Experiment 2.1

Aim: Write a program to compare the performance of different classification models in image recognition.

Software Required: Any Python IDE (e.g., PyCharm, Jupyter Notebook, GoogleColab)

Description: There are several classification models commonly used in image recognition tasks. Some popular ones are:

1. Convolutional Neural Networks (CNN): CNNs have revolutionized image recognition and achieved state-of-the-art performance in various tasks. They consist of multiple convolutional layers that automatically learn hierarchical features from images. Popular CNN architectures include AlexNet, VGGNet, GoogLeNet (Inception), ResNet, and DenseNet.
2. Support Vector Machines (SVM): SVMs are supervised learning models that can be used for image classification. They find an optimal hyperplane to separate different classes in feature space. SVMs are often combined with handcrafted features or extracted features from CNNs.
3. Random Forests: Random Forests are ensemble learning models that consist of multiple decision trees. They can be used for image classification by combining features extracted from images and making predictions based on the majority voting of the trees.
4. Gradient Boosting Models: Gradient boosting models, such as XGBoost and LightGBM, are also used in image classification tasks. These models build an ensemble of weak learners in a sequential manner and optimize a loss function to minimize prediction errors. They can handle complex relationships between features and provide high accuracy.
5. Deep Belief Networks (DBN): DBNs are deep learning models that have multiple layers of restricted Boltzmann machines (RBMs). They can learn hierarchical representations of images and perform classification tasks. However, CNNs have largely replaced DBNs in image recognition due to their superior performance.
6. Transfer Learning Models: Transfer learning allows pre-trained models, typically CNNs trained on large-scale datasets like ImageNet, to be utilized for image recognition tasks. By fine-tuning the pre-trained models on a smaller dataset specific to the target task, transfer learning enables effective classification even with limited data.
7. Ensemble Models: Ensemble models combine multiple classifiers to improve classification performance. Techniques like bagging, boosting, and stacking can be applied to combine the predictions of multiple models, such as CNNs, SVMs, or random forests, to obtain better accuracy and robustness.
8. Deep Convolutional Generative Adversarial Networks (DCGAN): While primarily used for image generation, DCGANs can also be utilized for image classification. By training a DCGAN on a specific dataset, the discriminator

network can be used as a classifier to distinguish between different classes of images.

Steps:

- Import the necessary libraries and modules.
- Load a dataset of labeled images for training and testing.
- Preprocess the images by resizing, normalizing, and augmenting if necessary.
- Split the dataset into training and testing sets.
- Define and initialize different classification models, such as support vector machines (SVM), random forest, convolutional neural networks (CNN), etc.
- Train each model using the training set.
- Evaluate the performance of each model using various metrics, such as accuracy, precision, recall, and F1 score, on the testing set.
- Compare the performance of the different models based on the evaluation results.
- Analyze the strengths and weaknesses of each model in terms of accuracy, computational efficiency, robustness, etc.
- Draw conclusions and discuss the implications of the findings.

Code:

```
# Step 1: Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


# Step 2: Load dataset
digits = load_digits()
X = digits.images.reshape((len(digits.images), -1))
y = digits.target

# Step 3: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Initialize models
models = [
    ('Logistic Regression', LogisticRegression(max_iter=1000)),
    ('SVM', SVC()),
    ('Random Forest', RandomForestClassifier())
]

# Step 5: Train and evaluate models
```

```
results = []
for name, model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results.append((name, accuracy))

# Step 6: Print results
for name, accuracy in results:
    print(f'{name}: {accuracy:.4f}')


# Step 7: Visualize results
names, accuracies = zip(*results)
plt.figure(figsize=(10, 5))
plt.bar(names, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Classification Model Performance')
plt.show()
```

Implementation:

```
[1] # Step 1: Import necessary libraries
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.datasets import load_digits
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn.svm import SVC
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score
```

```
    # Step 2: Load dataset
    digits = load_digits()
    X = digits.images.reshape((len(digits.images), -1))
    y = digits.target
```

```
[3] # Step 3: Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[4] # Step 4: Initialize models
    models = [
        ('Logistic Regression', LogisticRegression(max_iter=1000)),
        ('SVM', SVC()),
        ('Random Forest', RandomForestClassifier())
    ]
```

```
[5] # Step 5: Train and evaluate models
    results = []
    for name, model in models:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        results.append((name, accuracy))
```

```
    /usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
    STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

    Increase the number of iterations (max_iter) or scale the data as shown in:
        https://scikit-learn.org/stable/modules/preprocessing.html
    Please also refer to the documentation for alternative solver options:
        https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
      n_iter_i = _check_optimize_result(
```

```
[6] # Step 6: Print results
    for name, accuracy in results:
```

```
[6] # Step 6: Print results
    for name, accuracy in results:
        print(f'{name}: {accuracy:.4f}')
```

```
Logistic Regression: 0.9722
SVM: 0.9861
Random Forest: 0.9750
```

```
# Step 7: Visualize results
names, accuracies = zip(*results)
plt.figure(figsize=(10, 5))
plt.bar(names, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Classification Model Performance')
plt.show()
```