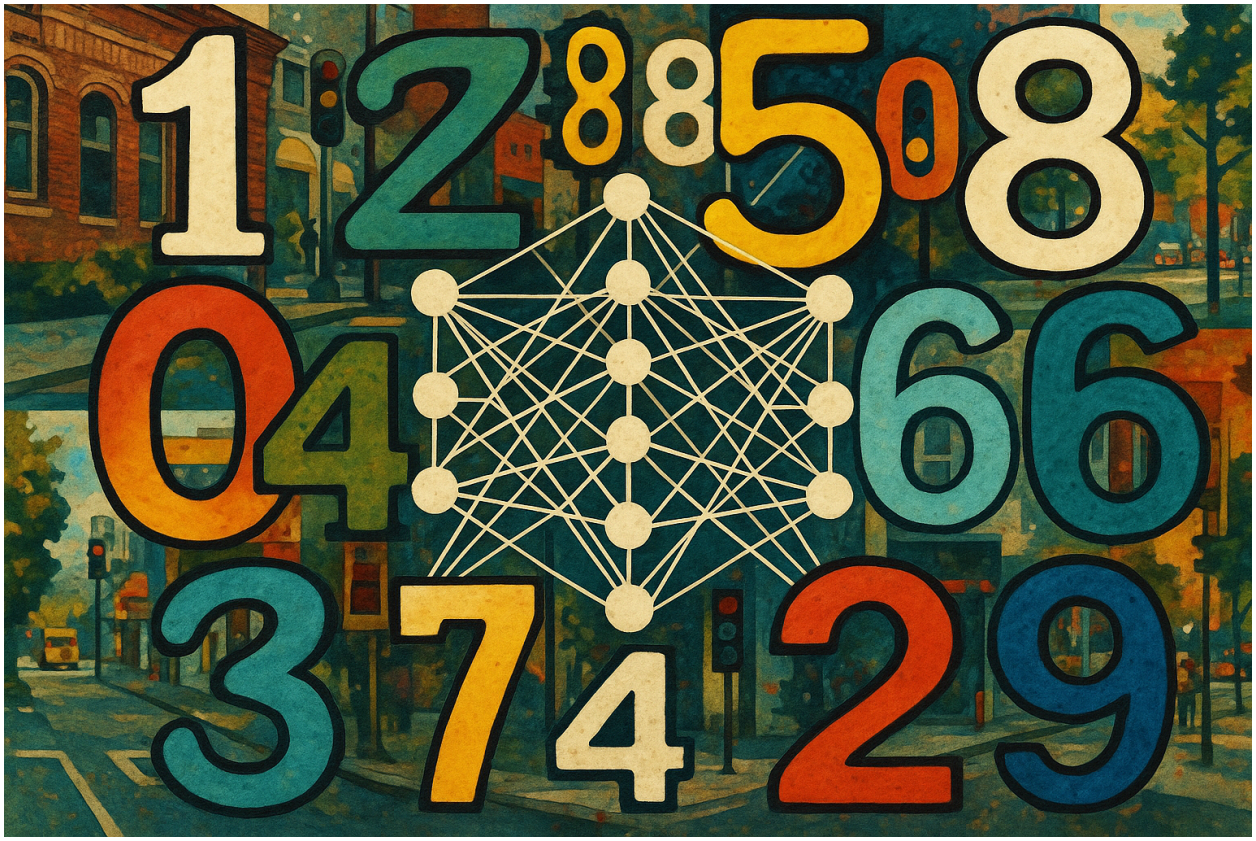
SVHN

Utilizing machine learning to classify house numbers.



Adam Nam
Zongze Li
Spencer Peng

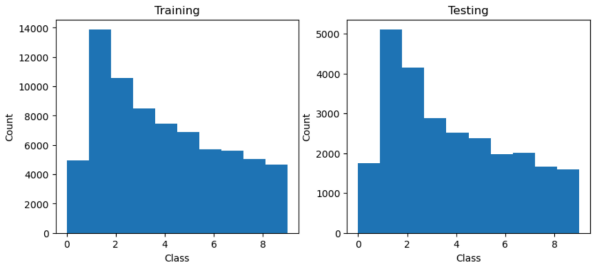
Dec 12, 2025
CS 178

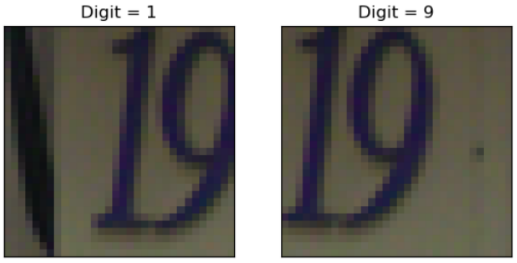
INTRODUCTION

Our group selected the Street View House Numbers (SVHN) dataset from Stanford University, which contains training and testing datasets with 32×32x3 RGB images labelled a digit from 0 to 9, corresponding with the centermost digit in the image.

Data Exploration

Before training our models, we noticed a higher frequency of images labeled as digits 1, 2, and 3, which raised concerns about potential overfitting toward these classes. Because of this imbalance, we considered how well a naïve baseline model would perform if it simply predicted the most common digit for every image which resulted in an accuracy of 6.8% on the training set and 6.7% on the testing set.

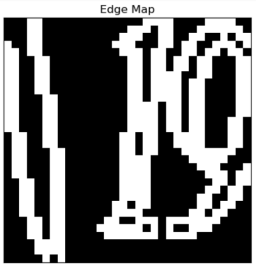




Another issue we observed was the significant amount of noise in the images. Since the dataset is collected from real-world house numbers, it naturally includes noise from neighboring digits, varying lighting conditions, and background elements. These challenges further emphasized the importance of applying careful preprocessing to improve model robustness and ensure reliable learning.

Data Proprocessing

We applied three main preprocessing techniques to our image dataset: normalization which was done by dividing all pixel intensities by 255, edge mapping, and HOG feature extraction.



Edge mapping was used to test whether emphasizing structural information could improve classification accuracy. We applied a simple gradient-based edge detector that computes horizontal and vertical derivatives using the kernel [−1,0,1][-1, 0, 1][−1,0,1]. This filter was convolved across the image in both directions to produce horizontal and vertical gradient maps. The gradient magnitude at each pixel was then calculated via the Pythagorean theorem. To generate a binary edge map, we kept only pixels whose gradient magnitude was at least half of the maximum magnitude in the image. The goal of this approach was to highlight meaningful structural boundaries while reducing the influence of noise and irrelevant texture.

HOG feature extraction was used primarily for the SVM model. Using the hog function from scikit-image, each grayscale image was converted into a histogram of oriented gradients. Like the edge map computation, HOG first obtains gradient magnitudes and orientations (via the arctangent of the vertical and horizontal derivatives). The image is divided into cells, and gradient magnitudes are accumulated into 9 orientation bins per cell according to the direction of the gradient. The resulting feature vector has the form [cell1_bin1, cell1_bin2, ..., cellN_bin9][cell_1_bin_1,\; cell_1_bin_2,\; ..., \; cell_N_bin_9][cell1_bin1,cell1_bin2,...,cellN_bin9]. These descriptors preserve local shape information while being robust to small illumination or noise variations. The motivation behind using HOG features was to provide the SVM with a structured, high-level representation that captures edges and orientations more effectively than raw pixels.

BASELINE MODELS

Before continuing with our core models, we decided to explore a few simple baseline models. Since we were still in the exploratory phase and understood that the dataset was complex, we did not focus on optimizing these baseline models. Nevertheless, even without extensive tuning, we observed a notable improvement in accuracy compared to the naïve baseline model. Our selected baseline models and their respective accuracies were as follows:

MODEL	TRAINING ACCURACY	TESTING ACCURACY
Logistic Regression	32.5%	25.7%
Decision Tree	99.9%	42.0%
Artificial Neural Network	78.0%	78.0%

RANDOM FOREST

We evaluated a Random Forest for classifying the SVHN dataset. A preliminary sanity check using a single decision tree outperformed our logistic regression baseline, motivating the use of an ensemble model..

