Assignment 2 - Part A

A CNN Model: training from Scratch

Arjun Kumar Gupta

Arjun Kumar Gupta,Milan Chatterjee

Ans 1.

> A small CNN model consisting of 5 convolutional layers where each layer followed by batch Norm,Relu activation and a max pool layer is build.After 5 such layers a dense layer followed by the output layer which contains 10 output neurons(since iNaturalist Dataset have 10 classes) is constructed.Also the number of filters ,the size of filters ,activation functions,number of neurons in dense layer etc can be changed in each of the layers.

> **a).** The number of computations in the first layer would be of the order = (I - **(k-1)**) *(I - **(k-1)**) * k * k * 3
> The same number of computation in first activation layer would be in the order (I - **(k-1)**) *(I - **(k-1)**) * m
> The number of computations in the 2nd layer would be of the order = (I - 2**(k-1)**) *(I - 2**(k-1)**) * k * k * m
> The same number of computation in second activation layer would be in the order (I - 2**(k-1)**) *(I - 2**(k-1)**) * m
> The number of computations in the third layer would be of the order = (I - 3**(k-1)**) *(I - 3**(k-1)**) * k * k * m
> The same number of computation in third activation layer would be in the order (I - 3**(k-1)**) *(I - 3**(k-1)**) * m
> The number of computations in the fourth layer would be of the order = (I - 4**(k-1)**) *(I - 4**(k-1)**) * k * k * m
> The same number of computation in fourth activation layer would be in the order (I - 4**(k-1)**) *(I - 4**(k-**

1)) * m
The number of computations in the fifth layer would be of the order = (I - 5**(k-1)**) *(I - 5**(k-1)**) * k * k * m
The same number of computation in fifth activation layer would be in the order (I - 5**(k-1)**) *(I - 5**(k-1)**) * m
The number of computations in the dense layer would be ((I - 5**(k-1)**) * (I - 5**(k-1)**) * m + 1) * n
The number of computaions in activation layers would be n +1
The number of computations in output layer would be (n +1) * 10 + 1
Therfore the total number of computations would be a sum of all these.

b). Assuming stride = 1, padding = 0, m filters in each layer,I * I image size in input layer,k * k size of each filter,n neurons in dense layer.

The number of parameters in 1st layer of CNN = m* k * k * 3
The number of parameters in 2nd layers of CNN = m * k * k * m
The number of parameters in 3nd layers of CNN = m * k * k * m
The number of parameters in 4nd layers of CNN = m * k * k * m
The number of parameters in 5nd layers of CNN = m * k * k * m
The number of parameters in dense layer = ((I - 5**(k-1)**) * (I - 5**(k-1)**) * m + 1) * n
The number of parameter in output layer = (n +1) * 10
Therefore the number of parameters in the convolutional layer is =
m* k * k * 3 + 4 * m * k * k * m + ((I - 5**(k-1)**) * (I - 5**(k-1)**) * m + 1) * n + (n +1) * 10 approx
Therefore the number of parameter in the 5 layers of CNN = 5 * m * k * k

Ans 2.

The model thus build was trained over the iNatural Dataset. The train dataset was divided into 9:1 ratio where the 10 % of the dataset was used as validation set for hyper parameter tuning.Here i have show the accuracy obtained over the different runs together with their respective validation and training los

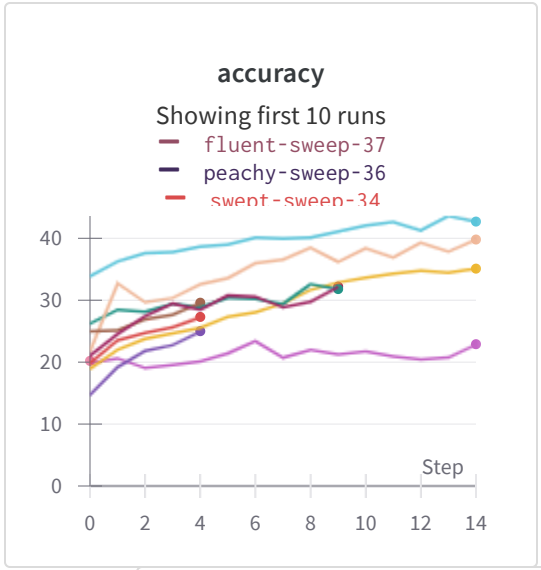
> Also the occuracy vs created plot,parallel co-ordinates plot and correlation plot is plotted from the sweep run.

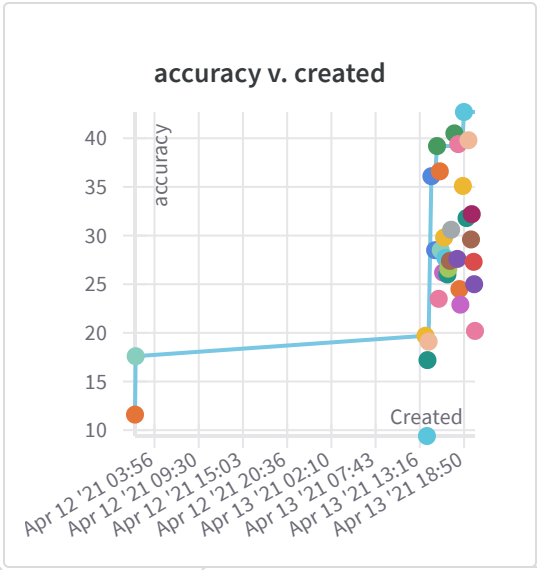
> The sweep hyper parameters over which the sweep was configured is show below:

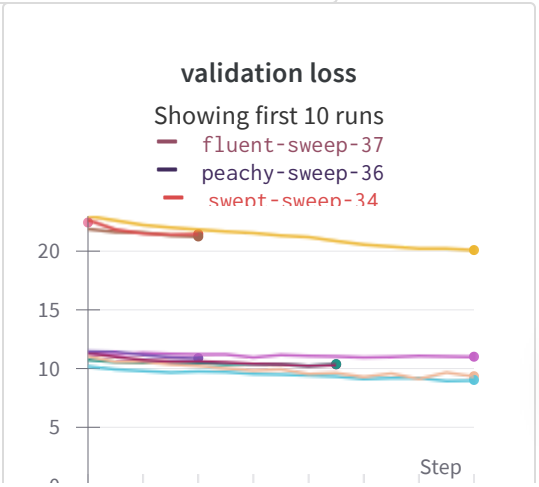
```
'parameters': {
    'lr': {
        'values': [0.0001, 0.0002,0.0004]
    },
    'activation_fn': {
        'values': ['relu', 'tanh','leaky_relu']
    },
    'num_of_filters': {
        'values' : [32,64,128,256]
    },
    'filter_size' : {
        'values' : [3,5,7]
    },
    'filter_organization':{
        'values': [2,1]
    },
    'batch_size':{
        'values': [100,200]
    },
    'dropdout_ratio': {
        'values': [0.2,0.3,0.4,0.5]
    },
    'dense_neuron_size':{
        'values' : [256,128,64]
    },
    'data_augmentation':{
        'values': [True,False]
    },
    'epochs':{
        'values' : [10,5,15]
    },
    'batch_norm':{
        'values' : [True,False]
```

