DA6401 Assignment 2

Student Details: Name: Siddhant Baranwal Roll No: DA24M021

WANDB Link:
https://api.wandb.ai/links/da24m021-indian-institute-of-technology-madras/ypk2orzf

Github Link:
https://github.com/Siddhant-DA24M021/da6401_assignment2.git

Submission Instructions : kindly make a PDF of your wandb.ai report, attach this page (with your details and links) as the front page to that, and then submit it.

Important Instructions:

- Students **must follow the updated submission format strictly**. Non-compliance will result in penalties.
- Any plagiarism will be reported and heavily penalized.
- You will lose marks if we can't access your Github repo and W&B links.
- Submitting non-modular, unstructured, or unreadable code will cause penalties.

DA24M021 - DA6401 - Assignment 2

This report is submitted by Siddhant Baranwal (DA24M021) in partial fulfillment of the assignment 2 of DA6401 course.

Siddhant Baranwal da24m021

Created on April 7 | Last edited on April 17

Github Link: - https://github.com/Siddhant-DA24M021/da6401_assignment2.git

Part A: Training from scratch

Question 1 (5 Marks)

The CNN model code is in the `model.py` file in the `github`.

- The model consists of 5 convolution layers.
- Each convolution layer is followed by an activation and a max-pooling layer.
- The model has been provided with support for batch-norm layer and dropout layer but it has been not activated initially.
- The model has one fully connected layer.
- The fully connected (dense) layer also has a batch-norm layer, dropout layer and an activation layer following it. The batch-norm layer and dropout layer are not activated initially.
- The model has an output layer of 10 neurons (representing number of classes in the iNaturalist dataset)
- The model has been defined according to the PyTorch code thus has the model structure in the `__init__` method and data passing logic in the `forward` method.

- The parameters of the model are: -
  - image_size: A tuple of integers of size 2.
  - in_channels: Integer (Default = 3). Number of channels in the images.
  - num_classes: Integer (Default=10). Number of output classes.
  - num_filters: A list of 5 integers(Default=[64, 64, 64, 64, 64]). Number of filters in each layer
  - kernel_size: A list of 5 integers(Default=[3, 3, 3, 3, 3]) Kernel size in each layer.
  - activation_fn: Activation function of torch.nn class (Default=nn.ReLU)
  - fc_layer_size: Integer (Default=2048). Fully-connected (dense) layer size
  - batchnorm: Boolean(Default=False). To use batchnorm layer or not.
  - dropout: Float(Default=0.0). Dropout ratio,

The code is flexible such that the number of filters, size of filters, and activation function of the convolution layers and dense layers can be changed. The number of neurons in the dense layer is also changeable.

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)

- **Assumptions:-**
- ***For forward propagation only*** *(Backward propagation has derivations and all).*
- ***Kernel size is small compared to Image dimensions (H >>>>> k and W >>>>> k)***

  such that: H-k+1 ≈ H and W-k+1 ≈ W

- No Batch norm and dropout layer being used (default settings).
- Number of filters in each layer (5 convolutional layer): - m
- Each filter size:- k x k

- Number of neurons in dense layer:- n
- Number of channels:- C
- Input Image Dimensions = (H, W)
- Maxpool window size = 2
- Output classes size:- O
- Activation layer computations are not considered.
- Number of operations in conv layer 1:
  - For one filter = H*W*C*k*k multiplications + H*W*C*k*k additions (including bias) = 2*H*W*C*k*k
  - For m filters = 2*H*W*m*C*k*k
- Number of operations in conv layer 2:
  - For one filter = H*W*m*k*k/4 multiplications + H*W*m*k*k/4 additions (including bias) = 2*H*W*m*k*k/4
  - For m filters = 2*H*W*m*m*k*k/4
  - Divided by 4 for maxpool size reduction
- Total number of computations in convolution layers = 2*H*W*m*C*k*k + 2*H*W*m*m*k*k * (1/4 + 1/16 + 1/64 + 1/256)
- Total number of computations in fully connected layer = (m*H*W/1024)*n multiplications + (m*H*W/1024)*n additions = 2*(m*H*W/1024)*n
- Total Number of computations in output layer = n*O multiplications + n*O additions = 2*n*O
- Total computations = 2*H*W*m*C*k*k + 2*H*W*m*m*k*k * (1/4 + 1/16 + 1/64 + 1/256) + 2*(m*H*W/1024)*n + 2*n*O
- Approx. Total computations (Additions and Multiplications) = `2HWmCkk + (85/128)HWmmkk + 2(mHW/1024)n + 2nO`

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)

- **Assumptions:-**
  - No Batch norm and dropout layer being used (default settings).
  - Number of filters in each layer (5 convolutional layer): - m

    - Each filter size:- k x k
    - Number of neurons in dense layer:- n
    - Number of channels:- C
    - Input Image Dimensions = (H, W)
    - Maxpool window size = 2
    - Output classes size:- O
- Number of parameters in each filter of first layer = C*k*k + 1
- Number of parameters in first convolution layer = m*(C*k*k + 1)
- Every layer has m filters, so after first layer, number of parameters in each filter= m*k*k + 1
- Total number of parameters in each convolution layer after first layer = m* (m*k*k + 1)
- Total number of parameters in all 5 convolution layer = m*(C*k*k + 1) + 4*m* (m*k*k + 1)
- Size of image after 5 convolution layer and maxpool layer = ((H-31k+31)/32 , (W-31k+31)/32)
- Size of flattened layer = m * (H-31k+31) * (W-31k+31) / 1024
- Number of parameters in dense layer = n * (m * (H-31k+31) * (W-31k+31) / 1024) + n
- Number of parameters in output layer = n*O + O
- Total number of parameters in the network = `m*(C*k*k+1) + 4*m*(m*k*k+1) + n*(m*(H-31*k+31)*(W-31*k+31)//1024)+ n + n*O +O`
- Assuming default parameters and image of size (224, 224), num of parameters = **3530186**

Question 2 (15 Marks)

Model is trained on iNaturalist dataset.

Hyperparameter sweep parameters: -

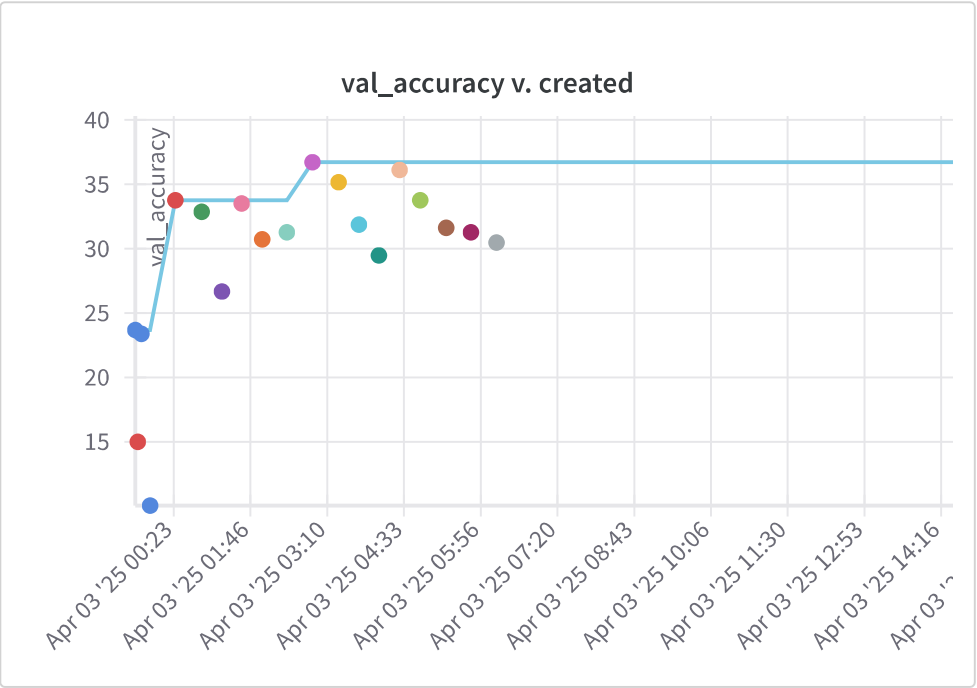
- data_augment: [True, False]
- batch_norm: [True, False]
- dropout: [0.0, 0.2, 0.3, 0.4]
- learning_rate: [0.01, 0.001, 0.0005, 0.0001]

- activation: [relu, leaky_relu, parametric_relu, gelu, silu, mish]
- num_filters: [equal16, equal32, equal64, doubling16, doubling32, halving256]
- kernel_size: [constant3, constant5, constant7, decreasing, increasing]
- fc_layer_size: [2048, 1024, 512]
- batch_size: [8, 16, 32]

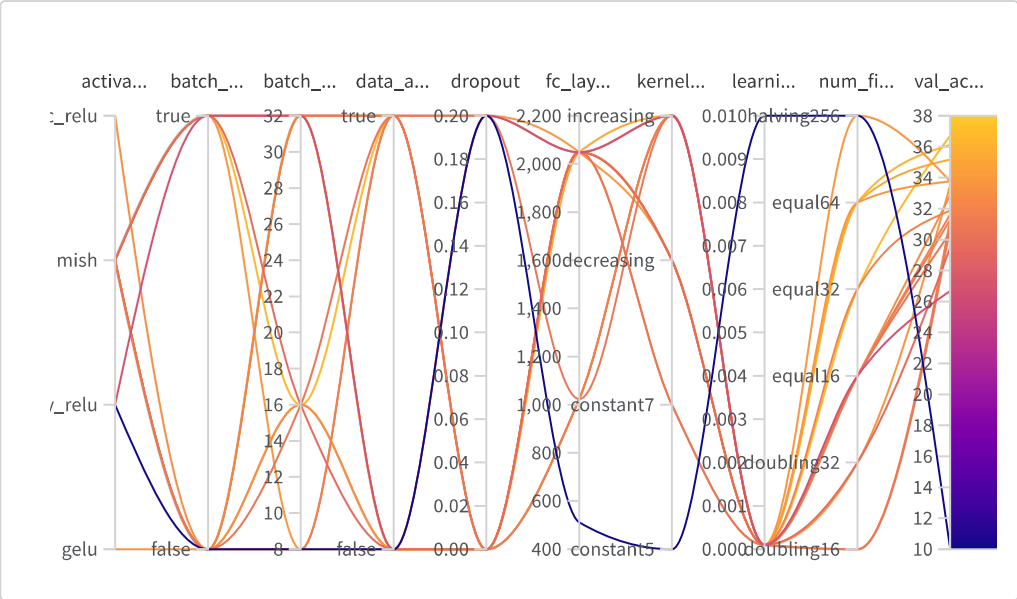
I tried the **"bayes"** sweep method in wandb which uses Bayesian Optimization to effectively search in search hyperparameter space by modeling performance and choosing the most promising configs. It's more sample-efficient than grid or random search.

Based on my sweeps I got the following plots which are automatically generated by wandb:

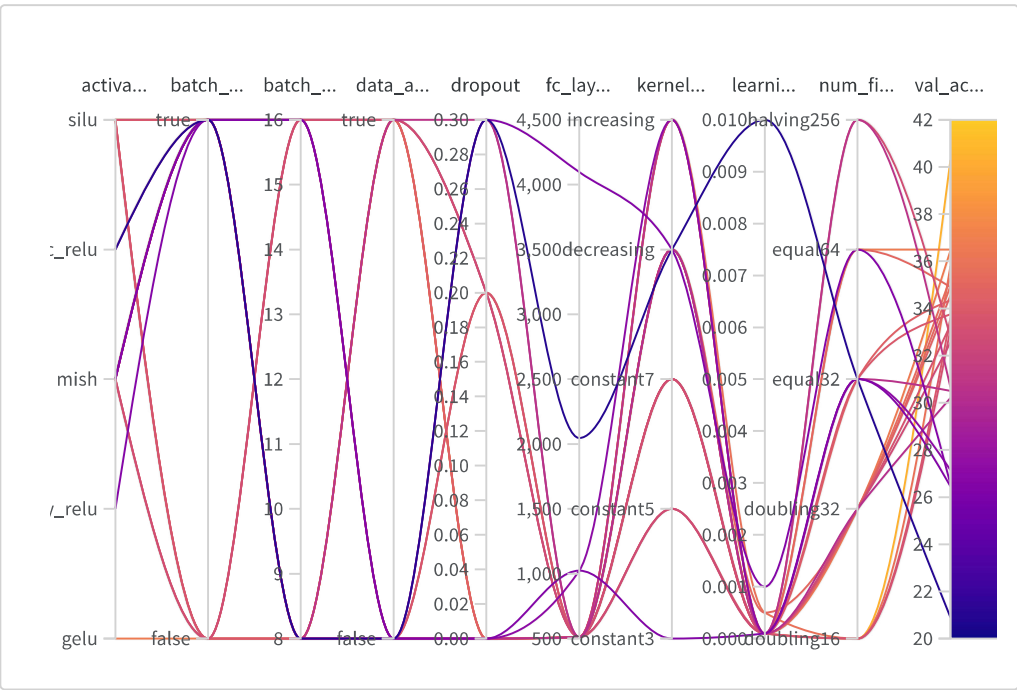
- validation accuracy v/s created plot



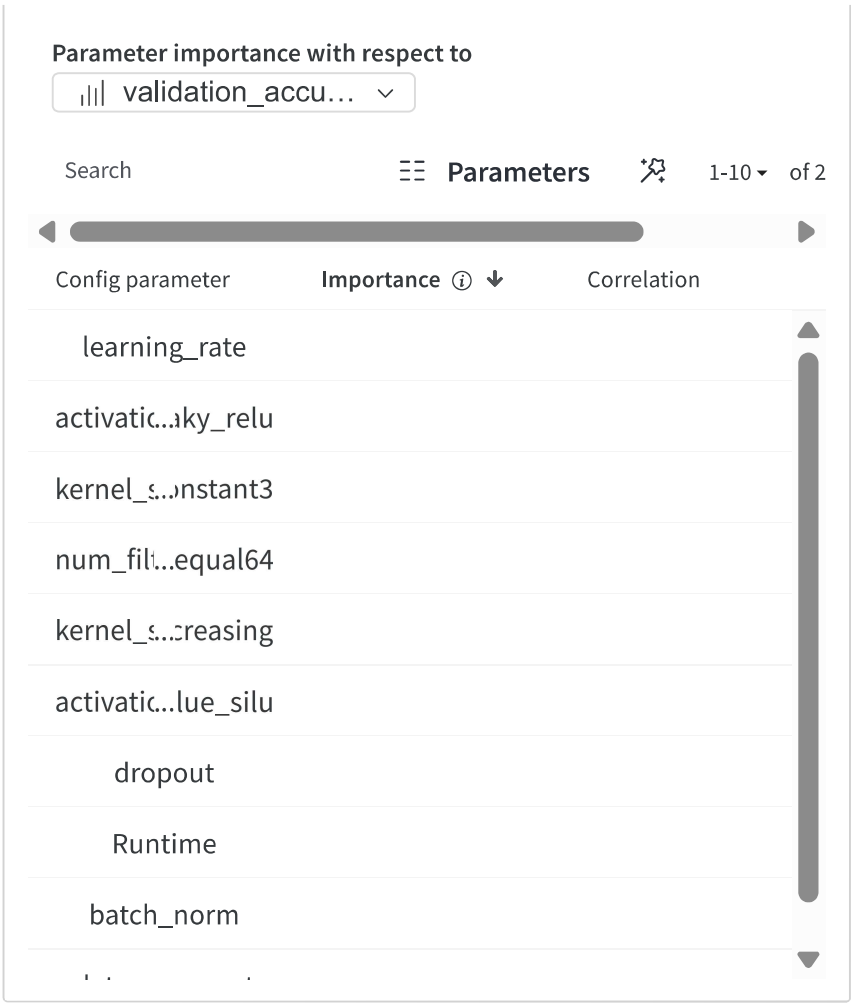
- parallel co-ordinates plot (Sweep-1) (16 runs)



- parallel co-ordinates plot (Sweep-2) (22 runs)



- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)



Question 3 (15 Marks)

Some observations:

- High batch_size results in lower loss as the gradient direction is closer to the true gradient when compared to lower batch_size.
- Data augmentation improved learning by incorporating some noise (works as regularization).
- Batch Normalization also helped in learning features effectively.
- Lower learning rate results in lower loss value and makes the learning more stable.
- Increasing number of filters in the later convolution layers also results in lower loss value.

- Decreasing number of filters in the later convolution layers also results in higher loss value.
- Decreasing the kernel sizes in later layers also helped in lowering error and increasing validation accuracy.
- Activation functions silu, gelu and mish performed better than leaky_relu and parametric_relu.
- High value of dropout resulted in lower validation accuracy. Maybe because the model is simple and not able to generalize well.

Question 4 (5 Marks)

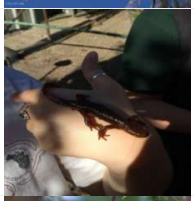
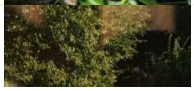
Applying best model on the test data

- Using the best model from your sweep and reporting the accuracy on the test set:-
  - **Test Loss of best model :- 1.756**
  - **Test Accuracy of best model :- 40.50%**
- A 10×3 grid containing sample images from the test data and predictions made by my best model.

```
runs.summary["Prediction Samples"]
```

Filter

	Image	True Label	Predicted Label
1		Mollusca	Reptilia
2		Aves	Mammalia
3		Mammalia	Mammalia

4		Fungi	Amphibia
5		Amphibia	Amphibia
6		Aves	Plantae
7		Fungi	Fungi
8		Plantae	Plantae
9		Plantae	Plantae
		Mammalia	Fungi

1 - 10 of 10 Export as CSV Columns...

Question 5 (10 Marks)

Github Repo Link :- https://github.com/Siddhant-DA24M021/da6401_assignment2/tree/main/partA

Part B : Fine-tuning a pre-trained model

Question 1 (5 Marks)

I researched and felt that EfficientNetV2 might be a good fit for this task.

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

I resized the image to match the dimensions as in the ImageNet dataset used for training EfficientNet_V2_S model.

Image size = (384, 384).

- **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?**

I replaced the last classifier layer of 1000 neurons with a layer that has only 10 neurons.

Question 2 (5 Marks)

I used `EfficientNetV2` for this task and tried finetuning the model using following strategies.

DIFFERENT FINETUNING STRATEGIES: -

1. **Full Finetuning**:- Finetuning is done on all the layers of the model with no layers frozen.

- It has maximum flexibility and high performance potential .
- It can learn task specific features deeply .
- But it involves huge computational costs.
- Train Loss: 0.121, Train Acc: 96.39% (After 10 Epochs)
- Val Loss: 0.502, Val Acc: 87.45% (After 10 Epochs)

2. **Freeze all layers except last classifier layer**:- Only the last classifier layer weights are learnt and all other weights are frozen.

- It can learn very fast and is memory efficient.
- It can avoid overfitting on small datasets.
- But it may not be very efficient in learning images from new domains.
- Train Loss: 0.793, Train Acc: 75.15% (After 10 Epochs)
- Val Loss: 0.667, Val Acc: 81.85% (After 10 Epochs)

3. **Freeze all layers except last k**:- It kind on balances between full finetuning and freezing all layers except last.

- It is more efficient and avoids huge computation cost of full finetuning .
- Is more robust in new domains than just training the last layer.
- Train Loss: 0.107, Train Acc: 96.65% (After 10 Epochs and with k=5)
- Val Loss: 0.476, Val Acc: 88.20% (After 10 Epochs and with k=5)

4. **Gradual Unfreezing of layers**:- Gradual unfreezing of layers helps in stabilizing the model and avoids rapid forgetting of the already learned features.

- It starts with last few layers trainable then with epochs unfreeze more layers.
- It helps the model in retaining the general knowledge while slowly learning on the new task.
- Train Loss: 0.146, Train Acc: 95.46% (After 10 Epochs and unfreezing one new layer after every 2 epoch)
- Val Loss: 0.413, Val Acc: 89.55% (After 10 Epochs and unfreezing one new layer after every 2 epoch)

Question 3 (10 Marks)

The strategy which gave the highest validation accuracy is **Gradual Unfreezing of layers.** So, for finetuning I trained the model on full train dataset with this strategy and evaluated on the test dataset.

After 10 Epochs: -

Train Loss: 0.122, Train Acc: 96.08%

Test Loss: 0.401, Test Acc: 89.55%

On comparing with the model I built: -

- Finetuning a pretrained model performed extremely well on train as well as test dataset.
- The model loaded is very big compared to my 5 layer CNN model.
- The EfficientNetV2 model has weights in the layers which are pretrained to identify different features and hence took a loss less effort to provide extremely good results.

Question 4 (10 Marks)

Github Repo Link :- https://github.com/Siddhant-DA24M021/da6401_assignment2/tree/main/partB

Self Declaration

I, Siddhant Baranwal (DA24M021), swear on my honor that I have written the code and the report by myself and have not copied it from the internet or other students.