Lab 11 Report

Name: Jivitesh M S
Roll No.: b24me1039

1(a) CNN Design and Implementation

Model Architecture

A sequential CNN model was designed using tensorflow.keras. The architecture is configurable but follows a standard pattern for image classification:

1. **Input Layer:** The model accepts 28x28x1 grayscale images from the MNIST dataset.
2. **First Convolutional Block:**
   - A Conv2D layer with 16 filters. The filter size is a variable parameter.
   - An activation function (e.g., 'relu' or 'sigmoid').
   - A pooling layer (either MaxPooling2D or AveragePooling2D) with a 2x2 window to downsample the feature maps.
3. **Second Convolutional Block:**
   - A Conv2D layer with 32 filters, again with a variable filter size.
   - An activation function.
   - A second pooling layer (either MaxPooling2D or AveragePooling2D).
4. **Classifier:**
   - A Flatten layer to convert the 2D feature maps into a 1D vector.
   - A Dense (fully connected) layer with 128 neurons and an activation function.
   - A final Dense output layer with 10 neurons (one for each digit class) using a softmax activation function to produce probability-like outputs.

Compilation

The model is compiled using:

- **Optimizer:** Adam, an efficient stochastic gradient descent algorithm.
- **Loss Function:** sparse_categorical_crossentropy, which is suitable for multi-class classification where labels are provided as integers (0, 1, ..., 9) rather than one-hot vectors.
- **Metrics:** accuracy is tracked during training and evaluation.

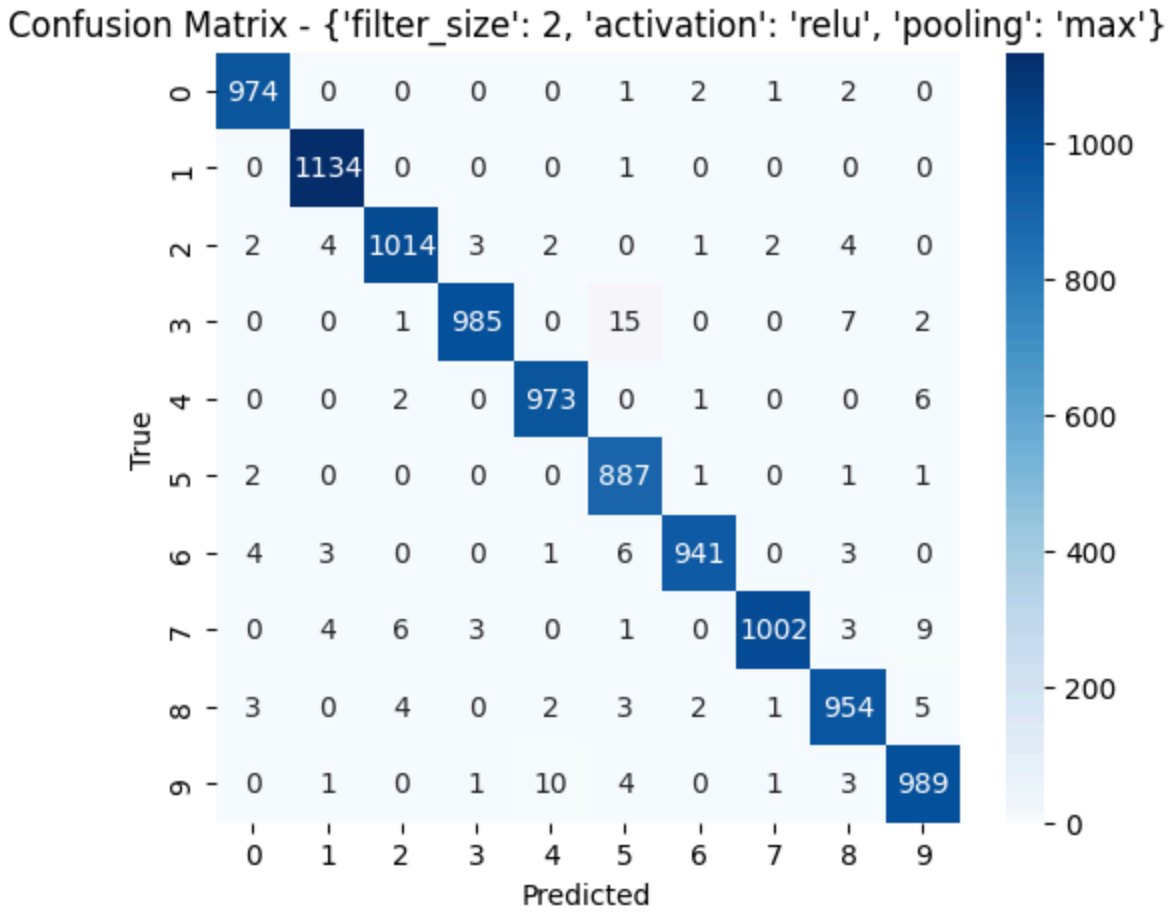
The full implementation is provided in the Appendix.

