ECE 3200 Final Project

Rohan Kalluraya (rak298) and Nimish Goel(ng449)

Introduction

In this project, we developed a digit identification algorithm capable of detecting which pair of digits (among 0, 2, 3, 5, 7, and 9) is present in a given handwritten image. Each unique pair corresponds to a specific class label. After preprocessing the data, we evaluated three different classification models. As our baseline models, we implemented Naive Bayes and Softmax Regression, achieving accuracies of 63% and 85.7% respectively. Finally, we built a neural network that achieved a 91.4% accuracy on the public test set.

Data pre-processing

For our **baseline models (Naive Bayes and Softmax Regression)**, we first applied a **Gaussian Blur** to each image to reduce noise. This was done using a 3×3 Gaussian filter, replacing each pixel with a weighted average of its neighbors. To extract meaningful features, we used the **Histogram of Oriented Gradients (HOG)** technique. The image was divided into 4×4 pixel cells, and gradients were calculated for each cell. These gradients were then binned into histograms and normalized over 2×2 cell blocks to increase robustness to lighting changes. This process resulted in a 1D feature vector representing each image.

For the **neural network**, we used a different preprocessing strategy. Each 28×28 image was flattened into a 784-dimensional vector. The pixel values (ranging from 0–255) were normalized to a [0, 1] range to improve training stability.

Models

1. Naive Bayes

Architecture

We used Gaussian Naive Bayes to train the model. The model has the “naive” assumption of conditional independence between every pair of features given the value of the class variable. The Gaussian Naive Bayes model assumes each element of the HOG vector follows a normal distribution, conditioned on the class. For a test image, it calculates the likelihood of its HOG features under each class, then predicts the class with the highest posterior probability.

Among all the Naive Bayes models, we used Gaussian Naive Bayes for image classification with HOG features because the continuous, real-valued nature of HOG descriptors fits naturally with the Gaussian distribution assumption. In contrast, models like MultinomialNB, BernoulliNB, and CategoricalNB are designed for discrete features — like word counts, binary indicators, or categorical labels — which don't match the gradient-based, floating-point values that HOG produces.

The likelihood of features conditioned on class follows the following formula. GaussianNB() optimizes the model for maximizing the likelihood of the observed train data using this formula by using MLE to find mean μ_y and variance σ_y.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

This model differs from the Multivariate Binomial distribution and multinomial binomial distribution model we learnt

in class as it doesn't involve text data and thus we can't simply count the number of times a feature occurs as the data is continuous and in this model, the features will not occur again so the learnt model will be highly inaccurate.

Optimization Method

The parameters mean and variance for the Gaussian Distribution of each feature conditioned on its class are estimated using maximum likelihood. During training, the mean and variance for each feature per class is directly computed and no optimization loop is necessary. This is fast and requires no tuning during training which is one of the reasons why Naive Bayes is used today.

Training and test performance

The training validation performance is 0.63 for this model which we obtained using a validation dataset created using Scikit's test-train split function on the train data. The public test performance was 63.94%.

2. Softmax Regression

Architecture

The HOG features define the inputs that get separated using softmax regression. The likelihood of a particular sample belonging to a class is modelled using a categorical variable. Then, maximum likelihood estimation is used to find the optimal parameters for each class. We exponentiate the logits for each class and normalize them to get the probability of finding each class. This is achieved using scikit's LogisticRegression with the default hyperparameter multi-class = 'multinomial.'

We are minimizing the cross-entropy loss(negative log likelihood) which is maximizing the likelihood estimate. The cross-entropy loss has the formula :

$$\text{Loss} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij} \cdot \log(p_{ij})$$

We are maximizing the probability assigned to the correct class for each sample(by averaging over all samples).

Optimization Method

The optimization method used is the Limited-memory Broyden–Fletcher–Goldfarb–Shanno(LBFGS) algorithm. This algorithm is similar to Newton's method of optimization that we learnt in class. Newton's method uses the Hessian to obtain parameters. It requires computing and inverting the full Hessian matrix which is computationally expensive. On the other hand, LBFGS uses an approximate Hessian matrix that is constructed using gradient history and not actual second derivatives. Update rule is similar to Newton's method except that it now uses the approximate Hessian matrix rather than the actual Hessian Matrix.

We used this optimization method over others as it's a second-order method, so it converges faster than first-order methods like SGD, especially on smooth convex problems (like softmax regression). It uses less memory than Newton's method and often needs fewer iterations to converge.

Training and test performance

The softmax regression was our best performing baseline model with an accuracy of 0.8574 on validation data created using Scikit's test-train split function on the train data. The public test accuracy on Kaggle is 85.739%.

3. Neural Network

Architecture

We fed the Neural Network the pixel values as the input. They represented a number from 0 to 255 for the RGB value. The dimension of the flattened 28*28 pixel value image was a 1D vector of size 784. The output was 15 neurons as they could belong to one of the classes. Through trial and error, we also had 2 hidden layers of sizes

ECE 3200 Final Project

Rohan Kalluraya (rak298) and Nimish Goel(ng449)

384 and 192.

Both the hidden layers used the following transformation on data they received : Linear transformation → ReLU → Dropout. Dropout is a regularization technique to prevent overfitting. It randomly sets some neuron outputs to zero during training. We had a dropout_rate=0.3 which means that 30% of neurons are "dropped" each pass. This helped us improve our public test performance.

The output layer performed a final linear transformation to be classified into one of the 15 categories.

We optimized for Pytorch cross-entropy loss which is :

$$\mathcal{L}(z, y) = -z_y + \log\left(\sum_{k=0}^{C-1} e^{z_k}\right)$$

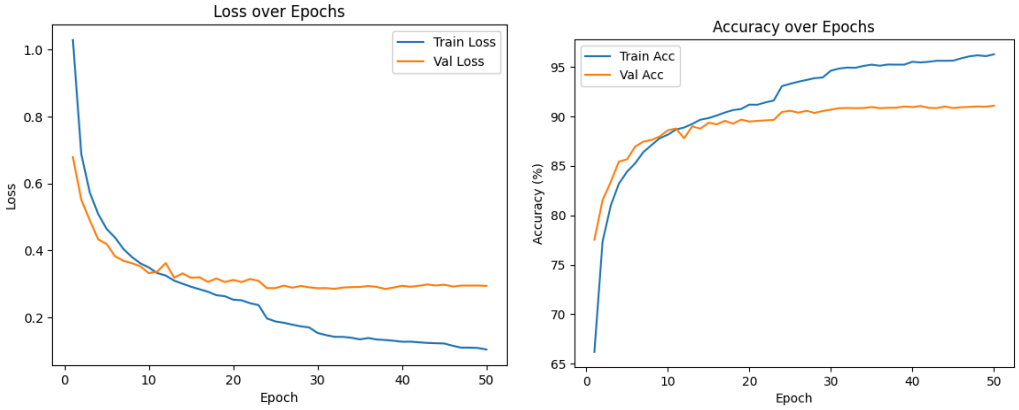
Optimization Method

We used the Adaptive Moment Estimation(Adam) optimizer. It adapts the learning rate for each parameter individually. Maintains two moving averages for each parameter: One for the gradient (momentum-like) One for the squared gradient (like RMSProp). We set the learning rate to be 0.001 which is a common default for Adam and usually works well. Adds L2 regularization (aka weight decay = 1e-5) to the loss to prevent overfitting by penalizing large weights.

We ran 50 epochs through the entire training data and chose the model that performed best on validation tests.

Training and test performance

Training validation performance is 91.07% for this model which we obtained using test data that we created using Scikit's test-train split function on the train data. The public test performance is 91.07%.



In these plots, we can see that as we increase the number of epochs, the validation loss continues to reduce and accuracy continues to increase. This means that the model is not overfitting after 50 epochs.

Conclusion and Future Work

In conclusion, we can see that neural networks with 2 hidden layers generalize well for this problem. Future ideas to try include implementing a CNN architecture since it outperforms MLP at image classification. They do not lose the spatial information which is lost in the MLP due to flattening during pre-processing. CNNs are insensitive to rotations, positions, etc. of the object in image in comparison to MLP. There is a possibility to try ensemble methods combining CNN, MLP and SVM, which may help increase test accuracy. In addition, for the MLP we implemented, we can include HOG features extracted and feed them along or without the raw data to experiment with the network performance. Another idea recommended by few papers was to perform a Hyperparameter grid search using loops or sklearn's *ParameterGrid* with different learning rates or batch sizes.

ECE 3200 Final Project

Rohan Kalluraya (rak298) and Nimish Goel(ng449)

AI Usage Declaration:

Used Claude for the Baseline notebook implementation debugging. Specifically, for feature extraction using HOG, I used Claude to understand the parameters to be used and to ensure that the features that I extracted were relevant to the classification. Also, I used Claude to understand how to use the OpenCV library for preprocessing. I did not have experience with such a classification problem in the past and will be able to use these libraries myself since I understand the documentation for these.

For the NN code, I used ChatGPT to understand the syntax involved in implementing a NN with appropriate layers and training it (along with validation). I asked it to explain the next steps after I had created the MLP class and it provided me with some debugging help as well as suggestions for a better loss function and learning rate scheduler for increasing accuracy. It also was used for debugging help with the epochs syntax, storing the model and then final prediction generation. We will be able to produce this model and data pipeline independently in the future because of asking GPT to explain the rationale behind each step it suggested as well as tinkering around with the number of epochs, neuron layers a little to reach 90% accuracy which helped provide a better idea about the architecture and optimization method as done in practice.

For both notebooks, ChatGPT was used to produce the .csv file in Kaggle submission format. Since Kaggle was new to us, I understood this and can use it independently in the future.

Referenced the following link to gain a basic understanding of the NN task.

https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/