MACHINE LEARNING REPORT

TASK ONE

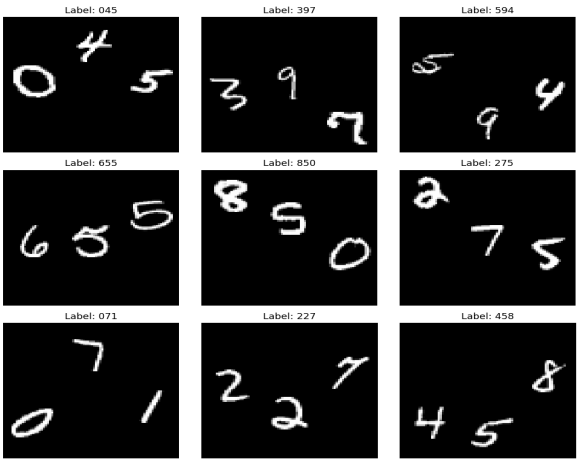
Visualizing the dataset

1. **Dataset Overview**

- I was able to do my visualisation of the dataset to have a look, so first things first I realised that the dataset had been already split between training, validation and test data.
- The Triple-MNIST contains images of size 84x84 pixels, where each picture contains 3 digits. The goal being to identify each of the three digits present in the image. The data set has 64,000 training images, 16,000 validation images and 20,000 test images. I also confirmed that the size is indeed 84x84 and the images is actually a grayscale.

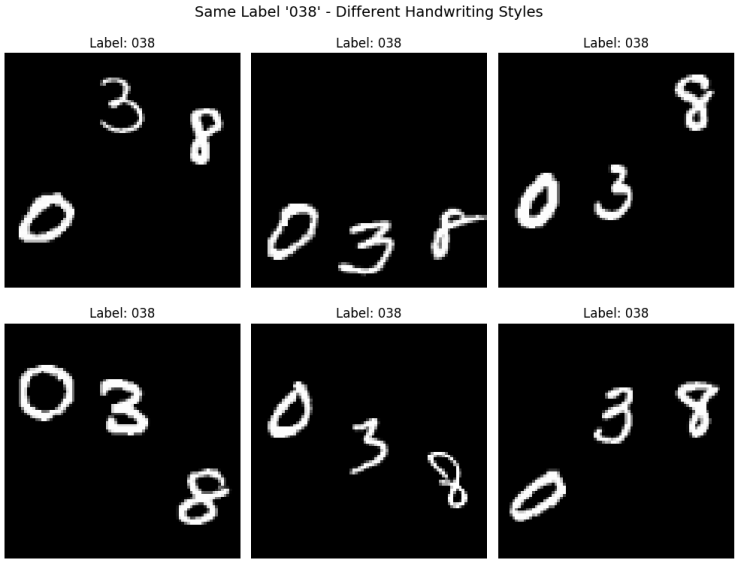
2. **Visual Exploration**

- I was aware of the basic MNIST dataset which contains 28x28 grayscale images of handwritten digits (0 through 9) with 60,000 training images and 10,000 test images.
- A 3x3 grid of random training samples was visualised to understand the structure of the images. Each 84 x 84 image contains three handwritten digits positioned at varying locations within the frame.



- I was also able to display 6 different images all with the same label **'038'.** Despite sharing the same label, each image shows distinct

handwriting styles and digit positions. This demonstrates the variation within each class that the model must learn to recognize it cannot simply memorize one pattern per label but must learn the general characteristics of each digit.



Comparison with Standard MNIST

- The Triple-MNIST problem differs from a standard classification in several ways:
    - **Multi-label output**: Predicting 3 digits instead of 1
    - **Larger output space**: 1000 possible classes vs 10 in the standard MNIST dataset.
    - **Data sparsity**: With a kind of limited training data spread across more classes, each class has fewer example to learn form.
    - **Spatial complexity**: The digits seem to be appearing at random rather than the expected center position for the standard MNIST.

Label Distribution Analysis

- I also did a label distribution analysis; it revealed that it contains 640 unique label combinations out of a possible 1000(000-999). Each label has exactly 100 training examples, resulting in a balanced dataset.
- This absence of **360 labels** from the training set suggests the test set may contain unseen combinations, requiring the model to generalize its understanding of individual digits rather than memorize specific 3-digit patterns.

- I further analysed and it revealed zero overlap between training and test labels. The training set contains 640 unique 3-digit combinations, while the test set contains 200 entirely different combinations. This design rigorously tests the model's ability to generalize it cannot rely on memorizing specific label patterns but must learn to recognize individual digits and combine that knowledge for unseen combinations.

TASK TWO:

LOGISTIC REGRESSION

2.1 Data Preprocessing

- To prepare the data for Logistic Regression model, each 84x84 grayscale image was flattened into a single 1D Vector of 7056 values (84 x 84 = 7056).
- The pixel values were normalized from 0-255 to 0-1 by dividing by 255. This normalization helps the model train more efficiently.
- After preprocessing, the training set shape was (64000, 7056), meaning 64,000 samples with 7,056 features each.

