Batch: HO-DL-1

Roll Number: 16010421073 **Experiment Number:5**

Name: Keyur Patel

Title of the Experiment: Convolutional Neural Network

Program:

Import requisite libraries using Tensorflow and Keras.

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

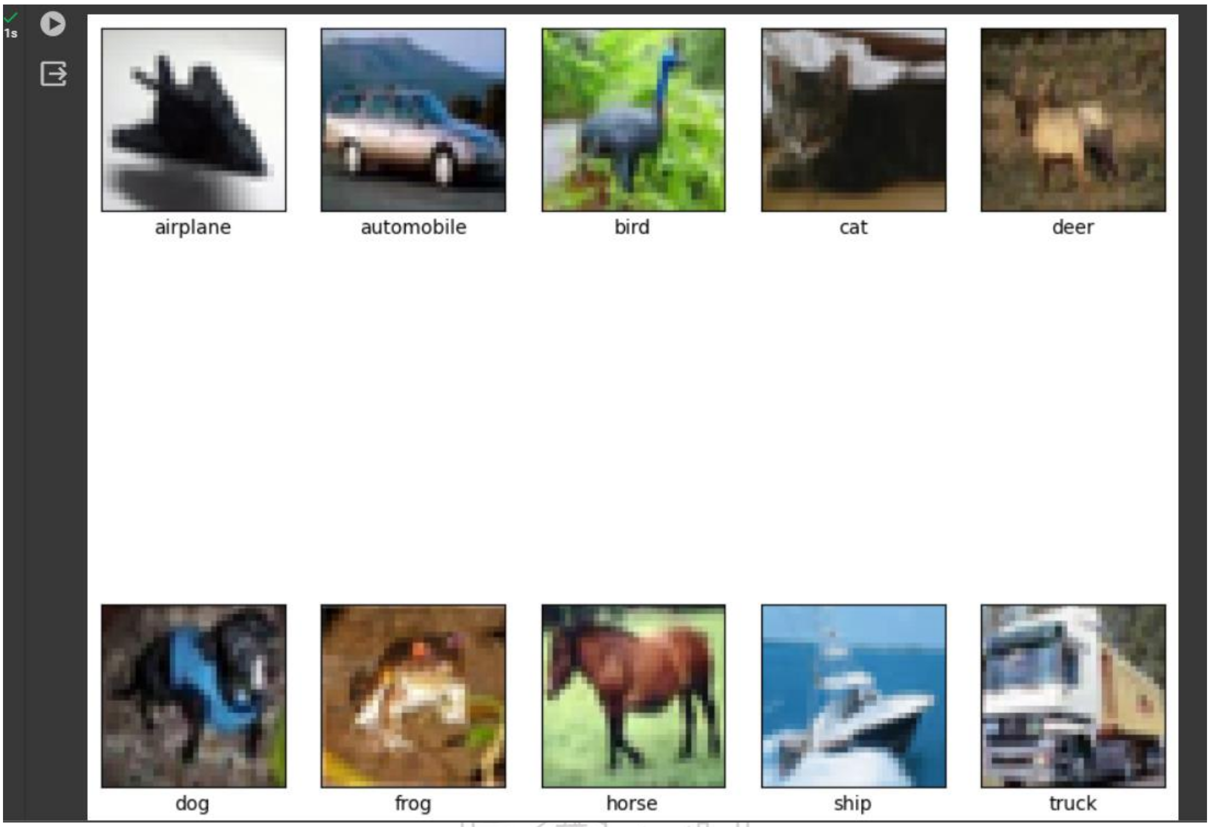
Load the selected dataset.

```
(X_train, y_train), (X_test, y_test) =
keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 11s 0us/step
```

Visualize and display random images belonging to each class.

```
class_names = ["airplane", "automobile", "bird", "cat", "deer","dog",
"frog", "horse", "ship", "truck"]
plt.figure(figsize=(10,10))
for i in range(10):
  idx = np.where(y_train == i)[0][0]
  plt.subplot(2, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(X_train[idx])
  plt.xlabel(class_names[i])
plt.show()
```



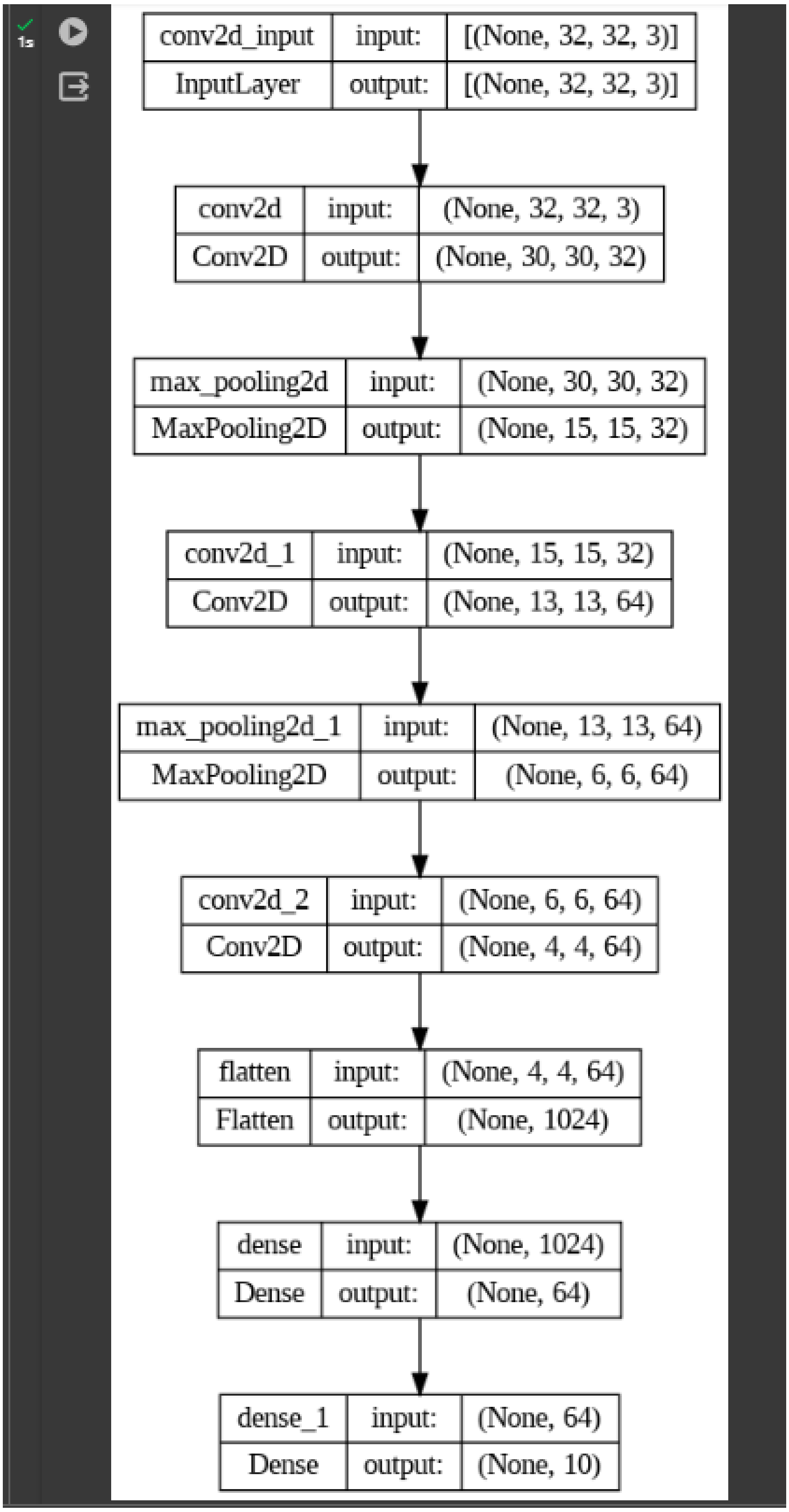
Develop the CNN model.

```
model = keras.Sequential([keras.layers.Conv2D(32, (3, 3),
activation='relu', input_shape=(32,32, 3)),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3), activation='relu'),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3), activation='relu'),
keras.layers.Flatten(),
  keras.layers.Dense(64, activation='relu'), keras.layers.Dense(10)
])
```

Print Model Summary and display architecture diagram.

```
model.summary()
keras.utils.plot_model(model, show_shapes=True)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
```



Compile and fit the model on train dataset.

```
model.compile(optimizer='adam',
```

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1250/1250 [==============================] - 52s 41ms/step - loss: 2.0608 - accuracy: 0.3215 - val_loss: 1.5217 - val_accuracy: 0.4412
Epoch 2/10
1250/1250 [==============================] - 51s 41ms/step - loss: 1.4520 - accuracy: 0.4758 - val_loss: 1.3459 - val_accuracy: 0.5205
Epoch 3/10
1250/1250 [==============================] - 48s 38ms/step - loss: 1.2835 - accuracy: 0.5467 - val_loss: 1.2598 - val_accuracy: 0.5471
Epoch 4/10
1250/1250 [==============================] - 49s 39ms/step - loss: 1.1765 - accuracy: 0.5842 - val_loss: 1.1601 - val_accuracy: 0.5940
Epoch 5/10
1250/1250 [==============================] - 47s 38ms/step - loss: 1.0680 - accuracy: 0.6252 - val_loss: 1.1579 - val_accuracy: 0.5929
Epoch 6/10
1250/1250 [==============================] - 49s 39ms/step - loss: 0.9982 - accuracy: 0.6507 - val_loss: 1.1673 - val_accuracy: 0.5971
Epoch 7/10
1250/1250 [==============================] - 48s 39ms/step - loss: 0.9269 - accuracy: 0.6766 - val_loss: 1.0661 - val_accuracy: 0.6322
Epoch 8/10
1250/1250 [==============================] - 47s 37ms/step - loss: 0.8545 - accuracy: 0.7027 - val_loss: 1.0463 - val_accuracy: 0.6430
Epoch 9/10
1250/1250 [==============================] - 49s 39ms/step - loss: 0.8006 - accuracy: 0.7215 - val_loss: 1.0995 - val_accuracy: 0.6425
Epoch 10/10
1250/1250 [==============================] - 49s 39ms/step - loss: 0.7459 - accuracy: 0.7380 - val_loss: 1.0908 - val_accuracy: 0.6447
```

Calculate training and the cross-validation accuracy.

```
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]
print("Training accuracy: {:.2f}%".format(train_acc*100))
print("Validation accuracy: {:.2f}%".format(val_acc*100))
```

```
Training accuracy: 73.80%
Validation accuracy: 64.47%
```

Redefine the model by using appropriate regularization technique to prevent overfitting.

```
from tensorflow.keras import regularizers
reg_model = keras.Sequential([keras.layers.Conv2D(32, (3, 3),
activation='relu', input_shape=(32,32, 3),
  kernel_regularizer=regularizers.l2(0.001)),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3), activation='relu',
  kernel_regularizer=regularizers.l2(0.001)),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3),
  activation='relu',
  kernel_regularizer=regularizers.l2(0.001)),
  keras.layers.Flatten(),
  keras.layers.Dense(64,
activation='relu',kernel_regularizer=regularizers.l2(0.001)),
  keras.layers.Dense(10)
])
```

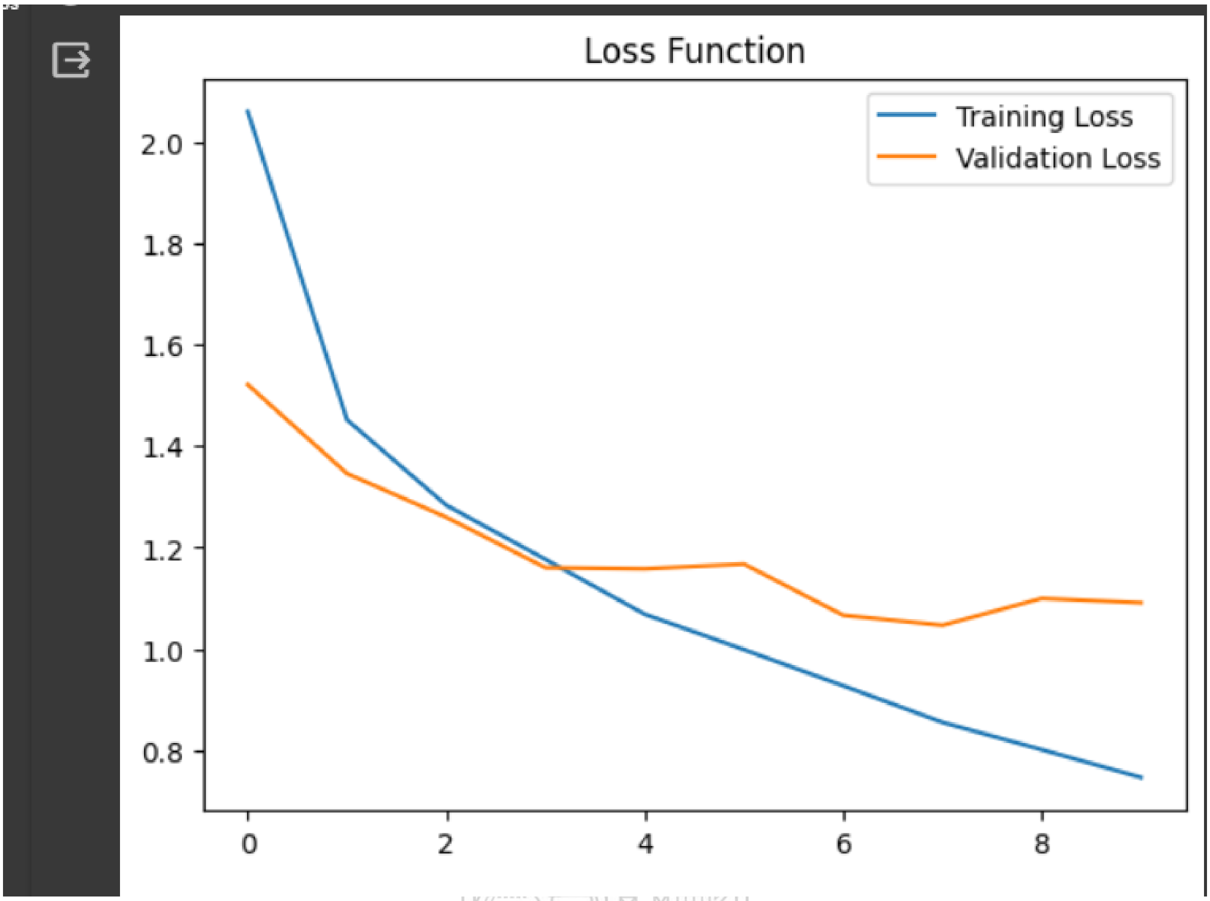
Fit the data on the regularized model.

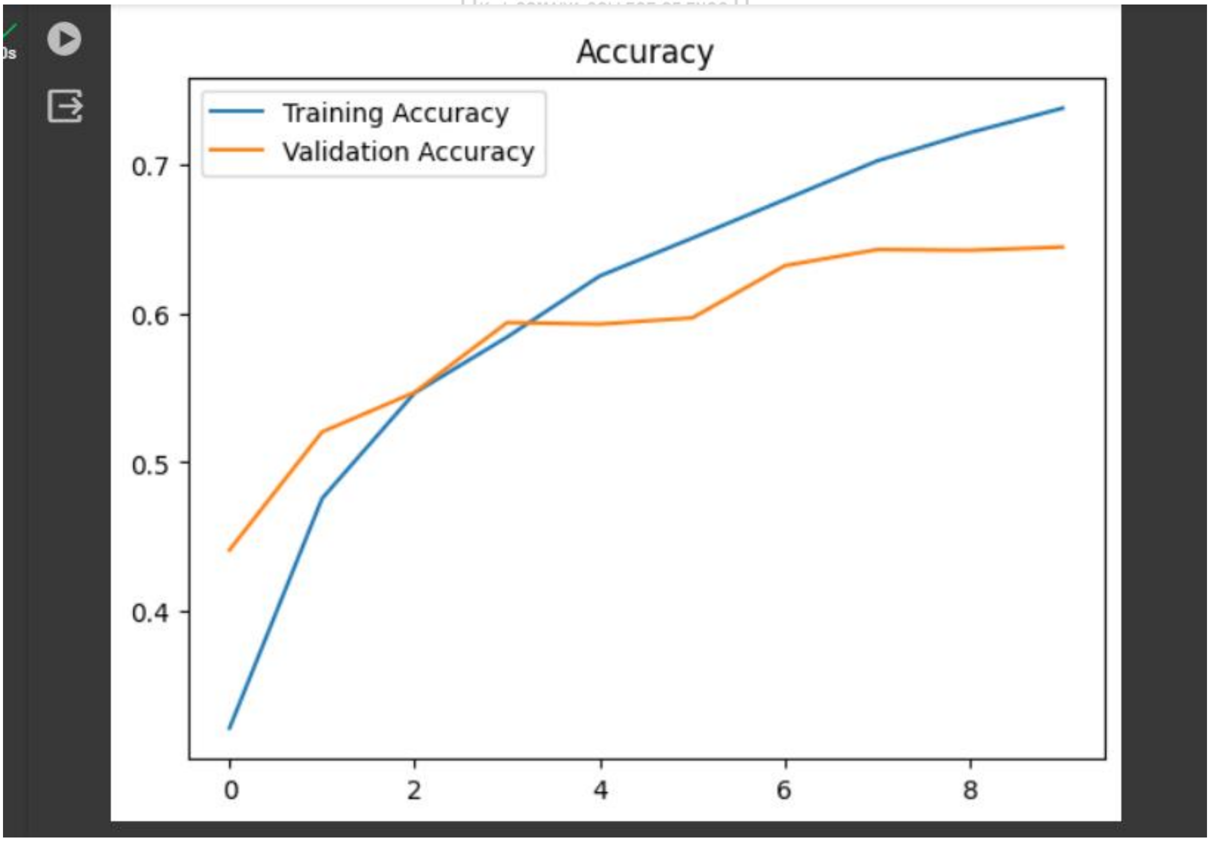
```
reg_model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
reg_history = reg_model.fit(X_train, y_train, epochs=10,
validation_split=0.2)
```

```
Epoch 1/10
1250/1250 [==============================] - 51s 40ms/step - loss: 1.9382 - accuracy: 0.3727 - val_loss: 1.6820 - val_accuracy: 0.4345
Epoch 2/10
1250/1250 [==============================] - 49s 39ms/step - loss: 1.5074 - accuracy: 0.5066 - val_loss: 1.4038 - val_accuracy: 0.5417
Epoch 3/10
1250/1250 [==============================] - 48s 38ms/step - loss: 1.3305 - accuracy: 0.5711 - val_loss: 1.3594 - val_accuracy: 0.5672
Epoch 4/10
1250/1250 [==============================] - 48s 38ms/step - loss: 1.2231 - accuracy: 0.6143 - val_loss: 1.2358 - val_accuracy: 0.6146
Epoch 5/10
1250/1250 [==============================] - 50s 40ms/step - loss: 1.1393 - accuracy: 0.6491 - val_loss: 1.2203 - val_accuracy: 0.6167
Epoch 6/10
1250/1250 [==============================] - 48s 38ms/step - loss: 1.0884 - accuracy: 0.6698 - val_loss: 1.1561 - val_accuracy: 0.6538
Epoch 7/10
1250/1250 [==============================] - 48s 38ms/step - loss: 1.0371 - accuracy: 0.6930 - val_loss: 1.1188 - val_accuracy: 0.6690
Epoch 8/10
1250/1250 [==============================] - 49s 39ms/step - loss: 1.0021 - accuracy: 0.7067 - val_loss: 1.1684 - val_accuracy: 0.6544
Epoch 9/10
1250/1250 [==============================] - 48s 38ms/step - loss: 0.9739 - accuracy: 0.7202 - val_loss: 1.1663 - val_accuracy: 0.6630
Epoch 10/10
1250/1250 [==============================] - 48s 39ms/step - loss: 0.9515 - accuracy: 0.7309 - val_loss: 1.2365 - val_accuracy: 0.6463
```

Calculate and plot loss function and accuracy using suitable loss function.

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss Function')
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Accuracy')
plt.show()
```





Display classification Report for un-regularized CNN model.

```
from sklearn.metrics import classification_report
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
print("Classification Report (Un-regularized):\n",
classification_report(y_test, y_pred_classes,
target_names=class_names))
```

```
313/313 [==============================] - 3s 11ms/step
Classification Report (Un-regularized):
              precision    recall  f1-score   support

    airplane       0.66      0.71      0.69      1000
  automobile       0.78      0.81      0.79      1000
        bird       0.56      0.48      0.52      1000
         cat       0.49      0.46      0.47      1000
        deer       0.57      0.58      0.57      1000
         dog       0.53      0.57      0.55      1000
        frog       0.85      0.58      0.69      1000
       horse       0.60      0.75      0.67      1000
        ship       0.78      0.77      0.77      1000
       truck       0.70      0.77      0.73      1000

    accuracy                           0.65     10000
   macro avg       0.65      0.65      0.65     10000
weighted avg       0.65      0.65      0.65     10000
```

Display classification Report for regularized CNN model.

```
y_pred = reg_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
print("Classification Report (Regularized):\n",
classification_report(y_test, y_pred_classes,
target_names=class_names))
```

```
313/313 [==============================] - 5s 16ms/step
Classification Report (Regularized):
              precision    recall  f1-score   support

    airplane       0.64      0.72      0.68      1000
  automobile       0.75      0.83      0.79      1000
        bird       0.52      0.51      0.51      1000
         cat       0.50      0.39      0.44      1000
        deer       0.58      0.64      0.61      1000
         dog       0.51      0.68      0.58      1000
        frog       0.73      0.77      0.75      1000
       horse       0.70      0.72      0.71      1000
        ship       0.84      0.71      0.77      1000
       truck       0.87      0.52      0.65      1000

    accuracy                           0.65     10000
   macro avg       0.66      0.65      0.65     10000
weighted avg       0.66      0.65      0.65     10000
```

Comment on output.

The CIFAR-10 dataset was used to train a model, and the results indicate that the regularization techniques employed were effective in preventing overfitting. This can be observed through the lower regularization loss and higher accuracy when compared to the non-regularized version of the model.

The classification report reveals that the model's performance varies across different classes, with some classes being classified more accurately than others. This suggests that further improvements could be made, such as acquiring additional training data or refining the model architecture, to enhance the overall classification performance.

CO3 : Assimilate fundamentals of Convolutional Neural Network.

Conclusion: We have successfully implemented a Convolutional Neural Network (CNN) model.