Majumdar, Aditya (am3713)
Mercer-Taylor, Andrew (ajm2209)
Ying, Robert (ry2242)

Identification of Printed Music

Computer Vision & Machine Learning for Mobile Platforms, Spring 2014

It is often difficult for people without a significant background in music to identify pieces of music from their printed scores. Since only the first page of a piece is labeled with the title and composer, it is often impractical for these people to try and search for musical metadata (i.e., title, composer, publication date, transcriber, etc.). This problem is especially prevalent among those who are trying to learn how to play a new piece or a new instrument. To compound the matter, the musical metadata is often insufficient in identifying the piece; for instance, Bach called more than 150 pieces “Prelude.”

The solution we envisioned was a mobile phone app that lets you take a picture of a sheet of music, and have the app return the name and composer of the piece. We felt that such an app would be very valuable for the above reasons, and would also be a cool way to apply some computer vision techniques. We chose to build an Android app because ⅔ of our group members have Android phones, and also because all three of us have prior experience with Android development. The idea is that our app will let users take a picture of whatever piece of sheet music they have lying around, and will segment the picture using the contours of the sheet music, extract interesting features, and then compare against features of various versions of classical music scores found in our database. Once we’ve matched up the user’s picture with the closest features in our database (using a heuristic), we find a relevant video recording of that piece being played on YouTube. The user can then play the recording immediately on their phone, or they can take another picture from within the app and begin the process anew.

In terms of building up our database, we downloaded PDFs of sheet music from IMSLP (International Music Score Library Project). We sometimes included several versions of the same piece, because sheet music often varies in size and formatting depending on the publishing company, so including feature sets from variations of the same piece gives us more tolerance when it comes to the recognition and matching stages. Our general process was to find features from each image on a measure by measure basis. The segmentation of individual measures was performed by identifying the contours in the image; we essentially used white space to find the points where we could separate the measures. From these processed images, we then extracted ORB binary descriptors of each measure, and stored this information in the database. As discussed later, ORB proved superior to SIFT due to speed and memory constraints.

Our initial plan was to recognize the music itself, transcribing note values and durations. However, the state of current research in this niche area of study is far from adequate to meet the challenge of reliably matching pieces. We experimented with Audiveris, an open source implementation of optical music recognition, and found that its performance was unreliable even on scans of music; with cell phone pictures, it was abysmal. The idea of transcribing the music was soon abandoned.

Our next approach proved much more promising. Rather than identifying the notes in the user image and searching for matching musical patterns in the transcribed database, we decided to create a method to directly compare the user image with database scans using computer vision techniques. An initial experiment showed the promise of SIFT descriptors. A set of one-measure snippets of pieces was manually created, and for each snippet, several skewed and distorted versions mimicking cell phone pictures were produced. SIFT descriptors were extracted for every scanned measure and every mock user image, and the Euclidean

distance between every image-scan pair was calculated. Descriptors of the mock user images were found to be closest to their original scans in every case.

We initially built our music recognition backend using SIFT and the FLANN nearest-neighbor search algorithm, which proved effective on small numbers of images. However, we found that the size of the SIFT descriptor caused the database to grow quickly with the addition of new images, to the point where even just a few thousand images exceeded the amount of memory available on the server. As a result, we decided to investigate other methods of identifying images.

After some experimentation, we found that the ORB binary descriptor was roughly as effective as SIFT in distinguishing between the various pieces, and came with the additional advantage of allowing us to calculate using Hamming distance instead of Euclidean distance. This both reduced the overall size of the database (to a few hundred megabytes) and the time it took to process a given image for matches against the search query.

To further reduce the processing time for each image search, we recognized that finding the correspondence score between the pieces in the database and the search image was trivially parallelizable. Thus, we split the search database into multiple processes, cutting the processing time by a factor of 8.

Since the search image is taken from a smartphone camera at a somewhat arbitrary angle and lighting condition, we found it necessary to process the image to get something that could be more easily compared to the database. To do this, we used an adaptive thresholding technique to help clean up the image and reduce the effect of the nonlinear light from the camera's flash, which also had the slight benefit of making the image processing itself faster. Additionally, the orientation invariance of the ORB descriptor allowed us to create a system

which was also largely orientation invariant, so that the orientation of the image taken by the user could be different from that of the scanned image.

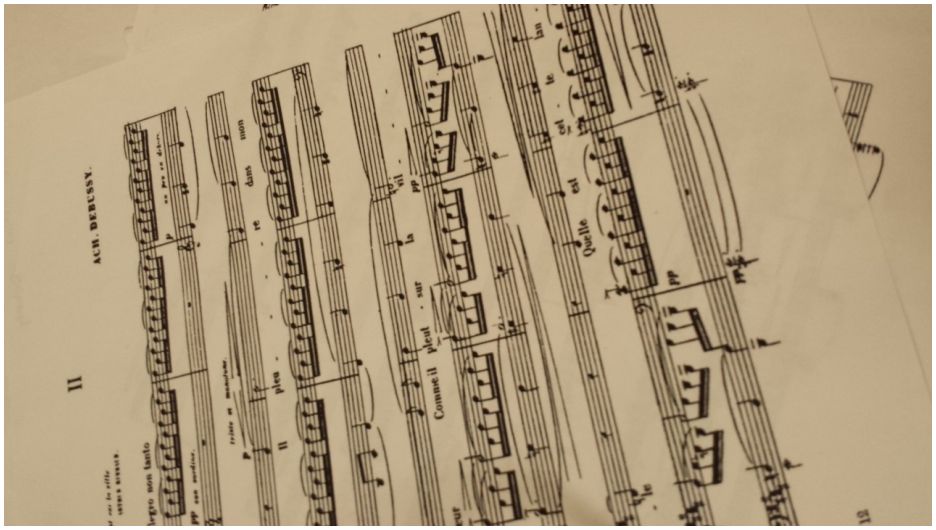
Because this app requires a fair amount of processing power and time to get through the workflow, most of the intensive work is done server-side. On the Android side, we decided to keep things simple and straightforward. The interface of the app itself simply takes advantage of Android's camera functionality. A single "capture" button, when pressed, takes a photo, uploads it to the server, and displays a load spinner prompting the user to wait for a response. When the server responds, it displays the title and composer of the piece identified, as well as a clickable link to a rendition on Youtube, if one can be found. Though we considered showing multiple results, the recognition proved so reliable that we decided to provide just the most likely piece. Ideally we would've started playing the Youtube recording of the piece immediately, but because some of these pieces are rather unique and not available on Youtube, we thought it would be more elegant to just return a link to the most relevant video result (if any exist), and let the user open it from within the app itself (only if they want to hear the music of course). On the Android side, most of the time was spent debugging various issues related to the consistency of flash and autofocusing - until we sorted out these bugs, we kept seeing inconsistent results, which was very frustrating, but that actually helped us fine tune aspects of the image processing. Otherwise, much of the time was just spent on fixing permission and networking issues within the Android framework. In the end, the Android experience became pretty smooth.

We take pride in what we've managed to accomplish with this app; it manages to recognize music more reliably than we had expected at the outset of the semester. That said, scaling the app to incorporate a wider database will require changes in the matching procedure. Currently, every piece in the database gets compared against the input, and even though we have optimized this procedure through parallelization and the use of the efficient ORB descriptor,

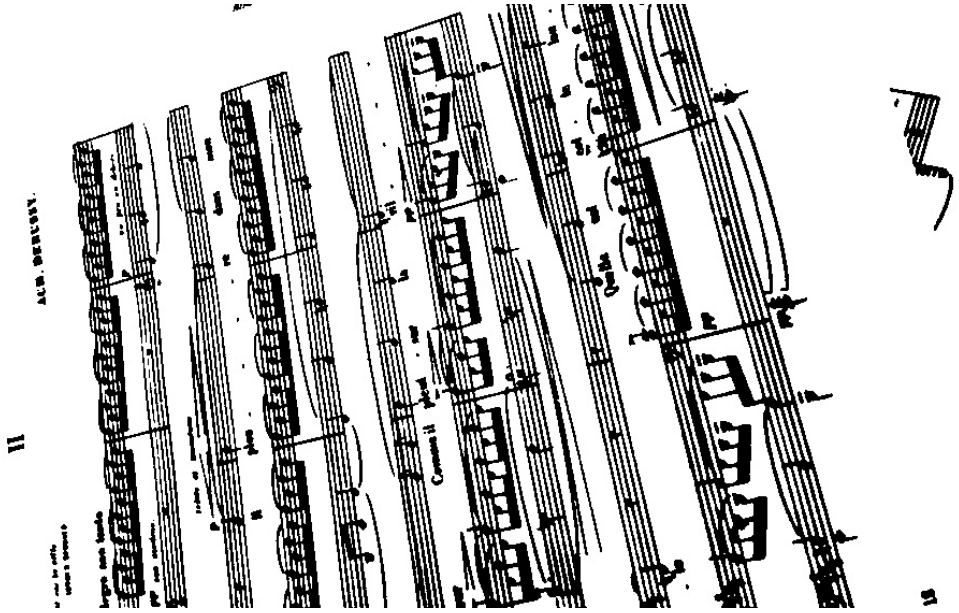
the fact remains that this O(n) approach limits the number of pieces we can incorporate while maintaining reasonably fast response times. A possible solution would be to use PCA to identify clusters of pieces, and to search only the clusters closest to a particular piece in the PCA projection. Another solution considered is using template matching to identify the key signatures of pieces, greatly reducing the search space. However, current optical music recognition software cannot reliably read the key signature, so this method would require some further research. Furthermore, although we are very confident in its capabilities, we have not attempted to quantify the accuracy of our app. This would require a sample of real user images of scores annotated with their actual metadata. We felt that creating such a database would not be worth the effort, or money.

Example:

This is the original image captured through our app:



The following is an intermediary image in our processing pipeline. It's a picture of the sheet music before it is segmented, but after the noise from the phone camera was reduced/removed.



After processing, we see JSON that looks like this:


```
{
 "maxval": 29,
 "link": "https://www.youtube.com/watch?v=sCgiCaG_dC4",
 "data": [
  [
   "Ballade (Debussy Claude)",
   -1
  ],
  [
   "Baryton Trio in D major Hob.XI45 (Haydn Joseph)",
   1
  ],
  [
   "Andante in G minor BWV 969 (Bach Johann Sebastian)",
```

```
    2
   ],
   [
    "Baryton Trio in D major Hob.XI34 (Haydn Joseph)",
    2
   ],
….….
   "Allemande in G minor BWV 837 (Bach Johann Sebastian)",
    20
   ],
   [
    "Allemande in G minor BWV 836 (Bach Johann Sebastian)",
    20
   ],
   [
    "Ariettes oublies (Debussy Claude)",
    29
   ]
  ],
  "title": "Ariettes oublies (Debussy Claude)"
}
```


We just parse the JSON to return the appropriate results to the user.

Group Contributions:

Robert took the lead in implementing ORB and the backend structure, worked on processing the captured images, and was able to parallelize much of the processing so the code would run faster. Andrew worked on the frontend and structure of the Android app itself, and performed initial validation of the idea using SIFT, in addition to resolving various network issues from the Android side. Aditya worked with both Robert and Andrew to connect the frontend and backend, integrated the YouTube API to return the link to the most relevant recording of the current piece, and worked on different aspects of the backend code to optimize the actual number of descriptor matches. All of us contributed when it came to idea formation, debugging, testing with multiple source images, acquiring music, and just researching and discussing solutions in general. The report and slide deck were also created equally.