COMPUTATIONAL INTELLIGENCE
DEEP LEARNING

Introduction to Keras

Adrian Horzyk

horzyk@agh.edu.pl

AGH University of
Science and Technology
Krakow, Poland

What is Keras?

Keras developed by François Chollet:

- Is an official high-level and high-performing API of TensorFlow used to specify and train different programs.
- Runs on top of TensorFlow, Theano, MXNet, or CNTK.
- Builds models by stacking layers and connecting graphs.
- Is actively developed by thousands of contributors across the world, e.g. Microsoft, Google, Nvidia, AWS.
- Is used by hundred thousands of developers, e.g. NetFlix, Uber, Google, Huawei, NVidia.
- Has good amount of documentation and easy to grasp all concepts.
- Supports GPU both of Nvidia and AMD and runs seamlessly on CPU and GPU.
- Is multi-platform (Python, R) and multi-backend.
- Allows for fast prototyping and leaves freedom to design and architecture

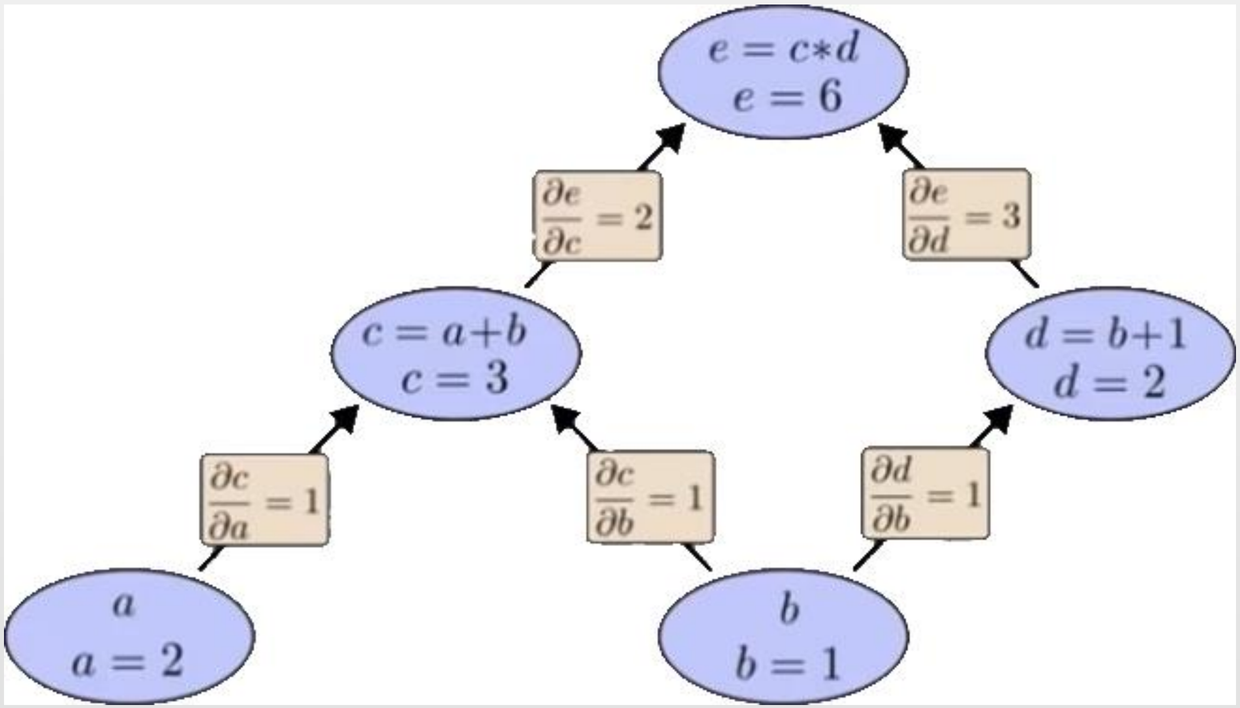
Keras Positive User Experience

Keras:

- **Follows best practices for reducing cognitive load**
- **Offers consistent and simple APIs.**
- **Minimizes the number of user actions required for common use cases.**
- **Provides a clear feedback upon user errors.**
- **More productive than many other frameworks.**
- **Integrates with lower-level Deep Learning languages like TensorFlow or Theano.**
- **Implements everything which was built in base language, e.g. TensorFlow.**
- **Produces models using GPU acceleration for various system like Windows, Linux, Android, iOS, Raspberry Pi.**

Working Principle

Keras is based on Computational Graphs like:

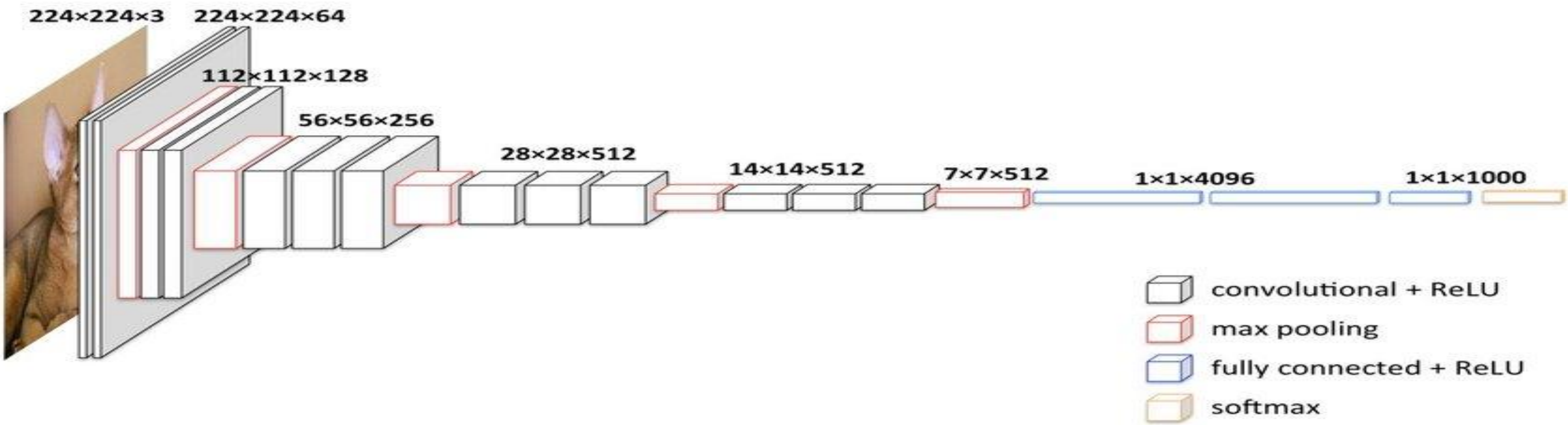


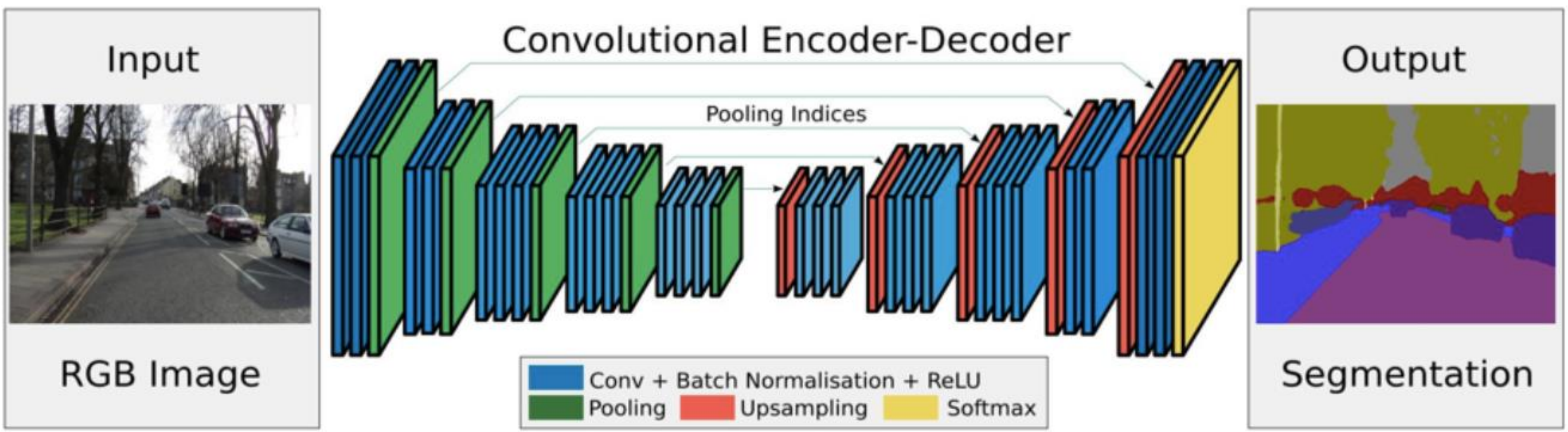
Where "a" and "b" are inputs used to compute "e" as an output using intermediate variables "c" and "d".

Computational Graphs allow to express complex expressions as a combination of simple operations.

Keras Sequential Models

We can create various sequential models which linearly stack layers and can be used for classification networks or autoencoders (consisting of encoders and decoders) like:

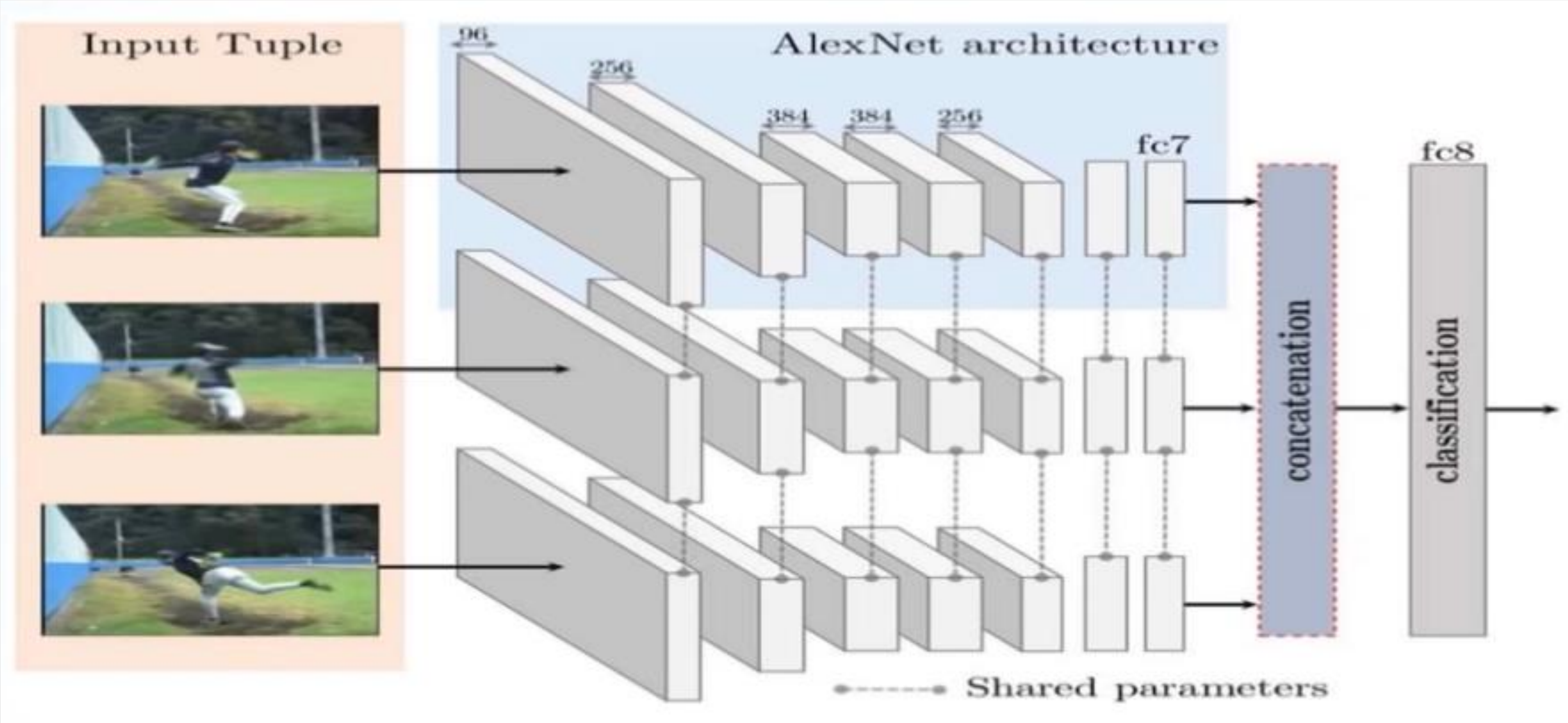




Keras Functional Models

Keras models can:

- Use multi-input, multi-output and arbitrary static graph topologies,
- Branch into two or more submodels,
- Share layers and/or weights.



Two Types of Execution of Keras Models

We can execute Keras model in two ways:

1. **Deferred (symbolic)**
   - Using Python to build a computational graph, next compiling and executing it.
   - Symbolic tensors **don't have a value** in the Python code.
2. **Eager (imperative)**
   - Here the Python runtime is the execution runtime, which is similar to the execution with Numpy.
   - Eager tensors **have a value** in the Python code.
   - With the eager execution, **value-dependent dynamic topologies** (tree-RNNs) can be constructed and used.

5 steps to implement a Neural Network

1. **Prepare Input (e.g. text, audio, images, video)** and specify the input dimension (size).
2. **Define the Model:** its architecture, build the computational graph, define sequential or functional style of the model and the kind of the network (MLP, CNN, RNN etc.).
3. **Specify the Optimizers** (Stochastic Gradient Descent (SGD), Root Mean Square (RMSprop), Adam etc.) to configure the learning process.
4. **Define the Loss Function (e.g. Mean Square Error (MSE), Cross Entropy, Hinge)** for checking the accuracy of the achieved prediction to adapt and improve the model.
5. **Train using training data, Test using testing/validation data, and Evaluate the Model.**

Installing TensorFlow and Keras

To start working with TensorFlow and Keras in Jupyter Notebook, you have to install them using the following commands in the Anaconda Prompt window:

conda install pip   # install pip in the virtual environment

pip install --upgrade tensorflow   # for python 2.7

pip3 install --upgrade tensorflow   # for python 3.*

It is recommended to install tensorflow with parameter –gpu to use GPU unit and make computations faster:

pip install tensorflow-gpu

$ pip install Keras

If successfully installed check in Jupyter Notebook the version of the TensorFlow using:

```
In [3]:  import tensorflow as tf
         print ("TensorFlow version: " + tf.__version__)

         TensorFlow version: 2.1.0
```

Implementing a CNN using Keras

We will try to create and train a simple Convolutional Neural Network (CNN) to tackle with handwritten digit classification problem using MNIST dataset:

Each image in the MNIST dataset is 28x28 pixels and contains a centred, grayscale digit form 0 to 9. Our goal is to classify these images to one of the ten classes using ten output neurons of the CNN network.

Simple CNN for MNIST classification

```python
'''Trains a simple ConvNet on the MNIST dataset. It gets more than 99% test accuracy after 12 epochs
(but there is still a lot of margin for parameter tuning). Training can take a few minutes!'''

# Import libraries
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

# Define hyperparameters
batch_size = 128
num_classes = 10
epochs = 12

# Input image dimensions
img_rows, img_cols = 28, 28

# Split the data between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

Simple CNN for MNIST classification

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define the sequential Keras model composed of a few layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu', input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model using optimizer
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])

# Train the model, validate, evaluate, and present scores
model.fit(x_train, y_train,batch_size=batch_size, epochs=epochs,
          verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Results of CNN MNIST classification

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 54s 900us/step - loss: 0.2651 - accuracy: 0.9186 - val_loss: 0.0572 - val_acc
uracy: 0.9807
Epoch 2/12
60000/60000 [==============================] - 58s 961us/step - loss: 0.0884 - accuracy: 0.9734 - val_loss: 0.0397 - val_acc
uracy: 0.9867
Epoch 3/12
60000/60000 [==============================] - 61s 1ms/step - loss: 0.0669 - accuracy: 0.9799 - val_loss: 0.0342 - val_accur
acy: 0.9883
Epoch 4/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0547 - accuracy: 0.9834 - val_loss: 0.0303 - val_accur
acy: 0.9902
Epoch 5/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0453 - accuracy: 0.9862 - val_loss: 0.0283 - val_accur
acy: 0.9909
Epoch 6/12
60000/60000 [==============================] - 60s 992us/step - loss: 0.0406 - accuracy: 0.9878 - val_loss: 0.0287 - val_acc
uracy: 0.9901
Epoch 7/12
60000/60000 [==============================] - 60s 998us/step - loss: 0.0365 - accuracy: 0.9887 - val_loss: 0.0285 - val_acc
uracy: 0.9909
Epoch 8/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0346 - accuracy: 0.9897 - val_loss: 0.0278 - val_accur
acy: 0.9902
Epoch 9/12
60000/60000 [==============================] - 62s 1ms/step - loss: 0.0310 - accuracy: 0.9903 - val_loss: 0.0382 - val_accur
acy: 0.9884
Epoch 10/12
60000/60000 [==============================] - 60s 995us/step - loss: 0.0308 - accuracy: 0.9906 - val_loss: 0.0277 - val_acc
uracy: 0.9913
Epoch 11/12
60000/60000 [==============================] - 63s 1ms/step - loss: 0.0295 - accuracy: 0.9908 - val_loss: 0.0259 - val_accur
acy: 0.9919
Epoch 12/12
60000/60000 [==============================] - 60s 1ms/step - loss: 0.0271 - accuracy: 0.9916 - val_loss: 0.0271 - val_accur
acy: 0.9919
Test loss: 0.027094655736458435
Test accuracy: 0.9919000267982483
```

Hyperparameters for MNIST classification

When dealing with deep learning models, we should modify hyperparameters of the model to get better performance of the model, e.g. for MNIST:

```
'''Trains a simple ConvNet on the MNIST dataset. It gets 99.64% test accuracy after 59 epochs
(but there is still a lot of margin for parameter tuning). Training can take a few hours!'''

# Import libraries
from __future__ import print_function
import os
#os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
import matplotlib.pyplot as plt
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau

# Define hyperparameters
batch_size = 128
num_classes = 10
epochs = 200

# Input image dimensions
img_rows, img_cols = 28, 28

# Split the data between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

Hyperparameters for MNIST classification

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define the sequential Keras model composed of a few layers
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu', input_shape=input_shape))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))
model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.30))
model.add(Conv2D(128,(3, 3), activation='relu',padding='same'))
model.add(Conv2D(128,(3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.40))
model.add(Conv2D(256,(3, 3), activation='relu',padding='same'))
model.add(Conv2D(256,(3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.50))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.35))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))
```

Hyperparameters for MNIST classification

```
# Compile the model using optimizer
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

# Learning rate
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.0001)

# Augmentation of training data
datagen = ImageDataGenerator(
        rotation_range=5, # rotate images
        zoom_range=0.2, # zoom images
        width_shift_range=0.15,  # shift images horizontally
        height_shift_range=0.15)  # shift images vertically
datagen.fit(x_train)

# Train the model, validate, evaluate, and present scores
'''model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))'''

# Train the model, validate, evaluate, and present scores
history = model.fit_generator(datagen.flow(x_train, y_train,batch_size=batch_size),
                              epochs=epochs,
                              verbose=1,
                              validation_data=(x_test, y_test),
                              steps_per_epoch=x_train.shape[0]//batch_size,
                              callbacks=[learning_rate_reduction])

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Hyperparameters for MNIST classification

```
# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc ='upper left')
plt.show()
```

```
racy: 0.9953
Epoch 54/200
468/468 [==============================] - 360s 768ms/step - loss: 0.0289 - accuracy: 0.9928 - val_loss: 0.0210 - val_accu
racy: 0.9956
Epoch 55/200
468/468 [==============================] - 362s 773ms/step - loss: 0.0311 - accuracy: 0.9929 - val_loss: 0.0209 - val_accu
racy: 0.9962
Epoch 56/200
468/468 [==============================] - 359s 767ms/step - loss: 0.0326 - accuracy: 0.9929 - val_loss: 0.0269 - val_accu
racy: 0.9940
Epoch 57/200

468/468 [==============================] - 357s 762ms/step - loss: 0.0305 - accuracy: 0.9932 - val_loss: 0.0208 - val_accu
racy: 0.9950
Epoch 58/200
468/468 [==============================] - 357s 763ms/step - loss: 0.0303 - accuracy: 0.9930 - val_loss: 0.0220 - val_accu
racy: 0.9950
Epoch 59/200
468/468 [==============================] - 349s 746ms/step - loss: 0.0317 - accuracy: 0.9927 - val_loss: 0.0182 - val_accu
racy: 0.9964
```

Let’s start with powerful computations!

- ✓ Questions?
- ✓ Remarks?
- ✓ Suggestions?
- ✓ Wishes?

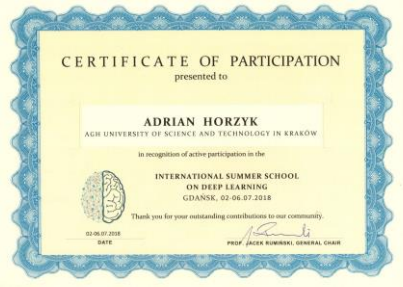
Bibliography and Literature

1. https://www.youtube.com/watch?v=XNKeayZW4dY
2. https://victorzhou.com/blog/keras-cnn-tutorial/
3. https://github.com/keras-team/keras/tree/master/examples
4. https://medium.com/@margaretmz/anaconda-jupyter-notebook-tensorflow-and-keras-b91f381405f8
5. https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html
6. http://coursera.org/specializations/tensorflow-in-practice
7. https://udacity.com/course/intro-to-tensorflow-for-deep-learning

Adrian Horzyk

horzyk@agh.edu.pl

Google: Horzyk



CERTIFICATE OF PARTICIPATION

presented to

ADRIAN HORZYK

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY IN KRAKÓW

in recognition of active participation in the

INTERNATIONAL SUMMER SCHOOL ON DEEP LEARNING

GDAŃSK, 02-06.07.2018

Thank you for your outstanding contributions to our community.

02-06.07.2018 DATE

PROF. JACEK RUMIŃSKI, GENERAL CHAIR

University of Science and Technology in Krakow, Poland