Intro to Image Understanding (CSC420)
Assignment 2

Due Date: Feb 14^{th}, 2025, 11:59 pm
Total: 200 marks

General Instructions:

- You are allowed to work directly with one other person to discuss the questions. However, you are still expected to write the solutions/code/report in your own words; i.e. no copying. If you choose to work with someone else, you must indicate this in your assignment submission. For example, on the first line of your report file (after your own name and information, and before starting your answer to Q1), you should have a sentence that says: *"In solving the questions in this assignment, I worked together with my classmate [name & student number]. I confirm that I have written the solutions/code/report in my own words"*.
- Your submission should be in the form of an electronic report (PDF), with the answers to the specific questions (each question separately), and a presentation and discussion of your results. For this, please submit a file named **report.pdf** to MarkUs directly.
- Submit documented codes that you have written to generate your results separately. Please store all of those files in a folder called **assignment2**, zip the folder and then submit the file **assignment2.zip** to MarkUs. You should include a **README.txt** file (inside the folder) which details how to run the submitted codes.
- Do not worry if you realize you made a mistake after submitting your zip file; you can submit multiple times on MarkUs.

Collaboration

In solving the questions in this assignment, I worked together with my classmate (Yousef Al Rawwash, 1007684873). I confirm that I have written the solutions/code/report in my own words.

Part I: Theoretical Problems (80 marks)

[Question 1] Image Pyramids (10 marks)

In Gaussian pyramids, the image at each level I_k is constructed by blurring the image at the previous level I_{k-1} and downsampling it by a factor of 2. A Laplacian pyramid, on the other hand, consists of the difference between the image at each level (I_k) and the upsampled version of the image in the next level of the Gaussian pyramid (I_{k+1}).

Given an image of size $2^n \times 2^n$ denoted by I_0, and its Laplacian pyramid representation denoted by $L_0, ..., L_{n-1}$, show how we can reconstruct the original image, using the minimum information from the Gaussian pyramid. Specify the minimum information required from the Gaussian pyramid and a closed-form expression for reconstructing I_0.

Hint: The reconstruction follows a recursive process; What is the base case that contains the minimum information?

The Laplacian pyramid is constructed using the Gaussian pyramid as follows:

$$L_k = I_k - U(I_{k+1}), \quad \text{for } k = 0, 1, \ldots, n-1.$$

Equivalently:

$$I_k = L_k + U(I_{k+1}).$$

This recursion allows us to reconstruct I_0 starting from I_n.
Starting from the coarsest level I_n, we compute:

$$I_{n-1} = L_{n-1} + U(I_n).$$

$$I_{n-2} = L_{n-2} + U(I_{n-1}).$$

Expanding the recursion gives the closed-form expression:

$$I_0 = L_0 + U(L_1 + U(L_2 + \cdots + U(L_{n-1} + I_n))).$$

This equation expresses the reconstruction of I_0 using only the Laplacian pyramid and the smallest Gaussian image I_n.

[Question 2] Activation Functions (10 marks)

Show that in a fully connected neural network with linear activation functions, the number of layers has effectively no impact on the network.

Hint: Express the output of a network as a function of its inputs and its weights of layers.

We show a multi-layer network with L layers and linear activation is equivalent to a single layer network. Let the network $f(x)$ be as shown below.

$$f(x) = \phi_L(W^{(L)}\phi_{(L-1)}(W^{(L-1)}\phi_{L-2}(W^{(L-2)}\phi_{L-1}(\ldots) + b^{(L-2)}) + b^{(L-1)}) + b^{(L)})$$

Where $\phi_i(z) = z$, so

$$f(x) = W^L W^{(L-1)} W^{(L-2)} \ldots W^{(1)} W^{(0)} x + \left(b^0 + b^1 + \cdots + b^{(L-2)} + b^{(L-1)} + b^{(L)}\right).$$

We can make a substitution using

$$\mathbf{W} = W^L W^{(L-1)} W^{(L-2)} \ldots W^{(1)} W^{(0)} \qquad \mathbf{b} = b^0 + b^1 + \cdots + b^{(L-2)} + b^{(L-1)} + b^{(L)}$$

to write $f(x)$ in terms of these new variables.

$$f(x) = \mathbf{W}x + \mathbf{b}$$

Which is a linear model that is equivalent in representation. Thus we have shown that a multi-layer MLP with linear activation can be represented as a single linear layer, thus the number of layers has effectively no impact on the network.

Although a multi-layer network with linear activations reduces to a single-layer linear model, there may still be learning benefits in using multiple layers. However, such networks remain significantly less expressive than non-linear models, which can leverage inexpensive activations like ReLU to learn complex patterns effectively.

