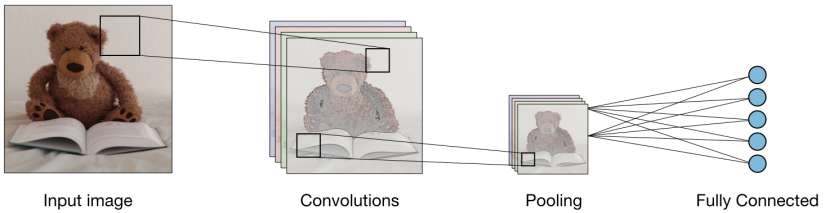
Welcome to Session 02 [Convolutional Neural Network (CNN)]

This is a Google Colaboratory (https://colab.research.google.com/notebooks/welcome.ipynb) notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select *CONNECT*.
2. Run all the notebook code cells: Select *Runtime* > *Run all*.

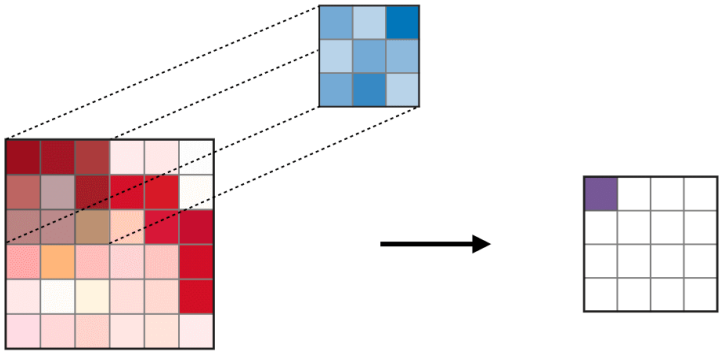
Overview

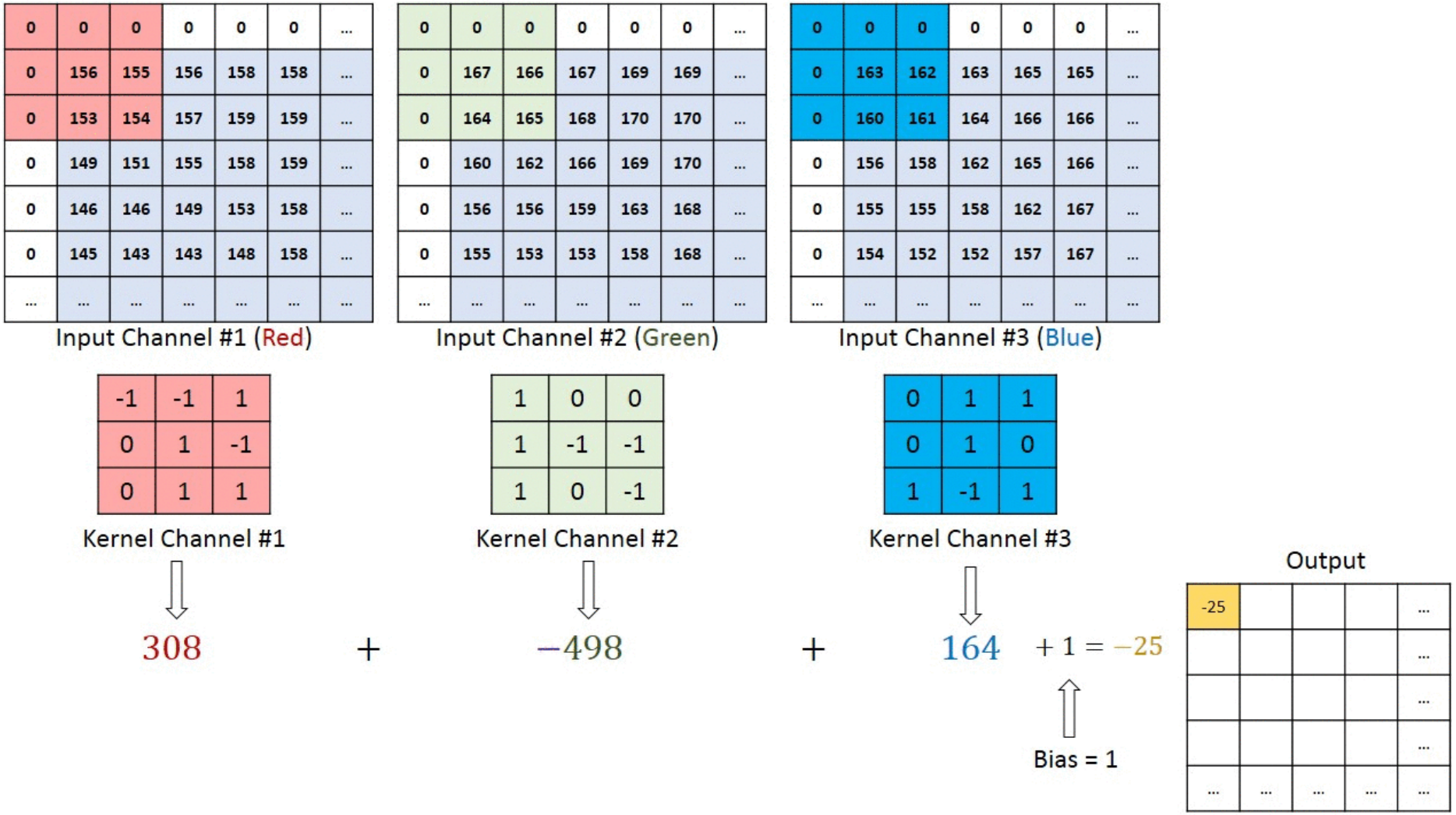
Architecture of a traditional CNN — Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



Types of Layer

Convolution layer (CONV) — The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size F and stride S. The resulting output O is called feature map or activation map.



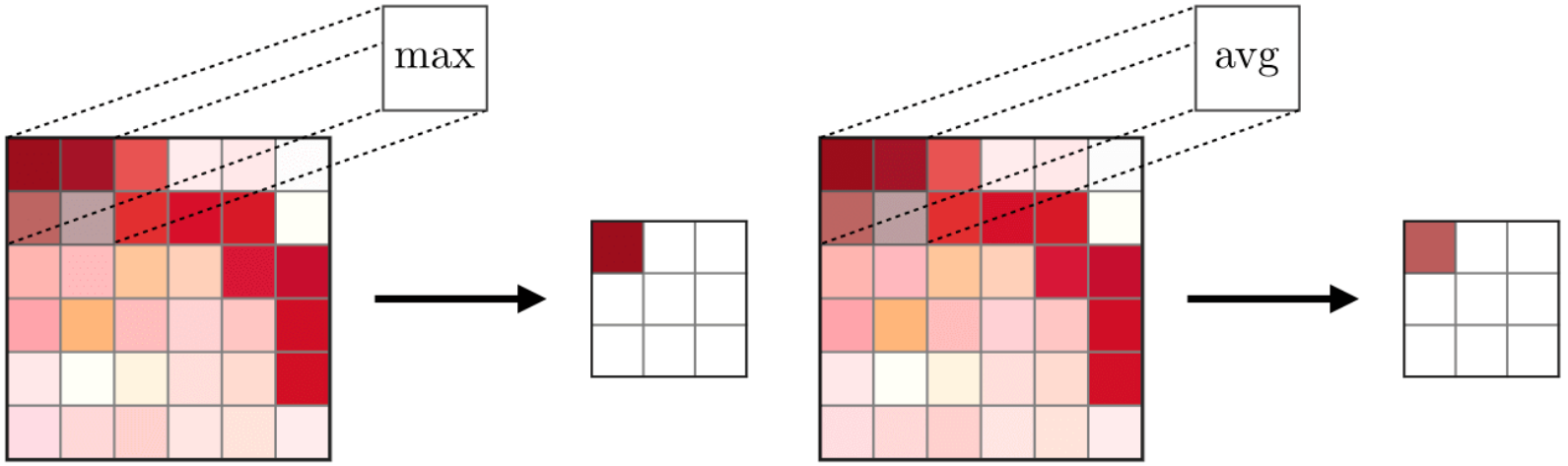


Remark: the convolution step can be generalized to the 1D and 3D cases as well.

Pooling (POOL) — The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view

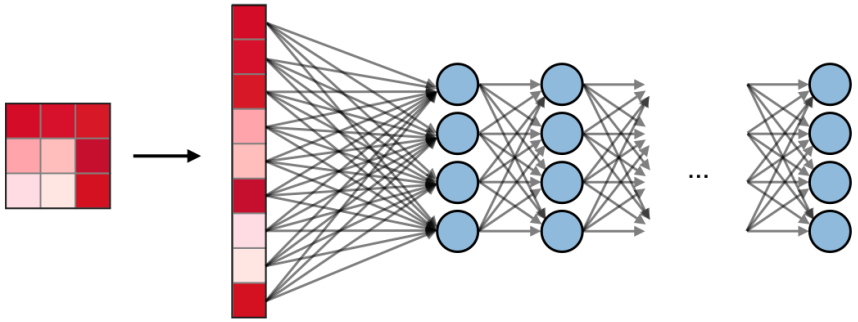
Illustration



Comments

- Preserves detected features
- Most commonly used
- Downsamples feature map
- Used in LeNet

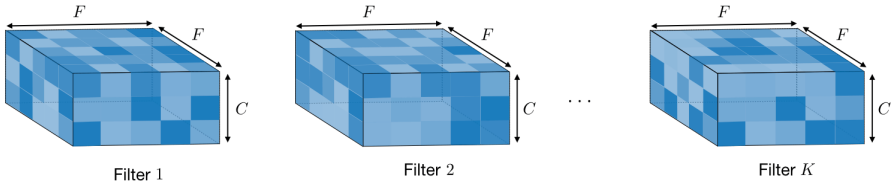
Fully Connected (FC) — The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



Filter hyperparameters

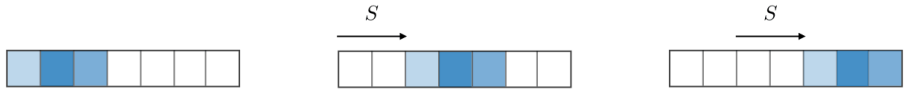
The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

Dimensions of a filter — A filter of size F×F applied to an input containing C channels is a F×F×C volume that performs convolutions on an input of size I×I×C and produces an output feature map (also called activation map) of size O×O×1.

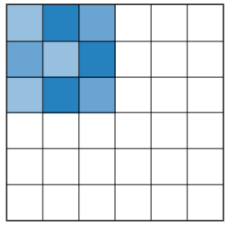
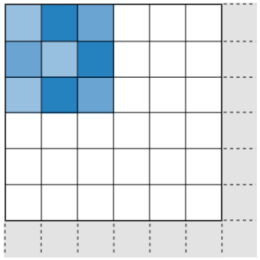
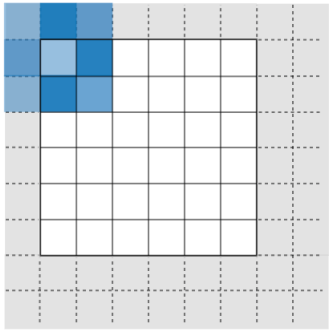


Remark: the application of K filters of size F×F results in an output feature map of size O×O×K.

Stride — For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



Zero-padding — Zero-padding denotes the process of adding P zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

Mode	Valid	Same	Full
Illustration			
Purpose	• No padding • Drops last convolution if dimensions do not match	• Output size is mathematically convenient • Also called 'half' padding	• Maximum padding such that end convolutions are applied on the limits of the input • Filter 'sees' the input end-to-end

Commonly used activation functions

Rectified Linear Unit — The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

Implementing The Model

Initialization

Mounting

In [0]:
```
# Mounting Gdrive

USE_G_COLAB = True

if USE_G_COLAB:
    from google.colab import drive


    drive.mount('/content/drive', force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk
8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Ao
ob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
# Project Root

root_dir = ''

if USE_G_COLAB:
    root_dir = '/content/drive/My Drive/workshops/2019_07_21/sessions_01/'
```

Install Requirement

In [0]:
```
!pip install -q tensorflow-gpu==2.0.0-beta1
!pip install -qU watermark
```

```
     |████████████████████████████████| 348.9MB 50kB/s
     |████████████████████████████████| 3.1MB 48.6MB/s
     |████████████████████████████████| 501kB 49.3MB/s
```

Custom Matplotlib Style

In [0]:
```
mpl_style = "https://gist.githubusercontent.com/m3hrdadfi/af8aca01094afb7d3e5b46de9ad8d509/raw/871ec5d721a3b438c3c896718ea4aafc91ea9744/gadfly.mplstyle"

!wget -q $mpl_style -O /root/.config/matplotlib/matplotlibrc
```

General Paramas

A random seed is a number used to initialize a pseudorandom number generator. For a seed to be used in a pseudorandom number generator, it does not need to be random

In [0]:
```
RANDOM_SEED = 141
```

Import requried packages

In [0]:
```
import tensorflow as tf

import requests
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils import shuffle

import matplotlib as mpl
import matplotlib.pyplot as plt

import cv2

import random
import sys
import io
import re
import time
from datetime import datetime
import os
import struct
import itertools
from tqdm import tqdm

from pprint import pprint


%matplotlib inline
mpl.rc_file(mpl.matplotlib_fname())

%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Persian MNIST

Data Set Information:

Attribute Information:

1. pixels
2. class:
    - 0
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9

In [0]:
```
!wget https://www.dropbox.com/s/op3ht07lfou9lbz/DigitDB.zip
!unzip DigitDB.zip
!ls
```

```
--2019-07-24 14:55:36--  https://www.dropbox.com/s/op3ht07lfou9lbz/DigitDB.zip
Resolving www.dropbox.com (www.dropbox.com)... 162.125.80.1, 2620:100:6030:1::a27d:5001
Connecting to www.dropbox.com (www.dropbox.com)|162.125.80.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/op3ht07lfou9lbz/DigitDB.zip [following]
--2019-07-24 14:55:36--  https://www.dropbox.com/s/raw/op3ht07lfou9lbz/DigitDB.zip
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc3d495a9fee00a812ada14c6e90.dl.dropboxusercontent.com/cd/0/inline/AlQODTFM7hxehLs
Mn6A_Gfpe-u2kDrnLNR9rHkQsNMDjyJJEHNV6fVIXaAupr6mGYxF1ZIE-1Q_XYmYzLEzn8iVjM3iX1e1iaVOVBqHuNWtmMg/file
# [following]
--2019-07-24 14:55:37--  https://uc3d495a9fee00a812ada14c6e90.dl.dropboxusercontent.com/cd/0/inline/
AlQODTFM7hxehLsMn6A_Gfpe-u2kDrnLNR9rHkQsNMDjyJJEHNV6fVIXaAupr6mGYxF1ZIE-1Q_XYmYzLEzn8iVjM3iX1e1iaVOV
BqHuNWtmMg/file
Resolving uc3d495a9fee00a812ada14c6e90.dl.dropboxusercontent.com (uc3d495a9fee00a812ada14c6e90.dl.dr
opboxusercontent.com)... 162.125.80.6, 2620:100:6030:6::a27d:5006
Connecting to uc3d495a9fee00a812ada14c6e90.dl.dropboxusercontent.com (uc3d495a9fee00a812ada14c6e90.d
l.dropboxusercontent.com)|162.125.80.6|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: /cd/0/inline2/AlQRAmjXIBvz5iF6PoNSmVOf6L4Sfjaqr2UhodiMoFvJTrYxB3WrHGuX2BQEsKTgEW13DMJRvh7G
V1SY1VMmygdrqCEojaGFgIffy-_1O79P7RrdM7Vfil1jWpLMVrS-gEc1hh3FyRVhDVHfNf9Fv3yT5yiaH7NabIPUytkqb3Mr1YhS
sU_NtWKp-hZKQAXCi7r6ZaT2NEWwoMYqVN1cN7b_8aSk8gy_iCVcOAVRVME4jnz0HAicTK6QHaYCfj1WKa0bUnx45J80n_-nnS4U
NuAOBqCqnQlUn0nZPHfNebFo3Fj1h4xVz9jCKB6SeC3IsUbFN5cn97LDqTh21zSOtRb5/file [following]
--2019-07-24 14:55:37--  https://uc3d495a9fee00a812ada14c6e90.dl.dropboxusercontent.com/cd/0/inline
2/AlQRAmjXIBvz5iF6PoNSmVOf6L4Sfjaqr2UhodiMoFvJTrYxB3WrHGuX2BQEsKTgEW13DMJRvh7GV1SY1VMmygdrqCEojaGFgI
ffy-_1O79P7RrdM7Vfil1jWpLMVrS-gEc1hh3FyRVhDVHfNf9Fv3yT5yiaH7NabIPUytkqb3Mr1YhSsU_NtWKp-hZKQAXCi7r6Za
T2NEWwoMYqVN1cN7b_8aSk8gy_iCVcOAVRVME4jnz0HAicTK6QHaYCfj1WKa0bUnx45J80n_-nnS4UNuAOBqCqnQlUn0nZPHfNeb
Fo3Fj1h4xVz9jCKB6SeC3IsUbFN5cn97LDqTh21zSOtRb5/file
Reusing existing connection to uc3d495a9fee00a812ada14c6e90.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 5290356 (5.0M) [application/zip]
Saving to: ‘DigitDB.zip’

DigitDB.zip         100%[===================>]   5.04M  8.54MB/s    in 0.6s

2019-07-24 14:55:38 (8.54 MB/s) - ‘DigitDB.zip’ saved [5290356/5290356]

Archive:  DigitDB.zip
  inflating: Train 60000.cdb
  inflating: RemainingSamples.cdb
  inflating: Test 20000.cdb
 DigitDB.zip   RemainingSamples.cdb  'Test 20000.cdb'
 drive         sample_data           'Train 60000.cdb'
```

In [0]:
```
def resize_image(src_image, dst_image_height, dst_image_width):
    src_image_height = src_image.shape[0]
    src_image_width = src_image.shape[1]

    if src_image_height > dst_image_height or src_image_width > dst_image_width:
        height_scale = dst_image_height / src_image_height
        width_scale = dst_image_width / src_image_width
        scale = min(height_scale, width_scale)
        img = cv2.resize(src=src_image, dsize=(0, 0), fx=scale, fy=scale, interpolation=cv2.INTER_CUB
IC)
    else:
        img = src_image

    img_height = img.shape[0]
    img_width = img.shape[1]

    dst_image = np.zeros(shape=[dst_image_height, dst_image_width], dtype=np.uint8)

    y_offset = (dst_image_height - img_height) // 2
    x_offset = (dst_image_width - img_width) // 2

    dst_image[y_offset:y_offset + img_height, x_offset:x_offset + img_width] = img

    return dst_image


def read_cdb(filepath):
    with open(filepath, 'rb') as f:
        data = f.read()
        offset = 0

        # read private header
        yy = struct.unpack_from('H', data, offset)[0]
        offset += 2

        m = struct.unpack_from('B', data, offset)[0]
        offset += 1

        d = struct.unpack_from('B', data, offset)[0]
        offset += 1

        h = struct.unpack_from('B', data, offset)[0]
```

```
        offset += 1

        w = struct.unpack_from('B', data, offset)[0]
        offset += 1

        total_rec = struct.unpack_from('I', data, offset)[0]
        offset += 4

        letter_count = struct.unpack_from('128I', data, offset)
        offset += 128 * 4

        img_type = struct.unpack_from('B', data, offset)[0]  # 0: binary, 1: gray
        offset += 1

        comments = struct.unpack_from('256c', data, offset)
        offset += 256 * 1

        reserved = struct.unpack_from('245c', data, offset)
        offset += 245 * 1

        if (w > 0) and (h > 0):
            normal = True
        else:
            normal = False

        images = []
        labels = []

        for i in tqdm(range(total_rec), position=0):

            start_byte = struct.unpack_from('B', data, offset)[0]  # must be 0xff
            offset += 1

            label = struct.unpack_from('B', data, offset)[0]
            offset += 1

            if not normal:
                w = struct.unpack_from('B', data, offset)[0]
                offset += 1

                h = struct.unpack_from('B', data, offset)[0]
                offset += 1
```

```
            byte_count = struct.unpack_from('H', data, offset)[0]
            offset += 2

            image = np.zeros(shape=[h, w], dtype=np.uint8)

            if img_type == 0:
                # Binary
                for y in range(h):
                    b_white = True
                    counter = 0
                    while counter < w:
                        wb_count = struct.unpack_from('B', data, offset)[0]
                        offset += 1

                        if b_white:
                            image[y, counter:counter + wb_count] = 0  # Background
                        else:
                            image[y, counter:counter + wb_count] = 255  # ForeGround
                        b_white = not b_white  # black white black white ...
                        counter += wb_count
            else:
                # GrayScale mode
                data = struct.unpack_from('{}B'.format(w * h), data, offset)
                offset += w * h
                image = np.asarray(data, dtype=np.uint8).reshape([w, h]).T

            images.append(image)
            labels.append(label)

        return images, labels


def load_data(datapath, img_height=32, img_width=32):
    images, labels = read_cdb(datapath)
    assert len(images) == len(labels)

    x = np.zeros(shape=[len(images), img_height, img_width], dtype=np.float32)
    y = np.zeros(shape=[len(labels)], dtype=np.int)

    for i in tqdm(range(len(images)), position=0):
        image = images[i]
        image = resize_image(src_image=image, dst_image_height=img_height, dst_image_width=img_width)
        image = image / 255
```

```
        image = np.where(image >= 0.5, 1, 0)

        x[i] = image
        y[i] = labels[i]

    x, y = shuffle(x, y, random_state=0)

    return x, y
```

Load the data

In [0]:
```
trainset_dir = 'Train 60000.cdb'
testset_dir = 'Test 20000.cdb'

x_train, y_train = load_data(trainset_dir)
x_test, y_test = load_data(testset_dir)

print('Train: %s, Labels: #%s' %(x_train.shape, len(y_train)))
print('Test: %s, Labels: #%s' %(x_test.shape, len(y_test)))
```

```
100%|██████████| 60000/60000 [00:07<00:00, 8401.26it/s]
100%|██████████| 60000/60000 [00:01<00:00, 58604.49it/s]
100%|██████████| 20000/20000 [00:02<00:00, 7866.61it/s]
100%|██████████| 20000/20000 [00:00<00:00, 55908.21it/s]

Train: (60000, 32, 32), Labels: #60000
Test: (20000, 32, 32), Labels: #20000
```

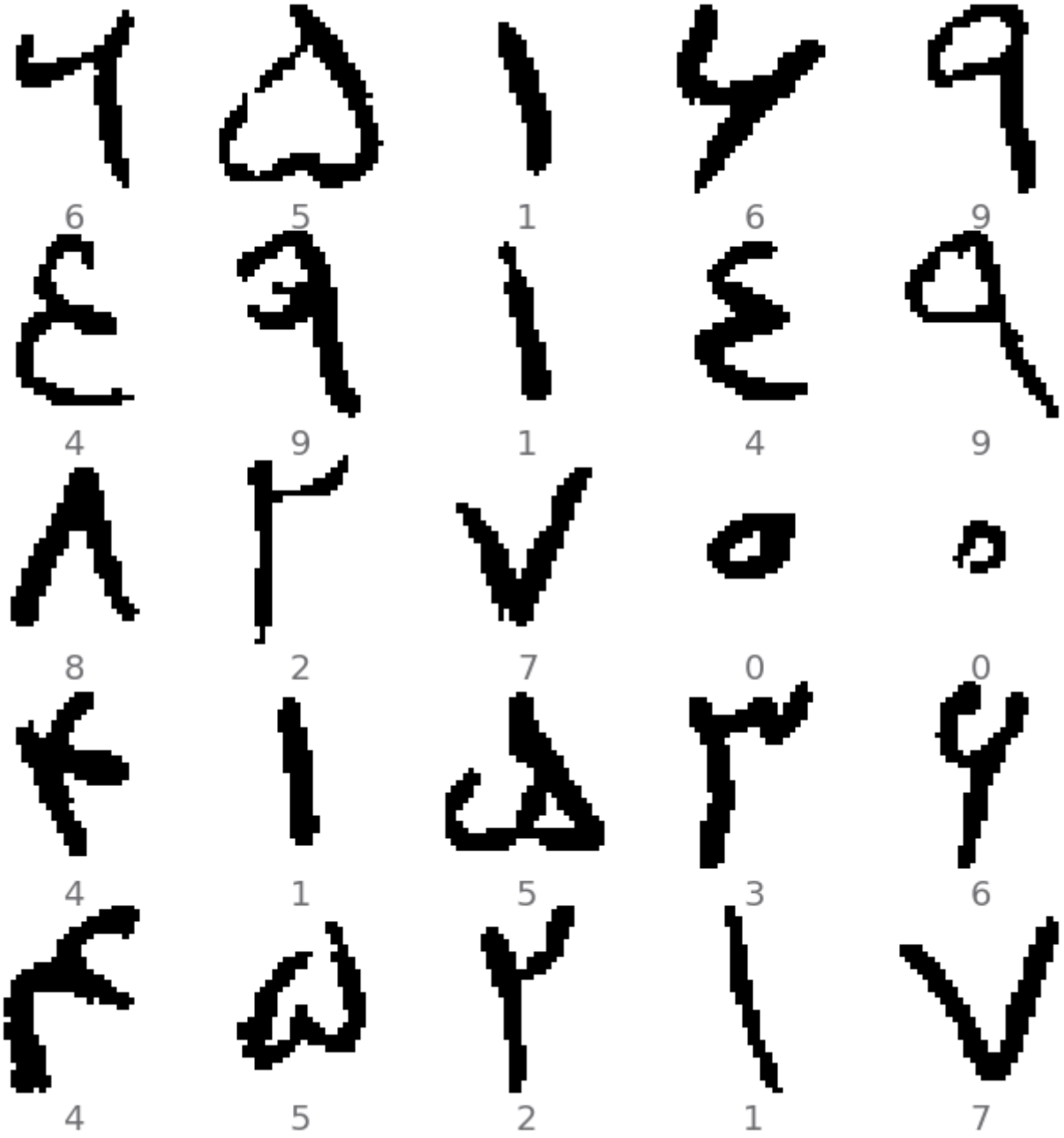
In [0]:
```
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.05, random_state=
RANDOM_SEED)

print('Train: %s, Labels: #%s' %(x_train.shape, len(y_train)))
print('Valid: %s, Labels: #%s' %(x_valid.shape, len(y_valid)))
print('Test: %s, Labels: #%s' %(x_test.shape, len(y_test)))
```

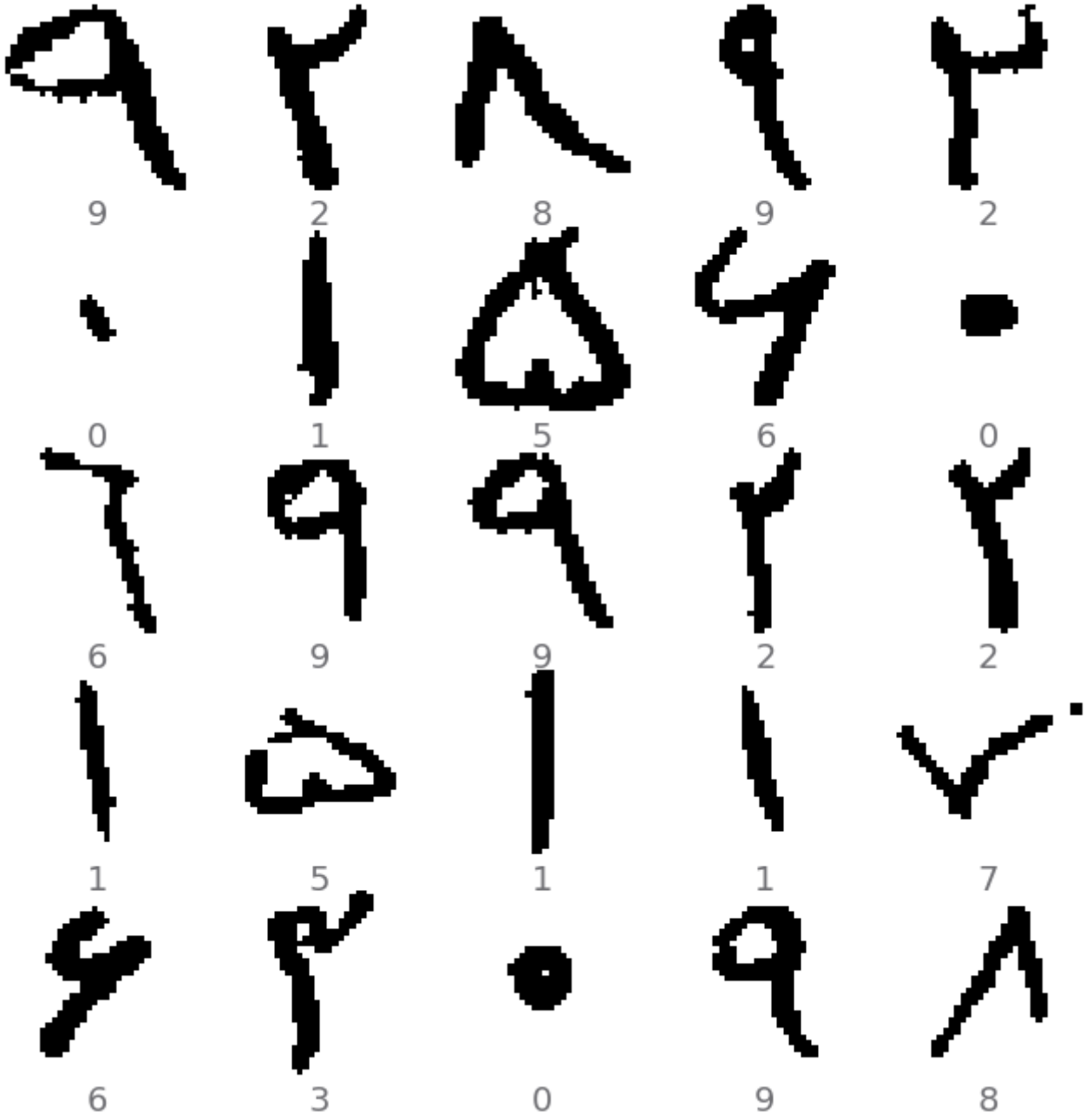
```
Train: (57000, 32, 32), Labels: #57000
Valid: (3000, 32, 32), Labels: #3000
Test: (20000, 32, 32), Labels: #20000
```

Visualizing the data

In [0]:
```
plt.figure(figsize=(8, 8))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(str(y_train[i]))
plt.show()
```


6 5 1 6 9
4 9 1 4 9
8 2 7 0 0
4 1 5 3 6
4 5 2 1 7

In [0]:
```
plt.figure(figsize=(8, 8))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i], cmap=plt.cm.binary)
    plt.xlabel(str(y_test[i]))
plt.show()
```


9 2 8 9 2
0 1 5 6 0
6 9 9 2 2
1 5 1 1 7
6 3 0 9 8

Preprocessing

In [0]:
```
```

```
Train: (57000, 32, 32, 1)
Valid: (3000, 32, 32, 1)
Test: (20000, 32, 32, 1)
```

Arch

In [0]:
```
def build_model(dnn_units):
    model, r, evaluate = None, None, None

    return model, r, evaluate
```

In [0]:
```
model, r, evaluate = build_model(64)
evaluate
```

Tensorboard

Hyperparams Tuning

In [0]:
```
def train_test_model(hparams):
    model = None
    accuracy = 0

    return accuracy
```

In [0]:
```
from datetime import datetime

!rm -rf ./logs/
logdir = 'logs/scaler/' + datetime.now().strftime('%Y%m%d-%H%M%S')
```

In [0]:
```
from tensorboard.plugins.hparams import api as hp


HP_DNN_UNITS = hp.HParam('dnn_units', hp.Discrete([16, 32, 64]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.4))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'rmsprop', 'sgd']))
METRIC_ACCURACY = 'accuracy'

with tf.summary.create_file_writer(logdir + '/hparam').as_default():
    hp.hparams_config(
        hparams=[HP_DNN_UNITS, HP_DROPOUT, HP_OPTIMIZER],
        metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
    )
```

In [0]:
```
def run(run_dir, hparams):
    with tf.summary.create_file_writer(run_dir).as_default():
        hp.hparams(hparams)  # record the values used in this trial
        accuracy = train_test_model(hparams)
        tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

In [0]:
```
session_num = 0

for num_units in HP_DNN_UNITS.domain.values:
    for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_value):
        for optimizer in HP_OPTIMIZER.domain.values:
            hparams = {
                HP_DNN_UNITS: num_units,
                HP_DROPOUT: dropout_rate,
                HP_OPTIMIZER: optimizer,
            }
            run_name = "run-%d" % session_num
            print('--- Starting trial: %s' % run_name)
            print({h.name: hparams[h] for h in hparams})
            run(logdir + '/hparam/' + run_name, hparams)
            session_num += 1
```

After Tuning

In [0]:
```
def build_optimal_model():
    model = None

    return model
```

In [0]:
```
optimal_model = build_optimal_model()
```

In [0]:
```
from datetime import datetime

!rm -rf ./logs/
logdir = 'logs/scaler/' + datetime.now().strftime('%Y%m%d-%H%M%S')
```

In [0]:
```
class ImagePredictionCallBack(tf.keras.callbacks.Callback):

    def __init__(self, log_dir, model):
        super(ImagePredictionCallBack, self).__init__()
```

In [0]:
```
ig_cb = ImagePredictionCallBack(log_dir=logdir, model=optimal_model)
tb_cb = tf.keras.callbacks.TensorBoard(log_dir=logdir, histogram_freq=1)
```

In [0]:
```
r = optimal_model.fit(x_train_ext, y_train,
              validation_data=[x_valid_ext, y_valid],
              epochs=2,
              batch_size=128,
              callbacks=[tb_cb, ig_cb],
              verbose=1)
```

Tensorboard

In [0]:
```
# %tensorboard --logdir logs/scaler/
```

In [0]: