CSC420 Assignment 2

Name: Liwei Yang
My teammate: Muxin Tian, student number 1005800885

1 Question 1

Proof.

Lemma 1: I_k can be reconstructed using I_{k-1} and L_k.

Proof. Assume we have a $2^n \times 2^n$ image I_0 with its respective Gaussian pyramid $\{I_i\}_{i=0}^{\leq n-1}$ Laplacian pyramid $\{L_i\}_{i=0}^{\leq n-1}$. We will show that, we can reconstruct I_{k-1} using I_k and L_{k-1}, for some $k \in \{1, \cdots, n\}$. Then, there exists some $x \in \mathbb{N}$ where I_k can be represented as a $p \times 2^x \times 2^x$ tensor (since it might not just be a gray-scale picture). We will firstly upsample I_k to an $p \times 2^{x+1} \times 2^{x+1}$image I'_k , by

$$\forall i, j \in \{1, \cdots, 2x\}, \ I'_k[p, i, j] = I_k[p, \lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil]$$

Then, after upsampling, we will convolve the upsampled image with Gaussian kernal g with $\sigma = \sqrt{2}$, size = 2, or:

$$I^\star_k = I'_k * g$$

Then, note that, by definition, we have:

$$\forall i, j \in \{1, \cdots, 2x\}, L_{k-1}[p, i, j] = \nabla^2 I_{k-1}[p, i, j]$$

Then, we will have

$$I_{k-1} = L_{k-1} + I^\star_k$$

□

Then, by our lemma, we know that we may reconstruct any level I_{k-1} with I_k and L_{k-1}. Then by simple induction, we know that we can reconstruct I_0 from I_{n-1}. Therefore, we only need the information from level I_{n-1} with dimension $p \times 1 \times 1$. □

2 Question 2

Proof. Suppose we have a linear fully connected (FC) neural network with n-layer hidden layers ($n \geq 0$). We will show that it has the same expressiveness as a single layer FC. Assume that we have a linear activation function $\phi = c$. **WLOG, assume** $c = 1$, since any other c can be simulated dividing output y by $\prod_{i=1}^{n} c^{(i)}$. Then, for any hidden layer i, we have that:

$$h^{(i)} = \phi\left(W^{(i)}\left(h^{(i-1)}\right) + b^{(i)}\right) = W^{(i)}h^{(i-1)} + b^{(i)}$$

Then, we may express the output y as a composition function as x, equivalently,

$$\begin{aligned}
y = f(x) &= h^{(n)} \circ \cdots h^{(1)}(x) \\
&= \left(W^{(n)}\left(W^{(n-1)}\left(\cdots\left(W^{(1)}x + b^{(1)}\right) + \cdots\right) + b^{(n-1)}\right) + b^{(n)}\right) && \text{(Linearity)} \\
&= \overbrace{b^{(n)} + W^{(n)}b^{(n+1)} + W^{(n)}\left(W^{(n-1)}b^{(n-2)}\right) + \cdots + \left(W^{(n)}\cdots W^{(2)}b^{(1)}\right)}^{\triangleq b^{\star}\text{ (Bias compartment)}} \\
&+ \overbrace{W^{(n)}W^{(n-1)}\cdots W^{(2)}W^{(1)}}^{\triangleq W^{\star}\text{ (Weight compartment)}}(x) \\
&= W^{\star}x + b^{\star}
\end{aligned}$$

