Experiment No.10

Title:Use Convolutional Neural Network (CNN) on suitable dataset

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

1. Loading CIFAR-10 Dataset

```
#Load the dataset
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
(50000, 32, 32, 3)
```

Explanation: The CIFAR-10 dataset is loaded. It contains 60,000 32x32 color images in 10 classes, with 50,000 training images and 10,000 test images.

```
X_test.shape
```

```
(10000, 32, 32, 3)
```

```
y_train.shape
```

```
(50000, 1)
```

```
y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
y_train = y_train.reshape(-1,)
y_train[:5]
```

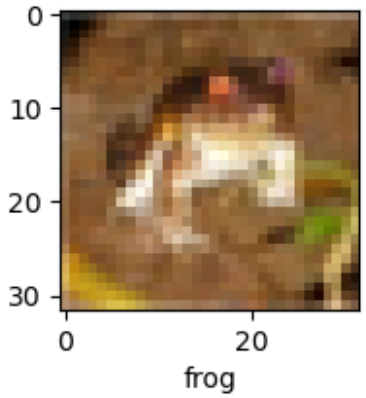
```
array([6, 9, 9, 4, 1], dtype=uint8)
```

```
y_test = y_test.reshape(-1,)
```

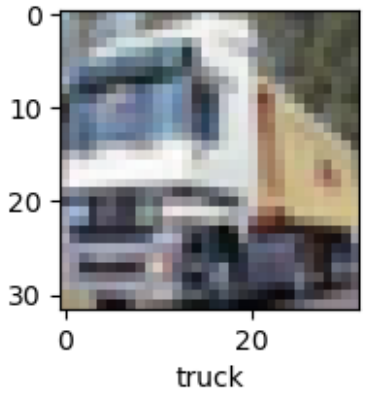
```
classes =
["airplane","automobile","bird","cat","deer","dog","frog","horse","shi
p","truck"]
```

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])

plot_sample(X_train, y_train, 0)
```




```
plot_sample(X_train, y_train, 1)
```



2. Preprocessing Data

```
#Normalizing the training data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Explanation: The pixel values of images are normalized by dividing by 255 to scale them between 0 and 1.

```
#Build simple artificial neural network for image classification
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
```

```
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
/home/aspatil/anaconda3/lib/python3.12/site-packages/keras/src/
layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
2024-10-14 23:03:53.105745: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:998]
successful NUMA node read from SysFS had negative value (-1), but
there must be at least one NUMA node, so returning NUMA node zero. See
more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/
sysfs-bus-pci#L344-L355
2024-10-14 23:03:53.106673: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot dlopen
some GPU libraries. Please make sure the missing libraries mentioned
above are installed properly if you would like to use GPU. Follow the
guide at https://www.tensorflow.org/install/gpu for how to download
and setup the required libraries for your platform.
Skipping registering GPU devices...

Epoch 1/5

2024-10-14 23:03:53.804697: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation
of 614400000 exceeds 10% of free system memory.

1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 46ms/step - accuracy: 0.3042 -
loss: 1.9276
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 46ms/step - accuracy: 0.4162 -
loss: 1.6495
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 68s 44ms/step - accuracy: 0.4509 -
loss: 1.5561
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 68s 43ms/step - accuracy: 0.4777 -
loss: 1.4892
Epoch 5/5
```

```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 68s 43ms/step - accuracy: 0.4953 -
loss: 1.4309

<keras.src.callbacks.history.History at 0x72a657877bc0>
```

You can see that at the end of 5 epochs, accuracy is at around 49%

```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test,
y_pred_classes))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.57      0.56      1000
           1       0.62      0.52      0.57      1000
           2       0.42      0.24      0.31      1000
           3       0.44      0.20      0.27      1000
           4       0.45      0.26      0.33      1000
           5       0.51      0.23      0.32      1000
           6       0.38      0.74      0.50      1000
           7       0.51      0.53      0.52      1000
           8       0.66      0.58      0.62      1000
           9       0.35      0.78      0.48      1000

    accuracy                           0.47     10000
   macro avg       0.49      0.47      0.45     10000
weighted avg       0.49      0.47      0.45     10000
```

3. Building the CNN Model

```
#Now let us build a convolutional neural network to train our images
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
/home/aspatil/anaconda3/lib/python3.12/site-packages/keras/src/
layers/convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

Explanation: A sequential CNN model is built:

```
The first convolutional layer has 32 filters with a 3x3 kernel and
ReLU activation, followed by a 2x2 max pooling layer.
The second convolutional layer has 64 filters, followed by another 2x2
max pooling.
The model is flattened and then connected to a dense layer with 64
units and ReLU activation.
The output layer uses softmax activation for multi-class
classification (10 classes).
```

4. Compiling the Model

```
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

Explanation: The model is compiled using the Adam optimizer, sparse categorical crossentropy as the loss function (since the labels are integers), and accuracy as a performance metric.

