CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK.

N.SHIVRAM (RA2011026010237)
G.MEGHANATH (RA2011026010219)
A.S.THARUN (RA2011026010217)

PROBLEM STATEMENT:

- The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

- The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

- This project aims to classify the given dataset into 10 unique classes through the use of Convolutional neural network

THE NEURAL NETWORK:

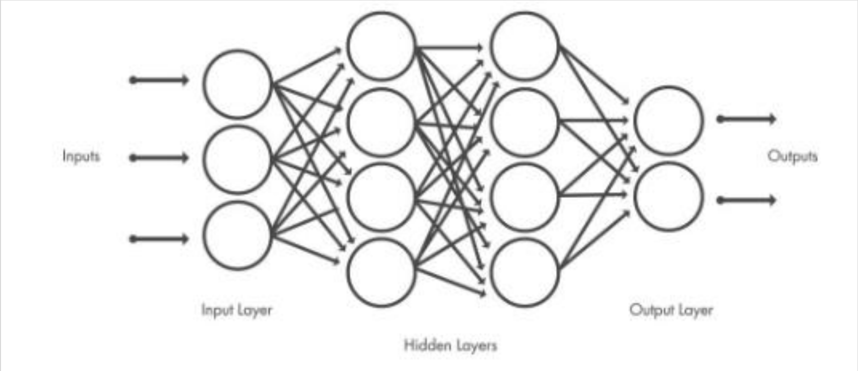
- A Convolutional Neural Network is used in this project to classify the dataset
- A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.
- Since our aim is to classify the dataset into 10 different classes, CNN is the best available neural network for the task.

ADVANTAGES OF CNN:

- CNNs eliminate the need for manual feature extraction—the features are learned directly by the CNN.
- CNNs produce highly accurate recognition results.
- CNNs can be retrained for new recognition tasks, enabling you to build on pre-existing networks.

WORKING OF A CNN:

- A convolutional neural network can have tens or hundreds of layers that each learn to detect different features of an image. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters can start as very simple features, such as brightness and edges, and increase in complexity to features that uniquely define the object.



- Like other neural networks, a CNN is composed of an input layer, an output layer, and many hidden layers in between.

- These layers perform operations that alter the data with the intent of learning features specific to the data. Three of the most common layers are: convolution, activation or ReLU, and pooling.

CONVOULUTION LAYER:

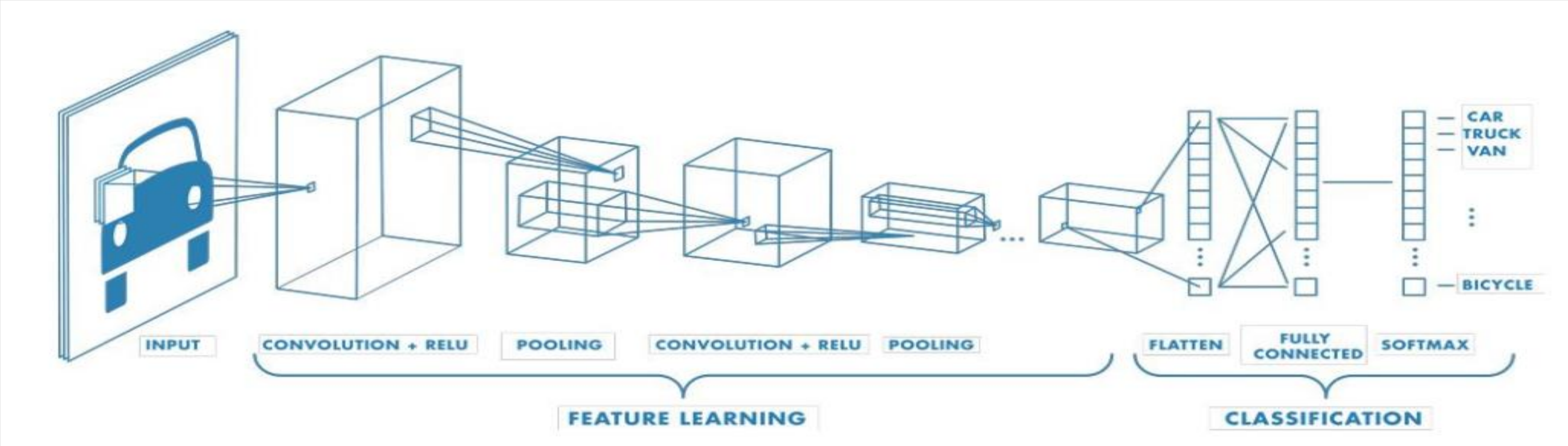
Convolution puts the input images through a set of convolutional filters, each of which activates certain features from the images.

RECTIFIED LINEAR UNIT (ReLU):

ReLu layer allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as activation, because only the activated features are carried forward into the next layer.

POOLING:

Pooling simplifies the output by performing nonlinear down sampling, reducing the number of parameters that the network needs to learn.



CNN — Model 1 — Simple ArchitectureInput Layer
Image 32 x 32 pixelsRGB
Convolutional LayerKernel Size: 3x3Filters: 32
32Feature Maps
Max PoolingPool Size: 2x2
Convolutional LayerKernel Size: 3x3Filters: 64
64Feature Maps
Max PoolingPool Size: 2x2
Flattening
(+54)
Output Layer

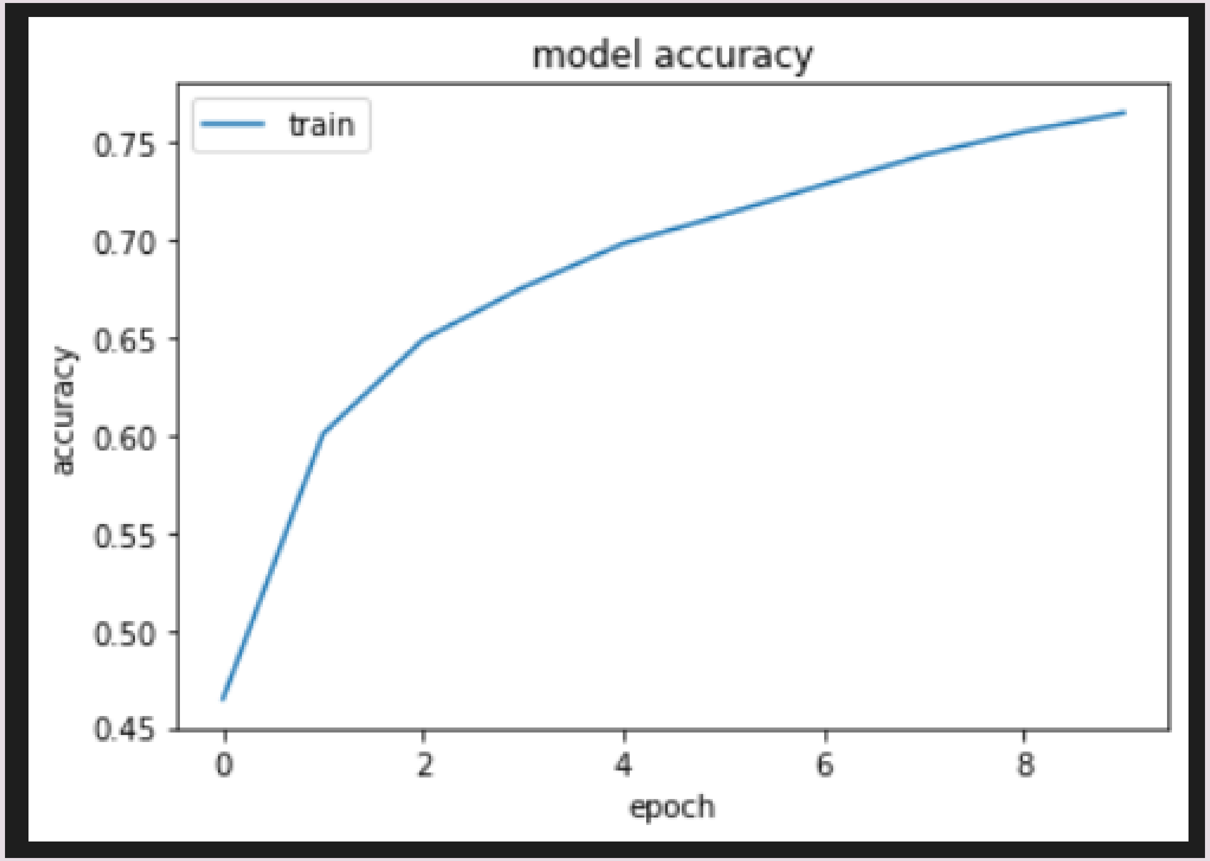
DATA FLOW:

- THE CNN HAS 2 sets of convulsion and pooling layers
- A 32x32 image is given as input to the first convolution layer.
- This convolution layer gives a 32 size filter map.
- This 32 size feature map is reduced to 2x2 by the pooling layer.
- This 2x2 feature map is again given to the second convolution layer.
- This layer gives a 64x64 feature map, which is again reduced to 2x2 feature map.
- This 2x2 map is given to the input or the flattening layer, which flattens the inputs.
- The flatten layer transfer the data to dense or hidden layer with 64 neurons.
- After data processing the hidden layer transfers the data to the output layers which have 10 neurons.

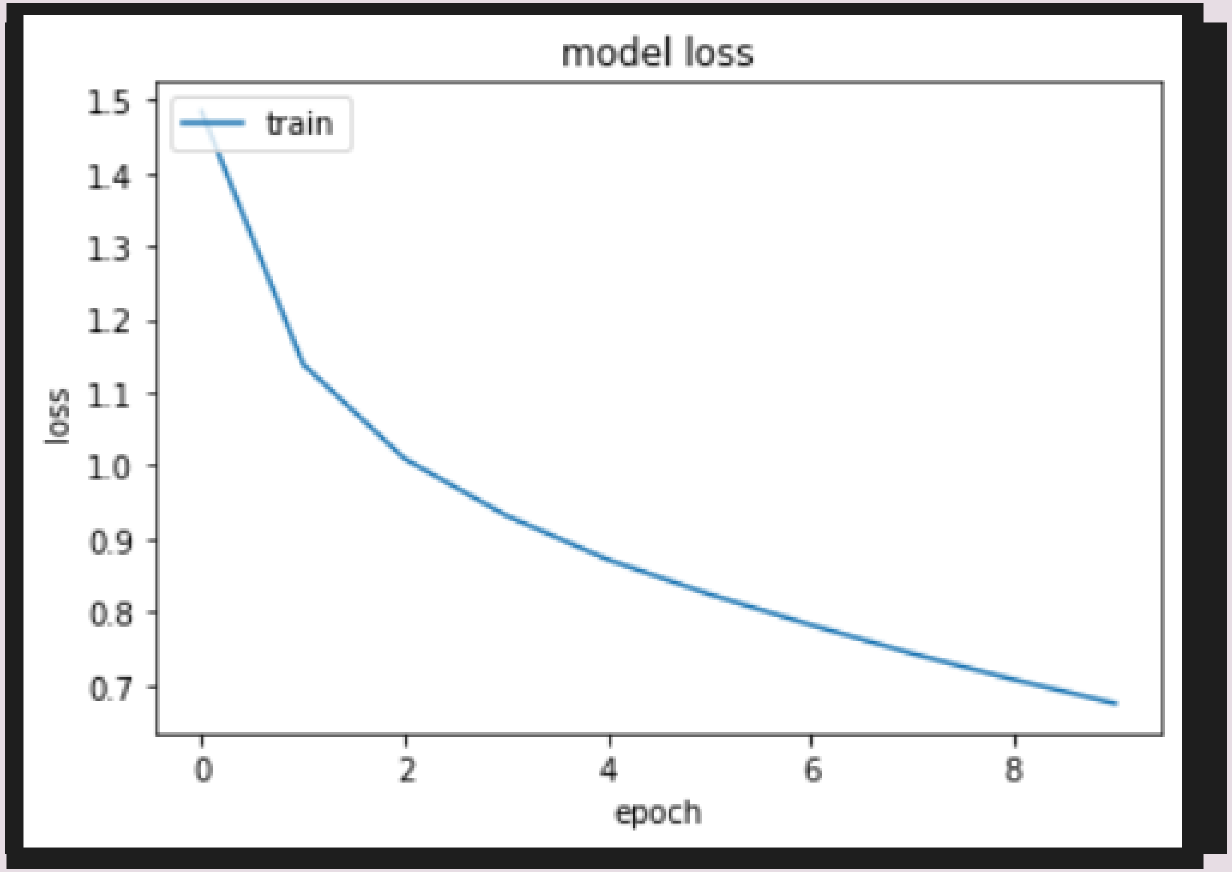
MODEL SUMMARY:

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

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

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

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 64)                147520

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 167,562
Trainable params: 167,562
Non-trainable params: 0
_________________________________________________________________
```

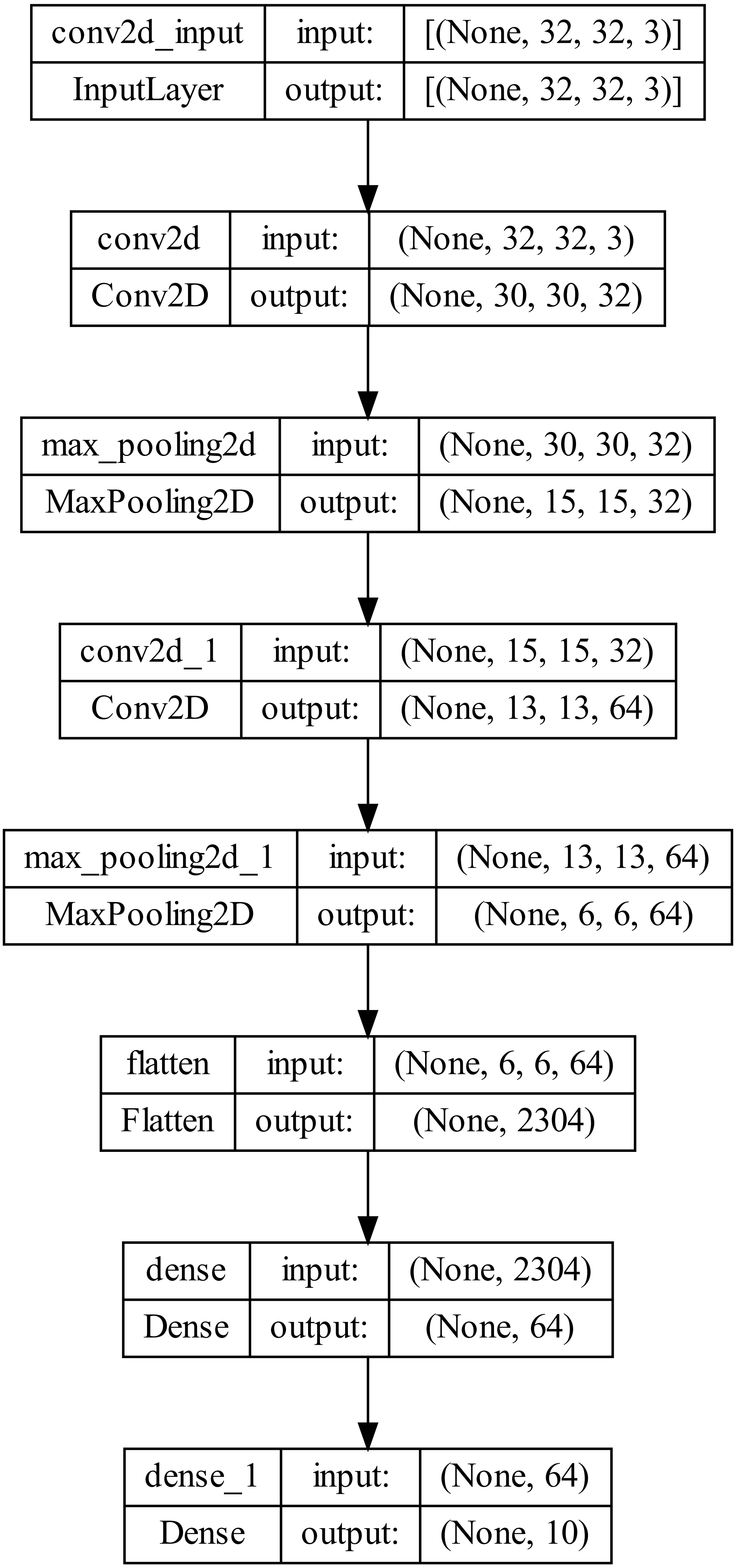
ACCURACY AND LOSS GRAPHS



ACCURACY



LOSS


conv2d_input
InputLayer
input:
output:
[(None, 32, 32, 3)]
[(None, 32, 32, 3)]
conv2d
Conv2D
input:
output:
(None, 32, 32, 3)
(None, 30, 30, 32)
max_pooling2d
MaxPooling2D
input:
output:
(None, 30, 30, 32)
(None, 15, 15, 32)
conv2d_1
Conv2D
input:
output:
(None, 15, 15, 32)
(None, 13, 13, 64)
max_pooling2d_1
MaxPooling2D
input:
output:
(None, 13, 13, 64)
(None, 6, 6, 64)
flatten
Flatten
input:
output:
(None, 6, 6, 64)
(None, 2304)
dense
Dense
input:
output:
(None, 2304)
(None, 64)
dense_1
Dense
input:
output:
(None, 64)
(None, 10)

CODE:

```python
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
```

[2] ✓ 23.6s Python

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

[3] ✓ 1.3s Python

```
(50000, 32, 32, 3)
```

DATA PROCESSING

```python
X_test.shape
```

[4] ✓ 0.5s Python

```
(10000, 32, 32, 3)
```

```python
y_train.shape
```

[5] ✓ 0.8s Python

```
(50000, 1)
```

```python
y_train[:5]
```

[6] ✓ 0.8s Python

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```python
y_train = y_train.reshape(-1,)
y_train[:5]
```

[7] ✓ 0.6s Python

```
array([6, 9, 9, 4, 1], dtype=uint8)
```

```python
y_test = y_test.reshape(-1,)
```

[8] ✓ 0.3s Python

+ Code + Markdown

```python
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

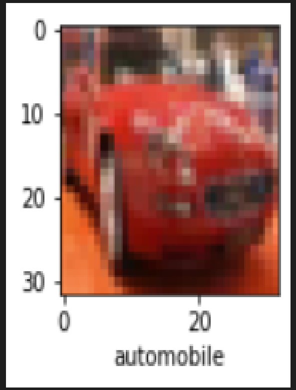
[9] ✓ 0.4s Python

```python
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
```

[10] ✓ 0.4s Python

```python
plot_sample(X_train, y_train, 5)
```

[11] ✓ 0.7s Python




```python
X_train = X_train / 255.0
X_test = X_test / 255.0
```

[12] ✓ 2.4s Python

CNN

```python
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

[34] Python

COMPILATION OF THE MODEL

```python
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

[35] Python

TRAINING OF THE CNN MODEL

```
1 cnn.fit(X_train, y_train, epochs=10)
```

[36] Python

```
Epoch 1/10
1563/1563 [==============================] - 40s 25ms/step - loss: 1.4798 - accuracy: 0.4674
Epoch 2/10
1563/1563 [==============================] - 34s 22ms/step - loss: 1.1286 - accuracy: 0.6028
Epoch 3/10
1563/1563 [==============================] - 35s 22ms/step - loss: 0.9963 - accuracy: 0.6513
Epoch 4/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.9148 - accuracy: 0.6793
Epoch 5/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.8516 - accuracy: 0.7027
Epoch 6/10
1563/1563 [==============================] - 35s 23ms/step - loss: 0.7992 - accuracy: 0.7240
Epoch 7/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.7459 - accuracy: 0.7404
Epoch 8/10
1563/1563 [==============================] - 35s 23ms/step - loss: 0.7022 - accuracy: 0.7551
Epoch 9/10
1563/1563 [==============================] - 34s 22ms/step - loss: 0.6607 - accuracy: 0.7694
Epoch 10/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.6208 - accuracy: 0.7844

<keras.callbacks.History at 0x19a56155940>
```

```python
from keras.models import load_model
loaded_model = load_model("annproject1.h5")
```

[13] ✓ 4.5s Python

```python
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = loaded_model.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test, y_pred_classes))
```

[13] Python

```
313/313 [==============================] - 4s 7ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.81      0.74      1000
           1       0.76      0.83      0.79      1000
           2       0.68      0.49      0.57      1000
           3       0.53      0.51      0.52      1000
           4       0.67      0.60      0.64      1000
           5       0.60      0.63      0.61      1000
           6       0.72      0.81      0.76      1000
           7       0.74      0.77      0.75      1000
           8       0.82      0.78      0.80      1000
           9       0.79      0.73      0.76      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.69     10000
weighted avg       0.70      0.70      0.69     10000
```

```python
loaded_model.evaluate(X_test,y_test)
```

[14] Python

```
313/313 [==============================] - 5s 14ms/step - loss: 0.9368 - accuracy: 0.6971

[0.9367502927780151, 0.6970999836921692]
```

```python
y_pred = loaded_model.predict(X_test)
y_pred[:5]
```

[15] Python

```
313/313 [==============================] - 3s 11ms/step

array([[5.3353757e-05, 3.4360433e-05, 2.4336561e-05, 9.3526763e-01,
        3.9762934e-05, 6.2665984e-02, 4.0117319e-04, 1.0616613e-04,
        1.4036268e-03, 3.6480708e-06],
       [1.2870802e-03, 5.9926391e-01, 5.6438996e-05, 5.2230496e-07,
        1.8217817e-07, 1.1667767e-07, 4.4908387e-07, 1.5154913e-07,
        3.9790127e-01, 1.4897849e-03],
       [1.0875955e-01, 7.1699786e-01, 1.1480342e-03, 3.4640683e-03,
        6.1841414e-04, 8.2107406e-04, 7.4084924e-04, 2.5710468e-03,
        1.1531218e-01, 4.9566925e-02],
       [9.0734780e-01, 6.4525456e-04, 1.2378589e-02, 1.0148327e-02,
        3.8362953e-03, 2.5029414e-04, 4.5067168e-04, 4.3927989e-04,
        6.4393640e-02, 1.0987990e-04],
       [1.3995085e-05, 1.0697803e-06, 2.7654157e-03, 4.9842246e-02,
        5.9531069e-01, 1.6057132e-02, 3.3562270e-01, 1.6582448e-05,
        3.7022788e-04, 2.8315108e-08]], dtype=float32)
```

```
[14]  1  loaded_model.summary()
✓ 0.1s                                                    Python
```

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

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

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

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 64)                147520

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 167,562
Trainable params: 167,562
Non-trainable params: 0
_________________________________________________________________
```

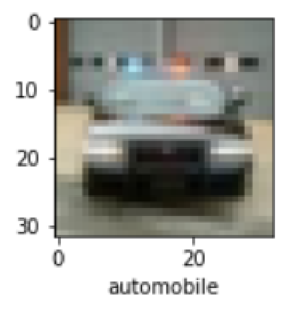
OUTPUT:

```
y_classes = [np.argmax(element) for element in y_pred]

```

[19] Python

```
plot_sample(X_test, y_test,66)
```

[25] Python




```

classes[y_classes[66]]
```

[26] Python

```
'automobile'
```