[Question 3] Back Propagation (10 marks)

Consider a neural network that represents the following function:

$$\hat{y} = \sigma(w_5\sigma(w_1x_1 + w_2x_2) + w_6\sigma(w_3x_3 + w_4x_4))$$

where x_i denotes input variables, $\hat{y}$ is the output variable, and σ is the logistic function:

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

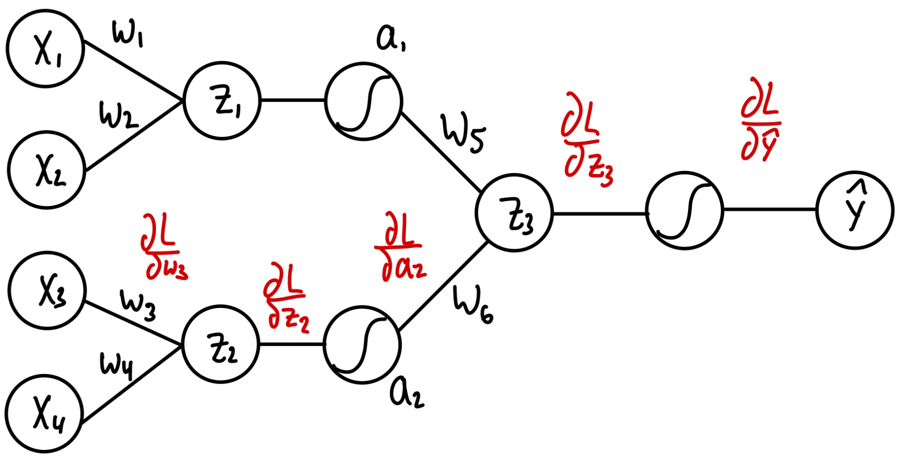
Suppose the loss function used for training this neural network is the **L2** loss, i.e. $L(y, \hat{y}) = (y - \hat{y})^2$. Assume that the network has its weights set as:

$$(w_1, w_2, w_3, w_4, w_5, w_6) = (-0.65,\ -0.55,\ 1.74,\ 0.79,\ -0.13,\ 0.93)$$

[3.a] (5 marks) Draw the computational graph for this function. Define appropriate intermediate variables on the computational graph. (break the logistic function into smaller components.)
[3.b] (5 marks) Given an input data point $(x_1, x_2, x_3, x_4) = (1.2,\ -1.1,\ 0.8,\ 0.7)$ with true label of 1.0, compute the partial derivative $\frac{\partial L}{w_3}$, by using the back-propagation algorithm. Indicate the partial derivatives of your intermediate variables on the computational graph. Round all your calculations to 4 decimal places.

Hint: For any vector (or scalar) $\mathbf{x}$, we have $\frac{\partial}{\partial \mathbf{x}}(||\mathbf{x}||_2^2) = 2\mathbf{x}$. Also, you do not need to write any code for this question! You can do it by hand.

[3.a]



x_1
x_2
x_3
x_4
w_1
w_2
w_3
w_4
w_5
w_6
z_1
z_2
z_3
a_1
a_2
$\hat{y}$
$\frac{\partial L}{\partial w_3}$
$\frac{\partial L}{\partial z_2}$
$\frac{\partial L}{\partial a_2}$
$\frac{\partial L}{\partial z_3}$
$\frac{\partial L}{\partial \hat{y}}$

Figure 1: Computational Graph for **[3.a]** and annotations (red) for **[3.b]**.

[3.b]
Using the chain rule path from Figure 1 and the given loss function, we can begin to derive the gradient of the loss w.r.t. w_3. Lets begin with the forward pass with our intermediate variables.

$$z_1 = w_1x_1 + w_2x_2 = (-0.65)(1.2) + (-0.55)(-1.1) = -0.1750$$

$$z_2 = w_3x_3 + w_4x_4 = (1.74)(0.8) + (0.79)(0.7) = 1.9450$$

$$a_1 = \sigma(z_1) = \frac{1}{1+e^{-z_1}} = 0.4564$$

$$a_2 = \sigma(z_2) = \frac{1}{1+e^{-z_2}} = 0.8749$$

$$z_3 = w_5\sigma(z_1) + w_6\sigma(z_2) = (-0.13)(0.4564) + (0.93)(0.8749) = 0.7543$$

$$\hat{y} = \sigma(z_3) = \frac{1}{1+e^{-z_3}} = 0.6801$$

Now we can compute the chain rule representation of $\frac{\partial L}{\partial w_3}$.

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial w_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial w_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_3}$$

And now we can compute these terms individually.

$$\frac{\partial L}{\partial \hat{y}} = 2(y-\hat{y})(-1) = -2(y-\hat{y}) = -2(1-0.6801) = -0.6398$$

$$\frac{\partial \hat{y}}{\partial z_3} = \frac{\partial \sigma(z_3)}{\partial z_3} = \sigma(z_3)(1-\sigma(z_3)) = (0.6801)(1-0.6801) = 0.2176$$

$$\frac{\partial z_3}{\partial a_2} = w_6 = 0.93$$

$$\frac{\partial a_2}{\partial z_2} = \frac{\partial \sigma(z_2)}{\partial z_2} = \sigma(z_2)(1-\sigma(z_2)) = 0.8749(1-0.8749) = 0.1094$$

$$\frac{\partial z_2}{\partial w_3} = x_3 = 0.8$$

Finally, we can plug in and compute.

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_3} = (-0.6398)(0.2176)(0.93)(0.1094)(0.8)$$

$$\boxed{\frac{\partial L}{\partial w_3} = -0.0113}$$

[Question 4] Convolutional Neural Network (15 marks)

In this problem, our goal is to estimate the computation overhead of CNNs by counting the FLOPs (floating point operations). Consider a convolutional layer C followed by a max pooling layer P. The input of layer C has 50 channels, each of which is of size 12×12. Layer C has 20 filters, each of which is of size 4×4. The convolution padding is 1 and the stride is 2. Layer P performs max pooling over each of the C's output feature maps, with 3×3 local receptive fields, and stride 1.
Given scalar inputs x_1, x_2, ..., x_n, we assume:

- A scalar multiplication $x_i.x_j$ accounts for one FLOP.
- A scalar addition $x_i + x_j$ accounts for one FLOP.
- A max operation $\max(x_1, x_2, ..., x_n)$ accounts for $n-1$ FLOPs.
- All other operations do not account for FLOPs.

