Keras is an open source deep-learning API for *Python*

Original author - François Chollet
Today - various developers
including *Google*'s *TensorFlow* team (since 2017)

Initial release - 27 March 2015
Stable release - 2.0.8 / 24 August 2017

Optional backends:
TensorFlow, *CNTK*, *Theano*

Motivation

Before:

- Undedicated libraries – self implementation
- Long unreadable code, multi-file (*Caffe* for example)
- Limited wrappers like *Lasagne*

With Keras:

- Simple syntax & full control, one *.py run file
- Includes all SOT NN layers & functions
- building complex graphs with ease
- growing community of contributors

(academia & industry)

Getting started

Documentation:
keras.io

Blog:
blog.keras.io

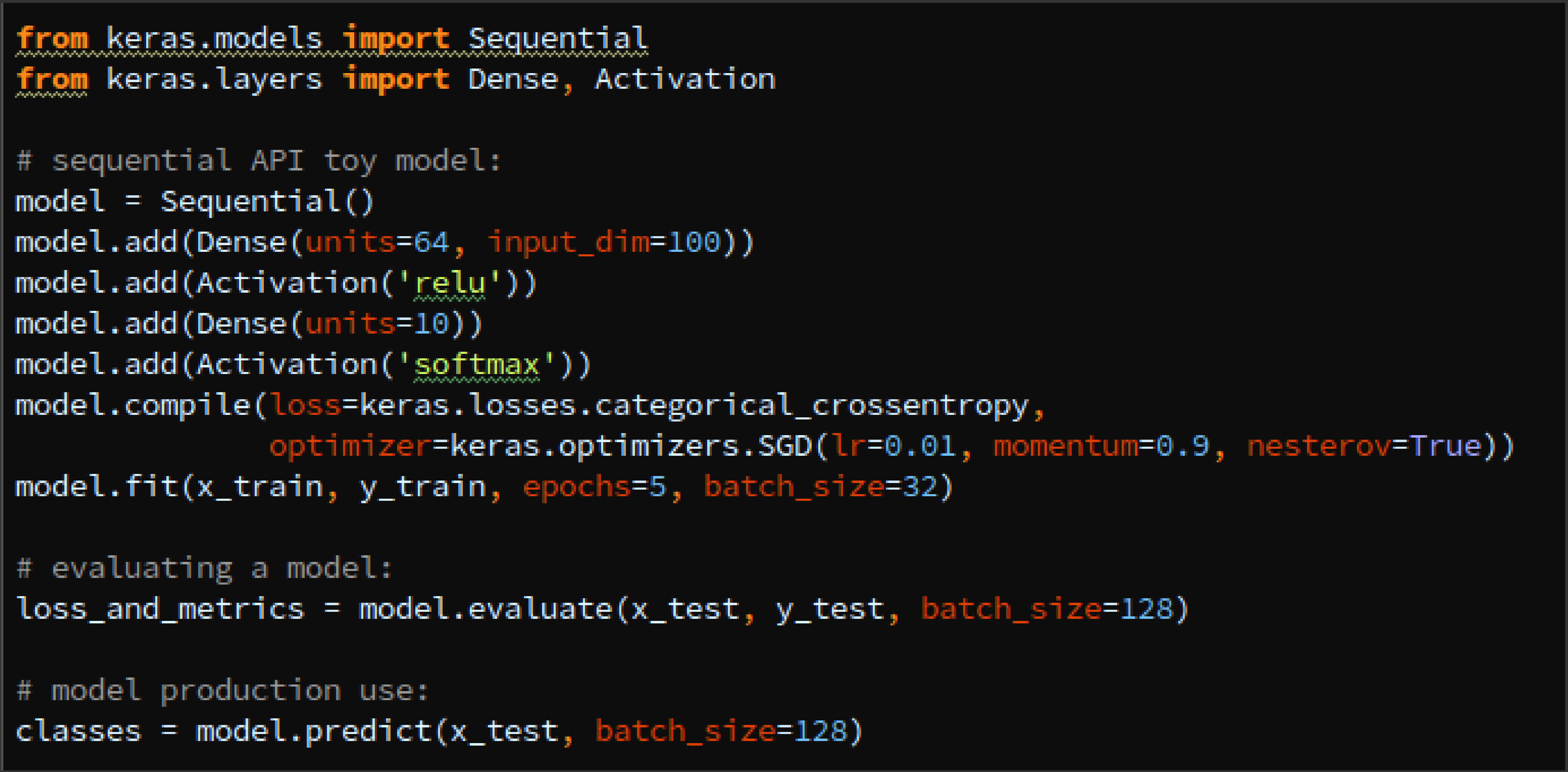
Installation: (or directly from git)

```
$ sudo pip install keras
```

(after pre-requisits & backend)

CUDA is supported by backends, and will be used if installed, unless flagged out.

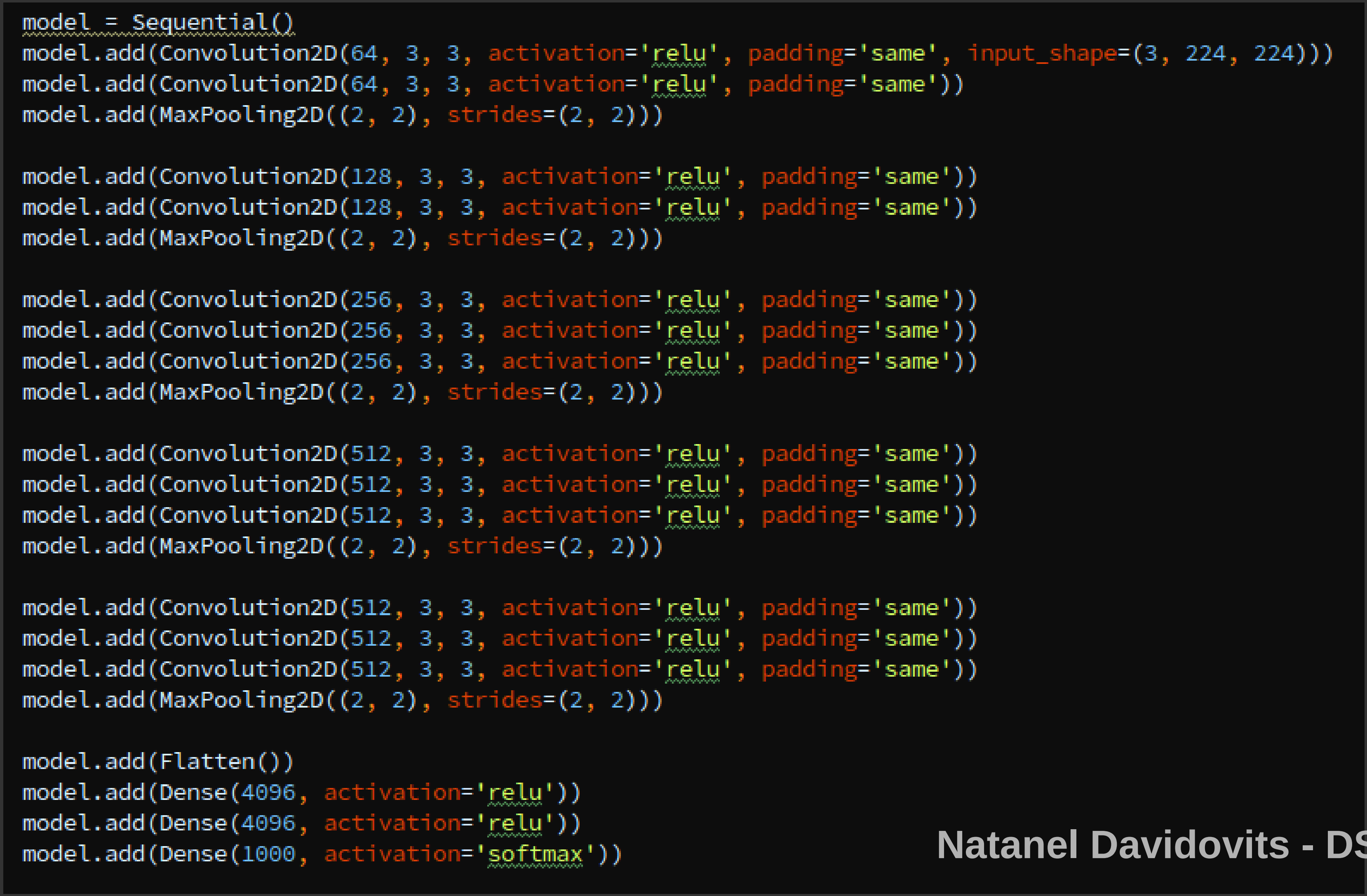
Getting started



```python
from keras.models import Sequential
from keras.layers import Dense, Activation

# sequential API toy model:
model = Sequential()
model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
model.fit(x_train, y_train, epochs=5, batch_size=32)

# evaluating a model:
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

# model production use:
classes = model.predict(x_test, batch_size=128)
```

Two APIs in Keras

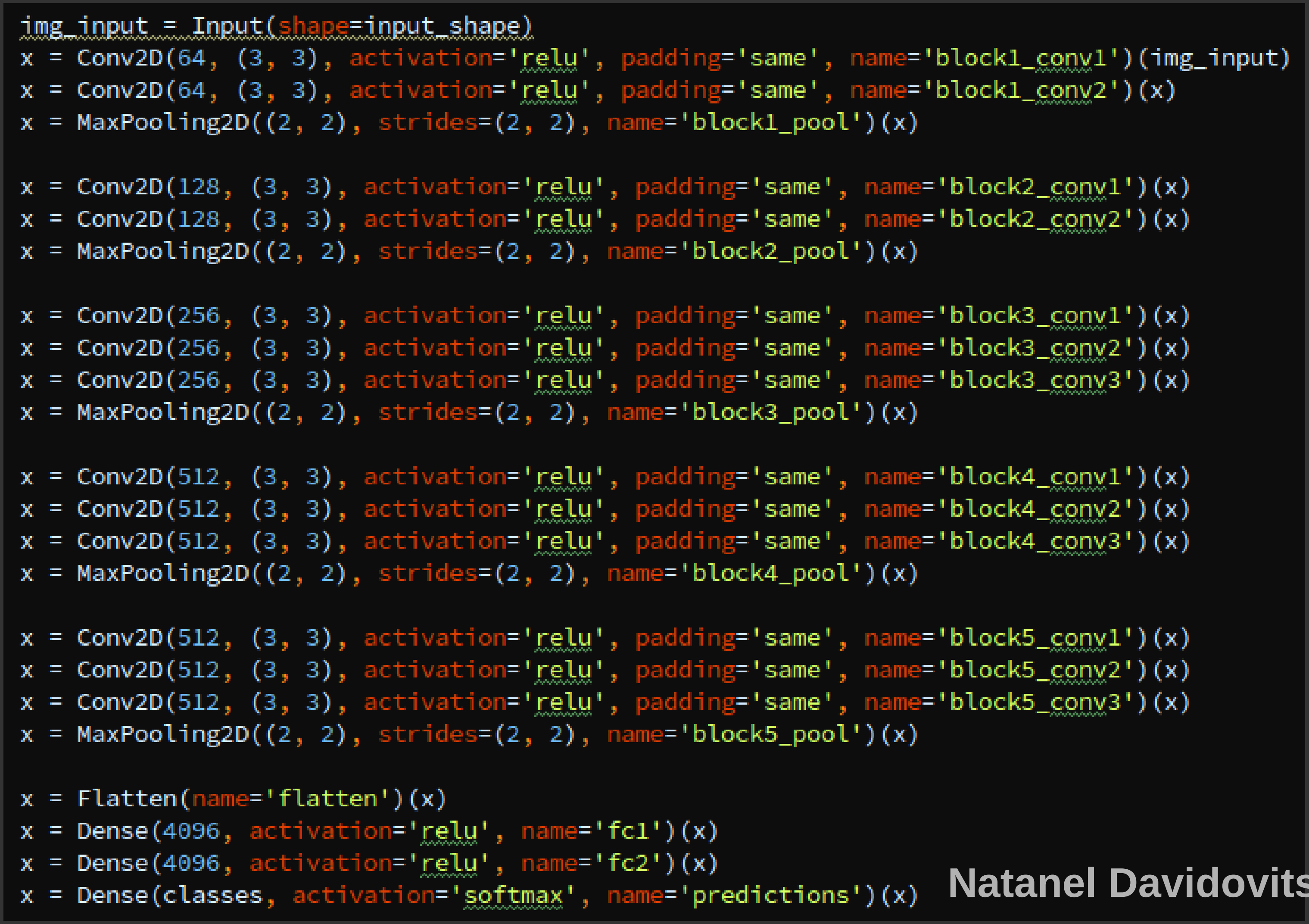
Sequential API - VGG16 example:



```
model = Sequential()
model.add(Convolution2D(64, 3, 3, activation='relu', padding='same', input_shape=(3, 224, 224)))
model.add(Convolution2D(64, 3, 3, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(128, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(128, 3, 3, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(256, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(256, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(256, 3, 3, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(512, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(512, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(512, 3, 3, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Convolution2D(512, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(512, 3, 3, activation='relu', padding='same'))
model.add(Convolution2D(512, 3, 3, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1000, activation='softmax'))
```

Two APIs in Keras

Functional API - VGG16 example:



```python
img_input = Input(shape=input_shape)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(img_input)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)

x = Flatten(name='flatten')(x)
x = Dense(4096, activation='relu', name='fc1')(x)
x = Dense(4096, activation='relu', name='fc2')(x)
x = Dense(classes, activation='softmax', name='predictions')(x)
```

Configuration files

Configuration file: ~/.keras/keras.json

```
{
"image_data_format": "channels_last",
"epsilon": 1e-07,
"floatx": "float32",
"backend": "tensorflow"
}
```

Configuration files

When using *Theano* backend: ~/.theanorc

```
[blas]
ldflags =

[global]
floatX = float32
device = gpu

[nvcc]
fastmath = True

[gcc]
cxxflags = -ID:\MinGW\include

[cuda]
root=/usr/local/cuda-8.0/
```

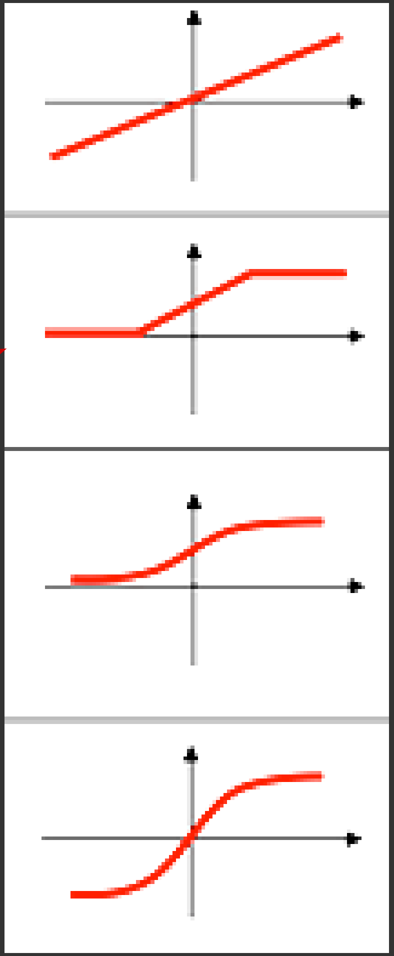
Documentation at: deeplearning.net/software/theano/library/config.html#

Basic layer types

- Core Layers – shown in example
- Convolutional Layers – shown in example
- Recurrent Layers – shown in example
- Pooling Layers – shown in example
- Locally-connected Layers
- Embedding Layers (including pretrained)
- Merge Layers – shown in example
- Advanced Activations Layers
- Normalization Layers
- Noise layers
- Layer wrappers
- Writing your own Keras layers

Activation functions

- linear (regression)
- softmax (classification)
- elu
- selu
- softplus
- softsign
- relu
- hard_sigmoidd
- sigmoid (logistic function / binary)
- tanh



and other advanced activations as well...

Loss functions

- mean_squared_error
- mean_absolute_error
- mean_absolute_percentage_error
- mean_squared_logarithmic_error
- squared_hinge
- hinge
- categorical_hinge
- logcosh
- categorical_crossentropy
- sparse_categorical_crossentropy
- binary_crossentropy
- kullback_leibler_divergence
- poisson
- cosine_proximity

Metrics

- binary_accuracy
- categorical_accuracy
- sparse_categorical_accuracy
- top_k_categorical_accuracy
- sparse_top_k_categorical_accuracy
- custom metrics (user defined)

Optimizers

- SGD
- RMSprop
- Adagrad
- Arguments
- References
- Adadelta optimizer.
- Arguments
- References
- Adam
- Adamax
- Nadam
- TFOptimizer - TensorFlow’s

Basic layer types

- Core Layers – shown in example
- Convolutional Layers – shown in example
- Recurrent Layers – shown in example
- Pooling Layers – shown in example
- Locally-connected Layers
- Embedding Layers (including pre-trained)
- Merge Layers – shown in example
- Advanced Activations Layers
- Normalization Layers
- Noise layers
- Layer wrappers
- Writing your own Keras layers

Customizing

Lambda layer – untrainable

(e.g. custom activation function)

Writing your own layer – trainable

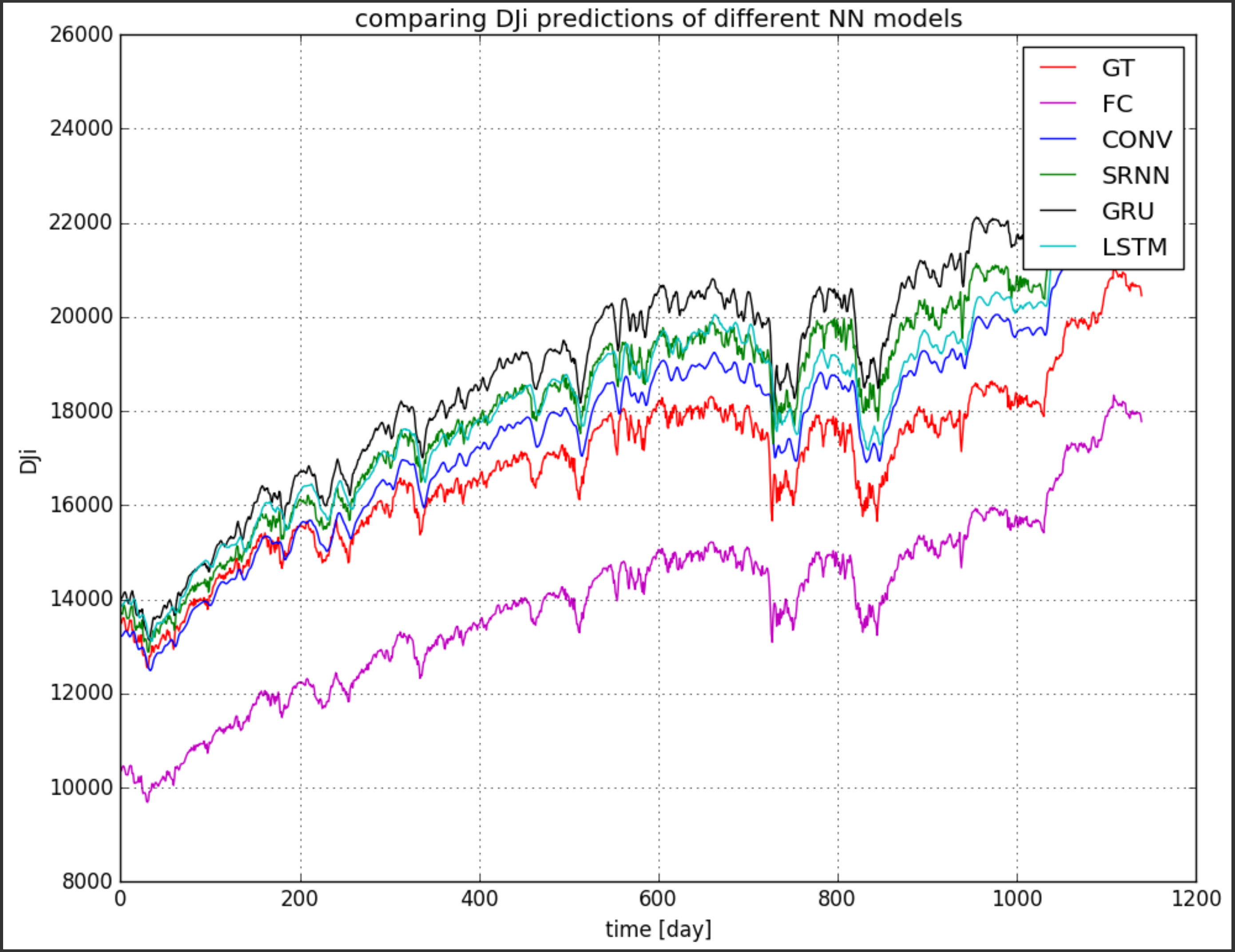
Writing your own metrics

Using your own generator as data flow

for pre-processing on-the-fly (during training)

Scikit-learn API for embedding a mixed model

Showcase – stocks Daily values vs. DJi



Natanel Davidovits - DS, VatBox