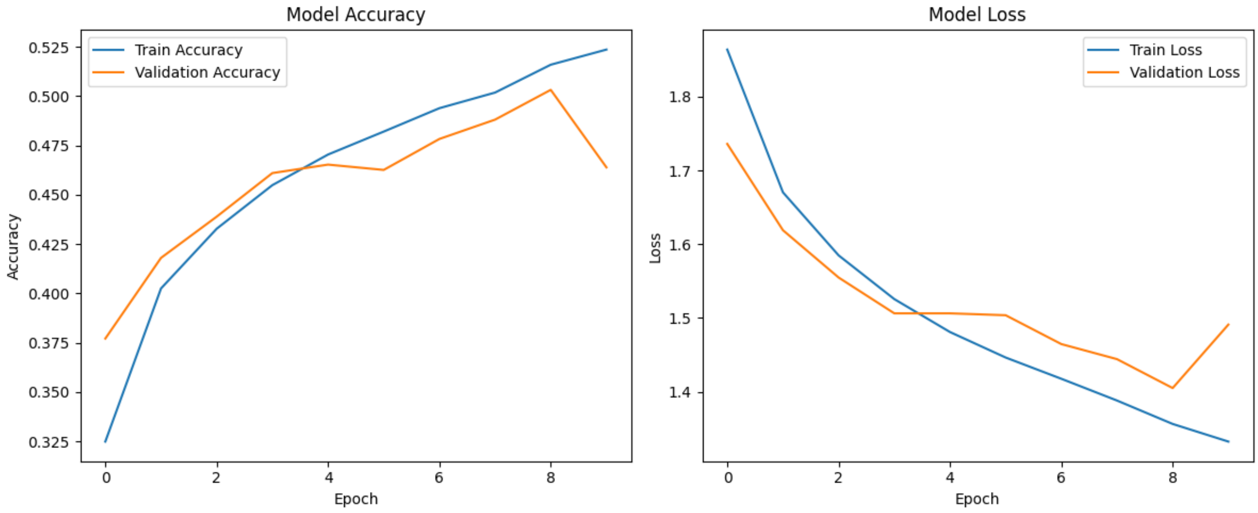
PROGRAM

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
model = Sequential([
    Flatten(input_shape=(32, 32, 3)),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train,epochs=10,batch_size=64,validation_data=(x_test, y_test))
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.tight_layout()

plt.show()
```

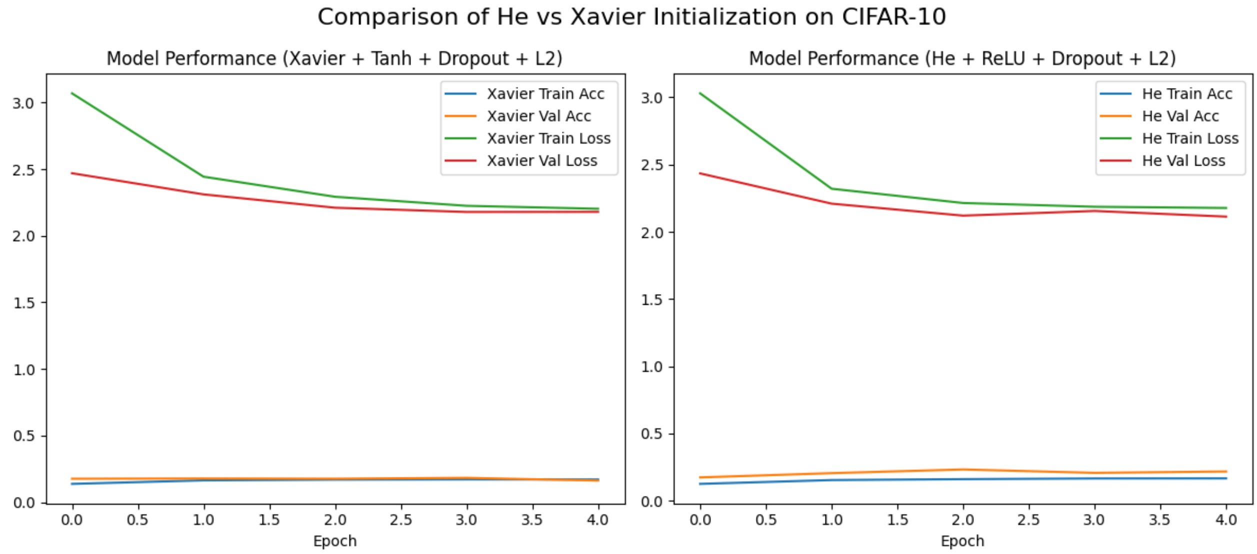
OUTPUT



PROGRAM

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.regularizers import l2

cifar10 = tf.keras.datasets.cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

y_train = y_train.flatten()

y_test = y_test.flatten()

reg = l2(0.001)

he_init = tf.keras.initializers.HeNormal(seed=0)

xav_init = tf.keras.initializers.GlorotNormal(seed=0)


model_he = Sequential()

model_he.add(Flatten(input_shape=(32, 32, 3)))

model_he.add(Dense(512, activation='relu', kernel_initializer=he_init, kernel_regularizer=reg))

model_he.add(Dropout(0.5))

model_he.add(Dense(256, activation='relu', kernel_initializer=he_init, kernel_regularizer=reg))

model_he.add(Dropout(0.5))

model_he.add(Dense(128, activation='relu', kernel_initializer=he_init, kernel_regularizer=reg))

model_he.add(Dropout(0.5))

model_he.add(Dense(10, activation='softmax'))

model_he.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history_he = model_he.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5,
batch_size=64)


model_xav = Sequential()

model_xav.add(Flatten(input_shape=(32, 32, 3)))

model_xav.add(Dense(512, activation='tanh', kernel_initializer=xav_init, kernel_regularizer=reg))

model_xav.add(Dropout(0.5))
model_xav.add(Dense(256, activation='tanh', kernel_initializer=xav_init, kernel_regularizer=reg))
model_xav.add(Dropout(0.5))
model_xav.add(Dense(128, activation='tanh', kernel_initializer=xav_init, kernel_regularizer=reg))
model_xav.add(Dropout(0.5))
model_xav.add(Dense(10, activation='softmax'))
model_xav.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history_xav = model_xav.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5,
batch_size=64)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history_xav.history['accuracy'], label='Xavier Train Acc')
plt.plot(history_xav.history['val_accuracy'], label='Xavier Val Acc')
plt.plot(history_xav.history['loss'], label='Xavier Train Loss')
plt.plot(history_xav.history['val_loss'], label='Xavier Val Loss')
plt.title("Model Performance (Xavier + Tanh + Dropout + L2)")
plt.xlabel("Epoch")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_he.history['accuracy'], label='He Train Acc')
plt.plot(history_he.history['val_accuracy'], label='He Val Acc')
plt.plot(history_he.history['loss'], label='He Train Loss')
plt.plot(history_he.history['val_loss'], label='He Val Loss')
plt.title("Model Performance (He + ReLU + Dropout + L2)")
plt.xlabel("Epoch")
plt.legend()
plt.tight_layout()
plt.suptitle("Comparison of He vs Xavier Initialization on CIFAR-10", fontsize=16, y=1.05)
plt.show()
```

