CSC420 A2

Ziang Chen 1004828449

October 28th, 2022

Q1:

Q2

We take X_i to be the output at ith layer

W_i be the weight of ith layer,

b_i be the bias

W_σ, b_σ be the scalars of the activation function

Observe that after $n+1$ layers,

$$
\begin{aligned}
X_{n+1} &= \sigma(W_n \circledast X_n + b_n) \\
&= W_\sigma \circledast (W_n \circledast X_n + b_n) + b_\sigma \\
&= (W_\sigma \circledast W_n) X_n + W_\sigma \circledast b_n + b_\sigma \\
&= W' \circledast X_n + b'
\end{aligned}
$$

Notice since the activation function is linear,

X_{n+1} can still be expressed as a linear combination of the input X_0, which means that the layers had no effect to the network output.

Q4: Convolutional Neural Network

Each filter in the filter bank is of size 4 x 4 x 50, We see that applying the filter requires 4 x 4 x 50 multiplications and (4 x 4 x 50) - 1 additions therefore 800 + 799 = 1599 FLOPS.

We know that the output size is (12 + (2 x 1) - 4) / 2 + 1 = 6 x 6

So each filter will have to be applied 36 times which leads to 36 x 1599 = 57564 FLOPS

Since there are 20 filters in the filter bank, we have 20 x 57564 = 1151280 FLOPS

Knowing the output size is 6 x 6 x 20, we see that the output size of max pooling is 6-3/1 + 1 = 4 x 4 x 20 = 320 times applying max pool, with each max pooling operation taking 3 x 3 - 1 = 8 FLOPS.

The total amount of FLOPS would be 1151280 + (320 x 8) = 1153840 without bias and 1153840 + (6 x 6 x 20) = 1154560 FLOPS with bias.

Q5

We know that the kernel size is 5x5

C1: (5 x 5 x 1 + 1) x 6 = 156

C3: (5 x 5 x 6 + 1) x 16 = 2416

C5: (5 x 5 x 16 + 1) x 120 = 48120

C6: (120 + 1) x 84 = 10164

C7: (84 + 1) x 10 = 850

Total trainable parameters: 61706

Q6

Q6

$$y = \frac{1}{1+e^{-x}}$$

$$\frac{dy}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1+e^{-x} - 1}{(1+e^{-x})^2}$$

$$= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

$$= y - y^2 = y(1-y)$$

Therefore, the input is not required for back propagation

Q7

a) $\tanh \in (0,1)$

$\sigma(x) \in (0,1)$

b) $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$

$\sigma(x) = \frac{1}{1+e^{-x}}$

$\sigma(2x) = \frac{1}{1+e^{-2x}}$

$2\sigma(2x) = \frac{2}{1+e^{-2x}}$

$2\sigma(2x) - 1 = \frac{1-e^{-2x}}{1+e^{-2x}} = \tanh(x)$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$
$$= 1 - [2\sigma(2x) - 1]^2$$
$$= 4\sigma(2x) - 4\sigma^2(2x)$$

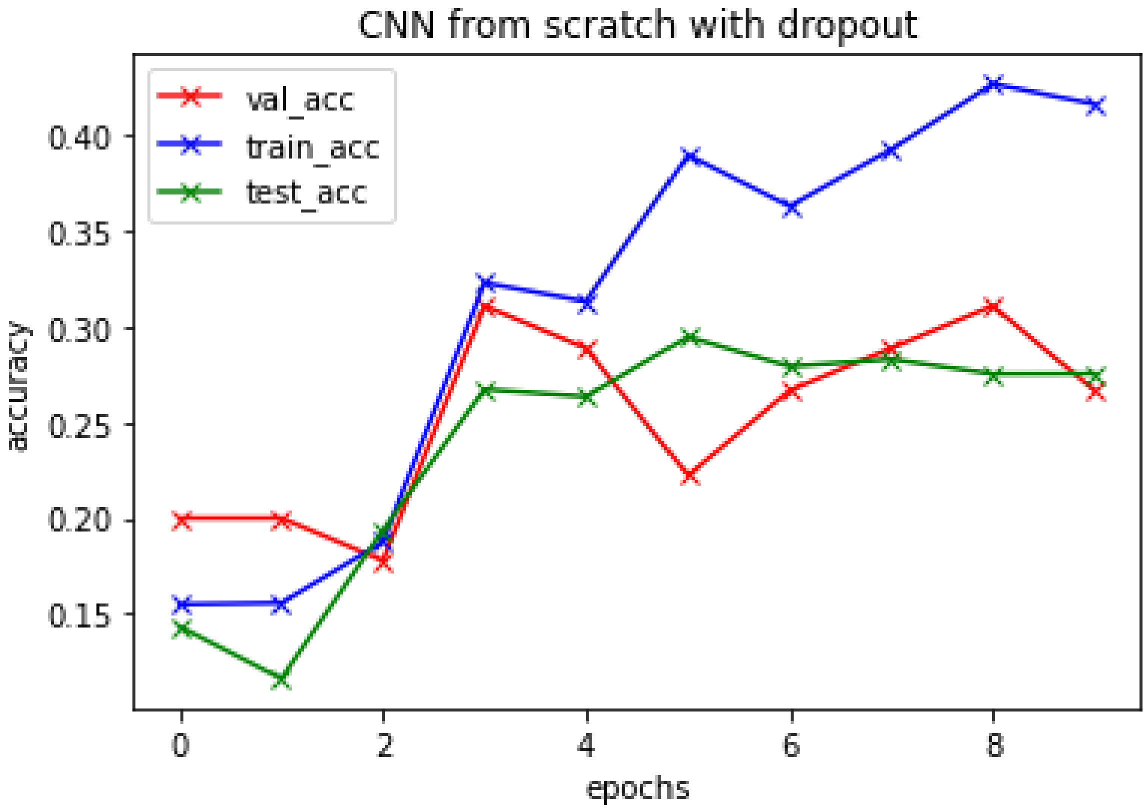
c) tanh's range helps models that output binary values, where as sigmoid is commonly used to models that output probabilities.

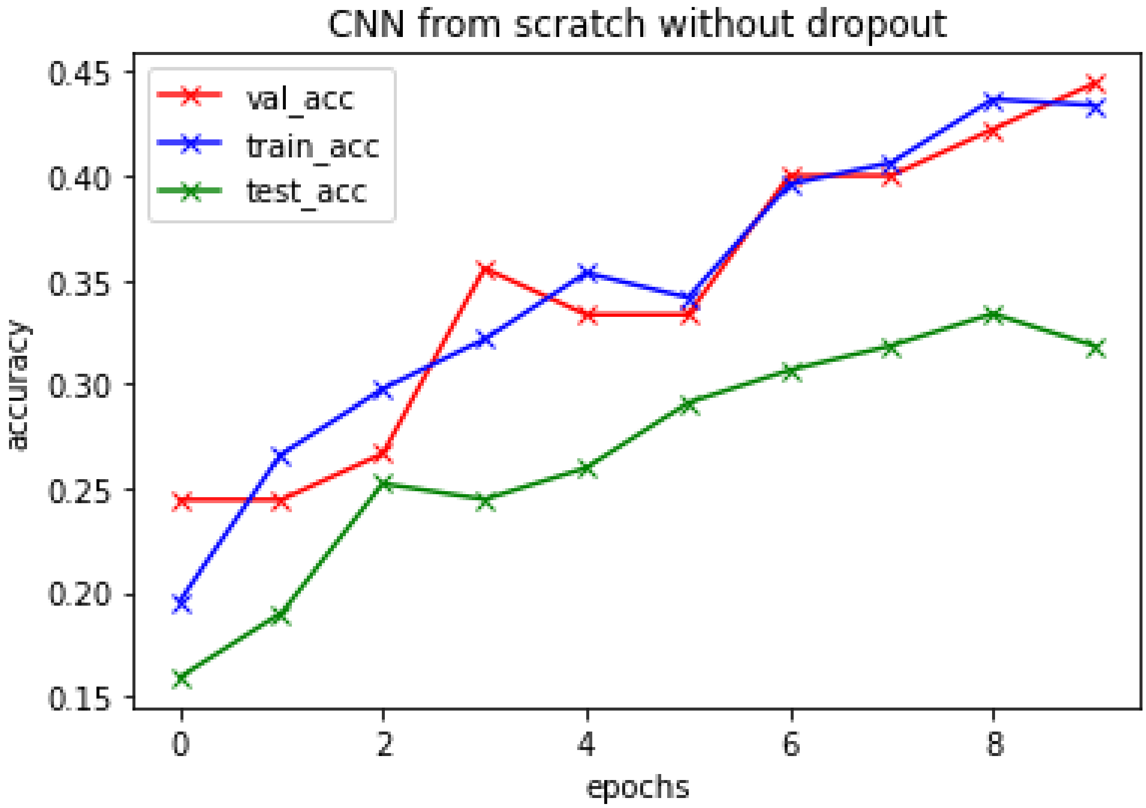
Part 2: Task 1

The two different datasets mainly differ in that DBI focuses solely on the animal itself while SDD occasionally has other objects(eg. human) within the image. This provides more variety and difficulty in the SDD dataset.

Part 2: Task 2

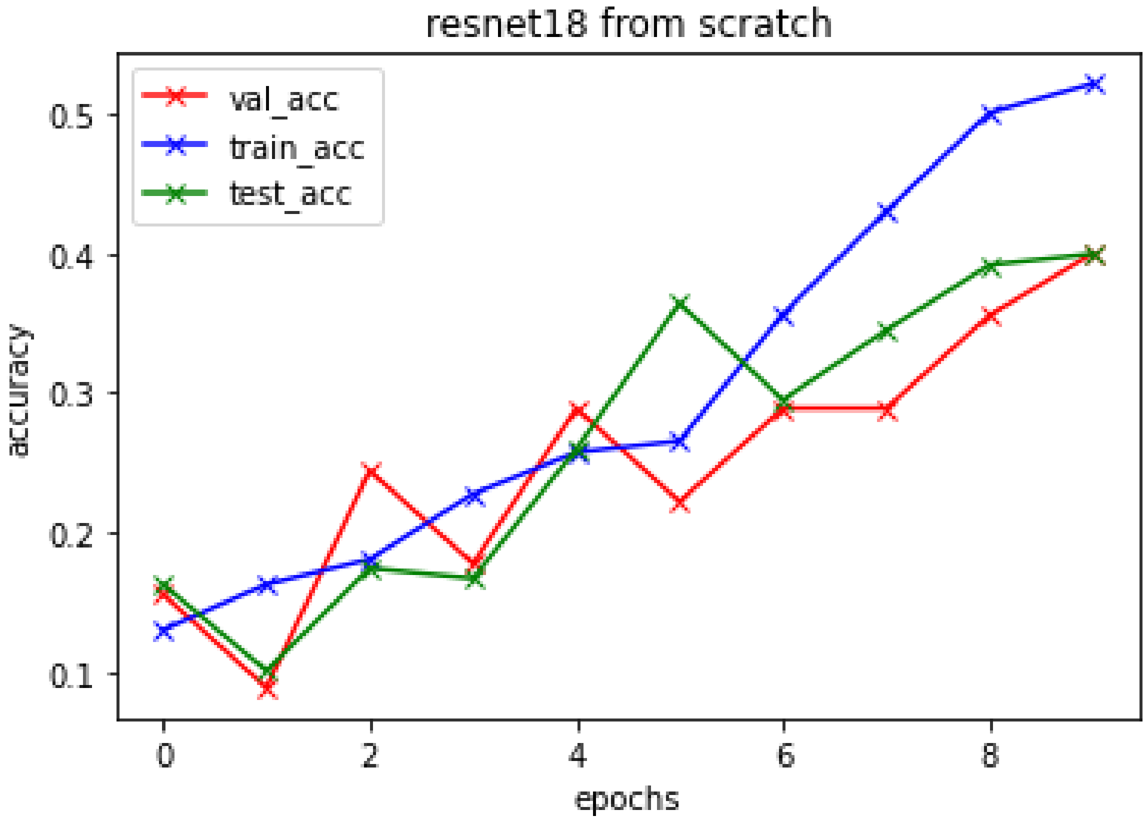
This is based on the started code given in: https://medium.com/@ankitvashisht12/classifying-dog-breed-using-pytorch-abc9f3c5128a




CNN from scratch without dropout
val_acc
train_acc
test_acc
accuracy
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
2
4
6
8
epochs

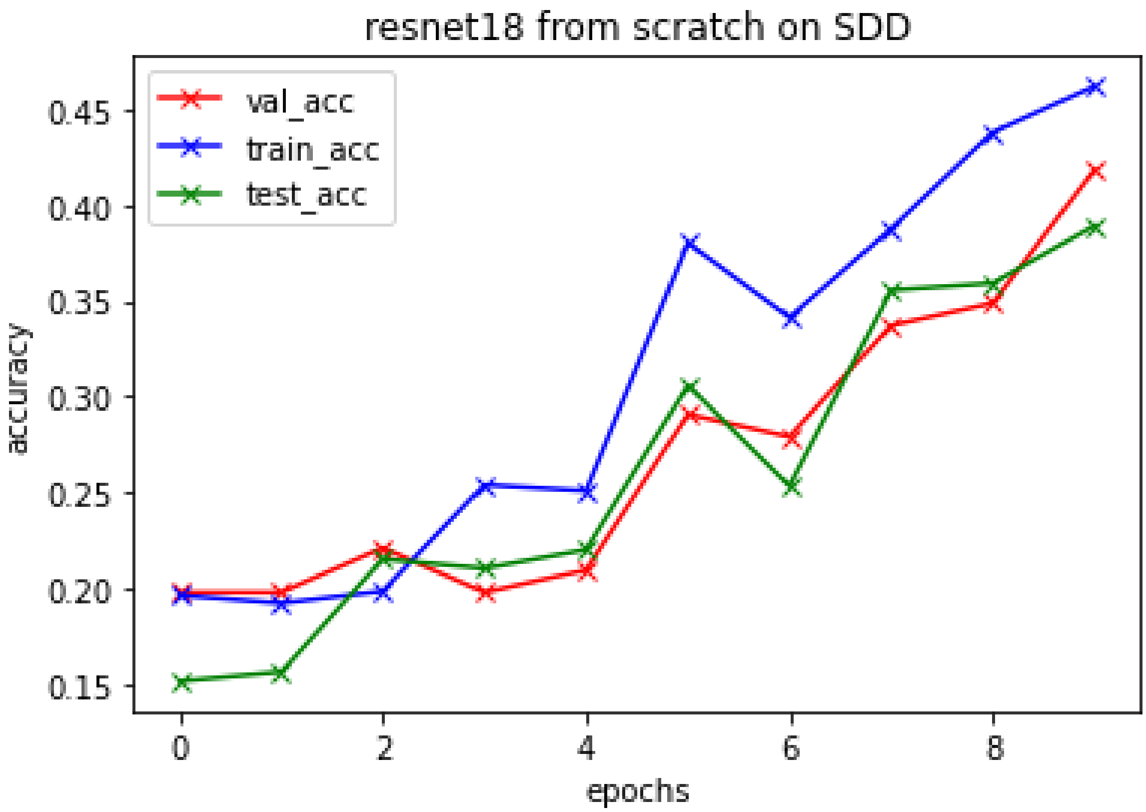
Part 2: Task 3

3a:



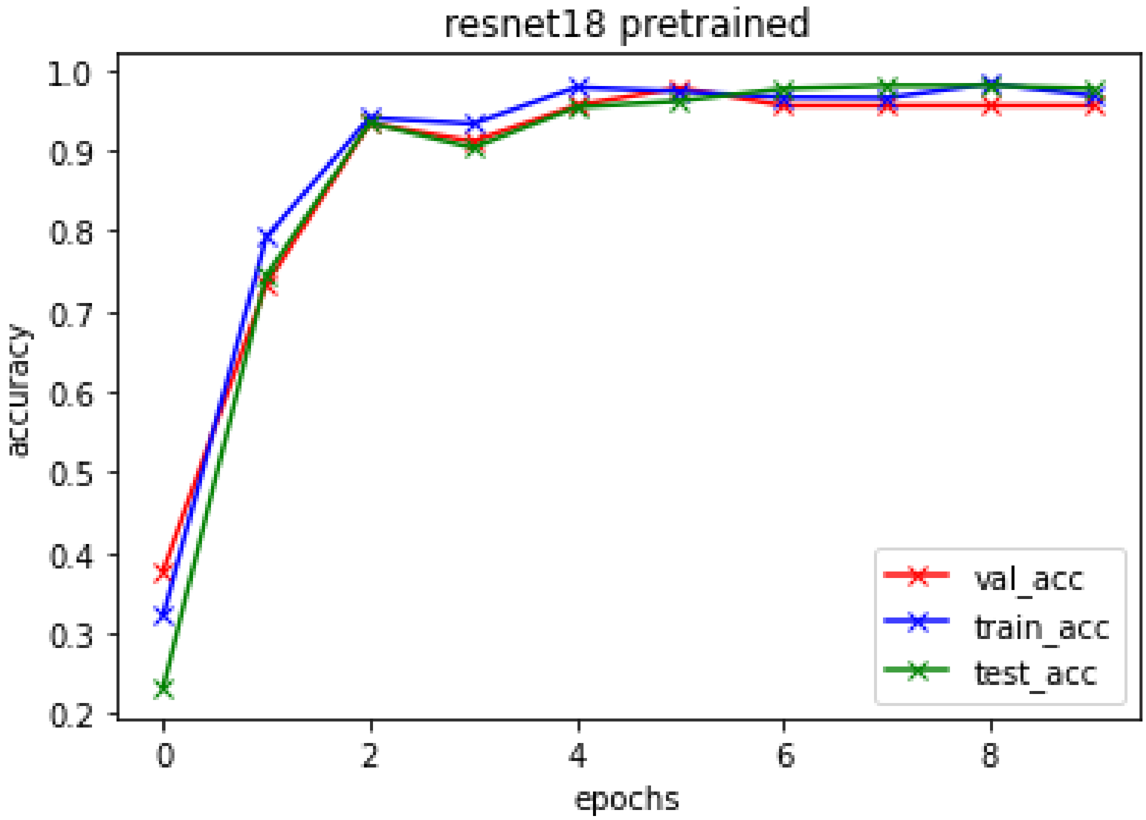
This model showed much higher accuracy compare to our custom CNN, one of the possible reasons why could be that it is much more complex in terms of the model size and trainable parameters.

3b:

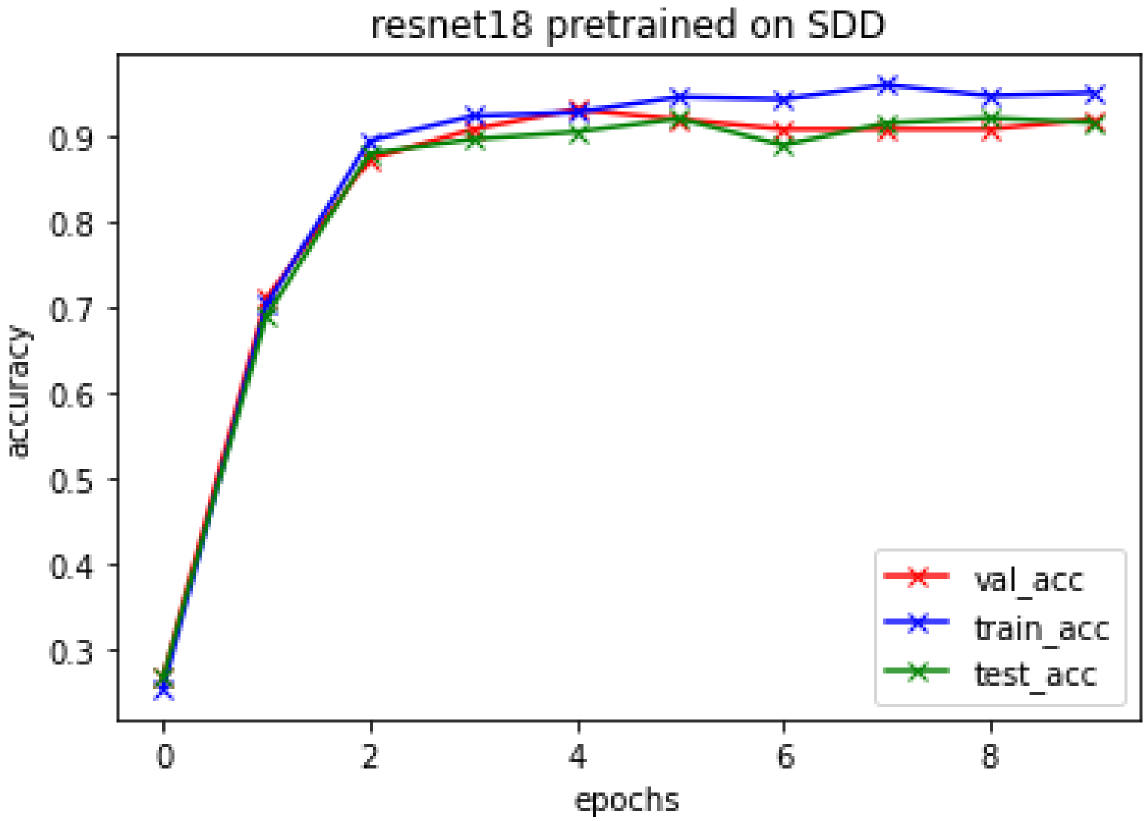


The model tapered off at around 40% of accuracy on both datasets, but DBI ended up being higher, this could be that the DBI dataset is less complicated in the objects included as we previously mentioned.

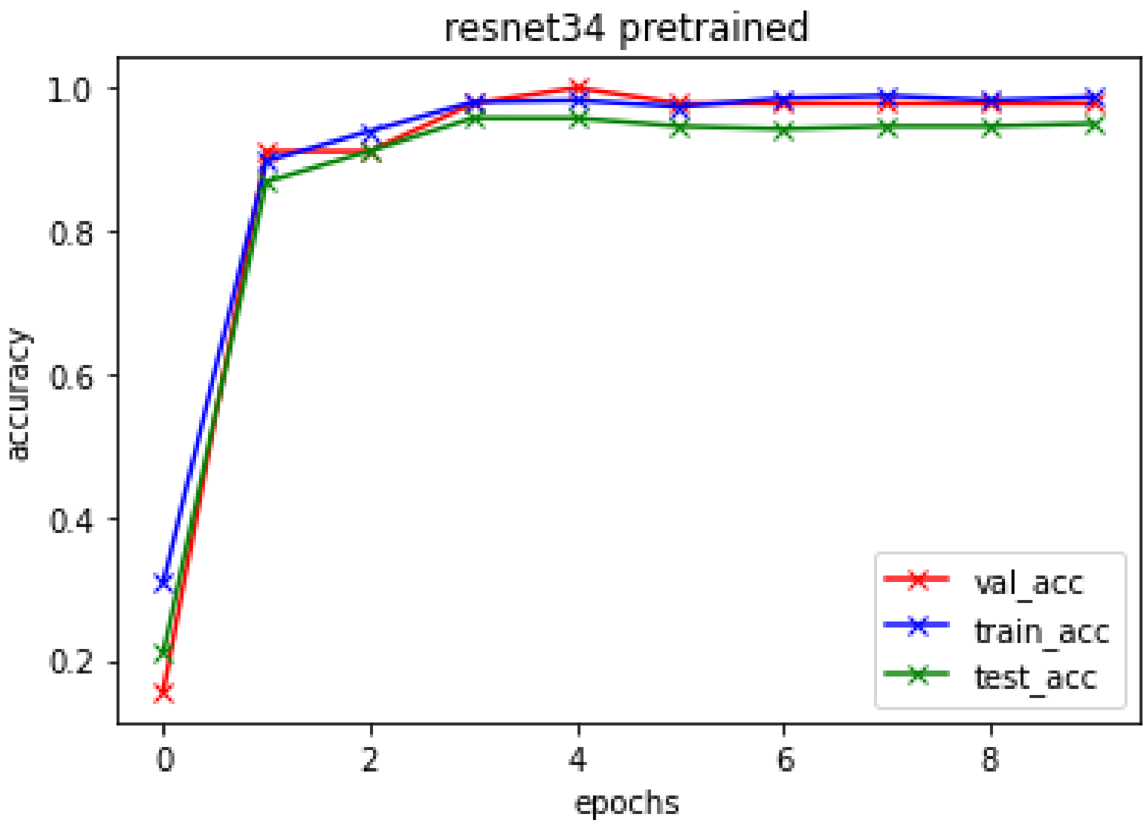
Part 2: Task 4



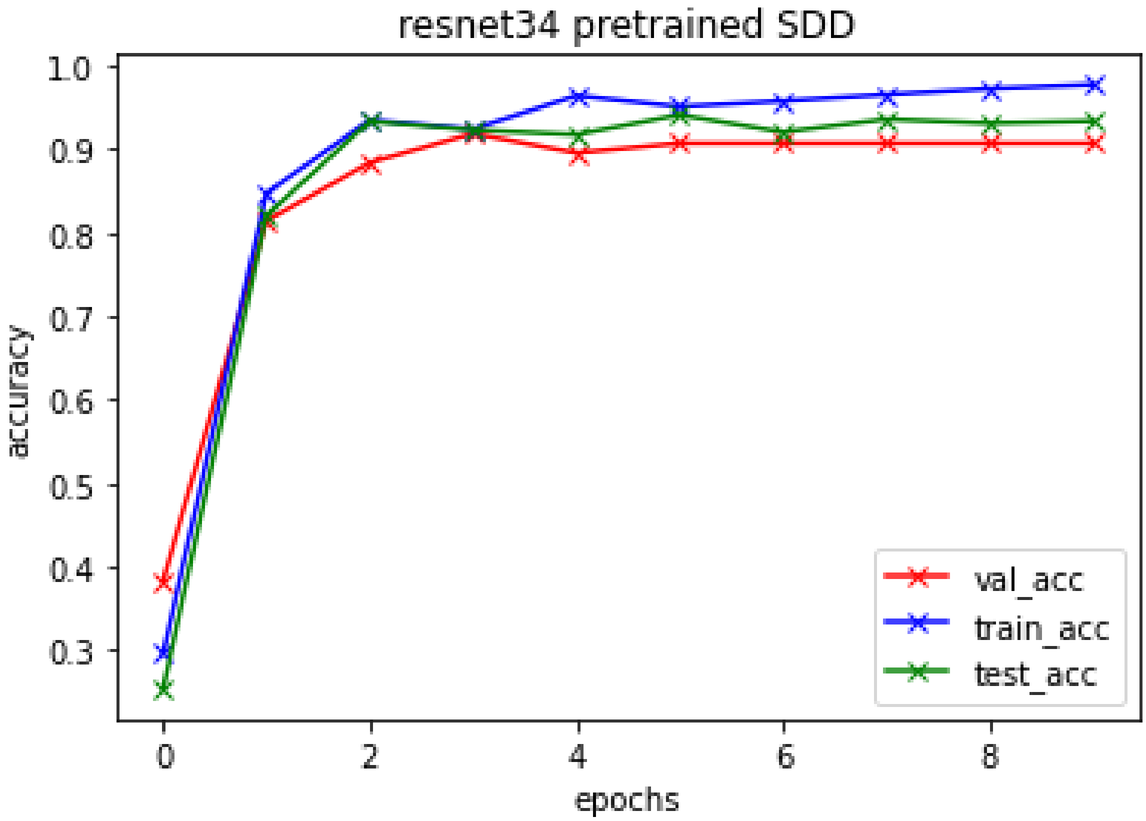
test acc = 0.9827



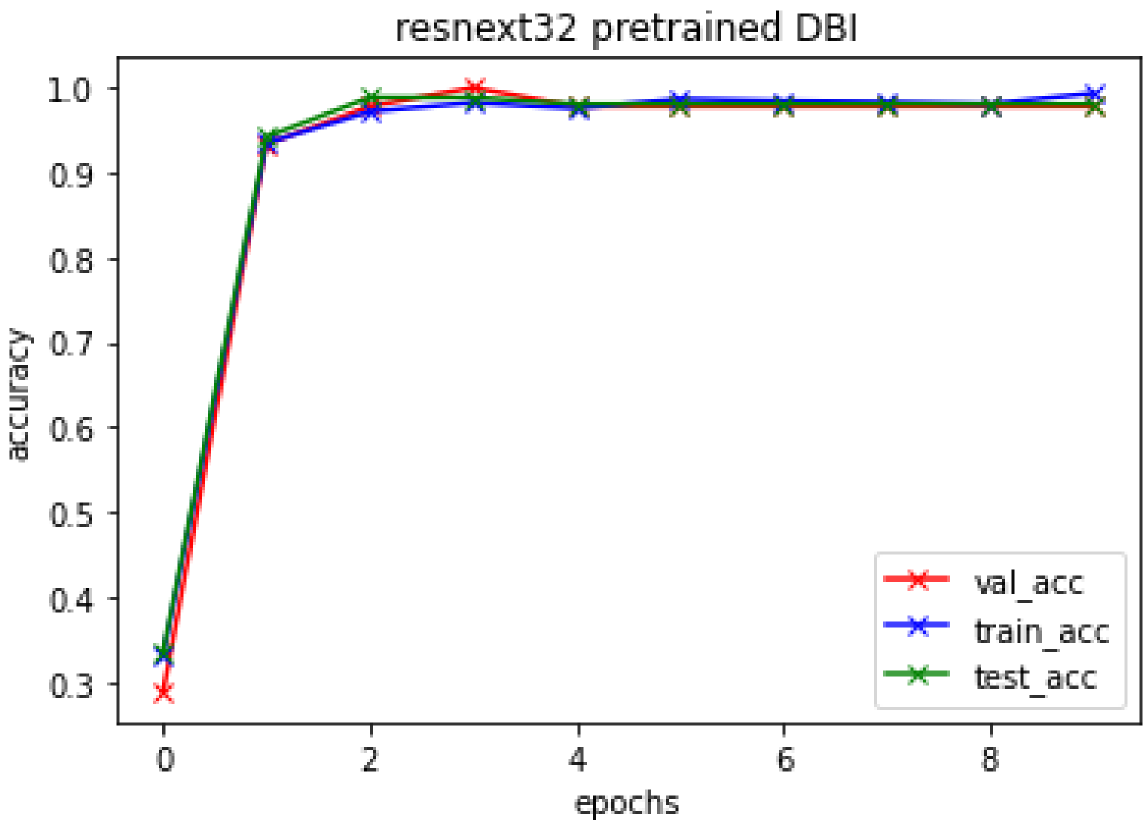
test acc = 0.9149



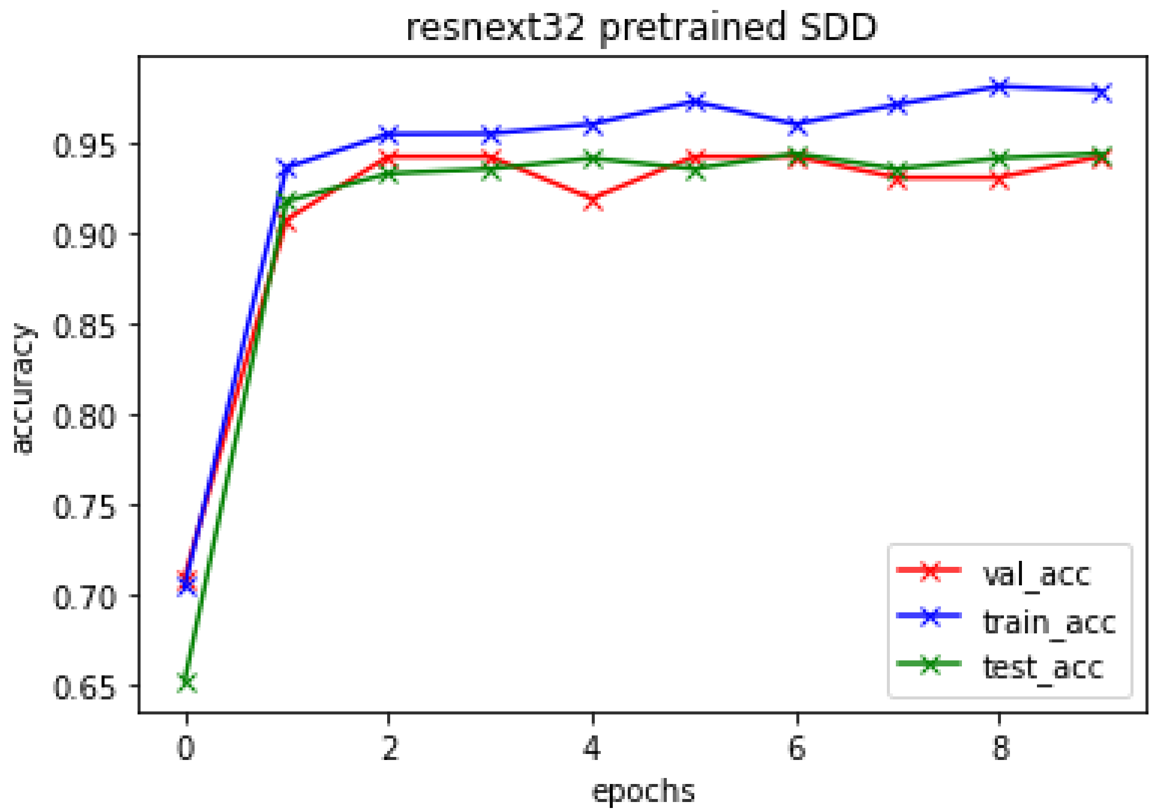
test acc = 0.9827



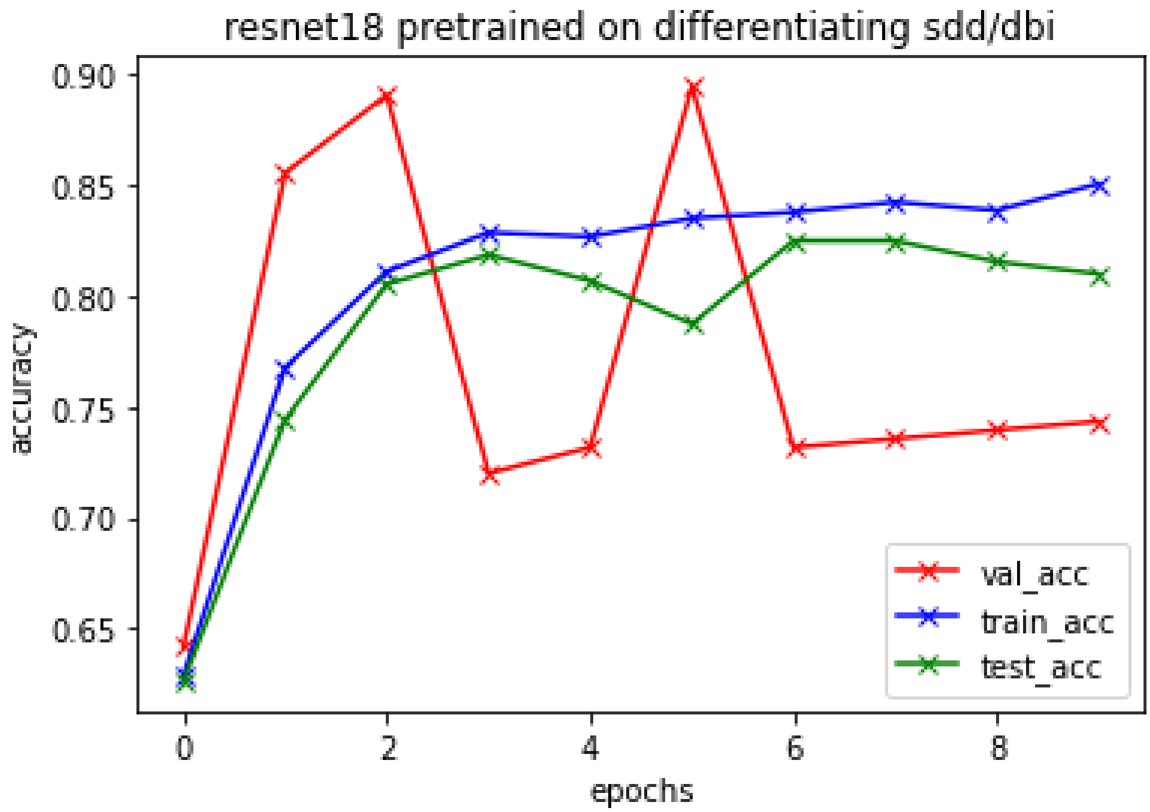
test acc = 0.9335



test acc = 0.9805



test acc = 0.9358
Most of these models experience a dropoff in SDD compare to DBI, while resnet18 and resnet34 performed identically in the DBI test, it suffered a harder droppoff in SDD.


resnet18 pretrained on differentiating sdd/dbi
0.90
0.85
0.80
0.75
0.70
0.65
accuracy
0
2
4
6
8
epochs
val_acc
train_acc
test_acc

```
[ ] class PretrainedResnet18SDDDBI(ImageClassificationBase):
        def __init__(self):
            super().__init__()

            self.network = models.resnet18(weights='ResNet18_Weights.DEFAULT')
            # Replace last layer
            for param in self.network.parameters():
              param.requires_grad = False
            num_ftrs = self.network.fc.in_features
            self.network.fc = nn.Sequential(
                nn.Linear(num_ftrs, 2),
                nn.LogSoftmax(dim=1)
            )

        def forward(self, xb):
            return self.network(xb)
```

```
[16] # set hyperparams
     num_epochs = 10
     opt_func = torch.optim.SGD

     max_lr = 0.01
     grad_clip = 0.1
     weight_decay = 1e-4
```

I used these hyper parameters because these are the values given in the medium article to test out different sets of pretrained model, which is close to the workload we have here. SGD is also a better choice compare to Adam as it better generalizes comparatively.