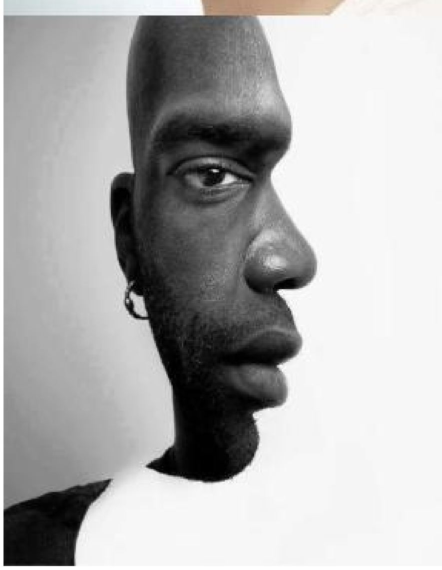
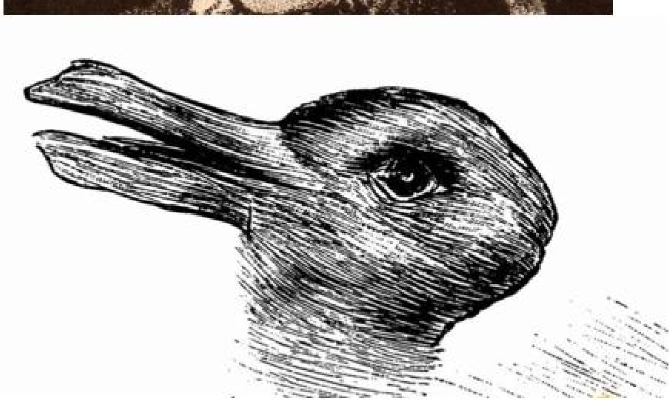
Convolutonal Neural Networks

April 19, 2020

1 Introduction

The human brain processes images based on features. Convolution Neural netoworks are used to process images and produce labels. An image is reprented as a matrix $M_{m \times n}$(B/W as 2D and color as 3D with 3rd dimension be the color channel $\{A, R, G, B\}$), each cell has a value of $[0, 255]$.





4 Steps to build a CNN 1. Convoluton 2. Max Pooling 3. Flattening 4. Full Connection

1.1 Step 1 : Convolution

The convolution is defined as a combined integral of two function and it shows how one function modifies the other, mathematically defined as $(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$. This paper[2] introduces the mathematical foundation about CNN.

Given a binary matrix $M \in \{0,1\}^{m\times n} | m_{i,j} \in M$ and a **Feature detector** matrix $F \in \{0,1\}^{3\times 3}$ (The feature detector is also called, **Kernel** or **Filter**), the convolution is defined as $M \otimes F \ni c_{i,j} = \sum f_{i,j}.m_{i,j}$ i.e. it result a matrix with elements as number of matches by **Striding** the filter over M. Typically the Stride is set to 2. The result is called the **feature map**. The Convolution process basically maps to another matrix which has higher weights to the region which corresponds higher similarity to the feature detector. This resembles the process human brain followes, it looks for features.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 2 & 1 \\ 1 & 4 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \end{bmatrix}$$

In the process there will me a Set of feature detectors (each highlights a specific property) called the Feature space $\mathcal{F} = \{F_i\}$ each feature would generate a feature map, thus a **Convolved Space** is generated. $\mathcal{C} = \{C_i = M \otimes F_i | \forall F_i \in \mathcal{F}\}$. Gimp[3] has a convolution tool, that does convolution using several feature detectors.

1.1.1 Rectified Linear Unit (ReLu) Layer

The rectifier function, as discussed in ANN section is defined as $\phi(x) = max(x, 0)$, increses the non-linearity. Images are natively non-linear in nature, however matrix transformation such as convolution tends to introduces linearity into the transformed on, thus to retain non-linearity in the image, ReLu is used [4]. The paper [5] proposes a parametric ReLu that improves accuracy without sacrificing any performance.

1.2 Step 2 : Max Pooling

Images which are given for training has various orientation, shades etc. which affects the matrix siginificantly. which is tackled by max-pooling, the property is called **spacial invarience**. The pooling is defied as srtiding a maxtrix $P_{2\times 2}$ over the feature map and record a statistics from the feature map. If he stat is Max then it's called max-pooling, other varient is **Min-pooling** or sub-sampling, **Average-pooling** [6]. A nice visualization of convolution process can be found here [7]

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 2 & 1 \\ 1 & 4 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 4 & 2 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

The pooling process contributes the following, 1. Maintains special invarience by keeping the features 2. Reduces the size which helps, preventing Over-fitting

1.3 Step 3 : Fattening

In this process a pooled feature map is converted into a vector, which will be used as the input layer of the neural network, the complete process is given below

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 2 & 1 \\ 1 & 4 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 4 & 2 & 0 \\ 0 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 0 \\ 4 \\ 2 \\ 0 \\ 0 \\ 2 \\ 1 \end{bmatrix}$$

1.4 Step 4 : Full Connection

The flattend vector is fed into the ANN, the hidden layers are ment to be Fully-connected (Dense). The ANN will use the features to learn relationship between them and corelates to the corresponsing label given to them. The output layer would have neuron equals to number of classes. During traing process, the hidden neuron gets trained and extracts features.

1.5 Softmax & Cross-Entropy Function

The output layer of an ANN is a measure of probabilstic approximation, for an n-class classification problem the output layer consists of n neurons, the sum of thei synaptic-wights is always 1. Now, how is it possible, to have values assigned to them without any mutual connection?, the answer is the neurons dont know this, the values are normalised by a function called **Softmax** [8] or Normalised-Exponential function. Mathematically its is a *generalization of logistic functio that squashes a vector* $X \in \mathbb{R}^k$ *of arbitrary values into a vector of same dimension with values within* $[0,1]$ *such that* $\sum X = 1$ Mathematiacally, softmax function is expressed as, $f_i(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$

The **Cross-enropy** function [9] is written as, $L_i = -log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$, $H(p,q) = -\sum_x p(x)log\Big(q(x)\Big)$ is a cost function like **MSE**, but preferred for CNN after using *softmax* which is called loss function in this context. After classifying a dog's picture, applying Softmax, assume that the two output neurons (dog, cat) become $q(x) = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$ the actual label was $p(x) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ the loss is $H(p,q)$

The follwing gives a concrete understanding of the usage of cross-entropy with an example.

[4]:
```
import pandas as pd

# two tables of classification result
```

```python
# using two NN are given
nn1 = pd.read_csv('cat_dog_NN1.csv')
nn2 = pd.read_csv('cat_dog_NN2.csv')
```

[5]:
```python
nn1 # result from ANN 1
```

[5]:

	row	dog_pred	cat_pred	dog	cat
0	1	0.9	0.1	1	0
1	2	0.1	0.9	0	1
2	3	0.4	0.6	1	0

[115]:
```python
nn2 # result from ANN 2
```

[115]:

	row	dog_pred	cat_pred	dog	cat
0	1	0.6	0.4	1	0
1	2	0.3	0.7	0	1
2	3	0.1	0.9	1	0

[146]:
```python
def cce(x,y):   #classificaion error, takes two lists
    sum = 0
    for i in range(len(x)):
        sum += round(x[i]) ^ y[i] #rounding [0,1] to {0,1}
    return round(sum/len(x),3)

def mse(x,y):  # MSE
    sum = 0
    for i in range(len(x)):
        sum += (x[i] - y[i])**2
    return round(sum/len(x),3)

def cef(x,y):  # Cross-entropy
    sum = 0
    import math
    for i in range(len(x)):
        sum += y[i]* math.log10(x[i])
    return round(-1*sum , 3)
```

[148]:
```python
dp1 = nn1['dog_pred'].values.tolist()
d1 = nn1['dog'].values.tolist()
cp1 = nn1['cat_pred'].values.tolist()
c1 = nn1['cat'].values.tolist()

dp2 = nn2['dog_pred'].values.tolist()
d2 = nn2['dog'].values.tolist()
cp2 = nn2['cat_pred'].values.tolist()
c2 = nn2['cat'].values.tolist()
```

```
nn1_cce = (cce(dp1,d1)+cce(cp1,c1))/2
nn2_cce = (cce(dp2,d2)+cce(cp2,c2))/2

nn1_mse = (mse(dp1,d1)+mse(cp1,c1))
nn2_mse = (mse(dp2,d2)+mse(cp2,c2))

nn1_cef = (cef(dp1,d1)+cef(cp1,c1))
nn2_cef = (cef(dp2,d2)+cef(cp2,c2))
```

[150]:
```
import math
print(f"Calssificatin error : NN1 = {nn1_cce}  \
      NN2 = {nn2_cce} \
      def = {round(math.sqrt((nn1_cce - nn2_cce)**2),3)}")

print(f"Mean Squared error  : NN1 = {nn1_mse}  \
      NN2 = {nn2_mse} \
      def = {round(math.sqrt((nn1_mse - nn2_mse)**2),3)}")

print(f"Cross-entropy error : NN1 = {nn1_cef}  \
       NN2 = {nn2_cef} \
      def = {round(math.sqrt((nn1_cef - nn2_cef)**2),3)}")
```

```
calssificatin error : NN1 = 0.333        NN2 = 0.333       def = 0.0
Mean Squared error  : NN1 = 0.254        NN2 = 0.706       def = 0.452
Cross-entropy error : NN1 = 0.49         NN2 = 1.377       def = 0.887
```

As it shows in the result, Cross-entropy distinguishes between the two resutls better than others, this happens due to its lograrithmic nature which can capture changes in a very small scale. However Cross-entropy is recomanded for classification problems only, for regression problems MSE is preffered.

2 References

1. https://www.researchgate.net/publication/2985446_Gradient-Based_Learning_Applied_to_Document_Recognition
2. https://pdfs.semanticscholar.org/450c/a19932fcef1ca6d0442cbf52fec38fb9d1e5.pdf?_ga=2.127701316.833949 358525363.1587235878
3. https://docs.gimp.org/2.6/en/plug-in-convmatrix.html
4. https://arxiv.org/pdf/1609.04112.pdf
5. https://arxiv.org/pdf/1502.01852.pdf
6. http://ais.uni-bonn.de/papers/icann2010_maxpool.pdf
7. https://www.cs.ryerson.ca/~aharley/vis/conv/
8. https://peterroelants.github.io/posts/cross-entropy-softmax/
9. https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

3 Implementing a CNN

In this implementation a set of images of cats and dogs are given, the CNN will be trained using them to predict if a given image is of a cat or that of a dog.

dataset : https://www.superdatascience.com/pages/deep-learning/

3.1 Building the CNN

3.1.1 step 1: Import libraries

[204]:
```
# importing libraries
from keras.models import Sequential      # for adding layers sequentially
from keras.layers import Conv2D          # for convolution
from keras.layers import MaxPooling2D    # for max-pooling
from keras.layers import Flatten         # for flatten
from keras.layers import Dense           # for Full-connection
from keras.preprocessing.image import ImageDataGenerator # for image processing

#initialising CNN object
CNN_classifier = Sequential()
```

3.1.2 Step 2: Convolution

[205]:
```
CNN_classifier.add(Conv2D(filters = 32,            # Number of filters␣
 ↪(feature-detector)
                          kernel_size = (3,3),      # filter size (row, col)
                          input_shape = (64,64,3),  # image dim : 3 ch of 64X64␣
 ↪res
                          activation = 'relu')      # activation func
                  )
```

3.1.3 Step 3: Max-pooling

Here the stride is taken as 2, thus the feature map would be of size $\left[\frac{m}{2}+1\times\frac{m}{2}+1\right]$ for odd m, n and $\left[\frac{m}{2}\times\frac{m}{2}\right]$ from even m, n

[206]:
```
CNN_classifier.add(MaxPooling2D(pool_size = (2,2))) #adds a max-pooling layer
```

3.1.4 Step 4: Flattening

Here the feature map space gets flattened into a single vector, there are two common qustions may arise. 1. How flattening preserves the information ? * ans : During max-pooling stage, the special stuctures are preserved and while flattening they are kept 2. Why not directly map image into flattened vector ? * ans : In that case, each input neuron would correspond to a single pixel without any correlation to the neighbouring pixels, thus not preserving any specual information. * Also, this would make the ANN over-complicated and may introduce over-fitting

[207]:
```
CNN_classifier.add(Flatten()) #adds a flattening layer
```

3.1.5 Step 5: Full-Connection

[208]:
```
CNN_classifier.add(Dense(units = 128,    # subject to trial, but must be 2^n
                         activation = 'relu')
                  )
CNN_classifier.add(Dense(units = 1,      #  output layer with binary␣
 ↪classification
                         activation = 'sigmoid')
                  )
```

3.1.6 Step 5 : Compile the CNN

[209]:
```
CNN_classifier.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics= ['accuracy']
                      )
```

3.2 Training the CNN

In order to avoid overfitting, a process called **Image Augmentation** is used. The process is used to train a network with relatively fewer number of images but ensures better accuracy. It takes training imges in batches, apply random transformation such as rotation, sharing etc. and makes several version of replica from a single source, these replicas are used to train the CNN. Thus, *Image augmentation is a trick to enrich an image dataset by transforming the existing images*

[210]:
```
# code source : https://keras.io/preprocessing/image/
# perform Image augmentation
train_datagen = ImageDataGenerator(rescale=1./255,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
[211]: # generating training set
train_set = train_datagen.flow_from_directory(
                                    'cnn_ds/ds/training_set',
                                    target_size=(64, 64),
                                    batch_size=32,
                                    class_mode='binary')
```

```
Found 8000 images belonging to 2 classes.
```

```
[212]: # generating test set
test_set = test_datagen.flow_from_directory(
                                    'cnn_ds/ds/test_set',   # test location
                                    target_size=(64, 64),   # resolution
                                    batch_size=32,
                                    class_mode='binary')    # classification type
```

```
Found 2000 images belonging to 2 classes.
```

```
[ ]: # Train CNN and validate with test set
CNN_classifier.fit_generator(train_set,
                                steps_per_epoch=8000,      # number of training␣
 ↪img
                                epochs=25,
                                validation_data=test_set, # test_set object
                                validation_steps=2000)     # number of test img
```

In order to improve the performance, the CNN must deep. There are two optins, either add more dense layer or add more convolution layers. Adding more convolution layer would privide better results. Just add the follwoing peice of codes before the flattening stage. Notice, `input_shape` is not given as it is only given to the first layer, for the following layers, keras would know what was the output nodes for the last layer, and set that as the input dimension of the current layer.

```
CNN_classifier.add(Conv2D(filters = 32,
                          kernel_size = (3,3),
                          activation = 'relu')
                  )
CNN_classifier.add(MaxPooling2D(pool_size = (2,2)))
```