Deep Learning Workshop II 2018

Organizing Committee:
Maggi Zhu
Aly El Gamal
Stanley Chan
Charles Bouman
Greg Buzzard
Dong Hye Ye
Amir Ziabari
Sri Yarlagada
Diyu Yang

Part 1: GPU Implementation of Deep Neural Networks Using Tensorflow and Keras

Amirkoushyar Ziabari

aziabari@purdue.edu

Aug 08 2018

Outline

- Deep Learning Frameworks
- Tensors
- What is Tensorflow (TF)?
- Tensorflow Structure
- CNN to classify MNIST
- GPU implementation
- Keras

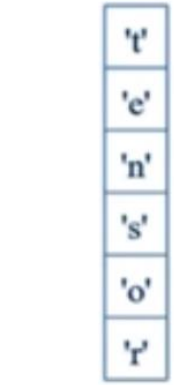
Deep Learning Frameworks

Framework	Distributed Execution	Architecture Optimization	Visualization	Community Support	Portability	Notes
Tensorflow	✓✓	✓✓	✓✓	✓✓	✓✓	• Google, • wide usage, ecosystem and community support • Visualization is superior
Pytorch	✓✓	✓✓	✓✓	✓✓	✓✓	• Facebook • Easy to use & technically impressive
CNTK	✓✓	✓✓	✓	-	✓✓	• Microsoft • Licensig issues
MXNet	✓	✓✓	✓	-	✓✓	• Behind other frameworks in DE, and lacks details
Caffe2	✓✓	✓✓	-	-	✓✓	• Facebook • Still moving
Caffe	-	✓✓	✓	✓	✓	• Facebook • Lacks future community support
Theano	✓	✓✓	✓	✓	✓	• University of Montreal • Lacks future community support

- **https://agi.io/2018/02/09/survey-machine-learning-frameworks/**

Tensors

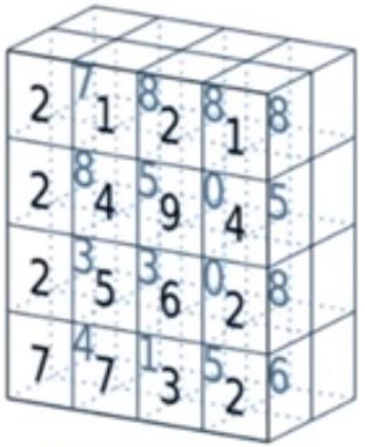
- Tensors are the standard way of representing data in Tensorflow (deep learning)
- Tensors are multidimensional arrays, an extension of matrices to data with higher dimensions



Tensor of dimension[1]

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

Tensor of dimensions[2]



Tensor of dimensions[3]

Tensor Rank

Rank	Math Entity	Python Example
0	Scalar (magnitude only)	s = 483
1	Vector (magnitude and direction)	v = [1.1, 2.2, 3.3]
2	Matrix (table of numbers)	m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3	3-Tensor (cube of numbers)	t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]
n	n-Tensor (you get the idea)	....