How many FLOPs layer C and P conduct in total during one forward pass, with and without accounting for bias?

The output dimension is given by,

$$\text{output\_size} = \frac{\text{input} - \text{filter\_size} + 2\text{padding\_size}}{\text{stride}} + 1$$

which gives $\frac{12-4+2\cdot 1}{2} + 1 = \frac{10}{2} + 1 = 6$.
To map a single value from the input $50 \times 12 \times 12$ to the output, $20 \times 6 \times 6$, there is a $50 \times 4 \times 4$ kernel used to compute the singular valued output. This corresponds to $50 \times 4 \times 4$ multiplication operations followed by summation. The summation is over the unique products so the total FLOPs from a single mapping must be $2 \times 50 \times 4 \times 4 - 1$, where the -1 is because for n terms, you only have $n-1$ operations to sum them.
This is the operation to compute a single value in the $20 \times 6 \times 6$ output tensor. So for $20 \times 6 \times 6$ values, there would be a linear scaling by the FLOPs per value. So the total becomes $20 \times 6 \times 6 \times (2 \times 50 \times 4 \times 4 - 1)$ FLOPs $= 1,151,280$ FLOPs to compute the output

of layer C without a bias term.
A bias term means that each output value will have an additional value added to it, which is a flop. Thus the output of layer C with a bias term is the same as before with an additional bias term per output value. Since there are $20 \times 6 \times 6$ output values, the total FLOPs with a bias term in layer C becomes $1,151,280 + 20 \times 6 \times 6 = 1,152,000$ FLOPs.

In layer P, we can compute the output shape using the non-padding equation,

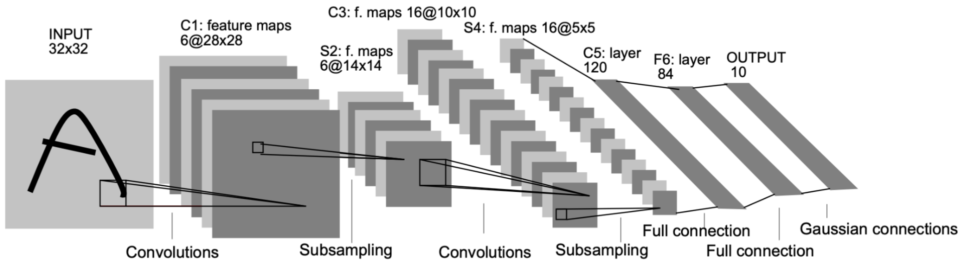
$$\text{output\_size} = \text{input} - \text{filter\_size} + 2\text{padding\_size} + 1$$

which gives $6 - 3 + 2 \cdot 0 + 1 = 4$.
This means there will be $20 \times 4 \times 4$ max-pooling operations. A single max operation over n values constitutes $n - 1$ FLOPs. So for a kernel of 3×3, it would be $9 - 1 = 8$ FLOPs. Thus the pooling layer P requires $8 \times 20 \times 4 \times 4 = 2560$ FLOPs with or without bias.

Thus, layer C and P conduct a total of $1,151,280 + 2560 = 1,153,840$ FLOPs without bias, and $1,152,000 + 2560 = 1,154,560$ FLOPs with bias.

[Question 5] Trainable Parameters (10 marks)

The following CNN architecture is one of the most influential architectures that was presented in the 90s. Count the total number of trainable parameters in this network. Note that the Gaussian connections in the output layer can be treated as a fully connected layer similar to $F6$.



We begin by counting the number of parameters in the first convolution layer. This layer goes from 32×32 to $6 \times 28 \times 28$. This means there are 6 filters, and assuming no padding and a stride of 1, we can compute the kernel size.

$$\text{output} = \text{input} - \text{filter\_size} + 2 \cdot \text{padding} + 1$$

$$\text{filter\_size} = \text{input} - \text{output} + 2 \cdot \text{padding} + 1$$

$$\text{filter\_size} = 32 - 28 + 1 = 5$$

Thus the filter is 5×5. Since we have 6 filters of shape 5×5, we have $6 \times 5 \times 5$ trainable weights in the first layer. Each filter will have a corresponding bias vector so there are

$6 \times 5 \times 5 + 6 = 156$ trainable parameters in the first layer.
The second layer is a sub-sampling layer and assuming max-pooling, there are no trainable parameters here.
The third layer involves convolutions with 16 kernels. The kernel size is $14 - 10 + 1 = 5$, so each kernel is of shape $6 \times 5 \times 5$. There are 16 kernels so there are $16 \times 6 \times 5 \times 5 + 16 = 2416$ learnable parameters in the third layer.
The fourth layer is a sub-sampling layer, so once again, there are no learnable parameters here.
The fifth layer is a linear feed forward from $16 \times 5 \times 5 = 400$ to 120. The number of parameters here is 120 biases plus $120 \cdot 400$ weights. Thus there are 48120 learnable parameters in the fifth layer.
Using the same formulation, the sixth layer has $120 \times 84 + 84 = 10164$ trainable parameters.
And the final layer has $84 \times 10 + 10 = 850$ trainable parameters.
Using these numbers, we can compute the total trainable parameters as

$$\text{total\_params} = 156 + 2416 + 48120 + 10164 + 850 = 61,706.$$

Thus, the total number of trainable parameters is $61,706$.

