```
Name: Borris A. Esplanada
Instructor: Engr. Roman Richard
Date: 6/23/24
Section: CPE 019 - CPE 32S1
```

Part 1: Try the MLP Notebook using the CIFAR10 Keras Dataset

```
# importing modules
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt
```

```
data = tf.keras.datasets.cifar10.load_data()
```

```
data
```

```
((array([[[[ 59,  62,  63],
           [ 43,  46,  45],
           [ 50,  48,  43],
           ...,
           [158, 132, 108],
           [152, 125, 102],
           [148, 124, 103]],

          [[ 16,  20,  20],
           [  0,   0,   0],
           [ 18,   8,   0],
           ...,
           [123,  88,  55],
           [119,  83,  50],
           [122,  87,  57]],

          [[ 25,  24,  21],
           [ 16,   7,   0],
           [ 49,  27,   8],
           ...,
           [118,  84,  50],
           [120,  84,  50],
           [109,  73,  42]],

          ...,

          [[208, 170,  96],
           [201, 153,  34],
           [198, 161,  26],
           ...,
           [160, 133,  70],
           [ 56,  31,   7],
           [ 53,  34,  20]],

          [[180, 139,  96],
           [173, 123,  42],
           [186, 144,  30],
           ...,
           [184, 148,  94],
           [ 97,  62,  34],
           [ 83,  53,  34]],

          [[177, 144, 116],
           [168, 129,  94],
           [179, 142,  87],
           ...,
           [216, 184, 140],
           [151, 118,  84],
           [123,  92,  72]]],


         [[[154, 177, 187],
           [126, 137, 136],
           [105, 104,  95],
```

```
                ...,
                [ 91,  95,  71],
                [ 87,  90,  71],
                [ 79,  81,  70]],
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
```

Convert the pixels into floating-point values.

```
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale # x_train = x_train/ 255
x_test /= gray_scale
```

We are converting the pixel values into floating-point values to make the predictions. Changing the numbers into grayscale values will be beneficial as the values become small and the computation becomes easier and faster. As the pixel values range from 0 to 256, apart from 0 the range is 255. So dividing all the values by 255 will convert it to range from 0 to 1

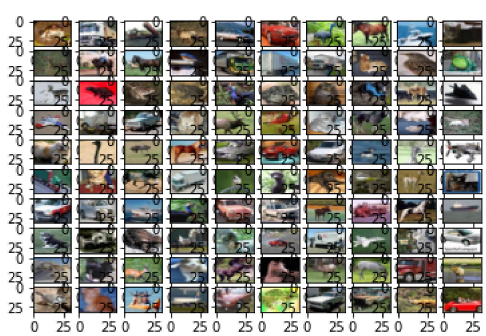
```
# Understand the structure of the dataset

print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)
```

```
Feature matrix: (50000, 32, 32, 3)
Target matrix: (10000, 32, 32, 3)
Feature matrix: (50000, 1)
Target matrix: (10000, 1)
```

```
# Data visualization

fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(x_train[k].reshape(32, 32, 3),
                        aspect='auto')
        k += 1
plt.show()
```



```
# Form the Input, hidden, and output layers.

model = Sequential([


    Flatten(input_shape=(32, 32, 3)),

    # dense layer 1
    Dense(128, activation='relu'),

    # dense layer 2
    Dense(64, activation='relu'),

    # dense layer 3
    Dense(32, activation='relu'),

    # output layer
  Dense(10, activation='sigmoid')
    ])


model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 3072)              0

 dense_14 (Dense)            (None, 128)               393344

 dense_15 (Dense)            (None, 64)                8256

 dense_16 (Dense)            (None, 32)                2080

 dense_17 (Dense)            (None, 10)                330

=================================================================
Total params: 404,010
Trainable params: 404,010
Non-trainable params: 0
_________________________________________________________________
```

Some important points to note:

The **Sequential model** allows us to create models layer-by-layer as we need in a multi-layer perceptron and is limited to single-input, single-output stacks of layers.

Flatten flattens the input provided without affecting the batch size. For example, If inputs are shaped (batch_size,) without a feature axis, then flattening adds an extra channel dimension and output shape is (batch_size, 1).

Activation is for using the sigmoid activation function.

The first two **Dense layers** are used to make a fully connected model and are the hidden layers.

The last Dense layer is the **output layer** which contains 10 neurons that decide which category the image belongs to.

```
# Compile the model

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

Compile function is used here that involves the use of loss, optimizers, and metrics. Here loss function used is **sparse_categorical_crossentropy**, optimizer used is **adam**.

```
# Fit the model

model.fit(x_train, y_train, epochs=50,
    batch_size=15000,
    validation_split=0.80)
```

