Assignment: 2

Aim: Image Classification**.**

Step 1: Load and Explore CIFAR-10 Dataset in Google Colab

- Import necessary libraries: TensorFlow, Keras, Matplotlib, and NumPy.
- Load the CIFAR-10 dataset using Keras's built-in function "datasets.cifar10.load_data()".
- Check the shape of the training dataset to ensure it contains 50,000 images with dimensions 32x32 pixels and 3 color channels.
- Load and explore the CIFAR-10 dataset using Google Colab, including loading the dataset from Google Drive and visualizing sample images for analysis.

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

```
[2] (X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
    X_train.shape

    Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
    170498071/170498071 [==============================] - 4s 0us/step
    (50000, 32, 32, 3)
```

Step 2: Explore Dataset Dimensions and Labels

- Confirm the testing dataset's dimensions: 10,000 images of size 32x32 pixels with 3 color channels.
- Acknowledge dataset sizes: 50,000 training images and 10,000 test images.
- Analyze the structure of training labels (y_train): initially a 2D array (50000, 1), displaying the first five labels as integers (0 to 9).

- Optimize label representation for classification by converting y_train to a 1D array.

```
[3] X_test.shape

    (10000, 32, 32, 3)

[4] y_train.shape

    (50000, 1)

    y_train[:7]

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

Step 3: Optimize Label Representation

- Reshape y_train and y_test arrays to a 1D format for improved classification ease.
- Display the first five labels from the reshaped y_train array, now represented as integers (0 to 9).
- Define a list classes containing the class names corresponding to the label integers for better interpretation in classification results.

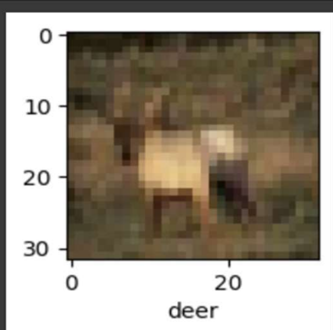
```
[7] y_train = y_train.reshape(-1,)
    y_train[:5]

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

[8] y_test = y_test.reshape(-1,)

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

Step 4: Visualize Sample Images

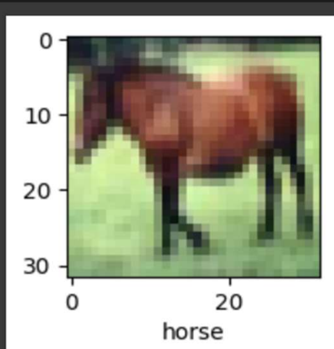
- Define a function plot_sample(X, y, index) to display a sample image along with its corresponding label.
- Set the figure size to 15x2 for better visualization.
- Plot the image at the specified index from the training dataset (X_train).
- Label the image with the corresponding class name using the classes list and the label (y_train[index]).

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

```
[12] plot_sample(X_train, y_train, 3)
```




```
[13] plot_sample(X_train, y_train, 7)
```



<u>Step 5: Neural Network Training Overview</u>

- Normalize pixel values of training and testing data to a range of 0 to 1.
- Define a sequential model with:
  1) Flatten layer to convert 3D image data into a 1D array.
  2) Dense layers with 3000 and 1000 neurons using sigmoid and ReLU activations respectively.
  3) Output layer with softmax activation for classification.
- Compile the model using stochastic gradient descent (SGD) optimizer, sparse categorical crossentropy loss, and accuracy metric.
- Train the model on the training data for 5 epochs.
- Review training progress, reaching approximately 39% accuracy after 5 epochs.

```
[14] X_train = X_train / 255.0
     X_test = X_test / 255.0
```

```
[16] ann = models.Sequential([
             layers.Flatten(input_shape=(32,32,3)),
             layers.Dense(3000, activation='sigmoid'),
             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)
```

```
Epoch 1/5
1563/1563 [==============================] - 146s 93ms/step - loss: 2.0426 - accuracy: 0.2542
Epoch 2/5
1563/1563 [==============================] - 146s 93ms/step - loss: 1.8765 - accuracy: 0.3260
Epoch 3/5
1563/1563 [==============================] - 146s 94ms/step - loss: 1.8112 - accuracy: 0.3547
Epoch 4/5
1563/1563 [==============================] - 156s 100ms/step - loss: 1.7590 - accuracy: 0.3743
Epoch 5/5
1563/1563 [==============================] - 146s 93ms/step - loss: 1.7173 - accuracy: 0.3901
<keras.src.callbacks.History at 0x7d9326411cc0>
```

Step 6: Model Evaluation with Classification Metrics

- Import necessary libraries: confusion_matrix, classification_report from sklearn.metrics, and numpy.
- Predict classes for the testing dataset using the trained model.
- Calculate predicted classes by selecting the index of the maximum probability from each prediction.
- Generate and display a classification report showing precision, recall, and F1-score for each class, along with overall accuracy.
- The report indicates the precision, recall, and F1-score for each class, as well as macro and weighted averages.

```
[17] 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 [==============================] - 10s 30ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.55      0.42      0.48      1000
           1       0.41      0.57      0.48      1000
           2       0.33      0.19      0.24      1000
           3       0.35      0.15      0.21      1000
           4       0.39      0.27      0.32      1000
           5       0.36      0.35      0.36      1000
           6       0.33      0.67      0.45      1000
           7       0.40      0.51      0.45      1000
           8       0.57      0.42      0.48      1000
           9       0.41      0.48      0.44      1000

    accuracy                           0.40     10000
   macro avg       0.41      0.40      0.39     10000
weighted avg       0.41      0.40      0.39     10000
```

Step 7: Convolutional Neural Network (CNN) Model Construction

- Define a sequential model (CNN) for a Convolutional Neural Network (CNN) architecture.
- Utilize Conv2D layers with ReLU activation to extract features from input images.

- Apply MaxPooling2D layers to reduce spatial dimensions and capture dominant features.
- Flatten the output from convolutional layers into a 1D array.
- Introduce fully connected Dense layers with ReLU activation to perform classification.
- Compile the model using the Adam optimizer, sparse categorical crossentropy loss, and accuracy metric.

```
[18] 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')
     ])
```

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

Step 8: CNN Model Training

- Train the CNN model (CNN) on the training data (X_train, y_train) for 7 epochs.
- Review the training progress, with each epoch displaying loss decreases and accuracy improves, indicating the model's learning process.

```
[20] cnn.fit(X_train, y_train, epochs=7)

     Epoch 1/7
     1563/1563 [==============================] - 62s 39ms/step - loss: 1.4849 - accuracy: 0.4629
     Epoch 2/7
     1563/1563 [==============================] - 66s 42ms/step - loss: 1.1448 - accuracy: 0.5980
     Epoch 3/7
     1563/1563 [==============================] - 61s 39ms/step - loss: 1.0049 - accuracy: 0.6492
     Epoch 4/7
     1563/1563 [==============================] - 61s 39ms/step - loss: 0.9177 - accuracy: 0.6797
     Epoch 5/7
     1563/1563 [==============================] - 61s 39ms/step - loss: 0.8479 - accuracy: 0.7057
     Epoch 6/7
     1563/1563 [==============================] - 60s 39ms/step - loss: 0.7907 - accuracy: 0.7248
     Epoch 7/7
     1563/1563 [==============================] - 62s 39ms/step - loss: 0.7417 - accuracy: 0.7419
     <keras.src.callbacks.History at 0x7d92b56efc70>
```

Step 9: Predicted Probabilities from CNN Model

- The output array displays the predicted probabilities for the first five test images across ten classes, generated by the trained Convolutional Neural Network (CNN) model.
- Each row represents a test image, and each column represents the probability of the image belonging to a specific class, ranging from 0 to 9.

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

```
313/313 [==============================] - 4s 11ms/step
array([[7.40700075e-03, 4.27877624e-03, 2.02666735e-03, 7.26002455e-01,
        4.60142561e-04, 1.33837268e-01, 3.18907797e-02, 3.15763536e-05,
        8.93085524e-02, 4.75671049e-03],
       [8.54847208e-03, 1.95791006e-01, 4.11347264e-06, 4.09195309e-06,
        4.93484400e-08, 3.45607219e-07, 9.34894295e-09, 2.40022082e-08,
        7.93549716e-01, 2.10218201e-03],
       [9.73862708e-02, 8.84638652e-02, 6.76183403e-03, 2.01888587e-02,
        1.93348643e-03, 1.43390882e-03, 2.35120868e-04, 1.49878336e-03,
        6.95230186e-01, 8.68676230e-02],
       [7.10481763e-01, 6.03079237e-03, 1.65384747e-02, 5.24569815e-03,
        2.40708645e-02, 8.40184948e-05, 5.39638917e-04, 1.14835944e-04,
        2.36401454e-01, 4.92537045e-04],
       [1.22831407e-04, 7.77254463e-05, 5.67302108e-02, 3.74471620e-02,
        9.37494412e-02, 5.17925806e-03, 8.06030750e-01, 2.43259547e-06,
        5.43321483e-04, 1.16756062e-04]], dtype=float32)
```

Step 10: Predicted and True Classes Comparison

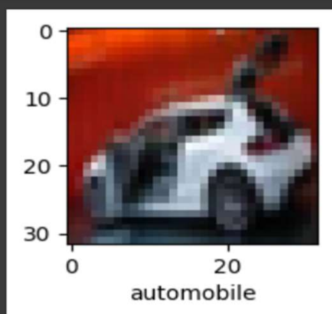
- Compare the predicted classes (y_classes) generated by the CNN model with the true classes (y_test) from the test dataset for the first five images.
- The arrays y_classes and y_test show the predicted and true classes respectively, indicating successful predictions when they match.

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

     [3, 8, 8, 0, 6]

[23] y_test[:5]

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

Step 11: Visualization and Prediction Verification

- Visualize the 6th image from the test dataset (X_test) along with its true label, showing an automobile.
- Verify the prediction result by checking the class name corresponding to the predicted class index, which matches the true label, confirming the prediction accuracy.

```
[25] plot_sample(X_test, y_test,6)
```

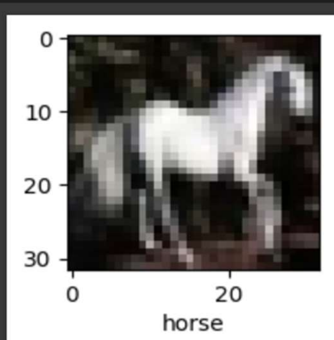



```
[27] classes[y_classes[6]]
```

```
'automobile'
```

- Visualize the 13th image from the test dataset (X_test) along with its true label, showing a Horse.

```
[35] plot_sample(X_test, y_test,13)
```




```
[34] classes[y_classes[13]]
```

```
'horse'
```