2.2 Logistic Regression Baseline

- A logistic Regression classifier was trained on the pre-processed training data (64,000 samples across 640 unique labels combinations). The model was configured with **max_iter=1000** to ensure convergence. Results:

  Training accuracy – 100%
  Validation accuracy – 0%

2.3 Analysis of Results

- The stark contrast between the training and validation accuracy reveals a fundamental limitation of this approach. Investigation shows that the validation set contains 160 unique labels, none of

which appear in the training set. This means the model is capable of scoring high training accuracy on the training data because it can memorize the pattern but it has zero generalisation capability.

- This occurs because logistic regression treats each combination as its own single discrete class meaning it sees the three digit combination as its own separate, indivisible entity.
- The model can only predict labels it has seen during training when presented with data it has never seen it can't recognise it.
- These finding highlights that treating the Triple-MNIST as a standard multi-class classification problem is fundamentally flawed, because the model learns to recognise specific combination rather than the underlying digit patterns.

2.4 CNN

Data preparation for CNN

- Unlike Logistic Regression which requires flattened 1D input (84×84 → 7,056 features), CNNs work with the original 2D image structure. This is important because CNNs can detect spatial patterns like edges, shapes, and digit features that get lost when flattening. **Steps we took:**
    1. Loaded images as 2D arrays - shape (64000, 84, 84)
    2. Adding channel dimensions - reshaped to (64000, 84, 84, 1) where 1 = grayscale
    3. Normalized pixel values – scaled from 0 – 255 to 0 – 1
    4. Label Encoding – converted string labels ("797", "038") to integers (0-639) because neural networks require numerical outputs.

Model Architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 82, 82, 32)	320
max_pooling2d (MaxPooling2D)	(None, 41, 41, 32)	0
conv2d_1 (Conv2D)	(None, 39, 39, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 19, 19, 64)	0
conv2d_2 (Conv2D)	(None, 17, 17, 64)	36,928
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 128)	2,367,616
dense_1 (Dense)	(None, 640)	82,560

Total params: 2,505,920

Training Configuration

- Optimiser: **Adam**
- Loss function: **Sparse Categorical Cross-entropy**
- Epochs: **5**
- Batch size: **32**

Results

- Accuracy
  - Training – 87.88%
  - Validation – 0%

Analysis

- The CNN achieved 87.88% accuracy on training data, demonstrating it successfully learned to recognise the 640-digit combinations it was trained on. However, validation accuracy was 0% because the validation set contains 160 completely different labels with zero overlap with training labels. The model can only predict labels it has seen during training.

Hyper parameter fine tuning

- **For Logistic Regression**, the primary hyperparameter tuned was the **regularisation strength (C)**. The C parameter controls the trade-off between fitting the training data and keeping the model simple:

    - **Lower C values** (e.g., 0.1): Stronger regularisation, simpler model, may underfit
    - **Higher C values** (e.g., 10.0): Weaker regularisation, more complex model, may overfit

Results

C value	Regularisation	Training Accuracy	Validation Accuracy
0.1	**Strong**	**91.64%**	**0.00%**
1.0	**Default**	**100.00%**	**0.00%**
10.0	**Weak**	**100.00%**	**0.00%**

- **For the CNN**, the hyper parameter tuned was the epochs and the learning rate:
    - **Epochs**: The number of times the model sees the entire training dataset. More epochs allow the model more opportunities to learn patterns.
    - **Learning rate:** Controls how much the model adjusts its weights after each batch. Lower values mean slower, more gradual learning.

Results

Epochs	Learning rate	Training Accuracy	Validation Accuracy
5	**0.001**	**87.17%**	**0.00%**
10	**0.001**	**88.35%**	**0.00%**
5	**0.0001 (slower)**	**78.99%**	**0.00%**

- Increasing epochs to 10 improved training accuracy as the model had more iterations to learn. The slower learning rate (0.0001)

resulted in lower accuracy because the model required more epochs to converge it was still improving when training stopped at epoch 5.

Analysis

- The goal of hyper parameter tuning is to improve performance on unseen data (validation set). For both models, tuning was unsuccessful in achieving this goal - validation accuracy remained at 0% across all configurations. The improvements in training accuracy (Logistic Regression reaching 100%, CNN improving to 88.35%) only indicate the models became better at memorising the training data, not at generalising. This confirms that hyperparameter tuning cannot overcome the fundamental limitation:
    - zero label overlap between training and validation sets. The problem requires a fundamentally different approach, explored in later tasks.

Test Set evaluation

- Both baseline models were evaluated on the test set using accuracy and **F1 score**. **Results:**

Model	Accuracy	F1 score
Logistic Regression	**0.00%**	**0.0000**
CNN	**0.00%**	**0.0000**

- Both models achieved **0% accuracy** and an F1 score of **0** on the test set, as neither model can predict labels that were not in the training data.

Confusion matrix

- A confusion matrix shows how predictions line up with actual labels correct predictions appear on the diagonal, errors appear off it. For this dataset though, generating one wasn't practical. From the analysis:
    - Logistic Regression predicted all 640 training labels across the test set.
    - CNN predicted 546 of its 640 training labels
    - Neither model predicted any of the 200 actual labels.
- A full confusion matrix would be a 200×640 grid, and since there's no overlap between predicted and actual labels, every single entry on the diagonal would be zero. Every prediction is a miss. The takeaway here isn't

complex: both models can only output labels they were trained on, and none of those labels exist in the test set.

