Assignment 2

Student Details:

Name: Mohit Singh

Roll no: DA24M010

WandB link: https://wandb.ai/da24m010-indian-institute-of-technology-madras/DA6401%20Assignments/reports/DA6401-Assignment-2--VmlldzoxMjExMDQwNA

GitHub link: https://github.com/DA24M010/da6401_Assignment2.git

DA6401 - Assignment 2

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

Mohit Singh da24m010

Created on April 3 | Last edited on April 18

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)
- 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)

1. Total Number of Computations:

Image shape taken is (224, 224, 3) as the input for the network.

assuming m filters in each layer of size $k \times k$ and n neurons in the dense layer

Conv1 computations : $3 * k^2 * 112^2 * m$

Conv2 computations : $m * k^2 * 56^2 * m$

Conv3 computations : $m * k^2 * 28^2 * m$

Conv4 computations: $m * k^2 * 14^2 * m$

Conv5 computations : $m * k^2 * 7^2 * m$

Dense layer computations : $(7*7*m) * n + n * 10 = 49*m*n + 10*n$

Total computations = Computations in (conv1 + conv2 + conv3 + conv4 + conv5 + dense) layer =

$$\mathbf{m * k^2 * (3*112^2 + 56^2 + 28^2 + 14^2 + 7^2) + 49*m*n + 10*n}$$

2. Total Number of Parameters:

bias term is also added in the parameters

Conv1 parameters : $m * 3 * k^2 + m$

Conv2-5 parameters : $m * m * k^2 + m$

Conv params = $m * 3 * k^2 + m + 4 * (m * m * k^2 + m)$ // As there are 4 layers in Conv2-5

$= m * 3 * k^2 + m + 4m^2k^2 + 4m$

$= 4m^2k^2 + 3mk^2 + 5m$

Dense layers parameters : Parameters in first dense layer + parameters in output layer = (49*m*n + n) + (n * 10 + 10)

Total parameters = Parameters in (conv + dense) layer = conv params + (49*m*n + n + 10n + 10) =

$4m^2k^2 + 3mk^2 + 5m + 49mn + 11n + 10$

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, ...
- 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.

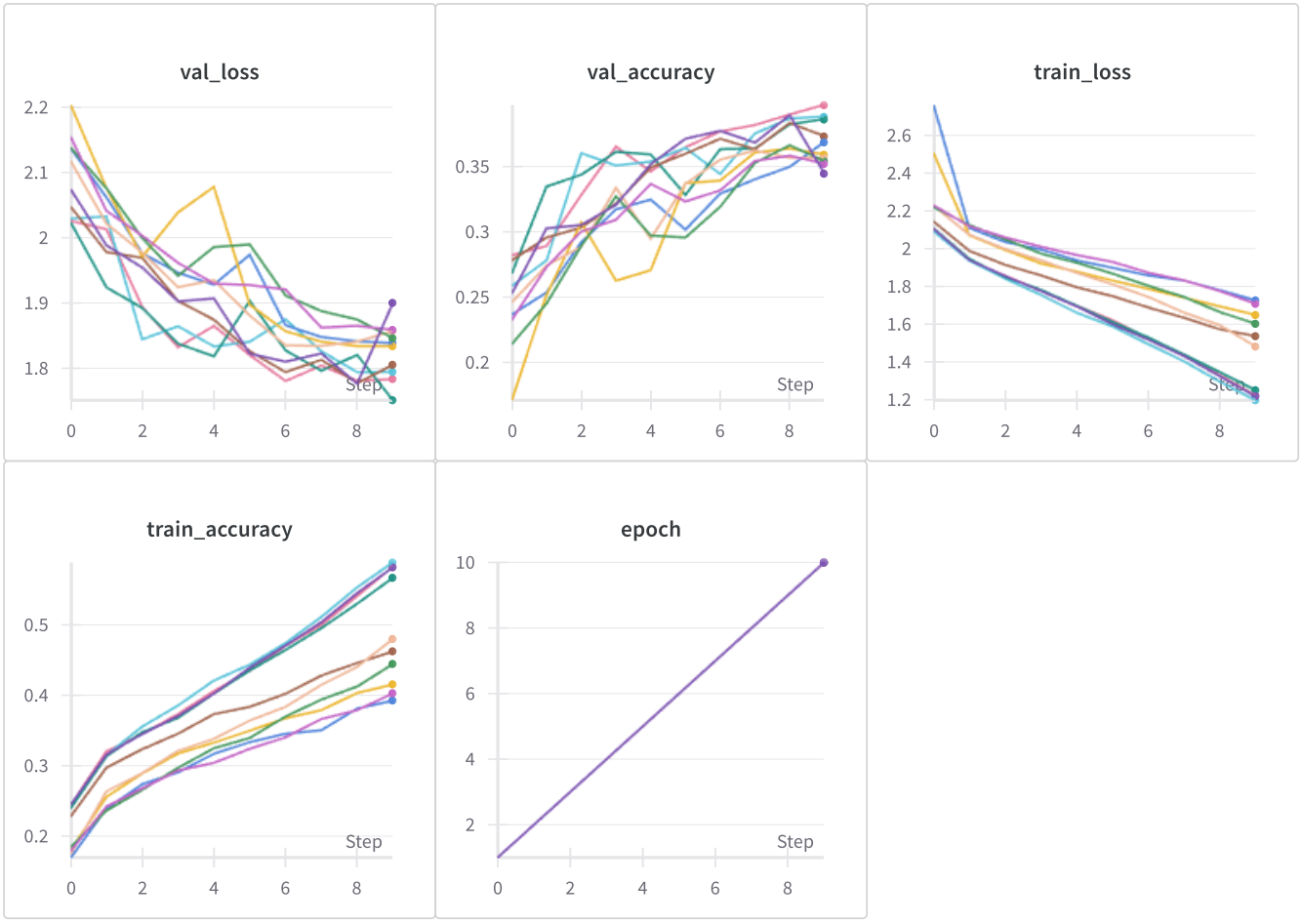
Hyperparameters Configurations:

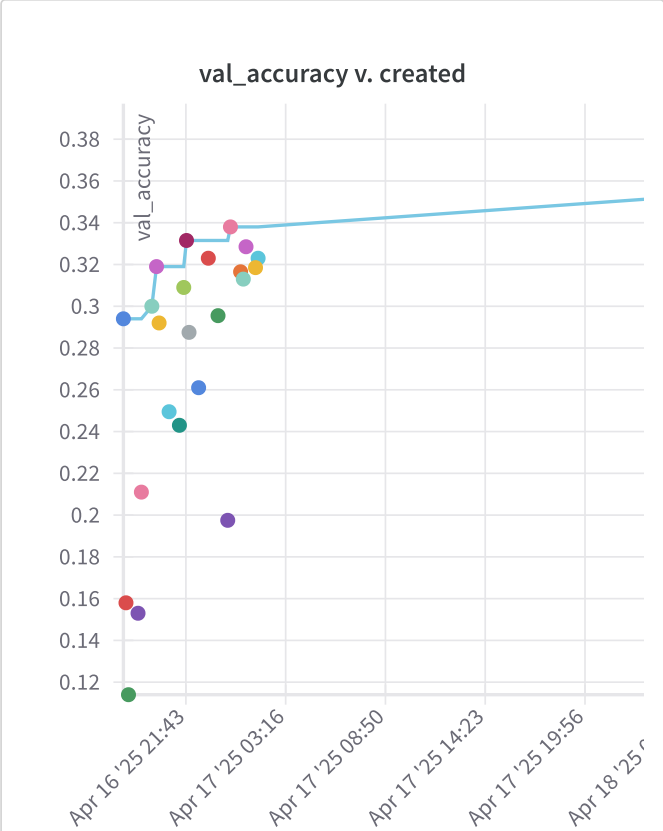
- **num_filters:** (Number of filters in the convolution layers) values = [32, 64, 128]
- **filter_multiplier**: (Multiplier for organising the filters, 1 means all conv layers have same filter, 0.5 means halving the filter value in subsequent layers, 2 means doubling the filter value) values = [0.5, 1, 2]
- **kernel_size**: (Size of k*k kernel used in conv layers) values = [3, 5]
- **activation**: (Activation function used in conv and dense layers) values = [ReLU, leaky ReLU, GELU, SILU, MISH]
- **dropout_rate**: (Dropout percentage applied in conv and dense layers) values = [0.2, 0.5, 0.7]

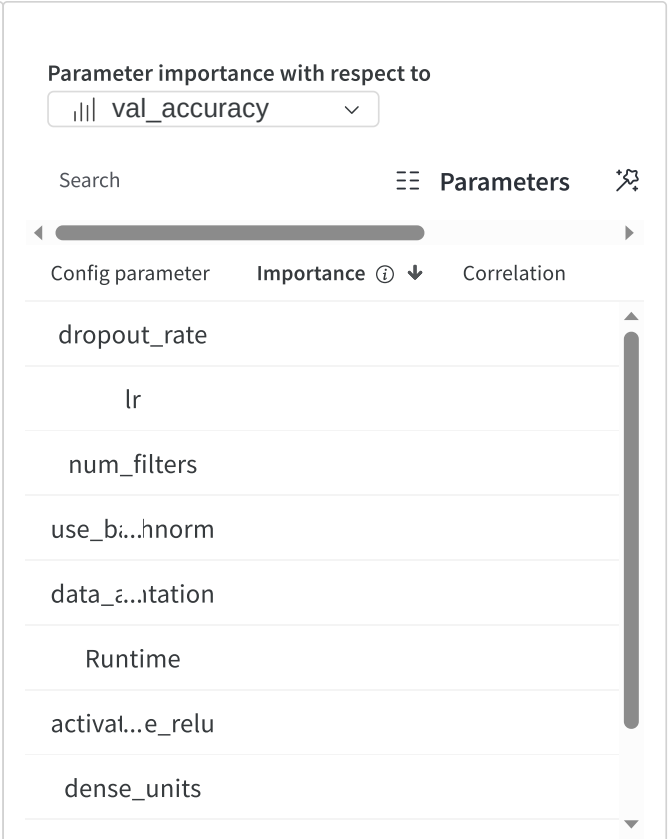
- **use_batchnorm**: (if True applies batchnorm in conv layers else no batch normalization) values = [True, False]
- **dense_units**: (number of nodes in the dense layer) values = [512, 1024]
- **data_augmentation**: (applies data augmentation(random rotation, flip, crop, jitter) in training dat if Tru) values = [True, False]
- **lr**: (learning rate) values = [0.01, 0.001, 0.0001]

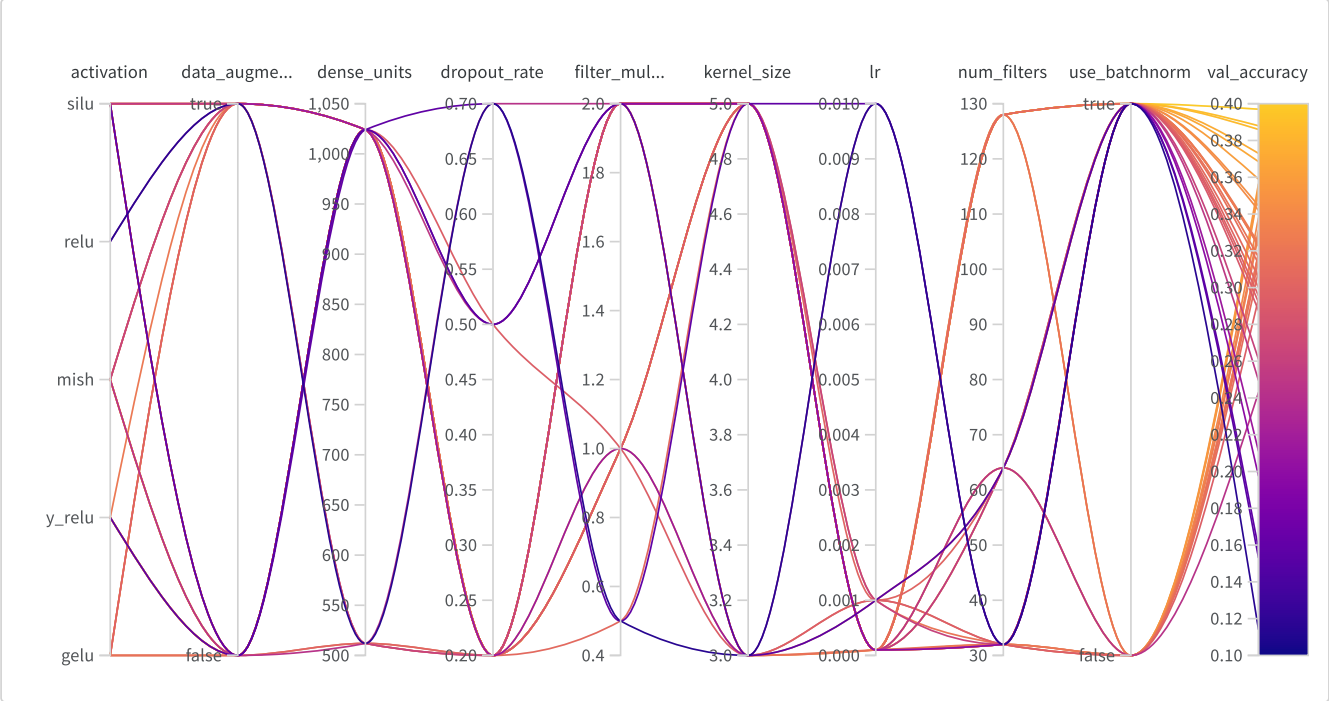
Strategy used to reduce the number of runs while still achieving a high accuracy:

- Set "method": "bayes" in the sweep configuration for more intelligent and efficient hyperparameter search.
- Specified the metric as "val_accuracy" with the goal to "maximize", ensuring the sweeps focus on improving validation accuracy.
- Tried various combinations of hyperparameters (like number of filters, filter size, activation functions, dense layer neurons etc.). Observed that experimenting with these combinations helped the model converge faster to higher accuracy.
- Adjusted the hyperparameter search space based on previous sweep results, narrowing down to promising ranges and excluding poor-performing configurations.









Question 3 (15 Marks)

Based on the above plots write down some insightful observations.

Observations:

- Dropout Rate and Learning Rate Are Highly Influential parameters, the parameter importance chart shows that both dropout_rate and lr (learning rate) have the highest impact on validation accuracy. The runs that achieved the highest validation accuracies consistently utilized a dropout rate of 0.2 and a learning rate of 0.0001.
- Runs with a higher num_filters like 64, 128 tend to achieve higher validation accuracy, as indicate by the positive correlation in the parameter importance chart and the parallel coordinates plot.
- Runs with a higher filter multiplier like 2, 1 tend to achieve higher validation accuracy, where val accuracy for 2 outperformed the filter_multiplier of 1.
- Increasing dense_units (number of neurons in the dense layer) generally leads to improved accuracy.
- Choice of activation function didn't drastically changed the validation accuracy. It had the same average for every activation.
- The use of data augmentation and batchnorm shows a slight positive effect on validation accuracy but its impact is less than that of other parameters maybe due to the data and model complexity.

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).

Best model hyperparameters:

"num_filters" : 128,

"filter_multiplier": 1,

"kernel_size": 5,

"activation": "mish",

"dropout_rate": 0.2,

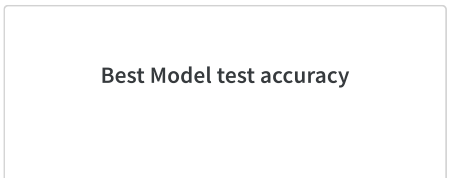
"use_batchnorm": True,

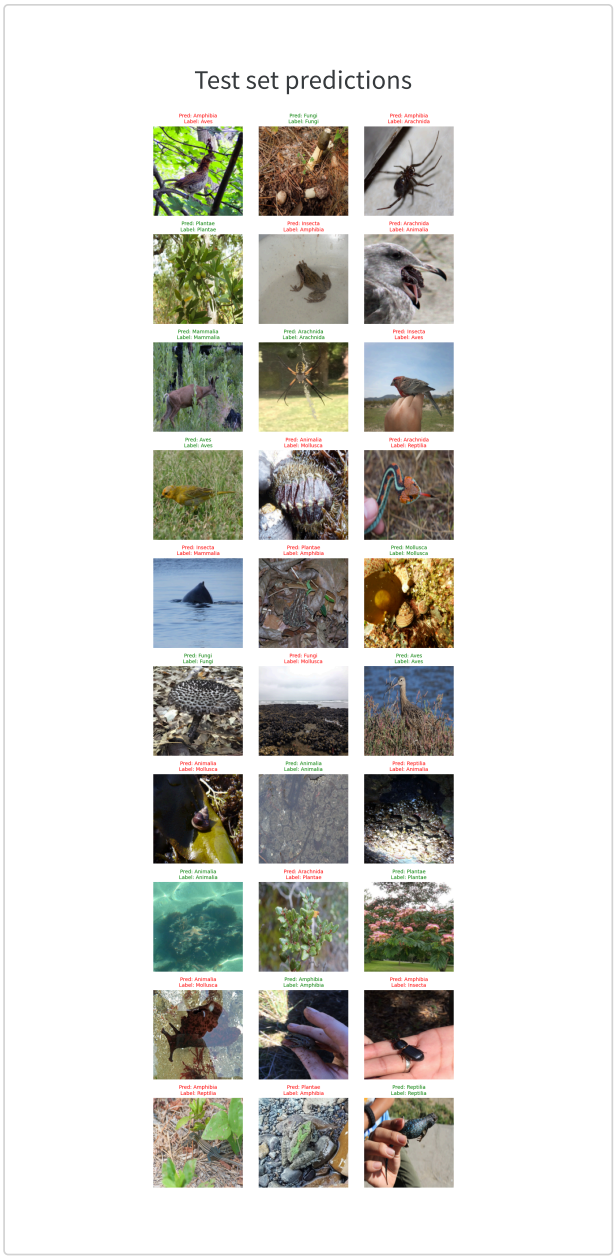
"dense_units": 1024,

"data_augmentation": False,

"lr": 0.0001

Best model test accuracy : 40.05 %





Question 5 (10 Marks)

Paste a link to your github code for Part A

gihub link : https://github.com/DA24M010/da6401_Assignment2.git

part A : https://github.com/DA24M010/da6401_Assignment2/tree/master/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.

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?

Matching the dimensions

I would use PyTorch's transformation pipeline to resize all images to the dimensions required by the pre-trained model and apply the same preprocessing steps used during ImageNet training. The input image size is 224*224 for VGG which is same to the size I am using in the CNN model training.

- 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?

Handling the Output Layer Mismatch

Remove the final fully connected layer of the pre-trained model and replace it with a new layer that outputs 10 classes. Or we can add some fully connected layers after the output layer of ImageNet and add a layer of size 10 at the last for the output in our dataset.

(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)

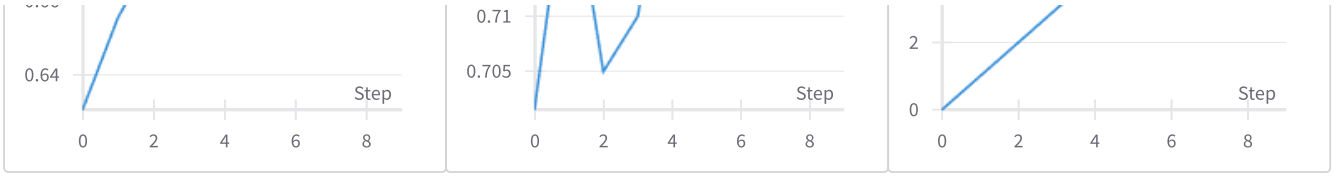
You will notice that `GoogLeNet`, `InceptionV3`, `ResNet50`, `VGG`, `EfficientNetV2`, `VisionTransformer` are very huge models as compared to the simple model that you implemented in Part A. Even fine-tuning on a small training data may be very expensive. What is a common trick used to keep the training 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).

- Strategy1: Froze all convolutional layers, added new fully connected layers after the convolutiona output, and trained only these new layers for the iNaturalist12K classification task.
- Strategy2: Froze all layers including the dense layers, inserted new layers after the original 1000-class output, and trained only these new dense layers to produce 10-class outputs.
- Strategy3: Froze the first 3 convolutional blocks of VGG to retain generic feature extraction, while fine-tuning the last 2 convolutional blocks and all fully connected layers till output layer of 10 to adapt to the iNaturalist dataset.

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.



Inference on training CNN model and fine tuning a large pre-trained model:

- **Time Requirements**: Fine-tuning the VGG model required only 12 minutes 39 seconds, as it involved training a small subset of model parameters while keeping others frozen. In contrast, training the CNN model from scratch took approximately 30 minutes on average with GPU.
- **Parameter Optimization:** The CNN trained from scratch required optimizing approximately 10,000,000 parameters, whereas fine-tuning only required updating a fraction of VGG's parameters, significantly reducing computational load.
- **Accuracy Comparison:** After 10 epochs, the fine-tuned VGG model achieved 72.35% test accuracy and 69.9% validation accuracy, substantially outperforming the best CNN model trained from scratch (40.05% test accuracy, 39.7% validation accuracy).
- **Efficiency-Performance Tradeoff:** Despite requiring less than half the training time, the fine-tune model delivered nearly double the accuracy of the model trained from scratch.
- **Storage Requirements:** The pre-trained VGG model required downloading approximately 500 MB of checkpoint files, whereas the model trained from scratch required no external downloads.
- **Inference Time:** Inference times for VGG model was relatively higher compared to the inference times for CNN model as the model complexity of VGG model is more compared to simple CNN model.

Question 4 (10 Marks)

Paste a link to your GitHub code for Part B

github link : https://github.com/DA24M010/da6401_Assignment2.git

part B : https://github.com/DA24M010/da6401_Assignment2/tree/master/partB

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

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

Question 1 (0 Marks)

Object detection is the task of identifying objects (such as cars, trees, people, animals) in images. Over the past 6 years, there has been tremendous progress in object detection with very fast and accurate models available today. In this question you will use a pre-trained YoloV3 model and use

it in an application of your choice. Here is a cool demo of YoloV2 (click on the image to see the demo on youtube).

Go crazy and think of a cool application in which you can use object detection (alerting lab mates of monkeys loitering outside the lab, detecting cycles in the CRC corridor, ...).

Make a similar demo video of your application, upload it on youtube and paste a link below (similar to the demo I have pasted above).

Also note that I do not expect you to train any model here but just use an existing model as it is. However, if you want to fine-tune the model on some application-specific data then you are free to do that (it is entirely up to you).

Notice that for this question I am not asking you to provide a GitHub link to your code. I am giving you a free hand to take existing code and tweak it for your application. Feel free to paste the link of your code here nonetheless (if you want).

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

Self Declaration

I, Mohit Singh (Roll no: DA24M010), 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.