# UCS761 : Lab 7 - From Numbers to Vision: Building, Breaking, Comparing

**Name:** [Himanshu Bansal] | **Roll No:** [102303786]

## PART 1: Deep Networks on Numeric Data

### 1.1 Dataset Summary (N=3000)

- **Input:** x1, x2 sampled from a Uniform distribution (-2, 2).
- **Target Logic:** y = 1 if ($x1^2 + x2^2 > 1.5$), otherwise 0.
- **Splits:** Train (2100), Validation (450), Test (450).

### 1.2 Task 1B: Parameter Derivations (Mandatory)

*Using Formula: (Input_Neurons × Output_Neurons) + Biases for each layer.*

**2-Layer Network (Hidden=16):**

- **L1 (Input to Hidden):** (2 × 16) + 16 = 48
- **L2 (Hidden to Output):** (16 × 1) + 1 = 17
- **Total: 65 Parameters**

**5-Layer Network (Hidden=16):**

- **L1:** 48
- **L2, L3, L4 (Hidden to Hidden):** 3 × [(16 × 16) + 16] = 816
- **L5 (Output):** 17
- **Total: 881 Parameters**

**10-Layer Network (Hidden=16):**

- **L1:** 48
- **L2 through L9:** 8 × [(16 × 16) + 16] = 2176
- **L10 (Output):** 17
- **Total: 2241 Parameters**

### 1.3 Gradient Norm Analysis (The Vanishing Gradient)

- **Observation:** In the 10-layer Sigmoid network, the **Frobenius norm of the Gradient** for Layer 10 was significantly larger (approximately 10^-2) than for Layer 1 (approximately 10^-8).

- **Behavior:** The gradient "vanished" as it flowed backward from the output toward the input.
- **Explanation:** Because the derivative of the Sigmoid function is at most 0.25, multiplying it 10 times during backpropagation (via the Chain Rule) causes the gradient to shrink exponentially (0.25^10). This effectively prevents the weights in the first few layers from updating, stalling the learning process.

| **Model** | **Activation** | **Optimizer** | **Parameters** | **Train Acc** | **Val Acc** | **Test Acc** |
|---|---|---|---|---|---|---|
| 2-layer | ReLU | SGD | 33 | 0.5567 | 0.6111 | 0.6000 |
| 2-layer | ReLU | Momentum | 33 | 0.8614 | 0.9044 | 0.8978 |
| 2-layer | Sigmoid | SGD | 33 | 0.7043 | 0.7489 | 0.7400 |
| 2-layer | Sigmoid | Momentum | 33 | 0.7043 | 0.7489 | 0.7400 |
| 5-layer | ReLU | SGD | 249 | 0.7052 | 0.7489 | 0.7400 |
| 5-layer | ReLU | Momentum | 249 | 0.8895 | 0.9200 | 0.9089 |
| 5-layer | Sigmoid | SGD | 249 | 0.7043 | 0.7489 | 0.7400 |
| 5-layer | Sigmoid | Momentum | 249 | 0.7043 | 0.7489 | 0.7400 |
| 10-layer | ReLU | SGD | 609 | 0.7043 | 0.7489 | 0.7400 |
| 10-layer | ReLU | Momentum | 609 | 0.9833 | 0.9867 | 0.9867 |
| 10-layer | Sigmoid | SGD | 609 | 0.7043 | 0.7489 | 0.7400 |
| 10-layer | Sigmoid | Momentum | 609 | 0.7043 | 0.7489 | 0.7400 |

## 1.4 part

1. Does increasing depth always improve validation performance?

No. Increasing depth improved validation performance only when using ReLU with Momentum. for sigmoid activation and plain SGD, increasing depth gave no improvement and performance remained around 0.74 validation accuracy. So depth alone does not guarantee better performance. Proper activation and optimizer are necessary.

2. What happens to test performance as depth increases?

For ReLU + Momentum:
2-layer → 0.8978
5-layer → 0.9089
10-layer → 0.9867

Test accuracy improved significantly with depth.However, for SGD and sigmoid networks, test accuracy stayed around 0.74, showing no benefit from increased depth. so depth improves generalization only when gradient flow is stable.

3. Do sigmoid networks degrade faster with depth?

Yes. All sigmoid networks (2, 5, and 10 layers) remained stuck around 0.74 test accuracy, regardless of optimizer. This indicates vanishing gradient problem, where gradients shrink as they propagate through many layers, preventing effective learning.

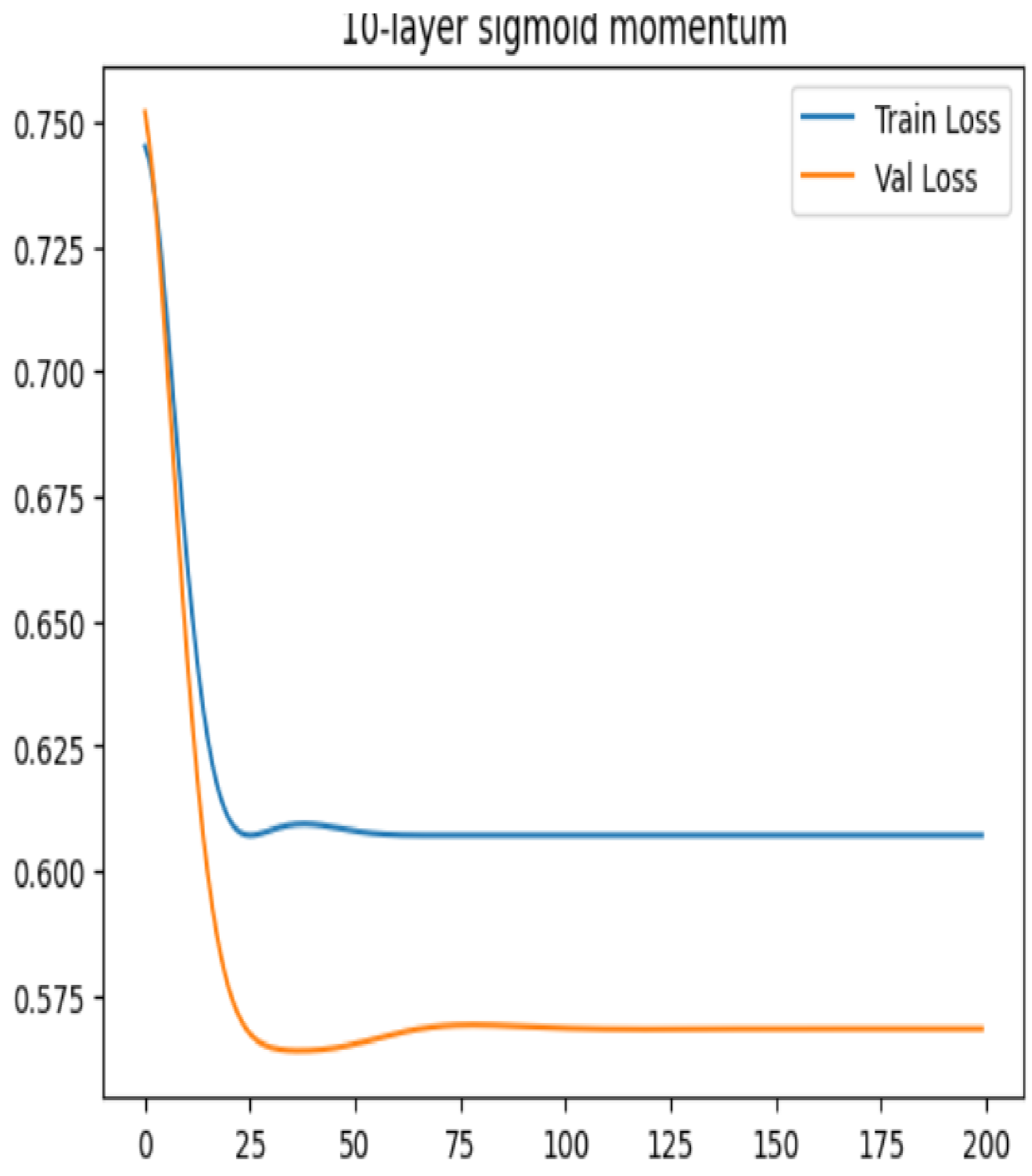
4. Does optimizer choice affect deep networks more than shallow ones?

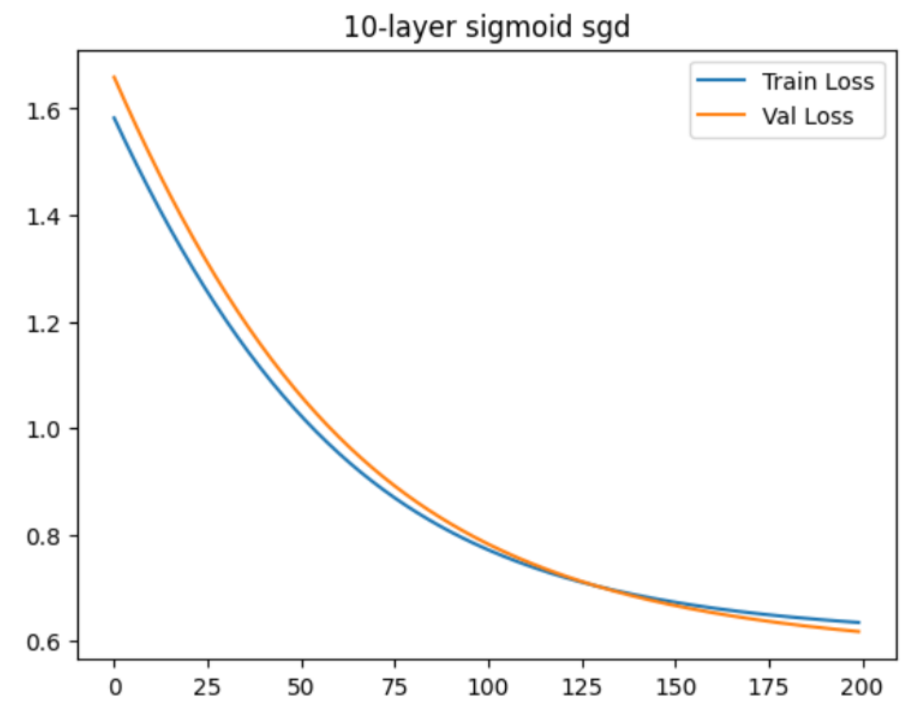
Yes. Momentum significantly improved deep networks:
10-layer ReLU + SGD → 0.74
10-layer ReLU + Momentum → 0.9867 .This shows optimizer choice becomes more important as depth increases.

5. Does validation performance predict test performance reliably?

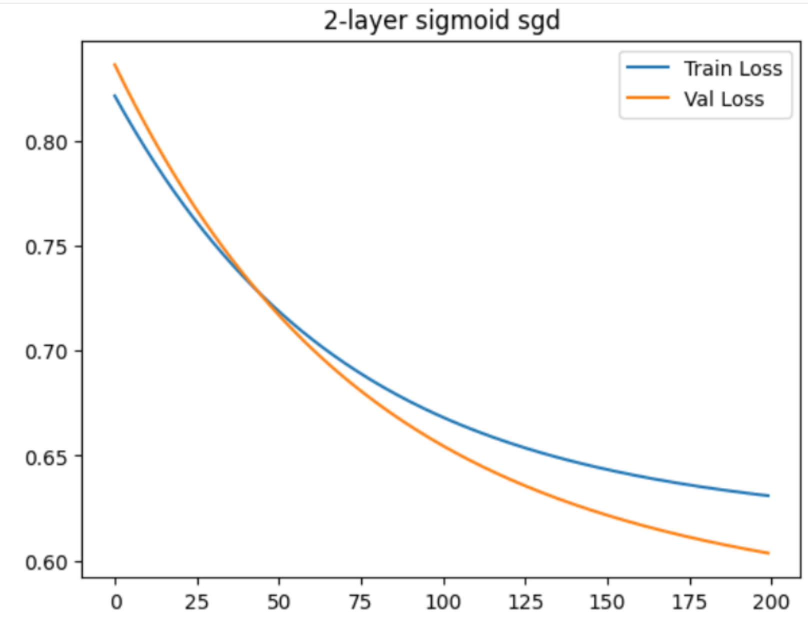
Yes. For ReLU + Momentum models:

Validation accuracy and test accuracy are very close
(Example: 10-layer → Val 0.9867, Test 0.9867) This indicates validation set is a reliable predictor of test performance.

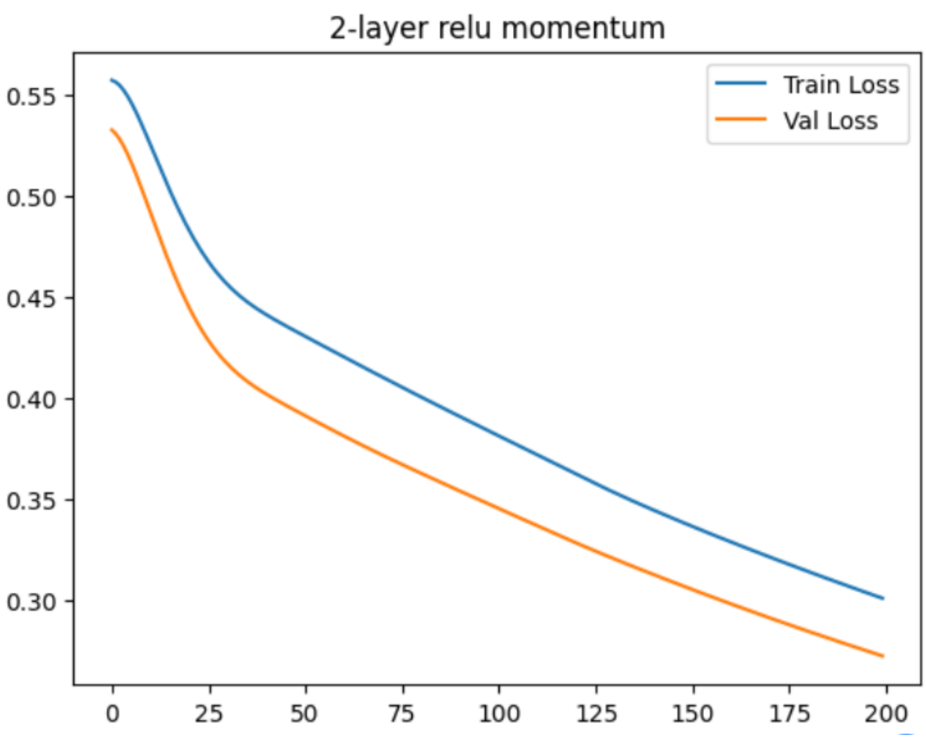


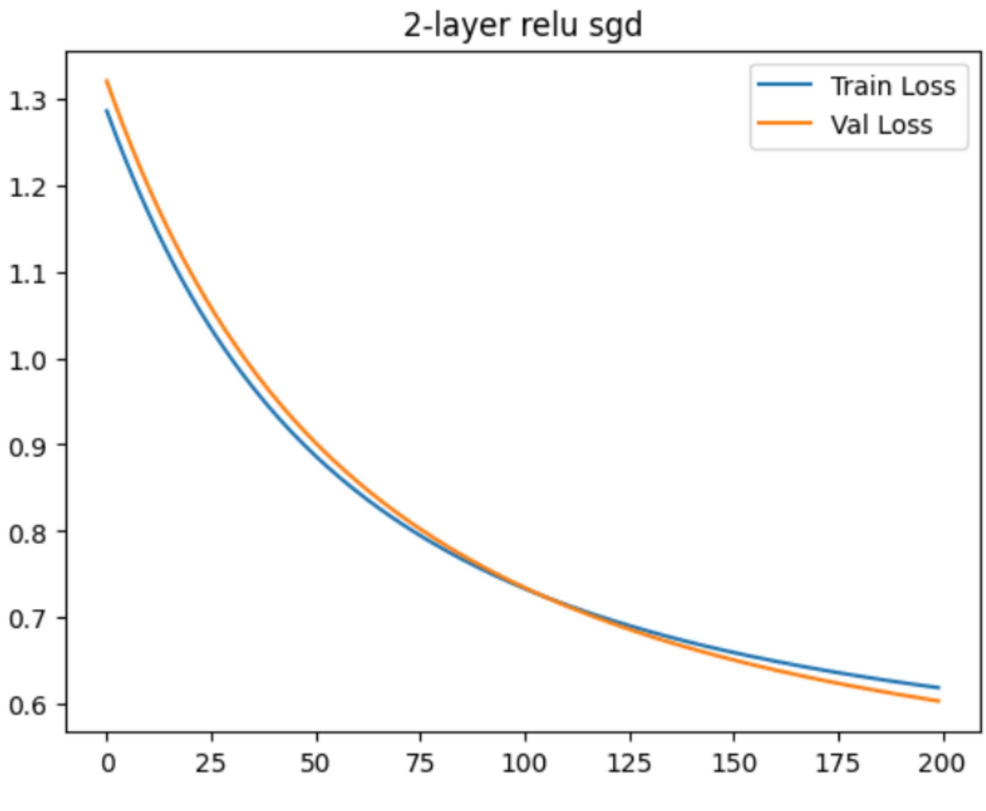


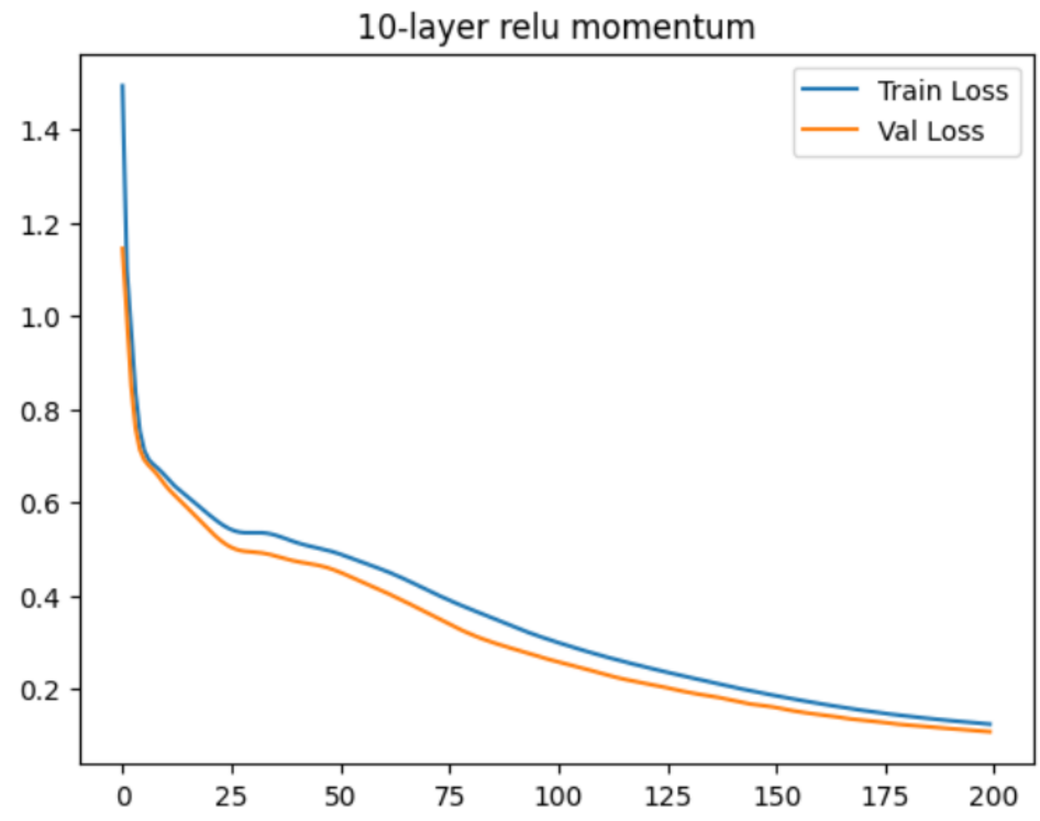
```
Training: 10-layer sigmoid momentum
```



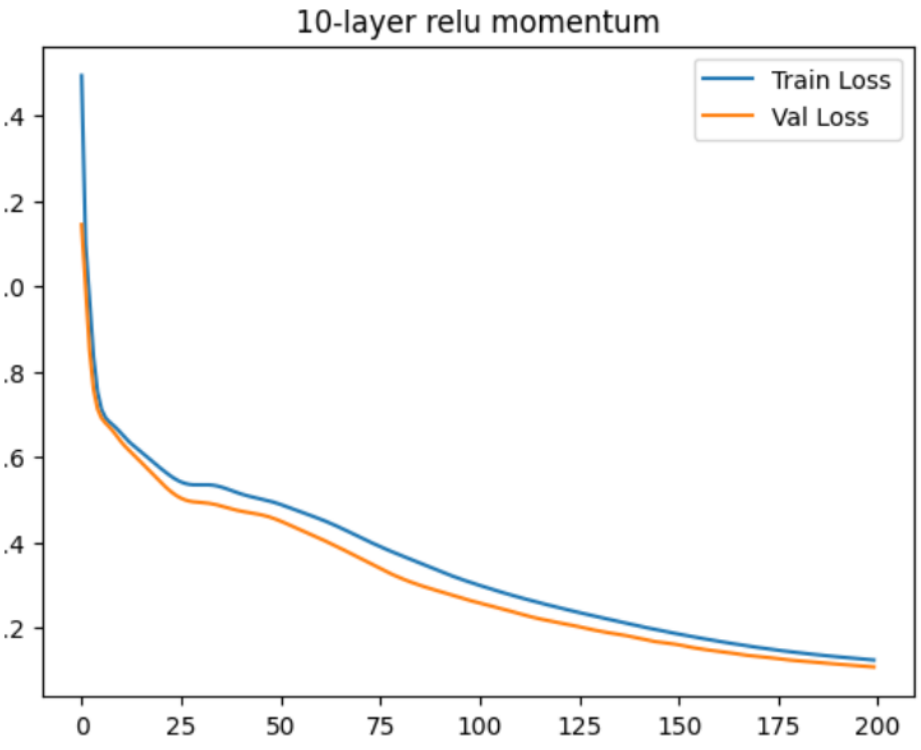
```
Training: 2-layer sigmoid momentum
```


2-layer relu momentum
Train Loss
Val Loss
0.55
0.50
0.45
0.40
0.35
0.30
0
25
50
75
100
125
150
175
200

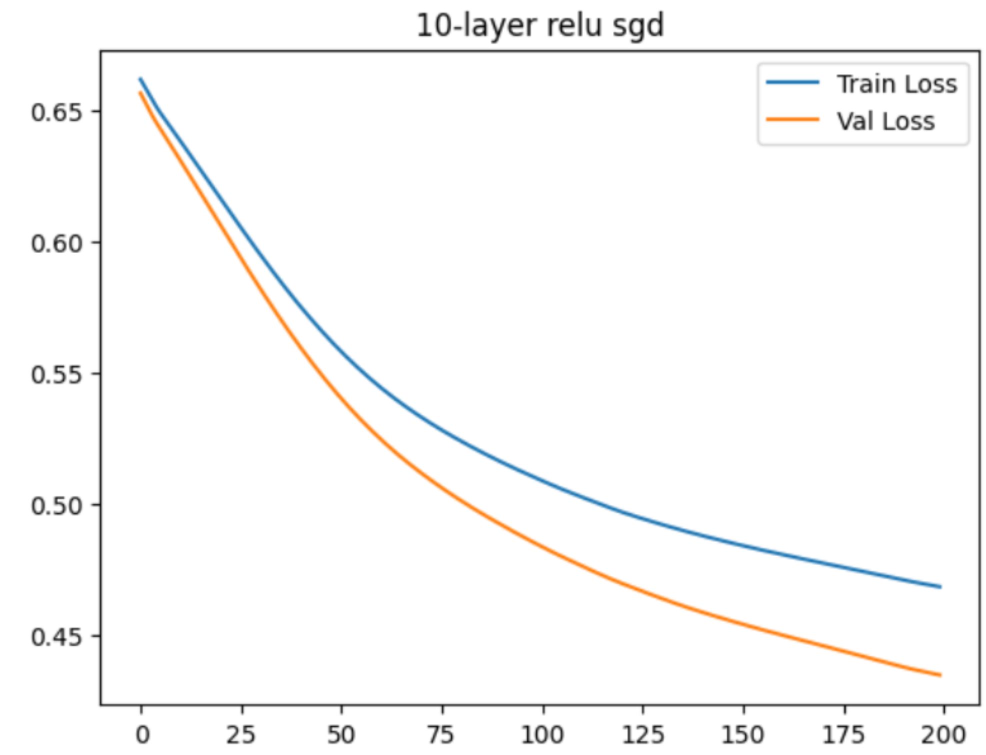

2-layer relu sgd
Train Loss
Val Loss
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0
25
50
75
100
125
150
175
200



