Program 3: CIFAR-10

Name: SOURABH S
Reg No: KH.EN.P2MCA25157
Course: MCA B (AI & DS)
Batch: 2025–2027

In [1]:
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

In [ ]:
```
# Load cifar10 dataset using cifar10 dataset module
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
#saving it as training set and test set( split it as )
```

In [ ]:
```
x_train.shape
#50000 images of 32x32 pixels in training set
```

Out[ ]:
```
(50000, 32, 32, 3)
```

In [4]:
```
x_test.shape
#10000 images of 32x32 pixels in test set)
```

Out[4]:
```
(10000, 32, 32, 3)
```

In [5]:
```
print(y_test)
#labels of test set
```

```
[[3]
 [8]
 [8]
 ...
 [5]
 [1]
 [7]]
```

In [6]:
```
#Displaying Sample Images from the MNIST Dataset
# Select random indices for displaying images
import numpy as np
indices = np.random.randint(0, x_train.shape[0], size=16)
```

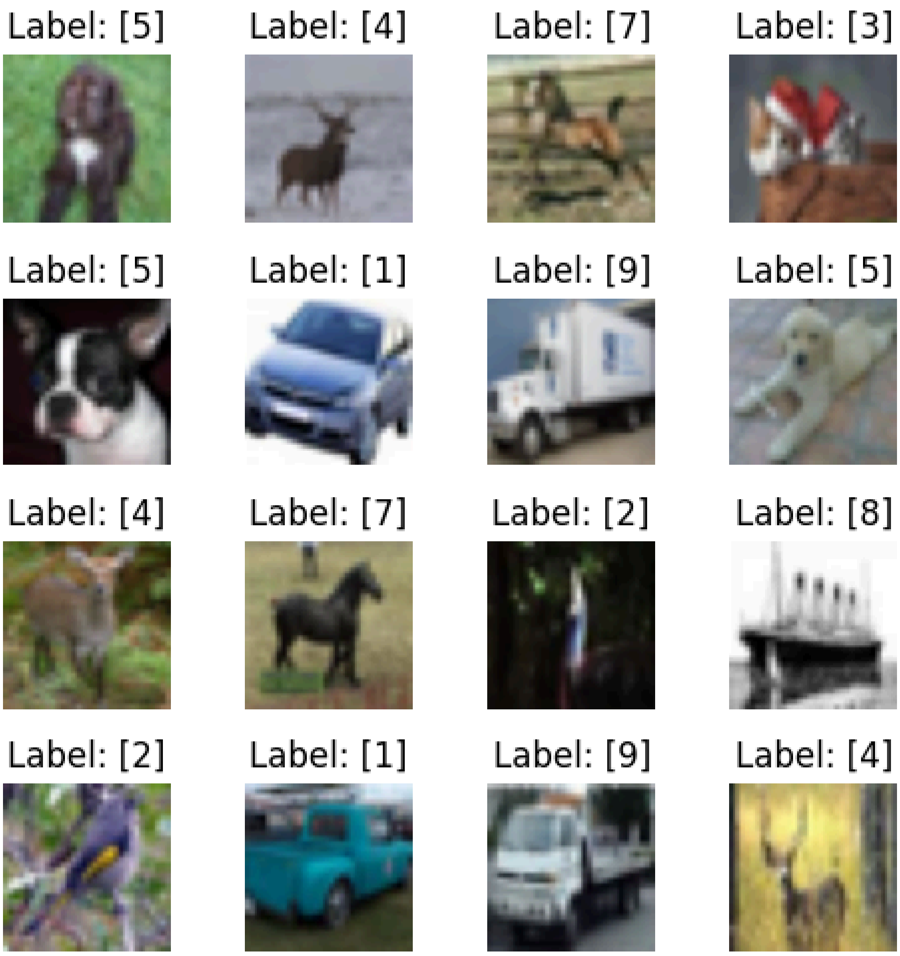
In [7]:
```
# Create a figure and subplots
fig, axes = plt.subplots(4, 4, figsize=(5, 5))

# Flatten the axes array for easy iteration
axes = axes.flatten()

for i, ax in enumerate(axes):
    image = x_train[indices[i]]
    label = y_train[indices[i]]
```

```
    ax.imshow(image)
    ax.set_title(f"Label: {label}")
    ax.axis('off')

plt.tight_layout()
plt.show()
```



In [8]:

```
# Reshape data for CNN input
#CNNs expect input data to have a specific shape, typically (num_samples, img_ro
img_rows, img_cols = 32, 32
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)#shape[0] = hc
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 3)
```

In [9]:

```
x_train.shape
#x_test.shape
#1 stands for 1 channel , if color images we will have 3 channels (RGB)
```

Out[9]: (50000, 32, 32, 3)

In [10]:

```
# Normalize pixel values to be between 0 and 1
x_train = x_train / 255.0 # To normalize the pixel values, we divide by 255 (the
x_test = x_test / 255.0
```

In [11]:

```
# Convert class vectors to binary class matrices

num_classes = 10 # Number of classes in the CIFAR-10 dataset
y_train = to_categorical(y_train, num_classes) #1 hotencoding the labels
y_test = to_categorical(y_test, num_classes)
```

In [12]:
```
print(y_test)
```

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

In [13]:
```
# Create a CNN model
model = Sequential()# 32 kernels , general we choose odd dimension for kernal si
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',strides=(1,1),padding
model.add(MaxPooling2D(pool_size=(2, 2)))#2 x 2 blank kernal
model.add(Flatten()) # only 1 convolution layer and 1 pooling layer, so we need
model.add(Dense(128, activation='relu'))#128 neurons or any no of neurons THIS I
model.add(Dense(num_classes, activation='softmax'))#according to the no of neuro
```

```
c:\Users\user\Desktop\MCA AI DS 2027\S2\Deep Learning\Lab\tfenv\lib\site-packages
\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do not pass an `in
put_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)
```

In [14]:
```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adamw', metrics=['accu
#this is a classification Problem so we use categorical_crossentropy as the loss
# 2 class classfiction , them binary_crossentropy use karte hai
# advanced optimizers : RMSprop, Adam, Adagrad, Adadelta
#accuracy to evaluate the model , trained well or not
```

In [15]:
```
# Train the model
#model.fit() standard attributes
# history variable will store the training history, which includes the loss and
# Trying to train model with x_train and y_train  , batch size of 128 each , tot
#epoch = can be decided by us ,
#get trained batch wise , accuracy is the trainign set accurayc , then the test
history=model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=1, valida
```

```
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 19s 44ms/step - accuracy: 0.4556 - loss: 1.5351 - va
l_accuracy: 0.5266 - val_loss: 1.3118
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 16s 42ms/step - accuracy: 0.5694 - loss: 1.2254 - va
l_accuracy: 0.5655 - val_loss: 1.2207
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 19s 48ms/step - accuracy: 0.6120 - loss: 1.1096 - va
l_accuracy: 0.5933 - val_loss: 1.1809
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 18s 45ms/step - accuracy: 0.6473 - loss: 1.0159 - va
l_accuracy: 0.6184 - val_loss: 1.0904
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 18s 45ms/step - accuracy: 0.6699 - loss: 0.9462 - va
l_accuracy: 0.6213 - val_loss: 1.0960
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 17s 44ms/step - accuracy: 0.6894 - loss: 0.8982 - va
l_accuracy: 0.6372 - val_loss: 1.0390
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 17s 44ms/step - accuracy: 0.7074 - loss: 0.8405 - va
l_accuracy: 0.6388 - val_loss: 1.0398
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 17s 43ms/step - accuracy: 0.7209 - loss: 0.8000 - va
l_accuracy: 0.6546 - val_loss: 0.9978
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 17s 44ms/step - accuracy: 0.7425 - loss: 0.7447 - va
l_accuracy: 0.6476 - val_loss: 1.0336
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 18s 45ms/step - accuracy: 0.7554 - loss: 0.7076 - va
l_accuracy: 0.6532 - val_loss: 1.0057
```

In [16]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1,048,704
dense_1 (Dense)	(None, 10)	1,290

Total params: 3,152,672 (12.03 MB)

Trainable params: 1,050,890 (4.01 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 2,101,782 (8.02 MB)

In [17]:

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6532 - loss: 1.0057
Test accuracy: 0.6531999707221985
```

In [18]:
```python
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

plt.savefig('./foo.png')
plt.show()
```