Image Classification Report

Kamal Yassin

Hussam Al Nabtiti

Introduction to Design Choices:

For this project, we decided to implement a convolutional neural network (CNN) method. The CNN model implemented demonstrates a methodical and efficient approach to solving the problem at hand. Throughout this report, we have thoroughly explained the design of the preprocessing steps, the training process, the learning algorithm, the results, and the possible limitations. All these elements of the implementation are deeply important for understanding why our approach is effective, or how it could possibly be improved.

Model & Architecture:

The model we built was implemented as a multi-layer convolutional neural network, which is a class of machine learning algorithms that aim to resolve image-related and image classification problems and tasks. Our architecture starts with a series of convolutional layers, which are essential in extracting the features of the images. These layers utilize filters in order to identify feature patterns such as textures, shapes, or even specific edges, which enables the model to learn and understand the structure of the hand-drawn digits. The CNN's convolutional layers use 3x3 filters which are ideal for encapsulating and isolating even the most minute details in the image while also keeping the costs associated with computing to a minimum.

In our model, we have opted to utilize batch normalization after each convolutional layer. This ensures that the outputs of the previous layer are normalized to facilitate a faster and more stable learning process. On the other hand, batch normalization also helps prevent covariate shift, ensuring that each layer receives a consistently distributed input, which helps speed up convergence.

A major addition to our architecture would be the addition of max-pooling layers. We have chosen to implement these layers after groups of convolutional layers to reduce the

dimensions of feature maps. This technique of downsampling ensures that essential features are retained while getting rid of redundant information which greatly reduces the computational complexity of the following layers. The max-pooling operation also introduces spatial invariance, which ensures that the model is able to recognize exact features regardless of their positions in the image inputs.

On the topic of layers, we have also utilized dropout layers, which are added to the architecture in order to prevent overfitting. By randomly disabling a set of neurons during training, the dropout layers force the neural network to develop more robust and generalized representations. The rates of the dropout are meticulously chosen, with smaller rates in the earlier layers that grow as we progress into fully connected layers (where the probability of overfitting is much higher). The fully connected layers collect the features learned by the convolutional layers and map them to output classes. The final fully connected layer consists of 10 neurons, each corresponding to one of the 10 possible digits (0-9), and uses softmax activation to output probabilities.

Preprocessing:

Before we began training, we opted to reshape the images from flat vectors into a 3D representation of 84 x 28 x 1, where 1 is indicative of a single grayscale color channel. The pixel values are then normalized to a range of 0-1 by dividing by 255. By opting to go with this normalization, we ensure that the input features are consistently scaled, which helps prevent numerical instability during training and hastens convergence.

We have utilized data augmentation to facilitate artificial expansion of the training dataset by applying transformations (such as rotation, zooming, and width/height changes.) These augmentations create possible real world variations of the input images, which simulates scenarios where digits may not be aligned/centered. This strategy improves the model's ability to generalize new data by reducing overfitting and encouraging the CNN to learn different features.

Training Process:

The training process uses the *"Adam"* optimizer, which is a gradient optimization algorithm that combines the advantages of momentum and adaptive learning rates. "Adam" is widely regarded for its efficiency and robustness, especially in training deep networks.

The loss function used is categorical crossentropy, which is appropriate for multiclass classification problems. This function measures the difference between the predicted probability distribution over the 10 digit classes and the true class labels, guiding the optimization process to minimize prediction errors. To ensure efficient and reliable training, two key callbacks are used:

- **Early Stopping**:
    - This monitors the validation accuracy and halts training if it does not improve for five consecutive epochs.
    - By restoring the best weights, early stopping prevents overfitting and saves computational resources
- **Reduce Learning Rate on Plateau:**
    - When validation accuracy plateaus, the learning rate is reduced by a factor of 0.5
    - This helps the optimizer fine-tune the model parameters to escape local minima

The model is trained for a maximum of 50 epochs with a batch size of 64. This batch size strikes a balance between computational efficiency and generalization performance. By splitting the dataset into manageable chunks, the model benefits from frequent updates to its parameters

Results and Interpretation:

The training process demonstrated steady improvement, achieving a validation accuracy of approximately 96.4% and a validation loss of 0.11 at its peak (epoch 17). This indicates the model successfully learned to generalize patterns from the training data. The early stopping mechanism halted training after 22 epochs, preventing overfitting and ensuring efficient use of computational resources.

On the test set, the model achieved an accuracy of 78.56%, which is significantly lower than the validation accuracy. This discrepancy could be attributed to overfitting to the training/validation data or differences in the distribution of the test data. The lower test accuracy

suggests the model may struggle with variations in the unseen test samples, such as noise, distortion, or other real-world inconsistencies not fully captured during training.

These results highlight the model's ability to perform well on validation data while revealing potential limitations in generalization to completely unseen data. Further analysis of misclassified examples could provide insights into specific failure cases, such as overlapping digits or noisy samples

Limitations and Potential Improvements:

The CNN performs well, however there is always room for improvement. Here are the areas where we can improve:

1. **Hyperparameter Tuning:** As they sit right now, the current hyperparameters (learning rate, number of filters, dropout rates, etc) were chosen based on a heuristic. These hyperparameters could be tuned (for example, using Bayesian Optimization) to further enhance performance.
2. **Transfer Learning:** By leveraging pretrained models, we could provide a stronger foundation and could potentially improve generalization to the test set.
3. **Regularization:** By implementing additional regularization techniques, such as weight decay, we could even further decrease overfitting.
4. **Data Augmentation:** The augmentation pipeline we have implemented could benefit from the addition of even more transformations such as shearing or distortions, which could potentially lead to a better simulation of real world variations.
5. **Test Performance**: Our test accuracy could be improved by addressing potential differences between the training/validation and test datasets. We could utilize techniques such as domain adaptation or even expanding the training dataset with samples that better represent real world variations.

Conclusion:

Our implementation of this CNN is a well-designed model that efficiently and effectively balances complexity and generalization. Its well planned architecture, combined with data

preprocessing, augmentation, and a robust training process enables it to achieve high validation accuracy. While the testing accuracy is only marginally lower, the model still demonstrates strong potential for improvement. Our outlined design choices, such as convolutional layers for feature extraction, batch normalization for stability, dropout for regularization, and augmentation for generalization, align with the solution to the task at hand. This model is a strong foundation to deal with the digit classification task, and can even be enhanced further with additional optimizations as outlined in this report.