Colab and Google Drive Setup

```
import os, sys
from google.colab import drive
drive.mount('/content/mnt')
nb_path = '/content/notebooks'
try:
  os.symlink('/content/mnt/My Drive/Colab Notebooks', nb_path)
except FileExistsError:
  print("path already set")
sys.path.insert(0,nb_path)
```

```
Drive already mounted at /content/mnt; to attempt to forcibly remount, call drive.mount("/content/mnt", force_remount=True).
path already set
```

```
# call once then re-comment
#!pip install --target=$nb.path --upgrade urllib3==1.24
#!pip install --target=$nb.path --upgrade folium==0.2.1
# !pip install --target=$nb_path mxnet-cu100mkl
```

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 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
```

```
# 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
```

```
!nvidia-smi
```

```
Mon Mar  9 08:32:33 2020       
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.59       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   69C    P8    34W / 149W |      0MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

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
tf.config.experimental.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

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
    #images_path = mnist_path + images_path
    folder = os.getcwd() + "/notebooks/"
    print(images_path)
    with gzip.open(folder + labels_path, 'rb') as labelsFile:
        labels = np.frombuffer(labelsFile.read(), dtype=np.uint8, offset=8)

    with gzip.open(folder + 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

# upload the 4 data file in the same folder of notebooks on your drive and check that they are
# lested below
os.listdir(os.getcwd() + "/notebooks")
```

```
['kernel.ipynb',
 'Untitled0.ipynb',
 'Untitled1.ipynb',
 'Untitled2.ipynb',
 'ComparisonOfClusteringMethods (1).ipynb',
 'HierarchicalClustering.ipynb',
 'ComparisonOfClusteringMethods.ipynb',
 'SilhouetteAnalysisIris.ipynb',
 'SilhouetteAnalysis.ipynb',
 'Untitled3.ipynb',
 'CPU - Fundamental Ops - Linear Algebra.ipynb',
 'GPU - Fundamental Ops - Linear Algebra.ipynb',
 'Copy of 0_colab',
 'Copy of 1_data',
 'Copy of 2_keras',
 'Copy of 4_predict',
 'Copy of 3_eager',
 'Teardown - MXnet - TF - Optimization.ipynb',
 'graphviz-0.8.4.dist-info',
 'chardet-3.0.4.dist-info',
 'urllib3',
 'dmlc_tracker',
 'urllib3-1.25.8.dist-info',
 'requests',
 'requests-2.23.0.dist-info',
 'numpy',
 'chardet',
 'graphviz',
 'idna',
 'mxnet',
 'mxnet_cu100mkl-1.5.1.post0.dist-info',
 'certifi',
 'certifi-2019.11.28.dist-info',
 'idna-2.9.dist-info',
 'numpy-1.18.1.dist-info',
 'bin',
 't10k-labels-idx1-ubyte.gz',
 't10k-images-idx3-ubyte.gz',
 'train-images-idx3-ubyte.gz',
 'train-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
# 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
# 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
def get_mxnet(optimized = True):
  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))
  if optimized:
    model_mx.hybridize()
  return model_mx


# TENSORFLOW -> KERAS
def get_tensorflow(optimized = True):
  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_initialize
  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()
  tf.keras.backend.clear_session()
  tf.config.optimizer.set_jit(optimized)
  return model_tf

#help(layers.Dense)
```

Optimization on/off

```
# MXNET
#model_mx.hybridize()

# TENSORFLOW
#tf.keras.backend.clear_session()
#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): # ad the warmup of tensorflow
        # 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)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.15 µs
```

```
# MXNET NO OPTIMIZATON
model_mx = get_mxnet(False)
```

```
%%time
trained_model_mx = training_procedure(model_mx, train_data_mx)
```

```
training acc at epoch 0: accuracy=0.877583
training acc at epoch 1: accuracy=0.967000
training acc at epoch 2: accuracy=0.976938
training acc at epoch 3: accuracy=0.983229
training acc at epoch 4: accuracy=0.986417
training acc at epoch 5: accuracy=0.987854
training acc at epoch 6: accuracy=0.990479
training acc at epoch 7: accuracy=0.991604
training acc at epoch 8: accuracy=0.993021
training acc at epoch 9: accuracy=0.993271
training acc at epoch 10: accuracy=0.994396
training acc at epoch 11: accuracy=0.993729
training acc at epoch 12: accuracy=0.994271
training acc at epoch 13: accuracy=0.995896
training acc at epoch 14: accuracy=0.996375
CPU times: user 35 s, sys: 7.08 s, total: 42.1 s
Wall time: 29.2 s
```

```
# MXNET OPTIMIZED
model_mx = get_mxnet(True)
```

```
%%time
trained_model_mx = training_procedure(model_mx, train_data_mx)
```

```
training acc at epoch 0: accuracy=0.880812
training acc at epoch 1: accuracy=0.970437
training acc at epoch 2: accuracy=0.979250
training acc at epoch 3: accuracy=0.983146
training acc at epoch 4: accuracy=0.986625
training acc at epoch 5: accuracy=0.988000
training acc at epoch 6: accuracy=0.990146
training acc at epoch 7: accuracy=0.990958
training acc at epoch 8: accuracy=0.992625
training acc at epoch 9: accuracy=0.993167
training acc at epoch 10: accuracy=0.993729
training acc at epoch 11: accuracy=0.994938
training acc at epoch 12: accuracy=0.995146
training acc at epoch 13: accuracy=0.995000
training acc at epoch 14: accuracy=0.994146
CPU times: user 25.2 s, sys: 4.99 s, total: 30.2 s
Wall time: 21 s
```

```
# TENSORFLOW WARMUP - NO OPTIMIZED
model_tf = get_tensorflow(False)
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
initial_weights = model_tf.get_weights()
model_tf.fit(train_generator_tf, steps_per_epoch=steps_per_epoch, epochs=1,
                    validation_data=validation_generator_tf, validation_steps=validation_steps,
                    shuffle=True, callbacks=[])
model_tf.set_weights(initial_weights)
```

```
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
240/240 [==============================] - 4s 17ms/step - loss: 0.7905 - accuracy: 0.8557 - val_loss: 0.1228 - val_accuracy: 0.9635
```

```
%%time
# TENSORFLOW NO OPTIMIZED
model_tf.fit(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: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 [==============================] - 2s 10ms/step - loss: 0.7191 - accuracy: 0.8703 - val_loss: 0.1063 - val_accuracy: 0.9672
Epoch 2/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0922 - accuracy: 0.9719 - val_loss: 0.0769 - val_accuracy: 0.9758
Epoch 3/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0677 - accuracy: 0.9789 - val_loss: 0.0675 - val_accuracy: 0.9786
Epoch 4/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0542 - accuracy: 0.9830 - val_loss: 0.0565 - val_accuracy: 0.9827
Epoch 5/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0448 - accuracy: 0.9859 - val_loss: 0.0519 - val_accuracy: 0.9840
Epoch 6/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0376 - accuracy: 0.9881 - val_loss: 0.0542 - val_accuracy: 0.9844
Epoch 7/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0316 - accuracy: 0.9898 - val_loss: 0.0522 - val_accuracy: 0.9849
Epoch 8/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0268 - accuracy: 0.9916 - val_loss: 0.0524 - val_accuracy: 0.9848
Epoch 9/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0251 - accuracy: 0.9918 - val_loss: 0.0612 - val_accuracy: 0.9816
Epoch 10/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0230 - accuracy: 0.9924 - val_loss: 0.0538 - val_accuracy: 0.9844
Epoch 11/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0207 - accuracy: 0.9926 - val_loss: 0.0574 - val_accuracy: 0.9853
Epoch 12/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0170 - accuracy: 0.9943 - val_loss: 0.0532 - val_accuracy: 0.9856
Epoch 13/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0155 - accuracy: 0.9947 - val_loss: 0.0531 - val_accuracy: 0.9860
Epoch 14/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0141 - accuracy: 0.9950 - val_loss: 0.0585 - val_accuracy: 0.9847
Epoch 15/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0141 - accuracy: 0.9949 - val_loss: 0.0618 - val_accuracy: 0.9843
CPU times: user 46 s, sys: 5.67 s, total: 51.6 s
Wall time: 35.7 s
<tensorflow.python.keras.callbacks.History at 0x7fd17721e908>
```

```
# TENSORFLOW WARMUP - OPTIMIZED
model_tf = get_tensorflow(True)
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
validation_steps = X_validation_tf.shape[0]//BATCH_SIZE
initial_weights = model_tf.get_weights()
model_tf.fit(train_generator_tf, steps_per_epoch=steps_per_epoch, epochs=1,
                    validation_data=validation_generator_tf, validation_steps=validation_steps,
                    shuffle=True, callbacks=[])
model_tf.set_weights(initial_weights)
```

```
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
240/240 [==============================] - 6s 25ms/step - loss: 0.7775 - accuracy: 0.8647 - val_loss: 0.1216 - val_accuracy: 0.9617
```

```
%%time
# TENSORFLOW - OPTIMIZED
model_tf.fit(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: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 [==============================] - 2s 10ms/step - loss: 0.7240 - accuracy: 0.8737 - val_loss: 0.1081 - val_accuracy: 0.9664
Epoch 2/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0900 - accuracy: 0.9716 - val_loss: 0.0760 - val_accuracy: 0.9757
Epoch 3/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0634 - accuracy: 0.9806 - val_loss: 0.0663 - val_accuracy: 0.9788
Epoch 4/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0516 - accuracy: 0.9844 - val_loss: 0.0716 - val_accuracy: 0.9754
Epoch 5/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0421 - accuracy: 0.9873 - val_loss: 0.0561 - val_accuracy: 0.9823
Epoch 6/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0344 - accuracy: 0.9891 - val_loss: 0.0629 - val_accuracy: 0.9806
Epoch 7/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0320 - accuracy: 0.9896 - val_loss: 0.0602 - val_accuracy: 0.9819
Epoch 8/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0266 - accuracy: 0.9912 - val_loss: 0.0505 - val_accuracy: 0.9845
Epoch 9/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0214 - accuracy: 0.9930 - val_loss: 0.0527 - val_accuracy: 0.9852
Epoch 10/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0217 - accuracy: 0.9929 - val_loss: 0.0577 - val_accuracy: 0.9837
Epoch 11/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0181 - accuracy: 0.9940 - val_loss: 0.0604 - val_accuracy: 0.9837
Epoch 12/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0166 - accuracy: 0.9945 - val_loss: 0.0584 - val_accuracy: 0.9841
Epoch 13/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0174 - accuracy: 0.9942 - val_loss: 0.0602 - val_accuracy: 0.9846
Epoch 14/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0121 - accuracy: 0.9961 - val_loss: 0.0554 - val_accuracy: 0.9857
Epoch 15/15
240/240 [==============================] - 2s 10ms/step - loss: 0.0117 - accuracy: 0.9960 - val_loss: 0.0649 - val_accuracy: 0.9827
CPU times: user 47.4 s, sys: 4.05 s, total: 51.5 s
Wall time: 36 s
<tensorflow.python.keras.callbacks.History at 0x7fd1770869e8>
```

6. Evaluate models

```
%%time
# MXNET
# 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 : 0.985000
CPU times: user 103 ms, sys: 39.3 ms, total: 142 ms
Wall time: 116 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: 0.9851
CPU times: user 1.54 s, sys: 223 ms, total: 1.77 s
Wall time: 1.7 s
```