DA6401 - Assignment 2

April 19, 2025

Name : Sakthe Balan G K

Roll No. : ME21B174

github repo link : `https://github.com/sakthe1010/da6401_assignment2`

wandb report link : `https://wandb.ai/sakthebalan1010-iit-madras/ASSIGNMENT%202/reports/ME21B174-DA6401-Assignment-2--VmlldzoxMjEzOTg0NQ`

ME21B174 DA6401 - Assignment 2

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

SAKTHE BALAN

Created on April 5 | 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 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 $k \times k$ 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)

Answer:

Assumptions:

- 5 convolutional layers followed by activation, optional batch normalization, and max pooling (with stride = 2, pooling kernel = 2×2).
- Each convolutional layer i has m_i filters, kernel size $k * k$, padding $k/2$ (to preserve spatial dimensions before pooling)
- The convolutional layers configuration is flexible and given as:

$$conv\_channels = [m_1, m_2, m_3, m_4, m_5]$$

- Input image dimension: $I * I * c$ (square images with c channels, typically $c = 3$ for RGB)
- After these 5 convolutional layers, the spatial dimensions reduce due to max pooling (halved at each layer), giving a final spatial dimension:

$$D = I/2^5 = I/32$$

- Afer convolutions, there is a fully connected (dense) layer with n neurons, folloed by an output layer with 10 neurons (10-class classification)

Generalized formula for total computations:

The total number of computations includes the forward pass multiplications and additions. Each convolution operation (multiply and add) counts as 2 operations.

Convolutional Layers Computations:

Each convolutional layer has:

- Input size: $W_{i-1} * W_{i-1} * m_{i-1}$ (with $W_0 = I, m_0 = c$)
- Output size (before pooling): same spatial dimension due to padding which is$W_{i-1} * W_{i-1} * m_i$

- Total computations per convolutional layer:

$$2 \times (k^2) \times (W_{i-1}^2 * m_{i-1}) \times m_i = 2W_{i-1}^2 m_{i-1} m_i k^2$$

After pooling, dimension reduces by half:

- $W_i = W_{i-1}/2$

Thus the total computations over 5 layers:

$$\text{Conv\_Ops} = 2k^2 \sum_{i=1}^{5} W_{i-1}^2 m_{i-1} m_i$$

Dense Layers Computations:

- Dense layer computations (flattening last feature maps of size $D * D * m_5$):

$$2 \times (D^2 m_5) \times n = 2D^2 m_5 n$$

- Output layer computations:

$$2 \times n \times 10 = 20n$$

Total Computations:

$$\text{Total Computations} = 2k^2 \sum_{i=1}^{5} W_{i-1}^2 m_{i-1} m_i + 2D^2 m_5 n + 20n$$

Generalized formula for total parameters:

Convolutional Layer Parameters:

Each convolutional filter has $k^2 m_{i-1} + 1$parameters (weights and bias). Thus, total parameters for convolutional layers is

$$\text{Conv\_Params} = \sum_{i=1}^{5} m_i (k^2 m_{i-1} + 1)$$

Dense Layer Parameters:

- Dense layer:

$$(D^2 m_5 + 1)n$$

- Output layer:

$$(n+1) \times 10$$

Total Parameters:

$$\text{Total Parameters} = \sum_{i=1}^{5} m_i(k^2 m_{i-1} + 1) + (D^2 m_5 + 1)n + 10(n+1)$$

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

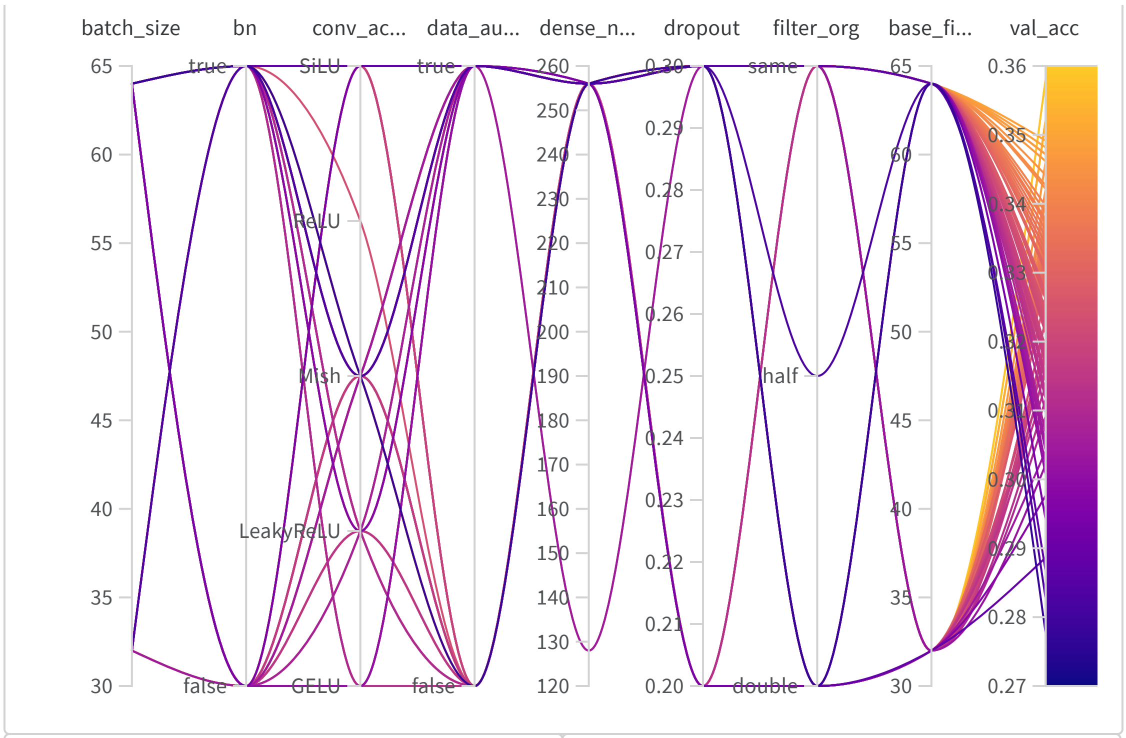
- number of filters in each layer : 32, 64, ...
- activation function for the conv layers: ReLU, GELU, SiLU, Mish, ...
- filter organisation: same number of filters in all layers, doubling in each subsequent layer, halving in each subsequent layer, etc
- data augmentation: Yes, No
- batch normalisation: Yes, No
- dropout: 0.2, 0.3 (BTW, where will you add dropout? You should read up a bit on this)

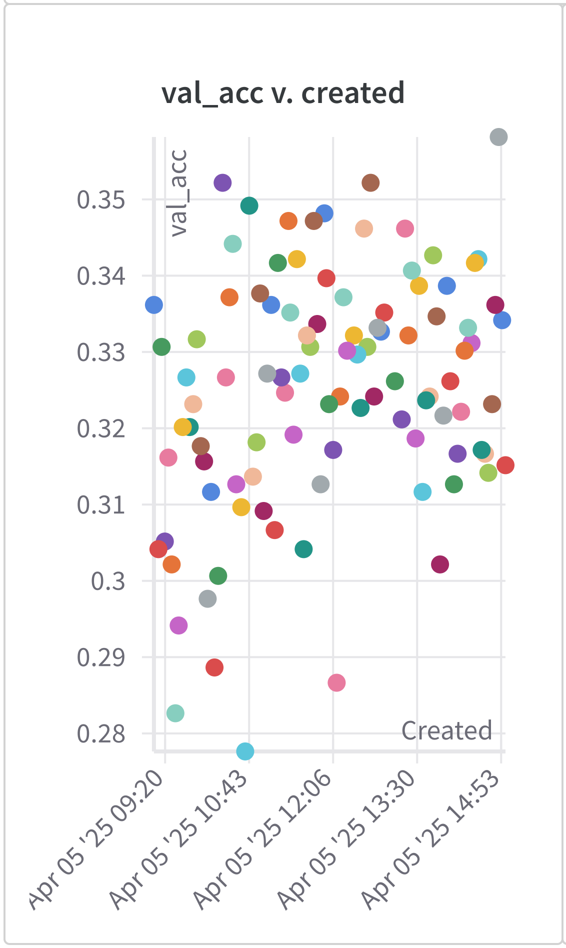
Based on your sweep please paste the following plots which are automatically generated by wandb:

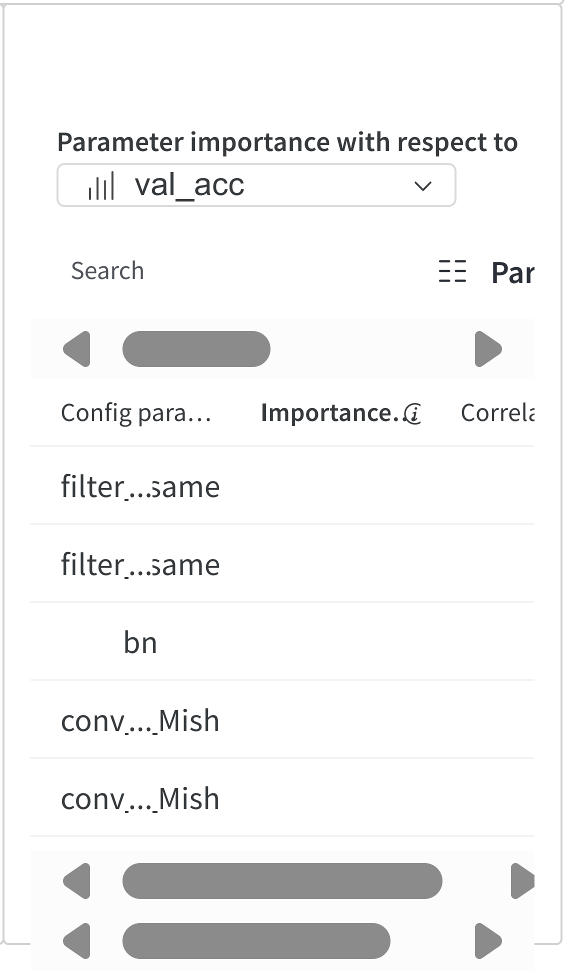
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

Answer:

Parallel coordinate plot






Parameter importance with respect to

val_acc

Search Par

Config para...	Importance	Correla
filter_...same		
filter_...same		
bn		
conv_...Mish		
conv_...Mish		

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.

List of hyperparameters and their values used for sweeps (best values in bold):

- convolution activation function: ["ReLU", "GELU", "SiLU", **"Mish"**, "LeakyReLU"]
- filter organization: [**"same"**, "double", "half"]
- base filter: [**32**, 64]
- dense neurons: [128, **256**]
- bn (batch normalization): [**true,** false]
- dropout: [0.2, **0.3**]
- data augmentation: [true, **false**]
- batch size: [32, **64**]

Strategies used:

- **Wandb Bayesian Sweep:** Prioritized runs based on promising hyperparameter configurations, significantly reducing the total number of experiments needed. (For example, the sweep automatically eliminated using 'half' configuration in filter organization after a single run with that setup. This is because that configuration produced low accuracy.)
- **Stratified validation split (20% from training data):** Ensured equal representation of classes.
- **Dynamic naming**: Ensured clarity and ease of comparison among experiments.

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:

1.Mish Activation Function Outperforms Others

- **Finding:** The network consistently achieved higher validation accuracy with Mish compared to ReLU, GELU, or SiLU.
- **Reason:** Mish is a smooth and non-monotonic activation that can preserve small gradients better than ReLU-like functions, aiding in deeper feature learning. It potentially helps the model converge to better minima, leading to improved performance.

2.“Same” Filter Organization Yields Better Results

- **Finding:** Keeping 32 filters in each conv layer (“same”) produced notably better accuracy than either doubling the filters each layer (“double”) or halving them (“half”).
- **Reason:**
  - “Double” sometimes overparameterized deeper layers, making it harder for the model to generalize well.
  - “Half” limited representational capacity, making it harder to learn diverse features.
  - “Same” struck a sweet spot: sufficient capacity without overwhelming parameter growth, thus enabling stable training and good generalization.

3.Batch Normalization + Dropout = Stability & Regularization

- **Finding:** A combination of batch normalization and 0.3 dropout consistently improved validation accuracy.
- **Reason:**
  - Batch Normalization helps mitigate internal covariate shift and stabilizes gradients.
  - Dropout provides additional regularization, preventing co-adaptation of neurons.

    - The synergy of BN + 0.3 dropout improved both convergence speed and final accuracy.

4.Data Augmentation Off Performed Better Than Expected

- **Finding:** Contrary to many scenarios where augmentation is beneficial, disabling data augmentation yielded slightly higher accuracy in this setup.
- **Possible Explanations:**
    - The nature of the iNaturalist images (lighting, perspective) may already be diverse enough, thus further augmentation sometimes introduces too much noise.
    - Mish + BN + Dropout already added sufficient regularization, making additional augmentation less helpful or even detrimental.
    - Dataset size may be adequate to learn robust features without heavy augmentation.

5.Larger Dense Layer Improved Classification

- **Finding:** Increasing the dense layer size from 128 to 256 consistently boosted performance.
- **Reason:** The additional capacity in the dense layer can capture more complex combinations of high-level features extracted by the convolutional backbone, improving final classification accuracy.

6.Batch Size of 64 Provides Balanced Training Dynamics

- **Finding:** A batch size of 64 struck a good balance between efficient GPU utilization and gradient stability.
- **Reason:**
    - Smaller batches can introduce unstable gradient estimates, while very large batches can hurt generalization.
    - 64 is a sweet spot, especially when combined with batch normalization, ensuring consistent statistics within each mini-batch.

Question 4 (5 Marks)

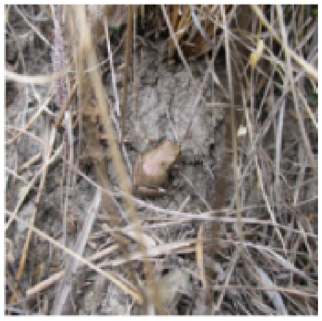
You will now apply your best model on the test data (You shouldn't have used test data so far. All the above experiments should have been done using train and validation data only).

- 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.

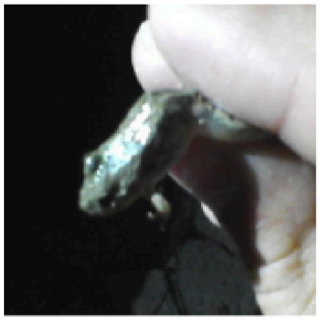
Answer:

1. The accuracy found on the test set used from the best model from the sweep turned out to be **0.341**
2. Shown below is the required 10*3 grid containing sample test images, and their predictions made by the best model. A separate script was written (refer a2_part1_q4.py for the code) to find the accuracy of the best model and produce the plot below.

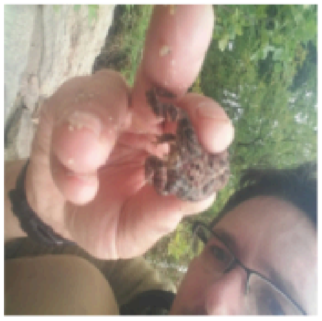
True: Amphibia
Pred: Mammalia



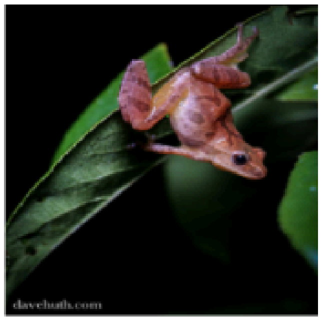
True: Amphibia
Pred: Fungi



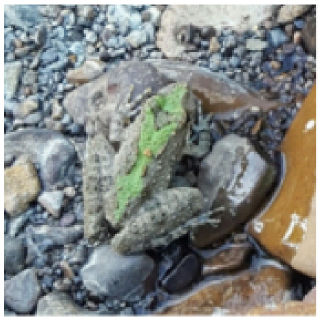
True: Amphibia
Pred: Fungi



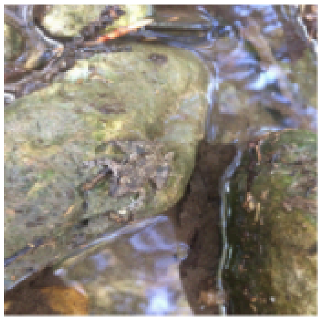
True: Amphibia
Pred: Amphibia



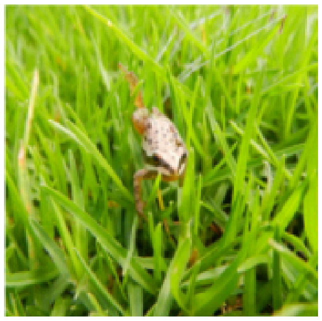
True: Amphibia
Pred: Amphibia



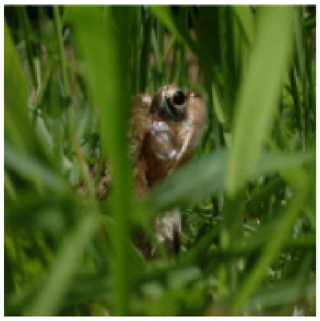
True: Amphibia
Pred: Amphibia



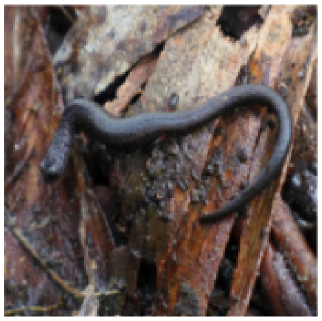
True: Amphibia
Pred: Aves



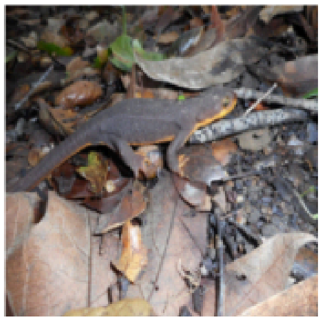
True: Amphibia
Pred: Insecta



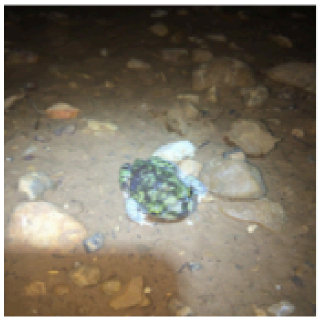
True: Amphibia
Pred: Amphibia



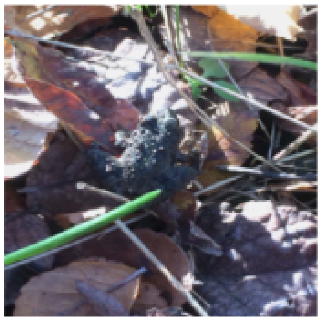
True: Amphibia
Pred: Insecta



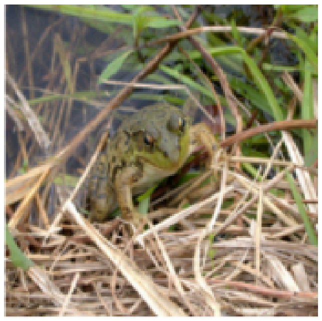
True: Amphibia
Pred: Animalia



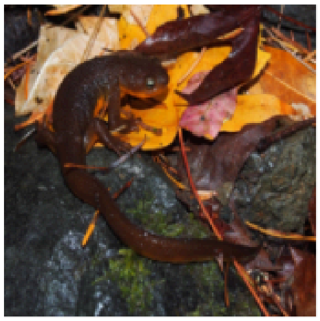
True: Amphibia
Pred: Animalia



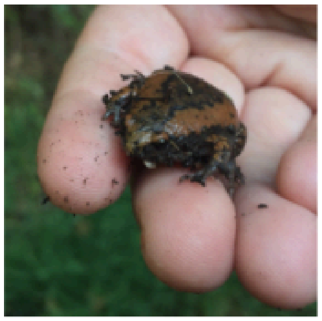
True: Amphibia
Pred: Mammalia



True: Amphibia
Pred: Fungi



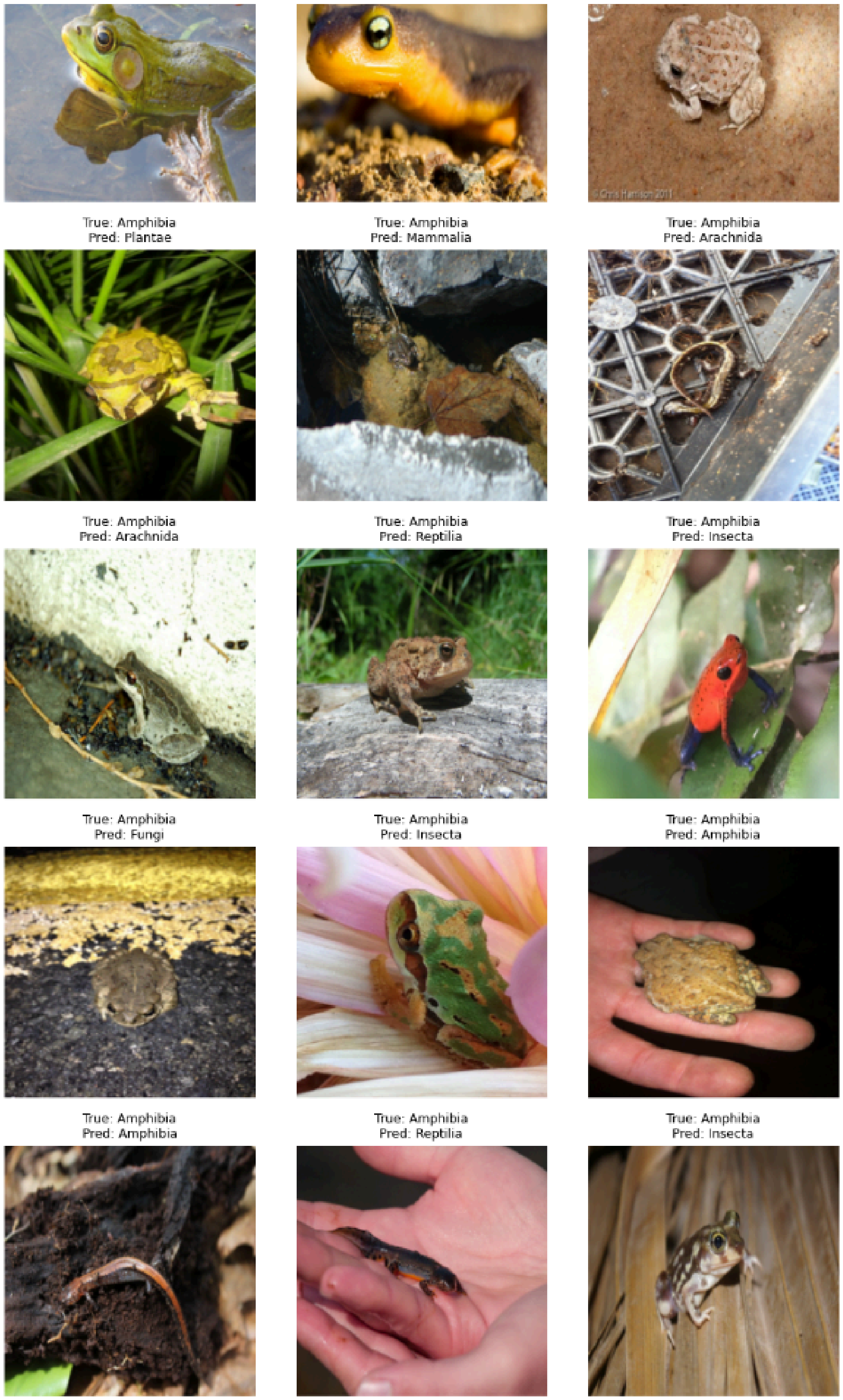
True: Amphibia
Pred: Insecta



True: Amphibia
Pred: Plantae

True: Amphibia
Pred: Insecta

True: Amphibia
Pred: Insecta



True vs Predicted test images

Question 5 (10 Marks)

Paste a link to your github code for Part A

Example:
https://github.com/sakthe1010/da6401_assignment2/tree/main/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 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:***

1. Image Dimension Handling:

The original pre-trained models on ImageNet typically expect image dimensions of 224×224 pixels. To align with this requirement, all images from the iNaturalist dataset are resized to this standard dimension during preprocessing. This resizing ensures compatibility with the model architecture, enabling the model to leverage the pre-trained feature extraction capabilities effectively.

2. Adjusting the Output Layer:

The original ResNet50 model, trained on ImageNet, contains an output layer with 1000 neurons, corresponding to the 1000 classes in the ImageNet dataset. However, our iNaturalist dataset contains only 10 classes. To adapt the pre-trained model to our task, the final fully connected (FC) layer is replaced with a new layer specifically designed for 10 output classes. This adjustment ensures that the model outputs predictions that are relevant and specific to our dataset.

Mathematically, this can be expressed as:

$$\text{New FC layer} = \text{Linear}\ (2048, 10)$$

where 2048 is the number of features from the last convolutional layer of ResNet50, and 10 corresponds to the number of classes in our dataset.

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:

The strategies I tried are mentioned below:

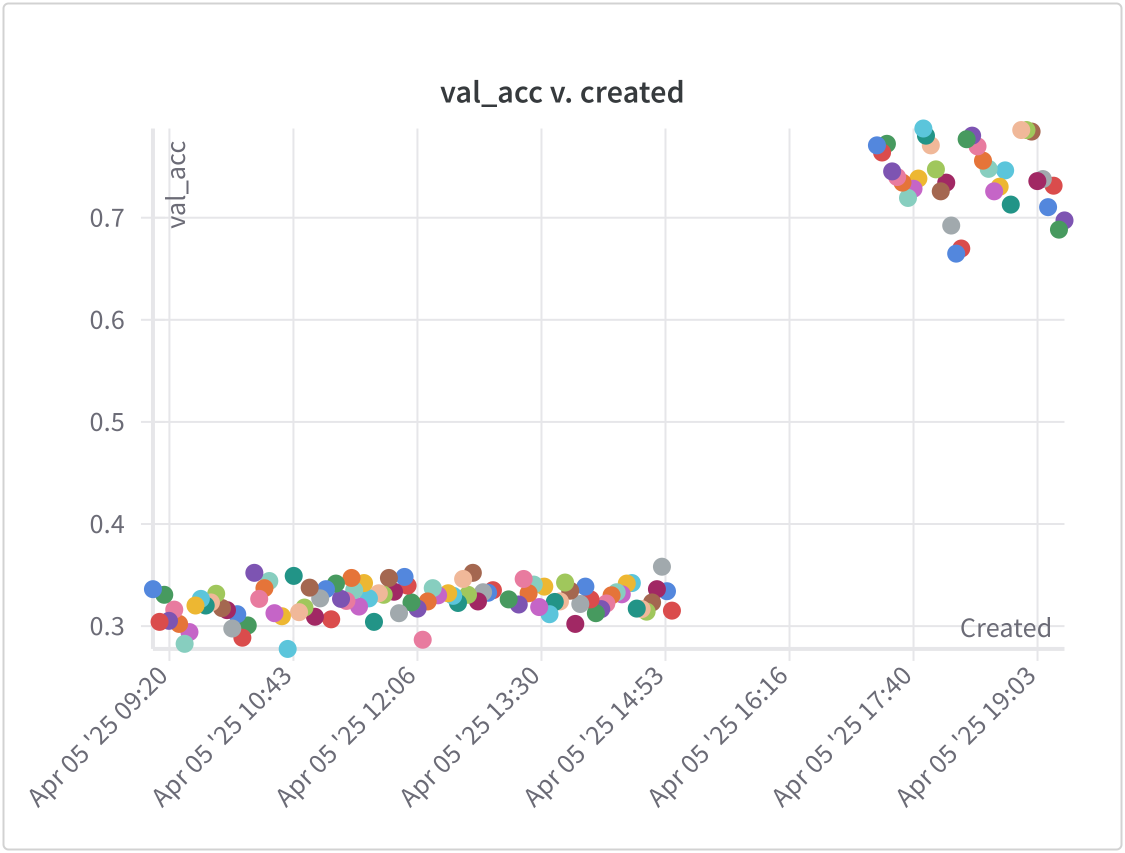
1. Freezing all layers except the final layer (Strat1): Reduce computation significantly by updating only the classification head.
2. Freezing all layers except last 2 layers (Strat2): Balance between efficiency and allowing meaningful model updates.
3. Full fine-tuning (Strat3): Fine-tuning the entire network, typically when ample computational resources and data are available

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:

I have tried the second strategy (strat2) as it strikes a good balance between training efficiency and accuracy, making it ideal for limited computational resources. I have fine-tuned the model, did a similar sweep as the previous task with required hyperparameters. The observations are as follows:



From the above plot you can clearly see the difference between training a model from scratch and fine-tuning a large pre-trained model. The small CNN model produces validation accuracy in the range of 0.3-0.4 whereas the ResNet50 model performs significantly better, producing validation accuracy in the range of 0.7-0.8.

- Fine-tuning a pretrained ResNet50 model resulted in achieving higher accuracy in fewer epochs compared to training a model from scratch.
- Pretrained models generally achieve significantly higher accuracy quickly due to transfer learning from ImageNet features.
- Training from scratch demands more epochs, computational power, and data augmentation techniques to reach similar performance to even limited fine-tuning strategies.

- ResNet50 contains 50 layers compared to our 5 layers in our small CNN model, hence the difference in accuracy.

Question 4 (10 Marks)

Paste a link to your GitHub code for Part B

Example:
https://github.com/sakthe1010/da6401_assignment2/tree/main/partB

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

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

Self Declaration

I, Sakthe Balan G K (Roll no: ME21B174), 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.