Baseline Performance

The baseline Random Forest trained on normalized pixels achieved 99.9% training accuracy and 68.2% testing accuracy, indicating substantial overfitting. Our goal was to reduce this gap while improving generalization.

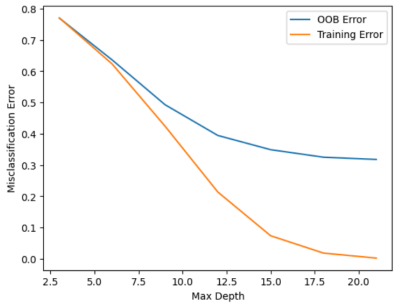
Out of Bag Error

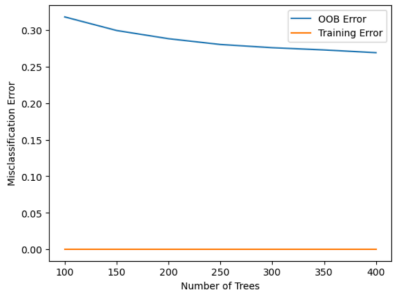
We optimized the model using OOB error rather than K-fold cross-validation. Because each tree trains on a bootstrap sample, the unused out-of-bag data naturally serves as a validation set for that tree. Averaging prediction errors over these instances provides an efficient estimate of generalization without requiring extra data or computation.

Optimization

We tuned max depth, number of trees, number of features per split, and minimum samples per leaf.

- Max Depth: Testing depths from 3–21 showed diminishing returns around 18–20. Although depth=20 worked well, the fully tuned model performed best with unlimited depth.
- Number of Trees: Increasing trees consistently lowered OOB error. We selected 350 trees, as improvements beyond this were negligible.
- Features per Split & Min Samples per Leaf: Adjusting these from their defaults which were square root of features and 1 sample per leaf provided no benefit, so we kept the default values.





The tuned model reached 99.9% training accuracy and 70.0% testing accuracy, a 1.8% improvement over the baseline, though overfitting remained.

Edge Map

We also trained the Random Forest on edge-mapped images to reduce noise effects. The default configuration yielded 92.9% training and 49.8% testing accuracy, lower overall performance but much less overfitting which showed some promise. However, OOB tuned parameters only reached 92.9% training accuracy and 51.6% testing accuracy, showing only minor improvement and confirming that edge maps were not effective for Random Forest classification in this task.

SUPPORT VECTOR MACHINE (SVM)

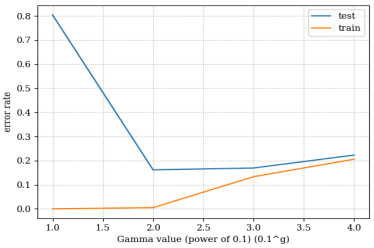
The motivation for using SVMs in combination with HOG features is to create a model that has some spatial awareness of features in a way that is independent from lighting or coloring. The SVM will learn different features based on the hog features in different locations of the image which will give the SVM the ability to learn some semblance of spatial awareness of the image. The HOG feature extraction will change every image into a feature vector of 324 orientation bins with the parameters, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)` for the hog function. The SVM will learn which features, orientations and cells are the most significant for classifying the numbers across all the images.

Baseline Performance

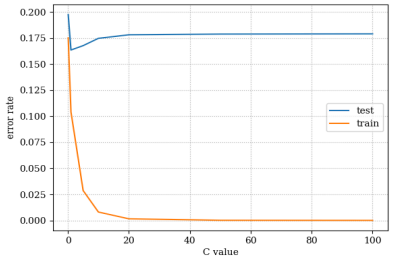
The baseline model has a training error of around 18% and a testing error of 20% across all 5 folds with no tweaking of the hyperparameters. This shows that the model is relatively well fit as the test and training error are quite similar showing it to be unlikely to be overfitting. It could, however, be underfitting but would require tuning hyperparameters to see. The model trained on the edgemaps represents a similar level of fitting with a train error of 48% and a test error of 48%, there is no indication of overfitting but it could likely be underfit due to very high error rates.

Optimization

The first parameter to be tuned was the kernel, 4 different kernels were tried, linear, polynomial, rbf and sigmoid. The best result for the normal image was rbf, which has a training error of 8.2% and a test error of 15%. The higher training error of rbf compared to the polynomial kernel indicated that it is less overfit than the polynomial kernel. These results show that the data of the features is relatively complex and requires a complex kernel to find relationships in the data. The rbf kernel tends to classify grouped or clustered data well which indicates that the feature vectors have many points close to each other that are of high significance, which makes sense due to the fact that numbers create connective shapes. This was validated with 5 fold cross validation.



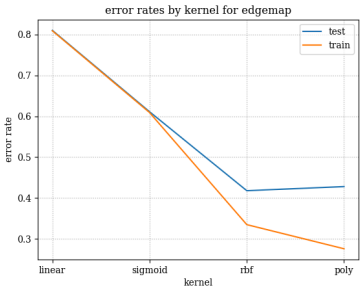
The second parameter to be tuned was the C value,with the values. A smaller C value like 1 performing better than larger values such as 10 indicates the dataset to be quite noisy as more misclassifications in the training data lead to overall better results, with the lowest test error rate at a C value of 1. This generally confirms the belief that the dataset is quite noisy and is clustered, as a large C value does not really have a bad test rate, showing that the data is relatively separated and easy to split, which makes sense as numbers oftentimes have vastly different shapes.

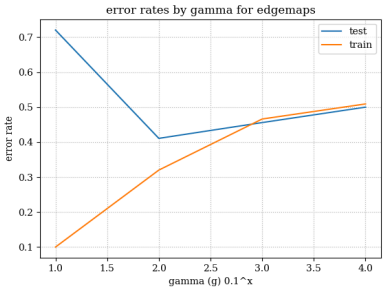


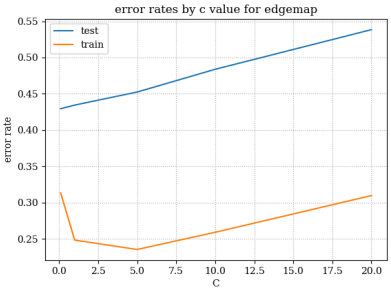
The Third parameter to be tuned was the gamma value. The gamma value affects the sphere of influence of a point which can be important for an image recognition system that requires some sense of spatial awareness. The data shows that there is very high overfitting at 0.1 which then sharply drops and then later starts to rise. This indicates a relatively low gamma value of around 0.01 or 0.001 as the optimal value to train the model on this can confirm the belief that the housing number data set is a relatively noisy and clustered dataset. The data clearly is overfitting at a high gamma like 0.1 but shows that the complexity of the model can be accurately represented with a smaller gamma like 0.01 or 0.001 while even smaller values likely cause the model to be too simple to accurately represent the data.

Edge Map

The same was overserved in the edgemap image. The higher training error of rbf and the lower compared to the polynomial kernel indicated that it is less overfit than the polynomial kernel. These results show that the data of the features is relatively complex and requires a complex kernel to find relationships in the data but does not require the level of complexity that a polynomial kernel can provide. It is likely that the levels of noise that is being removed by creating an edgemap is making the SVM optimize in ways that are unintended as the edgemaps will make the extracted HOG features to be much sharper and clean which could reduce the effectiveness of the SVM's ability to find an optimal middle ground as a lot of the noise that could guide the fitting is being removed.







The second parameter to be tuned was the gamma value. The values have a similar result to the version without the edgemap but to a much lower degree, with the lowest test error also at a gamma value 0.01. The results and graph seem to resemble the one from without an edgemap,

just at a lower accuracy.
The third parameter to be tuned was the C value. The data seems to follow a similar pattern for the model without the edgemap and has the lowest test error at C=0.1.
We believe the overall performance of the edgemaps failed because SVMs as a model benefit from noisy data to guide the support vectors to create good margins.

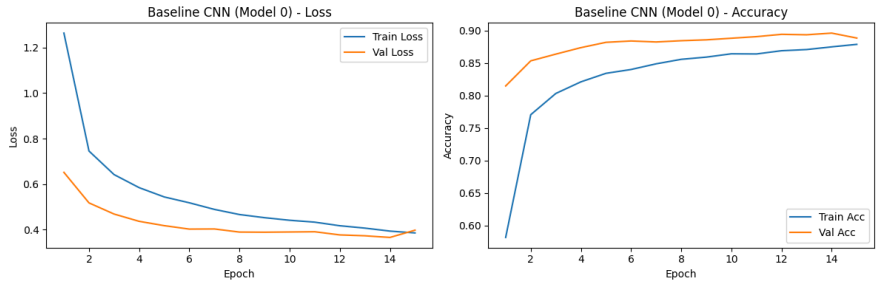
CONVOLUTIONAL NEURAL NETWORK (CNN)

We chose a convolutional neural network (CNN) because SVHN consists of small 32×32 RGB images where spatial structure matters. CNNs use local receptive fields, weight sharing, and pooling, which lets them learn edge and stroke patterns and build up to full digit shapes while being robust to small translations, lighting changes, and background clutter. Compared to a fully connected network, a CNN is far more parameter-efficient on images and leverages the 2D structure instead of treating pixels independently. Since CNNs are the standard for digit recognition tasks such as MNIST and SVHN, they are a natural fit for this project.

Baseline Performance

Our baseline CNN used two Conv–ReLU–MaxPool blocks (3→32→64 channels with 3×3 kernels and 2×2 pooling), followed by a 128-unit fully connected layer with ReLU, dropout p=0.5, and a 10-way output layer. Trained for 15 epochs with default optimizer settings, it achieved 87.9% training accuracy and 88.9% validation accuracy. The train/validation curves stayed close and validation accuracy slightly exceeded training accuracy near convergence, suggesting the model learned a useful representation without severe overfitting while leaving room for improvement.

MODEL	EPOCHS	TRAINING ACCURACY	VALIDATION ACCURACY
Baseline CNN (Model 0)	15	87.9%	88.9%

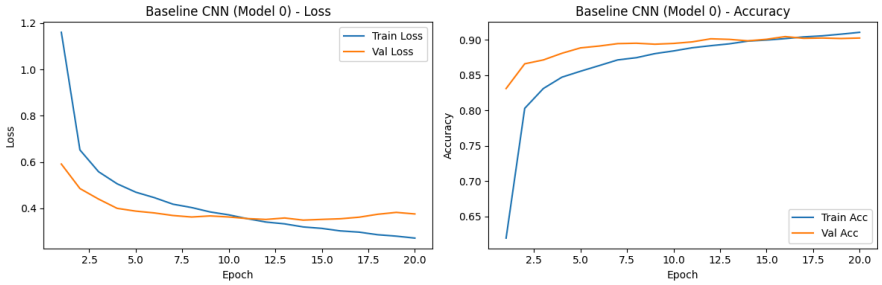


Baseline CNN after 15 epochs.

Epoch Tuning

We treated the number of epochs as a hyperparameter and compared 15 vs. 20 epochs with all other settings fixed. Extending training from 15 to 20 epochs improved training accuracy from 87.9% to 91.0% and validation accuracy from 88.9% to 90.2%. The learning curves suggested that performance was still improving but beginning to plateau around 20 epochs, with only mild overfitting. We therefore fixed 20 epochs as our default budget for later CNN experiments.

MODEL	EPOCHS	TRAINING ACCURACY	VALIDATION ACCURACY
Baseline CNN (Model 0)	15	87.9%	88.9%
Baseline CNN (Model 0)	20	91.0%	90.2%



Baseline CNN after 20 epochs.

Architecture Tuning

With the training budget fixed at 20 epochs, we varied the depth of the convolutional stack while keeping the dense layer (128 units) and dropout p=0.5 constant:

- **CNN A (shallow):** one conv layer with 32 filters
- **CNN B (baseline):** two conv layers with 32 and 64 filters
- **CNN C (deep):** three conv layers with 32, 64, and 128 filters

The shallow CNN A underfit, reaching only 70.1% training accuracy and 83.8% validation accuracy. CNN B significantly improved both metrics to 90.1%. CNN C achieved the best performance, with 95.5% training accuracy and 92.0% validation accuracy, showing that adding a third conv block improved feature quality at the cost of slightly more overfitting. Because pooling reduces spatial dimensions, CNN C actually used the fewest parameters among the three while achieving the highest validation accuracy, so we adopted CNN C as our final architecture.

MODEL	CONV FILTERS	DENSE UNITS	DROPOUT	PARAMETERS	EPOCHS	TRAINING ACC.	VALIDATION ACC.
CNN A (Shallow)	[32]	128	0.5	1,050,890	15	70.1%	83.8%
CNN B (Baseline)	[32, 64]	128	0.5	545,098	20	90.1%	90.1%
CNN C (Deep)	[32, 64, 128]	128	0.5	356,810	20	95.5%	92.0%

Regularization Tuning

Using CNN C, we compared five regularization configurations (all trained for 20 epochs with identical optimization settings):

1. No dropout, no L2
2. Dropout p=0.3, no L2
3. Dropout p=0.5, no L2
4. Dropout p=0.3 with L2=1e−4
5. Dropout p=0.5 with L2=1e−4

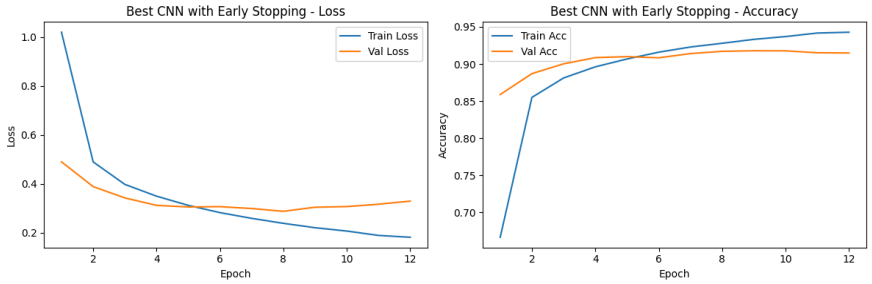
Without regularization, the model reached 98.6% training accuracy but only 91.0% validation accuracy, indicating overfitting. Dropout p=0.3 improved validation accuracy to 92.4% with a modest drop in training accuracy to 96.8%. Stronger dropout (p=0.5) slightly hurt validation performance, and adding L2 weight decay on top of dropout produced similar but not better results. We therefore chose dropout p=0.3 without L2 as our final configuration, as it gave the best validation accuracy with a simple setup.

EXPERIMENT NAME	DROPOUT	WEIGHT DECAY	TRAINING ACCURACY	VALIDATION ACCURACY
reg_1_no_dropout_no_l2	0.0	0	98.6%	91.0%
reg_2_dropout_0.3_no_l2	0.3	0	96.8%	92.4%
reg_3_dropout_0.5_no_l2	0.5	0	94.6%	91.8%
reg_4_dropout_0.3_l2_1e-4	0.3	1e-4	96.6%	92.3%
reg_5_dropout_0.5_l2_1e-4	0.5	1e-4	95.7%	91.9%

Early Stopping

Finally, we added early stopping to reduce unnecessary training and further control overfitting. Using our best architecture and regularization, we trained for up to 40 epochs with patience 4 based on validation loss. Validation accuracy rose quickly to about 91.7% by epoch 8, after which validation loss stopped improving and began to increase slightly while training loss continued to decrease. Early stopping triggered at epoch 12 and restored the weights from epoch 8. This achieved similar validation performance to our 20-epoch runs while effectively halving the training budget.

MODEL	EPOCH AT EARLY STOPPING	TRAINING ACCURACY	VALIDATION ACCURACY
Best CNN with Early Stopping	8	92.8%	91.7%

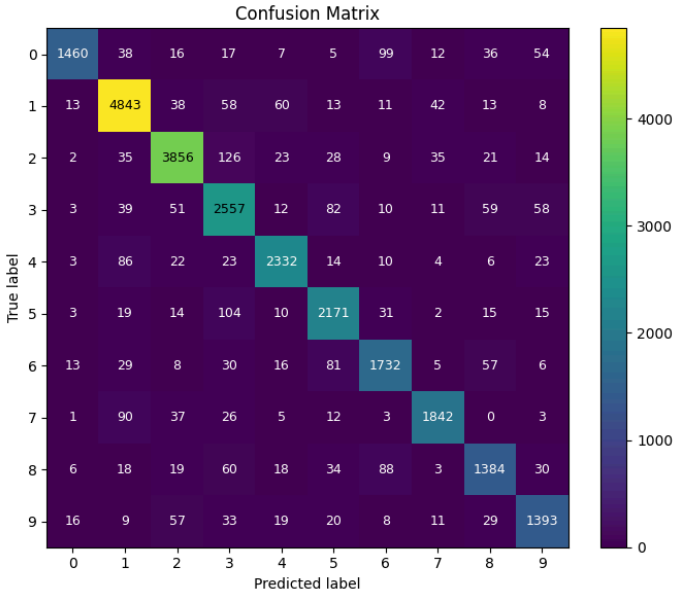


Loss & accuracy comparison for the best CNN model with Early Stopping.

Evaluation on Test Dataset

The final CNN (deep architecture, dropout p=0.3, early stopping) achieved 90.54% test accuracy and 0.3809 test loss, close to its validation performance (≈91–92%). Per-class accuracies were mostly above 90% for digits 1, 2, 4, 5, and 7, and in the low-to-upper-80s for the remaining digits. The confusion matrix shows most errors between visually similar digits (e.g., 2↔3, 5→3, 0↔6/9, 8↔3/6), indicating that the model learned reasonable digit structure but struggles on borderline cases where strokes are blurred, occluded, or low contrast.

MODEL	TESTING LOSS	TESTING ACCURACY
Final CNN	38.1%	90.5%



Confusion matrix for our final CNN model's performance on the test dataset.

CONCLUSION

After training and tuning our models on the SVHN data we found the following accuracies and F1 score:

MODEL	TRAINING ACCURACY	TESTING ACCURACY	F1 SCORE (WEIGHTED)
Random Forest	99.9%	70.1%	0.695
SVM	91.8%	85.2%	0.850
CNN	92.8%	90.5%	0.906

As expected, the CNN performed the best overall, achieving the highest test accuracy and weighted F1 score, and demonstrating strong generalization on this image-based task. The SVM performed second-best with reasonably strong generalization, though below the CNN, consistent with the fact that SVMs can handle feature-rich image data but lack the hierarchical feature extraction ability of deep networks. Lastly, the Random Forest achieved extremely high training accuracy but much lower test performance, indicating severe overfitting expected because decision-tree ensembles struggle with raw pixel inputs and lack spatial bias.

Edge Maps on SVHN

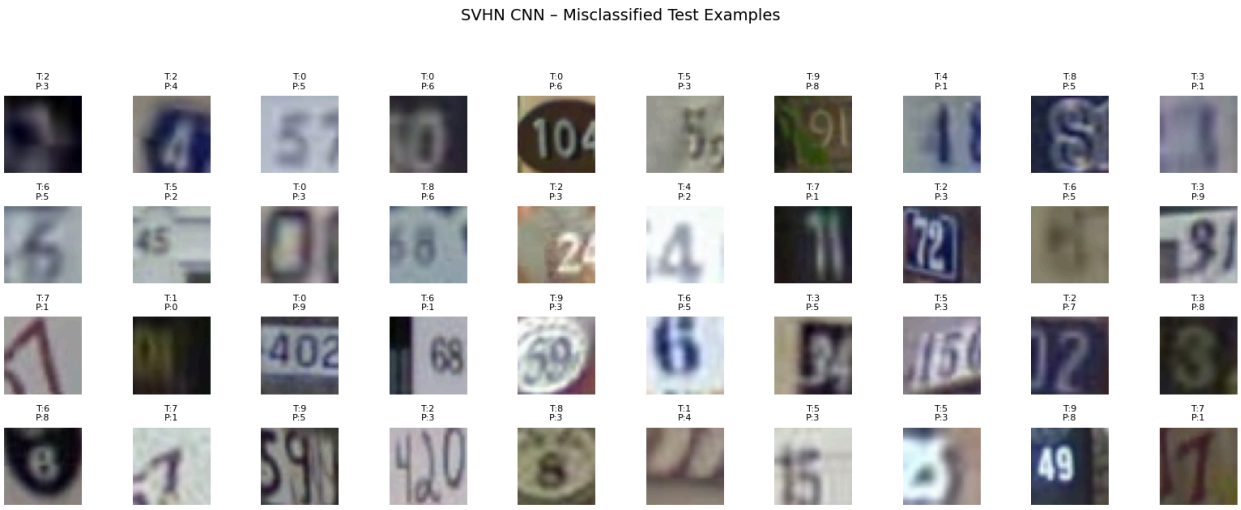
The edge maps did not perform well because much of the discriminative information in SVHN is encoded in the color channels, which help distinguish meaningful digit structure from background clutter. When the images are converted to edge maps, this color information is lost, making it more difficult for the models to separate the digit from irrelevant features such as building edges, shadows, or other background contours. Noisier datasets can also benefit certain optimization strategies like SVMs, whose support vectors can be better guided by noisy datasets. As a result, preprocessing steps that aggressively remove noise can

unintentionally discard useful information and ultimately hurt model performance.

CNN: Misclassified examples and error analysis

Inspecting misclassified test images confirms that many errors occur on genuinely difficult cases. Several crops contain multiple overlapping digits where the labeled digit occupies only part of the patch, so the network sometimes predicts the more salient or centered digit instead of the target. Other failures involve digits that are blurred, low contrast, partially occluded, or cut off at the frame boundary. Stylistic variations in stroke thickness and font also cause confusion, such as “5”s that resemble “3”s or flat-topped “2”s that look similar to “3”s.

These qualitative patterns align with the confusion matrix, which shows systematic mistakes like 2→3, 5→3, and 0→6/9. Overall, the CNN has a solid notion of digit shape when the digit is clear and isolated, but its predictions degrade when context, cropping, and noise make the single 32×32 patch ambiguous.



Example SVHN test images that our CNN misclassified. Each patch shows the true label (T) and the predicted label (P), illustrating common failure modes such as motion blur, low resolution, low contrast, cluttered or overlapping digits, and ambiguous digit shapes.

CNN: Looking ahead and possible improvements

The remaining errors suggest several directions for future work. On the preprocessing side, stronger data augmentation (e.g., random crops, small rotations, brightness/contrast jitter, Gaussian noise, and cutout) could make the model more robust to blur, occlusion, and low contrast. Architecturally, replacing our hand-designed CNN with a deeper residual network or a lightweight pretrained backbone adapted to 32×32 images could improve feature extraction, especially for subtle stroke patterns. Since many crops contain multiple digits, a more realistic system might also combine digit detection with sequence modeling rather than classifying each crop independently.

For evaluation, we focused mainly on overall accuracy. Future work could incorporate per-class F1 scores and calibration analysis to better understand where the model is overconfident or systematically biased. Combining these metrics with qualitative error inspection, as we did with the misclassified grids, would help target improvements on the most ambiguous digit pairs and the most challenging visual conditions.

Limitations

Our primary limitation in this study was the limited training time, which constrained both the number of models we could train and the preprocessing methods we could explore. With more time, we would experiment with encoding spatial information along with color into the input features to potentially improve the performance of the Random Forest and SVM models. Additionally, we could apply data augmentation techniques such as pixel shifts to make the models’ predictions more robust, and implement filters to better isolate the target digits in each image, reducing the impact of background clutter. We could also try applying other image processing techniques such as gaussian blurs or upscaling with bilinear interpolation or AI upscaling to increase the quality of the images.

DIVISION OF WORK

Adam Nam:

- Data exploration section
- Baseline models section
- Code, results, and graphs for Logistic Regression Classifiers, Decision Trees, and Random Forests
- Conclusion section

Zongze Li

- Data exploration section
- Baseline models section
- Code, results, and graphs for ANNs and CNNs
- Conclusion section

Spencer Peng

- Data exploration section
- Code, results, and graphs for SVMs and Edgemapping
- Conclusion section