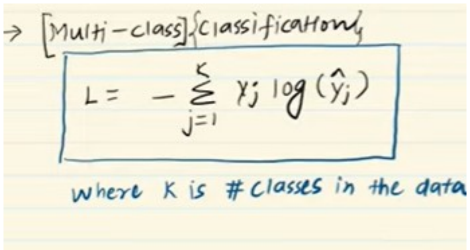
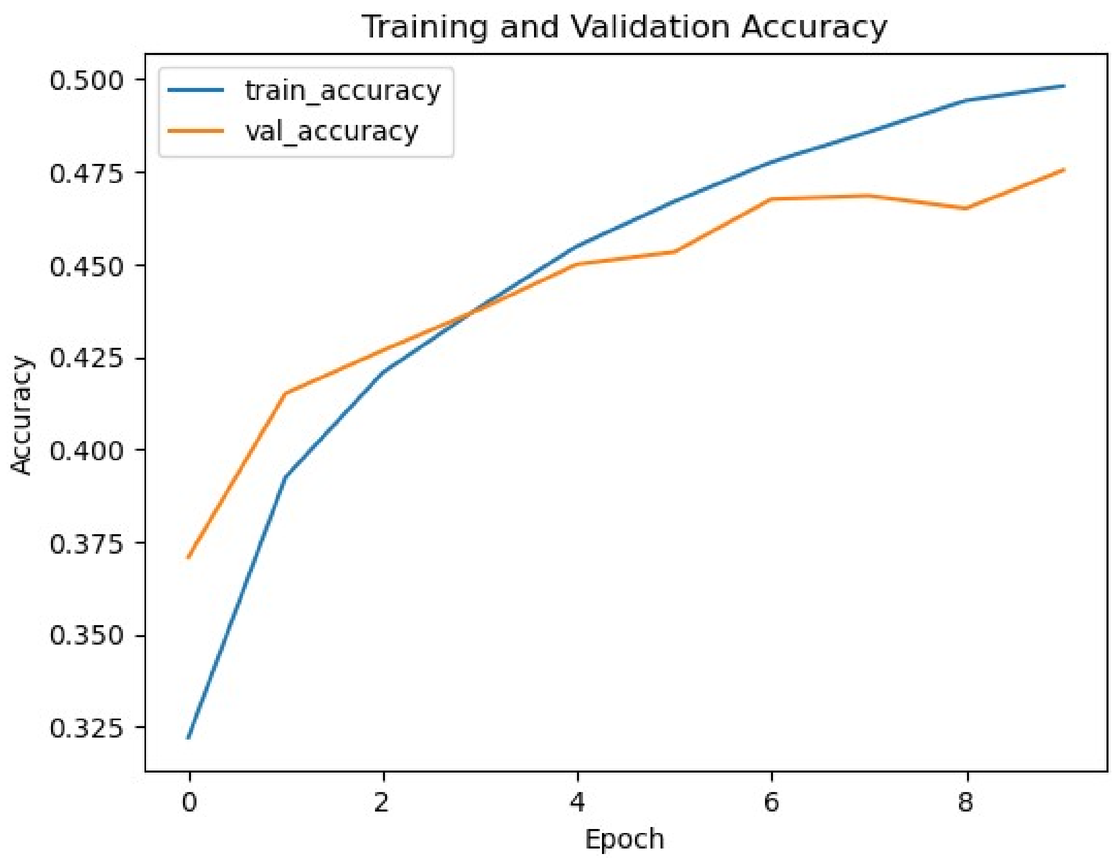
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the images to a range of 0 to 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Convert class vectors to binary class matrices (one-hot encoding)
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build the MLP model
model = Sequential()
model.add(Flatten(input_shape=(32, 32, 3)))  # Flatten the input
model.add(Dense(512, activation='relu'))      # First hidden layer
model.add(Dense(256, activation='relu'))      # Second hidden layer
model.add(Dense(10, activation='softmax'))    # Output layer

C:\Users\Nasreen\anaconda3\Lib\site-packages\keras\src\layers\
reshaping\flatten.py:37: UserWarning: Do not pass an
`input_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__(**kwargs)

model.summary()

Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ flatten_3 (Flatten)                  │ (None, 3072)                │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_10 (Dense)                     │ (None, 512)                 │
1,573,376 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dense_11 (Dense)                     │ (None, 256)                 │
131,328 │
├──────────────────────────────────────┼─────────────────────────────┤
```

```
───────────────────────┤
│ dense_12 (Dense)                            │ (None, 10)                             │
2,570 │
└─────────────────────────────────────────────┴────────────────────────────────────────┴
───────────────────────┘

 Total params: 1,707,274 (6.51 MB)

 Trainable params: 1,707,274 (6.51 MB)

 Non-trainable params: 0 (0.00 B)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history=model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)

Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 43s 31ms/step - accuracy: 0.2744 -
loss: 2.0646 - val_accuracy: 0.3708 - val_loss: 1.7652
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 37s 30ms/step - accuracy: 0.3909 -
loss: 1.7071 - val_accuracy: 0.4151 - val_loss: 1.6501
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 30ms/step - accuracy: 0.4163 -
loss: 1.6210 - val_accuracy: 0.4267 - val_loss: 1.6177
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 38s 30ms/step - accuracy: 0.4338 -
loss: 1.5774 - val_accuracy: 0.4378 - val_loss: 1.5954
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 37s 30ms/step - accuracy: 0.4525 -
loss: 1.5229 - val_accuracy: 0.4500 - val_loss: 1.5634
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 37s 30ms/step - accuracy: 0.4722 -
loss: 1.4776 - val_accuracy: 0.4533 - val_loss: 1.5585
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 29ms/step - accuracy: 0.4780 -
loss: 1.4534 - val_accuracy: 0.4676 - val_loss: 1.5214
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 41s 30ms/step - accuracy: 0.4875 -
loss: 1.4335 - val_accuracy: 0.4685 - val_loss: 1.5086
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 39s 31ms/step - accuracy: 0.4979 -
loss: 1.4162 - val_accuracy: 0.4651 - val_loss: 1.5241
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 36s 28ms/step - accuracy: 0.5005 -
loss: 1.3886 - val_accuracy: 0.4754 - val_loss: 1.4988
```

Categorical Cross Entropy is also known as Softmax Loss. It's a softmax activation plus a Cross-Entropy loss used for multiclass classification. Using this loss, we can train a Convolutional Neural Network to output a probability over the N classes for each image.




```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4854 - loss:
1.4673
Test accuracy: 0.4773

# Plot training & validation accuracy
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Training and Validation Accuracy
train_accuracy
val_accuracy
Accuracy
Epoch
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0
2
4
6
8