autokeras

April 18, 2020

AutoKeras implementation

This notebook uses Autokeras, an automated machine learning library built upon Tensorflow and Keras. The goal of AutoKeras is to make machine learning accessible for everyone.

This example takes a real-life dataset dogs vs cat and tries to train a well working image classifier without a user who decides which layers to use.

Please make sure that your driver configuration is correct (using Nvidia CUDA and cuDNN) and that the library makes its calculations on a GPU. Note that this code is written for AutoKeras v1.0.2 and may be updated for future versions.

Thank you for reading and enjoy exploring AutoKeras!

Robbe Decorte

Global imports and declarations

[ ]:
```
# Use this when running in the cloud or if the packages aren't already installed

!pip install autokeras
!pip install keras
!pip install tensorflow==2.1.0
```

[1]:
```
import pandas as pd
import numpy as np
import autokeras as ak

MAX_TRIES=5
SIZE=64
OUTPUT_NAME="autokeras-model-5"
```

Verify GPU status

[70]:
```
# Use the Keras GPU package
# conda install -c anaconda keras-gpu
from tensorflow.python.client import device_lib
```

```
[71]: # Show detailed information about the usable devices for Tensoflow
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 7502211016056982511
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 3142752665
locality {
  bus_id: 1
  links {
  }
}
incarnation: 12990106037986299612
physical_device_desc: "device: 0, name: GeForce GTX 1050, pci bus id:
0000:01:00.0, compute capability: 6.1"
]
```

```
[72]: # Only request device names
def get_available_devices():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos]
print(get_available_devices())
```

```
['/device:CPU:0', '/device:GPU:0']
```

Shuffle data and convert to numpy array

```
[18]: import os
from tqdm import tqdm
from PIL import Image
from random import shuffle

INPUT_PATH="data/train/"
```

```
[20]: def get_label(file):
    class_label = file.split('.')[0]
    if class_label == 'dog': label_vector = 0
    elif class_label == 'cat': label_vector = 1
    return label_vector
```

```
[21]: def get_data():
    data = []
```

```
    files = os.listdir(INPUT_PATH)
    for image in tqdm(files):

        label_vector = get_label(image)

        img = Image.open(INPUT_PATH + image).convert('L')
        img = img.resize((SIZE,SIZE))

        data.append([np.asarray(img),np.array(label_vector)])

    shuffle(data)
    return data
```

[22]:
```
data = get_data()
```

```
100%|
| 25000/25000 [01:54<00:00, 217.91it/s]
```

[23]:
```
# Save the data as a numpy array
np.save('data/images_shuffled.npy', data)
```

Data preprocessing and split to test and train

[5]:
```
data = np.load('data/images_shuffled.npy', allow_pickle=True)

# Split the labled data in train (20000 images) and test (5000 images)
train = data[:20000]
test = data[20000:]

print("Training dataset contains %d items" % len(train))
print("Testing dataset contains %d items" % len(test))
```

```
Training dataset contains 20000 items
Testing dataset contains 5000 items
```

[6]:
```
x_train = np.array([data[0] for data in train], 'float32')
x_test = np.array([data[0] for data in test], 'float32')
y_train = [data[1] for data in train]
y_test = [data[1] for data in test]
```

[7]:
```
x_train = np.array(x_train).reshape(-1,SIZE,SIZE,1)
x_test = np.array(x_test).reshape(-1,SIZE,SIZE,1)

x_train /= 255
x_test /= 255

y_train = np.array(y_train)
```

```
y_test = np.array(y_test)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(20000, 64, 64, 1)
(20000,)
(5000, 64, 64, 1)
(5000,)
```

Training and model evaluation

[ ]:
```
# Init image classifier.
clf = ak.ImageClassifier(max_trials=MAX_TRIES, name=OUTPUT_NAME)
# Feed the image classifier with training data.
clf.fit(x_train,y_train, verbose=2)
```

[ ]:
```
# This is a temporary workaround for a bug in the eager search algorithms
# https://github.com/keras-team/autokeras/issues/1045

clf = ak.AutoModel(
    inputs=[ak.ImageInput()],
    outputs=[ak.ClassificationHead()],
    tuner="random",
    max_trials=MAX_TRIES,
    name=OUTPUT_NAME,
    overwrite=False
)

clf.fit(x_train, y_train, verbose=2)
```

[53]:
```
score = clf.evaluate(x_test, y_test)
```

```
157/157 [==============================] - ETA: 13s - loss: 0.6545 - accuracy:
0.718 - ETA: 5s - loss: 0.6889 - accuracy: 0.796 - ETA: 4s - loss: 0.5824 -
accuracy: 0.81 - ETA: 3s - loss: 0.5444 - accuracy: 0.82 - ETA: 3s - loss:
0.6013 - accuracy: 0.82 - ETA: 3s - loss: 0.6256 - accuracy: 0.82 - ETA: 3s -
loss: 0.6035 - accuracy: 0.82 - ETA: 2s - loss: 0.6514 - accuracy: 0.82 - ETA:
2s - loss: 0.6221 - accuracy: 0.83 - ETA: 2s - loss: 0.6244 - accuracy: 0.83 -
ETA: 2s - loss: 0.6180 - accuracy: 0.82 - ETA: 2s - loss: 0.6278 - accuracy:
0.83 - ETA: 2s - loss: 0.6278 - accuracy: 0.83 - ETA: 2s - loss: 0.6330 -
accuracy: 0.83 - ETA: 2s - loss: 0.6378 - accuracy: 0.83 - ETA: 2s - loss:
0.6155 - accuracy: 0.83 - ETA: 1s - loss: 0.6196 - accuracy: 0.83 - ETA: 1s -
loss: 0.6450 - accuracy: 0.83 - ETA: 1s - loss: 0.6463 - accuracy: 0.83 - ETA:
1s - loss: 0.6567 - accuracy: 0.82 - ETA: 1s - loss: 0.6539 - accuracy: 0.82 -
```

```
ETA: 1s - loss: 0.6547 - accuracy: 0.82 - ETA: 1s - loss: 0.6527 - accuracy:
0.82 - ETA: 1s - loss: 0.6561 - accuracy: 0.82 - ETA: 1s - loss: 0.6449 -
accuracy: 0.82 - ETA: 1s - loss: 0.6329 - accuracy: 0.83 - ETA: 1s - loss:
0.6329 - accuracy: 0.83 - ETA: 1s - loss: 0.6454 - accuracy: 0.82 - ETA: 1s -
loss: 0.6401 - accuracy: 0.83 - ETA: 0s - loss: 0.6320 - accuracy: 0.83 - ETA:
0s - loss: 0.6352 - accuracy: 0.83 - ETA: 0s - loss: 0.6421 - accuracy: 0.82 -
ETA: 0s - loss: 0.6450 - accuracy: 0.82 - ETA: 0s - loss: 0.6423 - accuracy:
0.82 - ETA: 0s - loss: 0.6376 - accuracy: 0.82 - ETA: 0s - loss: 0.6380 -
accuracy: 0.82 - ETA: 0s - loss: 0.6375 - accuracy: 0.82 - ETA: 0s - loss:
0.6410 - accuracy: 0.82 - ETA: 0s - loss: 0.6458 - accuracy: 0.82 - ETA: 0s -
loss: 0.6505 - accuracy: 0.82 - ETA: 0s - loss: 0.6439 - accuracy: 0.82 - ETA:
0s - loss: 0.6378 - accuracy: 0.82 - ETA: 0s - loss: 0.6356 - accuracy: 0.82 -
ETA: 0s - loss: 0.6418 - accuracy: 0.82 - ETA: 0s - loss: 0.6398 - accuracy:
0.82 - 3s 17ms/step - loss: 0.6409 - accuracy: 0.8274
```

[64]:
```
print('Accuracy: {accuracy}\nLoss: {loss}'.format(accuracy=score[1],␣
 ↪loss=score[0]))
```

```
Accuracy: 0.8274000287055969
Loss: 0.6383044676065445
```

[65]:
```
model = clf.export_model()
model.save(OUTPUT_NAME + '/model.h5')
print('Model succesfully exported')
print(type(model))
```

```
Model succesfully exported
<class 'tensorflow.python.keras.engine.training.Model'>
```

Model visualization

[2]:
```
import matplotlib.pyplot as plt
import seaborn as sns;sns.set()
from tensorflow.python.keras.models import load_model
from tensorflow.keras.layers.experimental.preprocessing import Normalization
from sklearn.metrics import confusion_matrix
from tensorflow.keras.utils import plot_model
from sklearn.metrics import accuracy_score
```

[3]:
```
# Load model
cust = ak.CUSTOM_OBJECTS
cust['Normalization'] = Normalization

model = load_model(OUTPUT_NAME+'/model.h5', custom_objects=cust)

print("Model succesfully loaded")
```

```
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your
```

```
model is starting with a freshly initialized optimizer.
Model succesfully loaded
```

[8]:
```python
score = model.evaluate(x_test, y_test)
```

```
5000/5000 [==============================] - ETA: 10:23 - loss: 0.6545 -
accuracy: 0.718 - ETA: 1:41 - loss: 0.6116 - accuracy: 0.822 - ETA: 54s - loss:
0.6069 - accuracy: 0.8153 - ETA: 36s - loss: 0.6256 - accuracy: 0.820 - ETA: 27s
- loss: 0.6313 - accuracy: 0.825 - ETA: 21s - loss: 0.6260 - accuracy: 0.828 -
ETA: 18s - loss: 0.6213 - accuracy: 0.829 - ETA: 16s - loss: 0.6188 - accuracy:
0.830 - ETA: 14s - loss: 0.6388 - accuracy: 0.828 - ETA: 13s - loss: 0.6346 -
accuracy: 0.831 - ETA: 12s - loss: 0.6330 - accuracy: 0.833 - ETA: 11s - loss:
0.6372 - accuracy: 0.834 - ETA: 10s - loss: 0.6289 - accuracy: 0.836 - ETA: 9s -
loss: 0.6172 - accuracy: 0.838 - ETA: 8s - loss: 0.6184 - accuracy: 0.83 - ETA:
8s - loss: 0.6518 - accuracy: 0.83 - ETA: 7s - loss: 0.6552 - accuracy: 0.82 -
ETA: 7s - loss: 0.6558 - accuracy: 0.82 - ETA: 6s - loss: 0.6532 - accuracy:
0.82 - ETA: 6s - loss: 0.6547 - accuracy: 0.82 - ETA: 5s - loss: 0.6525 -
accuracy: 0.82 - ETA: 5s - loss: 0.6561 - accuracy: 0.82 - ETA: 4s - loss:
0.6431 - accuracy: 0.83 - ETA: 3s - loss: 0.6344 - accuracy: 0.83 - ETA: 3s -
loss: 0.6454 - accuracy: 0.82 - ETA: 3s - loss: 0.6401 - accuracy: 0.83 - ETA:
2s - loss: 0.6320 - accuracy: 0.83 - ETA: 2s - loss: 0.6365 - accuracy: 0.83 -
ETA: 2s - loss: 0.6492 - accuracy: 0.82 - ETA: 2s - loss: 0.6423 - accuracy:
0.82 - ETA: 1s - loss: 0.6383 - accuracy: 0.82 - ETA: 1s - loss: 0.6353 -
accuracy: 0.82 - ETA: 1s - loss: 0.6375 - accuracy: 0.82 - ETA: 1s - loss:
0.6410 - accuracy: 0.82 - ETA: 1s - loss: 0.6490 - accuracy: 0.82 - ETA: 0s -
loss: 0.6459 - accuracy: 0.82 - ETA: 0s - loss: 0.6395 - accuracy: 0.82 - ETA:
0s - loss: 0.6341 - accuracy: 0.82 - ETA: 0s - loss: 0.6418 - accuracy: 0.82 -
ETA: 0s - loss: 0.6389 - accuracy: 0.82 - ETA: 0s - loss: 0.6394 - accuracy:
0.82 - 6s 1ms/sample - loss: 0.6383 - accuracy: 0.8274
```

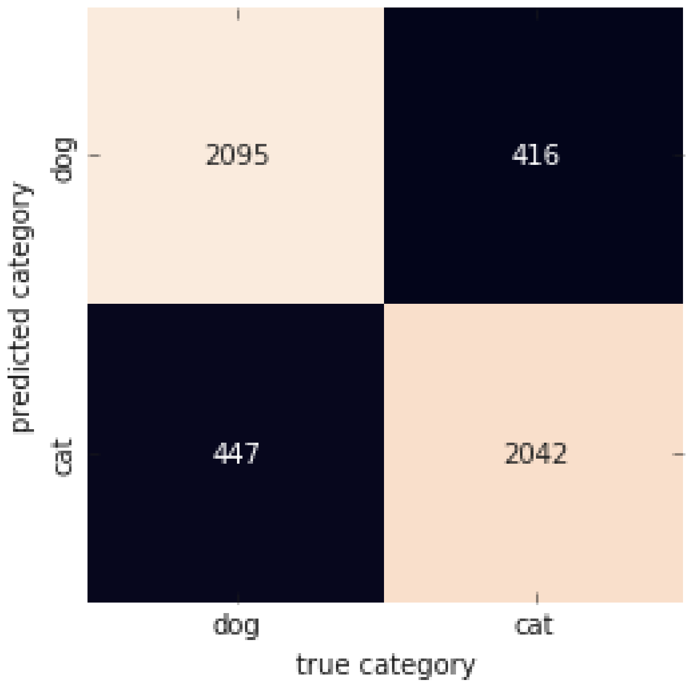
[9]:
```python
print('Accuracy: {accuracy}\nLoss: {loss}'.format(accuracy=score[1],␣
↪loss=score[0]))
```

```
Accuracy: 0.8274000287055969
Loss: 0.6383046453475952
```

[12]:
```python
predictions = model.predict(x_test)
```

[10]:
```python
plt.style.use('classic')
%matplotlib inline

mat = confusion_matrix(y_test, predictions.round())
labels = ['dog', 'cat']

sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,␣
↪xticklabels=labels, yticklabels=labels)
plt.xlabel('true category')
plt.ylabel('predicted category')
```

[10]: Text(90.80000000000001, 0.5, 'predicted category')



[13]:
```
instances = 0
correct_ones = 0

images = x_test.reshape(5000,64, 64)
incorrect_predictions = []

for i in range(0, len(predictions)):
    if i < 5:
        print("Prediction: ", predictions[i], ", Actual: ", y_test[i])
    if predictions[i].round() == y_test[i]:
        correct_ones += 1
    else:
        incorrect_predictions.append((i, images[i], predictions[i].round(4),
 ↪y_test[i]))
    instances += 1

print('\nMade {amountPred} predictions. {amountCorrect} of those are correct.
 ↪Approximately {percentage}%.'
      .format(amountPred=instances, amountCorrect=correct_ones,
 ↪percentage=round((correct_ones/instances) * 100)))
```

```
Prediction:  [0.5145127] , Actual:  0
Prediction:  [0.56603503] , Actual:  0
```

```
Prediction:  [0.99702567] , Actual:  0
Prediction:  [4.35466e-07] , Actual:  0
Prediction:  [0.98181605] , Actual:  1

Made 5000 predictions. 4137 of those are correct. Approximately 83%.
```

[14]:

```
# Show images that are classified to the wrong class
# Dog = 0, Cat = 1
%matplotlib inline

figure, axes = plt.subplots(nrows=6, ncols=4, figsize=(16,16))

for axes, item in zip(axes.ravel(), incorrect_predictions):
    index, image, predicted, expected = item
    axes.imshow(image, cmap=plt.cm.gray)
    axes.set_xticks([])
    axes.set_yticks([])
    axes.set_title(f'index: {index}\np: {predicted}; e: {expected}')
plt.tight_layout()
```

