KATTAM SANJAY

22WU0104055

AIML- A

DEEP LEARNING LAB - 2

1. Build and train a baseline model without any regularization.Record training and validation accuracy, loss, and observe overfitting symptoms (if any).

import tensorflow as tf

from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten from tensorflow.keras.utils import to\_categorical import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0 *# Normalize the*

*images*

y\_train, y\_test = to\_categorical(y\_train, 10), to\_categorical(y\_test, 10)

model = Sequential([ Flatten(input\_shape=(28, 28)), Dense(128, activation='relu'), Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

history = model.fit(x\_train, y\_train, epochs=4, validation\_data=(x\_test, y\_test))

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy') plt.plot(history.history['val\_accuracy'], label='Validation Accuracy') plt.title('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss') plt.plot(history.history['val\_loss'], label='Validation Loss') plt.title('Loss')

plt.legend()

plt.show()

*# Evaluate the model on test data*

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test) print(f'Test Accuracy: {test\_accuracy \* 100:.2f}%')

Downloading data from https://storage.googleapis.com/tensorflow/tf- keras-datasets/mnist.npz

11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/ flatten.py:37: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an

`Input(shape)` object as the first layer in the model instead.

super(). init (\*\*kwargs)

Epoch 1/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.8804 - loss: 0.4281 - val\_accuracy: 0.9613 - val\_loss: 0.1306

Epoch 2/4

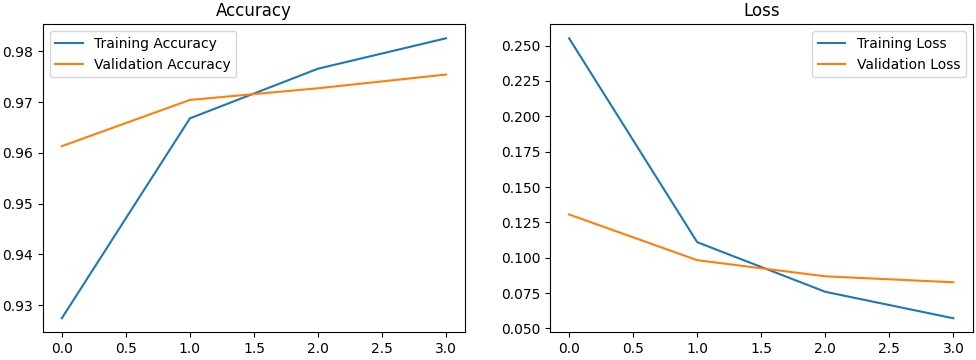
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 4ms/step - accuracy: 0.9651 - loss: 0.1176 - val\_accuracy: 0.9704 - val\_loss: 0.0983

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9759 - loss: 0.0780 - val\_accuracy: 0.9727 - val\_loss: 0.0869

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9839 - loss: 0.0537 - val\_accuracy: 0.9754 - val\_loss: 0.0826



313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9710 - loss: 0.0994

Test Accuracy: 97.54%

1. L1 and L2 Regularization:

Modify the baseline model by applying L1 and L2 regularization (individually and together).

Experiment with different regularization strengths (e.g., λ = 0.01, 0.1, 0.5) and evaluate their impact on model performance.

from tensorflow.keras.regularizers import l1, l2

l1\_model = Sequential([ Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu', kernel\_regularizer=l1(0.01)), Dense(10, activation='softmax')

])

l1\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

l1\_history = l1\_model.fit(x\_train, y\_train, epochs=4, validation\_data=(x\_test, y\_test))

l2\_model = Sequential([ Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu', kernel\_regularizer=l2(0.01)), Dense(10, activation='softmax')

])

l2\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

l2\_history = l2\_model.fit(x\_train, y\_train, epochs=4, validation\_data=(x\_test, y\_test))

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(l1\_history.history['accuracy'], label='L1 Training Accuracy') plt.plot(l1\_history.history['val\_accuracy'], label='L1 Validation Accuracy')

plt.title('L1 Regularization Accuracy') plt.legend()

plt.subplot(1, 2, 2)

plt.plot(l2\_history.history['accuracy'], label='L2 Training Accuracy') plt.plot(l2\_history.history['val\_accuracy'], label='L2 Validation Accuracy')

plt.title('L2 Regularization Accuracy') plt.legend()

plt.show()

