[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?

Gaussian pyramid levels are: $I_0, I_1, I_2 \cdots I_n$
Laplacian pyramid levels are: $L_0, L_1, L_2 \cdots L_n$
For each level, $L_k = I_k - \text{upsample}(I_{k+1})$
where upsample means expanded version of I_{k+1}, resized to match I_k.

Laplacian pyramid: $L_{n-1} = I_{n-1} - \text{upsample}(I_n)$
rearrange to reconstruct I_{n-1}
$I_{n-1} = L_{n-1} + \text{upsample}(I_n)$

To reconstruct any level I_k we need the laplacian at that level (L_k), and the gaussian image at the next level (I_{k+1})

The minimum information needed is
all laplacian pyramid levels: $L_0, L_1, L_2 \cdots L_n$
the final gaussian pyramid level: I_n

$I_{n-1} = L_{n-1} + (\text{upsample}(I_n)$
$I_{n-2} = L_{n-2} + \text{upsample}(I_{n-1})$
Continue this recursively up to $k=0$
closed form:
$I_0 = L_0 + \text{Upsample}(L_1 + \text{upsample}(L_2 + \cdots + \text{upsample}(L_{n+1} + I_n) \cdots))$

[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.

x = input vector
W_k = weight matrix of layer k
b_k = bias vector at layer k
h_k = activation of neurons at layer k

Forward Propagation:

$$h_1 = W_1 x + b_1$$
$$h_2 = W_2 h_1 + b_2$$
$$\vdots$$
$$h_L = W_L h_{L-1} + b_L$$

Substitute h_1 into h_2

$$h_2 = W_2(W_1 x + b_1) + b_2$$
$$= W_2 W_1 x + W_2 b_1 + b_2$$

Substitute for all layers

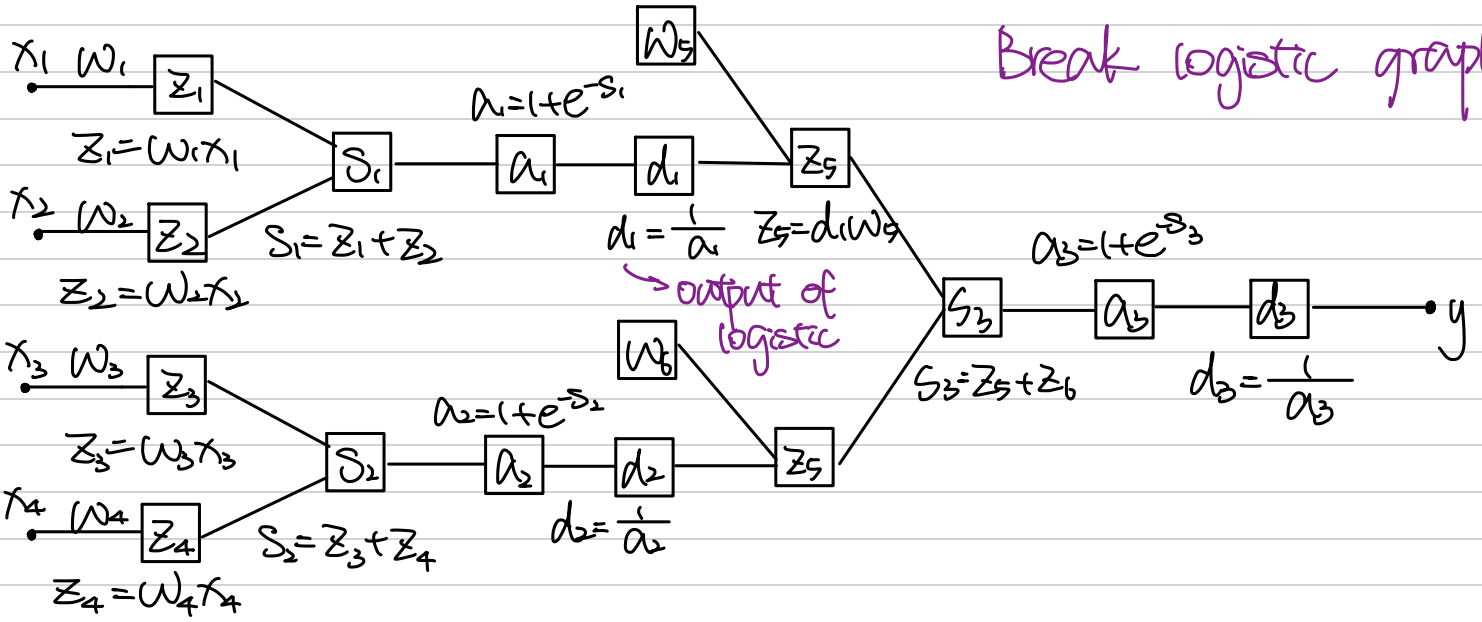
$$h_L = \underbrace{W_L W_{L-1} \cdots W_2 W_1 x}_{W_{eff}} + \underbrace{W_L W_{L-1} \cdots W_2 b_1 + W_L W_{L-1} \cdots b_2 + \cdots + b_L}_{b_{eff}}$$

