GAN: DELE CA2

Chew Yee Jing - P2415860
Vijeyakumar Dakshaa - P2415899

Research Problem

How can suitable Generative Adversarial Network (GAN) architectures be applied to generate 160 small black-and-white images across 16 classes using the EMNIST Letters dataset, and how can their quality be effectively evaluated and improved?

Overview of the 3 GAN Models Used

Unconditional DCGAN	Multi-DCGAN	Conditional GAN (CGAN / cDCGAN)
Description: ◦ Generates images solely from random noise without any label conditioning	**Description:** ◦ Separate DCGAN model trained independently for each class ◦ Generates images specific to a single class with dedicated generators	**Description:** ◦ Generator and discriminator conditioned on class labels ◦ Allows controlled image generation based on class input
Key Point: ◦ Simple architecture but cannot control which class of image is generated	**Key Point:** ◦ Better class-specific image quality but requires training 16 separate models	**Key Point:** ◦ Improves generation diversity and relevance by leveraging label information

BACKGROUND RESEARCH AND BASIC KNOWLEDGE

Is GAN Supervised or Unsupervised?

Classic GAN (DCGAN)

- Type: Unsupervised learning
- Learns data distribution without labels
- Discriminator only sees “real” vs “fake” — no class/category info

Conditional GAN (cGAN / cDCGAN)

- Type: Hybrid (Supervised + Unsupervised)
- Supervised part: Uses labels to condition both Generator & Discriminator → ensures correct class output
- Unsupervised part: Learns realistic features and style from unlabelled aspects of the data

BACKGROUND RESEARCH AND BASIC KNOWLEDGE

If you are asked to generate coloured images instead of black-and-white ones, do you think it would be easier or harder to produce better quality results?

Increased Data Complexity:

- Colored images have three channels (RGB) instead of one, adding significantly more information for the GAN to model.

Larger Model Requirements:

- Both the generator and discriminator must be expanded to process the extra channels, demanding more computation and longer training times.

Training Stability Challenges:

- Greater complexity can lead to instability during training, making precise hyperparameter tuning and regularization more important.

More Complex Evaluation:

- Quality assessment must consider not only shape and detail but also accurate and consistent color reproduction.

Exploratory Data Analysis

```
labels = df.iloc[:, 0]
images = df.iloc[:, 1:].values
```

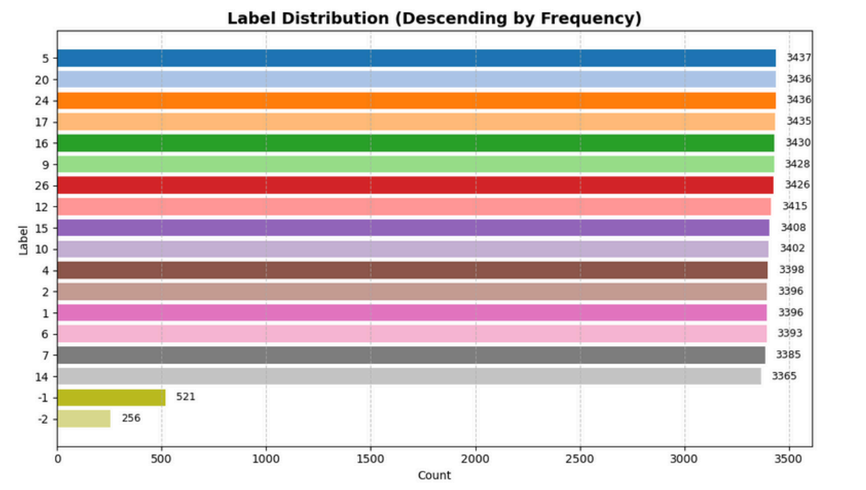
```
df.shape
```

```
(64828, 785)
```

	24	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.552	
count	55363.000000	55363.000000	55363.000000	55363.000000	55363.000000	55363.000000	55363.000000	55363.000000	55363.00000	55363.000000	...	55363.000000	5
mean	11.589003	1.787602	1.787602	1.787602	1.787710	1.791991	1.810921	1.819807	1.81009	2.259542	...	1.787602	
std	7.467979	17.352303	17.352303	17.352303	17.368913	17.359561	17.449014	17.500948	17.45474	19.855490	...	17.352303	
min	-2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	...	0.000000	
25%	5.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	...	0.000000	
50%	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	...	0.000000	
75%	16.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	...	0.000000	
max	26.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.00000	255.000000	...	255.000000	

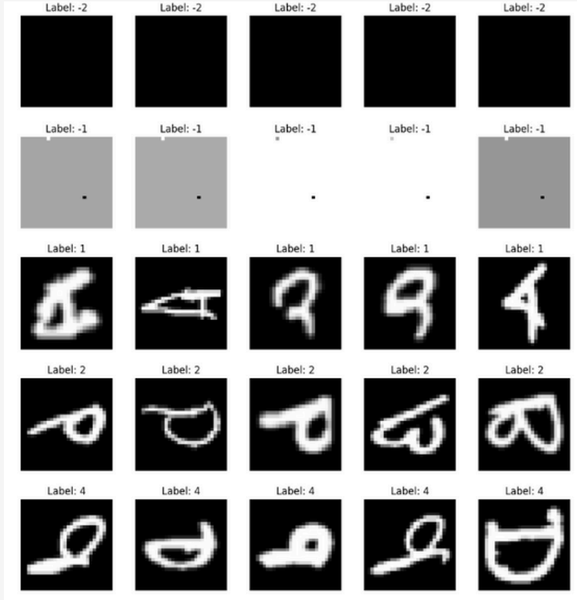
Labels Distribution

```
Unique labels: [-2, -1, 1, 2, 4, 5, 6, 7, 9, 10, 12, 14, 15, 16, 17, 20, 24, 26]
```




```
24
-2      256
-1      521
 1     3396
 2     3396
 4     3398
 5     3437
 6     3393
 7     3385
 9     3428
10     3402
12     3415
14     3365
15     3408
16     3430
17     3435
20     3436
24     3436
26     3426
```

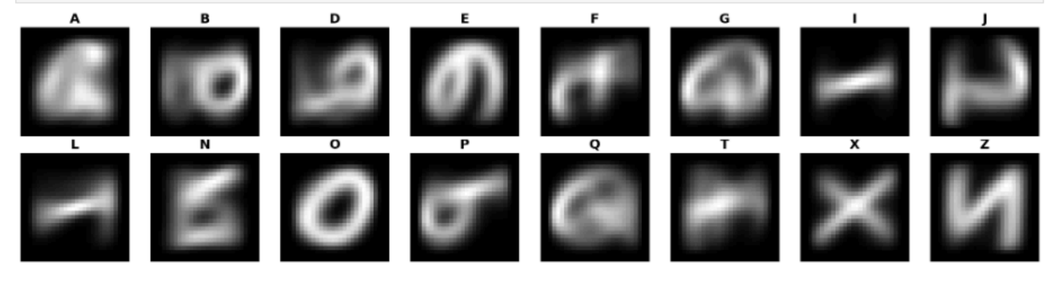
Sample Images by Class



- Each label corresponds to a letter in the EMNIST dataset
- Labels -2 and -1 appear to be mostly blank or minimal marks
- Other labels (e.g., 1, 2, 4, 5, 6) show handwritten letter variations.
- Variation in handwriting style and stroke thickness may affect GAN training quality.

Exploratory Data Analysis

Dataset Image Quality



- These are the per-letter average images for the 16 target classes, and they appear blurry due to variations in handwriting styles across samples.
- Hard to distinguish the features

What class(es) is/are relatively easier/harder to generate? Why?

Easier:

- Letters with distinct shapes and low variation, e.g., I, L, O, X.
- Few strokes, clear boundaries → less confusion for G.

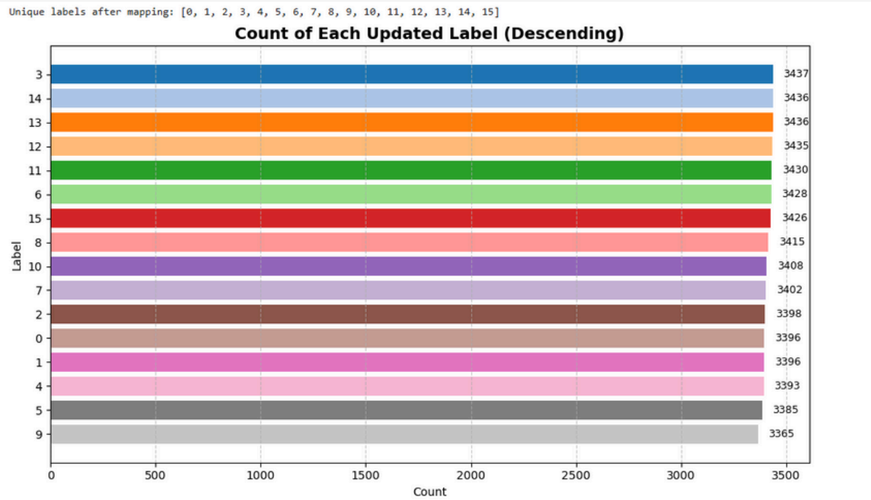
Harder:

- Letters with similar structures or fine details, e.g., C vs G, M vs N, E vs F.
- Greater style variability in handwriting → harder for G to capture consistently.

DATA CLEANING

Drop Duplicates

(64828, 785) ⟶ (55363, 785)



Filtered out unwanted classes

```
Unique labels after cleaning: [1, 2, 4, 5, 6, 7, 9, 10, 12, 14, 15, 16, 17, 20, 24, 26]
```

- Removed unwanted classes -1 and -2 (which were blank or placeholder images).
- Transposed (.T) the image array before plotting, which fixed the sideways orientation issue that EMNIST images have by default.

Re-mapped cleaned labels to a sequential range (0–15)

```
Unique labels after mapping: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

- Re-mapped cleaned labels to a sequential range (0–15)
- Required for label-aware GANs (Multi-DCGAN, Conditional DCGAN), not needed for Unconditional DCGAN.

FEATURE ENGINEERING

Normalize pixel values to [-1, 1]	• images = (images - 127.5) / 127.5 • Pixel values were scaled from the original [0, 255] range to [-1, 1], which is important for stabilizing GAN training and matching the output activation (`tanh`) of the generator.
Reshape	• images = images.reshape(-1, 28, 28) • The flat image arrays were reshaped into 28×28 pixel format to match the original image dimensions.
Fix orientation: rotate 90° clockwise then flip horizontally	• The EMNIST dataset stores images in a transposed and inverted format. We corrected this by rotating each image 90 degrees counter-clockwise and flipping it horizontally to restore proper orientation.
Reshape back to (-1, 28, 28, 1)	• Adds a channel dimension at the end. • Required because TensorFlow/Keras Conv2D layers expect 4D tensors: (batch, height, width, channels).

METRICS FOR EVALUATION

Eye-power 01

Discriminator Accuracy 02

G loss and D loss 03

Key Training Metrics - Ideal Ranges

- **Discriminator Accuracy: 0.55 – 0.75**

1. If accuracy is too high (e.g., >80% consistently) → the discriminator is overpowering the generator, so the generator struggles to improve.
2. If accuracy is too low (e.g., <50% consistently) → the generator is fooling the discriminator too easily, which usually means the discriminator isn't learning enough.
3. Around 50% can happen if both models are perfectly balanced, but in practice, that often means the discriminator is guessing randomly which isn't what we want for most of training.

- Generator Loss: around 1.0 – 2.5
- Discriminator Loss: around 0.5 – 2.0
- Mode Collapse Signs: more than 80% identical outputs

UNCONDITIONAL DCGAN

DCGAN (Deep Convolutional GAN)

- Uses convolutional layers in both Generator and Discriminator for image generation.
- Generator: Upsamples random noise into a 28×28 grayscale image.
- Discriminator: Distinguishes real vs. fake images.
- Unconditional: No labels given → outputs random classes from all 16 EMNIST categories.

BASELINE MODELLING – UNCONDITIONAL DCGAN

```
=== Generator Architecture ===
Model: "DCGAN_Generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 150)]             0

 dense_2 (Dense)             (None, 6272)              947072

 reshape_1 (Reshape)         (None, 7, 7, 128)         0

 up_sampling2d_2 (UpSampling  (None, 14, 14, 128)      0
 2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 128)       147584

 batch_normalization_3 (Batc  (None, 14, 14, 128)      512
 hNormalization)

 re_lu_2 (ReLU)              (None, 14, 14, 128)       0

 up_sampling2d_3 (UpSampling  (None, 28, 28, 128)      0
 2D)

 conv2d_6 (Conv2D)           (None, 28, 28, 64)        73792

 batch_normalization_4 (Batc  (None, 28, 28, 64)       256
 hNormalization)

 re_lu_3 (ReLU)              (None, 28, 28, 64)        0

 conv2d_7 (Conv2D)           (None, 28, 28, 1)         577

=================================================================
Total params: 1,169,793
Trainable params: 1,169,409
Non-trainable params: 384
```

Model	Layer / Operation	Output Shape	Why This Choice
Generator	Input	(latent_dim,)	Random noise vector to allow diverse image generation.
	Dense	(7×7×128)	Large feature map for more detail capture at start.
	Reshape	(7, 7, 128)	Convert dense output into spatial feature maps.
	UpSampling2D	(14, 14, 128)	Doubles resolution while preserving learned features.
	Conv2D (128)	(14, 14, 128)	Adds spatial detail after upsampling.
	BatchNormalization	(14, 14, 128)	Stabilizes activations and improves training convergence.
	ReLU	(14, 14, 128)	Encourages positive activations for better feature growth.
	UpSampling2D	(28, 28, 128)	Final resolution upsampling to target image size.
	Conv2D (64)	(28, 28, 64)	Adds finer detail features.
	BatchNormalization	(28, 28, 64)	Keeps generator training stable.
	ReLU	(28, 28, 64)	Activation to enhance feature learning.
	Conv2D (Output)	(28, 28, channels)	Outputs grayscale image, `tanh` for range [-1, 1] matching preprocessing.

```
latent_dim = 150
```

150 is a balanced choice.

Large enough to hold variety across 16 classes of images in CA2 GAN task, but small enough to train efficiently given the dataset.

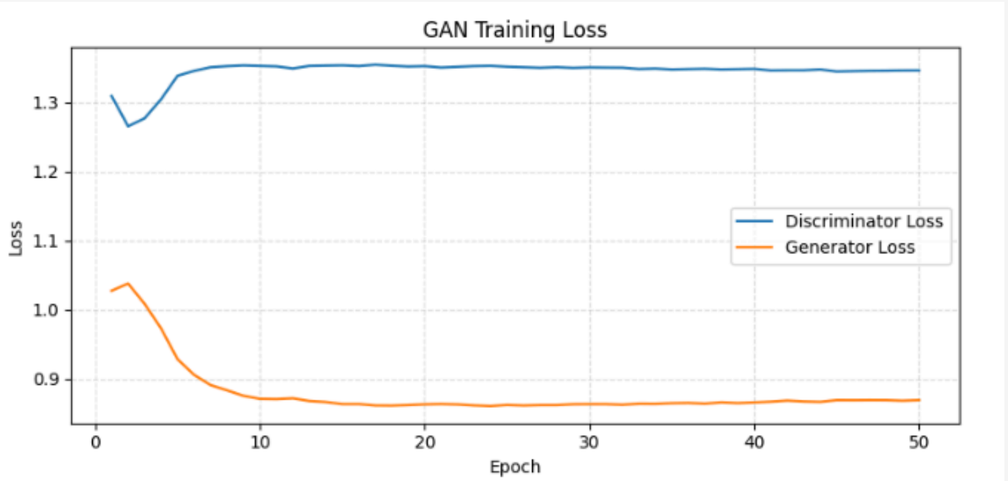
```
=== Discriminator Architecture ===
Model: "DCGAN_Discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_8 (Conv2D)           (None, 14, 14, 64)        640

 leaky_re_lu_2 (LeakyReLU)   (None, 14, 14, 64)        0

 dropout_2 (Dropout)         (None, 14, 14, 64)        0

 conv2d_9 (Conv2D)           (None, 7, 7, 128)         73856

 batch_normalization_5 (Batc  (None, 7, 7, 128)        512
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 7, 7, 128)         0

 dropout_3 (Dropout)         (None, 7, 7, 128)         0

 flatten_1 (Flatten)         (None, 6272)              0

 dense_3 (Dense)             (None, 1)                 6273

=================================================================
Total params: 81,281
Trainable params: 81,025
Non-trainable params: 256
```

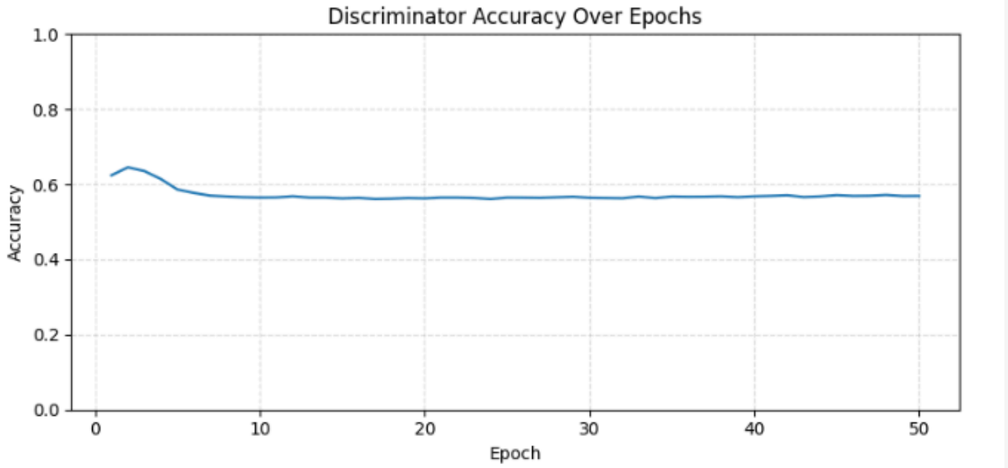
Model	Layer / Operation	Output Shape	Why This Choice
Discriminator	Input	(28, 28, channels)	Takes real or generated images for classification.
	Conv2D (64)	(14, 14, 64)	Extracts low-level patterns while reducing resolution.
	LeakyReLU(0.2)	(14, 14, 64)	Avoids dead neurons, keeps gradient flow.
	Dropout(0.3)	(14, 14, 64)	Regularization to prevent overfitting.
	Conv2D (128)	(7, 7, 128)	Learns deeper, mid-level features.
	BatchNormalization	(7, 7, 128)	Stabilizes training and improves gradient flow.
	LeakyReLU(0.2)	(7, 7, 128)	Allows small negative outputs for stability.
	Dropout(0.3)	(7, 7, 128)	Further regularization.
	Flatten	(6272,)	Converts feature maps to a vector for classification.
	Dense (Output)	(1,)	Sigmoid outputs probability of real/fake for adversarial loss.

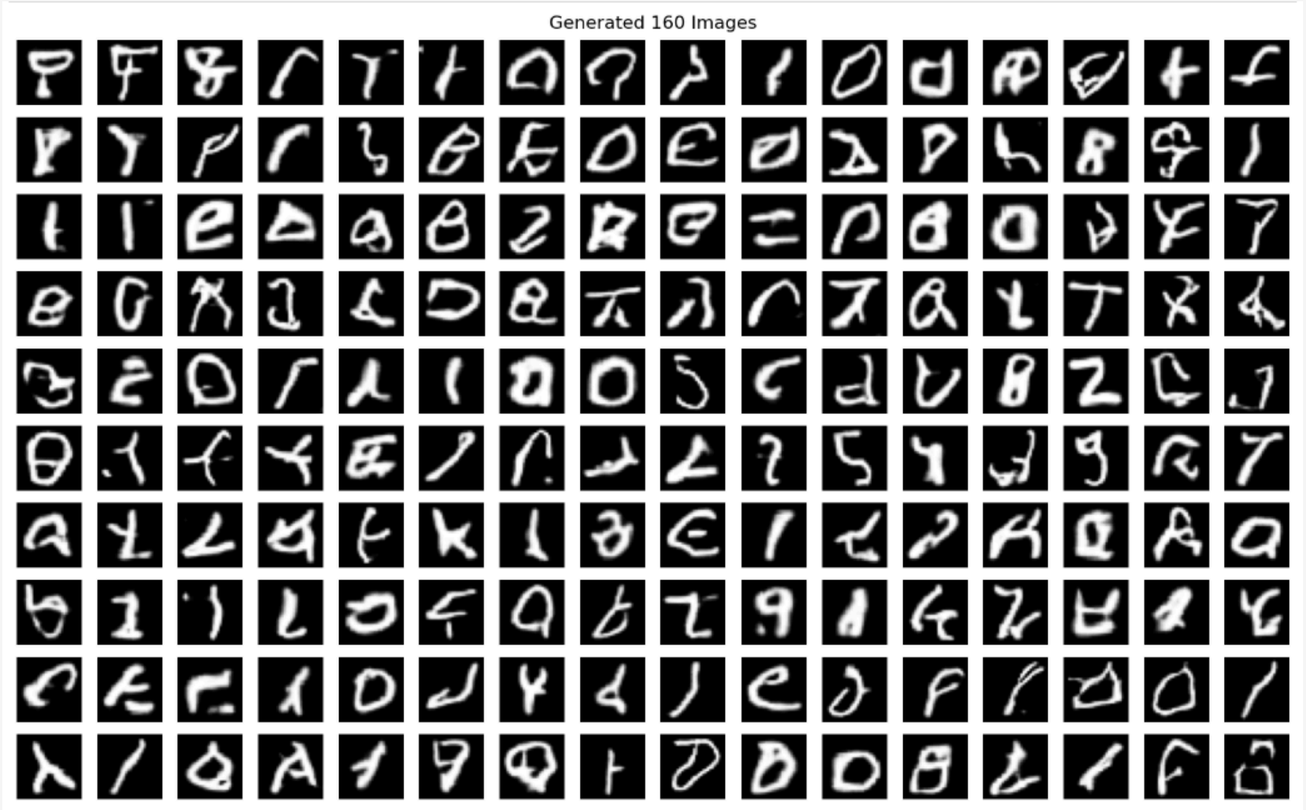
```
dcgan.compile(
    g_optimizer=tf.keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    d_optimizer=tf.keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5)
)
```

- Smaller, more stable updates for GAN training
- Lower momentum so updates respond faster to changes
- Common best practices for stable image generation

BASELINE MODELLING - UNCONDITIONAL DCGAN







- Losses stabilize early (~epoch 10) with Discriminator Loss ≈ 1.35 and Generator Loss ≈ 0.85, showing balanced training without divergence.
- Discriminator Accuracy ≈ 0.6 throughout, within the ideal 0.55–0.75 range for stable GAN training.
- Model reaches equilibrium quickly, providing a stable reference point for comparing future improvements.

MODEL IMPROVEMENT - UNCONDITIONAL DCGAN

Generator Architecture

Layer	Type	Parameters	Activation	Output Shape
Input	Input	`latent_dim`	–	`(latent_dim,)`
Dense	Fully Connected	Units = `256*7*7`	ReLU	`(256*7*7,)`
Reshape	Reshape	`(7, 7, 256)`	–	`(7, 7, 256)`
Conv2DTranspose	Transposed Conv	Filters = 256, Kernel = 3×3, Strides = 2, Padding = same	–	`(14, 14, 256)`
BatchNormalization	BN	Momentum = 0.8	–	`(14, 14, 256)`
Activation	ReLU	–	–	`(14, 14, 256)`
Conv2DTranspose	Transposed Conv	Filters = 128, Kernel = 3×3, Strides = 2, Padding = same	–	`(28, 28, 128)`
BatchNormalization	BN	Momentum = 0.8	–	`(28, 28, 128)`
Activation	ReLU	–	–	`(28, 28, 128)`
Conv2DTranspose	Transposed Conv	Filters = 64, Kernel = 3×3, Strides = 1, Padding = same	–	`(28, 28, 64)`
BatchNormalization	BN	Momentum = 0.8	–	`(28, 28, 64)`
Activation	ReLU	–	–	`(28, 28, 64)`
Conv2D	Conv	Filters = `channels`, Kernel = 3×3, Padding = same	Tanh	`(28, 28, channels)`

Discriminator Architecture

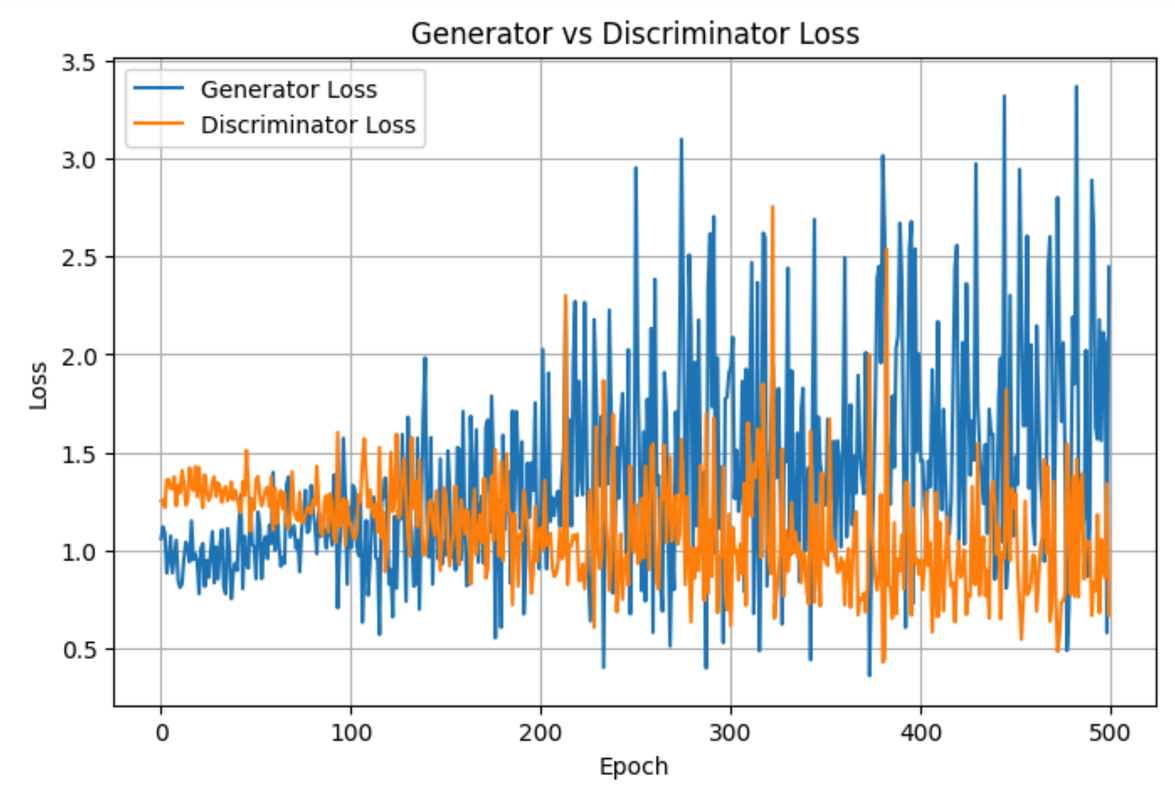
Layer	Type	Parameters	Activation	Output Shape
Input	Input	`img_shape`	–	`(28, 28, channels)`
Conv2D	Conv	Filters = 64, Kernel = 3×3, Strides = 2, Padding = same	LeakyReLU(0.2)	`(14, 14, 64)`
Dropout	Dropout	0.4	–	`(14, 14, 64)`
Conv2D	Conv	Filters = 128, Kernel = 3×3, Strides = 2, Padding = same	LeakyReLU(0.2)	`(7, 7, 128)`
BatchNormalization	BN	Default momentum	–	`(7, 7, 128)`
Dropout	Dropout	0.4	–	`(7, 7, 128)`
Conv2D	Conv	Filters = 256, Kernel = 3×3, Strides = 1, Padding = same	LeakyReLU(0.2)	`(7, 7, 256)`
BatchNormalization	BN	Default momentum	–	`(7, 7, 256)`
Dropout	Dropout	0.4	–	`(7, 7, 256)`
Flatten	Flatten	–	–	`(7*7*256,)`
Dense	Fully Connected	Units = 1	Sigmoid	`(1,)`

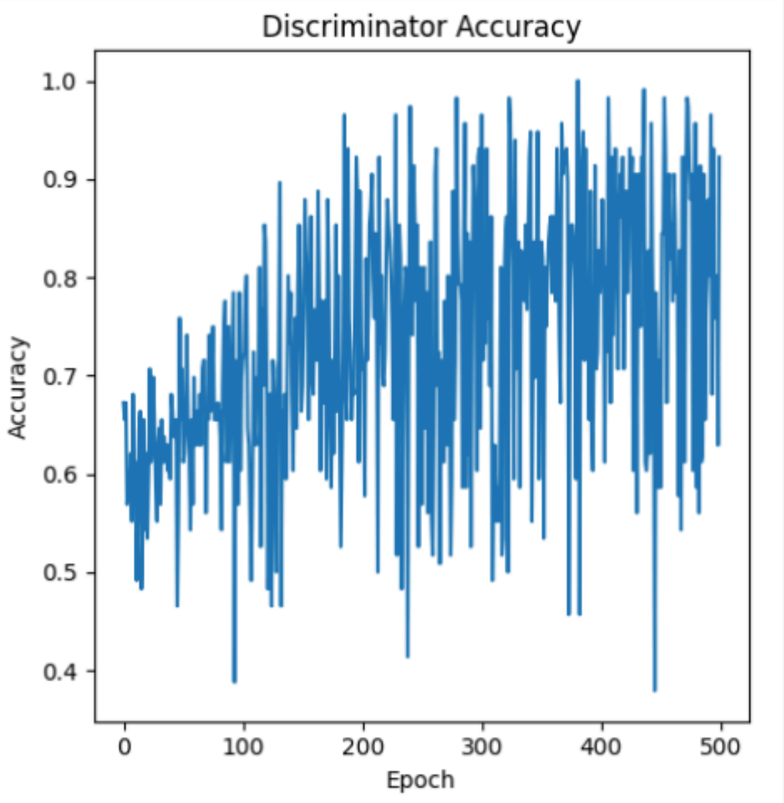
```python
def discriminator_loss(self, real_output, fake_output):
    real_loss = self.loss_fn(tf.ones_like(real_output) * 0.9, real_output)  # label smoothing
    fake_loss = self.loss_fn(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss
```

This stops the discriminator from assigning “perfect certainty” too quickly, which can make the generator’s job impossible.

Component	Baseline	Improved	Reason for Change
Generator – Dense Output	7×7×128	7×7×256	Learning complex patterns
Generator – Upsampling	UpSampling2D + Conv2D	Conv2DTranspose (stride=2)	Learns upsampling kernel directly
Generator – At 28×28	No extra conv	+ refinement Conv2DTranspose(64, stride=1)	Sharpens edges and adds fine details after reaching target size.
Generator – BatchNorm	Only some layers	After every upsampling (momentum=0.8)	Stabilizes training
Discriminator – Conv Blocks	2 blocks	3 blocks (extra stride=1 block)	Allows inspecting fine details without downsampling too much.
Discriminator – BatchNorm	Only 2nd block	All blocks	Improves gradient flow and stability across the network.
Discriminator – Dropout	0.3	0.4	Stronger regularization to prevent overfitting and maintain G–D balance.

Discriminator Accuracy and Loss Dynamics





Balanced adversarial training :

- Both losses fluctuate in a competitive range, showing neither network has greater dominance, which helps prevent mode collapse.

Improved realism over time:

- Rising discriminator accuracy of 0.6 to 0.9 with stable losses suggests the generator is producing increasingly convincing samples while the discriminator stays challenged.

MODEL EVALUATION – EYE POWER

✓ A B d e F g I J L N O P Q T X Z

Clear	Marginal	Nonsense
The generated image is recognizably the intended letter, with correct shape and minimal distortion.	The image resembles the target letter but has imperfections that make it less clear.	The image does not resemble the intended letter at all, instead looking like a random scribble or another unrelated symbol

MODEL EVALUATION – EYE POWER

Clear

Marginal

Rest are Nonsense

Clear	Marginal	Nonsense
The generated image is recognizably the intended letter, with correct shape and minimal distortion.	The image resembles the target letter but has imperfections that make it less clear.	The image does not resemble the intended letter at all, instead looking like a random scribble or another unrelated letter.
66	34	60

62.5%

37.5%

MULTI-DCGAN

MULTIPLE DCGAN (Deep Convolutional GAN)

- From Single DCGAN → Multi-DCGAN
- Single DCGAN: 1 Generator + 1 Discriminator → learns all 16 letters.
- Multi-DCGAN: 1 DCGAN per letter → 16 Generators + 16 Discriminators.
- Each model trains only on its target letter.

BASELINE MODELLING - Multiple DCGAN

Generator Architecture

Layer	Type	Parameters	Activation	Output Shape
Input	Input	`(latent_dim,)`	–	`(latent_dim,)`
Dense	Fully Connected	Units = `7×7×128`	ReLU	`(6272,)`
Reshape	Reshape	`(7, 7, 128)`	–	`(7, 7, 128)`
UpSampling2D	Upsampling	–	–	`(14, 14, 128)`
Conv2D	Conv	128 filters, 3×3, padding = same	ReLU	`(14, 14, 128)`
BatchNormalization	BN	Default momentum	–	`(14, 14, 128)`
UpSampling2D	Upsampling	–	–	`(28, 28, 128)`
Conv2D	Conv	64 filters, 3×3, padding = same	ReLU	`(28, 28, 64)`
BatchNormalization	BN	Default momentum	–	`(28, 28, 64)`
Conv2D	Conv	`channels` filters, 3×3, padding = same	Tanh	`(28, 28, channels)`

Discriminator Architecture

Layer	Type	Parameters	Activation	Output Shape
Input	Input	`(28, 28, channels)`	–	`(28, 28, channels)`
Conv2D	Conv	64 filters, 3×3, stride=2, padding = same	LeakyReLU(0.2)	`(14, 14, 64)`
Dropout	Dropout	0.2	–	`(14, 14, 64)`
Conv2D	Conv	128 filters, 3×3, stride=2, padding = same	LeakyReLU(0.2)	`(7, 7, 128)`
BatchNormalization	BN	Default momentum	–	`(7, 7, 128)`
Dropout	Dropout	0.3	–	`(7, 7, 128)`
Flatten	Flatten	–	–	`(6272,)`
Dense	Fully Connected	Units = 1	Sigmoid	`(1,)`

Generator

- Dense → Reshape to 7×7×128
- UpSampling2D + Conv2D(128) → BatchNorm → ReLU
- UpSampling2D + Conv2D(64) → BatchNorm → ReLU
- Output: Conv2D(channels), Tanh

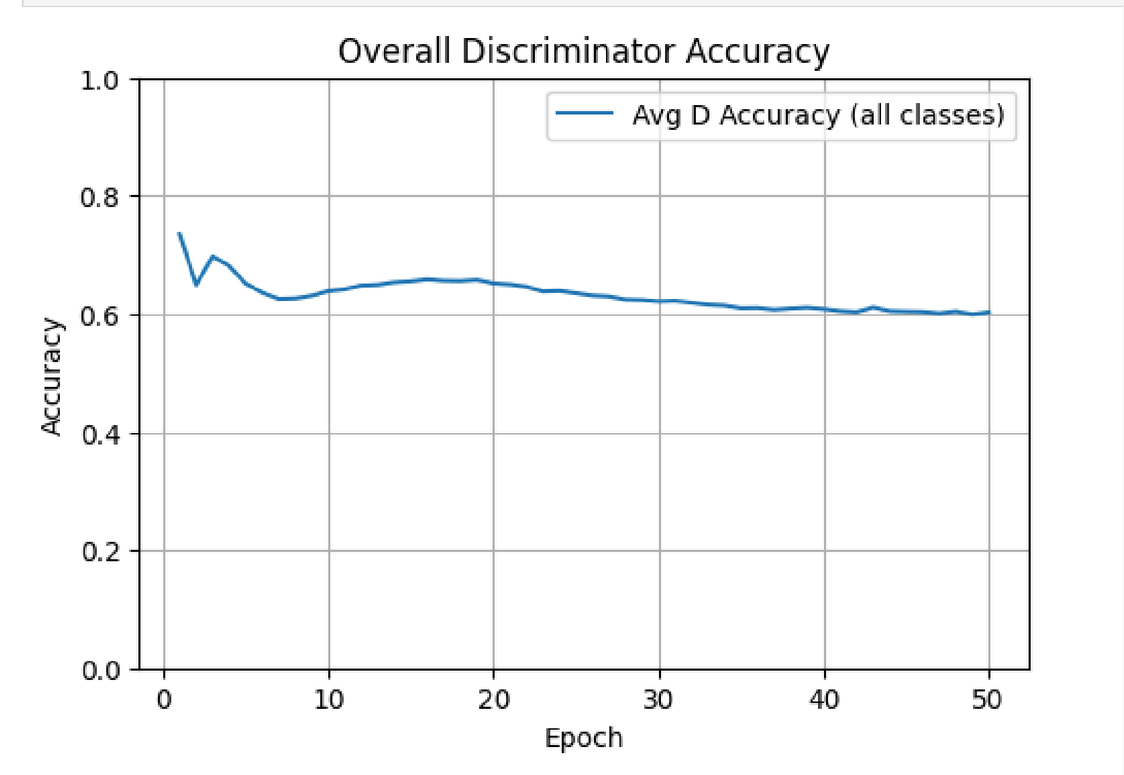
Discriminator

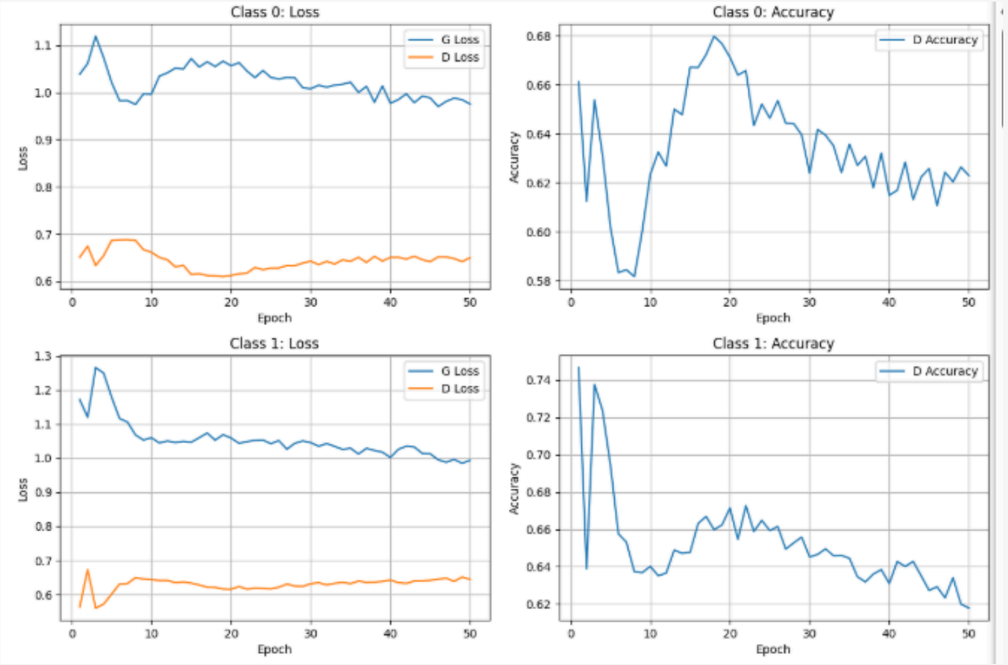
- Conv2D(64, stride=2) → LeakyReLU → Dropout(0.2)
- Conv2D(128, stride=2) → BatchNorm → LeakyReLU → Dropout(0.3)
- Flatten → Dense(1), Sigmoid

```
dcgan.compile(
    g_optimizer=Adam(0.0002, beta_1=0.5),
    d_optimizer=Adam(0.0002, beta_1=0.5)
)
```

```
=== Generator Summary ===
Model: "DCGAN_Generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_277 (InputLayer)      [(None, 150)]             0

 sequential_276 (Sequential)  (None, 28, 28, 1)        1169793

=================================================================
Total params: 1,169,793
Trainable params: 1,169,409
Non-trainable params: 384
_________________________________________________________________

=== Discriminator Summary ===
Model: "DCGAN_Discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_278 (InputLayer)      [(None, 28, 28, 1)]       0

 sequential_277 (Sequential)  (None, 1)                81281

=================================================================
Total params: 81,281
Trainable params: 81,025
Non-trainable params: 256
_________________________________________________________________
```

Discriminator Accuracy and Loss Dynamics



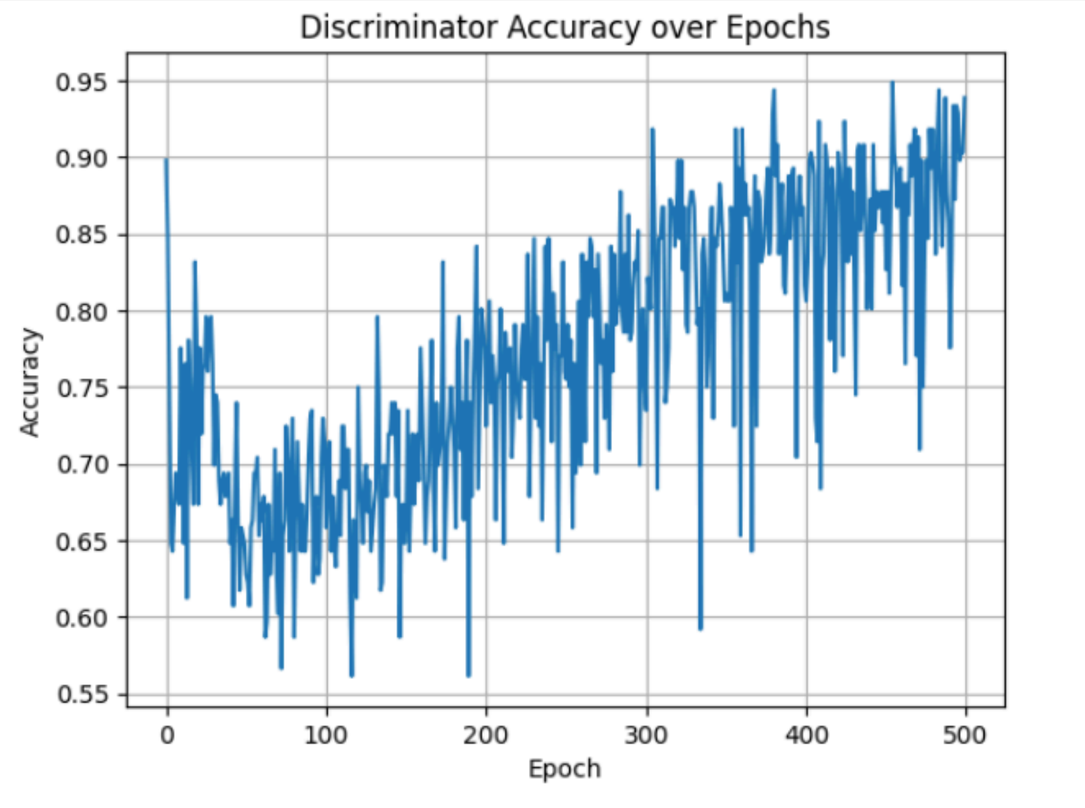


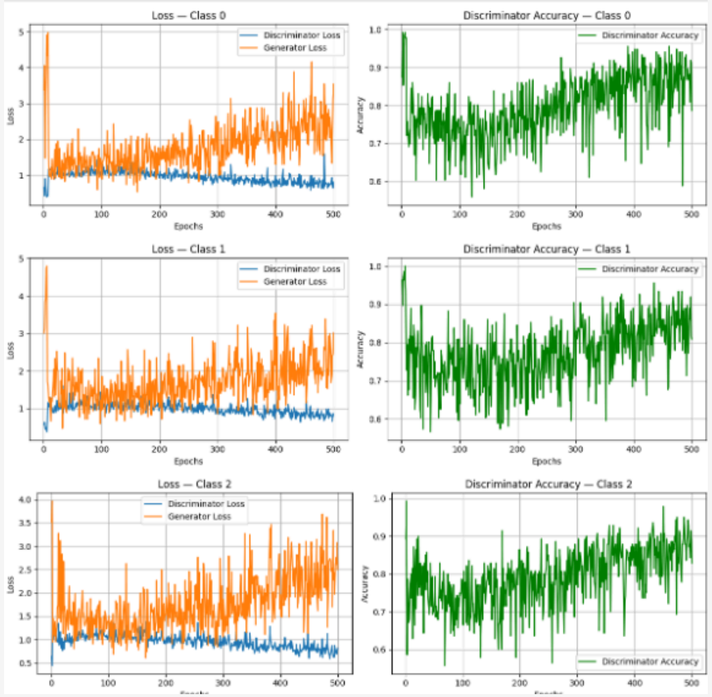
MULTI-DCGAN MODEL IMPROVEMENT

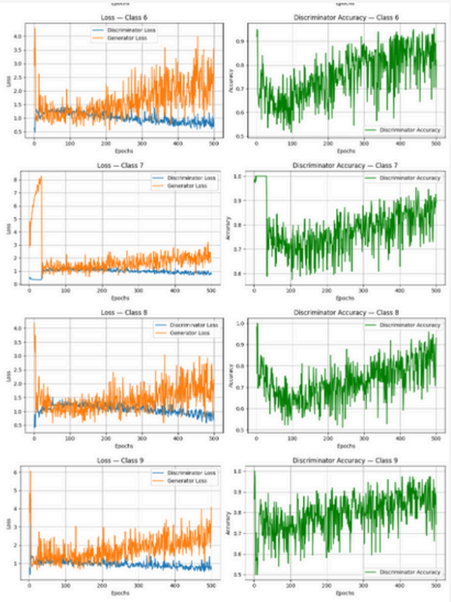
```
=== Generator Summary ===
Model: "model_178"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_147 (InputLayer)      [(None, 128)]             0

 sequential_146 (Sequential)  (None, 28, 28, 1)        2343553

=================================================================
Total params: 2,343,553
Trainable params: 2,318,017
Non-trainable params: 25,536
_________________________________________________________________

