Project ID for SYDE 522

Project Title:	Accelerometer Biometric
Project Member(s):	Mohamed Abo El Soud, Karim Kawambwa, WenChao Jiang, Omar Gouda
Summary of the Project:	Since everyone moves differently and accelerometers are fast becoming ubiquitous, this competition is designed to investigate the feasibility of using accelerometer data as a biometric for identifying users of mobile devices.
Data Used:	There have been uploaded approximately 60 million unique samples of accelerometer data collected from 387 different devices. These are split into equal sets for training and test. Samples in the training set are labeled with the unique device from which the data was collected. The test set is demarcated into 90k sequences of consecutive samples from one device. A file of test questions is provided in which you are asked to determine whether the accelerometer data came from the proposed device.
Source of Data Used:	https://www.kaggle.com/c/accelerometer-biometric-competition/data
Results Achieved:	This project is part of a competition. The competitioned has already ended. However we aim to work on trying to improve the results, or at least create an alternative solution.

Accelerometer Biometric

Bill Jiang, Mohammed Abo Al Soud, Omr Jouda, Karim Kawambwa
Department of Mechanical and Mechatronics Engineering
w52jiang@edu.uwaterloo.ca, maboelso@edu.uwaterloo.ca, ogouda@edu.uwaterloo.ca , kkawambw@edu.uwaterloo.ca

Abstract - *Mobile devices offer the opportunity to extract large amounts of information from daily user usage. Accelerometer data is one of the important pieces of data that can be extracted that determine the device's orientation at any time instant. The objective of this project is to determine whether the accelerometer data can be used to determine which mobile device the data came from. This would allow using accelerometers as a biometric for user identification. A large data set of accelerometer data is obtained from an online competition which is used to train and test a machine learning model. Several classification methods are used and their accuracies are compared to determine which method is most optimal to obtain accurate results. The final model is tested against an arbitrary list of accelerometer data and is used to predict which mobile device corresponds to certain accelerometer data sets.*

Keywords: accelerometer, mobile device, biometric, classification, model, training, user identification, testing, prediction, machine intelligence, gait, CNN, HMM

1 Introduction

There are various amounts of information that can be extracted from mobile devices. The widespread use of mobile devices allows opens up new possibilities to how this data can be gathered, handled and utilized to make informed decisions. One of the main pieces of information that can be derived is the accelerometer data that records information related to the device's orientation relative to a standard 3-axis coordinate system. This could serve various purposes such as a biometric security identification system that allows user authentication in a very friendly manner with minimal user interaction time. This could potentially be used to tackle the rising issue of mobile device users deactivating their security settings that involve significant interaction such as a PIN or password. The objective of this project is to develop a machine intelligence algorithm to investigate whether the accelerometer data can be used as a biometric for mobile user identification. Every mobile device user has different movement patterns, hence different accelerometer information. These patterns can be used to train a model to predict which user corresponds to certain accelerometer data. An arbitrary list is used to test the model and determine whether certain accelerometer data corresponds to a certain device.

1.1 Problem

The accelerometers used in mobile devices record information to determine the device's orientation relative to the coordinate system shown in the figure below. The x-axis always points to the right, y-axis to the top of the screen and the z-axis away from the screen.

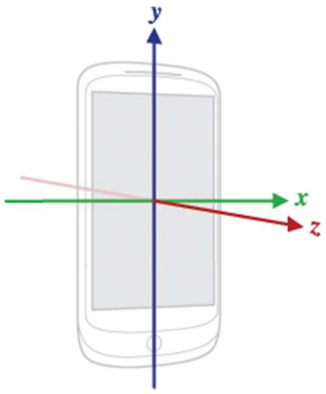

Figure 1. Coordinate system according to Android Sensor API [4]

The sensor's coordinate system never changes as the device moves and is independent of the screen's orientation. This is dependent on the device's natural orientation which could be either portrait or landscape. This allows for a fixed reference to be used as part of this classification process.

2 Data

The data used in this report is from the Kaggle sponsor known as Seal. The sponsor collected the x, y and z accelerometer data from several users within several months of normal device usage [3]. As indicated by the competition, this data was collected from user mobile devices that installed the Seal Android application that collected the accelerometer data in the background. An approximate 60 million unique samples from 387 different devices was collected. The data is split is split into equal sets of training and testing, and the presented in the following forms.

Table 1. Training Data Set Format

Field name	Description
T	Unix time
X	x co-ordinate acceleration
Y	y co-ordinate acceleration
Z	z co-ordinate acceleration
DeviceId	Unique Id of the device that generated the sample

Table 2. Test Data Set Format

Field name	Description
T	Unix time
X	x co-ordinate acceleration
Y	y co-ordinate acceleration
Z	z co-ordinate acceleration
SequenceId	Unique sequence number assigned to each question. Each group of samples is labeled with a unique SequenceId. Each SequenceId is matched to a professed DeviceId in the questions file.

Table 3. Questions Format

Field name	Description
QuestionId	Id of question
SequenceId	Unique number assigned to each sequence of samples
QuizDevice	Professed device that generated the sequence of accelerometer data in the test file

2.1 Sequential Data

Currently, many of the research into the field of machine learning has been dealing with image recognition. Where there exists a dataset of images and a classifier determines certain important features in the image. The applications of this research spans from text and handrwiting recognition, to driving systems for autonomous vehicles. Moreover, the machine learning algorithm seeks to define similarities and differences between the set of photos.

When dealing with images, the training dataset consists of the image, and a corresponding label. This is an example of what is known as supervised learning, where there is a clear input and output that allows the machine learning algorithm to make appropriate associations.

However, the dealing with accelerometer data represents a fundamental shift into the way in which the data is observed and analyzed. To further understand, it is important to have a clear grasp of the given dataset. The given datasets corresponds to coordinate in 3D space, which denote the acceleration of the individual as they move. Hypothetically, one could integrate the accelaration twice in order to get the current position of the individual in 3d space. Then, the test data contains acecelerometer data and it is checked in order to verify whether or not the test data belongs to the individual in question.

If a human being were to look at the training data and the test data, and is tasked with verification instead of the machine, they would likely attempt to perform the verification based on two attributes of the data: patterns and context. It is likely that the accelerometer data would exhibit certain patterns, specially if the individual is performing a continuous movement, such as walking. Furthermore, it is likely that each individual would have a pattern of movement based on the unique characteristics of the individual, such as height, weight, age and situatiosnal circumstance. Given a data sample, it is highly unlikely that the sample exists in isolation. Rather, the sample is part of a larger set of data, that are combined in order form a complete dataset. In other words, a sample cannot belong to a certain individual, if it cannot not possibly fit the current data given in the training set. For instance, if the sample data overlaps in time with the training data of an inidividual, then the sample does not belong to the same individual since the device cannot output two different sets of data at the same time. The sample must fit in-between gaps of the training data based on the time-stamp, with no possible overlap. If a sample starts immediately after a training dataset ends, then the values of the coordinates should match, as a jarring jump in the data would be incohesive. Overall, when judging accelerometer data is important to consider to consider the context in order to make valid judgements. This is a very different from the image recognition example mentioned earlier. The in the case of image recognition, each image is analyzed

separately in the machine learning agorithm, where each point in the accelerometer data is dependent on the point before it, and helps determine the probability of the location of the point after it. This influences the choice of machine learning methods used to deal with the data as well as the method in which the data might be prepared.

2.2 Exploits

There has been many exploits circulated on the internet that point towards a method in which one could efficiently determine the segment belonged to which device, from observing the timestamps and matching them accordingly.

From a posted question,

"What are some ways that you have tried to convert the data for this competition into standard train_X, train_Y and test_X format so that it can be fed to ML libraries to predict test_Y?"

It became apparent that people exploited this for the competition inferred from the response,

"Well, there seems to be leakage issues with this competition, and the host has acknowledged this, choosing to go on anyway, prizes and all. So I'm guessing there are quite a number of entries that try to exploit these issues, rather than employ ML approaches.

That being said, I tried to mimic the evaluation process. I divided the train set into 300 points each within every device label, transformed them into a single data point, then applied the same transform on the test set. I'm playing with various models and ensembles on these transformed sets."

By using the number of samples and timestamps alone, this was enough to predict the results with high accuracy. This was devoid of any machine learning and defeats the purpose of the study.

3 Methods

With the presented problem, machine learning is used to learn and determine the user of each device according to features extracted fom the data. The performce (accuracy) of any employed mchine learning method largely depends on good feture extraction which is a big problem to solve and requires good domain knowledge [1].

A total of three methods are used in order to try obtain the best accuracy where poor feature extraction may exist.

3.1 Convolution Neural Network (CNN)

This section discusses the use of Convolutional Neural Networks or CNN for human activity recognition using accelerometer data. This method can also be applied to identify different type of devices. CNN employs the use of features to determine similarities in patterns between two pieces of data [1]. Datasets are analyzed piece by piece and certain features common to a reference data set are attempted to be located. When CNN is presented with a new data set, there is no initial information as to where these certain features can be located. Hence, CNN attempts to match and locate feature in every possible location in the data set. The math used to perform this process is called convolution, hence the name CNN. The convolution process is repeated for each of the other key features until the complete data set is processed. Another powerful tool used by CNN is called pooling where large data sets are shrunk down into approximately quarter its original size while preserving the most relevant information required [1]. This is done using a small stepping window that goes across the data set recording the maximum value from the window at each step interval. CNN can now focus on determining if features exist in a dataset without being concerned where the feature is exactly located. This makes the process much easier to process where the same number of data sets is used, but each set is smaller in size than the original. The general architecture that is used for the project is shown in the Figure 2 below. It consists of having one convolution layer, followed by a layer of pooling and another convolution layer.

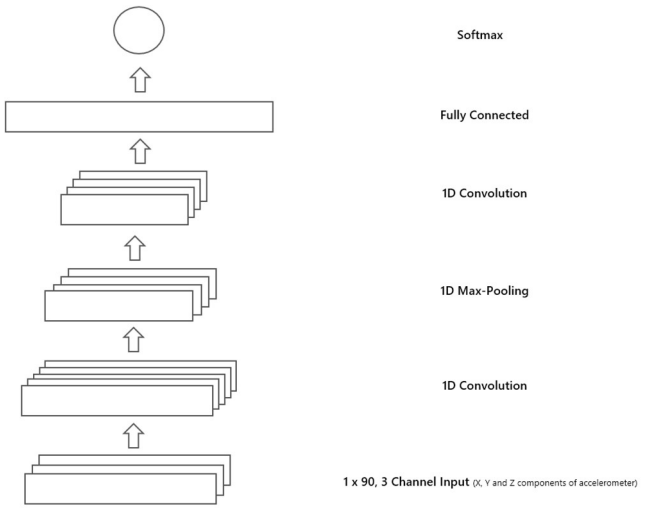

Figure 2: CNN Architecture [1]

The model is implemented in TensorFlow. The CNN features 2 x 1D convolutional layers 1x100, 3-channel input. Where the 1x100 samples the x,y, or z direction acceleration, which is capture seperatedly in the 3 channels. The max pooling layers aggreate the result from the 2 convolutional layers. A softmax layer at the very end aggreate all the result and chose the most likely device.

Since each sample is suppose to be 300 length long. It is also discovered that the device with least amount of data have at least 3000 samples. Since the data set is very large and difficult for the computer to handle, the data is sub sampled at 1/6 of total size. The data is then subdivided into training and testing data with 80:20 split. After training the model, the following results are recorded for each epoch of training. Using a small learning rate of 0.0001 to avoid effects of noise, the model obtain the best result of Training Loss: 6.74998, Training Accuracy: 0.663291, Testing Accuracy: 0.590099 shown in Figure 3. Figure 4 shows the training result across 300 epoch.

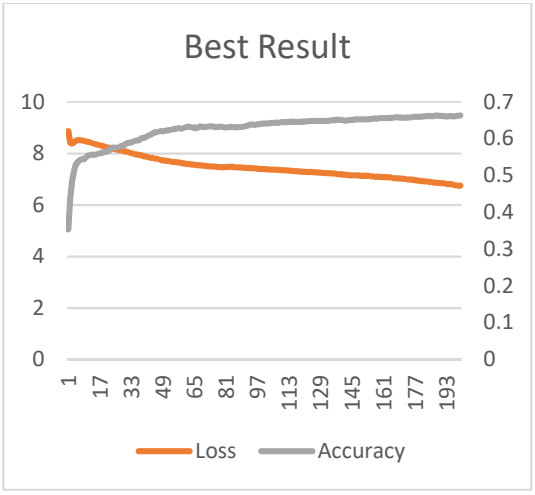

Figure 3: Best Result

Figure 4: Training result across 300 epoch

The progression start to stop around 200 epoch, which is how the best result were obtain. Although the result are not bad, it is promising. If better hareware were to be used and train on the full dataset, then the result could be significantly improved. In addition, the computer was unable to process the testing file since it is so large.

3.2 Hidden Markov Model (HMM)

Pattern recognition could be a used to determine the user based on the pattern from the gathered accelerometer data. The rerequied pattern recogniton would make use of the vitebri formulation of the Hidden Markov Model. Since the states of the daly activities are uunknown, the states can be infered by trainin the models for each device. The Viterbi-algorithm is used to find the most likely underlying explanation of a sequence of observation. The hidden states will correspond to the actions by the user and it is important to model with the appropriate state count otherwise the model might get over or under fit incase more to many or too few states are used respectively.

For the HMM, the score for each device could be improved using Expectation Maximization (EM) during the training sessions [6]. The EM methods include KMeans however it is much better to use Gaussian Mixture HMM (GMHMM) since the data is non-linear.

The HMM model is trained using the Baum Welch algorithm which resolves the learning problem with a given observation, what is the most likely model. Furthermore, the Veterbi algorithm resolves the decoding problem where with an observation the states can be inferred. Lastly, for evaluation purposes, the forward algorithm resolves the problem where given a sequence of observation and a model, what is the probability of the sequence being generated by the model.

3.3 Fourier Transformation with traditional classifier

The essense of this approach relies on some of the concepts mentioned about the sequential nature of the data. Specifically, since the data is sequential, it is likely that certain patterns that emerge from the data. Furthermore, since a pattern is a phenomena that repeats itself, one could also describe the data to be cyclical in nature, and would have a corresponding frequency.

A method is then needed to analyze the frequencies and magnitudes of patterns in descrete data. Forturnately, the fourier transform is a prime candidate for shaping to a more usable form.

3.3.1 Fourier Transformation

Fourier transformation is a an algorithm that decomposes data that is measured across time into the frequencies that make up the signal. The Fourier transformation would be applied to a set of data to form a corresponding set of data. When the data generated by the fourier transfomration is graphed, it shows peaks at the frequencies that are dominant in the original data. Figure 2 shows a sample signal used for demonstration.

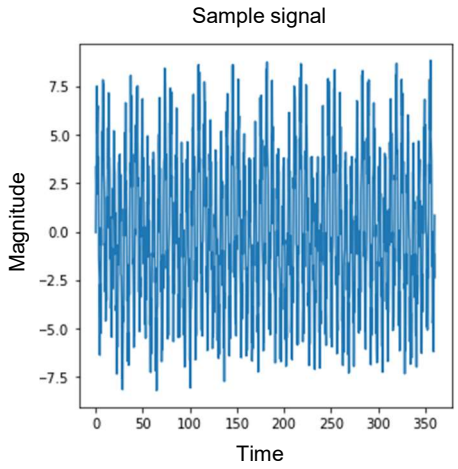

Figure 2. A sample signal constructed from sine waves

The signal shown in figure 2 is constructed from 4 sine waves, with frequencies of 10.4, 50, 400 and 60 and magnitudes 1.5, 1, 2 and 5, respectively. Figure 3 shows the corresponding fourier transform signal.

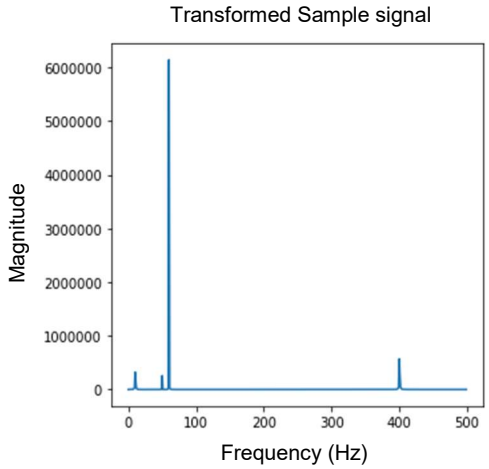

Figure 3. The sample signal transformed

One would notice that most of the transformed signal does not contain much activitiy. The transformed signal is mostly dormant with the exceptions of the 4 peaks in the graph. One could easily see that the peaks appear at the frequencies that was previously mentioned of the sine waves that make up the signal. Also the dominant frequency, which in this case would be 60 hertz as its sine wave as the largest amplitude, is also shown in the transformed signal with the largest magnitude.

3.3.2 Pipeline

There are a couple of observations that could be made from fourier transformation. The first of which is that observing the original at a certain window should not affect the observed frequencied, given that the window is large enough to capture the lower frequencies in the signal. This becomes crucial for a machine learning algorithm. As one can be given a small test set of data , one can transform the small test set into the frequency domain using the fourier transform, and the frequencies of the sample signal should closely match the frequencies of the training data of the same device.

Another issue that the fourier transform helps to solve is the issue with size. The training and test dataset is extremely large, which makes the training and testing the data extremely cumbersome and difficult. However, given that cyclical data tends to be uniform, it is possible to get the dominant frequencies, which typically occur at the lower end of the spectrum, and discard the rest of the transformed data. It is then possible to deal with a much more compressed version of the data using fourier transforms.

Hence the training data would then be grouped and separated by device. Due to the size of the dataset, the grouped data would then be saved into files, with each file labeled by the corresponding device number. Then each of the newly created files would be iterated through, where the “x”, “y”, “z” coordinates would be separated and grouped into arrays. The fourier transformation would then be applied to these arrays. In order to compress the data, any data above 150 hertz in the transformed signal would be truncated. This entails that the orignial array of 600,000 samples would be compressed into an array of only 150 values This transformed data would then be used used to train the classifier.

Given the test data, it would then be transformed using fourier transformation, in order to match the format in which the data was fed into the classifier for training. It is also important to note that each of the test datasets consists of 300 samples. After the test dataset is transformed, half of data at the end of the signal is eliminated due to the symmetrical nature of signal. The result is that each of the test datasets output an array of 150 samples, which conveniently matches the appropriate size that is fed into the classifier. Figure 4 shows an example of the transformed sample signal.

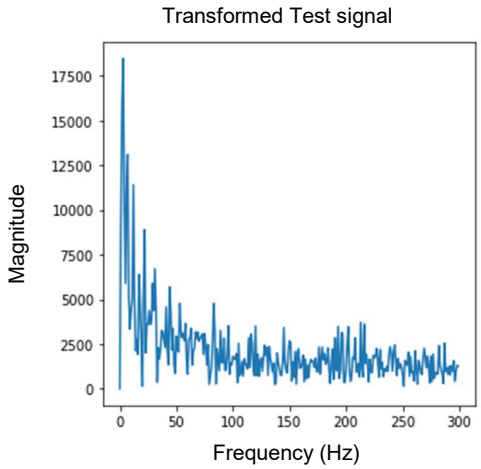

Figure 4. The sample signal transformed

4 Results

4.1 CNN

The CCN method obtain best result of Training Loss: 6.74998, Training Accuracy: 0.663291, Testing Accuracy: 0.590099 shown in Figure 3. This is with 1/6 of the data given. The hardware was also unable to handle the large test file size and therefore could not generate result for Kaggle.

4.2 HMM

The HMM method trained a model per device with each model containing 3 components. This was the best conponent count to use for the classification as determide by determinining the optimal component number using AI and BIC evaluation[6]. The 3 components may correspond to the 3 states where the user goes through the morning, afternoon and evening activities. The evalution was perfomed on a subset of the devices which produced a 50% accurcy score in identifying the devices from the csv with answers file of the posed questions. The score can be seen in the screenshot below.

Figure 5. Kaggle GMMHMM Submission Score

4.3 Fourier

For the Fourier Transfromation method, the results were less promising. The method failed to reliably estimate any of the test samples provided. Specifically every test sample would be predicted to belong to device 1017. This phenomena was recreated for the “x”, “y”, “z” data.

5 Discussion

The HMM methodoly took a long time to run and classify all the data. It ws not possible to have the entire dataset classified and scored in time to submit to Kaggle and get a accuracy score. This was due to a large dataset and it could be worth exploring other methods to speed this up and obtain resonable results. However, the subset of submitted data did show a 50% accuracy, proving that the method is possible with further exploration and imrovements. Furthermore, the implimentation made use of libraries that turned out to be deprecated and that slowed down the development time since work arounds and fixes needed to be implimented to get the code working accordingly. The large file size was also a huge problem for the CNN method, which used only 1/6 of the data and was unable to process result.csv.

A number of theories might explain the lackluster performance of the Fourier Transfromation method, but the dominant among them would be that the nature of the data of the test sample is vastly different from that of the training data. This might have caused the fourier transformations to differ significantly. This could be also explained as lower frequencies, might appear more dominant over the a large dataset, as opposed to a smaller one. Another theory would be that the data is too similar in order to create a discernable difference that could be recognized.

Using an ensemble method to combine all the methods and get a better final score overall could be viable option to pursue, though not guaranteed to provide better results than each method in isolation.

6 Conclusions

Even though the HMM failed to classify the entire dataset, 50% on a subset is great score. The time that it took to train and validate showed that more work is needed to make the system robust and fast as a fast-biometric security system. The provided dataset is quite large and classifying a user might take some time which both are very impractical. The system would time a long time to classify anything and hence it is more likely to be a system that helps stop fraudulent activities, such as bank systems. Furthermore, it is assumed that the user has their device in their pocket or in hand at all times during the recording of the accelerometer. This might not be necessarily true as many users could place their mobile devices on other surfaces making the data inconsistent in the classification.

The Fourier transformation method has not shown much promising results, so it might be advisable to recommend that using a Fourier transformation along with a supervised classifier does not yield accurate results. However, Fourier transformations are commonly used as a compression algorithm, and might prove useful when combined with other methods discussed in the report, such as the HMM, in order to vastly improve the runtime performance of the machine learning algorithm.

The CNN method requires large amount of computational power and was unable to process all the data given. However, it was still able to achieve almost 60% accuracy. With better hardware, this method should be explored further.

References

[1] A. Saeed, "Implementing a CNN for Human Activity Recognition in Tensorflow", *Aqibsaeed.github.io*, 2018. [Online]. Available: http://aqibsaeed.github.io/2016-11-04-human-activity-recognition-cnn/. [Accessed: 24- Apr- 2018].

[2] "Using Hidden Markov Models for accelerometer-based biometric gait recognition - IEEE Conference Publication", *Ieeexplore.ieee.org*, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/5759842/. [Accessed: 24- Apr- 2018].

[3] "Accelerometer Biometric Competition | Kaggle", *Kaggle.com*, 2018. [Online]. Available: https://www.kaggle.com/c/accelerometer-biometric-competition#description. [Accessed: 24- Apr- 2018].
[4] "Sensors Overview | Android Developers", *Developer.android.com*, 2018. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.html. [Accessed: 24- Apr- 2018].
[5] *Users-cs.au.dk*, 2018. [Online]. Available: http://users-cs.au.dk/cstorm/courses/PRiB_f12/slides/hidden-markov-models-1.pdf. [Accessed: 25- Apr- 2018].
[6] B. Christopher and B. Christopher, "Intro to Expectation-Maximization, K-Means, Gaussian Mixture Models with Python, Sklearn", *BLACKARBS LLC*, 2018. [Online]. Available: http://www.blackarbs.com/blog/intro-to-expectation-maximization-k-means-gaussian-mixture-models-with-python-sklearn/3/20/2017. [Accessed: 25- Apr- 2018].
[7] "(Deprecated) Hidden Markov Models", *Mahout.apache.org*, 2018. [Online]. Available: https://mahout.apache.org/docs/latest/algorithms/map-reduce/classification/hidden-markov-models.html. [Accessed: 25- Apr- 2018].
[8] *Inf.ed.ac.uk*, 2018. [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/asr/2012-13/asr03-hmmgmm-4up.pdf. [Accessed: 25- Apr- 2018].
[9] "HMMs Pattern Recognition", UKEssays, 2018. [Online]. Available: https://www.ukessays.com/essays/engineering/hmm-pattern-recognition-9997.php. [Accessed: 25- Apr- 2018].

Appendix

The following segments below show code for different methods used in this research.

6.1 Fourier transfoms with traditional classifiers

```
import matplotlib.pyplot as plt

def plot_single(X,t=None):
    plt.figure(figsize=(5, 5))
    if t is not None:
        plt.plot(X, t)
    else:
        plt.plot(X)
    plt.show()

from scipy.fftpack import fft, ifft
def get_fourier_freqs(data_set_unshaped):

    data_set = data_set_unshaped - np.mean(data_set_unshaped)
    data_length = len(data_set)

    y_train = fft(data_set)
    y_train = y_train[range(int(len(y_train)/2))]

    return abs(y_train[:150])




file_name = "train.csv"
no_of_lines = sum(1 for line in open(file_name))
with open(file_name) as train_file:
    current_device_file = None
    current_device_file_no = -1

    for line_no in range(0,no_of_lines):
        line = train_file.readline()
        if line == "T,X,Y,Z,Device\n":
            line = train_file.readline()
            line_no = line_no + 1
        t,x,y,z,device = line.split(',')
        device = device[:-1]
        if device != current_device_file_no:
            print("device: " + str(device))
            current_device_file_no = device
            current_device_file                                                    =
open("data/all_the_data/"+current_device_file_no+".csv", 'a')
        current_device_file.write(t + "," + x + "," + y + "," + z + '\n')

        if not (line_no % 1000):
            clear_output()
            print("device: " + str(line_no))



# gathered manually
device_ids=[7,8,9,12,23,25,26,27,33,37,39,45,47,51,52,57,58,65,67,68,70,71,73,74,7
5,78,79,81,87,89,90,91,92,94,95,96,99,104,105,108,110,111,116,117,120,122,124,126,
```

```
127,129,134,137,142,145,148,149,152,156,157,158,159,162,163,168,169,174,175,177,18
3,187,188,189,190,194,196,204,206,207,211,213,216,219,222,224,229,232,233,234,236,
237,239,240,261,263,268,269,270,271,273,274,275,277,281,282,283,284,285,289,290,29
1,294,296,297,298,299,302,306,309,312,313,314,323,325,333,335,338,341,343,344,345,
350,360,361,366,369,370,371,376,378,381,390,394,398,399,401,404,411,412,413,415,41
7,421,422,423,425,433,438,447,448,455,461,463,466,471,473,477,478,479,482,485,486,
487,491,492,494,501,503,505,507,509,514,515,518,520,523,524,528,531,533,534,536,53
7,539,547,550,552,553,554,556,557,562,568,571,573,574,575,577,579,580,581,583,589,
593,594,595,596,600,601,607,610,611,612,613,614,617,621,622,626,627,629,632,634,63
8,640,642,643,646,647,650,653,656,658,660,661,663,664,665,666,667,669,670,671,674,
675,676,678,679,680,681,682,683,684,687,688,690,691,692,694,696,698,699,700,703,70
5,706,709,710,711,713,714,720,721,722,727,728,729,730,732,735,736,738,739,745,746,
750,751,754,755,757,761,762,763,764,768,770,774,776,781,782,784,789,792,793,795,80
1,802,804,805,806,810,812,814,818,820,823,824,827,834,836,838,839,841,842,846,847,
848,854,857,859,860,862,864,868,870,871,877,880,882,883,887,890,895,897,900,911,91
2,913,919,933,941,943,945,953,955,956,967,973,977,979,983,987,991,992,996,997,998,
1000,1006,1015,1017,1027,1029,1031,1033,1035,1036,1037]


fourier_freqs_X = []

for device_id in device_ids:
    buffer = []
    print("device: "+ str(device_id))
    for   df   in   pd.read_csv("data/all_the_data/"+   str(device_id)   +".csv",
chunksize=1024):
        df.columns = ['T','X','Y','Z']
        buffer.extend(df['X'])
    fourier_freqs_single_device = get_fourier_freqs(buffer)
    fourier_freqs_X.append(fourier_freqs_single_device)

fourier_freqs_Y = []

for device_id in device_ids:
    buffer = []
    print("Y, device: "+ str(device_id))
    for   df   in   pd.read_csv("data/all_the_data/"+   str(device_id)   +".csv",
chunksize=1024):
        df.columns = ['T','X','Y','Z']
        buffer.extend(df['Y'])
    fourier_freqs_single_device = get_fourier_freqs(buffer)
    fourier_freqs_Y.append(fourier_freqs_single_device)


fourier_freqs_Z = []

for device_id in device_ids:
    buffer = []
    print("Z, device: "+ str(device_id))
    for   df   in   pd.read_csv("data/all_the_data/"+   str(device_id)   +".csv",
chunksize=1024):
        df.columns = ['T','X','Y','Z']
        buffer.extend(df['Z'])
    fourier_freqs_single_device = get_fourier_freqs(buffer)
    fourier_freqs_Z.append(fourier_freqs_single_device)



from sklearn.naive_bayes import GaussianNB
```

```
clf_X = GaussianNB()
clf_X.fit(fourier_freqs_X, device_ids)

clf_Y = GaussianNB()
clf_Y.fit(fourier_freqs_Y, device_ids)

clf_Z = GaussianNB()
clf_Z.fit(fourier_freqs_Z, device_ids)



file_name = "test.csv"
no_of_lines = sum(1 for line in open(file_name))

with open(file_name) as train_file:
    current_device_file = None
    current_device_file_no = -1

    for line_no in range(0,no_of_lines):
        line = train_file.readline()
        if line == "T,X,Y,Z,SequenceId\n":
            line = train_file.readline()
            line_no = line_no + 1
        t,x,y,z,device = line.split(',')
        device = device[:-1]
        if device != current_device_file_no:
            print(device)
            current_device_file_no = device
            current_device_file                                                    =
open("data/test_data/"+current_device_file_no+".csv", 'a')
        current_device_file.write(t + "," + x + "," + y + "," + z + '\n')



file_name = "questions.csv"
no_of_lines = sum(1 for line in open(file_name))

predictions = []

with open(file_name) as train_file:
    quiz_fourier_freqs
    for line_no in range(0,no_of_lines):
        line = train_file.readline()
        if line == "QuestionId,SequenceId,QuizDevice\n":
            line = train_file.readline()
            line_no = line_no + 1
        questionId,sequenceId,quizDevice = line.split(',')
        quizDevice = quizDevice[:-1]
        with open("data/test_data/"+sequenceId+".csv") as quiz_file:
            quiz_buffer_X = []
            quiz_buffer_Y = []
            quiz_buffer_Z = []
            for quiz_line_no in range(0,300):
                quiz_line = quiz_file.readline()
                t,x,y,z = quiz_line.split(',')
                z = z[:-1]
                quiz_buffer_X.append(float(x))
                quiz_buffer_Y.append(float(y))
                quiz_buffer_Z.append(float(z))
```

```
        quiz_fourier_freqs_X = get_fourier_freqs(quiz_buffer_X)
        quiz_fourier_freqs_Y = get_fourier_freqs(quiz_buffer_Y)
        quiz_fourier_freqs_Z = get_fourier_freqs(quiz_buffer_Z)
        prediction_X = str(clf_X.predict([quiz_fourier_freqs_X])[0])
        print("start")
        print("prediction X: " + prediction_X)
        prediction_X = str(clf_X.predict([quiz_fourier_freqs_X])[0])
        print("prediction Y: " + prediction_Y)
        prediction_Y = str(clf_Y.predict([quiz_fourier_freqs_Y])[0])
        print("prediction Z: " + prediction_Z)
        prediction_Z = str(clf_Z.predict([quiz_fourier_freqs_Z])[0])
        print("professed device: " + quizDevice)
```

Appendix B HMM Code

```
import pandas as pd
import numpy as np

import csv

train = {}
test = {}
qns = {}

for df in pd.read_csv('train.csv', chunksize=13981):
    devices = df['Device'].unique()

    for device in devices:
        if device not in train:
            train[device] = pd.DataFrame(columns=['T','X','Y','Z','Device'])
        train[device] = pd.concat([train[device], df[df['Device'] == device]])

for df in pd.read_csv("test.csv", chunksize=13981):
    seqs = df['SequenceId'].unique()

    for seq in seqs:
        if seq not in test:
            test[seq] = pd.DataFrame(columns=['T', 'X', 'Y', 'Z', 'SequenceId'])
        test[seq] = pd.concat([test[seq], df[df['SequenceId'] == seq]])

qns = pd.DataFrame(columns=['QuestionId', 'SequenceId', 'QuizDevice'])

for df in pd.read_csv("questions.csv", chunksize=13981):
    qns = pd.concat([qns, df])

import pandas as pd
import operator

from hmmlearn.hmm import GMMHMM

import warnings
warnings.filterwarnings('ignore')

def learn():
    models = {}
    to_train = [];
    for device, data in train.items():

    if device > 45: #use only subset of devices
```

```
                break
        d = np.column_stack([data.X.tolist(), data.Y.tolist(), data.Z.tolist()])
        if len(to_train) == 0:
            to_train = d
        else:
            to_train = np.concatenate([to_train, d])

        m = GMMHMM(n_components=3)
        m.fit(to_train)
        models[device] = m

#         print("Transition matrix")
#         print(m.transmat_)
#         print()

    return models


def evaluate(models):
    ans = {}
    ans ['QuestionId'] = []
    ans ['IsTrue'] = []

    for qn_index, qn in qns.iterrows():
        to_test = test[qn ['SequenceId']]

        if qn ['QuizDevice'] not in models.keys():
            ans['QuestionId'].append(qn ['QuestionId'])
            ans['IsTrue'].append(0)
            continue

        did = qn ['QuizDevice']

        model = models[did]
        t      =      np.column_stack([to_test.X.tolist(),      to_test.Y.tolist(),
to_test.Z.tolist()])
        score = model.score(t)

        if score > -2100:
            isTrue = 1
        else:
            isTrue = 0

        ans ['QuestionId'].append(qn ['QuestionId'])
        ans ['IsTrue'].append(isTrue)

    df = pd.DataFrame(ans)
    df.to_csv('hmm_answers.csv', ',', header=True, columns=["QuestionId","IsTrue"],
index=False)

if __name__=='__main__':
    evaluate(learn())
```

Appendix C CNN Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

```
import tensorflow as tf

%matplotlib inline
plt.style.use('ggplot')
```

In [2]:

```
def read_data(file_path):
    data = pd.read_csv(file_path,header = 0)
    return data
```

In [3]:

```
def feature_normalize(dataset):
    mu = np.mean(dataset,axis = 0)
    sigma = np.std(dataset,axis = 0)
    return (dataset - mu)/sigma
```

In [4]:

```
def plot_axis(ax, x, y, title):
    ax.plot(x, y)
    ax.set_title(title)
    ax.xaxis.set_visible(False)
    ax.set_ylim([min(y) - np.std(y), max(y) + np.std(y)])
    ax.set_xlim([min(x), max(x)])
    ax.grid(True)
```

In [5]:

```
def plot_activity(device,data):
    fig, (ax0, ax1, ax2) = plt.subplots(nrows = 3, figsize = (15, 10), sharex = True)
    plot_axis(ax0, data['T'], data['X'], 'X')
    plot_axis(ax1, data['T'], data['Y'], 'Y')
    plot_axis(ax2, data['T'], data['Z'], 'Z')
    plt.subplots_adjust(hspace=0.2)
    fig.suptitle(device)
    plt.subplots_adjust(top=0.90)
    plt.show()
```

In [6]:

```
def windows(data, size):
    start = 0
    while start < data.count() and data[start] == data[start+size]:
        yield int(start), int(start + size)
        start += size + 500
```

In [7]:

```
def segment_signal(data,window_size = 90):
    segments = np.empty((0,window_size,3))
    labels = np.empty((0))
    for (start, end) in windows(data['Device'], window_size):
        x = data["X"][start:end]
        y = data["Y"][start:end]
        z = data["Z"][start:end]
        if(len(dataset['T'][start:end]) == window_size and len(np.unique(data["Device"][start:end])) == 1):
            segments = np.vstack([segments,np.dstack([x,y,z])])
            labels = np.append(labels,stats.mode(data["Device"][start:end])[0][0])
    return segments, labels
```

In [8]:

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev = 0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.0, shape = shape)
    return tf.Variable(initial)

def depthwise_conv2d(x, W):
    return tf.nn.depthwise_conv2d(x,W, [1, 1, 1, 1], padding='VALID')

def apply_depthwise_conv(x,kernel_size,num_channels,depth):
    weights = weight_variable([1, kernel_size, num_channels, depth])
    biases = bias_variable([depth * num_channels])
    return tf.nn.relu(tf.add(depthwise_conv2d(x, weights),biases))

def apply_max_pool(x,kernel_size,stride_size):
    return tf.nn.max_pool(x, ksize=[1, 1, kernel_size, 1],
                          strides=[1, 1, stride_size, 1], padding='VALID')
```

In [9]:

```
dataset = read_data('train.csv')
```

In [10]:

```
dataset['X'] = feature_normalize(dataset['X'])
dataset['Y'] = feature_normalize(dataset['Y'])
dataset['Z'] = feature_normalize(dataset['Z'])
```

In [11]:

```
for device in np.unique(dataset["Device"]):
    subset = dataset[dataset["Device"] == device][:180]
    # plot_activity(device,subset)
```

In [12]:

```
segments, labels = segment_signal(dataset)
labels = np.asarray(pd.get_dummies(labels), dtype = np.int8)
reshaped_segments = segments.reshape(len(segments), 1,90, 3)
```

In [13]:

```
train_test_split = np.random.rand(len(reshaped_segments)) < 0.70
train_x = reshaped_segments[train_test_split]
train_y = labels[train_test_split]
test_x = reshaped_segments[~train_test_split]
test_y = labels[~train_test_split]
```

In [14]:

```
input_height = 1
input_width = 90
num_labels = labels.shape[1]
num_channels = 3

batch_size = 10
kernel_size = 60
depth = 60
num_hidden = 1000

learning_rate = 0.0001
training_epochs = 200

total_batches = train_x.shape[0] // batch_size
```

In [15]:

```
X = tf.placeholder(tf.float32, shape=[None,input_height,input_width,num_channels])
Y = tf.placeholder(tf.float32, shape=[None,num_labels])

c = apply_depthwise_conv(X,kernel_size,num_channels,depth)
p = apply_max_pool(c,20,2)
c = apply_depthwise_conv(p,6,depth*num_channels,depth//10)

shape = c.get_shape().as_list()
c_flat = tf.reshape(c, [-1, shape[1] * shape[2] * shape[3]])

f_weights_l1 = weight_variable([shape[1] * shape[2] * depth * num_channels * (depth//10), num_hidden])
f_biases_l1 = bias_variable([num_hidden])
f = tf.nn.tanh(tf.add(tf.matmul(c_flat, f_weights_l1),f_biases_l1))

out_weights = weight_variable([num_hidden, num_labels])
out_biases = bias_variable([num_labels])
y_ = tf.nn.softmax(tf.matmul(f, out_weights) + out_biases)
```

In [16]:

```
loss = -tf.reduce_sum(Y * tf.log(y_))
optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate).minimize(loss)

correct_prediction = tf.equal(tf.argmax(y_,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

In [17]:

```
cost_history = np.empty(shape=[1],dtype=float)

with tf.Session() as session:
    tf.global_variables_initializer().run()
    for epoch in range(training_epochs):
        for b in range(total_batches):
            offset = (b * batch_size) % (train_y.shape[0] - batch_size)
            batch_x = train_x[offset:(offset + batch_size), :, :, :]
            batch_y = train_y[offset:(offset + batch_size), :]
            _, c = session.run([optimizer, loss],feed_dict={X: batch_x, Y : batch_y})
            cost_history = np.append(cost_history,c)
        print ("Epoch: ",epoch," Training Loss: ",c," Training Accuracy: ",
               session.run(accuracy, feed_dict={X: train_x, Y: train_y}))

    print ("Testing Accuracy:", session.run(accuracy, feed_dict={X: test_x, Y: test_y}))
```

```
Epoch:  0  Training Loss:  8.87247  Training Accuracy:  0.353586
Epoch:  1  Training Loss:  8.41493  Training Accuracy:  0.439662
Epoch:  2  Training Loss:  8.38326  Training Accuracy:  0.486076
Epoch:  3  Training Loss:  8.43171  Training Accuracy:  0.51308
Epoch:  4  Training Loss:  8.49893  Training Accuracy:  0.530802
Epoch:  5  Training Loss:  8.51236  Training Accuracy:  0.538397
Epoch:  6  Training Loss:  8.51316  Training Accuracy:  0.54346
Epoch:  7  Training Loss:  8.50705  Training Accuracy:  0.545148
Epoch:  8  Training Loss:  8.49451  Training Accuracy:  0.544304
Epoch:  9  Training Loss:  8.47413  Training Accuracy:  0.549367
Epoch:  10  Training Loss:  8.45501  Training Accuracy:  0.553586
Epoch:  11  Training Loss:  8.42754  Training Accuracy:  0.555274
Epoch:  12  Training Loss:  8.40189  Training Accuracy:  0.556962
Epoch:  13  Training Loss:  8.37846  Training Accuracy:  0.556118
Epoch:  14  Training Loss:  8.35979  Training Accuracy:  0.556962
Epoch:  15  Training Loss:  8.33864  Training Accuracy:  0.55865
Epoch:  16  Training Loss:  8.31569  Training Accuracy:  0.560338
Epoch:  17  Training Loss:  8.29905  Training Accuracy:  0.561181
Epoch:  18  Training Loss:  8.27598  Training Accuracy:  0.563713
Epoch:  19  Training Loss:  8.25825  Training Accuracy:  0.563713
Epoch:  20  Training Loss:  8.23226  Training Accuracy:  0.567932
Epoch:  21  Training Loss:  8.21858  Training Accuracy:  0.568776
Epoch:  22  Training Loss:  8.1968  Training Accuracy:  0.575527
```

```
Epoch:  23  Training Loss:  8.18661  Training Accuracy:  0.576371
Epoch:  24  Training Loss:  8.16381  Training Accuracy:  0.575527
Epoch:  25  Training Loss:  8.14766  Training Accuracy:  0.576371
Epoch:  26  Training Loss:  8.13426  Training Accuracy:  0.578059
Epoch:  27  Training Loss:  8.11505  Training Accuracy:  0.580591
Epoch:  28  Training Loss:  8.0952  Training Accuracy:  0.580591
Epoch:  29  Training Loss:  8.07791  Training Accuracy:  0.585654
Epoch:  30  Training Loss:  8.06146  Training Accuracy:  0.588186
Epoch:  31  Training Loss:  8.03968  Training Accuracy:  0.58903
Epoch:  32  Training Loss:  8.01018  Training Accuracy:  0.589873
Epoch:  33  Training Loss:  7.99144  Training Accuracy:  0.592405
Epoch:  34  Training Loss:  7.97643  Training Accuracy:  0.594093
Epoch:  35  Training Loss:  7.95544  Training Accuracy:  0.595781
Epoch:  36  Training Loss:  7.94154  Training Accuracy:  0.596624
Epoch:  37  Training Loss:  7.92838  Training Accuracy:  0.601688
Epoch:  38  Training Loss:  7.90235  Training Accuracy:  0.602532
Epoch:  39  Training Loss:  7.88701  Training Accuracy:  0.603376
Epoch:  40  Training Loss:  7.87491  Training Accuracy:  0.606751
Epoch:  41  Training Loss:  7.84982  Training Accuracy:  0.609283
Epoch:  42  Training Loss:  7.83577  Training Accuracy:  0.61097
Epoch:  43  Training Loss:  7.82556  Training Accuracy:  0.61519
Epoch:  44  Training Loss:  7.80366  Training Accuracy:  0.617721
Epoch:  45  Training Loss:  7.79291  Training Accuracy:  0.617721
Epoch:  46  Training Loss:  7.77342  Training Accuracy:  0.619409
Epoch:  47  Training Loss:  7.75285  Training Accuracy:  0.621097
Epoch:  48  Training Loss:  7.73765  Training Accuracy:  0.619409
Epoch:  49  Training Loss:  7.72503  Training Accuracy:  0.621097
Epoch:  50  Training Loss:  7.7157  Training Accuracy:  0.621941
Epoch:  51  Training Loss:  7.70735  Training Accuracy:  0.623629
Epoch:  52  Training Loss:  7.68876  Training Accuracy:  0.624473
Epoch:  53  Training Loss:  7.67933  Training Accuracy:  0.62616
Epoch:  54  Training Loss:  7.66756  Training Accuracy:  0.627004
Epoch:  55  Training Loss:  7.65617  Training Accuracy:  0.627848
Epoch:  56  Training Loss:  7.64412  Training Accuracy:  0.629536
Epoch:  57  Training Loss:  7.63069  Training Accuracy:  0.627848
Epoch:  58  Training Loss:  7.61952  Training Accuracy:  0.628692
Epoch:  59  Training Loss:  7.60966  Training Accuracy:  0.63038
Epoch:  60  Training Loss:  7.603  Training Accuracy:  0.632068
Epoch:  61  Training Loss:  7.59104  Training Accuracy:  0.632911
Epoch:  62  Training Loss:  7.58485  Training Accuracy:  0.632068
Epoch:  63  Training Loss:  7.56951  Training Accuracy:  0.63038
Epoch:  64  Training Loss:  7.56532  Training Accuracy:  0.631224
Epoch:  65  Training Loss:  7.55179  Training Accuracy:  0.628692
Epoch:  66  Training Loss:  7.5509  Training Accuracy:  0.631224
Epoch:  67  Training Loss:  7.53929  Training Accuracy:  0.633755
Epoch:  68  Training Loss:  7.53323  Training Accuracy:  0.632911
Epoch:  69  Training Loss:  7.51863  Training Accuracy:  0.632068
Epoch:  70  Training Loss:  7.51516  Training Accuracy:  0.632068
Epoch:  71  Training Loss:  7.50542  Training Accuracy:  0.633755
Epoch:  72  Training Loss:  7.50278  Training Accuracy:  0.633755
Epoch:  73  Training Loss:  7.50442  Training Accuracy:  0.633755
Epoch:  74  Training Loss:  7.4915  Training Accuracy:  0.632068
Epoch:  75  Training Loss:  7.48361  Training Accuracy:  0.632068
Epoch:  76  Training Loss:  7.47811  Training Accuracy:  0.632068
Epoch:  77  Training Loss:  7.47648  Training Accuracy:  0.632911
Epoch:  78  Training Loss:  7.46747  Training Accuracy:  0.632911
Epoch:  79  Training Loss:  7.47075  Training Accuracy:  0.631224
Epoch:  80  Training Loss:  7.47478  Training Accuracy:  0.631224
Epoch:  81  Training Loss:  7.47545  Training Accuracy:  0.631224
Epoch:  82  Training Loss:  7.48265  Training Accuracy:  0.632068
Epoch:  83  Training Loss:  7.48371  Training Accuracy:  0.632911
Epoch:  84  Training Loss:  7.47897  Training Accuracy:  0.631224
Epoch:  85  Training Loss:  7.46643  Training Accuracy:  0.631224
Epoch:  86  Training Loss:  7.46336  Training Accuracy:  0.632068
Epoch:  87  Training Loss:  7.46318  Training Accuracy:  0.632068
Epoch:  88  Training Loss:  7.45128  Training Accuracy:  0.632068
Epoch:  89  Training Loss:  7.44718  Training Accuracy:  0.632911
Epoch:  90  Training Loss:  7.4423  Training Accuracy:  0.634599
Epoch:  91  Training Loss:  7.43855  Training Accuracy:  0.635443
Epoch:  92  Training Loss:  7.43204  Training Accuracy:  0.637975
Epoch:  93  Training Loss:  7.42448  Training Accuracy:  0.638819
Epoch:  94  Training Loss:  7.42072  Training Accuracy:  0.637975
Epoch:  95  Training Loss:  7.41903  Training Accuracy:  0.637975
Epoch:  96  Training Loss:  7.41152  Training Accuracy:  0.639663
Epoch:  97  Training Loss:  7.40618  Training Accuracy:  0.639663
Epoch:  98  Training Loss:  7.39901  Training Accuracy:  0.640506
Epoch:  99  Training Loss:  7.40475  Training Accuracy:  0.64135
Epoch:  100  Training Loss:  7.3891  Training Accuracy:  0.64135
Epoch:  101  Training Loss:  7.3829  Training Accuracy:  0.64135
Epoch:  102  Training Loss:  7.37886  Training Accuracy:  0.642194
Epoch:  103  Training Loss:  7.37766  Training Accuracy:  0.643038
Epoch:  104  Training Loss:  7.37595  Training Accuracy:  0.643038
Epoch:  105  Training Loss:  7.37227  Training Accuracy:  0.643038
Epoch:  106  Training Loss:  7.35802  Training Accuracy:  0.643882
Epoch:  107  Training Loss:  7.35862  Training Accuracy:  0.643038
```

```
Epoch:  108  Training Loss:  7.35348  Training Accuracy:  0.644726
Epoch:  109  Training Loss:  7.35128  Training Accuracy:  0.64557
Epoch:  110  Training Loss:  7.34515  Training Accuracy:  0.644726
Epoch:  111  Training Loss:  7.33798  Training Accuracy:  0.64557
Epoch:  112  Training Loss:  7.33819  Training Accuracy:  0.64557
Epoch:  113  Training Loss:  7.32448  Training Accuracy:  0.646414
Epoch:  114  Training Loss:  7.32029  Training Accuracy:  0.64557
Epoch:  115  Training Loss:  7.31177  Training Accuracy:  0.64557
Epoch:  116  Training Loss:  7.31022  Training Accuracy:  0.64557
Epoch:  117  Training Loss:  7.30013  Training Accuracy:  0.64557
Epoch:  118  Training Loss:  7.29822  Training Accuracy:  0.64557
Epoch:  119  Training Loss:  7.29427  Training Accuracy:  0.646414
Epoch:  120  Training Loss:  7.2901  Training Accuracy:  0.646414
Epoch:  121  Training Loss:  7.28819  Training Accuracy:  0.647257
Epoch:  122  Training Loss:  7.2782  Training Accuracy:  0.647257
Epoch:  123  Training Loss:  7.28564  Training Accuracy:  0.648101
Epoch:  124  Training Loss:  7.28237  Training Accuracy:  0.648101
Epoch:  125  Training Loss:  7.28208  Training Accuracy:  0.648101
Epoch:  126  Training Loss:  7.26532  Training Accuracy:  0.648101
Epoch:  127  Training Loss:  7.26173  Training Accuracy:  0.648101
Epoch:  128  Training Loss:  7.25646  Training Accuracy:  0.648101
Epoch:  129  Training Loss:  7.25388  Training Accuracy:  0.648101
Epoch:  130  Training Loss:  7.24449  Training Accuracy:  0.648101
Epoch:  131  Training Loss:  7.24013  Training Accuracy:  0.648101
Epoch:  132  Training Loss:  7.24181  Training Accuracy:  0.648945
Epoch:  133  Training Loss:  7.23115  Training Accuracy:  0.648945
Epoch:  134  Training Loss:  7.2203  Training Accuracy:  0.650633
Epoch:  135  Training Loss:  7.22236  Training Accuracy:  0.649789
Epoch:  136  Training Loss:  7.21559  Training Accuracy:  0.650633
Epoch:  137  Training Loss:  7.19477  Training Accuracy:  0.651477
Epoch:  138  Training Loss:  7.20064  Training Accuracy:  0.650633
Epoch:  139  Training Loss:  7.18597  Training Accuracy:  0.650633
Epoch:  140  Training Loss:  7.17128  Training Accuracy:  0.648945
Epoch:  141  Training Loss:  7.17709  Training Accuracy:  0.648945
Epoch:  142  Training Loss:  7.16779  Training Accuracy:  0.649789
Epoch:  143  Training Loss:  7.15189  Training Accuracy:  0.650633
Epoch:  144  Training Loss:  7.15813  Training Accuracy:  0.650633
Epoch:  145  Training Loss:  7.14963  Training Accuracy:  0.651477
Epoch:  146  Training Loss:  7.14622  Training Accuracy:  0.652321
Epoch:  147  Training Loss:  7.14639  Training Accuracy:  0.653165
Epoch:  148  Training Loss:  7.14433  Training Accuracy:  0.653165
Epoch:  149  Training Loss:  7.13192  Training Accuracy:  0.652321
Epoch:  150  Training Loss:  7.13617  Training Accuracy:  0.653165
Epoch:  151  Training Loss:  7.13172  Training Accuracy:  0.653165
Epoch:  152  Training Loss:  7.13392  Training Accuracy:  0.653165
Epoch:  153  Training Loss:  7.12139  Training Accuracy:  0.654008
Epoch:  154  Training Loss:  7.10685  Training Accuracy:  0.654852
Epoch:  155  Training Loss:  7.11075  Training Accuracy:  0.654852
Epoch:  156  Training Loss:  7.09273  Training Accuracy:  0.655696
Epoch:  157  Training Loss:  7.10198  Training Accuracy:  0.654852
Epoch:  158  Training Loss:  7.0871  Training Accuracy:  0.65654
Epoch:  159  Training Loss:  7.09239  Training Accuracy:  0.65654
Epoch:  160  Training Loss:  7.07817  Training Accuracy:  0.65654
Epoch:  161  Training Loss:  7.08971  Training Accuracy:  0.65654
Epoch:  162  Training Loss:  7.0772  Training Accuracy:  0.65654
Epoch:  163  Training Loss:  7.0784  Training Accuracy:  0.65654
Epoch:  164  Training Loss:  7.06282  Training Accuracy:  0.657384
Epoch:  165  Training Loss:  7.04473  Training Accuracy:  0.657384
Epoch:  166  Training Loss:  7.04505  Training Accuracy:  0.659072
Epoch:  167  Training Loss:  7.03515  Training Accuracy:  0.658228
Epoch:  168  Training Loss:  7.02322  Training Accuracy:  0.658228
Epoch:  169  Training Loss:  7.02619  Training Accuracy:  0.657384
Epoch:  170  Training Loss:  7.02029  Training Accuracy:  0.657384
Epoch:  171  Training Loss:  7.01331  Training Accuracy:  0.657384
Epoch:  172  Training Loss:  6.99669  Training Accuracy:  0.658228
Epoch:  173  Training Loss:  6.9954  Training Accuracy:  0.658228
Epoch:  174  Training Loss:  6.98832  Training Accuracy:  0.658228
Epoch:  175  Training Loss:  6.98026  Training Accuracy:  0.659916
Epoch:  176  Training Loss:  6.9698  Training Accuracy:  0.659916
Epoch:  177  Training Loss:  6.95732  Training Accuracy:  0.659916
Epoch:  178  Training Loss:  6.93957  Training Accuracy:  0.659916
Epoch:  179  Training Loss:  6.93365  Training Accuracy:  0.659916
Epoch:  180  Training Loss:  6.92492  Training Accuracy:  0.66076
Epoch:  181  Training Loss:  6.91584  Training Accuracy:  0.661603
Epoch:  182  Training Loss:  6.91147  Training Accuracy:  0.661603
Epoch:  183  Training Loss:  6.89123  Training Accuracy:  0.662447
Epoch:  184  Training Loss:  6.89893  Training Accuracy:  0.661603
Epoch:  185  Training Loss:  6.88471  Training Accuracy:  0.661603
Epoch:  186  Training Loss:  6.87054  Training Accuracy:  0.662447
Epoch:  187  Training Loss:  6.86117  Training Accuracy:  0.663291
Epoch:  188  Training Loss:  6.85897  Training Accuracy:  0.662447
Epoch:  189  Training Loss:  6.84896  Training Accuracy:  0.662447
Epoch:  190  Training Loss:  6.83697  Training Accuracy:  0.661603
Epoch:  191  Training Loss:  6.83932  Training Accuracy:  0.661603
Epoch:  192  Training Loss:  6.82057  Training Accuracy:  0.66076
```

```
Epoch:  193  Training Loss:  6.80573  Training Accuracy:  0.661603
Epoch:  194  Training Loss:  6.81126  Training Accuracy:  0.661603
Epoch:  195  Training Loss:  6.78921  Training Accuracy:  0.661603
Epoch:  196  Training Loss:  6.7726  Training Accuracy:  0.66076
Epoch:  197  Training Loss:  6.77315  Training Accuracy:  0.662447
Epoch:  198  Training Loss:  6.74386  Training Accuracy:  0.662447
Epoch:  199  Training Loss:  6.74998  Training Accuracy:  0.663291
Testing Accuracy: 0.590099
```

In [19]:

```
labels.shape
```

Out[19]:

```
(1690, 3)
```