Assignment 2

Student Details:
Name: Pradipkumar Vala
Roll No: DA24M012
WANDB Link:
https://wandb.ai/da24m012-iit-madras/A2/reports/DA6401-Assignment-2--VmlldzoxMjM1ODAxNw
Github Link:
https://github.com/Pradiprajvala/da6401_assignment2

DA6401 - Assignment 2

Learn how to use CNNs: train from scratch and finetune a pre-trained model as it is.

Pradipkumar Dilubhai Vala da24m012

Created on April 18 | Last edited on April 19

Instructions

- The goal of this assignment is twofold: (i) train a CNN model from scratch and learn how to tune the hyperparameters and visualize filters (ii) finetune a pre-trained model just as you would do in many real-world applications
- Discussions with other students is encouraged.
- You must use `Python` for your implementation.
- You can use any and all packages from `PyTorch`, `Torchvision` or PyTorch-Lightning. NO OTHER DL library such as `TensorFlow` or `Keras` is allowed. Please confirm with the TAs before using any new external library. BTW, you may want to explore PyTorch-Lightning as it includes `fp16` mixed-precision training, `wandb` integration and many other black boxes eliminating the need for boiler-plate code. Also, do look out for PyTorch2.0.
- You can run the code in a jupyter notebook on colab by enabling GPUs.
- You have to generate the report in the format shown below using `wandb.ai`. You can start by cloning this report using the clone option above. Most of the plots that we have asked for below can be (automatically) generated using the APIs provided by `wandb.ai`
- You also need to provide a link to your GitHub code as shown below. Follow good software engineering practices and set up a GitHub repo for the project on Day 1. Please do not write all code on your local machine and push everything to GitHub on the last day. The commits in GitHu should reflect how the code has evolved during the course of the assignment.
- You have to check Moodle regularly for updates regarding the assignment.

Problem Statement

In Part A and Part B of this assignment you will build and experiment with CNN based image classifiers using a subset of the iNaturalist dataset.

Part A: Training from scratch

Question 1 (5 Marks)

Build a small CNN model consisting of 5 convolution layers. Each convolution layer would be followed by an activation and a max-pooling layer.

After 5 such conv-activation-maxpool blocks, you should have one dense layer followed by the output layer containing 10 neurons (1 for each of the 10 classes). The input layer should be compatible with the images in the iNaturalist dataset dataset.

The code should be flexible such that the number of filters, size of filters, and activation function of the convolution layers and dense layers can be changed. You should also be able to change the number of neurons in the dense layer.

- What is the total number of computations done by your network? (assume m filters in each layer of size $k \times k$ and n neurons in the dense layer)

CNN Computation and Parameter Breakdown

Convolutional Layer Computation

- Each filter performs:
  `k² × c` operations per output position (where `c` is input channels)
- For `m` filters:
  `m × k² × c` operations per output position
- With padding (`filter_size // 2`), spatial dimensions remain the same
- After max-pooling, spatial dimensions are halved

Computation Breakdown

Layer	Computation
First Conv Layer	`m × k² × 3 × h × w` (3 input channels)
Second Conv Layer	`m × k² × m × (h/2) × (w/2)`
Third Conv Layer	`m × k² × m × (h/4) × (w/4)`
Fourth Conv Layer	`m × k² × m × (h/8) × (w/8)`
Fifth Conv Layer	`m × k² × m × (h/16) × (w/16)`
Dense Layer	`m × (h/32) × (w/32) × n`
Output Layer	`n × 10`

Total Computations

$$\text{Total} = m \cdot k^2 \cdot 3 \cdot h \cdot w + m^2 \cdot k^2 \cdot h \cdot w \cdot \left(\frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256}\right) + \frac{m \cdot h \cdot w \cdot n}{1024} + n \cdot 10$$

Total Computations (Assuming height and width = 224)

$$\text{Total computations} \approx 150,528 \cdot m \cdot k^2 + 16,589 \cdot m^2 \cdot k^2 + 49 \cdot m \cdot n + 10 \cdot n$$

- What is the total number of parameters in your network? (assume m filters in each layer of size $k \times k$ and n neurons in the dense layer)

Parameter Count Breakdown

Layer	Parameters
First Conv Layer	`m × (k² × 3 + 1)` (weights + biases)
Second Conv Layer	`m × (k² × m + 1)`
Third Conv Layer	`m × (k² × m + 1)`
Fourth Conv Layer	`m × (k² × m + 1)`
Fifth Conv Layer	`m × (k² × m + 1)`
Dense Layer	`(m × h/32 × w/32 + 1) × n`
Output Layer	`(n + 1) × 10`

Total Parameters

$$\text{Total Parameters} \approx m \cdot k^2 \cdot (3 + 4m) + 5m + \frac{m \cdot h \cdot w \cdot n}{1024} + 11n + 10$$

Total Parameters (Assuming height and width = 224)

$$\text{Total parameters} \approx 3 \cdot m \cdot k^2 + 4 \cdot m^2 \cdot k^2 + 5 \cdot m + 49 \cdot m \cdot n + 11 \cdot n + 10$$

Question 2 (15 Marks)

You will now train your model using the iNaturalist dataset. The zip file contains a train and a test folder. Set aside 20% of the training data, as validation data, for hyperparameter tuning. Make sure each class is equally represented in the validation data. **Do not use the test data for hyperparameter tuning.**

Using the sweep feature in wandb find the best hyperparameter configuration. Here are some suggestions but you are free to decide which hyperparameters you want to explore

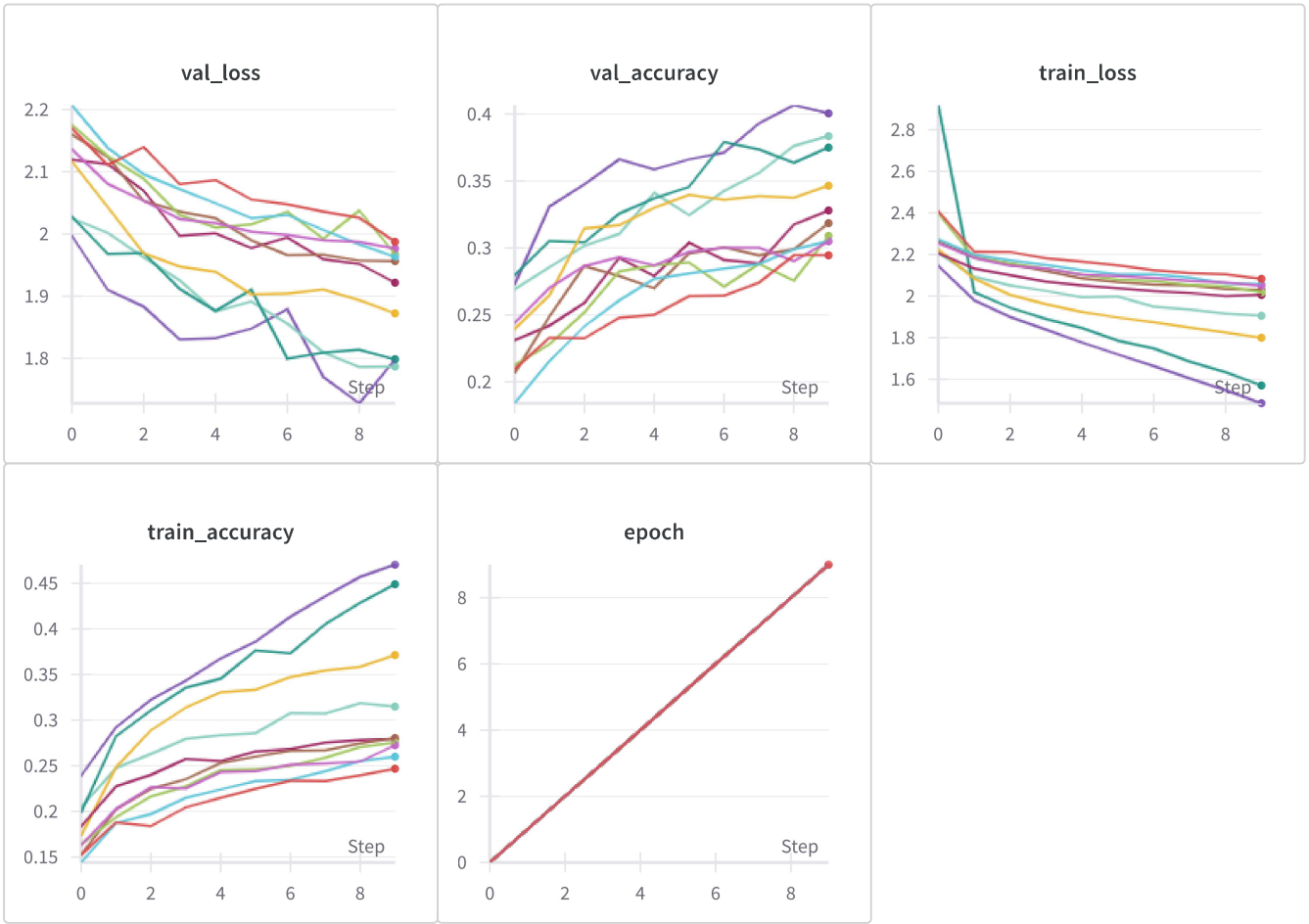
- number of filters in each layer : $32, 64, \ldots$
- activation function for the conv layers: ReLU, GELU, SiLU, Mish, ...

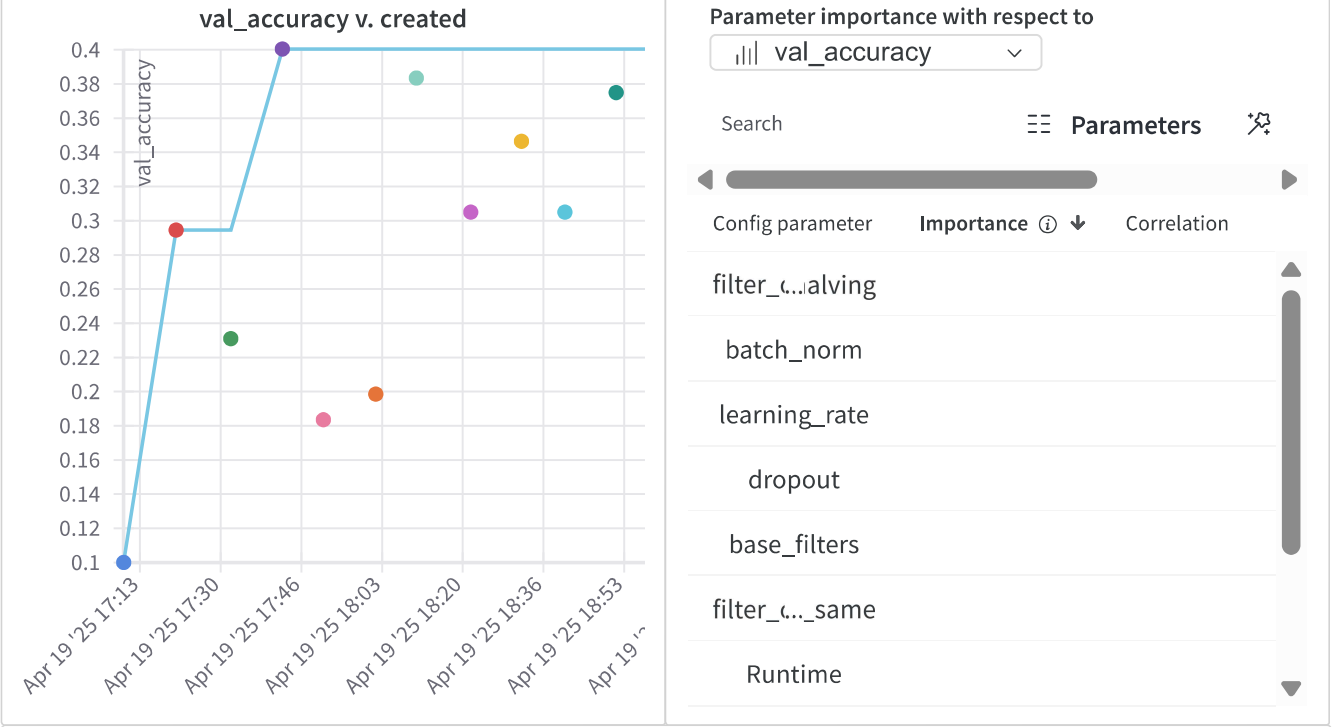
- filter organisation: same number of filters in all layers, doubling in each subsequent layer, halvin in each subsequent layer, etc
- data augmentation: Yes, No
- batch normalisation: Yes, No
- dropout: 0.2, 0.3 (BTW, where will you add dropout? You should read up a bit on this)

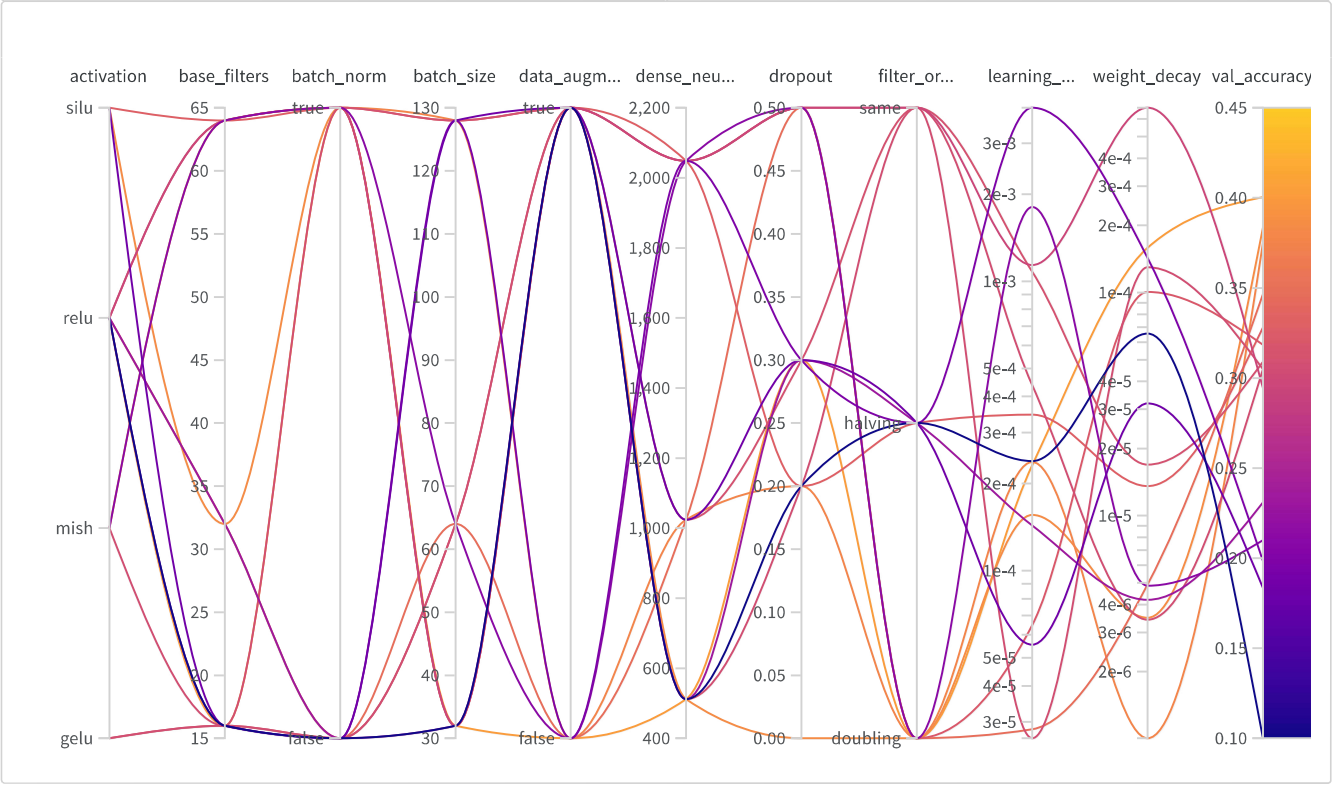
Based on your sweep please paste the following plots which are automatically generated by wandb:

- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

Also, write down the hyperparameters and their values that you sweeped over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried.







Answer

To reduce the number of runs while still achieving high accuracy, I used the following strategies:

Bayesian Optimization: Unlike grid or random search, Bayesian optimization intelligently selects the next set of hyperparameters based on past performance. This reduces the total number of runs needed to find optimal or near-optimal configurations.

Log-Uniform Sampling for learning_rate and weight_decay: Helps efficiently explore very small values that are typically optimal in deep learning.

Hyperparameters

Hyperparameter	Values / Range	Type
`batch_size`	[32, 64, 128]	Discrete
`learning_rate`	1e-5, 1e-2	Continuous
`weight_decay`	1e-6, 1e-3	Continuous
`base_filters`	[16, 32, 64]	Discrete
`filter_organization`	['same', 'doubling', 'halving']	Categorical
`activation`	['relu', 'gelu', 'silu', 'mish']	Categorical
`dense_neurons`	[512, 1024, 2048]	Discrete
`dropout`	[0.0, 0.2, 0.3, 0.5]	Discrete
`batch_norm`	[True, False]	Boolean
`data_augmentation`	[True, False]	Boolean
`epochs`	10 (fixed)	Constant

Question 3 (15 Marks)

Based on the above plots write down some insightful observations. For example,

- adding more filters in the initial layers is better
- Using bigger filters in initial layers and smaller filters in latter layers is better
- ... ...

(Note: I don't know if any of the above statements is true. I just wrote some random comments that came to my mind)

Answer

Insights from Hyperparameter Sweep

Based on the visualizations and Bayesian sweep results, several meaningful patterns emerged:

Observations Based on Hyperparameter Tuning Visualizations

1. **Filter Organization Strategy Matters**

- The `filter_organization.value_halving` configuration shows the highest importance in determining validation accuracy.
- This suggests that using **larger filters in early layers and smaller ones in deeper layers** significantly improves performance.
- Halving filters likely helps in capturing high-level to low-level features more effectively.

2. **Batch Normalization is Beneficial**
   - While not the most important parameter, `batch_norm` is **strongly positively correlated** with validation accuracy.
   - Models with `batch_norm = True` consistently perform better.
3. **Lower Dropout Rates Improve Accuracy**
   - Higher dropout rates correlate negatively with performance.
   - Models with **low dropout values (< 0.3)** tend to achieve better validation accuracy.
4. **Learning Rate Requires Careful Tuning**
   - `learning_rate` has moderate importance and positive correlation with accuracy.
   - Best-performing models typically use a learning rate in the range of `5e-4` to `2e-3`.
5. **Dense Layer Size Should Be Sufficient**
   - Models with **more neurons (~2000)** in the dense layers generally perform better.
   - Very small dense layers (< 800 neurons) underperform.
6. **Data Augmentation Shows Mixed Impact**
   - `data_augmentation` has low importance.
   - No consistent trend is observed; models with and without augmentation both yield varied performance.
7. **Improvement Plateau Over Time**
   - Validation accuracy improves sharply in early trials.
   - After reaching around **0.4 val_accuracy**, improvements become incremental.
   - Suggests **diminishing returns** and supports using early stopping or performance-based pruning.

Question 4 (5 Marks)

You will now apply your best model on the test data (You shouldn't have used test data so far. All the above experiments should have been done using train and validation data only).

- Use the best model from your sweep and report the accuracy on the test set.
- Provide a 10×3 grid containing sample images from the test data and predictions made by your best model (more marks for presenting this grid creatively).
- **(UNGRADED, OPTIONAL)** Visualise all the filters in the first layer of your best model for a random image from the test set. If there are 64 filters in the first layer plot them in an 8×8 grid.
- **(UNGRADED, OPTIONAL)** Apply guided back-propagation on any 10 neurons in the CONV5 layer and plot the images which excite this neuron. The idea again is to discover interesting patterns which excite some neurons. You will draw a 10×1 grid below with one image for each of the 10 neurons.

Model Predictions on Test Data

Pred: Mollusca
True: Mollusca

Pred: Animalia
True: Reptilia

Pred: Insecta
True: Insecta

Pred: Reptilia
True: Reptilia

Pred: Amphibia
True: Amphibia

Pred: Reptilia
True: Aves

Pred: Amphibia
True: Amphibia

Pred: Arachnida
True: Arachnida

Pred: Fungi
True: Mollusca

Pred: Aves
True: Aves

Pred: Animalia
True: Animalia

Pred: Mollusca
True: Mollusca

Pred: Mollusca
True: Mammalia

Pred: Animalia
True: Animalia

Pred: Animalia
True: Animalia

Pred: Amphibia
True: Amphibia

Pred: Reptilia
True: Reptilia

Pred: Mollusca
True: Plantae

Pred: Aves
True: Aves

Pred: Aves
True: Aves

Pred: Reptilia
True: Reptilia

Pred: Plantae
True: Plantae

Pred: Animalia
True: Animalia

Pred: Animalia
True: Amphibia

Pred: Aves
True: Aves

Pred: Mollusca
True: Fungi

Pred: Mammalia
True: Aves

Pred: Plantae
True: Plantae

Pred: Mammalia
True: Aves

Pred: Animalia
True: Reptilia

Question 5 (10 Marks)

Paste a link to your github code for Part A

Example: https://github.com/<user-id>/da6401_assignment2/partA;

- We will check for coding style, clarity in using functions and a `README` file with clear instructions on training and evaluating the model (the 10 marks will be based on this).
- We will also run a plagiarism check to ensure that the code is not copied (0 marks in the assignment if we find that the code is plagiarised).
- We will also check if the training and test data has been split properly and randomly. You will get marks on the assignment if we find any cheating (e.g., adding test data to training data) to get higher accuracy.

https://github.com/Pradiprajvala/da6401_assignment2/tree/main/partA

Part B : Fine-tuning a pre-trained model

Question 1 (5 Marks)

In most DL applications, instead of training a model from scratch, you would use a model pre-trained on a similar/related task/dataset. From `torchvision`, you can load **ANY ONE** model (`GoogLeNet`, `InceptionV3`, `ResNet50`, `VGG`, `EfficientNetV2`, `VisionTransformer` etc.) pre-trained on the ImageNet dataset. Given that ImageNet also contains many animal images, it stands to reason that using a model pre-trained on ImageNet maybe helpful for this task.

You will load a pre-trained model and then fine-tune it using the naturalist data that you used in the previous question. Simply put, instead of randomly initialising the weigths of a network you will use the weights resulting from training the model on the ImageNet data (`torchvision` directly provides these weights). Please answer the following questions:

- The dimensions of the images in your data may not be the same as that in the ImageNet data. How will you address this?

Answer

To ensure compatibility with the pre-trained model, I applied the following image transformations:

- **Resize and Crop**:
  Each image is first resized to 256 pixels on the shorter side using `transforms.Resize(256)`, followed by a center crop to 224×224 pixels using `transforms.CenterCrop(224)`. This standardizes all images to the expected input size for most ImageNet pre-trained models.
- **Normalization**:
  I normalized the images using the ImageNet mean and standard deviation values:
  - `mean = [0.485, 0.456, 0.406]`
  - `std = [0.229, 0.224, 0.225]`
    This step ensures that the input images have the same statistical properties as those used during the model's original training.

This preprocessing pipeline guarantees that all images from the naturalist dataset, regardless of their original dimensions, are properly formatted and normalized for optimal performance with the pre-trained model.

- ImageNet has 1000 classes and hence the last layer of the pre-trained model would have 1000 nodes. However, the naturalist dataset has only 10 classes. How will you address this?

Adapting the Pre-trained Model for a 10-Class Classification Task

To address the mismatch between the number of output classes in the pre-trained model and our target dataset, I replaced the final fully connected (FC) layer:

```
# Get the number of features in the last layer
num_ftrs = model.fc.in_features

# Replace the final fully connected layer with a new one that has 10 outputs
model.fc = nn.Linear(num_ftrs, 10)
```

(Note: This question is only to check the implementation. The subsequent questions will talk about how exactly you will do the fine-tuning.)

Question 2 (5 Marks)

You will notice that `GoogLeNet`, `InceptionV3`, `ResNet50`, `VGG`, `EfficientNetV2`, `VisionTransformer` are very huge models as compared to the simple model that you implemented in Part A. Even fine-tuning on a small training data may be very expensive. What is a common trick used to keep the training tractable (you will have to read up a bit on this)? Try different variants of this trick and fine-tune the model using the iNaturalist dataset. For example, '___'ing all layers except the last

layer, '___'ing upto k layers and '___'ing the rest. Read up on pre-training and fine-tuning to understand what exactly these terms mean.

Write down the at least 3 different strategies that you tried (simple bullet points would be fine).

Option 1: Fine-tune Only the Final Layer (Freeze Earlier Layers)

- Keep all convolutional layers frozen (weights fixed).
- Only train the new classification head.
- Most computationally efficient.
- May not adapt well to datasets that differ significantly from the original training data.

Option 2: Progressive Unfreezing (Gradual Fine-tuning)

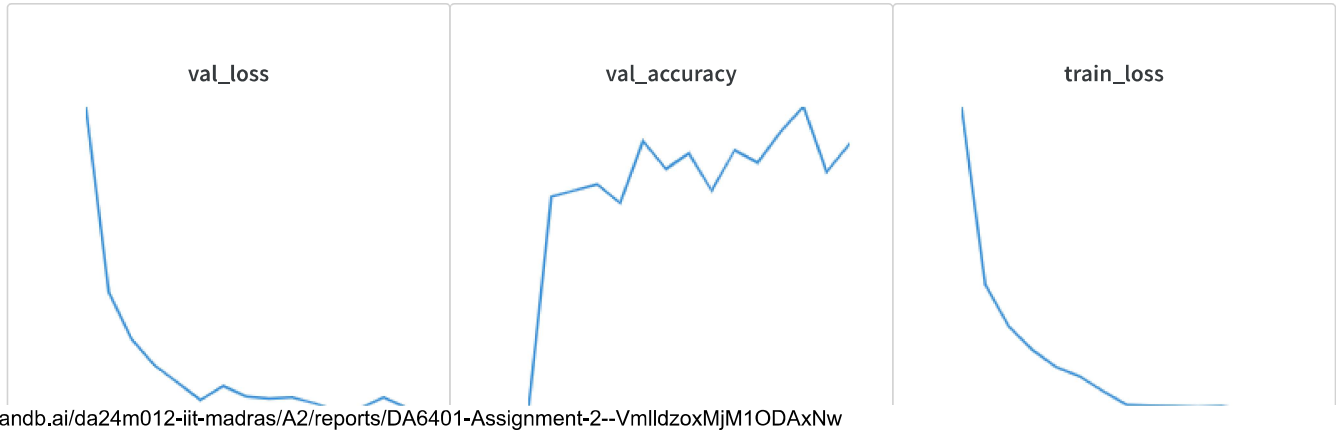
- Start by training only the final layers.
- Gradually unfreeze and train earlier layers.
- Allows the model to adapt high-level features first, then fine-tune more fundamental features.
- Often yields better performance than Option 1 while being more efficient than full fine-tuning.

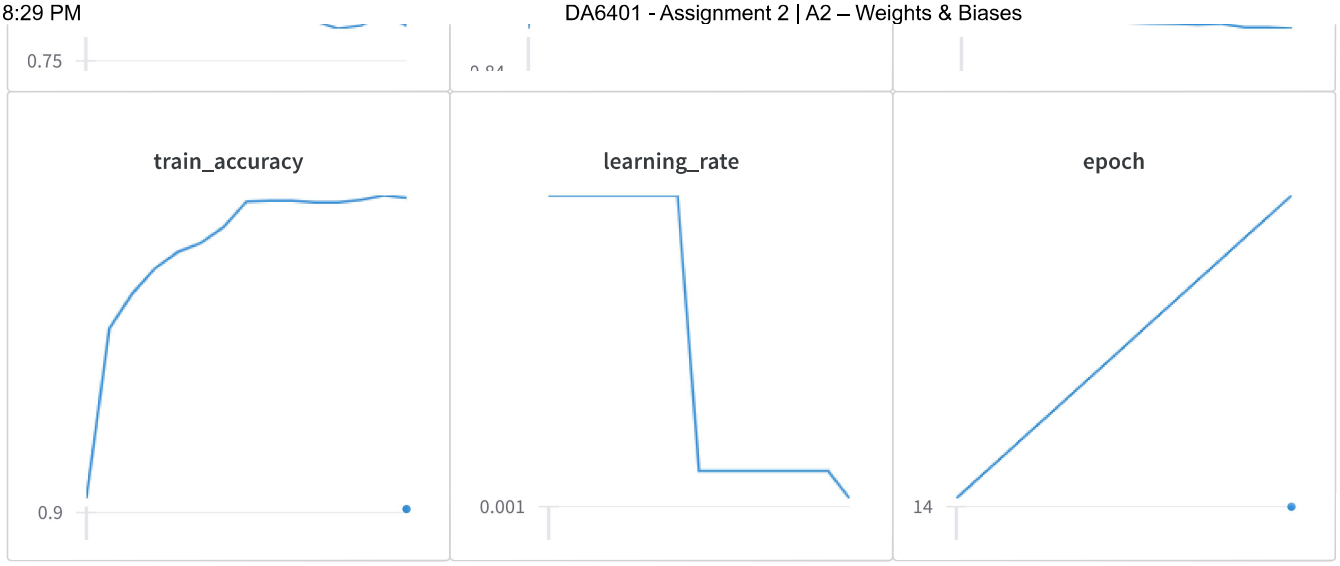
Option 3: Feature Extraction with a Custom Head

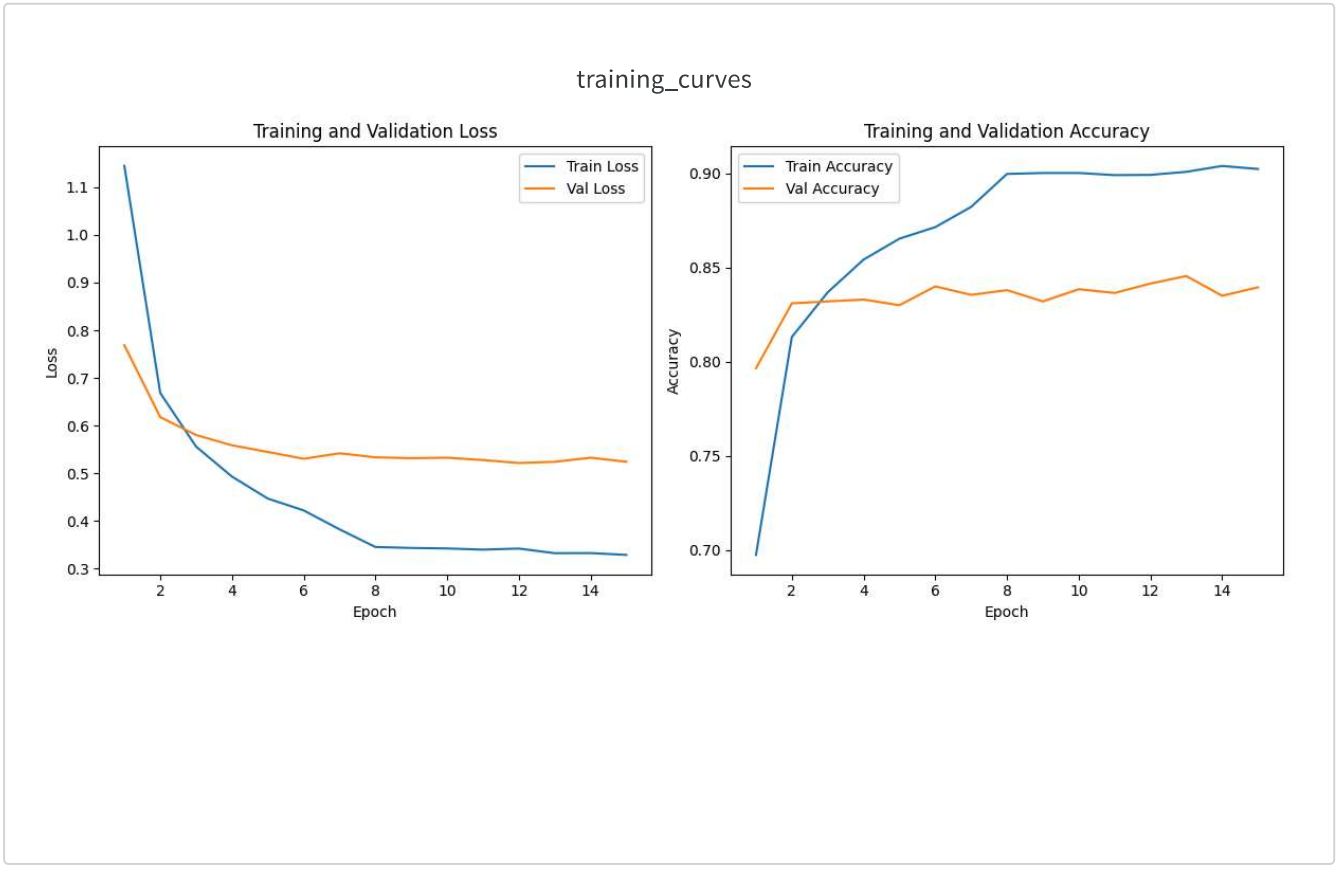
- Freeze all original network layers and use them only as feature extractors.
- Add and train a custom network head (e.g., multiple new layers) on top of the frozen base.
- Enables building a more complex and task-specific classifier.
- Suitable when the base features are transferable but the task is significantly different.

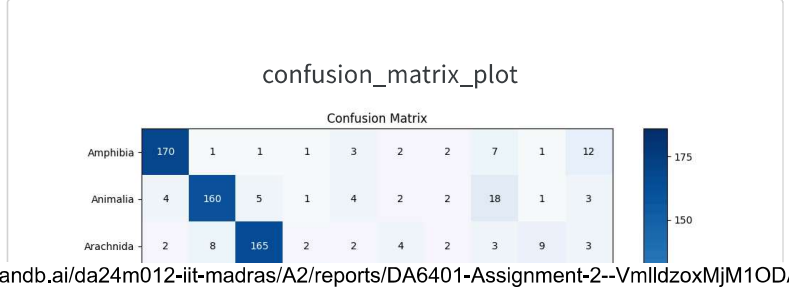
Question 3 (10 Marks)

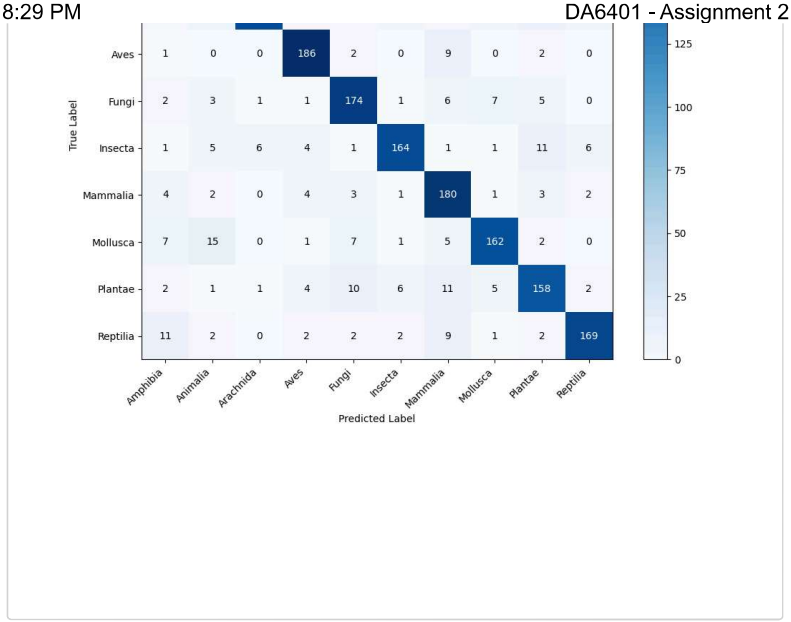
Now fine-tune the model using **ANY ONE** of the listed strategies that you discussed above. Based on these experiments write down some insightful inferences comparing training from scratch and fine-tuning a large pre-trained model.




0.75
train_accuracy
0.9
learning_rate
0.001
epoch
14


training_curves
Training and Validation Loss
Train Loss
Val Loss
Loss
Epoch
Training and Validation Accuracy
Train Accuracy
Val Accuracy
Accuracy
Epoch


confusion_matrix_plot
Confusion Matrix
Amphibia
Animalia
Arachnida



Insights from Fine-Tuning & Training from Scratch

Performance Comparison

Accuracy

- **Fine-tuning:** Achieved ~84% validation accuracy after 10 epochs.
- **Training from scratch:** Only reached 43% validation accuracy after 10 epochs.

Training Time

- Fine Tuned model took very less time compared to model from scratch to give better accuracy.

Learning Curve

- **Fine-tuning:** Shows rapid initial improvement with early plateau.
- **Training from scratch:** Exhibits much slower, gradual improvement.

Question 4 (10 Marks)

Paste a link to your GitHub code for Part B

Example: https://github.com/<user-id>/da6401_assignment2/partB

Follow the same instructions as in Question 5 of Part A.

https://github.com/Pradiprajvala/da6401_assignment2/tree/main/partB

(UNGRADED, OPTIONAL) Part C : Using a pre-trained model as it is

Self Declaration

I, Pradipkumar_Vala_DA24M012 (Roll no: DA24M012), swear on my honour that I have written the code and the report by myself and have not copied it from the internet or other students.