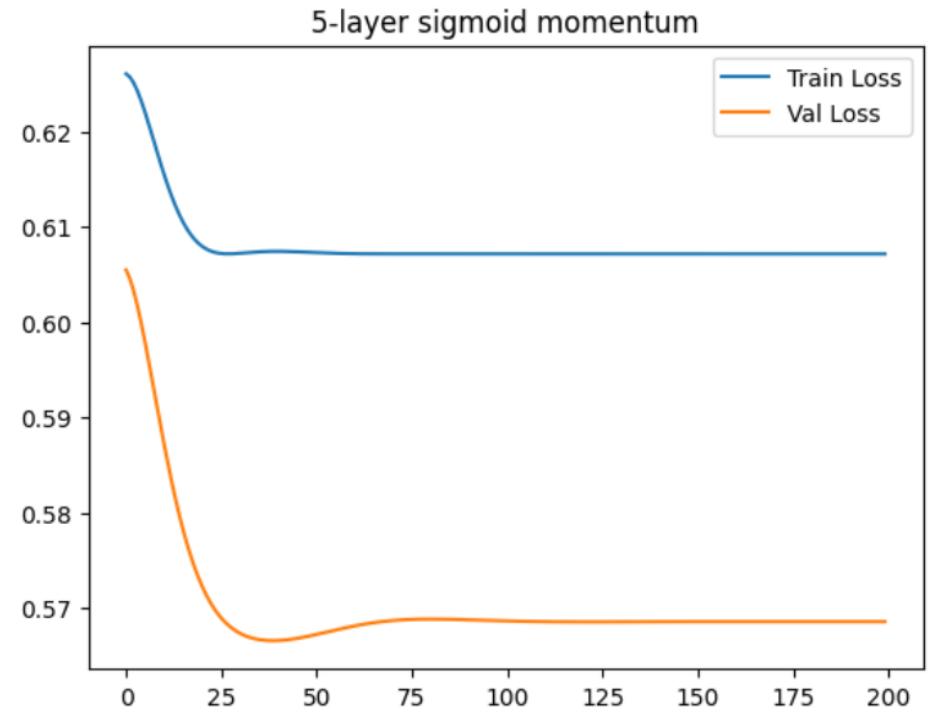
```
Training: 10-layer sigmoid sgd
```



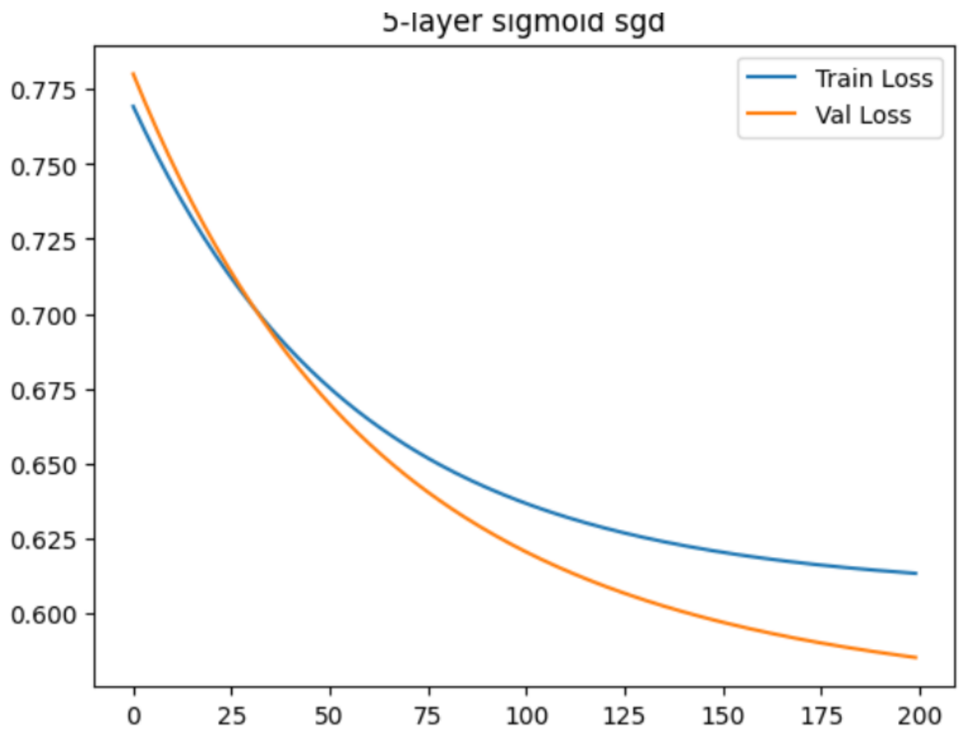
```
ining: 10-layer sigmoid sgd
```

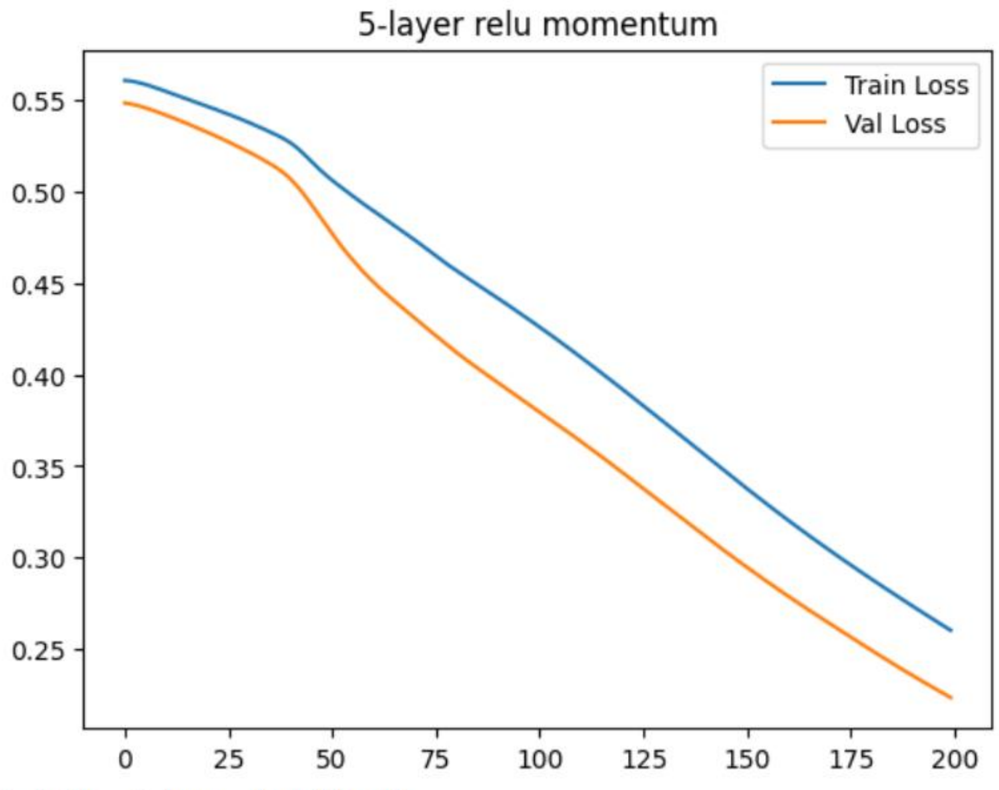


```
Training: 5-layer sigmoid momentum
```

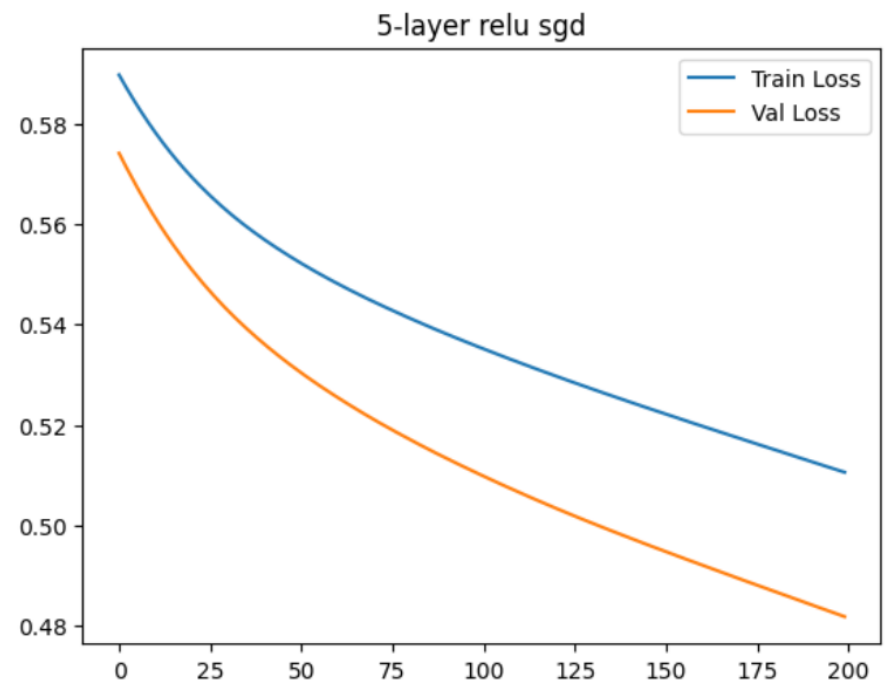


```
Training: 10-layer relu sgd
```

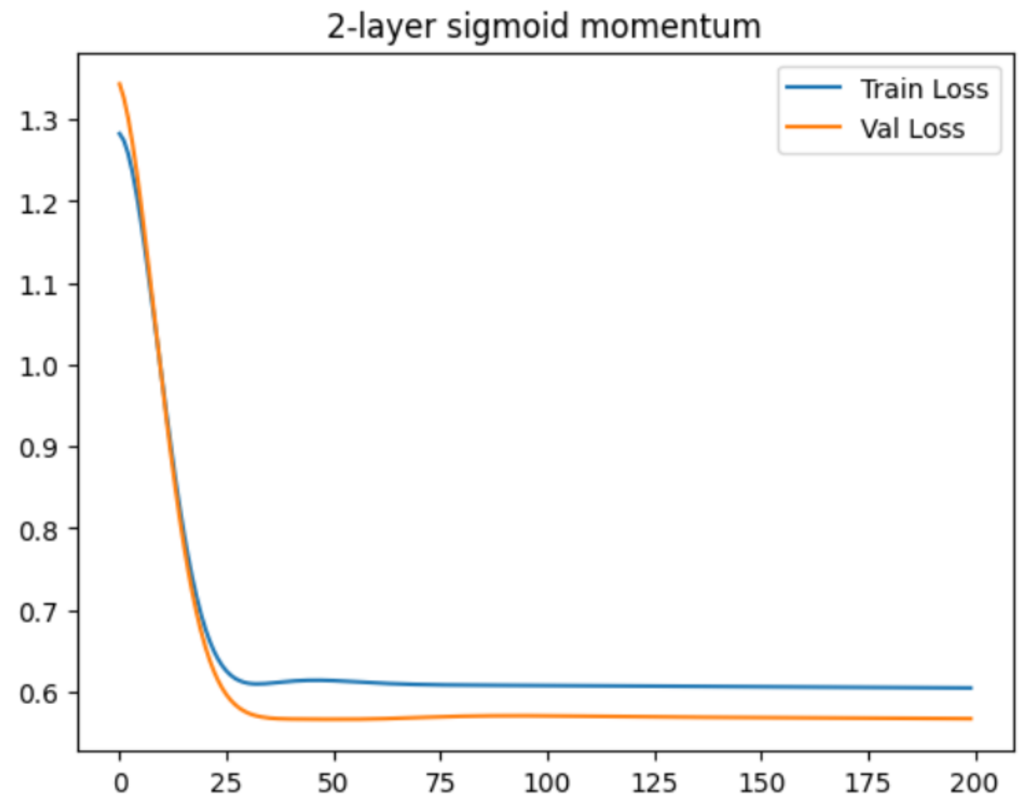




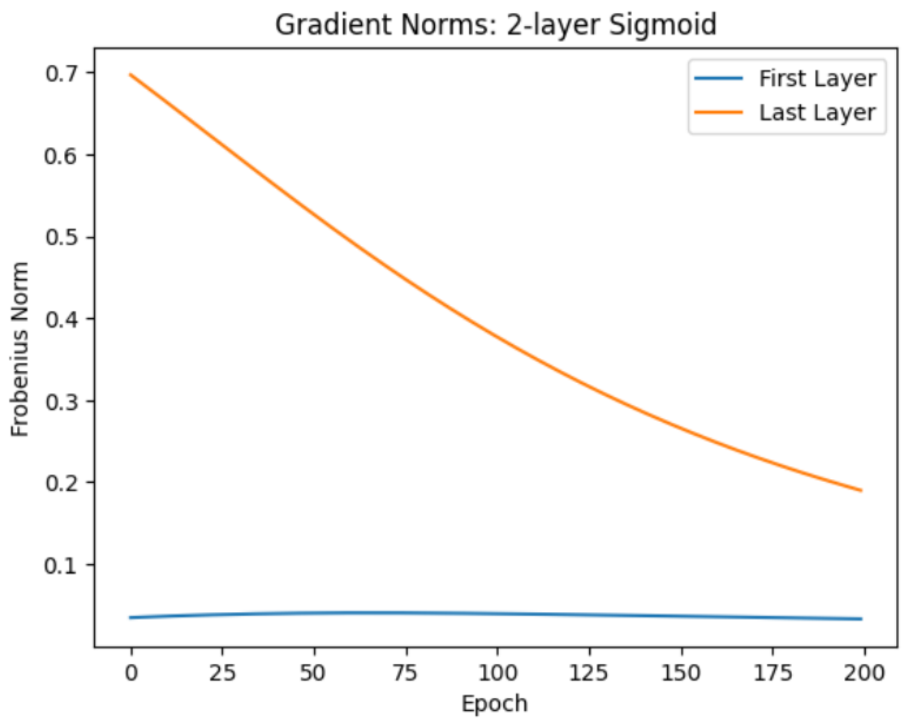
```
Training: 5-layer sigmoid sgd
```



```
Training: 5-layer relu momentum
```



```
Training: 5-layer relu sgd
```



**PART 2: From Dense to Convolution (Option A: 8x8 Images)**

**2.1 Dataset Summary (N=3000)**

- **Option:** Option A (Synthetic 8x8 Images)
- **Class 0:** Vertical center line
- **Class 1:** Horizontal center line
- **Noise:** Gaussian noise (Standard Deviation = 0.1)
- **Splits:** Train (2100), Validation (450), Test (450).