=== Discriminator Summary ===
Model: "model_179"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_148 (InputLayer)      [(None, 28, 28, 1)]       0

 sequential_147 (Sequential)  (None, 1)                624641

=================================================================
Total params: 624,641
Trainable params: 624,129
Non-trainable params: 512
```

Component	Baseline DCGAN	Improved DCGAN	Reason for Change
Generator – Dense+Reshape	7×7×128	7×7×256	More channels at low resolution increase capacity to model stroke details.
Generator – Upsampling	UpSampling2D + Conv2D ×2	Conv2DTranspose (stride=2) ×2	Learnable upsampling kernels, fewer artifacts.
Generator – Refinement	None	+ Conv2DTranspose(32/64, stride=1)	Sharpens edges and adds fine details at full resolution.
Generator – Norm/Act	BN after some convs, ReLU	BN after every block, LeakyReLU(0.2)	More consistent normalization, better gradient flow.
Generator – Output	Conv2D → tanh	Conv2D → tanh	Same output; range [-1, 1] for stable GAN training.
Discriminator – Blocks	2 conv blocks	3 conv blocks (extra stride=1 detail block)	Improves ability to inspect fine details without excessive downsampling.
Discriminator – Regularization	Dropout 0.2 / 0.3	Dropout 0.3 / 0.3 + BN in every block	Stronger regularization and stability.
Discriminator – Head	Flatten → Dense(1) → sigmoid	Flatten → Dense(64) → LeakyReLU → Dense(1) → sigmoid	Adds capacity and stabilizes gradients before final decision.
Optimizers	Adam(0.0002, β_1=0.5) for both G and D	Same settings	

MULTI-DCGAN MODEL IMPROVEMENT





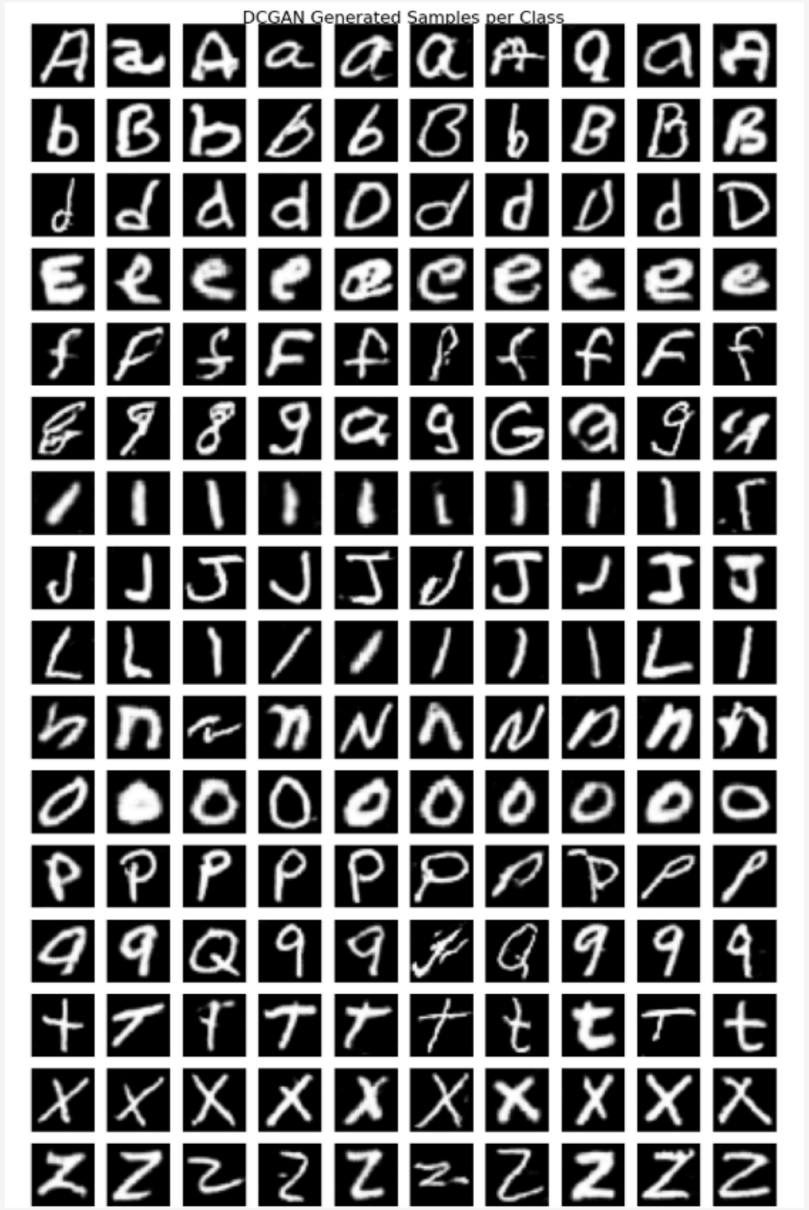


Discriminator Accuracy Observation

- Ideal GAN training: Discriminator accuracy should hover around 60–80%, allowing both networks to learn together.
- In this run, accuracy steadily increased to ~95%, indicating the discriminator became too strong.
- When the discriminator wins too often, the generator receives poor gradients, slowing its improvement.

Key takeaway: Training balance needs adjustment to keep accuracy in a competitive range.

MULTI-DCGAN MODEL IMPROVEMENT



MULTI-DCGAN MODEL IMPROVEMENT 2

Differences between previous and this model

Aspect	Version 1	Version 2
Final Layer Type	`Conv2D` (normal convolution)	`Conv2DTranspose` (transpose convolution)
Final Layer Kernel	3×3 kernel	2×2 kernel
Final Layer Stride	Default stride = 1 (Conv2D)	stride = 1
Final Layer Activation	Separate `Activation('tanh')` after Conv2D	`'tanh'` integrated inside final `Conv2DTranspose`
Upsampling Layers	3 × `Conv2DTranspose` + 1 Conv2D refinement layer	4 × `Conv2DTranspose` (last layer produces output)
Output Refinement	Final refinement via normal Conv2D	Refinement integrated into last `Conv2DTranspose`

Training Settings

Setting	Value / Choice	Why
Label Smoothing	Real labels = **0.9**, Fake = **0.0**	Prevents D overconfidence; gives G better gradients when D gets strong
G Optimizer	`Adam(lr=2e-4, beta_1=0.5)`	Slightly faster G to keep up with D; DCGAN-standard β_1 stabilizes momentum
D Optimizer	`Adam(lr=5e-5, beta_1=0.5)`	Slower D to avoid dominating (keeps D-acc ~60–80% "healthy" range)

```python
def discriminator_loss(self, real_output, fake_output):
    real_loss = self.loss_fn(tf.ones_like(real_output) * 0.9, real_output)  # label smoothing
    fake_loss = self.loss_fn(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss

dcgan.compile(
        g_optimizer=Adam(0.0002, beta_1=0.5),   # faster
        d_optimizer=Adam(0.00005, beta_1=0.5)   # slower
    )
```

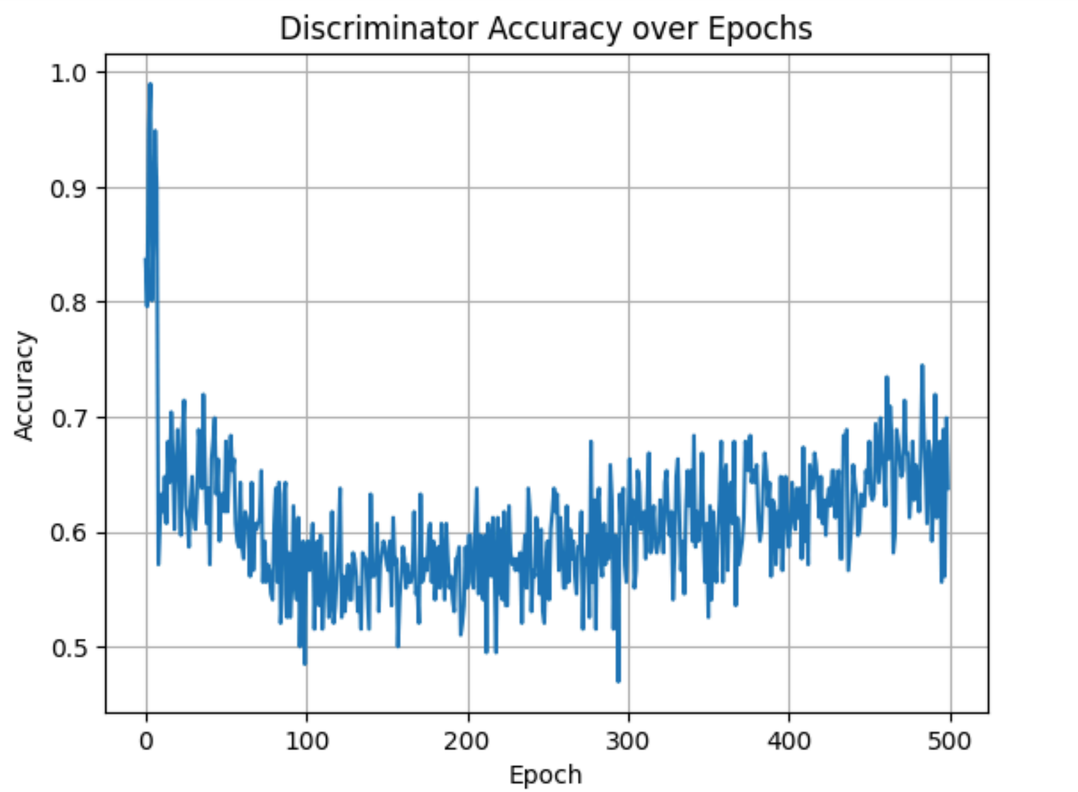
This stops the discriminator from assigning "perfect certainty" too quickly, which can make the generator's job impossible.

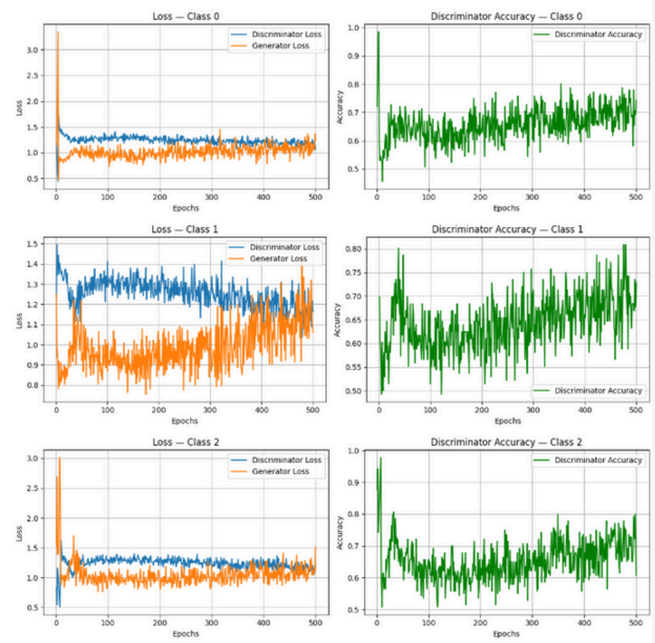
Parameter Choice Rationale

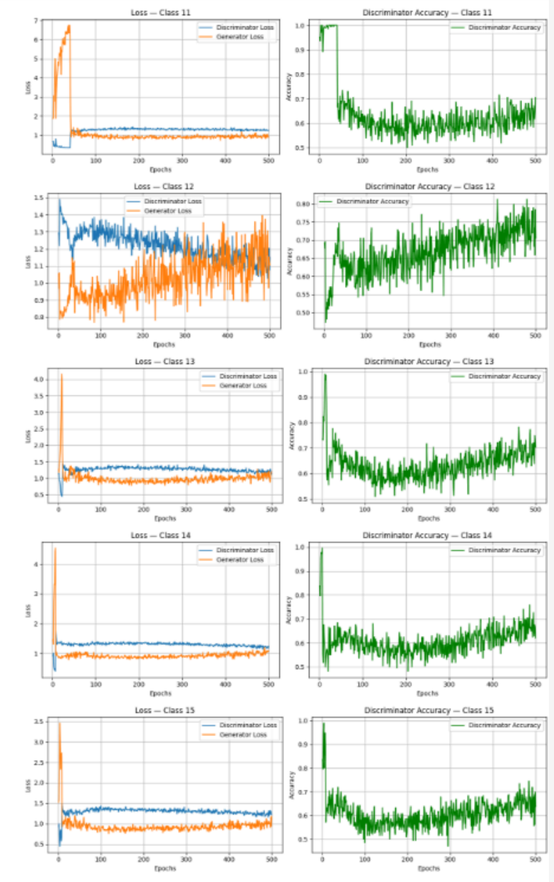
- Previous Issue:
  - Discriminator accuracy ~95% → too strong, weak gradients to G → slower improvement
- Adjustments:
  - Learning Rates: G = 2e-4, D = 5e-5 → boost G learning, slow D dominance
  - Label Smoothing: Real = 0.9, Fake = 0.0 → reduce D overconfidence, improve G gradient quality
  - Slight label noise → regularise and prevent D from memorising
  - Dropout + instance noise in D → reduce overfitting
  - Increased G capacity with refinement layer → better ability to challenge D
  - Batch size = 64 → balance stability and diversity

