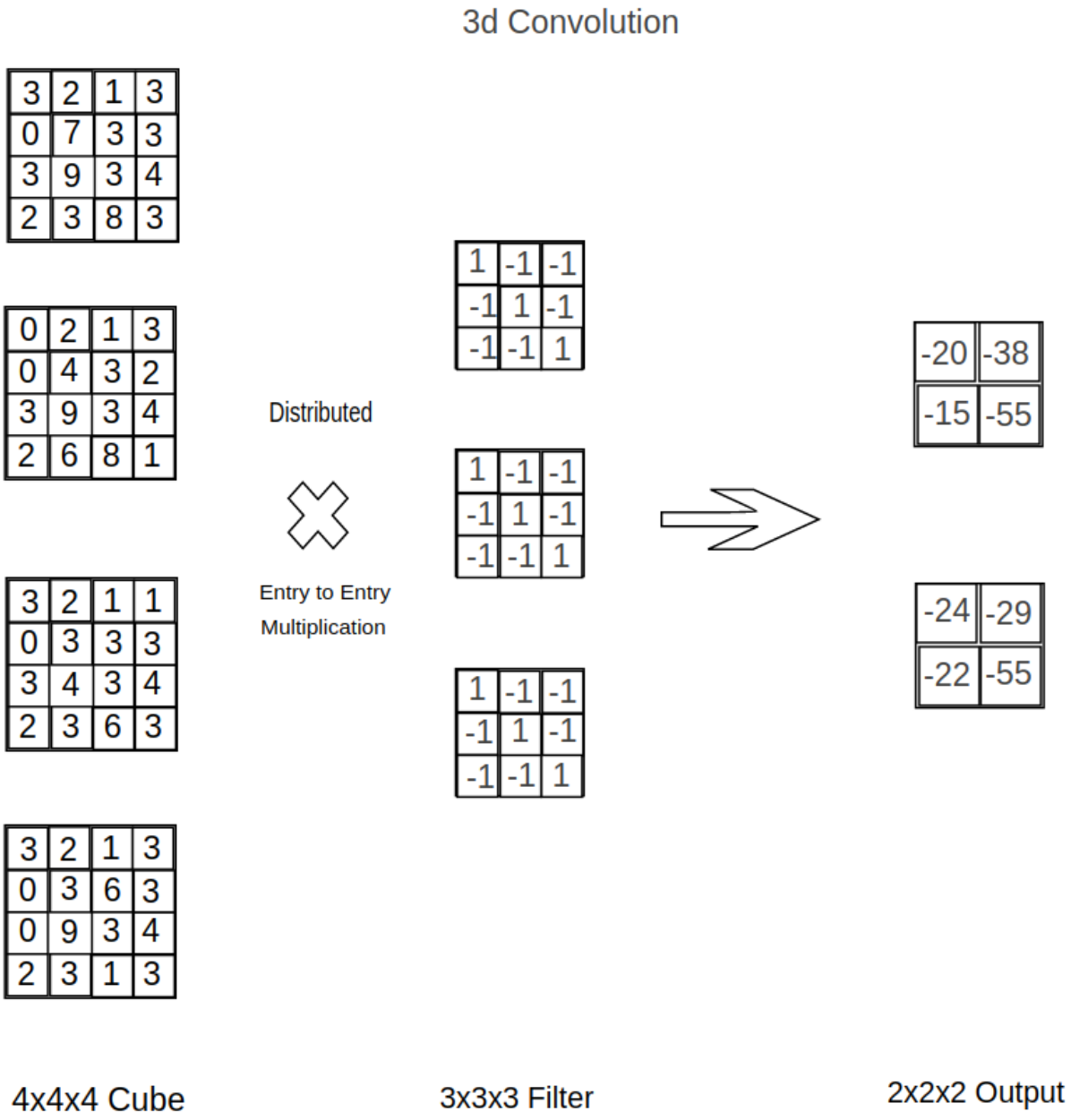


3D Convolution (own picture)

Step by Step Implementation: 3D Convolutional Neural Network in Keras

Michael Chan

In this article, we will be briefly explaining what a 3d CNN is, and how it is different from a generic 2d CNN. Then we will teach you step by step how to implement your own 3D Convolutional Neural Network using Keras.