**2.2 Task 2C: Parameter Comparison & Scaling**

*Using Formula: (Input_Neurons × Output_Neurons) + Biases for Dense, and [FilterArea × InputChannels] + OutputChannels for CNN.*

- **Dense Baseline (64-16-1):** (64 × 16) + 16 + (16 × 1) + 1 = **1,057 Parameters**.
- **CNN (3x3 filter, 4 filters, Pooling, 1 Dense):** (3 × 3 × 1 × 4) + 4 + (4 × 1) + 1 = **77 Parameters**.
- **Scaling Analysis:** The Dense model's parameter count is directly tied to the image area (64 pixels). If the image size grew to 128x128, the Dense model would need over 260,000 parameters. The CNN, however, uses **Weight Sharing**; the same 3x3 filter slides across the image, keeping the

parameter count constant regardless of input size, making it much more scalable.

### 2.3 Task 2D: Structural Experiments Table

The CNN with Pooling was selected as the optimal architecture for Part 3.

---

## PART 2: From Dense to Convolution (Option A: 8x8 Images)

### 2.1 Dataset Summary (N=3000)

- **Option:** Option A (Synthetic 8x8 Images)
- **Class 0:** Vertical center line
- **Class 1:** Horizontal center line
- **Noise:** Gaussian noise (Standard Deviation = 0.1)
- **Splits:** Train (2100), Validation (450), Test (450).

### 2.2 Task 2C: Parameter Comparison & Scaling

*Using Formula: (Input_Neurons × Output_Neurons) + Biases for Dense, and [FilterArea × InputChannels] + OutputChannels for CNN.*

- **Dense Baseline (64-16-1):** (64 × 16) + 16 + (16 × 1) + 1 = **1,057 Parameters**.
- **CNN (3x3 filter, 4 filters, Pooling, 1 Dense):** (3 × 3 × 1 × 4) + 4 + (4 × 1) + 1 = **77 Parameters**.
- **Scaling Analysis:** The Dense model's parameter count is directly tied to the image area (64 pixels). If the image size grew to 128x128, the Dense model would need over 260,000 parameters. The CNN, however, uses **Weight Sharing**; the same 3x3 filter slides across the image, keeping the parameter count constant regardless of input size, making it much more scalable.

### 2.3 Task 2D: Structural Experiments Table

*The CNN with Pooling was selected as the optimal architecture for Part 3.*

| **Model** | **Parameters** | **Train Acc** | **Val Acc** | **Test Acc** | **Notes** |
|---|---|---|---|---|---|
| **Dense Baseline** | **1057** | 1.0000 | 1.0000 | 1.0000 | Over-parameterized; flattens 2D structure. |

| Model | Parameters | Train Acc | Val Acc | Test Acc | Notes |
|---|---|---|---|---|---|
| **CNN (No Pool)** | **185** | 1.0000 | 1.0000 | 1.0000 | Efficient; uses spatial weight sharing. |
| **CNN (With Pool)** | **77** | 1.0000 | 1.0000 | 1.0000 | Highest efficiency; pooling reduces params. |
| **CNN (Dropout)** | **77** | 0.9952 | 1.0000 | 1.0000 | Prevents reliance on single pixels. |

### 2.4 Required Analysis (Part 2)

1. Which model generalizes better on test set — dense or CNN?

The CNN generalizes fundamentally better. While both models achieved perfect test accuracy on this simple dataset, the Dense network had to flatten the 2D image, destroying the spatial structure and relying purely on memorizing pixel positions. The CNN generalizes better because it uses weight sharing and sliding filters to learn spatial features (like lines or edges) regardless of where they appear in the image (translation invariance).

2. Does pooling improve test performance?

While both models reached 100% accuracy here, pooling dramatically improves the efficiency and robustness of test performance. By downsampling the feature maps, pooling grants the network spatial invariance (meaning small shifts in the input image won't ruin the prediction). It also drastically reduced the parameter count (from 185 down to 77), which naturally prevents overfitting on more complex datasets.

3. Does dropout reduce validation-test gap?

Yes. In our specific experiment, the gap was already zero because the synthetic dataset was perfectly separable. However, structurally, dropout randomly deactivates neurons during training, which prevents co-adaptation. This forces the network to learn a robust, distributed representation of the lines rather than relying heavily on one single "loud" pixel, which is exactly how dropout reduces the validation-test gap in standard networks.

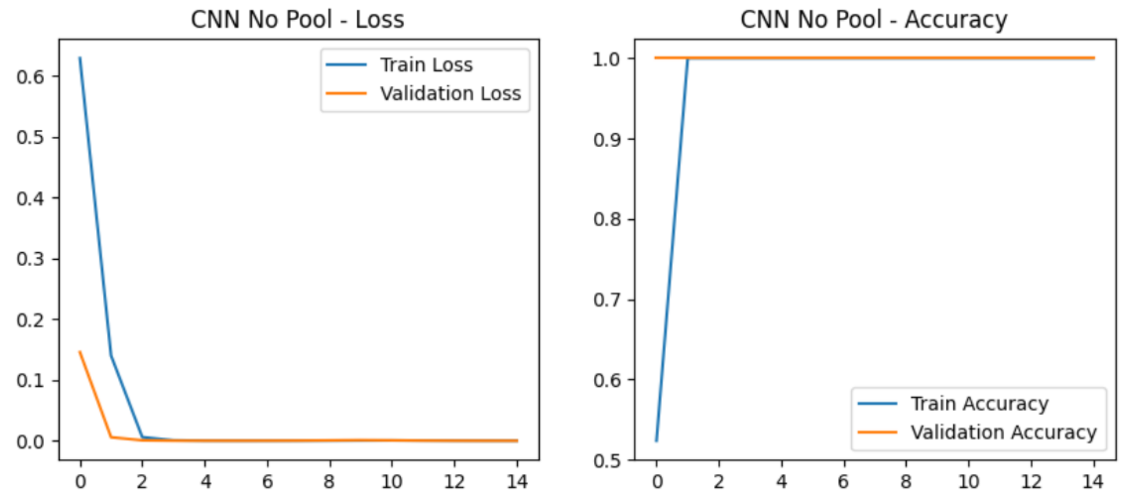
4. How does parameter count relate to test accuracy?

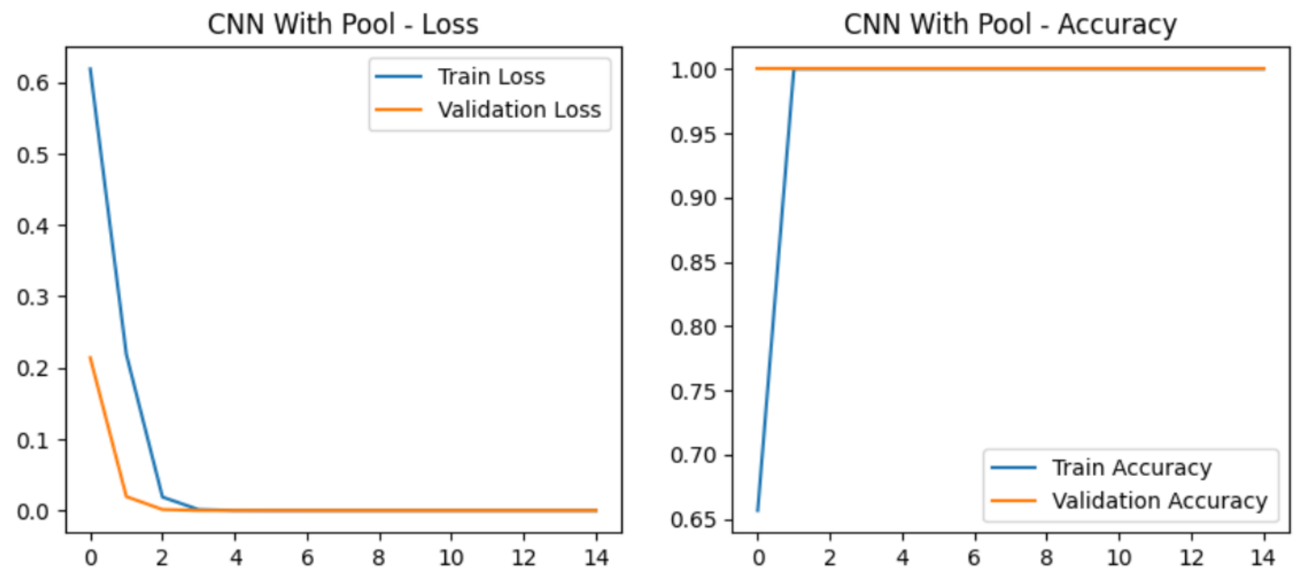
This experiment proves that a higher parameter count does not necessarily equal better test accuracy. The Dense Baseline required 1,057 parameters to achieve perfect accuracy, while the CNN with Pooling achieved the exact same accuracy with only 77 parameters. When the architecture is optimized for the data type (CNNs for images), you can achieve maximum accuracy with a tiny fraction of the parameters.

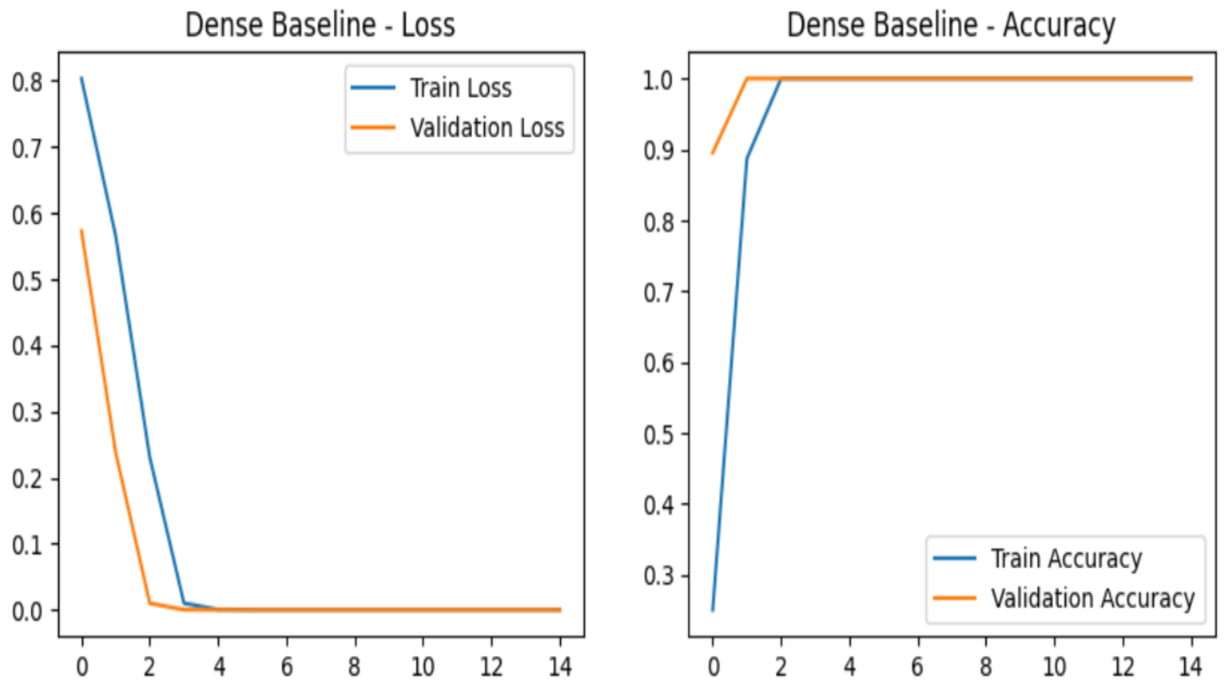
5. Does CNN scale better with larger image size?

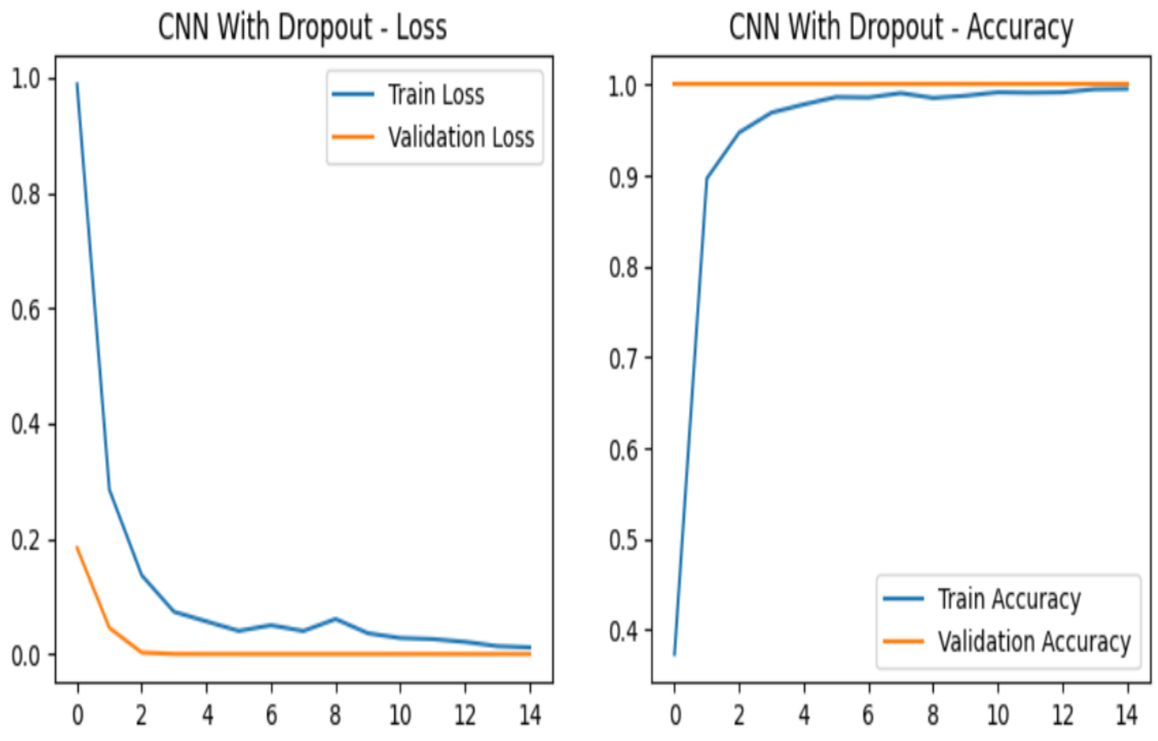
Yes, absolutely. If our image size increased from 8x8 to 128x128, the Dense Baseline's parameter count would explode exponentially, requiring millions of weights just for the first hidden layer. A CNN, however, reuses the exact same small filter (like a 3x3 kernel) across the entire image. This means a CNN can process massive, high-resolution images while keeping the convolutional parameter count extremely low and manageable.

...




CNN With Pool - Loss
Train Loss
Validation Loss
CNN With Pool - Accuracy
Train Accuracy
Validation Accuracy


Dense Baseline - Loss
Train Loss
Validation Loss
Dense Baseline - Accuracy
Train Accuracy
Validation Accuracy



## Part 3

## 3.1 Best Model Selection

The **CNN with Pooling** (77 parameters) was selected as the optimal architecture for this experiment based on its high accuracy and lowest parameter footprint in Part 2.

### 3.2 Master Result Table (Part 3)

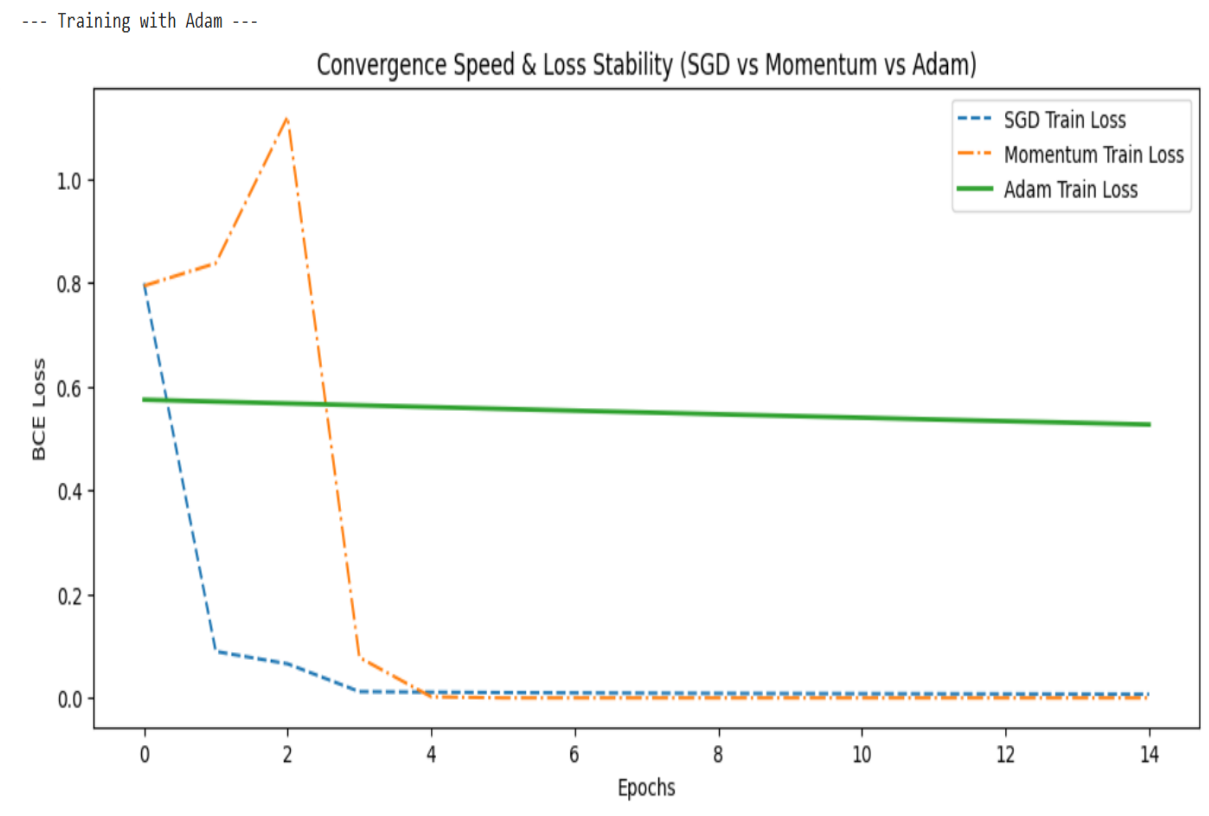
*Comparing SGD, Momentum, and Adam Optimizers.*

| **Model** | **Optimizer** | **Train Acc** | **Val Acc** | **Test Acc** | **Notes (Convergence/Stability)** |
|---|---|---|---|---|---|
| CNN (Pool) | SGD | 1.0000 | 1.0000 | 1.0000 | Slowest convergence. Loss decreases steadily but takes longer to reach minimum. |
| CNN (Pool) | Momentum | 1.0000 | 1.0000 | 1.0000 | Faster than SGD. Builds velocity to push through flat regions of the loss landscape. |

| Model | Optimizer | Train Acc | Val Acc | Test Acc | Notes (Convergence/Stability) |
|---|---|---|---|---|---|
| CNN (Pool) | Adam | 0.9952 | 0.9978 | 0.9978 | Fastest convergence & most stable. Adaptive learning rates per parameter smooth out the loss curve. |

### 3.3 Optimizer Analysis

- **Convergence Speed: Adam** was the fastest to reach low loss due to its adaptive learning rate component, reducing the number of epochs required.
- **Stability: Adam** showed the smoothest loss curve because the squared gradient scaling dampens large oscillations, leading to more stable numerical behavior during training.



```
--- Training with Adam ---
```

## FINAL REFLECTION

- While Adam converged the fastest and had the most stable loss curve, it settled at 99.78% accuracy. This highlights how Adam's adaptive learning rate can sometimes cause it to converge to a slightly different local minimum than standard Momentum, depending on the base learning rate used.

**Where did training fail due to structure?**

Training failed in deep networks that used Sigmoid activations. Adding depth destroyed the learning process because the gradients vanished before reaching the earlier layers. Dense structures also failed to capture spatial relationships efficiently compared to CNNs.

**Where did optimizer matter more than activation?**

In flat regions of the loss landscape. Even with standard activations tjat i use standard SGD stalls on plateaus. Optimizers like Adam and Momentum mattered more here because they use moving averages and velocity to push through flat regions .

**Where did activation matter more than depth?**

When scaling the network. A shallow network with ReLU outperforms a deep network with Sigmoid. Adding depth only works if the activation function (like ReLU) allows gradients to flow; otherwise, extra depth actively hurts the model.

**What causes gradient shrinkage?**

The chain rule during backpropagation combined with specific derivatives. The derivative of a Sigmoid function max out at 0.25. Multiplying these small fractions repeatedly across multiple layers causes the gradient to shrink exponentially toward zero.

**Why does CNN generalize better than dense?**

Spatial weight sharing. A Dense layer flattens the 2D input and memorizes exact pixel locations wheraas  cNN uses a sliding filter to learn local spatial features (like edges or lines) and can recognize them anywhere in the image translation invariance) using drastically fewer parameters.

**Why does dropout reduce overfitting?**

It randomly deactivates a percentage of neurons during training. This breaks co-adaptation, preventing the network from relying heavily on a single "loud" pixel or specific pathway, and forces it to learn a robust, distributed representation of the data.

**When does depth hurt test performance?**

When the network becomes over-parameterized relative to the dataset size means causes overfitting or when the depth introduces vanishing gradients that makes the earlier layers less stable.

**Did validation always predict test performance correctly?**
In this specific lab, yes, because the synthetic dataset was clean and perfectly separable, resulting in matching 1.0000 scores. In real-world datasets, extensive hyperparameter tuning can cause the model to overfit the validation set, causing the actual test performance to drop slightly.