```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
from time import time
import gc
```

```
from tensorflow.keras.applications.vgg19 import preprocess_input

# Load and preprocess reduced dataset
print("Loading and preprocessing data...")

n_samples = 10000

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Reduce dataset size
x_train = x_train[:n_samples]
y_train = y_train[:n_samples]

# Using 1/4 of n_samples for test set
x_test = x_test[:n_samples//4]
y_test = y_test[:n_samples//4]

print(f"Training samples: {x_train.shape[0]}")
print(f"Testing samples: {x_test.shape[0]}")
```

```
Loading and preprocessing data...
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Training samples: 10000
Testing samples: 2500
```

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

# Convert labels to one-hot encoding
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_test = lb.transform(y_test)
```

```
def create_feature_extractors():
    ## Create feature extractors from different VGG19 layers"""
    base_model = VGG19(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

    layers_to_extract = [
        'block3_conv2',  # Early layer
        'block4_conv3',  # Middle layer
        'block5_conv4'   # Deep layer
    ]

    feature_extractors = {
        layer: Model(inputs=base_model.input, outputs=base_model.get_layer(layer).output)
        for layer in layers_to_extract
    }

    return feature_extractors, layers_to_extract
```

```
#Extract the features in batches to manage the memory

def extract_features_in_batches(model, data, batch_size=32):

    num_samples = data.shape[0]
    features_list = []

    # Process data in batches
    for i in range(0, num_samples, batch_size):
        batch_data = data[i:min(i + batch_size, num_samples)]

        # Extract features for the batch
        batch_features = model.predict(batch_data, verbose=0)

        # Reshape features to 2D array
        batch_features_reshaped = batch_features.reshape(batch_features.shape[0], -1)
        features_list.append(batch_features_reshaped)

        # Clear memory
        del batch_data, batch_features
        gc.collect()

    # Combine all batches
    return np.concatenate(features_list, axis=0)
```

```
def evaluate_model(clf, X_train, X_test, y_train, y_test):

    # Train the model
    start_time = time()
    clf.fit(X_train, y_train)
    train_time = time() - start_time

    # Make predictions
    start_time = time()
    y_pred = clf.predict(X_test)
    predict_time = time() - start_time

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)

    return {
        'accuracy': accuracy,
        'precision': report['weighted avg']['precision'],
        'recall': report['weighted avg']['recall'],
        'f1': report['weighted avg']['f1-score'],
        'train_time': train_time,
        'predict_time': predict_time
    }
```

```
# Create feature extractors
print("Creating feature extractors...")
feature_extractors, layer_names = create_feature_extractors()

# Extract features for each layer
print("Extracting features for each layer : ")
features_train = {}
features_test = {}
```

```
for layer in layer_names:
    print(f"\nProcessing layer: {layer}")

    # Extract features for training data
    print("Extracting training features : ")

    features_train[layer] = extract_features_in_batches(
        feature_extractors[layer],
        x_train,
        batch_size=32
    )

    # Extract features for test data
    print("Extracting test features...")
    features_test[layer] = extract_features_in_batches(
        feature_extractors[layer],
        x_test,
        batch_size=32
    )

    # Print feature shapes
    print(f"Features from {layer}:")
    print(f"Train shape: {features_train[layer].shape}")
    print(f"Test shape: {features_test[layer].shape}")
```

```
Creating feature extractors...
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Extracting features for each layer :

Processing layer: block3_conv2
Extracting training features :
Extracting test features...
Features from block3_conv2:
Train shape: (10000, 16384)
Test shape: (2500, 16384)

Processing layer: block4_conv3
Extracting training features :
Extracting test features...
Features from block4_conv3:
Train shape: (10000, 8192)
Test shape: (2500, 8192)

Processing layer: block5_conv4
```

```
Extracting training features :
Extracting test features...
Features from block5_conv4:
Train shape: (10000, 2048)
Test shape: (2500, 2048)
```

```
# Define classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'KNN': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Decision Tree': DecisionTreeClassifier()
}
```

```
# Store results
results = []

# For each layer and classifier combination
for layer in layer_names:
    print(f"\nEvaluating models for layer: {layer}")

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(features_train[layer])
    X_test_scaled = scaler.transform(features_test[layer])

    # Evaluate each classifier
    for clf_name, clf in classifiers.items():
        print(f"Evaluating {clf_name}...")

        metrics = evaluate_model(
            clf,
            X_train_scaled,
            X_test_scaled,
            y_train.argmax(axis=1),
            y_test.argmax(axis=1)
        )

        results.append({
            'Layer': layer,
            'Classifier': clf_name,
            **metrics
        })
```

```
Evaluating models for layer: block3_conv2
Evaluating Logistic Regression...
Evaluating KNN...
Evaluating Random Forest...
Evaluating Decision Tree...

Evaluating models for layer: block4_conv3
Evaluating Logistic Regression...
Evaluating KNN...
Evaluating Random Forest...
Evaluating Decision Tree...

Evaluating models for layer: block5_conv4
Evaluating Logistic Regression...
Evaluating KNN...
Evaluating Random Forest...
Evaluating Decision Tree...
```

```
# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Find best combination
best_idx = results_df['accuracy'].idxmax()
best_combination = results_df.iloc[best_idx]

# Print results
print("\nResults Summary:")
print(results_df.round(4))
print("\nBest Combination:")
print(f"Layer: {best_combination['Layer']}")
print(f"Classifier: {best_combination['Classifier']}")
print(f"Accuracy: {best_combination['accuracy']:.4f}")
print(f"F1 Score: {best_combination['f1']:.4f}")
print(f"Precision: {best_combination['precision']:.4f}")
print(f"Recall: {best_combination['recall']:.4f}")
```

```
Results Summary:
           Layer           Classifier  accuracy  precision  recall      f1  \
0   block3_conv2  Logistic Regression    0.7116     0.7084  0.7116  0.7095
1   block3_conv2                  KNN    0.5236     0.6108  0.5236  0.5214
2   block3_conv2        Random Forest    0.5944     0.5895  0.5944  0.5901
3   block3_conv2        Decision Tree    0.3272     0.3299  0.3272  0.3282
```

```
4   block4_conv3  Logistic Regression    0.7092     0.7080  0.7092  0.7081
5   block4_conv3                  KNN    0.5488     0.5743  0.5488  0.5435
6   block4_conv3        Random Forest    0.6104     0.6081  0.6104  0.6076
7   block4_conv3        Decision Tree    0.3620     0.3601  0.3620  0.3608
8   block5_conv4  Logistic Regression    0.4784     0.4771  0.4784  0.4775
9   block5_conv4                  KNN    0.4156     0.4264  0.4156  0.4162
10  block5_conv4        Random Forest    0.4860     0.4861  0.4860  0.4835
11  block5_conv4        Decision Tree    0.2980     0.2998  0.2980  0.2977

    train_time  predict_time
0      63.4323        0.3553
1       0.1258       36.7296
2     144.8580        0.2294
3     277.0245        0.0285
4      40.6701        0.1460
5       0.0566       19.7361
6      77.1943        0.1810
7     105.2927        0.0200
8      79.5933        0.0710
9       0.0243        5.8503
10     21.2779        0.1768
11     13.6479        0.0045

Best Combination:
Layer: block3_conv2
Classifier: Logistic Regression
Accuracy: 0.7116
F1 Score: 0.7095
Precision: 0.7084
Recall: 0.7116
```