CS6910 - Assignment 2

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

Omlan Jyoti Mondal ge22m012

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 GitHub 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 $\mathrm{k} \times \mathrm{k}$ 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)

Answer:

(a)

Notationsused:

No.of filters in each layer = m (Given),

Size of each filter = k * k (including in max pooling layer),

padding = 0,

stride = 1,

1st CNN Layer:

filter Size = $k \times k \times 3$

Dimensions of the input image - (300,300,3) with m filters(Given) in each conv layer.

Dimensions of filters acting on input layer - (k,k,3)

Dimensions of the CONV1 layer - $(300-k+1,300-k+1,m) = (301-k,301-k,m)$

No. of multiplications = $(301-k \times 301-k) \times m \times k \times k \times 3$

No. of additions = $(301-k \times 301-k \times m) \times (k \times k -1) \times 3$

Dimensions of the output of the MaxPool1 layer = $(301-k-k+1,301-k-k+1,m)=(302-2k,302-2k,m)$

Dimensions of the CONV2 layer - $(302-2k-k+1,302-2k-k+1,m) = (303-3k,303-3k,m)$

No. of multiplications = $(303-3k \times 303-3k) \times m \times m \times k \times k$

No. of additions = $(303-3k \times 303-3k \times m) \times (k \times k -1) \times m$

Dimensions of the output of the MaxPool2 layer = $(303-3k-k+1, 303-3k-k+1, m) = (304-4k, 304-4k, m)$

Dimensions of the CONV3 layer - $(304-4k-k+1, 304-4k-k+1, m) = (305-5k, 305-5k, m)$

No. of multiplications = $(305-5k \times 305-5k) \times m \times m \times k \times k$

No. of additions = $(305-5k \times 305-5k \times m) \times (k \times k -1) \times m$

Dimensions of the output of the MaxPool3 layer = $(305-5k-k+1, 305-5k-k+1, m) = (306-6k, 306-6k, m)$

Dimensions of the CONV4 layer - $(306-6k-k+1, 306-6k-k+1, m) = (307-7k, 307-7k, m)$

No. of multiplications = $(307-7k \times 307-7k) \times m \times m \times k \times k$

No. of additions = $(307-7k \times 307-7k \times m) \times (k \times k -1) \times m$

Dimensions of the output of the MaxPool4 layer = $(307-7k-k+1, 307-7k-k+1, m) = (308-8k, 308-8k, m)$

Dimensions of the CONV5 layer - $(308-8k-k+1, 308-8k-k+1, m) = (309-9k, 309-9k, m)$

No. of multiplications = $(309-9k \times 309-9k) \times m \times m \times k \times k$

No. of additions = $(309-9k \times 309-9k \times m) \times (k \times k -1) \times m$

Dimensions of the output of the MaxPool5 layer = $(309-9k-k+1, 309-9k-k+1, m) = (310-10k, 310-10k, m)$

No. of neurons after flattening = $m \times 310-10k \times 310-10k$

No. of neurons in the dense layer = n

No. of multiplications = $m \times 310-10k \times 310-10k \times n$

No. of additions = $m \times 310-10k \times 310-10k \times n$

No. of neurons in output layer = 10

No. of multiplications = $10 \times n$

No. of additions = $10 \times n$

Therefore, total no. of multiplications = $((301-k \times 301-k) \times m \times k \times k \times 3) + ((303-3k \times 303-3k) \times m \times m \times k \times k) + ((305-5k \times 305-5k) \times m \times m \times k \times k) + ((307-7k \times 307-7k) \times m \times m \times k \times k) + ((309-9k \times 309-9k) \times m \times m \times k \times k) + (m \times 310-10k \times 310-10k \times n) + (10 \times n)$

Total no. of additions = $((301-k \times 301-k) \times m \times (k \times k -1) \times 3) + ((303-3k \times 303-3k) \times m \times m \times (k \times k - 1)) + ((305-5k \times 305-5k) \times m \times m \times (k \times k - 1)) + ((307-7k \times 307-7k) \times m \times m \times (k \times k - 1)) + ((309-9k \times 309-9k) \times m \times m \times (k \times k - 1)) + (m \times 310-10k \times 310-10k \times n) + (10 \times n)$

(b)

Layer 1:

No.of parameters: $k \times k \times 3 \times m$

Layer 2:

No.of parameters: $k \times k \times m \times m$

Layer 3:

No.of parameters: $k \times k \times m \times m$

Layer 4:

No.of parameters: $k \times k \times m \times m$

Layer 5:

No.of parameters: $k \times k \times m \times m$

No. of parameters in dense layer: $(m \times 310-10k \times 310-10k \times n) + n$

No. of parameters in output layer: $(10 \times n) + n$

Therefore, total no. of parameters in the network = $(k \times k \times 3 \times m) + 4(k \times k \times m \times m) + ((m \times 310-10k \times 310-10k \times n) + n) + ((10 \times n) + n)$

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, halving 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

The following are the hyperparameter configurations that I used for the wandb sweeps:

filter organisation: ['same','double','half']

Number of neurons in Dense Layer: [64, 128, 256, 512]

Batch Normalization: [True, False]

Data Augmentation: ['yes', 'no']

Batch Size: [32, 64]

Dropout: [0, 0.2, 0.4]

Epochs: [10]

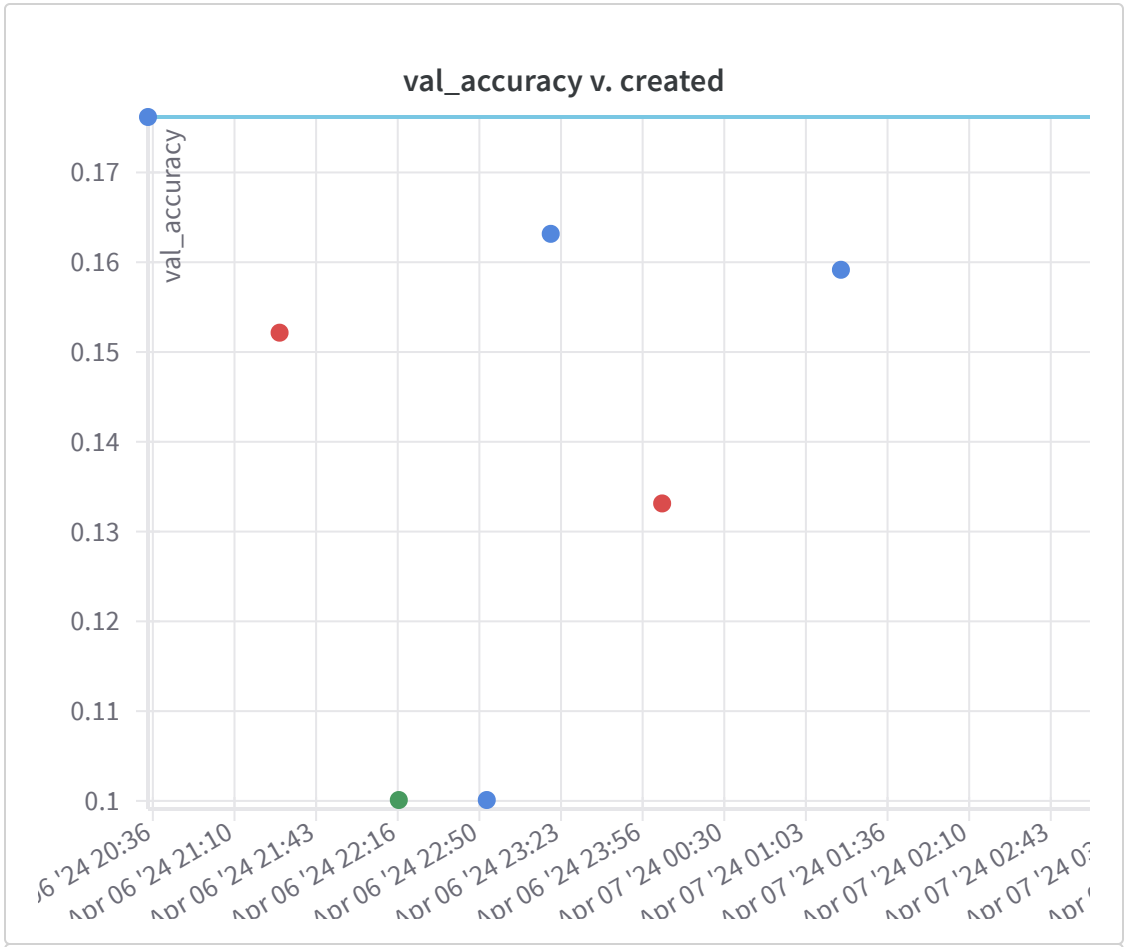
Learning Rate: [1e-3, 1e-4]

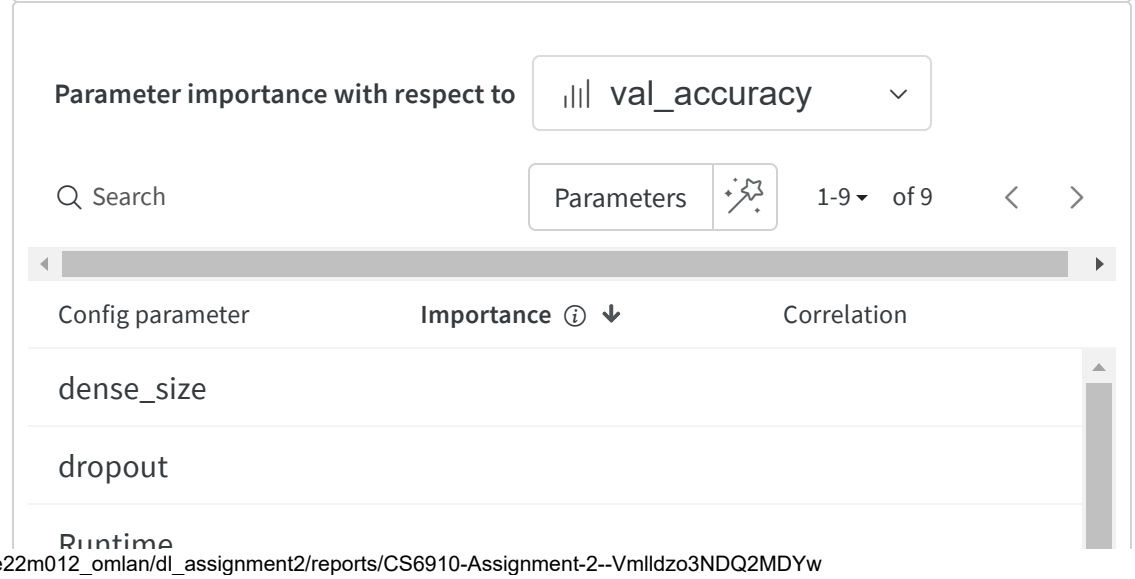
Strategies I followed:

Wandb has three hyperparameter searching strategies: Grid, Random, and Bayes.

1. When I explore hyperparameters, grid search method involves testing all possible combinations, making it quite computationally intensive. In contrast, random search method selects parameter sets randomly. Bayesian Optimization, however, models the function using a Gaussian process and selects parameters to maximize the probability of improvement. This technique requires specifying a metric to guide optimization, typically set as validation accuracy.

2. Initially, I conducted hyperparameter sweeps using the random search method. After identifying the best hyperparameters, I then utilized these optimized values to conduct additional sweeps using the grid search method.
3. Implementing techniques like Batch Normalization and Dropout proved effective for me in reducing the training time of the model.



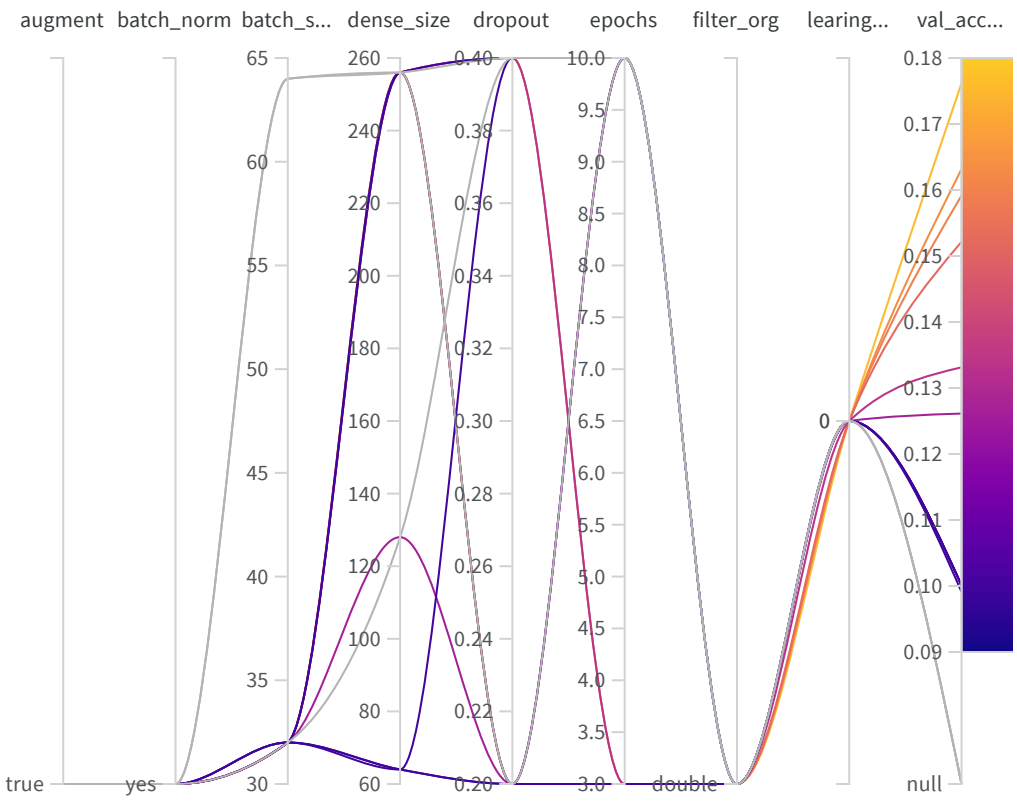


epochs

learing_rate

batch_size_

filter_org.value_dou…



Run set 14

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)

Based on the above plots and accuracies obtained, the following inferences can be made:

- Despite implementing data augmentation techniques, there was no noticeable improvement in model accuracy on either the validation or test datasets.

- Through experimentation, it became evident that ReLU activation outperformed other activation functions, and the Adam optimizer surpassed the SGD optimizer in terms of performance.
- Contrary to expectation, utilizing larger filter sizes in the initial layers did not yield better performance.
- It was observed that there exists a negative correlation between learning rate and model performance; increasing the learning rate led to a decline in model performance.
- The model demonstrated tendencies of overfitting as the number of parameters increased, indicating that increased model complexity could exacerbate overfitting.
- Maintaining uniform filter size (64) throughout the network consistently produced favorable model outcomes.
- A learning rate of 0.0001 consistently yielded high accuracy across experiments.
- Dropout regularization proved effective in curbing overfitting tendencies, as evidenced by the observation that the model tended to overfit without dropout.
- Batch normalization exhibited positive effects with specific hyperparameter configurations, contributing to improved model performance.
- Despite achieving good accuracy initially with specific hyperparameter settings, attempts to replicate the same results with the same configuration later on yielded different outcomes, the reason for which remains unclear.
- Employing a lower learning rate coupled with an increased number of epochs facilitated model convergence, resulting in lower loss and higher accuracy.
- Emphasizing a lower learning rate was found to enhance model performance, while maintaining consistent filter sizes across the network or even increasing them led to favorable model outcomes.

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).

Answer

Using the best set of hyperparameters reported in the plots, the model was evaluated and the highest reported test accuracy was 17.5% for 10 epoch. I tried multiple times in sweep but the accuracy wasn't increasing much.

Original: Mollusca, Predicted: Aves

Original: Reptilia, Predicted: Mammalia

Original: Aves, Predicted: Aves

Original: Amphibia, Predicted: Insecta

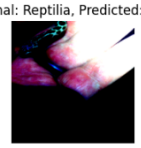
Original: Arachnida, Predicted: Insecta

Original: Mammalia, Predicted: Mammalia

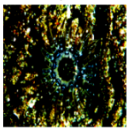
Original: Reptilia, Predicted: Mammalia

Original: Plantae, Predicted: Plantae

Original: Reptilia, Predicted: Fungi



Original: Animalia, Predicted: Plantae



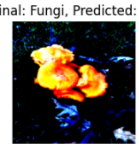
Original: Plantae, Predicted: Plantae

Original: Arachnida, Predicted: Arachnida

Original: Mollusca, Predicted: Aves

Original: Aves, Predicted: Aves

Original: Fungi, Predicted: Fungi

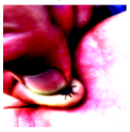


Original: Aves, Predicted: Aves

Original: Aves, Predicted: Insecta

Original: Plantae, Predicted: Plantae

Original: Arachnida, Predicted: Amphibia



Original: Mollusca, Predicted: Reptilia

Original: Animalia, Predicted: Fungi

Original: Plantae, Predicted: Aves

Original: Insecta, Predicted: Plantae

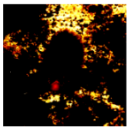
Original: Insecta, Predicted: Insecta

Original: Arachnida, Predicted: Insecta

Original: Amphibia, Predicted: Plantae

Original: Aves, Predicted: Plantae

Original: Mollusca, Predicted: Mollusca



Original: Amphibia, Predicted: Mammalia

Original: Animalia, Predicted: Mollusca

Question 5 (10 Marks)

Paste a link to your github code for Part A

https://github.com/amlananonto/CS6910_Assignment2/blob/main/dla2_part1.py

- 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 0 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?
- 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?

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

answer :

(a). Solution: To tackle this problem, I adjusted the image sizes to 300 x 300 dimensions using the "ImageDataGenerator" tool.

(b). Solution: The naturalist dataset comprises only 10 classes, whereas the ImageNet dataset contains 1000 nodes in its final layer. To resolve this discrepancy, I employed two approaches:

1. I disregarded the last layer of the ImageNet data by utilizing the "include_top=False" parameter in the pre-trained model. This involved removing the final layer from the base model and appending a dense layer with a size of 10 and a softmax activation function.
2. Similarly, I excluded the last layer of the ImageNet data using the "include_top=False" parameter in the pre-trained model. Subsequently, I removed the final layer from the base model and appended a dense layer with a variable size, followed by another dense layer with a size of 10 and a softmax activation function.

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).

Answer :

I employed the following techniques:

1. Freezing all layers within the base model. This involved setting the "trainable" attribute to False for all layers.
2. Freezing all layers except the final layer in the base model, denoted by "freez_before=1".
3. Freezing all layers except for the last k layers within the base model. This process entailed unfreezing the last k layers while keeping the rest frozen. The parameter "freez_before=k" determined the number of layers to freeze before unfreezing the last k layers.

To streamline this process, I utilized the concept of freezing before the last X layers, where the hyperparameters for "freeze_before" were set as [0, 50, 100]. This optimization led to a reduction in the duration of model training.

To explore various combinations and determine the optimal model, I leveraged WandB's sweep feature, incorporating the aforementioned hyperparameters.

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.

answer :

Here are the conclusions drawn from the analysis:

- Utilizing a pre-trained model accelerates convergence compared to training a model from the beginning. In this scenario, the optimal number of epochs would have been 10, although in some cases, 5 epochs produced satisfactory accuracies (>=80).
- Employing a large pre-trained network outperforms training a smaller network from scratch, as it eliminates the need to train all the layers.
- InceptionResNetV2 exhibits superior performance compared to other pre-trained models, achieving the highest accuracy. Conversely, ResNet50 pre-trained model achieved the lowest accuracy.
- Selecting smaller batch sizes such as 32 and 64 helps conserve RAM resources in Google Colab.
- Dropout is employed as a regularization technique to mitigate overfitting.
- The Adam optimizer outperforms other optimizers like RMSProp.
- Freezing the base model until just before the last 100 layers is determined to be the most effective fine-tuning approach among the specified values. This strategy yields improved accuracies by allowing the base model to adapt to the features of the new dataset.

Question 4 (10 Marks)

Paste a link to your GitHub code for Part B

https://github.com/amlananonto/CS6910_Assignment2/blob/main/DL_a2_Part2.py

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

Self Declaration

I, Omlan Jyoti Mondal (Roll no: GE22M012), 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.

Created with ❤️ on Weights & Biases.

https://wandb.ai/ge22m012_omlan/dl_assignment2/reports/CS6910-Assignment-2--Vmlldzo3NDQ2MDYw