**MNIST Number Detection using a Convolutional Neural Network**

Austin Ressler, Ayden Dillon, Lucas Chough, Aiden Colberg, Antonio Capporossi

The following are the tools used in creating our convolutional neural network architecture:

**Conv2D Layers**: These layers apply filters to the input image. Each filter detects specific patterns like edges or textures.

**MaxPooling2D**: Reduces the spatial dimensions, which lowers the computation and helps the model generalize.

**Flatten Layer**: Converts the 2D output of the convolutions into a 1D vector to connect with the dense layers.

**Dense Layers**: Adds fully connected layers for classification.

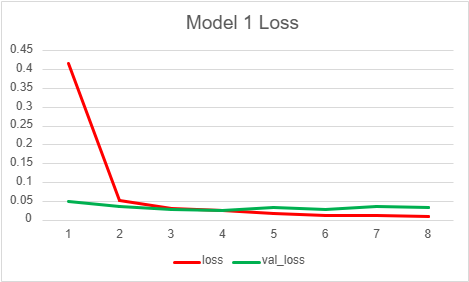
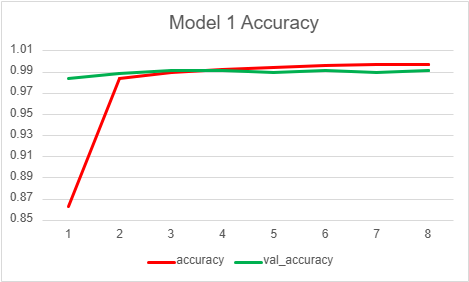
**Output Layer**: Uses softmax for multiclass classification (like the 10 digits of MNIST).

**Epochs**: Each model used 8 epochs

**Batch Size**: Each model maintained a batch size of 64

**Model 1**

In our initial model, we decided to create our convolutional neural network using very simple values and features. We started with a convolutional layer that had 32 filters of size 3x3, utilizing the ReLU activation function, which helps introduce non-linearity into the model. This layer was followed by a max pooling layer with a pool size of 2x2 and a stride of 2. We added a second convolutional layer, increasing the filter count to 64 while maintaining the same kernel size and activation function. This allowed the network to learn more complex patterns from the input data. Again, we followed this layer with another max pooling layer. We then included a third convolutional layer, again with 64 filters, allowing the model to deepen its understanding of the features present in the input images. In the first dense layer, we opted for 64 neurons with ReLU activation, providing the model with the capacity to learn and combine features extracted from the previous layers. Finally, we configured an output layer with 10 neurons and a softmax activation function, which allowed the model to output probabilities for each of the 10 classes, representing digits from 0 to 9. The model was compiled using the RMSprop optimizer, which is effective in handling non-stationary objectives, and we employed categorical crossentropy as the loss function, appropriate for multi-class classification tasks. This architecture sets the stage for more experimentation, letting us build upon this simple model by adjusting parameters, adding regularization techniques, or introducing more complex architectures as needed to improve performance on the dataset.



Analysis

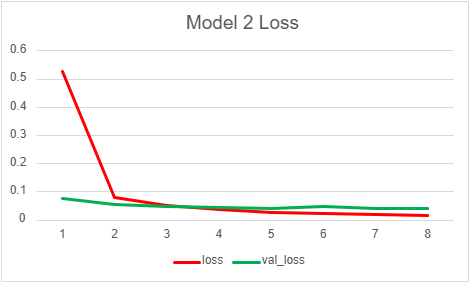
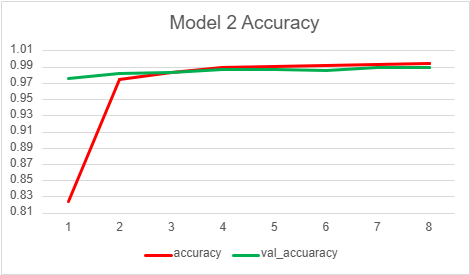
The training loss (red) starts high (around 0.4) and quickly drops, reaching a low level within the first couple of epochs. This suggests that the model is learning quickly and effectively minimizing error on the training data. After the initial sharp drop, the training loss stabilizes around a low value close to zero, indicating that the model has learned to fit the training data well.

The validation loss (green) is relatively low from the beginning and remains stable across epochs, staying at a similar level as the training loss. This low and stable validation loss suggests that the model generalizes well to unseen data and isn’t significantly overfitting the training set. Typically, overfitting would show as the validation loss increasing while the training loss continues to decrease. However, in this case, both losses are very close, implying that the model is balanced between training and validation performance. Both loss curves seem to converge and stabilize within the first few epochs. This indicates that the model reaches a good level of performance quickly and further training may not lead to significant improvements.

The CNN model appears to be well-trained and generalizes well to the validation set. Both the training and validation losses are low and stable, which is ideal.

**Model 2**

In our second model, the first convolutional layer's filter count was increased from 32 to 64, allowing the model to capture more complex patterns in the input images. This was followed by a max pooling layer, which retained the same configuration to effectively downsample the feature maps while preserving the most prominent features. The second convolutional layer's filter count was increased from 64 to 128, further enhancing the model's ability to learn intricate features. The third convolutional layer maintained 128 filters to continue refining the feature extraction process. In the first dense layer, the number of neurons was increased from 64 to 256, providing greater capacity for learning complex representations and relationships in the data. We also added a dropout layer with a rate of 0.5 to help prevent overfitting by randomly deactivating 50% of the neurons during training. Finally, the optimizer was changed from RMSprop to Adam, which is generally more effective due to its better learning rate, often leading to faster convergence and improved results. Collectively, these enhancements hope to create a more powerful and generalized model.



Analysis

Similar to the first model, there is a significant drop in training loss within the first epoch. This indicates that the model is quickly learning to minimize the error on the training data. After the initial drop, the training loss reaches a very low and stable value around or below 0.05, showing that the model effectively learns the training data. The validation loss starts relatively low and stabilizes close to the training loss. This suggests the model is generalizing well and isn’t overfitting significantly, as the validation loss doesn’t diverge from the training loss. With both loss and val\_loss closely following each other, it suggests that the model is well-calibrated and avoids overfitting. The stable and low validation loss implies that the model performs consistently on both training and unseen data. The loss values quickly converge and stabilize after the initial epoch, indicating the model reaches a good performance level early in training, with minimal additional gains from further epochs.

The second model appears well-trained, with both training and validation losses low and closely aligned. The quick stabilization suggests efficient learning, and the lack of overfitting indicates good generalization. Both this model and the first one seem well-tuned for the data, though this model starts with a higher initial loss than the first model. The validation loss isn’t quite as low as the first model, so we will try and make a few changes to see if we can get anything better.

**Model 3**

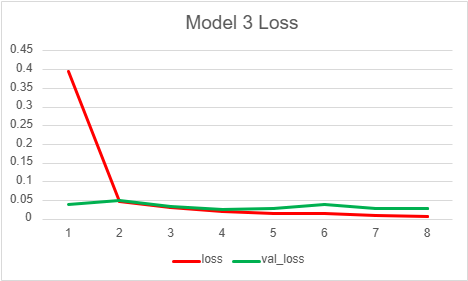
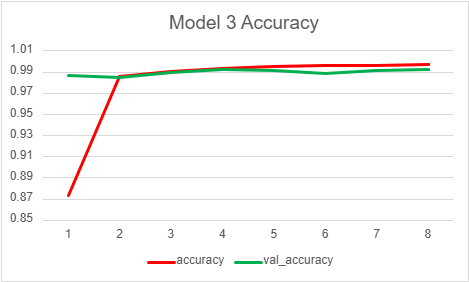
Finally, our third and final model incorporates elements from both the initial and the previously enhanced models, aiming to strike a balance between complexity and performance.

The architecture begins with a convolutional layer that utilizes 64 filters of size 3x3, activated by ReLU. This layer is designed to extract fundamental features from the input images, allowing the network to learn patterns early in the process. Following this, a max pooling layer with a pool size and stride of 2x2 is applied to reduce the spatial dimensions, helping to minimize computational load while retaining the most significant features.

The model then includes a second convolutional layer, again with 64 filters, allowing for the learning of more complex features without significantly increasing the model's complexity. This is followed by another max pooling layer to continue downsampling the feature maps. The third convolutional layer maintains the same configuration, with 64 filters, which helps in further refining the features extracted in previous layers.

After the convolutional and pooling layers, the output is flattened to transform the multi-dimensional feature maps into a one-dimensional vector, making it suitable for the fully connected layers. The first dense layer has 128 neurons, which provides sufficient capacity for the model to learn more abstract representations of the input data. Finally, the output layer consists of 10 neurons with a softmax activation function, designed to output class probabilities for digits 0 through 9.

The model is compiled using the RMSprop optimizer, which is effective for training deep networks, and categorical crossentropy as the loss function, appropriate for multi-class classification. This final model effectively combines attributes from both earlier versions: it maintains a straightforward architecture while incorporating an increased number of filters and neurons for improved learning capacity and generalization. This balanced design aims to optimize the model's performance on the classification task while keeping the architecture manageable.



Analysis

As we can see, the training loss starts high at around 0.4 and again drops sharply to a very low value after the first epoch. This indicates that the model quickly learns and minimizes errors on the training data. Both the training and validation loss stabilize after the first epoch, with only minor fluctuations. This suggests that the model reaches convergence quickly, and further training epochs bring little additional improvement. The validation loss starts at a low level and closely follows the training loss curve, remaining stable and close to the training loss across epochs. This pattern suggests that the model generalizes well and avoids significant overfitting. The closeness of the loss and validation loss curves indicates that this model generalizes well without overfitting. The similarity in loss values suggests consistent performance on both training and unseen data. Like the previous models, this one also shows efficient learning, with most gains achieved after the first epoch and steady convergence afterward. We can’t necessarily see from the graph, but this model had the highest accuracy, validation accuracy as well as had the lowest loss and validation loss out of the 3 models. This suggests that this is the best model of the 3. Sometimes the less complex models can be the better ones.

The third model displays effective learning, with both training and validation losses low and in close alignment. It generalizes very well, showing minimal overfitting. Compared to the other models, this one has the highest starting loss, but it converges quickly to a stable and the lowest loss level, making it the best candidate for a well-trained model.