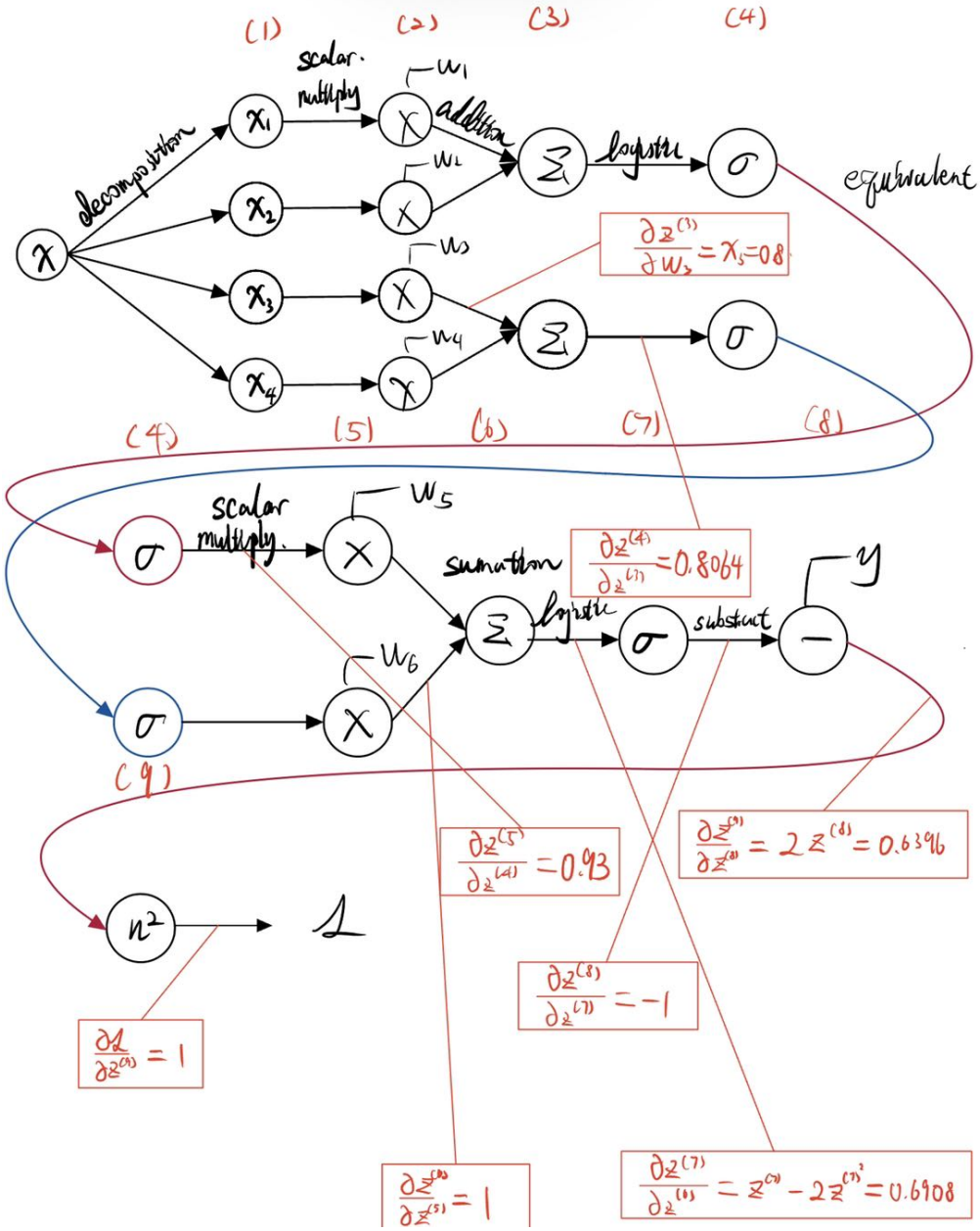
Where, we have shown that, a multi-linear FC with linear activation function is equivalent to a linear single-layer FC with corresponding bias $b^{\star}$ and weight $W^{\star}$, which are both independent to the input x. □

3 Question 3

3.1 Question 3a

Here, I only labeled the relevant nodes using $z^{(i)}$, for the ease of presentation. Also, the value of forward pass is not shown in the figure. It is viewable on demos.

https://www.desmos.com/calculator/6izo8yrxiw.



3.2 Question 3b

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(9)}} \frac{\partial z^{(9)}}{\partial z^{(8)}} \frac{\partial z^{(8)}}{\partial z^{(7)}} \frac{\partial z^{(7)}}{\partial z^{(6)}} \frac{\partial z^{(6)}}{\partial z^{(5)}} \frac{\partial z^{(5)}}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial w_3}$$

Where, we have:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z^{(9)}} &= 1 \\
\frac{\partial z^{(9)}}{\partial z^{(8)}} &= 0.6396 \\
\frac{\partial z^{(8)}}{\partial z^{(7)}} &= -1 \\
\frac{\partial z^{(7)}}{\partial z^{(6)}} &= 0.6908 \\
\frac{\partial z^{(6)}}{\partial z^{(5)}} &= 1 \\
\frac{\partial z^{(5)}}{\partial z^{(4)}} &= 0.93 \\
\frac{\partial z^{(4)}}{\partial z^{(3)}} &= 0.8064 \\
\frac{\partial z^{(3)}}{\partial w_3} &= 0.8
\end{aligned}$$

Then, by chain rule, we have:

$$\frac{\partial \mathcal{L}}{\partial w_3} = 1 \times 0.6396 \times -1 \times 0.6908 \times 1 \times 0.93 \times 0.8064 \times 0.8 = -0.2651$$

The detailed calculation for forward pass is available here https://www.desmos.com/calculator/6izo8yrxiw.

