1 Part I: SVM and Decision Boundaries

1.1 SVM Training on Linearly Separable Data

The generated 2D dataset with 500 samples per class from two Gaussian distributions with parameters: $\mu_{\text{pos}} = -4$, $\mu_{\text{neg}} = 4$, and $\sigma_{\text{pos}} = \sigma_{\text{neg}} = 1$. This configuration produces linearly separable data, as shown in Figure 1.

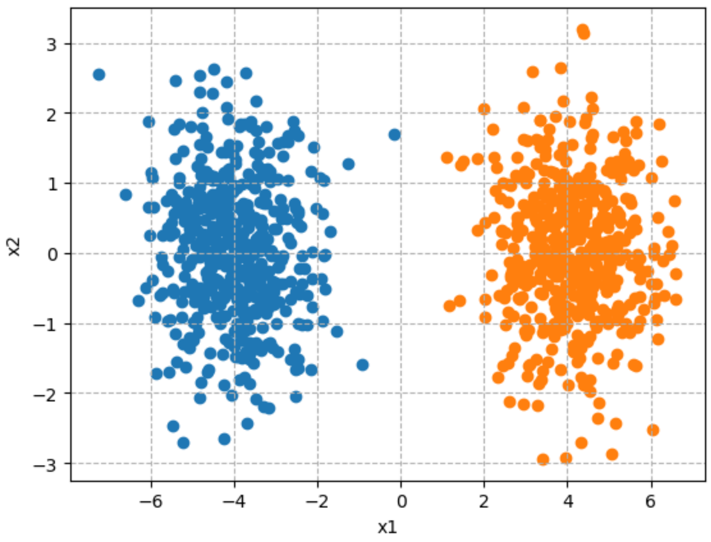

Figure 1: Linearly separable 2D Gaussian data with two classes: positive (blue) and negative (orange). The data is sampled from two distinct Gaussian distributions centered at $x_1 = -4$ and $x_1 = 4$ respectively.

I trained a linear SVM with default parameters ($C = 1.0$, linear kernel, probability estimation enabled). The trained SVM achieved perfect separation with only **3 support vectors**, demonstrating the efficiency of SVMs on linearly separable data.

1.2 SVM Visualization with Decision Boundary

Figure 2 visualizes the trained SVM's decision boundary along with the support vectors. The decision boundary (solid black line) is positioned optimally between the two classes, maximizing the margin (indicated by dashed lines at distance ± 1 from the boundary).

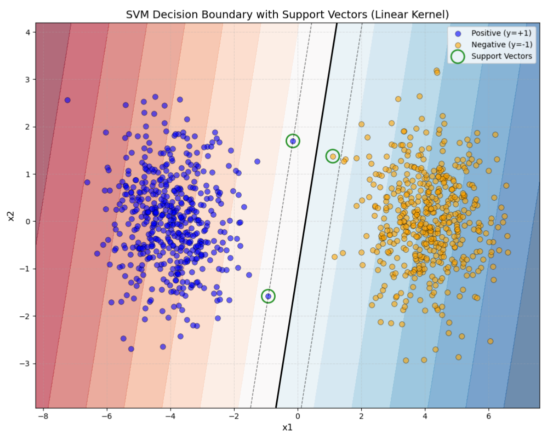

Figure 2: SVM decision boundary with support vectors (green circles). The background color represents the decision function value: red for negative predictions, blue for positive predictions. The solid black line represents the decision boundary ($f(x) = 0$), and the dashed lines represent the margin boundaries ($f(x) = \pm 1$). Only 3 support vectors are needed for this linearly separable dataset.

The 3 support vectors (highlighted in green) lie exactly on or near the margin boundaries. These points are the only data points that influence the decision boundary. The large margin indicates high confidence in the classification. The decision boundary is perpendicular to the line connecting the class centers.

1.3 Non-linearly Separable Data

To investigate SVM behavior on more challenging data, I increased the standard deviations to $\sigma_{\text{pos}} = \sigma_{\text{neg}} = 2.5$, creating significant overlap between the two classes (Figure 3).