[Question 6] Logistic Activation Function (10 marks)

For backpropagation in a node with logistic activation function, show that, in order to compute the gradient, as long as we have the output of the node, there is no need for the input.

Hint: Find the derivative of a neuron's output with respect to its inputs.

The logistic, or sigmoid, activation is shown below.

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

Following the hint, lets compute $\sigma'(x)$.

$$\begin{aligned}
\sigma'(x) = \frac{d}{dx}\left(1 + e^{-x}\right)^{-1} &= -1\left(1 + e^{-x}\right)^{-2}\frac{d}{dx}\left(1 + e^{-x}\right) \\
&= -1\left(1 + e^{-x}\right)^{-2}\left(0 - e^{-x}\right) \\
&= \frac{e^{-x}}{(1 + e^{-x})^2} \\
&= \left(\frac{e^{-x}}{1 + e^{-x}}\right)\frac{1}{1 + e^{-x}} \\
&= \left(\frac{(1 + e^{-x}) - 1}{1 + e^{-x}}\right)\sigma(x) \\
&= \left(\frac{(1 + e^{-x})}{1 + e^{-x}} + \frac{-1}{1 + e^{-x}}\right)\sigma(x)
\end{aligned}$$

$$\sigma'(x) = (1 - \sigma(x))\,\sigma(x)$$

Thus, we have shown that the gradient of the logistic function can be computed from the value of the function itself, without knowing the input value. As required.

[Question 7] Hyperbolic Tangent Activation Function (15 marks)

One alternative to the logistic activation function is the hyperbolic tangent function:

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

- (a) What is the output range for this function, and how it differs from the output range of the logistic function?
- (b) Show that its gradient can be formulated as a function of logistic function.
- (c) When do we want to use each of these activation functions?

[7.a]

Let's rewrite the tanh function to better illustrate its range.

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

At $x = 0$, the numerator is 0 and the denominator is 2, thus it is 0 at $x = 0$. As $x \to \infty$, the term $\frac{1}{e^{2x}}$ approaches 0, thus the function is 1 as x goes to infinity. As $x \to -\infty$, the same term becomes $e^{2|x|}$ (the inverse of what it currently is). This means the numerator will be a negative number, and the denominator will be positive. In the limit, this will approach -1.

Thus the range of the tanh function is $\tanh x \in (-1, 1)$ not inclusive.

The sigmoid function at $x = 0$ is $\frac{1}{2}$, and as $x \to \infty$, sigmoid approaches 1 as the exponential term approaches 0. As $x \to -\infty$, the denominator approaches infinity, so the sigmoid function approaches 0.

Thus $\sigma(x) \in (0, 1)$ not inclusive.

These ranges differ as tanh ranges from -1 to 1, while the sigmoid remains positive and ranges from 0 to 1.

[7.b]

Let's first compute the gradient and then see how we write it as a function of the logistic function. We can use trigonometry to simplify the derrivation.

$$\frac{d}{dx}\tanh x = \mathrm{sech}^2 x = 1 - \tanh^2 x$$

So lets try to represent tanh as a function of the logistic function.

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

From the derivations in Question 6, we know that

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

so we can substitue this in.

$$\tanh x = \sigma(2x) - (1 - \sigma(2x))$$

$$\tanh x = 2\sigma(2x) - 1$$

As required.

[7.c]
Sigmoid values from 0 to 1 can be interpreted as probabilities and used to place a likelihood on some classification, especially for binary classification as the inverse probability represents the other class. This is more applicable for the final output layers as this effect isn't observed in intermediate layers. Adding on to this, sigmoid is prone to vanishing gradients as it is really close to 0, so it is less prefered in hidden layers.
Conversely, since tanh is centered around 0 it provides greater flexibility and numerical stability. It also might be better for binary classification of classes $\{1, -1\}$.

Part II: Implementation Tasks (120 marks)

In this question, we train (or fine-tune) a few different neural network models to classify dog breeds. We also investigate their dataset bias and cross-dataset performances. All the tasks should be implemented using Python with a deep learning package of your choice, e.g. PyTorch or TensorFlow.

We use two datasets in this assignment.

1. Stanford Dogs Dataset
2. Dog Breed Images

The Stanford Dogs Dataset (SDD) contains over 20,000 images of 120 different dog breeds. The annotations available for this dataset include class labels (i.e. dog breed name) and bounding boxes. In this assignment, we'll only be using the class labels. Further, we will only use a small portion of the dataset (as described below) so you can train your models on Colab. Dog Breed Images (DBI) is a smaller dataset containing images of 10 different dog breeds.

To prepare the data for the implementation tasks, follow these steps:

1- Download both datasets and unzip them. There are 7 dog breeds that appear in both datasets:

- Bernese mountain dog
- Border collie
- Chihuahua
- Golden retriever
- Labrador retriever
- Pug
- Siberian husky

2- Delete the folders associated with the remaining dog breeds in both datasets. You can also delete the folders associated with the bounding boxes in the SDD.

3- For the 7 breeds that are present in both datasets, the names might be written slightly differently (e.g. Labrador Retriever vs. Labrador). Manually rename the folders so the names match (e.g. make them both *labrador_retriever*).

4- Rename the folders to indicate that they are subsets of the original datasets (to avoid potential confusion if you later want to use them for another project). For example, *SDDsubset* and *DBIsubset*. Each of these should now contain 7 subfolders (e.g. *border_collie*, *pug*, etc.) and the names should match.

5- Zip the two folders (e.g. *SDDsubset.zip* and *DBIsubset.zip*) and upload them to your Google Drive.

You can find sample code working with the SDD on the internet. If you want, you are welcome to look at these examples (particularly the one linked here) and use them as your starting code or use code snippets from them. You will need to modify the code as our questions are asking you to do different tasks, which are not the same as the ones in these online examples. But using and copying code snippets from these resources is fine. If you choose to use this (or any other online example) as your starting code, please acknowledge them in your submission (we provide a copy of this code in the assignment file). We also suggest that before starting to modify the starting code, you run them as is on your data (e.g. DBIsubset) to 1) make sure your dataset setup is correct and 2) to make sure you fully understand the starter code before you start modifying it.

Task I - Inspection (5 marks):
Look at the images in both datasets, and briefly explain if you observe any systematic differences between images in one dataset vs. the other.

The DBI images look far nicer in terms of visual appeal compared to the SDD images. These images are far better in terms of quality and centering and appear to be professionally done. In addition, all the photos in DBI have a single "person of interest" while the SDD dataset includes humans holding their pets, or other things in the way if the whole pet. The DBI set has no obstruction and the pet is clearly visible. SDD also has blurry and older pictures while DBI seems to have stock images one would need to pay for.
From these observations, I would say that the SDD set has more real world images while the DBI set has more professional and ideal images. This highlights a difference in the use case of each dataset. For generative tasks, I would prefer DBI, while for something like Siri TTS with image descriptions, perhaps the SDD dataset is more applicable.

Task II - simple CNN Training on the DBI (25 marks):
Construct a simple convolutional neural network (CNN) for classifying the images in DBI. For example, you can construct a network as follow:

- convolutional layer - 16 filters of size 3×3
- batch normalization
- convolutional layer - 16 filters of size 3×3
- max pooling (2×2)
- convolutional layer - 8 filters of size 3×3
- batch normalization
- convolutional layer - 8 filters of size 3×3
- max pooling (2×2)
- dropout (e.g. 0.5)

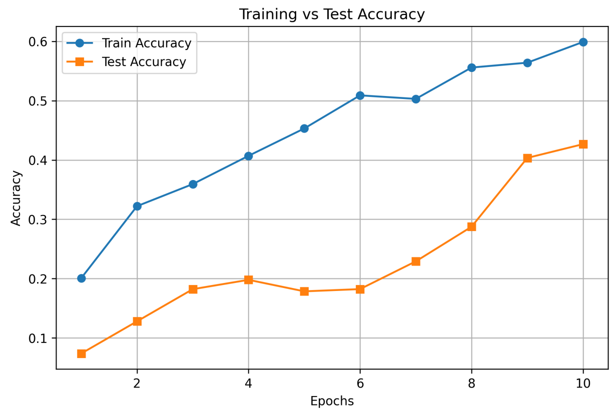
- fully connected (32)
- dropout (0.5)
- softmax

If you want, you can change these specifications; but if you do so, please specify them in your submission. Use RELU as your activation function, and cross-entropy as your cost function. Train the model with the optimizer of your choice, *e.g.*, SGD, Adam, RMSProp, etc. Use random cropping, random horizontal flipping, and random rotations for augmentation. Make sure to tune the parameters of your optimizer for getting the best performance on the validation set.

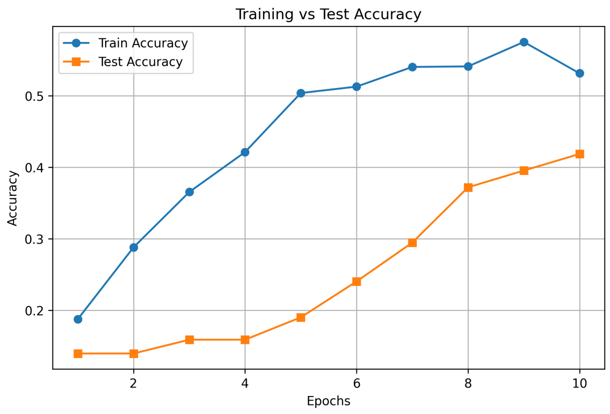
Plot the training, and test accuracy over the first 10 epochs. Note that the accuracy is different from the loss function; the accuracy is defined as the percentage of images classified correctly.

Train the same CNN model again; this time, without dropout. Plot the training and test accuracy over the first 10 epochs; and compare them with the model trained with dropout. Report the impact of dropout on the training and its generalization to the test set.

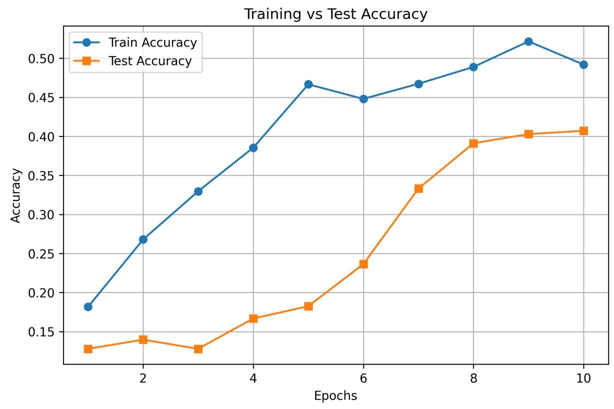
Figure 2 demonstrates the plots for various dropout rates from $p = 0 \in \{0, 0.1, 0.3, 0.5\}$. The plot reveals an interesting dynamic between accuracy and dropout. It seems that there is a hotspot for dropout where it works best. Here, we see that no dropout works well and achieves a high accuracy above 50%. Increasing it to $p = 0.1$ leads to a similar train accuracy, but lower val. accuracy. This indicates that the low dropout might not be effective enough, or it is too soon into training for a significant impact.
When increasing to $p = 0.3$, we see a lower train accuracy, but an identical val. acc. as in $p = 0$. This shows that with this much dropout, despite a lower train acc., the model generalizes better and gets the same val. acc..
Increasing to $p = 0.5$ leads to the worst performance, however there is small discrepancy between train and val acc., indicating that despite this model is overall worse, it is generalizing better than the no/low dropout models.



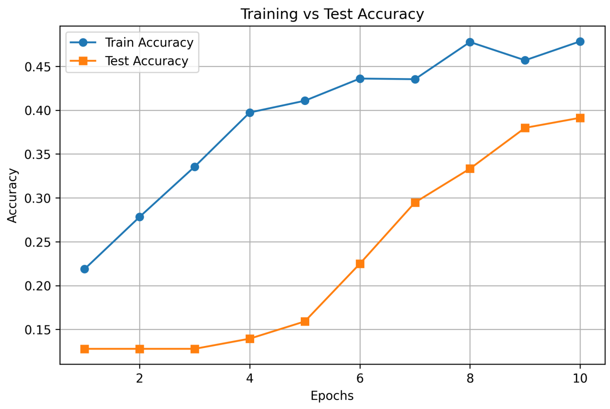
(a) Model 2 with dropout $p = 0$.



(b) Model 2 with dropout $p = 0.1$.



(c) Model 2 with dropout $p = 0.3$.



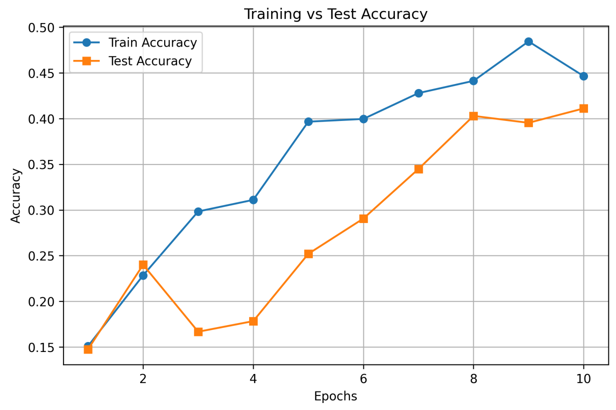
(d) Model 2 with dropout $p = 0.5$.

Figure 2: Model 2 Accuracy plots for 10 epochs.

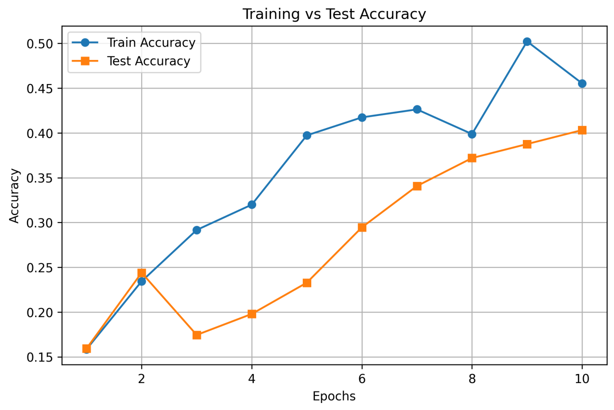
To further investigate, we will study the same effects on another model.

Below, Figure 3 is a different model I trained and its accuracy plots. This model was smaller so the accuracies are lower too, but again we see that the train and val. acc.s are closer when dropout increases. This confirms that dropout helps model generalization.
From this, it is clear that dropout helps model generalization, but might require longer training to achieve a similar performance.

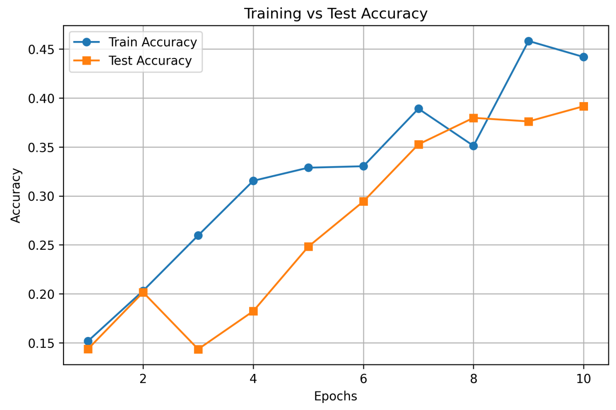
Note that I did change the architecture. Both model 1 and model 2 are different from the specified architecture above. The architectures can be found in "assignment2/task2/model.py" in lines 41 to 98.



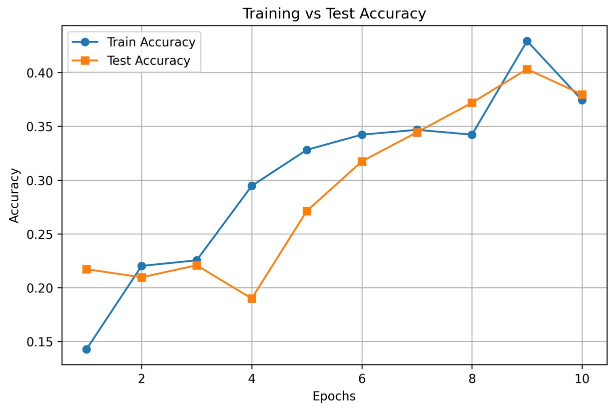
(a) Model 1 with dropout $p = 0$.



(b) Model 1 with dropout $p = 0.1$.



(c) Model 1 with dropout $p = 0.3$.

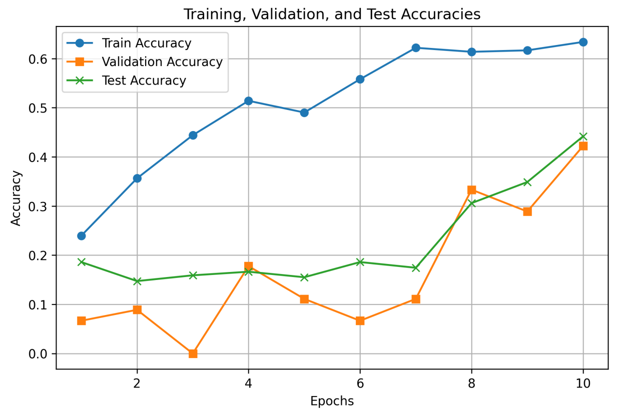


(d) Model 1 with dropout $p = 0.5$.

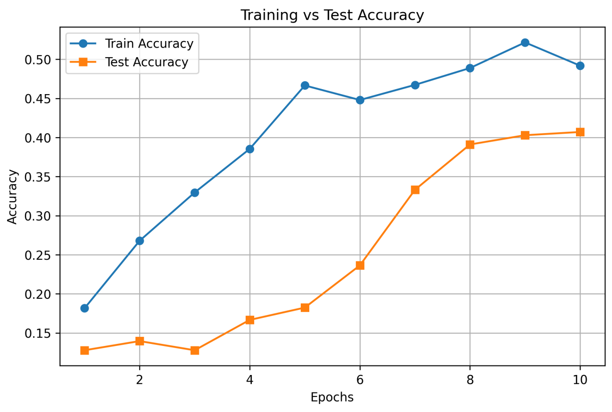
Figure 3: Model 1 Accuracy plots for 10 epochs.

Task III - ResNet Training on the DBI (25 marks):

[**III.a**] (15 marks) ResNet models were proposed in the "Deep Residual Learning for Image Recognition" paper. These models have had great success in image recognition on benchmark datasets. In this task, we use the ResNet-18 model for the classification of the images in the DBI dataset. To do so, use the ResNet-18 model from PyTorch, modify the input/output layers to match your dataset, and train the model from scratch; *i.e.*, do not use the pre-trained ResNet. Plot the training, validation, and testing accuracy, and compare those with the results of your CNN model.



(a) Accuracies of ResNet18.



(b) My CNN model (model 2) Accuracy plots.

Figure 4: Comparing ResNet18 with my CNN model.

Figure 4 above shows a comparison of ResNet18 and my CNN model training to 10 epoch 30. The comparison shows that the CNN model is better but has more or less converged. We know this as the re is a far less gap between the train and test accuracies, and greater consistency. I suspected that the ResNet18 should be better after more training, which is the case when trained to epoch 30. The plots for this are not included but can be generated if desired.

[III.b] (10 marks) Run the trained model on the entire SDD dataset and report the accuracy. Compare the accuracy obtained on the (test set of) DBI, vs. the accuracy obtained on the SDD. Which is higher? Why do you think that might be? Explain very briefly, in one or two sentences.

The final accuracy for ResNet on the DBI dataset is 41.8%, and on the SDD dataset is 27.7%. The DBI set accuracy is higher.
This makes sense because the model was trained on the DBI set and as mentioned in task 1, there are structural differences in the data that make it difficult to transfer the learning over. Despite this, we perfrom well better than random ($\frac{1}{7} = 14\%$) because we are still classifying the same thing so the learned filers are transferable.
As a side note, after training to 30 epochs, the DBI test accuracy is 57.3% and the SDD accuracy is 34.1%.

Task IV - Fine-tuning on the DBI (30 marks):
Similar to the previous task, use the following three models from PyTorch: ResNet18, ResNet34, and ResNeXt32. This time you are supposed to use the pre-trained models and fine-tune the input/output layers on DBI training data. Report the accuracy of these fine-tuned models on DBI test dataset, and also the entire SDD dataset. Discuss the cross-performance of these trained models. For example, are there cases in which two different models perform equally well on the test portion of the DBI but have significant performance differences when evaluated on the SDD?

Model	DBI Accuracy	SDD Accuracy
ResNet18	96.51%	90.76%
ResNet34	95.36%	92.04%
ResNeXt32	99.61%	95.74%

Table 1: Table of model accuracies on DBI and SDD datasets.

Table 1 above summarizes the accruacies of the three models on the two datasets. All models were trained using a 60-30-10 train-val-test split.
The ResNet18 and ResNet34 models perform comparably to one another in the DBI dataset, but the ResNet18 is just over a percent better. Their perfromance on the SDD dataset is flipped as the ResNet34 actually outperfroms the ResNet18 by over 1 percent. This suggests that the deeper ResNet34 is able to generalize better to unseen data, as expected.
ResNeXt32 significantly outperforms both on both datasets, highlighting the advantage of grouped convolutions for robust feature extraction. This suggests that increased model complexity leads to improved generalization across datasets, making ResNeXt32 the optimal choice for cross-performance tasks.
There aren't any extreme cases where two models perform equally well on DBI but significantly different on SDD, though ResNet18 and ResNet34 certainly meet this criterion to a lesser extent.

Task V - Dataset classification (20 marks):
Train a model that – instead of classifying dog breeds – can distinguish whether a given image is more likely to belong to SDD or DBI. To do so, first, you need to divide your data into training and test data (and possibly validation if you need those for tuning the hyper-parameters of your model). You need to either reorganize the datasets (to load the images using *torchvision.datasets.ImageFolder*) or write your own data loader function. You can start from a pre-trained model (of your choice) and fine-tune it on the training portion of the dataset. Include your network model specifications in the report, and make sure to include your justifications for that choice. Report your model's accuracy on the test portion of the dataset.

My model is a fine tuned ResNeXt32 using a 60-30-10 train-val-test split. I chose this because it worked well on the first run so there is no need to experiment further. Nonetheless, I did run two other models, as seen in Table 2.

Model	Test Accuracy
ResNet18	88.12%
ResNet34	88.51%
ResNeXt32	90.62%

Table 2: Table of test accuracy for models in predicting dataset.

This shows that my model is able to easily distinguish between the two datasets.

Task VI - How to improve performance on SDD? (10 marks):
If our goal were to have good performance on the SDD dataset, briefly discuss how to work towards this goal in each of the following cases:

- 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).
- At training time, we have access to the entire DBI dataset and a small portion (e.g. 10%) of the SDD dataset.
- 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.

(a)
This is a difficult task. One approach is to train on the entire dataset, then fine tune (even last layers only) on a subset of DBI images that are most similar to the description of the SDD dataset. This way we learn the entire DBI set and then are able to transfer this knowledge to more applicable images.
Another approach is to just train on the applicable images. However, this might be a small subset and would likely lead to poorer performance as the other less relevant images allow better feature extraction.

(b)
We can use on the entire DBI set plus the 10%. None of the SDD images should be in the test set as this is not a good allocation of it, rather they should be spread within the train and validation. It may even be beneficial to not train on the SDD set, rather to use it as validation and see how different architectures perform. This is definitely something I would play with and see where it leads. If the SDD data is used in the train set, then we should use data duplication to maximize it's utility.
Additionally, we could use clustering/supervised methods to pick the images from DBI that are most similar to SDD, as mentioned in part (a).

(c)
Given that we don't have the labels, we can try to make them ourselves using supervised learning techniques and then proceed as in (b). This obviously might have errors as we are using labels that are not known to be the ground truth.
We can also use the SDD images to find the most similar looking images in the DBI set and fine tune on these.
Also, we can manually label the images as they are dogs and humans would have no trouble doing this. It would of course take a lot of time and effort.

Task VII - Discussion (5 marks):
Briefly discuss how some of the issues that were examined in this exercise can have implications in real application, e.g. as related to bias or performance. For example, consider

the case where available training datasets are collected in one setting (e.g. the University of Toronto) and the goal is to deploy trained models in another setting (e.g. a hospital).

The scenario provided demonstrates a great example of the impact of data on a models bias and limitations. Data used in training has a significant impact on the model's usability. Things like camera angle, lighting, content of the image, and even the camera itself all play a role in the models usability.
Task II and III.a had us train and test on the same datasets, and even in just 10, epochs, we saw good perfrmance. However, Task III.b and IV used different test datasets and had lower accuracies than before. This shows that true generalization is a non-trivial task in real-world machine learning.
This means when training models for real-world applications, we must curate our dataset to that application. For example, if we are designing facial recognition for a smartphone, our dataset should have an appropriate size and distance, and other conditions for when people use their phones. In contrast, if we are developing a model for a dating app to find someone perfect match, for example, then we anticipate far clearer and more effort in the pictures, so we should train our model with that kind of dataset.

Summary of implementation tasks:

The train/val/test split is up to you. You can use the same split used in the sample code linked, i.e. train: 60%. validation 10%, test: 30%. But you can do other (reasonable) splits too if you want. Just specify in your report what you did.

	what we want to do	Train	Validation	Test
Task II	dog breed classification	DBI	DBI	DBI
Task III.a	dog breed classification	DBI	DBI	DBI
Task III.b	dog breed classification	DBI	DBI	SDD
Task IV	dog breed classification	DBI	DBI	both
Task V	dataset classification			