Epoch 1/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.7682 - loss: 5.0004 - val\_accuracy: 0.8525 - val\_loss: 1.1576

Epoch 2/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8462 - loss: 1.1673 - val\_accuracy: 0.8629 - val\_loss: 1.0535

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.8620 - loss: 1.0565 - val\_accuracy: 0.8663 - val\_loss: 1.0104

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8623 - loss: 1.0189 - val\_accuracy: 0.8708 - val\_loss: 0.9373

Epoch 1/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8583 - loss: 0.9560 - val\_accuracy: 0.9275 - val\_loss: 0.4172

Epoch 2/4

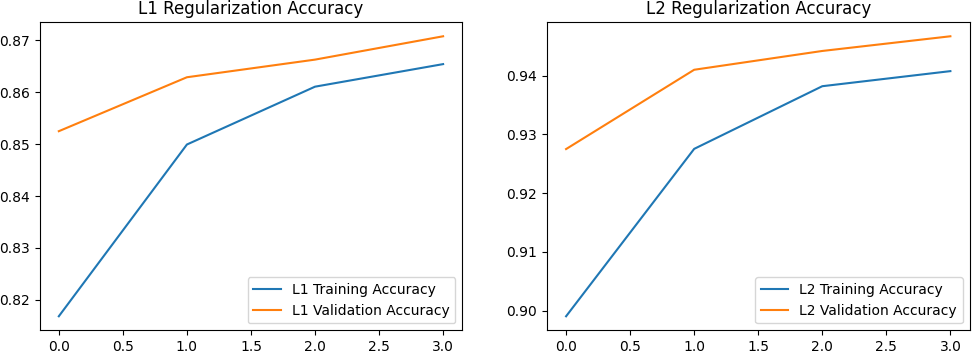
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9243 - loss: 0.4143 - val\_accuracy: 0.9410 - val\_loss: 0.3488

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.9365 - loss: 0.3609 - val\_accuracy: 0.9442 - val\_loss: 0.3328

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9395 - loss: 0.3425 - val\_accuracy: 0.9467 - val\_loss: 0.3166



1. Dropout:

Introduce dropout layers into your model architecture.

Experiment with different dropout rates (e.g., 0.2, 0.5) and observe how it affects overfitting and model performance.

from tensorflow.keras.layers import Dropout

*# Model with Dropout Regularization*

dropout\_model = Sequential([ Flatten(input\_shape=(28, 28)), Dense(128, activation='relu'), Dropout(0.5), *# Dropout rate of 50%* Dense(10, activation='softmax')

])

dropout\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) dropout\_history = dropout\_model.fit(x\_train, y\_train, epochs=4, validation\_data=(x\_test, y\_test))

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(dropout\_history.history['accuracy'], label='Dropout Training Accuracy')

plt.plot(dropout\_history.history['val\_accuracy'], label='Dropout Validation Accuracy')

plt.title('Dropout Regularization Accuracy') plt.legend()

plt.subplot(1, 2, 2)

plt.plot(dropout\_history.history['loss'], label='Dropout Training Loss')

plt.plot(dropout\_history.history['val\_loss'], label='Dropout Validation Loss')

plt.title('Dropout Regularization Loss') plt.legend()

plt.show()

Epoch 1/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8205 - loss: 0.5950 - val\_accuracy: 0.9532 - val\_loss: 0.1646

Epoch 2/4

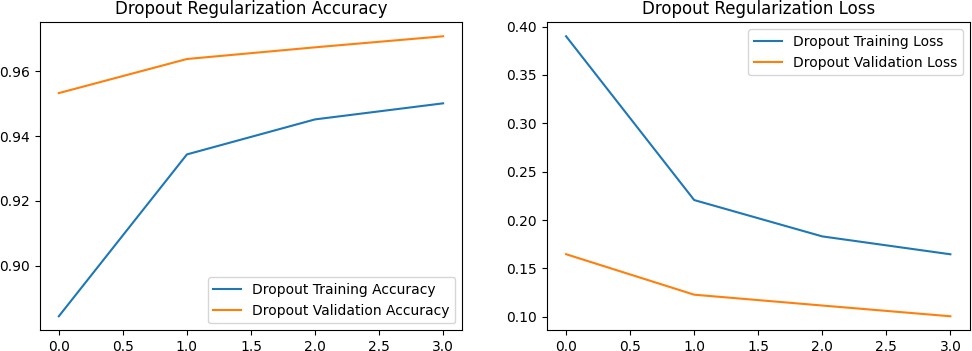
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9297 - loss: 0.2328 - val\_accuracy: 0.9637 - val\_loss: 0.1227

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9449 - loss: 0.1831 - val\_accuracy: 0.9673 - val\_loss: 0.1115

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9503 - loss: 0.1645 - val\_accuracy: 0.9707 - val\_loss: 0.1004



1. Early Stopping:

Implement early stopping by monitoring validation loss.

