Cairo University

Faculty of Computers and Information

AI322 – Supervised

Project

Owners	ID
Mahmoud Ehab Helmy	20220457
Abdelrahman Ibrahim	*20220519*
Moaz Gehad	*20220340*
Zeyad Mohamed	*20220146*

Initial CNN Model

```
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 Overview

The initial Convolutional Neural Network (CNN) model consists of two convolutional layers followed by a fully connected dense layer. It uses ReLU activation for non-linearity and a softmax layer for multi-class classification over the 10 MNIST digit classes. The architecture includes:

- **Conv2D(32) → MaxPooling2D → Conv2D(64) → Flatten**
- Followed by a **Dense(128)** hidden layer and a final **Dense(10)** output layer.

Training was performed for **5 epochs** using **SGD optimizer** (learning rate = 0.01, momentum = 0.9) and a **batch size of 128** with **10% validation split**.
This model served as a starting point to test and tune architecture depth, neuron count, batch size, and regularization techniques in later experiments.

Model Architecture

Layer (type)	Output Shape	# Neurons	# Parameters
Conv2D	(26, 26, 32) (relu)	21,632	320
MaxPooling2D	(13, 13, 32)	5,408	0
Conv2D	(11, 11, 64) (relu)	7,744	18,496
Flatten	(7744)	7,744	0
Dense	(128) (relu)	128	991,360
Dense	(10) (Softmax)	10	1,290

Comparison Table for number of Epochs

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)

We observed that the model achieves its best performance when trained for **18 to 20 epochs**, indicating that this range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs** for training, as it consistently delivers high accuracy while avoiding overfitting.

Comparison Table for number of Batch size

Metric	Batch Size 32	Batch Size 64	Batch Size 96	Batch Size 128
Final Accuracy	99.16%	99.01%	98.94%	98.85%
Accuracy (Epochs 1–5)	94.08%, 98.31%, 98.95%, 99.20%, 99.49%	92.12%, 97.49%, 98.37%, 98.74%, 99.05%	90.45%, 97.30%, 98.10%, 98.49%, 98.89%	88.42%, 96.87%, 97.89%, 98.35%, 98.68%
Average Train Time per Epoch	~0.00 s	~0.00 s	~0.00 s	~0.00 s
Average Test Time	0.51 s	0.73 s	0.68 s	0.74 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
# Epochs	18	18	18	18
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

Batch Size	code
32	cnn_model.fit(x_train_cnn, y_train_cat, epochs=18, batch_size=32, validation_split=0.1)
64	cnn_model.fit(x_train_cnn, y_train_cat, epochs=18, batch_size=64, validation_split=0.1)
96	cnn_model.fit(x_train_cnn, y_train_cat, epochs=18, batch_size=96, validation_split=0.1)
128	cnn_model.fit(x_train_cnn, y_train_cat, epochs=18, batch_size=128, validation_split=0.1)

We observed that the model achieves its best performance when trained for **18 to 20 epochs and batch size of 32**, indicating that those range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs and batch size of 32** for training, as it consistently delivers high accuracy while avoiding overfitting.

Comparison Table for size of LR

Metric	LR size 0.1	LR size 0.01	LR size 0.05	LR size 0.001	LR size 0.0001
Final Accuracy	10.28%	99.24%	98.59%	98.60%	96.47%
Accuracy (Epochs 1–5)	[0.904, 0.913, 0.885, 0.840, 0.862]	[0.939, 0.982, 0.988, 0.991, 0.994]	[0.954, 0.985, 0.990, 0.992, 0.993]	[0.866, 0.947, 0.962, 0.971, 0.976]	[0.60, 0.87, 0.89, 0.91, 0.92]
Average Train Time per Epoch	0.11 s	~0.00 s	0.00 s	~-0.01 s	~-0.02 s
Average Test Time	0.51 s	0.74 s	0.73 s	0.69 s	0.5 s
Total Parameters	2,022,934	2,022,934	2,022,934	2,022,934	2,022,934
Trainable Parameters	1,011,466	1,011,466	1,011,466	1,011,466	1,011,466
# Epochs	18	18	18	18	
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

LR SIZE	Code
0.1	SGD(learning_rate=0.1, momentum=0.9)
0.01	SGD(learning_rate=0.01, momentum=0.9)
0.05	SGD(learning_rate=0.05, momentum=0.9)
0.0001	SGD(learning_rate=0.0001, momentum=0.9)

We observed that the model achieves its best performance when trained for **18 to 20 epochs and batch size of 32 and LR of 0.01,** indicating that those range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs and batch size of 32 and LR of 0.01** for training, as it consistently delivers high accuracy while avoiding overfitting.

Comparison Table for different activation functions

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
Total Parameters	1,030,294	1,030,294	1,030,294	1,030,294
Trainable Parameters	515,146	515,146	515,146	515,146
# Epochs	18	18	18	18
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

Activation function	Code
relu	Conv2D(64, kernel_size=(3, 3), activation='relu'),
sigmoid	Conv2D(64, kernel_size=(3, 3), activation='sigmoid')
tanh	Conv2D(64, kernel_size=(3, 3), activation='tanh')
softplus	Conv2D(64, kernel_size=(3, 3), activation='softplus')

We observed that the model achieves its best performance when trained for **18 to 20 epochs and batch size of 32 and LR of 0.01,** and gives better results with **relu** activation function indicating that those range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs and batch size of 32 and LR of 0.01 and relu activation function** for training, as it consistently delivers high accuracy while avoiding overfitting.

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')

We observed that the model achieves its best performance when trained for **18 to 20 epochs and batch size of 32 and LR of 0.01 and** gives better results with **relu** activation function and **2 Convolution Layers 32-64** indicating that those range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs and batch size of 32 and LR of 0.01 and relu activation function and 2 Convolution Layers 32-64** for training, as it consistently delivers high accuracy while avoiding overfitting.

Comparison Table for different FC layers

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
Trainable Parameters	1,031,178	517,274	765,770	515,146
# Epochs	18	18	18	18
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

FC layers	Code
4 Fully connected Layers 128 - 96 - 64 -32	Dense(128, activation='relu'), Dense(96, activation='relu'), Dense(64, activation='relu'), Dense(32, activation='relu'), Dense(10, activation='softmax')
3 Fully connected Layers 64 - 32 -16	Dense(64, activation='relu'), Dense(32, activation='relu'), Dense(16, activation='relu'), Dense(10, activation='softmax')
2 Fully connected Layers 96 - 32	Dense(96, activation='relu') Dense(32, activation='relu') , Dense(10, activation='softmax')
1 Fully connected Layers 64	Dense(64, activation='relu'), Dense(10, activation='softmax')

We observed that the model achieves its best performance when trained for **18 to 20 epochs and batch size of 32 and LR of 0.01 and** gives better results with **relu** activation function and **2 Convolution Layers 32-64 with 1 Fully connected layer 64**
indicating that those range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs and batch size of 32 and LR of 0.01 and relu activation function and 2 Convolution Layers 32-64 with 1 Fully connected layer 64** for training, as it consistently delivers high accuracy while avoiding overfitting.

Comparison Table for different Optimizers

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
Trainable Parameters	515,146	515,146	515,146	515,146
# Epochs	18	18	18	18
Optimizers	SGD (momentum=0.9)	Adam	Adagrad	AdamW

Optimizer	Code
SGD	optimizer = SGD(learning_rate=0.01, momentum=0.9)
Adam	optimizer = Adam(learning_rate=0.01)
Adagrad	optimizer = Adagrad(learning_rate=0.01)
AdamW	optimizer = AdamW(learning_rate=0.01)

We observed that the model achieves its best performance when trained for **18 to 20 epochs and batch size of 32 and LR of 0.01 and** gives better results with **relu** activation function while using **SGD optimizer** and **2 Convolution Layers 32-64 with 1 Fully connected layer 64** indicating that those range provides the optimal balance between learning and generalization. Based on this observation, we will proceed with **18 epochs and batch size of 32 and LR of 0.01 and relu activation function and 2 Convolution Layers 32-64 with 1 Fully connected layer 64** for training and **SGD as optimizer**, as it consistently delivers high accuracy while avoiding overfitting.

Comparison Table for different Dropout

Metric	50% Dropout	25% Dropout	10% Dropout	No Dropout
Final Accuracy	99.17 %	99.17 %	99.13 %	99.11 %
Accuracy (Epochs 1–5)	[0.89, 0.95, 0.96, 0.975, 0.979]	[0.918, 0.973, 0.981, 0.985, 0.987]	[0.932, 0.979, 0.985, 0.989, 0.991]	[0.937, 0.982, 0.988, 0.991, 0.993]
Average Train Time per Epoch	0.00s	0.00s	0.00 s	0.00s
Average Test Time	0.74 s	0.73 s	0.73 s	0.5 s
Total Parameters	1,030,294	1,030,294	1,030,294	1,030,294
Trainable Parameters	515,146	515,146	515,146	515,146
# Epochs	18	18	18	18
Optimizers	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)	SGD (momentum=0.9)

Dropout	Code
50%	Dense(64, activation='relu'), Dropout(0.5), Dense(10, activation='softmax')
25%	Dense(64, activation='relu'), Dropout(0.25), Dense(10, activation='softmax')
10%	Dense(64, activation='relu'), Dropout(0.1), Dense(10, activation='softmax')
No dropout	Dense(64, activation='relu'), Dense(10, activation='softmax')

We added a **Dropout layer (rate = 0.25)** after the first dense layer:

```
Dense(64, activation='relu'),
Dropout(0.25),
Dense(10, activation='softmax')
```

- **Why here?**

Placed after the fully connected layer to prevent overfitting, since dense layers are more prone to it.

- **Why 0.25?**

After testing different rates, 0.25 gave the best validation accuracy with minimal training loss.

Final CNN Model

```
# 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)
```

Final CNN Model Description

The final Convolutional Neural Network (CNN) architecture was optimized through iterative experimentation to achieve high classification performance on the MNIST dataset. The model consists of two convolutional layers with 32 and 64 filters respectively, each followed by ReLU activation. A max pooling layer is used after the first convolution to reduce spatial dimensions. After flattening, a fully connected dense layer with 64 units and ReLU activation is applied, followed by a dropout layer with a rate of **0.25** to reduce overfitting. The output layer is a softmax layer with 10 units for classification.

The model was trained using the **SGD optimizer** with a **learning rate of 0.01** and **momentum 0.9**, for **18 epochs** with a **batch size of 32**, and **shuffle enabled** during training.

- **Final Training Accuracy:** 99.16%
- **Final Validation Accuracy:** 99.15%
- **Total Trainable Parameters:** 515,146
- **Total Parameters:** 1,030,294
- **Average Testing Time:** ~1.68 seconds

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.