1] What is a 3D Convolutional Neural Network?

A 3d CNN remains regardless of what we say a CNN that is very much similar to 2d CNN. Except that it differs in these following points (non-exhaustive listing):

3d Convolution Layers

Originally a 2d Convolution Layer is an entry per entry multiplication between the input and the different filters, where filters and inputs are 2d matrices. (fig.1)

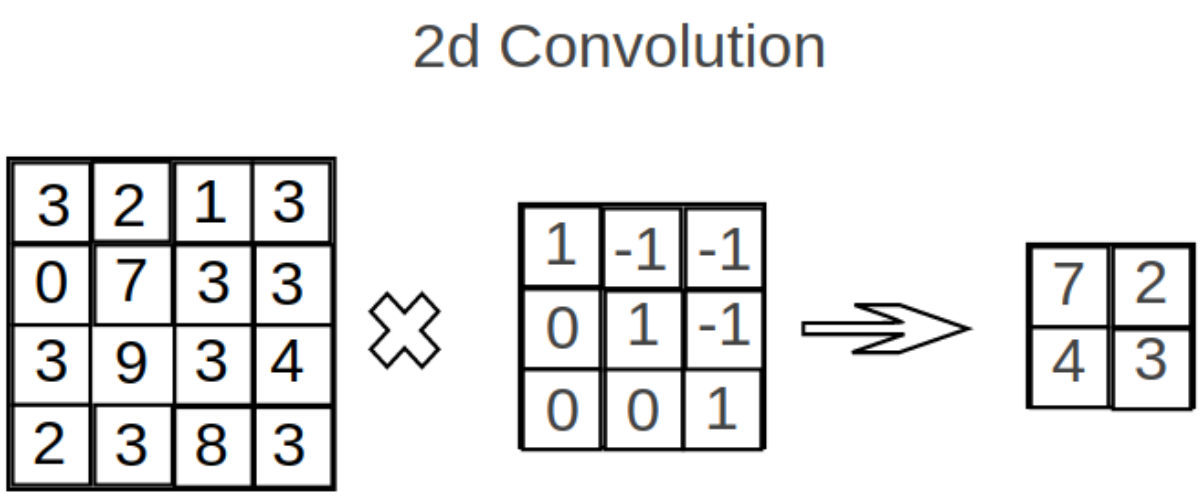


fig.1 (copyrighted: own)

In a 3d Convolution Layer, the same operations are used. We do these operations on multiple pairs of 2d matrices. (fig.2)

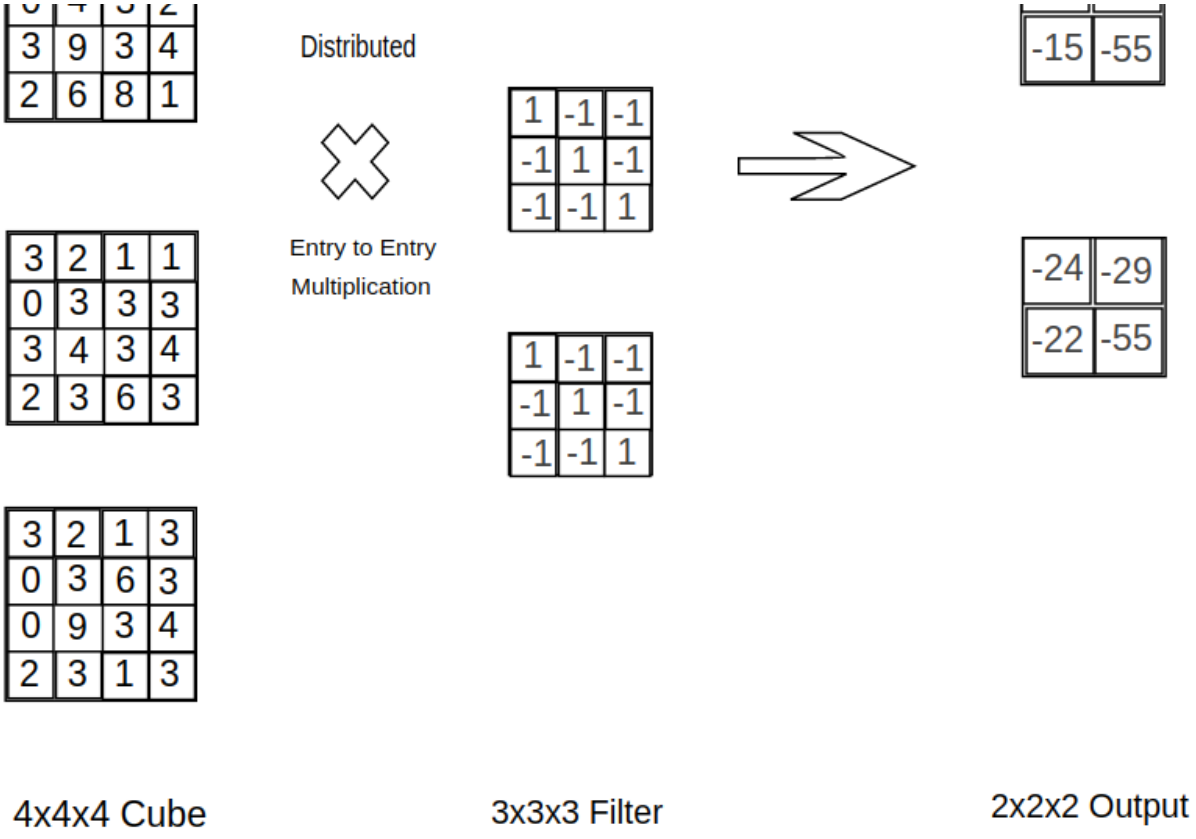
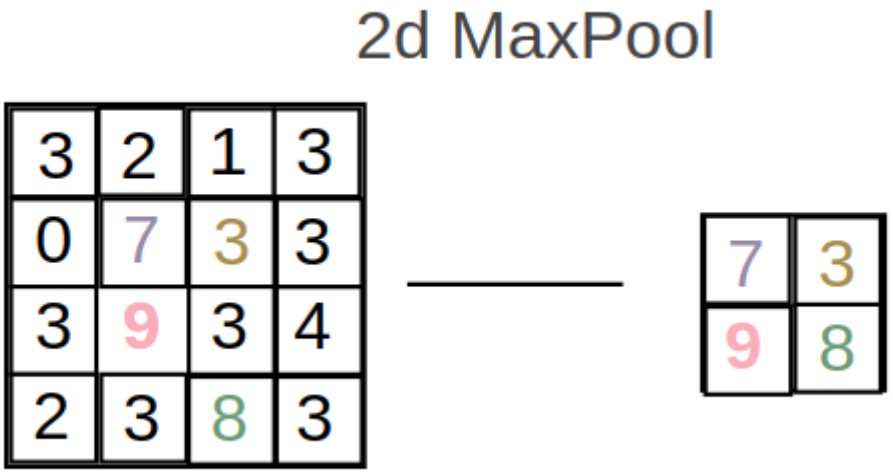


fig.2 (copyrighted: own)

Padding options and slides step options work the same way.

3d MaxPool Layers

2d Maxpool Layers (2x2 filter) is about taking the maximum element of a small 2x2 square that we delimitate from the input. (fig.3)



Now in a 3d Maxpool (2x2x2), we look for the maximum element in a width 2 cube. This cube represents the space delimited by the 2x2x2 zone from the input. (fig.4)

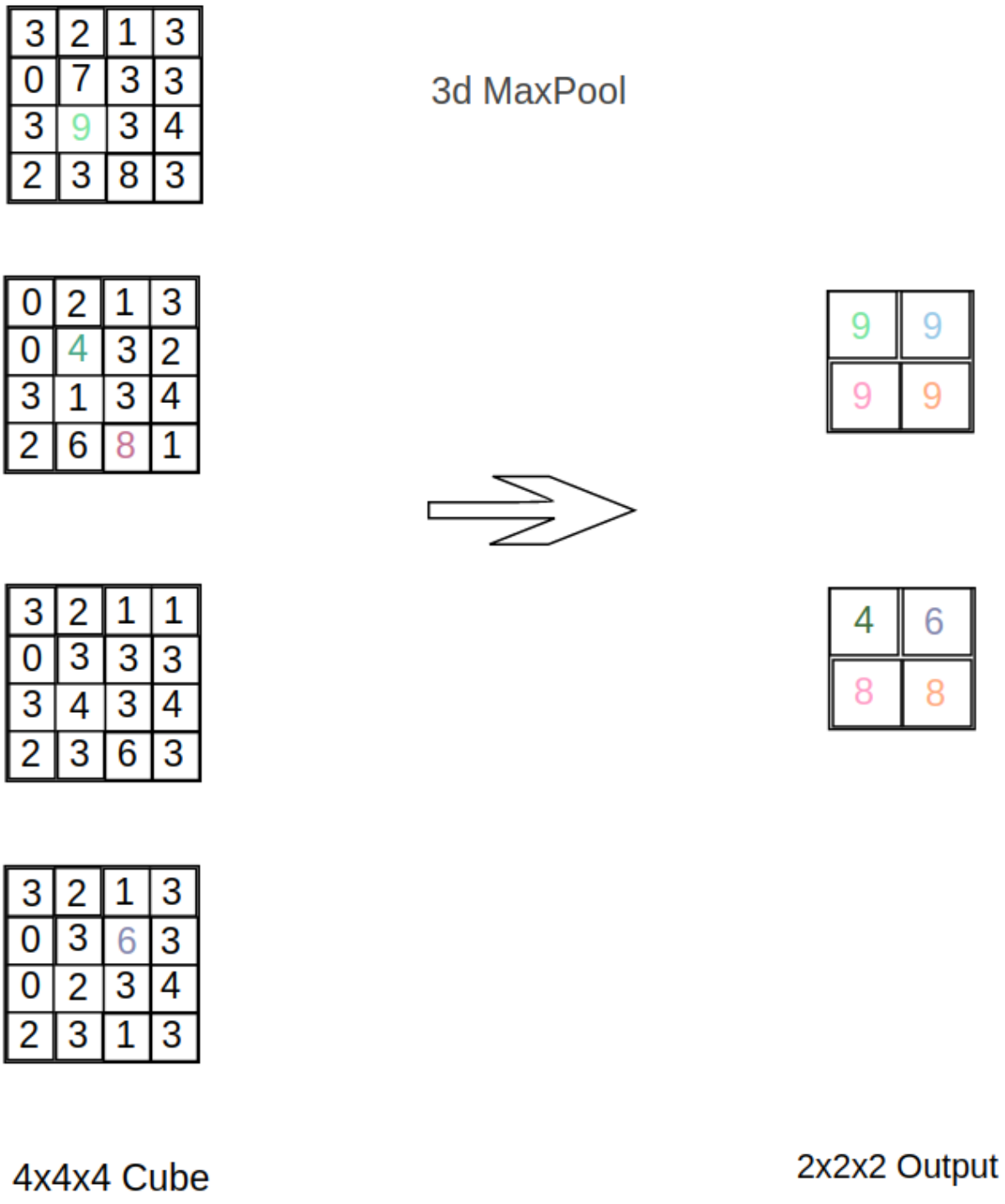


fig.4(copyrighted: own)

Note that the number of operations (compared to 2d CNN layers) is multiplied by the size of the filters used (regardless of the layer being Maxpool or Convolution) and also

So how does a data point for a 3d CNN look like?

One way to picture it is by using the following image (fig.5):

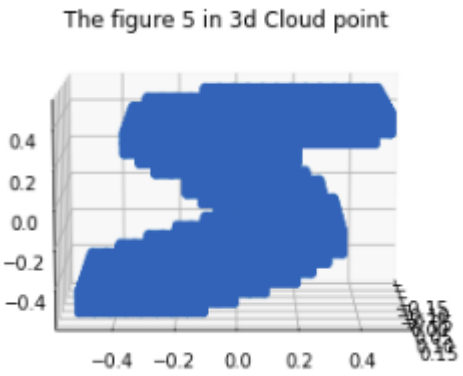


fig.5 (copyrighted: own)

Other existing datasets that you can use for your CNN are:

- RGB-D devices: Google Tango, Microsoft Kinect, etc.
- Lidar
- 3D reconstruction from multiple images

3] Preprocessing and Implementations

You can try for yourself the code on this dataset from Kaggle that we are using.

The required libraries to import are as follows:

```
import os
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"   #if like me you do not have a lot of m
os.environ["CUDA_VISIBLE_DEVICES"] = "" #then these two lines force keras to use your C
import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv3D, MaxPooling3D, Dropout, BatchNormalizat
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
import h5py
```

lib.py hosted with ❤ by GitHub view raw

To begin with, since the dataset is a bit specific, we use the following to helper functions to process them before giving them to the network.

```python
def array_to_color(array, cmap="Oranges"):
    s_m = plt.cm.ScalarMappable(cmap=cmap)
    return s_m.to_rgba(array)[:,:-1]


def rgb_data_transform(data):
    data_t = []
    for i in range(data.shape[0]):
        data_t.append(array_to_color(data[i]).reshape(16, 16, 16, 3))
    return np.asarray(data_t, dtype=np.float32)
```

helper.py hosted with ❤ by GitHub view raw

Plus, the dataset is stored as h5 file, so to extract the actual data points, we are required to read from h5 file, and use the to_categorical function to transform it into vectors. In this step, we also prepare for cross-validation.

```python
with h5py.File("./full_dataset_vectors.h5", "r") as hf:

    # Split the data into training/test features/targets
    X_train = hf["X_train"][:]
    targets_train = hf["y_train"][:]
    X_test = hf["X_test"][:]
    targets_test = hf["y_test"][:]
```

```
11
12      # Reshape data into 3D format
13      X_train = rgb_data_transform(X_train)
14      X_test = rgb_data_transform(X_test)
15
16      # Convert target vectors to categorical targets
17      targets_train = to_categorical(targets_train).astype(np.integer)
18      targets_test = to_categorical(targets_test).astype(np.integer)
19
```

h5.py hosted with ❤ by GitHub view raw

Finally, the model and the syntax for 3d CNN are as follows: (the architecture was picked without much refining since that is not the point of this article)

```
1  # Create the model
2  model = Sequential()
3  model.add(Conv3D(32, kernel_size=(3, 3, 3), activation='relu', kernel_initializer='he_u
4  model.add(MaxPooling3D(pool_size=(2, 2, 2)))
5  model.add(BatchNormalization(center=True, scale=True))
6  model.add(Dropout(0.5))
7  model.add(Conv3D(64, kernel_size=(3, 3, 3), activation='relu', kernel_initializer='he_u
8  model.add(MaxPooling3D(pool_size=(2, 2, 2)))
9  model.add(BatchNormalization(center=True, scale=True))
10 model.add(Dropout(0.5))
11 model.add(Flatten())
12 model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
```

```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])
model.summary()
# Fit data to model
history = model.fit(X_train, targets_train,
            batch_size=128,
            epochs=40,
            verbose=1,
            validation_split=0.3)
```

model.py hosted with ❤ by GitHub view raw

Note that the numbers of parameters will be a lot higher for the same number of layers compared to 2d CNN.

For your information, after a small sample training, we got the following accuracies and losses. (fig.6)

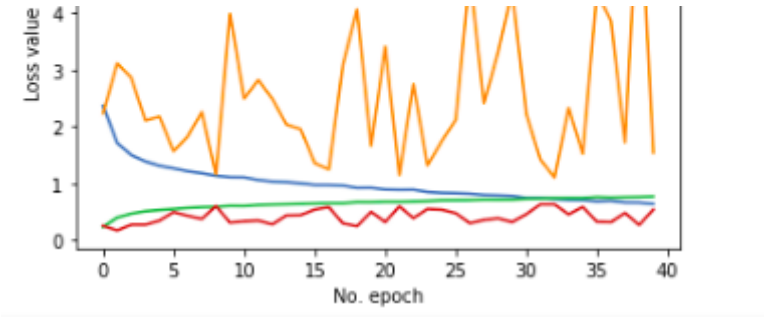


fig.6 (copyrighted: own)

4] But then a 3d? What for?

There happens to have many applications for a 3d CNN that are for instance:

- IRM data processing and therefore the inference
- self-driving
- Distance estimation

Alright, that's pretty much all. I hope you will try this technology out!

. . .

Thanks for reading and follow me, my website and my Facebook page if you liked it!

Machine Learning | Deep Learning | Artificial Intelligence | Data Science | Data Visualization