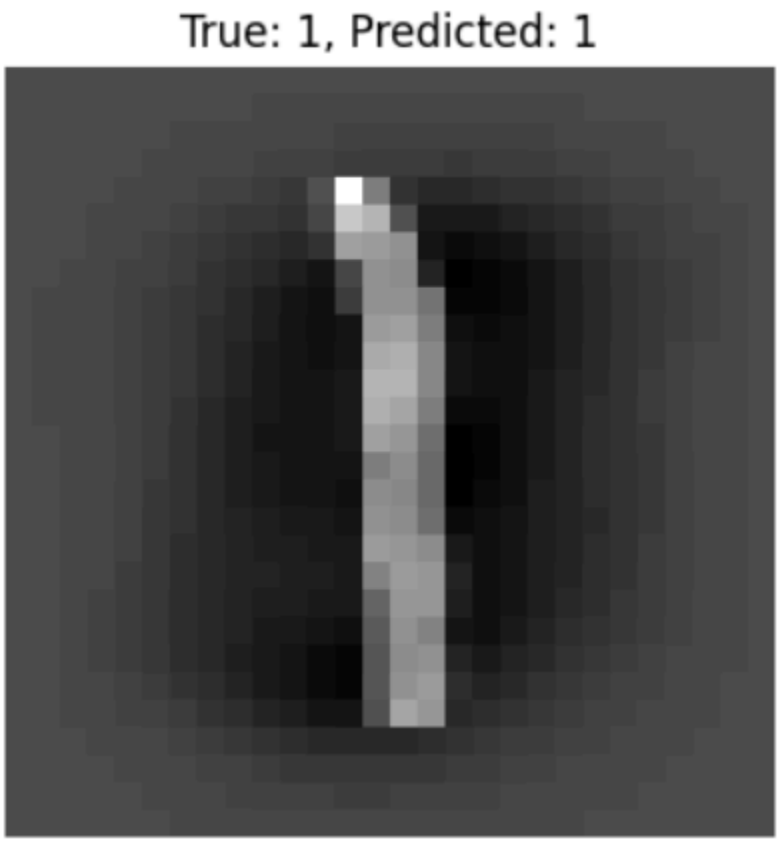
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import warnings
from sklearn.exceptions import ConvergenceWarning

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(-1, 28 * 28).astype(np.float32)
X_test = X_test.reshape(-1, 28 * 28).astype(np.float32)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

warnings.filterwarnings("ignore", category=ConvergenceWarning)

mlp = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=50,
alpha=1e-4,
                    solver='adam', random_state=1,
learning_rate_init=0.001)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
```

OUTPUT

Accuracy: 0.9792

PREDICTION PLOT

```
idx = np.random.randint(0, len(X_test))
image = X_test[idx].reshape(28, 28)
true_label = y_test[idx]
predicted_label = y_pred[idx]
plt.figure(figsize=(4, 4))
plt.imshow(image, cmap='gray')
plt.title(f"True: {true_label}, Predicted: {predicted_label}")
plt.axis("off")
plt.show()
```



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import SGD, Adam
import matplotlib.pyplot as plt
import numpy as np

(x_train, y_train), (x_test, y_test)
=keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0



def build_model(optimizer):
  model = Sequential([
  Flatten(input_shape=(28, 28)),
  Dense(128, activation='relu'),
  Dense(64, activation='relu'),
  Dense(10, activation='softmax')
  ])
  model.compile(optimizer=optimizer,
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])
  return model



optimizers = {
'Momentum-Based GD': SGD(learning_rate=0.001,
momentum=0.9),
'Stochastic GD': SGD(learning_rate=0.001),
'Adam': Adam(learning_rate=0.001),
}



history = {}
test_accuracies = {}
```

```
for name, optimizer in optimizers.items():
  print(f'Training with {name} optimizer...')
  model = build_model(optimizer)
  hist = model.fit(x_train, y_train, epochs=15,
  validation_data=(x_test, y_test), verbose=1)
  history[name] = hist
  test_loss, test_acc = model.evaluate(x_test, y_test,
  verbose=0)
  test_accuracies[name] = test_acc
```

```
plt.figure(figsize=(12, 5))
for name, hist in history.items():
  plt.plot(hist.history['val_accuracy'], label=f'{name} Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Validation Accuracy')
plt.legend()
plt.title('Comparison of Optimizers')
plt.show()
```

```
for name, acc in test_accuracies.items():
  print(f'{name} Test Accuracy: {acc:.4f}')
```