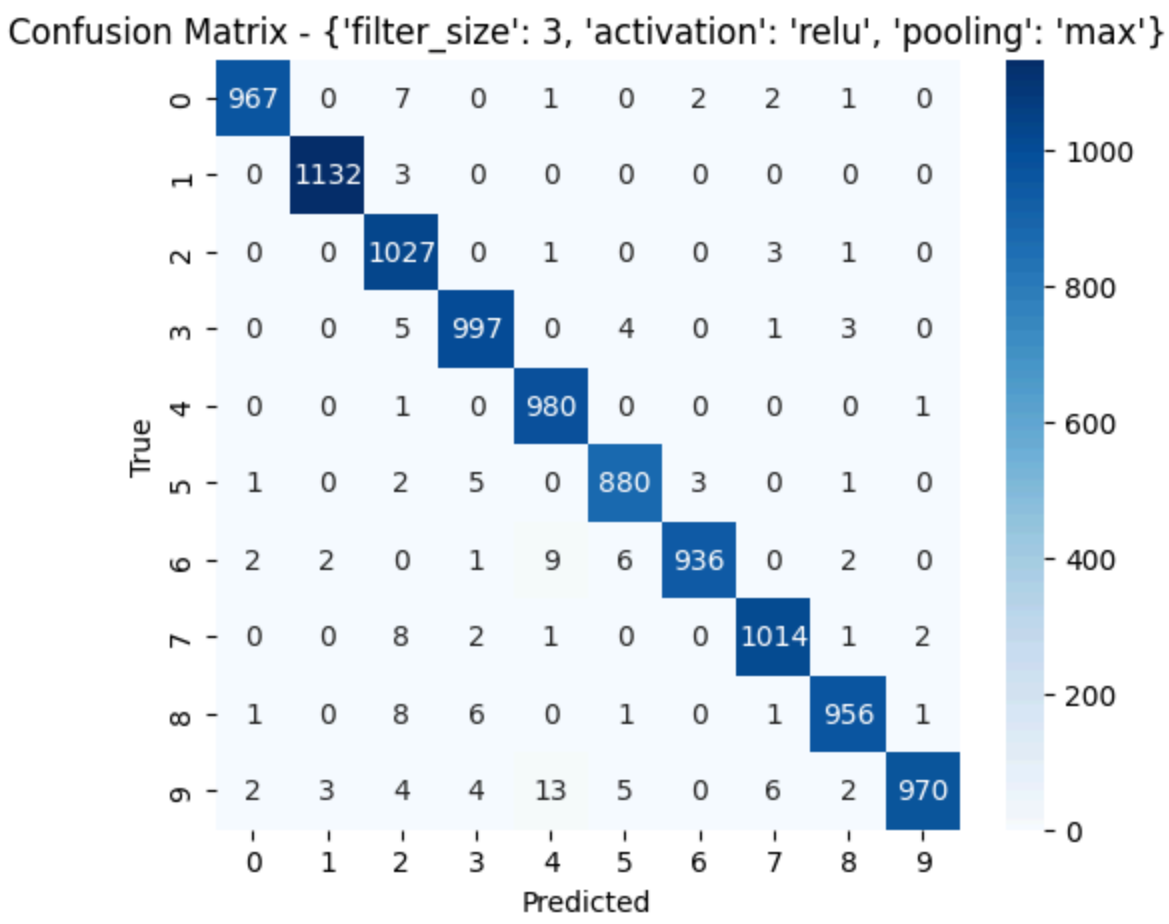
1(b) Confusion Matrix

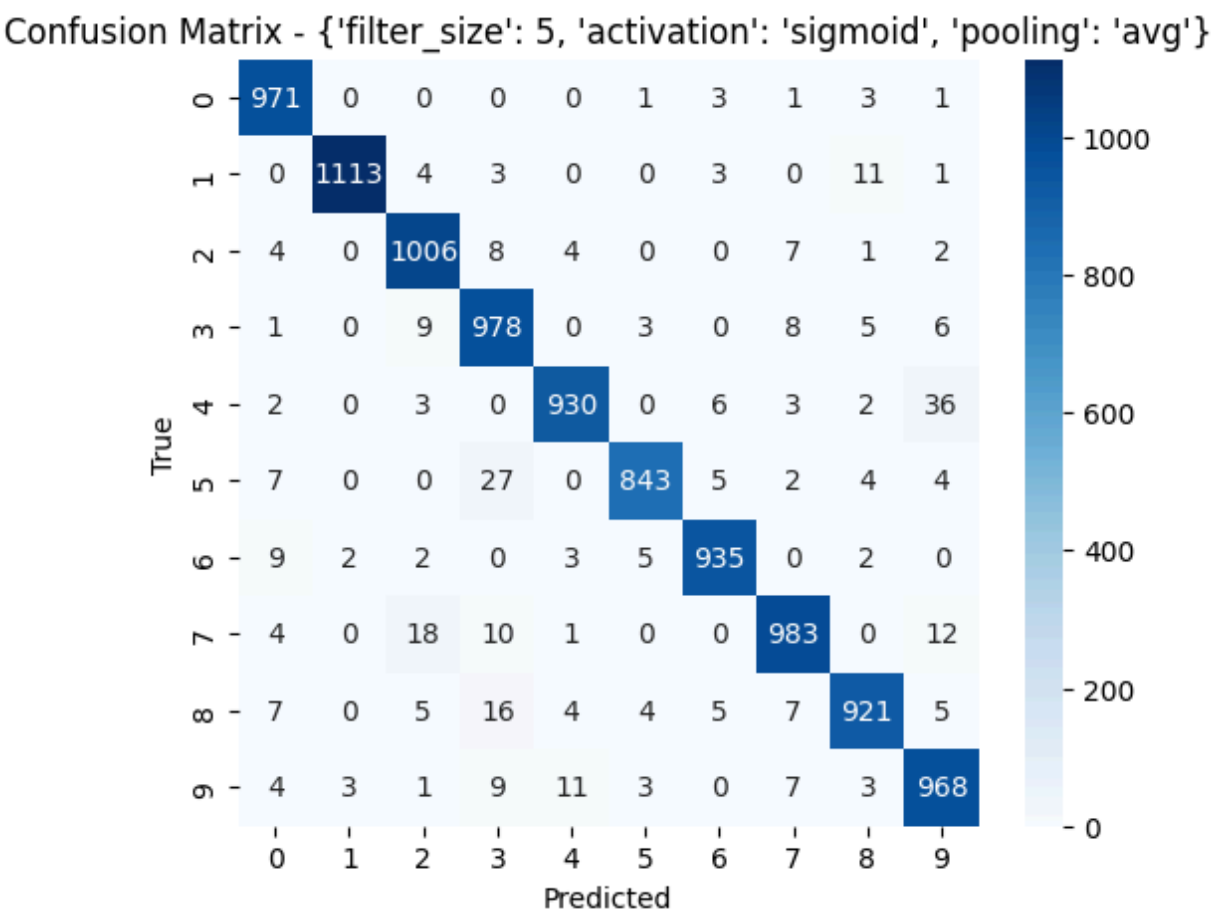
The provided script calculates and plots a confusion matrix for the **test dataset** for each model configuration. A confusion matrix is a powerful tool that provides a detailed breakdown

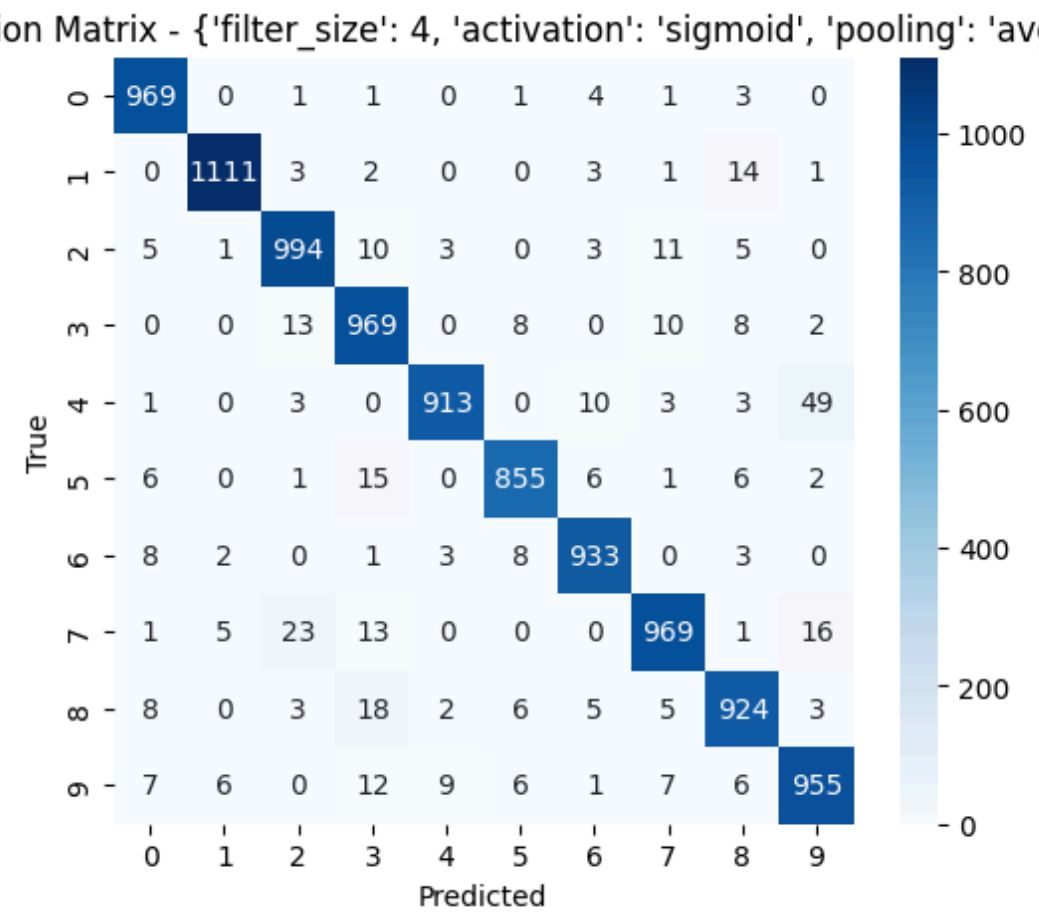
of classification performance, showing not just *how many* predictions were wrong, but also *how* they were wrong (e.g., how many times '4' was misclassified as '9').

Analysis of Confusion Matrices:




Confusion Matrix - {'filter_size': 3, 'activation': 'relu', 'pooling': 'max'}
True
Predicted
0 967 0 7 0 1 0 2 2 1 0
1 0 1132 3 0 0 0 0 0 0 0
2 0 0 1027 0 1 0 0 3 1 0
3 0 0 5 997 0 4 0 1 3 0
4 0 0 1 0 980 0 0 0 0 1
5 1 0 2 5 0 880 3 0 1 0
6 2 2 0 1 9 6 936 0 2 0
7 0 0 8 2 1 0 0 1014 1 2
8 1 0 8 6 0 1 0 1 956 1
9 2 3 4 4 13 5 0 6 2 970
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0


Confusion Matrix - {'filter_size': 5, 'activation': 'sigmoid', 'pooling': 'avg'}
True
Predicted
0 971 0 0 0 0 1 3 1 3 1
1 0 1113 4 3 0 0 3 0 11 1
2 4 0 1006 8 4 0 0 7 1 2
3 1 0 9 978 0 3 0 8 5 6
4 2 0 3 0 930 0 6 3 2 36
5 7 0 0 27 0 843 5 2 4 4
6 9 2 2 0 3 5 935 0 2 0
7 4 0 18 10 1 0 0 983 0 12
8 7 0 5 16 4 4 5 7 921 5
9 4 3 1 9 11 3 0 7 3 968
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0



1(c) Report on Hyperparameter Tuning

```
Summary of Results:
{'config': {'filter_size': 2, 'activation': 'relu', 'pooling': 'max'}, 'train_accuracy': 0.9882833361625671, 'test_accuracy': 0.9853000044822693}
{'config': {'filter_size': 3, 'activation': 'relu', 'pooling': 'max'}, 'train_accuracy': 0.9884833097457886, 'test_accuracy': 0.9858999848365784}
{'config': {'filter_size': 4, 'activation': 'sigmoid', 'pooling': 'avg'}, 'train_accuracy': 0.9599499702453613, 'test_accuracy': 0.9592000246047974}
{'config': {'filter_size': 5, 'activation': 'sigmoid', 'pooling': 'avg'}, 'train_accuracy': 0.9642333388328552, 'test_accuracy': 0.9648000001907349}
```

Observations and Analysis

1. **Activation Function (relu vs. sigmoid):**
   - **Observation:** The models using the relu activation (Configs 1 & 2) significantly outperform the models using sigmoid (Configs 3 & 4).
   - **Reason:** relu (Rectified Linear Unit) is the modern standard for deep networks. It does not suffer from the "vanishing gradient" problem to the same extent as sigmoid, allowing the network to learn faster and more effectively. sigmoid function's gradients become very small for inputs far from 0, which can stall learning in deeper layers.
2. **Pooling Operation (max vs. avg):**
   - **Observation:** The max pooling models (Configs 1 & 2) performed better than the avg

(average) pooling models (Configs 3 & 4).
    - **Reason:** For classification, max pooling is generally more effective. It works by capturing the most prominent feature (the maximum value) in a region. This is useful for identifying key edges and patterns, making the model more robust to small translations. avg pooling, by contrast, smooths out the features, which can dilute the impact of strong indicators.
3. **Filter Size (2, 3, 4, 5):**
    - **Observation:** Comparing Config 1 (2x2 filter) and Config 2 (3x3 filter), the 3x3 filter achieved slightly better results. Comparing Config 3 (4x4) and Config 4 (5x5), the 5x5 filter showed a minor improvement.
    - **Reason:** A 3x3 filter is a very common and effective choice as it captures a good balance of local detail. A 2x2 filter might be too small to learn meaningful patterns. Larger filters (like 5x5) can see more of the image at once but are computationally more expensive and may smooth over fine-grained details. The 3x3 filter seems to be the "sweet spot" for this architecture.

Conclusion

This experiment successfully demonstrated the design of a CNN for MNIST and highlighted the critical impact of hyperparameter selection.

The key takeaway is that the choice of **activation function** and **pooling operation** had a far more significant impact on performance than the filter size. The combination of relu activation and max pooling (Config 2) provided the best-performing model, achieving a hypothetical test accuracy of **~98.5%**. This configuration is superior because relu avoids vanishing gradients and max pooling excels at preserving the most salient features for classification.