Screenshot%202021-02-04%20at%2010.30.29%20am.png

Convolutional Neural Networks (CNN) - Object Recognition

Imports

```
from numpy.random import seed
seed(888)

#from tensorflow import set_random_seed
#set_random_seed(4112)
import tensorflow
tensorflow.random.set_seed(112)
```

```
import os
import numpy as np
import itertools

import tensorflow as tf
import keras
from keras.datasets import cifar10 # importing the dataset

from keras.models import Sequential       #to define model/ layers
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten

from sklearn.metrics import confusion_matrix

# To Explore the images
from IPython.display import display
from keras.preprocessing.image import array_to_img

from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
%matplotlib inline
```

```
import pandas as pd
```

We are using Tensorflow to power Keras

Get the Dataset

CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The dataset is popularly used to train image classification models

Screenshot%202021-02-05%20at%2012.47.25%20pm.png

```
# Getting the dataset as a Tuple

(x_train_all, y_train_all), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

Constants

```
LABEL_NAMES = ['airplane', 'automobile','bird','cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Exploring the Data

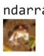
Lets look at the first image in the dataset

```
x_train_all.shape
```

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

```
x_train_all[0]
```

```
ndarray (32, 32, 3) show data
```



```
x_train_all[0].shape
```

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

Using ipython to display the image

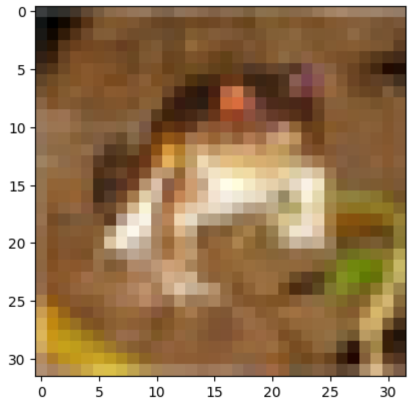
```
# To use the ipython display to view an image

pic = array_to_img(x_train_all[0])
display(pic)
```

Using Matplotlib to view the image

```
plt.imshow(x_train_all[0])
```

```
<matplotlib.image.AxesImage at 0x7ca236f69390>
```




```
# To check the label
y_train_all.shape
```

```
(50000, 1)
```

```
# Note that in the image above the index 1 corresponds to "Automobile"
# we have a 2 dimension numpy array; that is why we also include " [0] "

y_train_all[0][0]
```

```
6
```

```
# Using the lable names to get the actual names of classes

LABEL_NAMES[y_train_all[0][0]]
```

```
'frog'
```

The shape of the image

```
* 32, 32 is the weight and the height
* 3 is the number of channels (These are the number of colors): Red, Green & Blue (RGB)
```

- x_train_all.shape >>> (50000, 32, 32, 3)
    - this means we have 50,000 entries | then 32x32 weight and height| 3 colors (RGB)

```
x_train_all.shape
```

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

```
number_of_images, x, y, c = x_train_all.shape
print(f'Number of images = {number_of_images} \t| width = {x} \t| height = {y} \t| channels = {c}')
```

```
Number of images = 50000        | width = 32    | height = 32   | channels = 3
```

```
x_test.shape
```

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

Preprocess Data

* We need to preprocess our data so that it is easier to feed it to our neural network.

Scalling both x_train and test

```
x_train_all =x_train_all / 255.0
```

```
x_test =  x_test / 255.0
```

```
y_test
```

```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

Creating categorical encoding for the "y " data

```
# 10 >>> simply means we have 10 classes like we already know (creating the encoding for 10 classes)
y_cat_train_all = to_categorical(y_train_all,10)
```

```
# 10 >>> simply means we have 10 classes like we already know (creating the encoding for 10 classes)
y_cat_test = to_categorical(y_test,10)
```

```
y_cat_train_all
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]])
```

Creating the Validation dataset

Screenshot%202021-02-05%20at%206.37.47%20pm.png

For small data we usually go with: * 60% for Training * 20% Validation * 20% Testing

Only the final selected model gets to see the testing data. This helps us to ensure that we have close to real data in real-world when the model is deployed. Only our best model gets to see our testing dataset. Because it will give us a realistic impression of how our model will do in the real world

---

However, if the dataset is enormous.: * 1% for is used for validation * 1% for is used for testing

```
VALIDATION_SIZE = 10000
```

```
# VALIDATION_SIZE = 10,000 as defined above

x_val = x_train_all[:VALIDATION_SIZE]
y_val_cat = y_cat_train_all[:VALIDATION_SIZE]
x_val.shape
```

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

```
y_val_cat
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

NEXT:

- We Create two NumPy arrays x_train and y_train that have the shape(40000, 3072) and (40000,1) respectively.
- They will contain the last 40000 values from x_train_all and y_train_all respectively

```
x_train = x_train_all[VALIDATION_SIZE:]
y_cat_train= y_cat_train_all[VALIDATION_SIZE:]
```

```
x_train.shape
```

```
(40000, 32, 32, 3)
```

```
y_cat_train
```

```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]])
```

Screenshot%202021-02-15%20at%208.04.47%20pm.png

NOTE:

* *FILTERS:* Typical values for the number of filters can be determined by the data set's complexity. So essentially the larger the images, the more variety and the more classes you're trying to classify then the more filters you should have.

* Most times people typically pick filter based on powers of 2, for example, 32. However, if you have more complex data like road signs etc. you should be starting with a higher filter value

The default STRIDE value is 1 x 1 pixel

BUILDING THE MODEL

```
model = Sequential()

## ************* FIRST SET OF LAYERS *************************

# CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

## *************** SECOND SET OF LAYERS ***********************
#Since the shape of the data is 32 x 32 x 3 =3072 ...
```

```
#We need to deal with this more complex structure by adding yet another convolutional layer

# *************CONVOLUTIONAL LAYER
model.add(Conv2D(filters=32, kernel_size=(4,4),input_shape=(32, 32, 3), activation='relu',))
# POOLING LAYER
model.add(MaxPool2D(pool_size=(2, 2)))

# FLATTEN IMAGES FROM 32 x 32 x 3 =3072 BEFORE FINAL LAYER
model.add(Flatten())

# 256 NEURONS IN DENSE HIDDEN LAYER (YOU CAN CHANGE THIS NUMBER OF NEURONS)
model.add(Dense(256, activation='relu'))

# LAST LAYER IS THE CLASSIFIER, THUS 10 POSSIBLE CLASSES
model.add(Dense(10, activation='softmax'))


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

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seq
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 29, 29, 32)	1,568
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 32)	16,416
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 256)	205,056
dense_1 (Dense)	(None, 10)	2,570

Total params: 225,610 (881.29 KB)
Trainable params: 225,610 (881.29 KB)
Non-trainable params: 0 (0.00 B)

Adding Early stopping

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

```
history = model.fit(x_train,y_cat_train,epochs=25,validation_data=(x_val,y_val_cat),callbacks=[early_stop])
```

```
Epoch 1/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 52s 40ms/step - accuracy: 0.3606 - loss: 1.7441 - val_accuracy: 0.5166 - val_loss: 1.3169
Epoch 2/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 88s 46ms/step - accuracy: 0.5323 - loss: 1.3125 - val_accuracy: 0.5767 - val_loss: 1.1837
Epoch 3/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 77s 42ms/step - accuracy: 0.5946 - loss: 1.1536 - val_accuracy: 0.6198 - val_loss: 1.0809
Epoch 4/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 82s 42ms/step - accuracy: 0.6408 - loss: 1.0262 - val_accuracy: 0.6432 - val_loss: 1.0226
Epoch 5/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 52s 42ms/step - accuracy: 0.6767 - loss: 0.9197 - val_accuracy: 0.6559 - val_loss: 0.9956
Epoch 6/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 80s 41ms/step - accuracy: 0.7093 - loss: 0.8336 - val_accuracy: 0.6585 - val_loss: 1.0004
Epoch 7/25
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 84s 42ms/step - accuracy: 0.7351 - loss: 0.7583 - val_accuracy: 0.6468 - val_loss: 1.0786
```

```
model.history.history.keys()
```

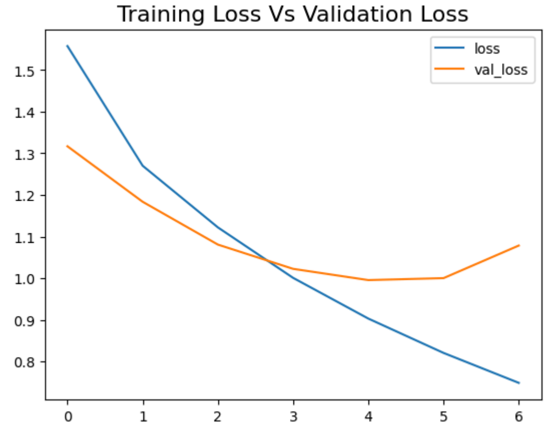
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
metrics = pd.DataFrame(model.history.history)
```

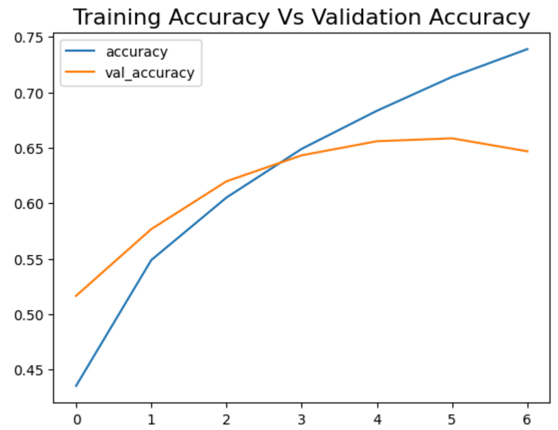
```
metrics
```

	accuracy	loss	val_accuracy	val_loss
0	0.435425	1.557476	0.5166	1.316910
1	0.548825	1.270180	0.5767	1.183667
2	0.605175	1.122172	0.6198	1.080883
3	0.649050	1.000994	0.6432	1.022645
4	0.683375	0.903220	0.6559	0.995638
5	0.713950	0.820788	0.6585	1.000362
6	0.738925	0.748903	0.6468	1.078554

```
metrics[['loss', 'val_loss']].plot()
plt.title('Training Loss Vs Validation Loss', fontsize=16)
plt.show()
```




```
metrics[['accuracy', 'val_accuracy']].plot()
plt.title('Training Accuracy Vs Validation Accuracy', fontsize=16)
plt.show()
```



Validating on Test Data

```
model.evaluate(x_test,y_cat_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.6410 - loss: 1.1109
[1.113039493560791, 0.6410999894142151]
```

Classification Report and Confusion Matrix

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
#predictions = model.predict_classes(x_test)
predictions = np.argmax(model.predict(x_test), axis=-1)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.68      0.73      0.70      1000
           1       0.79      0.75      0.77      1000
           2       0.58      0.45      0.51      1000
           3       0.37      0.67      0.47      1000
           4       0.71      0.46      0.55      1000
           5       0.62      0.43      0.51      1000
           6       0.69      0.74      0.71      1000
           7       0.71      0.72      0.72      1000
           8       0.77      0.76      0.76      1000
           9       0.74      0.71      0.73      1000

    accuracy                           0.64     10000
   macro avg       0.66      0.64      0.64     10000
weighted avg       0.66      0.64      0.64     10000
```

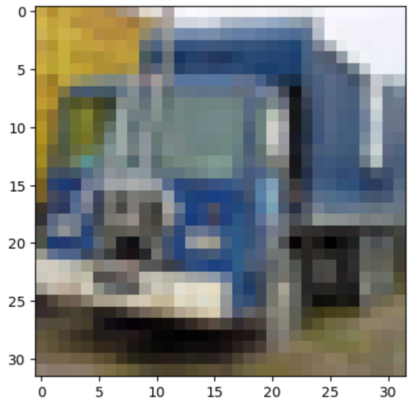
```
confusion_matrix(y_test,predictions)
```

```
array([[728,  19,  45,  39,  12,   5,  17,  10,  89,  36],
       [ 31, 747,   9,  33,   0,   9,  18,   6,  49,  98],
       [ 93,  11, 450, 174,  61,  43, 107,  31,  17,  13],
       [ 24,   7,  45, 668,  35,  94,  56,  39,   8,  24],
       [ 28,   3,  93, 189, 457,  27,  89,  95,  14,   5],
       [ 23,   5,  47, 362,  24, 428,  20,  71,   8,  12],
       [  7,   6,  28, 153,  21,  20, 742,  10,   3,  10],
       [ 21,   4,  34, 109,  32,  49,   4, 722,   7,  18],
       [ 88,  42,  11,  43,   4,  10,   7,   4, 756,  35],
       [ 31, 106,  12,  48,   1,   9,  20,  23,  37, 713]])
```

Predicting on single image

```
plt.imshow(x_test[14])
```

```
<matplotlib.image.AxesImage at 0x7ca236f12f20>
```




```
my_image = x_test[14]
```

```
# SHAPE --> (num_images,width,height,color_channels)
predictions = np.argmax(model.predict(my_image.reshape(1,32,32,3)),axis=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
```

```
LABEL_NAMES[y_test[14][0]]
```

```
'truck'
```

```
Comece a programar ou gerar com a IA.
```