Model Comparison

- On the test set, both models achieved identical performance: 0% accuracy and 0 F1 score. At first glance, this suggests neither model is better than the other. However, their training behaviour tells a different story.

Metric	Logistic Regression	CNN
Test Accuracy	**0.00%**	**0.00%**
Test F1 Score	**0.00**	**0.00**
Training Accuracy	**100.00%**	**87.88%**
Validation Accuracy	**0.00%**	**0.00%**

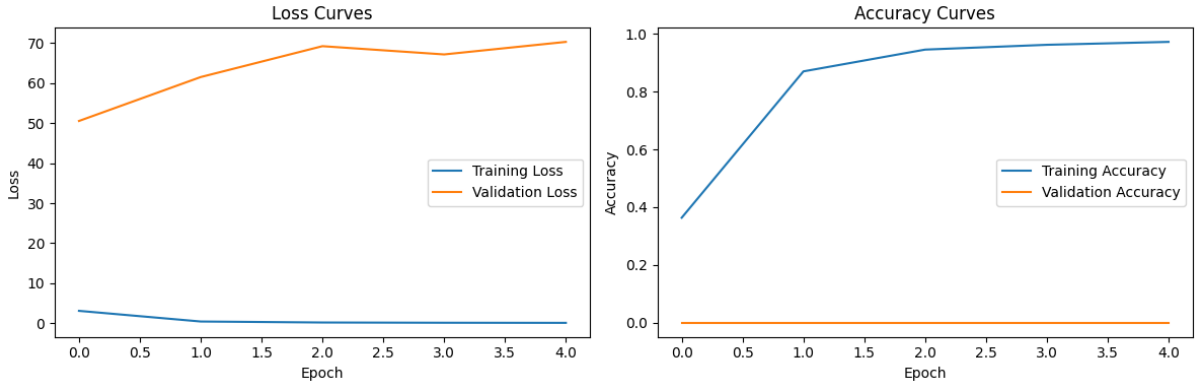
- Logistic Regression achieved 100% training accuracy, meaning it memorised every single training example. The CNN only reached 87.88%, so it didn't fully memorise the training data. Despite this difference, both models scored 0% on validation and test sets because neither can output labels they haven't seen before. In terms of what each model actually learned, the CNN has an advantage. Its convolutional layers extract spatial features like edges, curves, and digit shapes. These features would transfer to new digit combinations in a normal classification task. The CNN "sees" the digits, it just can't produce a label outside its training vocabulary. Logistic Regression has no such feature learning. It maps raw pixel values directly to 640 fixed classes. There's no understanding of what a digit looks like, just a memorised mapping from pixels to labels. If this were a standard task with overlapping labels between train and test, the CNN would likely outperform Logistic Regression. But under this zero-overlap design, that advantage doesn't show up in the results.
- It’s hard to pick an overall winner in this case but based on the design and architecture of the CNN it should have been able to extract spatial features like edges and digit shapes. This ability would typically give it the edge over a logistic regression model.

WHY?

- **Why Both Models Failed** Both models achieved 0% accuracy on the test set, and this was not due to poor implementation. The dataset was

intentionally designed with zero label overlap between training, validation, and test sets. Training contains 640 unique labels, validation has 160, and test has 200. None of these overlap. This creates a fundamental problem: both models can only predict labels they saw during training. Logistic Regression learned to map pixel patterns to 640 specific labels. The CNN funnels all its learning through a Dense (640) layer that outputs probabilities for those same 640 labels. When the test set contains completely different labels, every prediction is guaranteed to be wrong. No amount of tuning or training could fix this. The issue isn't overfitting or underfitting. It's that the models were built to solve a 640-class problem, and the test set asks for answers from a different set of 200 classes. This is why Task 4 exists. The solution is to stop treating the whole image as one classification problem with 640 classes. Instead, split the image into three parts and classify each digit individually as 0-9. That way, the model learns what individual digits look like, and can combine them to form any three-digit label, including ones it never saw during training.

TASK 3



Analysis of Training and Validation curves

- The training and validation loss curves from the baseline CNN were plotted to assess model performance.
- Training metrics (over 5 epochs):
  - Loss: decreased from 3.08 to 0.08
  - Accuracy: increased from 36.36% to 97.18%
- Validation metrics (over 5 epochs):
  - Loss: increased from 50.56 to 70.33
  - Accuracy: stayed at 0% throughout

- This pattern initially suggests severe overfitting. As the model learns the training data (loss decreasing, accuracy increasing), validation performance deteriorates (loss increasing). The training loss approaches zero while validation loss climbs higher, indicating the model is memorizing training patterns rather than learning generalizable features.
- However, the 0% validation accuracy from the very first epoch reveals a more fundamental issue than typical overfitting. With standard overfitting, you would expect some initial validation accuracy that gradually decreases as the model begins to memorize. The flat 0% from epoch 0 indicates the model cannot predict any validation labels correctly not because it's overfitting, but because it architecturally cannot output labels it has never seen.
- The increasing validation loss reflects the model's growing confidence in incorrect predictions. As training progresses, the model becomes more certain about its (wrong) predictions on validation data, leading to higher cross-entropy loss.
- This confirms the zero-overlap problem: the model cannot predict label combinations it was never trained on. This is the core limitation of treating 3-digit sequences as single classes, which will be addressed in Task 4.

