Edit Share

CS6910 - Assignment 2

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

Purvam Jain

Question 1 (5 Marks)

a) The number of computations done per image for a generic CNN with 5 layers with m filters of size $k \times k$ and n neurons in the dense layer can be computed as follows. Assume an initial image size of $d \times d$. Assume no padding and stride of 1. We count the number of multiplications only.

1. Multiplications in convolution between filters and feature maps:

- Layer 1: $m \times k \times k \times 3 \times (d-k+1) \times (d-k+1)$
- Layer 2: $m \times k \times k \times m \times (\frac{d-k+1}{2} - k + 1) \times (\frac{d-k+1}{2} - k + 1)$
- Layer 3: $m \times k \times k \times m \times (\frac{\frac{d-k+1}{2} - k + 1}{2} - k + 1) \times (\frac{\frac{d-k+1}{2} - k + 1}{2} - k + 1)$
- Layer 4: $m \times k \times k \times m \times (\frac{\frac{\frac{d-k+1}{2} - k + 1}{2} - k + 1}{2} - k + 1) \times (\frac{\frac{\frac{d-k+1}{2} - k + 1}{2} - k + 1}{2} - k + 1)$
- Layer 5: $m \times k \times k \times m \times (\frac{\frac{\frac{\frac{d-k+1}{2} - k + 1}{2} - k + 1}{2} - k + 1}{2} - k + 1) \times (\frac{\frac{\frac{\frac{d-k+1}{2} - k + 1}{2} - k + 1}{2} - k + 1}{2} - k + 1)$

2. Multiplications in the dense layers:

The flattened output of the convolution-relu-pooling layers has dimension $f^2 \times 1$ where:

$$f = \frac{\frac{\frac{\frac{d-k+1}{2}-k+1}{2}-k+1}{2}-k+1}{2}$$

The number of multiplications for $W_{m\times n}x_{n\times 1}$ is $m \times n$ and so the multiplications for the dense layer are $n \times f^2$ and for the output layer are $n \times 10$.

b) The number of parameters in the activation layers is zero. Only convolution and dense layers have parameters.

- **Convolution Layers**:

Layer 1 has $m \times k \times k \times 3$ parameters while other 4 convolution layers have $m \times k \times k \times m$ parameters.

- **Dense Layers**:

The dense layer weights are of size $n \times f^2$ and the biases are of size $n \times 1$, where f is the same as above.

There are 10 classes so the output layer weights are of size $10 \times n$ and the biases are of size 10×1.

Question 2 (15 Marks)

Some Fixed Parameters

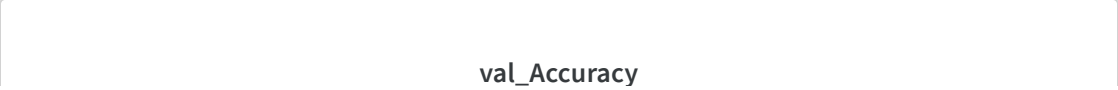
These parameters were fixed based on prior knowledge and experimental information.

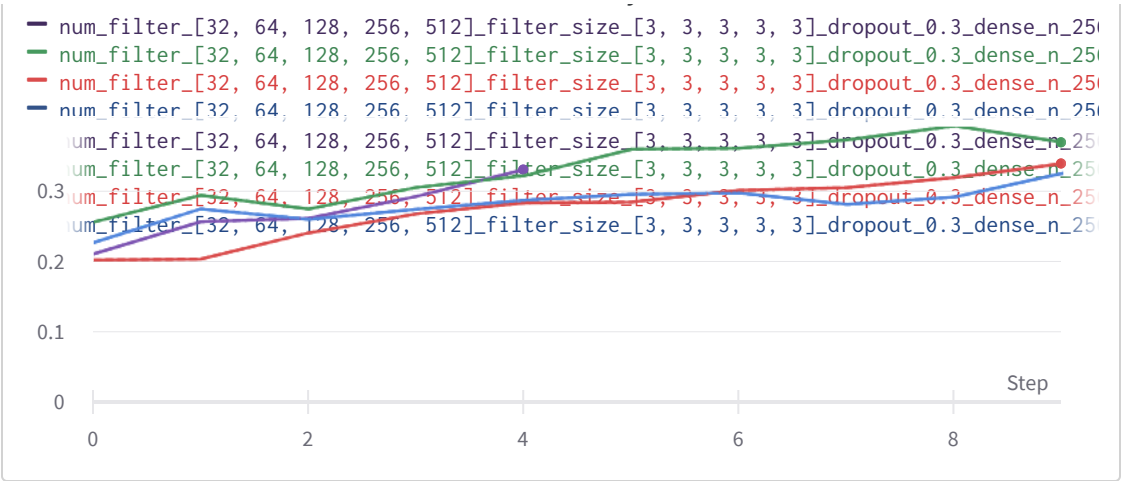
- Optimizer: Adam
- Learning Rate: 1e-3
- epochs: 10
- Loss Function: CrossEntropyLoss()

- Input Image size: 128*128
- Batch Size: 64

BatchNorm vs Data Augmentation

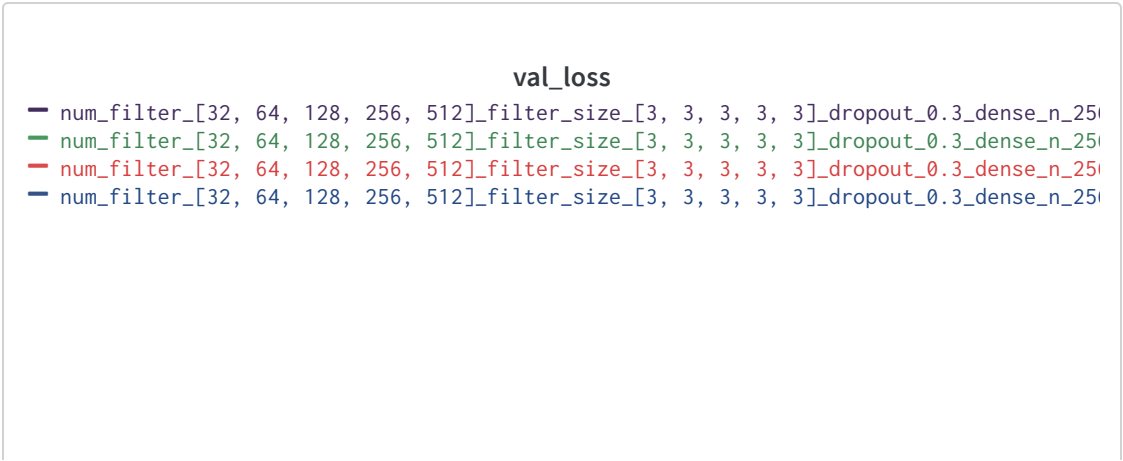
- First we only evaluate these two hyperparameters since these are independent of other hyperparameters (Note that last run couldn't complete so I made the observation based on just 5 epochs.) Here all other hyperparameters were set to be constant, as BatchNorm is applied to reduce training time and along data augmentation technique it is used to regularize to counter overfitting. They are independent of the other parameters as filter size and activation functions. From the graphs below we observe that **BatchNorm set to True with data augmentation set to false performs the best.**

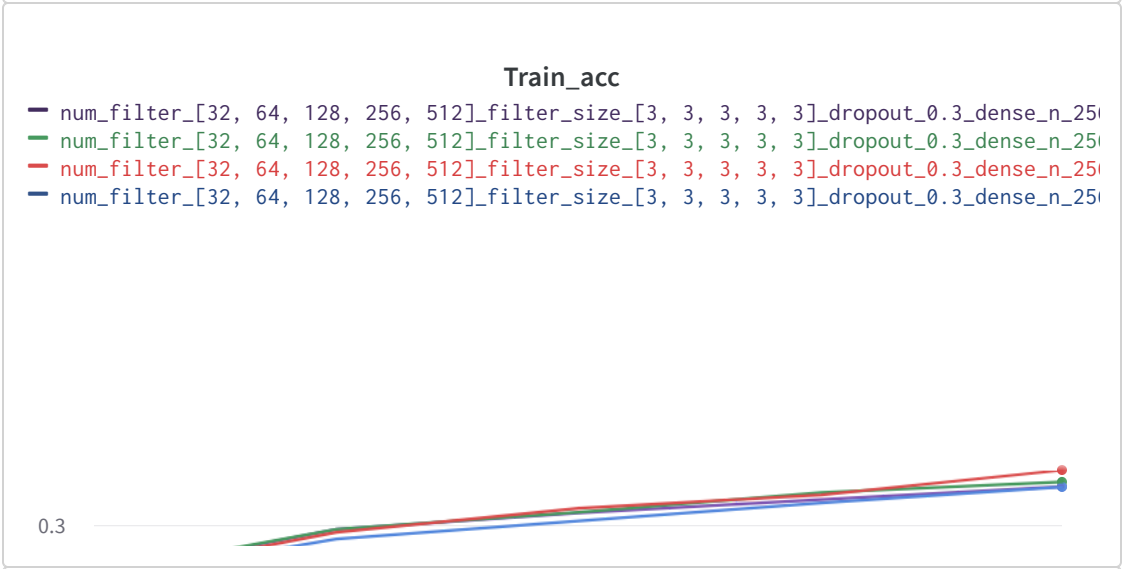


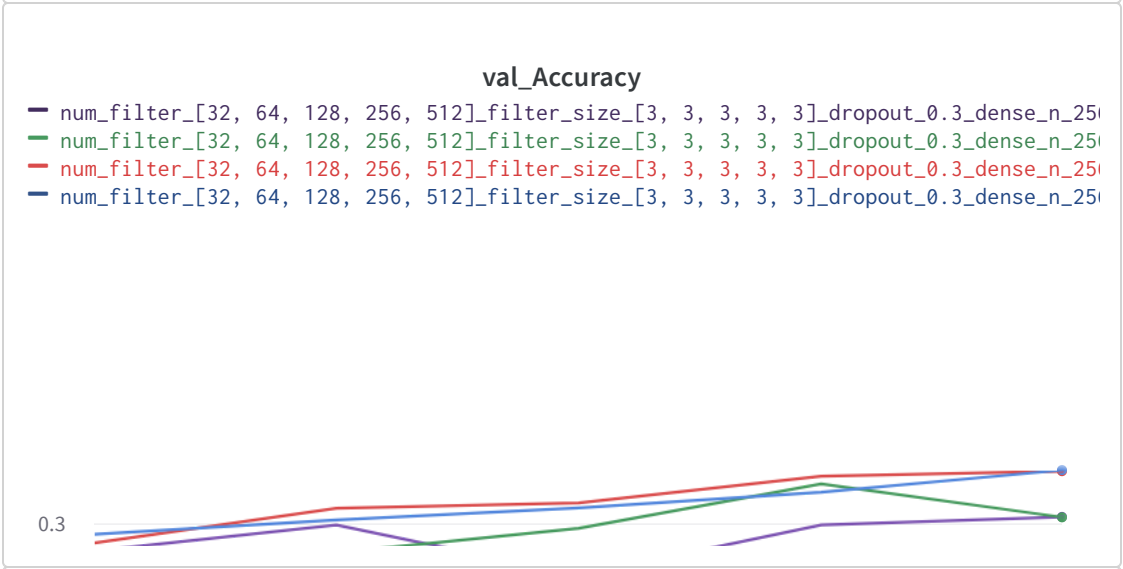


Activation Functions

- We experiment with ReLU, LeakyReLU, SiLU and Mish. They were all trained for 5 epochs and Mish seems to have outperformed all other functions with a slight advantage on Validation Loss while it is comparable to SiLU on validation accuracy. **Hence we select Mish activation function for further training.**





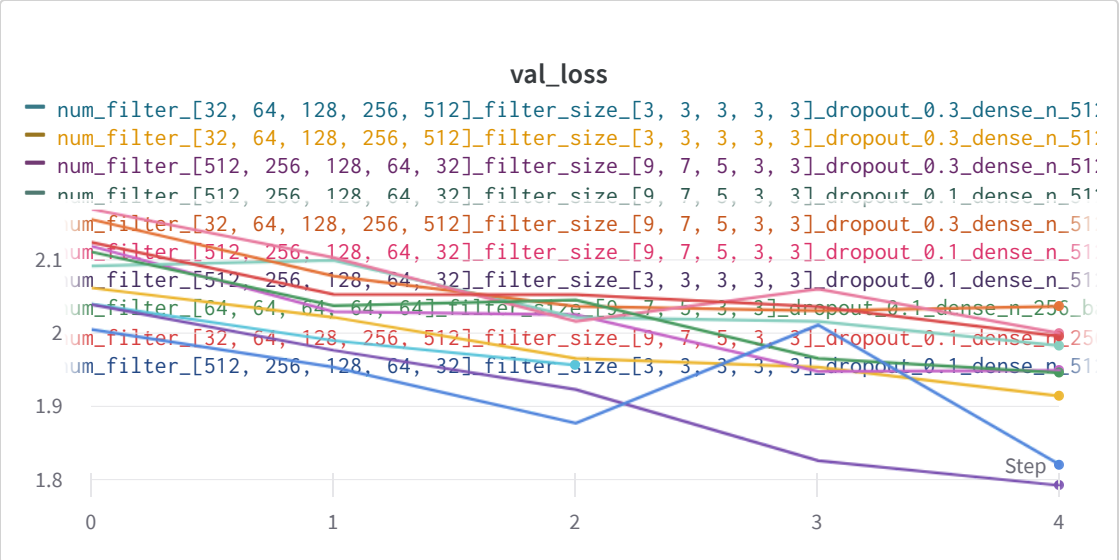


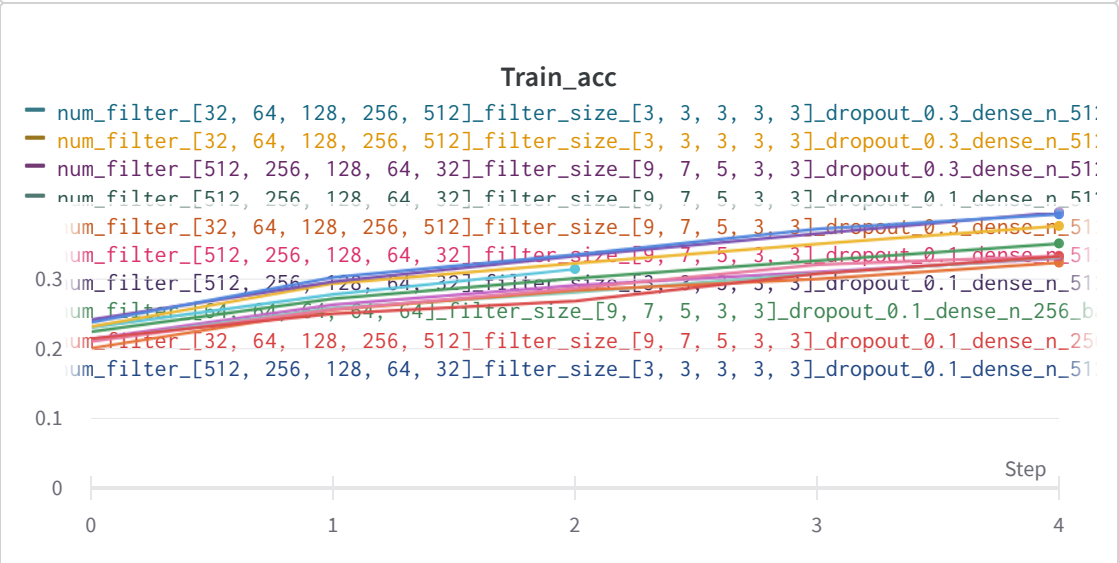
▾ Network Architecture

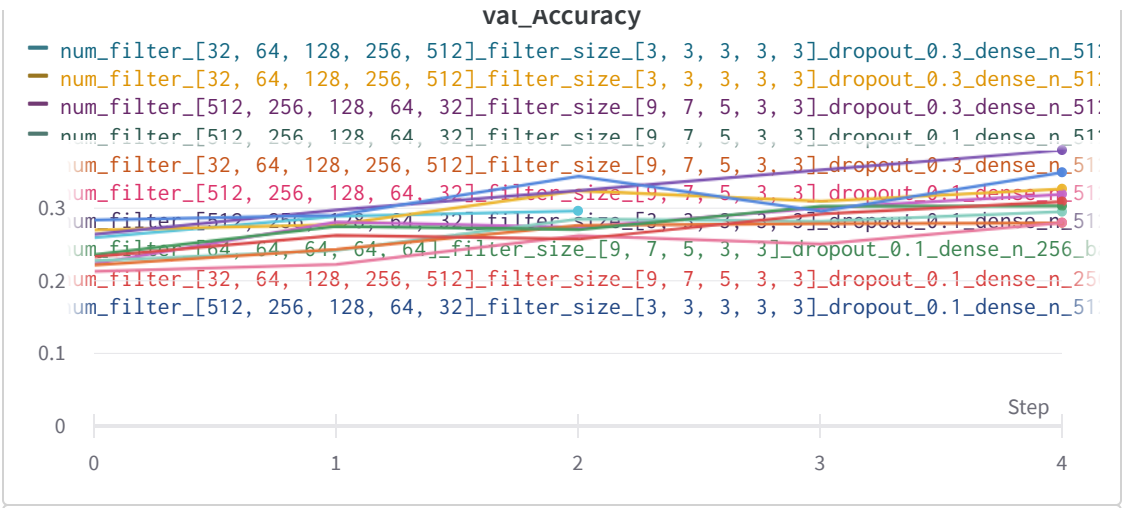
- Number of filters and their organization : [64,64,64,64,64], [512,256,128,64,32], [32,64,128,256,512]
- Dense Layer Size: 256, 512
- Dropout: 0.0, 0.1, 0.3
- Kernel Size: [3,3,3,3,3], [9,7,5,3,3]

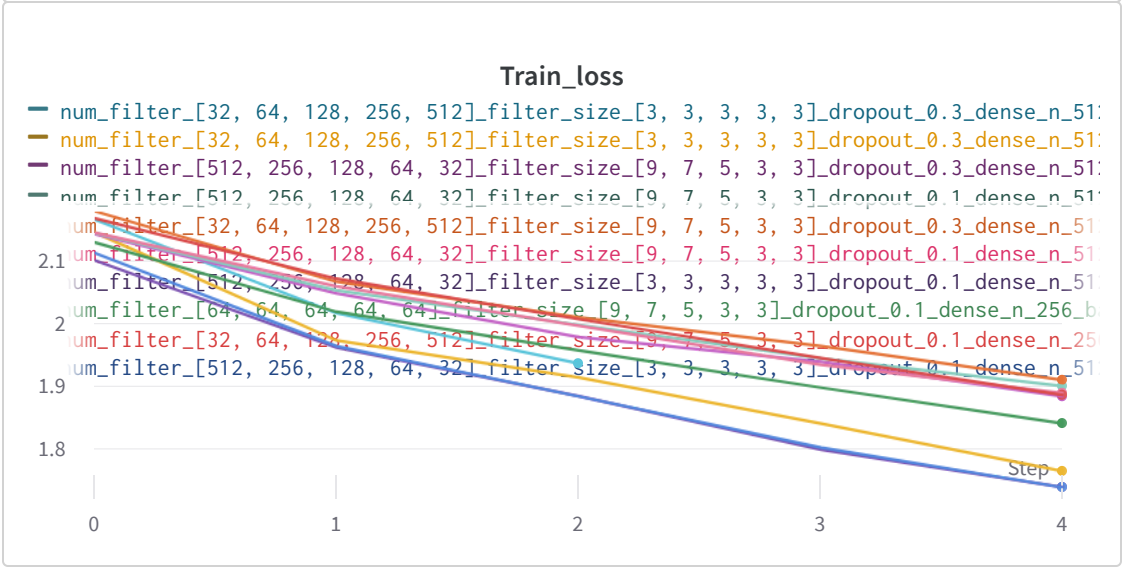
Upon running bayes sweep with the goal to reduce validation loss for 10 iterations following hyperparameters seemed to perform the best:

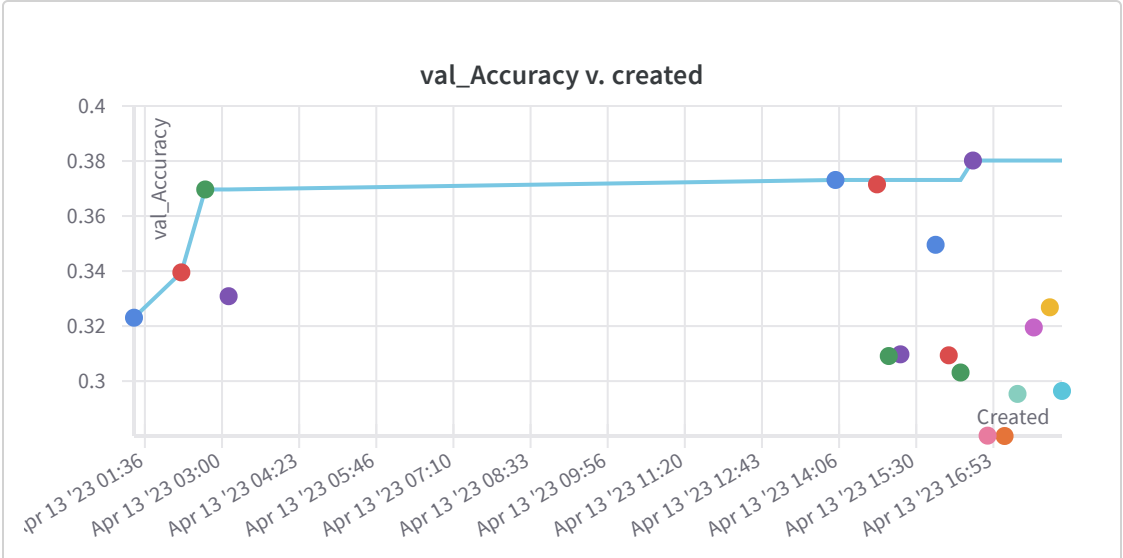
- Number of filters: [32,64,128,256,512]
- Dense Layer Size: 512
- Dropout: 0.1 (Dropout layer is added between the last two dense layers although different variations are possible)
- Kernel Size: [3,3,3,3,3]

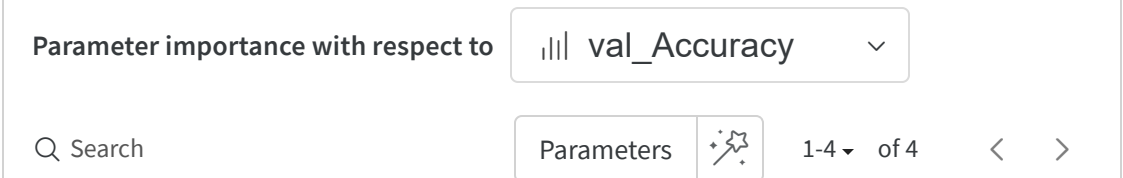




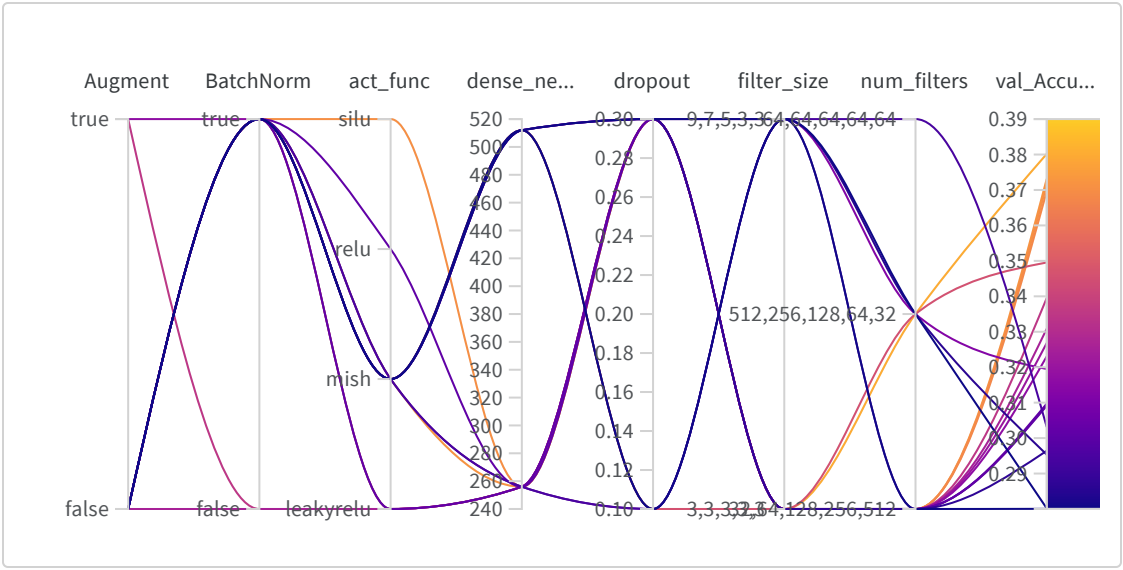

val_Accuracy
num_filter_[32, 64, 128, 256, 512]_filter_size_[3, 3, 3, 3, 3]_dropout_0.3_dense_n_51
num_filter_[32, 64, 128, 256, 512]_filter_size_[3, 3, 3, 3, 3]_dropout_0.3_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[9, 7, 5, 3, 3]_dropout_0.3_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_51
num_filter_[32, 64, 128, 256, 512]_filter_size_[9, 7, 5, 3, 3]_dropout_0.3_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[3, 3, 3, 3, 3]_dropout_0.1_dense_n_51
num_filter_[64, 64, 64, 64, 64]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_256_b
num_filter_[32, 64, 128, 256, 512]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_25
num_filter_[512, 256, 128, 64, 32]_filter_size_[3, 3, 3, 3, 3]_dropout_0.1_dense_n_51
0.3
0.2
0.1
0
Step
0
1
2
3
4


Train_loss
num_filter_[32, 64, 128, 256, 512]_filter_size_[3, 3, 3, 3, 3]_dropout_0.3_dense_n_51
num_filter_[32, 64, 128, 256, 512]_filter_size_[3, 3, 3, 3, 3]_dropout_0.3_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[9, 7, 5, 3, 3]_dropout_0.3_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_51
num_filter_[32, 64, 128, 256, 512]_filter_size_[9, 7, 5, 3, 3]_dropout_0.3_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_51
num_filter_[512, 256, 128, 64, 32]_filter_size_[3, 3, 3, 3, 3]_dropout_0.1_dense_n_51
num_filter_[64, 64, 64, 64, 64]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_256_b
num_filter_[32, 64, 128, 256, 512]_filter_size_[9, 7, 5, 3, 3]_dropout_0.1_dense_n_25
num_filter_[512, 256, 128, 64, 32]_filter_size_[3, 3, 3, 3, 3]_dropout_0.1_dense_n_51
2.1
2
1.9
1.8
Step
0
1
2
3
4


val_Accuracy v. created
0.4
0.38
0.36
0.34
0.32
0.3
val_Accuracy
Created
Apr 13 '23 01:36
Apr 13 '23 03:00
Apr 13 '23 04:23
Apr 13 '23 05:46
Apr 13 '23 07:10
Apr 13 '23 08:33
Apr 13 '23 09:56
Apr 13 '23 11:20
Apr 13 '23 12:43
Apr 13 '23 14:06
Apr 13 '23 15:30
Apr 13 '23 16:53


Parameter importance with respect to
val_Accuracy
Search
Parameters
1-4
of 4

Config parameter	Importance	Correlation
dropout		
dense_neurons		
BatchNorm		



Question 3 (15 Marks)

Based on the above plots write down some insightful observations. For example,

- Best filter organization strategy is doubling the number of filters instead of keeping it constant or halving number of filters which quite intuitive as is also clear from observing the state of the art models.
- For, size of filters, a constant size of (3,3) instead of decreasing filter size like other ImageNet models seems to work better. This could be due to small size of image and smaller network size.
- BatchNorm improves the performance as expected since it regularizes the weights for each layer.

- Data Augmentation seems to degrade the performance, because our network is small , data augmentation introduces regularization which rather proves to be counter effective. It maybe due to use of many transformations, but network might perform well with limited transformations as is visible from correlation table.
- Dropout seems to improve the validation set performance, this might be due to overfitting in the dense layers which is further visible from the negative correlation between validation loss and dense neurons.
- The current CNN architecture is not enough to capture the complexity of the iNaturalist dataset as at best we reach accuracy close to 40%. But if we train for a higher number of epochs, then we must introduce Data Augmentation to regularize the learning process, but still not much improvement is observed in the validation accuracy.

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).

The best model performed as : Test Loss : 2.388373 Test Acc : 0.406738

Amphibia

Animalia

Predicted label: Animalia

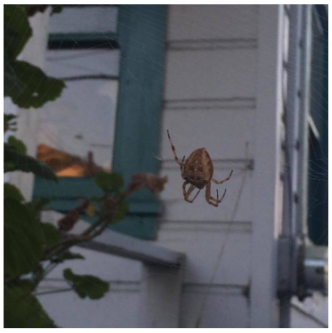
Predicted label: Fungi

Predicted label: Animalia

Arachnida

Predicted label: Arachnida

Predicted label: Animalia



Predicted label: Reptilia

Aves

Predicted label: Aves

Predicted label: Insecta

Predicted label: Aves

Fungi

Predicted label: Fungi

Predicted label: Reptilia

Predicted label: Mammalia

Insecta

Predicted label: Insecta

Predicted label: Insecta

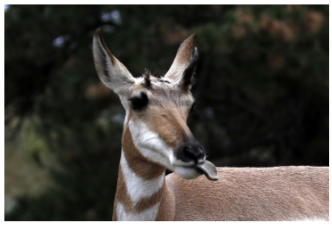
Predicted label: Insecta

Mammalia

Predicted label: Aves

Predicted label: Mammalia

Predicted label: Aves



Mollusca

Predicted label: Mammalia

Predicted label: Aves

Predicted label: Animalia

Plantae

Predicted label: Plantae

Predicted label: Aves

Predicted label: Plantae

Reptilia

Predicted label: Reptilia

Predicted label: Arachnida

Predicted label: Mammalia

Question 5 (10 Marks)

Paste a link to your github code for Part A

Example: https://github.com/wigglytuff-tu/CS6910_Assignment_2/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?

Answer: We can resize the input images to the the same size as is expected by these models, using transforms from torchvision. In general Imagenet models are trained on images of size (224,224,3). But, I trained InceptionetV3 which expects input image size to be (299,299,3).

- 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?

Answer: For this we delete the last dense layer and add another dense layer in its place with output features as 10 .

(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 `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).

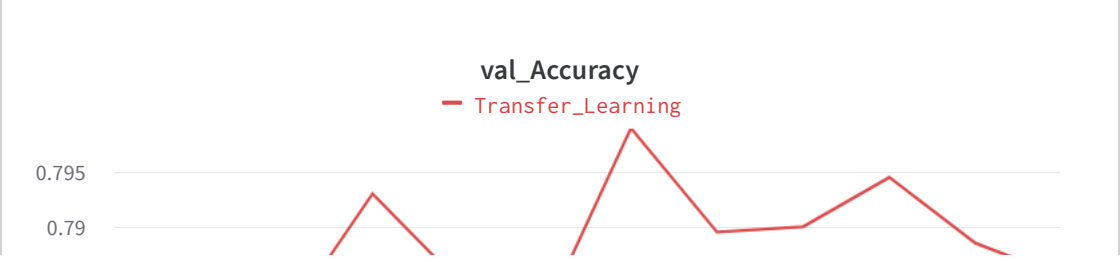
- Freezing all layers except for the last newly added layer with desired output features and training only that layer
- Freezing k initial layers as they might be general image features and training rest of the layers
- Freezing last layer, and then after training the last layer we can finetune or train rest of the layers

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.

I trained InceptionetV3 using the first strategy mentioned above.

- Pretrained models perform much better than models from scratch, this can be attributed to excellent feature extractors present in these models as they are trained on a large dataset of images , which also contains many animals and plants.
- They require a lot of time to train, even when training only the last dense layer.
- For scratch model validation accuracy plateaus at 40% while here it reaches 80%
- Better performance can be achieved by performing finetuning after training the dense layer



Question 4 (10 Marks)

Paste a link to your GitHub code for Part B

Example: https://github.com/wigglytuff-tu/CS6910_Assignment_2/tree/main/PartB

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

Self Declaration

I, Purvam Jain (Roll no: EE20B101), 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.

Created with ❤️ on Weights & Biases.

https://wandb.ai/purvam/CS6910-Assignment-2/reports/CS6910-Assignment-2--Vmlldzo0MDQ2Nzcy