Review Questions (Assignment-4)

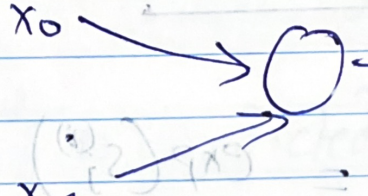
Name: Akshay Jain
StudentID: A20502846
Computer Vision AS4 Writeup (Review Q's)

Neural Networks

Sol^n (a) Template matching refers to the image processing where we find similar templates in a source image by giving a base template to compared on. The process of template matching is done by comparing each of the pixel values of the source image one at a time to the template image.

Let's take a Neural Network such that:

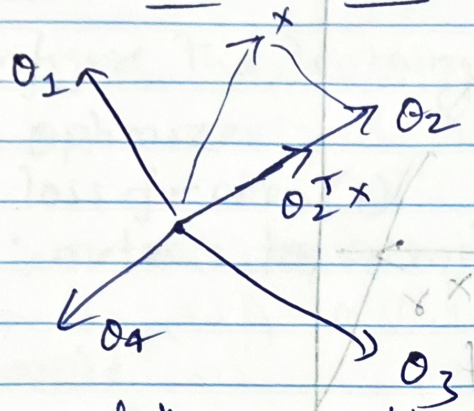
Having x_0



input $x_0 x_1 \dots x_n$
& output y

$$\hat{y} = f(x_j, \theta) = \theta^T x + \theta_0 \quad \text{— eq}^n \text{①}$$

⇒ Template Matching Interpretation can be understand using below fig. & (1) eqn ↓ Linear classifiers



let's say

↳ the rows of the weight matrix O^T represents templates

↳ with K rows we have K templates (one template per class)

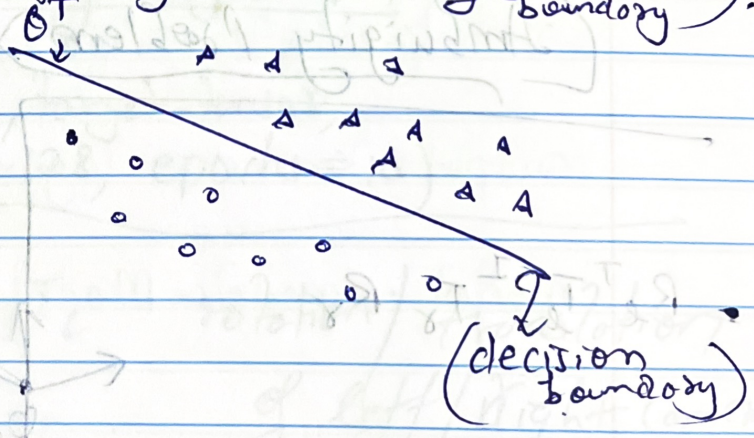
↳ the product $O^T x$ measure how well the input vector matches the template (dot product between vectors)

↳ high similarity to the template of a particular class indicates membership in this class.

⇒ Decision Boundary Interpretation

Similarly, we can define O^T for decision boundary interpretation that separates different examples from different classes by defining a boundary (or decision boundary).

Here; O^T separates class 'o' & 'Δ' by defining a boundary.



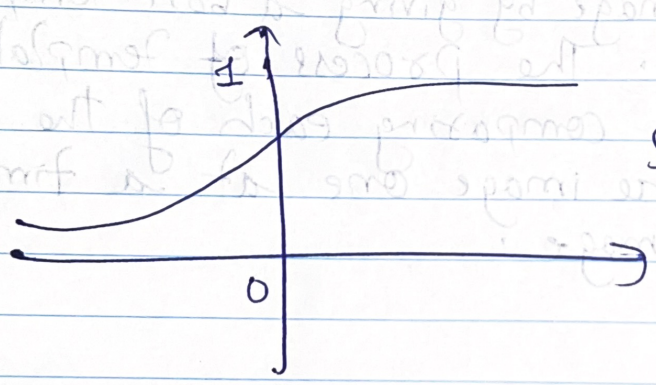
Sol^n(5) (i)(b) **Converting similarity score to probability**

Given similarity score (or decision boundary distance S_j) from a linear classifier (higher better), Compute:

$$\hat{y}_j^{(i)} = P(y=j / x^{(i)})$$

↳ In a **2-class classification case:**

$$\hat{y}_j^{(i)} = P(y=j / x^{(i)}) = \text{Sigmoid}(S_j^{(i)})$$



$\text{Sigmoid} = \frac{1}{1+e^{-x}}$

(Activation function for binary classification)

↳ In a **K-class classification:**

$$\hat{y}_j^{(i)} = P(y=j / x^{(i)}) = \frac{\exp(S_j^{(i)})}{\sum_{l=1}^{K} \exp(S_l^{(i)})}$$

(Softmax)

So, Softmax is used for binary classification & sigmoid function is used for multiclass classification.

Soln (I) (c)

Expression of L1 & L2 loss :

L1 & L2 loss are used to minimize error b/w known & predicted values in regression problems.

$$L1: L_i(\theta) = \sum_{j=1}^{K} |\hat{y}_j^{(i)} - y_j^{(i)}|$$

$\hat{y}_j^{(i)}$: $y_{known/true}$; $y_j^{(i)}$: $y_{predicted}$

where as;

$$L2: L_i(\theta) = \sum_{j=1}^{K} \left[\hat{y}_j^{(i)} - y_j^{(i)}\right]^2$$

$\hat{y}_j^{(i)}$ is predicted value & $y_j^{(i)} \in R$.

Huber Loss

As per wikipedia, In statistics the huber loss is a loss function used in robust regression that is less sensitive to outliers in data than the squared error loss & is given by:

$$L_\sigma(d) = \begin{cases} \frac{1}{2} d^2 & \text{if } d \le \sigma \\ \sigma\left(d - \frac{1}{2}\sigma\right), & \text{otherwise} \end{cases}$$

L_σ is "quadratic for small d"

"linear for large d"

with equal values & slopes of the different section at the two points where

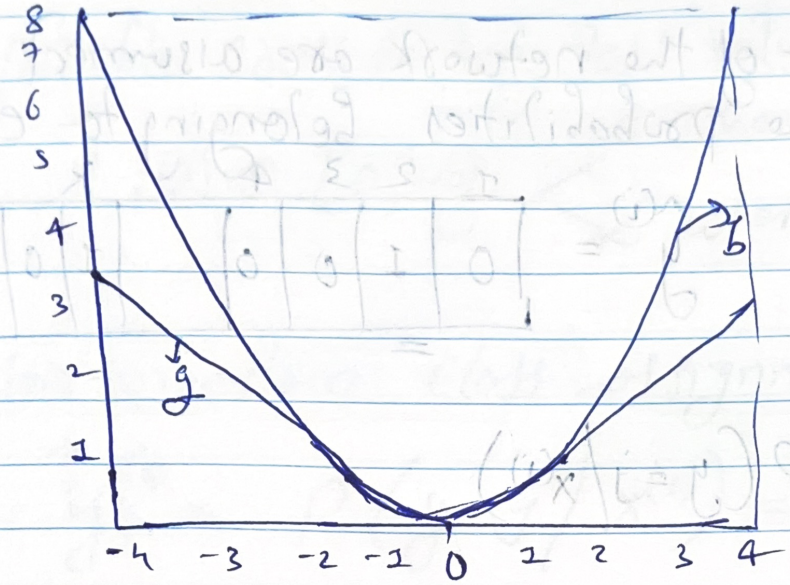
$$\boxed{|d| = \sigma} \quad \text{(equal values)}$$

Also, variable 'd' often refers as residuals, that is to the difference b/w observed & predicted values.

$$\boxed{d = y - f(x)}$$

$\therefore$ we can also express $\rightarrow$ $L_\sigma(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 \\ \sigma\left(|y - f(x)| - \frac{1}{2}\sigma\right) \end{cases}$

Graphical representation Huber loss.



Huber loss (g, $\sigma = 1$) & squared error loss (b) as a function of $(y - f(x))$

Cross entropy loss

It is a metric used to measure how well a classification model in machine learning / deep learning performs.

The loss (or error) is measured as a number between 0 & 1, with 0 being a perfect model.

⇒ In classification problems assume that the true class label $y^{(i)}$ is one-hot encoded.

[From Class Notes]

$x^{(i)} \in C_k \Rightarrow y^{(i)} =$

1	2	3	--	k	--
0	0	0		1	

Solution (1) (d)

Regularization refers to techniques that are used to calibrate machine learning / deep learning models in order to minimize the adjusted loss function & prevent overfitting or underfitting

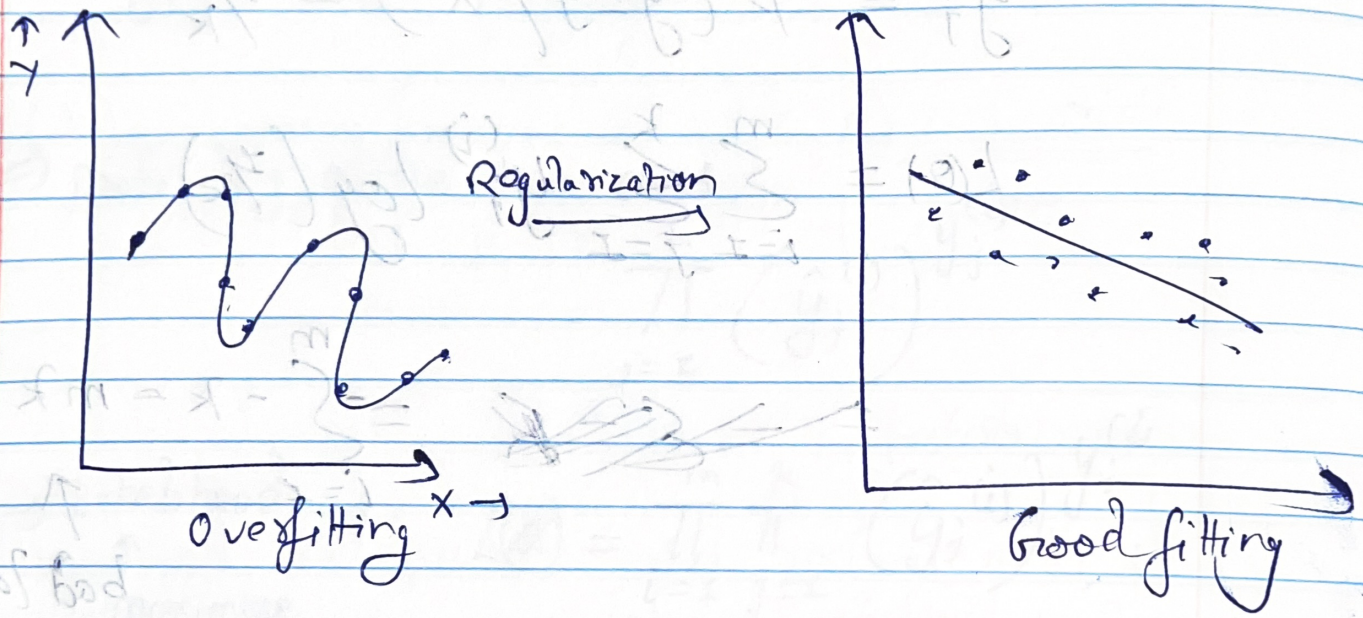
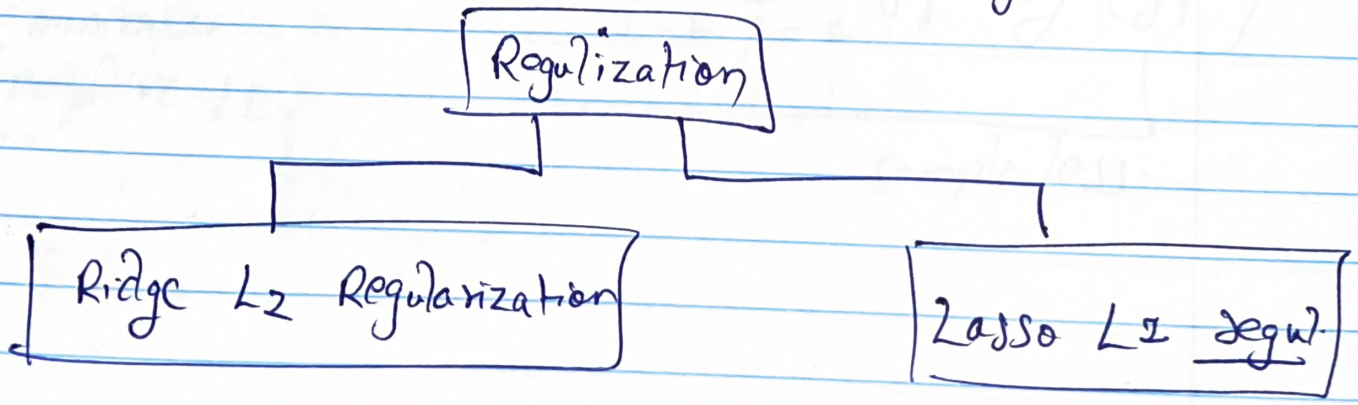


fig. Regularization on an overfitted model

These are 2 main types of regularization techniques:
1) Ridge Regularization (2) Lasso Regularization.



The L2 regularization is the most common type of regularization & also known as weight decay or ridge regularization.

During L2 regularization, the loss function of the neural network is extended by so-called regularization term, which is called (Ω)

$$\Omega(w) = \|w\|_2^2 = \sum_i \sum_j w_{ij}^2$$

(Regularization term)

Ω is defined as Euclidean Norm of the weight matrices, which is sum over all squared weight values of a weight matrix.

The regularization term is weighted by the scalar alpha divided by two & added to the regular loss function that is chosen for the current task.

New expression for loss function:

$$\hat{L}(w) = \alpha/2 \|w\|_2^2 + L(w)$$

(Regularization Loss during L2 regularization)

$$\hat{L}w = \alpha/2 \sum_i \sum_j w_{ij}^2 + L(w)$$

where, α = regularization rate

L1 regularization (Lasso): In this case we simply, we Ω (regularization term) as the sum of absolute values of the weight parameters in a weight matrix:

$$\Omega(w) = \|w\|_1 = \sum_i \sum_j |w_{ij}|$$

Like in L2 we can multiply regularization term by alpha & add the entire thing to the loss function.

$$\hat{L}(w) = \alpha \|w\|_1 + L(w)$$

↑↑ Loss function during L1 regularization

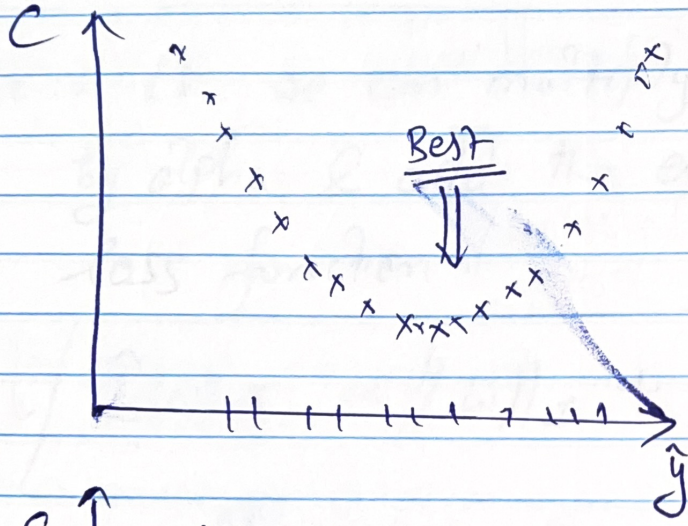
Solution (1) (e) Gradient descent is an iterative optimization algorithm for finding the minimum of function. To find a local minimum of a function using gradient

descent, we take steps proportional to the negative of the gradient of the function at the current point.

In other words, we move in the direction opposite to our gradients.

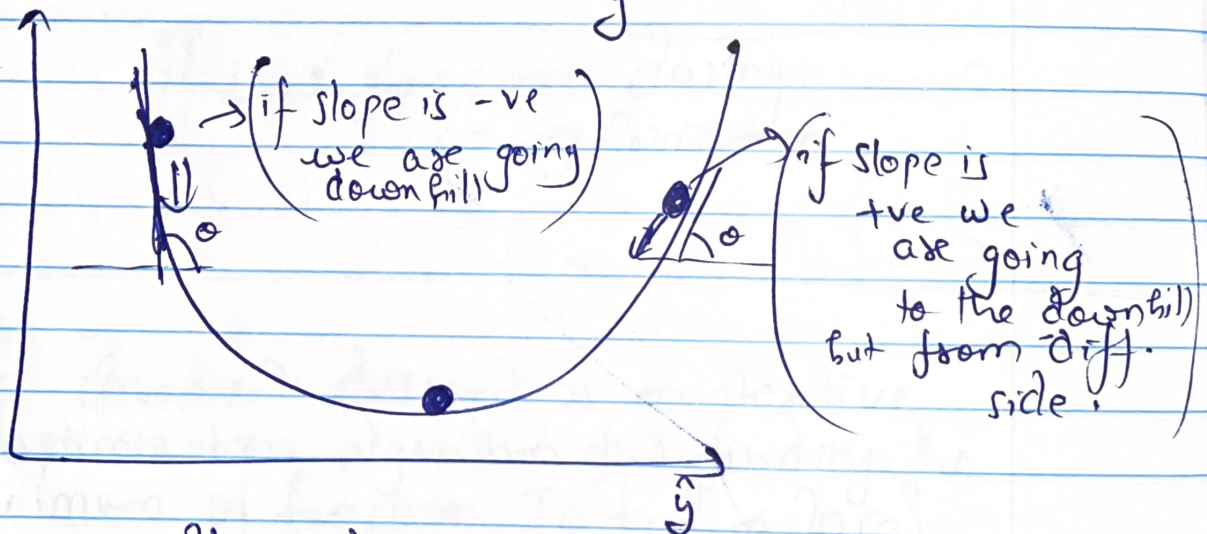
Solution (i) (f)