```
Epoch 18/50
1/1 [==============================] - 2s 2s/step - loss: 1.3952 - accuracy: 0.5163 - val_loss: 1.6579 - val_accuracy: 0.4227
Epoch 19/50
1/1 [==============================] - 2s 2s/step - loss: 1.4012 - accuracy: 0.5107 - val_loss: 1.6570 - val_accuracy: 0.4254
Epoch 20/50
1/1 [==============================] - 1s 1s/step - loss: 1.3976 - accuracy: 0.5142 - val_loss: 1.6512 - val_accuracy: 0.4253
Epoch 21/50
1/1 [==============================] - 3s 3s/step - loss: 1.3920 - accuracy: 0.5151 - val_loss: 1.6431 - val_accuracy: 0.4296
Epoch 22/50
1/1 [==============================] - 2s 2s/step - loss: 1.3827 - accuracy: 0.5191 - val_loss: 1.6412 - val_accuracy: 0.4299
Epoch 23/50
1/1 [==============================] - 2s 2s/step - loss: 1.3786 - accuracy: 0.5212 - val_loss: 1.6451 - val_accuracy: 0.4262
Epoch 24/50
1/1 [==============================] - 1s 1s/step - loss: 1.3812 - accuracy: 0.5174 - val_loss: 1.6517 - val_accuracy: 0.4270
Epoch 25/50
1/1 [==============================] - 2s 2s/step - loss: 1.3870 - accuracy: 0.5169 - val_loss: 1.6599 - val_accuracy: 0.4227
Epoch 26/50
1/1 [==============================] - 1s 1s/step - loss: 1.3968 - accuracy: 0.5103 - val_loss: 1.6625 - val_accuracy: 0.4244
Epoch 27/50
1/1 [==============================] - 1s 1s/step - loss: 1.3955 - accuracy: 0.5152 - val_loss: 1.6535 - val_accuracy: 0.4244
Epoch 28/50
1/1 [==============================] - 2s 2s/step - loss: 1.3886 - accuracy: 0.5149 - val_loss: 1.6435 - val_accuracy: 0.4294
Epoch 29/50
1/1 [==============================] - 2s 2s/step - loss: 1.3760 - accuracy: 0.5229 - val_loss: 1.6443 - val_accuracy: 0.4292
Epoch 30/50
1/1 [==============================] - 2s 2s/step - loss: 1.3739 - accuracy: 0.5221 - val_loss: 1.6479 - val_accuracy: 0.4278
Epoch 31/50
1/1 [==============================] - 2s 2s/step - loss: 1.3792 - accuracy: 0.5179 - val_loss: 1.6514 - val_accuracy: 0.4282
Epoch 32/50
1/1 [==============================] - 2s 2s/step - loss: 1.3819 - accuracy: 0.5201 - val_loss: 1.6539 - val_accuracy: 0.4251
Epoch 33/50
1/1 [==============================] - 2s 2s/step - loss: 1.3820 - accuracy: 0.5169 - val_loss: 1.6457 - val_accuracy: 0.4284
Epoch 34/50
1/1 [==============================] - 2s 2s/step - loss: 1.3718 - accuracy: 0.5253 - val_loss: 1.6436 - val_accuracy: 0.4291
Epoch 35/50
1/1 [==============================] - 2s 2s/step - loss: 1.3693 - accuracy: 0.5212 - val_loss: 1.6470 - val_accuracy: 0.4286
Epoch 36/50
1/1 [==============================] - 2s 2s/step - loss: 1.3711 - accuracy: 0.5210 - val_loss: 1.6521 - val_accuracy: 0.4282
Epoch 37/50
1/1 [==============================] - 2s 2s/step - loss: 1.3749 - accuracy: 0.5222 - val_loss: 1.6503 - val_accuracy: 0.4254
Epoch 38/50
1/1 [==============================] - 1s 1s/step - loss: 1.3774 - accuracy: 0.5190 - val_loss: 1.6495 - val_accuracy: 0.4283
Epoch 39/50
1/1 [==============================] - 2s 2s/step - loss: 1.3691 - accuracy: 0.5250 - val_loss: 1.6428 - val_accuracy: 0.4301
Epoch 40/50
1/1 [==============================] - 2s 2s/step - loss: 1.3638 - accuracy: 0.5262 - val_loss: 1.6431 - val_accuracy: 0.4292
Epoch 41/50
1/1 [==============================] - 2s 2s/step - loss: 1.3645 - accuracy: 0.5256 - val_loss: 1.6505 - val_accuracy: 0.4292
Epoch 42/50
1/1 [==============================] - 3s 3s/step - loss: 1.3680 - accuracy: 0.5249 - val_loss: 1.6508 - val_accuracy: 0.4252
Epoch 43/50
1/1 [==============================] - 2s 2s/step - loss: 1.3709 - accuracy: 0.5211 - val_loss: 1.6515 - val_accuracy: 0.4281
Epoch 44/50
1/1 [==============================] - 1s 1s/step - loss: 1.3668 - accuracy: 0.5232 - val_loss: 1.6457 - val_accuracy: 0.4283
Epoch 45/50
1/1 [==============================] - 1s 1s/step - loss: 1.3635 - accuracy: 0.5225 - val_loss: 1.6487 - val_accuracy: 0.4289
Epoch 46/50
1/1 [==============================] - 2s 2s/step - loss: 1.3608 - accuracy: 0.5250 - val_loss: 1.6438 - val_accuracy: 0.4295
Epoch 47/50
```

Some important points to note:

Epochs tell us the number of times the model will be trained in forwarding and backward passes.

Batch Size represents the number of samples, If it's unspecified, batch_size will default to 32.

Validation Split is a float value between 0 and 1. The model will set apart this fraction of the training data to evaluate the loss and any model metrics at the end of each epoch. (The model will not be trained on this data)

```
# Find the accuracy of the model

results = model.evaluate(x_test, y_test, verbose = 1)
print('test loss, test acc:', results)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.6397 - accuracy: 0.4284
test loss, test acc: [1.6396605968475342, 0.4284000098705292]
```

Conclusion

- To conclude, Building an MLP model needs data preparation, model architecture design, training, and evaluation. The effectiveness of the model depends on how well these steps are executed. In summary, through this assignment I've gained practical experience in implementing an MLP model for image classification, which is foundational knowledge in machine learning and neural networks.

We got the **accuracy** of our model 42% by using `model.evaluate()` on the test samples.

Google Collab Link: https://drive.google.com/file/d/1Fr7j0KHFZshGEat4znqmiJW54XntetLm/view?usp=sharing