Share Comment Star

DA6401 Introduction to Deep Learning - Assignment 2

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

Nandhakishore C S

Created on April 19 | Last edited on April 19

Library Used: PyTorch

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)

A small CNN model consisting of 5 convolution layers is built. Each convolution layer is be followed by an activation function and a max-pooling layer. The activation function, the structure of convolutional layer (kernel size, padding size and stride) are customisable. The max pooling layer is setup with a kernal size of 2×2 and stride 2.

After 5 such **conv-activation-maxpool** blocks, one dense layer followed by the output layer containing 10 neurons (1 for each of the 10 classes) is added. The number of neurons on the dense layer can be fixed by the user. The iNaturalist dataset dataset has images of different sizes (predominantly $800 \times 600 \times 3$ and are non - square in nature). For the model to handle all the images, by keeping the compute power of the machine in mind, the images are re-sized to a shape of $224 \times 224 \times 3$. This is achieved through torchvision's transforms module.

```
train_transform = transforms.Compose([
    transforms.Resize((input_size, input_size)),
    transforms.ToTensor(),
    mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]
])
```

The code is flexible such that the number of filters, size of filters, activation function of the convolution layers and dense layers, number of neurons in the dense layer can be changed. Refer the README file on the GitHub repository to know more.

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)

- In the given model configuration, a **Convolution-Activation-MaxPool** constitutes a block (denoted as (B)).
- Every layer has a kernel of size $k \times k$. After every block of operations, the image is convoluted to a different size. Let $w \times w \times k$ be the input given to one such block (B).
- For m filters, the number of computations done for one layer is given as: $m \times 2w^2k^2d$ (to account for additions after doing linear multiplications with kernel)
- For activation function (ReLU or similar type), the number of computations done (element wise operations on a vector): w^2
- The model implemented has a MaxPooling layer with a pool size of 2×2. Thus the number of computations per maxpool layer is calculated as: $4 \times \frac{w}{2} \times \frac{w}{2} = w^2$. For m such filter, mw^2 computations are done.
- Thus, for one block (B) total number of computations $2(mw^2k^2d + w^2m)$ at $(w = N, d = 3) = O(mk^2)$. This is neglecting the constants and addition terms
- After the input image is passed through the convolution layers, the size shrinks and the depth increases. Thus the number of computations shrinks after every layer. For b such blocks, we get $O(m^2k^2)$ computations.
  - Thus, the total computations for every block $= O(m^2k^2) + O(mk^2)$
- After all the layers, the size of the image reduces to $O(m)$, with $'n'$ neurons on the dense layer, there are $nO(m) + n$ computations (accounting for bias)
- **Total number of computations in the network** $= O(mn) + O(m^2k^2)$

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)

- With the number of computations known, the number of parameters in the model lay in the similar number.
- **Total number of parameters in the model** $= O(mn) + O(m^2k^2)$

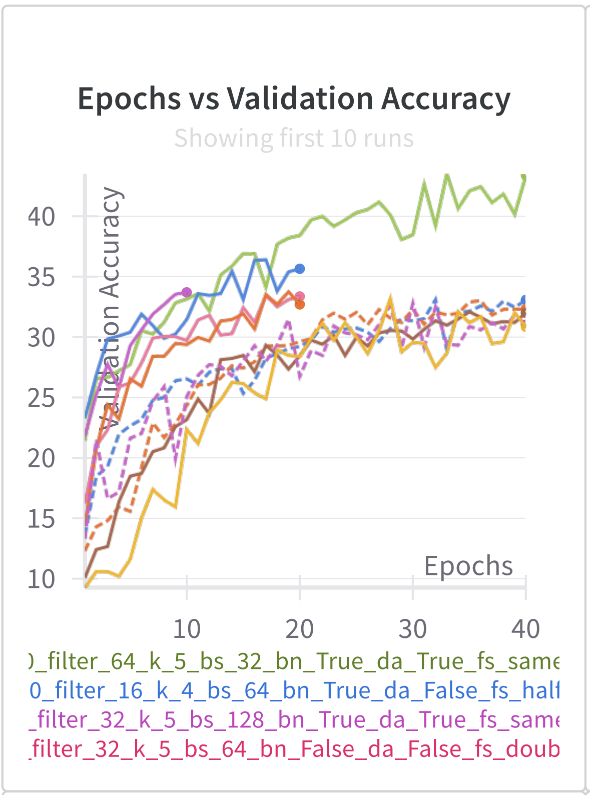
Question 2 (15 Marks)

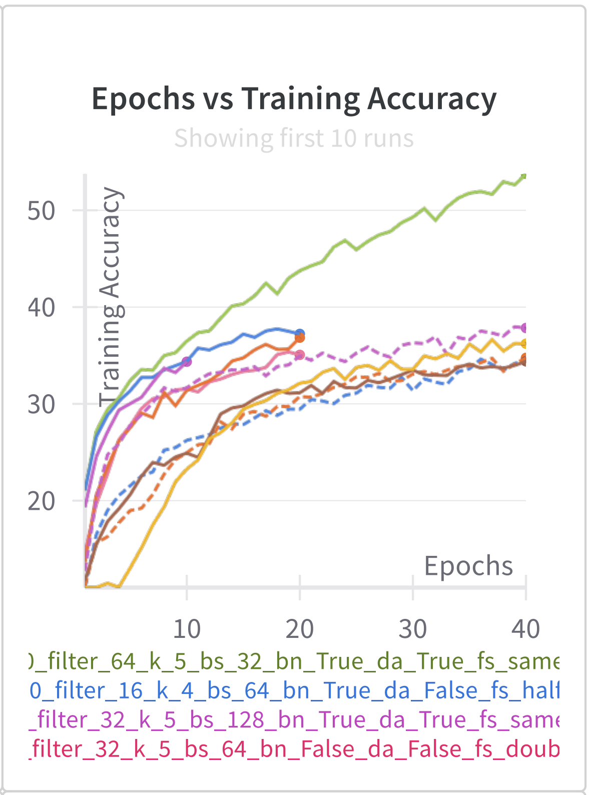
The iNaturalist dataset dataset when downloaded has data in two folders: Train and Val. As per the instructions, the Val folder is taken as the test data (renamed as test for ease of handling). A python script is written is to split 20% of the training data and to save it in a separate directory to use for validation. No test data is used for training and hyper parameter search.

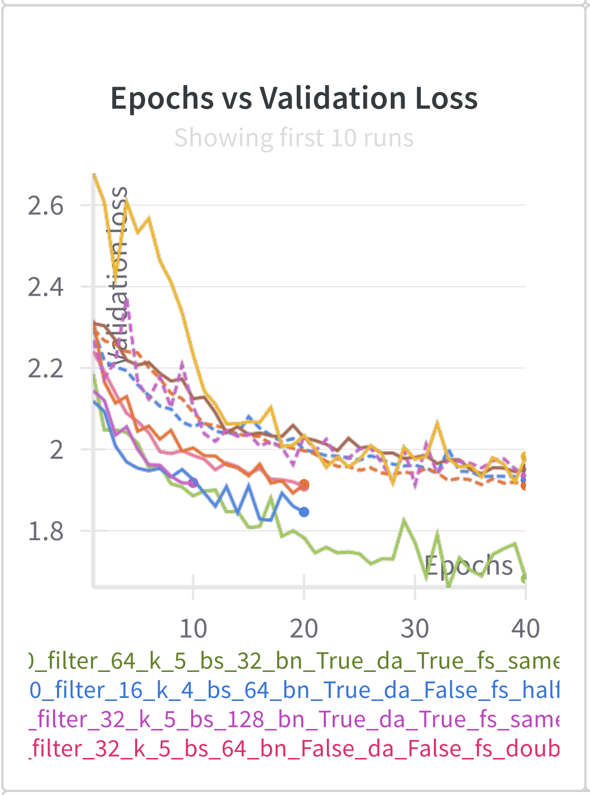
Using the sweep feature in wandb the best hyper parameter configuration is found and here are the hyper parameters explored.

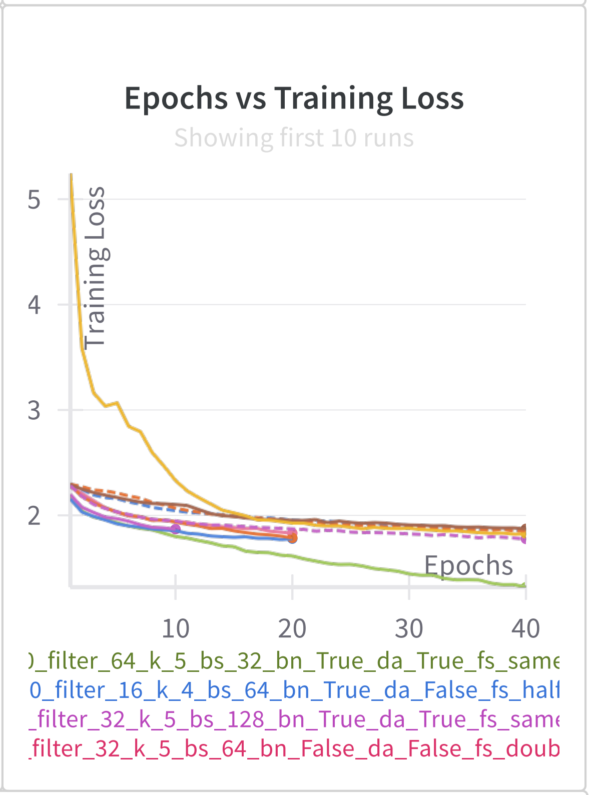
- **number of filters in each layer**: [16, 32, 64]
- **activation function for the conv layers:** ReLU, GELU, SiLU, Mish, ...
- **filter organisation**: same, double, half
- **data augmentation**: Yes, No
- **batch normalisation**: Yes, No
- **dropout**: values from uniform distribution from the interval [0.1, 0.6]
- **learning rate**: values from uniform distribution from the interval [1e-2, 1e-4]
- **epochs**: [20, 40]
- **weight decay**: values from uniform distribution from the interval [1e-3, 1e-5]
- **size of convolution layer kernel:** [3 ,4, 5]
- **bias in convolution layer**: Yes, No
- **padding of convolution layer:** no padding, padding to get output size as input size
- **number of neurons in dense layer:** [32, 64, 128]
- **batch size:** [32, 64, 128]
- **optimiser**: Adam, SGD

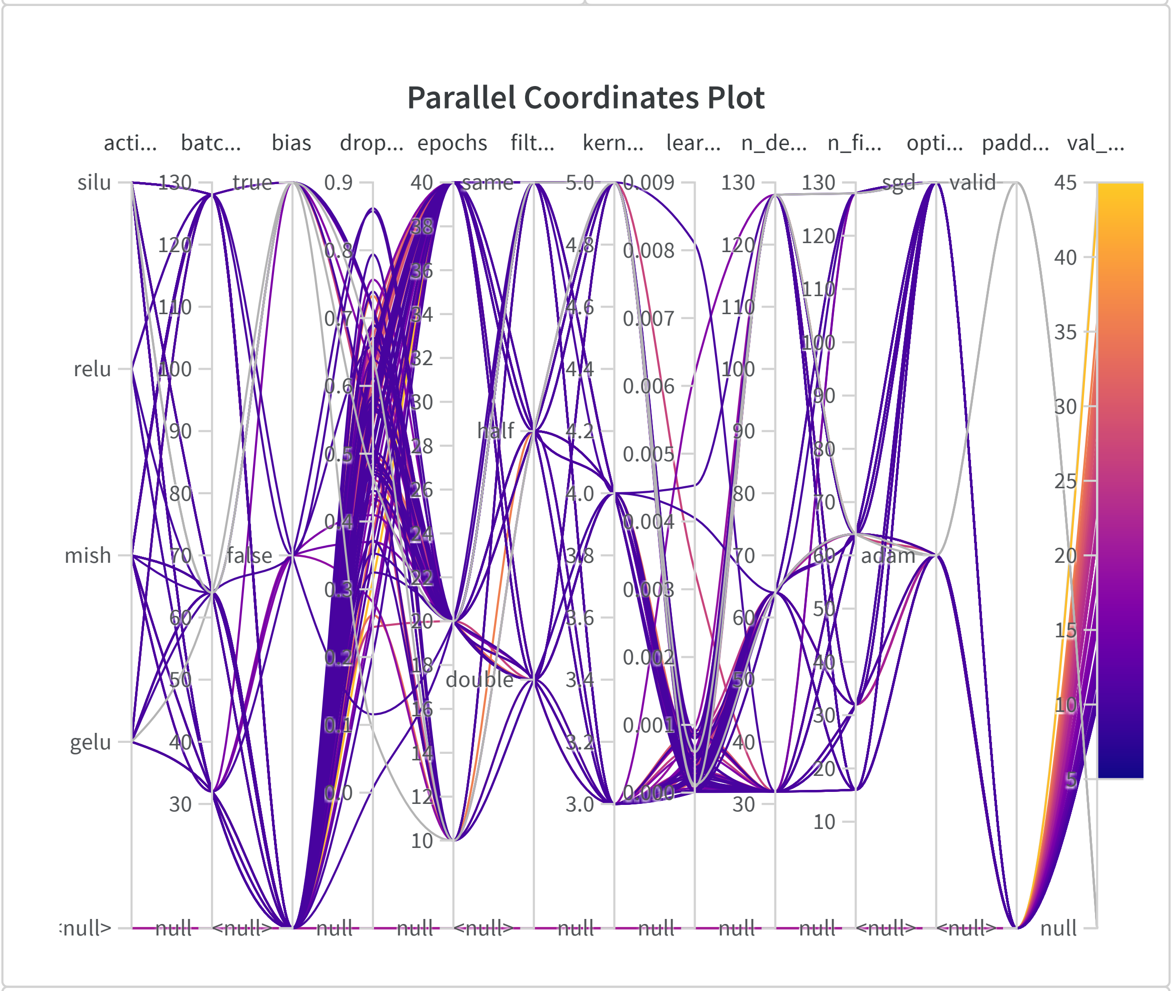
Using wandb agent, various sweeps were conducted and the observations are as follows:


Epochs vs Validation Accuracy
Showing first 10 runs
Validation Accuracy
40
35
30
25
20
15
10
Epochs
10
20
30
40
)_filter_64_k_5_bs_32_bn_True_da_True_fs_same
0_filter_16_k_4_bs_64_bn_True_da_False_fs_half
_filter_32_k_5_bs_128_bn_True_da_True_fs_same
filter_32_k_5_bs_64_bn_False_da_False_fs_doub


Epochs vs Training Accuracy
Showing first 10 runs
Training Accuracy
50
40
30
20
Epochs
10
20
30
40
)_filter_64_k_5_bs_32_bn_True_da_True_fs_same
0_filter_16_k_4_bs_64_bn_True_da_False_fs_half
_filter_32_k_5_bs_128_bn_True_da_True_fs_same
filter_32_k_5_bs_64_bn_False_da_False_fs_doub


Epochs vs Validation Loss
Showing first 10 runs
Validation Loss
2.6
2.4
2.2
2
1.8
Epochs
10
20
30
40
)_filter_64_k_5_bs_32_bn_True_da_True_fs_same
0_filter_16_k_4_bs_64_bn_True_da_False_fs_half
_filter_32_k_5_bs_128_bn_True_da_True_fs_same
filter_32_k_5_bs_64_bn_False_da_False_fs_doub

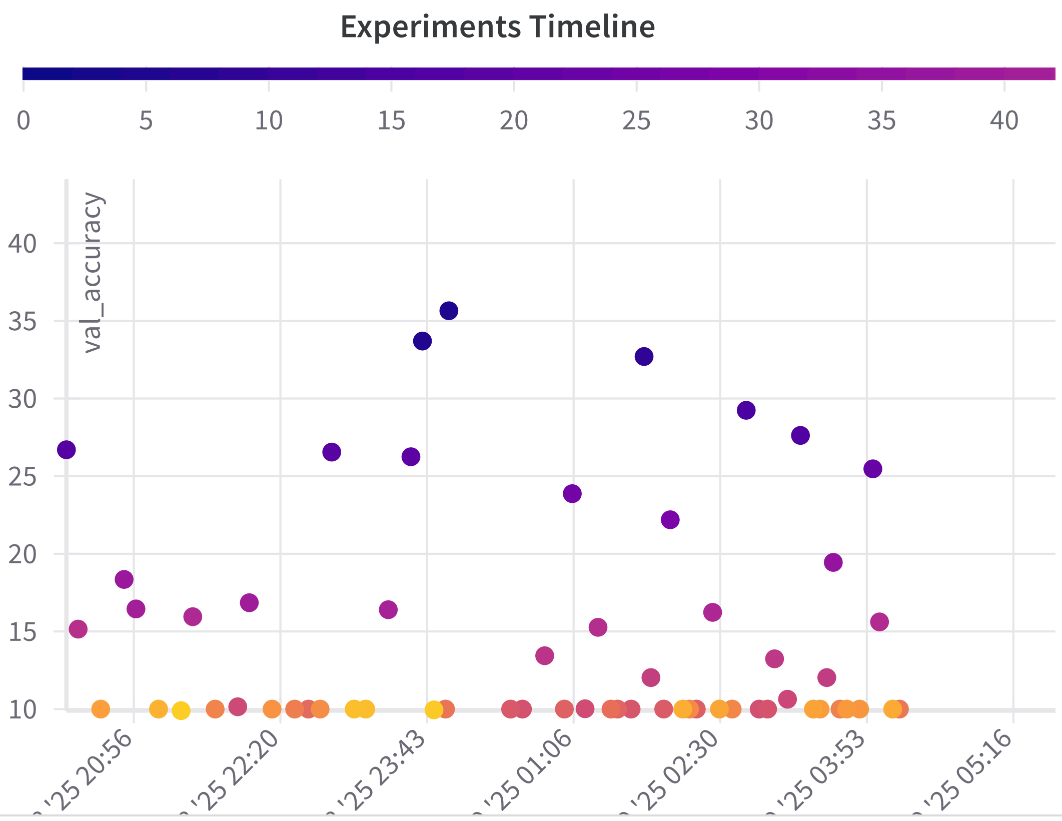

Epochs vs Training Loss
Showing first 10 runs
Training Loss
5
4
3
2
Epochs
10
20
30
40
)_filter_64_k_5_bs_32_bn_True_da_True_fs_same
0_filter_16_k_4_bs_64_bn_True_da_False_fs_half
_filter_32_k_5_bs_128_bn_True_da_True_fs_same
filter_32_k_5_bs_64_bn_False_da_False_fs_doub


Parallel Coordinates Plot
acti...
batc...
bias
drop...
epochs
filt...
kern...
lear...
n_de...
n_fi...
opti...
padd...
val_...
silu
relu
mish
gelu
<null>
true
false
same
half
double
sgd
adam
valid
null

Parameter importance with respect to val_accuracy

Search Parameters 1-10 of 14

Config parameter	Importance	Correlation
...		
dropout_rate		
n_filters		
n_dense_neurons		



```
model = ImageClassifier(
    input_size = (224, 224),
    n_layers = 10,
    in_channels = 3,
    n_classes = n_classes,
    kernel_size = 5,
    n_filters = 64,
    filter_strategy = 'same',
    padding_mode = 'same',
    n_epochs = 80,
    n_neurons = 128,
    activation = 'gelu',
    optimiser = 'adam',
    criterion = 'cross_entropy',
    learning_rate = 0.0001,
    weight_decay = 0.001,
    batch_norm = True,
    drop_out = 0.3,
    use_wandb = True,
    name = 'DA24M011',
    validation = True
).to(device)
```

The above snippet shows model initialisation with different hyper-parameters.

For doing strategic sweeps, the following steps were done:

- 'bayes' mode was enabled to search for better hyper-parameters in wandb
- Each sweep was limited to 28 to 32 runs and the hyper parameter ranges were narrowed down to look for configurations which gave better perfomance.
- Once a sweep was done, the results were analysed, the hyper parameter range was narrowed down the the process was repeated.

Question 3 (15 Marks)

Based on the above experiments, the following observations are made:

1. The The iNaturalist dataset dataset has 10 classes of various species as images of different sizes. The number of samples per class is not same and the dataset has class imbalance.
2. Based on the skewness in the class distribution, the contribution from the features of the samples in less numbers impacts the training of the model.
3. For the given 5 layered model with CNN + Activation + MaxPool layer, the following hyper parameters where found highly impactful:
   1. **Activation Function** : GELU
   2. **Convolution Kernel Size** : 5×5
   3. **Convolution Padding Type** : 'Same' - Padding in such a way that the input size and output size remains the same.
   4. **Dropout** : Dropout probabilities from the range $[0.3, 0.45]$ gives better validation accuracy.
   5. **Batch Normalisation** : The sweeps without batch normalisation gave better validation accuracy.
   6. **Data Augmentation** : The sweeps with Data Augmentation gave better validation accuracy.
   7. **Weight Decay** : Weight decay in the range $[0.001, 0.005]$ gives better validation accuracy.
   8. **Number of filters per Convolution-Activation-MaxPooling block:** Keeping the number of filters same as the initial number of filters

gives better validation accuracy than doubling or halving the number of filters after each Convolution-Activation-MaxPooling block

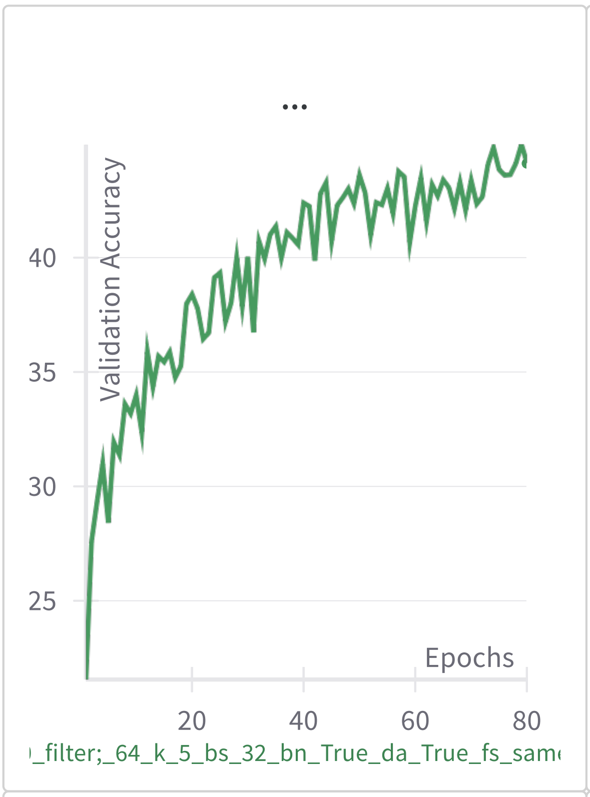
9. **Epochs:** Training the model from scratch with larger number of epochs gives better accuracy (40 epochs).

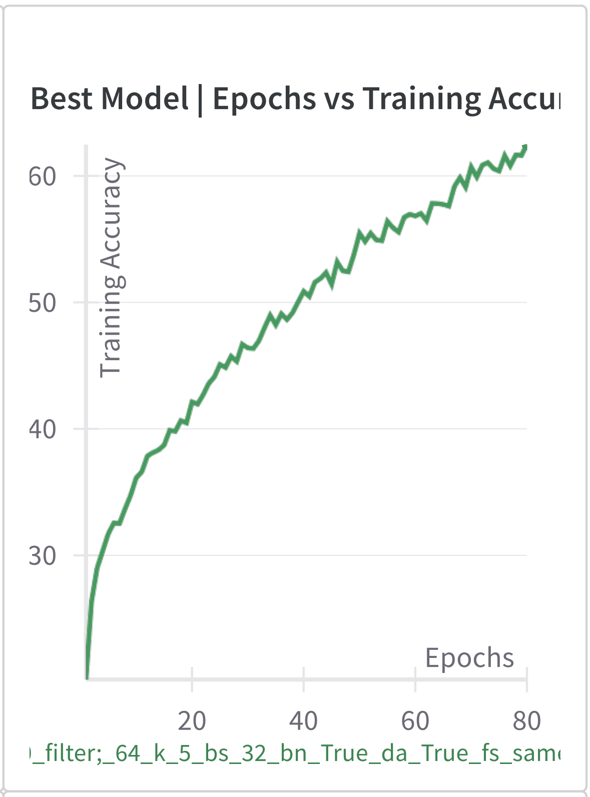
Question 4 (5 Marks)

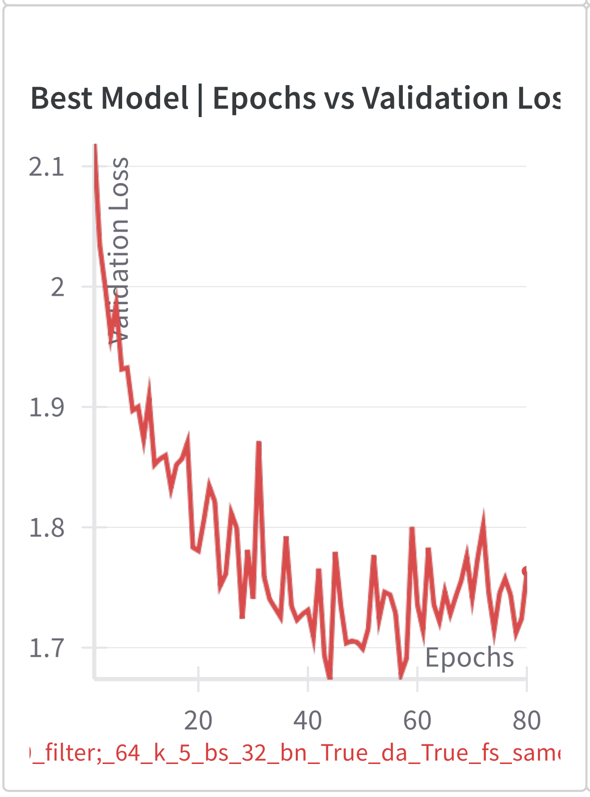
The **Best Model** chosen from the sweeps:

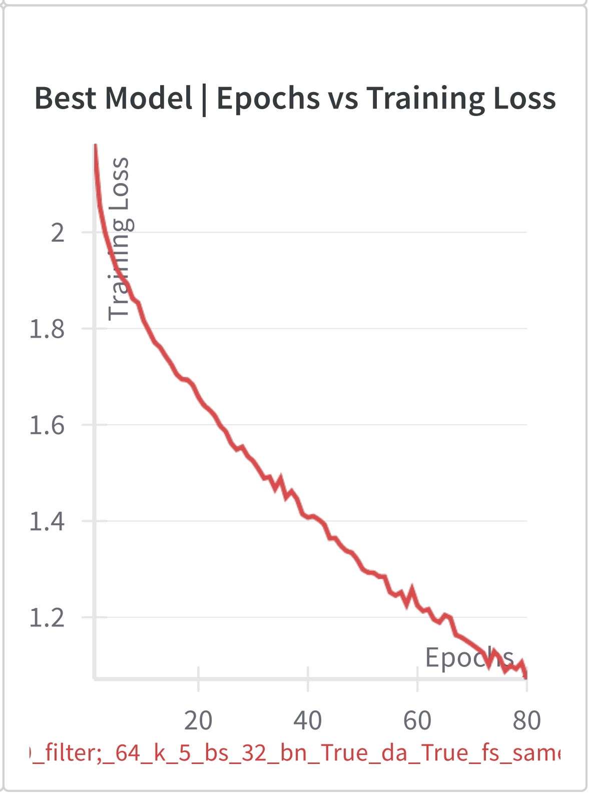
- **Training Accuracy**: 53.76 %
- **Validation Accuracy**: 43.38 %
- **Activation Function**: GELU
- **Learning Rate**: 0.00027
- **Optimiser**: Adam
- **Epochs**: 40
- **Number of Convolution Filters**: 64
- **Convolution Kernel Size**: 5×5
- **Batch Size**: 32
- **Batch Normalisation**: True
- **Data Augmentation**: True
- **Filter Strategy**: Same number of filters every Convolution-Activation-MaxPool block
- **Dropout**: 0.31396
- **Number of Neurons in Dense Layer**: 128
- **Weight Decay**: 0.00133

The best model was trained once again for 80 epochs to see if the validation accuracy increases, but the validation accuracy stalled after 43%



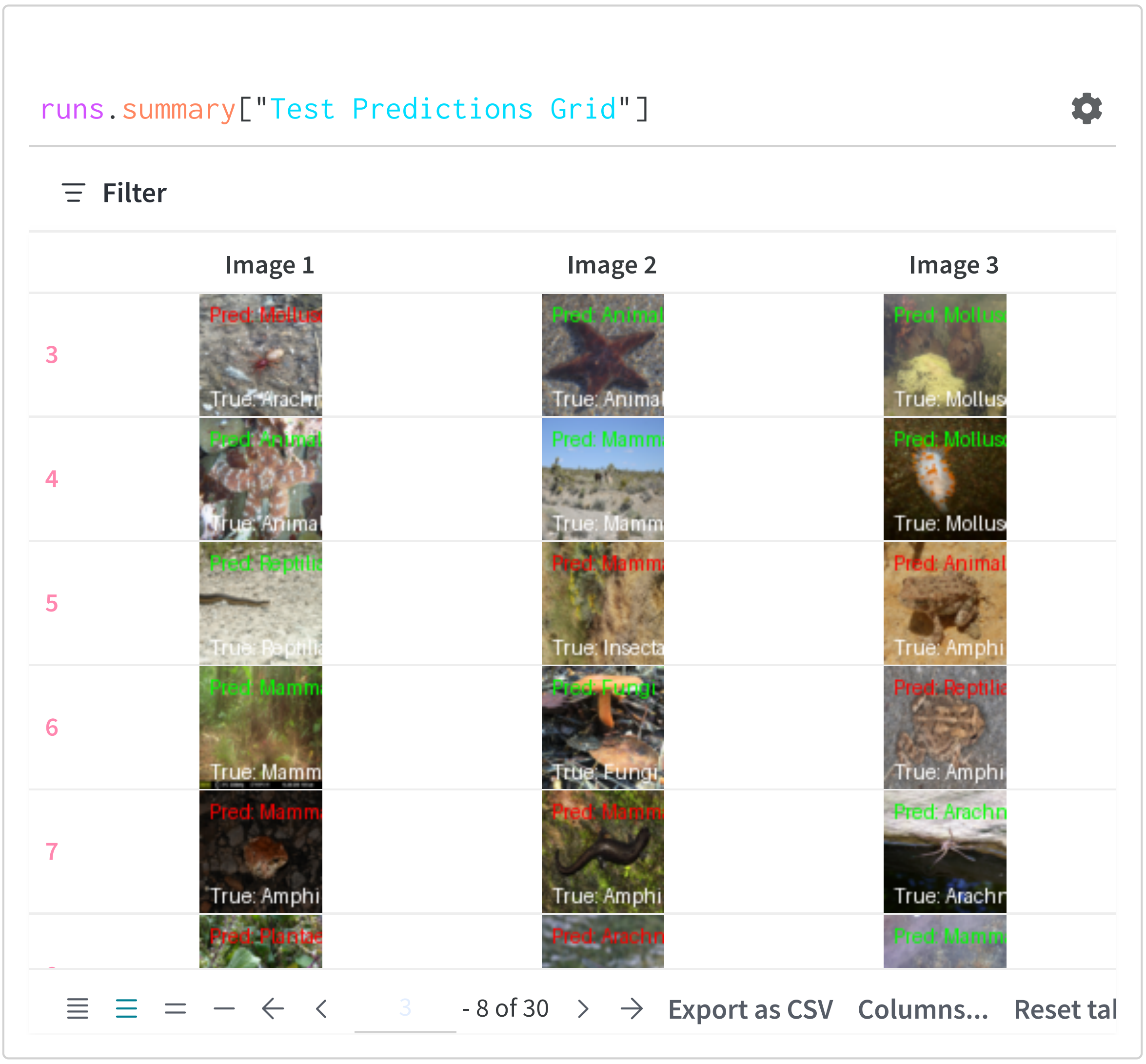






Best Model accuracy on test set: 44.82%

10×3 Image Grid with predictions done by the model:



```
runs.summary["Test Predictions Grid"]
```

Filter

	Image 1	Image 2	Image 3
3	Pred: Mollus True: Arachn	Pred: Animal True: Animal	Pred: Mollus True: Mollus
4	Pred: Animal True: Animal	Pred: Mamm True: Mamm	Pred: Mollus True: Mollus
5	Pred: Reptilia True: Reptilia	Pred: Mamm True: Insecta	Pred: Animal True: Amphi
6	Pred: Mamm True: Mamm	Pred: Fungi True: Fungi	Pred: Reptilia True: Amphi
7	Pred: Mamm True: Amphi	Pred: Mamm True: Amphi	Pred: Arachn True: Arachn
	Pred: Plantae	Pred: Arachn	Pred: Mamm

Question 5 (10 Marks)

GitHub Repository:
https://github.com/nandhakishorecs/da6401_assignment2/tree/main/PartA

Part B : Fine-tuning a pre-trained model

Question 1 (5 Marks)

Model Chosen: ResNet50

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

- Use a PyTorch's transforms based function to resize the dataset to a standard size (224×224) and loading it to train the 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?

- A user defined function is written to check for the last layer's dimension and the flattened layer is connected to a dense layer. The existing dense layer is removed and the new layer with 10 output nodes is attached.

Question 2 (5 Marks) - Different strategies to fine tune a pre-trained model

Three different strategies that you tried (simple bullet points would be fine).

- The CNN + Feature extraction layers are frozen and the weights corresponding to these layers are downloaded from the internet.

- Keeping the CNN layers, adding more fully connected layers with same number of neurons and training the model

- Freeze the first 'k' layers and replace the remaining dense layers and train the newly added dense layer.

Question 3 (10 Marks)

Chosen Strategy: Keep the feature extraction / CNN layer intact and remove the dense layer from the pre-trained model. A new dense layer with 10 outputs is added.

The newly added dense layer is then trained from scratch. This is achieved through a user defined Python function.

```
def set_parameter_requires_grad(model, freeze_layers):
    for name, param in model.named_parameters():
        param.requires_grad = not freeze_layers(name)
def get_model():
    model = models.resnet50(pretrained=True)

    # Replace final layer
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, 10)

    # Freeze layer1 and layer2, train layer3, layer4, and fc
    def freeze_layers(name): return any(x in name for x in ["layer1"
    set_parameter_requires_grad(model, freeze_layers)
    return model.to(device)
```

Question 4 (10 Marks)

GitHub repository:
https://github.com/nandhakishorecs/da6401_assignment2/tree/main/PartB

Self Declaration

I, **Nandhakishore C S** (Roll no: **DA24M011**), 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.