EXPERIMENTATION:

CNN AND YOLO ALGORITHM

CIFAR-10 DATASET

ROSHAN PETER

All the codes were done in python using the PyCharm IDE

The parameter values have been chosen carefully after many trial and error and we chose the one that gave better accuracy.

The final code that gave the best accuracy after the experimentation has been used in the final code.

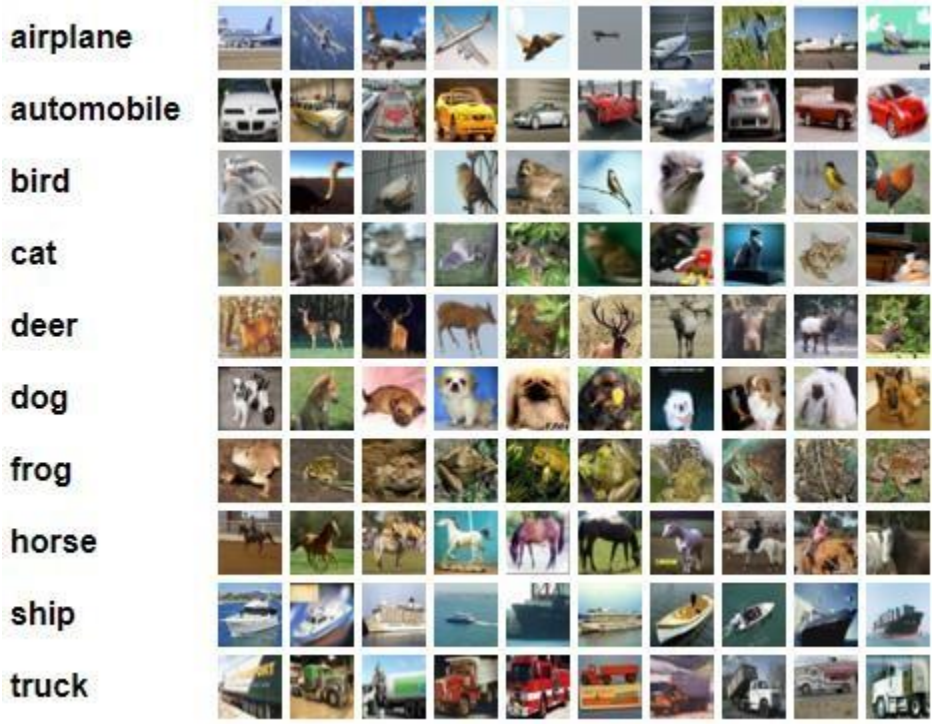
The accuracy vs epochs and loss vs epochs graphs were plotted using Tensorboard as we can also utilize the other features in it as well.

INDEX

CIFAR-10 DATASET

CIFAR-10 dataset is an inbuilt dataset in keras. It consists of 60000

(50000 for training and 10000 for testing), 32x32 color images in 10 classes with 6000 images per class. The classes are completely mutually exclusive.



Since the images in CIFAR-10 are low-resolution (32x32), this dataset can allow researchers to quickly try different algorithms to see what works.

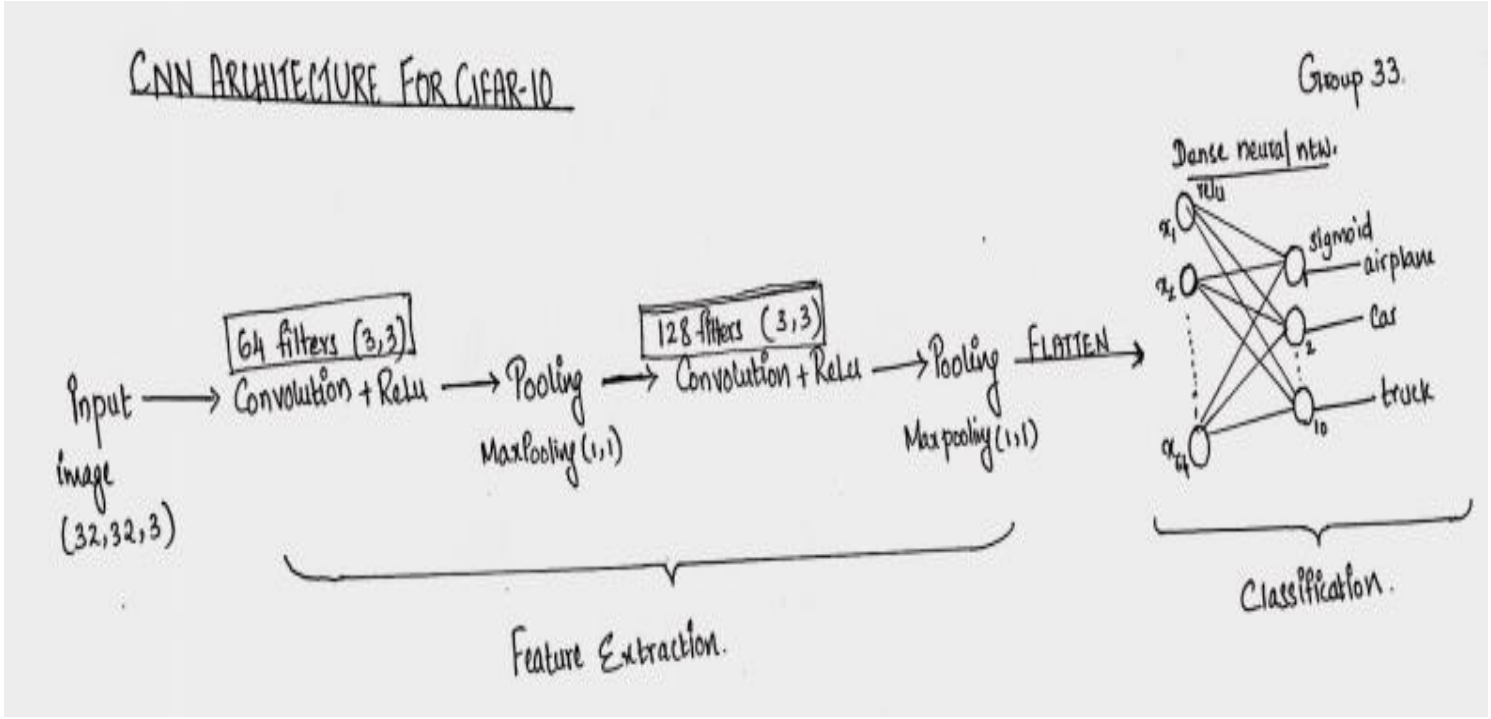
AIM

Build a convolutional neural network for cifar-10 dataset that provides a descent accuracy after performing experiments on the basic code.

Also provide a new application domain (Object detection using the Yolo algorithm)

ARCHITECTURE

We build our custom dataset. We ran the code several times to find the optimum value of parameters so that we could have the best accuracy.



FINAL CODE

The code that gave the best accuracy

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

# we receive the train and test data
(X_train, y_train), (X_test, y_test) =
tf.keras.datasets.cifar10.load_data()

# normalization of the pixels(0 to 1 range)
X_train = X_train/255
X_test = X_test/255

# changing it to to a list form from the matrix form
y_train = y_train.reshape(-1,)
y_test = y_test.reshape(-1,)

# building the model
model = keras.Sequential([
    layers.Conv2D(64, (3, 3), padding='same', activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), padding='same',
activation='relu'),
    layers.MaxPooling2D(),

    # dense network layer(for classification)
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='sigmoid')
    ])

# using tensorboard for plotting (loss vs epochs) and (accuracy
vs epochs)
tb_callback = tf.keras.callbacks.TensorBoard(log_dir='logs/',
histogram_freq=1)

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

```
model.fit(X_train, y_train, epochs=5, callbacks=[tb_callback])
model.summary()
print('this is the accuracy of the test data: ')
model.evaluate(X_test, y_test)

y_pred = model.predict(X_test)
yp = [np.argmax(i) for i in y_pred]

# different graphs and summary
from sklearn.metrics import classification_report,
confusion_matrix
print(classification_report(y_test, yp))
cm = confusion_matrix(y_test, yp)
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('predicted')
plt.ylabel('true')
plt.title('confusion matrix')
plt.show()
```