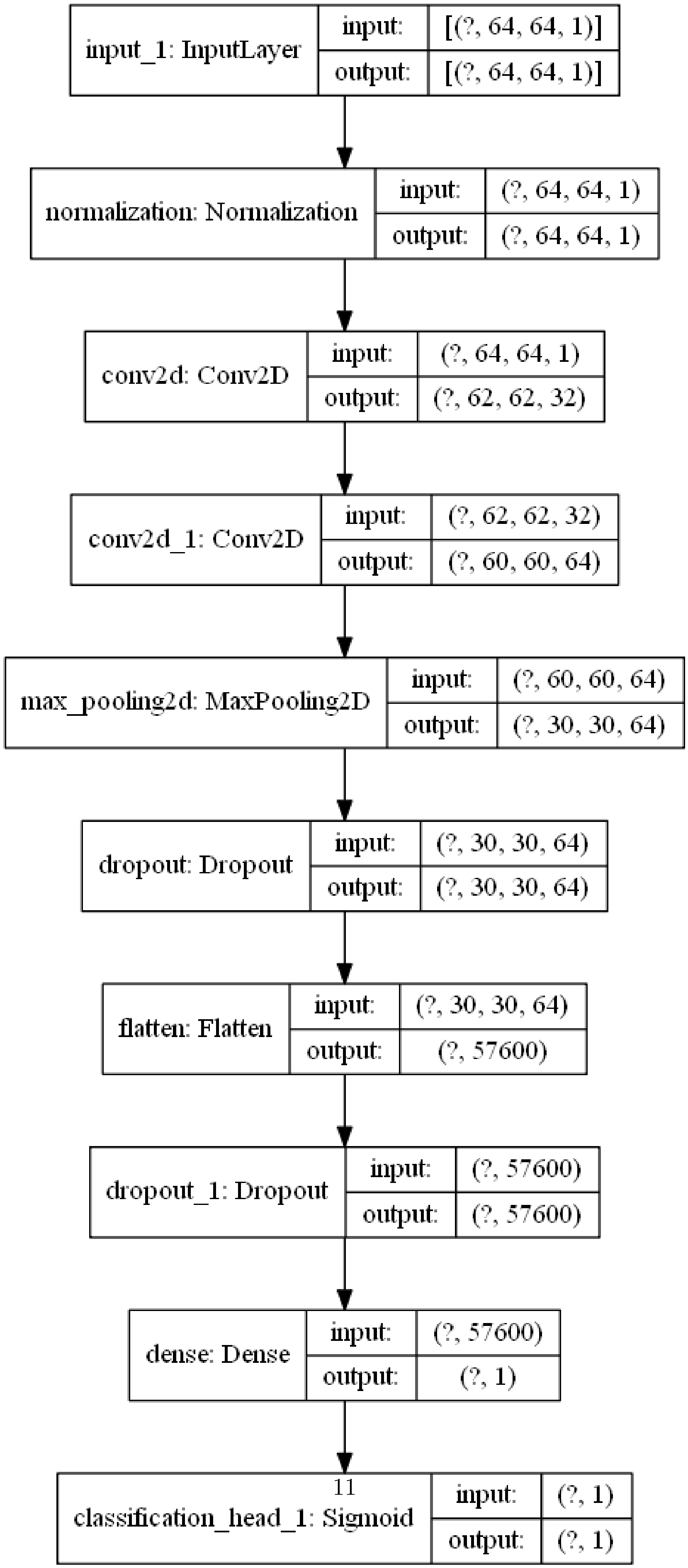
[67]:
```
model.summary()
```

```
Model: "model"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 64, 1)]       0
_________________________________________________________________
```

```
normalization (Normalization (None, 64, 64, 1)         3

_________________________________________________________________
conv2d (Conv2D)              (None, 62, 62, 32)        320

_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 60, 64)        18496

_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 30, 30, 64)        0

_________________________________________________________________
dropout (Dropout)            (None, 30, 30, 64)        0

_________________________________________________________________
flatten (Flatten)            (None, 57600)             0

_________________________________________________________________
dropout_1 (Dropout)          (None, 57600)             0

_________________________________________________________________
dense (Dense)                (None, 1)                 57601

_________________________________________________________________
classification_head_1 (Sigmo (None, 1)                 0
=================================================================
Total params: 76,420
Trainable params: 76,417
Non-trainable params: 3

_________________________________________________________________
```

[68]:
```python
plot_model(model, to_file=OUTPUT_NAME + '/model.png', show_shapes=True,␣
↪show_layer_names=True)
```

[68]:


input_1: InputLayer
input: [(?, 64, 64, 1)]
output: [(?, 64, 64, 1)]
normalization: Normalization
input: (?, 64, 64, 1)
output: (?, 64, 64, 1)
conv2d: Conv2D
input: (?, 64, 64, 1)
output: (?, 62, 62, 32)
conv2d_1: Conv2D
input: (?, 62, 62, 32)
output: (?, 60, 60, 64)
max_pooling2d: MaxPooling2D
input: (?, 60, 60, 64)
output: (?, 30, 30, 64)
dropout: Dropout
input: (?, 30, 30, 64)
output: (?, 30, 30, 64)
flatten: Flatten
input: (?, 30, 30, 64)
output: (?, 57600)
dropout_1: Dropout
input: (?, 57600)
output: (?, 57600)
dense: Dense
input: (?, 57600)
output: (?, 1)
classification_head_1: Sigmoid
input: (?, 1)
output: (?, 1)