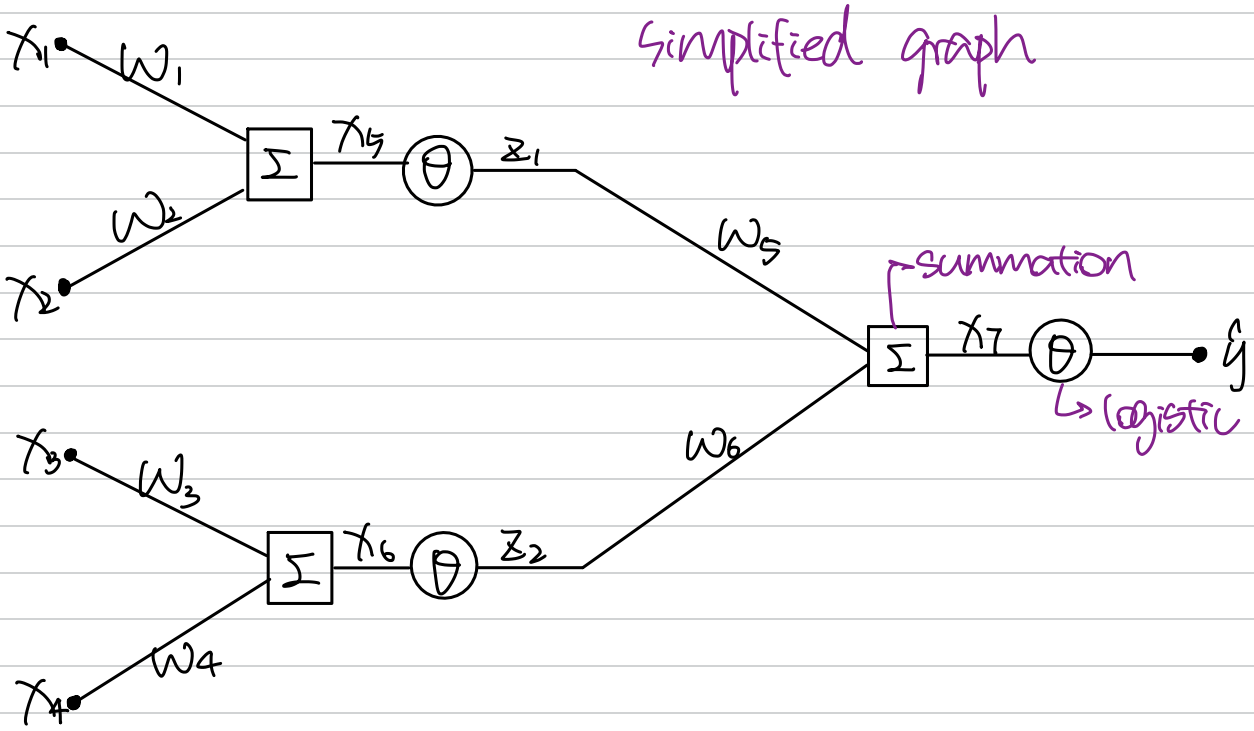
$W_{eff} = W_L W_{L-1} \cdots W_2 W_1$ → effective weight matrix
$b_{eff} = W_L W_{L-1} \cdots W_2 b_1 + W_L W_{L-1} \cdots b_2 + \cdots + b_L$
↳ effective bias

$$h_L = W_{eff} x + b_{eff}$$

∴ a deep linear network collapse into a single layer network with an equivalent weight and bias.

[**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.)





layer 1 layer 2

[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.

Forward Propagation

$x_5 = x_1 w_1 + x_2 w_2 = -0.65 \times 1.2 + (-0.55)(-1.1) = -0.1750$

$x_6 = x_3 w_3 + x_4 w_4 = 0.8 \times 1.74 + 0.7 \times 0.79 = 1.9450$

$z_1 = \sigma(x_5) = \sigma(-0.175) = \frac{1}{1+e^{-0.175}} = 0.4564$

$z_2 = \sigma(x_6) = \sigma(1.945) = \frac{1}{1+e^{-1.945}} = 0.8749$

$x_7 = z_1 \cdot w_5 + z_2 \cdot w_6 = 0.4564 \cdot (-0.13) + 0.8749 \cdot 0.93 = 0.7543$

$y = \sigma(x_7) = \frac{1}{1+e^{-0.7543}} = 0.6801$

Back Propagation

$L = (y - \hat{y})^2 = (-0.6801)^2 = 0.1023$

$\frac{\partial L}{\partial \hat{y}} = 2(y - \hat{y}) = -2(-0.68) = -0.6398$

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

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

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

$= (1 - \sigma(x)) \cdot \sigma(x)$

$\frac{\partial L}{\partial x_7} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_7} = 2(y - \hat{y})\, \sigma'(y)$

$= -0.6398 \cdot ((-\sigma(x_7)) \cdot \sigma(x_7)$

$= -0.6398 \cdot ((-0.6801) \cdot 0.6801 = -0.1592$

$\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial x_7} \cdot \frac{\partial x_7}{\partial w_6} = \frac{\partial L}{\partial x_6} z_2 = -0.1592 \cdot 0.8749 = -0.1248$

$\frac{\partial L}{\partial x_6} = \frac{\partial L}{\partial x_7} \cdot \frac{\partial x_7}{\partial x_6} = \frac{\partial L}{\partial x_7} \cdot w_6 \cdot \sigma'(x_6) = -0.1592 \cdot (1 - \sigma(x_6)) \sigma(x_6) \cdot w_6$

$= -0.1592 \cdot ((1 - 0.8749) \cdot 0.8749 \cdot 0.93$

$= -0.0142$

$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial x_6} \cdot \frac{\partial x_6}{\partial w_3} = \frac{\partial L}{\partial x_6} \cdot x_3 = -0.0142 \times 0.8 = -0.0113$

[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?

Convolutional layer

Input = 12x12x50

Filter = 4x4x50x20

Padding : 1

Stride : 2

Output size = $\frac{\text{Input size} - \text{kernel size} + 2\times\text{padding}}{\text{Stride}} + 1$

$= \frac{12 - 4 + 2\times 1}{2} - 1 = 6$

Output feature map size 6x6, and there are 20 output feature maps.

