



PROJECT “CNN”



Team Members

- Omar Hesham [1170065]
- Youssef Salah Mohamed [1170191]

Submitted to:

- Dr. Hossam Abdelmoniem
- Eng. Peter Salah

Table of Contents

Table of Contents	1
List of Figures.....	2
I. Introduction	3
Problem Definition:	3
Importance of CNN:	3
II. Methods and Algorithms.....	4
A. Data Preprocessing:	4
B. Uploading Data:	4
C. Extracting Features:	4
D. Splitting Data:	4
E. Model Architecture:.....	5
1. Feature learning:	5
2. Classification:.....	6
F. Compiling/Fitting model:	6
III. Results and Discussion	7
➤ Testing our model results:	7
➤ Testing AlexNet model results:	7
➤ Training time:.....	8
IV. Appendix with code	9
➤ Shots of code from colab:	9
▪ Imports	9
▪ Mounting Drive:.....	9
▪ Getting features and creating excel sheet:.....	10
▪ Saving data:.....	10
▪ Loading data:	10
▪ Splitting data:	10
▪ Building our model	11
▪ Compile and fitting our model:	11
▪ Evaluating our model:	11
▪ Building Alexnet model	12
▪ Alex Model cont.....	13
▪ Compile and fitting Alextnet model.....	13
▪ Evaluating Alextnet model.....	13
▪ Testing our model with internet images of trained classes	14

List of Figures

Figure 1 How CNN works.....	3
Figure 2 Model Architecture	5
Figure 3 32 vs 64 nodes	5
Figure 4 Pooling types	6
Figure 5 Our model results.....	7
Figure 6 Alexnet Model results	7
Figure 7 Bronotaus & Brain Test	8
Figure 8 Butterfly & Cannon Test.....	8
Figure 9 Binocular Test.....	8
Figure 10 Bonasi Test	8
Figure 12 Beaver & Bass Test	9
Figure 11 Barrel & Anchor Test	9

I. Introduction

Problem Definition:

Given a dataset containing images with different objects “about 11 different classes”. We want to design and implement Convolution Neural Network classifier which its role is to classify each image to its class.

Convolution Neural Network classifier efficiency mainly depends on the dataset quality, dataset quality depends on (image size, image brightness, number of images in each class compared to the other classes). Given a high variation in number of images between the classes and each other's, this leads to make the ones of larger numbers be dominated than the others of less number of images. So the first step is to make data preprocessing on the given dataset.

Importance of CNN:

- It becomes one of the most appealing approaches recently and has been an ultimate factor in a variety of recent success and challenging applications related to machine learning applications such as challenge ImageNet object detection, image classification, and face recognition. Therefore, we consider CNN as our model for our challenging problem of image classification
- Usage of CNN for segmentation and classification of the images in academic and business transactions. We use image recognition in different areas for example automated image organization, stock photography, face recognition, and many other related works.

