import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import mnist

# Load and preprocess data

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

X\_train\_full = X\_train\_full / 255.0

X\_test = X\_test / 255.0

# Split into train and validation sets

X\_train, X\_val = X\_train\_full[:-5000], X\_train\_full[-5000:]

y\_train, y\_val = y\_train\_full[:-5000], y\_train\_full[-5000:]

# Add channel dimension

X\_train = X\_train[..., np.newaxis]

X\_val = X\_val[..., np.newaxis]

X\_test = X\_test[..., np.newaxis]

# Set random seed for reproducibility

tf.random.set\_seed(42)

# Build the model

model = tf.keras.Sequential([

tf.keras.layers.Conv2D(32, 3, padding="same", activation="relu", kernel\_initializer="he\_normal", input\_shape=(28, 28, 1)),

tf.keras.layers.Conv2D(64, 3, padding="same", activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.MaxPooling2D(),

tf.keras.layers.Flatten(),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Dense(128, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(10, activation="softmax")

])

# Compile and train

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="nadam", metrics=["accuracy"])

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_val, y\_val))

# Save and evaluate

model.save("./models/my\_mnist\_cnn\_model.keras")

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print("Test accuracy:", test\_acc)

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense

from keras.callbacks import EarlyStopping, ModelCheckpoint

from keras.optimizers import SGD, Adadelta, Adam, RMSprop, Adagrad, Nadam, Adamax

import os

# Set seed

SEED = 2017

# Load dataset

data = pd.read\_csv('Data/winequality-red.csv', sep=';')

X = data.drop('quality', axis=1)

y = data['quality']

# Split dataset: train, val, test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=SEED)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=SEED)

# Create checkpoints folder if not exists

os.makedirs("checkpoints", exist\_ok=True)

# Model architecture

def create\_model(input\_dim):

model = Sequential([

Dense(100, input\_dim=input\_dim, activation='relu'),

Dense(50, activation='relu'),

Dense(25, activation='relu'),

Dense(10, activation='relu'),

Dense(1, activation='linear')

])

return model

# Define optimizers

optimizers = {

'sgd': SGD(),

'sgd-0001': SGD(learning\_rate=0.0001, decay=0.00001),

'adam': Adam(),

'adadelta': Adadelta(),

'rmsprop': RMSprop(),

'rmsprop-0001': RMSprop(learning\_rate=0.0001),

'nadam': Nadam(),

'adamax': Adamax()

}

# Training config

batch\_size = 128

epochs = 1000

results = []

# Training loop

for name, opt in optimizers.items():

model = create\_model(X\_train.shape[1])

model.compile(optimizer=opt, loss='mse', metrics=['accuracy'])

# Callbacks

ckpt\_path = f"checkpoints/best\_{name}.h5"

callbacks = [

EarlyStopping(monitor='val\_accuracy', patience=200, verbose=0, restore\_best\_weights=True),

ModelCheckpoint(ckpt\_path, monitor='val\_accuracy', save\_best\_only=True, verbose=0)

]

# Train model

history = model.fit(

X\_train, y\_train,

validation\_data=(X\_val, y\_val),

epochs=epochs,

batch\_size=batch\_size,

callbacks=callbacks,

verbose=0

)

best\_epoch = np.argmax(history.history['val\_accuracy'])

best\_val\_acc = history.history['val\_accuracy'][best\_epoch]

# Load best model and evaluate

best\_model = create\_model(X\_train.shape[1])

best\_model.compile(optimizer=opt, loss='mse', metrics=['accuracy'])

best\_model.load\_weights(ckpt\_path)

test\_loss, test\_acc = best\_model.evaluate(X\_test, y\_test, verbose=0)

results.append([name, best\_epoch, best\_val\_acc, test\_acc])

# Show results

res = pd.DataFrame(results, columns=['optimizer', 'best\_epoch', 'val\_accuracy', 'test\_accuracy'])

print(res)

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

import os

# Paths (change to your actual dataset path)

data\_dir = "path/to/dataset"

train\_dir = os.path.join(data\_dir, "train")

val\_dir = os.path.join(data\_dir, "val")

# Parameters

img\_size = (224, 224)

batch\_size = 32

epochs = 10

num\_classes = len(os.listdir(train\_dir))

# Data generators

train\_gen = ImageDataGenerator(rescale=1./255, rotation\_range=30,

width\_shift\_range=0.2, height\_shift\_range=0.2,

horizontal\_flip=True)

val\_gen = ImageDataGenerator(rescale=1./255)

train\_data = train\_gen.flow\_from\_directory(train\_dir, target\_size=img\_size,

batch\_size=batch\_size, class\_mode='categorical')

val\_data = val\_gen.flow\_from\_directory(val\_dir, target\_size=img\_size,

batch\_size=batch\_size, class\_mode='categorical')

# Load pre-trained ResNet50 base

base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

base\_model.trainable = False

# Add custom classification layers

x = GlobalAveragePooling2D()(base\_model.output)

x = Dense(512, activation='relu')(x)

x = Dropout(0.5)(x)

output = Dense(num\_classes, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=output)

# Compile and train (initial training)

model.compile(optimizer=Adam(1e-3), loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(train\_data, validation\_data=val\_data, epochs=epochs)

# Fine-tuning

base\_model.trainable = True

for layer in base\_model.layers[:100]:

layer.trainable = False

model.compile(optimizer=Adam(1e-4), loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(train\_data, validation\_data=val\_data, epochs=epochs // 2)

# Save the model

model.save("fine\_tuned\_resnet50.h5")

# Imports required packages

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import numpy as np

# Load and prepare data

(X\_train\_full, y\_train\_full), (X\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()

class\_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# Normalize data

X\_train\_full, X\_test = X\_train\_full / 255.0, X\_test / 255.0

# Target classes: "Pullover" and "T-shirt/top"

class\_0\_index = class\_names.index("Pullover")

class\_1\_index = class\_names.index("T-shirt/top")

# Separate 2-class and 8-class datasets

mask\_2\_classes = np.isin(y\_train\_full, [class\_0\_index, class\_1\_index])

X\_train\_2\_classes\_full = X\_train\_full[mask\_2\_classes]

y\_train\_2\_classes\_full = y\_train\_full[mask\_2\_classes]

mask\_8\_classes = ~mask\_2\_classes

X\_train\_8\_classes\_full = X\_train\_full[mask\_8\_classes]

y\_train\_8\_classes\_full = y\_train\_full[mask\_8\_classes]

# Preprocess 8-class dataset

X\_train\_8\_classes, X\_val\_8\_classes, y\_train\_8\_classes, y\_val\_8\_classes = train\_test\_split(

X\_train\_8\_classes\_full, y\_train\_8\_classes\_full, test\_size=5000, random\_state=42, stratify=y\_train\_8\_classes\_full)

pixel\_means\_8 = X\_train\_8\_classes.mean(axis=0, keepdims=True)

pixel\_stds\_8 = X\_train\_8\_classes.std(axis=0, keepdims=True)

X\_train\_8\_scaled = (X\_train\_8\_classes - pixel\_means\_8) / pixel\_stds\_8

X\_val\_8\_scaled = (X\_val\_8\_classes - pixel\_means\_8) / pixel\_stds\_8

label\_encoder\_8 = LabelEncoder()

y\_train\_8\_encoded = label\_encoder\_8.fit\_transform(y\_train\_8\_classes)

y\_val\_8\_encoded = label\_encoder\_8.transform(y\_val\_8\_classes)

# Build and train model for 8 classes (pretraining)

model\_pretrain = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=[28, 28]),

tf.keras.layers.Dense(100, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dense(100, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dense(100, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dense(8, activation="softmax")

])

model\_pretrain.compile(loss="sparse\_categorical\_crossentropy", optimizer=tf.keras.optimizers.SGD(learning\_rate=0.001), metrics=["accuracy"])

model\_pretrain.fit(X\_train\_8\_scaled, y\_train\_8\_encoded, epochs=20, validation\_data=(X\_val\_8\_scaled, y\_val\_8\_encoded))

model\_pretrain.save("./models/my\_fashion\_mnist\_model.keras")

# Preprocess 2-class dataset

X\_train\_2, X\_val\_2, y\_train\_2, y\_val\_2 = train\_test\_split(X\_train\_2\_classes\_full, y\_train\_2\_classes\_full, test\_size=3000, random\_state=42, stratify=y\_train\_2\_classes\_full)

pixel\_means\_2 = X\_train\_2.mean(axis=0, keepdims=True)

pixel\_stds\_2 = X\_train\_2.std(axis=0, keepdims=True)

X\_train\_2\_scaled = (X\_train\_2 - pixel\_means\_2) / pixel\_stds\_2

X\_val\_2\_scaled = (X\_val\_2 - pixel\_means\_2) / pixel\_stds\_2

label\_encoder\_2 = LabelEncoder()

y\_train\_2\_encoded = label\_encoder\_2.fit\_transform(y\_train\_2)

y\_val\_2\_encoded = label\_encoder\_2.transform(y\_val\_2)

# Build and train model from scratch

tf.keras.backend.clear\_session()

tf.random.set\_seed(42)

model\_scratch = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=[28, 28]),

tf.keras.layers.Dense(100, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dense(100, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dense(100, activation="relu", kernel\_initializer="he\_normal"),

tf.keras.layers.Dense(1, activation="sigmoid")

])

model\_scratch.compile(loss="binary\_crossentropy", optimizer=tf.keras.optimizers.SGD(learning\_rate=0.001), metrics=["accuracy"])

model\_scratch.fit(X\_train\_2\_scaled, y\_train\_2\_encoded, epochs=20, validation\_data=(X\_val\_2\_scaled, y\_val\_2\_encoded))

# Prepare test set for 2 classes

mask\_test\_2\_classes = np.isin(y\_test, [class\_0\_index, class\_1\_index])

X\_test\_2 = X\_test[mask\_test\_2\_classes]

y\_test\_2 = y\_test[mask\_test\_2\_classes]

y\_test\_2\_encoded = label\_encoder\_2.transform(y\_test\_2)

X\_test\_2\_scaled = (X\_test\_2 - pixel\_means\_2) / pixel\_stds\_2

model\_scratch.evaluate(X\_test\_2\_scaled, y\_test\_2\_encoded)

# Transfer Learning

model\_transfer = tf.keras.models.load\_model("./models/my\_fashion\_mnist\_model.keras")

model\_transfer.pop()

model\_transfer.add(tf.keras.layers.Dense(1, activation="sigmoid"))

X\_train\_2\_subset, \_, y\_train\_2\_subset, \_ = train\_test\_split(X\_train\_2\_scaled, y\_train\_2\_encoded, train\_size=0.60, stratify=y\_train\_2\_encoded)

for layer in model\_transfer.layers[:-1]:

layer.trainable = False

tf.keras.backend.clear\_session()

tf.random.set\_seed(42)

model\_transfer.compile(loss="binary\_crossentropy", optimizer=tf.keras.optimizers.SGD(learning\_rate=0.001))

model\_transfer.fit(X\_train\_2\_subset, y\_train\_2\_subset, epochs=5, validation\_data=(X\_val\_2\_scaled, y\_val\_2\_encoded))

# Fine-tune entire model

for layer in model\_transfer.layers[:-1]:

layer.trainable = True

model\_transfer.compile(loss="binary\_crossentropy", optimizer=tf.keras.optimizers.SGD(learning\_rate=0.001))

model\_transfer.fit(X\_train\_2\_subset, y\_train\_2\_subset, epochs=100, validation\_data=(X\_val\_2\_scaled, y\_val\_2\_encoded))

model\_transfer.evaluate(X\_test\_2\_scaled, y\_test\_2\_encoded)

# STEP 1: IMPORT LIBRARIES AND LOAD DATA

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

import random

# Load Fashion MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()

# Display a sample image

plt.imshow(X\_train[0], cmap='gray')

plt.title(f"Label: {y\_train[0]}")

plt.show()

# Display dataset shapes

print("Training Set Shape:", X\_train.shape)

print("Test Set Shape:", X\_test.shape)

# STEP 2: DATA VISUALIZATION

# Display 15x15 grid of random images

fig, axes = plt.subplots(15, 15, figsize=(17, 17))

axes = axes.ravel()

for i in range(225):

idx = random.randint(0, len(X\_train)-1)

axes[i].imshow(X\_train[idx], cmap='gray')

axes[i].set\_title(y\_train[idx], fontsize=8)

axes[i].axis('off')

plt.subplots\_adjust(hspace=0.4)

plt.show()

# STEP 3: DATA PREPROCESSING (NORMALIZE AND ADD NOISE)

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

# Add Gaussian noise

def add\_noise(data, noise\_factor=0.3):

noisy = data + noise\_factor \* np.random.randn(\*data.shape)

return np.clip(noisy, 0., 1.)

noise\_train = add\_noise(X\_train)

noise\_test = add\_noise(X\_test)

# Visualize a noisy image

plt.imshow(noise\_train[22], cmap='gray')

plt.title("Noisy Image")

plt.show()

# STEP 4: BUILD & TRAIN AUTOENCODER

autoencoder = tf.keras.models.Sequential([

tf.keras.layers.Input(shape=(28, 28, 1)),

# Encoder

tf.keras.layers.Conv2D(16, 3, strides=2, padding='same', activation='relu'),

tf.keras.layers.Conv2D(8, 3, strides=2, padding='same', activation='relu'),

# Bottleneck

tf.keras.layers.Conv2D(8, 3, padding='same', activation='relu'),

# Decoder

tf.keras.layers.Conv2DTranspose(16, 3, strides=2, padding='same', activation='relu'),

tf.keras.layers.Conv2DTranspose(1, 3, strides=2, padding='same', activation='sigmoid')

])

autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),

loss='binary\_crossentropy')

autoencoder.summary()

# Train the autoencoder

autoencoder.fit(noise\_train.reshape(-1, 28, 28, 1),

X\_train.reshape(-1, 28, 28, 1),

epochs=10,

batch\_size=200,

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

X\_test.reshape(-1, 28, 28, 1)))

# STEP 5: EVALUATE MODEL

loss = autoencoder.evaluate(noise\_test.reshape(-1, 28, 28, 1),

X\_test.reshape(-1, 28, 28, 1))

print(f'Test Loss: {loss:.3f}')

# Predict denoised images

predicted = autoencoder.predict(noise\_test[:10].reshape(-1, 28, 28, 1))

# STEP 6: VISUALIZE RESULTS

fig, axes = plt.subplots(2, 10, figsize=(20, 4), sharex=True, sharey=True)

for i in range(10):

axes[0][i].imshow(noise\_test[i], cmap='gray')

axes[0][i].set\_title("Noisy")

axes[0][i].axis('off')

axes[1][i].imshow(predicted[i].reshape(28, 28), cmap='gray')

axes[1][i].set\_title("Denoised")

axes[1][i].axis('off')

plt.tight\_layout()

plt.show()

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

# Generate sine wave data

def generate\_sine\_sequence(n):

x = np.linspace(0, 50, n)

return np.sin(x)

# Parameters

n\_steps = 100

window\_size = 10

# Prepare data

sequence = generate\_sine\_sequence(n\_steps)

X = [sequence[i:i+window\_size] for i in range(n\_steps - window\_size)]

y = [sequence[i+window\_size] for i in range(n\_steps - window\_size)]

X = np.array(X).reshape(-1, window\_size, 1)

y = np.array(y)

# Define RNN model

model = Sequential([

SimpleRNN(10, activation='relu', input\_shape=(window\_size, 1)),

Dense(1)

])

model.compile(optimizer='adam', loss='mse')

# Train model

model.fit(X, y, epochs=100, verbose=1)

# Predict

predicted = model.predict(X)

# Compare expected vs predicted

print("Expected:", y[:5])

print("Predicted:", predicted[:5].flatten())

import tensorflow as tf

import numpy as np

# Load and preprocess text

with open("shakespeare.txt", "r") as file:

text = file.read().lower()

# Create character-index mappings

chars = sorted(set(text))

char\_to\_idx = {c: i for i, c in enumerate(chars)}

idx\_to\_char = {i: c for i, c in enumerate(chars)}

vocab\_size = len(chars)

# Prepare training sequences

seq\_length = 100

X, y = [], []

for i in range(len(text) - seq\_length):

X.append([char\_to\_idx[c] for c in text[i:i+seq\_length]])

y.append(char\_to\_idx[text[i + seq\_length]])

X = np.array(X) / vocab\_size # Normalize

X = X.reshape((X.shape[0], seq\_length, 1))

y = np.array(y)

# Build the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.LSTM(128, return\_sequences=True, input\_shape=(seq\_length, 1)),

tf.keras.layers.LSTM(128),

tf.keras.layers.Dense(vocab\_size, activation="softmax")

])

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam")

model.fit(X, y, epochs=20, batch\_size=64)

# Function to generate text

def generate\_text(seed, length=200):

result = seed.lower()

for \_ in range(length):

input\_seq = [char\_to\_idx.get(c, 0) for c in result[-seq\_length:]]

input\_seq = np.array(input\_seq) / vocab\_size

input\_seq = input\_seq.reshape(1, seq\_length, 1)

pred\_idx = np.argmax(model.predict(input\_seq, verbose=0))

result += idx\_to\_char[pred\_idx]

return result

# Generate and print text

print(generate\_text("shall i compare thee to a summer's day? "))

import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Load dataset

data = pd.read\_csv("diabetes.csv")

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

# Preprocessing

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2)

# Build model

def build\_model(optimizer):

model = tf.keras.Sequential([

tf.keras.layers.Dense(16, activation='relu', input\_shape=(X.shape[1],)),

tf.keras.layers.Dense(8, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

])

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

return model

# Compare Adam and RMSprop

optimizers = ['adam', 'rmsprop']

for opt in optimizers:

print(f"\nTraining with {opt.upper()}")

model = build\_model(opt)

history = model.fit(X\_train, y\_train, epochs=50, validation\_split=0.2, verbose=0)

loss, acc = model.evaluate(X\_test, y\_test)

print(f"{opt.upper()} Test Accuracy: {acc:.4f}")

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load and preprocess sneaker dataset (assume folders: train/sneaker1, sneaker2, etc.)

img\_size = (128, 128)

batch\_size = 32

train\_datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

train\_data = train\_datagen.flow\_from\_directory("sneaker\_dataset", target\_size=img\_size,

batch\_size=batch\_size, subset='training', class\_mode='categorical')

val\_data = train\_datagen.flow\_from\_directory("sneaker\_dataset", target\_size=img\_size,

batch\_size=batch\_size, subset='validation', class\_mode='categorical')

# MobileNetV1

mobilenet = tf.keras.applications.MobileNet(input\_shape=img\_size + (3,), include\_top=False, weights='imagenet')

mobilenet.trainable = False

model = tf.keras.Sequential([

mobilenet,

tf.keras.layers.GlobalAveragePooling2D(),

tf.keras.layers.Dense(train\_data.num\_classes, activation='softmax')

])

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

model.fit(train\_data, validation\_data=val\_data, epochs=5)

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load and preprocess MRI dataset (folder structure per class)

img\_size = (224, 224)

datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2,

rotation\_range=20, zoom\_range=0.2, horizontal\_flip=True)

train\_data = datagen.flow\_from\_directory("glioma\_dataset", target\_size=img\_size,

batch\_size=32, subset='training', class\_mode='categorical')

val\_data = datagen.flow\_from\_directory("glioma\_dataset", target\_size=img\_size,

batch\_size=32, subset='validation', class\_mode='categorical')

# ResNet50 with transfer learning

base\_model = tf.keras.applications.ResNet50(input\_shape=img\_size + (3,), include\_top=False, weights='imagenet')

base\_model.trainable = False

model = tf.keras.Sequential([

base\_model,

tf.keras.layers.GlobalAveragePooling2D(),

tf.keras.layers.Dense(256, activation='relu'),

tf.keras.layers.Dense(train\_data.num\_classes, activation='softmax')

])

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

model.fit(train\_data, validation\_data=val\_data, epochs=5)

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

# Load and preprocess wine quality dataset

data = pd.read\_csv("wine.csv") # Assumes normal wine quality data

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

X\_train, X\_test = train\_test\_split(data\_scaled, test\_size=0.2)

# Autoencoder

input\_dim = X\_train.shape[1]

autoencoder = tf.keras.Sequential([

tf.keras.layers.Dense(32, activation='relu', input\_shape=(input\_dim,)),

tf.keras.layers.Dense(16, activation='relu'),

tf.keras.layers.Dense(32, activation='relu'),

tf.keras.layers.Dense(input\_dim, activation='linear')

])

autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.fit(X\_train, X\_train, epochs=50, batch\_size=16, validation\_data=(X\_test, X\_test))

# Detect anomalies

reconstructions = autoencoder.predict(X\_test)

mse = np.mean(np.square(X\_test - reconstructions), axis=1)

threshold = np.percentile(mse, 95)

anomalies = mse > threshold

print(f"Detected {np.sum(anomalies)} anomalies in wine production.")

import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load and preprocess data

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv'

temps = pd.read\_csv(url)['Temp'].values.astype(np.float32)

mean, std = temps.mean(), temps.std()

temps = (temps - mean) / std

# Create sequences

def make\_seq(data, window=7):

X, y = [], []

for i in range(len(data) - window):

X.append(data[i:i+window])

y.append(data[i+window])

return np.array(X).reshape(-1, window, 1), np.array(y)

X, y = make\_seq(temps)

split = int(0.8 \* len(X))

X\_train, X\_test, y\_train, y\_test = X[:split], X[split:], y[:split], y[split:]

# Build and train RNN model

model = tf.keras.Sequential([

tf.keras.layers.SimpleRNN(64, input\_shape=(7, 1), return\_sequences=True),

tf.keras.layers.SimpleRNN(32),

tf.keras.layers.Dense(1)

])

model.compile(optimizer='adam', loss='mse')

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=30, batch\_size=32, verbose=0)

# Predict and plot

y\_pred = model.predict(X\_test)

y\_pred = y\_pred \* std + mean

y\_test = y\_test \* std + mean

plt.plot(y\_test, label='Actual')

plt.plot(y\_pred, label='Predicted')

plt.legend(); plt.title("Actual vs Predicted Temperatures")

plt.show()

import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load and normalize data

temps = pd.read\_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv')['Temp'].values.astype(np.float32)

mean, std = temps.mean(), temps.std()

temps = (temps - mean) / std

# Create sequences

def make\_seq(data, window=7):

X, y = [], []

for i in range(len(data) - window):

X.append(data[i:i+window])

y.append(data[i+window])

return np.array(X).reshape(-1, window, 1), np.array(y)

X, y = make\_seq(temps)

split = int(0.8 \* len(X))

X\_train, X\_test, y\_train, y\_test = X[:split], X[split:], y[:split], y[split:]

# Build and train model

model = tf.keras.Sequential([

tf.keras.layers.LSTM(64, input\_shape=(7, 1)),

tf.keras.layers.Dense(1)

])

model.compile(optimizer='adam', loss='mse')

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=30, batch\_size=32, verbose=0)

# Predict and plot

y\_pred = model.predict(X\_test)

plt.plot(y\_test \* std + mean, label='Actual')

plt.plot(y\_pred.flatten() \* std + mean, label='Predicted')

plt.legend(); plt.title('LSTM: Actual vs Predicted Temp'); plt.show()

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D

# Load and normalize data

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

x\_train, x\_test = x\_train[..., np.newaxis]/255., x\_test[..., np.newaxis]/255.

# Add Gaussian noise

noise = 0.5

x\_train\_noisy = np.clip(x\_train + noise \* np.random.randn(\*x\_train.shape), 0., 1.)

x\_test\_noisy = np.clip(x\_test + noise \* np.random.randn(\*x\_test.shape), 0., 1.)

# Build autoencoder

model = Sequential([

Conv2D(32, (3,3), activation='relu', padding='same', input\_shape=(28,28,1)),

MaxPooling2D((2,2), padding='same'),

Conv2D(64, (3,3), activation='relu', padding='same'),

MaxPooling2D((2,2), padding='same'),

Conv2D(64, (3,3), activation='relu', padding='same'),

UpSampling2D((2,2)),

Conv2D(32, (3,3), activation='relu', padding='same'),

UpSampling2D((2,2)),

Conv2D(1, (3,3), activation='sigmoid', padding='same')

])

model.compile(optimizer='adam', loss='binary\_crossentropy')

model.fit(x\_train\_noisy, x\_train, epochs=5, batch\_size=128, validation\_data=(x\_test\_noisy, x\_test))

# Predict

decoded = model.predict(x\_test\_noisy)

# Show results

n = 10

plt.figure(figsize=(18, 6))

for i in range(n):

for j, img in enumerate([x\_test\_noisy, decoded, x\_test]):

ax = plt.subplot(3, n, i + j\*n + 1)

plt.imshow(img[i].reshape(28, 28), cmap='gray')

ax.axis('off')

plt.tight\_layout()

plt.show()