Classification of Images from the CIFAR-10 Dataset using ANN (MLP) & CNN

OBJECTIVE:

The objective of the "Classification of Images from the CIFAR-10 Dataset using ANN (MLP) & CNN" lab is to equip learners with the knowledge and skills to build and evaluate image classification models using Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN). The course begins with fundamental concepts of neural networks and image data representation. It advances to implementing Multilayer Perceptrons (MLP) and CNNs using Python libraries such as TensorFlow and Keras. Learners will gain hands-on experience in preprocessing image data, designing network architectures, and training models to classify images into ten different categories from the CIFAR-10 dataset. By the end of the lab, learners will be proficient in applying ANN and CNN techniques to image classification tasks, enabling them to tackle similar problems in real-world applications.

Classification of Images from the CIFAR-10 Dataset using ANN (MLP) & CNN. The CIFAR-10 dataset consists of color 60,000 images each with 32 x 32 pixel in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.

Class labels are:

airplane : 0, automobile : 1, bird : 2, cat : 3, deer : 4, dog : 5, frog : 6, horse : 7, ship : 8, truck : 9.

Import Tensorflow

```
#!pip install matplotlib
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
import matplotlib
matplotlib.__version__
```

```
'3.8.0'
```

```
tf.__version__
```

```
'2.17.1'
```

Check for GPU

```
physical_devices = tf.config.experimental.list_physical_devices('GPU')
#physical_devices
print("Num GPUs Available: ", len(physical_devices))
#tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

```
Num GPUs Available:  0
```

Load Dataset

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

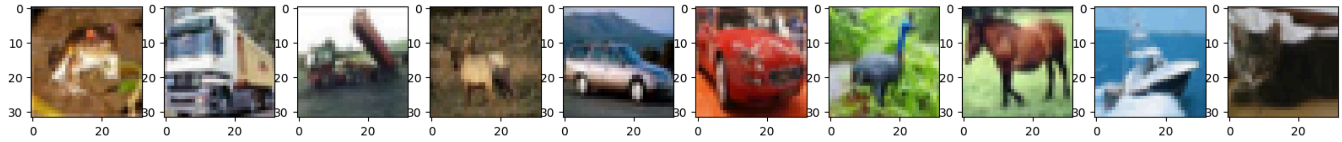
Show some sample images of data set with corresponding labels.

```
cifar10_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
print('Example training images and their labels: ' + str([x[0] for x in y_train[0:10]]))
print('Corresponding classes for the labels: ' + str([cifar10_classes[x[0]] for x in y_train[0:10]]))

fig, axarr = plt.subplots(1, 10)
fig.set_size_inches(20, 6)

for i in range(10):
    image = x_train[i]
    axarr[i].imshow(image)
plt.show()
```

```
Example training images and their labels: [6, 9, 9, 4, 1, 1, 2, 7, 8, 3]
Corresponding classes for the labels: ['frog', 'truck', 'truck', 'deer', 'automobile', 'automobile', 'bird', 'horse', 'ship', 'cat'
```




```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((50000, 32, 32, 3), (50000, 1), (10000, 32, 32, 3), (10000, 1))
```

```
model.compile(optimizer ='adam',
              loss = 'categorical_crossentropy',
              metrics =['accracy'])
```

Preparing the dataset Normalize the input data

```
X_train = x_train / 255.0
X_test = x_test / 255.0
# Every Neuron is expected to have value from 0 to 1 to converge quickly(Gradient Descent)
```

```
# Import necessary libraries
from tensorflow.keras.utils import to_categorical


train_labels = [0, 1, 2, 1, 0]  # Example labels

# Convert labels to one-hot encoding
train_labels_one_hot = to_categorical(train_labels)

# Print the one-hot encoded labels
print(train_labels_one_hot)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

MLP Network I/p Layer - Flatten Hidden layer - 2048, AF = 'RELU' O/p Layer - 10 , AF-Softmax

```
from tensorflow import keras
from keras.layers import Dense
from keras.layers import Flatten
```

```
ann = keras.Sequential()
ann.add(Flatten(input_shape=(32,32,3)))
ann.add(Dense(2048,activation='relu'))
ann.add(Dense(10,activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_c
  super().__init__(**kwargs)
```

```
ann.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 2048)	6,293,504
dense_1 (Dense)	(None, 10)	20,490

Total params: 6,313,994 (24.09 MB)
Trainable params: 6,313,994 (24.09 MB)
Non-trainable params: 0 (0.00 B)

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

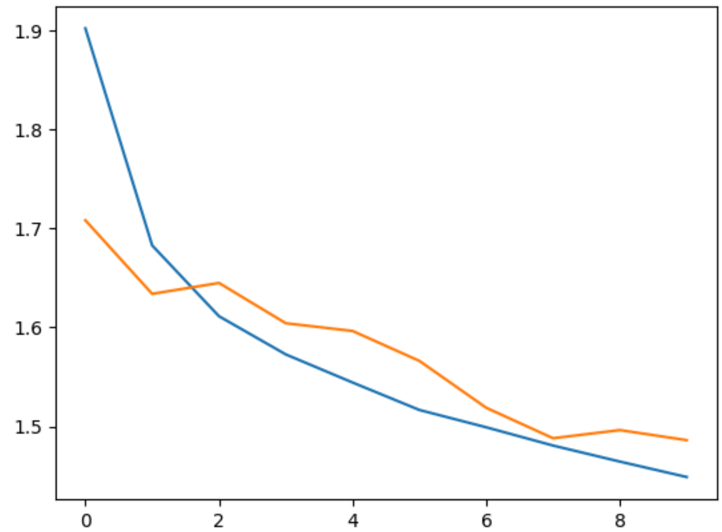
```
history = ann.fit(X_train ,y_train,epochs=10,validation_data=(X_test,y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 142s 90ms/step - accuracy: 0.2903 - loss: 2.2475 - val_accuracy: 0.4005 - val_loss: 1.7081
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 141s 90ms/step - accuracy: 0.3911 - loss: 1.6958 - val_accuracy: 0.4150 - val_loss: 1.6338
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 147s 93ms/step - accuracy: 0.4176 - loss: 1.6256 - val_accuracy: 0.4204 - val_loss: 1.6448
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 197s 90ms/step - accuracy: 0.4415 - loss: 1.5707 - val_accuracy: 0.4180 - val_loss: 1.6042
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 141s 89ms/step - accuracy: 0.4497 - loss: 1.5573 - val_accuracy: 0.4271 - val_loss: 1.5964
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 145s 91ms/step - accuracy: 0.4612 - loss: 1.5165 - val_accuracy: 0.4460 - val_loss: 1.5661
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 204s 92ms/step - accuracy: 0.4686 - loss: 1.4955 - val_accuracy: 0.4612 - val_loss: 1.5189
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 210s 97ms/step - accuracy: 0.4788 - loss: 1.4603 - val_accuracy: 0.4728 - val_loss: 1.4881
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 192s 91ms/step - accuracy: 0.4808 - loss: 1.4647 - val_accuracy: 0.4737 - val_loss: 1.4963
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 207s 94ms/step - accuracy: 0.4892 - loss: 1.4413 - val_accuracy: 0.4732 - val_loss: 1.4862
```

With the below simple function we will be able to plot our training history.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7d7034772230>]
```



CNN Model

```
from tensorflow import keras
from keras.layers import Conv2D, Dense, Flatten, MaxPooling2D, Dropout
```

```
cnn = keras.Sequential()
cnn.add(Conv2D(32, kernel_size= (3,3), strides=(1,1), padding='same', activation='relu', input_shape = (32,32,3)))
cnn.add(MaxPooling2D((2,2)))
cnn.add(Conv2D(64, kernel_size= (3,3), strides=(1,1), padding='same', activation='relu'))
cnn.add(MaxPooling2D((2,2)))
cnn.add(Conv2D(128, kernel_size= (3,3), strides=(1,1), padding='same', activation='relu'))
cnn.add(MaxPooling2D((2,2)))
cnn.add(Conv2D(256, kernel_size= (3,3), strides=(1,1), padding='same', activation='relu'))
cnn.add(MaxPooling2D((2,2)))
cnn.add(Flatten())
cnn.add(Dense(64,activation='relu'))
cnn.add(Dropout(0.3))
cnn.add(Dense(10,activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
cnn.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
conv2d_3 (Conv2D)	(None, 4, 4, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 64)	65,600
dropout (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 10)	650

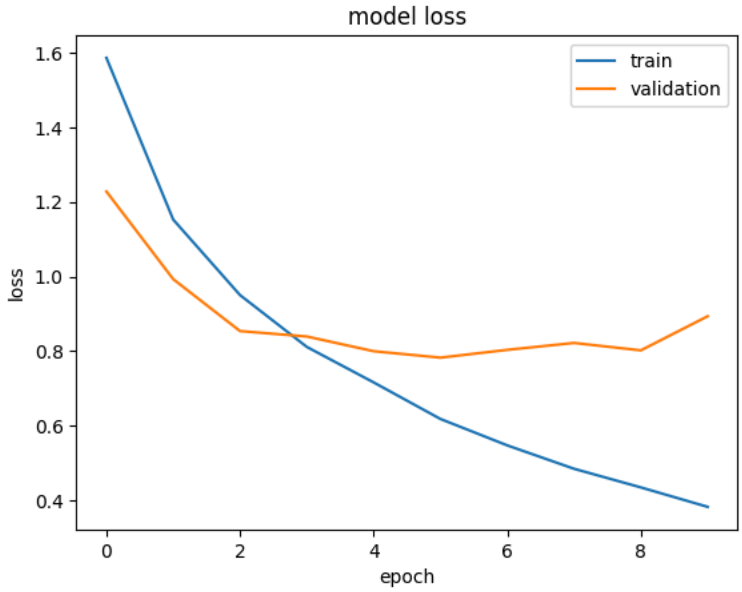
Total params: 454,666 (1.73 MB)
Trainable params: 454,666 (1.73 MB)

```
cnn.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
history = cnn.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 187s 118ms/step - accuracy: 0.3167 - loss: 1.8214 - val_accuracy: 0.5587 - val_loss: 1.2280
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 185s 118ms/step - accuracy: 0.5706 - loss: 1.2106 - val_accuracy: 0.6520 - val_loss: 0.9928
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 219s 130ms/step - accuracy: 0.6603 - loss: 0.9682 - val_accuracy: 0.7083 - val_loss: 0.8535
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 246s 119ms/step - accuracy: 0.7151 - loss: 0.8216 - val_accuracy: 0.7161 - val_loss: 0.8389
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 219s 130ms/step - accuracy: 0.7556 - loss: 0.7053 - val_accuracy: 0.7320 - val_loss: 0.7995
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 250s 123ms/step - accuracy: 0.7906 - loss: 0.6070 - val_accuracy: 0.7352 - val_loss: 0.7822
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 192s 116ms/step - accuracy: 0.8193 - loss: 0.5260 - val_accuracy: 0.7431 - val_loss: 0.8031
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 202s 117ms/step - accuracy: 0.8389 - loss: 0.4687 - val_accuracy: 0.7432 - val_loss: 0.8217
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 183s 117ms/step - accuracy: 0.8573 - loss: 0.4134 - val_accuracy: 0.7554 - val_loss: 0.8017
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 183s 117ms/step - accuracy: 0.8731 - loss: 0.3604 - val_accuracy: 0.7501 - val_loss: 0.8933
```

```
def plotLosses(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.show()
```

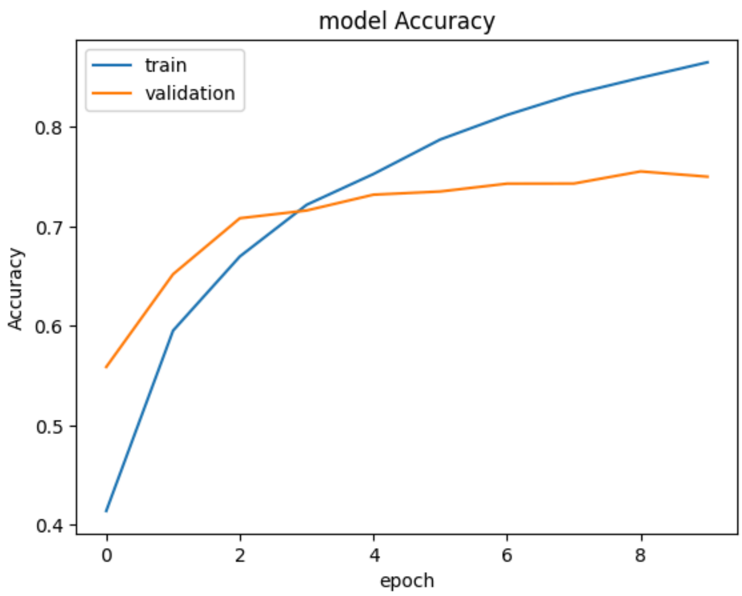
```
plotLosses(history)
```




```
def plotAccuracy(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('epoch')

    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
```

```
plotAccuracy(history)
```




```
from keras.models import load_model
cnn.save('model111.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
# Load the model
model = tf.keras.models.load_model('model111.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
```

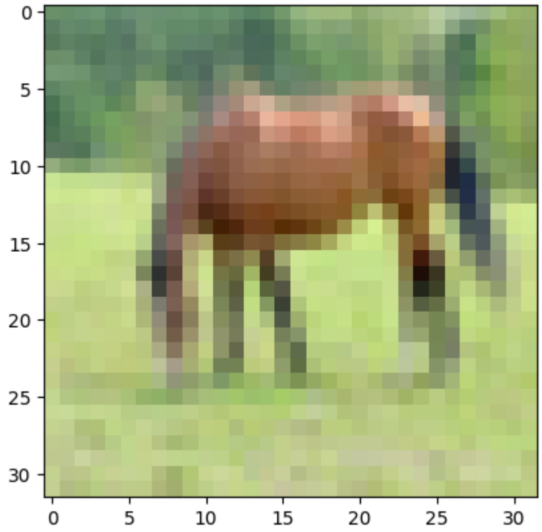
```
import numpy as np
# Add a batch dimension to the input
x_test_sample = np.expand_dims(x_test[20], axis=0)
```

```
# Now pass it to the model for prediction
model.predict(x_test_sample)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
array([[0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]], dtype=float32)
```

```
plt.imshow(x_test[60])
```

```
&lt;matplotlib.image.AxesImage at 0x7d7031866bf0&gt;
```




```
# Example: if you have class names like this
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']  # replace with your actual cla

# Get the prediction probabilities
predictions = model.predict(x_test_sample)

# Get the index of the class with the highest probability
predicted_class_index = np.argmax(predictions)

# Get the corresponding class name
predicted_class_name = class_names[predicted_class_index]
print(f"The predicted class is: {predicted_class_name}")
```