Training

```
1563/1563 [==============================] - 77s 49ms/step - loss: 1.6030 - accuracy: 0.4193
Epoch 2/5
1563/1563 [==============================] - 77s 49ms/step - loss: 1.0009 - accuracy: 0.6508
Epoch 3/5
1563/1563 [==============================] - 77s 49ms/step - loss: 0.8467 - accuracy: 0.7082
Epoch 4/5
1563/1563 [==============================] - 78s 50ms/step - loss: 0.7307 - accuracy: 0.7480
Epoch 5/5
1563/1563 [==============================] - 77s 49ms/step - loss: 0.6417 - accuracy: 0.7773
```

Evaluating the accuracy of the test data

```
this is the accuracy of the test data:
313/313 [==============================] - 4s 13ms/step - loss: 0.8331 - accuracy: 0.7161
```

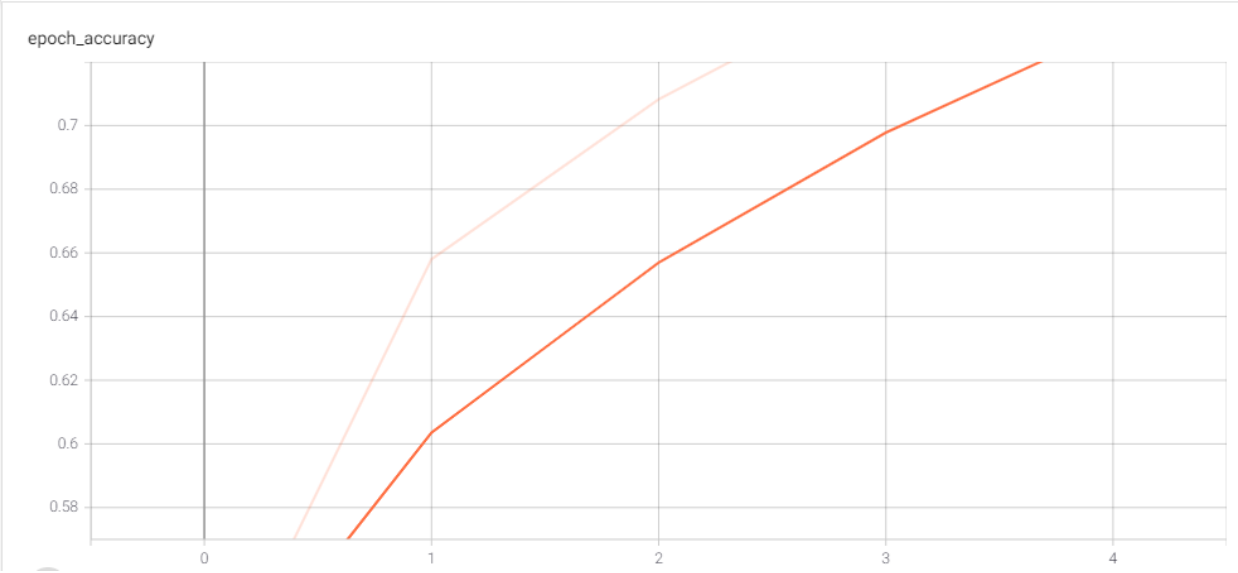
Summary of the model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 64)                524352
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 600,650
Trainable params: 600,650
Non-trainable params: 0
_________________________________________________________________
```

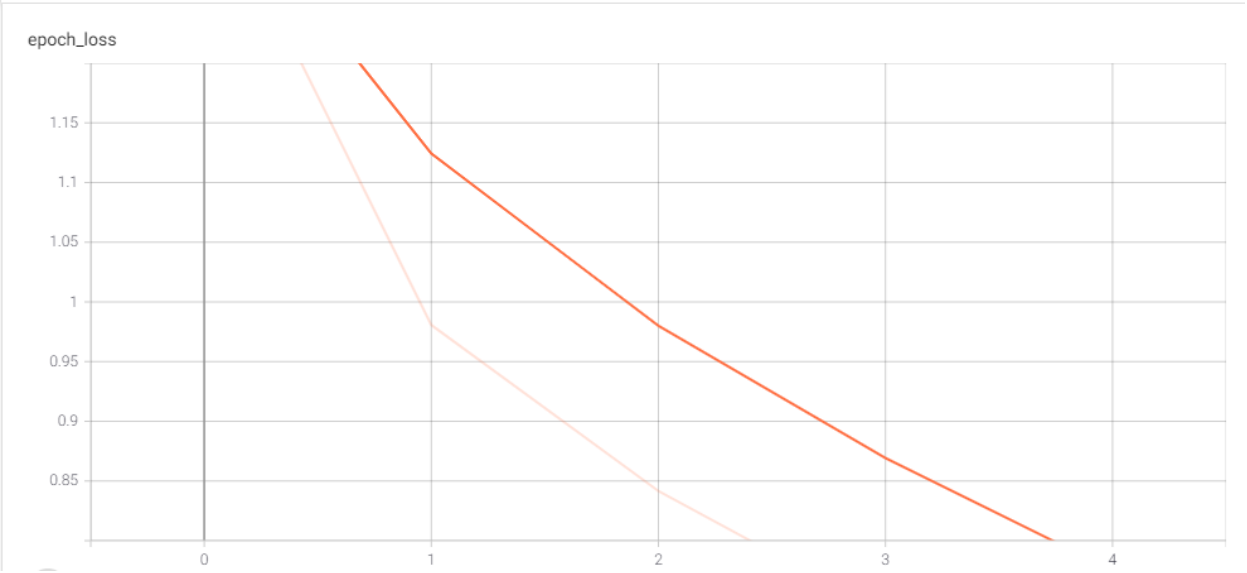
Classification report

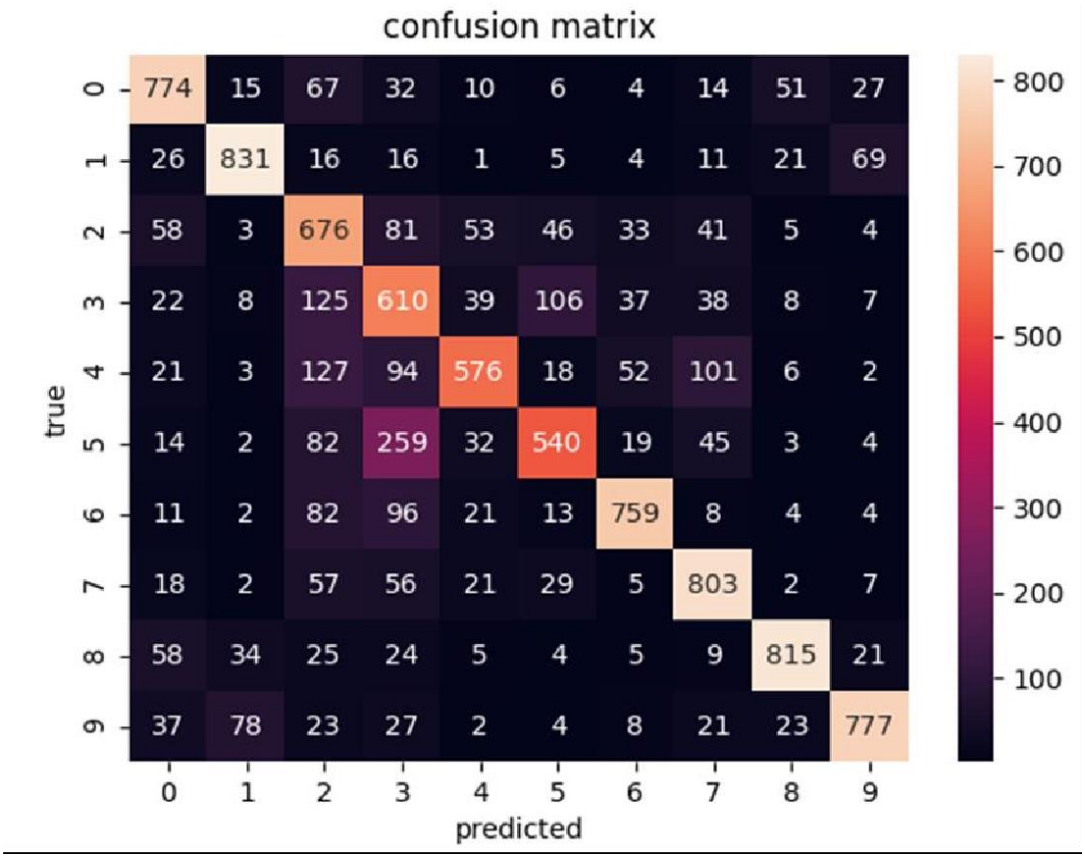
	precision	recall	f1-score	support
0	0.74	0.77	0.76	1000
1	0.85	0.83	0.84	1000
2	0.53	0.68	0.59	1000
3	0.47	0.61	0.53	1000
4	0.76	0.58	0.65	1000
5	0.70	0.54	0.61	1000
6	0.82	0.76	0.79	1000
7	0.74	0.80	0.77	1000
8	0.87	0.81	0.84	1000
9	0.84	0.78	0.81	1000
accuracy			0.72	10000
macro avg	0.73	0.72	0.72	10000
weighted avg	0.73	0.72	0.72	10000

Accuracy vs. epochs



Loss vs. epochs





EXPERIMENTS

Using artificial neural network (ANN)

Our initial thought was why we could not do the same thing using artificial neural network and so we tried and the accuracy we got was not that great compared to that using convolutional neural network.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

(X_train, y_train), (X_test, y_test) =
keras.datasets.cifar10.load_data()
X_train = X_train/255
X_test = X_test/255

model = keras.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(3000, activation='relu'),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='sigmoid')
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)
model.fit(X_train, y_train, epochs=5)
print('This is the evaluation result: ')
model.evaluate(X_test, y_test)
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 114s 72ms/step - loss: 2.2571 - accuracy: 0.2777
Epoch 2/5
1563/1563 [==============================] - 115s 73ms/step - loss: 1.6938 - accuracy: 0.3948
Epoch 3/5
1563/1563 [==============================] - 116s 74ms/step - loss: 1.6154 - accuracy: 0.4259
Epoch 4/5
1563/1563 [==============================] - 115s 73ms/step - loss: 1.5587 - accuracy: 0.4392
Epoch 5/5
1563/1563 [==============================] - 115s 74ms/step - loss: 1.5206 - accuracy: 0.4562
This is the evaluation result:
313/313 [==============================] - 3s 9ms/step - loss: 1.5374 - accuracy: 0.4467
```

CNN model with Adam as optimizer

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

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

X_train = X_train/255
X_test = X_test/255

y_train = y_train.reshape(-1,)
y_test = y_test.reshape(-1,)


model = keras.Sequential([
    layers.Conv2D(32, (3, 3), padding='valid',
activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), padding='valid',
activation='relu'),
    layers.MaxPooling2D(),

    # dense network layer
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='sigmoid')
    ])
```

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

model.fit(X_train, y_train, epochs=5)
print('this is the accuracy of the test data: ')
model.evaluate(X_test, y_test)
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 29s 18ms/step - loss: 1.7549 - accuracy: 0.3576
Epoch 2/5
1563/1563 [==============================] - 28s 18ms/step - loss: 1.1809 - accuracy: 0.5810
Epoch 3/5
1563/1563 [==============================] - 28s 18ms/step - loss: 1.0333 - accuracy: 0.6374
Epoch 4/5
1563/1563 [==============================] - 28s 18ms/step - loss: 0.9323 - accuracy: 0.6774
Epoch 5/5
1563/1563 [==============================] - 28s 18ms/step - loss: 0.8690 - accuracy: 0.6963
this is the accuracy of the test data:
313/313 [==============================] - 2s 4ms/step - loss: 0.9599 - accuracy: 0.6625
```

CNN model using Stochastic Gradient Descent (SGD)

```
model.compile(
    optimizer='SGD',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 29s 18ms/step - loss: 2.1639 - accuracy: 0.2061
Epoch 2/5
1563/1563 [==============================] - 28s 18ms/step - loss: 1.6770 - accuracy: 0.3985
Epoch 3/5
1563/1563 [==============================] - 28s 18ms/step - loss: 1.4598 - accuracy: 0.4753
Epoch 4/5
1563/1563 [==============================] - 28s 18ms/step - loss: 1.3537 - accuracy: 0.5179
Epoch 5/5
1563/1563 [==============================] - 28s 18ms/step - loss: 1.2734 - accuracy: 0.5500
this is the accuracy of the test data:
313/313 [==============================] - 2s 4ms/step - loss: 1.3268 - accuracy: 0.5351
```

As it is very clear from the output that the Adam optimizer gives more accuracy than SGD optimizer we will be using the Adam as our optimizer in our code henceforth.

CNN model with loss as mean square error (MSE)

```
model.compile(
    optimizer='adam',
    loss='MSE',
    metrics=['accuracy']
)
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 26s 17ms/step - loss: 20.5403 - accuracy: 0.0999
Epoch 2/5
1563/1563 [==============================] - 27s 17ms/step - loss: 20.6234 - accuracy: 0.1007
Epoch 3/5
1563/1563 [==============================] - 28s 18ms/step - loss: 20.5191 - accuracy: 0.1001
Epoch 4/5
1563/1563 [==============================] - 27s 17ms/step - loss: 20.4686 - accuracy: 0.1008
Epoch 5/5
1563/1563 [==============================] - 27s 17ms/step - loss: 20.4933 - accuracy: 0.0992
this is the accuracy of the test data:
313/313 [==============================] - 1s 4ms/step - loss: 20.5000 - accuracy: 0.1000
```

CNN model with padding

If we do padding, the corner pixels get to participate much more in the convolutional process. Padding preserves the image, hence it's also called 'same' convolution.

```
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), padding='same', activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
    layers.MaxPooling2D(),
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 38s 24ms/step - loss: 1.6716 - accuracy: 0.3918
Epoch 2/5
1563/1563 [==============================] - 38s 25ms/step - loss: 1.0870 - accuracy: 0.6163
Epoch 3/5
1563/1563 [==============================] - 39s 25ms/step - loss: 0.9262 - accuracy: 0.6775
Epoch 4/5
1563/1563 [==============================] - 39s 25ms/step - loss: 0.8314 - accuracy: 0.7079
Epoch 5/5
1563/1563 [==============================] - 38s 24ms/step - loss: 0.7632 - accuracy: 0.7342
this is the accuracy of the test data:
313/313 [==============================] - 2s 6ms/step - loss: 0.8541 - accuracy: 0.7113
```

Padding has helped us improve the accuracy even further, so we will be incorporating this change into our code.

CNN model with dropout layers

Dropout regularization helps to tackle the overfitting problem in deep learning. After doing dropout, the model can't now rely on one input as it might be dropped out at random. Neurons will not learn redundant details of inputs.

```
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), padding='same', activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
    layers.MaxPooling2D(),
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 42s 26ms/step - loss: 1.7803 - accuracy: 0.3545
Epoch 2/5
1563/1563 [==============================] - 42s 27ms/step - loss: 1.1601 - accuracy: 0.5844
Epoch 3/5
1563/1563 [==============================] - 42s 27ms/step - loss: 0.9799 - accuracy: 0.6598
Epoch 4/5
1563/1563 [==============================] - 41s 26ms/step - loss: 0.8945 - accuracy: 0.6868
Epoch 5/5
1563/1563 [==============================] - 41s 26ms/step - loss: 0.8330 - accuracy: 0.7078
this is the accuracy of the test data:
313/313 [==============================] - 2s 6ms/step - loss: 0.9195 - accuracy: 0.6856
```

Dropping the neurons is not helping us improve the accuracy in this case. So we will drop the idea of Dropout and move ahead with further investigations and methods to improve the accuracy.

Increasing the number of filters and pooling (2, 2)

Pooling is used to reduce the size of the feature map. This helps in reducing the computations. It also helps to reduce overfitting as there are less parameters. The model also becomes more tolerant towards variation and distortion.

```
model = keras.Sequential([
    layers.Conv2D(64, (3, 3), padding='valid',
activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3, 3), padding='valid',
activation='relu'),
    layers.MaxPooling2D((2,2)),

    # dense network layer
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='sigmoid')
    ])
```

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

Output

```
Epoch 1/5
1563/1563 [==============================] - 59s 38ms/step - loss: 1.6638 - accuracy: 0.3982
Epoch 2/5
1563/1563 [==============================] - 59s 38ms/step - loss: 1.0966 - accuracy: 0.6173
Epoch 3/5
1563/1563 [==============================] - 61s 39ms/step - loss: 0.9423 - accuracy: 0.6736
Epoch 4/5
1563/1563 [==============================] - 56s 36ms/step - loss: 0.8435 - accuracy: 0.7058
Epoch 5/5
1563/1563 [==============================] - 56s 36ms/step - loss: 0.7813 - accuracy: 0.7292
this is the accuracy of the test data:
313/313 [==============================] - 3s 9ms/step - loss: 0.9168 - accuracy: 0.6891
```

Just by increasing the filter size we can increase the accuracy, theoretically more the filters more the accuracy but the complexity increases. So we will move ahead with the new filter size in our experiments. (2, 2) pooling didn't help us improve the accuracy.

Data augmentation

CNN by itself doesn't take care of rotation and scaling. Data augmentation generates new rotated and scaled samples from existing training samples. That means, more the training samples better will be the learning process.

```
data_aug = keras.Sequential([
    layers.experimental.preprocessing.RandomZoom(0.2, 0.3,
input_shape=(32, 32, 3)),
    layers.experimental.preprocessing.RandomFlip('vertical')
])
```

```
model = keras.Sequential([
    data_aug,
    layers.Conv2D(32, (3, 3), padding='same', activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
    layers.MaxPooling2D(),

    # dense network layer
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='sigmoid')
    ])
```

Output

```
Epoch 1/5
1563/1563 [==============================] - 44s 27ms/step - loss: 1.9024 - accuracy: 0.2944
Epoch 2/5
1563/1563 [==============================] - 44s 28ms/step - loss: 1.4778 - accuracy: 0.4706
Epoch 3/5
1563/1563 [==============================] - 47s 30ms/step - loss: 1.3214 - accuracy: 0.5266
Epoch 4/5
1563/1563 [==============================] - 45s 29ms/step - loss: 1.2319 - accuracy: 0.5645
Epoch 5/5
1563/1563 [==============================] - 44s 28ms/step - loss: 1.1832 - accuracy: 0.5822
this is the accuracy of the test data:
313/313 [==============================] - 2s 6ms/step - loss: 1.1396 - accuracy: 0.6002
```

In our case data augmentation didn’t help us improve the accuracy

REAL WORLD APPLICATION

The application of out code is object detection using the Yolo algorithm. We decided to choose Yolo algorithm because it's the state of the art algorithm to perform object detection. It's way faster than the other approaches(R CNN, FAST R CNN, FASTER R CNN) because it does detection only in one single iteration thus saving us so much time and computational power. The application range of this in our living world using Yolo algorithm for object detection is huge.

Because of the speed of this algorithm it can be used to detect objects in a video with up to 50-60 FPS(the latest version of Yolo), the scope for this varies to real world applications such as autonomous cars, military applications, airport security…….

```
import cv2
import numpy as np

# loading the yolo algorithm
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

classes = np.array(open('coco.names.txt').read().splitlines())

layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3)) #
different colour boxes for different detected objects

# loading the image
img = cv2.imread("truck_photo.jpg")
img = cv2.resize(img, None, fx=0.1, fy=0.1)
height, width, channels = img.shape

# detecting the objects present in the image
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0,
0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layers) # contains the information of
the detected objects
```

```
# Showing the information of the detected objects on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            # Object detected
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# using non max supression incase the algorithm detects two or
more classes for the same object
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

font = cv2.FONT_HERSHEY_SIMPLEX
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = colors[i]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y + 30), font, 1, color, 3)


cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:

Output: