Experiment No 8

Aim: To define the CNN to predict the knowledge from image classification Visualizing the learned CNN Model.

Theory:

Convolutional Neural Networks (CNNs) are a class of deep learning models primarily designed for processing and analyzing grid-like data, with a strong emphasis on their application in computer vision tasks. CNNs have proven to be exceptionally effective in image classification, object detection, and various other visual recognition tasks.

Key components include:

- **Convolutional Layers**: CNNs use convolutional layers to process input data, where a filter slides over the data to detect local patterns. The output is called a feature map.
- **Pooling Layers**: Pooling reduces the size of feature maps while retaining essential information, commonly using max or average pooling. This helps reduce computation and prevents overfitting.
- **Activation Functions**: Non-linear functions like ReLU introduce non-linearity, making the model more powerful and effective.
- **CNN Architecture:** CNNs consist of multiple convolutional, activation, and pooling layers, followed by fully connected layers. Early layers capture simple features, while deeper layers capture complex patterns.
- **Weight Sharing:** Filters are shared across the input space, allowing the network to detect the same features in different parts of the input, enhancing efficiency.
- **Striding**: Filters can move with a stride, reducing the spatial dimensions of the output feature maps.
- **Padding:** Adding pixels to the input before convolution preserves feature map dimensions, important for deeper layers.
- **Transfer Learning:** Pre-trained CNNs on large datasets (e.g., ImageNet) are fine-tuned for specific tasks, improving performance on smaller datasets.
- **Applications:** CNNs excel in image classification, object detection, face recognition, medical image analysis, and even extend to NLP and speech recognition.
- **Training:** CNNs are trained using stochastic gradient descent (SGD) with backpropagation to adjust weights based on the loss function.

Conclusion:

A simple CNN was built to classify images as either horizontal or vertical. The trained model learned to distinguish between the two orientations, achieving accuracy on both the training and validation datasets. The model's architecture and learned features can be visualized to better understand its performance.

CNN on CIFAR-10 Dataset

Import necessary libraries

In [12]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Load and preprocess the CIFAR-10 dataset#

In [2]:
```
# Load the CIFAR-10 dataset
(X_train, Y_train), (X_test, Y_test) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
X_train, X_test = X_train / 255.0, X_test / 255.0

# Print the shapes of the dataset
print(f'Training data shape: {X_train.shape}, Training labels shape: {Y_train.shape}')
print(f'Test data shape: {X_test.shape}, Test labels shape: {Y_test.shape}')
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 676s 4us/step
Training data shape: (50000, 32, 32, 3), Training labels shape: (50000, 1)
Test data shape: (10000, 32, 32, 3), Test labels shape: (10000, 1)
```

In [6]:
```
# Build a CNN model
model = models.Sequential([
    # First convolutional layer with 32 filters, 3x3 kernel, and ReLU activation
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),  # Max pooling layer
    # Second convolutional layer
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),  # Max pooling layer
    # Third convolutional layer
    layers.Conv2D(64, (3, 3), activation='relu'),

    # Flatten the feature maps and connect to a dense layer
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # Output layer for 10 classes
])
```

In [7]:
```
# Print the model summary
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 64)	36,928
flatten_1 (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 64)	65,600
dense_3 (Dense)	(None, 10)	650

Total params: 122,570 (478.79 KB)

Trainable params: 122,570 (478.79 KB)

Non-trainable params: 0 (0.00 B)

Compile the model

In [8]:
```
# Compile the model with an optimizer, loss function, and performance metrics
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Train the model

In [9]:
```
# Train the model on the training data
history = model.fit(X_train, Y_train, epochs=10,
                    validation_data=(X_test, Y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 32s 16ms/step - accuracy: 0.3526 - loss: 1.7425 - val_accuracy: 0.5635 -
val_loss: 1.2154
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5880 - loss: 1.1658 - val_accuracy: 0.6297 -
val_loss: 1.0609
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.6531 - loss: 0.9846 - val_accuracy: 0.6669 -
val_loss: 0.9618
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 22s 14ms/step - accuracy: 0.6973 - loss: 0.8634 - val_accuracy: 0.6950 -
val_loss: 0.8949
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.7207 - loss: 0.7974 - val_accuracy: 0.7071 -
val_loss: 0.8498
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 22s 14ms/step - accuracy: 0.7442 - loss: 0.7277 - val_accuracy: 0.7073 -
val_loss: 0.8507
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 22s 14ms/step - accuracy: 0.7649 - loss: 0.6692 - val_accuracy: 0.7175 -
val_loss: 0.8253
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 22s 14ms/step - accuracy: 0.7807 - loss: 0.6251 - val_accuracy: 0.7183 -
val_loss: 0.8316
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 22s 14ms/step - accuracy: 0.7935 - loss: 0.5827 - val_accuracy: 0.7210 -
val_loss: 0.8423
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.8094 - loss: 0.5402 - val_accuracy: 0.7158 -
val_loss: 0.8467
```

Evaluate the model

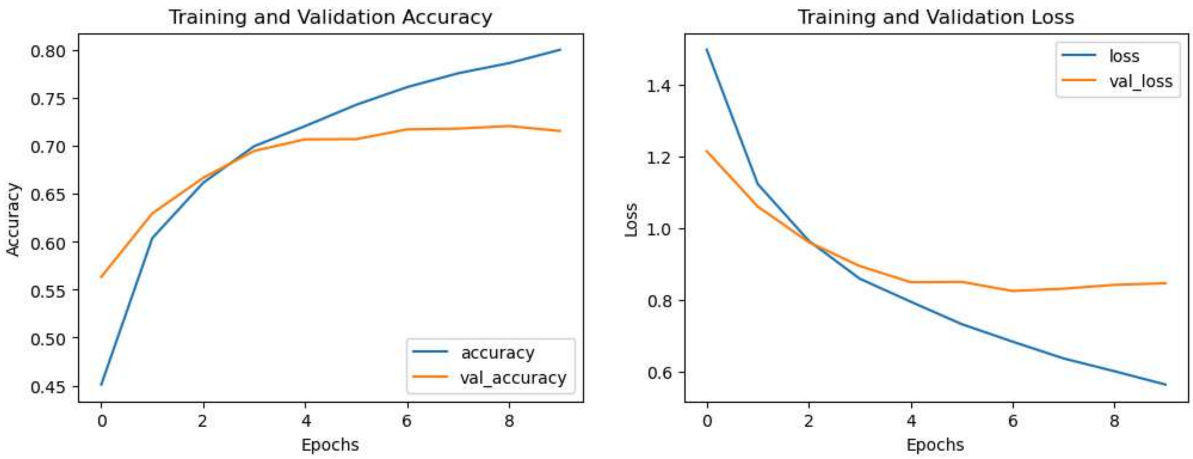
In [39]:
```
# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(X_test, Y_test, verbose=2)
print(f'Test accuracy: {test_acc*100}')
```

```
313/313 - 3s - 9ms/step - accuracy: 0.7158 - loss: 0.8467
Test accuracy: 71.57999873161316
```

Visualize training results

In [13]:

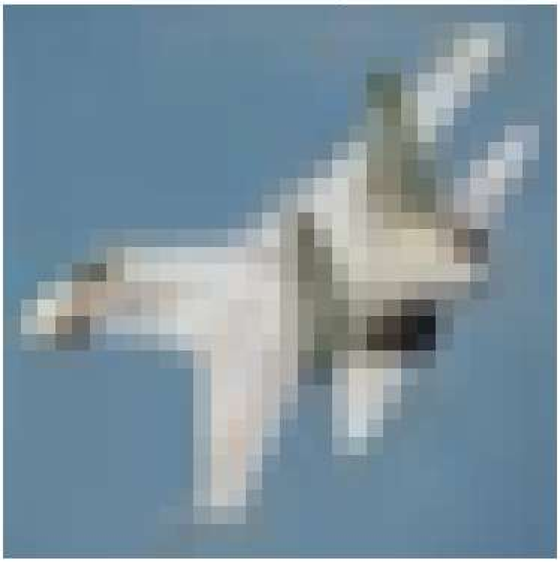
```
# Plot the accuracy and loss curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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

plt.show()
```



Sample Classification

In [32]:
```
import numpy as np
import matplotlib.pyplot as plt

#dictionary to map label indices to class labels
class_names = {0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat',
               4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse',
               8: 'ship', 9: 'truck'}

# Select a specific example from the test dataset (e.g., the 10th image)
example_index = 10  # You can change this index to any number between 0 and 9999

# Extract the selected test image and its true label
example_image = X_test[example_index]
true_label_index = Y_test[example_index][0]  # Flatten the label index
true_label_name = class_names[true_label_index]  # Get the class name

# Display the selected image clearly
plt.imshow(example_image)
plt.title(f'True Label: {true_label_name}')
plt.axis('off')  # Turn off the axis to make the image display cleaner
plt.show()
```

True Label: airplane



In [33]:
```
# Reshape the image to match the input shape expected by the model
example_image_reshaped = np.expand_dims(example_image, axis=0)  # Add a batch dimension

# Use the trained model to predict the class
predicted_probs = model.predict(example_image_reshaped)

# Get the predicted class by finding the index with the highest probability
predicted_label_index = np.argmax(predicted_probs)
predicted_label_name = class_names[predicted_label_index]

print(f'Predicted Label: {predicted_label_name}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
Predicted Label: airplane
```

In [34]:
```
# Compare the true label with the predicted label
if predicted_label_index == true_label_index:
    print(f"Correct! The model predicted {predicted_label_name} and the true label is {true_label_name}."
else:
    print(f"Incorrect! The model predicted {predicted_label_name} but the true label is {true_label_name}
```

```
Correct! The model predicted airplane and the true label is airplane.
```