1. Import libraries

```
import os
import sys
import numpy as np
import gzip
import pandas as pd
from time import time
print("OS: ", sys.platform)
print("Python: ", sys.version)
# MXnet
import mxnet as mx
from mxnet import nd, autograd
from mxnet import gluon
from mxnet.gluon import nn
print("MXNet version", mx.__version__) # Matteo 1.5.1
# Tensorflow
from sklearn.model_selection import train_test_split
%tensorflow_version 2.x
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
print("Tensorflow version (by Google): ", tf.__version__)
```

```
OS:  linux
Python:  3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0]
MXNet version 1.5.1
TensorFlow 2.x selected.
Tensorflow version (by Google):  2.1.0
```

```
#! pip install mxnet
#!pip install mxnet-cu100mkl
```

```
Collecting mxnet-cu100mkl
  Downloading https://files.pythonhosted.org/packages/bb/eb/68921d5ffb80fd5cba483ab0dc955ed4aa257acc5c3b00c05dc03e3737c2/mxnet_cu100mkl-1.5.1.p
     |████████████████████████████████| 576.6MB 27kB/s
Requirement already satisfied: numpy<2.0.0,>1.16.0 in /tensorflow-2.1.0/python3.6 (from mxnet-cu100mkl) (1.18.1)
Requirement already satisfied: requests<3,>=2.20.0 in /tensorflow-2.1.0/python3.6 (from mxnet-cu100mkl) (2.23.0)
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.6/dist-packages (from mxnet-cu100mkl) (0.8.4)
Requirement already satisfied: idna<3,>=2.5 in /tensorflow-2.1.0/python3.6 (from requests<3,>=2.20.0->mxnet-cu100mkl) (2.9)
Requirement already satisfied: certifi>=2017.4.17 in /tensorflow-2.1.0/python3.6 (from requests<3,>=2.20.0->mxnet-cu100mkl) (2019.11.28)
Requirement already satisfied: chardet<4,>=3.0.2 in /tensorflow-2.1.0/python3.6 (from requests<3,>=2.20.0->mxnet-cu100mkl) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /tensorflow-2.1.0/python3.6 (from requests<3,>=2.20.0->mxnet-cu100mkl
Installing collected packages: mxnet-cu100mkl
Successfully installed mxnet-cu100mkl-1.5.1.post0
WARNING: The following packages were previously imported in this runtime:
  [mxnet]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
# Check cuda version
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:01_CDT_2018
Cuda compilation tools, release 10.0, V10.0.130
```

Set GPU usage

```
# MXNET
gpus = mx.test_utils.list_gpus()
print(gpus)
ctx =  [mx.gpu()] if gpus else [mx.cpu(0), mx.cpu(1)]
print(ctx)
```

```
range(0, 1)
[gpu(0)]
```

```
# TENSORFLOW
```

Control reproducibility

The most common form of randomness used in neural networks is the random initialization of the network weights. Although randomness can be used in other areas, here is just a short list:

- Randomness in Initialization, such as weights.
- Randomness in Regularization, such as dropout.
- Randomness in Layers, such as word embedding.
- Randomness in Optimization, such as stochastic optimization.

source: https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

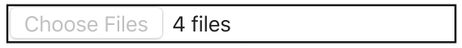
```
import random
np.random.seed(42)
random.seed(42)
for computing_unit in ctx:
    mx.random.seed(42, ctx = computing_unit)
tf.random.set_seed(42)
```

2. Read dataset - General Train/Test split

```
def read_mnist(images_path: str, labels_path: str):
    #mnist_path = "data/mnist/"
    #images_path = mnist_path + images_path
    print(images_path)
    with gzip.open(labels_path, 'rb') as labelsFile:
        labels = np.frombuffer(labelsFile.read(), dtype=np.uint8, offset=8)

    with gzip.open(images_path,'rb') as imagesFile:
        length = len(labels)
        # Load flat 28x28 px images (784 px), and convert them to 28x28 px
        features = np.frombuffer(imagesFile.read(), dtype=np.uint8, offset=16) \
                        .reshape(length, 784) \
                        .reshape(length, 28, 28, 1)
    return features, labels


from google.colab import files
uploaded = files.upload()
```



