6. TRANSFER LEARNING MODEL ON A REVIEW DATASET

PROGRAM CODE:-

```
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tokenizer =AutoTokenizer.from_pretrained("nlptown/bert-base-multilingual-uncased-sentiment")
model=AutoModelForSequenceClassification.from_pretrained("nlptown/bert-base-multilingual-uncased-sentiment").to(device)

def predict_review(review):
    enc = tokenizer(review, return_tensors='pt', truncation=True, padding=True).to(device)
    with torch.no_grad():
        output = model(**enc).logits.softmax(dim=1)
        sentiment = output.argmax().item()
    labels = ["Very Negative", "Negative", "Neutral", "Positive", "Very Positive"]
    print(f"Sentiment: {labels[sentiment]}")

review = input("Enter a review: ")
predict_review(review)
```

OUTPUT

```
tokenizer_config.json: 100%   39.0/39.0 [00:00<00:00, 577B/s]
config.json: 100%   953/953 [00:00<00:00, 23.5kB/s]
vocab.txt: 100%   872k/872k [00:00<00:00, 18.1MB/s]
special_tokens_map.json: 100%   112/112 [00:00<00:00, 3.39kB/s]
model.safetensors: 100%   669M/669M [00:03<00:00, 176MB/s]
Enter a review: it was really good
Sentiment: Very Positive
```

7.AUTOENCODER USING OXFORD FLOWER DATASET

PROGRAM CODE

```
# Import TensorFlow for deep learning
import tensorflow as tf
# Import TensorFlow Datasets to load and manage datasets
import tensorflow_datasets as tfds
# Import Matplotlib for visualizing images
import matplotlib.pyplot as plt

# Load the Oxford Flowers 102 dataset with images and labels
# 'as_supervised=True' gives us (image, label) pairs
# 'with_info=True' provides dataset information like label names and splits
dataset, info = tfds.load('oxford_flowers102', as_supervised=True, with_info=True)

# Define the image size to which all images will be resized
IMG_SIZE = 128  # Resize all images to 128x128 for simplicity

# Define a function to preprocess each image
def preprocess_image(image, label):
    # Resize the image to IMG_SIZE x IMG_SIZE
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    # Normalize the pixel values to the range [0, 1]
    image = image / 255.0
    # Return the image as both input and target (for autoencoder training)
    return image, image

# Apply the preprocessing function to the training data, batch it, and prefetch for performance
train_data = dataset['train'].map(preprocess_image).batch(32).prefetch(tf.data.AUTOTUNE)
# Apply the preprocessing function to the test data, batch it, and prefetch for performance
test_data = dataset['test'].map(preprocess_image).batch(32).prefetch(tf.data.AUTOTUNE)

# Define the Autoencoder using the Sequential API
autoencoder = tf.keras.Sequential([
    # ENCODER
    # First convolutional layer: 32 filters, 3x3 kernel, ReLU activation
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(IMG_SIZE,
IMG_SIZE, 3)),
    # Downsample with MaxPooling
```

```
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),

    # Second convolutional layer: 64 filters, 3x3 kernel, ReLU activation
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    # Downsample again with MaxPooling
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),

    # Third convolutional layer: 128 filters, 3x3 kernel, ReLU activation
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    # Downsample once more with MaxPooling (latent space representation)
    tf.keras.layers.MaxPooling2D((2, 2), padding='same'),

    # DECODER
    # First transpose convolutional layer: Reverse of encoding, 128 filters, ReLU activation
    tf.keras.layers.Conv2DTranspose(128, (3, 3), activation='relu', padding='same'),
    # Upsample to increase resolution
    tf.keras.layers.UpSampling2D((2, 2)),

    # Second transpose convolutional layer: 64 filters, ReLU activation
    tf.keras.layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same'),
    # Upsample again
    tf.keras.layers.UpSampling2D((2, 2)),

    # Third transpose convolutional layer: 32 filters, ReLU activation
    tf.keras.layers.Conv2DTranspose(32, (3, 3), activation='relu', padding='same'),
    # Final upsample
    tf.keras.layers.UpSampling2D((2, 2)),

    # Output layer: 3 filters (RGB image), sigmoid activation to normalize output to [0, 1]
    tf.keras.layers.Conv2DTranspose(3, (3, 3), activation='sigmoid', padding='same')
])

# Compile the autoencoder model
# 'adam' is used as the optimizer for faster convergence
# 'binary_crossentropy' is used as the loss because the output is normalized to [0, 1]
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder using the training data
# Validation is done using the test data to monitor performance
autoencoder.fit(train_data, epochs=1, validation_data=test_data)
```

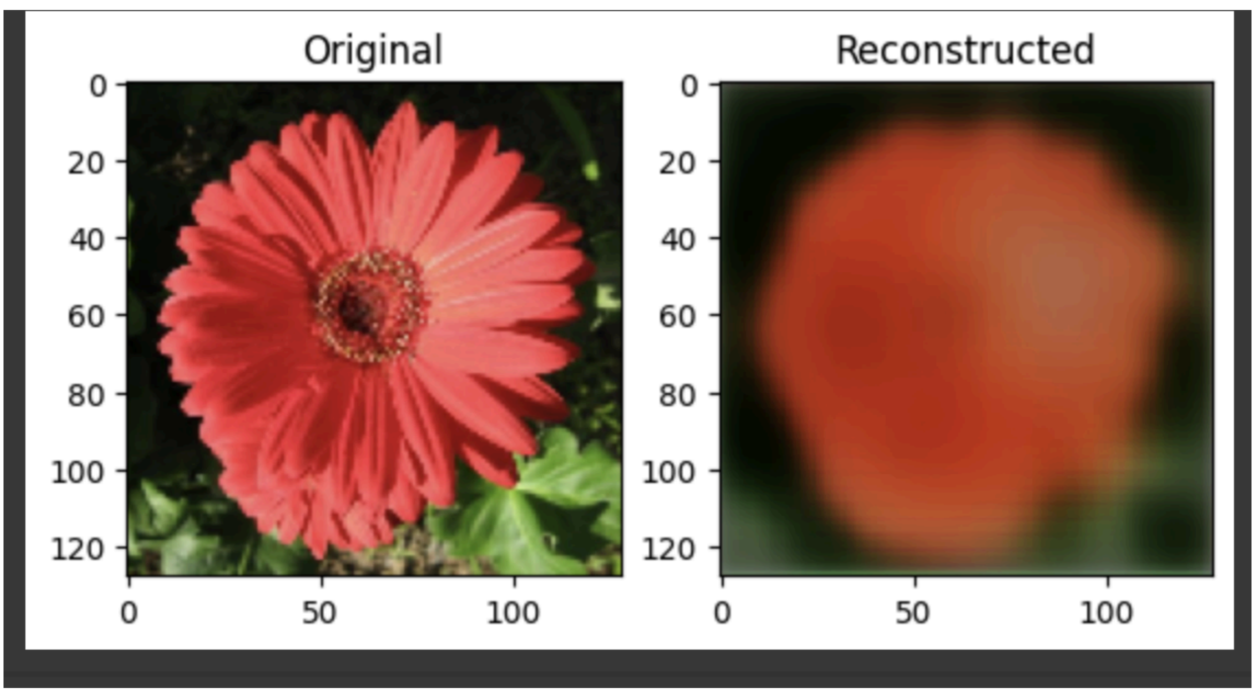
Get the first batch of test images (32 images), and select the first one from the batch
original_image = next(iter(test_data))[0][0]
# Use the trained autoencoder to reconstruct the selected image
reconstructed_image = autoencoder.predict(original_image[None, ...]) # Add batch dimension to image

# Display the original image on the left
plt.subplot(1, 2, 1)
plt.imshow(original_image)
plt.title("Original")

# Display the reconstructed image on the right
plt.subplot(1, 2, 2)
plt.imshow(reconstructed_image[0]) # Remove batch dimension
plt.title("Reconstructed")

# Show both images side by side
plt.show()

OUTPUT:



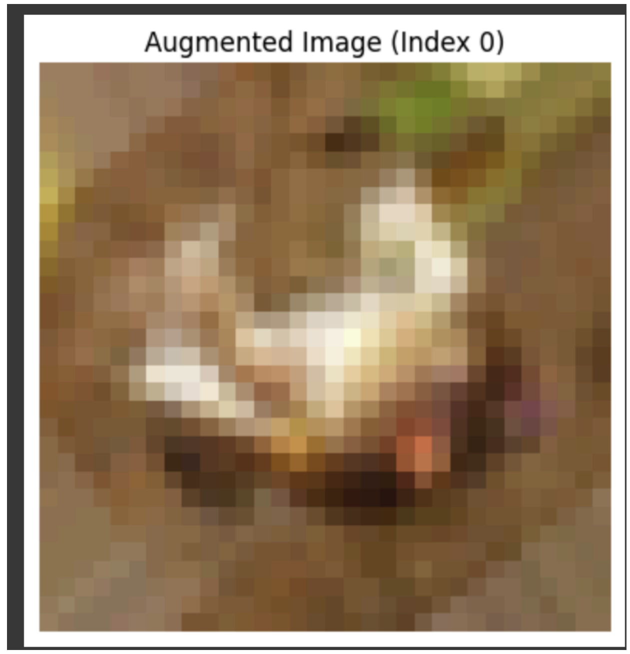
8.DATA AUGMENTATION USING CIFAR-10 DATASET

PROGRAM CODE:

```
# Import TensorFlow for data processing and model development
import tensorflow as tf
# Import Matplotlib for visualizing images
import matplotlib.pyplot as plt
# Import ImageDataGenerator from Keras for image augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load CIFAR-10 dataset (already included in TensorFlow)
# CIFAR-10 contains 60,000 32x32 color images in 10 classes
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Select the first image from the training dataset for demonstration
original_image = X_train[0]

# Reshape the image to add a batch dimension (required by ImageDataGenerator)
# Input shape must be (batch_size, height, width, channels)
image = original_image.reshape((1, 32, 32, 3))

 #Create an ImageDataGenerator object to define augmentation transformations
datagen = ImageDataGenerator(
    rotation_range=50,       # Randomly rotate images within a range of degrees
    zoom_range=0.2,          # Randomly zoom in or out up to 20%
    vertical_flip=True     # Randomly flip the image vertically
)

# Fit the data generator to the image (necessary when using data augmentation)
datagen.fit(image)

plt.imshow(original_image)    # Display the original image
plt.title("Original Image (Index 0)")  # Add a title to the plot
plt.axis('off')               # Remove axes for better visualization

augmented_images = datagen.flow(image)  # Create a generator
augmented_image = next(augmented_images)[0]

plt.imshow(augmented_image.astype('uint8'))    # Display the augmented image
```

```
plt.title("Augmented Image (Index 0)")  # Add a title to the plot
plt.axis('off')                 # Remove axes for better visualization
plt.show()
```

OUTPUT:-



9. LeNet -5 CNN ARCHITECTURE USING FASHION-MNIST DATASET

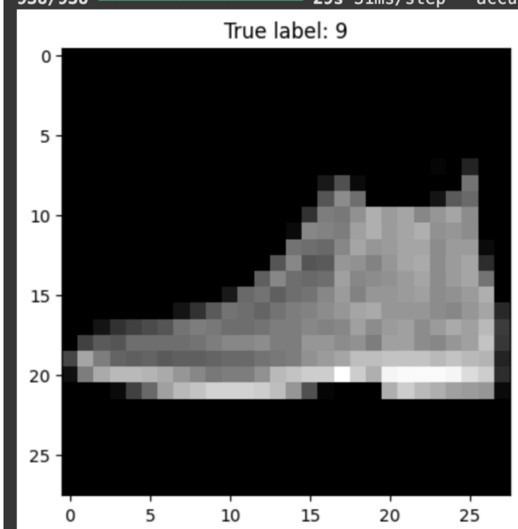
PROGRAM CODE:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

# Load and preprocess the Fashion-MNIST dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

# Build the LeNet-5 model
model = models.Sequential([
    layers.Conv2D(6, (5, 5), activation='tanh', input_shape=(28, 28, 1)),
    layers.AvgPool2D((2, 2)),
    layers.Conv2D(16, (5, 5), activation='tanh'),
    layers.AvgPool2D((2, 2)),
    layers.Flatten(),
    layers.Dense(120, activation='tanh'),
    layers.Dense(84, activation='tanh'),
    layers.Dense(10, activation='softmax')
])

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
# Display a sample image from the test set
plt.imshow(x_test[0].reshape(28, 28), cmap='gray')
plt.title(f"True label: {np.argmax(y_test[0])}")
plt.show()

# Make a prediction
prediction = model.predict(x_test[0].reshape(1, 28, 28, 1))
```

```
predicted_label = np.argmax(prediction)

# Display the predicted label
print(f"Predicted label: {predicted_label}")
```

OUTPUT:-

```
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 35s 34ms/step - accuracy: 0.7119 - loss: 0.7928 - val_accuracy: 0.8223 - val_loss: 0.4837
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 28s 30ms/step - accuracy: 0.8396 - loss: 0.4389 - val_accuracy: 0.8488 - val_loss: 0.4119
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 44s 33ms/step - accuracy: 0.8638 - loss: 0.3726 - val_accuracy: 0.8520 - val_loss: 0.3933
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 41s 33ms/step - accuracy: 0.8717 - loss: 0.3475 - val_accuracy: 0.8620 - val_loss: 0.3729
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 29s 31ms/step - accuracy: 0.8803 - loss: 0.3260 - val_accuracy: 0.8640 - val_loss: 0.3649
```



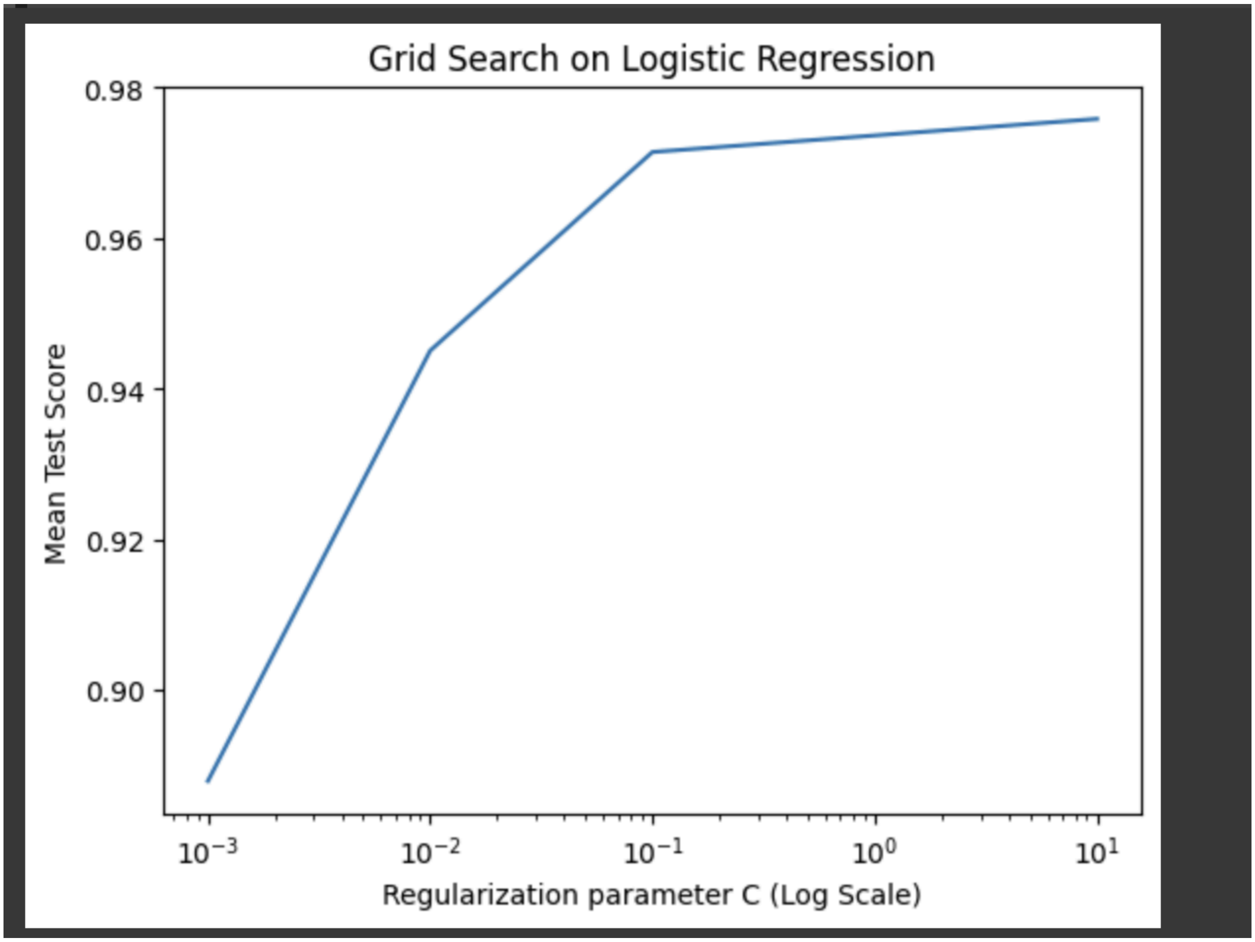

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step
Predicted label: 9
```

10. HYPERPARAMETER TUNING ON A BREAST CANCER DATASET

PROGRAM CODE:-

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split and scale the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Define the model and parameters for tuning
model = LogisticRegression(max_iter=10000)
params = {'C': [0.001, 0.01, 0.1, 1, 10]}  # Different regularization strengths

# Hyperparameter tuning using GridSearchCV
grid_search = GridSearchCV(model, params, cv=5)
grid_search.fit(X_train, y_train)

# Plot the results
plt.plot(params['C'], grid_search.cv_results_['mean_test_score'])
plt.xscale('log')
plt.xlabel('Regularization parameter C (Log Scale)')
plt.ylabel('Mean Test Score')
plt.title('Grid Search on Logistic Regression')
plt.show()
```

OUTPUT:


Grid Search on Logistic Regression
0.98
0.96
0.94
0.92
0.90
Mean Test Score
10^-3
10^-2
10^-1
10^0
10^1
Regularization parameter C (Log Scale)

11.MODEL DEPLOYMENT USING MNIST DIGITS DATASET

PROGRAM CODE:-

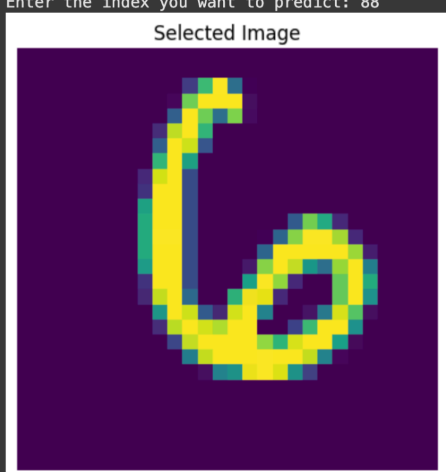
```
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load and preprocess MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build the model
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=1)

# Save the trained model
model.save("mnist_model.keras")
print("Model saved as 'mnist_model.keras'.")

import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.datasets import mnist

# Load data and model
```

```
(_, _), (X_test, _) = mnist.load_data()
X_test = X_test / 255.0  # Normalize
model = load_model("mnist_model.keras")

# Predict for a specific index
index = int(input("Enter the index you want to predict: "))
plt.imshow(X_test[index])
plt.title("Selected Image")
plt.axis("off")
plt.show()

prediction = np.argmax(model.predict(X_test[index].reshape(-1, 28, 28, 1)))
print(f"Predicted Digit: {prediction}")
```

OUTPUT:-

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 37s 19ms/step - accuracy: 0.9001 - loss: 0.3371
Model saved as 'mnist_model.keras'.
```

```
Enter the index you want to predict: 88
```




```
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
Predicted Digit: 6
```