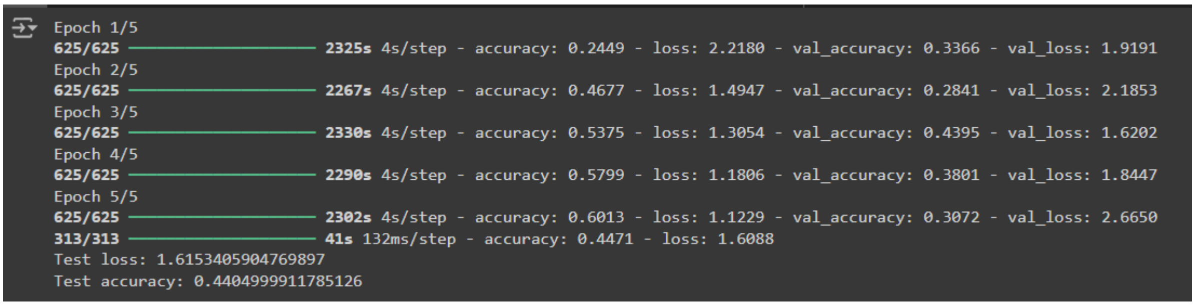
Practical – 6

Practical: Transfer learning with pre-trained models.

Tasks: Loading pre-trained CNN models (e.g., VGG, ResNet, Inception) using TensorFlow or Keras. Fine-tuning pre-trained models on a custom dataset for a specific computer vision task Evaluating the performance of fine-tuned models and comparing it with training from scratch.

Code:

```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import numpy as np

# Load and preprocess CIFAR-10 dataset
def load_cifar10_data():
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0
    y_train = to_categorical(y_train, 10)
    y_test = to_categorical(y_test, 10)
    return (x_train, y_train), (x_test, y_test)

# Load pre-trained model without the top layers
def get_base_model(input_shape):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
    return base_model

# Add custom layers on top of the base model
def build_model(base_model, num_classes):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model

# Fine-tuning the model
def fine_tune_model(model, base_model, learning_rate, num_layers_to_freeze):
    # Freeze the base layers
    for layer in base_model.layers[:num_layers_to_freeze]:
        layer.trainable = False
    # Compile the model
    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='categorical_crossentropy',
  metrics=['accuracy'])
    return model

# Main function
def main():
    input_size = (32, 32, 3)
    num_classes = 10  # CIFAR-10 has 10 classes
    learning_rate = 1e-4
    num_layers_to_freeze = 50
    epochs = 5
    batch_size = 64

    # Load data
    (x_train, y_train), (x_test, y_test) = load_cifar10_data()

    # Load and build model
    base_model = get_base_model(input_size)
    model = build_model(base_model, num_classes)

    # Fine-tune model
    model = fine_tune_model(model, base_model, learning_rate, num_layers_to_freeze)

    # Callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)

    # Train the model
    history = model.fit(
        x_train, y_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_split=0.2,
        callbacks=[early_stopping, checkpoint]
    )

    # Evaluate the model
    loss, accuracy = model.evaluate(x_test, y_test)
    print(f'Test loss: {loss}')
    print(f'Test accuracy: {accuracy}')

if __name__ == "__main__":
    main()
```



```
Epoch 1/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 2325s 4s/step - accuracy: 0.2449 - loss: 2.2180 - val_accuracy: 0.3366 - val_loss: 1.9191
Epoch 2/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 2267s 4s/step - accuracy: 0.4677 - loss: 1.4947 - val_accuracy: 0.2841 - val_loss: 2.1853
Epoch 3/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 2330s 4s/step - accuracy: 0.5375 - loss: 1.3054 - val_accuracy: 0.4395 - val_loss: 1.6202
Epoch 4/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 2290s 4s/step - accuracy: 0.5799 - loss: 1.1806 - val_accuracy: 0.3801 - val_loss: 1.8447
Epoch 5/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 2302s 4s/step - accuracy: 0.6013 - loss: 1.1229 - val_accuracy: 0.3072 - val_loss: 2.6650
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 132ms/step - accuracy: 0.4471 - loss: 1.6088
Test loss: 1.6153405904769897
Test accuracy: 0.4404999911785126
```