Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Image Processing and Computer Vision - II Laboratory (DJ19DSL702)

AY: 2024-25

Experiment 5

Bhuvi Ghosh
60009210191

(Transfer Learning on Image Classification)

Aim: To compare the performance of different transfer learning strategies on an image classification task.

Theory:

Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a second, related task. Instead of training a model from scratch, transfer learning allows leveraging the knowledge a model has gained from a large, diverse dataset (like ImageNet) to apply it to a smaller, more specific dataset. This is especially useful when the new dataset is limited in size or the task is closely related to the original problem. By reusing the learned features from a pre-trained model, transfer learning reduces computational resources, training time, and often improves the performance of the new task. It is widely used in areas like image classification, natural language processing, and speech recognition.

Datasets:

1. Pre-trained model: Use a pre-trained model like VGG16, ResNet50, or MobileNet, trained on ImageNet.
2. Custom dataset: A smaller image dataset specific to the problem (e.g., classifying images of specific objects, animals, or scenes).

Approach:

The experiment will test three different transfer learning strategies:

3. Fine-Tuning the Entire Model: The pre-trained model is fully retrained on the new dataset.

4. Freezing Some Layers: Some layers of the pre-trained model are frozen (not updated), while others are trained on the new dataset.
5. Using the Model as a Feature Extractor: The pre-trained model is used to extract features, and only a new classifier is trained on top.

Experiment Design:

Step 1: Preprocessing the Data

- Preprocess the custom dataset (resize, normalize, augment).
- Split the dataset into training, validation, and test sets.

Step 2: Transfer Learning Methods

Method 1: Fine-Tuning the Entire Model

- Load the pre-trained model (e.g., VGG16 or ResNet50).
- Replace the final classification layer to match the number of classes in your dataset.
- Train all layers of the model using the new dataset.

Method 2: Freezing Some Layers

- Load the pre-trained model.
- Freeze the initial layers (e.g., first 10 layers in a 50-layer network).
- Replace the final classification layer.
- Train only the non-frozen layers on the new dataset.

Method 3: Using the Model as a Feature Extractor

- Load the pre-trained model.
- Freeze all the layers except the final classification layer.
- Replace the final classification layer.
- Train only the classifier on the new dataset.

Step 3: Training

- Train each of the three models using appropriate loss functions (e.g., cross-entropy for classification).
- Use a learning rate schedule or optimizer like Adam or SGD.

Step 4: Evaluation

- Evaluate the models on the test set using metrics like accuracy, precision, recall, and F1-score.
- Record the training time and performance for each method.

Step 5: Analysis

- Compare the performance of the three methods in terms of: Model accuracy on the test set.

Expected Outcomes:

- **Fine-tuning the entire model** might yield the best performance but could be computationally expensive.
- **Freezing some layers** allows faster training while retaining some learning from the pre-trained model.
- **Feature extraction** might be the quickest but may offer slightly lower performance depending on the dataset.

Code:

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


def load_and_preprocess_data():

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

    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0


    x_train = x_train.reshape((-1, 28, 28, 1))
    x_test = x_test.reshape((-1, 28, 28, 1))


    y_train = to_categorical(y_train, num_classes=10)
    y_test = to_categorical(y_test, num_classes=10)


    val_size = 10000
    x_val, y_val = x_train[-val_size:], y_train[-val_size:]
    x_train, y_train = x_train[:-val_size], y_train[:-val_size]

    return (x_train, y_train), (x_val, y_val), (x_test, y_test)


(x_train, y_train), (x_val, y_val), (x_test, y_test) = load_and_preprocess_data()

def create_model_fine_tuning(num_classes):
    base_model = models.Sequential([
        layers.Input(shape=(28, 28, 1)),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])

    return base_model

def create_model_freezing_layers(num_classes):
    base_model = create_model_fine_tuning(num_classes)


    for layer in base_model.layers[:4]:
        layer.trainable = False

    return base_model
```

```python
def create_model_feature_extractor(num_classes):
    base_model = create_model_fine_tuning(num_classes)


    for layer in base_model.layers:
        layer.trainable = False

    return base_model


def train_model(model, x_train, y_train, x_val, y_val, epochs=10, batch_size=64):
    model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(), metrics=['accuracy'])
    history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_val, y_val))
    return history


def evaluate_model(model, x_test, y_test):
    test_loss, test_acc = model.evaluate(x_test, y_test)
    return test_loss, test_acc


num_classes = 10


print("Fine-Tuning the Entire Model:")
model_fine_tuning = create_model_fine_tuning(num_classes)
history_fine_tuning = train_model(model_fine_tuning, x_train, y_train, x_val, y_val)
test_loss_fine_tuning, test_acc_fine_tuning = evaluate_model(model_fine_tuning, x_test, y_test)

print("\nFreezing Some Layers:")
model_freezing = create_model_freezing_layers(num_classes)
history_freezing = train_model(model_freezing, x_train, y_train, x_val, y_val)
test_loss_freezing, test_acc_freezing = evaluate_model(model_freezing, x_test, y_test)


print("\nUsing the Model as a Feature Extractor:")
model_feature_extractor = create_model_feature_extractor(num_classes)
history_feature_extractor = train_model(model_feature_extractor, x_train, y_train, x_val, y_val)
test_loss_feature_extractor, test_acc_feature_extractor = evaluate_model(model_feature_extractor, x_test, y_test)


print("\nTest Accuracy:")
print(f"Fine-Tuning: {test_acc_fine_tuning:.4f}")
print(f"Freezing Some Layers: {test_acc_freezing:.4f}")
print(f"Feature Extraction: {test_acc_feature_extractor:.4f}")


def plot_history(history, title):
    plt.plot(history.history['accuracy'], label='train accuracy')
    plt.plot(history.history['val_accuracy'], label='val accuracy')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

```
[2]  def create_model_feature_extractor(num_classes):
         base_model = models.Sequential([
             layers.Input(shape=(28, 28, 1)),
             layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
             layers.MaxPooling2D(),
             layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
             layers.MaxPooling2D(),
             layers.Flatten(),
         ])

         for layer in base_model.layers:
             layer.trainable = False

         model = models.Sequential()
         model.add(base_model)
         model.add(layers.Dense(128, activation='relu'))
         model.add(layers.Dense(num_classes, activation='softmax'))

         return model
```

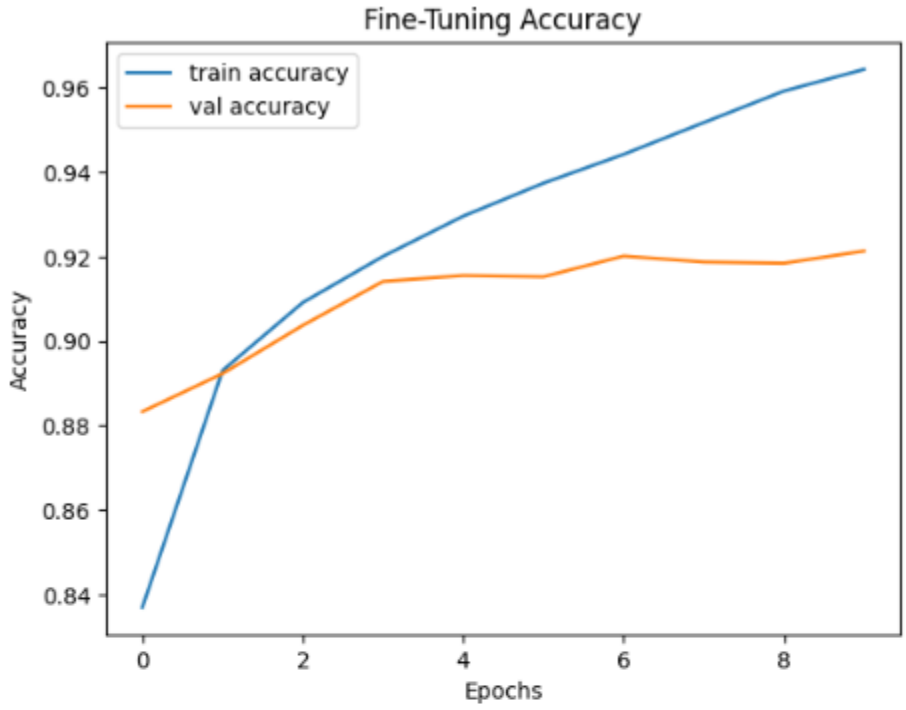
```
print("\nUsing the Model as a Feature Extractor:")
model_feature_extractor = create_model_feature_extractor(num_classes)
history_feature_extractor = train_model(model_feature_extractor, x_train, y_train, x_val, y_val, epochs=20)  # Adj
test_loss_feature_extractor, test_acc_feature_extractor = evaluate_model(model_feature_extractor, x_test, y_test)


print("\nTest Accuracy for Feature Extraction:")
print(f"Feature Extraction: {test_acc_feature_extractor:.4f}")
```

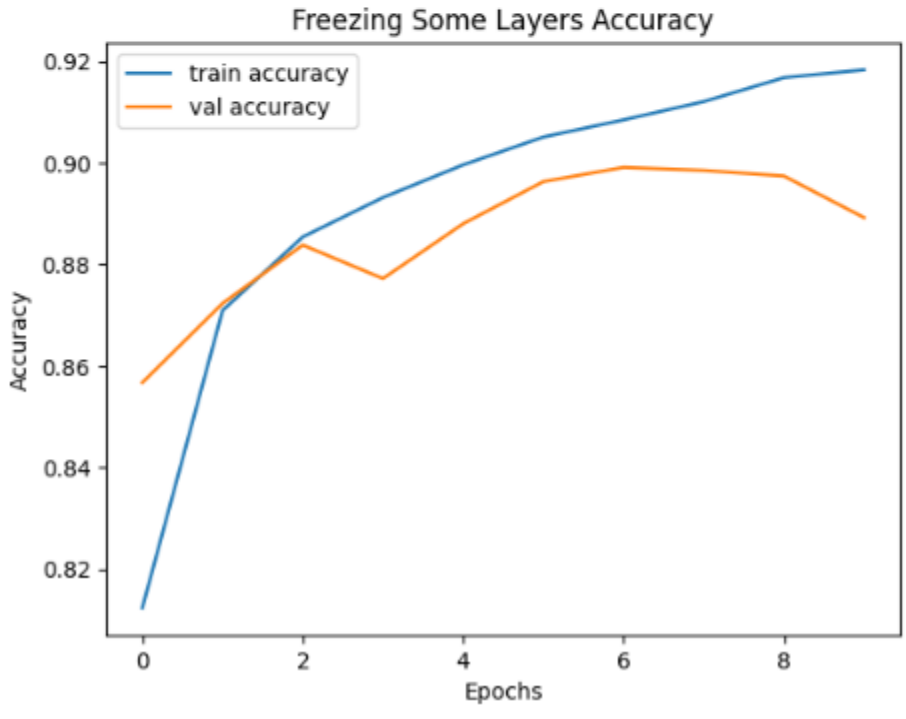
Output:

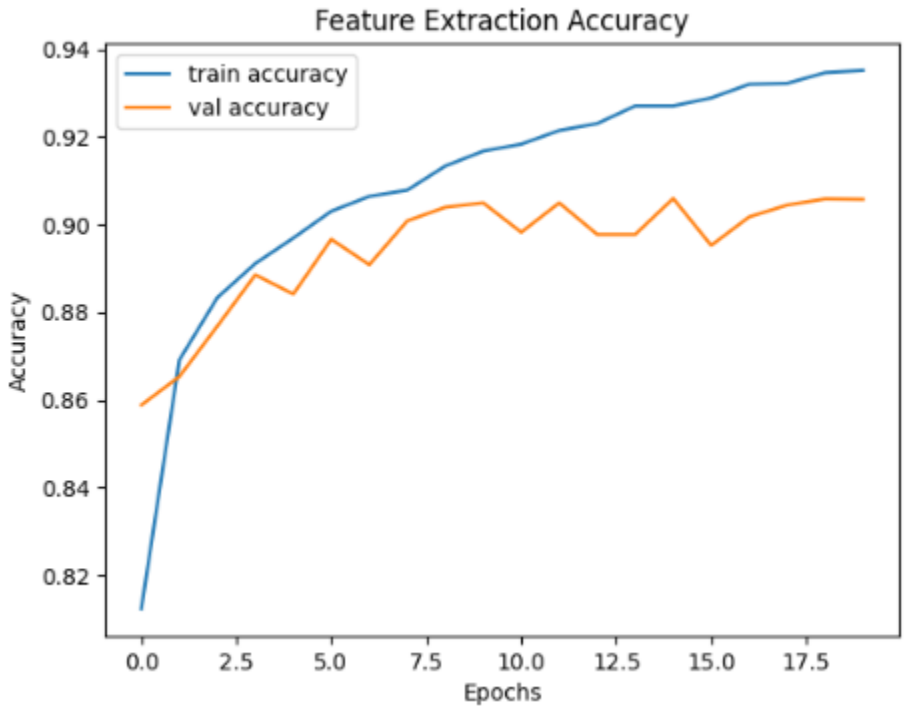
```
Fine-Tuning the Entire Model:
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 73s 90ms/step - accuracy: 0.7709 - loss: 0.6494 - val_accuracy: 0.8833 - val_loss: 0.3269
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 81s 88ms/step - accuracy: 0.8898 - loss: 0.3060 - val_accuracy: 0.8923 - val_loss: 0.2957
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 89ms/step - accuracy: 0.9086 - loss: 0.2481 - val_accuracy: 0.9037 - val_loss: 0.2689
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 68s 87ms/step - accuracy: 0.9194 - loss: 0.2233 - val_accuracy: 0.9141 - val_loss: 0.2413
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 71s 91ms/step - accuracy: 0.9303 - loss: 0.1900 - val_accuracy: 0.9155 - val_loss: 0.2320
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 69s 88ms/step - accuracy: 0.9378 - loss: 0.1708 - val_accuracy: 0.9152 - val_loss: 0.2363
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 88ms/step - accuracy: 0.9468 - loss: 0.1440 - val_accuracy: 0.9201 - val_loss: 0.2261
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 88ms/step - accuracy: 0.9517 - loss: 0.1298 - val_accuracy: 0.9187 - val_loss: 0.2330
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 69s 88ms/step - accuracy: 0.9601 - loss: 0.1110 - val_accuracy: 0.9184 - val_loss: 0.2413
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 82s 88ms/step - accuracy: 0.9656 - loss: 0.0921 - val_accuracy: 0.9213 - val_loss: 0.2493
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.9166 - loss: 0.2815

Freezing Some Layers:
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 29s 36ms/step - accuracy: 0.7451 - loss: 0.7992 - val_accuracy: 0.8567 - val_loss: 0.4006
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 27s 35ms/step - accuracy: 0.8691 - loss: 0.3755 - val_accuracy: 0.8723 - val_loss: 0.3550
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 27s 35ms/step - accuracy: 0.8824 - loss: 0.3331 - val_accuracy: 0.8838 - val_loss: 0.3254
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 35ms/step - accuracy: 0.8935 - loss: 0.2981 - val_accuracy: 0.8772 - val_loss: 0.3338
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.8984 - loss: 0.2814 - val_accuracy: 0.8880 - val_loss: 0.3115
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 27s 35ms/step - accuracy: 0.9051 - loss: 0.2638 - val_accuracy: 0.8963 - val_loss: 0.2897
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 32s 40ms/step - accuracy: 0.9091 - loss: 0.2508 - val_accuracy: 0.8991 - val_loss: 0.2812
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9129 - loss: 0.2420 - val_accuracy: 0.8985 - val_loss: 0.2807
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 35ms/step - accuracy: 0.9161 - loss: 0.2361 - val_accuracy: 0.8974 - val_loss: 0.2834
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.9191 - loss: 0.2265 - val_accuracy: 0.8892 - val_loss: 0.3064
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8816 - loss: 0.3234
```

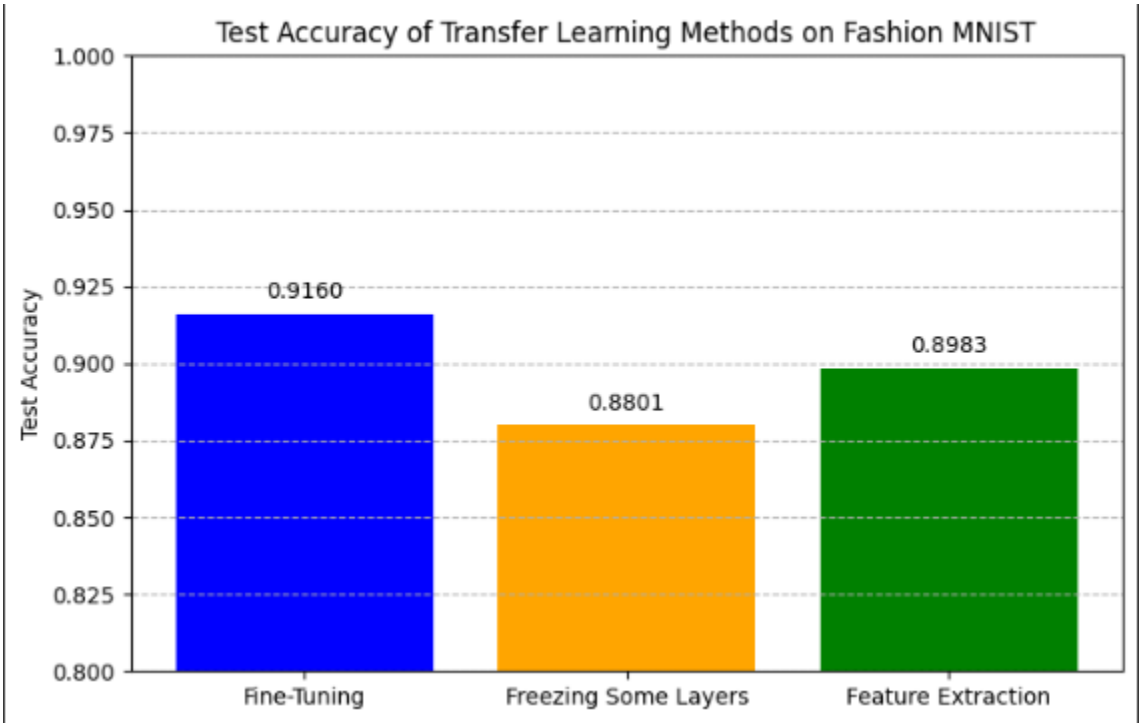
```
Using the Model as a Feature Extractor:
Epoch 1/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 29s 36ms/step - accuracy: 0.7409 - loss: 0.7819 - val_accuracy: 0.8588 - val_loss: 0.4011
Epoch 2/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.8663 - loss: 0.3800 - val_accuracy: 0.8653 - val_loss: 0.3733
Epoch 3/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 43s 39ms/step - accuracy: 0.8794 - loss: 0.3417 - val_accuracy: 0.8768 - val_loss: 0.3468
Epoch 4/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 39s 36ms/step - accuracy: 0.8910 - loss: 0.3127 - val_accuracy: 0.8885 - val_loss: 0.3117
Epoch 5/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 42s 36ms/step - accuracy: 0.8957 - loss: 0.2908 - val_accuracy: 0.8841 - val_loss: 0.3269
Epoch 6/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - accuracy: 0.9045 - loss: 0.2666 - val_accuracy: 0.8966 - val_loss: 0.2875
Epoch 7/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 35ms/step - accuracy: 0.9056 - loss: 0.2609 - val_accuracy: 0.8908 - val_loss: 0.3014
Epoch 8/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - accuracy: 0.9076 - loss: 0.2521 - val_accuracy: 0.9008 - val_loss: 0.2746
Epoch 9/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - accuracy: 0.9130 - loss: 0.2384 - val_accuracy: 0.9039 - val_loss: 0.2683
Epoch 10/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9143 - loss: 0.2370 - val_accuracy: 0.9049 - val_loss: 0.2617
Epoch 11/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - accuracy: 0.9192 - loss: 0.2238 - val_accuracy: 0.8982 - val_loss: 0.2779
Epoch 12/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9193 - loss: 0.2237 - val_accuracy: 0.9049 - val_loss: 0.2596
Epoch 13/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9236 - loss: 0.2084 - val_accuracy: 0.8977 - val_loss: 0.2907
Epoch 14/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9255 - loss: 0.2073 - val_accuracy: 0.8977 - val_loss: 0.2818
Epoch 15/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - accuracy: 0.9267 - loss: 0.2018 - val_accuracy: 0.9059 - val_loss: 0.2531
Epoch 16/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 27s 35ms/step - accuracy: 0.9299 - loss: 0.1910 - val_accuracy: 0.8952 - val_loss: 0.2928
Epoch 17/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 35ms/step - accuracy: 0.9302 - loss: 0.1884 - val_accuracy: 0.9017 - val_loss: 0.2638
Epoch 18/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.9311 - loss: 0.1846 - val_accuracy: 0.9044 - val_loss: 0.2748
Epoch 19/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 36ms/step - accuracy: 0.9337 - loss: 0.1789 - val_accuracy: 0.9058 - val_loss: 0.2626
Epoch 20/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - accuracy: 0.9356 - loss: 0.1722 - val_accuracy: 0.9057 - val_loss: 0.2606
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8975 - loss: 0.2844

Test Accuracy for Feature Extraction:
Feature Extraction: 0.8983
```

Graph for comparision:






Feature Extraction Accuracy
train accuracy
val accuracy
Accuracy
Epochs
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Test Accuracy of Transfer Learning Methods on Fashion MNIST
Test Accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.9160
0.8801
0.8983
Fine-Tuning
Freezing Some Layers
Feature Extraction