Each output pixel requires:

multiplication : 4x4x50 = 800

addition : 4x4x50 - 1 = 799

If including bias, add 1 more addition per element.

Total FLOPs for layer C:

Without bias:

multiplications : 4x4x50 x (6x6x20) = 576000

additions : (4x4x50 - 1) x (6x6x20) = 575280

576000 + 575280 = 1151280 FLOPs

With bias

1151280 + (6x6x20) = 1151280 + 720 = 1152000 FLOPs

Max pooling layer:

Input : 6x6x20

pool size : 3x3

Stride : 1

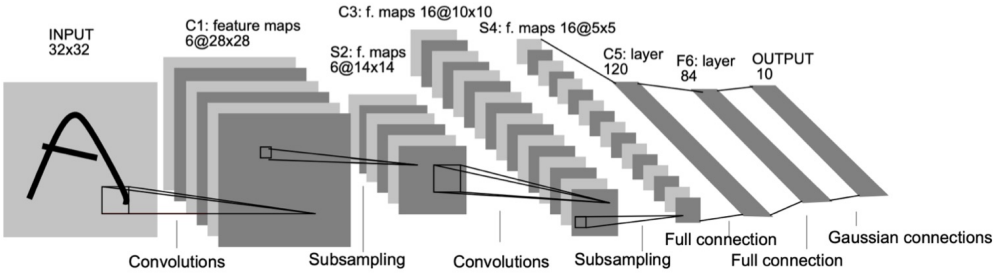
output size : $\frac{6-3}{1} + 1 = 4$

Each max operation over 3×3 area requires 8 comparisons
Output size = 4×4, and there are 20 channels.
Total FLOPs = 4×4×20×8 = 2560

Total FLOPs for both layers:
With bias: 1151280 + 2560 = 1153840 FLOPs
Without bias: 1152000 + 2560 = 1154560 FLOPs

[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 $F6$.



C1 convolutional layer:
Input: 32x32x1
Output: 6 feature maps of 28x28
Filter size: 32 - 28 + 1 = 5 5x5
Parameters: (filter size * filter size * input channel + bias) * output channel
= (5x5x1 + 1) x6 = 156

S2 Subsampling layer
No trainable parameters.

C3 convolutional layer
Input: 14x14x6
Output: 16 feature maps of 10x10
Filter size: 14 - 10 + 1 = 5 5x5
Parameters: (filter size * filter size * input channel + bias) * output channel
= 5x5x6 + 1) x16
= 2416

S4 Subsampling layer
No trainable parameters.

C5 Fully Connected Layer
Input: 5x5x16
Output: 120
Parameters: (5x5x16+1)x120 = 48120

C6 Fully Connected Layer
Input: 120
Output: 84

parameters : (120 + 1) · 84 = 10164

C7 Gaussian connection

Input : 84

Output : 10

parameters = 84 + 1) × 10 = 850

Total parameters : 156 + 2416 + 48120 + 10164 + 850

= 61706

[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.

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

where z is the weighted sum of inputs

$$z = \sum_i w_i x_i + b$$

$$y = \sigma(z)$$

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

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

$$= \frac{e^{-z}}{1+e^{-z}} \cdot \frac{1}{1+e^{-z}}$$

$$= \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) \cdot \frac{1}{1+e^{-z}}$$

$$= (1 - \sigma(z)) \cdot \sigma(z)$$

$$= (1-y) \cdot y$$

Gradient of the logistic function only depends on the output y, not on the input x_i or weight sum z.

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

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

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

- (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?

a) Range of $\tanh(x)$:

as $x \to \infty$, $e^{-2x} \to 0$ $\quad \tanh(x) = \frac{1-0}{1+0} = 1$

as $x \to -\infty$, $e^{-2x} \to \infty$ $\quad \tanh(x) = \frac{1-\infty}{1+\infty} = -1$

the range of $\tanh(x)$ is $(-1, 1)$

Range of $\sigma(x)$:

as $x \to \infty$, $e^{-2x} \to 0$ $\quad \sigma(x) = \frac{1}{1+0} = 1$

as $x \to -\infty$, $e^{-2x} \to \infty$ $\quad \sigma(x) = \frac{1}{1+\infty} = 0$

the range of $\sigma(x)$ is $(0, 1)$

tanh is centered at zero with range $(-1, 1)$, while logistic function is positive with range $(0, 1)$

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

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

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

$$\frac{\partial}{\partial x}\tanh(x) = \frac{\partial}{\partial x}(2\sigma(2x) - 1)$$

$$= 2\sigma'(2x)$$

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

$$= 4(1 - \sigma(2x)) \cdot \sigma(2x)$$

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

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

$$\frac{\partial}{\partial x}\tanh(x) = 4 \cdot (1 - \frac{\tanh(x)+1}{2}) \cdot \frac{\tanh(x)+1}{2}$$