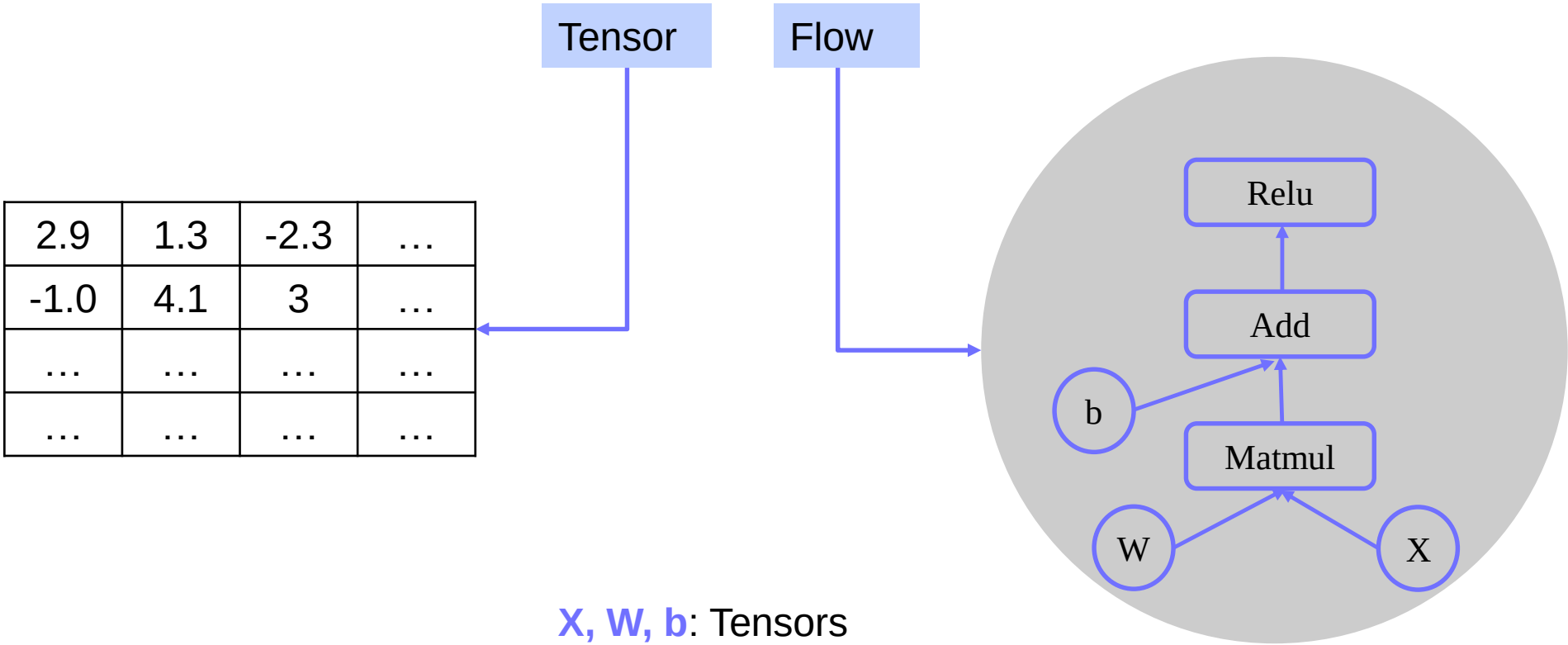
Tensor Data Types

- In addition to dimensionality Tensors have different data types

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a tensor is a byte array.
DT_BOOL	tf.bool	Boolean.

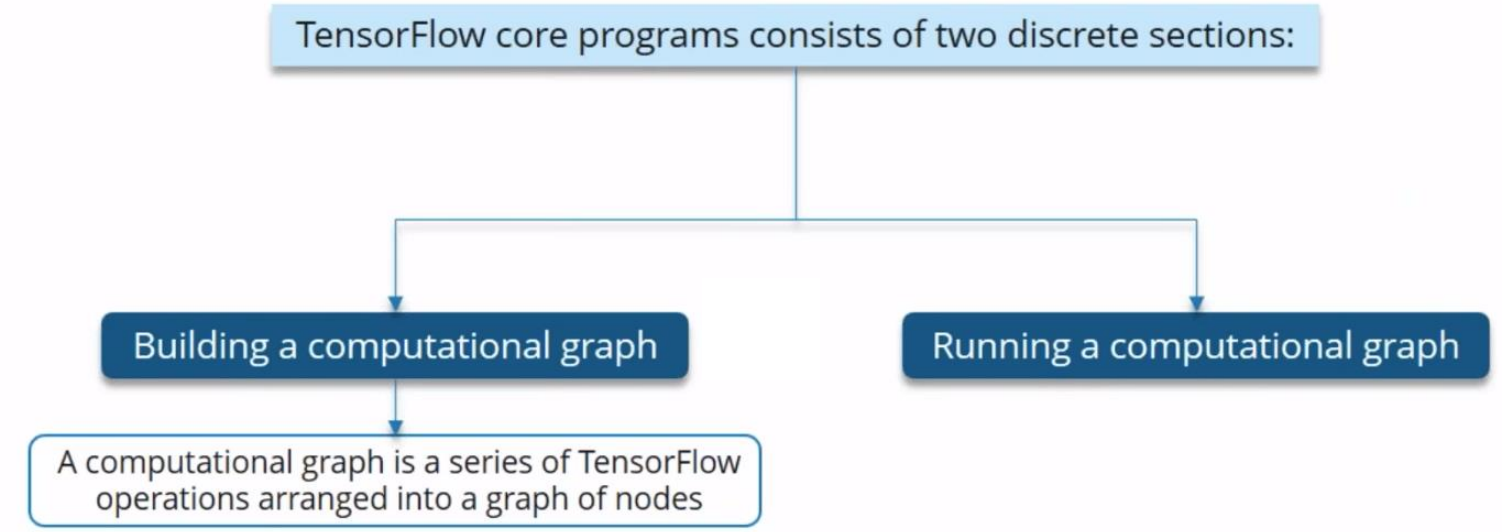
What is Tensorflow?

- Tensorflow (TF): a python library to implement deep networks
  - Very simple to install on all operating systems (tensorflow.org/install)
  - Pycharm, ipython, etc can be used to run TF on windows
- In Tensorflow, computation is approached as a dataflow graph



X, W, b: Tensors
Relu, Add, Matmul: functions

Tensorflow Structure




```
import tensorflow as tf

a = tf.constant(5.0,tf.float32)
b = tf.constant(6.0)
c = a*b
```

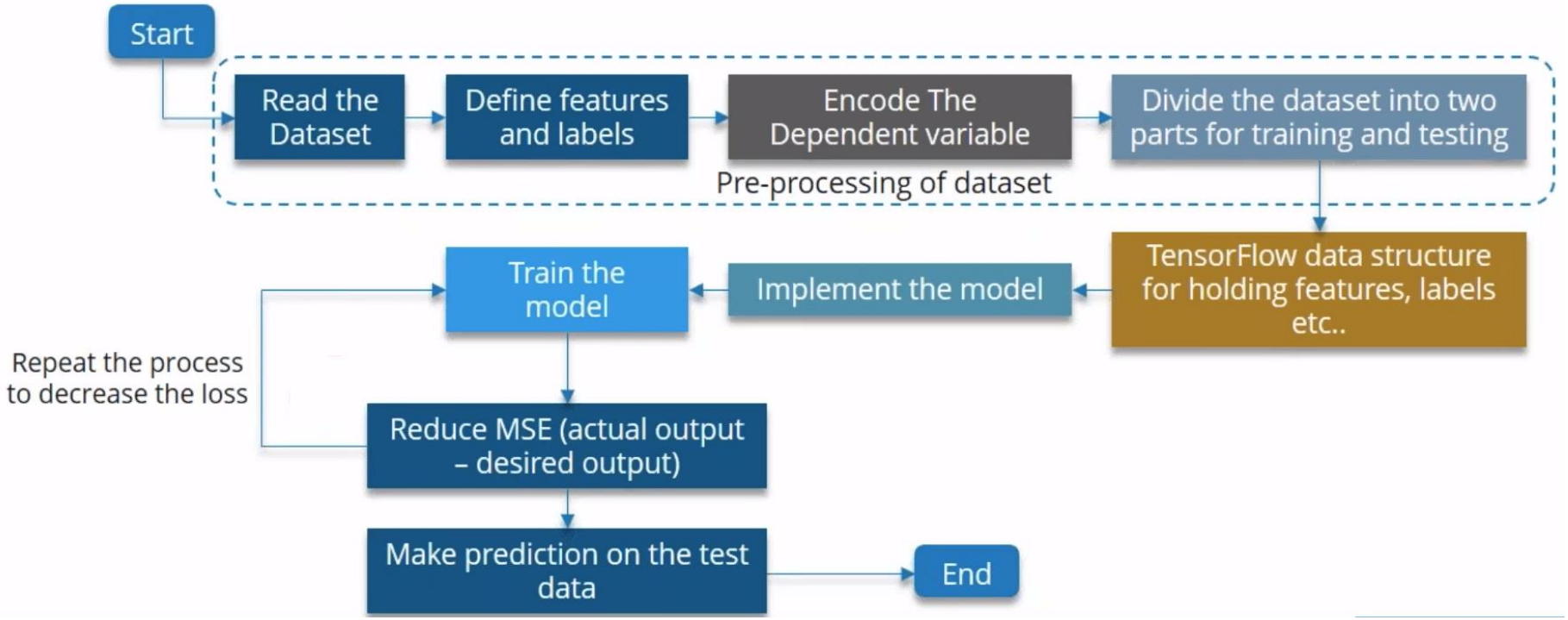
Build a computational graph

Run the computational graph

Goal

- **Develop a Convolutional Neural Network To Classify MNIST DATA**

Training Process



Code Structure

Build the computational Graph

0. Import the necessary libraries
1. Read the input data; define parameters, constants..
2. Define input/target size, type
   Assign space for input/target
3. Define weights and biases