4 Question 4

We first compute the size of layer P. Since the stride is 2, we have

$$12 \times \frac{1}{2} = 6$$

thus the pooling layer is 6×6.

Without counting bias: Here, we will have:

$$20 \times 6 \times 6 \times (4 \times 4 \times 50 \times 2 - 1) + 8 \times 16 \times 20 = 1153840$$

Thus, a total of 1153840 FLOPs will be conducted during one forward pass, if we ignore bias.

With counting bias: Here, we will have:

$$20 \times 6 \times 6 \times (4 \times 4 \times 50 \times 2) + 8 \times 16 \times 20 = 1154560$$

Thus, a total of 1154560 FLOPs will be conducted during one forward pass, if we do count bias.

5 Question 5

I assumed the original 32×32 to be a grey scale picture.
We firstly compute the filter size f_{C_1}, f_{C_2} and stride s_{C_1}, s_{C_2} where all each parameters $\in \mathbb{N}^+$.

$$(32 - f_{C_1}) \times \frac{1}{s_{C_1}} + 1 = 28 \implies \begin{cases} f_{C_1} = 5 \\ s_{C_1} = 1 \end{cases}$$
$$(14 - f_{C_2}) \times \frac{1}{s_{C_2}} + 1 = 10 \implies \begin{cases} f_{C_2} = 5 \\ s_{C_2} = 1 \end{cases}$$

We first consider the number of trainable parameters in each layer, each denote as $t_{<layer>}$

$$\begin{aligned}
t_0 &= 0 && \text{(input)} \\
t_{C_1} &= \left(f_{C_1}^2 \times 1 + 1\right) \times 6 = 156 \\
t_{S_2} &= 0 && \text{(Pooling)} \\
t_{C_3} &= \left(f_{C_2}^2 \times 6 + 1\right) \times 16 = 2416 \\
t_{S_4} &= 0 && \text{(Pooling)} \\
t_{C_5} &= 120 \times 16 \times 5 \times 5 + 120 = 48120 \\
t_{F_6} &= 120 \times 84 + 84 = 10164 \\
t_7 &= 10 \times 84 + 10 = 850
\end{aligned}$$

Then, we will have the total number of trainable parameters to be:

$$\begin{aligned}
t &= t_0 + t_{C_1} + t_{S_2} + t_{C_3} + t_{S_4} + t_{C_5} + t_{F_6} + t_7 \\
&= 0 + 156 + 0 + 2416 + 0 + 48120 + 10164 + 850 \\
&= 61706
\end{aligned}$$

6 Question 6

Proof. Suppose we have a $n-$layers FC neural network, where each layer has h_i neurons.
Assume the neurons in the FC-NN is ordered from up to top.
For a specific layer i, where $1 \leq i \leq n$, we let the weight of the layer i be $w_{i,j}$, where j is the position from up to down, and $1 \leq j \leq h_i$. Similarly, we define the input and output of that neurons as $x_{i,j}$ and $y_{i,j}$, respectively.
Then, with input $x_{i,j}$ and logistic activation function σ, we have:

$$y_{i,j} = \sigma\left(\sum_{m=1}^{h_{i-1}} (w_{i,j})_m \, x_{i,j}\right)$$

Then, we define a function:

$$z_{i,j} \triangleq \left(\sum_{m=1}^{h_{i-1}} (w_{i,j})_m \, x_{i,j}\right), \qquad \text{we will have } \sigma(z_{i,j}) = y_{i,j}$$

Then, we will have:

$$\begin{aligned}
\frac{\partial y_{i,j}}{\partial x_{i,j}} &= \frac{\partial y_{i,j}}{\partial z_{i,j}} \frac{\partial z_{i,j}}{\partial x_{i,j}} = \frac{e^{-z}}{(1+e^{-z})^2 \, 2} (w_{i,j})_m \\
&= \frac{1}{(1+e^{-z})^2 \, 2} e^{-z} (w_{i,j})_m \\
&= \left(\sigma(z_{i,j}) - \sigma(z_{i,j})^2\right) (w_{i,j})_m \\
&= \left(y_{i,j} - y_{i,j}^2\right) (w_{i,j})_m
\end{aligned}$$

Then, note that, for intermediate hidden layer, the input of this layer i is the output of the previous layer $i-1$. Therefore, we will have $x_{i,j} = y_{i-1,j}$. Thus, we will have

$$\frac{\partial y_a}{\partial y_b} = \prod_{i=b}^{a-1} \frac{\partial y_{i+1}}{\partial y_i}$$

for some $n \geq a \geq b \geq 1$. Furthermore, finding each $\frac{\partial y_{i+1}}{\partial y_i}$ only requires for us to know the output of that layer. Thus, we do not to know the input for the backward pass of gradient. □

7 Question 7

7.1 Question 7a

The codomain of **tanh** is $(-1, 1)$, and the codomain of **logistics** is $(0, 1)$.

7.2 Question 7b

We will differentaite the original function regarding x:

$$\begin{aligned}
\frac{d}{dx}\,\textbf{tanh}(x) &= \frac{d}{dx}\frac{1-e^{-2x}}{1+e^{-2x}} = \frac{4\mathrm{e}^{2x}}{\left(\mathrm{e}^{2x}+1\right)^2} && \text{(Quotient rule)} \\
&= 4e^{2x} \times \left(\frac{1}{e^{2x}+1}\right)^2 = 4\left(\frac{1}{\sigma(2x)} - 1\right) \times \left(\sigma(2x)\right)^2 \\
&= 4\sigma(2x) - 4\sigma(2x)^2 \\
&= 4\left(\sigma(2x) - \sigma(2x)^2\right)
\end{aligned}$$

7.3 Question 7c

In **almost every case**, we prefer ReLU over these two activation functions for a better propagation of gradient. However, when we are restricted to these two functions, here are the vantages of each of them:
Sigmoid (logistics)

- We can computer the gradient of a logisitc layer directly from the output, without needing the input.
- Sigmoid has smaller gradient, so the weight updates are milder and slower. This may help to reduce the chance of stucking in a local minima.
- Compare to tanh, the computation of sigmoid is faster.

tanh

- The gradient of tanh is significantly greater (2-4 times) at the center, compared to sigmoid.
- Therefore, during propagation, the gradient tend to be greater and we will have a more significant update for the weights, leading to faster convergence (training time).

Part 2

I used the starter code given in the pdf file.

Task I

From inspection, I conclude the following differences of the two datasets.

1. SDD
    a. The images in SDD dataset have different sizes. Although most of them are within 10-100kb range, some of the images can be as large as a few megabytes.
    b. The subject of the image is more diverse, sometimes it contains people, table, and more objects that may be distractive for the model.
    c. Generally, I believe that this dataset is a better representation of general dog images compare to DBI. However, it may be harder to train on.
2. DBI
    a. The size of the images is more even more inconsistent (smoother distribution). The size of most of the image ranges from 10kb to 1mb.
    b. The images are less diverse and focuses mainly on the dog. There are no distractive objects for the AI to recognize.
    c. I believe that the dataset is a more specific representation of dog images. It is easier to train a better model on it. However, the model trained from this dataset may have poor generalization on other dataset or real live pictures.

Task II

I used the exact parameters as given in the assignment, with the following hyperparameters.

Algorithm 1: Model configuration

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

batch = 128
max_lr = 0.015
grad_clip = 0.1
weight_decay = 1e-4
```

The code of the model is available with specific comments. [1] We used a slightly higher learn rate with a larger batch size. I believe such setting would give a better generalization and higher performance gain per epoch.

[1] Code available: https://colab.research.google.com/drive/18- njyB_iQ41g9WsztQGMQlGB58lcvKUQ?usp=sharing

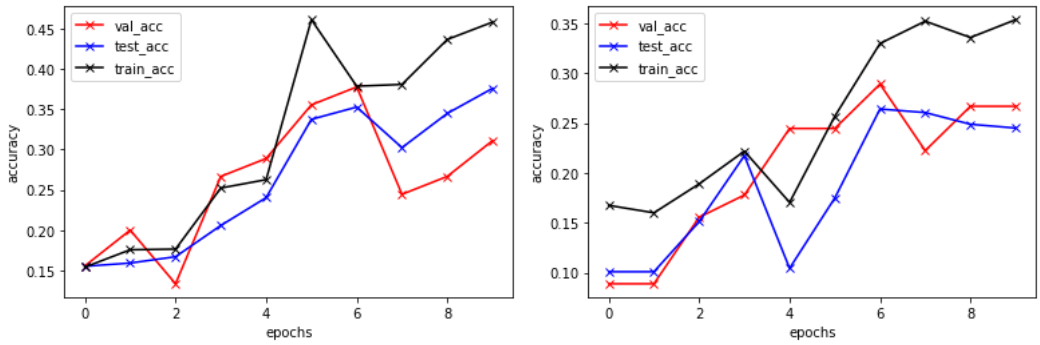


Figure 1: Accuracy of the customized model. The left figure is the model without dropout, while the right one has in-place dropout enabled.

As shown in **Figure 1**, a gradual improve in accuracy of prediction is observed across both models. The model without dropout achieved higher accuracy in the training set. However, the difference between the validation set and the training set model with dropout enabled is slightly lower (32.5% higher) when compared to the one without dropout (47.3%).

I believe that mechanism behind the above observation is that, when the dropout functionality is enabled, the network learns slower as some neurons will be dropped and will not gain any experience during some epochs. However, the dropout also improves the ability of the model to generalize the acquired knowledge, making the accuracy of the validation set closer to the training set.

Task III

Task III.a

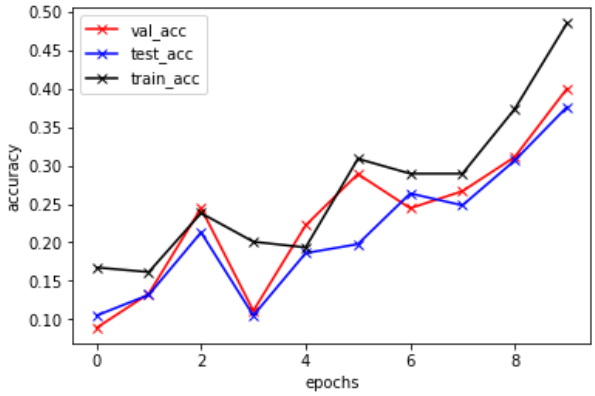


Figure 2: Accuracy of the Resnet18 model.

The above figure (**Figure 2**) reflects the accuracy of the Resnet18 model during the first 10 epochs of training. It is clear that, the Resnet18 model has better accuracy across all sets, and has better generalization than all of the customized models.

Task III.b

Algorithm 2: Evaluation of the performance

```
SDDcorrect = 0
SDDtot = 0
models = [model3_DBI, model4_DBI, model5_DBI]
SDD_Accuracy = []
for m in models:
  # print(type(m))
  for i in range(len(SDDdataset)):
    if predict_single_usig_different_model(*SDDdataset[i], m):
      SDDcorrect += 1
    SDDtot += 1
    # if i % 100 == 0:
      # print (float(i) / float(len(DBIdataset)))
  a = float(SDDcorrect) / float(SDDtot)
  SDD_Accuracy.append(a)
```

Table 1: Accuracy of the Resnet18 on SDD and test set of DBI

Model	Accuracy on SDD	Accuracy on DBI (test set)
Resnet18	0.3074	0.3605

It is clear that the model has better accuracy on the test set of DBI than the whole set of SDD. I believe that it is because of that the generalization ability of the model is not strong enough to train on a specific dataset while generalize all the knowledges to a bigger, less specific set.

Task IV

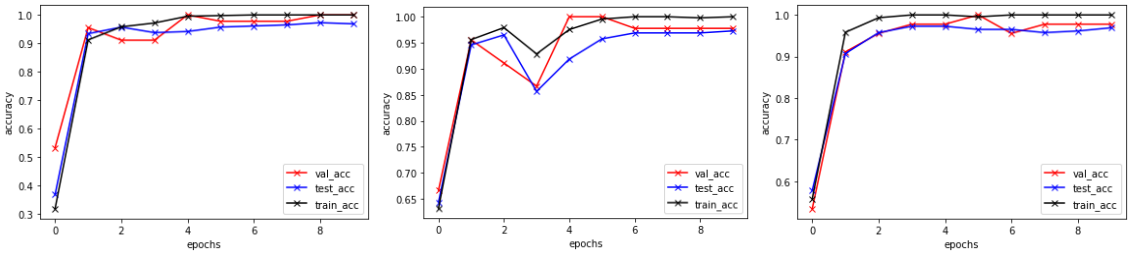


Figure 3: Accuracy of the pre-trained models on DBI dataset. The models listed from left to right are Resnet18, Resnet34, ResNeXt32-50, respectively.

Table 2: Accuracy of the pre-trained models on DBI dataset.

Model	Accuracy on SDD	Accuracy on DBI (test)
Resnet18	0.8929	0.9876
Resnet34	0.9152	0.9891
ResNeXt32	0.9167	0.9897

We used the above algorithm (**Algorithm 2**) to assess the accuracy of the three models on SDD (similarly, on DBI).

As shown above (**Figure 3, Table 2**), the relative ranking accuracy of the models are (in descending order) ResNeXt32-50, Resnet34, and Resnet18. Interestingly, Resnet34 had a greater difference in the accuracy in test set and training set.

The performance of the model is very consistent on DBI. However, there are slight difference in SDD, with Resnet 18 being lower than the other two. However, both models preserved high accuracy. I believe this might be attributed to a larger batch size.

