Small Image Classification Using Convolutional Neural Network (CNN)

In this notebook, we will classify small images cifar10 dataset from tensorflow keras datasets. There are total 10 classes as shown below. We will use CNN for classification

Start coding or generate with AI.

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

Load the dataset

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

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

```
X_test.shape
```

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

```
X_train.shape
```

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

Here we see there are 50000 training images and 1000 test images

```
y_train.shape
```

```
(50000, 1)
```

```
X_train[1]
```

```
array([[[154, 177, 187],
        [126, 137, 136],
        [105, 104,  95],
        ...,
        [ 91,  95,  71],
        [ 87,  90,  71],
        [ 79,  81,  70]],

       [[140, 160, 169],
        [145, 153, 154],
        [125, 125, 118],
        ...,
        [ 96,  99,  78],
        [ 77,  80,  62],
        [ 71,  73,  61]],

       [[140, 155, 164],
        [139, 146, 149],
        [115, 115, 112],
        ...,
        [ 79,  82,  64],
        [ 68,  70,  55],
        [ 67,  69,  55]],

       ...,

       [[175, 167, 166],
        [156, 154, 160],
        [154, 160, 170],
        ...,
        [ 42,  34,  36],
        [ 61,  53,  57],
        [ 93,  83,  91]],

       [[165, 154, 128],
        [156, 152, 130],
        [159, 161, 142],
```

```
        ...,
        [103,  93,  96],
        [123, 114, 120],
        [131, 121, 131]],

       [[163, 148, 120],
        [158, 148, 122],
        [163, 156, 133],
        ...,
        [143, 133, 139],
        [143, 134, 142],
        [143, 133, 144]]], dtype=uint8)
```

```
y_train[:5]
```

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

y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to now 1D array

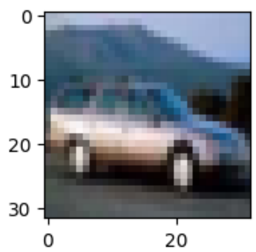
```
y_train = y_train.reshape(-1,)
y_train[:5]
```

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

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

```
plt.figure(figsize = (15,2))
plt.imshow(X_train[4])
```

```
<matplotlib.image.AxesImage at 0x7a11c35fc280>
```



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

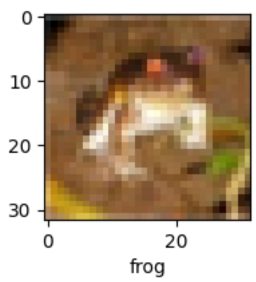
```
classes[9]
```

```
'truck'
```

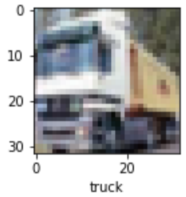
Let's plot some images to see what they are

```
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)
```



Normalize the images to a number from 0 to 1. Image has 3 channels (R,G,B) and each value in the channel can range from 0 to 255. Hence to normalize in 0-->1 range, we need to divide it by 255

Normalizing the training data

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

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)
```

```
Epoch 1/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.8074 - accuracy: 0.3561
Epoch 2/5
1563/1563 [==============================] - 2s 1ms/step - loss: 1.6208 - accuracy: 0.4285
Epoch 3/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.5380 - accuracy: 0.4585
Epoch 4/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.4808 - accuracy: 0.4806
Epoch 5/5
1563/1563 [==============================] - 2s 2ms/step - loss: 1.4326 - accuracy: 0.4928
<tensorflow.python.keras.callbacks.History at 0x295ab873c10>
```

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))
```

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

           0       0.63      0.45      0.53      1000
           1       0.72      0.46      0.56      1000
           2       0.33      0.46      0.39      1000
           3       0.36      0.25      0.29      1000
           4       0.44      0.37      0.40      1000
           5       0.34      0.46      0.39      1000
           6       0.56      0.47      0.51      1000
           7       0.39      0.67      0.50      1000
           8       0.64      0.60      0.62      1000
           9       0.59      0.53      0.55      1000

    accuracy                           0.47     10000
   macro avg       0.50      0.47      0.47     10000
weighted avg       0.50      0.47      0.47     10000
```

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')
])
```

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

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

```
Epoch 1/10
1563/1563 [==============================] - 2s 2ms/step - loss: 1.4407 - accuracy: 0.4810
Epoch 2/10
1563/1563 [==============================] - 2s 2ms/step - loss: 1.1084 - accuracy: 0.6109
Epoch 3/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.9895 - accuracy: 0.6574
Epoch 4/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.9071 - accuracy: 0.6870
Epoch 5/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.8416 - accuracy: 0.7097
Epoch 6/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.7847 - accuracy: 0.7262
Epoch 7/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.7350 - accuracy: 0.7448
Epoch 8/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.6941 - accuracy: 0.7574
Epoch 9/10
1563/1563 [==============================] - 2s 1ms/step - loss: 0.6516 - accuracy: 0.7731
Epoch 10/10
1563/1563 [==============================] - 2s 2ms/step - loss: 0.6187 - accuracy: 0.7836
<tensorflow.python.keras.callbacks.History at 0x296555783d0>
```

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

```
cnn.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.9022 - accuracy: 0.7028
[0.9021560549736023, 0.7027999758720398]
```

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

```
array([[4.3996371e-04, 3.4844263e-05, 1.5558505e-03, 8.8400185e-01,
        1.9452239e-04, 3.5314459e-02, 7.2777577e-02, 6.9044131e-06,
        5.6417785e-03, 3.2224660e-05],
       [8.1062522e-03, 5.0841425e-02, 1.2453231e-07, 5.3348430e-07,
        9.1728407e-07, 1.0009186e-08, 2.8985988e-07, 1.7532484e-09,
        9.4089705e-01, 1.5346886e-04],
       [1.7055811e-02, 1.1841061e-01, 4.6799007e-05, 2.7727904e-02,
        1.0848254e-03, 1.0896578e-03, 1.3575243e-04, 2.8652203e-04,
        7.8895986e-01, 4.5202184e-02],
       [3.1300801e-01, 1.1591638e-02, 1.1511055e-02, 3.9592334e-03,
        7.7280165e-03, 5.6289224e-05, 2.3531138e-04, 9.4204297e-06,
        6.5178138e-01, 1.1968113e-04],
       [1.3230885e-05, 2.1221960e-05, 9.2594400e-02, 3.3585075e-02,
        4.4722903e-01, 4.1028224e-03, 4.2241842e-01, 2.8064171e-05,
        6.6392668e-06, 1.0745022e-06]], dtype=float32)
```

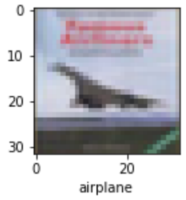
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[3, 8, 8, 8, 4]
```

```
y_test[:5]
```

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

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




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

    'ship'

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

    'ship'

Exercise

Use CNN to do handwritten digits classification using MNIST dataset. You can use this notebook as a reference: https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/main/1_digits_recognition/digits_recognition_neural_network.ipynb

Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN. Once you have worked on this problem on your own, you can check my solution by clicking on this link: Solution