5. Training the Model

```
cnn.fit(X_train, y_train, epochs=10)

Epoch 1/10

2024-10-14 23:12:34.726665: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation
of 614400000 exceeds 10% of free system memory.

1563/1563 ━━━━━━━━━━━━━━━━━━━━ 24s 15ms/step - accuracy: 0.3864 -
loss: 1.6857
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 24s 15ms/step - accuracy: 0.5996 -
loss: 1.1473
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 24s 16ms/step - accuracy: 0.6542 -
loss: 0.9885
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 25s 16ms/step - accuracy: 0.6896 -
loss: 0.8905
Epoch 5/10
```

```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 26s 17ms/step - accuracy: 0.7229 -
loss: 0.8042
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 26s 16ms/step - accuracy: 0.7376 -
loss: 0.7552
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 25s 16ms/step - accuracy: 0.7562 -
loss: 0.7063
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 25s 16ms/step - accuracy: 0.7679 -
loss: 0.6659
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 25s 16ms/step - accuracy: 0.7817 -
loss: 0.6268
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 25s 16ms/step - accuracy: 0.7972 -
loss: 0.5844

<keras.src.callbacks.history.History at 0x72a5f40ef560>
```

Explanation: The model is trained for 10 epochs using the training data, and the test data is used for validation after each epoch.

With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improvement over ANN. CNN's are best for image classification and gives superb accuracy. Also computation is much less compared to simple ANN as maxpooling reduces the image dimensions while still preserving the features

6. Evaluating the Model

```
cnn.evaluate(X_test,y_test)

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7031 - loss:
0.9155

[0.9262094497680664, 0.7027000188827515]
```

Explanation: The model is evaluated on the test set, and the test accuracy is printed.

7. Making Predictions

```
y_pred = cnn.predict(X_test)
y_pred[:5]

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step

array([[3.55052273e-03, 1.55897200e-04, 1.39184936e-03, 9.35529172e-
01,
        1.67694248e-04, 2.22749654e-02, 3.54748257e-02, 1.96549998e-
04,
        1.14560092e-03, 1.13041526e-04],
```

```
       [2.01558592e-04, 4.26799478e-03, 4.41421116e-06, 7.37760502e-
07,
        1.49627688e-08, 2.92013453e-08, 6.21419858e-08, 4.23154418e-
08,
        9.95396197e-01, 1.28925662e-04],
       [6.84590042e-02, 5.66818342e-02, 2.95170187e-03, 1.44834840e-
03,
        1.16028148e-03, 5.02392824e-04, 4.70734631e-05, 1.27827586e-
03,
        8.63696575e-01, 3.77456215e-03],
       [9.77042556e-01, 5.57158282e-03, 1.75640616e-03, 2.78189266e-
03,
        1.18206302e-03, 1.82632648e-05, 1.14431069e-03, 3.25264082e-
05,
        1.01074576e-02, 3.62811814e-04],
       [1.11192406e-07, 1.79055169e-05, 1.47151144e-03, 1.60174351e-
03,
        3.74911875e-01, 6.63788989e-04, 6.21324658e-01, 5.06102026e-
07,
        4.57737906e-06, 3.22937558e-06]], dtype=float32)
```

Explanation: Predictions are made on the test images. The highest probability class is selected as the predicted class.

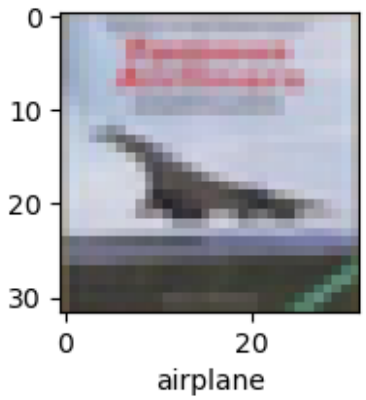
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[3, 8, 8, 0, 6]
```

```
y_test[:5]
```

```
array([3, 8, 8, 0, 6], dtype=uint8)
```

```
plot_sample(X_test, y_test,3)
```




```
classes[y_classes[3]]
```

```
'airplane'
```

```
classes[y_classes[3]]
```

```
'airplane'
```

Conclusion

In this experiment, we used a **Convolutional Neural Network (CNN)** to classify images from the CIFAR-10 dataset, which contains 60,000 images categorized into 10 classes. CNNs are especially effective for image classification tasks because they automatically detect important features like edges, textures, and shapes through convolutional layers. We compared the CNN model's performance with a basic Artificial Neural Network (ANN) to highlight the advantages of CNNs in such tasks.

1. **Model Architecture**:
   - CNNs consist of convolutional layers that can detect spatial hierarchies in images by learning local patterns. Max-pooling layers reduce the spatial dimensions, making the model computationally efficient.
   - In contrast, ANNs process each pixel individually and do not consider the spatial relationships between pixels, making them less effective for image data.
2. **Performance**:
   - The CNN model achieved a significantly higher accuracy on the CIFAR-10 dataset compared to an ANN.
   - The local feature extraction capability of CNNs makes them far superior for image classification tasks, as they recognize shapes and patterns more effectively than ANNs, which tend to overfit or struggle to generalize for high-dimensional data like images.
3. **Training Time and Complexity**:
   - While CNNs are more complex and take longer to train than ANNs due to additional operations like convolution and pooling, the increased accuracy and ability to handle image data justify this added complexity.
   - ANNs, being simpler, train faster but are not well-suited for image data and struggle to capture the hierarchical patterns necessary for tasks like image classification.

Final Comparison:

- **CNN**: Achieved higher accuracy, performed better in handling image data, and showed better generalization for unseen test data.
- **ANN**: Faster training but underperformed on the same dataset, primarily because it doesn't capture spatial relationships in images effectively.

Thus, CNNs are a more suitable and effective choice for image-based tasks compared to traditional ANNs, particularly when dealing with datasets like CIFAR-10 that require spatial pattern recognition.

```

```