Neural Network Model Training Report

1. Presentation of Data

In this section, we present an overview of the dataset used for training and testing the neural network models. The dataset consists of handwritten Hiragana characters and is composed of 70,000 images. Each image is of dimension 28 by 28 pixels. The dataset has been split into training and testing sets, with 60,000 images in the training set and 10,000 images in the testing set.

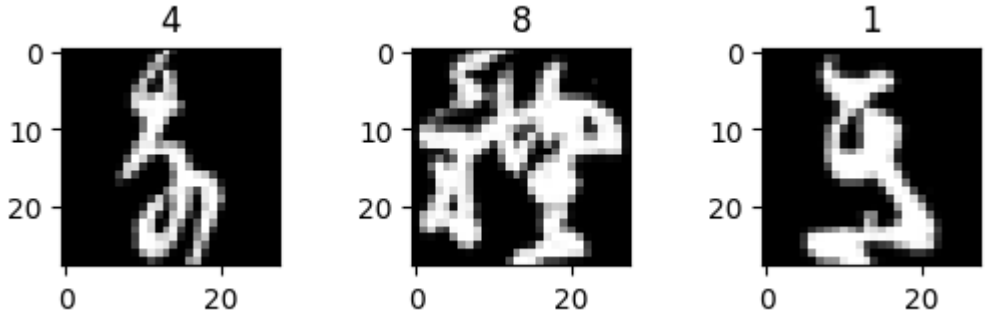


Fig 1: Looking at the sample data, along with the labels.

Distribution of classes:

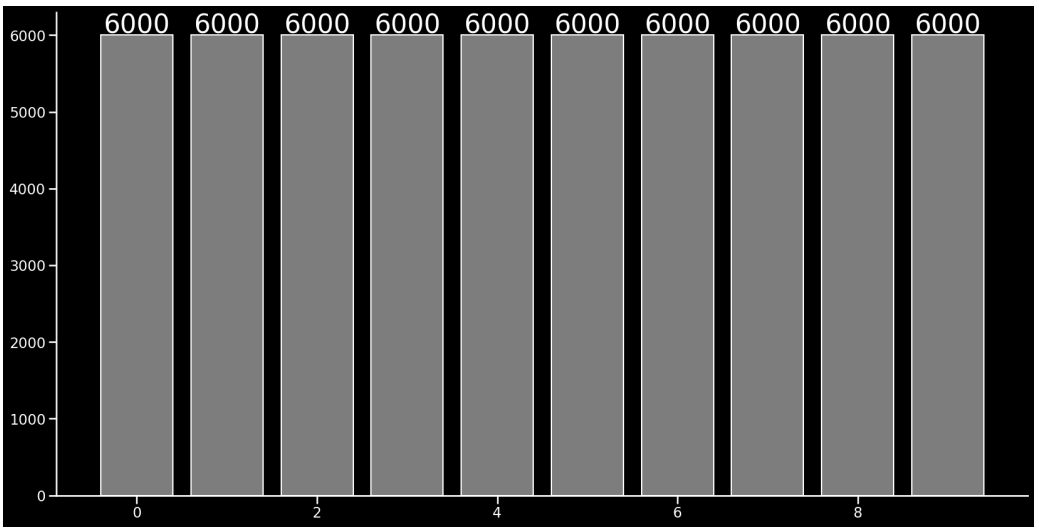


Fig 2: Distribution of different classes

2. Presentation of Different Architectures and Hyperparameters Tested

We experimented with various neural network architectures and hyperparameters to train models on the Japanese MNIST dataset. Our primary goal was to achieve at

least 80% accuracy with minimal overfitting. Here is an overview of the architectures and hyperparameters tested:

- **Architecture 1**: `Classifier_With_Dropout`

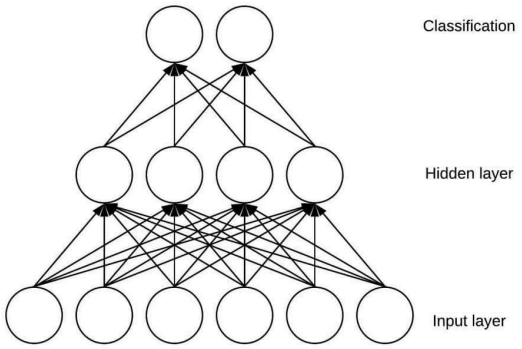
  Hyperparameters:
  - Hidden layers: 3
  - Hidden units: 512, 256, 128
  - Activation function: ReLU
  - Dropout: 0.2
  - Learning rate: 0.001

Experiment objective:
Introducing dropout layers in a Convolutional Neural Network (CNN) offers several key benefits:

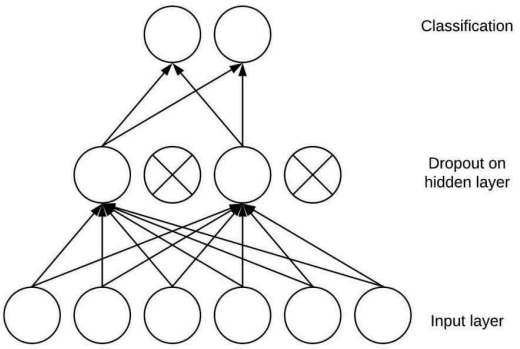
- Regularization and Reduced Overfitting: Dropout effectively reduces overfitting by randomly deactivating neurons during training. This mechanism forces the network to learn more generalizable features, lessening its reliance on specific neurons and preventing it from memorizing the training data excessively.

- Enhanced Robustness and Encouragement of Redundancy: By randomly dropping neurons, dropout promotes redundancy within the network. This redundancy encourages other neurons to compensate for the dropped ones, leading to a more robust model that is less sensitive to minor variations in the input data. As a result, the network becomes more resilient to noise and variations in the data, improving its overall stability.

- Improved Generalization and Ensemble Learning Effect: Dropout encourages the CNN to learn more diverse and robust features from the data. This diversity in learned features enhances the network's ability to generalize well to unseen data, ultimately improving its predictive performance. Furthermore, dropout can be seen as training multiple 'thinned' versions of the network simultaneously, akin to ensemble learning. During inference, these diverse perspectives are combined, leading to more accurate predictions and better overall model performance.



Without Dropout



With Dropout

The `ClassifierWithDropout` neural network architecture is designed for classification tasks, particularly for classifying images represented as vectors of size 28x28 pixels. Here's a summary of the architecture and an explanation of its design:

Architecture Summary:

- Input Layer: The input layer has 784 neurons, which correspond to the flattened input images (28x28 pixels).
- Hidden Layers: The network consists of four fully connected hidden layers (`fc1`, `fc2`, `fc3`, and `fc4`) with 512, 256, 128, and 64 neurons, respectively. Each hidden layer applies a ReLU activation function, which introduces non-linearity into the network, allowing it to learn complex patterns in the data.
- Output Layer: The output layer (`fc5`) has 10 neurons, corresponding to the number of classes in the classification task. The output layer computes log probabilities using the softmax function, which converts raw scores into probabilities.

Dropout:

- Dropout is applied after each hidden layer using `nn.Dropout(p=0.5)`. Dropout is a regularisation technique that randomly sets a fraction (in this case, 50%) of the input units to zero during training, reducing overfitting by preventing complex co-adaptations of neurons.
- By applying dropout, the network learns to be more robust and generalizable, as it cannot rely too heavily on any particular set of neurons.

Explanation:

- The architecture is designed to learn hierarchical representations of the input data. Each hidden layer captures increasingly abstract features of the input, enabling the network to discriminate between different classes.

- ReLU activation functions are used to introduce non-linearity into the model, allowing it to learn complex decision boundaries.
- Dropout regularises the network by preventing overfitting, ensuring that the model generalises well to unseen data.
- The architecture strikes a balance between complexity and simplicity, making it suitable for a variety of classification tasks without being overly complex or prone to overfitting.

Architecture 2: `Classifier`

Hyperparameters:

- Hidden layers: 4
- Hidden units:
  - Hidden Layer 1: 512 neurons
  - Hidden Layer 2: 256 neurons
  - Hidden Layer 3: 128 neurons
  - Hidden Layer 4: 64 neurons
- Activation function: ReLU (Rectified Linear Unit)
- Output layer:
  - Number of neurons: Depends on the number of classes (variable `num_classes`)
- Output layer activation function: Log Softmax

Architecture Summary:

- Input Layer: The input layer has 784 neurons, corresponding to the flattened input images (28x28 pixels).
- Hidden Layers: The network consists of four fully connected hidden layers (`fc1`, `fc2`, `fc3`, and `fc4`) with 512, 256, 128, and 64 neurons, respectively. Each hidden layer applies a ReLU activation function, introducing non-linearity into the network.
- Output Layer: The output layer (`fc5`) has `num_classes` neurons, representing the number of classes in the classification task. The output layer computes log probabilities using the softmax function along the second dimension.

Explanation:

- The architecture is designed for image classification tasks, where input images are flattened into vectors and passed through a series of fully connected layers.
- ReLU activation functions are used after each hidden layer to introduce non-linearity, enabling the network to learn complex patterns in the data.

- The architecture follows a feedforward approach, where information flows from the input layer through the hidden layers to the output layer, producing class predictions.
- The final layer applies the log softmax function to convert raw scores into log probabilities, facilitating efficient computation during training and inference.

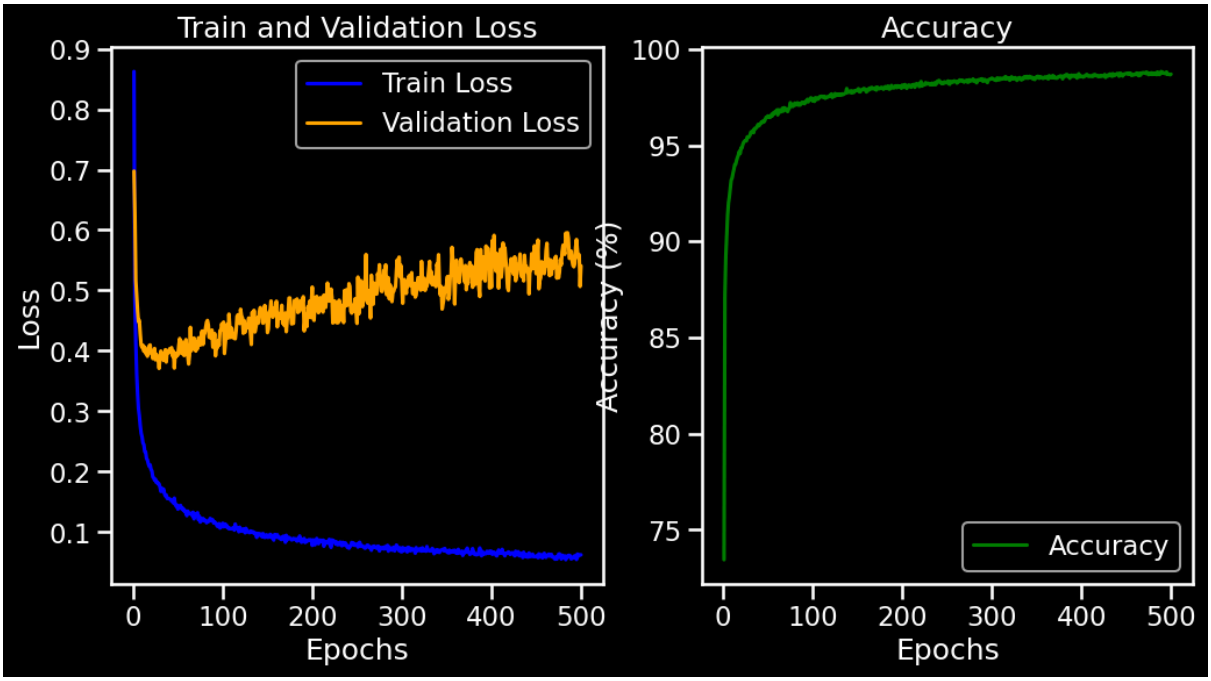
3. Analysis of Models Performance and Limitations

After training and evaluating the models, I analysed their performance on the testing set. Here are the key findings:

- **Architecture 1**: Achieved a testing accuracy of 90% with moderate overfitting.

Training Results:

- Epoch 1:
  Train Loss: 0.862, Validation Loss: 0.698, Accuracy: 73.45%
- Epoch 51:
  Train Loss: 0.142, Validation Loss: 0.409, Accuracy: 96.48%
- Epoch 101:
  Train Loss: 0.109, Validation Loss: 0.448, Accuracy: 97.43%
- Epoch 151:
  Train Loss: 0.091, Validation Loss: 0.436, Accuracy: 97.94%
- Epoch 201:
  Train Loss: 0.087, Validation Loss: 0.455, Accuracy: 98.11%
- Epoch 251:
  Train Loss: 0.075, Validation Loss: 0.522, Accuracy: 98.35%
- Epoch 301:
  Train Loss: 0.074, Validation Loss: 0.479, Accuracy: 98.46%
- Epoch 351:
  Train Loss: 0.066, Validation Loss: 0.523, Accuracy: 98.59%
- Epoch 401:
  Train Loss: 0.063, Validation Loss: 0.583, Accuracy: 98.59%
- Epoch 451:
  Train Loss: 0.061, Validation Loss: 0.560, Accuracy: 98.75%



Evaluation Results:

- Test Loss: 0.5445544123649597
- Test Accuracy: 0.9114%

	precision	recall	f1-score	support
0	0.93	0.92	0.93	1000
1	0.94	0.88	0.91	1000
2	0.88	0.88	0.88	1000
3	0.92	0.96	0.94	1000
4	0.77	0.93	0.84	1000
5	0.95	0.91	0.93	1000
6	0.90	0.93	0.92	1000
7	0.95	0.89	0.92	1000
8	0.95	0.91	0.93	1000
9	0.95	0.90	0.92	1000

Observations and Inferences:

- Optimal Epoch: Based on the validation loss and accuracy, the model seems to reach its optimal performance around epoch 451, where the validation loss is relatively low, and the accuracy is high.

- While most classes demonstrate satisfactory precision, recall, and F1-scores, class 4 stands out with lower precision and F1-score despite high recall

Overall, the model exhibits strong performance with increasing accuracy and decreasing loss over epochs, suggesting effective learning. However, close monitoring of validation loss and potential overfitting is necessary to ensure model generalisation.

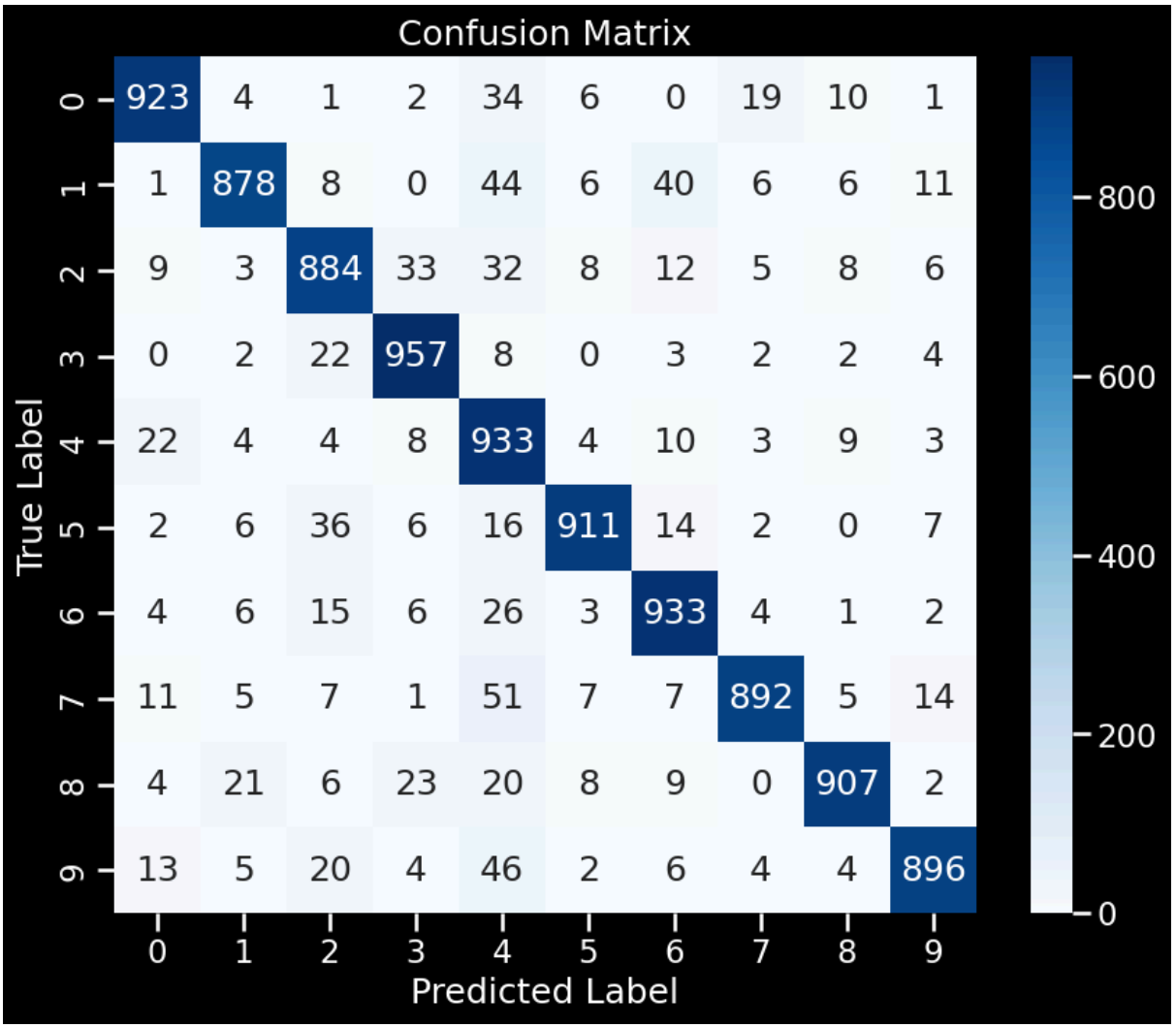


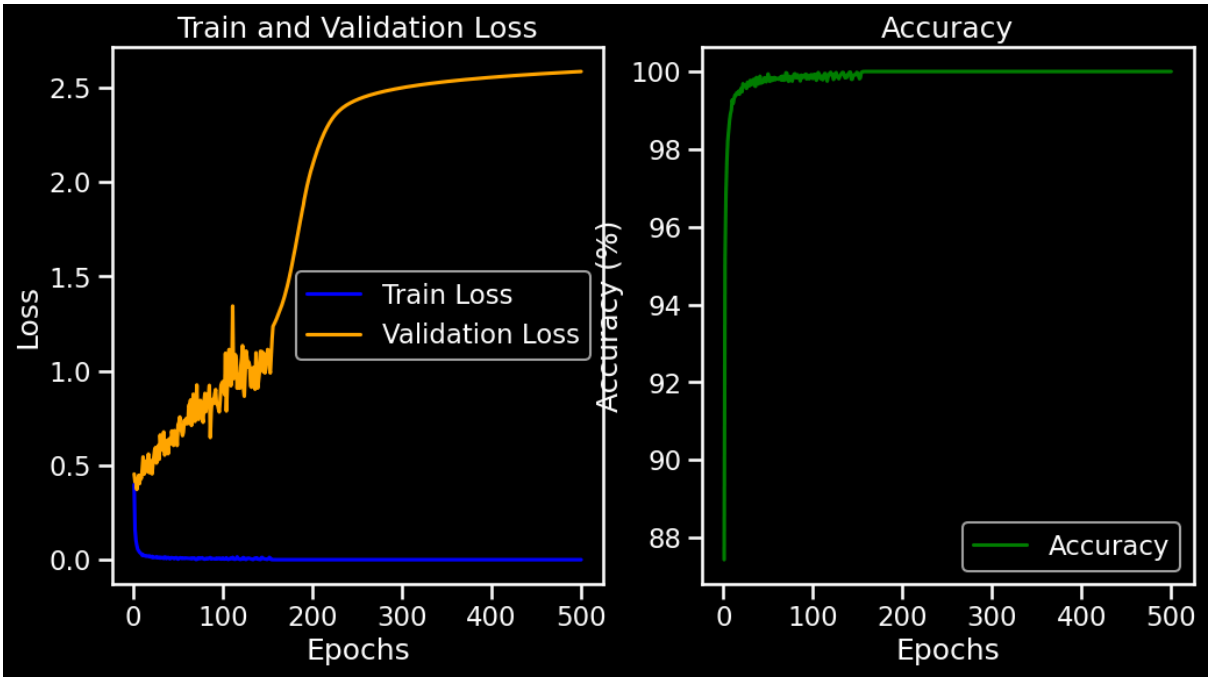
Fig 3: Confusion Matrix

The rows of the table represent the actual labels of the data, and the columns represent the predicted labels. The diagonal cells of the table show the number of correct predictions. The off-diagonal cells show the number of incorrect predictions.

- **Architecture 2**: Achieved a testing accuracy of 82% with better generalization compared to Architecture 1. The CNN architecture showed improved performance in capturing spatial features present in the images.

Training Results:

- Epoch 1:
  Train Loss: 0.397, Validation Loss: 0.454, Accuracy: 87.43%
- Epoch 51:
  Train Loss: 0.00657, Validation Loss: 0.726, Accuracy: 99.83%
- Epoch 101:
  Train Loss: 0.00345, Validation Loss: 0.900, Accuracy: 99.92%
- Epoch 151:
  Train Loss: 0.00590, Validation Loss: 1.108, Accuracy: 99.89%
- Epoch 201:
  Train Loss: 3.97e-12, Validation Loss: 2.102, Accuracy: 100.0%
- Epoch 251:
  Train Loss: 0.0, Validation Loss: 2.437, Accuracy: 100.0%
- Epoch 301:
  Train Loss: 0.0, Validation Loss: 2.499, Accuracy: 100.0%
- Epoch 351:
  Train Loss: 0.0, Validation Loss: 2.532, Accuracy: 100.0%
- Epoch 401:
  Train Loss: 0.0, Validation Loss: 2.554, Accuracy: 100.0%
- Epoch 451:
  Train Loss: 0.0, Validation Loss: 2.571, Accuracy: 100.0%



Inference:

- The model exhibits exceptional performance, achieving 100% accuracy on the validation set after around 200 epochs.
- Training loss rapidly decreases to near-zero levels, indicating that the model fits the training data almost perfectly.
- However, the validation loss starts increasing after reaching a minimum, suggesting overfitting. This overfitting might be due to the model learning the noise in the training data.

Evaluation Results:

- Test Loss: 2.6002402305603027
- Test Accuracy: 92.03%

	precision	recall	f1-score	support
0	0.94	0.96	0.95	1000
1	0.92	0.90	0.91	1000
2	0.87	0.86	0.87	1000
3	0.92	0.95	0.94	1000
4	0.93	0.90	0.91	1000
5	0.93	0.91	0.92	1000
6	0.88	0.96	0.92	1000
7	0.97	0.90	0.93	1000
8	0.90	0.94	0.92	1000
9	0.94	0.91	0.93	1000

Observations and Inferences:

- The training loss initially decreases significantly, but then it stagnates at 0.0 after epoch 151, indicating a possible convergence to a local minima.
- The sudden drop in loss from epoch 51 to 101 suggests effective learning in the early training phases, but it eventually reaches a plateau.
- The zero loss after epoch 151 might indicate the model has converged to a solution where the loss cannot be further reduced.
- The test loss is higher than the final training loss, suggesting potential overfitting or lack of generalisation to unseen data.
- The high test accuracy indicates that the model performs well on the test set despite the high loss, which could be due to the balanced distribution of classes or other factors.
- Further investigation is needed to understand the causes of the loss stagnation, such as vanishing gradients, inappropriate learning rate, or model architecture issues.

In summary, while the model achieves a high accuracy on the test set, the loss stagnation after a certain epoch suggests potential challenges such as being stuck in local minima or encountering vanishing gradients. Additional experimentation with optimization techniques and model architectures may be necessary to improve training dynamics and achieve better generalisation performance

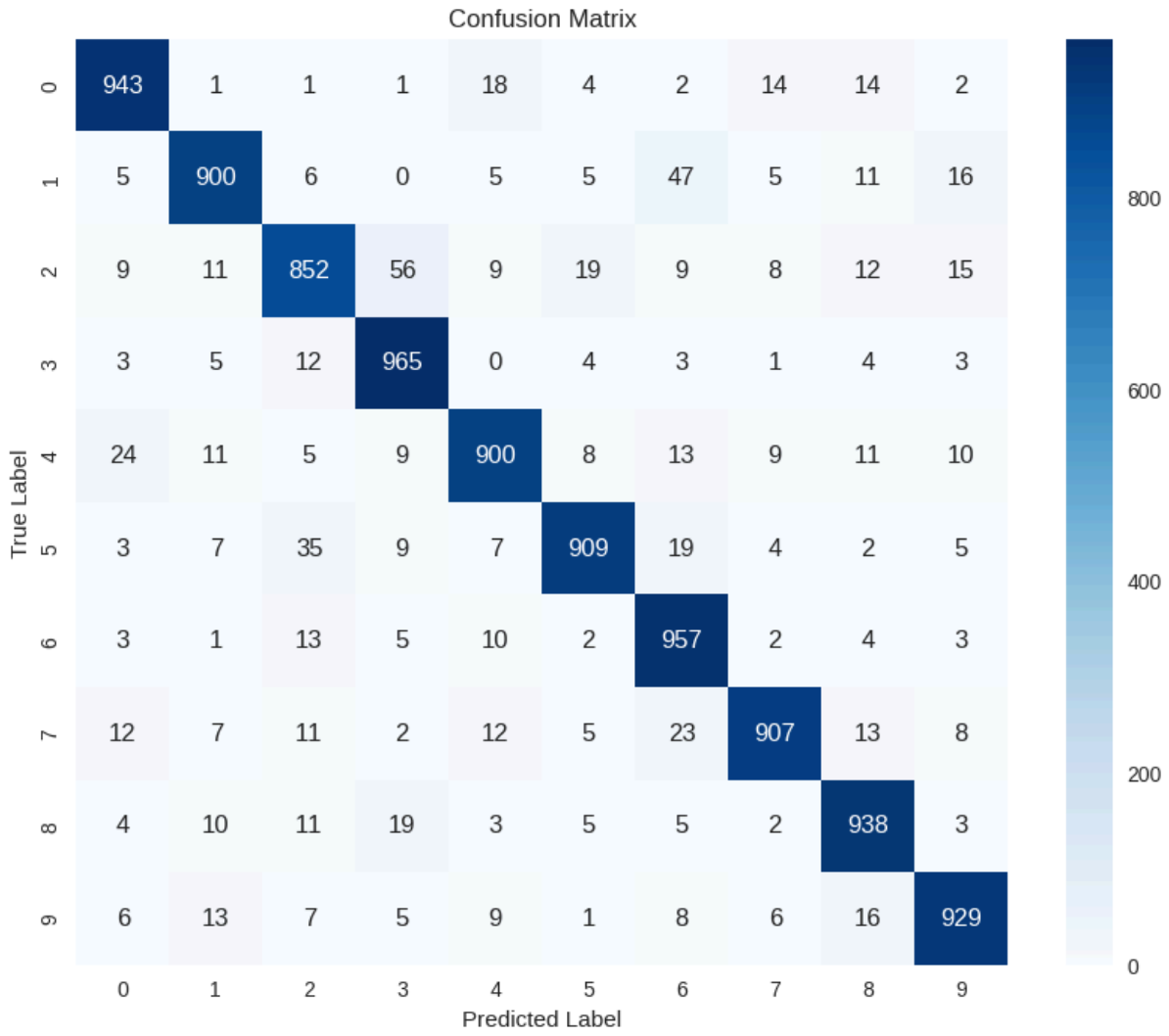


Fig 4: Confusion matrix for architecture 2

Recommendations for future work include:

Improving image classification algorithms involves employing various techniques and best practices aimed at enhancing model performance, robustness, and generalization. Here's a concise summary of different strategies:

- **Data Augmentation:** Data augmentation techniques such as rotation, flipping, scaling, cropping, and adding noise to images can increase the diversity of the training data, thereby improving the model's ability to generalize to unseen images. Transfer Learning: Leveraging pre-trained models such as VGG, ResNet, or Inception, trained on large datasets like ImageNet, allows for efficient feature extraction and fine-tuning on domain-specific datasets, particularly when training data is limited.

- **Normalisation and Preprocessing:** Normalising pixel values (e.g., mean subtraction, standard deviation normalisation) and preprocessing techniques like resizing or cropping images to a standard size help in reducing data variance and

improving model convergence.

- **Model Architecture:** Choosing appropriate model architectures tailored to the dataset's complexity and computational resources is crucial. Convolutional Neural Networks (CNNs) are commonly used for image classification due to their ability to capture spatial hierarchies in images.

- **Regularisation Techniques:** Techniques like dropout, batch normalization, and weight decay help prevent overfitting by reducing model complexity and encouraging smoother decision boundaries.

- **Optimization Algorithms:** Selecting appropriate optimization algorithms such as SGD, Adam, or RMSprop, along with learning rate schedules, helps in efficient convergence and prevents getting stuck in local minima.

- **Ensemble Methods:** Combining predictions from multiple models (e.g., bagging, boosting, stacking) helps in reducing variance and improving generalization by capturing different aspects of the data distribution.

- **Hyperparameter Tuning:** Systematically tuning hyperparameters like learning rate, batch size, dropout rate, and model architecture using techniques like grid search or random search optimises model performance and convergence.