OUTPUT

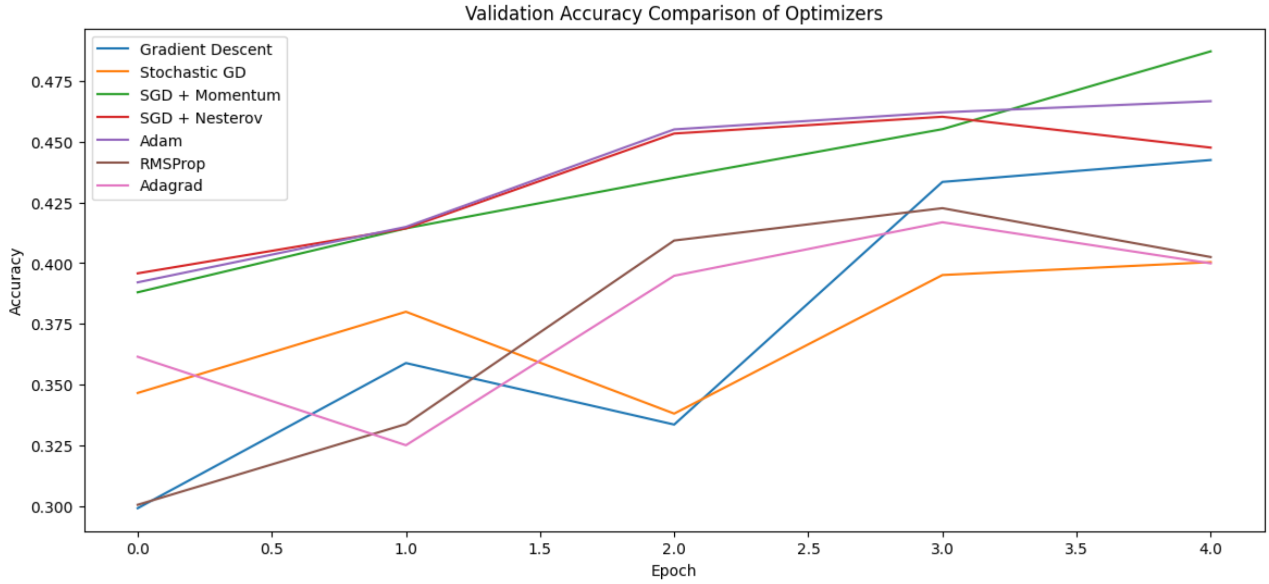


Comparison of He vs Xavier Initialization on CIFAR-10
Model Performance (Xavier + Tanh + Dropout + L2)
Xavier Train Acc
Xavier Val Acc
Xavier Train Loss
Xavier Val Loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch
Model Performance (He + ReLU + Dropout + L2)
He Train Acc
He Val Acc
He Train Loss
He Val Loss
Epoch

PROGRAM

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
def build_model():
    model = Sequential([
        Flatten(input_shape=(32, 32, 3)),
        Dense(512, activation='relu'),
        Dense(256, activation='relu'),
        Dense(128, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])
    return model
optimizers = {
    "Gradient Descent": tf.keras.optimizers.SGD(learning_rate=0.01),
    "Stochastic GD": tf.keras.optimizers.SGD(learning_rate=0.01),  # mini-batch SGD (default in Keras)
    "SGD + Momentum": tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
    "SGD + Nesterov": tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=True),
    "Adam": tf.keras.optimizers.Adam(learning_rate=0.001),
    "RMSProp": tf.keras.optimizers.RMSprop(learning_rate=0.001),
    "Adagrad": tf.keras.optimizers.Adagrad(learning_rate=0.01),
}
histories = {}
for name, opt in optimizers.items():
    print(f"\nTraining with {name} optimizer...")
    model = build_model()
    model.compile(optimizer=opt, loss="categorical_crossentropy", metrics=["accuracy"])
    history = model.fit(x_train, y_train,epochs=5,batch_size=64,validation_data=(x_test,
                    y_test),verbose=1)
    histories[name] = history
plt.figure(figsize=(14, 6))
for name, history in histories.items():
    plt.plot(history.history['val_accuracy'], label=name)
plt.title("Validation Accuracy Comparison of Optimizers")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

OUTPUT



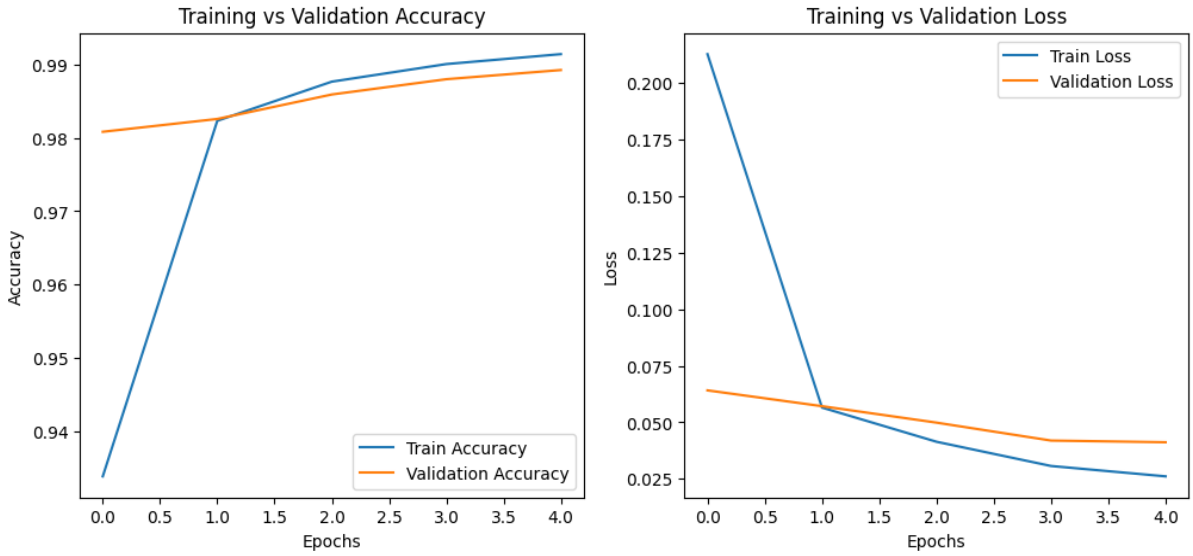
EXP NO :6

PROGRAM

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = (train_images / 255.0)[..., tf.newaxis]
test_images = (test_images / 255.0)[..., tf.newaxis]
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',loss='categorical_crossentropy',  metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=5,batch_size=64,validation_split=0.2)
_, test_acc = model.evaluate(test_images, test_labels)
print(f"\nTest Accuracy : {round(test_acc * 100, 4)}%")
plt.figure(figsize=(12, 5))
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.title('Training vs Validation Accuracy')

plt.legend()

# Loss plot

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training vs Validation Loss')

plt.legend()

plt.show()
```

OUTPUT

```
Test Accuracy : 98.97%
```

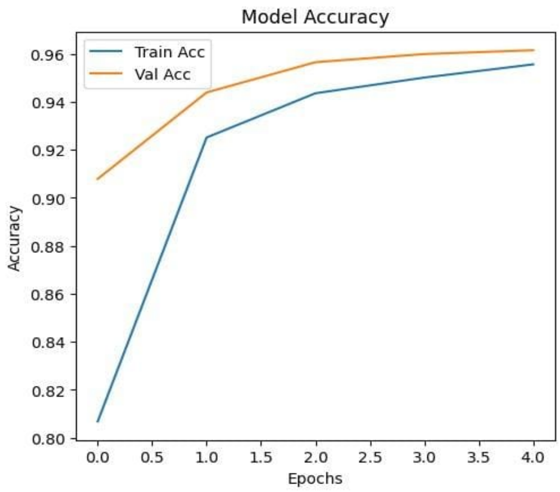


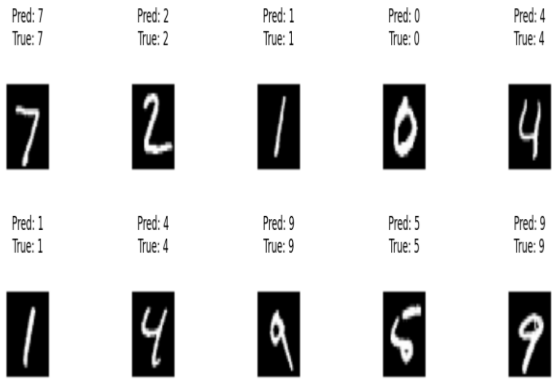
PROGRAM

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras import models, layers

from tensorflow.keras.applications import VGG16

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

y_train = tf.keras.utils.to_categorical(y_train, 10)

y_test = tf.keras.utils.to_categorical(y_test, 10)

npad = ((0, 0), (10, 10), (10, 10))

x_train = np.pad(x_train, npad, 'constant', constant_values=255)

x_test = np.pad(x_test, npad, 'constant', constant_values=255)

x_train = np.array([np.stack((img, img, img), axis=-1) for img in x_train])

x_test = np.array([np.stack((img, img, img), axis=-1) for img in x_test])

x_train = x_train.astype("float32") / 255.0

x_test = x_test.astype("float32") / 255.0

vgg_model = VGG16(weights="imagenet", include_top=False, input_shape=(48, 48, 3))

vgg_model.trainable = False  # freeze pretrained layers

model = models.Sequential([

   vgg_model,

   layers.Flatten(),

   layers.Dense(128, activation="relu"),

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

])

model.compile(optimizer="adam",  loss="categorical_crossentropy",  metrics=["accuracy"])

history = model.fit(x_train, y_train,epochs=5, batch_size=64,validation_data=(x_test,
y_test),verbose=1)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)

print(f"\n Test Accuracy: {test_acc:.4f}")

print(f" Test Loss: {test_loss:.4f}")

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label="Train Acc")
plt.plot(history.history['val_accuracy'], label="Val Acc")
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
predictions = model.predict(x_test[:10])
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_test[:10], axis=1)
plt.figure(figsize=(12, 4))
for i in range(10):
   plt.subplot(2, 5, i+1)
   plt.imshow(x_test[i])
   plt.title(f"Pred: {predicted_classes[i]}\nTrue: {true_classes[i]}")
   plt.axis("off")
plt.tight_layout()
plt.show()
```

OUTPUT

Test Accuracy: 0.9615



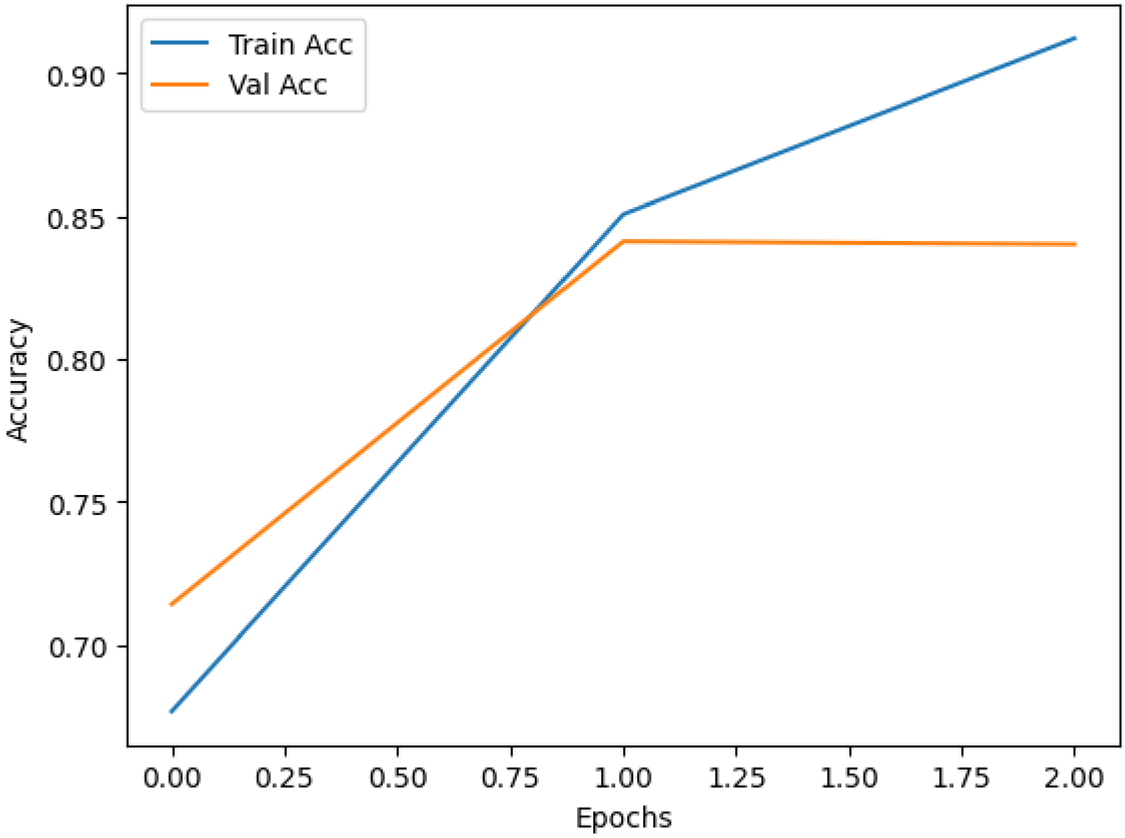


PROGRAM

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
import matplotlib.pyplot as plt
max_features = 10000
maxlen = 500
batch_size = 32
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
print(f"Training sequences: {len(input_train)}")
print(f"Test sequences: {len(input_test)}")
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,epochs=5, batch_size=batch_size,validation_split=0.2)
test_loss, test_acc = model.evaluate(input_test, y_test, verbose=0)
print(f"\nTest Accuracy: {test_acc:.4f}")
print(f" Test Loss: {test_loss:.4f}")


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['acc'], label="Train Acc")
plt.plot(history.history['val_acc'], label="Val Acc")
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")

plt.legend()

plt.show()
```

<u>OUTPUT</u>

Test Accuracy: 0.8326

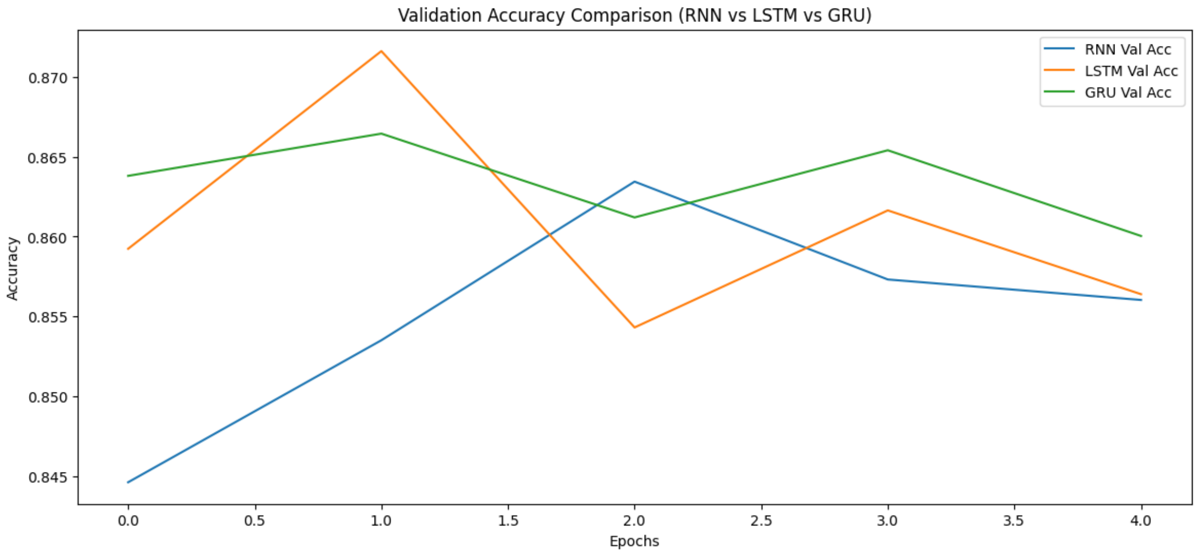


PROGRAM

```
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
max_features = 10000
maxlen = 200
(xtrain, ytrain), (xtest, ytest) = imdb.load_data(num_words=max_features)
xtrain = pad_sequences(xtrain, maxlen=maxlen)
xtest = pad_sequences(xtest, maxlen=maxlen)
def build_model(cell_type="RNN"):
   model = Sequential()
   model.add(Embedding(max_features, 32, input_length=maxlen))
    if cell_type == "RNN":
      model.add(SimpleRNN(32, activation='relu'))
   elif cell_type == "LSTM":
      model.add(LSTM(32))
   elif cell_type == "GRU":
      model.add(GRU(32))
   model.add(Dense(1, activation="sigmoid"))
   model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
   return model

histories = {}
for cell in ["RNN", "LSTM", "GRU"]:
   print(f"\nTraining {cell} model...")
   model = build_model(cell)
   history = model.fit(
      xtrain, ytrain,epochs=5,batch_size=64,validation_data=(xtest, ytest),verbose=1 )
   histories[cell] = history
```

```
plt.figure(figsize=(14,6))

for cell, history in histories.items():

    plt.plot(history.history['val_accuracy'], label=f"{cell} Val Acc")

plt.title("Validation Accuracy Comparison (RNN vs LSTM vs GRU)")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend()

plt.show()
```

OUTPUT


Validation Accuracy Comparison (RNN vs LSTM vs GRU)
RNN Val Acc
LSTM Val Acc
GRU Val Acc
Accuracy
Epochs