CourseName: Computer Vision Lab **Course Code:** CSP-422

Experiment:2.1

Aim:

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

Software Required:

Any IDE (Jupyter Notebook, Pycharm, Google Colab).

Relevance Of the Experiment:

The experiment aims to compare the performance of different classification models in image recognition. Image recognition is a fundamental task in computer vision and has various practical applications, such as object detection, facial recognition, and medical imaging. By comparing different classification models, researchers and practitioners can determine which models are most effective for specific image recognition tasks, enabling the development of more accurate and efficient systems.

Description:

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

- 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.
- 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.
- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

Pseudo code/Algorithms:

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.

CourseName: Computer Vision Lab **Course Code:** CSP-422

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.

Flowchart/Steps:

Implementation:

```
import numpy as np
from sklearn import datasets, model_selection
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load an image dataset (e.g., CIFAR-10)
(X_train, y_train), (X_test, y_test) = keras.datasets.cifar10.load_data()

# Flatten the images and scale the pixel values to the range [0, 1]
X_train = X_train.reshape(-1, 32*32*3) / 255.0
X_test = X_test.reshape(-1, 32*32*3) / 255.0

# Split the data into training and testing sets
X_train, X_val, y_train, y_val = model_selection.train_test_split(X_train, y_train,
test_size=0.2, random_state=42)
```

```
# Define and train classification models
models = [
    ("Random Forest", RandomForestClassifier(n_estimators=100, random_state=42)),
    ("Support Vector Machine", SVC(kernel='linear', C=1)),
    ("Neural Network", MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=100,
random_state=42)),
]

for name, model in models:
    print(f"Training {name}...")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    print(f"{name} Validation Accuracy: {accuracy:.2%}")

# Create a deep neural network model using TensorFlow/Keras
def create_neural_network():
    model = keras.Sequential([
        layers.Input(shape=(32*32*3,)),
        layers.Dense(128, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

print("Training Deep Neural Network...")
nn_model = create_neural_network()
nn_model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

CourseName: Computer Vision Lab **Course Code:** CSP-422

```
# Evaluate the deep neural network
y_nn_pred = nn_model.predict(X_val)
nn_accuracy = accuracy_score(y_val, np.argmax(y_nn_pred, axis=1))
print(f"Deep Neural Network Validation Accuracy: {nn_accuracy:.2%}")
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 513s 3us/step
Training Random Forest...
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:1151: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
Random Forest Validation Accuracy: 45.38%
Training Support Vector Machine...
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Support Vector Machine Validation Accuracy: 38.09%
Training Neural Network...
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:1102: DataConversionWarning: A colu
mn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (100) reached and the optimization hasn't converged yet.
  warnings.warn(
Neural Network Validation Accuracy: 48.06%
Training Deep Neural Network...
Epoch 1/10
1250/1250 [==============================] - 13s 9ms/step - loss: 1.9079 - accuracy: 0.3087 - val_loss: 1.8048 - val_accuracy:
0.3507
Epoch 2/10
1250/1250 [==============================] - 9s 7ms/step - loss: 1.7307 - accuracy: 0.3817 - val_loss: 1.6722 - val_accuracy:
0.3957
Epoch 3/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1.6643 - accuracy: 0.4049 - val_loss: 1.6549 - val_accuracy:
0.4069
Epoch 4/10
1250/1250 [==============================] - 9s 7ms/step - loss: 1.6110 - accuracy: 0.4275 - val_loss: 1.6469 - val_accuracy:
0.4045
Epoch 5/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1.5789 - accuracy: 0.4378 - val_loss: 1.5905 - val_accuracy:
0.4313
Epoch 6/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1.5472 - accuracy: 0.4521 - val_loss: 1.5571 - val_accuracy:
0.4463
Epoch 7/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1.5277 - accuracy: 0.4565 - val_loss: 1.5802 - val_accuracy:
0.4328
Epoch 8/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1.5111 - accuracy: 0.4609 - val_loss: 1.5673 - val_accuracy:
0.4327
Epoch 9/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1.4937 - accuracy: 0.4656 - val_loss: 1.5809 - val_accuracy:
0.4415
Epoch 10/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1.4822 - accuracy: 0.4694 - val_loss: 1.5577 - val_accuracy:
0.4424
313/313 [==============================] - 1s 2ms/step
Deep Neural Network Validation Accuracy: 44.24%
```