Share Comment Star ...

Assignment 2

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

Argha Boksi cs21d407, Shubham Mallik Thakur

Created on March 23 | Last edited on April 3

Part A: Training from scratch

Question 1 :

The CNN model was built using Keras. The model has five convolutional layers. Each convolutional layer is followed by a ReLU activation and a max pooling layer. After 5 such conv-relu-maxpool blocks of layers we have one dense layer followed by the output layer containing 10 neurons (1 for each of the 10 classes). The code is flexible such that the number of filters, size of filters, activation function in each layer and number of neurons in the dense layer can be changed. One needs to update the "sweep_configuration" object in our code in order to make such changes.

(a) Total no. of computations :

We made the following assumptions while calculating the total number of computations :

i) There are 'm' filters in each layer

ii) Filter size is k x k

iii) There are 'n' neurons in the fully connected layer.

iv) Input image size (256, 256, 3)

We considered the following computations :

i) Multiplications

ii) Additions

iii) ReLU operations

iv) Max operations in the Maxpool layers

v) Softmax Calculation in the output layer

First Convolutional layer :

For one pixel - 3 * k * k multiplications, 3 * (k * k) additions

For the image - 256 * 256 * 3 * k * k multiplications, 256 * 256 * 3 * (k * k - 1) additions

ReLU 1 Layer :

Output size : m * 256 * 256 (because we are considering Same convolution with appropriate padding)

Relu operations - m * 256 * 256

MaxPooling 1 Layer :

Reduces the output size to half , hence no. of max operations - m * 128 * 128

Second Convolutional layer :

For one pixel - m * k * k multiplications, m * (k * k) additions

For the image - 128 * 128 * m * k * k multiplications, 128 * 128 * m * (k * k - 1) additions

ReLU 2 Layer :

Output size : m * 128 * 128 (because we are considering Same convolution with appropriate padding)

Relu operations - m * 128 * 128

MaxPooling 2 Layer :

Reduces the output's size to half, hence no. of max operations - m * 64 * 64

Third Convolutional Layer :

For one pixel - m * k * k multiplications, m * (k * k) additions

For the image - 64 * 64 * m * k * k multiplications, 64 * 64 * m * (k * k - 1) additions

ReLU 3 Layer :

Output size : m * 64 * 64 (because we are considering Same convolution with appropriate padding)

Relu operations - m * 64 * 64

MaxPooling 3 Layer :

Reduces the output's size to half, hence number of max operations - m * 32 * 32

Fourth Convolutional Layer :

For one pixel - m * k * k multiplications, m * (k * k) additions

For the image - 32 * 32 * m * k * k multiplications, 32 * 32 * m * (k * k - 1) additions

ReLU 4 Layer :

Output size : m * 32 * 32

Relu operations - m * 32 * 32

MaxPooling 4 Layer :

Reduces the output's size to half, hence number of max operations - m * 16 * 16

Fifth Convolutional layer :

For one pixel - m * k * k multiplications, m * (k * k) additions

For the image - 16 * 16 * m * k * k multiplications, 16 * 16 * m * (k * k - 1) additions

ReLU 5 Layer :

Output size : m * 16 * 16

Relu operations - m * 16 * 16

MaxPooling 5 Layer :

Reduces the output's size to half, hence number of max operations - m * 8 * 8

Fully Connected Layer :

Number of multiplications - (m * 8 * 8) * n

Number of additions for multiplication - (m * 8 * 8 - 1) * n

Number of additions for bias - n

Relu operations - n

Output Layer :

Number of multiplications - n * 10

Number of additions for multiplication - (n - 1) * 10

Number of additions for bias - 10

Number of softmax operations - 10

(b) Total no. of parameters :

First Convolutional layer :

For one filter - 3 * k * k parameters

For 'm' filters - m * 3 * k * k parameters

Second Convolutional layer :

For one filter - m * k * k parameters

For 'm' filters - m * m * k * k parameters

Third, Fourth, Fifth Convolutional Layer :

For one filter - m * k * k parameters

For 'm' filters - m * m * k * k

For the three layers - 3 * m * m * k * k

Bias parameters :

We have a total of 5*m filters. One bias parameter corresponding to each of those filters. Hence 5*m bias parameters.

Fully Connected Layer :

((m * 8 * 8) * n)+n parameters

Output Layer :

(n * 10 + 10) parameters

Question 2 :

We trained our model using the iNaturalist dataset. 10% of the training data was set aside for hyper parameter tuning.

We created a "sweep_configuration" object which contains different hyperparameter values we tried during our experimentation.

Hyperparameters and their corresponding values:

filter_shape : [(3, 3),(3, 3),(3, 3),(3, 3),(3, 3)], [(3, 3),(5, 5),(3, 3),(3, 3),(3, 3)], [(5, 5),(5, 5),(5, 5),(5, 5),(5, 5)], [(5, 5), (5, 5),(3, 3),(3, 3),(3, 3)], [(3, 3),(5, 5),(5, 5),(3, 3),(3, 3)]

dropout_probability : [0, 0.25, 0.20, 0.30]

learning_rate : [1e-3, 1e-4]

activation_function : ['relu']

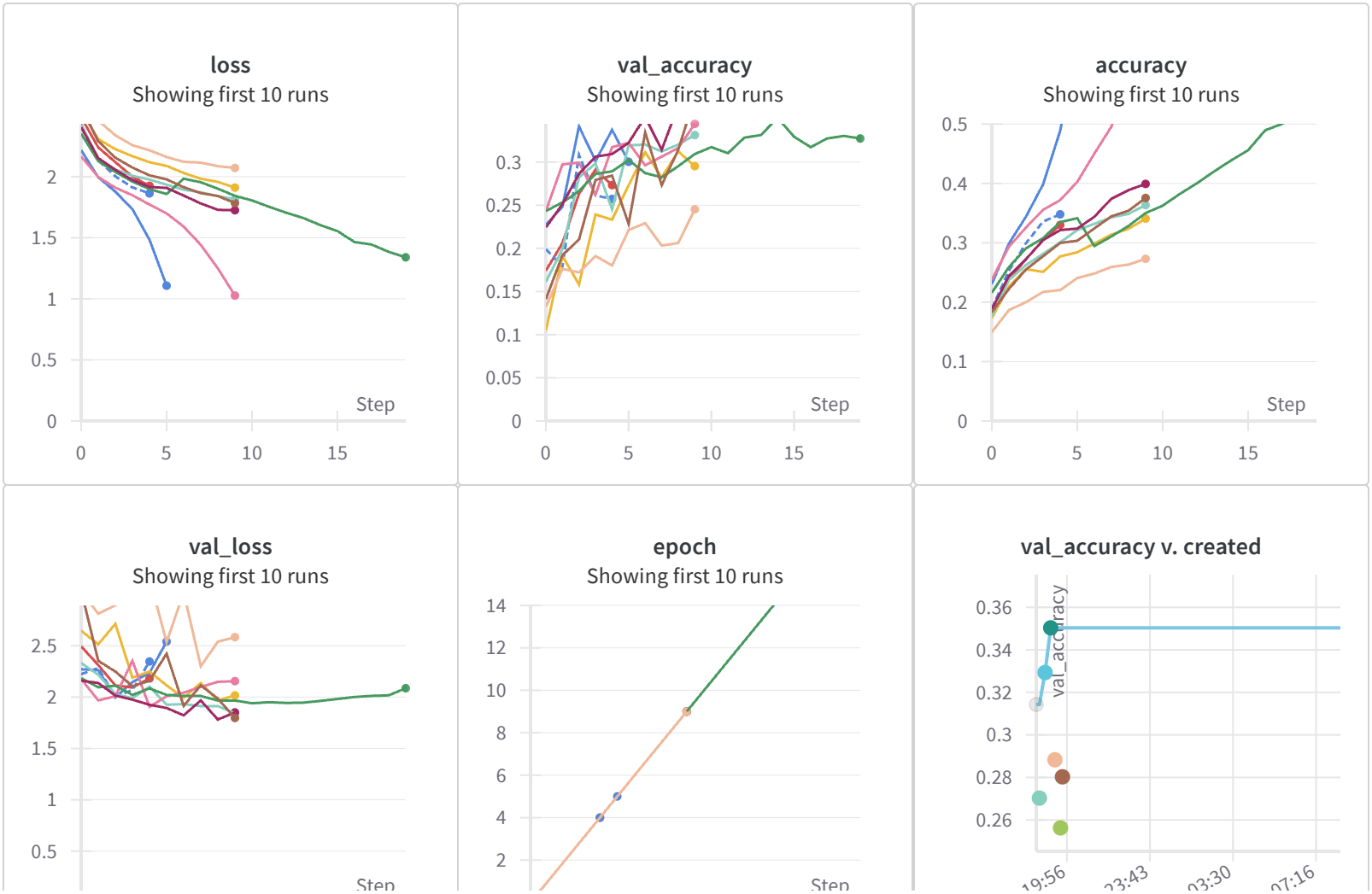
batch_normalization : ['true', 'false']

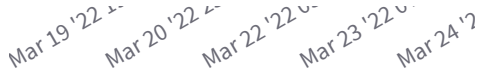
filter_organization : [64, 64, 128, 512, 512], [32, 64, 128, 512, 512], [32, 32, 64, 64, 128], [32, 64, 64, 128, 128], [32, 64, 128, 128, 512], [64, 64, 64, 64, 64], [512, 128, 128, 64, 32]

data_augmentation : ['true', 'false']

FC_layer_size : [32, 64, 128, 256, 512]

num_of_epochs : [5, 10, 20]