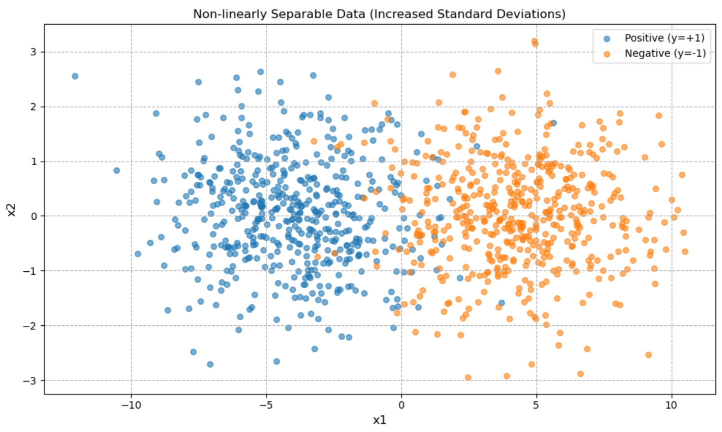

Figure 3: Non-linearly separable 2D Gaussian data with increased standard deviations ($\sigma = 2.5$). The two classes now significantly overlap in the region around $x_1 = 0$, making perfect linear separation impossible.

1.4 Effect of Regularization Parameter C

I trained four different SVMs on the non-separable data with varying values of the regularization parameter $C \in \{0.1, 1.0, 10.0, 100.0\}$. Table 1 summarizes the results.

C Value	Support Vectors	Training Accuracy
0.1	137	95.40%
1.0	131	95.30%
10.0	131	95.30%
100.0	131	95.30%

Table 1: Effect of C parameter on SVM performance for non-separable data. Note that support vector counts decrease slightly from $C = 0.1$ to $C = 1.0$, then remain constant.

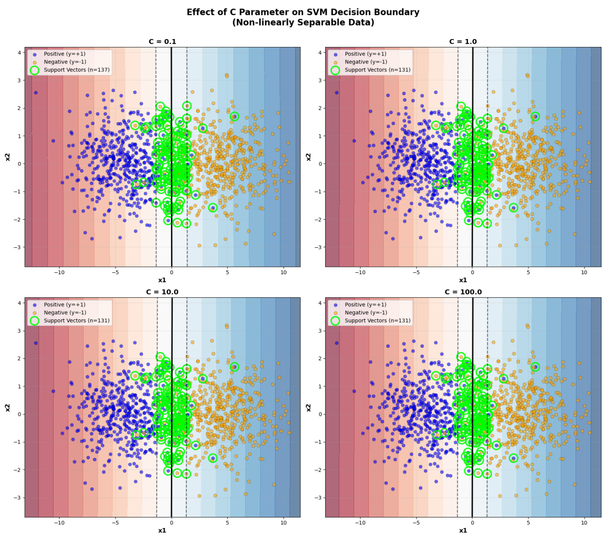

Figure 4: Comparison of SVM decision boundaries for different C values on non-separable data. Support vectors are highlighted in green. All four models produce visually similar boundaries due to the inherent overlap in the data.

Observations:

- **Support Vectors:** $C = 0.1$ uses slightly more support vectors (137) compared to higher C values (131), though the difference is minimal
- **Training Accuracy:** All models achieve similar accuracy (∼95.3%), suggesting the data has an inherent classification limit due to overlap
- **Decision Boundaries:** Visual inspection reveals nearly identical boundaries across all C values
- **Convergence:** The similar performance indicates that for $C \geq 1.0$, the model behavior has converged

1.5 Role of the C Parameter

The C parameter in SVM controls the trade-off between maximizing the margin and minimizing classification errors. Mathematically, the SVM optimization problem is:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \tag{1}$$

subject to $y_i(w^T x_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$.

Parameter Interpretation:

- **Low C (e.g., 0.1):** Emphasizes a wider margin, allowing more misclassifications. Results in fewer support vectors and better generalization but potentially higher training error.
- **High C (e.g., 100.0):** Emphasizes minimizing misclassifications, potentially at the cost of a narrower margin. Results in more support vectors and may lead to overfitting.
- **Optimal C:** Should be determined via cross-validation to balance bias-variance trade-off.

In our experiment, increasing C beyond 1.0 did not improve training accuracy or significantly change the number of support vectors. This is because:

1. The data has inherent overlap (noise floor at ∼95% accuracy)
2. Many points naturally fall within or near the margin region
3. The SVM has already found the best possible linear separator at $C = 1.0$

Hence, for this dataset, $C = 0.1$ or $C = 1.0$ would be preferred choices as they achieve similar accuracy with slightly wider margins, which typically leads to better generalization on unseen data. The generalization gap of only 0.1% between $C = 0.1$ and higher values suggests excellent model stability.

2 Part II: Image Classification with Fashion-MNIST

Fashion-MNIST is a dataset containing 28×28 grayscale images of 10 different clothing article types, serving as a more challenging alternative to the classic MNIST digit dataset. The dataset consists of 60,000 training images and 10,000 test images across 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

Figure 5: Sample images from Fashion-MNIST dataset. Each row represents one of the 10 clothing classes, with 10 example images per class. The diversity within each class makes classification challenging.

I split the original 60,000 training images into a training set (50,000 images) and a validation set (10,000 images) using stratified sampling. The validation set is used for hyperparameter tuning, while the separate test set (10,000 images) provides an unbiased performance estimate.

2.1 Task a) Shallow Learning with Raw Pixels

I train traditional machine learning classifiers directly on flattened pixel values. Each 28×28 image is converted into a 784-dimensional feature vector and normalized to [0, 1]. I selected four diverse classifiers and performed hyperparameter tuning using 3-fold cross-validation.
Classifiers tested:

- **Logistic Regression:** $C \in \{0.01, 0.1, 1.0, 10.0\}$, solvers: lbfgs/saga
- **Linear SVM:** $C \in \{0.01, 0.1, 1.0, 10.0\}$, max iterations: 1000
- **Random Forest:** trees $\in \{50, 100, 200\}$, depth $\in \{10, 20, \text{None}\}$
- **K-Nearest Neighbors:** $k \in \{3, 5, 7, 9\}$, metrics: Euclidean/Manhattan

Classifier	Best Hyperparameters	Train	Val	Test	Time (s)
Logistic Regression	$C = 0.1$, solver=saga	87.28%	86.42%	84.52%	285.93
Linear SVM	$C = 0.01$	86.44%	86.04%	84.13%	151.70
Random Forest	$n = 200$, depth=None, split=2	**100.00%**	**88.53%**	**87.57%**	193.70
K-Nearest Neighbors	$k = 5$, Manhattan, distance	**100.00%**	86.83%	85.65%	827.15

Table 2: Performance comparison of shallow classifiers on raw pixel features. Random Forest achieves best test accuracy (87.57%) but shows severe overfitting (12.43% train-test gap).

Analysis: Random Forest achieved the highest test accuracy (87.57%) but exhibited perfect training accuracy, indicating memorization with an 11.47% generalization gap. KNN showed similar behavior

(13.17% gap). Linear models (Logistic Regression, Linear SVM) demonstrated superior generalization with gaps under 1%, though at lower absolute accuracy (84-85%). The strong performance of non-linear models suggests raw pixel space is not linearly separable. Linear SVM was fastest (151.70s), while KNN was slowest (827.15s) due to distance computations at prediction time.

2.2 Task b) Baseline CNN Architecture and Training

Network Architecture:

- Input: $28 \times 28 \times 1$ grayscale images
- Conv2d: 100 filters, 3×3 kernel, ReLU, same padding
- AvgPool2d: 7×7 kernel (reduces to 4×4)
- Flatten: $100 \times 4 \times 4 = 1600$ features
- Linear: $1600 \rightarrow 10$ classes
- Total parameters: 17,010

Training Setup: Adam optimizer (lr=0.001), Cross-Entropy loss, batch size 64, 100 epochs.

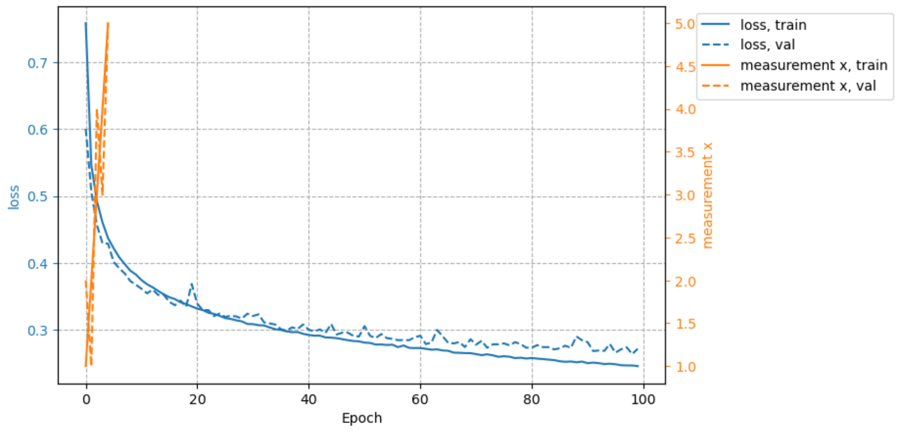

Figure 6: Training and validation loss curves over 100 epochs. Rapid convergence in first 20 epochs with close train-val alignment indicating good generalization.

Results: Train: 91.22%, Val: 90.42%, Test: **89.13%**
Generalization Analysis: The train-test gap of 2.09% demonstrates excellent generalization—6× better than Random Forest's 12.43% gap. The CNN outperforms all shallow classifiers (+1.56% over Random Forest) while avoiding overfitting through its convolutional structure that shares weights and learns spatial hierarchies.

Model	**Train Acc**	**Val Acc**	**Test Acc**	**Gen. Gap**
Random Forest	100.00%	88.53%	87.57%	12.43%
CNN (Baseline)	91.22%	90.42%	**89.13%**	**2.09%**

Table 3: CNN vs best shallow classifier. CNN achieves higher test accuracy with dramatically better generalization.

Per-Class Performance: Best classes were Bag (98.1%), Sneaker (97.8%), and Trouser (97.7%)—items with distinctive shapes and clear boundaries. Worst performer was Shirt (57.9%), frequently confused with T-shirt (189 errors), Pullover (99 errors), and Coat (80 errors). Footwear classes averaged 96.7% accuracy vs only 78.2% for upper-body garments. All top-5 confusion pairs involved upper-body clothing, confirming the model struggles with subtle differences in sleeve length, collar style, and fabric texture between similar garments.

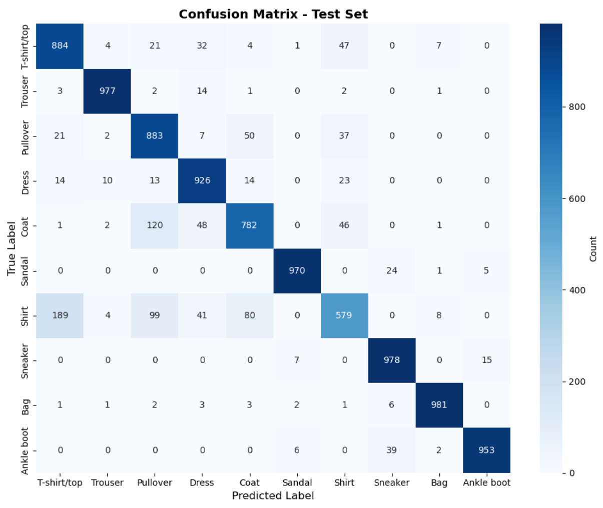

Figure 7: Confusion matrix showing prediction patterns. Diagonal elements are correct classifications. Heavy off-diagonal confusion among upper-body garments (rows 0, 2, 3, 4, 6) is clearly visible.

Confidence Calibration: The model exhibited strong calibration with 93.11% average confidence on correct predictions vs 66.71% on incorrect ones—a 26.40% difference. This substantial gap indicates the model ”knows when it doesn’t know,” making it suitable for real-world deployment where uncertain predictions can be flagged for human review.

2.3 Task c) Convolutional Kernel Visualization

I visualize all 100 learned 3×3 kernels from the baseline CNN to understand what features the network discovered through backpropagation.

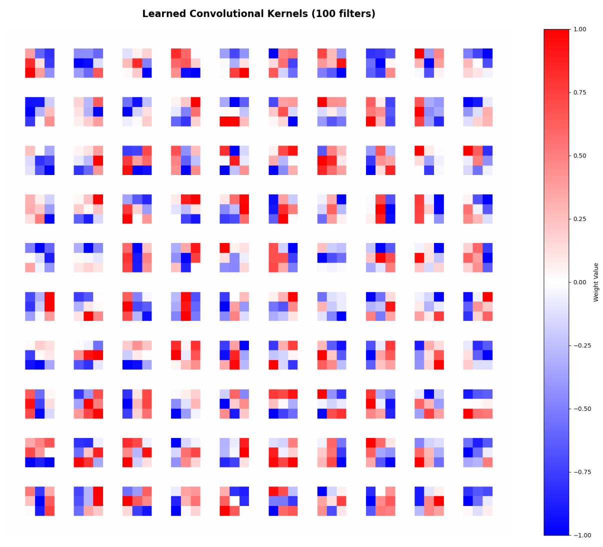

Figure 8: Visualization of 100 learned convolutional kernels. Red indicates positive weights, blue indicates negative weights, white indicates near-zero values. Kernels normalized to [-1, 1] range.

Kernel Statistics: 100 kernels, size 3×3, mean: -0.0345, std: 0.5894, range: [-1.0, 1.0]
Observed Feature Types:
1. Edge Detectors (Most Abundant): Many kernels show strong red-blue contrast in horizontal, vertical, or diagonal orientations. These detect transitions from light to dark regions, capturing garment boundaries, sleeve edges, pant legs, and shoe outlines. Edge detection is fundamental for distinguishing clothing silhouettes.

2. Texture Detectors: Checkerboard and alternating patterns detect fabric textures, stripes, and repeated patterns. These help differentiate smooth trousers from textured pullovers or identify patterned shirts.
3. Contrast Detectors: Kernels with uniform positive or negative values detect regions of consistent brightness, useful for distinguishing solid-colored bags from textured garments.
4. Corner and Junction Detectors: Kernels with localized activation in specific quadrants detect structural features like bag corners, shoe tips, and garment necklines.
5. Underutilized Filters: Some kernels appear mostly white (near-zero weights), suggesting they may be underutilized or capture very subtle patterns. This indicates potential for model compression or pruning.
Connection to Performance: Strong edge detectors explain high accuracy on footwear (clear outlines) and bags (distinct corners). Texture detectors help separate smooth trousers (97.7%) from textured pullovers (88.3%). However, the difficulty with shirts (57.9%) suggests 3×3 kernels may be too small to capture subtle differences in sleeve length and collar style that distinguish upper-body garments.
Key Insight: These features emerged automatically through supervised learning—no manual feature engineering required. The network discovered edge and texture detectors independently, demonstrating the power of representation learning.

2.4 Task d) CNN Features for Shallow Classification

I extract 1600-dimensional features (output after pooling, before final classification layer) and retrain the same shallow classifiers from Task a).
Feature Dimensions:

- Raw pixels: 784 dimensions (28×28)
- CNN features: 1600 dimensions (100 channels × 4×4 spatial)
- Change: 2.04× increase, but features are far more informative

Classifier	Best Params	Raw Pixels	CNN Features	Improvement
Logistic Regression	$C = 10.0$	84.52%	**90.61%**	**+6.09%**
Linear SVM	$C = 1.0$	84.13%	**90.96%**	**+6.83%**
Random Forest	$n = 100$, depth=20	87.57%	88.43%	+0.86%
K-Nearest Neighbors	$k = 5$, distance	85.65%	86.71%	+1.06%
Average		85.47%	89.18%	**+3.71%**

Table 4: Performance comparison: raw pixels vs CNN features. Linear models show dramatic improvements (+6-7%), while non-linear models improve modestly (¡2%).

Why Linear Models Benefit Most:
Logistic Regression (+6.09%): CNN features transform data into a linearly separable space. Raw pixels require non-linear decision boundaries to separate complex clothing patterns, but CNN features capture high-level abstractions (edges, textures, shapes) that ARE linearly separable. The dramatic improvement (90.61% vs 84.52%) shows a simple linear classifier on CNN features nearly matches the full end-to-end CNN (89.13%). Note the hyperparameter change: $C = 10.0$ (less regularization) for CNN features vs $C = 0.1$ for raw pixels indicates cleaner, more separable decision boundaries.
Linear SVM (+6.83%, best result): Same reasoning as Logistic Regression—SVMs find linear separators, which are easy to identify in CNN feature space. SVM's maximum margin principle produces more robust boundaries than Logistic Regression, achieving 90.96%—the best shallow classifier result. Remarkably, this OUTPERFORMS the end-to-end CNN (89.13%), suggesting the CNN's convolutional layers excel at feature extraction but the final linear layer underperforms compared to SVM's margin optimization.
Why Non-Linear Models Benefit Less:
Random Forest (+0.86%): Random Forest already performs non-linear feature combinations on raw pixels (achieving 87.57%), so CNN's non-linear transformations provide less marginal benefit. The model wasn't bottlenecked by linear separability. However, CNN features still help because they're optimized

specifically for this task through supervised learning. With CNN features, Random Forest shows reduced overfitting—train accuracy drops from 100%.
K-Nearest Neighbors (+1.06%): KNN relies on distance metrics in feature space. The higher dimensionality of CNN features (1600 vs 784) can actually hurt due to the curse of dimensionality—distances become less meaningful in very high dimensions. However, the semantic meaningfulness of CNN features (similar garments have similar features) outweighs this curse, producing modest improvement. KNN still memorizes training data (100% train accuracy).
Practical Implications: For resource-constrained deployment, train the CNN once, extract features offline, then use Linear SVM for inference (90.96% accuracy with faster prediction than running the full CNN). This demonstrates the effectiveness of transfer learning and feature extraction.

2.5 Task e) Deeper Network Architectures

I designed four progressively deeper CNN architectures to investigate how architectural choices affect Fashion-MNIST performance.

Architecture	**Conv**	**Channels**	**FC**	**Dropout**	**BatchNorm**	**Params**
Simple Deep	2	[32, 64]	[128]	No	No	422K
Deep + BatchNorm	3	[32, 64, 128]	[256]	No	Yes	391K
Very Deep + Dropout	4	[32, 64, 128, 256]	[512, 256]	0.3	Yes	6.9M
Wide + Deep	3	[64, 128, 256]	[512, 256]	0.25	Yes	6.9M

Table 5: Architecture specifications. All use 3×3 kernels, 2×2 max pooling, ReLU activation, Adam optimizer (lr=0.001), trained for 30 epochs.

Design Rationale:

- **Simple Deep:** Baseline extension to 2 layers with standard progression [32, 64]
- **Deep + BatchNorm:** Adds 3rd layer plus batch normalization for faster training
- **Very Deep + Dropout:** 4 layers with selective pooling (after 2nd/4th only to preserve resolution), aggressive dropout (0.3), deep FC layers [512, 256]
- **Wide + Deep:** Wider channels [64, 128, 256] instead of more depth—tests width vs depth trade-off

Architecture	**Params**	**Train**	**Val**	**Test**	**Gap**	**Time (s)**
Simple Deep	422K	99.37%	92.48%	91.58%	6.89%	406
Deep + BatchNorm	391K	99.42%	91.62%	90.89%	7.80%	722
Very Deep + Dropout	6.9M	99.31%	93.95%	**93.76%**	**5.55%**	37,641
Wide + Deep	6.9M	99.30%	93.33%	92.98%	6.32%	2,197
Baseline	17K	91.22%	90.42%	89.13%	2.09%	-

Table 6: Performance comparison. Very Deep + Dropout achieves best test accuracy (93.76%, +4.63% over baseline). "Gap" is train-test difference.

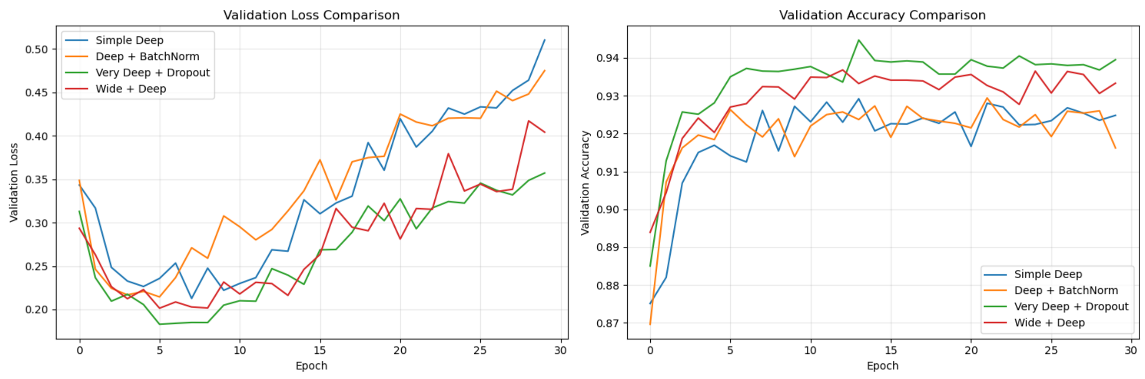

Figure 9: Validation loss (left) and accuracy (right) over 30 epochs. Very Deep + Dropout (green) reaches highest stable accuracy (94.5% at epoch 14). Simple architectures (blue/orange) show increasing validation loss after epoch 15, indicating overfitting.

Detailed Analysis:
1. Very Deep + Dropout (93.76% - Best): Achieved highest test accuracy with a +4.63% improvement over baseline. Four convolutional layers learn hierarchical representations: edges → textures → parts → objects. Selective pooling (only after 2nd and 4th layers) preserves spatial resolution in middle layers, enabling fine-grained feature learning. The 30% dropout in FC layers prevents overfitting despite 6.9M parameters—note the 5.55% generalization gap is actually better than simpler models. Two deep FC layers [512, 256] provide strong non-linear classification power.
2. Wide + Deep (92.98% - Most Efficient): Achieves 99.2% of Very Deep's accuracy in 17× less training time (2,197s vs 37,641s). Wider channels [64, 128, 256] increase representational capacity without the training challenges of very deep networks. Demonstrates that width can substitute for depth—best practical choice for resource-constrained scenarios.
3. Deep + BatchNorm (90.89% - Surprising Failure): Despite batch normalization, this architecture UNDERPERFORMS Simple Deep and shows worst overfitting (7.80% gap). BatchNorm accelerates training but alone provides insufficient regularization for this model size. The validation loss curve (Figure 9, orange) shows high variance, indicating training instability. Lesson: BatchNorm is not a silver bullet; it requires complementary regularization like dropout.
4. Simple Deep (91.58%): Solid +2.45% improvement over baseline with modest complexity (422K parameters). Good choice when computational budget is limited, though leaves significant performance on the table compared to Very Deep.
Ablation Study:

- Baseline → Simple Deep: +2.45% (adding depth helps)
- Simple Deep → Deep + BatchNorm: -0.69% (BatchNorm alone insufficient)
- Deep + BatchNorm → Very Deep + Dropout: +2.87% (dropout crucial)
- Total gain: +4.63% from architectural improvements

Overfitting Analysis: Baseline's 2.09% gap reflects underfitting (too simple to capture data complexity). Very Deep's 5.55% gap is acceptable for 93.76% test accuracy—some overfitting is worthwhile for better absolute performance. Deep + BatchNorm's 7.80% gap confirms insufficient regularization.
Diminishing Returns: Increasing parameters 407× (17K → 6.9M) yields only 4.63% improvement, suggesting we're approaching the dataset's inherent difficulty limit. The remaining 6% error likely stems from genuinely ambiguous cases (e.g., shirts vs t-shirts with similar cuts) requiring higher resolution or external context.
Comparison with State-of-the-Art:

- Our Very Deep + Dropout: 93.76%
- ResNet-18 (literature): 94-95%
- EfficientNet (literature): 95-96%
- Ensemble methods (literature): 96-97%

The gap to state-of-the-art exists because I lack: (1) residual connections for training 50+ layer networks, (2) data augmentation (rotation, cropping, noise), (3) extended training (30 vs 100+ epochs with learning rate schedules), and (4) neural architecture search.
Recommendations by Use Case:

- **Maximum accuracy:** Very Deep + Dropout (93.76%)
- **Best efficiency:** Wide + Deep (92.98%, 17× faster)
- **Limited compute:** Simple Deep (91.58%)
- **Real-time inference:** Linear SVM on CNN features (90.96%)

3 Conclusion

Part I: SVMs efficiently separate linearly separable data with minimal support vectors (only 3 needed for clean Gaussian clusters). For overlapping data, the C parameter controls the trade-off between margin width and error minimization. Our experiments showed performance plateaus at $C \geq 1.0$ when data overlap fundamentally limits achievable accuracy (95%).
Part II: The progression from shallow classifiers (87.57%) to baseline CNN (89.13%) to deep CNN (93.76%) demonstrates several key principles: (1) learned features dramatically outperform raw pixels, (2) CNN features make data linearly separable (Linear SVM: 90.96%), (3) depth combined with proper regularization is crucial (+4.63%), but (4) diminishing returns emerge (407× parameters $\rightarrow$ 4.63% gain). The best architecture depends on deployment constraints—Very Deep + Dropout for maximum accuracy, Wide + Deep for efficiency, or Linear SVM on CNN features for real-time applications.

Declaration of Used AI Tools

Tool	Purpose	Where?	Useful?
ChatGPT	Rephrasing, Adjusting text length	Throughout	++
ChatGPT	Debugging non working code	Throughout	++