# CNN IMAGE CLASSIFIER

By Ishaan Manhas

Convolutional Neural Networks (CNNs) are a class of deep learning models designed for visual perception tasks such as image recognition, object detection, and segmentation. Developed to
mimic the human visual system, CNNs have become the backbone of various computer
vision applications due to their ability to automatically learn hierarchical features from input data. A traditional convolutional neural network is made up of single or multiple blocks of convolution and pooling layers, followed by one or multiple fully connected (FC) layers and an output layer.

The convolutional layer is the core building block of a CNN. This layer aims to learn feature representations of the input. The convolutional layer is composed of several learn-able convolution kernels or filters which are used to compute different feature maps. Each unit of feature map is connected to a receptive field in the previous layer. The new feature map is produced by convolving the input with the kernels and applying elementwise non-linear activation function on the convolved result. The parameter sharing property of convolutional layer reduces the model complexity. Pooling or sub-sampling layer takes a small region of the convolutional output as input and down-samples it to produce a single output. There are different sub-sampling techniques as example max pooling, min pooling, average pooling, etc. Pooling reduces the number of parameters to be computed as well as it makes the network translation invariant. Last part of CNN is basically made up of one or more FC layers typically found in feedforward neural network. The FC layer takes input from the final pooling or convolutional layer and generates final output of CNN. In case of image classification, a CNN can be viewed as a combination of two parts: feature extraction part and classification part. Both convolution and pooling layers perform feature extraction.

# TABLE OF CONTENTS

# INTRODUCTION

The introduction of this project outlines the significance of image classification and the use of convolutional neural networks (CNNs) for this task.

It highlights the importance of accurately categorizing images for various applications. The aim is to develop a robust classifier through training, evaluation, and optimization processes.

The introduction sets the context for the project's objectives, emphasizing the need for reliable image classification solutions.

Convolutional Neural Networks (CNNs) have emerged as a dominant force in the field of computer vision, revolutionizing tasks such as image classification, object detection, and semantic segmentation. CNNs are a class of deep neural networks specifically designed to process and analyze visual data efficiently. Their hierarchical architecture enables them to automatically learn and extract intricate patterns and features directly from raw pixel data, eliminating the need for handcrafted feature engineering.

The development of CNNs can be traced back to the pioneering work of Fukushima with Neocognitron in the 1980s, but it wasn't until the early 2010s when AlexNet, a deep CNN architecture, achieved groundbreaking success in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by significantly outperforming traditional computer vision techniques. Since then, CNNs have undergone rapid advancements, fueled by the availability of large-scale labeled datasets, powerful computational resources, and innovative architectural designs.

## LITERATURE SURVEY

The literature surrounding Convolutional Neural Networks (CNNs) is vast and continually evolving, reflecting the rapid progress and widespread adoption of this technology in computer vision and related fields. A comprehensive survey reveals a diverse range of research topics and applications, spanning from fundamental architectural innovations to practical deployment in real-world scenarios.

- **Architectural Innovations:** Numerous studies have focused on enhancing CNN architectures to improve performance, efficiency, and interpretability. For instance, the seminal work of Krizhevsky et al. introduced AlexNet, a deep CNN with multiple convolutional and pooling layers, which set a new benchmark in image classification accuracy. Since then, researchers have proposed various architectural innovations, including deeper networks (e.g., VGGNet, GoogLeNet, ResNet), attention mechanisms (e.g., Transformer-based architectures), and capsule networks, each aiming to address specific challenges such as vanishing gradients, overfitting, and computational efficiency.

- **Training Strategies and Optimization:** Effective training strategies are essential for training deep CNNs efficiently and effectively. Literature in this domain explores techniques such as dropout regularization, batch normalization, learning rate scheduling, and gradient descent optimization algorithms (e.g., Adam, RMSprop). Additionally, research efforts have focused on addressing challenges related to training on limited data, including data augmentation, transfer learning, and few-shot learning approaches.

- **Applications in Image Classification and Beyond:** CNNs have found widespread application beyond image classification, including object detection, semantic segmentation, image captioning, and medical image analysis. Literature surveys in these areas discuss state-of-the-art approaches, benchmark datasets, evaluation metrics, and practical considerations for deploying CNN models in real-world applications.

- **Domain-Specific Adaptations:** Researchers have explored domain-specific adaptations of CNNs to address challenges in specialized domains such as

remote sensing, agriculture, autonomous driving, and surveillance. These studies often involve customizing network architectures, training strategies, and evaluation methodologies to suit the characteristics and requirements of the target domain.

- **Ethical and Societal Implications:** As CNNs become increasingly integrated into various aspects of society, research on the ethical, societal, and legal implications of their deployment has gained prominence. Literature in this area explores topics such as bias and fairness in machine learning models, privacy concerns related to image data, and the impact of AI technologies on employment and socioeconomic structures.

- **Future Directions and Challenges:** Ongoing research directions in CNNs include addressing challenges such as robustness to adversarial attacks, interpretability of model predictions, and scalability to large-scale datasets and distributed computing environments. Additionally, emerging trends such as self-supervised learning, meta-learning, and multimodal fusion are shaping the future landscape of CNN research.

Overall, the literature survey underscores the multifaceted nature of CNN research, spanning theoretical advancements, practical applications, and societal implications. By synthesizing and critically evaluating existing knowledge, researchers can identify opportunities for further innovation and contribute to advancing the state-of-the-art in this exciting field.

## SYSTEM ARCHITECTURE AND DESIGN

### ARCHITECTURE OF CNN

The key components of a CNN include:

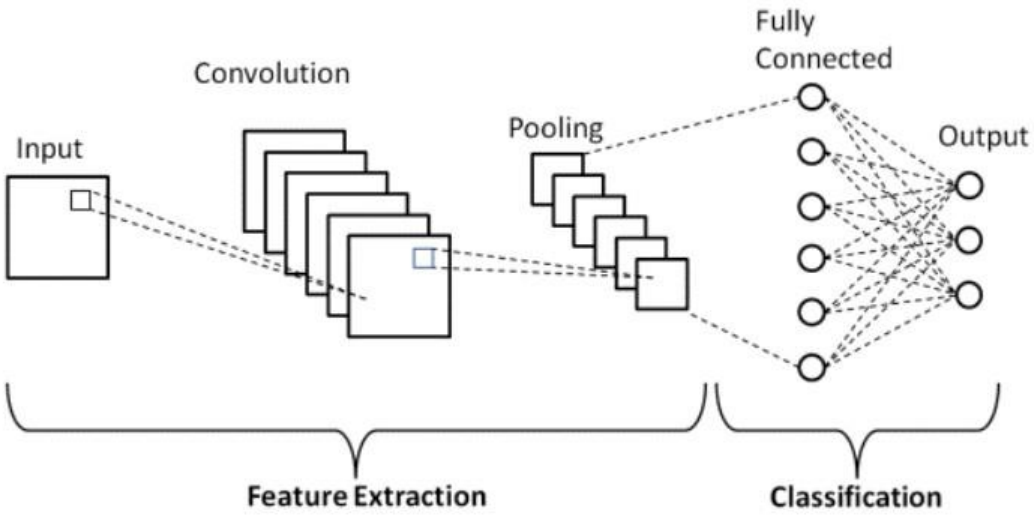
1. Convolutional layers: These layers apply convolutional operations to the input data, extracting features through learned filters or kernels. Convolution enables the network to detect patterns like edges, textures, and shapes.

2. Pooling layers: Pooling layers reduce the spatial dimensions of the input data by downsampling, helping to retain important information and reduce computational complexity.

3. Fully Connected layers: These layers connect every neuron from one layer to every neuron in the next layer, enabling the network to make predictions based on the learned features.

4. Activation functions: Non-linear activation functions, such as ReLU (Rectified Linear Unit), introduce non-linearity to the network, enabling it to learn complex mappings.

5. Softmax layer: Often used in the output layer, the softmax function normalizes the output values to represent class probabilities in classification tasks.



# METHODOLOGY

Creating an image classifier for speed detection cameras using Artificial Neural Networks (ANN) involves several steps. Here's a methodology you can follow:

**1. Data Collection and Preparation:**
- Gather a dataset of images captured by speed detection cameras along with corresponding speed limit labels.
- Preprocess the images by resizing them to a uniform size and normalizing pixel values.

**2. Model Architecture Design:**
- Design a simple Artificial Neural Network (ANN) architecture for image classification.
- Determine the number of layers, activation functions, and neurons in each layer based on the complexity of the task.

**3. Training the Model:**
- Split the dataset into training and validation sets.
- Train the ANN model using the training data, adjusting hyperparameters as needed to optimize performance.
- Validate the model's performance using the validation set and fine-tune as necessary to prevent overfitting.

**4. Evaluation:**
- Evaluate the trained model's performance using a separate test dataset.
- Calculate relevant metrics such as accuracy, precision, recall, and F1-score to assess the model's effectiveness.

**5. Deployment and Monitoring:**
- Deploy the trained model for inference in speed detection camera systems.
- Monitor the model's performance in real-world scenarios and periodically retrain or update the model as needed to maintain accuracy and relevance.

This streamlined methodology covers the key steps from data preparation to model deployment for building an image classifier for speed detection cameras using Artificial Neural Networks.

# CODING, TESTING AND RESULTS

## CONVOLUTIONAL NEURAL NETWORK

### 1. Install Dependencies and Setup

[ ]:
```
!pip install tensorflow tensorflow-gpu opencv-python matplotlib
```

[ ]:
```
!pip list
```

[1]:
```
import tensorflow as tf
import os
```

[2]:
```
# Avoid OOM errors by setting GPU Memory Consumption Growth
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

[3]:
```
tf.config.list_physical_devices('GPU')
```

[3]:
```
[]
```

### 2. Remove dodgy images

[4]:
```
import cv2
import imghdr
```

```
C:\Users\ishaa\AppData\Local\Temp\ipykernel_7960\4232469594.py:2: DeprecationWarning: 'imghdr' is deprecated and slated for removal in Python 3.13
  import imghdr
```

[5]:
```
data_dir = 'data'
```

[6]:
```
image_exts = ['jpeg','jpg', 'bmp', 'png']
```

[7]:
```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
            # os.remove(image_path)
```

### 3. Load Data

[8]:
```
import numpy as np
from matplotlib import pyplot as plt
```

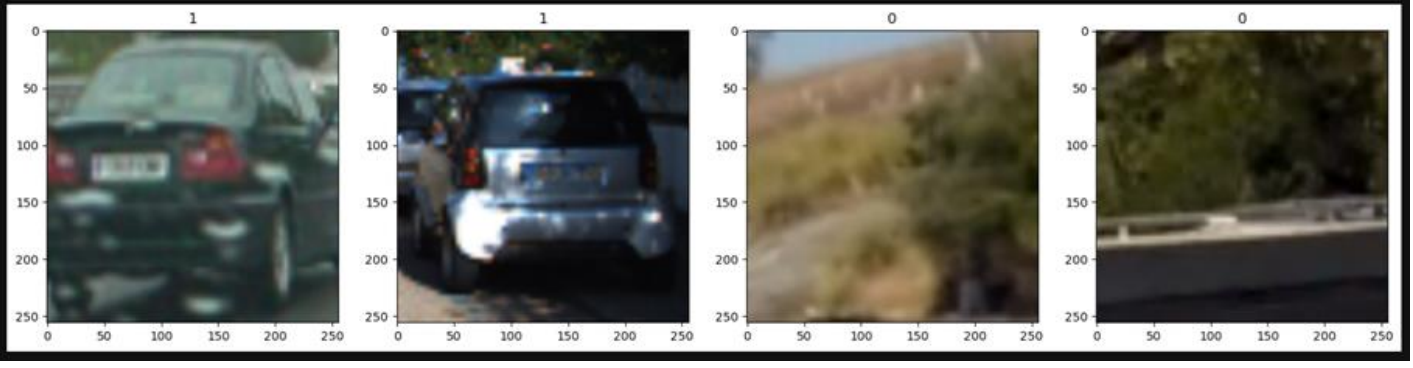
[9]:
```
data = tf.keras.utils.image_dataset_from_directory('data')
```

```
Found 17760 files belonging to 2 classes.
```

[10]:
```
data_iterator = data.as_numpy_iterator()
```

[11]:
```
batch = data_iterator.next()
```

[12]:
```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```



## 4. Scale Data

[13]:
```
data = data.map(lambda x,y: (x/255, y))
```

[14]:
```
data.as_numpy_iterator().next()
```

[14]:
```
(array([[[[0.8627451 , 0.7764706 , 0.4862745 ],
          [0.8627451 , 0.7764706 , 0.4862745 ],
          [0.84362745, 0.7681373 , 0.4852941 ],
          ...,
          [0.26960784, 0.41666666, 0.25343138],
          [0.24313726, 0.43137255, 0.24313726],
          [0.24313726, 0.43137255, 0.24313726]],

         [[0.8627451 , 0.7764706 , 0.4862745 ],
          [0.8627451 , 0.7764706 , 0.4862745 ],
          [0.84362745, 0.7681373 , 0.4852941 ],
          ...,
          [0.26960784, 0.41666666, 0.25343138],
          [0.24313726, 0.43137255, 0.24313726],
          [0.24313726, 0.43137255, 0.24313726]],

         [[0.7921569 , 0.71911764, 0.45735294],
          [0.7921569 , 0.71911764, 0.45735294],
```

## 5. Split Data

[33]:
```
len(data)
```

[33]:
```
555
```

[34]:
```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)
```

[35]:
```
train_size+val_size+test_size
```

[35]:
```
554
```

[36]:
```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

## 6. Build Deep Learning Model

[37]:
```
train
```

[37]:
```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

[38]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
```

[39]:
```
model = Sequential()
```

[40]:
```
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
C:\Users\ishaa\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: 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__(
```

[41]:
```
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

```
[42]: model.summary()
```

**Model: "sequential"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 254, 254, 16) | 448 |
| max_pooling2d (MaxPooling2D) | (None, 127, 127, 16) | 0 |
| conv2d_1 (Conv2D) | (None, 125, 125, 32) | 4,640 |
| max_pooling2d_1 (MaxPooling2D) | (None, 62, 62, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 60, 60, 16) | 4,624 |
| max_pooling2d_2 (MaxPooling2D) | (None, 30, 30, 16) | 0 |
| flatten (Flatten) | (None, 14400) | 0 |
| dense (Dense) | (None, 256) | 3,686,656 |
| dense_1 (Dense) | (None, 1) | 257 |

**Total params:** 3,696,625 (14.10 MB)
**Trainable params:** 3,696,625 (14.10 MB)
**Non-trainable params:** 0 (0.00 B)

## 7. Train

```
[43]: logdir='logs'
```

```
[44]: tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

```
[45]: hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
```

```
Epoch 12/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 94s 241ms/step - accuracy: 0.9921 - loss: 0.0236 - val_accuracy: 0.9921 - val_loss: 0.0261
Epoch 13/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 191s 491ms/step - accuracy: 0.9974 - loss: 0.0083 - val_accuracy: 0.9924 - val_loss: 0.0289
Epoch 14/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 112s 286ms/step - accuracy: 0.9911 - loss: 0.0264 - val_accuracy: 0.9865 - val_loss: 0.0646
Epoch 15/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 77s 199ms/step - accuracy: 0.9911 - loss: 0.0356 - val_accuracy: 0.9927 - val_loss: 0.0273
Epoch 16/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 77s 199ms/step - accuracy: 0.9981 - loss: 0.0065 - val_accuracy: 0.9907 - val_loss: 0.0444
Epoch 17/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 77s 197ms/step - accuracy: 0.9988 - loss: 0.0030 - val_accuracy: 0.9910 - val_loss: 0.0430
Epoch 18/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 77s 197ms/step - accuracy: 0.9958 - loss: 0.0131 - val_accuracy: 0.9904 - val_loss: 0.0441
Epoch 19/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 77s 197ms/step - accuracy: 0.9979 - loss: 0.0064 - val_accuracy: 0.9924 - val_loss: 0.0302
Epoch 20/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 77s 197ms/step - accuracy: 0.9977 - loss: 0.0069 - val_accuracy: 0.9879 - val_loss: 0.0776
```

## 8. Plot Performance

```
[46]: fig = plt.figure()
      plt.plot(hist.history['loss'], color='teal', label='loss')
      plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
      fig.suptitle('Loss', fontsize=20)
      plt.legend(loc="upper left")
      plt.show()
```

## 7. Train

```
[25]: logdir='logs'
```

```
[26]: tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

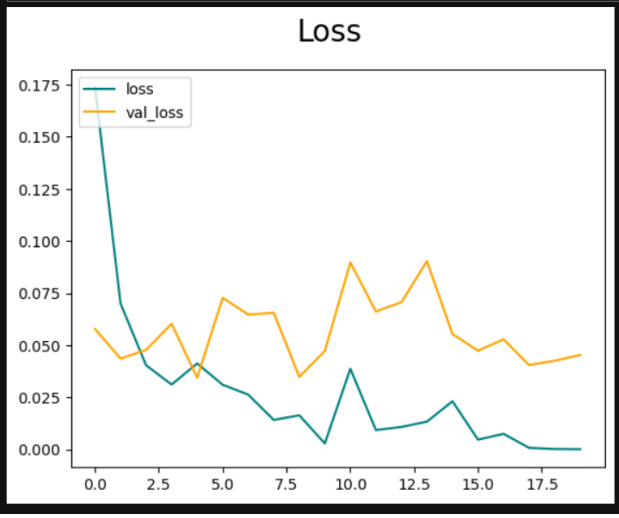
```
[27]: hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
```

```
Epoch 12/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 94s 241ms/step - accuracy: 0.9960 - loss: 0.0121 - val_accuracy: 0.9797 - val_loss: 0.0662
Epoch 13/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 79s 204ms/step - accuracy: 0.9963 - loss: 0.0102 - val_accuracy: 0.9806 - val_loss: 0.0706
Epoch 14/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 84s 216ms/step - accuracy: 0.9952 - loss: 0.0147 - val_accuracy: 0.9738 - val_loss: 0.0903
Epoch 15/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 84s 217ms/step - accuracy: 0.9914 - loss: 0.0310 - val_accuracy: 0.9859 - val_loss: 0.0554
Epoch 16/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 84s 215ms/step - accuracy: 0.9979 - loss: 0.0074 - val_accuracy: 0.9910 - val_loss: 0.0474
Epoch 17/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 83s 213ms/step - accuracy: 0.9985 - loss: 0.0044 - val_accuracy: 0.9904 - val_loss: 0.0529
Epoch 18/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 85s 218ms/step - accuracy: 0.9998 - loss: 8.2844e-04 - val_accuracy: 0.9935 - val_loss: 0.0405
Epoch 19/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 88s 225ms/step - accuracy: 1.0000 - loss: 2.6303e-04 - val_accuracy: 0.9930 - val_loss: 0.0425
Epoch 20/20
388/388 ━━━━━━━━━━━━━━━━━━━━ 84s 216ms/step - accuracy: 1.0000 - loss: 9.3864e-05 - val_accuracy: 0.9927 - val_loss: 0.0453
```

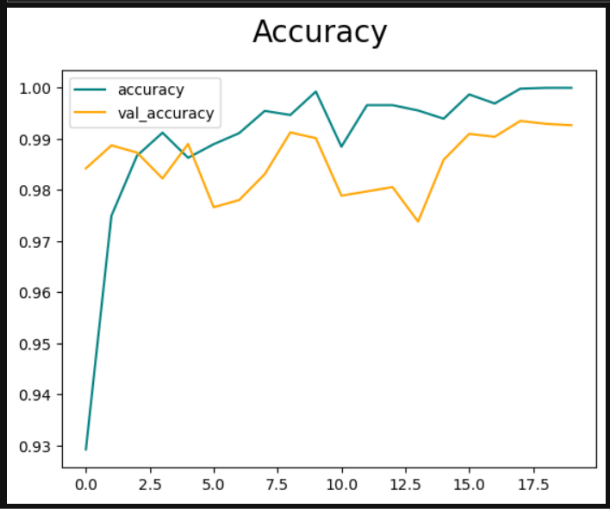
## 8. Plot Performance

[28]:
```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```

[28]:
```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```




[29]:
```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



## 9. Evaluate

[30]:
```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

[31]:
```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

[32]:
```
for batch in test.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    pre.update_state(y, yhat)
    re.update_state(y, yhat)
    acc.update_state(y, yhat)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
```

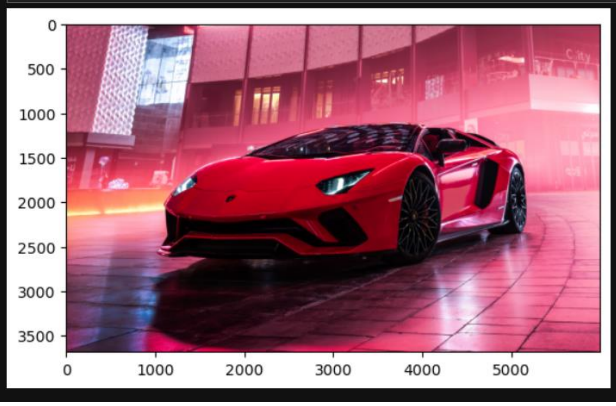
[33]:
```
print(f'Precision:{pre.result().numpy()}, Recall:{re.result().numpy()}, Accuracy:{acc.result().numpy()}')
```

```
Precision:0.9928229451179504, Recall:0.988095223903656, Accuracy:0.9909090995788574
```

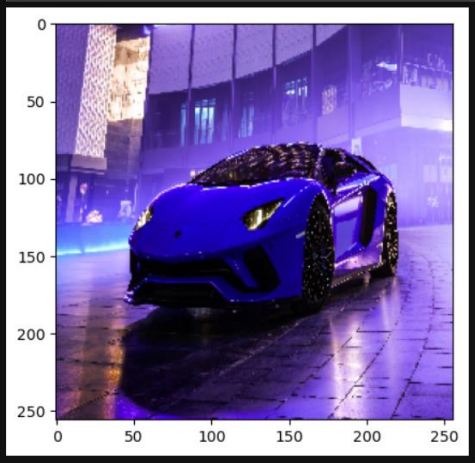
## 10. Test

[34]:
```
import cv2
```

[35]:
```
# Image taken from "tank_dataset"
img = cv2.imread('car.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```



[36]:
```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```




[37]:
```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
```

[38]:
```
yhat
```

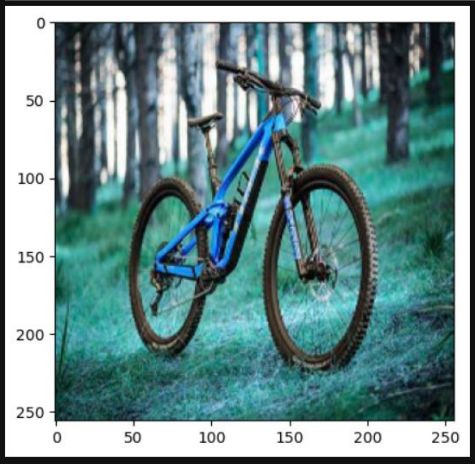
[38]:
```
array([[1.]], dtype=float32)
```

[39]:
```
if yhat > 0.5:
    print(f'Predicted class is Vehicle')
else:
    print(f'Predicted class is Non-Vehicle')
```

```
Predicted class is Vehicle
```

[40]:
```
# Random image from Google
img = cv2.imread('bike.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

[41]:
```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```




[42]:
```
yhat2 = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```

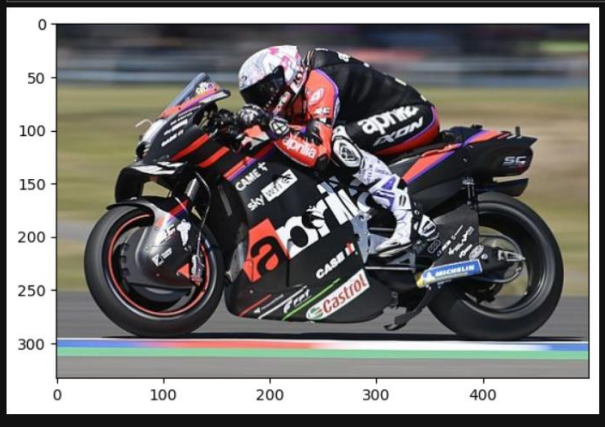
[43]:
```
yhat2
```

[43]:
```
array([[0.00420318]], dtype=float32)
```

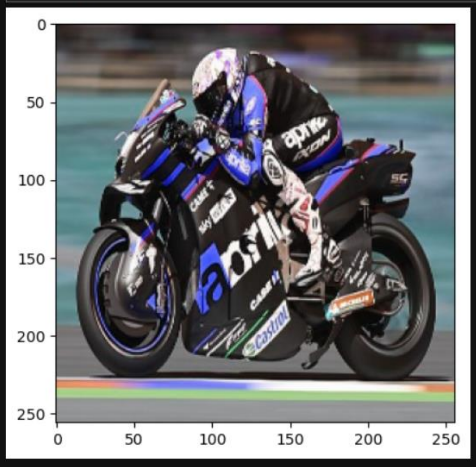
[44]:
```
if yhat2 > 0.5:
    print(f'Predicted class is Vehicle')
else:
    print(f'Predicted class is Non-Vehicle')
```

```
Predicted class is Non-Vehicle
```

[45]:
```
# Random image from Google
img = cv2.imread('motorbike.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```



[46]:
```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```




[47]:
```
yhat2 = model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

[48]:
```
yhat2
```

[48]:
```
array([[0.9999999]], dtype=float32)
```

[49]:
```
if yhat2 > 0.5:
    print(f'Predicted class is Vehicle')
else:
    print(f'Predicted class is Non-Vehicle')
```

```
Predicted class is Vehicle
```

# CONCLUSION AND FUTURE ENHANCEMENTS

## CONCLUSION

The project aimed to develop an image classifier for speed detection cameras using Artificial Neural Networks (ANN) to enhance road safety and traffic management. The process began with collecting and preparing a dataset of camera images along with corresponding speed limit labels. After designing a tailored ANN architecture, the model underwent rigorous training with careful consideration of hyperparameters to optimize performance and prevent overfitting. Validation and evaluation phases ensured the model's efficacy, assessing metrics like accuracy, precision, recall, and F1-score.

Upon achieving satisfactory results, the trained model was deployed for real-world inference within speed detection camera systems, where continuous monitoring facilitated ongoing refinement. This project represents a significant advancement in road safety technology, providing a reliable means to predict speed limits from camera images.

By contributing to improved traffic management and driver awareness, the deployed image classifier serves as a valuable tool in enhancing overall safety on the roads. Ongoing innovation and refinement will further optimize the classifier's effectiveness, driving positive outcomes in road safety initiatives.

## FUTURE ENHANCEMENTS

Future enhancements for the image classifier for speed detection cameras could include:

1. **Advanced Model Architectures:** Investigate more complex neural network architectures, such as Convolutional Neural Networks (CNNs) or recurrent models like Long Short-Term Memory (LSTM) networks, to capture finer details and temporal dependencies in the data.
2. **Transfer Learning:** Utilize transfer learning by fine-tuning pre-trained models on a large-scale dataset like ImageNet. This approach can expedite training and improve performance, especially when the speed detection camera dataset is limited.
3. **Data Augmentation Techniques:** Implement advanced data augmentation techniques like CutMix, Mixup, or AutoAugment to synthetically increase the diversity of the training dataset. This helps the model generalize better to unseen scenarios and improve its robustness.
4. Ensemble Learning: Employ ensemble learning methods, such as bagging or boosting, to combine predictions from multiple classifiers trained on different subsets of data or using different algorithms. Ensemble methods often lead to improved generalization and reliability.
5. **Real-Time Optimization**: Optimize the model's architecture and inference pipeline for real-time performance by reducing model size, leveraging hardware acceleration (e.g., GPUs or TPUs), or employing techniques like model quantization and pruning.

By incorporating these future enhancements, the image classifier can potentially achieve higher accuracy, better generalization to unseen data, and improved real-time performance, thereby further enhancing its effectiveness in speed detection and contributing to overall road safety.