Indian Institute of Information Technology, Vadodara
(Gandhinagar Campus)
Summer Research Internship - 2022

<u>Extraction of melodies with pitch classes From images of HCM compositions in Bhatkhande notation</u>

under the mentorship of
Dr.Pratik Shah

Submitted by
Manjot Singh (201951090)

Abstract - Conversion of Music Notes from images of Bhatkhande Notation System into MIDI file(audio file) involving Images Processing using OpenCV, Deep convolutional neural network modeling using Tensorflow and Keras and extracting those notes to be converted into a playable audio file using music21 toolkit for computer-aided musicology. The source code and other involved files for this project can be found here.

I. Introduction

The Indian Classical Music holds a very high place in our Indian culture, to the point that it is synonymous to Indian festivals, religious customs, marriages, exotic cuisine and Traditional Clothing. Doesn't matter who you are or what part of India you belong to, the minute you hear Indian music, the nostalgia and the amusement kicks in. Even if you are not from India, it would still create a sense of elegance and simplistic pleasures.

But despite its legendary status in the music culture, it seems that it is not being practiced heavily by the masses. Part of the reason is that it is less accessible and not being able to be visualized from reading about it in books and online articles. We are living in the age of social media and Short Clips platforms which tend to cause low attention span. "TikTok use disorder (TTUD) is positively linked to memory loss, and it is also positively linked to depression, anxiety, and stress. Depression, anxiety, and stress are positively linked to memory loss. Furthermore, depression, anxiety, and stress have a mediating effect between TTUD and memory loss." [1]

This project tries to bridge that gap between written medium and being able to hear and picture the beauty of the melodies. To be able to just convert your favorite melodies instantly from unexciting texts in your old books to audio files in your phone or laptop almost seems like a dream come true for a music enthusiast.

II. Problem Statement

Now that we understand the importance of the project, we need to understand and break down what are the necessary steps we need to take to solve this problem.

Task: From images of HCM compositions in Bhatkhande notation, extract the melodies with pitch classes.

So, initially this image needs to be converted into a supported audio file.

III. Proposed Solution

A lot of approaches had to be considered for this project. The final approach ended up being diving this task into 3 parts:

- NOTATION RECOGNITION AND PROCESSING
- NOTATION CLASSIFICATION
- AUDIO EXTRACTION

Or in simpler terms, this is divided as:

- What to read in this image?
- What is the meaning of the part that I just read in the image?
- What sound to produce in response to what I just read.

PART A - NOTATION RECOGNITION AND PROCESSING

For this, OpenCV is used to convert the notation image to a data array of the image. First comes Preprocessing of the image in which all the extra noise on the image is removed, like the color of the paper, different shades of color etc, so that it would not interfere with the detection[3]. After this we end up with a binary version of the image where only two colors exist, Black (0,0,0) and White(255,255,255).

After that, contours(outline along the boundary) of the image are detected, extracted and drawn onto another blank image.

Using these contours, Bounding Rectangles are drawn on the co-ordinates of these contours and saved on another blank image.

After getting all the bounding rectangles of all the contours out of the image, selective filtering of the contours is done for classification and thus removing the unnecessary clutter like vertical lines[2], heading, numberings etc.
We get x, y, h, w from contours as properties. For filtering, 4 conditions were set:

- **h / w < 3**, i.e. *Height / Width should be less than 3* (This removes all the tall lines which have a lot more height than width).
- **w / h < 2**, i,e. *Width / Height should be less than 2* (This removes all the horizontal lines and heading which have a lot more width than height).
- **h > 20,** i.e. *h should be greater than 20px* (This removes smaller elements such as numbers or dots)
- **w> 18,** i.e. *w should be greater that 18px* (This removes marks and numberings)

And Voila, we ended up with the desired elements needed for the classification. These elements are cut out of the original binary images in order for the classification.[5]

After this, we need to pass these bounded cutout elements in an array for further classification in the Deep CNN Model.

PART B - NOTATION CLASSIFICATION

A lot of approaches had to be considered for this part, since there was no dataset available for the hindi music notation, Devnagri Hindi Text Classification ended up being used and only selective elements, namely (सा रे गा मा प ध नि) were ended up getting passed to the model. These images are resized to 32px X 32px for uniformity, converted to binary as well and normalized.[4]

Data Augmentation is also used to add random flips, random rotation and random zoom to counter overfitting of the model. in addition to the normal Deep CNN model with max pooling, batch normalization, and adding multiple dense layers.

In [42]:
```
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 32, 32, 3)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 13, 13, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 5, 64)          18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 5, 5, 64)          256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 64)          36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 1, 1, 64)          256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 128)               8320
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_5 (Batch (None, 64)                256
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 455
=================================================================
Total params: 84,135
Trainable params: 83,367
Non-trainable params: 768
_________________________________________________________________
None
```

In [43]:
```
model.evaluate(X_test_scaled, y_test)
```

```
66/66 [==============================] - 0s 3ms/step - loss: 0.0273
```

Out[43]:
```
[0.02734367363154888, 0.9904761910438538]
```

PART C - NOTATION CLASSIFICATION

The image array from PART A is used here to classify the acquired images from value 0 to 6 and then using a dictionary convert to Sa, Re, Ga, Ma, Pa, Dha, Ni and Saa. This array is then converted to stream using music21 library and converted to MIDI file and used as audio file.

IV. EXPERIMENTS

- **Trackbars for customized Width, Height and Aspect Ratio selection.**

  Since this project was mainly carried out in Jupyter Notebooks, I ended up hard coding the values required for this particular image since Jupyter doesn't allow interactive windows. But it was brought to my attention that these height and width values may not work on different images

with smaller or larger text. So a commercial version of this project works, say for React or Android, i tried for an interface to which includes a trackbar inorder for the correct texts to be recognized.

- **<u>Iterative approach based on Pixel value threshold</u>**

I ended up with an Contour detected based approach, but previously I was set on an Iterative based approach/ Sliding Window approach. I manually handpicked points on top left of the text necessarily and counted the number of pixels used to show that piece of text , i.e सा took 342 pixels on average, रे took 275 pixels on average, गा 313 pixels on average, मा took 361 on average, प took 256 and ध took 310. In short, in the 1600 X 1200 image, I planned to pick a 45 X 45 window on the top left and move it right and bottom and pass only those pixels that satisfy constraints based on these conditions. I was later made aware that my approach was fine but the implementation of it was somewhat naive. But by then, the contours based approach worked so I went with a contour based approach. This Pixels data with sliding window approach is still here.

- **<u>Improving Datasets</u>**

Since the dataset I received was somewhat different that the same I used the model on. E.g. not being able to read the works properly even after preprocessing.

Sa getting classified as Maa.

I fiddled around with the data set but the end solution is that the model needs to be trained using more of the text from the original sample to match.

- **<u>Asking for too many output variables</u>:**

There are a lot more variations of music notes in the bhatkhande notation system. When I added those into the mix I had to synthesize them using openCV, more precisely using open CV, horizontal lines and dots were drawn above and below these very same notations. The problem ended up being that the model accuracy plummeted in the test cases since it created a lot of confusion for the model to detect what notes are written in the sample image.

- **<u>Testing on other notation images from the web</u>**

Here only one problem arises, the constraint of hard coded values. For the searching purposes of the notes, a kernel size needs to be specified or in simpler words, consider the previous example of iterative a based approach in which a window size is decided upon to search in the whole document and fortunately this window size does not stay the same and changes based on document to document since this base example '***part1.jpeg***' is a high resolution 1600 X 1200 image whereas other examples are usually 360 X 360 or 480 X 480, resulting in different height and width thresholds.

For filtering purposes too, certain conditions were applied as stated above, for example only the bounding rectangle of a certain size would be considered for prediction using the model but at least the size of the window needs to be calculated using trial and error method.

- **<u>Trying to remove the meed node (a variation when another note is on top of a node adding that same variation)</u>**

To remove this kind of note a lot of approaches had to be considered, few of them are:

- For the first approach it was observed that meed note are somewhat lesser in frequency, that's why all the coordinates of the notes were sorted by the y axis and a difference array

was created to detect the horizontal frequency of the notes. unfortunately this did not plan out since frequency does not have anything to do with this variation. So in the cases that all notes have meed variation, the whole need row would be included and in the cases that the main notes are lesser in frequency would also be discarded.

- Second approach was based on the principle of search by iteration, here for any note the area just above the note was passed into a function to search for any coordinates. Unfortunately this would only work if we know that the meed coordinate is directly above our current note but in actuality the meed can be found anywhere above the current note without any specific syntax for coordination.

  For a 2D based searching,searching would become a Tedious, resource expensive and time consuming task resulting in this approach to be discarded .

- Third was calculating the euclidean distance between the current note and the notes all above. This was the most bound to work approach and it did for the most part but in this case for any other test case it ends up getting stuck on the same problem of hard coded values. In other words, the minimum distance between two notes needs to be declared such that it is classified as a meed node..

For all the earlier work, absolute properties of the notes were being modified and manipulated, like drawing rectangles around one note, allowing that rectangle to be included based on its physical properties, which is somewhat easier. The problem arises when relative positionings need to be processed in order for them to work, which is somewhat tricky and needs more experience to be handled properly.

- **<u>Discarding the Handwritten Devanagari Dataset</u>**

  The model previously had a lot of issues with accuracy, mainly because the properties of handwritten and printed text are different and the possibility of misclassification increases exponentially. So to counter this issue, a custom dataset had to be generated by screenshotting and cropping text on written medium such as books and online images. Even though this process was tedious, it ended up making the model accuracy from 45.3239% to (current) 84.2105%, almost twice the original. This custom image dataset can be found here.

- **<u>Testing on Random Notation from the Internet</u>**

  The model needs to be sufficient enough to be used on any notation image that contains the notes. So to check that eligibility, 4 test cases were ran the results were promising.
  - First classifications are made using the model.
  - Then those values are checked one by one using an incremental iterator and the value is stored in an array/list in order to be cross checked against the predictions.
  - After that, heat map and midi files are generated through these test cases resources and data of which can be found here.

<u>TEST 1:</u>

Problem Statement:

Results:

Accuracy: **78.84%**

TEST 2:

Problem Statement:

Results:

Accuracy: **78.94%**

TEST 3:

Problem Statement:

Results:

Accuracy: **69.3333%**

TEST 4:

Problem Statement:

V. Conclusion

Only one two parameters were changed to achieve these results in the test (Search Window and Kernel Size) and once changed these parameters were kept the same for all the test cases(all 4). Accuracy could have been improved a lot, the new dataset really did wonders for this model, only hits taken by the accuracy were by the accidental detection of numbers as notes. If it were only the notes and not the number, the accuracy could have been above 90%. So, more tuning needs to be done in order of the hard coded values to be automatically computed instead of relying on the user input.

This project really made me aware of a lot more concepts in Machine Learning, Deep Learning and Image Manipulation. The best part about it is how different parts of this project do different things and it all comes together in the end ever so seamlessly.

References

[1] Alpaydin, E. (2021). *Machine Learning, revised and updated edition*. MIT Press.

[2] *Canny edge detector*. OpenCV. (n.d.). Retrieved July 25, 2022, from https://docs.opencv.org/4.x/da/d5c/tutorial_canny_detector.html

[3] Connectedreams.com. (2016, December 1). *Beginner's Guide to Computer Vision*. Medium. Retrieved July 25, 2022, from https://medium.com/readers-writers-digest/beginners-guide-to-computer-vision-23606224b720

[4] Kang, N. (2019, February 4). *Introducing deep learning and neural networks - deep learning for rookies (1)*. Medium. Retrieved July 25, 2022, from https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883

[5] Lee, S. (2022, January 30). *Lines detection with Hough Transform*. Medium. Retrieved July 25, 2022, from https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549

[6] Sha, P., & Dong, X. (2021, August 21). *Research on adolescents regarding the indirect effect of depression, anxiety, and stress between TikTok use disorder and memory loss*. International journal of environmental research and public health. Retrieved July 25, 2022, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8393543/

[7] Springer. (2014). Image Processing. In *Computer vision*.