Speech Recognition with Tensorflow

Course Project Report

Submitted in partial fulfilment of the requirements for the

degree of

Master of Technology in

Computer Science and Engineering

Under the guidance of

Dr. Annappa B

172CS007 Tanmay Badhe
172CS009 Debojyoti M
172CS015 Kemanth PJ

Contents

Speech Recognition with Tensorflow

1. Abstract

TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. This project uses Tensorflow to create a Speech Recognition application with Convolutional Neural Networks.

2. Introduction

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, often replacing its closed-source predecessor, DistBelief. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015. TensorFlow is Google Brain's second generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. Tensorflow was originally created for tasks that require heavy numerical computations and was geared towards the problem of machine learning, and deep neural networks.Due to a C C++ backend, TensorFlow is able to run faster than pure Python code. It provides both a Python and a C++ API. But the Python API is more complete and it's generally easier to use. TensorFlow application uses a structure known as a data flow graph and its structure is based on the execution of this graph. A data flow graph has two basic units. 1. A node represents a mathematical operation 2. An edge represents a multi-dimensional array, known as a tensor. So this high-level abstraction reveals how the data flows between operations. The standard usage is to build a graph and then execute after the session is created, by using the 'run' and 'eval' operations. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays.

2.1 Advantages Of Tensorflow:

1. **Flexibility**: we need to express our computation as a data flow graph to use TensorFlow. It is a highly flexible system which provides multiple models or multiple versions of the same model can be served simultaneously. The architecture of TensorFlow is highly modular, which means we can use some parts individually or can use all the parts together. Such flexibility facilitates non-automatic migration to new models/versions, A/B testing experimental models, and canarying new models.
2. **Portability**: TensorFlow has made it possible to play around an idea on our laptop without having any other hardware support. It runs on GPUs, CPUs, desktops, servers, and mobile computing platforms. we can deploy a trained model on our mobile as a part of our product, and that's how it serves as a true portability feature.
3. **Research and Production**: It can be used to train and serve models in live mode to real customers. To put it simply, rewriting codes is not required and the industrial researchers can apply their ideas to products faster. Also, academic researchers can share codes directly with greater reproducibility. In this way it helps to carry out research and production processes faster.
4. **Auto Differentiation**: It has automatic differentiation capabilities which benefits gradient based machine learning algorithms. We can define the computational architecture of your predictive model, combine it with our objective function and add data to it- TensorFlow manages derivatives computing processes automatically. We can compute the derivatives of some values with respect to some other values results in graph extension and we can see exactly what's happening.
5. **Performance**: TensorFlow allows us to make the most of your available hardware with its advanced support for threads, asynchronous computation, and queues. Just assign compute elements of your TensorFlow graph to different devices and let it manage the copies itself. It also facilitates us with the language options to execute our computational graph. TensorFlow iPython notebook helps in keeping codes, notes, and visualization in a logically grouped and interactive style.

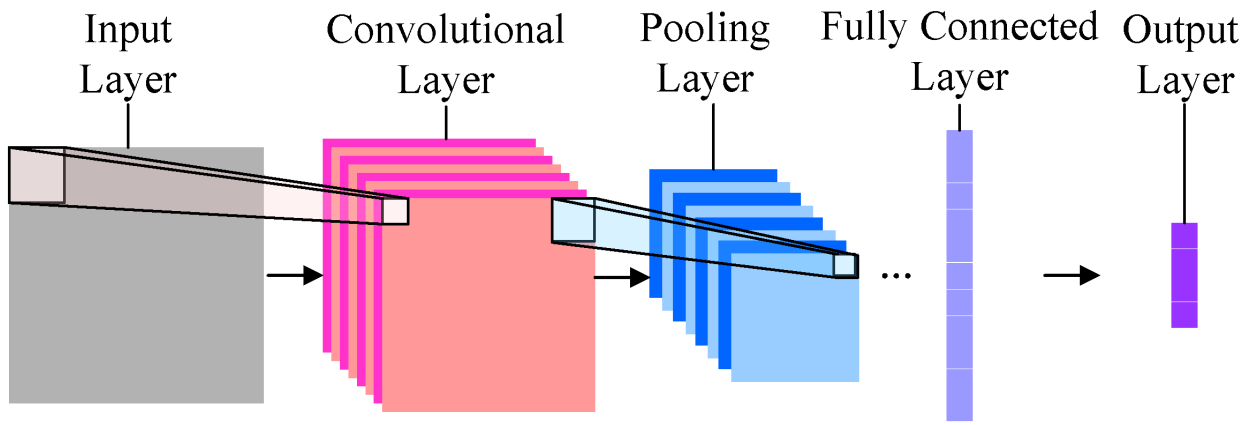
3. Speech Recognition with Tensorflow

Speech recognition is the process of extracting text transcriptions or some form of meaning from speech input. Speech analytics can be considered as the part of the voice processing, which converts human speech into digital forms suitable for storage or transmission computers. This application is capable of detecting the word spoken in the audio file, it uses convolutional neural networks to classify different audio sounds. The convolutional neural networks itself is implemented using tensorflow.

3.1 Convoltional Neural Networks

The Convolutional neural network(CNN) is a deep learning architecture that has numerous application in computer vision and natural language processing. The CNN classifies objects based on number of features matched. Steps involed in creating CNN

1. Convolution
2. Pooling
3. Flattening
4. Full Connection



- **Convolution**
  ConvNets derive their name from the “convolution” operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.
- **Pooling**
  Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.
- **Flattening**
  Convert the 2D matrix to a column vector so that it can passed through an artificial neural network
- **Full Connection**
  The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

4. Implementation of Speech Recognizer in Tensorflow

First we need to import the libararies required by the application

In [8]:

```
#import required libraries

##for numerical computations
import numpy as np

##for converting to mfcc
import librosa

##for file handling
import os

##for one shot encoding
from keras.utils import to_categorical

##split dataset into train and test set
from sklearn.model_selection import train_test_split

##tensorflow
import tensorflow as tf
```

```
Using TensorFlow backend.
```

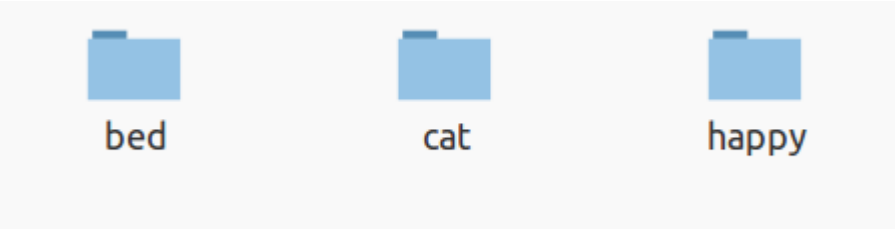
4.1. Dataset

The dataset is located in folder named data, we set DATA_PATH variable to that folder.

In [7]:

```
DATA_PATH = "./data/"
```

The data consists of three classes, viz. bed, cat and happy.



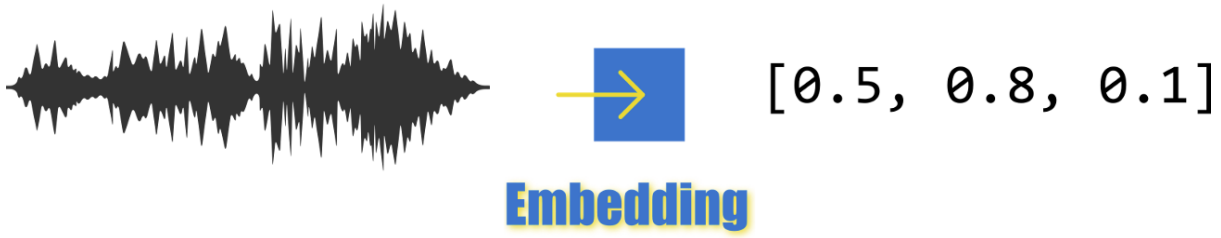
Each folder contains approximately 1700 audio files. The name of the folder is actually the label of those audio files. The task will be to classify an audio between **bed, cat and happy** - these three

classes.

4.2. Preprocessing Audio Files

We need to prepare a fixed size vector for each audio file and feed the vector into the Convolutional Net.An **embedding** is a mapping from discrete objects, such as words, to vectors of real numbers.

Audio Embedding



In sound processing, the **mel-frequency cepstrum** (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Steps

1. Read the audio file from DATA_PATH.
2. Perform downsample operation.
3. Compute MFCC using librosa library.
4. MFCC vectors might vary in size for different audio input,To overcome this problem we need to pad the output vectors with zero.

The function **wav2mfcc** performs the aforementioned steps.

In [31]:

```
def wav2mfcc(file_path, max_pad_len=11):
    wave, sr = librosa.load(file_path, mono=True, sr=None)
    wave = wave[::3]
    mfcc = librosa.feature.mfcc(wave, sr=16000)
    pad_width = max_pad_len - mfcc.shape[1]
    mfcc = np.pad(mfcc, pad_width=((0, 0),
                    (0, pad_width)), mode='constant')
    return mfcc
```

The function **get_labels** gets the name of different categories of data present in the dataset folder and encodes them

In [9]:

```
def get_labels(path=DATA_PATH):
    labels = os.listdir(path)
    label_indices = np.arange(0, len(labels))
    return labels, label_indices, to_categorical(label_indices)
```

In order to avoid the computation of mfcc again and again we store the calculated data in a numpy array using the **save_data_to_array** function and can be resued for further computations

In [10]:

```
def save_data_to_array(path=DATA_PATH, max_pad_len=11):
    labels, _, _ = get_labels(path)

    for label in labels:
        # Init mfcc vectors
        mfcc_vectors = []

        wavfiles = [path + label + '/' + wavfile
                    for wavfile in os.listdir(path + '/' + label)]
        for wavfile in wavfiles:
            mfcc = wav2mfcc(wavfile, max_pad_len=max_pad_len)
            mfcc_vectors.append(mfcc)
        np.save(label + '.npy', mfcc_vectors)
```

The function **get_train_test** splits the entire dataset into training set and test set. The training set contains 60 percent of the data and test set contains 40 percent of the data.

In [11]:

```
def get_train_test(split_ratio=0.6, random_state=42):
    # Get available labels
    labels, indices, _ = get_labels(DATA_PATH)

    # Getting first arrays
    X = np.load(labels[0] + '.npy')
    y = np.zeros(X.shape[0])

    # Append all of the dataset into one single array, same goes for y
    for i, label in enumerate(labels[1:]):
        x = np.load(label + '.npy')
        X = np.vstack((X, x))
        y = np.append(y, np.full(x.shape[0], fill_value= (i + 1)))

    assert X.shape[0] == len(y)

    return train_test_split(X, y,
                            test_size= (1 - split_ratio),
                            random_state=random_state, shuffle=True)
```

Now we use get_train_test to split the dataset, We also use one hot encoding to convert categorical data to appropriate formats which is suitable for computation by the CNN

In [19]:

```
X_train, X_test, y_train, y_test = get_train_test()
X_train = X_train.reshape(X_train.shape[0], 20, 11, 1)
X_test = X_test.reshape(X_test.shape[0], 20, 11, 1)
y_train_hot = to_categorical(y_train)
y_test_hot = to_categorical(y_test)
```

4.3 Building the Convolutional Neural Network

The function **init_weights** initializes the weights of the network with some random values using tensorflow's random distribution function.

Initializing the weights of the CNN

In [12]:

```
def init_weights(shape):
    init_random_dist = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(init_random_dist)
```

Initializing the bias of the CNN

This function **init_bias** initializes the bias with some constant value

In [13]:

```
def init_bias(shape):
    init_bias_vals = tf.constant(0.1, shape=shape)
    return tf.Variable(init_bias_vals)
```

Creating the Convolutional layer

The **conv2d** is a function that takes feature matrix x, and weight matrix W as input and returns a 2d convolutional layer

In [14]:

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],
                        padding='SAME')
```

The 2x2 pooling layer is created by **max_pool_2by2**

In [17]:

```
def max_pool_2by2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

convolutional_layer method uses conv2d to perform convolution and passes it through a relu activation function

In [16]:

```
def convolutional_layer(input_x, shape):
    W = init_weights(shape)
    b = init_bias([shape[3]])
    return tf.nn.relu(conv2d(input_x, W) + b)
```

normal_full_layer creates a fully connected layer

In [18]:

```
def normal_full_layer(input_layer, size):
    input_size = int(input_layer.get_shape()[1])
    W = init_weights([input_size, size])
    b = init_bias([size])
    return tf.matmul(input_layer, W) + b
```

The actual CNN is then created with the help of above helper functions.

In [21]:

```
## using a 2x2 feature detector for convolution
## There is a single input channel
## And there will be 32 output features,
## hence shape = [2, 2, 1, 32]
convo_1 = convolutional_layer(x,shape=[2,2,1,32])

## flattening the features extracted into an 1D array
## The size of the array will be 20x11x32
convo_2_flat = tf.reshape(convo_1,[-1,20*11*32])

## creating a fully connected laer with 1024 neurons
## It takes the 1D array conv_2_flat as input
full_layer_one = tf.nn.relu(normal_full_layer(convo_2_flat,1024))

## during training phase we need to randomly turn off
## some neurons to find new paths in the network and
## increase accuracy
full_one_dropout = tf.nn.dropout(full_layer_one,keep_prob=hold_prob)

## creating the final output layer with 3 neurons
## to classify the audio into 3 categories
y_pred = normal_full_layer(full_one_dropout,3)
```

Defining the cost function of the network. In this case cross entropy is being used. The objective of the CNN is to optimize the cost function. We use Adam optimizer for our application. The learning rate α is 0.001

In [23]:

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true,
                                               logits=y_pred))
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
train = optimizer.minimize(cross_entropy)
```

4.3 Training the CNN

Tensorflow uses placeholders as special variables to feed data into Deep Neural Networks. We define three placeholders as described below-

1. x: input
2. y_true: correct labels
3. hold_prob: probability for dropout layer

In [20]:

```
x = tf.placeholder(tf.float32,shape=[None,20,11,1])
y_true = tf.placeholder(tf.float32,shape=[None,3])
hold_prob = tf.placeholder(tf.float32)
```

The graph nodes corresponding to the CNN are created by Tensorflow, and the variables are initialized

In [24]:

```
init = tf.global_variables_initializer()
```

saver saves the model parameters and weights

In [25]:

```
saver = tf.train.Saver()
```

Now we create a Tensorflow session. The graph is initialized and the session is run

In [27]:

```
with tf.Session() as sess:
    sess.run(init)
    for i in range(250):
        #print("Training Epoch : " + str(i))
        sess.run(train, feed_dict={x: X_train,
                                   y_true: y_train_hot, hold_prob: 0.5})

        # PRINT OUT A MESSAGE EVERY 100 STEPS
        if i%100 == 0:

            print('Currently on step {}'.format(i))
            print('Accuracy is:')
            # Test the Train Model
            matches = tf.equal(tf.argmax(y_pred,1),tf.argmax(y_true,1))

            acc = tf.reduce_mean(tf.cast(matches,tf.float32))
            print(sess.run(acc,feed_dict={x:X_test,
                                          y_true:y_test_hot,hold_prob:1.0})

    saver.save(sess,'models/cnn_model.ckpt')
```

```
Currently on step 0
Accuracy is:
0.3261079
Currently on step 100
Accuracy is:
0.9238921
Currently on step 200
Accuracy is:
0.9349711
```

4.4 Creating a GUI and predicting a new audio clip

Importing libraries for creating GUI

In [28]:

```
from tkinter import filedialog
from tkinter import Tk,Label,Button,Canvas
```

We now create a browse button for browsing and loading audio clips. After loading the clip, we calculate the mfcc of the audio. A Tensorflow session is created. The trained CNN is loaded and the saved parameters are restored. Prediction is made on the audio clip which is then output and the output is displayed on canvas.

In [29]:

```
def browse_button():
    # Allow user to select a directory and store it in global var
    # called folder_path
    global folder_path
    global path
    filename = filedialog.askopenfile()
    sample = wav2mfcc(filename.name)
    print(filename.name)
    path = "aplay " + filename.name[50:]
    sample_reshaped = sample.reshape(1, 20, 11, 1)
    labels = ["happy", "cat", "bed"]
    with tf.Session() as sess :
        saver.restore(sess,'models/cnn_model.ckpt')
        predict = tf.argmax(y_pred,1)
        pred = sess.run(predict,
                        feed_dict={x:sample_reshaped,
                                   y_true: y_train_hot, hold_prob: 1.0})
        ans = labels[pred[0]]
        canvas.create_text(350, 25,
                text = "The detected word is : " + ans, font=("Purisa", 25))
```

The main code which creates the gui with the help of browse button

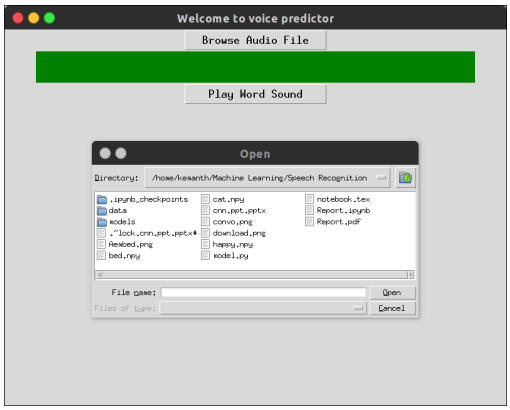
In [39]:

```
root = Tk()
root.title("Welcome to voice predictor")
root.geometry('800x600')
button2 = Button(text="Browse Audio File",
                 command=browse_button, height = 1,
                 width = 20,font=("Ariel", 15))
button2.pack()
canvas = Canvas(width=700, height=50, bg='green')
canvas.pack()
play = lambda : os.system(path)
button = Button(root, text = 'Play Word Sound',
                command = play, height = 1, width = 20,font=("Purisa", 15))
button.pack()
root.mainloop()
```

```
/home/kemanth/Machine Learning/Speech Recognition/data/happy/0
b09edd3_nohash_0.wav
INFO:tensorflow:Restoring parameters from models/cnn_model.ckp
t
```

4.4 A sample run

The input window



Predicting Output