```
=== Generator Summary ===
Model: "model_358"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_329 (InputLayer)      [(None, 128)]             0

 sequential_328 (Sequential)  (None, 28, 28, 1)        2343393

=================================================================
Total params: 2,343,393
Trainable params: 2,317,857
Non-trainable params: 25,536
_________________________________________________________________

=== Discriminator Summary ===
Model: "model_359"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_330 (InputLayer)      [(None, 28, 28, 1)]       0

 sequential_329 (Sequential)  (None, 1)                624641

=================================================================
Total params: 624,641
Trainable params: 624,129
Non-trainable params: 512
```

MULTI-DCGAN MODEL IMPROVEMENT 2







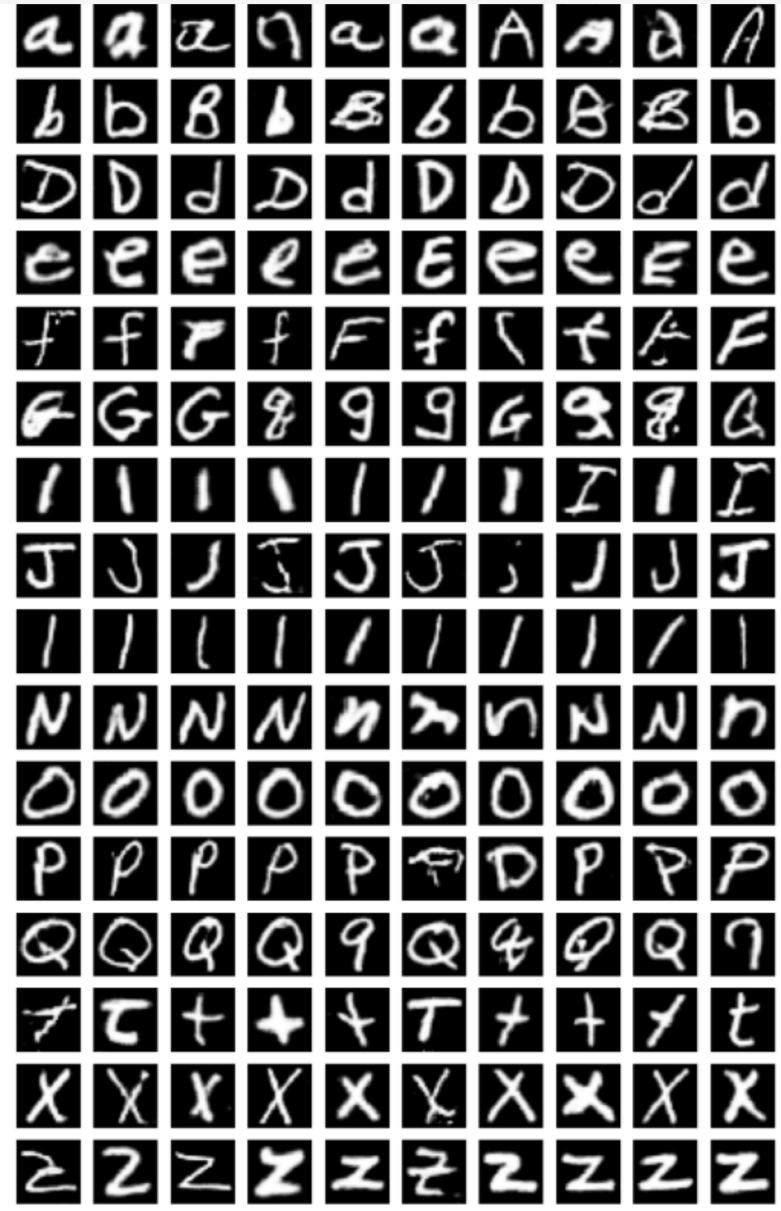
Before: D accuracy climbed to ~95% and stayed high → D overpowered G.

Now: Accuracy hovers mostly in the 0.55–0.7 range, which is within the ideal 60–80% band for balanced GAN training.

The lower, more stable D accuracy means:

- G receives stronger gradients → better learning
- Training is competitive rather than one-sided

MULTI-DCGAN MODEL IMPROVEMENT 2



- Clear letter shapes across all 16 classes, minimal “nonsense” samples.
- Consistent stroke style within each class, showing the generator learned stable patterns.
- Matches the improved discriminator accuracy curve balanced training allowed the generator to keep improving.

Key Success Factors:
Slower D, stronger G, regularisation (dropout + instance noise), and increased G capacity.

CGAN & CONDITIONAL DCGAN

Feature	CGAN	CDCGAN
Architecture	Dense + Conv mix	Fully convolutional
Image Quality	Basic	Sharper, more realistic
Training Stability	Moderate	Higher (with BN, deeper layers)
Computation	Lower	Higher
Use Case in Project	Quick baseline	Improved results

CGAN BASELINE MODEL

Layer (type)	Output Shape	Param #	Connected to
input_106 (InputLayer)	[(None, 1)]	0	[]
embedding_52 (Embedding)	(None, 1, 150)	2400	['input_106[0][0]']
input_105 (InputLayer)	[(None, 150)]	0	[]
flatten_78 (Flatten)	(None, 150)	0	['embedding_52[0][0]']
multiply_26 (Multiply)	(None, 150)	0	['input_105[0][0]', 'flatten_78[0][0]']
dense_90 (Dense)	(None, 256)	38656	['multiply_26[0][0]']
batch_normalization_101 (Batch Normalization)	(None, 256)	1024	['dense_90[0][0]']
leaky_re_lu_139 (LeakyReLU)	(None, 256)	0	['batch_normalization_101[0][0]']
dense_91 (Dense)	(None, 512)	131584	['leaky_re_lu_139[0][0]']
batch_normalization_102 (Batch Normalization)	(None, 512)	2048	['dense_91[0][0]']
leaky_re_lu_140 (LeakyReLU)	(None, 512)	0	['batch_normalization_102[0][0]']
dense_92 (Dense)	(None, 784)	402192	['leaky_re_lu_140[0][0]']
reshape_42 (Reshape)	(None, 28, 28, 1)	0	['dense_92[0][0]']

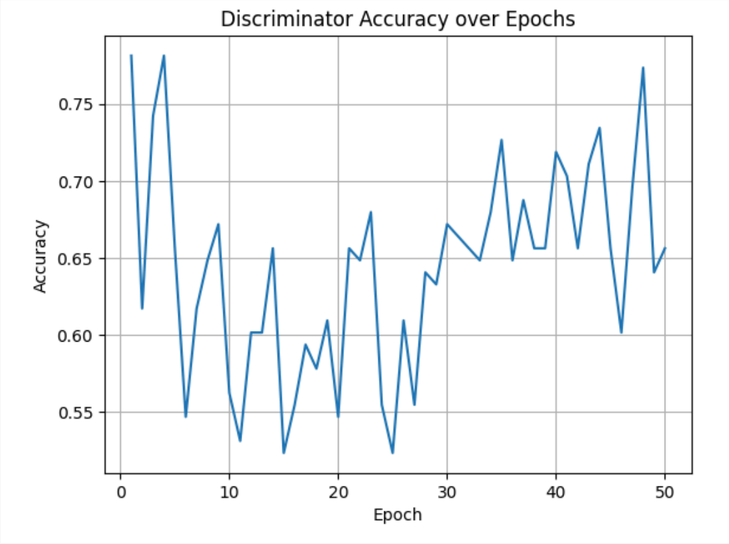
Total params: 577,904
Trainable params: 576,368
Non-trainable params: 1,536

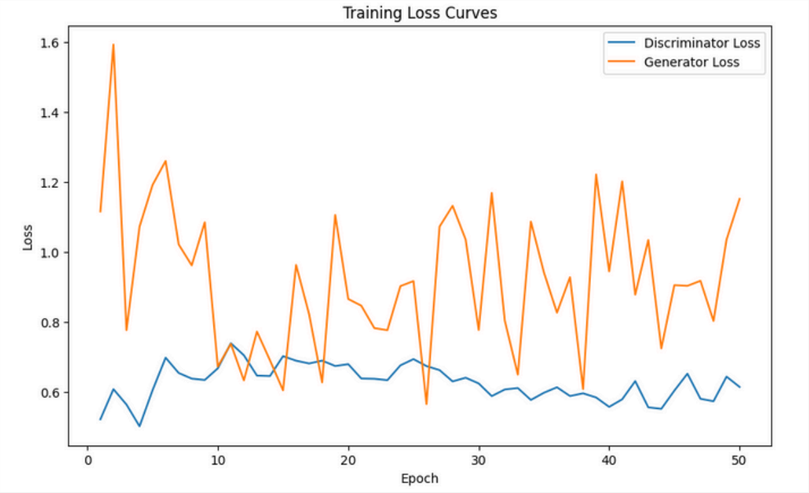
Total params: 577,904
Trainable params: 576,368
Non-trainable params: 1,536

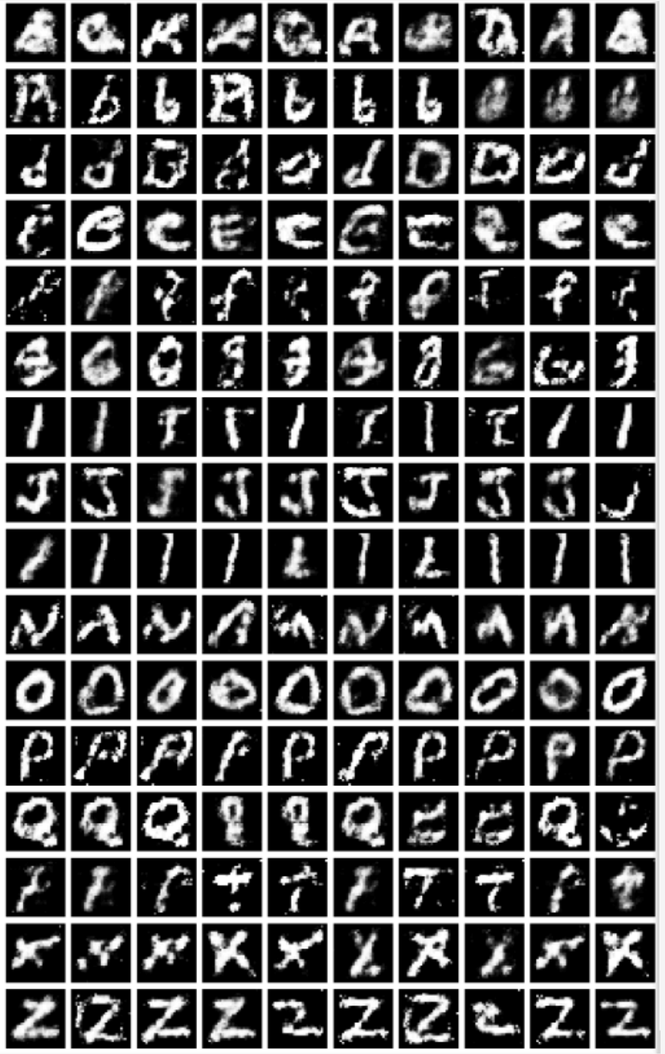
Model: "model_52"

Layer (type)	Output Shape	Param #	Connected to
input_108 (InputLayer)	[(None, 1)]	0	[]
input_107 (InputLayer)	[(None, 28, 28, 1)]	0	[]
embedding_53 (Embedding)	(None, 1, 784)	12544	['input_108[0][0]']
flatten_80 (Flatten)	(None, 784)	0	['input_107[0][0]']
flatten_79 (Flatten)	(None, 784)	0	['embedding_53[0][0]']
concatenate_26 (Concatenate)	(None, 1568)	0	['flatten_80[0][0]', 'flatten_79[0][0]']
dense_93 (Dense)	(None, 256)	401664	['concatenate_26[0][0]']
leaky_re_lu_141 (LeakyReLU)	(None, 256)	0	['dense_93[0][0]']
dense_94 (Dense)	(None, 1)	257	['leaky_re_lu_141[0][0]']

Total params: 414,465
Trainable params: 414,465
Non-trainable params: 0



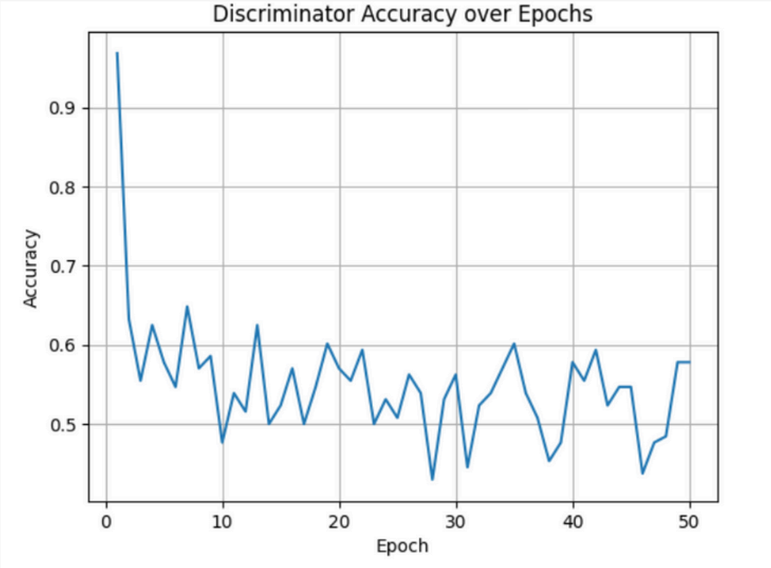




CDCGAN BASELINE MODEL

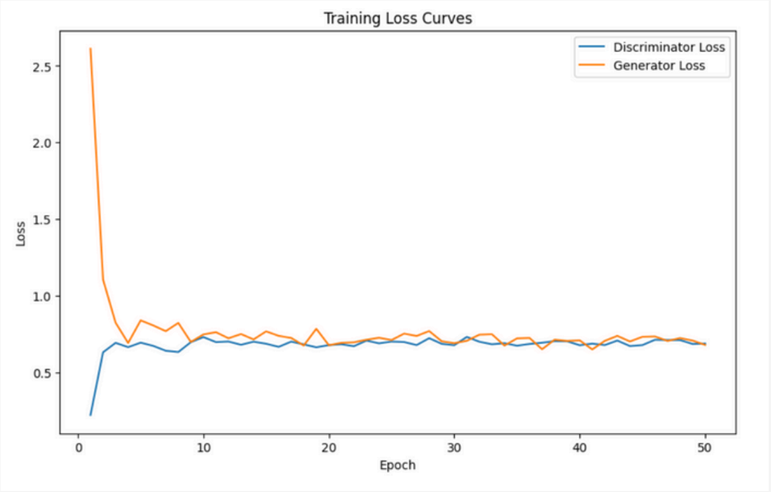
Layer (type)	Output Shape	Param #	Connected to
input_114 (InputLayer)	[(None, 1)]	0	[]
embedding_56 (Embedding)	(None, 1, 150)	2400	['input_114[0][0]']
input_113 (InputLayer)	[(None, 150)]	0	[]
flatten_84 (Flatten)	(None, 150)	0	['embedding_56[0][0]']
multiply_28 (Multiply)	(None, 150)	0	['input_113[0][0]', 'flatten_84[0][0]']
dense_97 (Dense)	(None, 12544)	1881600	['multiply_28[0][0]']
batch_normalization_107 (Batch Normalization)	(None, 12544)	50176	['dense_97[0][0]']
leaky_re_lu_147 (LeakyReLU)	(None, 12544)	0	['batch_normalization_107[0][0]']
reshape_45 (Reshape)	(None, 7, 7, 256)	0	['leaky_re_lu_147[0][0]']
conv2d_transpose_46 (Conv2DTranspose)	(None, 14, 14, 128)	524288	['reshape_45[0][0]']
batch_normalization_108 (Batch Normalization)	(None, 14, 14, 128)	512	['conv2d_transpose_46[0][0]']
leaky_re_lu_148 (LeakyReLU)	(None, 14, 14, 128)	0	['batch_normalization_108[0][0]']
conv2d_transpose_47 (Conv2DTranspose)	(None, 28, 28, 64)	131072	['leaky_re_lu_148[0][0]']
batch_normalization_109 (Batch Normalization)	(None, 28, 28, 64)	256	['conv2d_transpose_47[0][0]']
leaky_re_lu_149 (LeakyReLU)	(None, 28, 28, 64)	0	['batch_normalization_109[0][0]']
conv2d_60 (Conv2D)	(None, 28, 28, 1)	3137	['leaky_re_lu_149[0][0]']

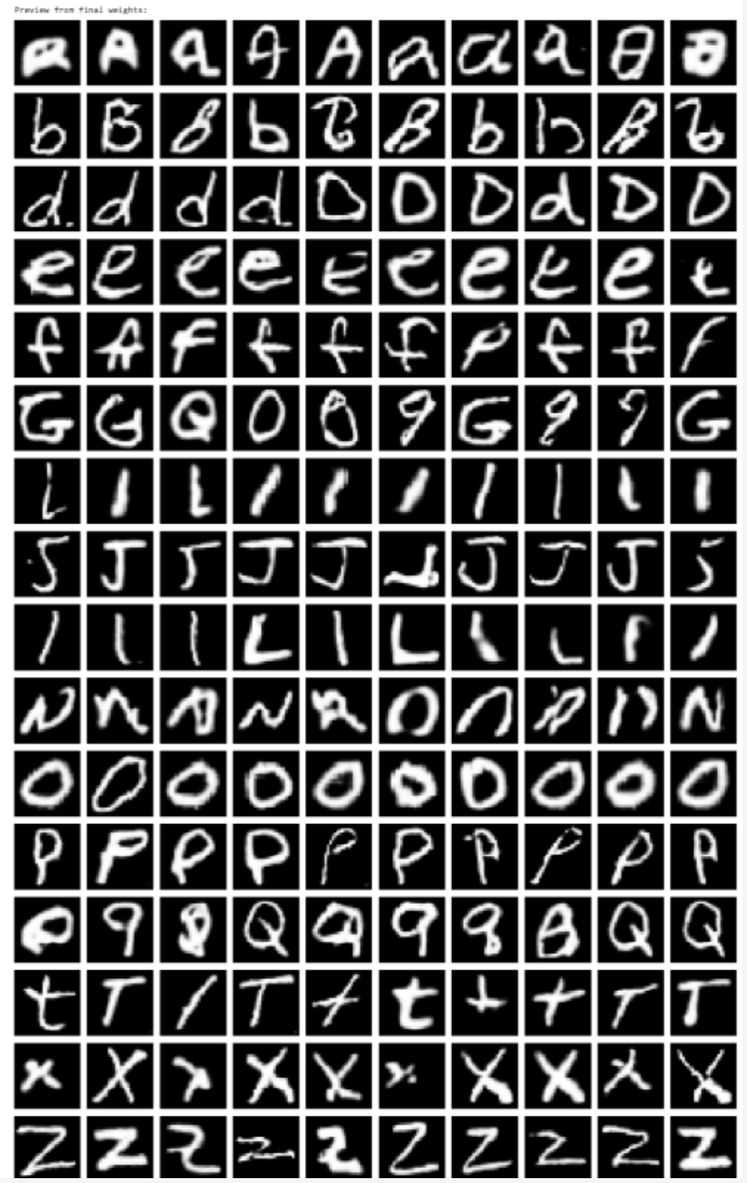
Total params: 2,593,441
Trainable params: 2,567,969
Non-trainable params: 25,472



Layer (type)	Output Shape	Param #	Connected to
input_116 (InputLayer)	[(None, 1)]	0	[]
embedding_57 (Embedding)	(None, 1, 784)	12544	['input_116[0][0]']
flatten_85 (Flatten)	(None, 784)	0	['embedding_57[0][0]']
input_115 (InputLayer)	[(None, 28, 28, 1)]	0	[]
reshape_46 (Reshape)	(None, 28, 28, 1)	0	['flatten_85[0][0]']
concatenate_28 (Concatenate)	(None, 28, 28, 2)	0	['input_115[0][0]', 'reshape_46[0][0]']
conv2d_61 (Conv2D)	(None, 14, 14, 64)	2112	['concatenate_28[0][0]']
leaky_re_lu_150 (LeakyReLU)	(None, 14, 14, 64)	0	['conv2d_61[0][0]']
dropout_57 (Dropout)	(None, 14, 14, 64)	0	['leaky_re_lu_150[0][0]']
conv2d_62 (Conv2D)	(None, 7, 7, 128)	131200	['dropout_57[0][0]']
batch_normalization_110 (Batch Normalization)	(None, 7, 7, 128)	512	['conv2d_62[0][0]']
leaky_re_lu_151 (LeakyReLU)	(None, 7, 7, 128)	0	['batch_normalization_110[0][0]']
dropout_58 (Dropout)	(None, 7, 7, 128)	0	['leaky_re_lu_151[0][0]']
flatten_86 (Flatten)	(None, 6272)	0	['dropout_58[0][0]']
dense_98 (Dense)	(None, 1)	6273	['flatten_86[0][0]']

Total params: 152,641
Trainable params: 152,385
Non-trainable params: 256

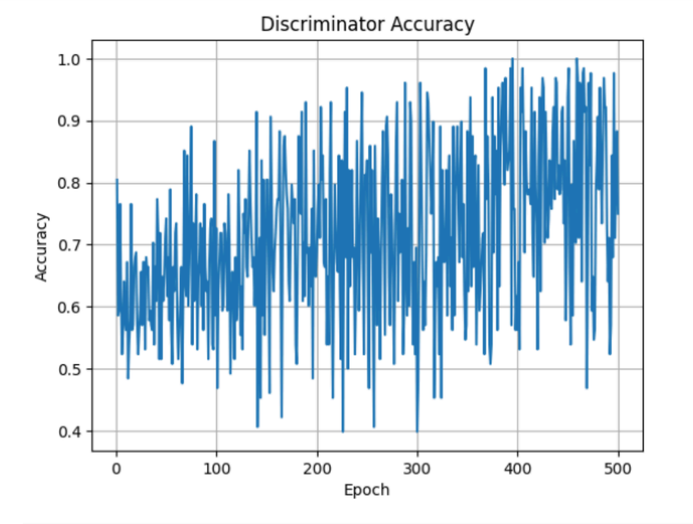




CDCGAN MODEL IMPROVEMENT 1

Layer (type)	Output Shape	Param #	Connected to
input_82 (InputLayer)	[(None, 1)]	0	[]
embedding_40 (Embedding)	(None, 1, 150)	2400	['input_82[0][0]']
input_81 (InputLayer)	[(None, 150)]	0	[]
flatten_60 (Flatten)	(None, 150)	0	['embedding_40[0][0]']
multiply_20 (Multiply)	(None, 150)	0	['input_81[0][0]', 'flatten_60[0][0]']
dense_56 (Dense)	(None, 12544)	1881600	['multiply_20[0][0]']
batch_normalization_84 (BatchNormalization)	(None, 12544)	50176	['dense_56[0][0]']
leaky_re_lu_111 (LeakyReLU)	(None, 12544)	0	['batch_normalization_84[0][0]']
reshape_35 (Reshape)	(None, 7, 7, 256)	0	['leaky_re_lu_111[0][0]']
conv2d_transpose_42 (Conv2DTranspose)	(None, 14, 14, 128)	524288	['reshape_35[0][0]']
batch_normalization_85 (BatchNormalization)	(None, 14, 14, 128)	512	['conv2d_transpose_42[0][0]']
leaky_re_lu_112 (LeakyReLU)	(None, 14, 14, 128)	0	['batch_normalization_85[0][0]']
conv2d_transpose_43 (Conv2DTranspose)	(None, 28, 28, 64)	131072	['leaky_re_lu_112[0][0]']
batch_normalization_86 (BatchNormalization)	(None, 28, 28, 64)	256	['conv2d_transpose_43[0][0]']
leaky_re_lu_113 (LeakyReLU)	(None, 28, 28, 64)	0	['batch_normalization_86[0][0]']
conv2d_53 (Conv2D)	(None, 28, 28, 1)	577	['leaky_re_lu_113[0][0]']

Total params: 2,590,881
Trainable params: 2,565,409
Non-trainable params: 25,472

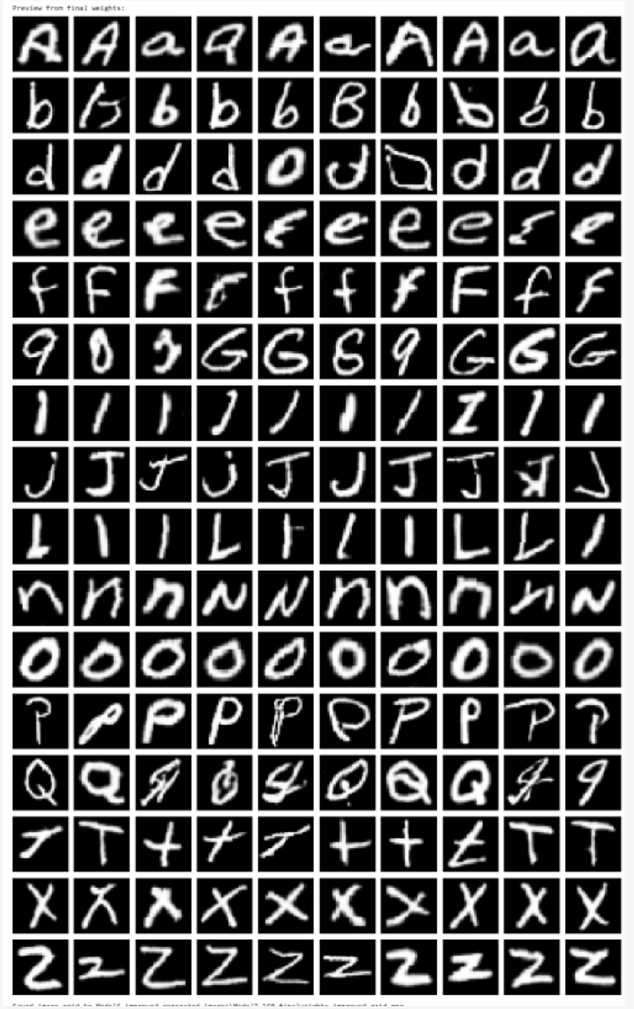


Observations

- D accuracy drifts upward, stays > 0.8–1.0 in late epochs.
- G loss (orange) explodes over time, while D loss (blue) remains low & flat.

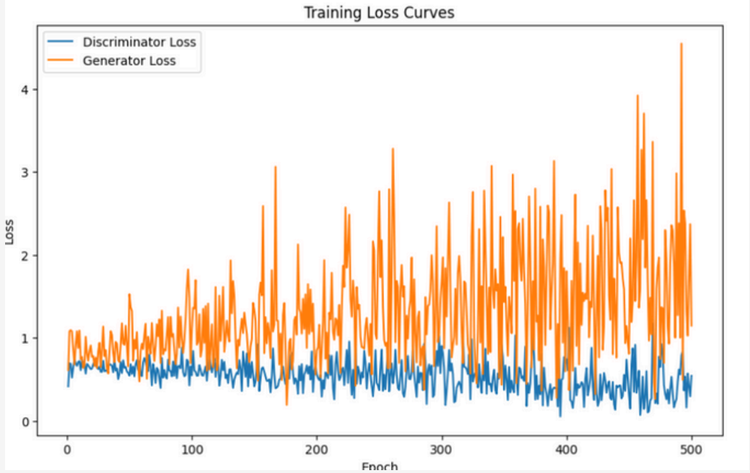
Interpretation

- Discriminator becomes too strong.
- D learns to easily separate real vs fake → high confidence, low loss.
- Generator struggles to receive useful gradient signals.



Layer (type)	Output Shape	Param #	Connected to
input_84 (InputLayer)	[(None, 1)]	0	[]
embedding_41 (Embedding)	(None, 1, 784)	12544	['input_84[0][0]']
flatten_61 (Flatten)	(None, 784)	0	['embedding_41[0][0]']
input_83 (InputLayer)	[(None, 28, 28, 1)]	0	[]
reshape_36 (Reshape)	(None, 28, 28, 1)	0	['flatten_61[0][0]']
concatenate_20 (Concatenate)	(None, 28, 28, 2)	0	['input_83[0][0]', 'reshape_36[0][0]']
conv2d_54 (Conv2D)	(None, 14, 14, 64)	2112	['concatenate_20[0][0]']
leaky_re_lu_114 (LeakyReLU)	(None, 14, 14, 64)	0	['conv2d_54[0][0]']
dropout_44 (Dropout)	(None, 14, 14, 64)	0	['leaky_re_lu_114[0][0]']
conv2d_55 (Conv2D)	(None, 7, 7, 128)	131200	['dropout_44[0][0]']
batch_normalization_87 (BatchNormalization)	(None, 7, 7, 128)	512	['conv2d_55[0][0]']
leaky_re_lu_115 (LeakyReLU)	(None, 7, 7, 128)	0	['batch_normalization_87[0][0]']
dropout_45 (Dropout)	(None, 7, 7, 128)	0	['leaky_re_lu_115[0][0]']
conv2d_56 (Conv2D)	(None, 7, 7, 256)	295168	['dropout_45[0][0]']
batch_normalization_88 (BatchNormalization)	(None, 7, 7, 256)	1024	['conv2d_56[0][0]']
leaky_re_lu_116 (LeakyReLU)	(None, 7, 7, 256)	0	['batch_normalization_88[0][0]']
dropout_46 (Dropout)	(None, 7, 7, 256)	0	['leaky_re_lu_116[0][0]']
flatten_62 (Flatten)	(None, 12544)	0	['dropout_46[0][0]']
dense_57 (Dense)	(None, 1)	12545	['flatten_62[0][0]']

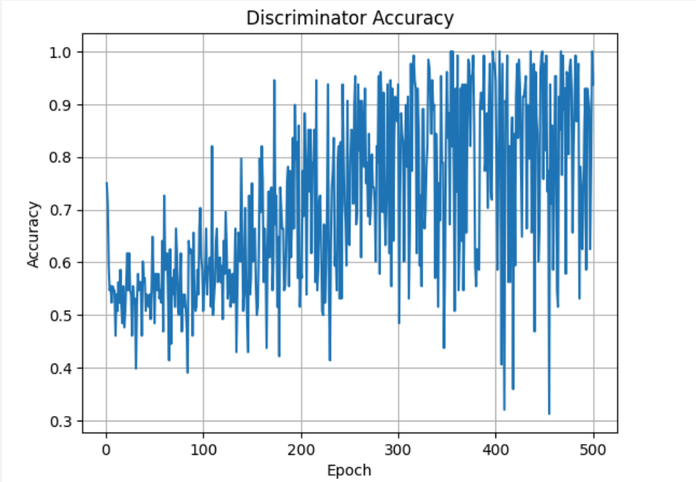
Total params: 455,105
Trainable params: 454,337
Non-trainable params: 768



CDCGAN MODEL IMPROVEMENT 2

Why the Change? (Based on Previous Results)

- Discriminator accuracy in V1 drifted >0.8–1.0 → D too strong.
- Needed to boost G capacity and reduce D's dominance for balanced training.

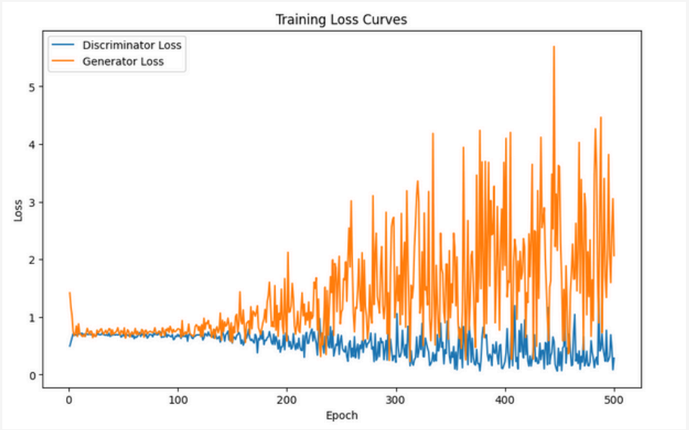


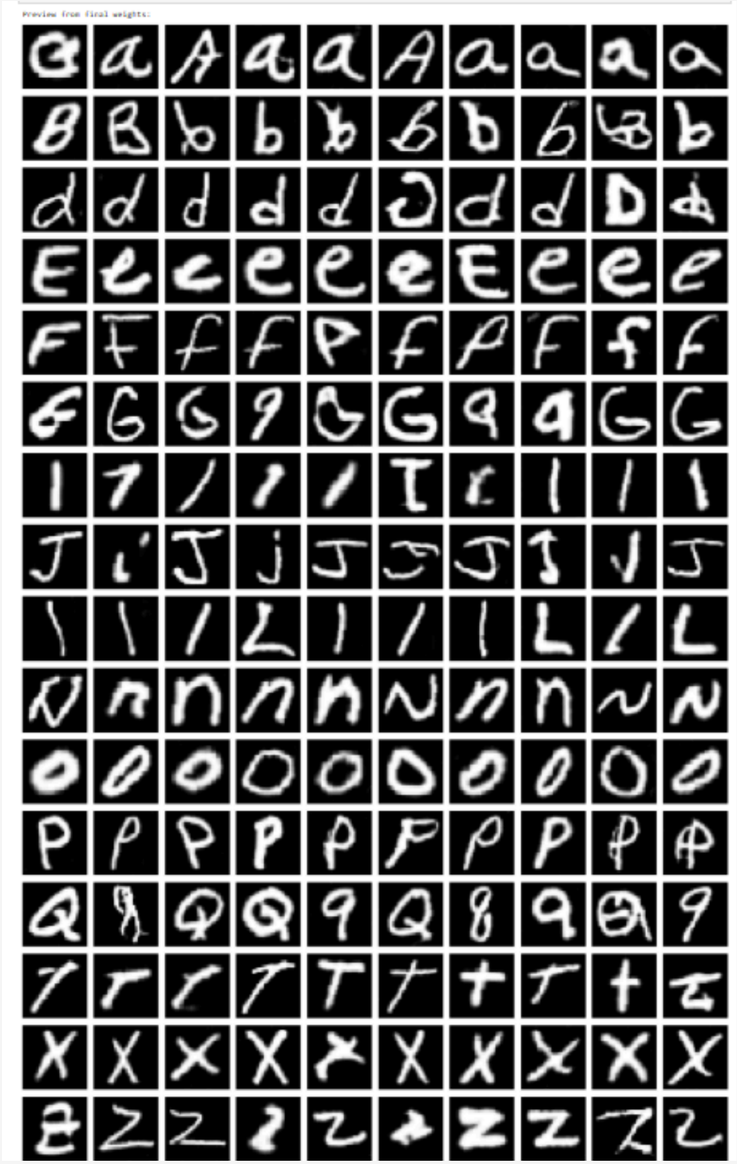
Generator (G)

- Added: Conv2DTranspose(256, stride=1) at 7×7 before first upsample.
- Purpose: Early feature refinement → richer details, sharper strokes.

Discriminator (D)

- Removed: 3rd Conv block (256 filters).
- Purpose: Prevent D from overpowering G → better gradient flow.





CDCGAN MODEL IMPROVEMENT 3

Key Changes

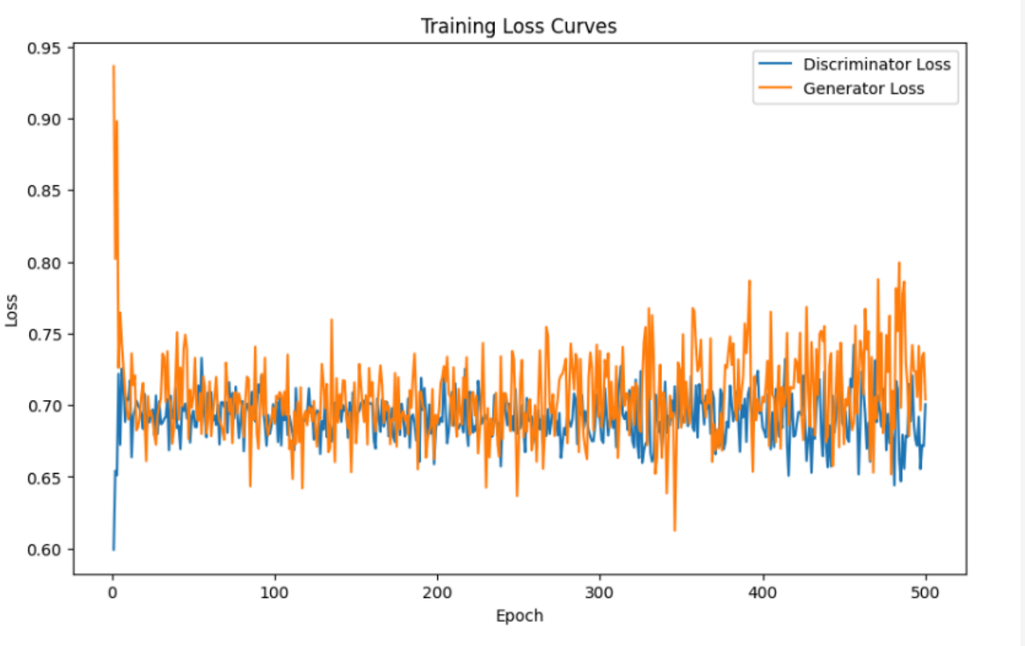
- Generator
  - Removed refine layer at 7×7 (Conv2DTranspose 256, stride=1).
  - Direct upsampling: 128 (→14×14) → 64 (→28×28).
  - Changed output layer kernel size from 3×3 to 7×7 for smoother outputs.
- Discriminator
  - Kept same 2-block design: Conv(64) → Conv(128) → Flatten → Dense(1).
  - Maintains dropout.
- Optimizers
  - Adam (β_1=0.5) for both G & D.
  - G LR: 0.0002; D LR: 0.0001 for training stability.
  - Lower D LR + simpler G/D pairing reduces discriminator dominance, giving G more opportunity to improve image quality.

The combination of a simpler generator, a stable yet weaker discriminator, and a lower D learning rate is designed to maintain balanced adversarial dynamics.

This gives the generator more opportunity to improve image quality while avoiding instability or early domination by either network.

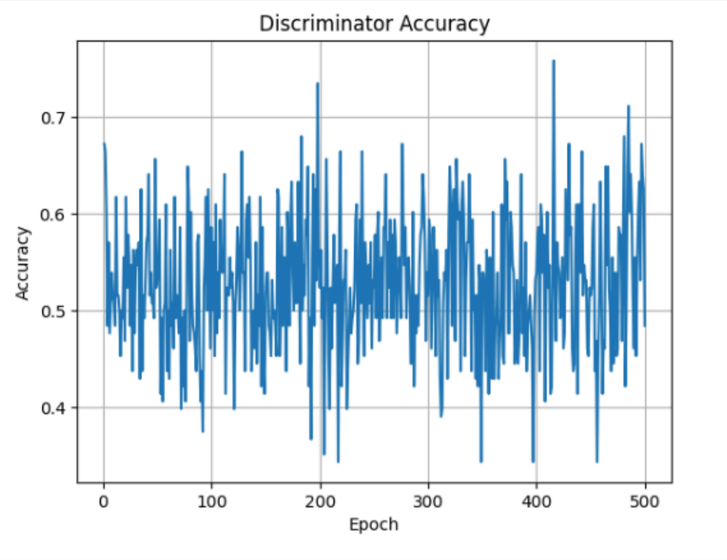
Layer (type)	Output Shape	Param #	Connected to
input_62 (InputLayer)	[(None, 1)]	0	[]
embedding_30 (Embedding)	(None, 1, 150)	2400	['input_62[0][0]']
input_61 (InputLayer)	[(None, 150)]	0	[]
flatten_33 (Flatten)	(None, 150)	0	['embedding_30[0][0]']
multiply_12 (Multiply)	(None, 150)	0	['input_61[0][0]', 'flatten_33[0][0]']
dense_44 (Dense)	(None, 12544)	1881600	['multiply_12[0][0]']
batch_normalization_71 (BatchNormalization)	(None, 12544)	50176	['dense_44[0][0]']
leaky_re_lu_69 (LeakyReLU)	(None, 12544)	0	['batch_normalization_71[0][0]']
reshape_27 (Reshape)	(None, 7, 7, 256)	0	['leaky_re_lu_69[0][0]']
conv2d_transpose_36 (Conv2DTranspose)	(None, 14, 14, 128)	524288	['reshape_27[0][0]']
batch_normalization_72 (BatchNormalization)	(None, 14, 14, 128)	512	['conv2d_transpose_36[0][0]']
leaky_re_lu_70 (LeakyReLU)	(None, 14, 14, 128)	0	['batch_normalization_72[0][0]']
conv2d_transpose_37 (Conv2DTranspose)	(None, 28, 28, 64)	131072	['leaky_re_lu_70[0][0]']
batch_normalization_73 (BatchNormalization)	(None, 28, 28, 64)	256	['conv2d_transpose_37[0][0]']
leaky_re_lu_71 (LeakyReLU)	(None, 28, 28, 64)	0	['batch_normalization_73[0][0]']
conv2d_38 (Conv2D)	(None, 28, 28, 1)	3137	['leaky_re_lu_71[0][0]']

Total params: 2,593,441
Trainable params: 2,567,969
Non-trainable params: 25,472

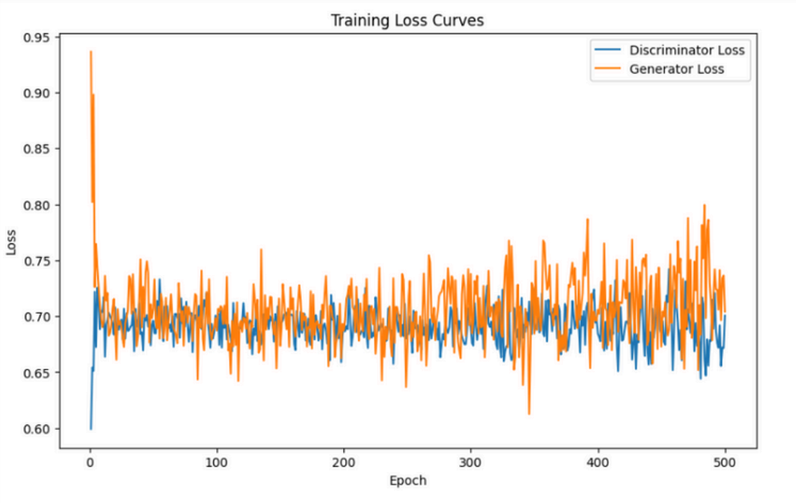


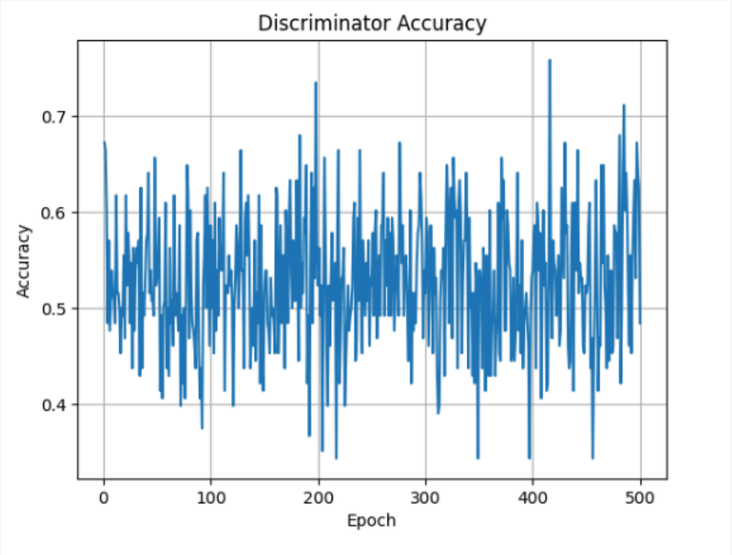
Layer (type)	Output Shape	Param #	Connected to
input_64 (InputLayer)	[(None, 1)]	0	[]
embedding_31 (Embedding)	(None, 1, 784)	12544	['input_64[0][0]']
flatten_34 (Flatten)	(None, 784)	0	['embedding_31[0][0]']
input_63 (InputLayer)	[(None, 28, 28, 1)]	0	[]
reshape_28 (Reshape)	(None, 28, 28, 1)	0	['flatten_34[0][0]']
concatenate_18 (Concatenate)	(None, 28, 28, 2)	0	['input_63[0][0]', 'reshape_28[0][0]']
conv2d_39 (Conv2D)	(None, 14, 14, 64)	2112	['concatenate_18[0][0]']
leaky_re_lu_72 (LeakyReLU)	(None, 14, 14, 64)	0	['conv2d_39[0][0]']
dropout_30 (Dropout)	(None, 14, 14, 64)	0	['leaky_re_lu_72[0][0]']
conv2d_40 (Conv2D)	(None, 7, 7, 128)	131200	['dropout_30[0][0]']
batch_normalization_74 (BatchNormalization)	(None, 7, 7, 128)	512	['conv2d_40[0][0]']
leaky_re_lu_73 (LeakyReLU)	(None, 7, 7, 128)	0	['batch_normalization_74[0][0]']
dropout_31 (Dropout)	(None, 7, 7, 128)	0	['leaky_re_lu_73[0][0]']
flatten_35 (Flatten)	(None, 6272)	0	['dropout_31[0][0]']
dense_45 (Dense)	(None, 1)	6273	['flatten_35[0][0]']

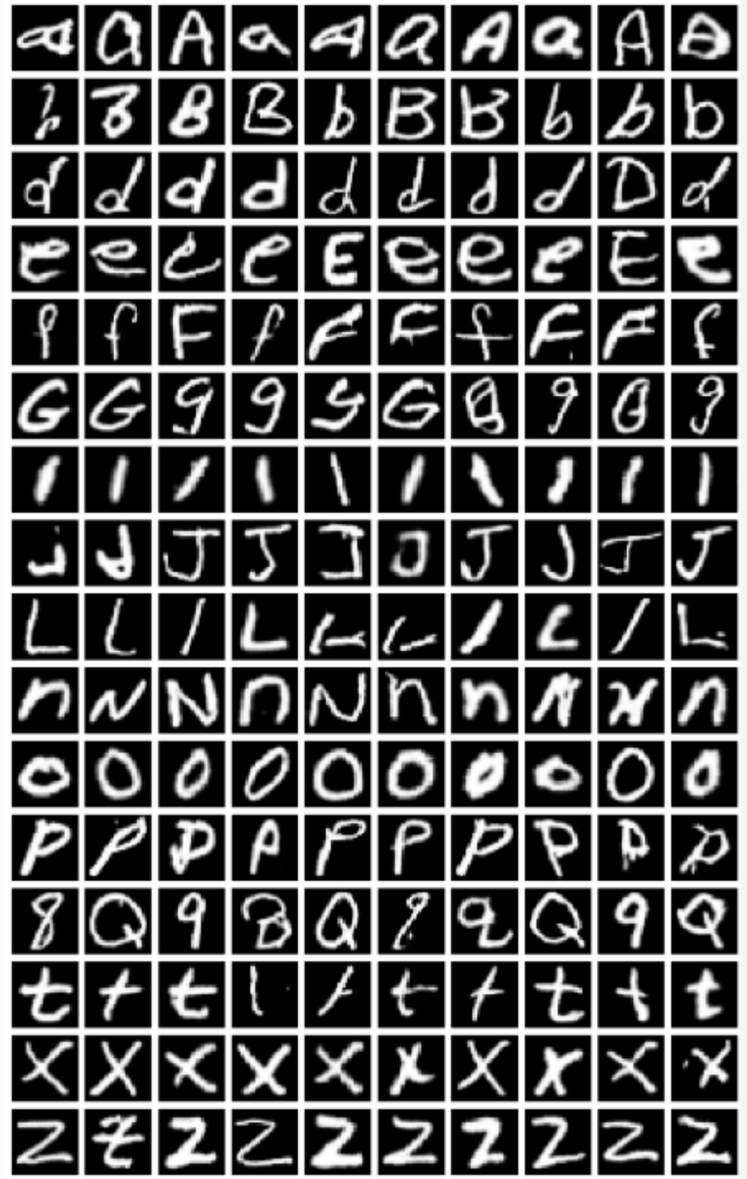
Total params: 152,641
Trainable params: 152,385
Non-trainable params: 256



CDCGAN MODEL IMPROVEMENT 3







Observations (From Plots)

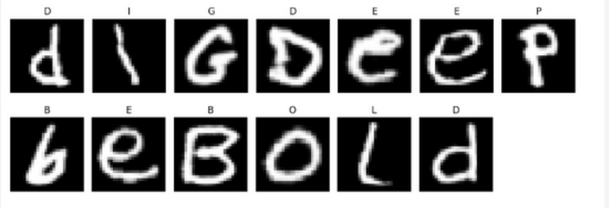
- Loss curves: Generator and Discriminator losses remain close and stable (~0.65–0.75) across 500 epochs
- D accuracy: Oscillates mainly between 0.45–0.7, within the healthy range for balanced learning.

What Changed

- Increased generator capacity (7×7×256 trunk, larger final 7×7 conv).
- Added dropout in D and label smoothing (real=0.9) for regularisation.
- Learning rates: G=2e-4, D=5e-5 → slowed D, boosted G learning.

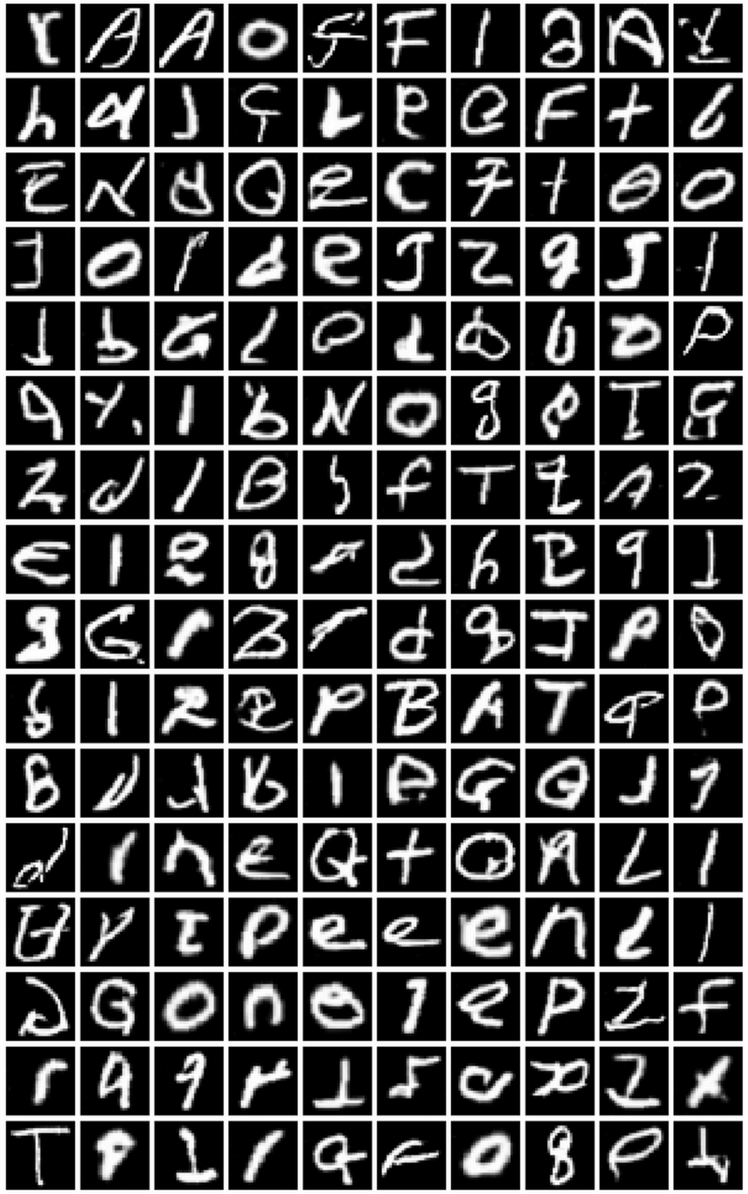
Result

- Balanced training → both networks improve together.
- Sharper outputs with fewer “nonsense” samples.

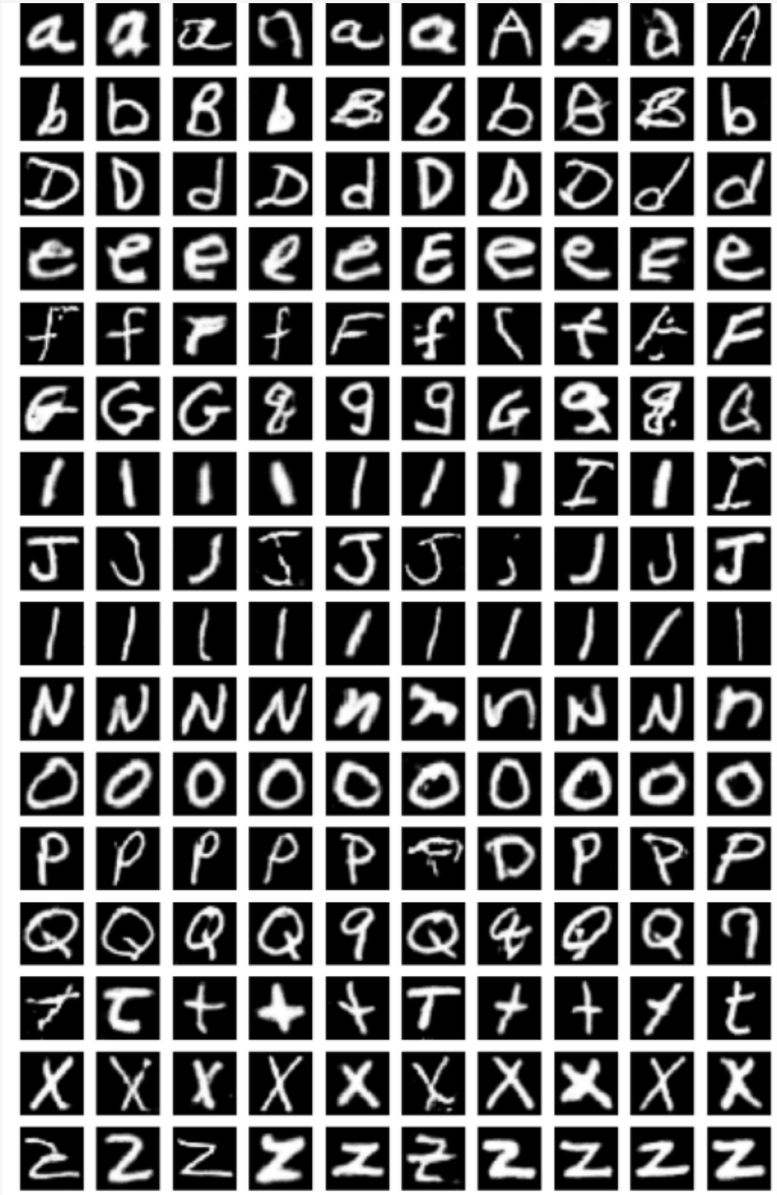


After Model Improvement....

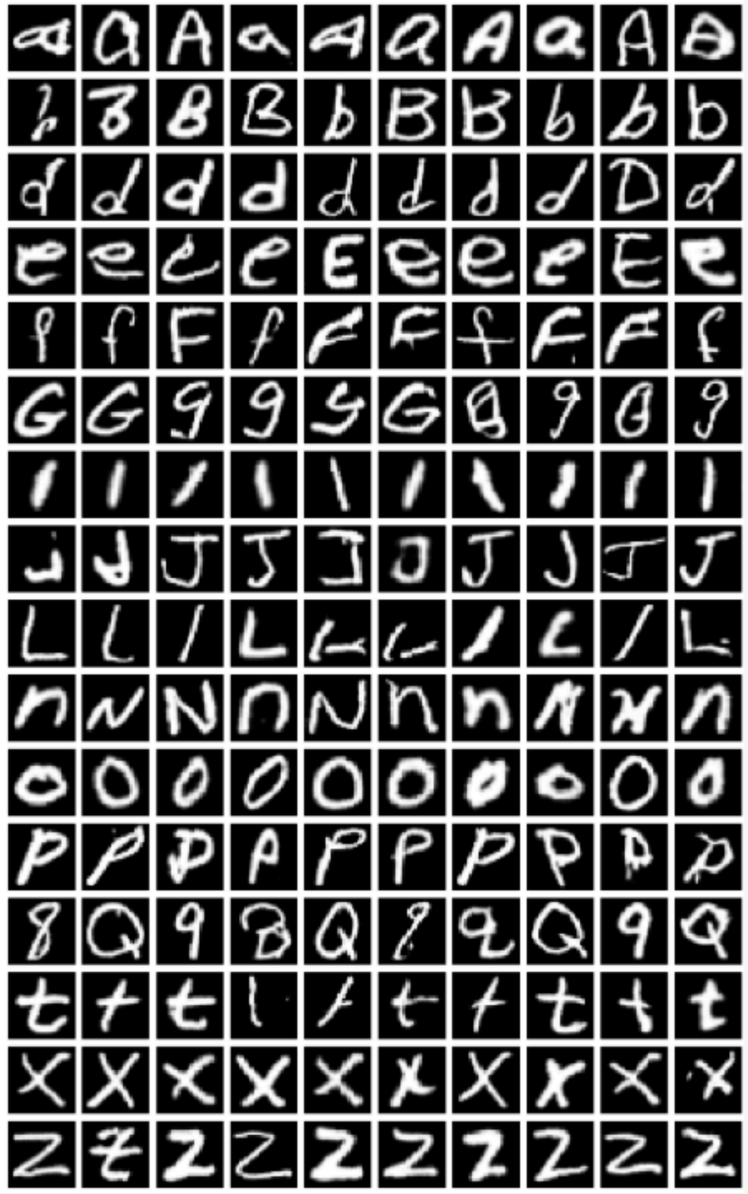
Unconditioned DCGAN



Multiple DCGANs



CDCGAN



CONCLUSION

	Key Strengths	Limitations	Performance
DCGAN (Baseline)	• Simple to implement. • Learns general letter features.	• No class control. • Output diversity but random classes.	• Generates plausible EMNIST letters, but cannot target specific classes.
Multi-DCGAN	• High per-class quality (specialized training). • Models focus only on one distribution.	• Very long total training time. • Large storage (many weights).	• Sharp, consistent letters per class, but training/storage heavy.
CGAN (Conditional)	• Generates any class from one model. • Efficient storage and faster training than Multi-DCGAN.	• More complex training (must learn label conditioned features).	• Quality close to Multi-DCGAN if tuned well, with slightly faster training.