Gradient descent is basically the optimization done on the neural networks for minimizing the cost function.
So, basically if let say there are thousands of weights that need to be adjusted & out of them which to choose so this 'Gradient Descent' approach helps to do optimization.



$$C = \frac{1}{2}(\hat{y} - y)^2$$

↓
Cost function



Now, we will see the slope of our cost function, which will help to do optimization.

(Gradient Descent method)

So, using slope method we can easily find the best optimization.

Limitations of Gradient Descent

It is good for problems having convex shape function but when the function is not convex rather hybrid it will do optimization in local minima.

Optimization in local Minima

Stochastic Gradient Descent (SGD)

It is defined as a modified gradient descent technique for doing optimization globally.

⇒ In and all brief summary of difference b/w gradient descent & stochastic gradient descent as per class notes:

Gradient Descent

→ Iterative optimization algorithm.

→ start with guess θ_0

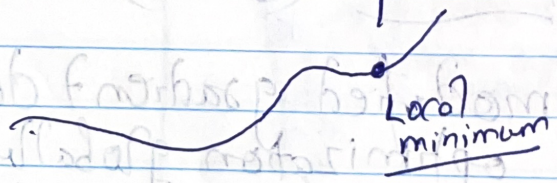
→ Repeat

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \nabla L(\theta^{(i)})$$

(η: learning rate)

→ Stop condition

$$|L\theta^{(i+1)} - L\theta^{(i)}| < \tau$$



SGD

→ Randomly order examples.

→ For $j = 1 \dots m$ (loss of j^{th} example)

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \nabla L_j(\theta^{(i)})$$

→ More frequent but less accurate updates.

SGD
update after every example

update using a subset of examples.

GD
update based on all examples.

Solution (I) (g)

Learning rate (δ or η) is one such hyper-parameter that defines the adjustment in the weights of our network w.r.t the loss gradient descent. It determines how fast or slow we will move towards the optimal weight.

The Gradient Descent algo. estimates the weights of the model in many iterations by minimizing a cost function at every step.

As per algorithm:

Repeat until convergence {

$$W_j = W_j - \delta \, \partial F(W_j) / \partial W_j$$

}

where; → W_j is the weight
→ θ is the theta
→ $F(W_j)$ is the cost function.

2 Cases

1) if δ is too small we will need too many iterations to converge to best values.

2) if δ is (very large) we will skip the optimal solution

Hence, using a good δ is crucial.

Following approaches can be used for finding good δ.

1) Choose a fixed learning rate.

The standard gradient descent procedure uses a fixed learning rate (eg: 0.01) that is determined by trial & error.

2) Use learning rate annealing

Learning rate annealing entails starting with a high learning rate & then gradually reducing the learning rate linearly during training. The learning rate can decrease to a value close to 0.

3) Use Cyclical learning rate.

Proposed by Leslie (2017)
Instead of monotonically decreasing the learning rate, this method lets the learning rate cyclically vary b/w reasonable boundaries.

4) Use an Adoptive learning rate.

Another option is to use δ that adopts based on the error output of the model.

Solution (1) (h) ⇒ Momentum is an extension to the gradient descent optimization algorithm, often referred to as Gradient Descent with Momentum.

⇒ It is designed to accelerate the optimization process, e.g. decrease the number of function evaluations required to reach the optima, or to improve the capability of the optimization algorithm e.g. result in a better final result.

⇒ A problem with GD algorithm is that the progression of the search can bounce around the search space based on the gradient. For example, the search may progress downhill towards the minima, but during this progression, it may move in another direction, even uphill, depending on the gradients of specific points (sets of parameter) encountered during the search.

∴ This can slow down the progress of the search.

Momentum involves adding an additional hyperparameter that controls the amount of history to include in the updation equation. i.e. the step to a new point in the search space.

Value of hyperparameter is in the range 0.0 to 1.0 & often close to 1.0. such as 0.8, 0.9 or 0.99.

We can break gradient descent update equation into 2 parts:

1) the calculation of the change to the position.
2) the update of the old position to the new position.

→ The change in the parameters is calculated as the gradient for the point scaled by the step size.

- $\text{change\_x} = \text{step\_size} * f'(x)$

New position is calculated by simply subtracting:

- $x = x - \text{change\_x}$

If we think of updates over time, then the update at the current iteration or time(t) will add the change used at the previous time (t-1) weighted by the momentum hyperparameter as follows:

- $\text{change\_x}(t) = \text{step\_size} * f'(x(t-1)) + \text{momentum} * \text{change\_x}(t-1)$

The update to the position:

$$x(t) = x(t-1) - \text{change\_x}(t)$$

Note: The change in the position accumulates magnitude & dir^n of changes over the iterations of the search, proportional to the size of momentum hyperparameters.

Neural Network

Solution 1) e) i)

Forword Propagation: As the name suggests, the input data is fed in the forword direction through the network. Each hidden layer accepts the input data, processes it as per the activation function & passes to the successive layer.

In order to generate some output, the input data should be fed in forword dir^n only.

~~They~~

At each neuron in hidden or output layer, the processing happens in 2 steps:

1) Pre-activation 2) Activation.

