In [3]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
```

Pre processing img data

In [4]:
```
train_dir = "cifar-10-img/cifar-10-img/train"
test_dir = "cifar-10-img/cifar-10-img/test"

train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
)

test_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
)

# here batch_size is the number of images in each batch
train_batch_size = 5000
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(32, 32),
    batch_size=train_batch_size,
    class_mode='categorical'
)
test_batch_size = 1000
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(32, 32),
    batch_size=test_batch_size,
    class_mode='categorical'
)
```

```
Found 40079 images belonging to 10 classes.
Found 9921 images belonging to 10 classes.
```

Selecting only first batch with 5000 images as train and test data

In [5]:
```
x_train, y_train =  train_generator[0]
x_test, y_test = test_generator[0]

print(len(x_train))
print(len(x_test))
```

```
5000
1000
```

a. Load in a pre-trained CNN model trained on a large dataset

In [6]:
```
# Load VGG16 without top layers
weights_path = "vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
base_model = VGG16(weights=weights_path, include_top=False, input_shape=(32, 32, 3)
```

b. Freeze parameters (weights) in model's lower convolutional layers

In [7]:
```
for layer in base_model.layers:
   layer.trainable = False
```

c. Add custom classifier with several layers of trainable parameters to model

In [8]:
```
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
predictions = Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy
```

d. Train classifier layers on training data available for task

In [9]:
```
# Train the model
model.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_test, y_te
```

```
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 65s 767ms/step - accuracy: 0.2858 - loss: 1.9748 - val_ac
curacy: 0.3860 - val_loss: 1.6505
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 60s 761ms/step - accuracy: 0.4368 - loss: 1.5820 - val_ac
curacy: 0.4820 - val_loss: 1.4761
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 59s 746ms/step - accuracy: 0.4848 - loss: 1.4490 - val_ac
curacy: 0.5050 - val_loss: 1.4197
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 59s 742ms/step - accuracy: 0.5258 - loss: 1.3451 - val_ac
curacy: 0.5160 - val_loss: 1.3857
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 60s 761ms/step - accuracy: 0.5300 - loss: 1.3036 - val_ac
curacy: 0.5260 - val_loss: 1.3878
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 60s 764ms/step - accuracy: 0.5596 - loss: 1.2402 - val_ac
curacy: 0.5340 - val_loss: 1.3614
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 81s 751ms/step - accuracy: 0.5696 - loss: 1.1968 - val_ac
curacy: 0.5390 - val_loss: 1.3443
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 60s 756ms/step - accuracy: 0.5964 - loss: 1.1397 - val_ac
curacy: 0.5430 - val_loss: 1.3512
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 59s 750ms/step - accuracy: 0.6054 - loss: 1.1018 - val_ac
curacy: 0.5360 - val_loss: 1.3331
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 59s 743ms/step - accuracy: 0.6202 - loss: 1.0582 - val_ac
curacy: 0.5320 - val_loss: 1.3575
```

Out[9]: <keras.src.callbacks.history.History at 0x21ddcdc1160>

e. Fine-tune hyper parameters and unfreeze more layers as needed

In [10]:
```python
base_model = VGG16(weights=weights_path, include_top=False, input_shape=(32, 32, 3)
# freeze all layers first
for layer in base_model.layers:
   layer.trainable = False
# unfreeze last 4 layers of base model
for layer in base_model.layers[len(base_model.layers) - 4:]:
   layer.trainable = True
# fine-tuning hyper parameters
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = Dense(512, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
predictions = Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
```

```
# training fine tuned model
model.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_test, y_te
```

```
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 105s 1s/step - accuracy: 0.1576 - loss: 2.2179 - val_accu
racy: 0.1740 - val_loss: 2.0656
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 138s 1s/step - accuracy: 0.2368 - loss: 1.9183 - val_accu
racy: 0.3770 - val_loss: 1.5908
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 153s 1s/step - accuracy: 0.4748 - loss: 1.4207 - val_accu
racy: 0.4490 - val_loss: 1.6163
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 94s 1s/step - accuracy: 0.5984 - loss: 1.1576 - val_accur
acy: 0.5770 - val_loss: 1.2232
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 143s 1s/step - accuracy: 0.6732 - loss: 0.9533 - val_accu
racy: 0.5870 - val_loss: 1.2751
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 140s 1s/step - accuracy: 0.7122 - loss: 0.8254 - val_accu
racy: 0.6100 - val_loss: 1.2426
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 148s 1s/step - accuracy: 0.7548 - loss: 0.7015 - val_accu
racy: 0.5850 - val_loss: 1.3624
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 138s 1s/step - accuracy: 0.7888 - loss: 0.6338 - val_accu
racy: 0.6230 - val_loss: 1.3001
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 84s 1s/step - accuracy: 0.8150 - loss: 0.5188 - val_accur
acy: 0.6020 - val_loss: 1.3733
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.8332 - loss: 0.5054 - val_accur
acy: 0.5870 - val_loss: 1.8079
```

Out[10]: <keras.src.callbacks.history.History at 0x21de57b2350>

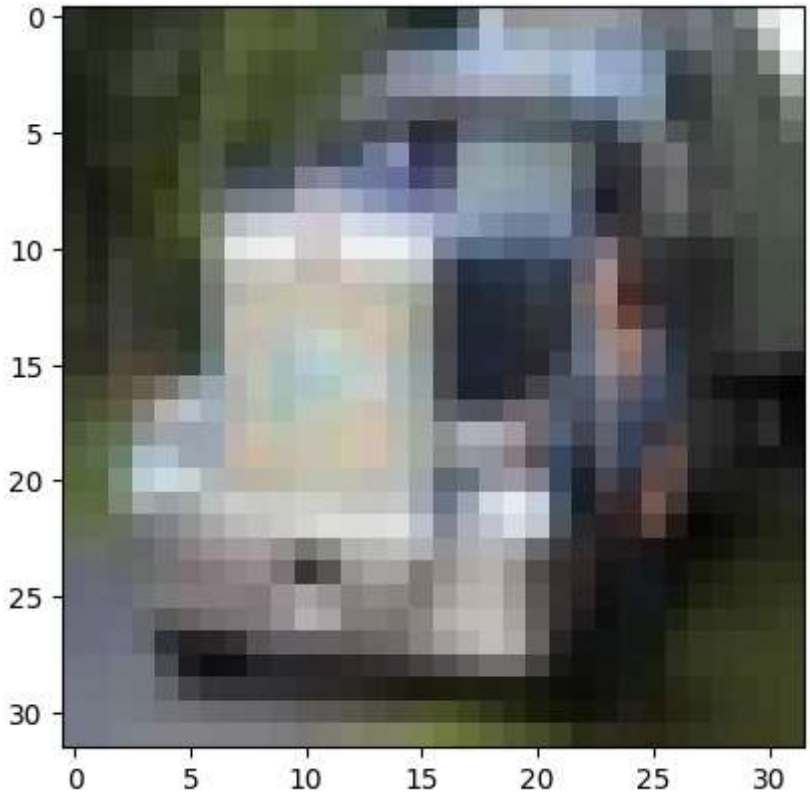
In [11]:
```
import matplotlib.pyplot as plt
predicted_value = model.predict(x_test)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 302ms/step
```

In [12]:
```
labels = list(test_generator.class_indices.keys())
```

In [13]:
```
n = 890
plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
```

```
Preditcted:  truck
Actual:  truck
```



In [ ]: