Student: Yuliia Zozulia

One Page Summary

Topic: Classification of Musical Instruments by Sound

Problem Statement: The goal is to develop the model which will allow us to recognize music instrument from the sound of it. I will use convolutional neural network, implementing it using Keras. As dataset, I will use The NSynth Dataset, which contains sounds of solitary musical notes from various instrument families.

Dataset Description: NSynth is an audio dataset containing 305,979 musical notes, each with a unique pitch, timbre, and envelope. For 1,006 instruments from commercial sample libraries, authors generated four second, monophonic 16kHz audio snippets, referred to as notes with indicating the instrument family the record came from.

- Large size (zipped): 23.5 Gb, 305,979 record.
- Subset that was used: 1.13 Gb, 9,460 record for training; 371 Mb, 3,039 files for testing.

https://magenta.tensorflow.org/datasets/nsynth

Overview of Technology: Keras – high-level neural networks API, written in Python and capable of running on top of TensorFlow – was used to implement the model to solve the problem. It has very user-friendly user interface and offers consistent & simple APIs. Librosa package was used to extract features from raw sound file.

Overview of Steps:

1. Defined problem statement
2. Install and configure environment
3. Find suitable dataset and obtain data
4. Clean and preprocess data
5. Implement CNN model, find suitable configuration
6. Evaluate model performance
7. Output predictions

Hardware: PC with Windows 10 Home (64 bit) running on AMD FX-8320E Eight-Core Processor 3.5 GHz and equipment with 8.00 Gb RAM. No CUDA-supported GPU.

Software: Anaconda, Jupiter Notebook, Python, Keras, TensorFlow, Keras, Librosa,

Lessons Learned: I used Convolutional Neural Network to create a model for classification music instruments by sound. Keras provides very user-friendly user interface and consistent & simple APIs for NN development. It’s really pleasure to work with it.

Pros: Using NSynth, which contains audio record of the solitary note, as dataset, developed model, implemented thought Keras, trains very fast and produce very good results (98% test accuracy).

Cons: It would be nice to expand the model to learn how to recognize instrument not from the single note, but from actual piece of music with a single predominant instrument.

YouTube URLs:

2 minutes: https://youtu.be/koA5rjbmMZk

15 minutes: https://youtu.be/wFLID1VJqik

Problem Statement:

The goal is to develop the model which will allow us to recognize music instrument from the sound of it. I will use convolutional neural network, implementing it using Keras - high-level neural networks API, running on top of TensorFlow. As dataset, I will use The NSynth Dataset, which contains sounds of solitary musical notes from various instrument families.

Why this topic?

I have musical background (piano), and I often listen to the classical music, where a lot of different instruments are performing. It's always good to know which of them is playing now, and often found myself trying to guess while listening, so idea of making computer to recognize them for me inspired me. Besides this, I felt curiosity about the complexity of this problem – are sounds produced by different instrument that difficult to recognize, or this is a simple problem?

During the term we worked a lot on image recognizing, so I was curious about working with sound.

Technologies used:

I used Keras – high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

https://keras.io/

I run it on top of TensorFlow™ is an open source software library for high performance numerical computation.

As environment management system, Anaconda Navigator was used:

https://anaconda.org/anaconda/anaconda-navigator

Programing language – Python:

https://www.python.org/

Development environment – Jupyter:

http://jupyter.org/

A python package for music and audio analysis – librosa.

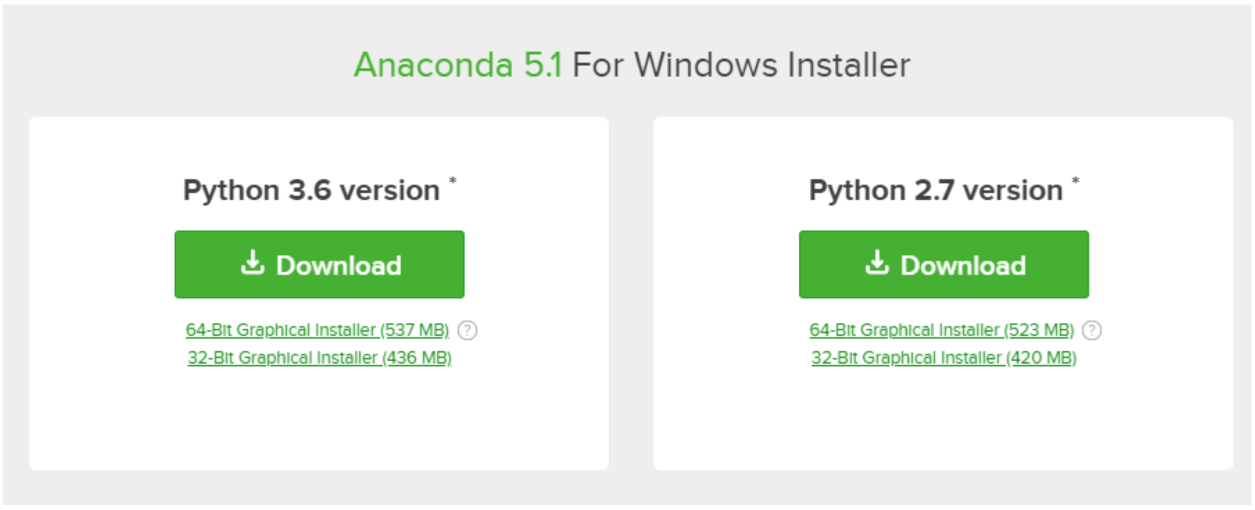
https://github.com/librosa/librosa

Setup instructions:

To set up Anaconda, go to:

https://www.anaconda.com/download

Pick OS (Windows 64-Bit in my case) and Python version (Python 3.6), click "Download" and follow steps of Graphical Installer.



Anaconda 5.1 For Windows Installer

Python 3.6 version *

Download

64-Bit Graphical Installer (537 MB)
32-Bit Graphical Installer (436 MB)

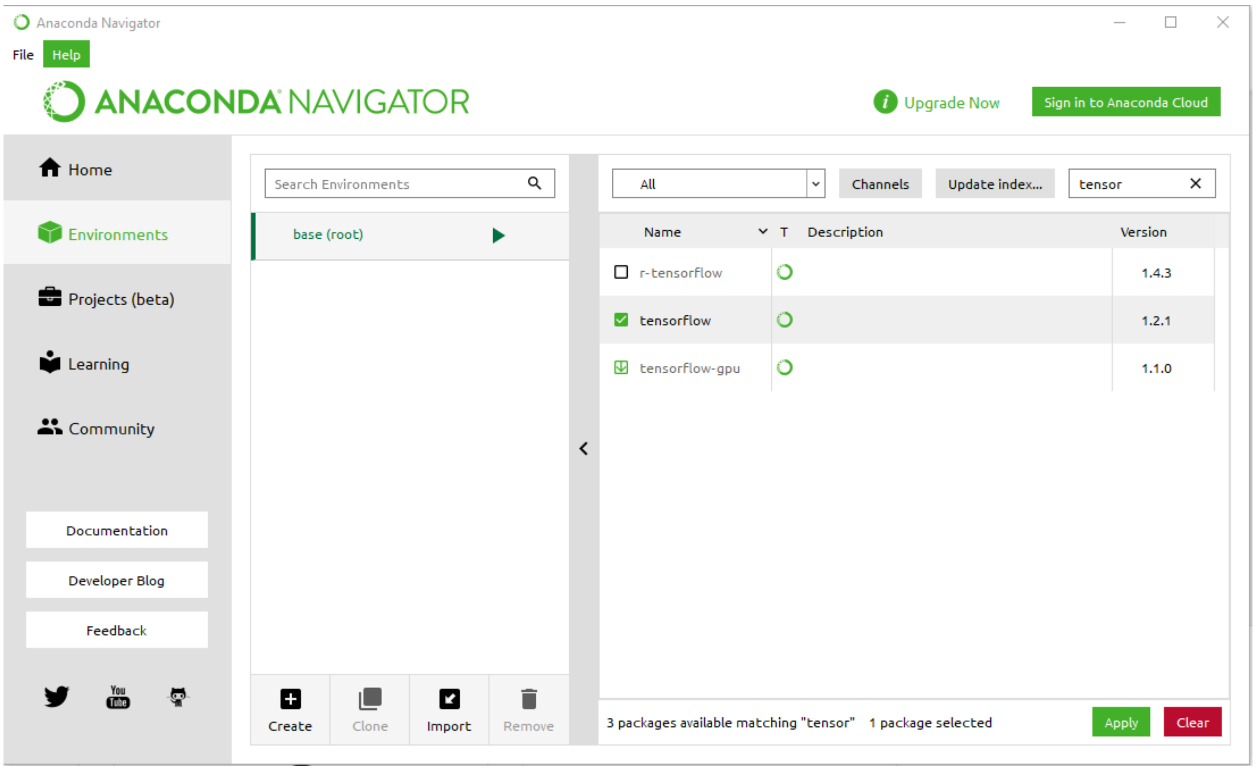
Python 2.7 version *

Download

64-Bit Graphical Installer (523 MB)
32-Bit Graphical Installer (420 MB)

Once done, you have installed Anaconda, Python and Jupiter Notebook.

Open Anaconda Navigator, go to Environment, pick appropriate Environment (base(root)) in my case, then in "Search Packages" type "Tensotflow". If not installed already, select checkbox near "Tensorflow" and press "Apply".



	Name	T	Description	Version
☐	r-tensorflow			1.4.3
✓	tensorflow			1.2.1
☐	tensorflow-gpu			1.1.0

3 packages available matching "tensor" 1 package selected

Do this whenever you need to install new package.

Packages used in the project:

TensorFlow, Numpy, Keras, Matplotlib, Librosa, shutil, IPython.

Make sure you have all of them installed to proceed. If not, follow steps described for tensorflow no install other packages as well.

The only package which I didn't find from Anaconda Navigator interface was librosa, which can be installed from the conda-forge channel:

```
conda install -c conda-forge librosa
```

To execute program, lunch Jupiter Notebook, and open file with source code.

Hardware description:

I used my PC, which has Windows 10 Home (64 bit) running on AMD FX-8320E Eight-Core Processor 3.5 GHz and equipment with 8.00 Gb RAM. It also has AMD Radeon R7 200 GPU, which is useless (for this project), because it isn't supported by CUDA.

Preprocessing the data take about 20 minutes, and my model itself trains very fast (about 56 second per epoch), so hardware without GPU doesn't create a problem here.

Dataset Description:

To perform classification of Musical Instruments by sound, we need to find large dataset with records of sound with information about performer.

I used The NSynth Dataset from Magenta

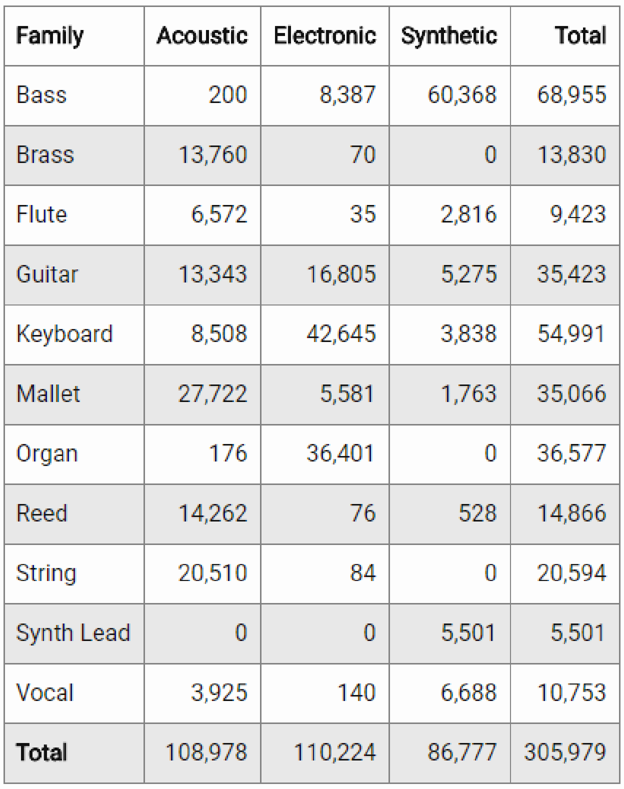
https://magenta.tensorflow.org/datasets/nsynth

Authors intended to create high-quality and large-scale sounds dataset to much those in the image domain (such as MNIST, CIFAR and ImageNet).

NSynth is an audio dataset containing 305,979 musical notes, each with a unique pitch, timbre, and envelope. For 1,006 instruments from commercial sample libraries, authors generated four second, monophonic 16kHz audio snippets, referred to as notes, by ranging over every pitch of a standard MIDI piano (21-108) as well as five different velocities (25, 50, 75, 100, 127). The note was held for the first three seconds and allowed to decay for the final second. This dataset is widely used for music syntheses.

Each sample contains large description with a lot of information, like note, pitch, velocity and so on, but, as I intended to train my model on sound file only, I didn't take this information into the account.

Dataset contain information about instrument family. Frequency counts for instrument classes with Sources as columns a Families as rows:



Family	**Acoustic**	**Electronic**	**Synthetic**	**Total**
Bass	200	8,387	60,368	68,955
Brass	13,760	70	0	13,830
Flute	6,572	35	2,816	9,423
Guitar	13,343	16,805	5,275	35,423
Keyboard	8,508	42,645	3,838	54,991
Mallet	27,722	5,581	1,763	35,066
Organ	176	36,401	0	36,577
Reed	14,262	76	528	14,866
String	20,510	84	0	20,594
Synth Lead	0	0	5,501	5,501
Vocal	3,925	140	6,688	10,753
Total	108,978	110,224	86,777	305,979

Data reading and preprocessing:

The full dataset is split into three sets:

- Train [tfrecord | json/wav]: A training set with 289,205 examples. Instruments do not overlap with valid or test.
- Valid [tfrecord | json/wav]: A validation set with 12,678 examples. Instruments do not overlap with train.
- Test [tfrecord | json/wav]: A test set with 4,096 examples. Instruments do not overlap with train.

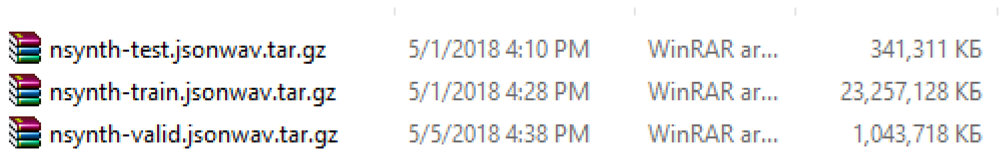
We are interested only in wav files. Here are the links for downloading:

http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-train.jsonwav.tar.gz

http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-valid.jsonwav.tar.gz

http://download.magenta.tensorflow.org/datasets/nsynth/nsynth-test.jsonwav.tar.gz

Total size of 3 datasets is 23.5 Gb.



nsynth-test.jsonwav.tar.gz	5/1/2018 4:10 PM	WinRAR ar...	341,311 КБ
nsynth-train.jsonwav.tar.gz	5/1/2018 4:28 PM	WinRAR ar...	23,257,128 КБ
nsynth-valid.jsonwav.tar.gz	5/5/2018 4:38 PM	WinRAR ar...	1,043,718 КБ

Trains set is 21.1 Gb.

I used "nsynth-valid" folder as train set and "nsynth-test" as validation. "nsynth-train" is too huge, while I got model with high accuracy on much smallel dataset ("nsynth-valid" folder).

Unzip test and validation folders to folder which contains your Jupitar notebook.

It's important not to mix folders, because instruments (from which records were taken) has not to overlap in train and validation set.

```
import os

train_path = "./nsynth-valid/audio/"
val_path = "./nsynth-test/audio/"
```

Function for cleaning the data. As all file names looks like

“flute_acoustic_002-106-050.wav”

We can split name using “_” and use first element as instrument family and second as type (acoustic, electronic, synthetic). I would like to proceed only with acoustic and electronic, so let’s delete synthetic. Also, dataset contains records of vocal, with is not instrument, so let’s delete them too.

```
def clean_data(path):
    for filename in os.listdir(path):
        sp = filename.split('_')
        if((sp[1]=="synthetic") or (sp[0]=="vocal") or (sp[0]=="synth")):
            os.remove(path+filename)
```

As mentioned above, we can use first part of file name as instrument family indemnificatory, so there is the function to get all possible labels:

```
def get_labels(path):
    labels = set([])
    for filename in os.listdir(path):
        sp = filename.split('_')
        labels.add(sp[0])
    labels = list(labels)
    return (labels)
```

Here and below I'll use orange color as background if I've taken this code from somewhere (with reference to the source).

This function to convert wav file to mfcc is taken (with some modifications) from Manash's blog:

https://blog.manash.me/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b

(also used in Lecture 12):

```
def wav2mfcc(file_path, max_len=11):
    wave, sr = librosa.load(file_path, mono=True, sr=None)
    wave = wave[::3]
    mfcc = librosa.feature.mfcc(wave, sr=16000)
    mfcc = mfcc[:, :max_len]
    return mfcc
```

Function to read data from given folder and convert it to pair X, Y where X is MFCC transformation of file and Y is label of instrument family the sound was recorded from.

```
def get_data(path, max_len):
    labels = get_labels(train_path)
    X=[]
    Y=[]
    files_list = os.listdir(path)
    shuffle(files_list)
    for filename in files_list:
        sp = filename.split('_')
        x=wav2mfcc(path+filename, max_len)
        if(sp[0] not in labels):
            print(sp[0] + "Error!")
        y = labels.index(sp[0])

        X.append(x)
        Y.append(y)
    X=np.array(X)
    Y=np.array(Y)
    return X,Y
```

Actual cleaning:

```
clean_data(train_path)
clean_data(val_path)
```

After cleaning, we got 1.13 Gb, 9,460 records for training; 371 Mb, 3,039 files for testing.

Actual file reading and saving processed arrays to .npy files (for future reference):

```
t = 20 #We'll trim data to this length

X_train, Y_train = get_data(train_path, t)
np.save("X_train" + '.npy', X_train)
np.save("Y_train" + '.npy', Y_train)

X_val, Y_val = get_data(val_path, t)
np.save("X_val" + '.npy', X_val)
np.save("Y_val" + '.npy', Y_val)
```

Reshaping to feed to Conv2D:

```
from keras.utils import to_categorical
mfcc_output_size = 20 #constant
channel = 1
epochs = 10
batch_size = 20

X_train = np.load("X_train" + '.npy')
Y_train = np.load("Y_train" + '.npy')

X_val = np.load("X_val" + '.npy')
Y_val = np.load("Y_val" + '.npy')

X_train = X_train.reshape(X_train.shape[0], mfcc_output_size, t, channel)
Y_train = to_categorical(Y_train)

X_val = X_val.reshape(X_val.shape[0], mfcc_output_size, t, channel)
Y_val = to_categorical(Y_val)
```

Let’s check what we’ve get:

```
labels = get_labels(train_path)

print(labels)
print (X_train.shape)
print (X_val.shape)
```

