INSTRUMENT RECOGNITION

Aleks Aner Axel Cuong Joe Thomas

Business Requirement Document

- We did not have any significant updates in our BRD

Management Plan

- We have updated our Sprint Board and added a Burndown Chart.

Architecture and Design Document

- We did not have any significant updates in our Architecture and Design Documents

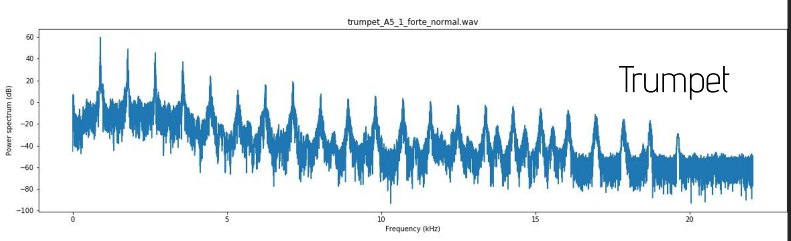
Processing Steps

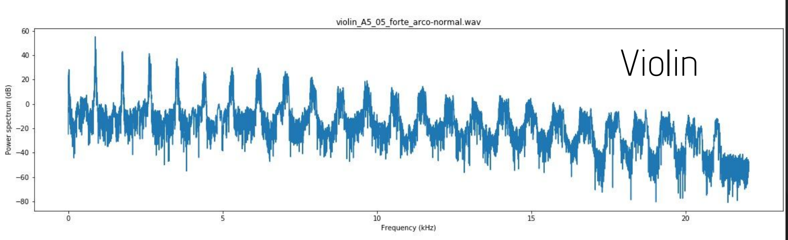
```python
fs, Audiodata = wavfile.read(r'C:\Users\Axel\Documents\All-Samples-WAV\clarinet\\'+soundWav)
#tittle for the graph will be the name of the file.
plt.title(soundWav,size=16)
#FFT
n = len(Audiodata)
AudioFreq = fft(Audiodata)
AudioFreq = AudioFreq[0:int(np.ceil((n+1)/2.0))] #Half of the spectrum
MagFreq = np.abs(AudioFreq) # Magnitude
MagFreq = MagFreq / float(n)

# power spectrum
MagFreq = MagFreq**2

if n % 2 > 0: # ffte odd
    MagFreq[1:len(MagFreq)] = MagFreq[1:len(MagFreq)] * 2
else:# fft even
    MagFreq[1:len(MagFreq) -1] = MagFreq[1:len(MagFreq) - 1] * 2

freqAxis = np.arange(0,int(np.ceil((n+1)/2.0)), 1.0) * (fs / n)

plt.plot(freqAxis/1000.0, 10*np.log10(MagFreq)) #Power spectrum
plt.title(soundWav)
plt.xlabel('Frequency (kHz)'); plt.ylabel('Power spectrum (dB)')
plt.savefig(soundWav.rstrip(".wav") + '.png')
plt.clf()
```

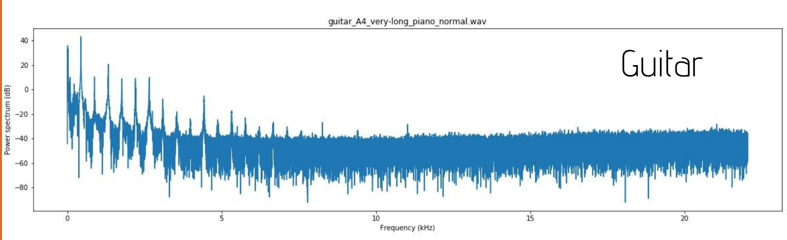
- Convert music file to a Power Spectral Density (PSD) graph using a Fourier transform

- Python libraries used:
  - `from scipy.io import wavfile`
  - `import numpy as np`
  - `import matplotlib.pyplot as plt`
  - `import os`
  - `from scipy.fftpack import fft`

Our Input Data







- X = .wav graphs
- Y = String/Integer labels

- Sets of 1,000 for training data

- Takes input graphs and outputs integer labels that can be decoded to strings of instruments

Classes

- We have completed a model with 3 instruments at the beginning.
    - Guitar
    - Trumpet
    - Violin

- We are planning to add the following classes to our model:
    - Bass Clarinet
    - Bassoon
    - Cello
    - Clarinet
    - Contrabassoon
    - Double Bass
    - English Horn
    - Flute
    - French Horn
    - Oboe
    - Saxophone
    - Trombone
    - Tuba
    - Viola

- Total: 17 Classes

Pre-Processing of Input Data

Requirements:

- 360x1440
- Gray scale (small)
- Values [0-1]
- Random

```python
#Imports the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import os, random, cv2, pickle
from PIL import Image

#List of all labels used for recognition
CATEGORIES = ["Flute", "Guitar", "Saxophone", "Trumpet", "Tuba", "Violin"]

#Initializes the array of official data
training_data = []

#Accessing My Google Drive
from google.colab import drive
drive.mount("drive")
base_image_path = "drive/My Drive/musical data/"

#Creates a list of all files from the specified folder
def loadImages(path):
    image_files = sorted([os.path.join(path, file)
                          for file in os.listdir(path)])
    return image_files

#Creates a dataset from the linked google drive based on labels in CATEGORIES
def create_training_data():
    for category in CATEGORIES:
        class_num = CATEGORIES.index(category)
        image_path = base_image_path + CATEGORIES[class_num]
        dataset = loadImages(image_path)
        #Subsets of total data can be generated by limiting the iterations of 'dataset'
        for img in dataset[30:200]:
            try:
                pil_im = Image.open(img)                        #opens image
                gray_im = pil_im.convert('LA')                  #converts to grayscale
                std_size = (360, 1440)                          #sets a standard size for images
                gray_im = gray_im.resize(std_size)              #applies standard size
                img_arr = np.asarray(gray_im)                   #converts image to numerical array
                img_arr = np.reshape(img_arr, [360,1440,2])
                img_arr = np.true_divide(img_arr, 255.0)        #converts values to [0-1] scale for better model evaluation
```

Pre-Processing of Input Data

<u>Requirements:</u>

- 360x1440
- Gray scale (small)
- Values [0-1]
- Random

```
            training_data.append([img_arr[:,:,[0]], class_num])   #appends to official dataset with shape(360,1440,1) and adds label
        except Exception as e:      #produces exception if the image cannot be read
          print('Fail: ' + e)
          pass

#builds the official dataset and shuffles it for model integrity
create_training_data()
random.shuffle(training_data)

#outputs data to the user to verify incoming information
print(len(training_data))
for sample in training_data[:10]:
    print(sample[1])

#creates final lists of features and labels from dataset
X = []
y = []
for features, label in training_data:
    X.append(features)
    y.append(label)

#changes directory for easy colab storage access of datasets
%cd '/content/'

#Saves all data to corresponding pickle files
pickle_out = open("X_train.pickle","wb") #<--easy pivotting between generating test and train
#pickle_out = open("X_test.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y_train.pickle","wb")
#pickle_out = open("y_test.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()

pickle_in = open("X_train.pickle", "rb")
#pickle_in = open("X_test.pickle", "rb")
X = pickle.load(pickle_in)

print('Done')
```

Input size	Description
1 × 43 × 128	mel-spectrogram
32 × 45 × 130	3 × 3 convolution, 32 filters
32 × 47 × 132	3 × 3 convolution, 32 filters
32 × 15 × 44	3 × 3 max-pooling
32 × 15 × 44	dropout (0.25)
64 × 17 × 46	3 × 3 convolution, 64 filters
64 × 19 × 48	3 × 3 convolution, 64 filters
64 × 6 × 16	3 × 3 max-pooling
64 × 6 × 16	dropout (0.25)
128 × 8 × 18	3 × 3 convolution, 128 filters
128 × 10 × 20	3 × 3 convolution, 128 filters
128 × 3 × 6	3 × 3 max-pooling
128 × 3 × 6	dropout (0.25)
256 × 5 × 8	3 × 3 convolution, 256 filters
256 × 7 × 10	3 × 3 convolution, 256 filters
256 × 1 × 1	global max-pooling
1024	flattened and fully connected
1024	dropout (0.50)
11	sigmoid

MFCC -> PSD
43x128 -> 360x1440

Layers

- 1 x 360 x 1440 (Input)
- 32 x 356 x 1436 5 x 5 s = 1 (Conv)
- Math: (360-5)/1 + 1 = 356
- 32 x 352 x 1432 5 x 5 s = 1 (Conv)
- 32 x 176 x 716 2 x 2 s = 2 (Pool)
- Math: (352-2)/2 + 1 = 176
- 32 x 176 x 716 Dropout (0.25)
- 64 x 172 x 712 5 x 5 s = 1 (Conv)
- 64 x 168 x 708 5 x 5 s = 1 (Conv)
- 64 x 84 x 354 2 x 2 s = 2 (Pool)
- 64 x 84 x 354 Dropout (0.25)
- Continued: 84 -> 38 -> 15

Total Number of layers: 1 + 4 x 4 + 1 + 1 = 19

Layers

Layers:

- Input (1)
- Hidden (14)
- Output(1)

Filter size: 3x3
Pooling stride: 2
Dropout: 0.25

```
#Imports the necessary libraries
import tensorflow as tf
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten,\
    Conv2D, MaxPooling2D
from keras.utils import to_categorical
import pickle
import joblib

#Loads in testing and training data
X_train = pickle.load(open("X_train.pickle","rb"))
y_train = pickle.load(open("y_train.pickle","rb"))
X_test = pickle.load(open("X_test.pickle","rb"))
y_test = pickle.load(open("y_test.pickle","rb"))

#converts the data to model-friendly formats
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train_bin = to_categorical(y_train)
y_test_bin = to_categorical(y_test)

model = Sequential()  #initializes the model
#runs first convolution with 32 filters, 3x3 kernal, and (360,1440,1) input from our converted images
model.add(Conv2D(32, (3,3), input_shape =(360,1440,1)))
model.add(Activation("relu")) #normalizes the data
model.add(Conv2D(32, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2))) #pools data to reduce input size and boost feature relevance

#repeats the layers with added filters
model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())  #squashes data to prepare for labeling
model.add(Dense(64))  #condenses to prepare for softmax
model.add(Activation("relu")) #normalization
```

Layers

Layers:

- Input (1)
- Hidden (14)
- Output(1)

- Softmax output
- Why joblib?
- Test vs Train data

```
model.add(Flatten())  #squashes data to prepare for labeling
model.add(Dense(64))  #condenses to prepare for softmax
model.add(Activation("relu")) #normalization

model.add(Dropout(0.25))  #runs dropout to reduce overfitting

model.add(Dense(6, activation='softmax')) #condenses to the number of lables(3) and selects the highest

#uses categorical crossentropy to prepare for multiple value single lable processing
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])

#trains the model
history = model.fit(X_train, y_train_bin, epochs=4, batch_size=32, validation_split=0.1)

#saves the model
filename = 'finalized_model.sav'
joblib.dump(model, filename)

#tests the model and prints results
test_results = model.evaluate(X_test, y_test_bin, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]*100}%')
```

Performance

- The model performance could be improved with more sample data. ~200 -> ~500 -> ~1000
- Synthetic data can be generating by making distorted versions of existing data
- Speed can be improved by lowering the dimension of the input array or cutting out unnecessary operations
- Additional Epochs have not yet seen a limit to their positive effect on model accuracy

Accuracy

3 Instruments Before

```
60
316
Train on 284 samples, validate on 32 samples
Epoch 1/2
284/284 [==============================] - 1090s 4s/step - loss: 2.8135 - accuracy: 0.3873 - val_loss: 1.1221 - val_accuracy: 0.2812
Epoch 2/2
284/284 [==============================] - 1096s 4s/step - loss: 1.0853 - accuracy: 0.4120 - val_loss: 1.1178 - val_accuracy: 0.3125
90/90 [==============================] - 79s 882ms/step
Test results - Loss: 1.1070311811235216 - Accuracy: 33.33333432674408%
```

3 Instruments After

```
416
416
Train on 374 samples, validate on 42 samples
Epoch 1/4
374/374 [==============================] - 473s 1s/step - loss: 15.1490 - accuracy: 0.4064 - val_loss: 0.9963 - val_accuracy: 0.4524
Epoch 2/4
374/374 [==============================] - 474s 1s/step - loss: 0.8376 - accuracy: 0.6096 - val_loss: 0.6496 - val_accuracy: 0.7619
Epoch 3/4
374/374 [==============================] - 480s 1s/step - loss: 0.4696 - accuracy: 0.8476 - val_loss: 0.4396 - val_accuracy: 0.7857
Epoch 4/4
374/374 [==============================] - 478s 1s/step - loss: 0.2669 - accuracy: 0.9171 - val_loss: 0.2933 - val_accuracy: 0.9048
416/416 [==============================] - 113s 271ms/step
Test results - Loss: 0.16162086163575834 - Accuracy: 96.63461446762085%
----------------------------------------------------------------------------
```

6 Instruments

```
Train on 833 samples, validate on 93 samples
Epoch 1/4
833/833 [==============================] - 1007s 1s/step - loss: 4.8751 - accuracy: 0.2257 - val_loss: 1.5762 - val_accuracy: 0.5269
Epoch 2/4
833/833 [==============================] - 996s 1s/step - loss: 0.9948 - accuracy: 0.6146 - val_loss: 0.5078 - val_accuracy: 0.7849
Epoch 3/4
833/833 [==============================] - 994s 1s/step - loss: 0.4314 - accuracy: 0.8667 - val_loss: 0.4644 - val_accuracy: 0.8387
Epoch 4/4
833/833 [==============================] - 989s 1s/step - loss: 0.2063 - accuracy: 0.9280 - val_loss: 0.1098 - val_accuracy: 0.9785
180/180 [==============================] - 49s 270ms/step
Test results - Loss: 0.2227531333764394 - Accuracy: 91.66666865348816%
```

Accuracy

Personal Requirements:

- >70% accuracy
- Multiple labels in use
- Integration with User Interface
- Data is precise within a 10% tolerance

Professional Requirements:

- >90% accuracy
- Polyphonic identification
- All-in-one integration
- 5% precision tolerance with justification

How well fit is the model?

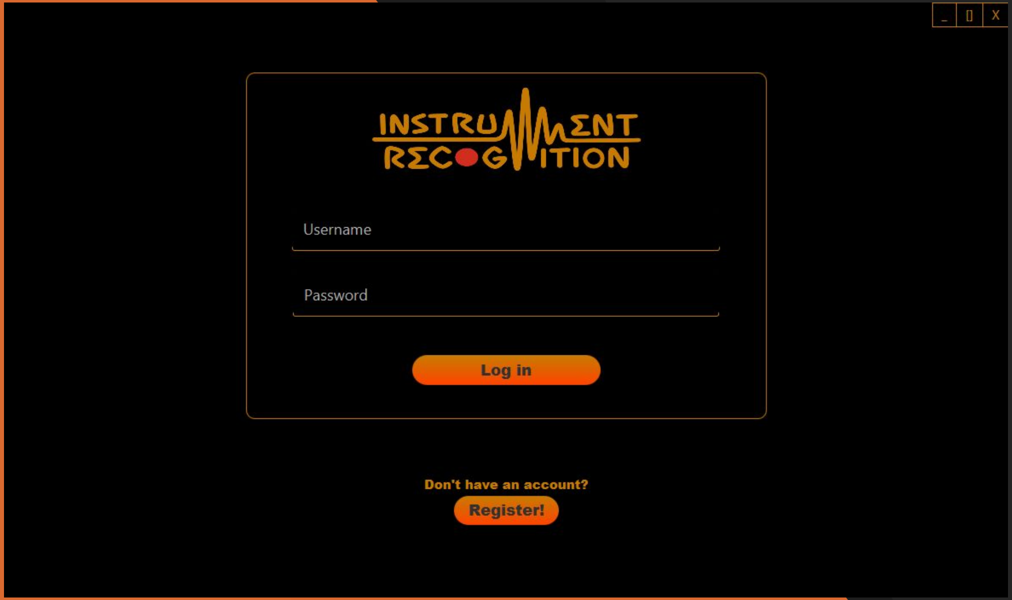
Our current model can alternate between over and under-fitting and it is up to the data scientist processing the results to determine the case. A higher accuracy than test accuracy can indicate over-fitting and vice-versa can show under-fitting.

Transfer Learning Application

Do you think transfer learning could be applied in this case?

Absolutely,
our application allows for the identification of audio clips based on their unique graphical representations. So long as the data is quantifiably unique, our product can recognize any kind of audio. Ex. include: Deep space audio signals, machine fault tolerance

Other Product Features

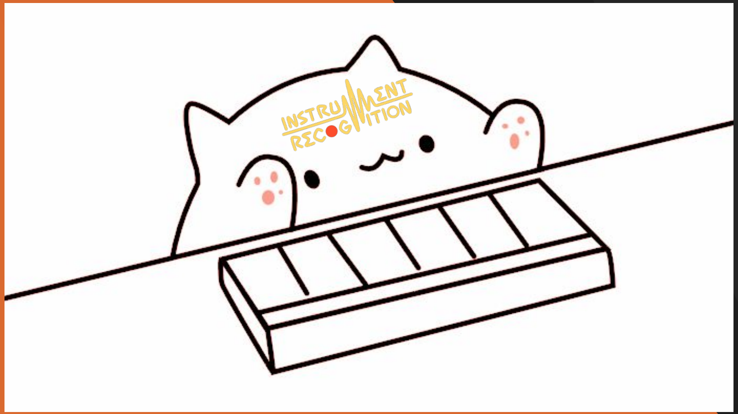


* Preview of the Login/Registration Page

- User Interface
  - Registration
  - Login
  - Database connectivity
  - Add files to the list
  - Delete files from the list
  - Play sound files
  - Get instrument classification
  - Show PSD graph

Demo!

Applications Outside of Music



- We have only started our journey
  - So far, we only have a relatively small number of instruments in our algorithm
  - We are focused on adding more instruments as our project grows

- The future
  - Currently, we have not discussed any plans of expanding into different fields of sound recognition
  - There are many different sound recognition algorithms out there

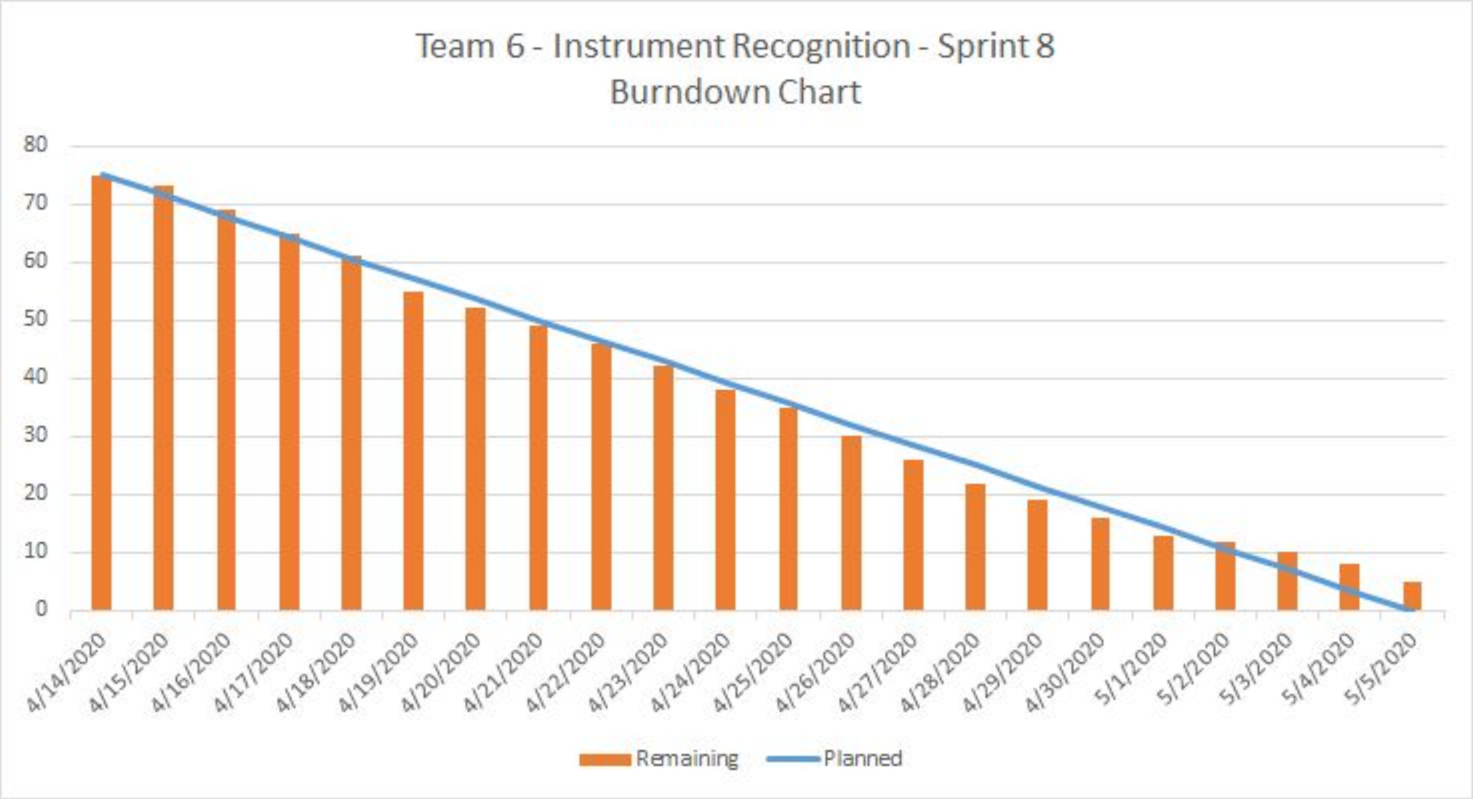
Sprint 8 Retrospective

- Sprint Goals
    - Train ML model
    - Work on getting custom data set(s)
    - Select new instruments for training
    - Add the chosen instruments for training
    - Host ML model on the Cloud
    - Complete the UI

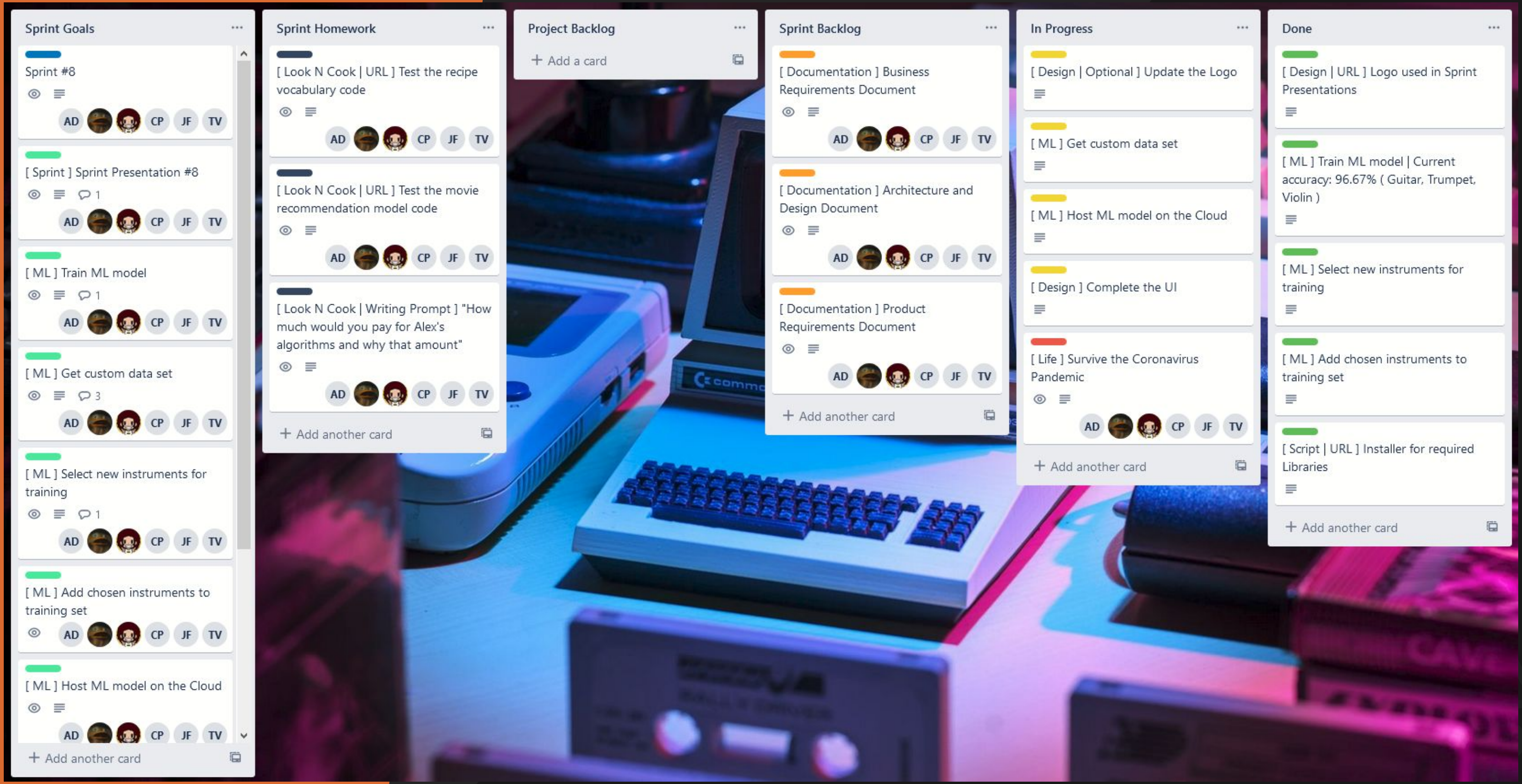
- Points
    - Planned: 75
    - Achieved: 70

- Unable to finish:
    - Host ML model on the cloud
        - Still researching ways to deploy the code, will get it to the class by next Wednesday latest.
    - Complete the UI

Burndown Chart



Sprint Board



INSTRUMENT RECOGNITION

Thank you for *listening!*