```
Accuracy of <class '__main__.DogBreedResnet18Pretrained'>  on SDD: 0.902676399026764
Accuracy of <class '__main__.DogBreedResNeXt32Pretrained'>  on SDD: 0.9180859691808597
Accuracy of <class '__main__.DogBreedResnet34Pretrained'>  on SDD: 0.9159232224925655
```

(To be honest, the loss in performance is way below my expectation, here is the data of a second run to confirm that the data is not just high by chance)

Task V

Algorithm 3: Balancing the datasets and data division

```
import os
import random

files = os.listdir('../Task V/SDD copy')
files_DBI = os.listdir('../Task V/DBI')
print(files)
no = len(files)
to_delete = no - len(files_DBI)

lst = list(range(0, no, 1))
random.shuffle(lst)
print(lst)

delete_files = lst[:to_delete]

for i in range(len(delete_files)):
    os.remove('../Task V/SDD copy/' + files[delete_files[i]])
```

I used the above algorithm to randomly delete the excessive samples in the SDD sets to maintain the balance across the two datasets (before clipping there are approximately ~1200 images in SDD while only ~600 in DBI). This operation makes the expected value of getting the correct answer from a random guessing is 0.5. These operations made the performance gain is clearer and the training performance is better.

After the modification of the dataset, I set the training set ratio to be 0.7, which is higher than the one we use in previous trails. I use this parameter because of that I believe a higher ratio would give a better generalization across images. **I did not use a validation set.**[2]

[2] Code here: https://colab.research.google.com/drive/1IFr9sqLUg7hDeGkB18ZlfvW4Kqi9TMwR?usp=sharing

For the model, I chose **ResNeXt32-50**, which is the model with highest accuracy in the previous trails. The hyperparameters I used us consistent with **Algorithm 1 (max_lr = 0.015, batch = 128)**. I also modified the number of outputs of the last layer to **2**.

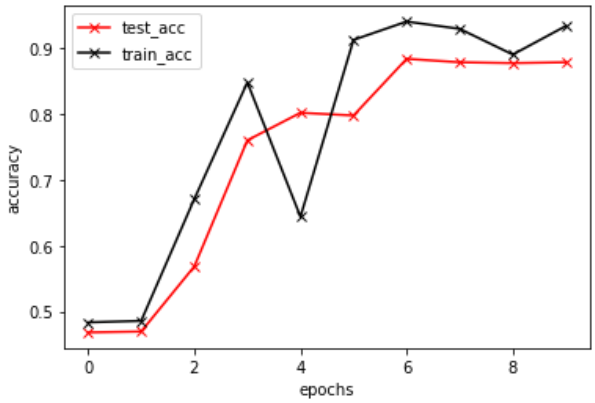


Figure 4: Accuracy of the pre-trained ResNeXt32-50 model on classifying the images into two datasets (DBI, SDD).

My model achieved good accuracy after 10 epochs of training on the **test set (0.8906)**, and good accuracy on the training set (0.9324).