Kinjal Kachi
Assignment-3

Computer Vision
Fall 2020
ANumber A20449343

Computer Vision:

1) Neural Networks

a) Template matching interpretation.

- The linear classifier is given as the function of weights and input as :-

$$\hat{y} = f(x, \theta) = \underbrace{\theta^T}_{K \times n} \underbrace{x}_{n \times 1} + \underbrace{\theta_0}_{K \times 1}$$

- Template matching is used to find similarity between the feature vector and templates
- Rows at $\theta^T$ are templates. If we have k different classes we have k templates representation as ${\theta^T}_{k \times n}$.
- $\theta^T x$ measures how well x matches with the k templates by performing dot product.
- The high the values of $\theta^T x$ with a template from all k-templates, high the membership in $k^{th}$ class.

The matrices are shown as follows:

$$\hat{y} = \begin{bmatrix} n_0^T \\ \vdots \\ n_n^T \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_0 \\ \vdots \\ d_k \end{bmatrix} = \begin{bmatrix} n_1 \cdot x \\ \vdots \\ n_k \cdot x \end{bmatrix} + \begin{bmatrix} d_0 \\ \vdots \\ d_k \end{bmatrix}$$

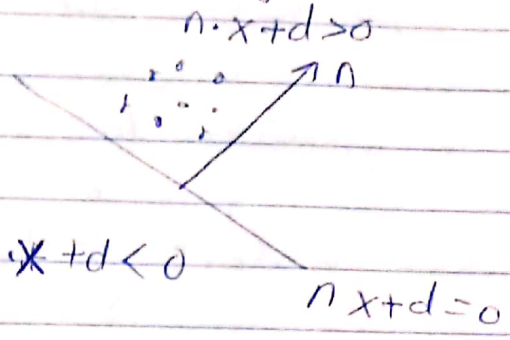
Decision Boundary Interpretation.

- Here we calculate the distance from the decision boundary to classify the feature.
- Rows of $\theta^T$ are parameters of k linear discriminant

function

- Each linear discriminant separates one class from "all others"
- Positive values for the function says that the examples belongs to that class and negative value suggests that it doesn't

$$\hat{y} = \begin{bmatrix} n_1 x \\ \vdots \\ n_k x \end{bmatrix} + \begin{bmatrix} d_0 \\ \vdots \\ d_k \end{bmatrix}$$




n = normal
d = distance from origin

(b) Converting similarity score to probability

- Two class classification.

Here we use sigmoid function to classify the example into one of the binary class. It takes the features as input & outputs the probability

$$\hat{y}_j^{(i)} = P(y=j \mid x^{(i)}) = \text{sigmoid}\left(s_j^{(i)}\right)$$

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

k - classifier

Here use ~~softmax~~ activation function.

$$\hat{y}_j^{(i)} = P(y=j \mid x^{(i)}) = \frac{\exp\left(s_j^{(i)}\right)}{\sum_{i=1}^{k} \exp\left(s_i^{(i)}\right)}$$

(c) Loss functions:-

$$L_1 : L_i(\theta) = \sum_{j=1}^{k} \left| \hat{y}_j^{(i)} - y_i^{(i)} \right|$$

where,

$$L_2 : L_i(\theta) = \sum_{j=1}^{k} \left( \hat{y}_j^{(i)} - y_j^{(i)} \right)^2$$

Huber loss:

$$L_i(\theta) = \sum_{j=1}^{k} S_\sigma \left( \hat{y}_j^{(i)} - y_j^{(i)} \right)$$

- Cross Entropy loss:

$$L(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log \left( \hat{y}_j^{(i)} \right)$$

d) Regularization:

We add regularizaⁿ to the loss function in order to get a simpler solution i.e lower weights. It helps the model generalize better:-

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} L_i\left(f(x_i; w), y_i\right) + \lambda R(\theta)$$

Here, $R(\theta)$ is regularization term

$\lambda$ = weight of regularizaⁿ (Hyperparameter)

(e) In gradient descent, we try to find the minima of the function.

- We take the gradient to decide the direction in which we need to move forward.

