```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, roc_auc_score, roc_curve

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam, RMSprop

# 1. Load dataset
df =
pd.read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabet
es.data.csv',
                 header=None)
df.columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
              'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

# 2. Preprocess
X = df.drop('Outcome', axis=1)
y = df['Outcome']

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, random_state=42)

# 3. Define a simple NN model
def build_model(optimizer):
    model = Sequential([
        Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
        Dense(8, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# 4. Train using Adam
adam_model = build_model(Adam(learning_rate=0.001))
adam_history = adam_model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, verbose=0)

# 5. Train using RMSProp
rms_model = build_model(RMSprop(learning_rate=0.001))
```

```
rms_history = rms_model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=32,
verbose=0)

# 6. Evaluate on test set
def evaluate_model(model, name):
    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    print(f"\n{name} Optimizer Evaluation:")
    print(classification_report(y_test, y_pred, digits=4))
    auc = roc_auc_score(y_test, model.predict(X_test))
    print(f"AUC: {auc:.4f}")
    return auc

auc_adam = evaluate_model(adam_model, "Adam")
auc_rms = evaluate_model(rms_model, "RMSProp")

# 7. Plot training history
def plot_history(histories, key='accuracy'):
    plt.figure(figsize=(12, 5))
    for name, history in histories:
        plt.plot(history.history[key], label=f'{name} {key}')
    plt.title(f'Training {key}')
    plt.xlabel('Epochs')
    plt.ylabel(key)
    plt.legend()
    plt.grid(True)
    plt.show()

plot_history([('Adam', adam_history), ('RMSProp', rms_history)], key='accuracy')
plot_history([('Adam', adam_history), ('RMSProp', rms_history)], key='loss')

# 8. Compare ROC Curves
def plot_roc(model, name):
    y_probs = model.predict(X_test)
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc_score(y_test, y_probs):.2f})')

plt.figure(figsize=(8, 6))
plot_roc(adam_model, "Adam")
plot_roc(rms_model, "RMSProp")
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
—---------------------------------
import os
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# 1. Define paths
base_dir = "path_to_dataset"  # Replace with actual path
train_dir = os.path.join(base_dir, "train")
val_dir = os.path.join(base_dir, "val")
test_dir = os.path.join(base_dir, "test")

# 2. Parameters
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 10

# 3. Data Augmentation
train_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    shear_range=0.1,
    horizontal_flip=True
)

val_gen = ImageDataGenerator(rescale=1./255)
test_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')
test_data = test_gen.flow_from_directory(test_dir, target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False)

# 4. Load ResNet50
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze initial layers

# 5. Add Custom Classification Head
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(256, activation='relu')(x)
```

```
x = Dropout(0.5)(x)
output = Dense(train_data.num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output)

# 6. Compile & Train
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

history = model.fit(train_data, validation_data=val_data, epochs=EPOCHS)

# 7. Fine-tune (unfreeze top layers)
base_model.trainable = True
for layer in base_model.layers[:100]:
    layer.trainable = False

model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])

fine_tune_epochs = 5
history_finetune = model.fit(train_data, validation_data=val_data, epochs=fine_tune_epochs)

# 8. Evaluate
loss, acc = model.evaluate(test_data)
print(f"Test Accuracy: {acc*100:.2f}%")

# Classification Report
y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)
print(classification_report(y_true, y_pred, target_names=test_data.class_indices.keys()))

# 9. Visualize Training
def plot_metrics(history, label):
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='Train')
    plt.plot(history.history['val_accuracy'], label='Val')
    plt.title(f'{label} Accuracy')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train')
    plt.plot(history.history['val_loss'], label='Val')
    plt.title(f'{label} Loss')
    plt.legend()
    plt.show()

plot_metrics(history, "Initial Training")
plot_metrics(history_finetune, "Fine-tuning")
```

```
# 10. Save Model
model.save("glioma_resnet_model.h5")
—-------------------------------------------------------
# --- Sneaker Classification with MobileNet-v1 and GoogLeNet (InceptionV3) ---
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2, InceptionV3
from tensorflow.keras.applications.mobilenet import preprocess_input as mobilenet_preprocess
from tensorflow.keras.applications.inception_v3 import preprocess_input as
inception_preprocess
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Input
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
import numpy as np
import os

# Change this to your dataset path
DATASET_PATH = 'sneaker_dataset'  # folder with subfolders like Nike/, Adidas/, etc.

IMG_SIZE = (128, 128)
BATCH_SIZE = 32
EPOCHS = 5

# ---- Common ImageDataGenerator ----
def create_data_generators(preprocess_func):
    datagen = ImageDataGenerator(
        preprocessing_function=preprocess_func,
        validation_split=0.2,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True
    )
    train_gen = datagen.flow_from_directory(
        DATASET_PATH,
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='training'
    )
    val_gen = datagen.flow_from_directory(
        DATASET_PATH,
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='validation'
```

```
    )
    return train_gen, val_gen


# ---- Model Builder ----
def build_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    output = Dense(num_classes, activation='softmax')(x)
    return Model(inputs=base_model.input, outputs=output)


# ---- Train and Evaluate ----
def train_and_evaluate(model, train_gen, val_gen, name="Model"):
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    print(f"\nTraining {name}...")
    model.fit(train_gen, validation_data=val_gen, epochs=EPOCHS)

    print(f"\nEvaluating {name}...")
    val_gen.reset()
    preds = model.predict(val_gen)
    y_pred = np.argmax(preds, axis=1)
    y_true = val_gen.classes

    print(f"\nClassification Report for {name}:\n")
    print(classification_report(y_true, y_pred, target_names=list(val_gen.class_indices.keys())))

    return model


# =====================
# Train MobileNet-v1
# =====================
train_gen_m, val_gen_m = create_data_generators(mobilenet_preprocess)
mobilenet_base = MobileNetV2(input_shape=IMG_SIZE + (3,), include_top=False,
weights='imagenet')
mobilenet_base.trainable = False
mobilenet_model = build_model(mobilenet_base, num_classes=train_gen_m.num_classes)
mobilenet_model = train_and_evaluate(mobilenet_model, train_gen_m, val_gen_m,
"MobileNetV2")

# =====================
# Train GoogLeNet (InceptionV3)
# =====================
train_gen_g, val_gen_g = create_data_generators(inception_preprocess)
```

```
googlenet_base = InceptionV3(input_shape=IMG_SIZE + (3,), include_top=False,
weights='imagenet')
googlenet_base.trainable = False
googlenet_model = build_model(googlenet_base, num_classes=train_gen_g.num_classes)
googlenet_model = train_and_evaluate(googlenet_model, train_gen_g, val_gen_g, "GoogLeNet
(InceptionV3)")

# =====================
# Compare Computational Efficiency
# =====================
def print_model_summary_stats(model, name):
    total_params = model.count_params()
    print(f"\n{name} - Total Trainable Parameters: {total_params}")

print_model_summary_stats(mobilenet_model, "MobileNetV2")
print_model_summary_stats(googlenet_model, "GoogLeNet (InceptionV3)")
```

—--

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, classification_report
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Input, LSTM, RepeatVector, TimeDistributed

# --- Load dataset (use your path) ---
df = pd.read_csv('winequality-red.csv')  # Or 'winequality-white.csv'

# ------------------------------
# 1. AUTOENCODER for Anomaly Detection
# ------------------------------
print("\n--- Autoencoder for Anomaly Detection ---")

# Scale data between 0 and 1
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(df.drop("quality", axis=1))

# Split data
X_train, X_test = train_test_split(X_scaled, test_size=0.2, random_state=42)

# Build autoencoder
input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoded = Dense(16, activation='relu')(input_layer)
encoded = Dense(8, activation='relu')(encoded)
decoded = Dense(16, activation='relu')(encoded)
decoded = Dense(input_dim, activation='sigmoid')(decoded)
```

```
autoencoder = Model(inputs=input_layer, outputs=decoded)

autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X_train, X_train, epochs=50, batch_size=32, validation_split=0.1, verbose=0)

# Anomaly scores
reconstructions = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - reconstructions, 2), axis=1)
threshold = np.percentile(mse, 95)
anomalies = mse > threshold

print(f"Anomalies Detected: {np.sum(anomalies)} / {len(X_test)}")

# ------------------------------
# 2. WINE QUALITY PREDICTION (MLP Regression)
# ------------------------------
print("\n--- Wine Quality Prediction (MLP Regression) ---")

X = df.drop("quality", axis=1)
y = df["quality"]
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Build simple regression model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)  # Output: quality score
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1, verbose=0)

y_pred = model.predict(X_test).flatten()
print("MSE:", mean_squared_error(y_test, y_pred))

# ------------------------------
# 3. RNN (LSTM) for Fermentation Process Modeling
# ------------------------------
print("\n--- RNN (LSTM) for Time-Series Modeling ---")

# For demonstration, create time sequences artificially
# Normally, use temporal fermentation data

timesteps = 5
features = X_train.shape[1]

# Artificial reshaping into sequences
def to_sequences(X, y, timesteps):
```

```
    Xs, ys = [], []
    for i in range(len(X) - timesteps):
        Xs.append(X[i:i+timesteps])
        ys.append(y[i+timesteps])
    return np.array(Xs), np.array(ys)

X_seq, y_seq = to_sequences(X_scaled, y.values, timesteps)
X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(X_seq, y_seq,
test_size=0.2, random_state=42)

# Build LSTM
model_lstm = Sequential([
    LSTM(64, input_shape=(timesteps, features)),
    Dense(32, activation='relu'),
    Dense(1)
])

model_lstm.compile(optimizer='adam', loss='mse')
model_lstm.fit(X_train_seq, y_train_seq, epochs=30, batch_size=32, validation_split=0.1,
verbose=0)

lstm_pred = model_lstm.predict(X_test_seq).flatten()
print("LSTM MSE:", mean_squared_error(y_test_seq, lstm_pred))
```