Improvement Technique 1: Dropout Regularisation

- Dropout randomly disables neurons during training to stop the model from memorising.
- Dropout layers were added **after each convolutional block (25%) and before the final dense layer (50%).**

Model	Training Accuracy
Base CNN	87.88%
CNN + Dropout	86.95%

- The slight drop in training accuracy shows dropout is doing its job. The model cannot memorise as easily.

Improvement Technique 2: Data Augmentation

- Data augmentation creates variations of training images by rotating, shifting, and zooming. This usually helps models generalise by seeing more variety.

Model	Training Accuracy
Base CNN	87.88%

CNN + Augmentation	0.16%

- The model failed to learn anything. With three digits side by side, even small rotations and shifts distort the image too much. Parts of digits get cut off or blend into each other, making the label unrecognisable.

Test Set Evaluation

Model	Test Accuracy	F1 Score
Base CNN	0.00%	0.00
CNN + Dropout	0.00%	0.00
CNN + Data Augmentation	0.00%	0.00

- Neither technique improved test set performance. This is not due to incorrect implementation, but because the problem cannot be solved through regularisation or augmentation. The training, validation, and test sets contain completely different 3-digit combinations with zero overlap. A model trained on 640 combinations cannot predict 200 combinations it has never seen.

Why Regularisation Techniques Failed

- Neither dropout nor data augmentation improved test performance because they address overfitting, not architectural constraints.

- **Dropout's Limitation:** Dropout successfully reduced overfitting, as shown by the decrease in training accuracy from 87.88% to 86.95%. By randomly disabling neurons during training, dropout prevented the model from co-adapting on specific training patterns. However, this doesn't solve the zero-overlap problem. The Dense(640) output layer can only predict the 640 specific combinations it was trained on. Dropout cannot enable the model to output the 200 test combinations that don't exist in its vocabulary—regularisation reduces confidence in training classes but cannot create new output classes.

- **Data Augmentation's Catastrophic Failure:** Data augmentation failed completely (0.16% accuracy) due to the spatial structure of

Triple-MNIST images. Unlike single-digit MNIST where small rotations and shifts are harmless, Triple-MNIST contains three digits positioned side-by-side within fixed regions (columns 0-27, 28-55, 56-83). Even minor transformations cause severe distortions:

- Rotations cause digits to tilt and overlap with adjacent positions
- Width/height shifts move digits into wrong regions (e.g., first digit bleeding into the middle position)
- Zoom operations crop critical parts of digits

The Architectural Root Cause: Both techniques failed because the model's output layer is structurally limited to 640 fixed classes. No amount of regularisation or data variation can enable it to predict labels from a different set of 200 classes. This demonstrates that the problem requires a fundamental architectural redesign (addressed in Task 4), not optimisation of the existing architecture.

TASK 4

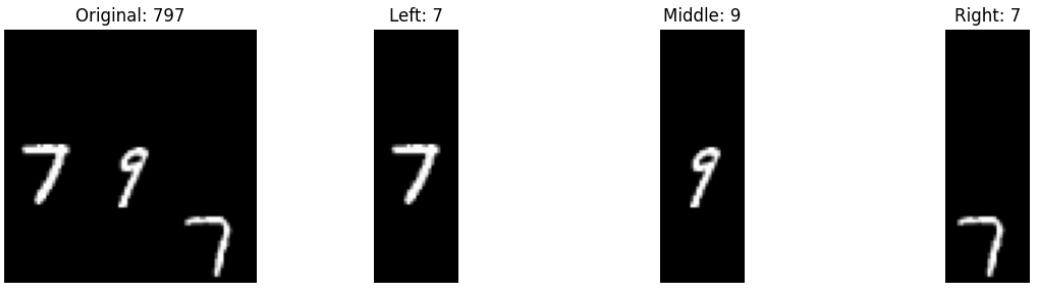
4.1 Problem Statement

- The results from Tasks 2 and 3 showed that treating each 3-digit combination as its own class does not work. The model got 0% accuracy on validation and test sets because these sets had label combinations that were not in the training data. If the model only learned to recognise 640 specific combinations, it cannot predict the 200 different combinations in the test set.
- To fix this, I took a different approach. Instead of classifying entire images, the model should learn to recognise individual digits (0-9). Then any 3-digit combination can be predicted by identifying each digit separately and joining the results together.

4.2 Image Splitting Strategy

- Each 84x84 image was split into three vertical slices of 84x28 pixels. The first slice (columns 0-27) contains the first digit, the middle slice (columns 28-55) contains the second digit, and the right slice (columns 56-83) contains the third digit.
- The slices are 84 pixels tall rather than 28 because the digits appear at different vertical positions within each column. Taking the full

height ensures the digit is captured regardless of where it sits vertically.



