**[IIT-Delhi-Artificial Intelligence and Machine Learning For Industry- Batch02](https://lmsportal.timespro.com/course/view.php?id=438)**

Shailesh Ganpat Belose

[shaileshbelose@gmail.com](mailto:shaileshbelose@gmail.com)

+91 9595833595

Contents

[Overview: 1](#_Toc174114572)

[Part A) ( 50 marks) 1](#_Toc174114573)

[Project structure: 2](#_Toc174114574)

[Epoch Execution: 3](#_Toc174114575)

[Classification Performance Across Epochs: 7](#_Toc174114576)

[Misclassified Data and circumstances 7](#_Toc174114577)

[Part B) (50 marks) 11](#_Toc174114578)

# Overview:

Assignment : Module 4 (Deep Learning)

## Part A) ( 50 marks)

Please train a network for image classification on mini-imagenet datasets (mentioned

below). You should experiment with the following network parameters:

1. number of convolutional (conv) layers

2. fully connected (fc) layers

3. number of filters in different layers

4. Max Pooling

5. training time (number of epochs)

6. Stride

to come up with a study of the effect of these parameters on the classification

performance. Try to improve the performance as much as possible by modifying these

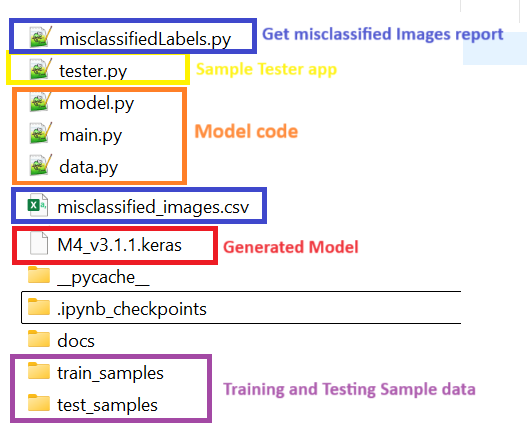
parameters. Please present the results of such a study in the form of a table that shows

the classification performance as a function of these parameters. Also look at some of

the images that are mis-classified and see if there is an explanation for such

mis-classifications.

## Project structure:



### Epoch Execution:

Epoch 1/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 75s 172ms/step - accuracy: 0.1031 - loss: 3.5202 - val\_accuracy: 0.1424 - val\_loss: 3.3395 - learning\_rate: 0.0010

Epoch 2/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 77s 179ms/step - accuracy: 0.2120 - loss: 2.8380 - val\_accuracy: 0.2258 - val\_loss: 2.7869 - learning\_rate: 0.0010

Epoch 3/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 76s 176ms/step - accuracy: 0.2744 - loss: 2.5677 - val\_accuracy: 0.3100 - val\_loss: 2.4273 - learning\_rate: 0.0010

Epoch 4/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 78s 181ms/step - accuracy: 0.3237 - loss: 2.3751 - val\_accuracy: 0.3370 - val\_loss: 2.3080 - learning\_rate: 0.0010

Epoch 5/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 78s 180ms/step - accuracy: 0.3517 - loss: 2.2293 - val\_accuracy: 0.3567 - val\_loss: 2.2232 - learning\_rate: 0.0010

Epoch 6/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 80s 185ms/step - accuracy: 0.3942 - loss: 2.1150 - val\_accuracy: 0.3076 - val\_loss: 2.3971 - learning\_rate: 0.0010

Epoch 7/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 82s 189ms/step - accuracy: 0.4031 - loss: 2.0704 - val\_accuracy: 0.3773 - val\_loss: 2.1492 - learning\_rate: 0.0010

Epoch 8/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 83s 192ms/step - accuracy: 0.4307 - loss: 1.9720 - val\_accuracy: 0.4058 - val\_loss: 2.1195 - learning\_rate: 0.0010

Epoch 9/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 84s 195ms/step - accuracy: 0.4401 - loss: 1.9244 - val\_accuracy: 0.4448 - val\_loss: 1.9131 - learning\_rate: 0.0010

Epoch 10/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 83s 192ms/step - accuracy: 0.4629 - loss: 1.8376 - val\_accuracy: 0.4785 - val\_loss: 1.8734 - learning\_rate: 0.0010

Epoch 11/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 82s 189ms/step - accuracy: 0.4763 - loss: 1.7832 - val\_accuracy: 0.4900 - val\_loss: 1.7960 - learning\_rate: 0.0010

Epoch 12/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 84s 195ms/step - accuracy: 0.4966 - loss: 1.7591 - val\_accuracy: 0.4939 - val\_loss: 1.7831 - learning\_rate: 0.0010

Epoch 13/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 84s 196ms/step - accuracy: 0.5105 - loss: 1.6699 - val\_accuracy: 0.4442 - val\_loss: 2.0536 - learning\_rate: 0.0010

Epoch 14/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 85s 197ms/step - accuracy: 0.5252 - loss: 1.6337 - val\_accuracy: 0.4794 - val\_loss: 1.8033 - learning\_rate: 0.0010

Epoch 15/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 86s 199ms/step - accuracy: 0.5276 - loss: 1.6062 - val\_accuracy: 0.5155 - val\_loss: 1.6873 - learning\_rate: 0.0010

Epoch 16/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 85s 198ms/step - accuracy: 0.5557 - loss: 1.5342 - val\_accuracy: 0.4891 - val\_loss: 1.8055 - learning\_rate: 0.0010

Epoch 17/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 86s 200ms/step - accuracy: 0.5525 - loss: 1.5359 - val\_accuracy: 0.5297 - val\_loss: 1.6851 - learning\_rate: 0.0010

Epoch 18/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 201ms/step - accuracy: 0.5764 - loss: 1.4652 - val\_accuracy: 0.5015 - val\_loss: 1.7617 - learning\_rate: 0.0010

Epoch 19/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 201ms/step - accuracy: 0.5745 - loss: 1.4421 - val\_accuracy: 0.4867 - val\_loss: 1.7984 - learning\_rate: 0.0010

Epoch 20/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 86s 199ms/step - accuracy: 0.5856 - loss: 1.4117 - val\_accuracy: 0.5339 - val\_loss: 1.6774 - learning\_rate: 0.0010

Epoch 21/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 84s 194ms/step - accuracy: 0.5925 - loss: 1.3733 - val\_accuracy: 0.5352 - val\_loss: 1.6322 - learning\_rate: 0.0010

Epoch 22/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 84s 195ms/step - accuracy: 0.6200 - loss: 1.2926 - val\_accuracy: 0.5494 - val\_loss: 1.6323 - learning\_rate: 0.0010

Epoch 23/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 85s 195ms/step - accuracy: 0.6257 - loss: 1.2697 - val\_accuracy: 0.5252 - val\_loss: 1.7026 - learning\_rate: 0.0010

Epoch 24/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 86s 196ms/step - accuracy: 0.6357 - loss: 1.2239 - val\_accuracy: 0.5433 - val\_loss: 1.6299 - learning\_rate: 0.0010

Epoch 25/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 198ms/step - accuracy: 0.6399 - loss: 1.2158 - val\_accuracy: 0.5118 - val\_loss: 1.7547 - learning\_rate: 0.0010

Epoch 26/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 88s 200ms/step - accuracy: 0.6416 - loss: 1.2096 - val\_accuracy: 0.5464 - val\_loss: 1.6305 - learning\_rate: 0.0010

Epoch 27/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 88s 199ms/step - accuracy: 0.6554 - loss: 1.1624 - val\_accuracy: 0.5070 - val\_loss: 1.8831 - learning\_rate: 0.0010

Epoch 28/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 89s 202ms/step - accuracy: 0.6655 - loss: 1.1200 - val\_accuracy: 0.5764 - val\_loss: 1.5384 - learning\_rate: 5.0000e-04

Epoch 29/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 88s 201ms/step - accuracy: 0.6855 - loss: 1.0308 - val\_accuracy: 0.5679 - val\_loss: 1.5530 - learning\_rate: 5.0000e-04

Epoch 30/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 200ms/step - accuracy: 0.7030 - loss: 0.9835 - val\_accuracy: 0.5782 - val\_loss: 1.5058 - learning\_rate: 5.0000e-04

Epoch 31/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 200ms/step - accuracy: 0.7175 - loss: 0.9307 - val\_accuracy: 0.5821 - val\_loss: 1.5067 - learning\_rate: 5.0000e-04

Epoch 32/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 86s 198ms/step - accuracy: 0.7177 - loss: 0.9291 - val\_accuracy: 0.5721 - val\_loss: 1.6049 - learning\_rate: 5.0000e-04

Epoch 33/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 88s 201ms/step - accuracy: 0.7214 - loss: 0.9243 - val\_accuracy: 0.5752 - val\_loss: 1.5878 - learning\_rate: 5.0000e-04

Epoch 34/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 200ms/step - accuracy: 0.7381 - loss: 0.8603 - val\_accuracy: 0.5873 - val\_loss: 1.5461 - learning\_rate: 2.5000e-04

Epoch 35/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 201ms/step - accuracy: 0.7514 - loss: 0.8168 - val\_accuracy: 0.5994 - val\_loss: 1.4928 - learning\_rate: 2.5000e-04

Epoch 36/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 87s 201ms/step - accuracy: 0.7597 - loss: 0.7999 - val\_accuracy: 0.6009 - val\_loss: 1.4816 - learning\_rate: 2.5000e-04

Epoch 37/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 89s 203ms/step - accuracy: 0.7592 - loss: 0.7769 - val\_accuracy: 0.5830 - val\_loss: 1.5817 - learning\_rate: 2.5000e-04

Epoch 38/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 89s 202ms/step - accuracy: 0.7669 - loss: 0.7609 - val\_accuracy: 0.5867 - val\_loss: 1.5584 - learning\_rate: 2.5000e-04

Epoch 39/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 88s 202ms/step - accuracy: 0.7796 - loss: 0.7202 - val\_accuracy: 0.5921 - val\_loss: 1.5240 - learning\_rate: 2.5000e-04

Epoch 40/40

413/413 ━━━━━━━━━━━━━━━━━━━━ 88s 201ms/step - accuracy: 0.7877 - loss: 0.7089 - val\_accuracy: 0.5979 - val\_loss: 1.5119 - learning\_rate: 1.2500e-04

104/104 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.5859 - loss: 1.4943

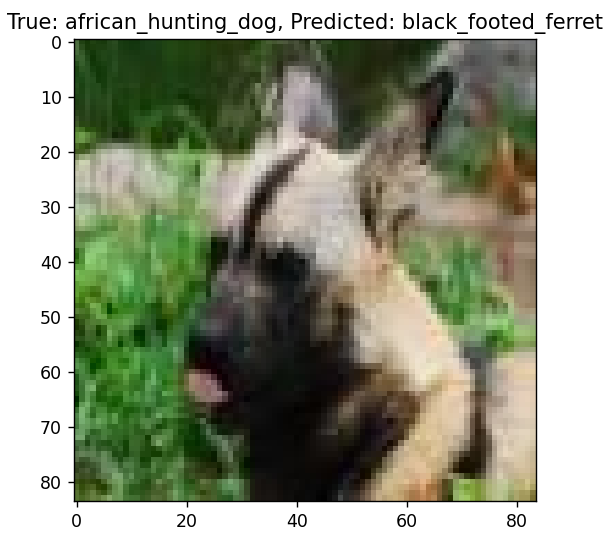
Test accuracy: 0.6009091138839722

### Classification Performance Across Epochs:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Epoch** | **Accuracy** | **Loss** | **Validation Accuracy** | **Validation Loss** | **Learning Rate** |
| 1 | 0.1031 | 3.5202 | 0.1424 | 3.3395 | 0.001 |
| 5 | 0.3517 | 2.2293 | 0.3567 | 2.2232 | 0.001 |
| 10 | 0.4629 | 1.8376 | 0.4785 | 1.8734 | 0.001 |
| 15 | 0.5276 | 1.6062 | 0.5155 | 1.6873 | 0.001 |
| 20 | 0.5856 | 1.4117 | 0.5339 | 1.6774 | 0.001 |
| 25 | 0.6399 | 1.2158 | 0.5118 | 1.7547 | 0.001 |
| 30 | 0.703 | 0.9835 | 0.5782 | 1.5058 | 0.0005 |
| 35 | 0.7514 | 0.8168 | 0.5994 | 1.4928 | 0.00025 |
| 40 | 0.7877 | 0.7089 | 0.5979 | 1.5119 | 0.000125 |
| **Test** | **0.6009** | **1.4943** | **0.6009** | **1.4943** | - |

### Misclassified Data and circumstances

Found some wrong predicted labels against classes from the model, Here we can see the sample images as.



A dog lying on the ground

Description automatically generated

A deer standing in the woods

Description automatically generated with medium confidenceA close up of a bug

Description automatically generated

Many more are there which are listed in cs file named as *misclassified\_images.csv.*

## Part B) (50 marks)

From the models you have trained for Part - A, consider the one that gave you highest

accuracy on test data and perform the following experiments on the same test data.

Given a fully-trained high-performance image classifier model, the question arises

whether the model has really learnt the location of the object in the image or if the

model just classifies the image based on surrounding or contextual cues. In this regard,

to understand the behavior of your model, for some of the selected images (∼ 10

images) from test-set of your dataset, perform occlusion sensitivity experiment as

follows: For each pixel position i(along x-direction), j (along y-direction):

1. consider a window (N × N, choose an appropriate value of N) around (i, j) and replace

the content of the window with gray pixels. Refer [1] figure 7, 8 for more information 2. Pass the modified image (with respect to the position (i, j)) through the model and

note down the probability for the true class into an array. i.e., confidence(i, j). Plot the

confidence array as an image and comment on the observations.

Here in tester app we are browsing one image and iterating occlusion of 28X28 block on 84X84 size image in 3 column 3 rows format. Which looks like below, then we are simply predicting its class\_name and confidence.

A screenshot of a computer screen

Description automatically generatedA screenshot of a duck

Description automatically generatedA collage of a group of bottles

Description automatically generatedA collage of images of a person riding a unicycle

Description automatically generated