Set appropriate patience and minimum delta values. Compare results with and without early stopping.

from tensorflow.keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, min\_delta=0.001, restore\_best\_weights=True)

early\_stopping\_model = Sequential([ Flatten(input\_shape=(28, 28)), Dense(128, activation='relu'), Dense(10, activation='softmax')

])

early\_stopping\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

early\_stopping\_history = early\_stopping\_model.fit(x\_train, y\_train, epochs=4, validation\_data=(x\_test, y\_test), callbacks=[early\_stopping])

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1) plt.plot(early\_stopping\_history.history['accuracy'], label='Early Stopping Training Accuracy') plt.plot(early\_stopping\_history.history['val\_accuracy'], label='Early

Stopping Validation Accuracy') plt.title('Early Stopping Accuracy') plt.legend()

plt.subplot(1, 2, 2)

plt.plot(early\_stopping\_history.history['loss'], label='Early Stopping Training Loss')

plt.plot(early\_stopping\_history.history['val\_loss'], label='Early Stopping Validation Loss')

plt.title('Early Stopping Loss') plt.legend()

plt.show()

Epoch 1/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8812 - loss: 0.4252 - val\_accuracy: 0.9575 - val\_loss: 0.1432

Epoch 2/4

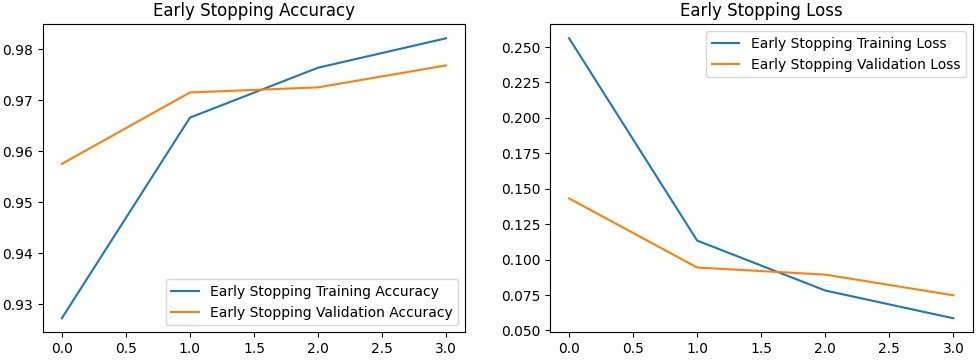
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9644 - loss: 0.1242 - val\_accuracy: 0.9715 - val\_loss: 0.0944

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9774 - loss: 0.0764 - val\_accuracy: 0.9725 - val\_loss: 0.0894

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9837 - loss: 0.0554 - val\_accuracy: 0.9768 - val\_loss: 0.0748



1. Data Augmentation:

Apply data augmentation techniques (e.g., flipping, rotation, cropping, brightness/contrast adjustment).

Train the model using augmented data and evaluate its impact on performance.

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator( rotation\_range=20, width\_shift\_range=0.2, height\_shift\_range=0.2, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, fill\_mode='nearest'

)

datagen.fit(x\_train.reshape(-1, 28, 28, 1)) *# Reshape to 4D for augmentation*

augmented\_model = Sequential([ Flatten(input\_shape=(28, 28)), Dense(128, activation='relu'), Dense(10, activation='softmax')

])

augmented\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

augmented\_history = augmented\_model.fit(datagen.flow(x\_train.reshape(- 1, 28, 28, 1), y\_train, batch\_size=32), epochs=4,

validation\_data=(x\_test.reshape(-1, 28, 28, 1), y\_test))

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1) plt.plot(augmented\_history.history['accuracy'], label='Augmented Training Accuracy') plt.plot(augmented\_history.history['val\_accuracy'], label='Augmented Validation Accuracy')

plt.title('Data Augmentation Accuracy') plt.legend()

plt.subplot(1, 2, 2)

plt.plot(augmented\_history.history['loss'], label='Augmented Training Loss')

plt.plot(augmented\_history.history['val\_loss'], label='Augmented Validation Loss')

plt.title('Data Augmentation Loss') plt.legend()

plt.show() Epoch 1/4

/usr/local/lib/python3.11/dist-packages/keras/src/trainers/ data\_adapters/py\_dataset\_adapter.py:121: UserWarning: Your `PyDataset` class should call `super(). init (\*\*kwargs)` in its constructor.

`\*\*kwargs` can include `workers`, `use\_multiprocessing`,

`max\_queue\_size`. Do not pass these arguments to `fit()`, as they will be ignored.

self.\_warn\_if\_super\_not\_called()

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 36s 19ms/step - accuracy: 0.4647 - loss: 1.5753 - val\_accuracy: 0.8185 - val\_loss: 0.5836

Epoch 2/4

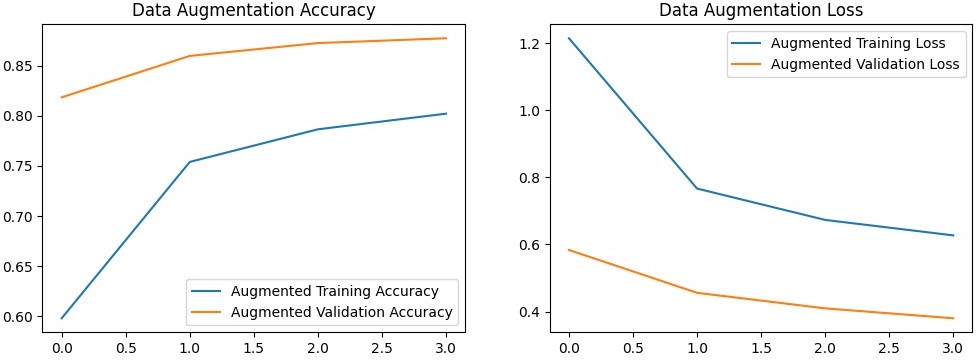
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 33s 17ms/step - accuracy: 0.7394 - loss: 0.8043 - val\_accuracy: 0.8599 - val\_loss: 0.4560

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 33s 18ms/step - accuracy: 0.7818 - loss: 0.6833 - val\_accuracy: 0.8727 - val\_loss: 0.4097

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 37s 20ms/step - accuracy: 0.7964 - loss: 0.6447 - val\_accuracy: 0.8775 - val\_loss: 0.3802



1. Combined Regularization:

Combine two or more regularization techniques (e.g., L2 + Dropout + Data Augmentation). Compare the results with individual techniques and the baseline model.

combined\_model = Sequential([ Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu', kernel\_regularizer=l2(0.01)), Dropout(0.5),

Dense(10, activation='softmax')

])

combined\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

combined\_history = combined\_model.fit(datagen.flow(x\_train.reshape(-1, 28, 28, 1), y\_train, batch\_size=32), epochs=4,

validation\_data=(x\_test.reshape(-1, 28, 28, 1), y\_test))

*#*

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1) plt.plot(combined\_history.history['accuracy'], label='Combined Training Accuracy') plt.plot(combined\_history.history['val\_accuracy'], label='Combined Validation Accuracy')

plt.title('Combined Regularization Accuracy') plt.legend()

plt.subplot(1, 2, 2)

plt.plot(combined\_history.history['loss'], label='Combined Training Loss')

plt.plot(combined\_history.history['val\_loss'], label='Combined Validation Loss')

plt.title('Combined Regularization Loss') plt.legend()

plt.show()

Epoch 1/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 36s 18ms/step - accuracy: 0.3050 - loss: 2.5203 - val\_accuracy: 0.6614 - val\_loss: 1.4873

Epoch 2/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 21ms/step - accuracy: 0.4669 - loss: 1.8027 - val\_accuracy: 0.6828 - val\_loss: 1.3834

Epoch 3/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 36s 19ms/step - accuracy: 0.4935 - loss: 1.7462 - val\_accuracy: 0.6900 - val\_loss: 1.3334

Epoch 4/4

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 37s 20ms/step - accuracy: 0.5054 - loss: 1.7031 - val\_accuracy: 0.7466 - val\_loss: 1.2293

