BEYOND CURRICULUM

Experiment 9

Objective: Implement an Inception V3 for image classification.

Explanation:

InceptionV3 is a deep convolutional neural network designed for image classification, leveraging an advanced architecture with multiple-sized convolutional filters in parallel to capture multi-scale features. It employs factorized convolutions, asymmetric convolutions, and label smoothing to enhance efficiency and accuracy while reducing computational costs. The model, pre-trained on ImageNet, can be fine-tuned for custom datasets. In TensorFlow/Keras, InceptionV3 is implemented as a feature extractor or fine-tuned with additional layers. Its optimized design improves feature representation, making it effective for large-scale image classification tasks with high precision while maintaining computational efficiency.

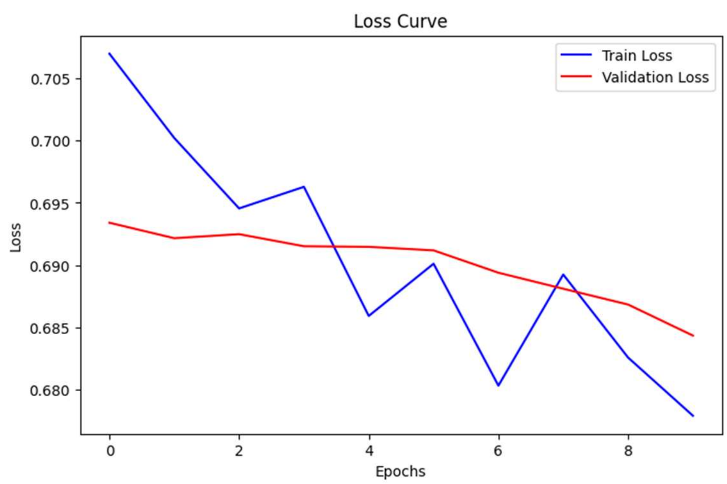
Code & Output:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Enable mixed precision training (if supported)
tf.keras.mixed_precision.set_global_policy('mixed_float16')
```

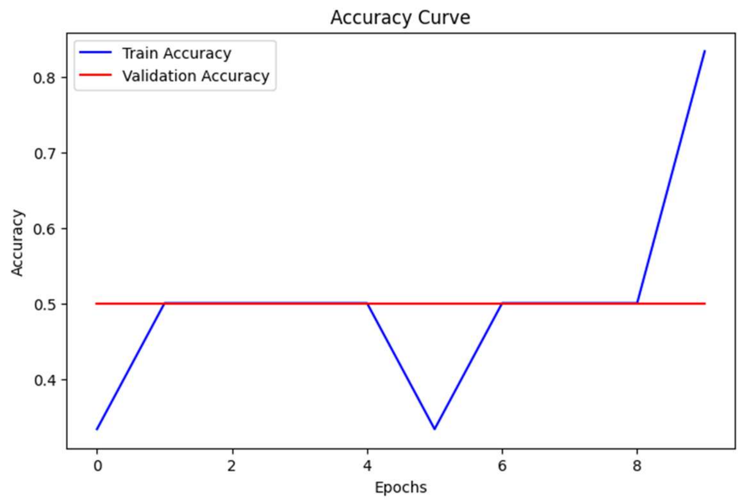
```python
# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Convert grayscale to RGB by expanding dimensions
x_train = np.repeat(x_train[..., np.newaxis], 3, -1)
x_test = np.repeat(x_test[..., np.newaxis], 3, -1)

# Normalize pixel values
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('float32') / 255.0

# One-hot encode labels
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

# Create TensorFlow dataset with optimized pipeline
def preprocess(image, label):
    image = tf.image.resize(image, (150, 150))  # Adjusted for InceptionV3 input size
    return image, label

train_dataset = (tf.data.Dataset.from_tensor_slices((x_train, y_train))
                 .map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
                 .cache()
                 .batch(32)
                 .prefetch(tf.data.AUTOTUNE))

test_dataset = (tf.data.Dataset.from_tensor_slices((x_test, y_test))
                .map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
```

```
            .cache()
            .batch(32)
            .prefetch(tf.data.AUTOTUNE))

# Load InceptionV3 model without top layer
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
for layer in base_model.layers:
    layer.trainable = False  # Freeze convolutional layers

# Add custom layers on top
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dense(10, activation='softmax', dtype='float32')(x)  # Ensure correct dtype with mixed precision

model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = model.fit(train_dataset, epochs=5, validation_data=test_dataset)  # Reduced epochs to 5

# Plot training and validation loss
```

```
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Train Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='red')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss Curve')
plt.show()
```




```
# Plot training and validation accuracy
plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Train Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Curve')
plt.show()
```



Learning Outcome(s):

- Successfully implemented an Inception V3 for image classification.
- Visualizing results using Matplotlib.

BEYOND CURRICULUM

Experiment 10

Objective: Implement Bi-directional LSTM for text classification.

Explanation:

A Bi-directional Long Short-Term Memory (Bi-LSTM) network enhances text classification by capturing both past and future dependencies in a sequence. Unlike standard LSTMs, which process text in one direction, Bi-LSTM consists of two LSTMs running in opposite directions, improving context understanding. This model is effective for sentiment analysis, spam detection, and document classification. Implemented in TensorFlow/Keras, it typically includes an embedding layer, Bi-LSTM layers, and dense layers for classification. Bi-LSTM improves accuracy by leveraging complete context, making it superior for processing long-range dependencies in natural language processing tasks.

Code & Output:

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

import matplotlib.pyplot as plt


# Sample dataset

texts = ["I love this product!", "This is the worst experience ever.", "Absolutely fantastic!", "Not good at all.", "Great service and fast delivery.", "Terrible quality, do not buy!", "Excellent and reliable.", "Would not recommend."]

labels = [1, 0, 1, 0, 1, 0, 1, 0]  # 1: Positive, 0: Negative
```

```
# Tokenization and padding
max_words = 10000
max_len = 20
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post')

# Convert labels to numpy array
labels = np.array(labels)

# Split dataset into training and validation sets
split = int(0.8 * len(texts))
x_train, x_val = padded_sequences[:split], padded_sequences[split:]
y_train, y_val = labels[:split], labels[split:]

# Define Bi-LSTM model
model = Sequential([
    Embedding(input_dim=max_words, output_dim=64, input_length=max_len),
    Bidirectional(LSTM(64, return_sequences=True)),
    Bidirectional(LSTM(32)),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

Compile model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

Train model

history = model.fit(x_train, y_train, epochs=50, validation_data=(x_val, y_val), batch_size=2)

Plot training and validation loss

plt.figure(figsize=(8, 5))

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

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

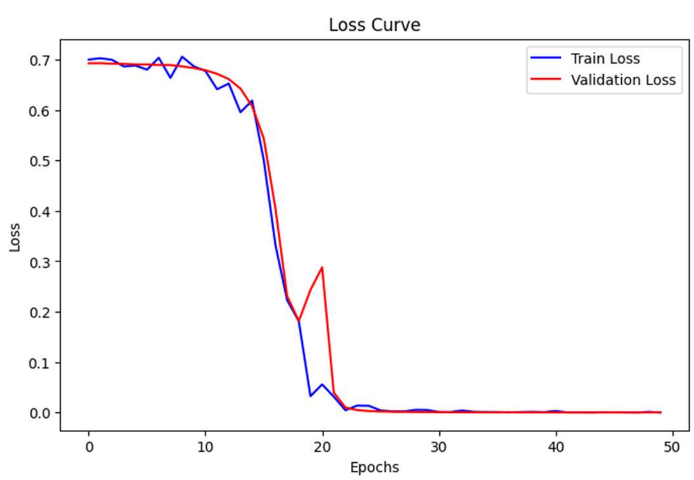
plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.title('Loss Curve')

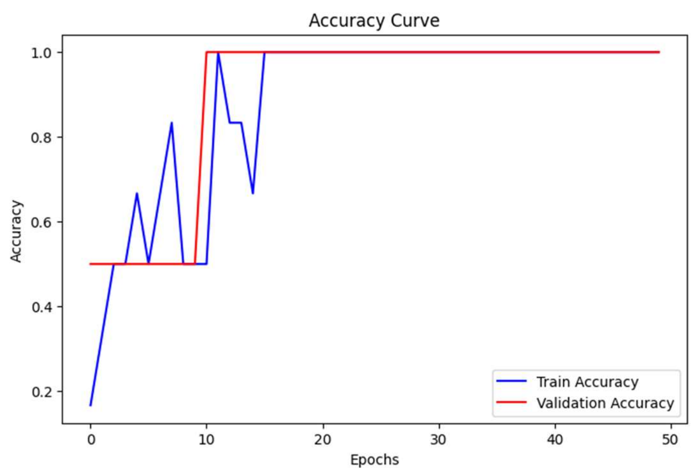
plt.show()



Plot training and validation accuracy

plt.figure(figsize=(8, 5))

```
plt.plot(history.history['accuracy'], label='Train Accuracy', color='blue')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Curve')
plt.show()
```



Learning Outcome(s):

- Successfully implemented Bi-directional LSTM for text classification.
- Visualizing results using Matplotlib.