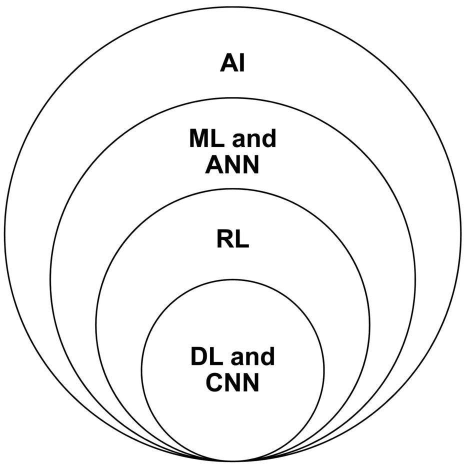
Build your First Image Processing Project with CNN

Introduction

CNN's were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits. It was mostly used in the postal sectors to read zip codes, pin codes, etc. The important thing to remember about any deep learning model is that it requires a large amount of data to train and also requires a lot of computing resources. This was a major drawback for CNNs at that period and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.
In 2012 Alex Krizhevsky realized that it was time to bring back the branch of deep learning that uses multi-layered neural networks. The availability of large sets of data, to be more specific ImageNet datasets with millions of labeled images and an abundance of computing resources enabled researchers to revive CNNs.



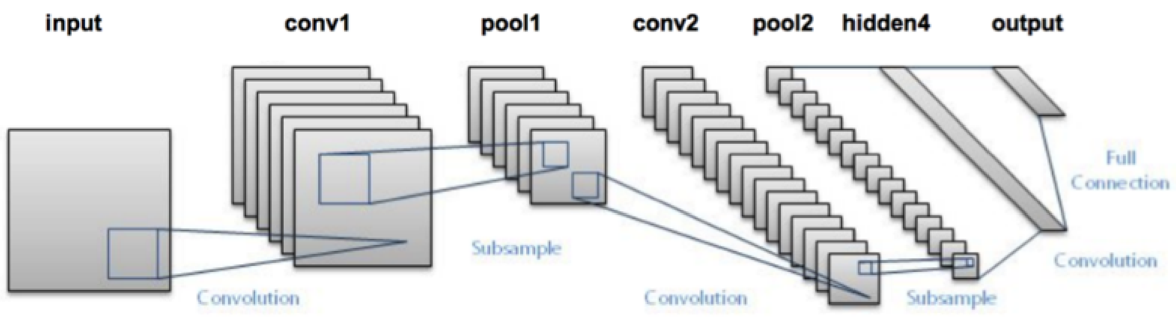
Deep learning is a booming field at the current time and most of the projects and problem statements use deep learning in any sort of work. If you have to pick a deep learning technique for solving any computer vision problem statement then many of you including myself will go with a conventional neural network.
In this article, we will build our first image processing project using CNN and understand its power and why it has become so popular. In this article, we will walk through every step of developing our own convolutional model and build our first amazing project.

Why Image Classification?

Image classification is a technique where the system takes an input image and appropriately classifies it. Now-a-days, image classification is used by various organisations to simplify their work and make the process streamline and fast. Have you ever wondered about why your system is capable of identifying you and your family's faces, this all happens when Image Processing comes into account.
As technology advancement takes place there are new algorithms and neural networks becoming more powerful and capable to handle very large size images and videos, process them, and conclude them with proper subtitles.

Brief on Convolutional Neural Network(CNN)

In deep learning, a **convolutional neural network** (**CNN/ConvNet**) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics **convolution** is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.



Different layers in Convolutional Neural Network

Problem Statement

Self- driving cars are taking over the world, where the driver is fully dependent on the car. To achieve this high accuracy, it is important that the cars must understand the traffic signals correctly. Here, in this project we are going to develop a traffic sign identification problem.
There are many different traffic signs like speed limit, traffic signals, indicating directions(left or right), etc. The dataset we are working on contains 50000 images of 43 classes which are numbered from 0 to 42.

You can download the dataset from here.

Let's start!

Step 1: Load the Dataset

We'll start with importing the libraries. We will use the Keras library to load each layer that works on top of TensorFlow. So, please install TensorFlow before importing deep learning layers.

Load Dataset

```
!pip install tensorflow
!pip install keras
!pip install scikit-learn
```

Import the libraries

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
from keras.utils import to_categorical
from keras.layers import Conv2D, Dense, Flatten, MaxPool2D, Dropout
```

Now, we'll load all the images in a single list in the form of an array that will describe the pixels of the image and another list which will contain labels of the corresponding image. To feed image data to the model we need to convert it into a NumPy array.
The training dataset contains a different folder with the name of classes named 0 to 42. with the help of the os module we will iterate through each class folder and append the image and respective label to the list. We also have CSV files that contain the actual label category name.

```
imgs_path = "GTSRB_dataset/Train"
data = []
labels = []
classes = 43
for i in range(classes):
    img_path = os.path.join(imgs_path, str(i)) #0-42
    for img in os.listdir(img_path):
        im = Image.open(p + '/' + img)
        im = im.resize((30,30))
        im = np.array(im)
        data.append(im)
        labels.append(i)
data = np.array(data)
labels = np.array(labels)

print("success")
```

Explore sample Image

Let's look at one sample image using the pillow library.

```
path = "gtsrb-german-traffic-sign/Train/0/00000_00004_00029.png"
img = Image.open(i0)
img = img.resize((30, 30))
sr = np.array(img)
plt.imshow(img)

plt.show()
```

Step 2: Split Dataset into train and test

We will use the to_categorical method to convert labels into one-hot encoding.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
print("training shape: ",x_train.shape, y_train.shape)
print("testing shape: ",x_test.shape, y_test.shape)
y_train = to_categorical(y_train, 43)
```

```
y_test = to_categorical(y_test, 43)
```

Step 3: Build a CNN model

Now we will start developing a convolutional neural network to classify images for correct labels. CNN is best to work with image data.

The architecture of our CNN model

- Conv2D layer – we will add 2 convolutional layers of 32 filters, size of 5*5, and activation as relu
- Max Pooling – MaxPool2D with 2*2 layers
- Dropout with a rate of 0.25.0
- 2 Convolutional layer of 64 filters and size of 3*3
- Dropout with a rate of 0.25
- Flatten layer to squeeze the layers into 1 dimension
- Dense, feed-forward neural network(256 nodes, activation=”relu”)
- Dropout Layer(0.5)
- Dense layer(nodes=46, activation=”softmax”)

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu", input_shape=x_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation="relu"))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation="softmax"))
```

- MaxPool2D – Maximum pooling layer is used to reduce the size of images
- Dropout – Dropout is a regularization technique to reduce overfitting
- Flatten – to convert the parrel layers to squeeze the layers
- Dense – for feed-forward neural network

The last layer will have an activation function as softmax for Multi-class classification.

Step 4: Train and Validate the Model

Let's first compile the model. During compiling we need to describe the loss function and optimizer to use.

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

- Loss Function – to calculate the loss done by model. we will use categorical cross-entropy as we have a multiclass classification problem statement
- Optimizer – Optimize to optimize the loss function

Let’s fit the train and test data to model and start training the convolutional model. we need to define a number of epochs to train for and batch size to consider while training the model.

```
epochs = 15
history = model.fit(x_train, y_train, epochs=epochs, batch_size=64, validation_data=(x_test, y_test))
```

It will take some time to run so please keep patience till it runs.Now let us plot an accuracy and loss graph using Matplotlib.

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Training accuracy")
plt.plot(history.history['val_accuracy'], label="val accuracy")
plt.title("Accuracy")
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.legend()
plt.figure(1)
plt.plot(history.history['loss'], label="training loss")
plt.plot(history.history['val_loss'], label="val loss")
plt.title("Loss")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

Step 5: Test the Model

The dataset contains a test folder that has different test images and a test.csv file that contains details related to the image path and respective labels. Again we will load the data using pandas and resize it to the shape of 30*30 pixels and convert it to a NumPy array. After processing test images we will check the accuracy of the model against actual labels.

```
from sklearn.metrics import accuracy_score
test = pd.read_csv("gtsrb-german-traffic-sign/Test.csv")
test_labels = test['ClassId'].values
test_img_path = "../input/gtsrb-german-traffic-sign"
test_imgs = test['Path'].values
test_data = []
test_labels = []
for img in test_imgs:
    im = Image.open(test_img_path + '/' + img)
    im = im.resize((30,30))
    im = np.array(im)
    test_data.append(im)
test_data = np.array(test_data)
predictions = model.predict_classes(test_data)
print("accuracy: ", accuracy_score(test_labels, predictions))
```

Step 6: Save the Model

Save the model for future use as well, we will use the dump model to create a GUI for Traffic Classification Project.

```
model.save('traffic_classifier.h5')
```

Hence we have successfully built and evaluated our Convolutional neural network for the image classification task. Now we will work on the frontend part and deploy our model on GUI using the python Tkinter library.