4.3 Single-Digit Dataset Creation

- After splitting, each original image becomes three separate training samples. The 64,000 training images produced 192,000 single-digit images. Each digit label was extracted from the 3-digit string. For example, an image labelled "797" produces three samples with labels 7, 9, and 7.
- This changes the problem from 640 classes (3-digit combinations) to just 10 classes (digits 0-9). The same process was applied to validation (48,000 samples) and test (60,000 samples) sets.

4.4 Training the Single-Digit CNN

- A CNN was trained to classify single digits (0-9). The architecture was the same as Task 2 but with an input shape of 84x28x1 and an output layer of 10 classes instead of 640.
- Training results (over 5 epochs):
    - Training accuracy: 99.71%
    - Validation accuracy (during training): 99.24%
- The model learned quickly and showed no obvious signs of overfitting. The validation set was then used to evaluate performance on full 3-digit predictions before moving to the test set.

4.5 Prediction and Concatenation

- To predict a 3-digit label from a full 84x84 image:
    - 1. Split the image into three 84x28 pieces
    - 2. Predict each digit using the trained CNN
    - 3. Concatenate the three predictions to form the final label

- For example, if the model predicts 7, 9, 7 for the three pieces, the final prediction is "797".
- The model was first evaluated on the validation set (16,000 images) to check performance before final testing. Validation accuracy was 97.53%, confirming the approach works. No hyperparameter tuning was needed given the strong results.

4.6 Evaluation Results

- **Test set performance:**
  - Test accuracy: 97.60%
  - F1 score: 0.9853

Comparison with Task 2:

Approach	Test Accuracy	F1 Score
Task 2: 640 Class CNN	0.00%	0.0000
Task 4: Split + 10 -class CNN	97.60%	0.9853

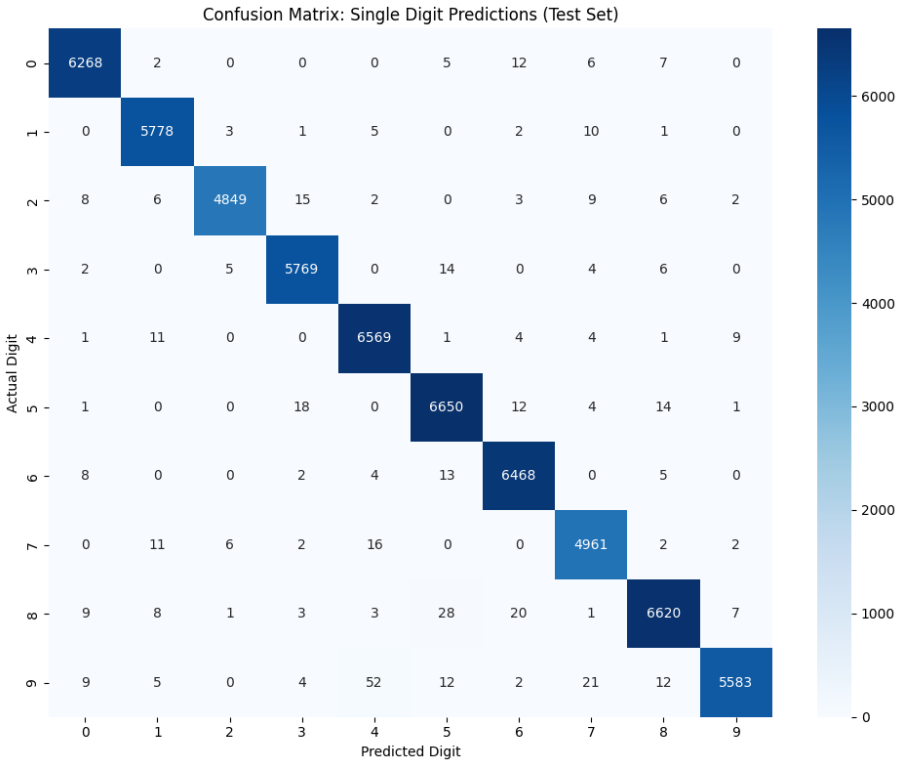
- The splitting approach solved the zero-overlap problem completely. By learning to recognise individual digits, the model can now predict any 3-digit combination, including labels it never saw during training.

4.7 Confusion Matrix Analysis

- The confusion matrix shows how well the model predicts each individual digit on the test set.
- Per digit accuracy:

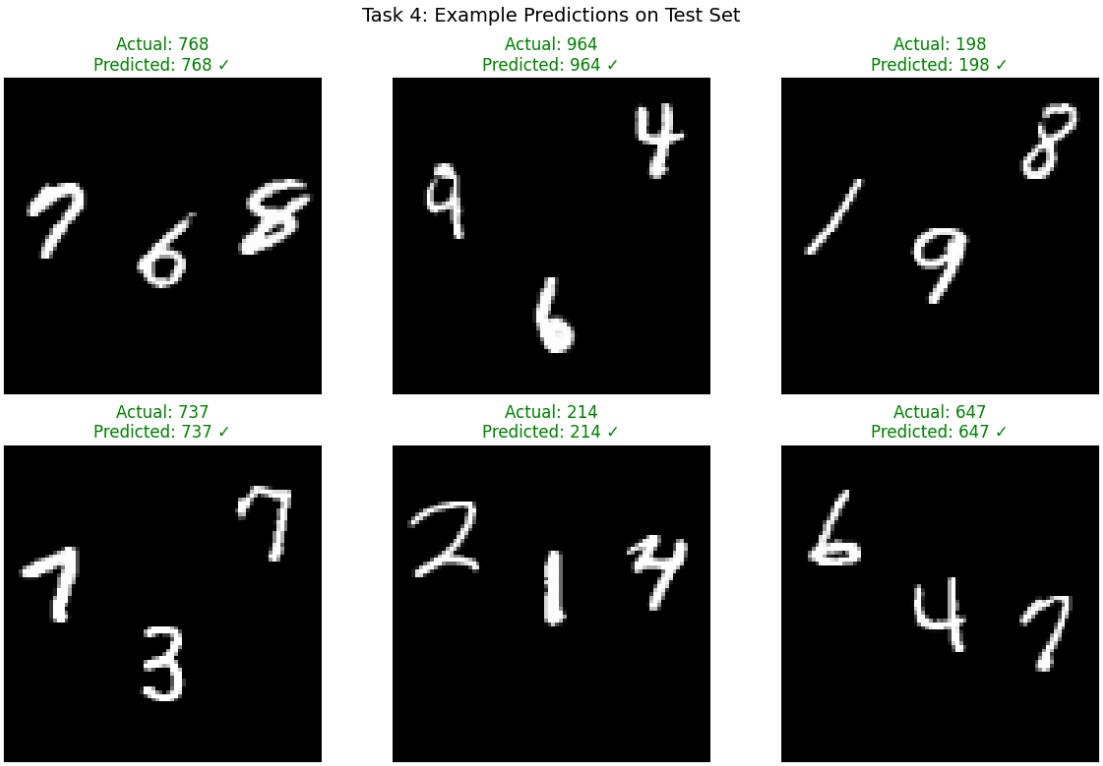
Digit	Accuracy
0	99.49%
1	99.62%
2	98.96%
3	99.47%
4	99.53%
5	99.25%
6	99.51%
7	99.22%
8	98.81%
9	97.95%

- All digits achieved above 97% accuracy. Digit 9 had the lowest accuracy (97.95%), possibly due to visual similarity with other digits. Digit 1 had the highest (99.62%). The balanced performance across all digits confirms the model generalises well.



4.8 Example Predictions

- Six random test images were selected to visualise the model's predictions. All six were predicted correctly, showing the model reliably identifies digits across different handwriting styles and positions.



4.9 Why Task 4 Outperformed Task 2

The dramatic difference in performance—97.60% vs 0.00%—stems from fundamental differences in how each approach handles the problem's structure.

- **Zero-overlap catastrophic failure:** Task 2 achieved 0% because its Dense(640) output layer can only predict the 640 specific combinations it was trained on. When the test set contains 200 entirely different combinations, the model cannot output labels that don't exist in its vocabulary. Every prediction is guaranteed to be wrong.
- **Compositional generalisation:** Task 4 treats digits as reusable building blocks. Once the model learns to recognise "7", it can use that knowledge in any combination—"789", "372", or "007". This enables the model to construct any of the 1000 possible combinations from just 10 learned patterns. Task 2 cannot do this because it treats each combination as an independent, unrelated class.
- **More training samples per class:** In Task 2, each of the 640 combinations had approximately 100 training samples (64,000 ÷ 640). In Task 4, each digit class has 19,200 samples (192,000 ÷ 10).

This 20× increase in samples per class leads to better feature learning and generalisation for each digit.
- **Architectural alignment:** Task 2 processes the entire 84×84 image through a single classification pipeline, forcing the model to learn complex spatial relationships across all three digits positions simultaneously. Task 4 explicitly isolates each digit region (84×28 slices), allowing the model to focus on learning individual digit patterns. This decomposition matches the compositional structure of the problem and enables the model to learn simpler, more transferable features.

This demonstrates a key principle in machine learning: model architecture must align with the underlying structure of the data. Task 2's monolithic approach was fundamentally incompatible with the compositional nature of multi-digit labels, while Task 4's decomposition strategy leveraged this structure for superior generalisation.

Task 5

TASK 5a

5a.1 Problem Understanding

- Tasks 2 and 3 showed that treating the 3-digit label as a single 1000-class problem fails. Task 4 solved this by splitting images, but Task 5(a) asks for a solution that uses the multi-label structure without splitting.
- The idea is that each image has three separate targets: digit 1 (0-9), digit 2 (0-9), and digit 3 (0-9). Instead of one output layer with 1000 classes, the model has three output layers with 10 classes each.
- The 1000-class approach treats "123" and "124" as unrelated classes, even though they share two digits. The multi-output approach recognises they differ only in the third digit.

5a.2 Multi-Output CNN Architecture

- I designed a CNN with a shared backbone and three separate output heads:

    - **Shared backbone:** Three convolutional layers that process the full 84×84 image. The first two layers have Conv2D → MaxPooling, and the third has Conv2D only. Filter sizes are 32, 64, and 64.
    - **Shared dense layer:** After flattening, a Dense layer with 128 units captures features useful for all three predictions.
    - **Three output heads:** Each head is a Dense layer with 10 units and softmax activation. One head predicts digit 1, another predicts digit 2, and the third predicts digit 3.
- The shared backbone learns edges, curves, and digit shapes that apply regardless of position. Each output head then specialises for its specific digit position.
- Since the test labels are new combinations of the same 10 digits, a model that recognises individual digits should generalise to unseen combinations.
- This design follows principles from multi-task learning, where related tasks benefit from shared representations (Caruana, 1997). Training all three digit classifiers together encourages the model to learn more robust features.

5a.3 Training Configuration

- Optimiser: Adam (default learning rate)
- Loss: Sparse categorical cross-entropy for each output head
- Epochs: 5
- Metrics: Per-head accuracy for digit1, digit2, digit3
- Batch size: 32
- Validation: 10% split from training data (validation_split=0.1)
- The model was compiled with three loss functions (one per output) and trained end-to-end.

5a.4 Evaluation Results

- Test set performance:

Metric	Result
Test Accuracy	90.50%
Test F1 Score	0.9412

- Validation Accuracy: 90.95%

- Comparison with other approaches:

Approach	Test Accuracy	F1 Score
Task 2	0.00%	0.0000
Task 4	97.60%	0.9853
Task 5a: Multi-output CNN	90.50%	0.9412

- The multi-output CNN achieved 90.50% accuracy on unseen 3-digit combinations. This is a massive improvement over the 0% baseline from Task 2

5a.5 Comparison with Task 4

- Task 4 (97.60%) outperformed Task 5a (90.50%) by about 7.10 percentage points. This makes sense:
    - Task 4 explicitly isolates each digit region before classification. There is no interference between positions.
    - Task 5a relies on the network to implicitly learn which part of the image corresponds to which output. This is harder.
    - The shared backbone in Task 5a must balance features for all three positions, which can lead to compromises.

5a.6 Conclusion

- The multi-output approach works. By respecting the structure of the labels (three separate digits rather than one combined class), the model can generalise to unseen digit combinations.
- Task 4 is still better for this dataset because the digits are cleanly separated into columns. But Task 5a shows that a single end-to-end model can also solve the problem without manual preprocessing

TASK 5b

5b.1 Motivation and Approach

- Task 5b explored an alternative solution: using Generative Adversarial Networks (GANs) to synthesise new training images to augment the dataset.

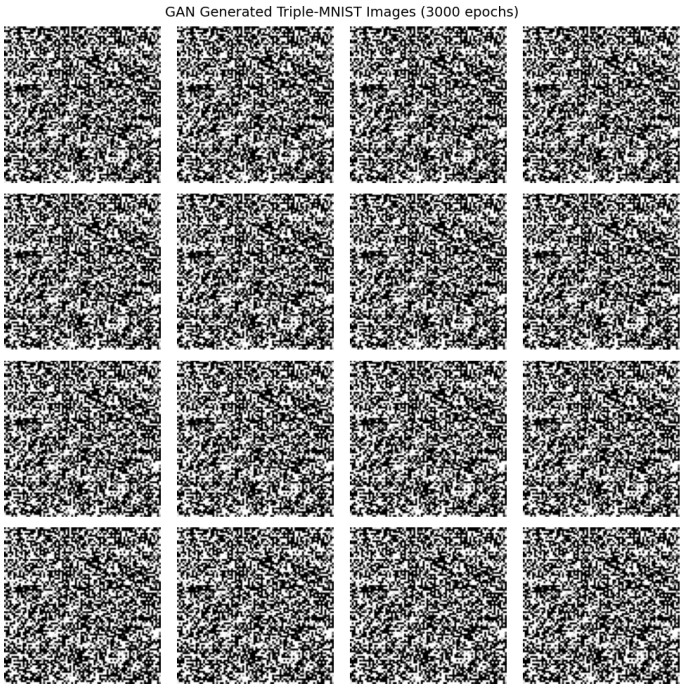
5b.2 Approach 1: Full-Image GAN (84×84)

Architecture:

Generator:
- Input: 100-dimensional noise vector (latent space)
- Dense layer projecting to 7 x 7 x 256 feature map
- **Four** Conv2DTranspose layers with BatchNormalisation and **ReLU**
- Output: 112×112×1 image with tanh activation, then cropped to 84×84

Discriminator:
- Input: 84×84×1 image
- Three Conv2D layers (32→64→128 filters) with LeakyReLU
- Dropout (30%) after each layer for stability
- Output: Single sigmoid probability (real/fake)

Training Configuration:
- Epochs: 3,000
- Batch size: 64
- Optimiser: Adam (learning rate=0.0002, β_1=0.5)
- Label smoothing: Real labels=0.9, Fake labels=0.1
- Discriminator trained 2× per generator step



5b.3 Approach 2: Single-Digit GAN (84×28)

- Given the failure of the full-image approach, a simplified GAN was trained on individual digit slices (84×28 pixels) extracted from the training images.