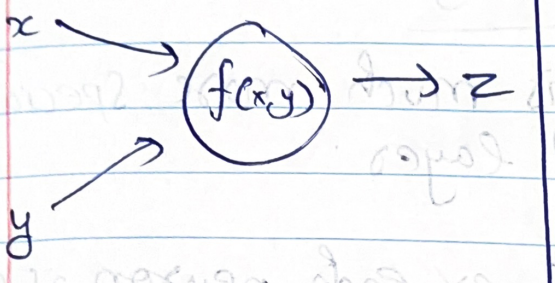
Backword Propagation:

It is the practice of fine ~~training~~ tuning the weights of a neural net based on the error rate (i.e.) loss) obtained in the previous epoch (i.e. iteration).

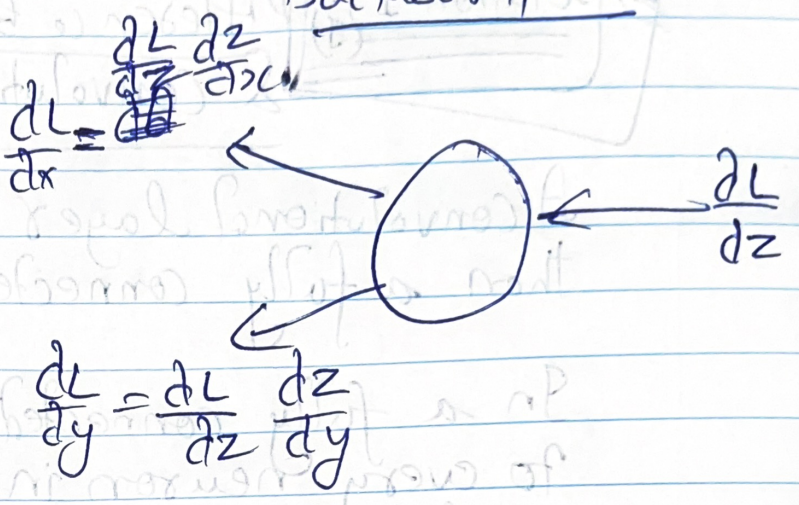
Proper training of weights ensures lower error rates, making the model reliable.

Solution 1)

Forward pass



Backward pass



Overall steps: (of propagation)

→ In forward propagate step, the data flows through the network. To get the outputs.

→ the loss function is used to calculate total error.

*→ then we use backward propagation algo. to calculate the gradient of the loss function with respect to each weight & bias.

*→ Finally, we use gradient descent to update weights & biases at each layer.

→ we repeat each step to minimize total error of the neural network.

Review Question
1. Neural Network.

Solution 1) (j) Difference between fully connected & convolution layer:

A convolutional layer is much more specialized than a fully connected layer.

In a fully connected layer each neuron is connected to every neuron in the previous layer & each connection has it's own weight.
It is totally general purpose connection pattern & makes no assumptions about the features in the data.
Also, very expensive in memory & computation.

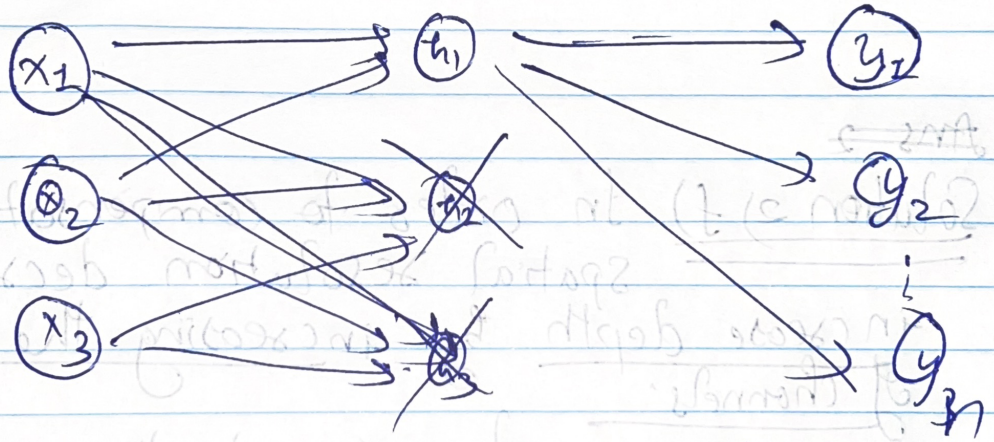
In contrast, in convolutional layer each neuron is only connected to a few nearby neurons in the previous layer, & the same set of weights is used for every neuron.
The fewer number of connections & weights make convolution layers relatively cheap in terms of memory & compute power needed.

Solution 1) (K) Dropout layer is used to regularize deep neural networks.

→ It is a technique which is used to prevent overfitting.

→ Dropout works by randomly setting the outgoing edges of hidden unit to '0' at each update of the training phase.

fig.



Dropout (h_2 & h_3)

In above fig. we have dropout h_2 & h_3 hence the hidden layer will only consist of h_1. for evaluation of output (y_i).

Review Question 2
Convolution Neural Network.

Solution 2) a)

RGB Image 4x4

Channel R

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

3x3 filter

1	1	1
1	1	1
1	1	1

$*$

⇒ first element of 4x4 matrix on convolution

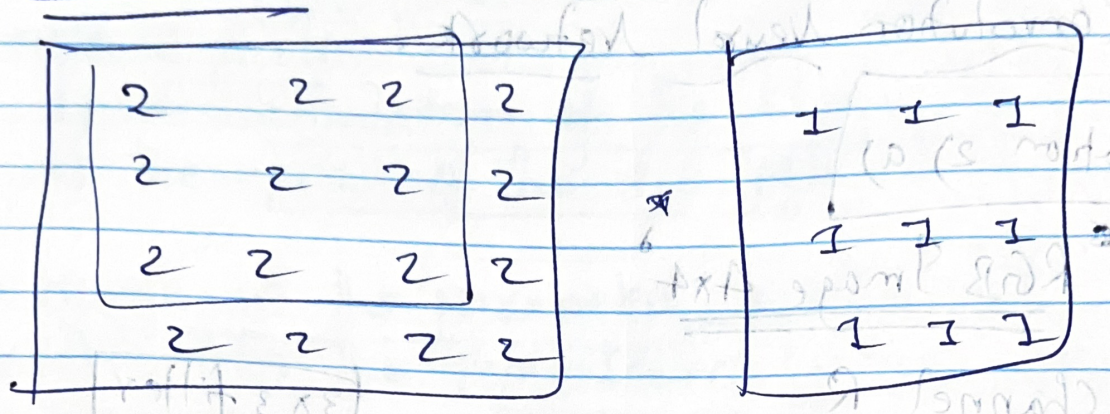
$= (1\times1 + 1\times1 + 1\times1) \times 3$

$= \boxed{9}$

⇒ Similarly, for all the elements as all the elements are same so output will also remain same $= \boxed{9}$

9	9
9	9

Channel G



$= (2+2+2) \cdot 3$

$= 18$

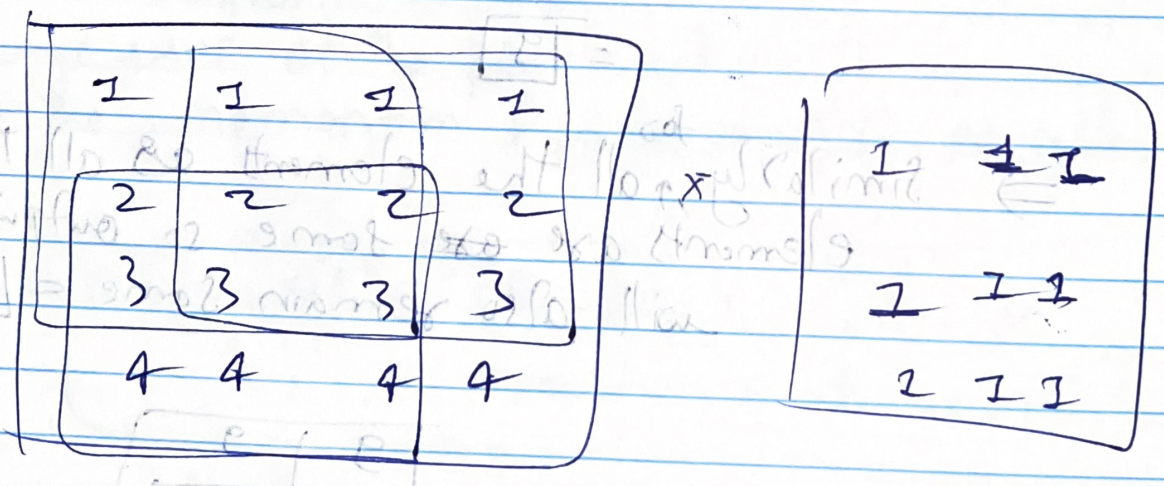
Similarly, other elements on convolution will be $= 18$

$\therefore$ Matrix on convolution $\Rightarrow$

18	18
18	18

2x2

Channel B



First element $= \begin{pmatrix} 1+1+1+ \\ 2+2+2+ \\ 3+3+3 \end{pmatrix} = 3+6+9$

$= 18$

Second element $= 18$

Third element $= \begin{pmatrix} 2+2+2 \\ 3+3+3 \\ 4+4+4 \end{pmatrix} = (6+9+12)$

$= 27$

Forth element = 27

So, matrix will be :

18	18
27	27

2x2

So, we have

9 9	18 18	18 18
9 9	18 18	27 27

RGB channel convolution matrix

⇒ Element wise addition of all 3 channel :

$$\begin{bmatrix} 9+18+18 & 9+18+18 \\ 9+18+27 & 9+18+27 \end{bmatrix} = \begin{pmatrix} 45 & 45 \\ 54 & 54 \end{pmatrix}$$

⇒ On Averaging each element with 1/9 while convolution elements will be :

$$\begin{bmatrix} 1+2+2 & 1+2+2 \\ 1+2+3 & 1+2+3 \end{bmatrix} = \begin{bmatrix} 5 & 5 \\ 6 & 6 \end{bmatrix}$$

Solution 2) b) with Zero padding

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

6×6

$*$

1	1	1
1	1	1
1	1	1

3×3

$$= \begin{bmatrix} 4 & 0+6 & 0+6 & 4 \\ 6 & 9 & 9 & 6 \\ 6 & 9 & 9 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix}$$

2nd element $= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

$= \boxed{6}$

4th row or (13th element)

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \boxed{4}$$

14th element

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} = 6$$

4th element

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = 4$$

5th element

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} = 6$$

∴ final convolved matrix with zero padding =

$$\begin{bmatrix} 4 & 6 & 6 & 4 \\ 6 & 9 & 9 & 6 \\ 6 & 9 & 9 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix}$$

Channel G

0	0	0	0	0	0
0	2	2	2	2	0
0	2	2	2	2	0
0	2	2	2	2	0
0	2	2	2	2	0
0	0	0	0	0	0

*

1	1	1
1	1	1
1	1	1

First element = $2 \times 4 = \boxed{8}$

2^{nd} & 3^{rd} $= 2 \times 6 = 12$

4^{th} element $= \boxed{8}$

$5^{th} = 2 \times 6$

$6^{th} = 2 \times 9 = 18$

Convolved G matrix $=$

$$\begin{bmatrix} 8 & 12 & 12 & 8 \\ 12 & 18 & 18 & 12 \\ 12 & 18 & 18 & 12 \\ 8 & 12 & 12 & 8 \end{bmatrix}$$

Channel B

0	0	0	0	0	0
0	1	1	1	1	0
0	2	2	2	2	0
0	3	3	3	3	0
0	4	4	4	4	0
0	0	0	0	0	0

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

1st element = 1×2 + 2×2 = 6

2nd / 3rd = 1×3 + 2×3 = 9

4th = 1×2 + 2×2 + 3×2 = 2 + 4 + 6 = 12

5th = 1×3 + 2×3 + 3×3 = 3 + 6 + 9 = 18

6th = 1×2 + 2×2 + 3×2

7th = 2×2 + 3×2 + 4×2 = 4 + 6 + 8 = 18

8th = 2×3 + 3×3 + 4×3 = 27

13th = 6 + 8 = 14

14th = 9 + 12 = 21

$$= \begin{bmatrix} 6 & 9 & 9 & 6 \\ 12 & 18 & 18 & 12 \\ 18 & 27 & 27 & 18 \\ 14 & 21 & 21 & 14 \end{bmatrix}$$

Element wise add^ of all channel

$$= \begin{bmatrix} 4 & 6 & 6 & 4 \\ 6 & 9 & 9 & 6 \\ 6 & 9 & 9 & 6 \\ 4 & 6 & 6 & 4 \end{bmatrix} + \begin{bmatrix} 8 & 12 & 12 & 8 \\ 12 & 18 & 18 & 12 \\ 12 & 18 & 18 & 12 \\ 8 & 12 & 12 & 8 \end{bmatrix} +$$

$$+ \begin{bmatrix} 6 & 9 & 9 & 6 \\ 12 & 18 & 18 & 12 \\ 18 & 27 & 27 & 18 \\ 14 & 21 & 21 & 14 \end{bmatrix} = \begin{bmatrix} 18 & 27 & 27 & 18 \\ 30 & 45 & 45 & 30 \\ 30 & 54 & 54 & 30 \\ 26 & 39 & 39 & 26 \end{bmatrix}$$

Matrix on Convolution (with zero padding) =

$$\begin{bmatrix} 18 & 27 & 27 & 18 \\ 30 & 45 & 45 & 30 \\ 30 & 54 & 54 & 30 \\ 26 & 39 & 39 & 26 \end{bmatrix}$$

Solution 2) c) Dilated kernel (with rate = 2)

$$K = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}_{5\times5}$$

R Channel (Convolution)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{6\times6} * \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}_{5\times5}$$

1^{st} o/p $= 0+0+(1+1)+0+(1+1)$

$= 2+2 = 4$

Similarly, 2^{nd} / 3^{rd} / 4^{th} elements.

~~Solution 2) 2) (i)~~
~~2) (i)~~

$$= \begin{array}{|c|c|}\hline 4 & 4 \\ \hline 4 & 4 \\ \hline \end{array}_{2\times 2}$$ (feature map from R channel)

G channel

$$\begin{array}{|cccccc|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} * K =$$

1^{st} el. $= (2+2) + (2+2) = 8$

Similarly other elements.

So, convolved matrix (feature map) for G.

$$= \begin{array}{|c|c|}\hline 8 & 8 \\ \hline 8 & 8 \\ \hline \end{array}_{2\times 2}$$

B channel

$$\begin{array}{|cccccc|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 \\ 0 & 3 & 3 & 3 & 3 & 0 \\ 0 & 4 & 4 & 4 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} * K = \begin{array}{|c|c|}\hline 12 & 12 \\ \hline 8 & 8 \\ \hline \end{array}_{2\times 2}$$

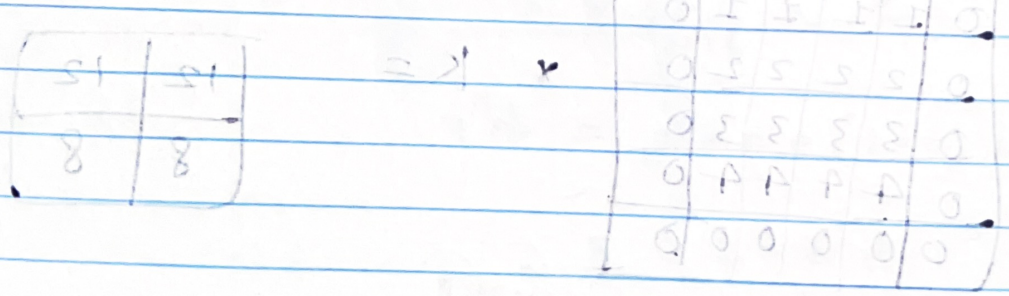
Element wise add^n of all channel

$$= \begin{array}{|c|c|} \hline 4+8+12 & 4+8+12 \\ \hline 4+8+8 & 4+8+8 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|} \hline 24 & 24 \\ \hline 20 & 20 \\ \hline \end{array}$$

Solution 2) d) Template matching is the process of moving the template over the entire image & calculating the similarity between the template & the covered window on the image.

It is implemented through 2D convolution.
In convolution, the value of output pixel is computed by multiplying elements of two matrices & summing the results.
One of these matrices represents the image itself, while the other matrix is the template which is known as convolution kernel.



Solution 2) e) Multi Scale Analysis with a fixed window size

We can achieve mutiple scale analysis by increasing the depth of the image. So, by this way same region in an image with a fixed window size can store more information at different depths.
Hence, multiscale achieved.

~~Ans 2~~

Solution 2) f) In order to compensate spatial resolution decrease we increase depth by increasing the number of channels.
So, as we go deeper into the network, we have more & more units & as we know deeper layers have larger receptive fields & corresponds to more specific features. & hence we can detect complex spatial shapes. That is the major purpose for doing so.

~~Ans~~ Solution 2) g)

Formula to calculate feature map shape

$$fm = \left[\frac{i - K + 2p}{s}\right] + 1$$

(i → image, K → kernel, p → padding, s → stride)

$$fm = \left[\frac{128 - 3 + 0}{1}\right] + 1 = 126$$

So, the size of resulting tensor for 16 convolution filter of size 3x3x32 :

$$\Rightarrow \boxed{126 \times 126 \times 16}$$

Solution 2) h) (with stride = 2)

$$fm = \left[\frac{128 - 3}{2}\right] + 1$$

$$= (125/2) + 1 = 62 + 1$$

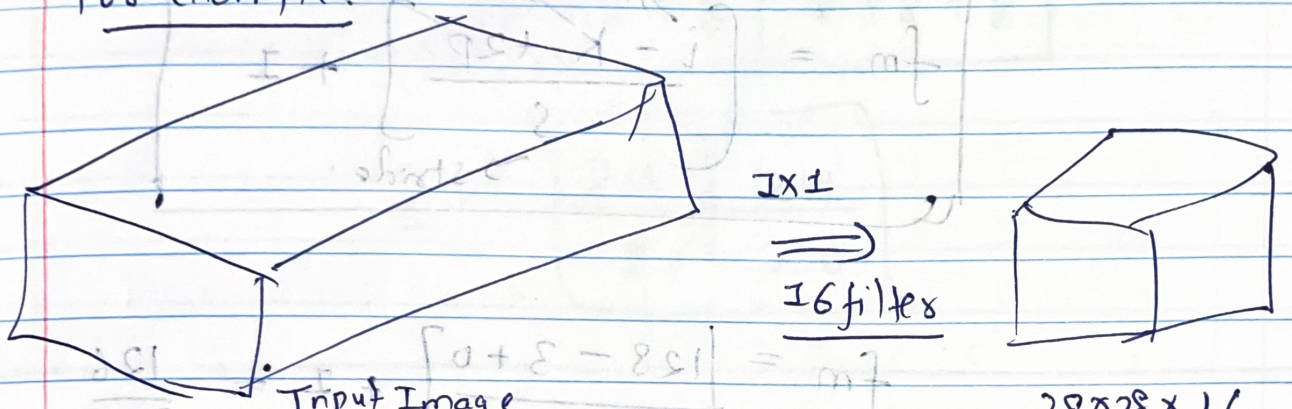
$$fm = \boxed{63}$$

So, the size of resulting tensor :

$$\Rightarrow \boxed{63 \times 63 \times 16}$$

Solution 2) i) 1×1 convolution operation involves convolving the input with the filter of size 1×1.

For example:



So, as we can see, in above operation using 1×1 filter (16) the depth of input image reduced/shrink from 192 to 16.

So, 1×1 is used for 'Dimension Reduction' & in making computation less expensive.

(e)(f)(j)

Solution 2) j) A convolutional layers generally used with ConvNet, is a class of neural networks that specializes in processing data that has a grid like topology, such as an image.

Convolution layers is the core building block of CNN. It performs a dot product b/w 2 matrices. where one matrix is the set of learnable parameters known as kernel & other portion is restricted portion of the receptive field.

Difference between early & deep layers:

Major difference is that early convolution layers can learn features like lines, edges etc. whereas;
Deep layers can do for more complex features like face detection, object recognition etc.

Solution 2) k) Purpose of pooling

Pooling = down sampling spatial dimensions (depth unchanged)

① So, the main purpose of is it helps in reducing the number of networ coefficients.

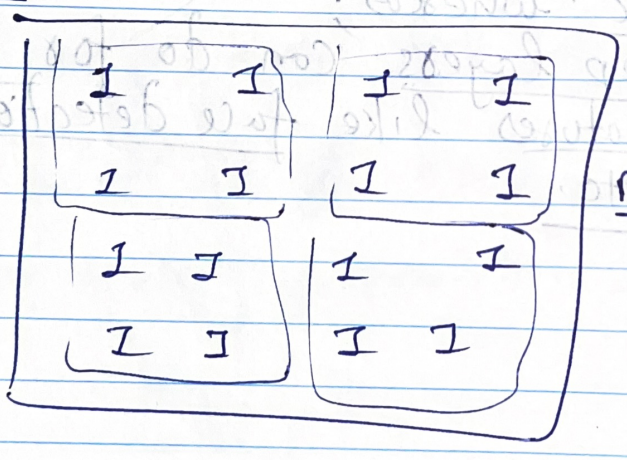
② Also, Pooling supports multiple scale analysis

③ It makes the operations / convolutions computationally less expensive.

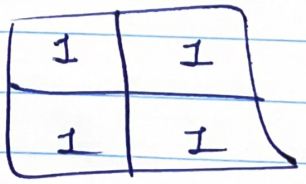
The most common approach used in pooling is max pooling.

Solution 2) l)

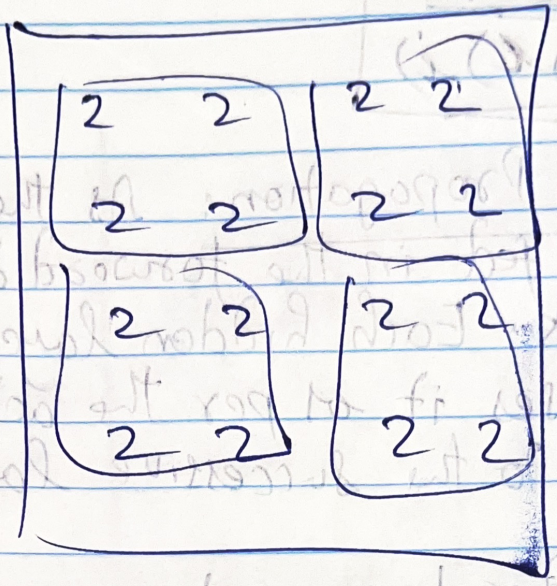
R Channel

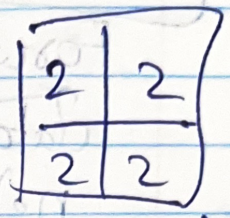


Max Pooling with stride = 2



6 channel





Max Pooling w.

8 channel