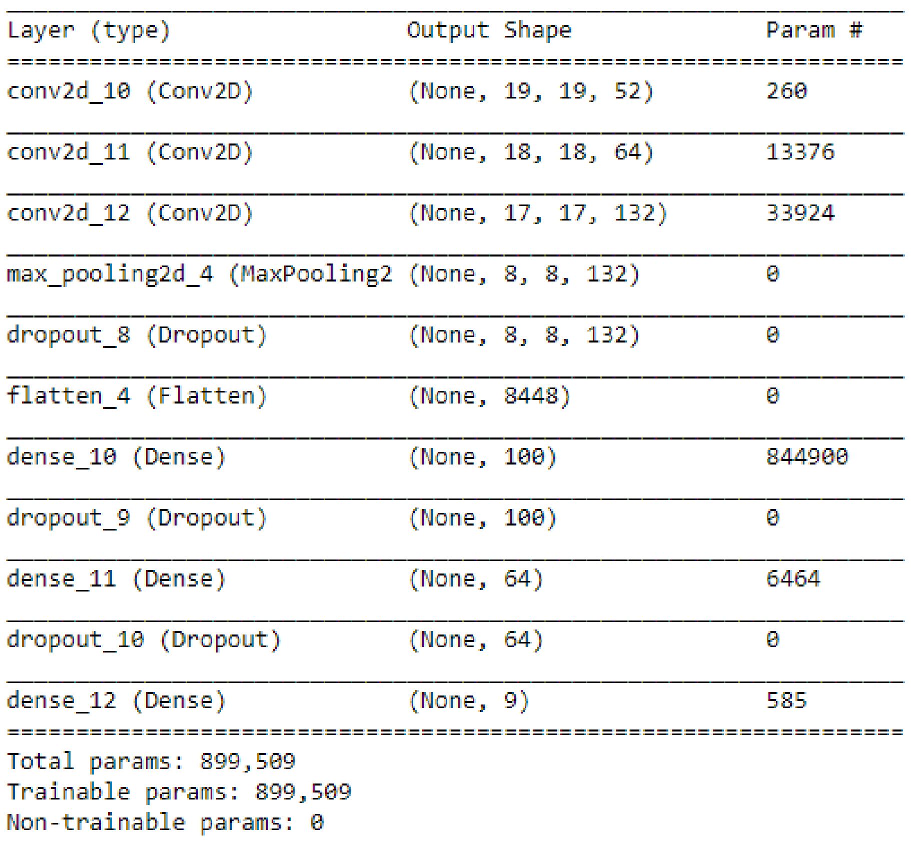
```
['keyboard', 'flute', 'mallet', 'guitar', 'bass', 'reed', 'string', 'brass',
'organ']
(9459, 20, 20, 1)
(3039, 20, 20, 1)
```

So we have 9 classes of instruments; 9459 samples in training set (20x20 each) and 3039 in validation set.

Building a model:

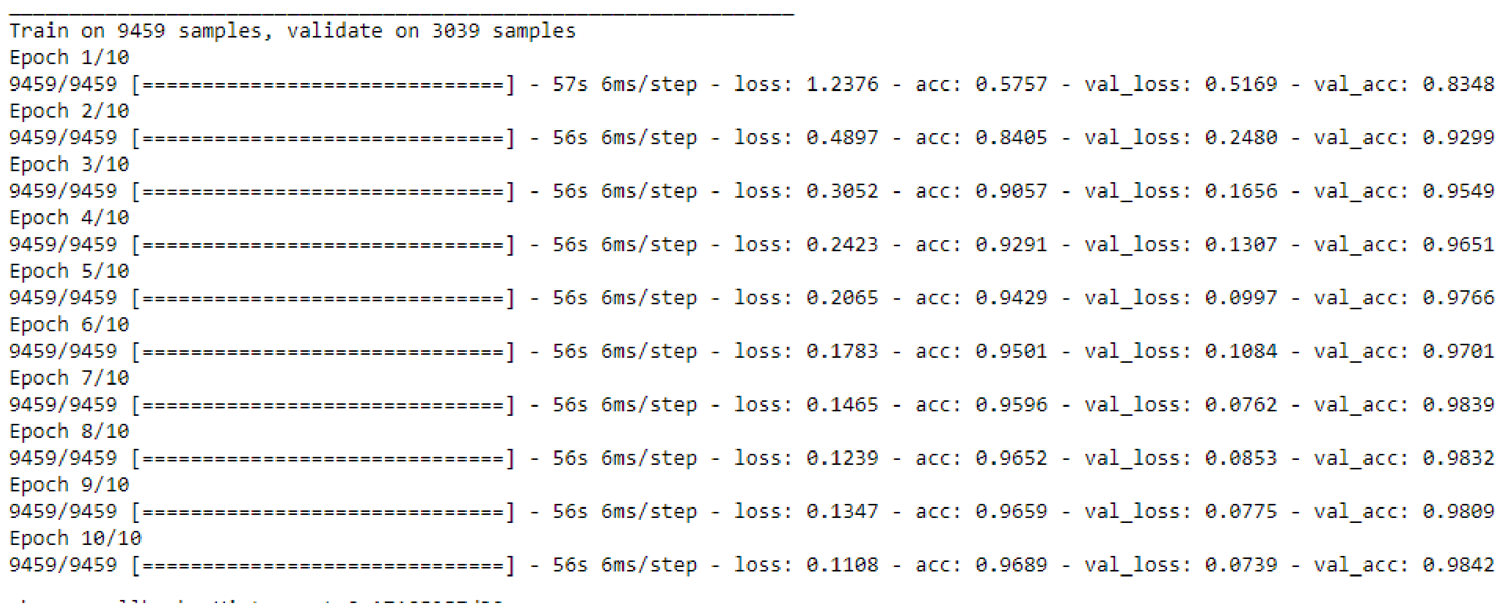
Defining the model (similar to the one from Homework12):

```
import keras
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

def build_model():
    model = Sequential()
    model.add(Conv2D(52,     kernel_size=(2,     2),     activation='relu',
input_shape=(mfcc_output_size, t, channel)))
    model.add(Conv2D(64, kernel_size=(2, 2), activation='relu'))
    model.add(Conv2D(132, kernel_size=(2, 2), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(64,          kernel_regularizer=regularizers.l2(0.001),
activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(len(labels), activation='softmax'))
    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adam(),
                  metrics=['accuracy'])
    return model
model = build_model()
model.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_10 (Conv2D)           (None, 19, 19, 52)        260       
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 18, 18, 64)        13376     
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 17, 17, 132)       33924     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 132)         0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 8, 8, 132)         0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 8448)              0         
_________________________________________________________________
dense_10 (Dense)             (None, 100)               844900    
_________________________________________________________________
dropout_9 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 64)                6464      
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_12 (Dense)             (None, 9)                 585       
=================================================================
Total params: 899,509
Trainable params: 899,509
Non-trainable params: 0
```

```
model.fit(X_train,    Y_train,    batch_size=batch_size,    epochs=epochs,
verbose=1, validation_data=(X_val, Y_val))
```



```
_________________________________________________________________
Train on 9459 samples, validate on 3039 samples
Epoch 1/10
9459/9459 [==============================] - 57s 6ms/step - loss: 1.2376 - acc: 0.5757 - val_loss: 0.5169 - val_acc: 0.8348
Epoch 2/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.4897 - acc: 0.8405 - val_loss: 0.2480 - val_acc: 0.9299
Epoch 3/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.3052 - acc: 0.9057 - val_loss: 0.1656 - val_acc: 0.9549
Epoch 4/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.2423 - acc: 0.9291 - val_loss: 0.1307 - val_acc: 0.9651
Epoch 5/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.2065 - acc: 0.9429 - val_loss: 0.0997 - val_acc: 0.9766
Epoch 6/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.1783 - acc: 0.9501 - val_loss: 0.1084 - val_acc: 0.9701
Epoch 7/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.1465 - acc: 0.9596 - val_loss: 0.0762 - val_acc: 0.9839
Epoch 8/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.1239 - acc: 0.9652 - val_loss: 0.0853 - val_acc: 0.9832
Epoch 9/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.1347 - acc: 0.9659 - val_loss: 0.0775 - val_acc: 0.9809
Epoch 10/10
9459/9459 [==============================] - 56s 6ms/step - loss: 0.1108 - acc: 0.9689 - val_loss: 0.0739 - val_acc: 0.9842
```

Validation error after 10 epochs is **98.4%.** Each epoch took about 1 minute, so the whole training process was completed in 10 minutes. Much faster than reading the files. There is no need to continue training, as validation error starts to fluctuate after 7th epoch.

Function for predicting from wav file (from Homework12):

```
def predict(filepath, model):
    sample = wav2mfcc(filepath, t)
    sample_reshaped = sample.reshape(1, mfcc_output_size, t, channel)
    return labels[np.argmax(model.predict(sample_reshaped))]
```

Function for prediction for all files in the folder, which returns list of targeted and predicted instrument labels:

```
def predict_set(path, model):
    target=[]
    predicted=[]

    for filename in os.listdir(path):
        sp = filename.split('_')
        target.append(sp[0])
        predicted.append(predict(path+filename, model))

    return target, predicted
```

Predicting labels for all files in Validation set:

```
target, predicted = predict_set(val_path, model)
```

Let’s make a confusion matrix:

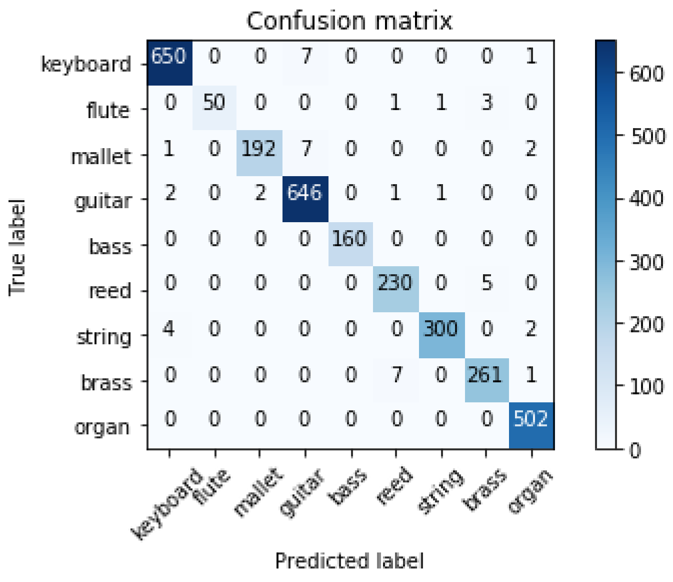
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(target, predicted, labels=labels)
```

Function for matrix visualization. Taken from here:

http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

```
import itertools
import numpy as np
import matplotlib.pyplot as plt

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


plot_confusion_matrix(confusion_matrix, classes=labels,
                      title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[650   0   0   7   0   0   0   0   1]
 [  0  50   0   0   0   1   1   3   0]
 [  1   0 192   7   0   0   0   0   2]
 [  2   0   2 646   0   1   1   0   0]
 [  0   0   0   0 160   0   0   0   0]
 [  0   0   0   0   0 230   0   5   0]
 [  4   0   0   0   0   0 300   0   2]
 [  0   0   0   0   0   7   0 261   1]
 [  0   0   0   0   0   0   0   0 502]]
```



Confusion matrix looks very good! Among miss-predicted values is keyboard predicted as guitar (7 times) and reed predicted as brass (5 times).

Here are some examples of predicting files from validation set:

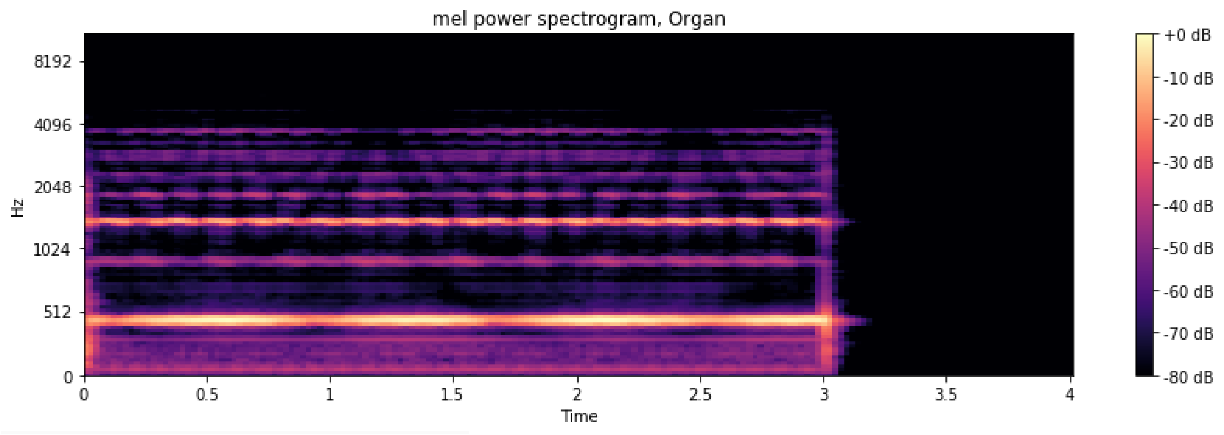
```
filename = "organ_electronic_001-048-127.wav"
print("Targeted : " + filename.split('_')[0])
print("Predicted: " + predict(test_path+'/' + filename, model=model))
ipd.Audio(test_path+'/' + filename)
```

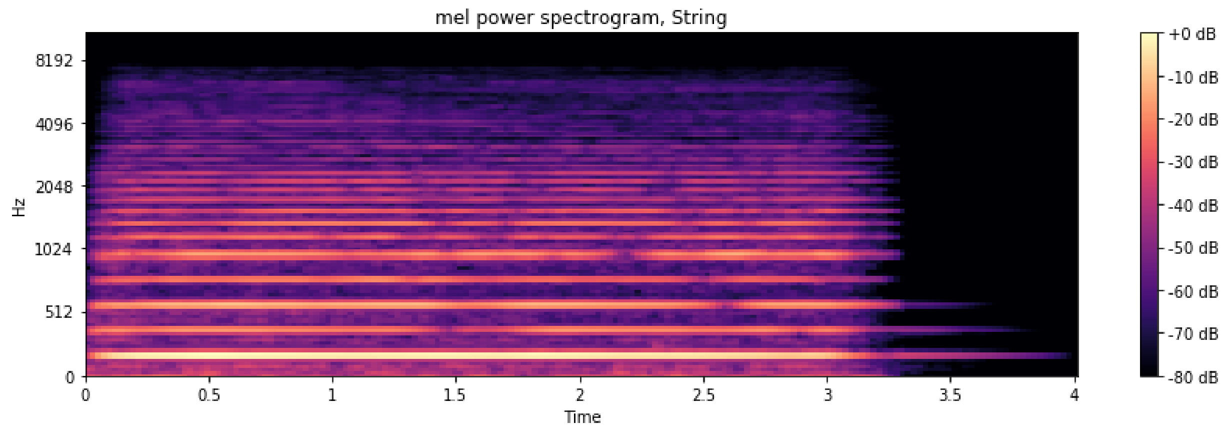
```
Targeted : organ
Predicted: organ
```

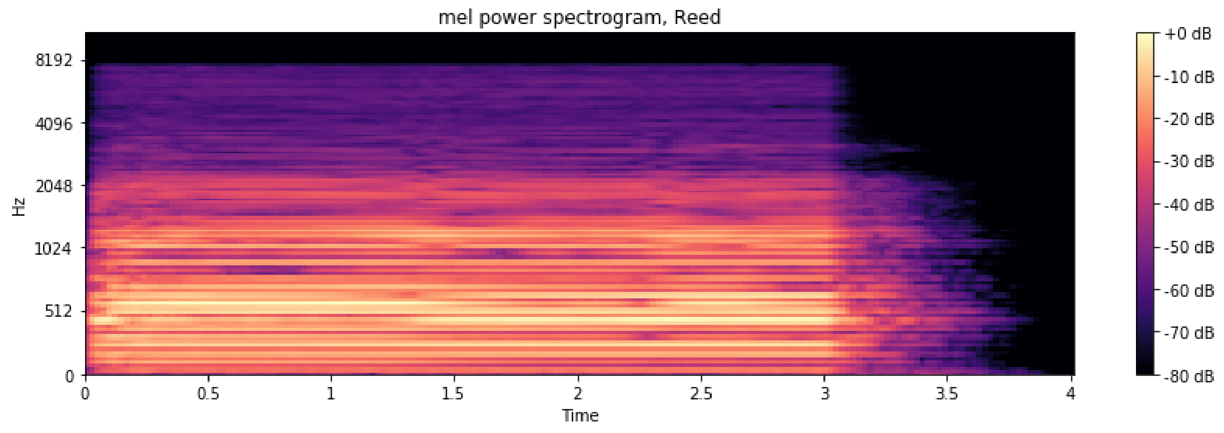
```
filename = "string_acoustic_080-048-025.wav"
print("Targeted : " + filename.split('_')[0])
print("Predicted: " + predict(test_path+'/' + filename, model=model))
ipd.Audio(test_path+'/' + filename)
Targeted : string
Predicted: string
```

Let’s take a look at mel power spectrogram for several different instruments (code taken from librosa demo):
http://nbviewer.jupyter.org/github/librosa/librosa/blob/master/examples/LibROSA%20demo.ipynb

```
audio_path = train_path + '/organ_electronic_104-070-127.wav'
y, sr = librosa.load(audio_path)
S = librosa.feature.melspectrogram(y, sr=sr, n_mels=128)
# Convert to log scale (dB). We'll use the peak power (max) as reference.
log_S = librosa.power_to_db(S, ref=np.max)
plt.figure(figsize=(12,4))
librosa.display.specshow(log_S, sr=sr, x_axis='time', y_axis='mel')
plt.title('mel power spectrogram, Organ')
plt.colorbar(format='%+02.0f dB')
plt.tight_layout()
ipd.Audio(audio_path)
```







Conclusions

I used Convolutional Neural Network to create a model for classification music instruments by sound. As datasource, I choose very clean dataset NSynth from Magenta. It contains the record of a solitary note per file. The model, implemented thought Keras, trains very fast (~10 epochs, each takes around 1 minute, was enough) and produces very good results (98% test accuracy).

Keras provides very user-friendly user interface and consistent & simple APIs for NN development. It's really pleasure to work with it.

I would like to expand the model to learn how to recognize instrument not from the single note, but from actual piece of music with a single predominant instrument. I found suitable dataset – IRMAS (https://www.upf.edu/web/mtg/irmas). But model, similar to the one I successfully used for NSynth Dataset, when trained on IRMAS dataset (with modification to accommodate more complex data), gives me only 47.5% testing accuracy (number of classes – 9). Although training accuracy was ~97%, no matter how I tried to apply regularization of such model.

Also, it would be interesting to use other sound features, which can be extracted from sound file using Librosa library, like spectral centroid, roll-off frequency, tonal centroid features, zero-crossing rate, etc. It didn't make sense to with NSynth (as we already have ~98% test accuracy) but it might help with more complex IRMAS dataset. I feel like to include these features, we shouldn't just put them as additional dimension to convolutional network. I tried to build network which takes several inputs – to feed first input (mfcc) to convolutional part of the network, then flatten it and feed to second input (vector of other abovementioned feature) to dense layer, but my network failed to converge. I'm looking forward to investigate matter further.

So, as I discovered, problem of recognizing musical instrument from the sound of it can be successfully and easily solved if we deal with the record of the single note, but requires more sophisticated approaches if we want to classify instruments from an actual piece of music.

References:

Code and technical info:

https://keras.io/

https://anaconda.org/anaconda/anaconda-navigator

https://www.python.org/

http://jupyter.org/

https://github.com/librosa/librosa

https://blog.manash.me/building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19c12b

http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

http://nbviewer.jupyter.org/github/librosa/librosa/blob/master/examples/LibROSA%20demo.ipynb

Data source:

https://magenta.tensorflow.org/datasets/nsynth