CSC420 Assignment 2

Sailing Ni

Feb 2023

1 Image Pyramids

Represent Laplacian Pyramid L_k by definition:

$$L_k = I_k - U(I_{k+1}), \text{ where } k \in [0, n-1] \text{ and } U \text{ is linear expanding operator}$$

To recover I_k using opposite downsampling operation:

$$I_k = L_k + U(I_{k+1}), \text{ where } k \in [0, n-1]$$

Express I_0 recursively:

$$\begin{aligned}
I_0 &= L_0 + U(I_1) \\
&= L_0 + U(L_1 + U(I_2)) \\
&= L_0 + U(L_1) + U^2(I_2)) \\
&= L_0 + U(L_1) + U^2(L_2 + U(I_3))) \\
&= L_0 + U(L_1) + U^2(L_2) + U^3(I_3) \\
&= ... \\
&= L_0 + U(L_1) + U^2(L_2) + U^3(I_3) + ... + U^{n-1}(L_{n-1}) + U^n(I_n) \\
&= \sum_{i=0}^{n-1} U^i(L_i) + U^n(I_n)
\end{aligned}$$

Therefore, the minimum information is I_n, and the base case is 1 pixel.
To reconstruct the original image, we expand base case I_n, summed by series of upsampling Laplacian Pyramid L_k at all levels of $k \in [0, n-1]$.

2 Activation Functions

Assume there are n layers of linear activations.
Let x_i be the output after $i-th$ layers, where x_0 is the input for $i \in [1, n]$.
Let W_i denote weights and B_i denote bias.
Apply two activation layers on x_{i-1}:

$$\begin{aligned} x_i = \sigma(x_{i-1}) &= W_{i-1}x_{i-1} + B_{i-1} \\ x_{i+1} = \sigma(x_i) &= W_i x_i + B_i \\ &= W_i(W_{i-1}x_{i-1} + B_{i-1}) + B_i \\ &= W_i W_{i-1} x_{i-1} + W_i B_{i-1} + B_i \end{aligned}$$

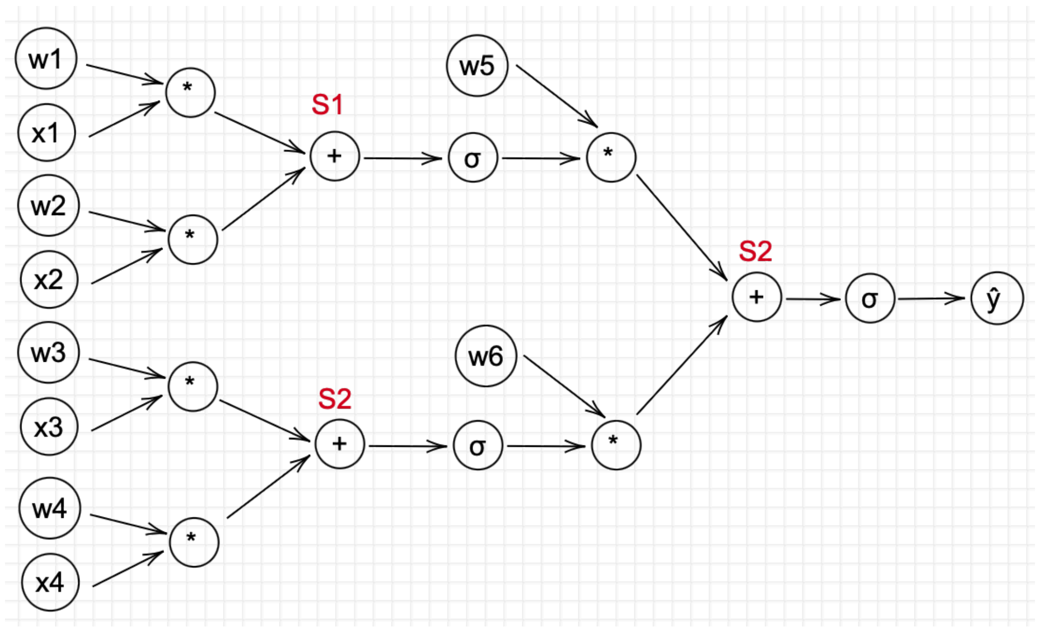
Let $W = W_i W_{i-1}$ and $B = W_i B_{i-1} + B_i$

$$= W x_{i-1} + B$$

Even though we applied two layers on x_{i-1}, it is effectively only 1 linear activation layer.
Therefore, any number of linear layers are eventually a single linear perceptron.
In other words, no matter how many layers applied, it effectively has no impact on the network.

3 Back Propgation

[3.a]: Computational graph



[3.b]: Partial Derivative

Given: $(x_1, x_2, x_3, x_4) = (1.2, -1.1, 0.8, 0.7)$; $y = 1.0$;
$(w_1, w_2, w_3, w_4, w_5, w_6) = (-0.65, -0.55, 1.74, 0.79, -0.13, 0.93)$;

$$\begin{aligned}
s_1 &= w_1x_1 + w_2x_2 = -0.175\\
s_2 &= w_3x_3 + w_4x_4 = 1.945\\
\sigma(s1) &= 0.456\\
\sigma(s2) &= 0.875\\
s3 &= w_5\sigma(s1) + w_6\sigma(s2) = 0.754\\
\hat{y} &= \sigma(s3) = 0.680\\
\frac{\partial L}{\partial \hat{y}} &= \frac{\partial[(y-\hat{y})^2]}{\partial \hat{y}} = -2(y-\hat{y}) = -0.640\\
\frac{\partial \hat{y}}{\partial s_3} &= \hat{y}(1-\hat{y}) = 0.218\\
\frac{\partial s_3}{\partial s_2} &= \frac{\partial s_3}{\partial \sigma(s_2)}\frac{\partial \sigma(s_2)}{\partial s_2} = w_6\sigma(s_2)(1-\sigma(s2)) = 0.102\\
\frac{\partial s_2}{\partial w_3} &= x_3 = 0.8\\
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial w_3} = -0.0113
\end{aligned}$$

4 Convolutional Neural Network

- Input size: 12 * 12 * 50

- Layer C: Convolutional Layer

# of filter positions = $\frac{input_size+padding*2-kernel_size}{stride} + 1 = \frac{12+2*1-4}{2} + 1 = 6$

Output size of layer C is 6*6*20, and each filter is of size 4*4*50.

Without bias, each filter performs $(4 * 4 * 50)$ times of multiplications and $(4 * 4 * 50 - 1)$ times of additions. Add 1 to each filter to include bias.

Therefore: w/o bias: # of flops $= 6^2 * 20 * (4 * 4 * 50 * 2 - 1) = 1,151,280$

w/ bias: # of flops $= 6^2 * 20 * (4 * 4 * 50 * 2) = 1,152,000$

- Layer P: Pooling layer

# of filter positions = $\frac{input_size+padding*2-kernel_size}{stride} + 1 = \frac{6+2*0-3}{1} + 1 = 4$

Output size of P is 4*4*20. Each 3*3 matrix performs max: # of flops= $20 * 4 * 4 * (3^2 - 1) = 2,560$

- Conclusion: output

w/o bias: Total # flops $= 1,151,280 + 2,560 = 1,153,840$

w/ bias: Total # flops $= 1,152,000 + 2,560 = 1,154,560$

5 Trainable Parameters

- C1: kernel size $k = 32 - 28 + 1 = 5$
  # Trainable Params $= k^2 * c * N = 5^2 * 1 * 6 = 150$
- S2: Downsampling does not require trainable parameters.
  # Trainable Params $= 0$
- C3: kernel size $k = 14 - 10 + 1 = 5$
  # Trainable Params $= k^2 * c * N = 5^2 * 6 * 16 = 2,400$
- S4: Downsampling does not require trainable parameters.
  # Trainable Params $= 0$
- C5: # Trainable Params $= 16 * 5 * 5 * 120 + 120 = 48,120$
- F6: # Trainable Params $= 84 * 120 + 84 = 10,164$
- C7: # Trainable Params $= 10 * 84 + 10 = 850$
- Total # Trainable Params $\Sigma = 150 + 0 + 2,400 + 0 + 48,120 + 10,164 + 850 = 61,684$

6 Logistic Activation Function

Let x be the input, y be the output of the node, where y is a sigmoid function i.e. $y = \frac{1}{1+e^{-x}}$

$$\begin{aligned}\frac{\partial y}{\partial x} &= \frac{0 + e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1 + e^{-x} - 1}{(1+e^{-x})^2} \\ &= \frac{1 + e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\ &= y - y^2\end{aligned}$$

We can see the gradient is a function of y.
Therefore, we only need output value y to compute the gradient of logistic function, without information about inputs.

7 Hyperbolic Tangent Activation Function

(a): Output Range

$$tanh(x) = \frac{-1 - e^{-2x} + 2}{1 + e^{-2x}} = -1 + \frac{2}{1 + e^{-2x}}$$
$$\lim_{x \to -\infty} tanh(x) = -1; \lim_{x \to \infty} tanh(x) = 1$$

Therefore,. output range of $tanh(x)$ is $(-1, 1)$, while output range of logistic function is $(0, 1)$. Major difference is $tanh(x)$ can have negative values but logistic functions are always positive.

(b): Formulate as function of logistic function

Note: logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\begin{aligned}
tanh'(x) &= \frac{\partial(-1 + \frac{2}{1+e^{-2x}})}{\partial x} = \frac{\partial(\frac{2}{1+e^{-2x}})}{\partial x} \\
&= \frac{0(1 + e^{-2x}) - 2(-2)e^{-2x}}{(1 + e^{-2x2})} \\
&= \frac{4e^{-2x}}{(1 + e^{-2x})^2} \\
&= 4\frac{e^{-2x}}{1 + e^{-2x}}\frac{1}{e^{-2x}} \\
&= 4\frac{1}{1 + e^{2x}}\frac{1}{1 + e^{-2x}} \\
&= 4\sigma(-2x)\sigma(2x)
\end{aligned}$$

(c): $tanh(x)$ vs $\sigma(x)$

- Logistic function has output range of $(0, 1)$, which can be used in probability predication, since any probabilities lies between 0 and 1.
- tanh(x) function preserves input signs and has the advantage that negative inputs will be mapped strongly negative. It is useful in binary classification.

8 Implementation Tasks

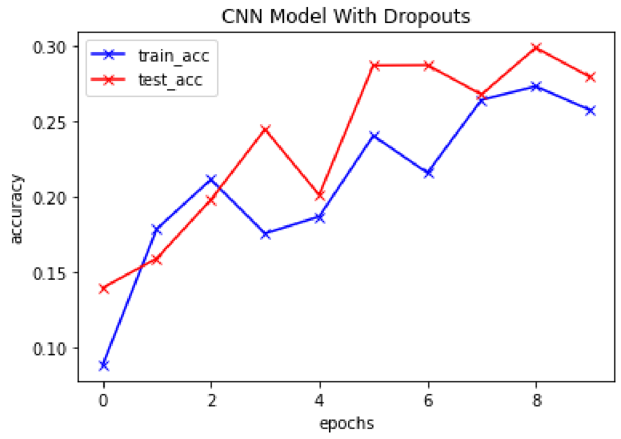
See `a2_part2ipynb` for code and solutions.

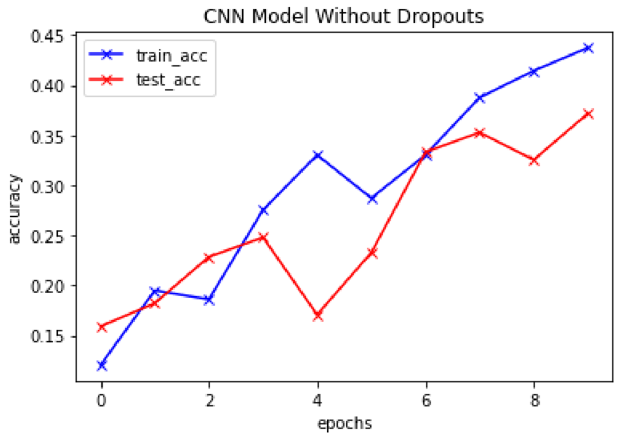
Task I: Inspection

- In DBI, dogs in images of relatively plain and clear backgrounds and color of dogs are in great contrast with the background color. It is easy to separate the dogs from the background.
- In SDD, the images are overall of lower resolution than DBI. The backgrounds are much messier and dogs are surrounded with other objects. We can see objects like human beings, decorations, food, blankets, flowers, other animals etc., and some images contain captions and watermarks. Also, many SDD images have background color that are similar to dog colors. It is harder to separate the dogs from backgrounds.

Task II: simple CNN Training on the DBI

Training and testing accuracy over the first 10 epochs:





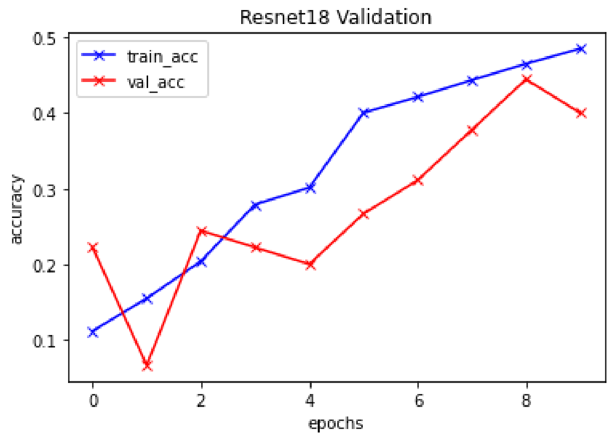
Comments:

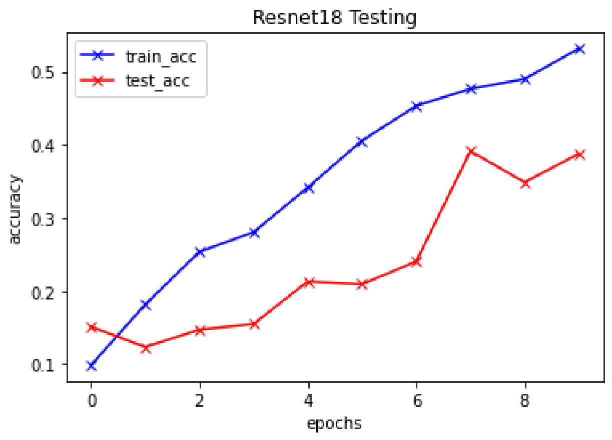
- Overall, both cases show similar accuracy results. In the case of CNN model with dropouts, test accuracy ends up at around 28%, while without dropouts it is around 37%.
- Without dropout, the training accuracy increases faster and smoother over time. With dropouts, test accuracy is often higher than train accuracy, which might indicate underfitting.
- Dropout randomly sets activations to zero during the training process to avoid overfitting. Normally, using dropouts can increase accuracy. However, in this case, dropouts have led to decrease in accuracy.
- This underfitting problem can be attributed to the small dataset size we are using, where dropouts make it harder to fit.

Task III: ResNet Training on the DBI

<u>III.a: ResNet-18 on DBI</u>

Training, validation, and testing accuracy over the first 10 epochs:



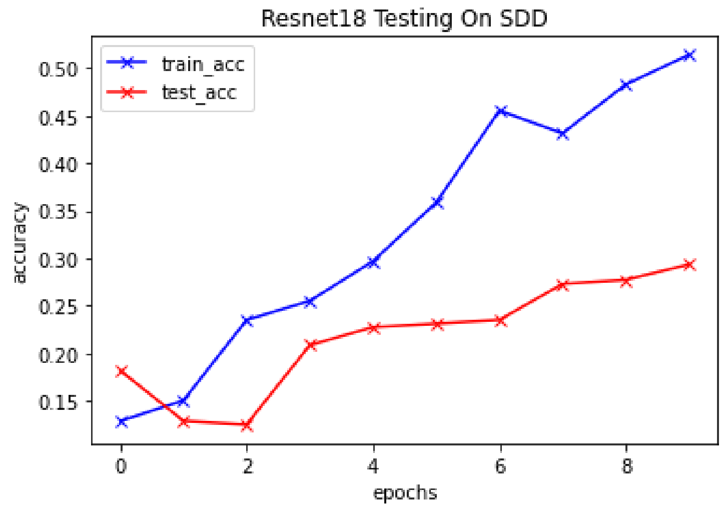


<u>Comments:</u>

- After 10 epochs, validation and testing accuracies are 40% and 39%.
- Compared with CNN model, the test accuracy for ResNet-18 model is slightly higher than CNN model and it increases more smoothly on increase of epochs.

<u>III.b: ResNet-18 on SDD</u>

Training and testing accuracy on SDD, over the first 10 epochs:

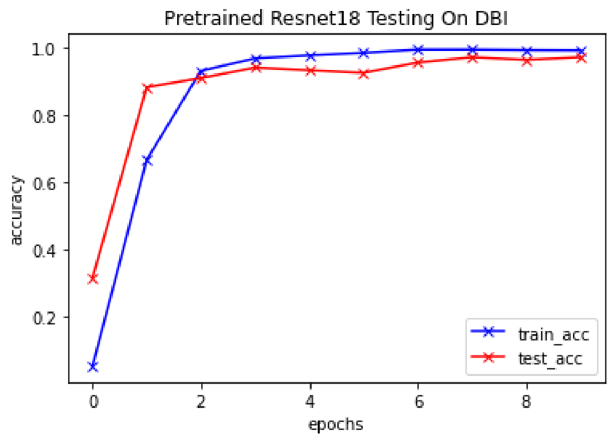


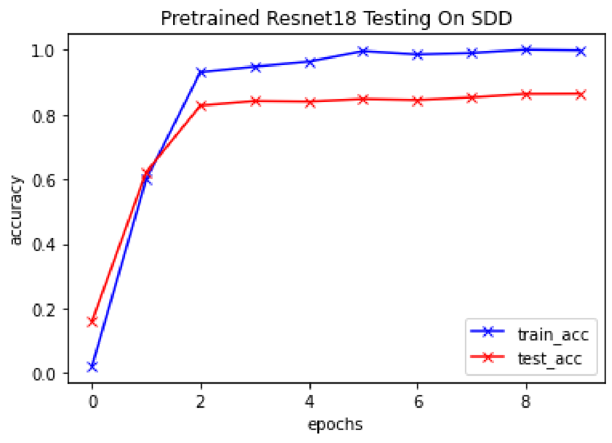
<u>Comments:</u>

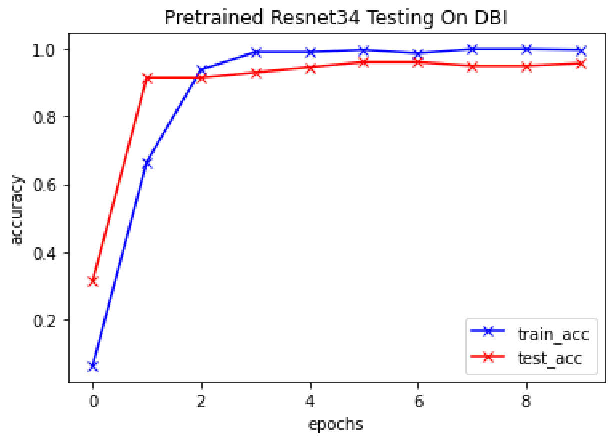
- After 10 epochs, testing accuracy on SDD is around 29%, lower than on DBI.
- This can be explained by the natural differences between DBI and SDD, as discussed in (I): background are more complex in SDD dataset with more irrelevant objects present, while most backgrounds in DBI are plain. Therefore, the DBI-trained model likely might lack the ability of distinguish irrelevant objects and thus incorrectly classify the dogs.

