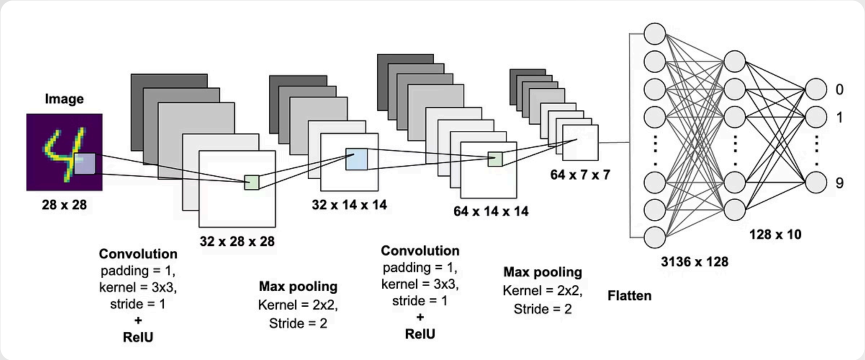

Supervised Learning CNN Project Overview

This project develops a CNN model for MNIST digit classification.

We optimized architecture, training parameters, and regularization for best accuracy.

M **by Moaz Gehad**

Made with GAMMA

Data Preprocessing Steps

- **Normalization:** Scale pixel values to a 0-1 range for model stability. Large input values would cause unstable gradients and poor training behavior.
- **Reshaping:** Adjust input images into the 3D format expected by CNN layers.
- **Encoding Target:** Convert digit labels into one-hot vectors for classification.

Normalize to range [0, 1]

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Reshape for CNN

```
x_train_cnn = x_train.reshape(len(x_train), 28, 28, 1)
x_test_cnn = x_test.reshape(len(x_test), 28, 28, 1)

print(x_train.shape, x_test.shape, x_train_cnn.shape, x_test_cnn.shape)
```

```
(60000, 28, 28) (10000, 28, 28) (60000, 28, 28, 1) (10000, 28, 28, 1)
```

One-hot encode labels for ANN/CNN

```
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

print(y_train_cat.shape, y_test_cat.shape)
```

Made with GAMMA

```
def build_cnn():
    model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)), MaxPooling2D(pool_size=(2,2),strides=(2,2)),
        Conv2D(64, kernel_size=(3, 3), activation='relu'), Flatten(), Dense(128, activation='relu'), Dense(10, activation='softmax') ])
    optimizer = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
cnn_model = build_cnn()
history = cnn_model.fit(x_train_cnn, y_train_cat, epochs=5, batch_size=128, validation_split=0.1)
```

Initial CNN Model Architecture

Layers

- Conv2D(32) + MaxPooling
- Conv2D(64)
- Flatten
- Dense(128)
- Dense(10) Softmax

Training

5 epochs, SGD optimizer (lr=0.01, momentum=0.9), batch size 128

Purpose

Baseline to tune depth, neurons, batch size, and regularization

Made with GAMMA

Epochs Impact on Accuracy

Metric	5 Epochs	10 Epochs	15 Epochs	20 Epochs
Final Accuracy	98.22%	98.85%	98.76%	99.04%
Accuracy (Epochs 1-5)	0.89, 0.96, 0.97, 0.98, 0.98	0.88, 0.97, 0.97, 0.98, 0.98	0.88, 0.96, 0.97, 0.98, 0.98	0.87, 0.96, 0.97, 0.98, 0.98
Avg. Train Time	-0.01 s	-0.01 s	-0.00 s	-0.00 s
Average Test Time	0.72 s	0.73 s	0.48 s	0.72 s
Trainable Parameters	1,011,466	1,011,466	1,011,466	1,011,466
Total Parameters	2,022,934	2,022,934	2,022,934	2,022,934
LR	0.01	0.01	0.01	0.01
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

# Epochs	Code
5	cnn_model.fit(x_train_cnn, y_train_cat, epochs=5, batch_size=128, validation_split=0.1)
10	cnn_model.fit(x_train_cnn, y_train_cat, epochs=10, batch_size=128, validation_split=0.1)
15	cnn_model.fit(x_train_cnn, y_train_cat, epochs=15, batch_size=128, validation_split=0.1)
20	cnn_model.fit(x_train_cnn, y_train_cat, epochs=20, batch_size=128, validation_split=0.1)

1 **5 to 20 Epochs**

Accuracy improves from 98.22% to 99.04%

2 **Optimal Epochs**

18-20 epochs balance learning and generalization

3 **Training Choice**

Selected 18 epochs for consistent high accuracy

Made with GAMMA

Batch Size Effects

Batch Size 32

Highest accuracy: 99.16%

Fast training and testing times

Batch Size 128

Lowest accuracy: 98.85%

Slower test time

Made with GAMMA

Learning Rate Comparison

Best LR

0.01 achieves 99.24% accuracy

Poor LR

0.1 yields only 90.28% accuracy

Other LRs

0.05 and 0.001 perform well but less than 0.01

Made with GAMMA

Metric	relu	sigmoid	tanh	softplus
Final Accuracy	99.17 %	98.15 %	98.8 %	97.81 %
Accuracy (Epochs 1–5)	[0.938, 0.983, 0.988, 0.991, 0.994]	[0.105, 0.106, 0.548, 0.914, 0.938]	[0.934, 0.977, 0.984, 0.988, 0.991]	[0.269, 0.941, 0.968, 0.975, 0.980]
Average Train Time per Epoch	0.00s	0.13s	0.00 s	0.01 s
Average Test Time	0.52 s	0.61 s	0.63 s	0.73 s

Activation Functions Tested

ReLU

Highest accuracy: 99.17%

Fastest training time

Sigmoid

Lowest accuracy: 98.15%

Slower training

Tanh & Softplus

Moderate accuracy and speed

Made with GAMMA

Convolutional Layer Variations

- 2 Conv Layers (32-64 filters) gave best accuracy: 99.11%
- 1, 3, and 2 Conv Layers (16-32) performed slightly lower
- Training time and parameters balanced at 2 Conv Layers 32-64

Comparison Table for different convolution layers

Metric	1 Conv Layer	2 Conv Layers 32-64	3 Conv Layers	2 Conv Layers 16-32
Final Accuracy	98.83 %	99.11 %	99.02 %	99.09 %
Accuracy (Epochs 1–5)	[0.925, 0.975, 0.984, 0.988, 0.992]	[0.942, 0.982, 0.988, 0.992, 0.994]	[0.945, 0.984, 0.99, 0.992, 0.995]	[0.939, 0.982, 0.988, 0.992, 0.994]
Average Train Time per Epoch	0.00s	0.13s	0.00 s	0.00s
Average Test Time	0.73 s	0.51 s	0.79 s	0.73 s
Total Parameters	1,387,926	2,022,934	1,441,430	1,003,670
Trainable Parameters	693,962	1,011,466	720,714	501,834
# Epochs	18	18	18	18
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

Conv Layers	Code
1 Conv Layer	Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1))
2 Conv Layers	Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)) Conv2D(64, kernel_size=(3, 3), activation='relu')
3 Conv Layers	Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)), Conv2D(64, kernel_size=(3, 3), activation='relu') Conv2D(64, kernel_size=(3, 3), activation='relu')
2 Conv Layers 16-32	Conv2D(16, (3,3), activation='relu', input_shape=(28,28,1)) Conv2D(32, kernel_size=(3, 3), activation='relu')

Made with GAMMA

Metric	4 Fully connected Layers 128 - 96 - 64-32	3 Fully connected Layers 64-32-16	2 Fully connected Layers 96 - 32	1 Fully connected layer 64
Final Accuracy	95.29 %	98.87 %	99.1 %	99.13 %
Accuracy (Epochs 1–5)	[0.40, 0.73, 0.88, 0.90, 0.90]	[0.91, 0.97, 0.98, 0.99, 0.99]	[0.933, 0.982, 0.988, 0.992, 0.994]	[0.939, 0.982, 0.989, 0.992, 0.994]
Average Train Time per Epoch	-0.05 s	0.00 s	0.00s	0.00s
Average Test Time	0.91 s	0.65 s	1.38 s	0.32 s
Total Parameters	2,062,358	1,034,550	1,531,542	1,030,294

Fully Connected Layers Comparison

1 FC Layer (64 units)

Highest accuracy: 99.13%

Fastest test time

More FC Layers

Lower accuracy and longer test times due to increasing parameters

Made with GAMMA

Metric	SGD	Adam	Adagrad	AdamW
Final Accuracy	99.13 %	97.69 %	98.82 %	98.08 %
Accuracy (Epochs 1–5)	[0.939, 0.982, 0.989, 0.992, 0.994]	[0.954, 0.976, 0.981, 0.982, 0.983]	[0.912, 0.972, 0.98, 0.984, 0.986]	[0.956, 0.975, 0.979, 0.983, 0.983]
Average Train Time per Epoch	0.00s	0.01s	0.00 s	0.00s
Average Test Time	0.32 s	0.55 s	0.73 s	0.54 s
Total Parameters	1,030,294	1,545,440	1,030,294	1,545,440

Optimizers

SGD

Best accuracy: 99.13%

Consistent training and test speed

Used Momentum: 0.9

ADAM

Least Accuracy: 97.69%

Adagrad

Accuracy: 98.82%

Highest test time : ~73s

AdamW

Moderate Accuracy: 98.08%

Made with GAMMA

Dropout Regularization

Dropout Rates Tested

0%, 10%, 25%, 50% dropout after dense layer.

Best Rate

25% dropout gave best validation accuracy with minimal loss.

Placement

Added after fully connected layer to reduce overfitting.

Made with GAMMA

Final CNN Model Summary

2 Conv Layers (32 & 64 filters) with ReLU activation

MaxPooling after first Conv, FC Dense(64) + Dropout(0.25)

Trained 18 epochs, batch size 32, LR 0.01, SGD optimizer(0.9)

Final accuracy: 99.16%, Val Accuracy: 99.15%, test time ~1.68s

Conclusion

This model showed steady improvement across the early epochs (e.g., ~91.6% accuracy after epoch 1), and ultimately achieved strong generalization, making it a robust solution for handwritten digit recognition.

```
# 18 Epochs - batch_size = 32 - learning_rate = 0.01 -
#  2 Conv Layer Small dense (32,64) - 1 FC layer 64 -
#  relu - optimizer : SGD - 25% dropout - shuffle

def build_cnn():
    model = Sequential([
        Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
        MaxPooling2D(pool_size=(2,2),strides=(2,2)),
        Conv2D(64, kernel_size=(3, 3), activation='relu'),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.25),
        Dense(10, activation='softmax')
    ])
    optimizer = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

cnn_model = build_cnn()
history = cnn_model.fit(x_train_cnn, y_train_cat,
                        epochs=18, batch_size=32,
                        validation_split=0.1, shuffle = True)
```

Made with GAMMA

Calculating Model Parameters Layer-by-Layer

1

Conv Layer 1 (32 filters)

Parameters = (Filter Size 3x3 × Input Channels 1 + Bias 1) × Filters 32 = 320

2

Conv Layer 2 (64 filters)

Parameters = (3x3 × 32 + 1) × 64 = 18,496

3

Fully Connected Layer (64 units)

Parameters = (Input Features from Flattened Conv2 [7744] × 64) + 64 = 495,680

4

Total Parameters

Sum of all layers ≈ 515,146 parameters, confirming model size.

Layer	Output Shape	Parameters	Notes
Input	(28, 28, 1)	0	60000 training samples
Conv2D(32, 3x3)	(26, 26, 32)	320	(3×3×1 + 1)×32
MaxPooling2D(2x2)	(13, 13, 32)	0	No parameters
Conv2D(64, 3x3)	(11, 11, 64)	18,496	(3×3×32 + 1)×64
Flatten	(7744,)	0	Reshape only
Dense(64)	(64,)	495,680	(7744 + 1) × 64
Dropout(0.25)	(64,)	0	No parameters
Dense(10)	(10,)	650	(64 + 1) × 10
Total	–	**515,146**	All trainable

Made with GAMMA