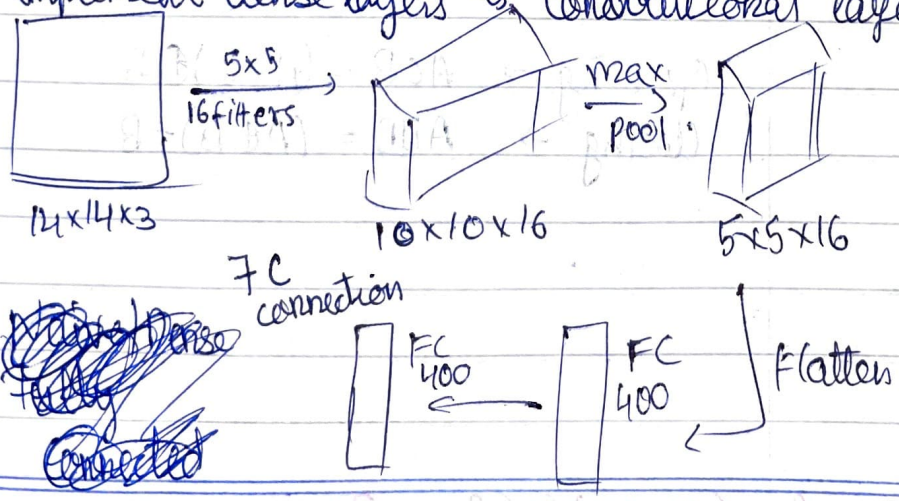
→ Not actual bounding box detection

- 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

→



7C connection

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

→ eg maxpool gives 5×5×16 (400) then our FC will have 400 neurons ∴ we will use 400 conv filters of size 5×5×16

→ First channel of ~~feature~~ filter (out of 16) is placed on first output of maxpool channel (out of 16). ∴ 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

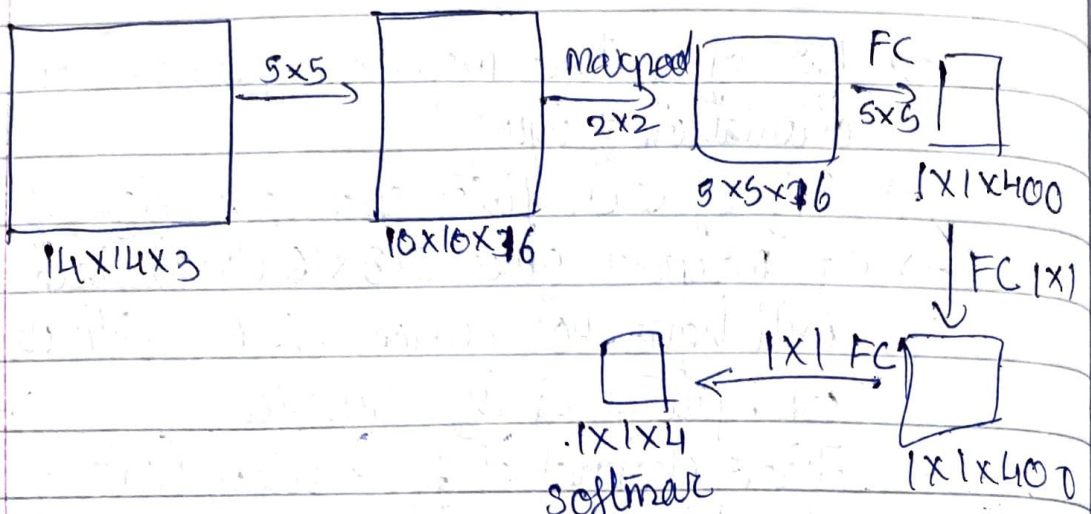
→ This will give us 1×1×400 outputs.

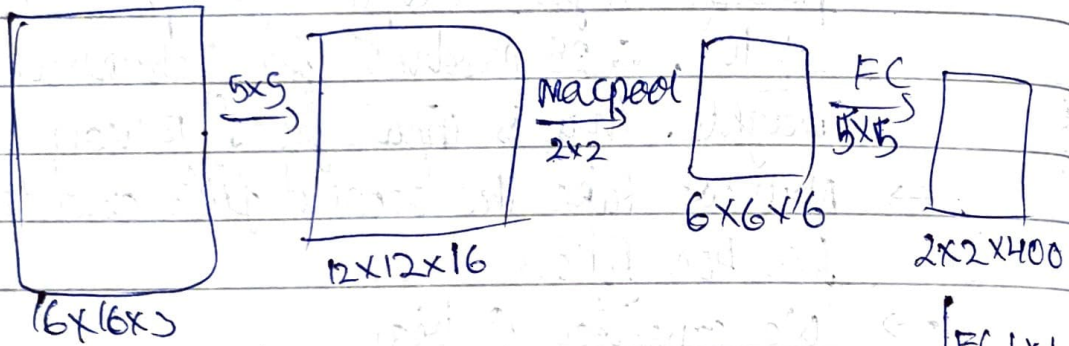
→ We will apply same concept again. We want 400 neurons (FC units). ∴ We will use 400 filters of size 1×1×400

→ Last layer has 4 neurons (softmax). ∴ We will use 4 filters of size 1×1×400

→ 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 14x14x3 for training and 16x16x3 for prediction:





~~softmax~~ argmax on channels for eg 1st row 1st column across all channels

This can be said as convolution on 4 patches of 14x14x3

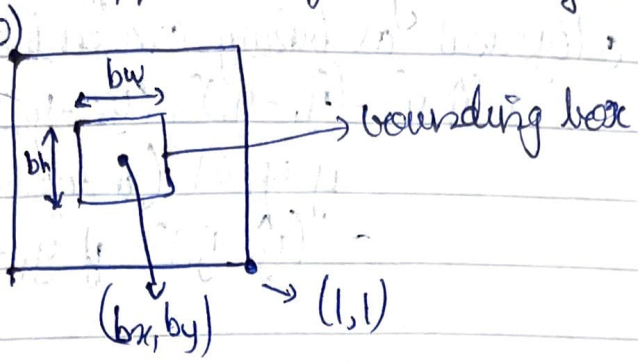
⇒ This allows input on any size.. Then ~~argmax~~ 2D matrix gives class of patches.

DL

- ✱ Classification with localization

→ Regression approach for bounding box.

→



b_x, b_y => center of object (co-ordinates)
b_h => height of object (number value)
b_w => width of object (number value)

→ Consider 4 classes in an image:
1 → pedestrian
2 → car
3 → cycle
4 → background
Hence 4 neurons in final for softmax.
We need 4 more neurons for b_x, b_y, b_h, b_w
∴ Total 8 neurons in output layer
1st 4 → softmax
2nd 4 → b_x, b_y, b_h, b_w in that order

→ This can be changed to $\langle P_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3 \rangle$ 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 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 $y_1 = 1$~~

if $y_1 = 1$ (~~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 is divided into grids (19×19×8 in original YOLO)
→ Target is created for each grid cell.
→ Here we consider 3×3 grid ⇒ 9 cells
∴ We will have 3×3×8 (3 object classes, 1 background, 4 ~~img~~ object detail values)
→ Each grid will have extremeties: (0,0) and (1,1)
→ If two objects present then we consider center of the grid. The ~~of~~ grid is classified as the object to which center grid belongs

~~P.T.O~~

P.T.O

- Non-max suppression

→ Used when more than 1 bounding box on a single object.

→ Retains only the best bounding box

→ Each output is given by $< P_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3 >$

→ First discard bounding boxes with $P_c < x$ where x 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 ≥ 0.5 with the box output in the previous step

DL

- Anchor Boxes

→ When a grid has more than one object, we use anchor boxes

→ Hyperparameter - No of anchor boxes.

→ If # 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 ~~input~~ shape 3×3×8 (Target values) If 2 anchor boxes are assigned then the shape becomes 3×3×16

→ In case of multiple objects in 1 grid, there must be a shape or other distinctive feature. If same type of 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 ~~decision~~ detection. Detection with having IOU > threshold

→ False Positive (FP): wrong detection. Detection having IOU < threshold (Ground truth does not have bounding box with sufficient IOU)

→ False Negative (FN): Ground truth not detected