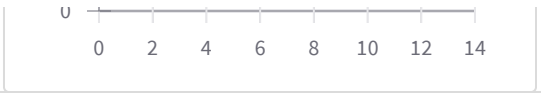
Fig 1. Sweep config parameters

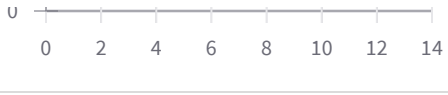
The strategy that i tried to reduce the number of runs still achieving the high accuracy was a techniques call "bayes" while configuring the method in sweep config.The method makes sure that the hyper parameters for which the accuracy was high than the others would run more with some slight changes to hyper parameters i.e the variables would be set up in such a way that the hyper parameters are more or less close to the hyper parameter in the best accuracy obtained till that point.Also before configuring the sweep we also tested with some configuration that made much sense in order to remove unnecessary runs.For example choosing learning rate to be always less than 0.0005 as in above the oscillations in the minimum region were very large thus resulting in not convergence quickly.

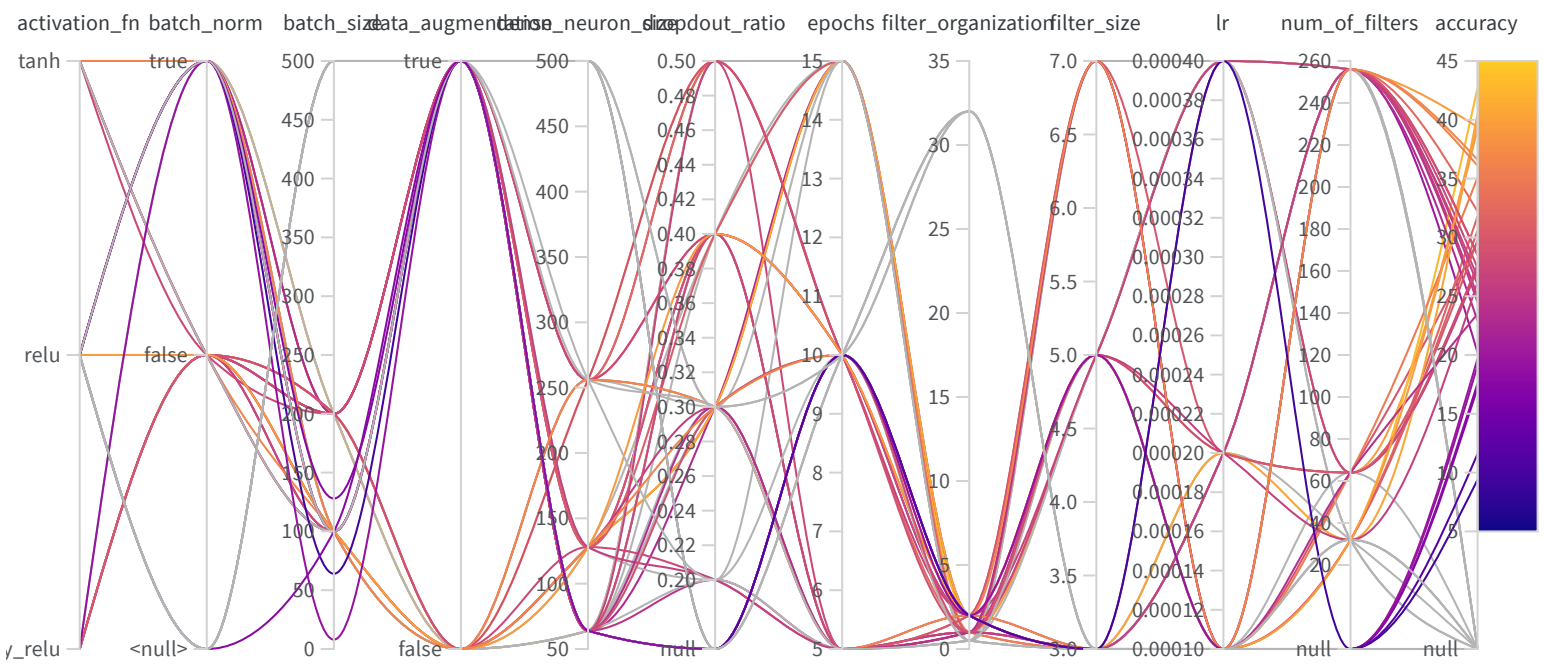












Parameter importance with respect to accuracy

Search | Parameters | Rows per page 10 | 1-10 of 18

Config parameter	Importance	Correlation
data_augmentation		
epochs		
num_of_filters		
Runtime		
dropout_ratio		

dropout_ratio

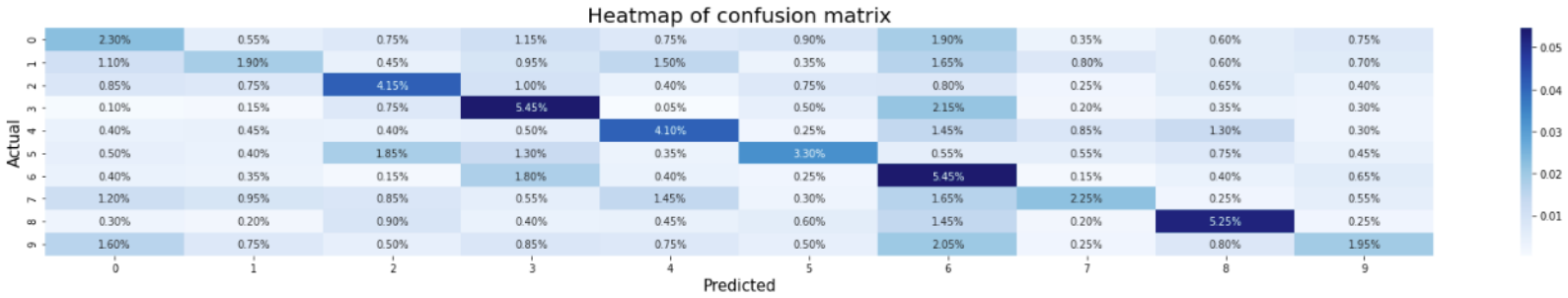
Run set 50

Ans 3

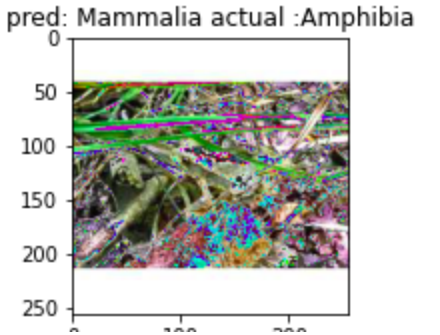
- Smaller number of filters in initial layers and then doubling the number of filters in later layers works better than larger number of filters in initial layers and then halving them in later layers.
- Dropout with value probability 50% in the dense layers works better than no dropout or with dropout but less probability.
- If the number of filters are large then large filter sizes e.g 7 works better than small filter sizes.
- Batch normalization after every layer works better than no normalization.
- The learning rate with smaller values especially 0.0001 or 0.0002 works better than 0.0004 or higher values.
- tanh activation works better and leads to higher accuracy in less time than relu or leaky relu activation function.
- The number of filters if high and remains high in the later layers then works better than lower number of filters in all the layers.
- More number of neurons in the dense layer works better than less number of neurons in the dense layer.

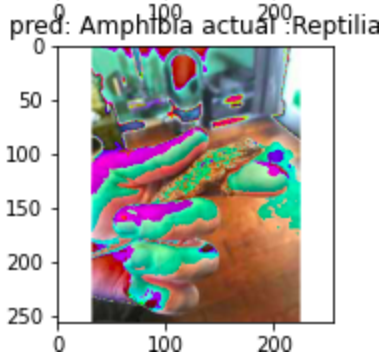
Ans 4.

> **a)**.The best model obtained from the sweep was tested over the test dataset. An accuracy of about 36.10 % was recorded.Here is a confusion matrix.

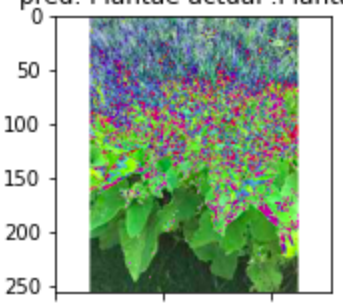

Heatmap of confusion matrix
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
2.30% 0.55% 0.75% 1.15% 0.75% 0.90% 1.90% 0.35% 0.60% 0.75%
1.10% 1.90% 0.45% 0.95% 1.50% 0.35% 1.65% 0.80% 0.60% 0.70%
0.85% 0.75% 4.15% 1.00% 0.40% 0.75% 0.80% 0.25% 0.65% 0.40%
0.10% 0.15% 0.75% 5.45% 0.05% 0.50% 2.15% 0.20% 0.35% 0.30%
0.40% 0.45% 0.40% 0.50% 4.10% 0.25% 1.45% 0.85% 1.30% 0.30%
0.50% 0.40% 1.85% 1.30% 0.35% 3.30% 0.55% 0.55% 0.75% 0.45%
0.40% 0.35% 0.15% 1.80% 0.40% 0.25% 5.45% 0.15% 0.40% 0.65%
1.20% 0.95% 0.85% 0.55% 1.45% 0.30% 1.65% 2.25% 0.25% 0.55%
0.30% 0.20% 0.90% 0.40% 0.45% 0.60% 1.45% 0.20% 5.25% 0.25%
1.60% 0.75% 0.50% 0.85% 0.75% 0.50% 2.05% 0.25% 0.80% 1.95%
0.05
0.04
0.03
0.02
0.01

b).


pred: Mammalia actual :Amphibia
0
50
100
150
200
250
0
100
200

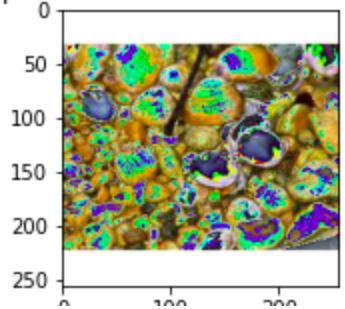

pred: Amphibia actual :Reptilia
0
50
100
150
200
250
0
100
200

pred: Plantae actual :Plantae

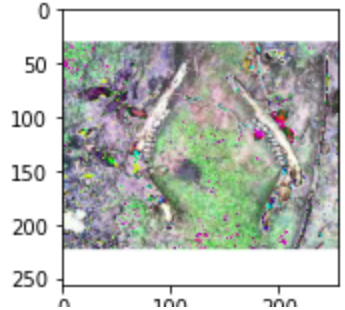


pred: Amphibia actual :Reptilia

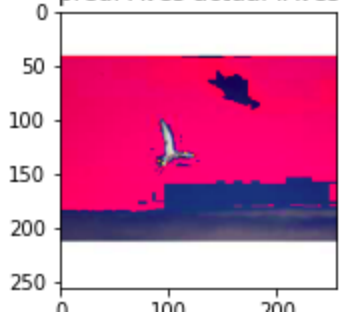
pred: Mammalia actual :Mollusca

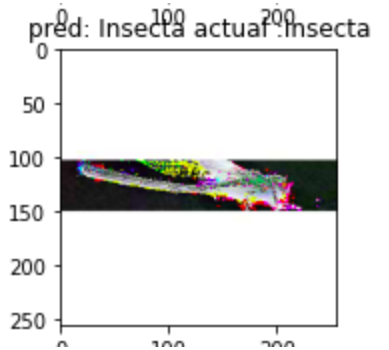


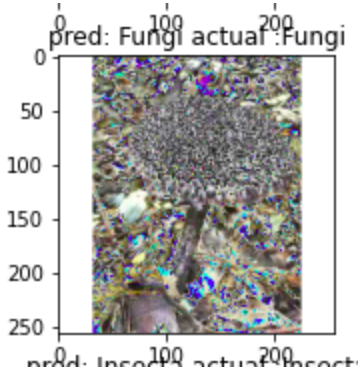
pred: Mammalia actual :Mammalia

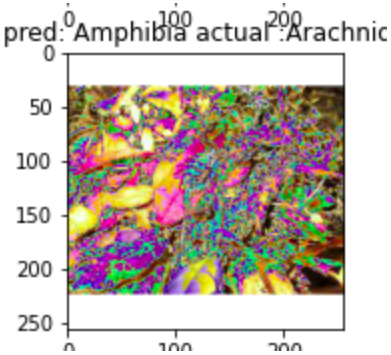


pred: Aves actual :Aves

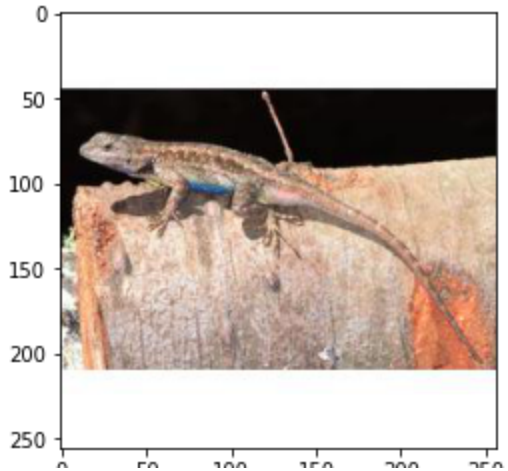






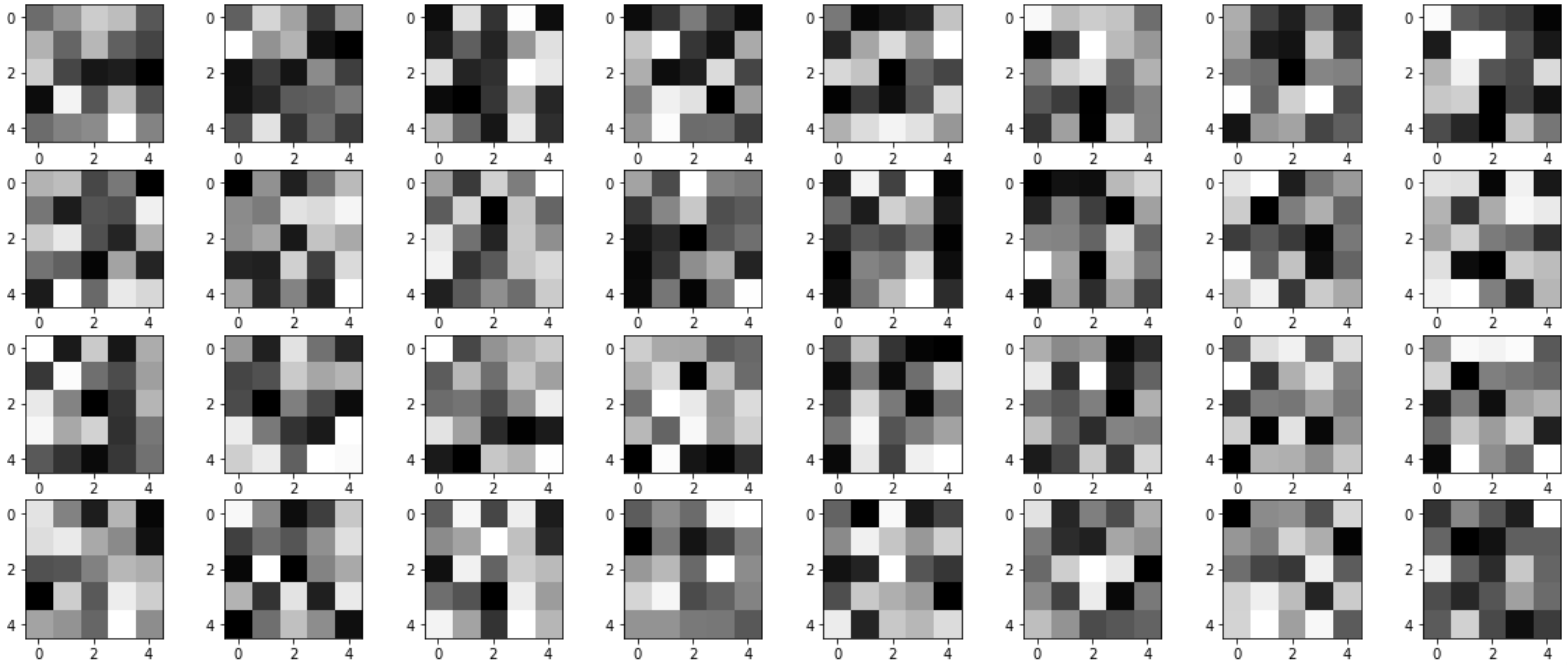


c). The image of a lizard is visualized after 1st convolution layer operation.Here is the original image.



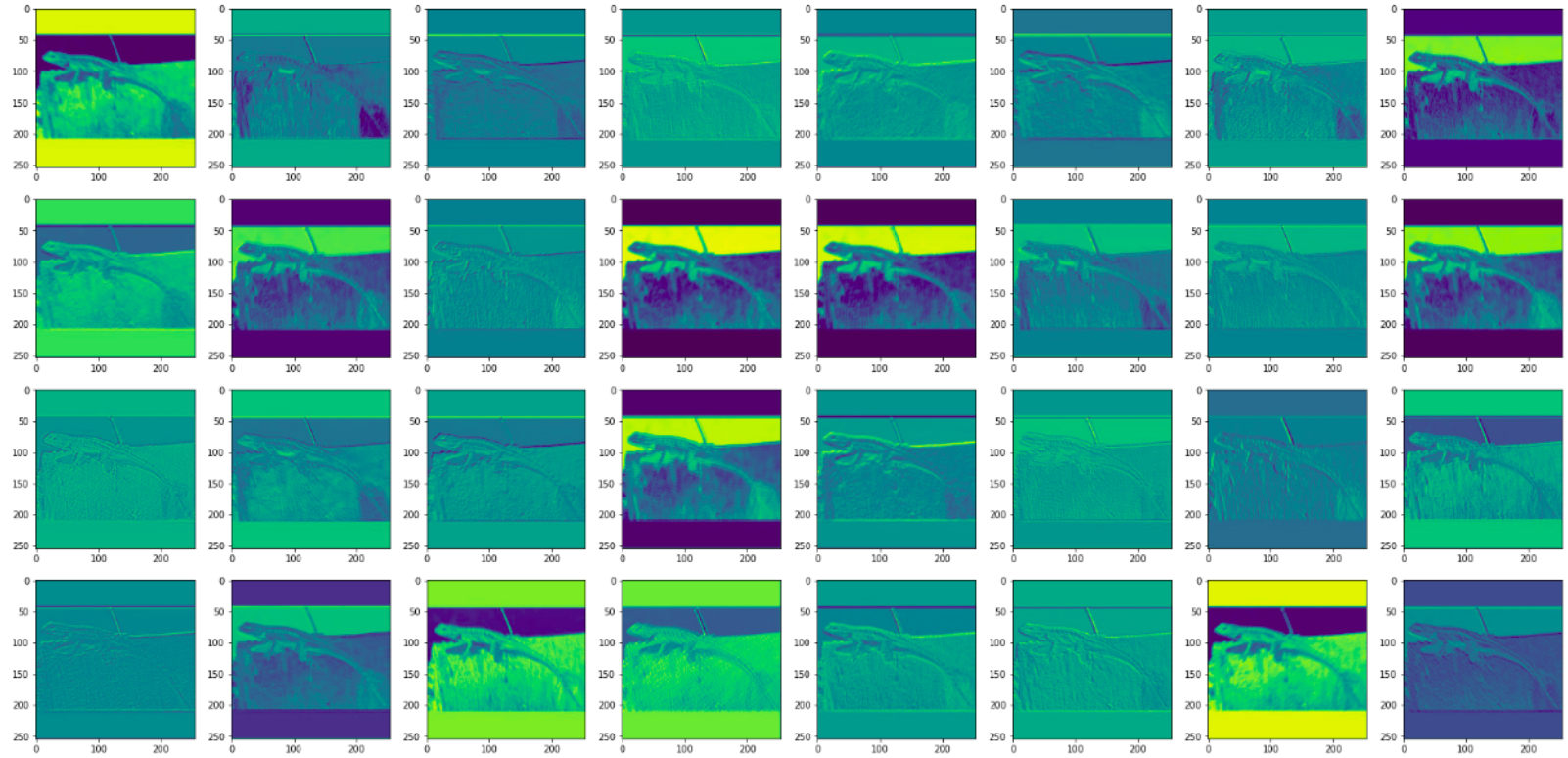
Original lizard

The visual representation of the filters in the first layer convolutional layer is given below:



The imges of the lizard after the 1st layer convolutional operation is given below:

```
torch.Size([32, 254, 254])
```



```
Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
```

Ans 5.

> Guided Backpropagation combines vanilla backpropagation at ReLUs (leveraging which elements are positive in the preceding feature map) with DeconvNets (keeping only positive error signals).Since we are only interested in what image features the neuron detects so when propagating the gradient, we set all the negative gradients to 0. We don't care if a pixel suppresses(negative value) a neuron somewhere along the part to our neuron.

Ans 6.

Here this the github link to the code:

https://github.com/theindianwriter/CS6910-assignment_2

Created with ❤️ on Weights & Biases.

https://wandb.ai/theindianwriter/cs6910-assignment2-part_a/reports/Assignment-2-Part-A--Vmlldzo2MTQxMjg

Assignment 2 Part B

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

Mitesh Khapra

Arjun Gupta, Milan Chatterjee

Instructions

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 keras, you can load any model (InceptionV3, InceptionResNetV2, ResNet50, Xception, 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 (keras directly provides these weights). Please answer the following questions:

(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?

Ans: For this we have to resize our images to the same size as used in the pretrained model we are using. For example for inception models input size is (299,299) and for most of the other models it is (224,224). This can be seen in the methods initialize_*model() and Dataset*_Transform() part of the code.

(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?

Ans: As we know that number of classes should be equal to number of neurons in the last layer, and the last layer in cnn is fully connected layer. So to address the above problem we have to initialize the model with num_*classes = 10, so that fully connected layer can have 10 neurons . This is achieved in the initializ_model() method using the code*

num_ftrs = model_ft.fc.in_features

model_ft.fc = nn.Linear(num_ftrs, num_classes)

Your implementation should be modular so that it allows to swap in any model (InceptionV3, InceptionResNetV2, ResNet50, Xception).

/comment: Yes our implementation is modular , just provide appropriate model_*name from a list of model*_name to choose from in the method

#startTraining(model_name ="resnet50", num_classess =10, batch_size =32, num_epochs=5,feature_extract=False,no_layers_to_unfreeze=1,lr=0.001,gamma=0.1,opt = "sgd").

In case that model does not exist it will show invalid model name , exiting. In that user can add that model in the initialize_moodel() method.

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

Question 2 (5 Marks)

You will notice that InceptionV3, InceptionResNetV2, ResNet50, Xception 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 different strategies that you tried (simple bullet points would be fine).

Ans:

pretraining means model we are going to use is already trained on similar task. Fine tuning means our input may not be of similar shape and size as that of pretrained model also we may not have that much computational resource power or time on which pretrained model was trained . Finetuning is the tweaks and tricks to be applied so that we can get better result on our task using little modification in the pretrained model

Background knowledge: Last layers of cnn captures /play most important role in image classification / object detection problem so this has to be kept in mind while developing strategies.

Strategies to keep the training tractable:

- Feature Extracting: Freezing all layers except last fully connected layer
  - this done in our code keeping the feature_*extract flag as true , and no*_layers_*to*_unfreeze
  - In the case only the weights of last layer will be updated

- Freezing upto k layers from the last
  - this done in our code keeping the feature_*extract flag as true , and no*_layers_*to*_unfreeze =k
  - In the case only the weights of last k layers will be updated
- Using different optimizer and scheduler:
  - Different optimizers are helpful in faster convergence like sgd with moenyum, adagrad, adam etc
  - Choosing a learning rate scheduler, which decreases the learning rate of the optimizer overtime and helps prevent non-convergence due to large learning rates.
  - lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=gamma) i.e. decrease learning rate by a factor of gamma after every 7 epoch
- We used different hyperparameters like which model to use, learning rate, optimizer, batch size etc

Question 3 (15 Marks)

Now finetune the model using different strategies that you discussed above and different hyperparameter choices. Based on these experiments write down some insightful inferences (once again you will find the sweep function to be useful to plot and compare different choices).

Here are some examples of inferences that you can draw:

- Using a huge pre-trained network works better than training a smaller network from scratch (as you did in Part A)
- InceptionV3 works better for this task than ResNet50
- Using a pre-trained model, leads to faster convergence as opposed to training a model from scratch
- ... .

(Note: I don't know if any of the above statements is true. I just wrote some random comments that came to my mind)
Of course, provide evidence (in the form of plots) for each inference.

Of course, provide appropriate plots for the above inferences (mostly automatically generated by wandb). The more insightful and thorough your inferences and the better the supporting evidence (in terms of plots), the more you will score in this question.

Answer:

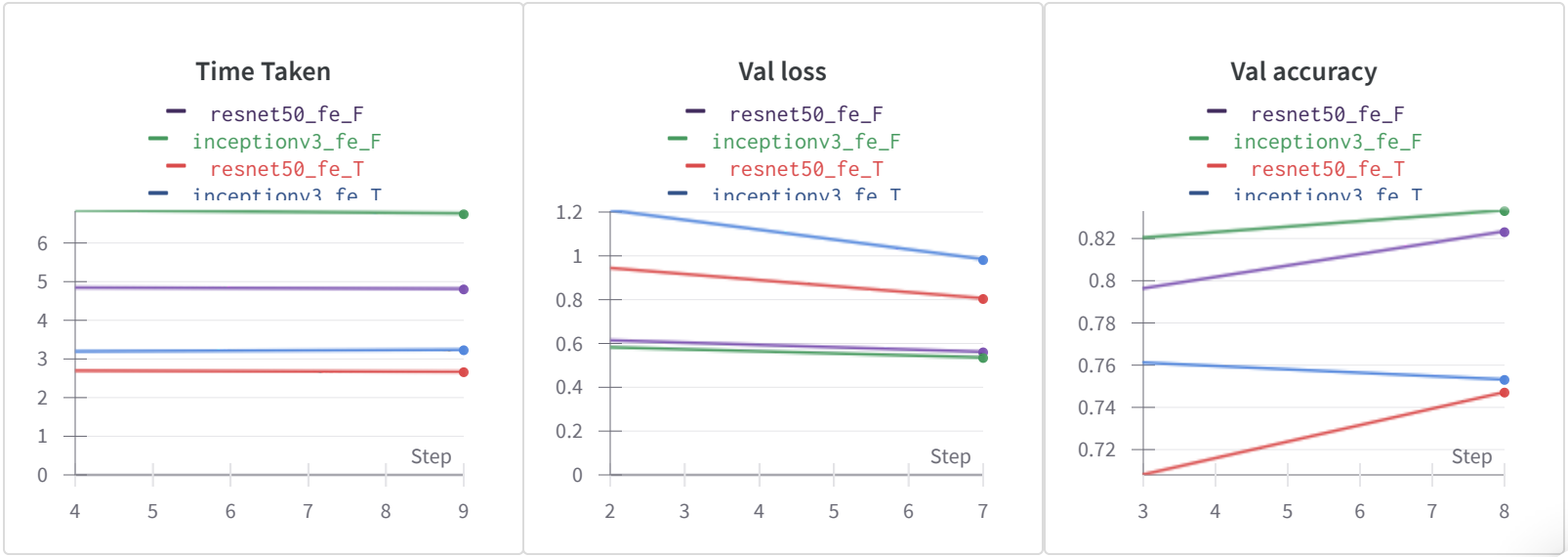
For drawing better inferences we run 2 separate sweeps. In one sweep we took 2 models inceptionv3 and resnet50 and use pretrained as False one time and True another time i.e. 4 combinations

Let us see plots corresponding to this.

Note: resnet50_fe *F means model is resnet 50and feature*_extract flag is set to False in that case all the layers of the model are trained and when feature_extract is set to true only the outermost layer is trained

model_name	feature_ext	Time Taken	Train Loss	Train accura	Val accuracy	Val loss
resnet50	false	4.798	0.7526	0.7462	0.823	0.5594
inceptionv3	false	6.743	1.022	0.753	0.833	0.5337
resnet50	true	2.657	1.026	0.6706	0.747	0.803
inceptionv3	true	3.225	1.94	0.6609	0.753	0.9806

Feature Extract Comparison

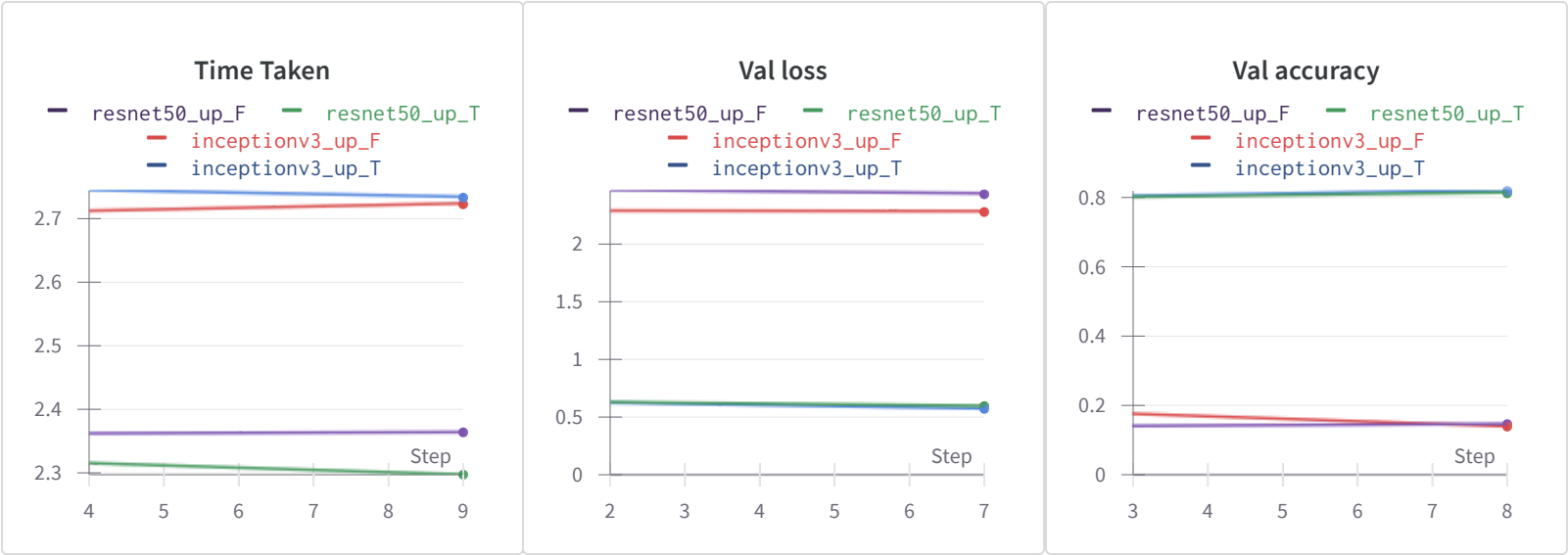


- From above graph we can clearly conclude that when feature extraction is set to False it takes almost half the time to train the model. for ex resnet 50 with feature extraction takes 2.657min while without feature extraction takes 4.798min similarly inceptionv3 with feature extarction takes 3.225min and without feature extraction takes 6.743min. At the same time accuracy varies around 8% , here resnet50 without feature extraction gives 82.3% of validation accuracy and with feature extraction gives 74.7% of validation accuracy while it is 83.33% and 75.3% for inceptionv3 model.
- So we can conclude if we go for the intermediate solution i.e train more than 1 layers from the last and not just all layers in the model by that we will get better accuracy training just layer as well as less time than training the whole network. This is one trick we used and got validation accuracy of 82.6% using resnet101 and training just last 3 layers.

- Let us analyse the second case i.e. use pretrained = False and use Pretrained = True. Lets consider the same plots naming convention is same as above instead of fe we use up to denote use pretrained flag

model_name	use_pretrai	Time Taken	Train Loss	Train accura	Val accuracy	Val loss
resnet50	false	2.364	2.342	0.1578	0.146	2.432
resnet50	true	2.297	0.7477	0.7497	0.812	0.5948
inceptionv3	false	2.723	3.002	0.1338	0.139	2.278
inceptionv3	true	2.733	1.02	0.7558	0.819	0.5711

Pretrained comparison

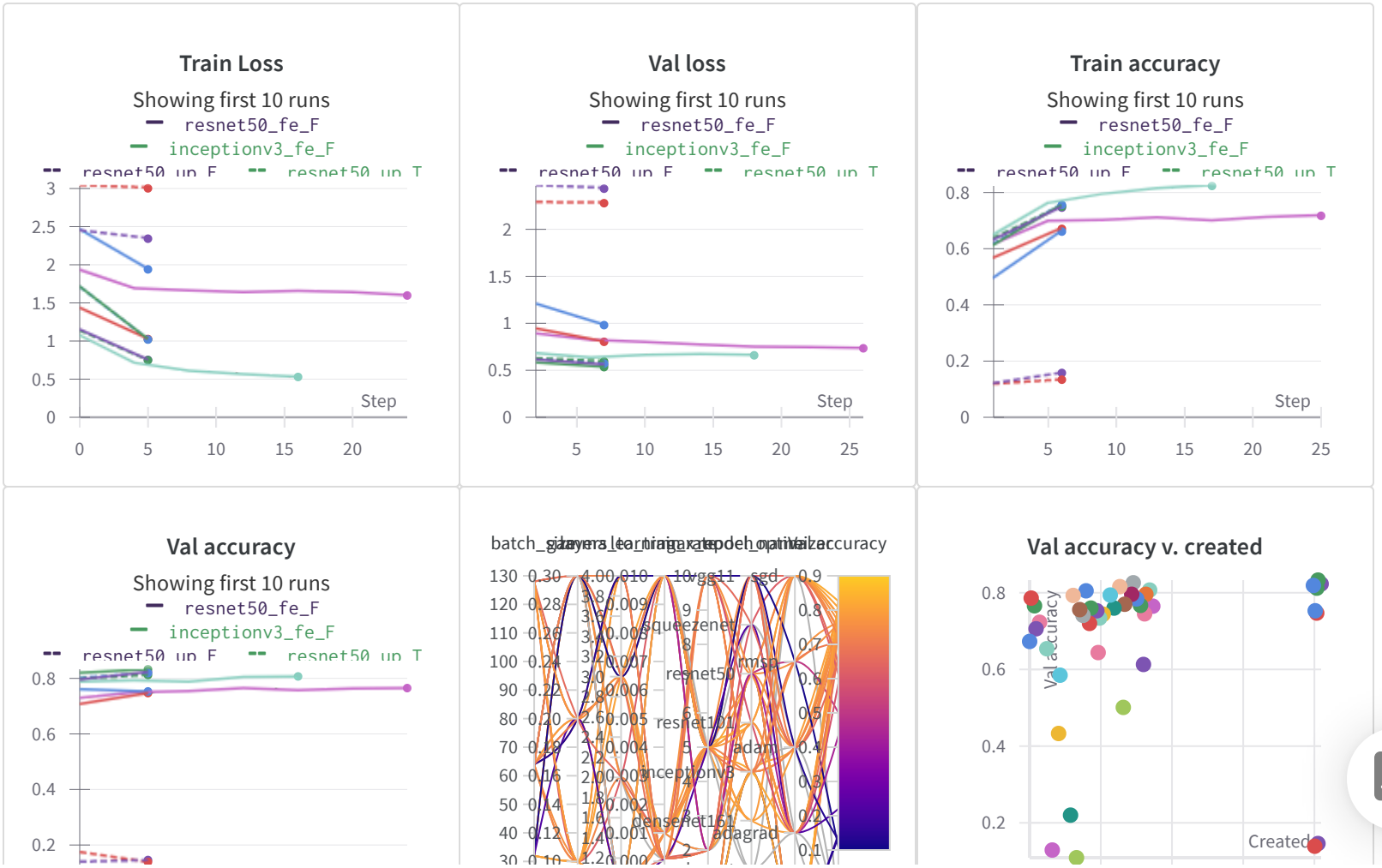


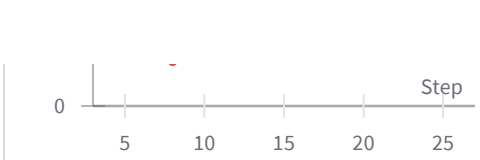
- From the above graphs it is clear that when pretrained version of the model is used (i.e. weight initialization is not random rather it is already arrived by pretraining the network on similar tasks) instead of training from scratch (i.e. which weight initialization are random) it almost take same time to train the model but there is a huge difference in accuracy when we are using pretrained model we are getting accuracy in the range of 80s but when we are training the same model from scratch it is just in the range of 15s.
- This happens because giant companies with massive computational resource have already pretrained the network on huge dataset which we dont have access to , so we can take advantage of the pretrained model and just train the last few layers.
-
- Now Lets see the broader picture i.e. over all comparisons

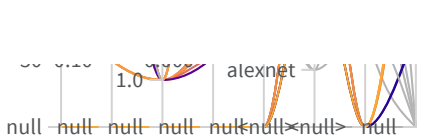
model_name	Train accura	Val accuracy
densenet161	0.7532	0.794
densenet161	0.7489	0.796
densenet161	0.7463	0.784
inceptionv3	0.717	0.765
inceptionv3	0.6722	0.744
inceptionv3	0.6153	0.706
inceptionv3	0.7972	0.816
inceptionv3	0.6591	0.723
inceptionv3	0.6671	0.734
inceptionv3	0.6995	0.77
inceptionv3	0.7421	0.796
resnet101	0.7926	0.826
resnet101	0.7202	0.766
resnet101	0.7356	0.786
resnet50	0.8233	0.807
resnet50	0.6947	0.753
resnet50	0.7461	0.76
resnet50	0.4666	0.501
squeezenet	0.5637	0.613
squeezenet	0.2002	0.22
squeezenet	0.6271	0.654
vgg11	0.7063	0.756
vgg11	0.6581	0.745
vgg11	0.09634	0.109

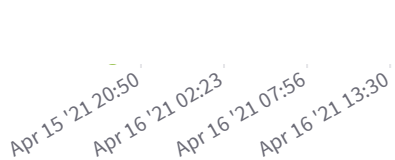
resnet101	0.7943	0.793
resnet101	0.5972	0.585
resnet101	0.7891	0.805

Figure1: Comparison between different models and validation accuracies




Step
0
5
10
15
20
25


alexnet
1.0
null
null
null
null
null
null
null
null


Apr 15 '21 20:50
Apr 16 '21 02:23
Apr 16 '21 07:56
Apr 16 '21 13:30

Run set 50

- From the above plots following conclusion can be drawn
- Models from resnet family like resnet50, resnet101 etc , inceptonv3 and densenet161 have comparable performances in terms of validation accuracy, resnet101 gives validation accuracy of 82.6% and inceptionv3 gives accuracy of 81.6%
- squeezenet, vgg11, and alexnet are not as suitable as inceptionv3 and resnet families
- Figure1 and validation accuracy versus created chart gives justification for the same.
- Huge pre-trained network works better than training a smaller network from scratch because more huge network learn complex relationship better than small network.
- This is very evident from the fact that validation accuracy in case of partA was in the range of 35s but in case of part B it is crossing even 80 it can reach even 86+ if we finetune the model by by taking the model which have accuracies in higher range and for those using different optimizer to make them out from local maxima
- This is also true that pre-trained model, leads to faster convergence as opposed to training a model from scratch but in some cases the difference may not be much but there will be huge difference in accuracy.
- Also we can see that there is no much change in the validation accuracy with number of epochs , so no need to go for upto 15 epochs , 5-7 epochs should be good enough

Question 4 (10 Marks)

Paste a link to your github code for Part A

Example: https://github.com/<user-id>/cs6910_assignment2/partB

Follow the same instructions as in Question 6 of Part A.

https://github.com/theindianwriter/CS6910-assignment_2/blob/main/DL_Assign_2_PartB_withWandB.ipynb

https://github.com/theindianwriter/CS6910-assignment_2/blob/main/DL_Assign_2_PartB.ipynb

Created with ❤️ on Weights & Biases.

https://wandb.ai/cs20m038/DL_Assign_2B/reports/Assignment-2-Part-B--Vmlldzo2MTM5MTY

Assignment 2 Part C

Object Detection using YoloV3

Arjun Kumar Gupta

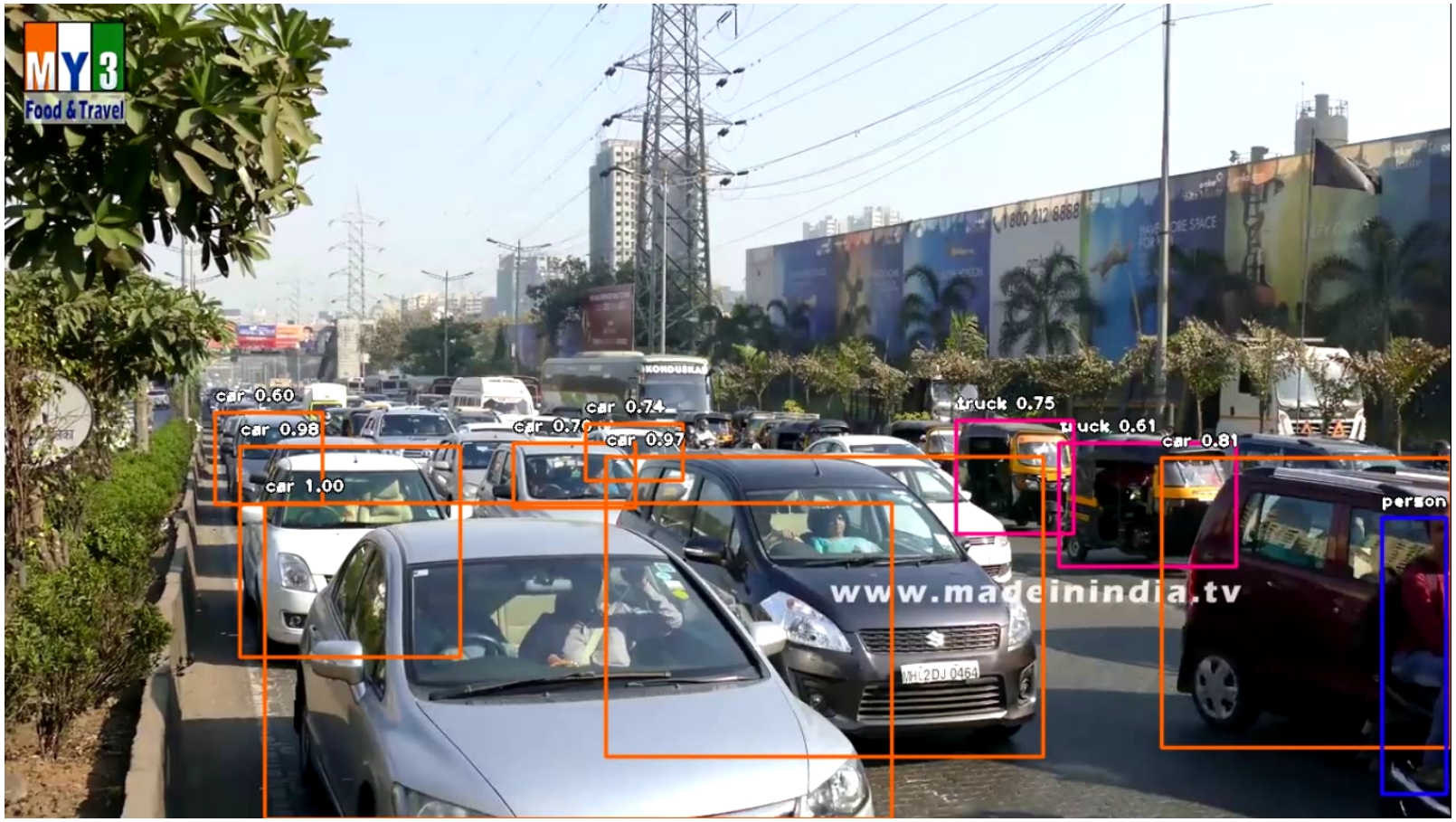
Arjun Kumar Gupta, Milan Chatterjee

Ans 1.

> YoloV3 model has been used to detect objects especially vehicles in this application.In our day to day life traffic plays an important role and traffic analysis is one of the ways in which a traffic can be handled.A key component of traffic analysis is detecting the amount of cars that pass certain points.This can be done once all the vehicles are detected in a frame which is done by this application. The code of the application has been inspired by ImageAI Object Detection model which has been build over YoloV3 model. Some snippets of the demo video has been pasted below :


MY 3
Food & Travel
car 0.97
car 0.61
car 0.97
car 1.00
car 0.96
car 0.94
car 0.67
car 0.51
traffic_analysis_video.avi


MY 3
Food & Travel
car 0.67
car 0.86
car 0.99
car 0.99
car 0.64
person 0.51
car 0.94
car 0.99
www.madeinindia.tv



> The application can also detect vehicles from a live street cam and thus this could be used to also detect vehicles which are breaking traffic rules and a snapshot of their number plate can be captured.

The link of a demo video of our application is given below:

https://drive.google.com/file/d/1vORb2Kyps5jmVmEaI8ruemVbrWiYod53/view?usp=sharing

> The github link is given here:
> https://github.com/theindianwriter/CS6910-assignment_2

Self Declaration:

CS20M015 (55 %) - Arjun Kumar Gupta

- In part A loading,preprocessing the images and making them ready for training.
- Implementation of Model, designing as well as code.
- Tuning the hyper parameters with the help of train set.
- Configuring wandb sweep for Part A and Part B.
- Analysing the results of part A and saving, loading the best model in the drive.
- Testing the model performance on the test set and doing visualization.
- In part C changing some portions of the code to suit our application.
- Preparing report for part A and part C.
- Helping to debug in part A.
- 80 % in Part A, 20 % in part B, 55% in part C

CS20M038 (45 %) - Milan Chatterjee

- In part B loading, preprocessing of images and making them ready for training against different models.
- Researching over different pretraincd models and implementing to use all of them.
- Tuning the hyper parameters and making comparisons between different models in Part B
- Helping in debugging of part A.
- Observing the sweep results and making insightful interpretations of the results.
- Testing the different models in part B with test dataset and reporting accuracy.
- Researching and giving ideas for development of application in part C.
- Generating the demo video in Part C.
- Making Report for Part B.

- 80 % in part B, 45 % in part C, 20 % in part A

> We, Arjun Kumar Gupta CS20M015 and Milan Chatterjee CS20M028, swear on our honour that the above declaration is correct.

The github link for all the parts of the assignemt is :

https://github.com/theindianwriter/CS6910-assignment_2

The number of cheat days used is 5 days 18 hours

Created with ❤️ on Weights & Biases.

https://wandb.ai/theindianwriter/cs6910-assignment2-part_a/reports/Assignment-2-Part-C--Vmlldzo2MTQ4NDc