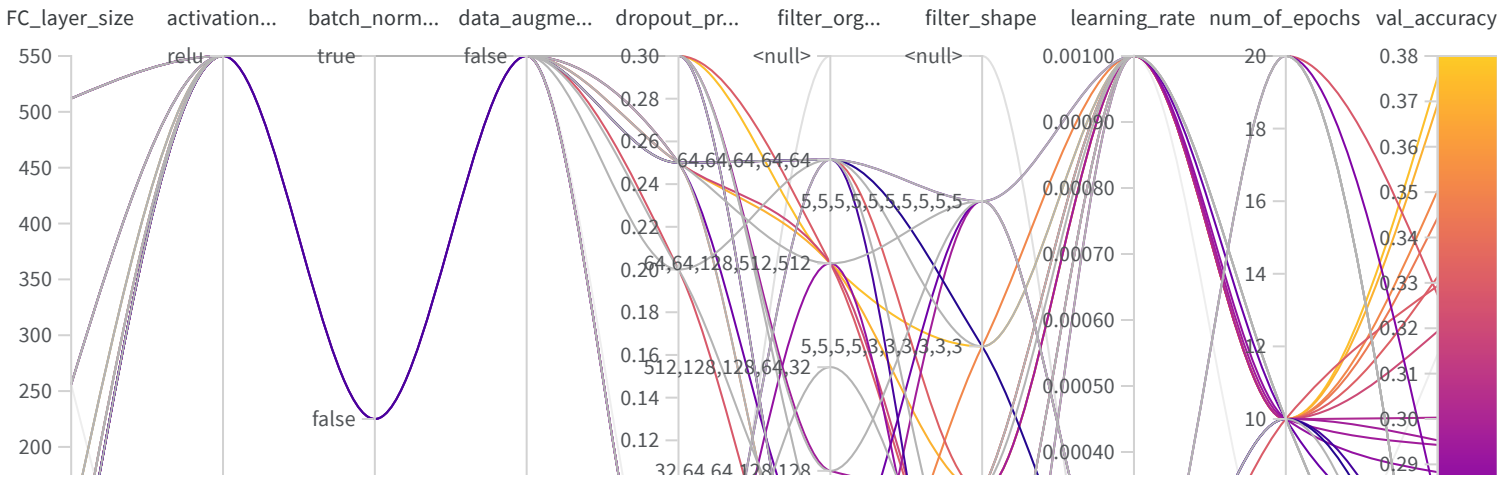


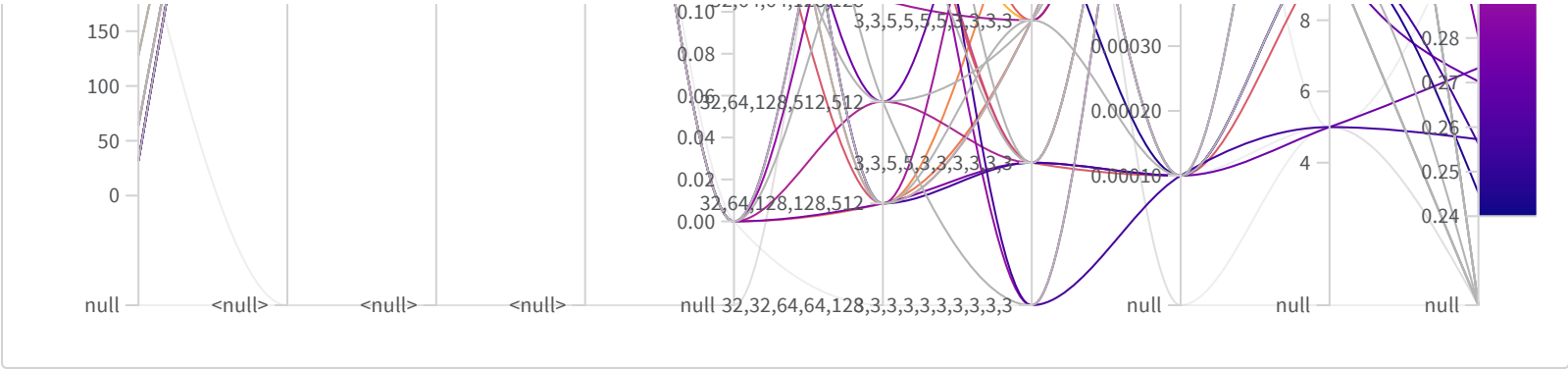
Parameter importance with respect to val_accuracy

Search

Parameters 1-9 of 9

Config parameter	Importance	Correlation
learning_rate		
Runtime		
dropout_probability		
FC_layer_size		
num_of_epochs		
batch_normalization.value_true		





For efficient hyperparameter search we picked "Bayesian Search" method provided by wandb. "Bayesian Sweeps" create a probabilistic model of metric score as a function of the hyperparameters, and choose parameters with high probability of improving the metric(in our case the metric is "val_accuracy" and the goal is "maximize").

Question 3:

Based on our experimentation, the following inferences can be made:

1) The two most important hyperparameters are - dropout probability and learning rate, as can be seen from the correlation summary table. We observed the importance of these two hyperparameters during our experimentation as well. As an example, when wandb agent generated the following hyperparameter configuration :

filter_shape : [[3,3], [5,5], [3,3], [3,3], [3,3]]

filter_organization : [32, 64, 128, 128, 512]

FC_layer_size : 32

dropout_probability : 0

batchnorm : true

number of epochs : 16

we observed a training accuracy of 96.87%. But the validation accuracy was only 28%. That means in absence of dropout probability our model was overfitting on the training data. Dropout acts as a regularization method and allows the model to generalize on unseen data.

2) We observed that increasing the number of filters as we go deeper into the network gave better validation accuracies. The runs that had same number of filters across layers did not give good results. The following filter organization gave us the best validation accuracy(37.54%) : [64, 64, 128, 512, 512]. This method of increasing the number of filters is observed in networks such as AlexNet, ZFNet, VGGNet.

3) We tried (3,3) and (5,5) filters across different layers. We tried three different strategies -

i) same filter_shape for all the layers

ii) (3, 3) filters for initial layers and (5, 5) filters for deeper layers

iii) (5, 5) filters for initial layers and (3, 3) filters for deeper layers

Apart from these three strategies we also tried some random combinations. We found using different filter shapes across layers is a better strategy than using the same filter shape for all layers. The following configuration gave us the best validation accuracy(37.54%) : [(5, 5), (5,5), (3, 3), (3, 3), (3, 3)]. Two other configurations also gave good results on the validation data - [(3,3), (5,5), (3,3), (3,3), (3,3)] and [(3,3), (5,5), (5,5), (3,3), (3,3)].

4) Batch normalization is used to speed up training and it allows us to use higher learning rates. It makes training a deep neural net easier. We observed the same during our runs. In absence of

batch norm we observed training accuracy in the range 33-36% and validation accuracy in the range 24-28%. By default after analysing a few runs bayesian sweeps chose parameters with high probability of improving the metric. So most of the times bayesian sweep set "batch_norm = true" during our experimentations. With batch norm both training accuracy and validation accuracy improved.

5) We encountered overfitting whenever dropout probability was set to zero. Whenever dropout was included in the model architecture there was no overfitting. In the presence of dropout(mostly 0.25) the training accuracy was in the range(55-65%) and the validation accuracy was in the range(30-38%).

6) As explained in the previous point we didn't encounter overfitting with dropout. Hence the Bayesian sweep selected dropout(0.25) most of the time. And whenever it included dropout, it didn't pick data augmentation. Both dropout and data augmentation address the issue of overfitting. During our experimentation Bayesian sweep gave more importance to dropout than data augentation most of the time. Hence data augmentation was not done too many times by the bayesian sweep.

7) The size of the dense layer did not affect the validation accuracy too much. When we had relatively fewer neurons(32, 64) in the dense layer dropout probability of 0.10 and 0.20 worked better. Whereas with large no. of neurons(512) in the dense layer dropout probability of 0.30 worked better. This is because - as we increase the no. of neurons in the dense layer, no. of parameters increases and regularization becomes necessary to prevent overfitting.

8) We used Adam optimization algorithm for training our network. We tried two different learning rates - 1e-3, 1e-4. Training the network for more number of epochs resulted in better performance. This is also supported by the positive correlation associated with "num_of_epochs".

Question 4 :

(a) We obtained a validation accuracy of 37.54% for our best model.

Hyperparameters associated with the best model :

filter_shape : [(5, 5),(5,5),(3, 3),(3, 3),(3, 3)]

filter_organization : [64, 64,128, 512, 512]

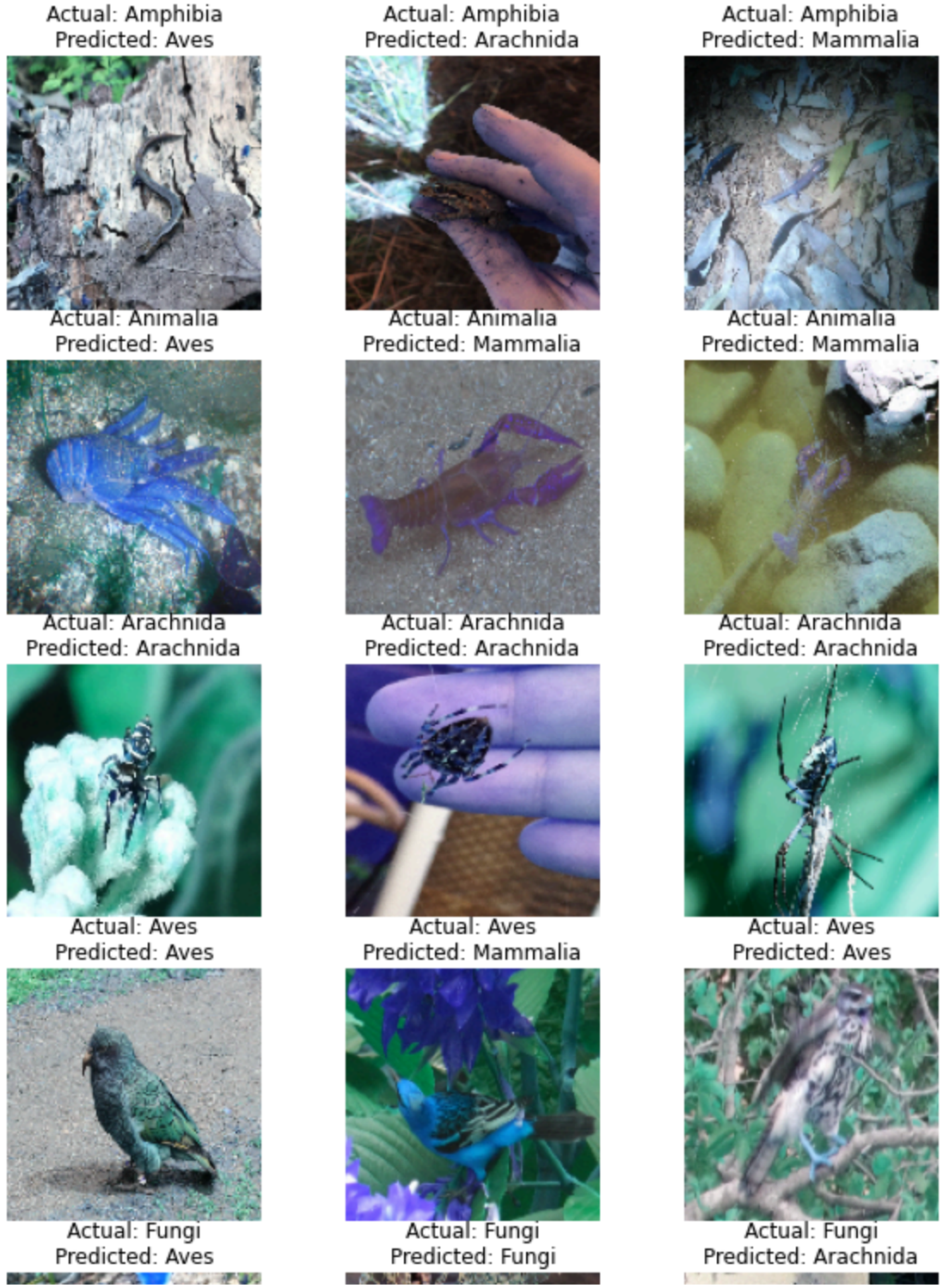
FC_layer_size : 512

dropout_probability : 0.3

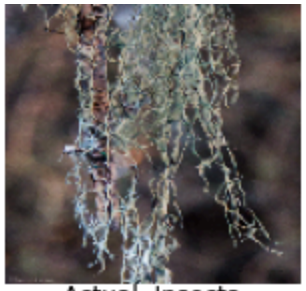
batch-normalization : true

Using the best set of hyperparameters reported above, the model was evaluated and the test accuracy was **30.39%**.

(b) A 10 x 3 grid containing sample images from the test data and predictions made by our best model have been displayed below.


Actual: Amphibia
Predicted: Aves
Actual: Amphibia
Predicted: Arachnida
Actual: Amphibia
Predicted: Mammalia
Actual: Animalia
Predicted: Aves
Actual: Animalia
Predicted: Mammalia
Actual: Animalia
Predicted: Mammalia
Actual: Arachnida
Predicted: Arachnida
Actual: Arachnida
Predicted: Arachnida
Actual: Arachnida
Predicted: Arachnida
Actual: Aves
Predicted: Aves
Actual: Aves
Predicted: Mammalia
Actual: Aves
Predicted: Aves
Actual: Fungi
Predicted: Aves
Actual: Fungi
Predicted: Fungi
Actual: Fungi
Predicted: Arachnida

Actual: Insecta
Predicted: Aves


Actual: Insecta
Predicted: Insecta

Actual: Insecta
Predicted: Mammalia

Actual: Mammalia
Predicted: Mammalia

Actual: Mammalia
Predicted: Mammalia

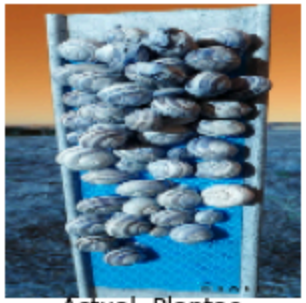
Actual: Mammalia
Predicted: Aves

Actual: Mollusca
Predicted: Arachnida

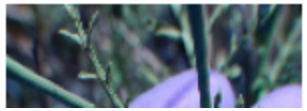
Actual: Mollusca
Predicted: Arachnida

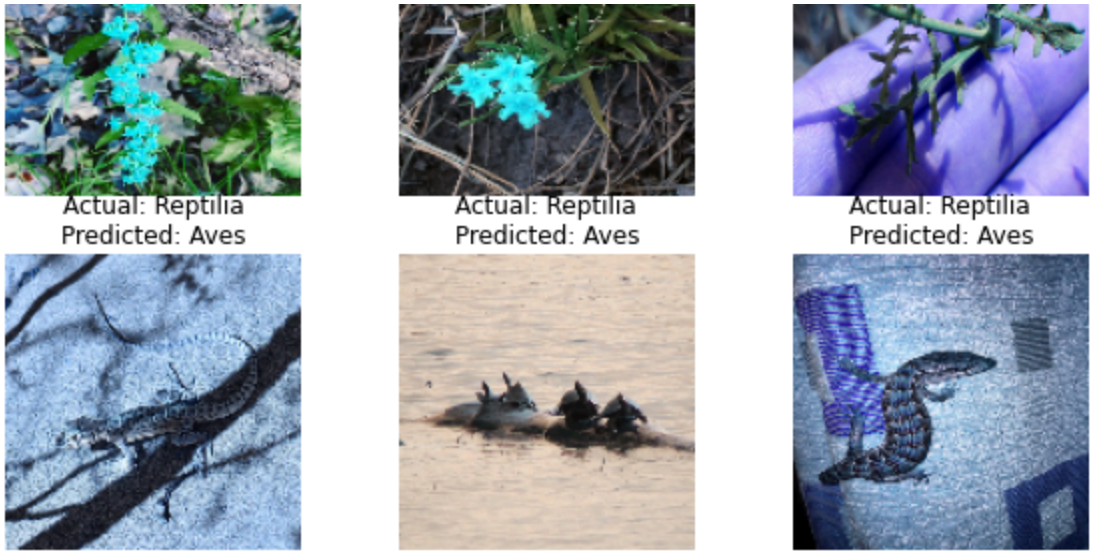
Actual: Mollusca
Predicted: Aves

Actual: Plantae
Predicted: Plantae

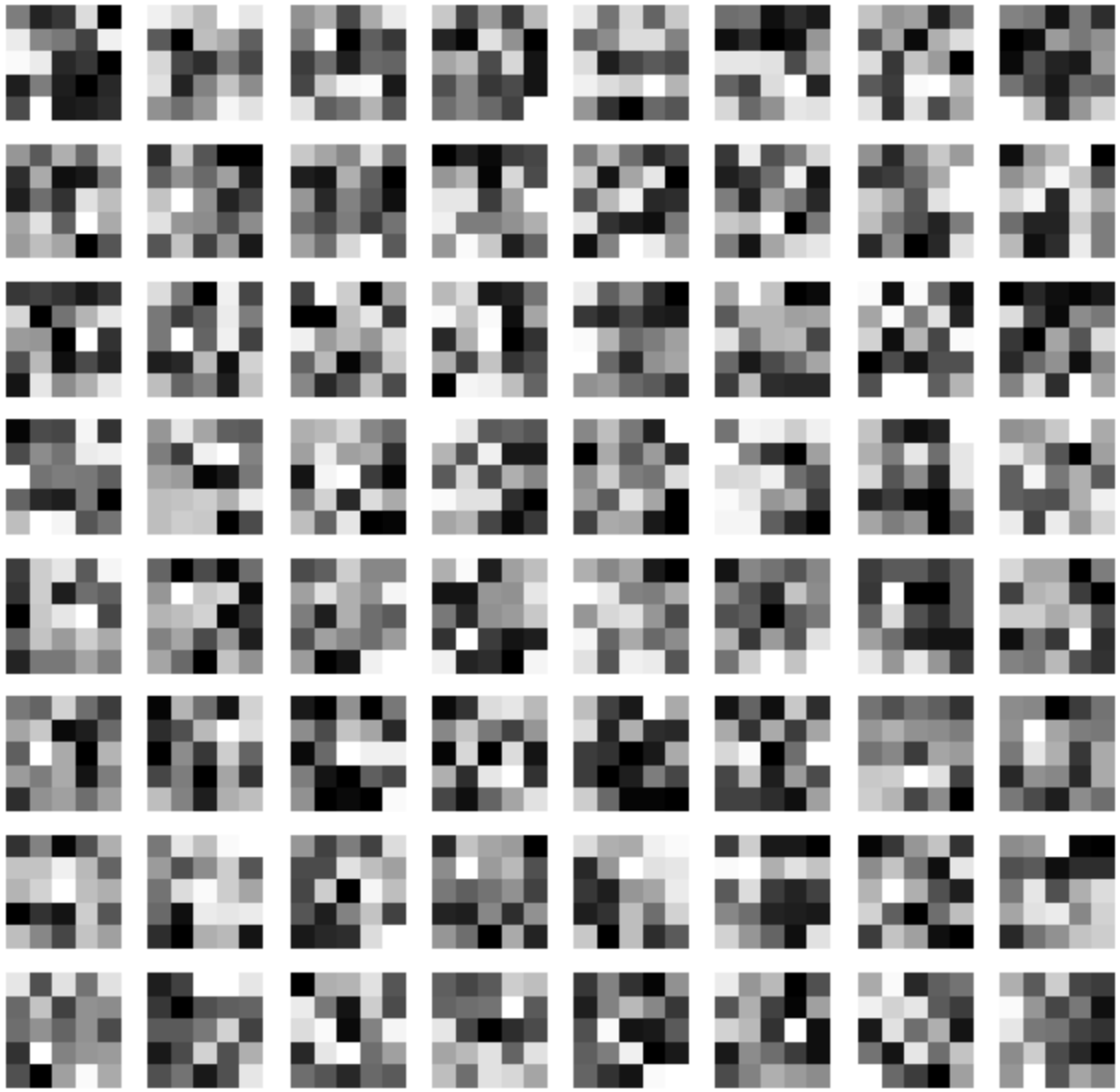

Actual: Plantae
Predicted: Plantae

Actual: Plantae
Predicted: Arachnida

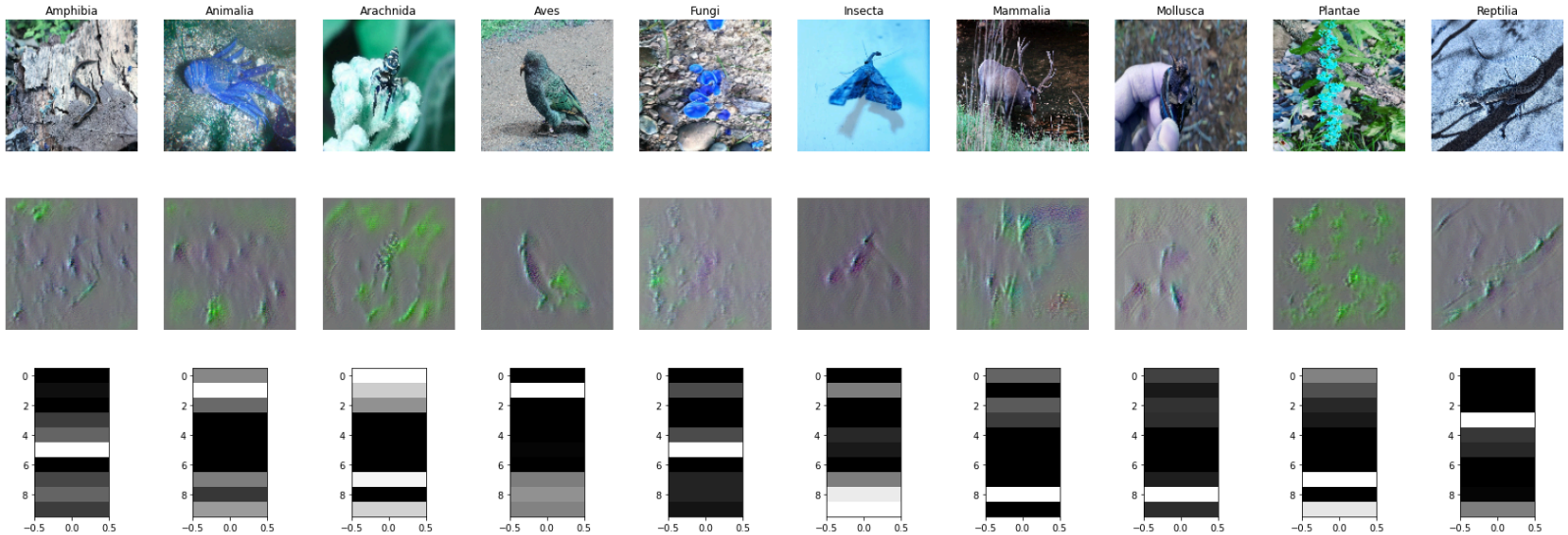




(c) Our best model has 64 filters in the first layer. The following 8 x 8 grid displays the 64 filters. As we know the earlier layers of a neural network detect simpler patterns like edges, somewhat later layers might detect parts of objects and then even later layers detect parts of complete objects. We have visualized the 64 filters of the first layer which look for simpler patterns across the input images. All the 64 filters correspond to 64 specific patterns.



- Question 5 :



We have randomly selected 10 neurons from the 5th convolutional layer, applied guided backpropagation, and visualized them for all the 10 available classes. We have also plotted the gradients so that we can see which pixels are more influential according to the model while training. Let us look, as an example, at the image of pigeon. The gradients are only significant for the pixels which actually correspond to the pigeon.

Question 6 :

Github link for Part A -

https://github.com/Shubhammt/cs6910_assignment2/tree/main/Part%20A

Part B : Fine-tuning a pre-trained model

Question 1:

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

We loaded four models(InceptionV3, ResNet50, InceptionResNetV2, Xception) from 'keras.applications'. The pre-trained models in 'keras.applications' take in an argument 'input_shape' which resizes the model based on the input size. The training and test images are resized to 128 x 128 pixels. This does not affect any filter weights in the first part of the network as the filters are indifferent to input size.

(b) 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?

The pre-trained models in 'keras.applications' take in an argument 'include_top' which can be set to 'False'. In that case the fully-connected output layers of the model used to make predictions is not loaded, allowing a new output layer to be added and trained. To address the problem of different number of classes, we loaded the pre-trained models without the fully connected layers using "include_top = False" argument to the model. Then we added our own fully connected layers to predict for 10 classes instead of 1000.

Our implementation is modular, we have kept an attribute called “pretrained_models” within the “sweep_configuration” object. This attribute contains all the model names. It allows Wandb agent to select whichever model it wants to run.

Question 2

(a) For fine tuning large models we employ the trick of freezing the top few layers of our model.

(b) We are allowed to do so because we are importing some models which are pretrained on imagenet dataset and their filters have already learned to detect useful patterns in the images.

(c) This helps us achieve the target of keeping our training tractable as we now no longer need to train the entire huge model and just need to update the weights of the remaining layers.

(d) Thus we can get reasonably good accuracy in short amount of time as compared to training the full model.

(e) The natural question is to determine how many layers to freeze. This can be taken as a hyperparameter, which can be experimented with while training.

Question 3 :

We performed a Bayesian Search sweep over the following hyperparameters :

freeze_before : [50, 60, 70]

pretrained_models : ['InceptionV3', 'InceptionResNetV2', 'ResNet50V2', 'Xception']

FC_layer_size : [32, 64, 128, 256, 512]

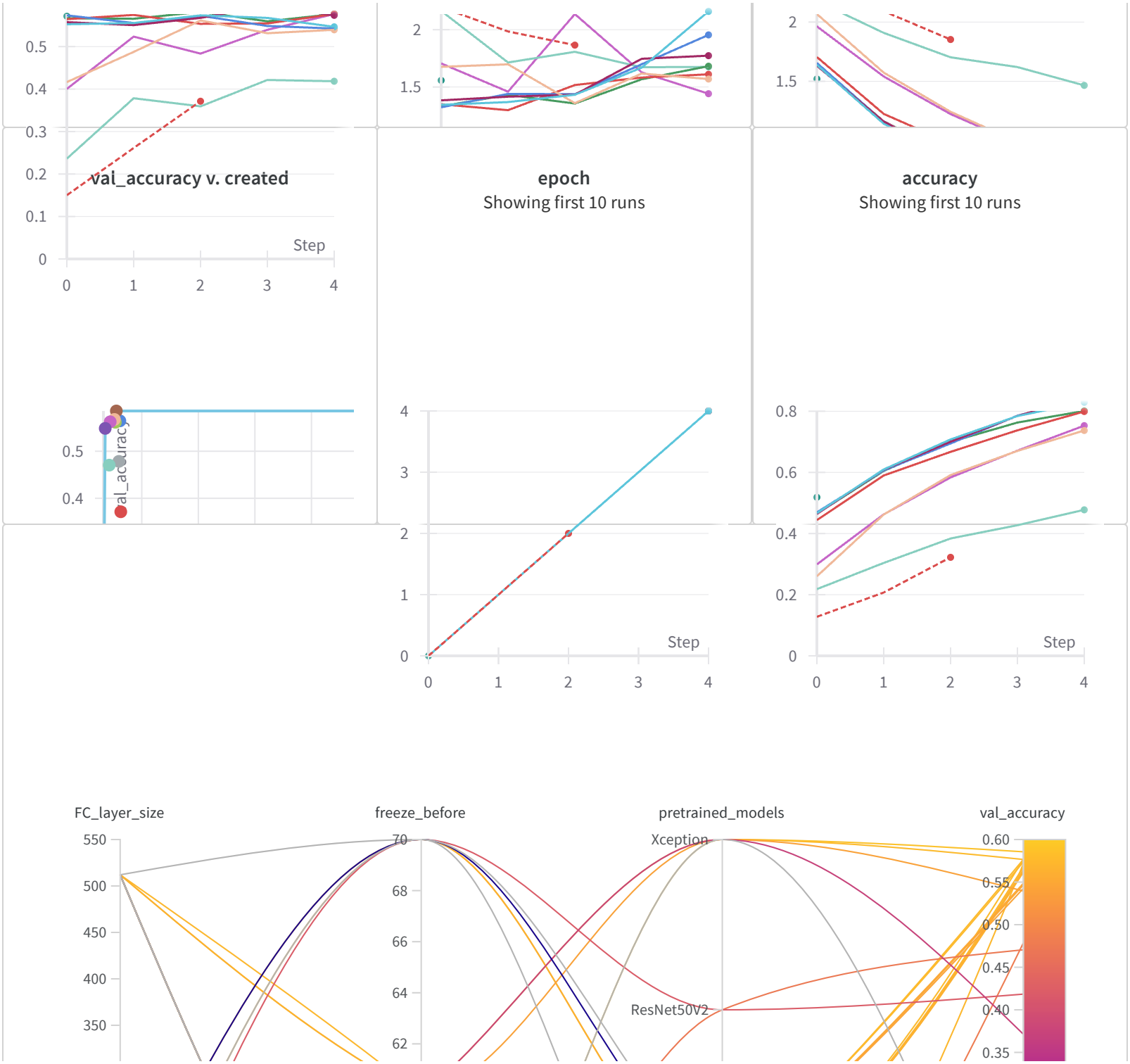
num_of_epochs': [5]

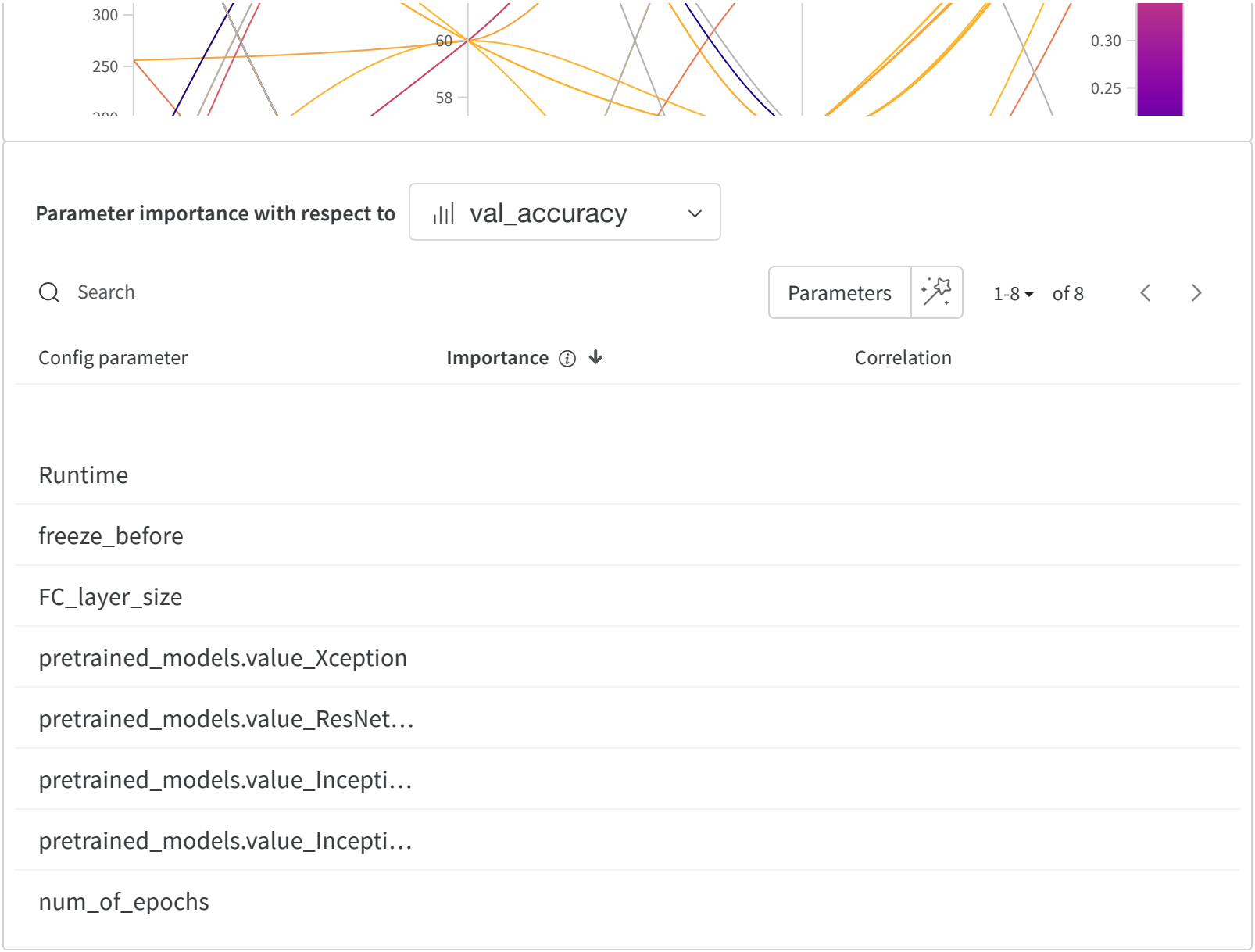
We used Adam as the optimizer.

The highest validation accuracy obtained was **58.56%** for the following model :

Xception with 512 neurons in the FC layer, number of epochs = 5

val_accuracy	val_loss	loss
Showing first 10 runs	Showing first 10 runs	Showing first 10 runs


val_accuracy v. created
Step
epoch
Showing first 10 runs
accuracy
Showing first 10 runs
Step
Step
val_accuracy
FC_layer_size
freeze_before
pretrained_models
val_accuracy
Xception
ResNet50V2



Some inferences:

- Among pretrained models, InceptionV3 appeared to be the best choice. For InceptionV3 the validation accuracy was close to 60% most of the time. This inference is backed by the high positive correlation value reflected in the table.
- We observed that "freeze_before" has a lot of impact on validation accuracy. Whenever its large, it tends to reduce the validation accuracy. This is surprising because a large "freeze_before" indicate a large number of trainable parameters and such a model should be more capable of capturing patterns in the data and hence has increased performance.
- Fine tuning a pretrained model with large number of layers provides much better performance than a simpler network trained from scratch. Clearly our validation accuracy improved drastically from Part A to Part B. Also we noticed that fine tuning a pretrained model is relatively faster than training a model from scratch.
- Increasing fully connected layer size seems to increase validation accuracy. This is opposite to what we observed in Part A of this assignment, where the correlation between the two was negative.
- InceptionResNetV2 seems to train very fast as compared to InceptionV3 and Xception.

Question 4 :

Github link for Part B :

https://github.com/Shubhammt/cs6910_assignment2/tree/main/Part%20B

Part C : Trespassing alert

Question 1

Camera traps are used to remotely monitor the behavior of some rare and lesser known animals in their natural habitat. Often these habitats are disturbed by presence of illegal trespassers who many a times deal with animal trafficking. The camera traps are very well hidden from both animals and humans alike, and sometimes finds these unwanted trespassers, but it only records the footage and is not able to alert any authorities due to absence of modules which tell humans apart from the other fauna. This can however be achieved by real time object-detection functionalities provided by YOLO models.

We trained a YOLOv3 model on a custom dataset of images having both both humans and animals in the frame. This model is then tested on a video taken from youtube. We have added the dataset in the github repo as well.

Self Declaration :

Shubham Mallik Thakur(MA20M022) : (50% contribution)

Argha Boksi (CS21D407): (50% contribution)

We, Shubham Mallik Thakur and Argha Boksi, swear on our honour that the above declaration is correct.

Date: 03-04-2022

Created with ❤️ on Weights & Biases.

https://wandb.ai/shubham-argha/Assignment%202/reports/Assignment-2--VmlldzoxNzI3OTc5