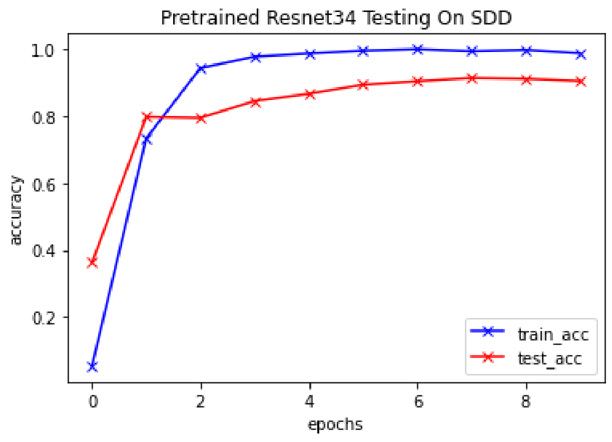
Task IV: Fine-tuning on the DBI

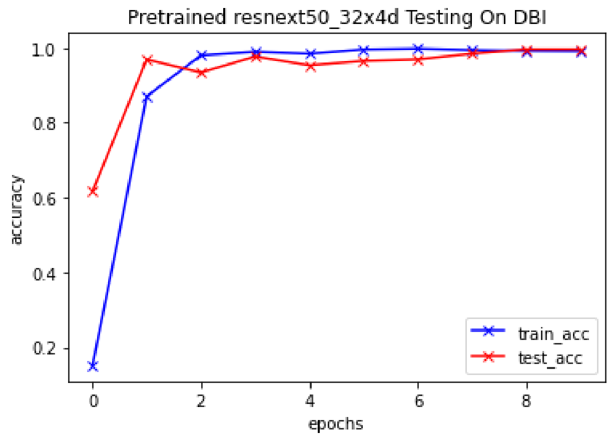
Use three pretrained models on DBI from PyTorch: `ResNet18`, `ResNet34`, and `Resnext50_32x4d`.
<u>Left:</u> Testing Accuracy on DBI; <u>Right:</u> Testing Accuracy on SDD.

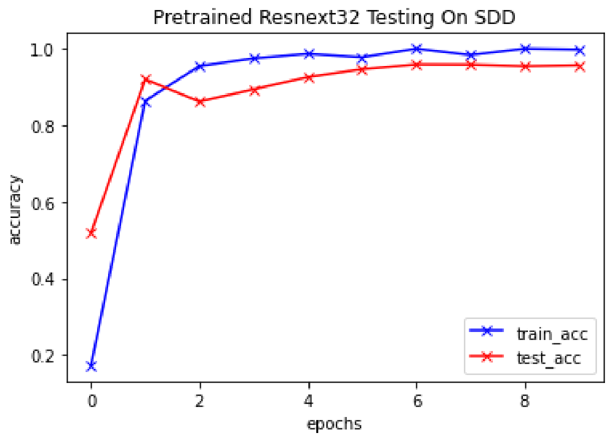










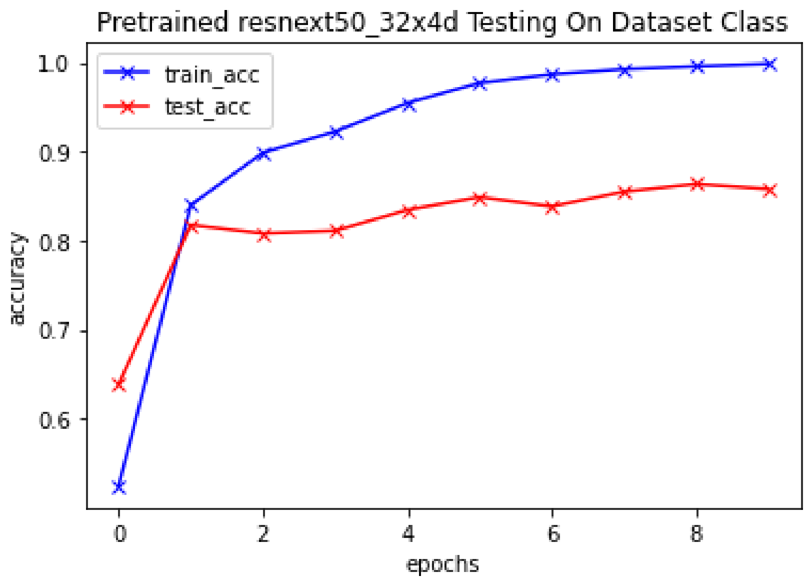


Summary of different models' testing accuracy after 10 epochs:

Model	Accuracy (DBI)	Accuracy (SDD)
ResNet18	0.9728	0.8640
ResNet34	0.9575	0.9048
Resnext50_32x4d	0.9961	0.9568

Comments:
- Overall, all trained models performed way better than the models we trained from scratch. Notice similar trend when we compare DBI accuracy to SDD accuracy. Testing accuracy on DBI is almost perfect: all three models' accuracies are over 95%. Testing on SDD, however, is at lower accuracy with best performing only at 95% and least performing at 86%.
- `Resnext50_32x4d` is the best performing pretrained model in both testing on DBI and SDD, with accuracies of 99.6% and 95.7% respectively.
- `ResNet18` and `ResNet34` performed equally well in testing on DBI, with both accuracies above 95%. While in SDD testing, `ResNet34`'s 90.5% is better than the 86.5% of `ResNet18`.

Task V: Dataset classification
- Start from restructuring the folders by flattening the original dog breed classifications.
Dataset structure 'datasets-DBI/SDD-Dog_Breed-images' is changed to 'datasets-DBI/SDD-images'. In the new dataset, all images are labelled either 'DBI_subest' or 'SDD_subset'.
- Split the dataset into 30% test and 70% train randomly.
- Start from the pretrained `Resnext50_32x4d` model, and fine-tune it by training on the new dataset we created. I pick `Resnext50_32x4d` because of its superior performance than the other two pre-trained models.
- Hyperparams are: 10 epochs, max learning rate of 0.01, and SGD optimizer. After 10 epochs, the accuracy seems to plateaued around 86% and the curve starts to flatten.
- Accuracy results are plotted below, where we can the model fits really fast in the first epoch and has almost non-decreasing accuracy over time.



Task VI: How to improve performance on SDD?

1. **At training time, we have access to the entire DBI dataset, but none of the SDD dataset. All we know is a high level description of SDD and its differences with DBI (similar to the answer you provided for Task I of this question).**
   <u>Answer:</u>
   - In this case, we do not have any portion of data from SDD dataset.
   - We can apply adversarial training and train the model to distinguish between the source domain and the target domain. The idea is to force the model to learn features that are invariant to the differences between the domains, which can improve its performance on the target domain.

2. **At training time, we have access to the entire DBI dataset and a small portion (e.g. 10%) of the SDD dataset.**
   <u>Answer:</u>
   - In this case, we have a limited labeled sample from the SDD dataset. - We can use transfer learning techniques to pre-train a model on the DBI dataset and then fine-tune it on the limited labeled examples from the SDD dataset.
   - It may also be helpful to use data augmentation techniques to increase the size of the labeled SDD dataset and improve the model's ability to generalize to new examples.

3. **At training time, we have access to the entire DBI dataset and a small portion (e.g. 10%) of the SDD dataset but without the SDD labels for this subset.**
   <u>Answer:</u>
   - In this scenario, we have a limited labeled sample from the SDD dataset, but we cannot use it to directly train the model. Instead, we can use it to validate the performance of the model on the target population and to guide the training process.
   - We can apply active learning techniques to select the most informative examples from the unlabeled SDD dataset and use them to guide the training process.
   - It may also be helpful to use transfer learning techniques to pre-train a model on the DBI dataset and then fine-tune it on the limited unlabeled examples from the SDD dataset.

Task VII: Discussion

- We can apply transfer learning to address the cross-testing scenario. Reusing a pre-trained model that was trained on similar task, but on a different source domain. You pick among a few pretrained models to choose the best fit.
- We can use data augmentations in our training to create synthetic data points that adapt to the differences between the datasets.
- If possible, collect some labeled examples from the testing dataset to fine-tune the pre-trained model on testing data to improve performance on the target dataset.
- Apply adversarial training, train the model to distinguish between the source domain and the target domain. The idea is to force the model to learn features that are invariant to the differences between the domains, which can improve its performance on the target domain.