- **train-labels-idx1-ubyte.gz**(application/x-gzip) - 28881 bytes, last modified: 29/12/2019 - 100% done
- **train-images-idx3-ubyte.gz**(application/x-gzip) - 9912422 bytes, last modified: 29/12/2019 - 100% done
- **t10k-labels-idx1-ubyte.gz**(application/x-gzip) - 4542 bytes, last modified: 29/12/2019 - 100% done
- **t10k-images-idx3-ubyte.gz**(application/x-gzip) - 1648877 bytes, last modified: 29/12/2019 - 100% done

```
Saving train-labels-idx1-ubyte.gz to train-labels-idx1-ubyte.gz
Saving train-images-idx3-ubyte.gz to train-images-idx3-ubyte.gz
Saving t10k-labels-idx1-ubyte.gz to t10k-labels-idx1-ubyte.gz
Saving t10k-images-idx3-ubyte.gz to t10k-images-idx3-ubyte.gz
```

```
! ls
```

```
sample_data                 train-images-idx3-ubyte.gz
t10k-images-idx3-ubyte.gz   train-labels-idx1-ubyte.gz
t10k-labels-idx1-ubyte.gz
```

```
# LOAD TRAIN AND TEST ALREADY SPLIT
train = {}
test = {}
train['features'], train['labels'] = read_mnist('train-images-idx3-ubyte.gz', 'train-labels-idx1-ubyte.gz')
test['features'], test['labels'] = read_mnist('t10k-images-idx3-ubyte.gz', 't10k-labels-idx1-ubyte.gz')
print(test['features'].shape[0], '-> # of test images.')
print(train['features'].shape[0], '-> # of training images (train + validation).')
# CREATE TRAIN AND VALIDATION SPLIT
validation = {}
train['features'], validation['features'], train['labels'], validation['labels'] = train_test_split(train['features'], train['labels'], test_size=0
print("    ", train['features'].shape[0], '-> # of (actual) training images.')
print("    ", validation['features'].shape[0], '-> # of validation images.')
```

```
train-images-idx3-ubyte.gz
t10k-images-idx3-ubyte.gz
10000 -> # of test images.
60000 -> # of training images (train + validation).
     48000 -> # of (actual) training images.
     12000 -> # of validation images.
```

3. Create a reader for each Framework

```
# GENERAL PARAMETERS
EPOCHS = 15
BATCH_SIZE = 200
```

```
# MXNET
```

```
# MXNET
# convert from NHWC to NCHW that is used by MXNET
# https://stackoverflow.com/questions/37689423/convert-between-nhwc-and-nchw-in-tensorflow
X_train_mx = mx.ndarray.transpose(mx.nd.array(train['features']), axes=(0, 3, 1, 2))
y_train_mx = mx.nd.array(train['labels'])
X_validation_mx = mx.ndarray.transpose(mx.nd.array(validation['features']), axes=(0, 3, 1, 2))
y_validation_mx = mx.nd.array(validation['labels'])
X_test_mx = mx.ndarray.transpose(mx.nd.array(test['features']), axes=(0, 3, 1, 2))
y_test_mx = mx.nd.array(test['labels'])
# create data iterator
train_data_mx = mx.io.NDArrayIter(X_train_mx.asnumpy(), y_train_mx.asnumpy(), BATCH_SIZE, shuffle=True)
val_data_mx = mx.io.NDArrayIter(X_validation_mx.asnumpy(), y_validation_mx.asnumpy(), BATCH_SIZE)
test_data_mx = mx.io.NDArrayIter(X_test_mx.asnumpy(), y_test_mx.asnumpy(), BATCH_SIZE)
```

```
X_train_mx.shape
```

```
(48000, 1, 28, 28)
```

```
type(X_train_mx.asnumpy())
```

```
numpy.ndarray
```

```
# TENSORFLOW
# convert in multiple output for tensorflow
X_train_tf, y_train_tf = train['features'], to_categorical(train['labels'])
X_validation_tf, y_validation_tf = validation['features'], to_categorical(validation['labels'])
# create data generator
train_generator_tf = ImageDataGenerator().flow(X_train_tf, y_train_tf, batch_size=BATCH_SIZE)
validation_generator_tf = ImageDataGenerator().flow(X_validation_tf, y_validation_tf, batch_size=BATCH_SIZE)
```

```
X_train_tf.shape
```

```
(48000, 28, 28, 1)
```

4. Create models

```
# MXNET -> GLUON
# IDENTICAL TO LeNet paper: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
model_mx = nn.HybridSequential()
model_mx.add(nn.Conv2D(channels=6, kernel_size=5, activation='relu'),
        nn.AvgPool2D(pool_size=2, strides=2),
        nn.Conv2D(channels=16, kernel_size=3, activation='relu'),
```

```
        nn.AvgPool2D(pool_size=2, strides=2),
        nn.Flatten(),
        nn.Dense(120, activation="relu"),
        nn.Dense(84, activation="relu"),
        nn.Dense(10))


# TENSORFLOW -> KERAS
model_tf = keras.Sequential()
init_tf = tf.keras.initializers.GlorotNormal(seed=1)
model_tf.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(28,28,1), kernel_initializer = init_tf, bias_initializer =
model_tf.add(layers.AveragePooling2D(pool_size=(2, 2), strides=2))
model_tf.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu', kernel_initializer = init_tf, bias_initializer = init_tf))
model_tf.add(layers.AveragePooling2D(pool_size=(2, 2), strides=2))
model_tf.add(layers.Flatten())
model_tf.add(layers.Dense(units=120, activation='relu', kernel_initializer = init_tf, bias_initializer = init_tf))
model_tf.add(layers.Dense(units=84, activation='relu', kernel_initializer = init_tf, bias_initializer = init_tf))
model_tf.add(layers.Dense(units=10, activation = 'softmax', kernel_initializer = init_tf, bias_initializer = init_tf))
#model.summary()


#help(layers.Dense)
```

Optimization on/off

```
# MXNET
model_mx.hybridize()


# TENSORFLOW
tf.config.optimizer.set_jit(True)
```

5. Train Models

```
%%time
# MXNET
def training_procedure(handwritten_net, train_data):
    global EPOCHS
```

```
    global ctx
    handwritten_net.initialize(mx.init.Xavier(), ctx=ctx, force_reinit=True)
    #handwritten_net(init = mx.init.Xavier(), ctx=ctx)
    optim = mx.optimizer.Adam(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08, lazy_update=True)
    trainer = gluon.Trainer(handwritten_net.collect_params(), optim)
    # Use Accuracy as the evaluation metric.
    metric = mx.metric.Accuracy()
    softmax_cross_entropy_loss = gluon.loss.SoftmaxCrossEntropyLoss()

    for i in range(EPOCHS):
        # Reset the train data iterator.
        train_data.reset()
        # Loop over the train data iterator.
        for batch in train_data:
            # Splits train data into multiple slices along batch_axis
            # and copy each slice into a context.
            data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
            # Splits train labels into multiple slices along batch_axis
            # and copy each slice into a context.
            label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
            outputs = []
            # Inside training scope
            with autograd.record():
                for x, y in zip(data, label):
                    z = handwritten_net(x)
                    # Computes softmax cross entropy loss.
                    loss = softmax_cross_entropy_loss(z, y)
                    # Backpropogate the error for one iteration.
                    loss.backward()
                    outputs.append(z)
            # Updates internal evaluation
            metric.update(label, outputs)
            # Make one step of parameter update. Trainer needs to know the
            # batch size of data to normalize the gradient by 1/batch_size.
            trainer.step(batch.data[0].shape[0])
        # Gets the evaluation result.
        name, acc = metric.get()
        # Reset evaluation result to initial state.
        metric.reset()
        print('training acc at epoch %d: %s=%f'%(i, name, acc))
    return handwritten_net

trained_model_mx = training_procedure(model_mx, train_data_mx)
```

```
training acc at epoch 0: accuracy=0.877583
training acc at epoch 1: accuracy=0.967292
training acc at epoch 2: accuracy=0.977083
training acc at epoch 3: accuracy=0.983437
training acc at epoch 4: accuracy=0.985938
training acc at epoch 5: accuracy=0.987708
training acc at epoch 6: accuracy=0.990958
training acc at epoch 7: accuracy=0.991396
training acc at epoch 8: accuracy=0.992958
training acc at epoch 9: accuracy=0.994021
training acc at epoch 10: accuracy=0.993708
training acc at epoch 11: accuracy=0.993667
training acc at epoch 12: accuracy=0.993979
training acc at epoch 13: accuracy=0.994833
training acc at epoch 14: accuracy=0.996271
CPU times: user 17.1 s, sys: 2.58 s, total: 19.7 s
Wall time: 14.2 s
```

```
%%time
# TENSORFLOW
chosen_tf_optimizer = keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
model_tf.compile(loss=keras.losses.categorical_crossentropy, optimizer=chosen_tf_optimizer, metrics=['accuracy'])
steps_per_epoch = X_train_tf.shape[0]//BATCH_SIZE
validation_steps = X_validation_tf.shape[0]//BATCH_SIZE
model_tf.fit_generator(train_generator_tf, steps_per_epoch=steps_per_epoch, epochs=EPOCHS,
                       validation_data=validation_generator_tf, validation_steps=validation_steps,
                       shuffle=True, callbacks=[])
```

```
WARNING:tensorflow:From <timed exec>:8: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed ir
Instructions for updating:
Please use Model.fit, which supports generators.
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 240 steps, validate for 60 steps
Epoch 1/15
240/240 [==============================] - 12s 52ms/step - loss: 0.7762 - accuracy: 0.8671 - val_loss: 0.1152 - val_accuracy: 0.9642
Epoch 2/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0921 - accuracy: 0.9714 - val_loss: 0.0781 - val_accuracy: 0.9749
Epoch 3/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0639 - accuracy: 0.9808 - val_loss: 0.0691 - val_accuracy: 0.9785
Epoch 4/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0488 - accuracy: 0.9852 - val_loss: 0.0597 - val_accuracy: 0.9808
Epoch 5/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0399 - accuracy: 0.9879 - val_loss: 0.0641 - val_accuracy: 0.9800
Epoch 6/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0325 - accuracy: 0.9900 - val_loss: 0.0535 - val_accuracy: 0.9841
Epoch 7/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0275 - accuracy: 0.9913 - val_loss: 0.0472 - val_accuracy: 0.9864
Epoch 8/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0243 - accuracy: 0.9922 - val_loss: 0.0591 - val_accuracy: 0.9831
Epoch 9/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0226 - accuracy: 0.9929 - val_loss: 0.0510 - val_accuracy: 0.9846
Epoch 10/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0199 - accuracy: 0.9935 - val_loss: 0.0496 - val_accuracy: 0.9852
Epoch 11/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0179 - accuracy: 0.9936 - val_loss: 0.0546 - val_accuracy: 0.9850
Epoch 15/15
240/240 [==============================] - 2s 7ms/step - loss: 0.0099 - accuracy: 0.9969 - val_loss: 0.0498 - val_accuracy: 0.9877
CPU times: user 39.5 s, sys: 3.82 s, total: 43.3 s
Wall time: 35.2 s
```

6. Evaluate models

```
%%time
# MXNET
# TEST THE NETWORK
metric = mx.metric.Accuracy()
# Reset the test data iterator.
```

```
test_data_mx.reset()
# Loop over the test data iterator.
for batch in test_data_mx:
    # Splits test data into multiple slices along batch_axis
    # and copy each slice into a context.
    data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
    # Splits validation label into multiple slices along batch_axis
    # and copy each slice into a context.
    label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
    outputs = []
    for x in data:
        outputs.append(model_mx(x))
    # Updates internal evaluation
    metric.update(label, outputs)
print('MXnet - Test %s : %f'%metric.get())
assert metric.get()[1] > 0.90
```

```
MXnet - Test accuracy : 0.987800
CPU times: user 80.1 ms, sys: 16.3 ms, total: 96.3 ms
Wall time: 75.3 ms
```

```
%%time
# TENSORFLOW
score = model_tf.evaluate(test['features'], to_categorical(test['labels']), verbose=0)
#print('Test loss:', score[0])
print('TensorFlow - Test accuracy:', score[1])
assert score[1] > 0.90
```

```
TensorFlow - Test accuracy: 0.9872
CPU times: user 1.35 s, sys: 302 ms, total: 1.65 s
Wall time: 1.82 s
```