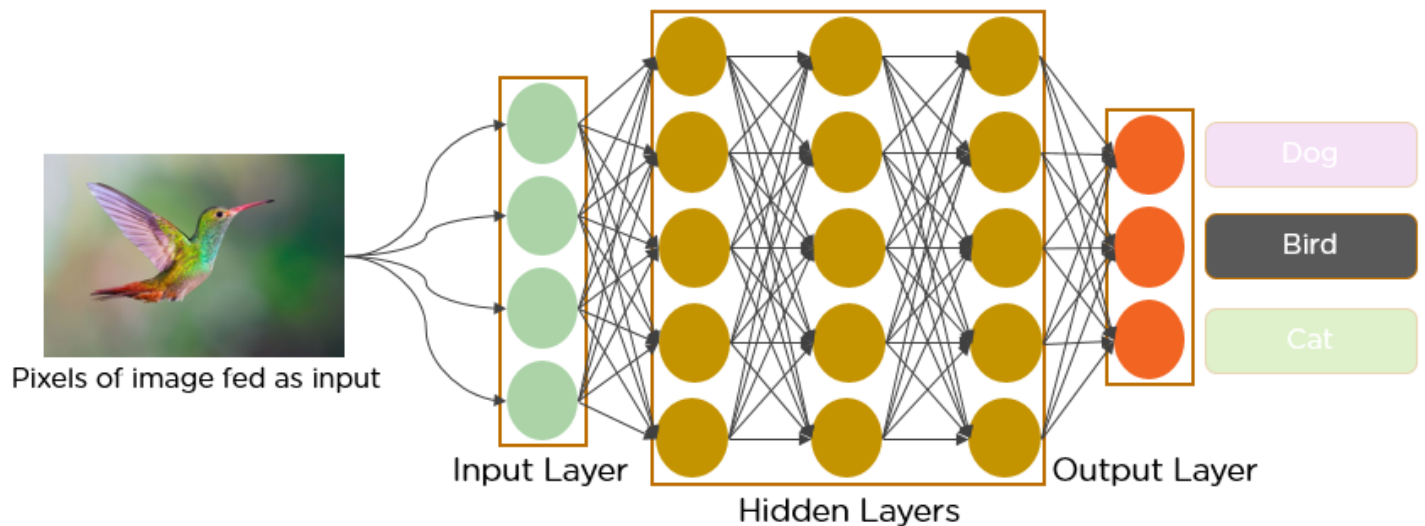


Figure 1 How CNN works

II. Methods and Algorithms

A. Data Preprocessing:

- we download dataset from http://www.vision.caltech.edu/Image_Datasets/Caltech101/
- We have 11 classes “11 different objects” but every class has different number of images than the other also some classes have small number of images, this variation/small number of images may lead in a not well-trained classifier
- So we had to do image augmentation to decrease this variation and make variation in number of images decrease, this results in homogenous dataset.
- After we did image augmentation, we moved to next step.

B. Uploading Data:

- Next step was uploading this dataset on google drive to extract features of images from it.

C. Extracting Features:

- We created an excel file to save labels of each image (11 labels for 11 classes) by looping on each folder as extracting features step.
- We did resize operation on each image to 100x100x3 to decrease computational power and memory consumption since we compared it with 300x300x3 results and we noticed that there is no significant difference in models accuracy.
- Both features and labels are saved in numpy array named X_data, Y_data respectively.
- We saved X_data, Y_data to load them without loading dataset again but before doing normalization step to save memory.
- After saving step we move to normalization step (dividing each value by 255).

D. Splitting Data:

- Here comes the step of splitting data into test, validate and train with ratios 15,15 and 70% respectively.

E. Model Architecture:

- Now it's time to build our model.
- Firstly we'll start with explaining model architecture

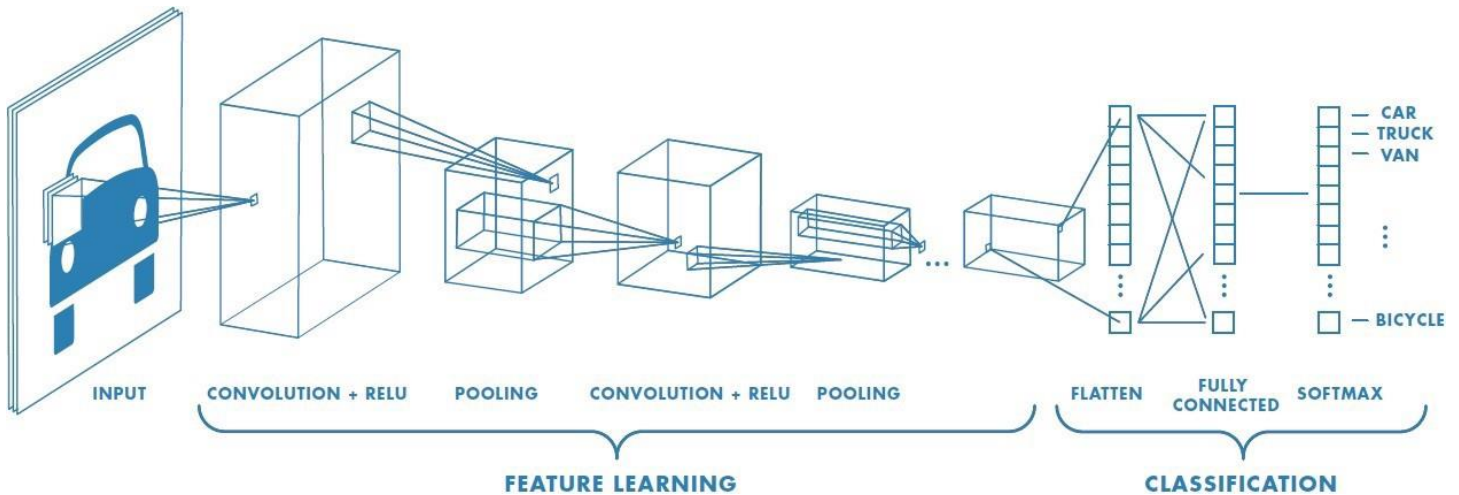


Figure 2 Model Architecture

1. Feature learning:

- It's by adding number of layers
- Firstly, we are using a sequential model
- We add convolution layer with 32 nodes and kernel size (3x3), padding "the same" since the size of image is already small, and activation 'relu', kernel_initializer 'he_uniform' and specify the the input size (100x100x3).
- Then we apply batch normalization since it makes the landscape of the corresponding optimization problem be significantly more smooth. This ensures, in particular, that the gradients are more predictive and thus allow for use of larger range of learning rates and faster network convergence. it mainly works by calculating the mean and standard deviation of each input variable to a layer per mini-batch and using these statistics to perform the standardization.
- Then we add another convolution layer with 32 nodes again and another batch normalization is done on this layer.
- Now it's time to add pooling layer which is responsible for reducing spatial size of the convolved feature which results in decreasing computational power required to process the data.

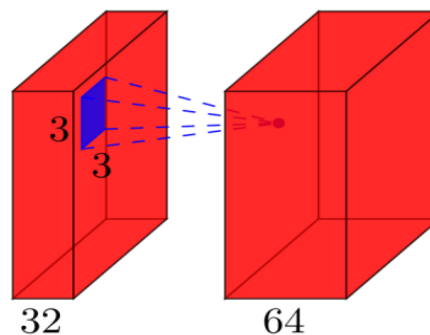


Figure 3 32 vs 64 nodes

- Pooling layer is also responsible for extracting dominant features , thus maintaining the process of effectively training of the model.
- Here comes a question which type of pooling do we use? We chose Max pooling instead of average pooling as max pooling acts as a noise suppressant. It discards the noisy activations altogether and performs de-noising along dimensionality reduction. On the other hand, Average pooling simply performs dimensionality reduction as a noise surpassing so we can say that Max pooling performs a lot better than average pooling.

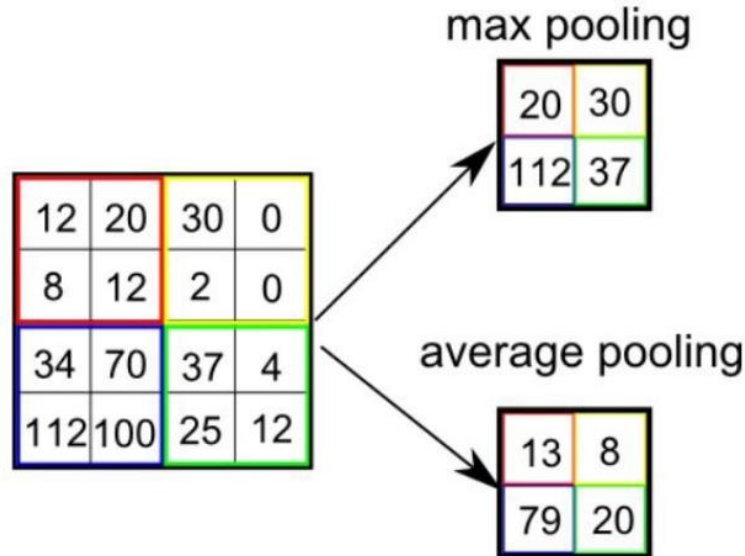


Figure 4 Pooling types

- We found that accuracy wasn't good enough so we repeated the previous step one more time but with 64 nodes.
- To avoid overfitting we used dropout function with 20%

2. Classification:

- To move to classification step , we flattened the final output and fed it into a regular neural network for classification purposes.
-
- With a classifier of dense 128 nodes and 11 output nodes for the 11 classes.
- We used softmax activation function to distinguish between dominating and certain low-level features in images and classify them.

F. Compiling/Fitting model:

- Using SGD function with learning rate 0.001 and momentum 0.9 as an optimizer and using Categorical crossentropy as loss function from keras library since adam optimizer didn't result in high accuracy.
- Finally we fit model with training data with 30 epochs and batch size 64.

III. Results and Discussion

➤ Testing our model results:

- A. After 40 epochs in fitting our model we get results of loss: 4.9627e-05 and accuracy:100% in training set. And loss: 0.5056 and accuracy:95.23% in the validation set.
- B. Then we evaluate the test set and get loss: 0.3379 and accuracy: 94.02%

```
2343/2343 [=====] - 2s 747us/sample - loss: 5.7154e-05 - acc: 1.0000 - val_loss: 0.4997 - val_acc: 0.9523
Epoch 29/30
2343/2343 [=====] - 2s 745us/sample - loss: 5.3199e-05 - acc: 1.0000 - val_loss: 0.5023 - val_acc: 0.9523
Epoch 30/30
2343/2343 [=====] - 2s 745us/sample - loss: 4.9627e-05 - acc: 1.0000 - val_loss: 0.5056 - val_acc: 0.9523
```

```
[ ] ## evaluate results with test data ( X, Y)
    results = model.evaluate(X_test, y_test, batch_size=64)
    print("test loss, test acc:", results)
```

```
test loss, test acc: [0.3379521191832554, 0.9402391]
```

Figure 5 Our model results

➤ Testing AlexNet model results:

Then we searched for open source code that do CNN we found AlexNet and its results was

- A. After 40 epochs in fitting our model we get results of loss: 0.0295 and accuracy:98.8% in training set. And loss: 0.3031 and accuracy:92.84% in the validation set,
- B. Then we evaluate the test set and get loss: 0.370 and accuracy: 93.02%

We found that our model has higher accuracy than AlexNet model with nearly 1% in the test set and 2.2% in the validation accuracy

To check our model with different data we download an image to each class from internet and uploaded it in drive then started to predict each one and showing them all with their results. We found that 10 from the 11 class predict right and only one failed which is nearly equal 91% accuracy

```
Epoch 29/30
2343/2343 [=====] - 2s 990us/sample - loss: 0.0211 - acc: 0.9915 - val_loss: 0.5120 - val_acc: 0.9125
Epoch 30/30
2343/2343 [=====] - 2s 989us/sample - loss: 0.0295 - acc: 0.9880 - val_loss: 0.3031 - val_acc: 0.9284
```

```
[ ] ## evaluate results with test data ( X, Y)
    results = model_Alex.evaluate(X_test, y_test, batch_size=64)
    print("test loss, test acc:", results)
```

```
test loss, test acc: [0.37000213462518033, 0.9302789]
```

Figure 6 Alexnet Model results

➤ Training time:

We running our code on colab GPU since it is faster than none or even our CPU.
It took 1min and 3 sec in training with 40 epochs.
In building model and compiling it didn't take a time nearly 1 sec.

➤ Testing images from internet:

Class: [8] ****butterfly****



/content/drive/MyDrive/pattern/O
Class: [9] ****cannon****

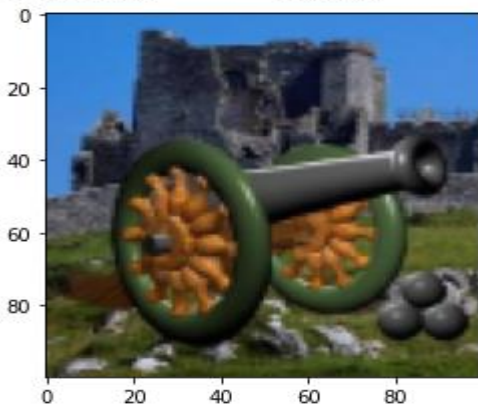


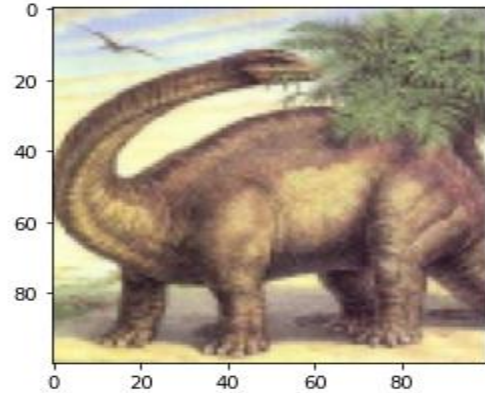
Figure 8 Butterfly & Cannon Test

Class: [5] ****bonasi****



Figure 10 Bonasi Test

Class: [7] ****bronotaus****



/content/drive/MyDrive/pattern/O
Class: [6] ****brain****

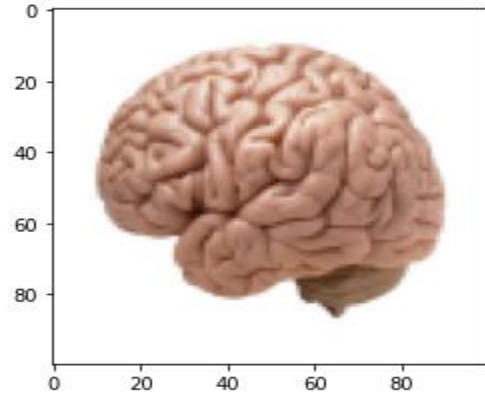


Figure 7 Bronotaus & Brain Test

Class: [5] ****bonasi****

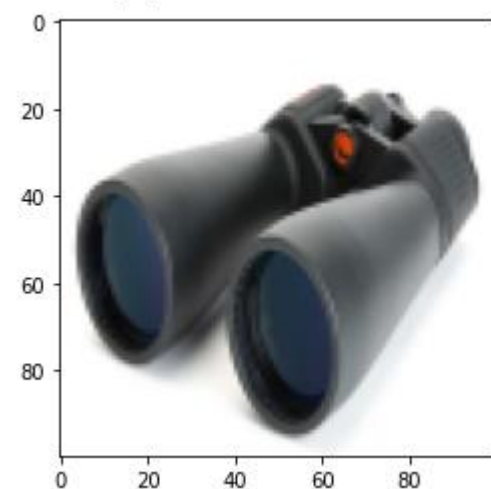


Figure 9 Binocular Test

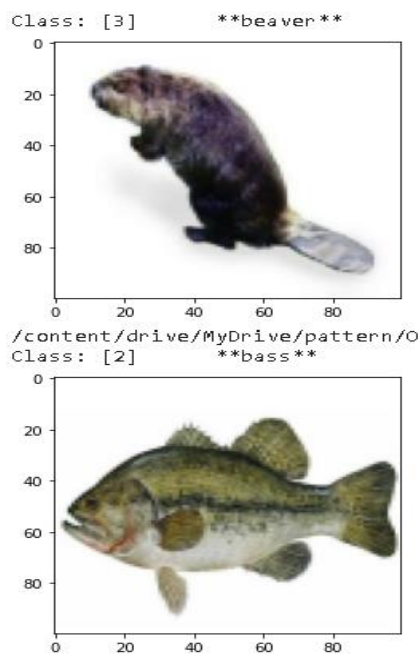


Figure 11 Beaver & Bass Test

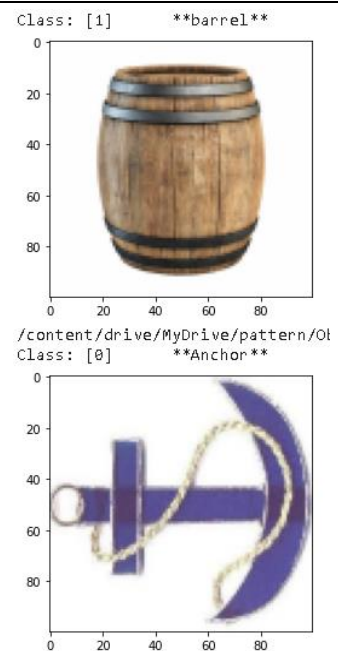


Figure 12 Barrel & Anchor Test

IV. Appendix with code

➤ Shots of code from colab:

■ Imports:

```
## IMPORTS
import xlwt , xlrd
from xlwt import Workbook
import os
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras import datasets, layers, models
from keras.layers import Activation, Dense
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras import optimizers
####
import tensorflow as tf

# from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
# from tensorflow.python.framework import ops
```

■ Mounting Drive:

```
## Connect to Drive
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

- Getting features and creating excel sheet:

```
wb = Workbook()

# Add_sheet database.
sheet1 = wb.add_sheet('database')
# Drive link
rootdir = r'/content/drive/MyDrive/pattern/dataset'
# Excel headers
sheet1.write(0, 0, 'ID')
sheet1.write(0, 1, 'CLASS')
i=1
val=0
IMG_SIZE=100
imgs=[]
for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        ## Take the path of the image
        img_path=(os.path.join(subdir, file))
        img=cv2.imread(img_path)
        img = cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), (IMG_SIZE,IMG_SIZE), interpolation = cv2.INTER_AREA)
        imgs.append(img)
        ## Append the class in the excel file
        sheet1.write(i, 0, i)
        sheet1.write(i, 1, val-1)
        i=i+1
        val=val+1

wb.save('database.xls')
```

- Saving data:

```
## Save the data to load it after first run
np.save('X_data1',X_data)
np.save('Y_data1',Y_data)
```

- Loading data:

```
## Load data
X_data = np.load(r'/content/drive/MyDrive/pattern/X_data1.npy')
Y_data = np.load(r'/content/drive/MyDrive/pattern/Y_data1.npy')
```

- Splitting data:


```
## Divide the Data ( 70% train , 15% validation, 15% test) in shuffle
X_train, X_test, y_train, y_test = train_test_split(X_data,Y_data, test_size=0.3, shuffle=True)
X_test, X_validate, y_test,y_validate = train_test_split(X_test,y_test, test_size=0.5, shuffle=True)
```

▪ Building our model

```
[ ] ## Use a Sequential model with (two 32Conv2d-layers)and their normalization and maxpooling then (two 64Conv2d-layers)and their normalizati
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(100, 100, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))

#####
model.summary()
```


 # Flatten the model and add 128 dense with 11 classes and activation function "softmax"

```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(layers.Dense(11))
model.add(Activation("softmax"))
#####
model.summary()
```

▪ Compile and fitting our model:

```
## using SGD function with learning rate 0.001 and momentum 0.9 as an optimizer and using Categoricalcrossentropy
## fit the model with 30 epoches and batch size 64
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt,
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=30, batch_size=64, validation_data=(X_validate, y_validate))
```

▪ Evaluating our model:

 ## evaluate results with test data (X, Y)

```
results = model.evaluate(X_test, y_test, batch_size=64)
print("test loss, test acc:", results)
```

■ Building Alexnet model

```
▶ ## Use online open source " Alex model " to compare the results
model_Alex = models.Sequential()

# 1st Convolutional Layer
model_Alex.add(layers.Conv2D(filters = 96, input_shape = (100, 100, 3),
                             kernel_size = (11, 11), strides = (4, 4),
                             padding = 'same'))
model_Alex.add(Activation('relu'))
# Max-Pooling
model_Alex.add(layers.MaxPooling2D(pool_size = (2, 2),
                                    strides = (2, 2), padding = 'same'))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())

# 2nd Convolutional Layer
model_Alex.add(layers.Conv2D(filters = 256, kernel_size = (11, 11),
                              strides = (1, 1), padding = 'same'))
model_Alex.add(Activation('relu'))
# Max-Pooling
model_Alex.add(layers.MaxPooling2D(pool_size = (2, 2), strides = (2, 2),
                                    padding = 'same'))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())
```

```
▶ # 3rd Convolutional Layer
model_Alex.add(layers.Conv2D(filters = 384, kernel_size = (3, 3),
                              strides = (1, 1), padding = 'same'))
model_Alex.add(Activation('relu'))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())

# 4th Convolutional Layer
model_Alex.add(layers.Conv2D(filters = 384, kernel_size = (3, 3),
                              strides = (1, 1), padding = 'same'))
model_Alex.add(Activation('relu'))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())

# 5th Convolutional Layer
model_Alex.add(layers.Conv2D(filters = 256, kernel_size = (3, 3),
                              strides = (1, 1), padding = 'same'))
model_Alex.add(Activation('relu'))
# Max-Pooling
model_Alex.add(layers.MaxPooling2D(pool_size = (2, 2), strides = (2, 2),
                                    padding = 'same'))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())

# Flattening
model_Alex.add(layers.Flatten())
```

- Alex Model cont.

```
# 1st layers.Dense Layer
model_Alex.add(layers.Dense(4096, input_shape = (100*100*3, )))
model_Alex.add(Activation('relu'))
# Add Dropout to prevent overfitting
model_Alex.add(layers.Dropout(0.4))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())

# 2nd layers.Dense Layer
model_Alex.add(layers.Dense(4096))
model_Alex.add(Activation('relu'))
# Add Dropout
model_Alex.add(layers.Dropout(0.4))
# Batch Normalisation
model_Alex.add(layers.BatchNormalization())

# Output Softmax Layer
model_Alex.add(layers.Dense(11))
model_Alex.add(Activation('softmax'))
```

- Compile and fitting Alexnet model

```
[ ] ## using SGD function with learning rate 0.001 and momentum 0.9 as an optimizer and using CategoricalCrossentropy as loss function
## fit the model with 30 epoches and batch size 64
opt = SGD(lr=0.001, momentum=0.9)
model_Alex.compile(optimizer=opt,
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])
history = model_Alex.fit(X_train, y_train, epochs=30, batch_size=64, validation_data=(X_validate, y_validate))
```

- Evaluating Alexnet model

```
[ ] ## evaluate results with test data ( X, Y)
results = model_Alex.evaluate(X_test, y_test, batch_size=64)
print("test loss, test acc:", results)
```

- Testing our model with internet images of trained classes

```
[ ] ## Add image from each class downloaded from the internet in Drive. and start testing them
folder = r'/content/drive/MyDrive/pattern/Objects'

for filename in os.listdir(folder):
    img_path = (os.path.join(folder,filename))
    print(img_path)
    img = (cv2.imread(img_path))
    img = cv2.resize(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), (100,100), interpolation = cv2.INTER_AREA)
    X_pred = np.reshape(img, (1,img.shape[0],img.shape[1],3))
    y_pred = model.predict(X_pred)
    y_pred = np.argmax(y_pred, axis = 1)
    if y_pred==0:
        coressponding_class='**Anchor**'
    elif y_pred==1:
        coressponding_class='**barrel**'
    elif y_pred==2:
        coressponding_class='**bass**'
    elif y_pred==3:
        coressponding_class='**beaver**'
    elif y_pred==4:
        coressponding_class='**binocular**'
    elif y_pred==5:
        coressponding_class='**bonasi**'
    elif y_pred==6:
        coressponding_class='**brain**'
    elif y_pred==7:
        coressponding_class='**bronotaus**'
    elif y_pred==8:
        coressponding_class='**butterfly**'
    elif y_pred==9:
        coressponding_class='**cannon**'
    else:
        coressponding_class='**carside**'

plt.imshow(img)
print('Class:',y_pred,' ',coressponding_class)
plt.show()
```

```
##### Classes #####
##      carside      ----->10  ##      binocular      ----->4
##
##      cannon       ----->9   ##      beaver         ----->3
##
##      butterfly    ----->8   ##      bass           ----->2
##
##      bronotaus     ----->7   ##      barrel        ----->1
##
##      brain        ----->6   ##      anchor         ----->0
##
##      bonasi       ----->5
```