ಗ್ಲೋಬಲ್ ಅಕಾಡೆಮಿ ಆಫ್ ಟೆಕ್ನಾಲಜಿ, ಬೆಂಗಳೂರು

Global Academy of Technology, Bengaluru

Department of Artificial Intelligence & Machine Learning	Internal Test No: 2
Semester	5th

Subject Name	Deep Learning Principles & Practices	Subject Code	2	1	A	M	L	5	3
Time: 90 Mins.				Max. Marks: 40					

Q. No.	Questions	Marks
1	*Examine the implications of initializing weights with excessively large or small values in a deep neural network. Discuss the potential challenges associated with weight initialization and the strategies employed to address these challenges.*	**10**

⇒ Initializing weights in Neural Network

Let the initial weight value be "Zero".

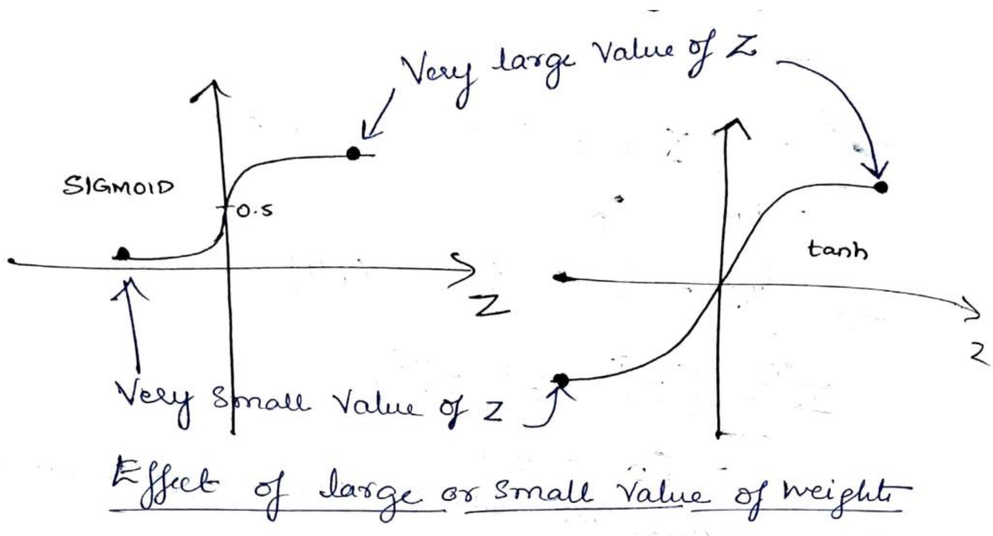
If the weights on the first layer is given as

$$W^{[1]} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \\ W_{41} & W_{42} & W_{43} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$ 2M

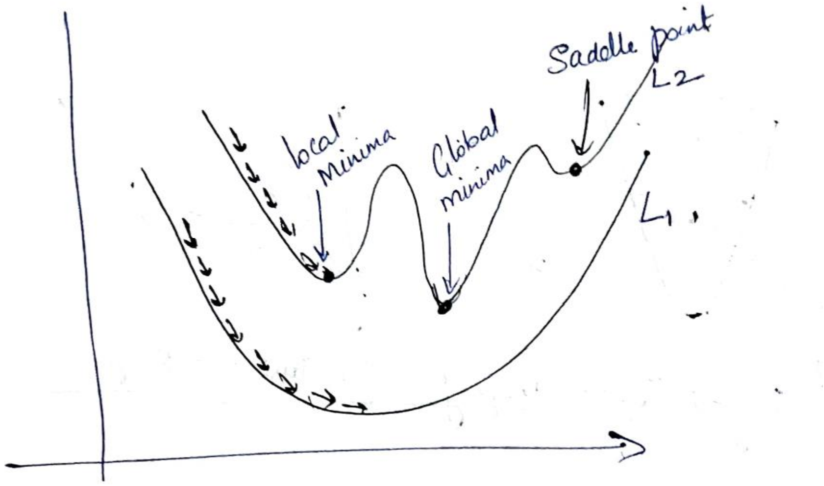
$$Z^{[1]} = [x_1 \; x_2 \; x_3 \; x_4] * \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \\ W_{41} & W_{42} & W_{43} \end{bmatrix}$$

$$\therefore Z^{[1]} = [x_1 \; x_2 \; x_3 \; x_4] * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$ 2M

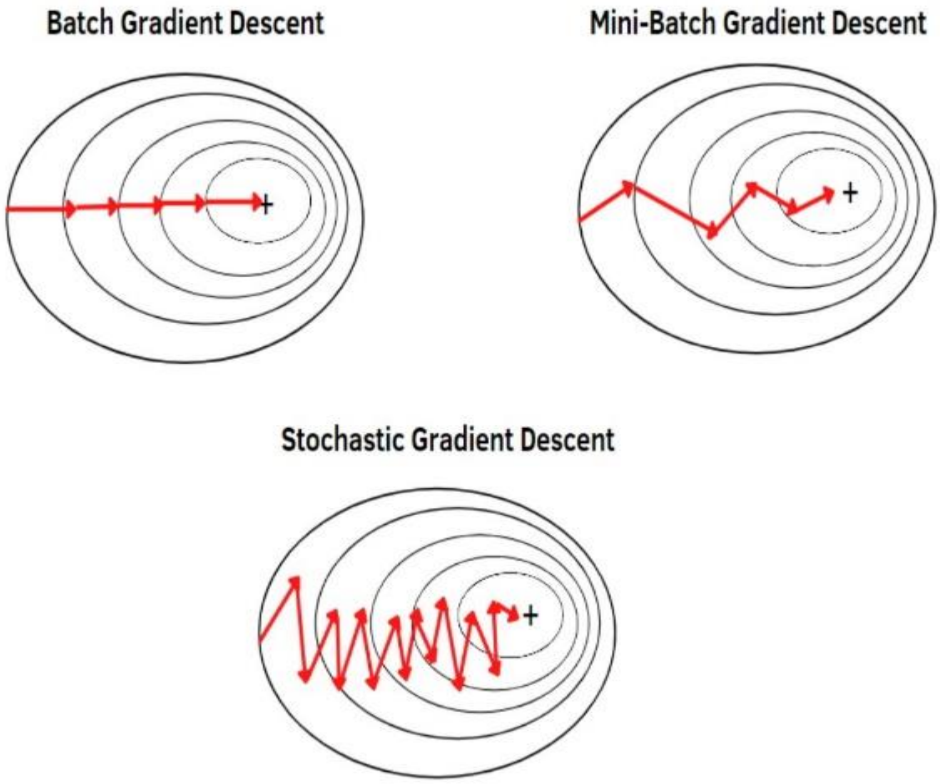
Individually, for each neuron in the hidden layer 1, we have

	$Z_1^{[1]} = [X_1.0 + X_2.0 + X_3.0 + X_0.0]$ $= 0$ $Z_2^{[1]} = [X_1.0 + X_2.0 + X_3.0 + X_0.0]$ $= 0$ $Z_3^{[1]} = [X_1.0 + X_2.0 + X_3.0 + X_0.0]$ $Z^{[1]} = [Z_1^{[1]} \; Z_2^{[1]} \; Z_3^{[1]}]$ $= [0 \; 0 \; 0]$ Assume Same activation function is applied to all neuron in a hidden layer. $A_1^{[1]} = A_2^{[1]} = A_3^{[1]}$ ⇒ Neuron are said to be 'symmetric' ⇒ using "zero" as the value to initiate the weights is practically going to make the use of multiple neurons in hidden layers redundant. That is why the weights should not be initialized as 0. However, the bias can be initialized as 0. If the option of initializing is with random values. 2 ways by using normal distribution — by using a uniform distribution. → If the weights are initialized with large values, the value of Z will be relatively large. If observed in Sigmoid and tanh function, the slope  Effect of large or small value of weights	2M 2M

<table>
<tr>
<td></td>
<td>Leads to Vanishing Gradient and Exploding gradients. These challenges are overcome by using some optimum solutions. To strike the balance between these extremes, 2 types of weight initializations are used.
1) Kaiming He initialization
2) Xavier initialization</td>
<td>2M</td>
</tr>
<tr>
<td>2</td>
<td>Explore the challenges posed by local minima and saddle points in the context of batch gradient descent. Elaborate on the difficulties these points present and discuss how stochastic gradient descent (SGD) offers solutions to mitigate these challenges.



Fig(c) Batch Gradient Descent for Non-Convex Loss Curve


</td>
<td>10

2M

3M</td>
</tr>
</table>

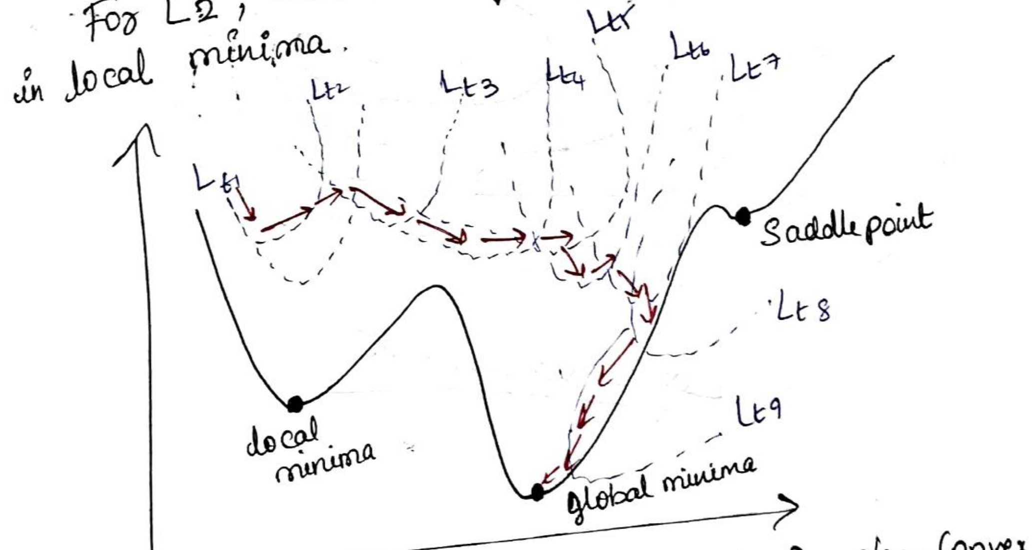
→ A mini-batch gradient descent follow a middle path. It is not as smooth as Batch gradient descent and as oscillating as Stochastic gradient descent.

→ Although, Batch gradient descent seems to give the smoothest path to reach the point of lowest loss, there are 2 main Challenges.

① loading the entire training data to memory Can be a big challenge.

② They may stuck to a local minima.

∴ In all practical Situations, the loss curve is not a Simple Convex Curve (L_1) as Show in Fig(c) but a more Complex, non-Convex Curve (L_2). For L_2, there is a possibility to get trapped in local minima.



Fig(d) Stochastic Gradient Descent for Non-Convex loss curve

→ In Stochastic gradient descent, the possibility to get trapped in local minima is much less.
In Fig(d), each training record t_1, t_2, t_3 etc., there is a Separate loss Curve $L_{t_1}, L_{t_2}, L_{t_3}$ etc., respectively. 2M

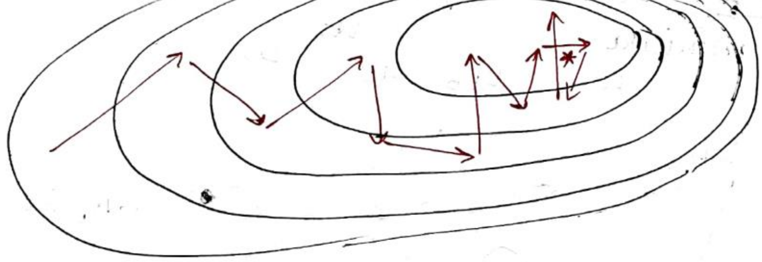
→ This leads to a higher possibility to escape local minima and reach the global minima for the curve.

→ Therefore, the possibility of missing the global optimal point is much less in case of Stochastic gradient descent.

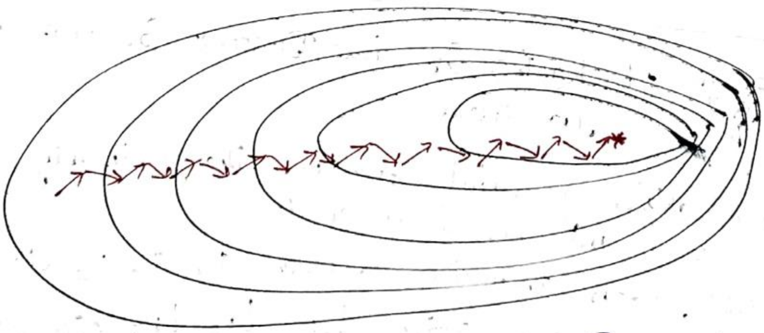
In certain points of loss curve, the gradient may also be 0. Such points known as Saddle points.

→ An SGD can overcome Saddle points too, whereas the batch gradient descent faces a serious challenge.

↳ disadvantage of SGD is that the model parameter update is so frequent that it is computationally expensive & also takes a significantly high training time for large training datasets



Large learning rate



fig(e)

Effect of Learning rate on Gradient Descent — Small learning rate

3M

3	*a) Interpret the challenges faced by traditional ANN to deal with image and what are the building blocks of CNN.*	**10**

⇒ Challenges Faced by Traditional ANN to work with image data

In case of MNIST dataset, it is one of the most popular image datasets being adopted in deep learning literature, traditional ANN yields pretty good results.

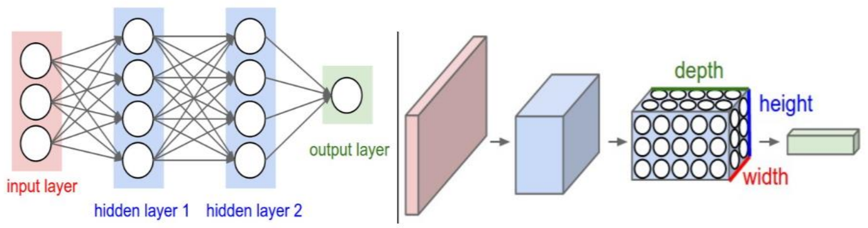
However, MNIST is a very specific dataset in which all images are extremely uniform in shape and the images are all centered.

→ In real-world image-related problems, we hardly come across such uniform and standardized images.

→ Another problem for any image dataset is the number of trainable parameters

3M

Traditional ANN will take a long training time as well as memory
CNN has certain components as a part of the architecture which optimizes the number of trainable parameters
ANN does not capture the spatial features (i.e.,features which exploit location information) of an image well.
Therefore, CNN perform non-linear transformation in different locations of the image in the form of convolutions, so the feature can be considered spatial.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

3M

b) Calculate the padding size for an image of size 50 x 50 and a 3x3 filter used for convolution, assuming 'same' padding?

For same padding

Padding size, $P = \frac{f-1}{2}$ where f denotes the dimension of the filter 2M

$$P = \frac{3-1}{2} = 1$$

Padding size required is 1 2M
for same padding the padding exists.

4

a) What will be the dimensions of the output feature map for an input feature map of size 380x270x64 which passes through a pooling layer having filter size 2x2 and a stride size of 2? **10**

3M

$$\left(\frac{n_h - f}{s} + 1\right) \times \left(\frac{n_w - f}{s} + 1\right) \times n_c$$

n_h=380, n_w=270, n_c =64, f=2 and s=2 2M

therefore , the dimension of the output feature map will be 190×135×64

b) Enumerate the various types of pooling employed in CNN architectures and delve into the details of one specific pooling type, supported by a practical example by highlighting the significance of the pooling layer in Convolutional Neural Networks (CNNs) and its diverse applications.

Pooling

→ An effective technique to reduce the number of trainable parameter is by using pooling layers.

→ pooling layers help to succesfully downsample an image, thus reducing its dimension and reducing the number of trainable parameter

→ pooling layer helps to progressively reduce the spatial size of the representation

→ A pooling window is represented by its maximum or average or sum value; So it is independent of the spatial coordinates of the values within a specific window.

→ For an input feature map having dimensions $n_h \times n_w \times n_c$, if a $f \times f$ pooling filter is applied with a stride size 's', the dimension of the o/p map will be

$$\left(\frac{n_h - f}{s} + 1\right) \times \left(\frac{n_w - f}{s} + 1\right) \times n_c$$

3-types of pooling 3M

1. Max pooling
2. Average pooling
3. Sum pooling

original image

1	5	2	4
5	-7	7	-1
13	-3	4	7
5	9	24	9

⇒

5	7
13	24

Max pooling

1	3
6	11

Avg pooling

4	12
24	44

Sum pooling

for 2x2 pooling filter and a stride size of 2

For max pooling → maximum value of the subset is considered.

for average pooling → average of the pixel values of the different subsets are taken.

i.e., $\frac{1+5+5+(-7)}{4} = 1$

|||ly 3, 6 & 11 respectively.

For sum pooling → Sum of the values in the subset are done.

2M

5 *What are the trades-offs between batch, minibatch, and stochastic gradient descent in terms of convergence speed, computational efficiency, and generalization? Explain with neat diagram how gradient descent differs from other gradient descent variants*. **10**

1. Batch Gradient Descent:

- **Convergence Speed:** Slower compared to other variants, especially on large datasets, as it processes the entire dataset before updating weights.
- **Computational Efficiency:** Computationally expensive, as it requires storing and processing the entire dataset in each iteration.
- **Generalization:** May converge to a more accurate minimum due to the use of the entire dataset, but might be prone to getting stuck in local minima.

2. Mini-Batch Gradient Descent:

- **Convergence Speed:** Faster than batch gradient descent due to processing smaller subsets (batches) of the dataset in each iteration.
- **Computational Efficiency:** More computationally efficient than batch gradient descent, especially for large datasets. Utilizes parallelism better.
- **Generalization:** Balances between batch and stochastic; tends to generalize better than batch but may still find a good minimum.

3M

3. Stochastic Gradient Descent (SGD):

- **Convergence Speed:** Fastest convergence, as it updates weights based on individual data points.
- **Computational Efficiency:** Highly efficient due to processing one data point at a time. Suitable for online learning scenarios.
- **Generalization:** May oscillate around the minimum, but the noisy updates can help escape local minima, leading to better generalization.

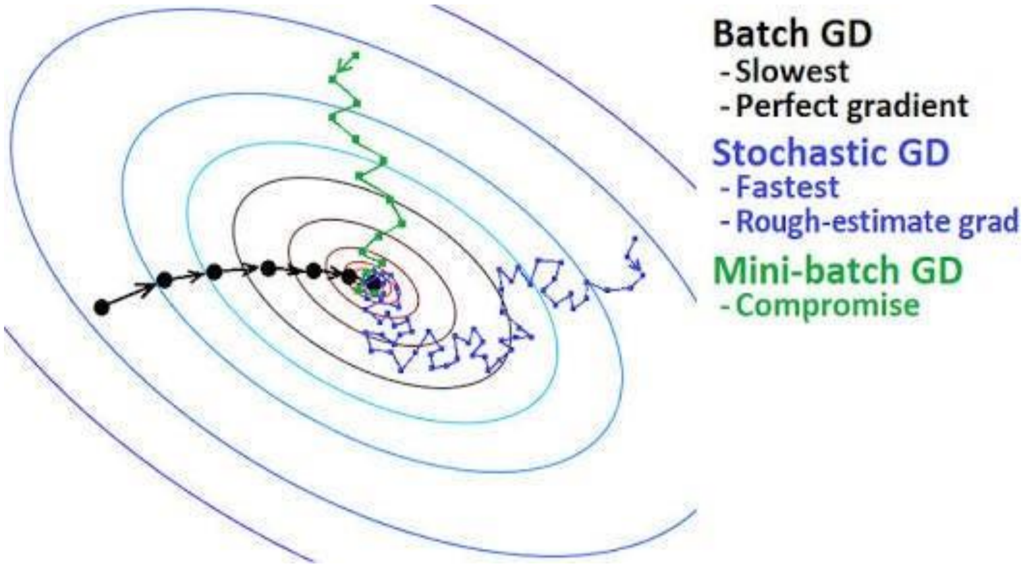
Trade-Offs:

- **Convergence Speed:**
  - Batch: Slow
  - Mini-Batch: Moderate
  - Stochastic: Fast
- **Computational Efficiency:**
  - Batch: Low
  - Mini-Batch: Moderate to High
  - Stochastic: High

3M

- **Generalization:**
    - Batch: May converge to a more accurate minimum but may overfit.
    - Mini-Batch: Balances between batch and stochastic; good generalization.
    - Stochastic: Better at escaping local minima, potentially better generalization.

The choice between these variants depends on the dataset size, computational resources, and the trade-off between convergence speed and generalization. Mini-batch gradient descent is often a practical compromise in many scenarios.



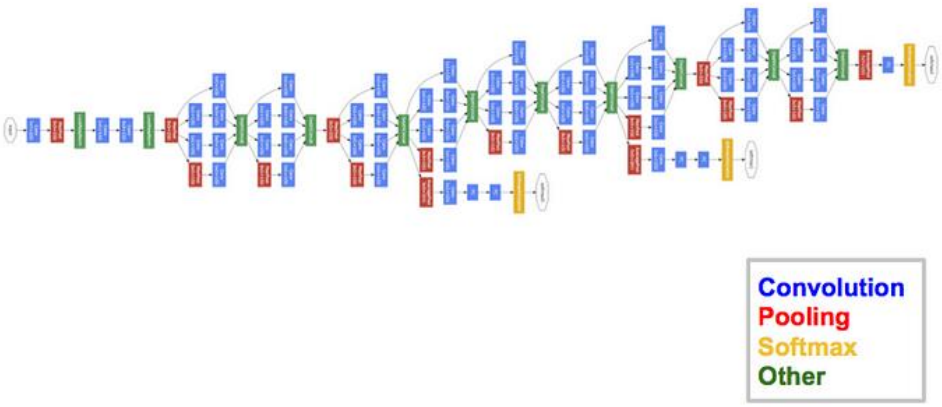
4M

6

Write short notes on:
(a) VGG 16
(b) Google Net

10

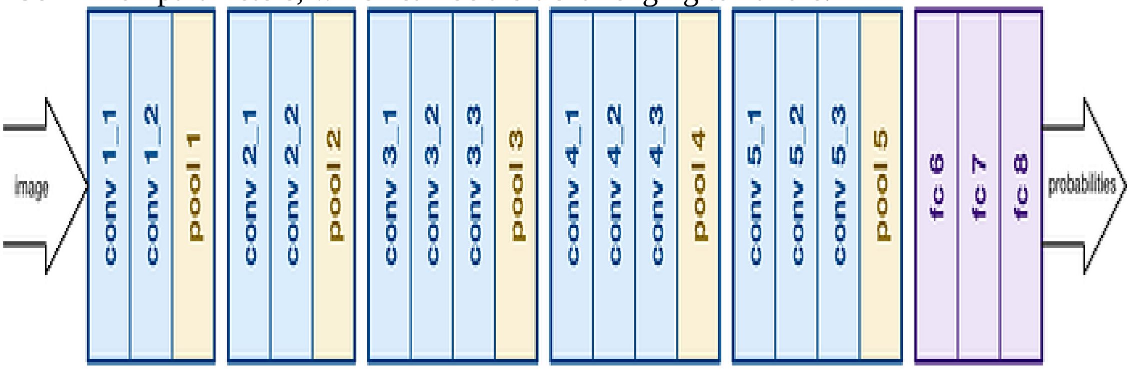
GoogLeNet/Inception(2014)
The winner of the ILSVRC 2014 competition was GoogLeNet(a.k.a. Inception V1) from Google. It achieved a top-5 error rate of 6.67%! This was very close to human level performance which the organisers of the challenge were now forced to evaluate. As it turns out, this was actually rather hard to do and required some human training in order to beat GoogLeNets accuracy. After a few days of training, the human expert (Andrej Karpathy) was able to achieve a top-5 error rate of 5.1%(single model) and 3.6%(ensemble). The network used a CNN inspired by LeNet but implemented a novel element which is dubbed an inception module. It used batch normalization, image distortions and RMSprop. This module is based on several very small convolutions in order to drastically reduce the number of parameters. Their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million.



5M

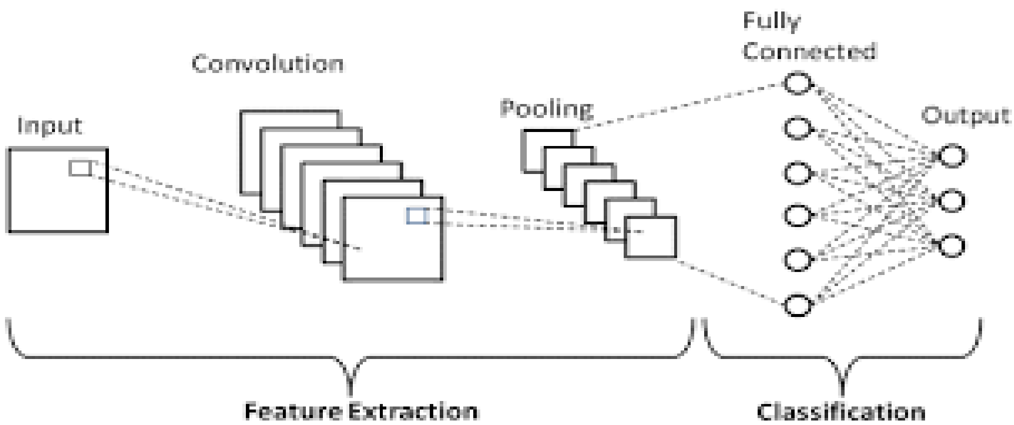
VGGNet (2014)

The runner-up at the ILSVRC 2014 competition is dubbed VGGNet by the community and was developed by Simonyan and Zisserman. VGGNet consists of 16 convolutional layers and is very appealing because of its very uniform architecture. Similar to AlexNet, only 3x3 convolutions, but lots of filters. Trained on 4 GPUs for 2–3 weeks. It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor. However, VGGNet consists of 138 million parameters, which can be a bit challenging to handle.



5M

7 *Develop a high-level overview of a custom CNN architecture tailored for a specific computer vision application, highlighting the key design choices and layers.* **10**



To design a Convolutional Neural Network (CNN) architecture for object recognition in the context of autonomous vehicles, ensuring real-time processing, high accuracy, and adaptability to varying environmental conditions. **3M**

Key Design Choices:

1. **Input Layer:**
    - RGB Image Input: Accepts input images in the standard RGB format (3 channels).
    - Image Preprocessing: Incorporates preprocessing layers to handle illumination changes, noise, and distortions.
2. **Convolutional Layers:**
    - **Convolutional Blocks:** Stacks multiple convolutional blocks with varying kernel sizes.
    - **Striding and Padding:** Utilizes striding and padding to control spatial dimensions and capture multi-scale features effectively.
    - **Activation Function:** Leaky ReLU for non-linearity to avoid vanishing gradients.
3. **Pooling Layers:**
    - **Max-Pooling:** Applied after convolutional blocks for downsampling and spatial hierarchy extraction.

	• **Global Average Pooling (GAP):** Utilized in later layers for spatial information reduction and feature summarization. 4. **Normalization and Regularization:** • **Batch Normalization:** Enhances convergence and stability during training. • **Dropout:** Regularizes the network by randomly dropping neurons to prevent overfitting. 5. **Skip Connections:** • **Residual Connections:** Employed to facilitate the flow of gradients and ease the training of deeper networks. • **Feature Concatenation:** Incorporates skip connections between different layers to enhance feature reuse. 6. **Fully Connected Layers:** • **Flatten Layer:** Precedes fully connected layers to transition from convolutional layers. • **Dense Layers:** Multiple dense layers with gradual reduction in neurons to extract high-level features. 7. **Output Layer:** • **Softmax Activation:** Applied to produce class probabilities for multi-class object recognition. • **Sigmoid Activation:** Used for binary classification tasks. **Additional Considerations:** • **Transfer Learning:** • Pre-training on large datasets like ImageNet and fine-tuning for the target task to leverage learned features. • Potential use of architecture variants inspired by successful models like Efficient Net or Mobile Net for efficiency. • **Data Augmentation:** • Implementation of data augmentation techniques (rotation, scaling, flipping) to increase model robustness and handle diverse scenarios. • **Optimization and Regularization:** • Adaptive learning rate strategies (e.g., Adam optimizer) for efficient convergence. • L2 regularization for weight decay. **Evaluation Metrics:** • **Performance Metrics:** Precision, recall, F1-score, and accuracy for comprehensive evaluation. • **Latency:** Assessment of inference time for real-time deployment on autonomous vehicles.	**2M** **3M** **2M**
8	*Apply the principles of transfer learning to explain how pre-trained CNN models, such as ResNet, can be fine-tuned for a new image classification task. Provide specific steps in the process.* **Steps for Fine-Tuning a Pre-trained ResNet Model:** 1. **Select a Pre-trained ResNet Model:** • Choose a ResNet model that was pre-trained on a large and diverse dataset, such as ImageNet. The pre-trained model will have learned generic features that can be beneficial for a variety of tasks. 2. **Remove the Fully Connected Layers:** • The last few layers of a pre-trained ResNet model typically include fully connected layers for the specific task it was trained on (e.g., ImageNet classification). Remove these layers to retain the convolutional base. 3. **Add New Fully Connected Layers:** • Append new fully connected layers at the end of the pre-trained ResNet model. These layers will be specific to the new image classification task. Adjust the number of neurons in the output layer based on the number of classes in the new task. 4. **Freeze Convolutional Layers:**	**10** **2M**

- Freeze the weights of the convolutional layers in the pre-trained ResNet model. This prevents these layers from being updated during the initial training on the new task.

2M

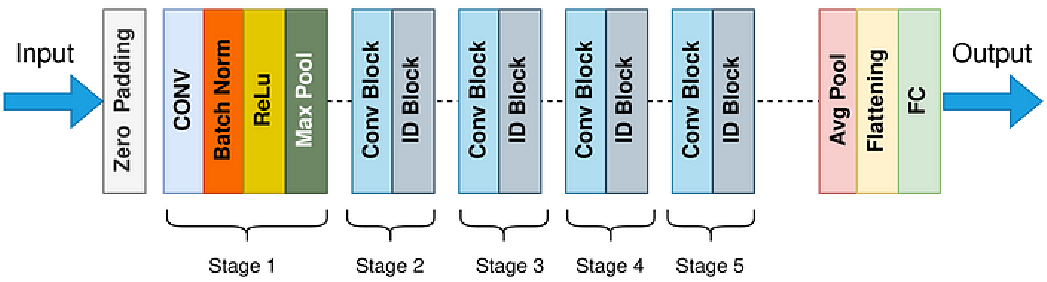
5. **Compile the Model:**
   - Compile the fine-tuned model using an appropriate loss function (e.g., categorical crossentropy for multi-class classification) and an optimizer (e.g., Adam).
6. **Data Preprocessing:**
   - Preprocess the new dataset using the same preprocessing steps applied to the original dataset used to train the pre-trained ResNet model. This may include normalization, resizing, and data augmentation.
7. **Fine-Tuning:**
   - Train the model on the new dataset using the compiled model and frozen convolutional layers. This step allows the new fully connected layers to learn task-specific features while preserving the knowledge embedded in the pre-trained convolutional base.
8. **Unfreeze Convolutional Layers (Optional):**
   - Optionally, unfreeze some of the top layers of the convolutional base and continue training. This allows the model to adapt to more task-specific features present in the new dataset.
9. **Hyperparameter Tuning:**
   - Fine-tune hyperparameters such as learning rate, batch size, and regularization strength to optimize model performance on the new task.
10. **Evaluate and Validate:**
    - Evaluate the fine-tuned model on a validation set to ensure it generalizes well to unseen data. Adjust the model architecture or hyperparameters if necessary.
11. **Inference on Test Set:**
    - Once satisfied with the model's performance, use it for inference on the test set to assess its effectiveness in real-world scenarios.

2M

Benefits of Fine-Tuning with Transfer Learning:

- **Faster Convergence:** Utilizes pre-learned features, accelerating convergence on the new task.
- **Data Efficiency:** Effective even with limited labeled data for the new task.
- **Generalization:** Transfers knowledge from one domain to another, improving model generalization.

ResNet50 Model Architecture



4M

COs Addressed and Cognitive Level

CO No.	Course Outcomes	RBT Level
21CO53.1	Understand and Analyse the fundamentals that drive deep learning networks	L2
21CO53.2	Build, train and apply fully connected neural networks	L3
21CO53.3	Analyze convolutional networks and their role in image processing.	L3
21CO53.4	Implementation of deep learning techniques to solve real-world problems.	L3

Signature of Course Coordinator **Signature of Module Coordinator** **Signature of HOD**