Rationale:

- Smaller image size (84×28 vs 84×84) is easier to learn
- 192,000 single-digit training samples (64,000 × 3 digits)
- Reduced complexity: generating one digit instead of three

Architecture:

Generator:

- Input: 100-dimensional noise vector
- Dense layer projecting to 21×7×256 feature map
- **Two Conv2DTranspose** layers upsampling to 84×28
- BatchNormalisation + ReLU after each upsampling
- Output: Conv2D(1, 3×3, tanh)
- Total parameters: 4,457,729

Discriminator:

- Input: 84×28×1 image

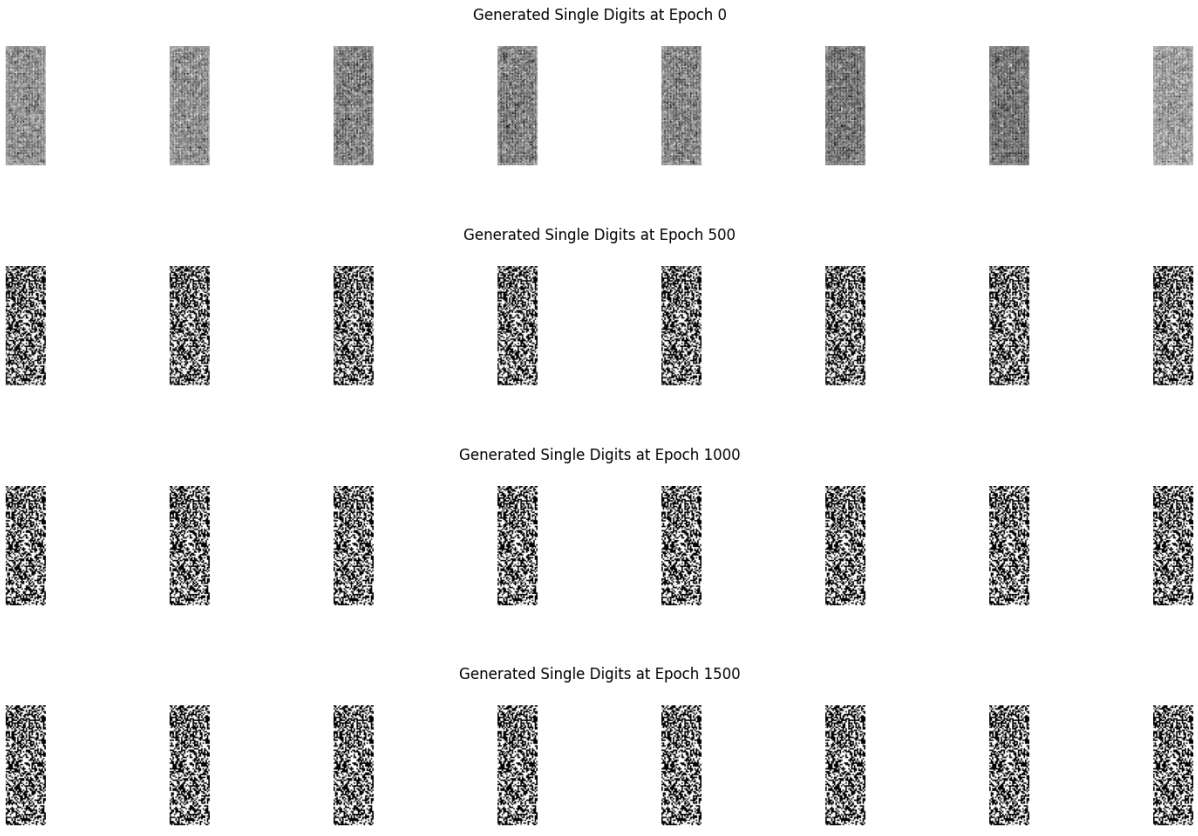
- Three Conv2D layers with LeakyReLU, BatchNormalisation and Dropout (0.25)
- Output: Single sigmoid probability
- Total parameters: 170,977

Training Configuration:

- Epochs: 2,000
- Batch size: 64
- Training data: 192,000 single-digit images
- Label smoothing **(real=0.9, fake=0.1)**

Results:

Epoch	Discriminator Loss	Generator Loss
0	0.7044	0.7030
500	0.9700	0.1784
1000	1.1558	0.1489
1500	1.2210	0.1384



5b.4 Analysis of Failure

- Training never stabilised: generator loss dropped rapidly while discriminator loss rose steadily suggests possible mode collapse.

- Discriminator accuracy remained at 0% throughout, meaning it could not tell real images from fake ones.

- Stability tweaks (label smoothing, dropout, discriminator-heavy training) did not improve sample quality.

- The architectures were likely too simple for Triple-MNIST's complexity (three digits per image).

5b.5 Conclusion

- Both full-image and single-digit GANs produced noise-like samples, so augmentation was not usable.
- This reinforces that aligning model design with data structure (Task 4) was more effective than brute-force data generation.
- Despite the failure, this exploration demonstrates the difficulty of GAN training and validates the architectural approach taken in Task 4.