Importing important libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, MaxPooling2D, GlobalAveragePooling2D, Add, Flatten, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
```

Load the CIFAR-10 dataset and spliting of train and test sets

In [2]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

First 25 images of the training dataset

In [3]:
```
fig, ax = plt.subplots(5, 5)
k = 0
for i in range(5):
    for j in range(5):
        ax[i][j].imshow(x_train[k], aspect='auto')
        k += 1
plt.show()
```

Normalization of pixel values to the range of 0-1

In [4]:
```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

Convert class labels to one-hot encoded vectors

In [5]:
```
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Custom CNN architecture with skip connections

In [6]:
```
def custom_cnn(input_shape, num_classes):
    # Input layer
    inputs = Input(shape=input_shape)

    # Convolutional layers with batch normalization and max pooling

    #Layer1
    conv1 = Conv2D(32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(inputs)
    conv1 = BatchNormalization()(conv1)
    conv1 = Conv2D(32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(conv1)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv1)

    #Layer2
    conv2 = Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(pool1)
    conv2 = BatchNormalization()(conv2)
    conv2 = Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(conv2)
    conv2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv2)

    #Layer3
    conv3 = Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(pool2)
    conv3 = BatchNormalization()(conv3)
    conv3 = Conv2D(128, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(conv3)
    conv3 = BatchNormalization()(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv3)

    #Layer4
    conv4 = Conv2D(256, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(pool3)
    conv4 = BatchNormalization()(conv4)
    conv4 = Conv2D(256, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(conv4)
    conv4 = BatchNormalization()(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv4)

    # Skip Connections
    skip_connection1 = Conv2D(64, kernel_size=(1, 1), strides=(1, 1), padding='same')(pool1)
    skip_connection1 = Add()([skip_connection1, conv2])

    skip_connection2 = Conv2D(256, kernel_size=(1, 1), strides=(1, 1), padding='same')(pool3)
    skip_connection2 = Add()([skip_connection2, conv4])

    # Global average pooling
    global_pool = GlobalAveragePooling2D()(pool4)

    # Flatten the output from the final_pool layer of Global_Average_Pooling
    flatten = Flatten()(global_pool)

    # Fully connected layers
    fc1 = Dense(256, activation='relu')(flatten)
    fc2 = Dense(128, activation='relu')(fc1)
    fc3 = Dense(num_classes, activation='softmax')(fc2)
    outputs=fc3

    # Model creation
    model = Model(inputs=inputs, outputs=outputs)

    return model
```

Define the input shape and number of classes

In [7]:
```
input_shape = x_train.shape[1:]
num_classes = 10
```

In [8]:
```
x_train.shape[1:]
```

Out[8]:
```
(32, 32, 3)
```

Create an object/instance of the custom CNN model

In [9]:
```
model = custom_cnn(input_shape, num_classes)
```

```
Metal device set to: Apple M1
```

Compilation of the model

In [10]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Training the model

In [11]:
```
model.fit(x_train, y_train, batch_size=128, epochs=10, validation_data=(x_test, y_test))
```

```
Epoch 1/10
2023-06-30 00:49:04.673616: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
391/391 [==============================] - 25s 63ms/step - loss: 1.1361 - accuracy: 0.5927 - val_loss: 2.1144 - val_accuracy: 0.3447
Epoch 2/10
391/391 [==============================] - 24s 62ms/step - loss: 0.7084 - accuracy: 0.7524 - val_loss: 0.8935 - val_accuracy: 0.7007
Epoch 3/10
391/391 [==============================] - 25s 63ms/step - loss: 0.5435 - accuracy: 0.8109 - val_loss: 0.7485 - val_accuracy: 0.7587
Epoch 4/10
391/391 [==============================] - 24s 62ms/step - loss: 0.4212 - accuracy: 0.8532 - val_loss: 0.6676 - val_accuracy: 0.7785
Epoch 5/10
391/391 [==============================] - 25s 64ms/step - loss: 0.3333 - accuracy: 0.8839 - val_loss: 0.6233 - val_accuracy: 0.7976
Epoch 6/10
391/391 [==============================] - 24s 62ms/step - loss: 0.2541 - accuracy: 0.9107 - val_loss: 0.7514 - val_accuracy: 0.7869
Epoch 7/10
391/391 [==============================] - 25s 63ms/step - loss: 0.2016 - accuracy: 0.9293 - val_loss: 0.7756 - val_accuracy: 0.7805
Epoch 8/10
391/391 [==============================] - 25s 63ms/step - loss: 0.1549 - accuracy: 0.9457 - val_loss: 0.9381 - val_accuracy: 0.7676
Epoch 9/10
391/391 [==============================] - 25s 63ms/step - loss: 0.1322 - accuracy: 0.9529 - val_loss: 0.7304 - val_accuracy: 0.8098
Epoch 10/10
391/391 [==============================] - 29s 74ms/step - loss: 0.1058 - accuracy: 0.9631 - val_loss: 0.7557 - val_accuracy: 0.8105
```

Out[11]:
```
<keras.callbacks.History at 0x169e9ca30>
```

Evaluate the model on the test set

In [12]:
```
_, accuracy = model.evaluate(x_test, y_test)
print("Test accuracy:", accuracy)
```

```
313/313 [==============================] - 3s 11ms/step - loss: 0.7557 - accuracy: 0.8105
Test accuracy: 0.8105000257492065
```

In [13]:
```
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Test Accuracy: 81.05%
```

Model Summary

In [14]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_6 (Conv2D)           (None, 4, 4, 256)         295168

 batch_normalization_6 (Batc  (None, 4, 4, 256)        1024
 hNormalization)

 conv2d_7 (Conv2D)           (None, 4, 4, 256)         590080

 batch_normalization_7 (Batc  (None, 4, 4, 256)        1024
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 global_average_pooling2d (G  (None, 256)              0
 lobalAveragePooling2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 256)               65792

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 1,276,074
Trainable params: 1,274,154
```

```
Non-trainable params: 1,920
_________________________________________________________________
```

In [ ]: