Experiment- 5

Student Name: Kush Bhasin **UID:** 20BCS7677
Branch: BE(CSE) **Section/Group:** 20BCS-1/B
Semester: 7th
Subject Name: Computer Vision Lab
Subject Code: 20CSP- 422

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

2. **System Requirements:**
   - Python 3.9
   - Visual Studio Code

3. **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.

4. Steps:

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

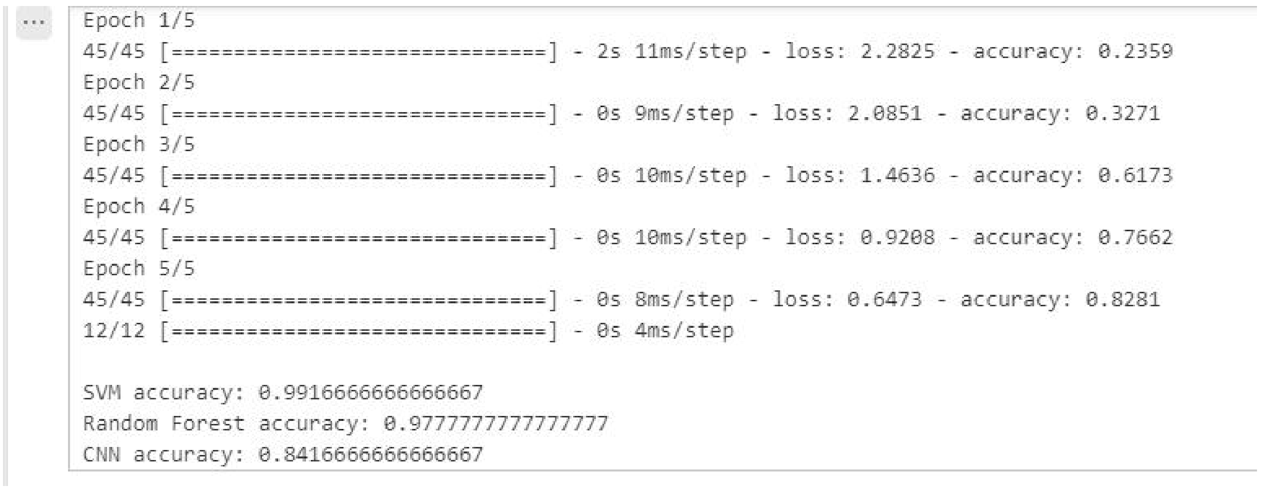
5. Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from keras.utils import to_categorical
from sklearn import datasets
from sklearn.metrics import accuracy_score, classification_report

def load_dataset():
    digits = datasets.load_digits()
    X, y = digits.data, digits.target
    return X, y
def preprocess_dataset(X, y):
    X = X.reshape(-1, 8, 8, 1)
    X = X / 255.0
    y_one_hot = to_categorical(y, num_classes=10)
    return X, y_one_hot
# Load and preprocess the dataset
X, y = load_dataset()
X, y = preprocess_dataset(X, y)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Define and initialize classification models
svm_model = SVC()
rf_model = RandomForestClassifier()
# CNN model using Keras
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(8, 8, 1), padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
# Compile the CNN model
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Train the models
svm_model.fit(X_train.reshape(X_train.shape[0], -1), np.argmax(y_train, axis=1))
rf_model.fit(X_train.reshape(X_train.shape[0], -1), np.argmax(y_train, axis=1))
cnn_model.fit(X_train, y_train, epochs=5, batch_size=32)
# Evaluate the models
svm_preds = svm_model.predict(X_test.reshape(X_test.shape[0], -1))
rf_preds = rf_model.predict(X_test.reshape(X_test.shape[0], -1))
cnn_preds = np.argmax(cnn_model.predict(X_test), axis=1)
# Accuracy for each model
svm_accuracy = accuracy_score(np.argmax(y_test, axis=1), svm_preds)
rf_accuracy = accuracy_score(np.argmax(y_test, axis=1), rf_preds)
cnn_accuracy = accuracy_score(np.argmax(y_test, axis=1), cnn_preds)
# Classification reports for each model
svm_report = classification_report(np.argmax(y_test, axis=1), svm_preds)
rf_report = classification_report(np.argmax(y_test, axis=1), rf_preds)
cnn_report = classification_report(np.argmax(y_test, axis=1), cnn_preds)
# Visualize results using a bar chart
models = ['SVM', 'Random Forest', 'CNN']
```

```
accuracies = [svm_accuracy, rf_accuracy, cnn_accuracy]
print("\nSVM accuracy:", svm_accuracy)
print("Random Forest accuracy:", rf_accuracy)
print("CNN accuracy:", cnn_accuracy)
plt.bar(models, accuracies, color=['blue', 'green', 'red'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Comparison')
plt.ylim(0, 1.0)
plt.show()
# Print classification reports
print("SVM Classification Report:")
print(svm_report)
print("\nRandom Forest Classification Report:")
print(rf_report)
print("\nCNN Classification Report:")
print(cnn_report)
```

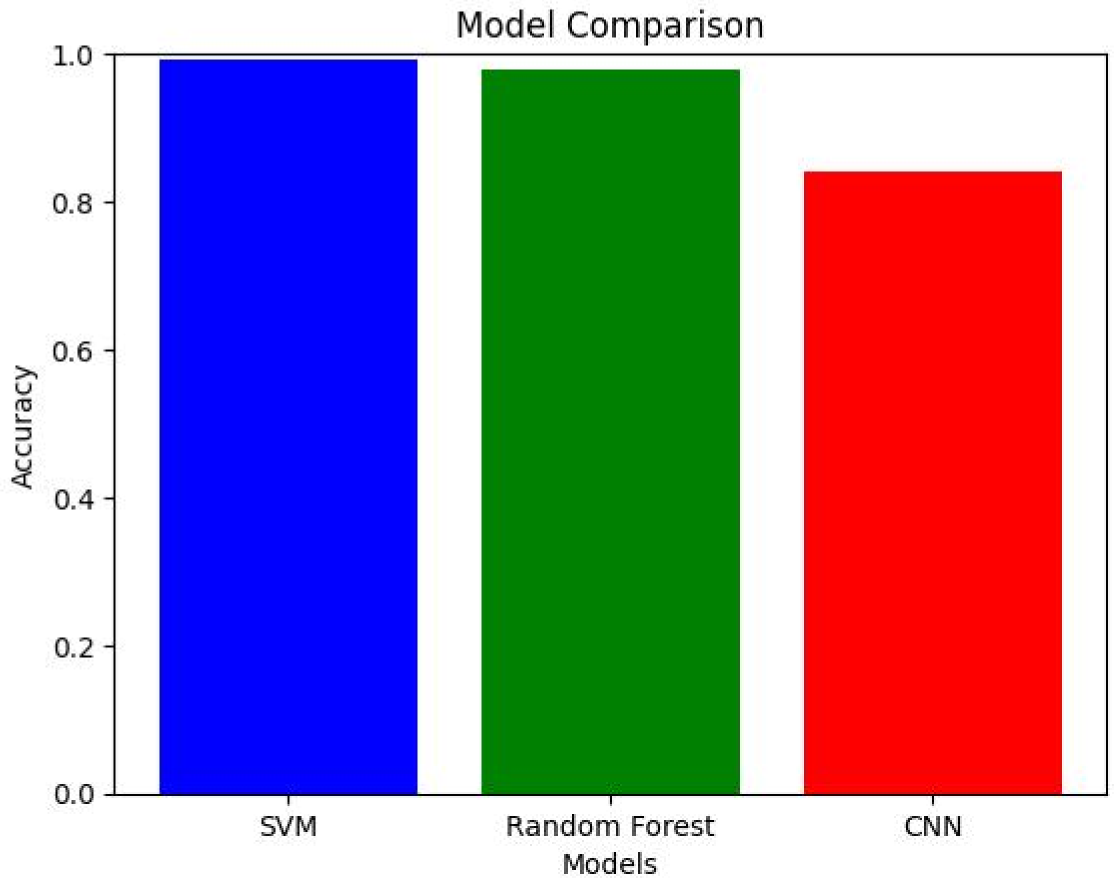
6. Output:



```
Epoch 1/5
45/45 [==============================] - 2s 11ms/step - loss: 2.2825 - accuracy: 0.2359
Epoch 2/5
45/45 [==============================] - 0s 9ms/step - loss: 2.0851 - accuracy: 0.3271
Epoch 3/5
45/45 [==============================] - 0s 10ms/step - loss: 1.4636 - accuracy: 0.6173
Epoch 4/5
45/45 [==============================] - 0s 10ms/step - loss: 0.9208 - accuracy: 0.7662
Epoch 5/5
45/45 [==============================] - 0s 8ms/step - loss: 0.6473 - accuracy: 0.8281
12/12 [==============================] - 0s 4ms/step

SVM accuracy: 0.9916666666666667
Random Forest accuracy: 0.9777777777777777
CNN accuracy: 0.8416666666666667
```

...



...

```
SVM Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        31
           1       0.94      1.00      0.97        33
           2       1.00      1.00      1.00        28
           3       1.00      1.00      1.00        39
           4       1.00      1.00      1.00        40
           5       1.00      1.00      1.00        41
           6       1.00      1.00      1.00        32
           7       1.00      0.98      0.99        43
           8       1.00      0.93      0.97        30
           9       0.98      1.00      0.99        43

    accuracy                           0.99       360
```