4. Define and construct the model
   (e.g. Convolutional Neural Net)
5. Define loss function
6. Choose optimization technique
7. Define Training operation
8. Define Initialization operation
9. Define events logs and saving operations

Launch the computational Graph

10. Define Session and run initialization
11. Train the model (run the training op.)
    Print outputs, Save (or restore) model and events logs

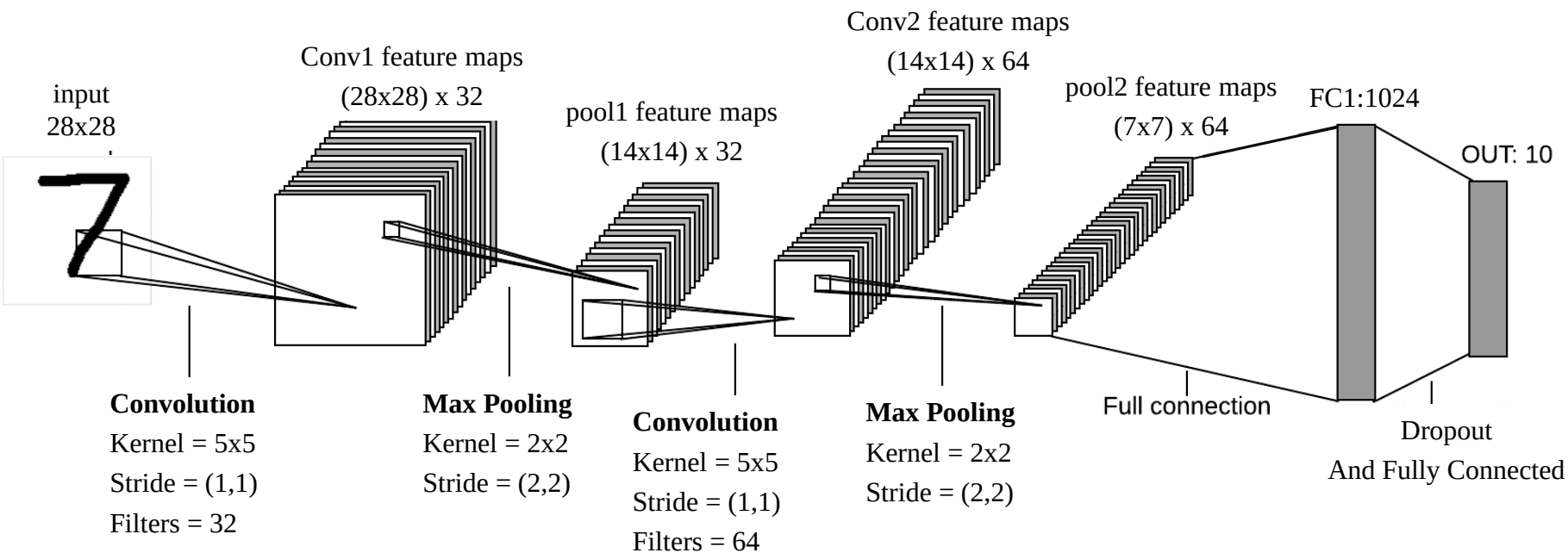
MNIST Dataset

MNIST Dataset Overview

This example is using MNIST handwritten digits. The dataset contains 60,000 examples for training and 10,000 examples for testing. The digits have been size-normalized and centered in a fixed-size image (28x28 pixels) with values from 0 to 1. For simplicity, each image has been flattened and converted to a 1-D numpy array of 784 features (28*28).

More info: http://yann.lecun.com/exdb/mnist/

CNN Network


input
28x28
Conv1 feature maps
(28x28) x 32
pool1 feature maps
(14x14) x 32
Conv2 feature maps
(14x14) x 64
pool2 feature maps
(7x7) x 64
FC1:1024
OUT: 10
Convolution
Kernel = 5x5
Stride = (1,1)
Filters = 32
Max Pooling
Kernel = 2x2
Stride = (2,2)
Convolution
Kernel = 5x5
Stride = (1,1)
Filters = 64
Max Pooling
Kernel = 2x2
Stride = (2,2)
Full connection
Dropout
And Fully Connected

Review the code

GPU implementation

Supported devices

On a typical system, there are multiple computing devices. In TensorFlow, the supported device types are `CPU` and `GPU`. They are represented as `strings`. For example:

- `"/cpu:0"`: The CPU of your machine.
- `"/device:GPU:0"`: The GPU of your machine, if you have one.
- `"/device:GPU:1"`: The second GPU of your machine, etc.

If a TensorFlow operation has both CPU and GPU implementations, the GPU devices will be given priority when the operation is assigned to a device. For example, `matmul` has both CPU and GPU kernels. On a system with devices `cpu:0` and `gpu:0`, `gpu:0` will be selected to run `matmul`.

Manual Device Placement

1. **To perform particular operation on a device of your choice**
2. **with tf.device ()**
    1. Creates a context such that all operations with the context will have the same device assignment

Allowing GPU memory growth

- **By default, TF maps nearly all of the GPU memory of all GPUs visible to the process**

- **In some cases it is desirable for the process to only allocate a subset of the available memory, or to only grow the memory usage as it is needed by the process**

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```

Allow soft placement

If you would like TensorFlow to automatically choose an existing and supported device to run the operations in case the specified one doesn't exist, you can set **allow_soft_placement** to **True** in the configuration option when creating the session.

Websites

- https://www.tensorflow.org/api_docs/python/tf/ConfigProto
- https://www.tensorflow.org/guide/using_gpu
- https://www.tensorflow.org/api_docs/python/tf/device

Assignment 1

- **Change code 1 to run on a gpu**
  - Set allow growth and soft placement to True
  - Set the graph to be on the cpu, while optimization and loss calculation to be on the gpu
  - If you confront memory error on google colab when calculating the test accuracy, then please write the code to input the test data in batches

Keras

What is Keras?

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Why Keras?

- User friendly
- Modularity
- Easy Extensibility
- Work with python

Keras Structure

0. Import the necessary libraries

1. Read the input data; define parameters, constants..

2. Define model structure. (**Sequential** model: linear stack of layers, or more complicated **Keras functional API** user defined model)

3. Add layers of the model

4. Compile the model

5. Fit the model

6. Evaluate the model

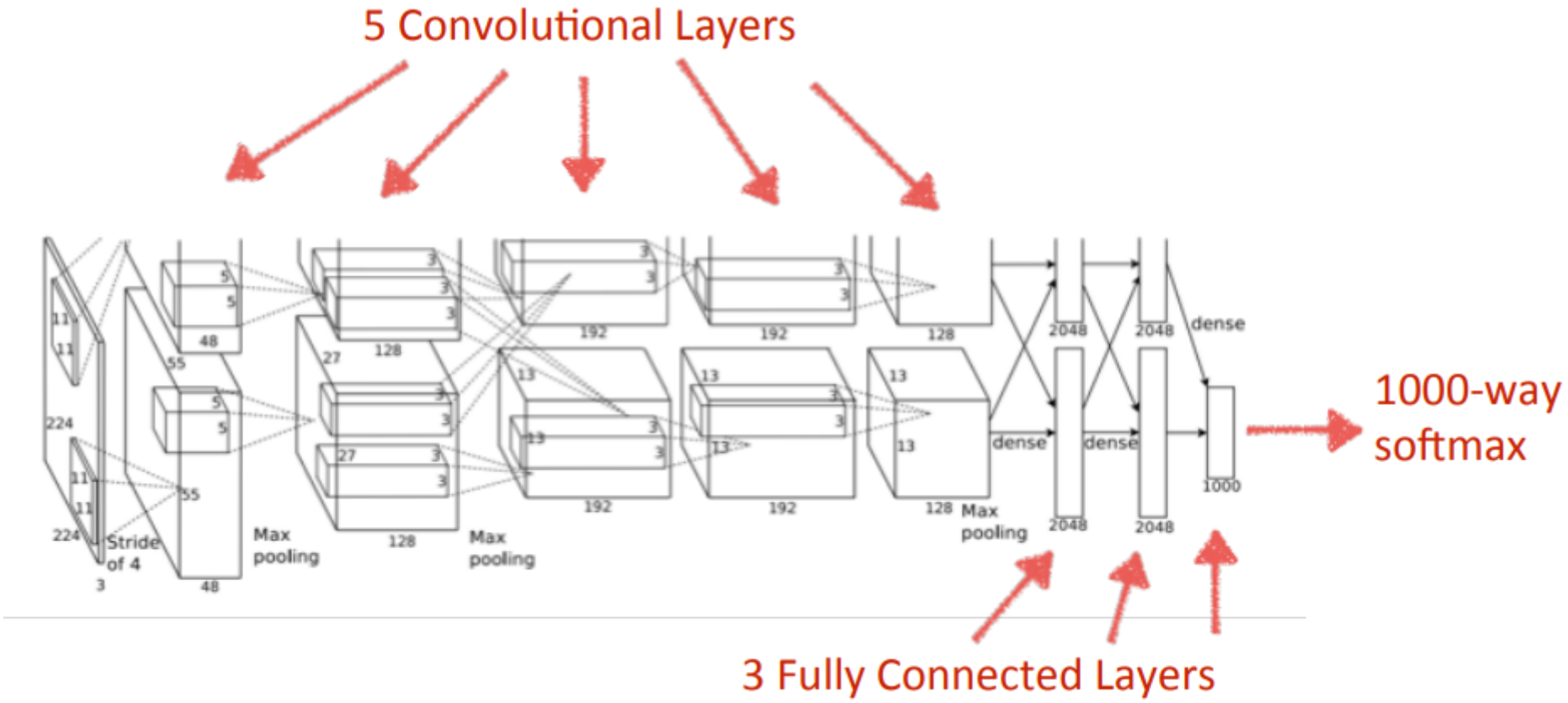
https://keras.io/

Let’s convert our TF code to Keras

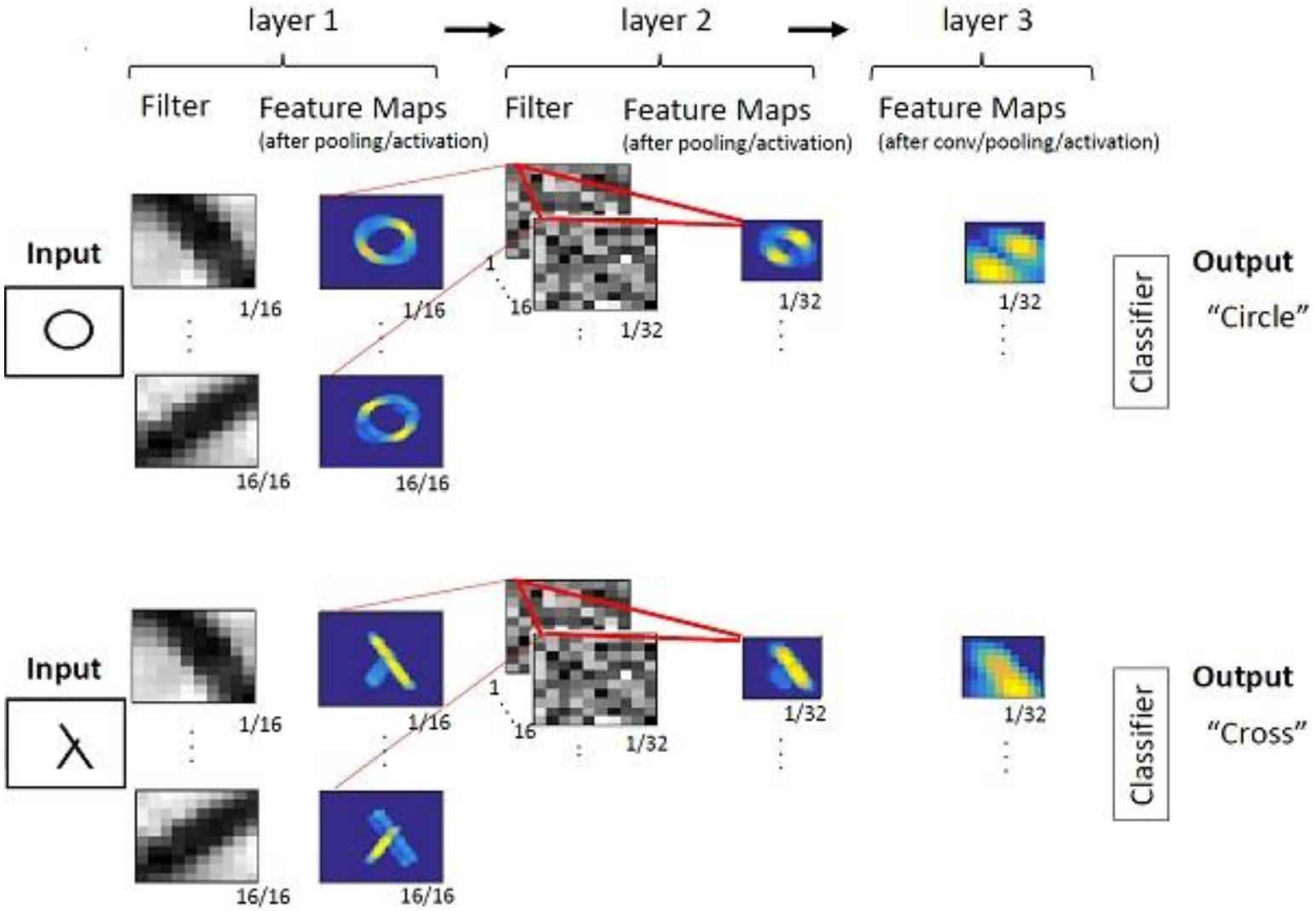
Show how to run Keras on GPU and Multi-GPUs

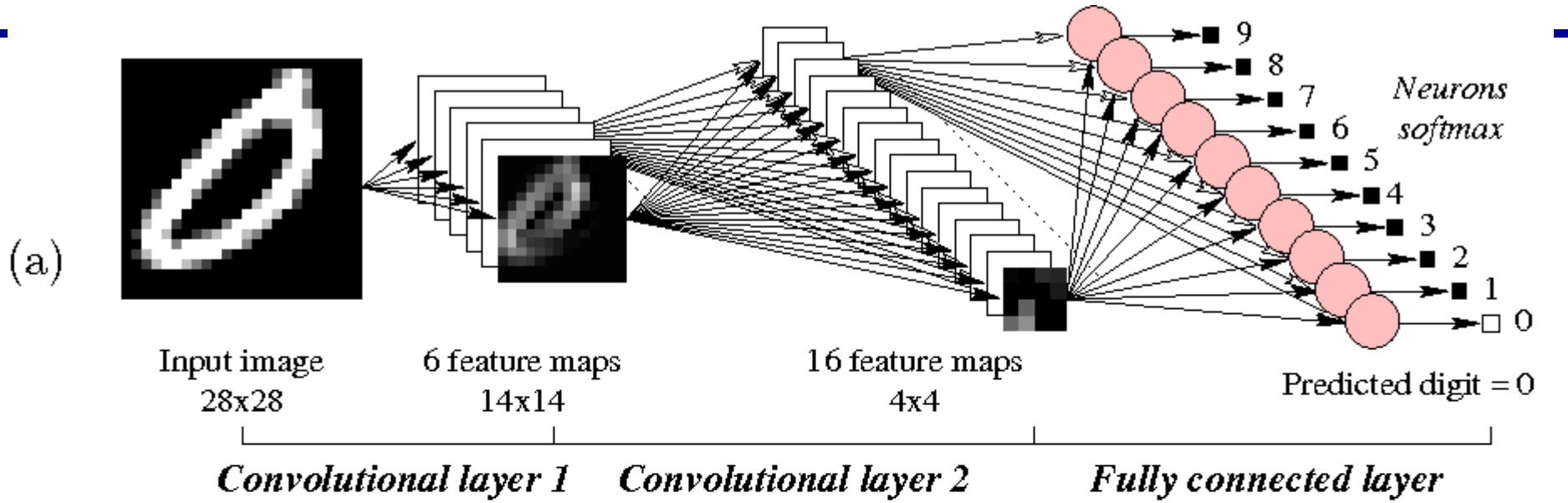
Assignment 2

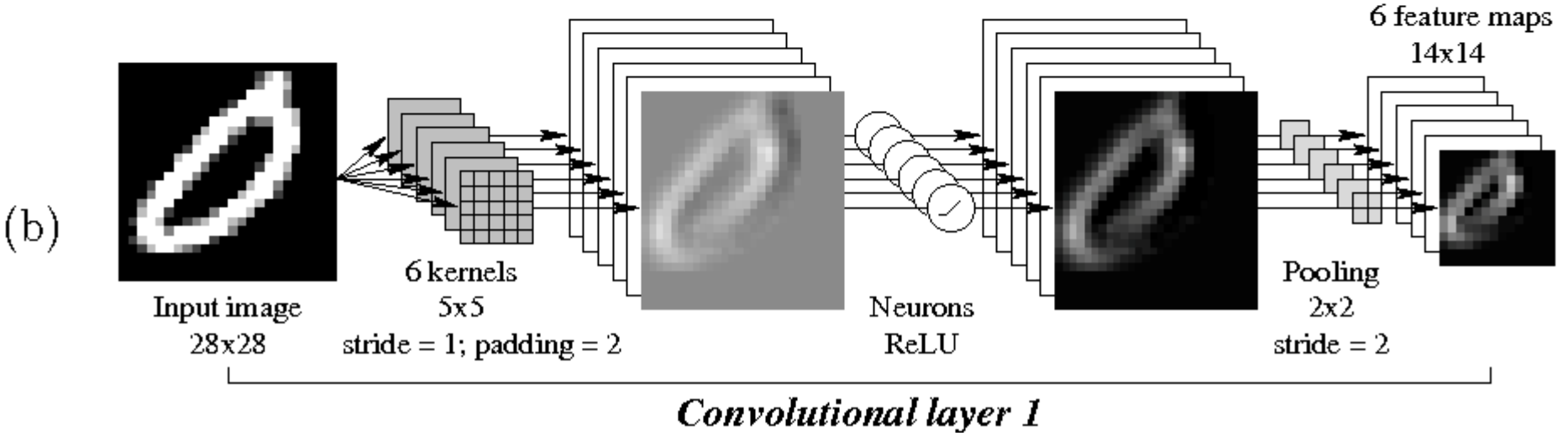
- Convert the tensorflow code for AlexNet into Keras



Backup


layer 1
layer 2
layer 3
Filter
Feature Maps
(after pooling/activation)
Filter
Feature Maps
(after pooling/activation)
Feature Maps
(after conv/pooling/activation)
Input
1/16
1/16
16/16
16/16
1
16
1/32
1/32
1/32
Classifier
Output
"Circle"
Input
1/16
1/16
16/16
16/16
1
16
1/32
1/32
1/32
Classifier
Output
"Cross"


(a)
9
8
7
6
5
4
3
2
1
0
Neurons
softmax
Input image
28x28
6 feature maps
14x14
16 feature maps
4x4
Predicted digit = 0
Convolutional layer 1
Convolutional layer 2
Fully connected layer


(b)
6 feature maps
14x14
Input image
28x28
6 kernels
5x5
stride = 1; padding = 2
Neurons
ReLU
Pooling
2x2
stride = 2
Convolutional layer 1

- `training`: Either a Python boolean, or a TensorFlow boolean scalar tensor (e.g. a placeholder). Whether to return the output in training mode (normalized with statistics of the current batch) or in inference mode (normalized with moving statistics). **NOTE**: make sure to set this parameter correctly, or else your training/inference will not work properly.

Batch Norm at test time

- **Training (mini-batch)**

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$s^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

- Testing (one-example)

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.