$$= (4 - 2(\tanh(x)+1) \cdot \frac{\tanh(x)+1}{2}$$

$$= 2\tanh(x) + 2 - (\tanh(x)+1)^2$$

$$= 2\tanh(x) + 2 - \tanh^2(x) - 2\tanh(x) - 1$$

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

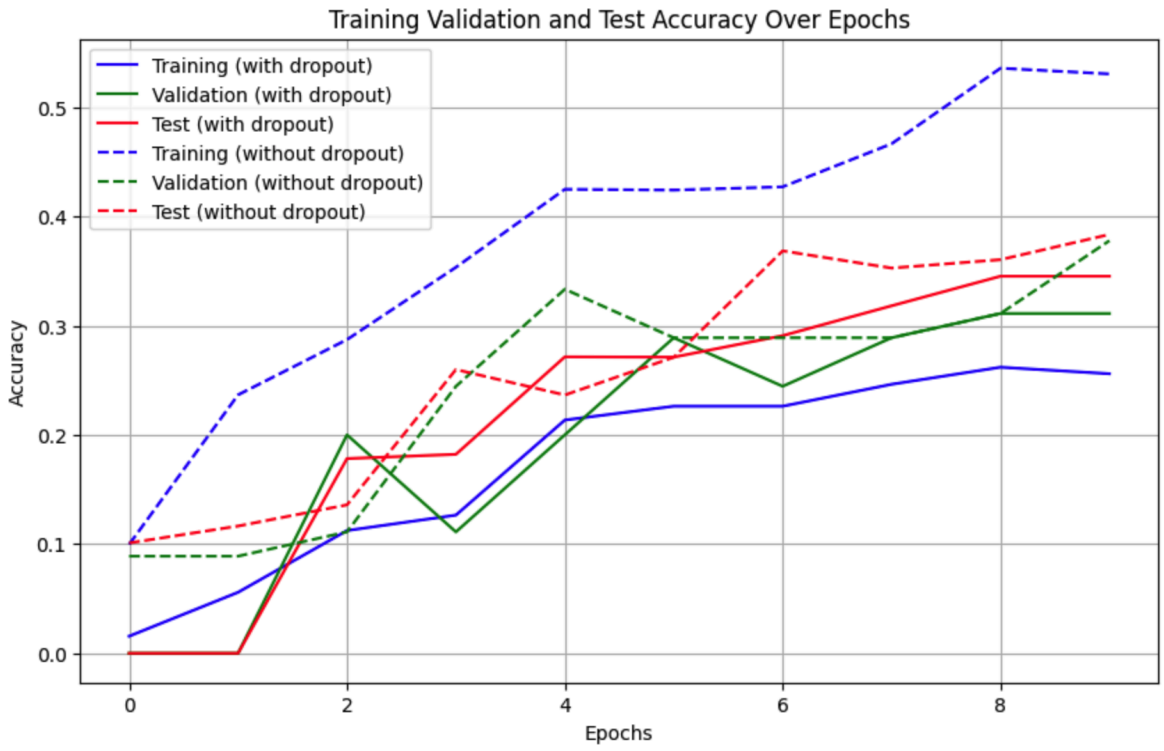
C) Use sigmoid when
- used in binary classification problems.
- dealing with data that is naturally bounded between 0 and 1
- In gates of LSTM/GRU cells where values need to be in [0, 1]
- probabilistic interpretation is needed

Use tanh when
- zero-centered inputs are important.
- stronger gradient is needed near the origin.
- susceptible to vanishing gradient but less than sigmoid.

1. Inspection

	SDD	DBI
Image Quality and Resolution	SDD images have higher resolution and more consistent image quality.	DBI images show more variation in image quality. Some images appear to be from amateur photographers or casual pet photos.
Background	SDD images have more controlled backgrounds, and with dogs as clear focal point.	DBI contains more natural settings with varied backgrounds from indoor home environments to outdoor scenes.
Dog Poses	SDD images show the full body of the dog in a side or standing pose. The breed-specific features are clearly visible.	DBI has more variety in poses, including close-up shots, action shots, and partial views of the dogs.
Lighting	SDD maintains more consistent lighting conditions across images.	DBI shows greater variation in lighting from indoor lighting to natural lighting conditions.

2. CNN Training on the DBI

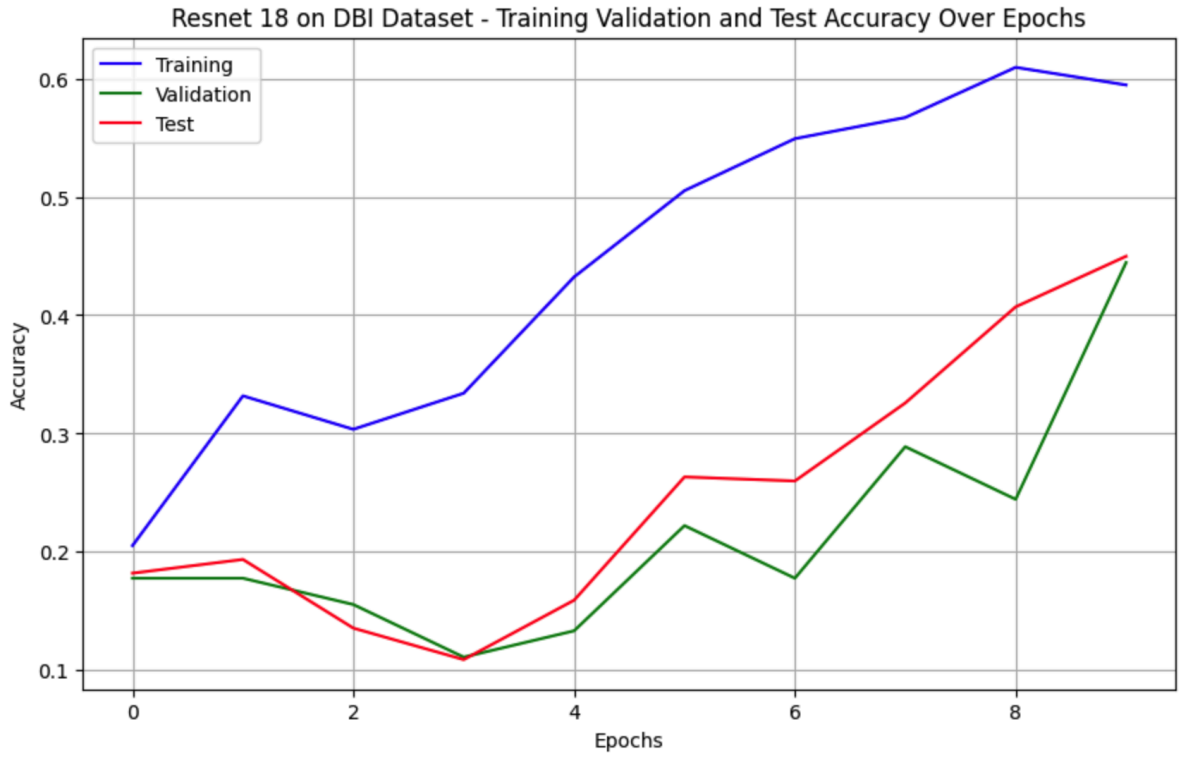


The model trained without dropout achieves higher training accuracy compared to the model trained with dropout, because dropout randomly disables neurons during training. This makes it harder for the model to memorize training data. Additionally, the model trained with dropout

achieves more stable and higher test accuracy compared to the model trained without dropout. The without dropout model shows greater fluctuations in test accuracy. This indicates that it is possible to overfit the training data. The model without dropout shows a gap between training and test accuracy; whereas, the model with dropout has a smaller gap between training and test accuracy. This demonstrates that dropout has better generalization. Dropout acts as a regularizer, preventing overfitting and helping the model generalize better to unseen data.

3. ResNet Training on the DBI

a.

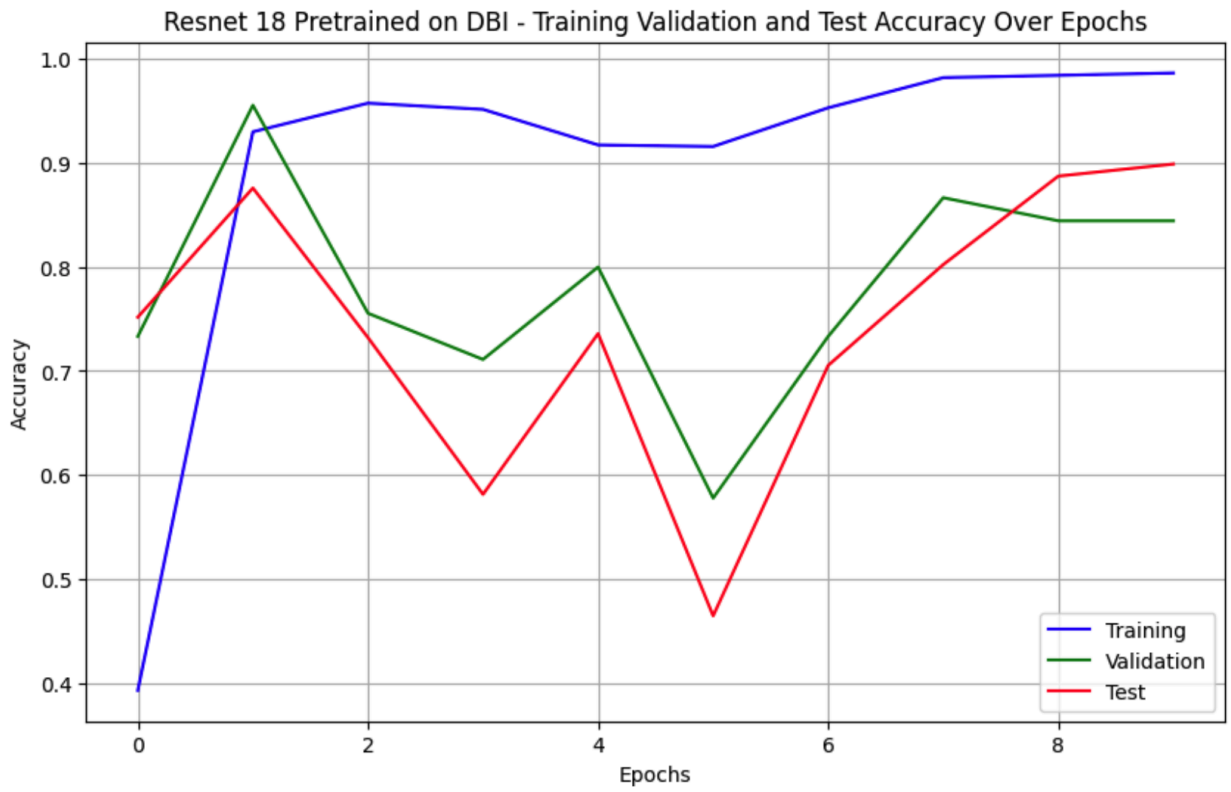


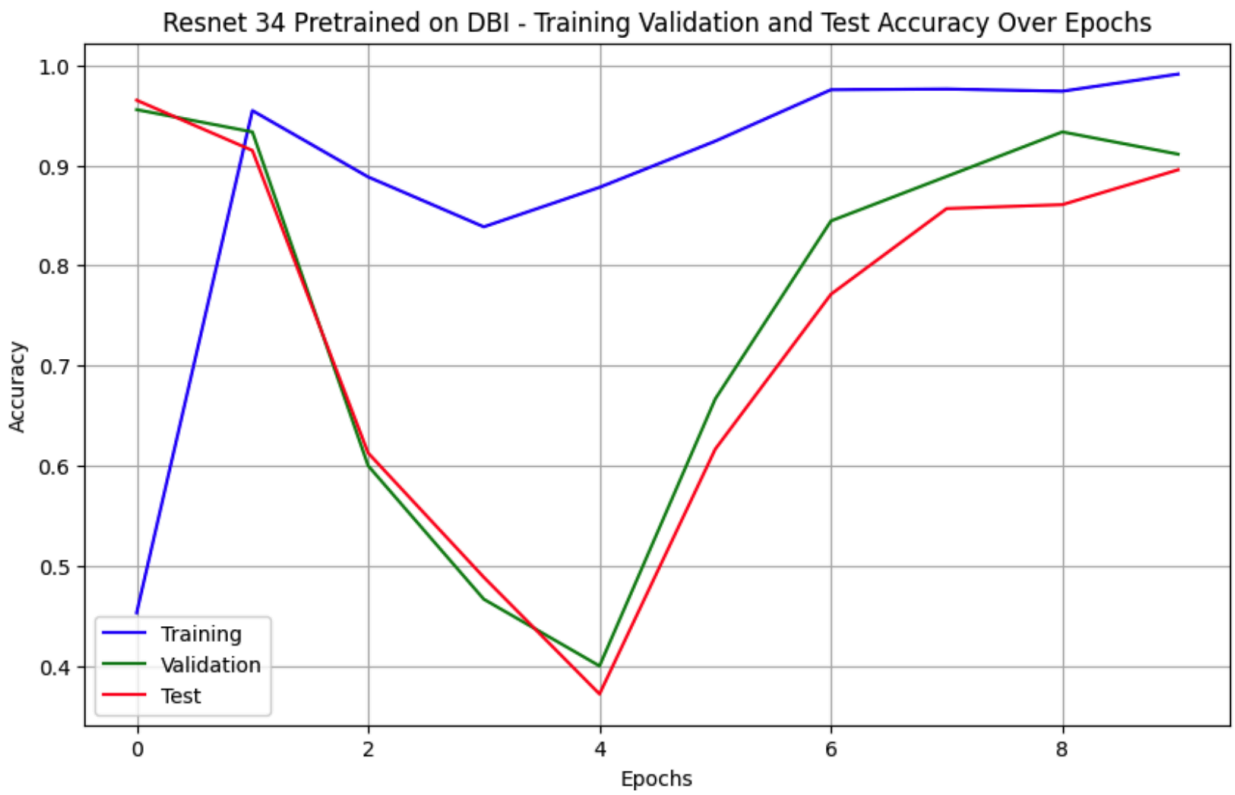
CNN training accuracy increases more slowly with dropout, peaking below 40% in 10 epochs, while without dropout, it surpasses 50%. ResNet-18 Training accuracy improves significantly, reaching over 60% by epoch 10. ResNet-18 trains faster and achieves higher training accuracy compared to CNN.
CNN validation accuracy shows moderate growth, staying around 30-35%. ResNet-18 starts low but gradually improves, reaching nearly 40% in later epochs. ResNet-18 generalizes better to validation data. CNN struggles more with validation accuracy, especially when dropout is used. CNN Model peaks around 35-40% with dropout and slightly higher without dropout. ResNet-18 Model peaks near 40%, showing steady improvement. ResNet-18 achieves comparable or better test accuracy than CNN.

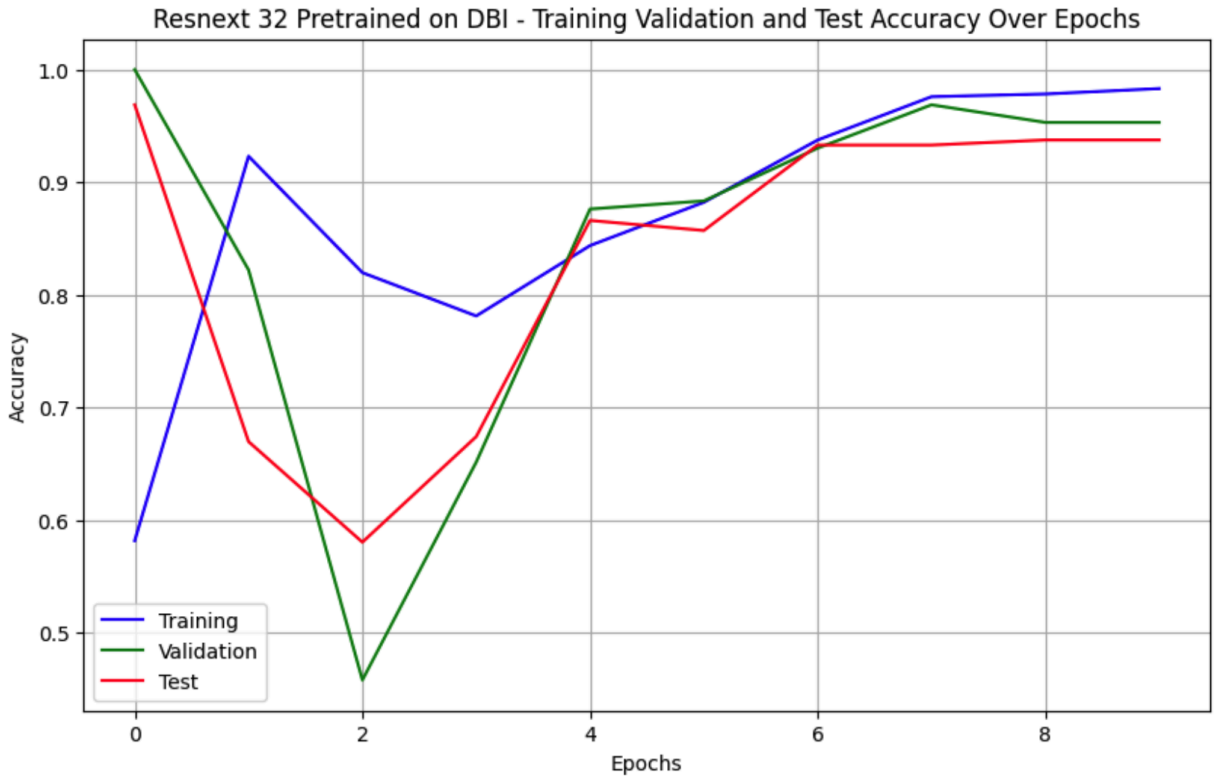
b.

The accuracy on the SDD dataset (19.30%) is significantly lower than the accuracy on the DBI test set (~40%). This suggests that the model trained on DBI does not generalize well to SDD. This is likely due to differences in dataset characteristics such as image distribution, class imbalance, or feature complexity. The lower accuracy on SDD may indicate that the model has overfitted to DBI-specific features.

4. Fine-tuning on the DBI



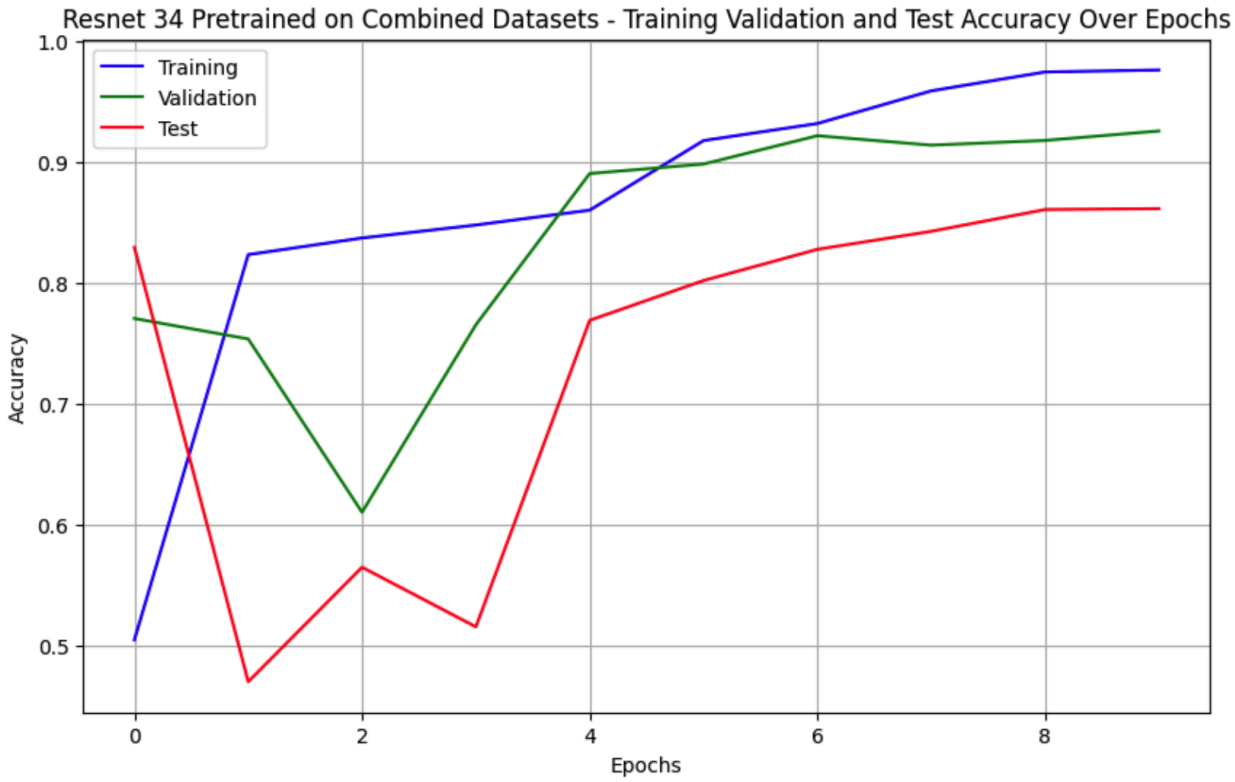

Resnet 34 Pretrained on DBI - Training Validation and Test Accuracy Over Epochs
Accuracy
Epochs
Training
Validation
Test

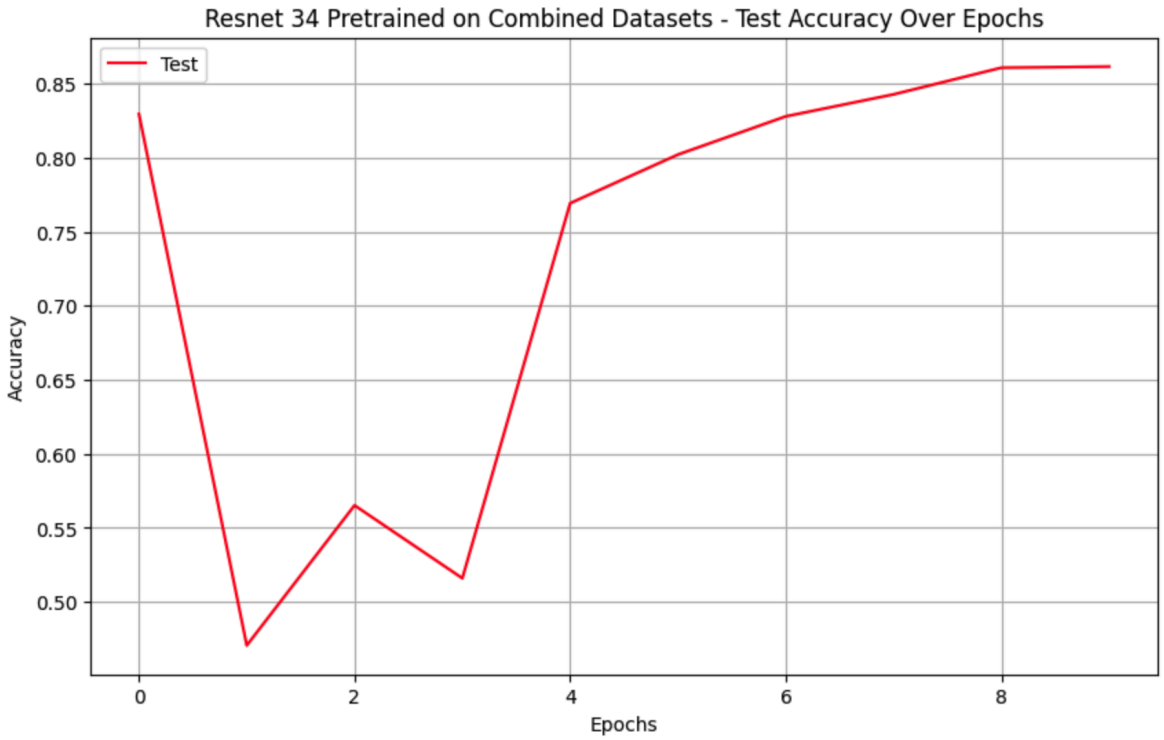

Resnext 32 Pretrained on DBI - Training Validation and Test Accuracy Over Epochs
Accuracy
Epochs
Training
Validation
Test

Model	Accuracy on SDD
Resnet 18 Pretrained	35.0%
Resnet 34 Pretrained	31.3%
Resnext 32 Pretrained	36.5%

All three models show similar training patterns on DBI, with rapid initial learning followed by fluctuations. ResNeXt32 achieves the highest final test accuracy around 93%, followed by ResNet34 (89%), and ResNet18 (85%). ResNet18 shows volatility in validation/test performance, with dramatic drops around epoch 5. ResNet34 experiences a severe performance dip around epoch 4 but recovers more smoothly. ResNeXt32 demonstrates the most stable learning trajectory after epoch 4.
All models show similar patterns in their DBI training curves initial spike, dip around epoch 4, then recovery. However, their final SDD performances don't directly correlate with their training stability on DBI. They all experience a dramatic performance drop (55-58%) when tested on SDD. This suggests a fundamental domain shift between DBI and SDD datasets.

5. Dataset classification





I chose Resnet 34 as the pretrained model, because it is a good balance between model complexity and computational efficiency. The model is well-suited for fine-tuning on a binary classification task like distinguishing between SDD and DBI images. Its deeper architecture compared to ResNet-18 allows for capturing more complex hierarchical features. In this way, it improves generalization without significantly increasing computation cost. On the other hand, ResNet-18 was not chosen because it is shallower. This limits its ability to learn fine differences between the two datasets and increasing the risk of underfitting. While ResNeXt-32 offers a more advanced architecture with grouped convolutions, it was not selected because its added complexity and computational requirements. The computation overhead outweighs the potential benefits for this specific task. Given the relatively small dataset, a model as deep as ResNeXt-32 could lead to overfitting, whereas ResNet-34 strikes the right balance between performance and training efficiency.

The choice of network specifications, including the learning rate, optimizer, weight decay, and gradient clipping, was fine-tuned to optimize model performance. A learning rate of 0.001was chosen for stable convergence. This allows the model to adapt effectively without large oscillations or slow updates. The Adam optimizer was selected due to its adaptive learning rate capabilities. This makes it well-suited for fine-tuning pre-trained models like ResNet-34, because it efficiently handles noisy gradients and converges faster than standard SGD. To prevent overfitting, a weight decay of 10^−4 was applied, ensuring that the model's parameters do not grow excessively large, which helps maintain generalization on the test set. Finally, gradient clipping at 0.1 was used to prevent exploding gradients. This stabilizes training and ensuring smooth learning updates without excessively restricting weight adjustments.

6. How to improve performance on SDD

Case 1:

We can focus on making the model robust to the known differences between datasets, like the indoor outdoor shift and varying camera angles. Additionally, we can use data augmentation techniques that simulate these differences. We can also implement domain randomization techniques to help the model generalize better to new environments.

Case 2:

We can use the labeled SDD portion as a validation set to identify where the model struggles most with domain shift. Moreover, we can implement domain adaptation techniques treating DBI as source domain and SDD as target domain. We can use the labeled SDD portion to fine-tune the model after initial training on DBI. In the meantime, we need to balance the training to prevent overfitting to the small SDD portion.

Case 3:

We can use unsupervised domain adaptation techniques to bridge the gap between datasets, and use consistency regularization approaches where the model should make similar predictions under different transforms of the unlabeled SDD data. We can also implement techniques like entropy minimization on the unlabeled SDD portion to encourage more confident predictions.

7. Discussion

The differences between training and deployment environments can have real-world implications for model performance and fairness. Models trained on university data might underperform in hospitals due to different lighting conditions, equipment layouts, and traffic patterns. Hospitals have unique features like medical equipment, wheelchairs, and gurneys that may be rare or absent in university settings. The model might fail to account for time-sensitive situations like medical emergencies where normal movement patterns are disrupted. University populations tend to skew younger and more able-bodied compared to hospital populations. Hospitals see more people with mobility aids, varied walking speeds, and assisted movement. This could lead to biased or unsafe predictions for elderly patients, people with disabilities, or medical staff assisting patients.