- The gradient of the function point into the direction where the function changes the most.
- In order to find minima, we move in opposite direction of the gradient

(f) Gradient Descent can be slow when there are lot of parameters and examples in data
- It updates the new parameters by looking at each example for every iteration
- Update is more frequent in gradient descent.
- On the other hand, stochastic gradient descent (SGD) uses different approaches.
- It looks at each eg. and updates the parameters on the go.
- SGD takes less time to converge than gradient descent.
- The problem with SGD is that as it is taking each example & updating parameters, it would not converge in the right values.

| SGD | Minibatch | GD |
|---|---|---|
| Update after every example | Update after each minibatch | Update after looking at all examples |

(g) learning rate:
- Learning rate values are hyperparameters for gradient descent & can be made smaller as iteration progresses. It is called 'learning rate decay'. We can use following approaches.

(i) Step decay:

With every iterations k, we reduce the learning rate by some specified function.

(2) Expotential decay

The learning rate is decayed exponentially with each iteration.

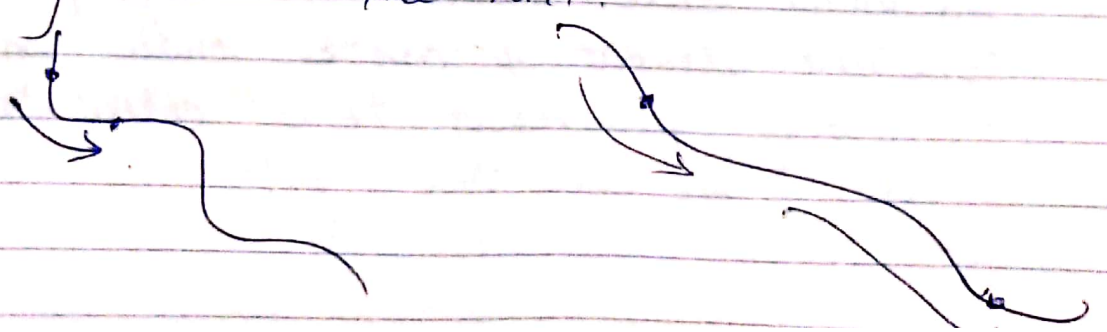
$n_\lambda = n_0 d^{-k/t}$.

(3) Fractional decay $n_t = n_0/(1+kt)$

With each iteration, reduce the learning rate by some fraction.

(4) Momentum.

- We can add momentum on top of gradient descent to obtain better results.
- If the momentum is not added, GD could get ~~Stuck~~ Stuck at the local minimum & coverages at that point.
- Adding momentum helps the algorithm understand to keep moving in the direction of descent even if the optimal values according to the equation are found
- This phenomenon is a similar to a ball rolling down the hill.



(i) The backpropagation algorithm calculates the gradients of the loss function with respect to the network's weights.

- We start with some initial guess for the weights and then update these weights using gradient descent for each node.
- Forward Pass

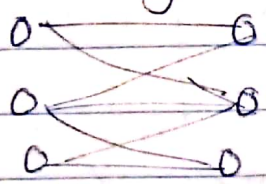
In forward pass, we simply push the input to compute all the intermediate values at the nodes. By doing this, we have all the required values to calculate the gradients using chain rule for each node.

- Backward Pass

In backward pass we push the gradient towards the beginning starting from the end node. We calculate the derivate at each node wrt loss using chain rule. In the backward pass we update the weights using gradients to minimize the loss function.

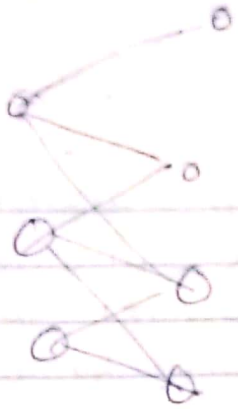
- In backward pass, gradients are propagated towards the beginning from the end node in order to update the networks node parameters.

(ii) In a fully connected layer, each neuron receives input from every element of the previous layer.



Fully connected layer

In a convolutional layer, neuron ~~rete~~ receives input from only a restricted subarea of previous layer.



Convolutional layer.

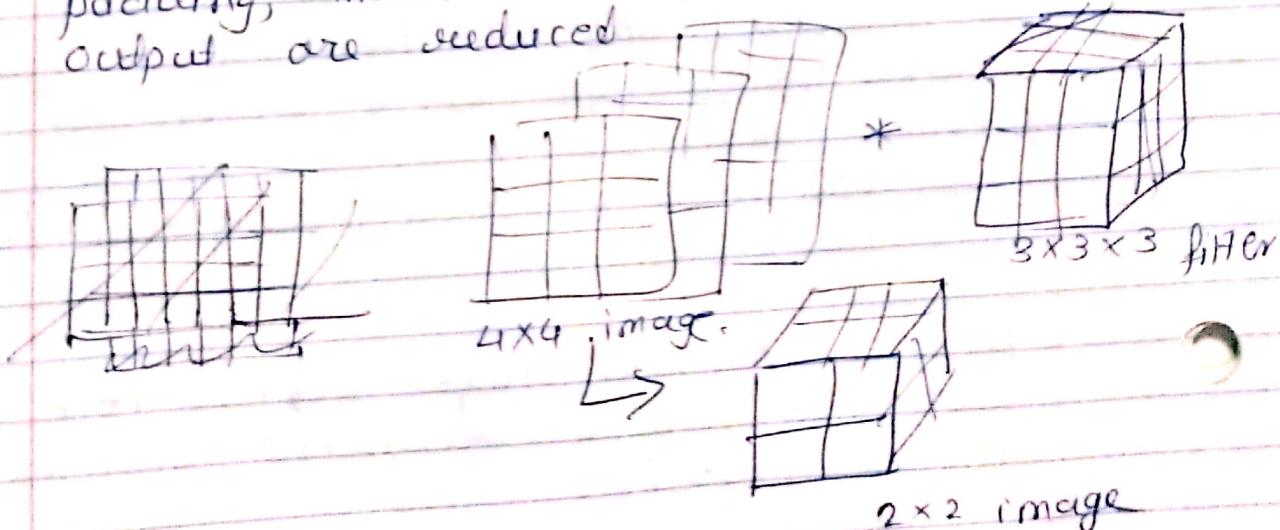
(k) Dropout

- Dropout means we remove some units in the layers of neural network during training randomly.
- At each training stage, individual nodes are either dropping out of the network with probability $1-p$, so that the reduced network is left.
- We we dropout in order to prevent overfitting & avoid co-dependency among neurons, improve training speed.

Q2] Convolutional Neural Networks

(a) I is a 4×4 RGB image
filter is 3×3×3 matrix with all 1s

- Convolute without zero padding.
When we apply the filter without zero padding, the number of dimensions of output are reduced



As we convolve the filter with image, we get 2×2×3 matrix as output. 2×2 is result of filter with the respective channel of the input image

R =

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

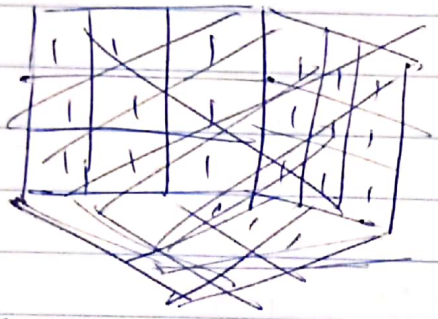
G =

| | | | |
|---|---|---|---|
| 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 2 |

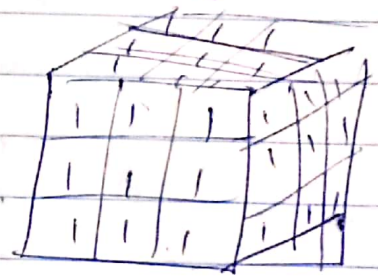
B =

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 |

filter -



All 1s



3×3×3

The results for each channel, RGB are.

| | R | G | B | final |
|---|---|---|---|---|
| (1, 1) | 9 | 18 | 18 | 45 |
| (1, 2) | 9 | 18 | 18 | 46 |
| (2, 1) | 9 | 18 | 27 | 54 |
| (2, 2) | 9 | 18 | 27 | 54 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 9 | + | 18 | 18 | + | 18 | 18 | → | 45 | 45 |
| 9 | 9 | | 18 | 18 | | 27 | 27 | | 54 | 54 |

output w/o zero padding

(b) Convolution with zero padding for R channel looks like.

R

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Other channels (G and B) will look similar with shape 6X6.
Similarly apply the filter as we did in Q 2(a), we get results for each channel as follows

R =

| | | | |
|---|---|---|---|
| 4 | 6 | 6 | 4 |
| 6 | 9 | 9 | 6 |
| 6 | 9 | 9 | 6 |
| 4 | 6 | 6 | 4 |

+

G =

| | | | |
|---|---|---|---|
| 8 | 12 | 12 | 8 |
| 12 | 18 | 18 | 12 |
| 12 | 18 | 18 | 12 |
| 14 | 21 | 21 | 14 |

Adding up values of each channel to get final output

| | | | |
|---|---|---|---|
| 18 | 27 | 27 | 18 |
| 30 | 45 | 45 | 30 |
| 36 | 54 | 54 | 36 |
| 26 | 39 | 39 | 26 |

Output with p zero padding

(c) Atrous convolution with dilation rate of 2. Lets do this for each channel & then add them to get final output

$R =$

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

when convolving with filter with all ones, we get,

Iteration 1: $(0\times1)+(0\times1)+(0\times1)+(0\times1)+(1\times1)+(1\times1)+(0\times1)+(1\times1)+(1\times1)=4$

Iteration 2: 4
Iteration 3: 4
Iteration 4: 4

$R =$

| | |
|---|---|
| 4 | 4 |
| 4 | 4 |

Similarly :-

$G =$

| | |
|---|---|
| 8 | 8 |
| 8 | 8 |

$B =$

| | |
|---|---|
| 12 | 12 |
| 8 | 8 |

final output = R+G+B

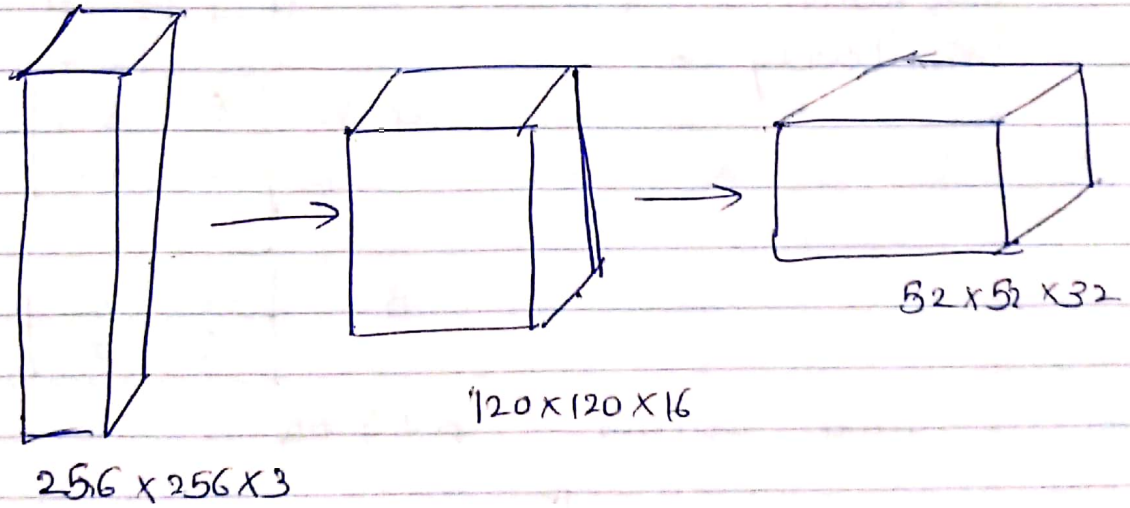
| | |
|---|---|
| 4+8+12 | 4+8+12 |
| 4+8+12 | 4+8+8 |

Q2d) Template matching is the technique used to find small parts of an image which matches template image.
It slides the ~~win~~ window across image that will provide a percent match with the template. When image is convoluted with the filter, it is expected to give portions of the image that match the ~~portio~~ particular template to give a higher response

Q2 e) Multi-Scale analysis can be achieved ~~to~~ with a fixed window size by reducing an image size by continuously convolving it with filter. This will lead to a larger receptive field in the convluted layer., better feature extraction and accounts for differently scaled objects.
eg: We start with 8x8 size image and fixed window size of 2x2. After first convolution, we get 4x4 output, running another convolution gives an image ~~get~~ of 2x2. As the scale of image reduces the objects in the image get more receptive area and would actually lead to feature extraction.

(f) When applying convolution to an image, the size of the image is reduced the spatial dimension due to pooling or stride

- If ~~we~~ we let the spatial dimension reduces we lose information from the image.
- In order to do so, we increase the depth of the image to compensate for spatial resolution decrease.
- This is done by applying number of filters to get the depth of n.
- As the depth increases, we have more no. of coefficients which represent the information in the image at each channel.
- Each filter we apply store a specific information in the image & these n filters stacked as whole give the required information with reduced spatial dimension



(g) Input = 128 x 128 x 32 No. of filters = 16
filter size = 3 x 3 x 3
The output size of tensor ~~fto~~ after convolution is given by:-

$$O = \frac{I - f + 2P}{S} + I$$

I = Input size of input
f = filter size
P = Padding
S = ~~Sth~~ Stride

$$\therefore O = \frac{128 - 3 + 2(0)}{1} + 1 = 126$$

∴ Size of output = 126 x 126 x 16
∴ (As ~~use~~ we use 16 filters)

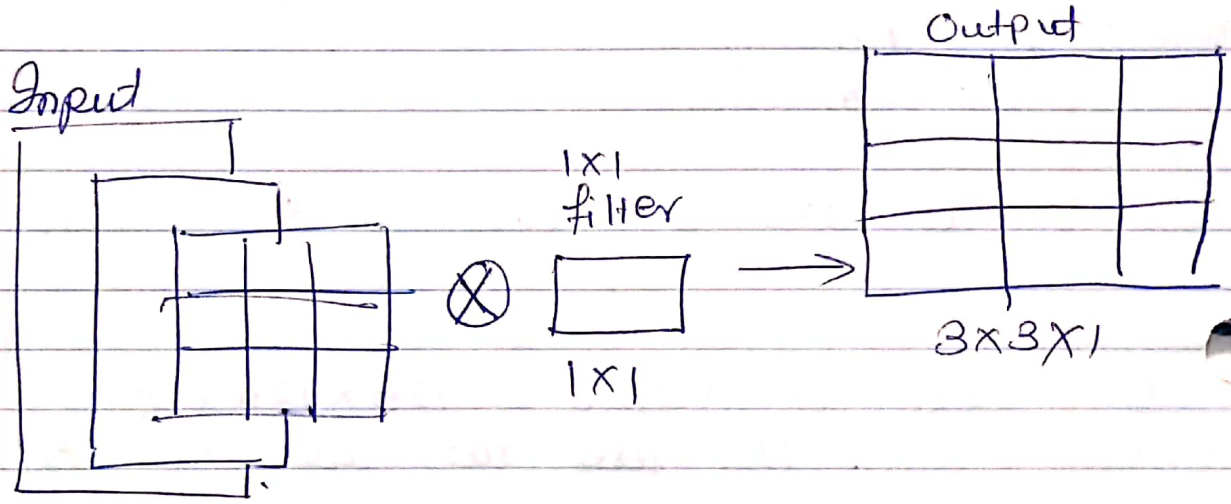
(h) With stride of 2, $O = \frac{128 - 3 + 2(0) + 1}{2}$

$= 63.5$

**Size of output tensor = 64 x 64 x 32**

(i) 1 x 1 convolution
We apply (x) convolution to reduce the number of channels in the image.
A 1x1 filter will only have a single parameter of weight for each channel in the input and like the application of any filter results a single output value.

- This allows 1x1 filter to act like a neuron with an input from the same position across each of the feature maps in the input.
- A single neuron can be convolved along the image resulting in a feature map with same width & height as input. & reduced channel to 1.




(k) Pooling

- Pooling layer is added in the neural network to downsample spatial dimensions of the input from previous layer.
- The ~~pix~~ pooling operater is always almost of size 2x2. It is applied with stride of 2 pixels.
- The result of using a pooling layer & creating downsampled features is a summarized versions of the features detected in the input.
- Pooling supports multiple scale analysis & ~~its~~ it helps in reducing the number of coefficients.

- This in turn results in fewer calculations in training phase to find optimal weights for each neuron.

(j) Max. Pooling on image I.
Max. pooling filter is of dimension 2×2 & applied with stride of 2.
We will apply this filter with each channel respectively.

R =

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Max. pooling →

| | |
|---|---|
| 1 | 1 |
| 1 | 1 |

G =

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Max-Pooling →

| | |
|---|---|
| 2 | 2 |
| 2 | 2 |

B =

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 |

Max. Pooling →

| | |
|---|---|
| 2 | 2 |
| 4 | 4 |

Now we have values for each channel, we take max. value from it to get the final output

R

| | |
|---|---|
| 1 | 1 |
| 1 | 1 |

G

| | |
|---|---|
| 2 | 2 |
| 2 | 2 |

| | |
|---|---|
| 2 | 2 |
| 4 | 4 |

⇒ Max pooling output

| | |
|---|---|
| 2 | 2 |
| 4 | 4 |