OUTPUT:

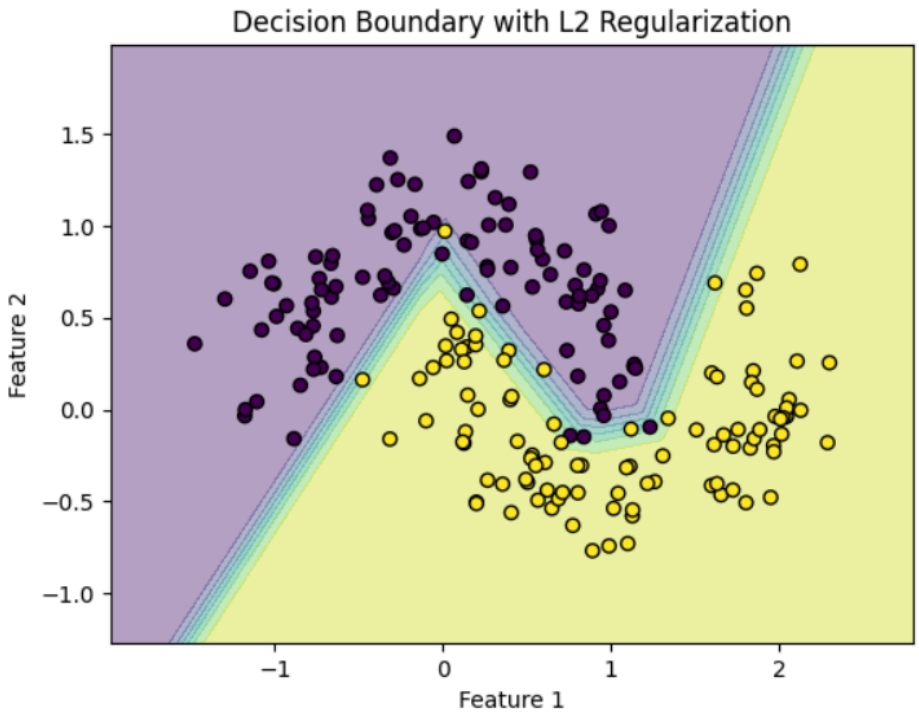
Momentum-Based GD Test Accuracy: 0.9703
Stochastic GD Test Accuracy: 0.9239
Adam Test Accuracy: 0.9778

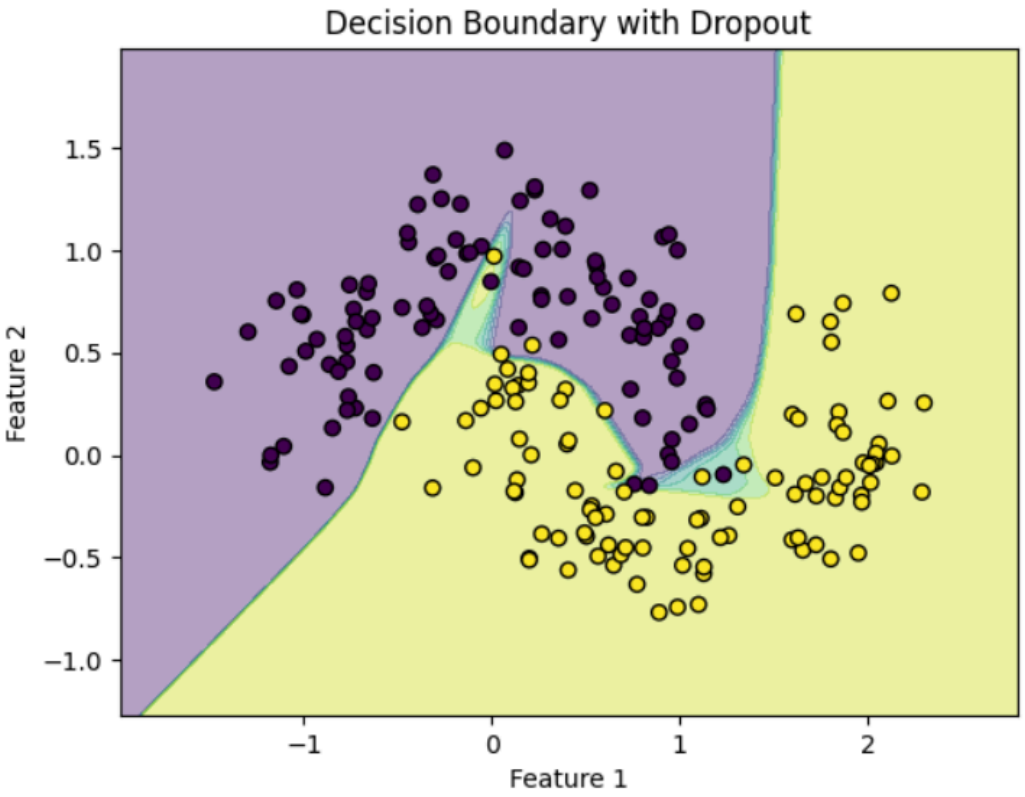
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers

# Generate dataset
X, y = make_moons(n_samples=200, noise=0.2, random_state=42)

# Function to create and train a model
def create_train_model(use_regularization=False, use_dropout=False,
use_both=False):
    model = Sequential()

    if use_both:  # L2 Regularization + Dropout
        model.add(Dense(128, activation='relu', input_shape=(2,),
kernel_regularizer=regularizers.l2(0.01)))
        model.add(Dropout(0.3))
        model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))
        model.add(Dropout(0.3))

    elif use_regularization:
        model.add(Dense(128, activation='relu', input_shape=(2,),
kernel_regularizer=regularizers.l2(0.01)))
        model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.01)))

    elif use_dropout:
        model.add(Dense(128, activation='relu', input_shape=(2,)))
        model.add(Dropout(0.3))
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.3))

    model.add(Dense(1, activation='sigmoid'))
```

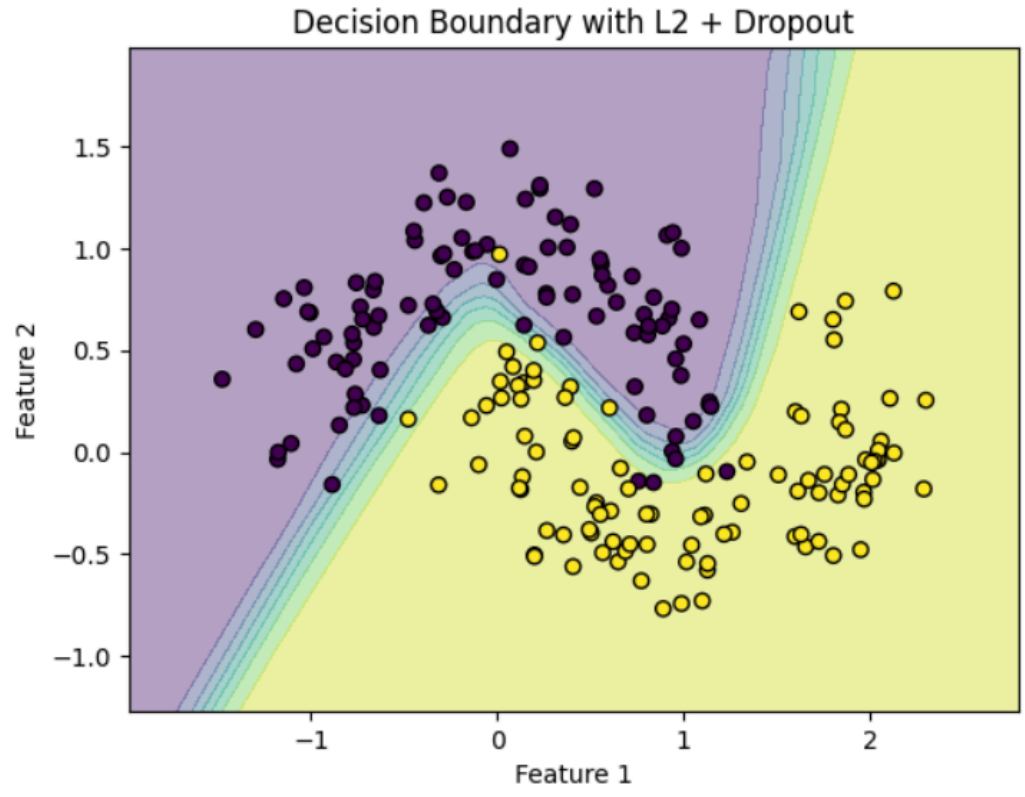
```
    # Compile model
    model.compile(optimizer=Adam(learning_rate=0.01),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    # Train model
    history = model.fit(X, y, epochs=1000, verbose=0,
validation_split=0.2)

    # Evaluate model
    loss, accuracy = model.evaluate(X, y, verbose=0)

    return model, accuracy, history

# Train all models
model_reg, acc_reg, history_reg =
create_train_model(use_regularization=True)
model_drop, acc_drop, history_drop =
create_train_model(use_dropout=True)
model_both, acc_both, history_both =
create_train_model(use_both=True)

# Print results
print(f"Accuracy with L2 Regularization: {acc_reg:.4f}")
print(f"Accuracy with Dropout: {acc_drop:.4f}")
print(f"Accuracy with L2 + Dropout: {acc_both:.4f}")

# Function to plot decision boundary
def plot_decision_boundary(model, title):
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.4)
```

```
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title(title)
    plt.show()

# Plot decision boundaries
plot_decision_boundary(model_reg, "Decision Boundary with L2 Regularization")
plot_decision_boundary(model_drop, "Decision Boundary with Dropout")
plot_decision_boundary(model_both, "Decision Boundary with L2 + Dropout")
```

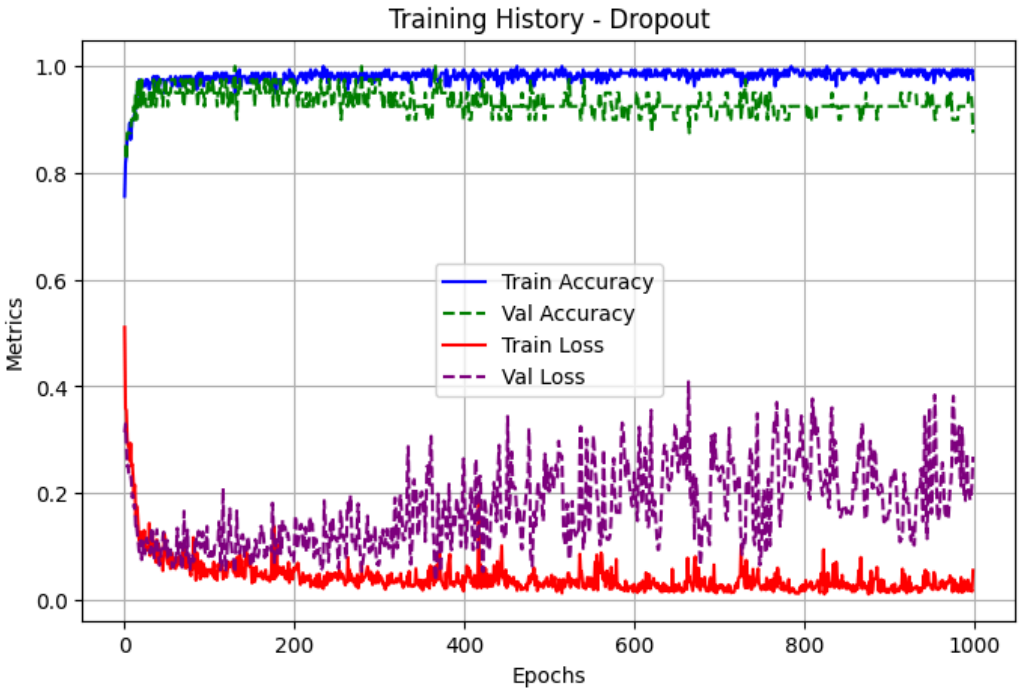
OUTPUTS :



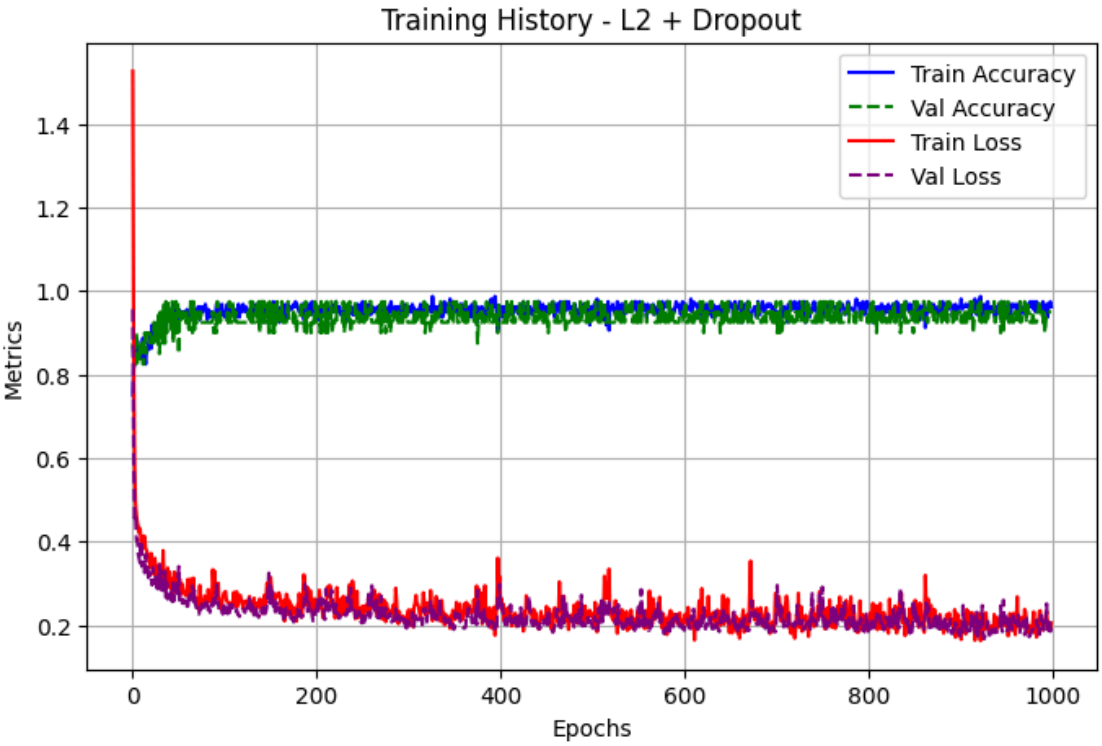

Decision Boundary with Dropout
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2


Decision Boundary with L2 + Dropout
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2

```
def plot_training_history_separate(histories, labels):
    colors = ['blue', 'green', 'red', 'purple']
    for history, label in zip(histories, labels):
        plt.figure(figsize=(8, 5))
        plt.plot(history.history['accuracy'], label='Train Accuracy',
color=colors[0])
        plt.plot(history.history['val_accuracy'], linestyle="dashed",
label='Val Accuracy', color=colors[1])
        plt.plot(history.history['loss'], label='Train Loss',
color=colors[2])
        plt.plot(history.history['val_loss'], linestyle="dashed",
label='Val Loss', color=colors[3])
        plt.xlabel("Epochs")
        plt.ylabel("Metrics")
        plt.legend()
        plt.title(f"Training History - {label}")
        plt.grid(True)
        plt.show()
        final_epoch = len(history.history['accuracy']) - 1 )
```




Training History - L2 Regularization
Train Accuracy
Val Accuracy
Train Loss
Val Loss
Metrics
Epochs
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000


Training History - L2 + Dropout
Train Accuracy
Val Accuracy
Train Loss
Val Loss
Metrics
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000