A PROJECT REPORT ON G.I.S.A.

(GESTURE INTERPRETER FOR SPECIALLY ABLED)

MEMBERS

ARJUN PARMAR

BHAVNA MATWANI

MANAV KHORASIYA

RIYA SINGHAL

YAMINI KABRA

Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to ***"DRISHTI-A Revolutionary Concept"*** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my mentors Ms. Deeksha Goyal and Ms. Shubhi Agarwal for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

Abstract

It is observed that specially abled people are unable to communicate properly due to the lack of knowledge about their sign language amongst common people. This system aims at encountering this problem and proposing a feasible solution. The aim is to build a human responsive system that can aid specially abled (blind, deaf and aphonic) people by interpreting the sign language (i.e. hand gestures). Sign language of such people is mostly composed of hand gestures. Gesture recognition is a technique that uses the concepts of computer vision which is used for the interaction between human and computer or it can be for any automation implementation purpose. Further it is proposed to convert these gestures to text and speech so that it can be easily understood by others.

Contents

1. Content Page
2. Digit Recognitions
3. Motion in Space
4. Alphabet Recognition
    i. Sign Language Recognition
    ii. Alphabet Tracing with Motion in Space
5. Text to Speech

Digit Recognition

For digit recognition, first part is to segment the raw image. So, in first part we do the Segmentation of image. Our aim to extract skin pixels from raw image which is done by Segmentation.

To segment the image, we have lots of noisy elements in background. To overcome this problem, we do segmentation in two way:1) Segmentation according to fix value of HSV, YCrCb and RGB colour space of skin pixel 2) Segmentation according to variable HSV value of skin pixel

Skin Segmentation

We used average HSV values for skin to extract the skin area from the frame. This did not prove to be accurate since the HSV values are affected by intensity of light. To incorporate the effect of lighting conditions, the conditions of YCrCb, HSV and RGB models were added. Due to this, the skin area could be extracted more accurately.

What is a colour space?
When colours need to be used in digital media like cameras and laptops, colours need to be presented in numbers because the digital media can only comprehend numbers. Therefor colour space is a set of rules that allows describing colours with numbers.

What is HSV colour space?
Hue Saturation Value
HUE
Hue is the colour portion of the model, expressed as a number from 0 to 360 degrees.

SATURATION

Saturation describes the amount of Grey in a particular colour, from 0 to 100 percent. Reducing this component toward zero introduces Greyer and produces a faded effect. Sometimes, saturation appears as a range from just 0-1, where 0 is grey, and 1 is a primary colour.

VALUE (OR BRIGHTNESS)

Value works in conjunction with saturation and describes the brightness or intensity of the colour, from 0-100 percent, where 0 is completely black, and 100 is the brightest and reveals the most colour.

Uses of HSV

Designers use the HSV colour model when selecting colours for paint or ink because HSV better represents how people relate to colours than the RGB colour model does.

What is RGB colour space?

Red Green Blue

RGB colour model is the additive colour model using Red, green and blue colours. The main use of the RGB colour model is for displaying images in electronic devices. In this process of the RGB colour model if the three colours are superimposed with the least intensity then the black colour is formed and if it is added with the full intensity of light, then the white colour is formed. To make a different array of colours these primary colours should be superimposed in different intensities. According to some studies, the intensity of each primary colours can vary from 0 to 255 and which results in the creation of almost 16,777,216 colours.

What is YCrCb colour model?

Y is the luma component of the colour. Luma component is the brightness of the colour. That means the light intensity of the colour. The human eye is more sensitive to this component.

Cb and Cr is the blue component and red component related to the chroma component. That means "Cb is the blue component relative to the green component. Cr is the red component relative to the green component." These components are less sensitive to the human eyes.

Since the Y component is more sensitive to the human eye, it needs to be more correct and Cb and Cr are less sensitive to the human eye. Therefore, it needs not to be more accurate. When in JPEG compression, it uses these sensitivities of the human eye and eliminate the unnecessary details of the image.

Now we know all colour spaces so according to our approaches let see both algorithms:

1) Segmentation according to fix value of HSV, YCrCb and RGB colour space of skin pixel

In this method we find the range of HSV, YCbCr, RGB value for human skin pixel which are given below:

We got two type of ranges from research paper which are given below:

(0<=h) and (h<=50) and (0.23<s) and (s<0.68) and (r>95) and (g>40) and (b>20) and (r>g) and (r>b) and (r-g)>15

OR

(r>95) and (g>40) and (b>20) and (r>g) and (r>b) and (r-g)>15 and (cr>135) and (cb>85) and (y>80) and (cr<=(1.5862*cb+20)) and (cr>=(0.3448*cb+76.2069)) and (cr>=(-4.5652*cb+234.5652)) and (cr<=-1.15*cb+301.75) and (cr<=-2.2857*cb+432.85)

From these two ranges we proposed below algorithm:

We take input image from Camera feed and compare it with above range and got output image with pixels which satisfy above one of two condition. Then, we skeletonized it with distance transform and feed into our recognizer model and got predicted output.

The output of this algorithm is Figure 5 which contain the pixel position which satisfy above condition.

In this algorithm, the term distance transform is also included so firstly we have to know about it, Let see about Distance transform.

Distance Transform is morphology for Skeletonization.

Distance Transform

The distance transform is an operator normally only applied to binary images. The result of the transform is a grey level image that looks similar to the input image, except that the grey level intensities of points inside foreground regions are changed to show the distance to the closest boundary from each point.

One way to think about the distance transform is to first imagine that foreground regions in the input binary image are made of some uniform slow burning inflammable material. Then consider simultaneously starting a fire at all points on the boundary of a foreground region and letting the fire burn its way into the interior. If we then label each point in the interior with the amount of time that the fire took to first reach that point, then we have effectively computed the distance transform of that region. Figure 1 shows a distance transform for a simple rectangular shape.

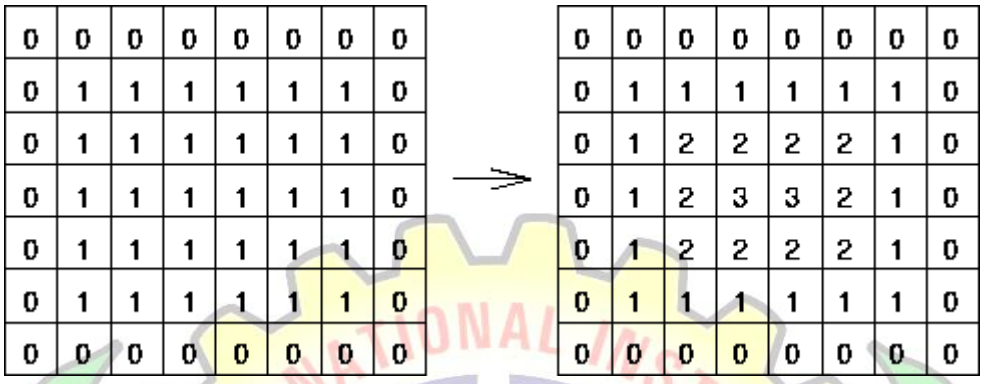


Figure 1 *The distance transform of a simple shape. Note that we are using the `chessboard' distance matric*

How It Works

There are several different sorts of distance transform, depending upon which distance matric is being used to determine the distance between pixels. The example shown in Figure 1 uses the `chessboard' distance metric but both the Euclidean and `city block' metrics can be used as well.

Even once the metric has been chosen, there are many ways of computing the distance transform of a binary image. One intuitive but extremely inefficient way of doing it is to perform multiple successive erosions with a suitable structuring element until all foreground regions of the image have been eroded away. If each pixel is labeled with the number of erosions that had to be performed before it disappeared, then this is just the distance transform. The actual structuring element that should be used depends upon which distance metric has been chosen. A 3×3 square element gives the `chessboard' distance transform, a cross shaped element gives the `city block' distance transform, and a disk shaped element gives the Euclidean distance transform. Ofcourse it is not actually possible to generate a good disk shaped element on a discrete grid on a small scale, but there are algorithms that vary the structuring element on each erosion so as to approximate a circular element.

Example

Figure 2 *Source image*

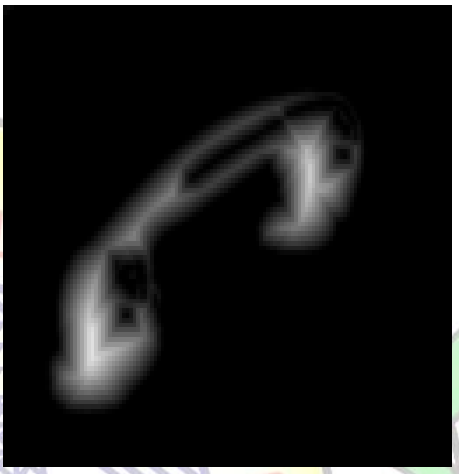


Figure 3 *Distance Transform of Source Image*

Note: The source image for distance transform must be grayscale image. If image is RGB , then convert it into grayscale. Now we are familiar with all terms in algorithm. Let's see the output of this algorithm:

Figure 4 *Source image*

Figure 5 *Location of Skin pixels*

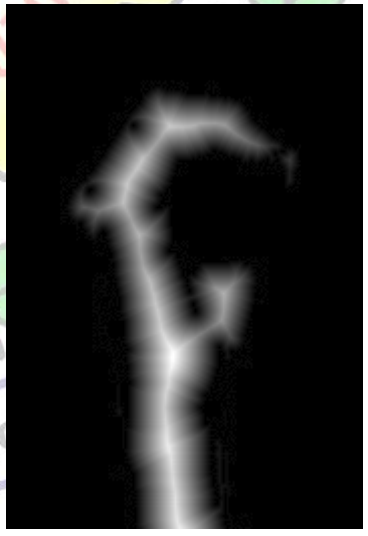


Figure 6 *Distance Transform of Skin Pixels*

Further we propose one more algorithm which can extract the skin pixel according to only HSV value with trackbar which is given below

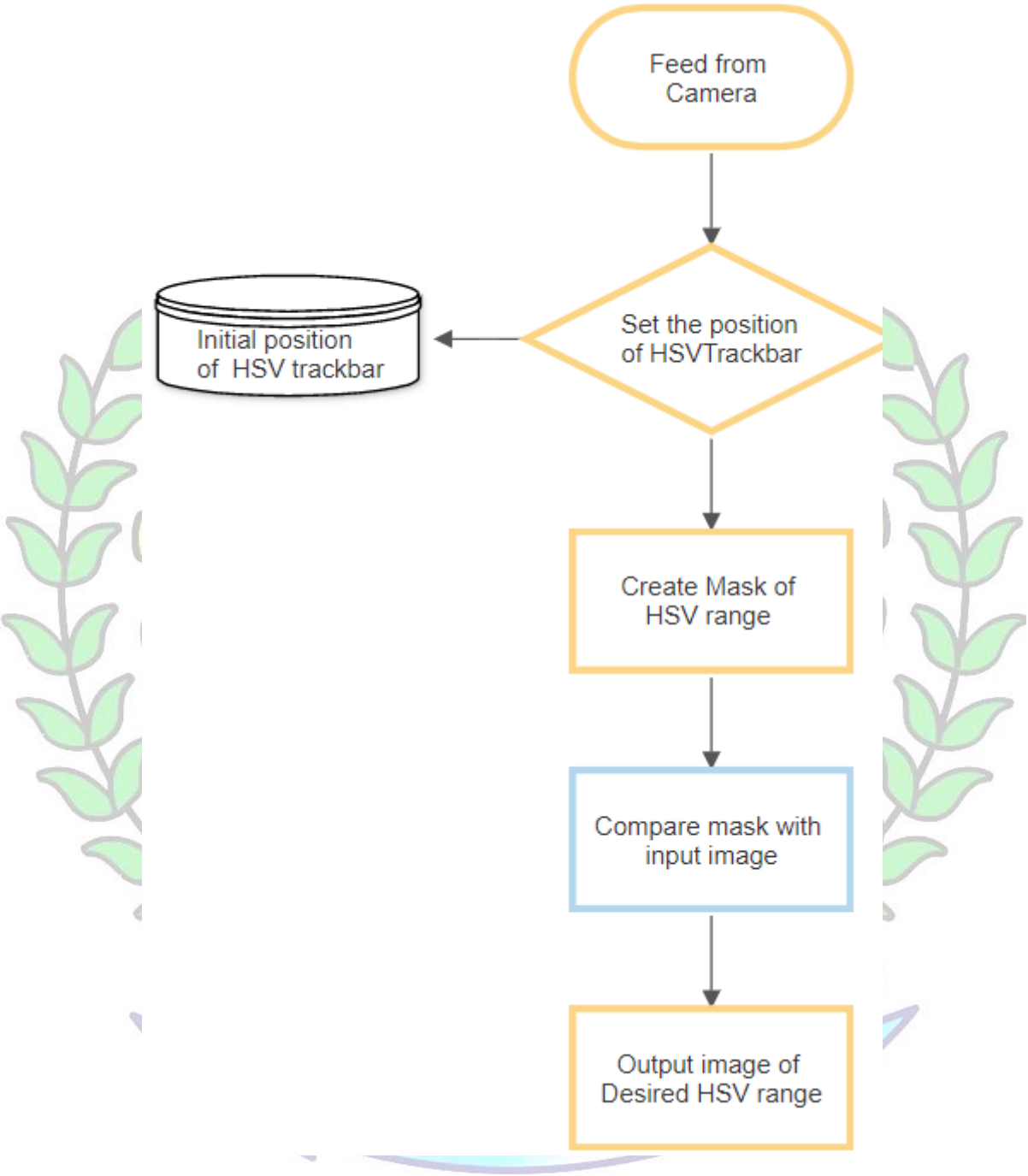


Figure 7 *Flow diagram of HSV Trackbar*

Let's see the output of this algorithm:

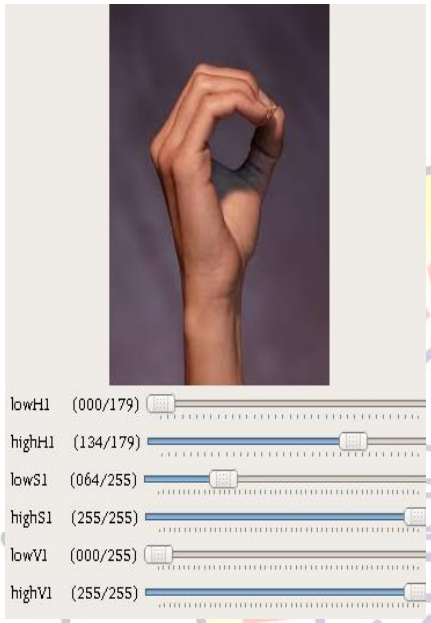


Figure 8(a) *Source image with HSV Trackbar*

Figure 8(b) *Skin Detection using HSV Trackbar*

Figure 9 *Distance Transform of fig.8*

A)Initially we tried to implement google's hand tracking algorithm, but it was not successful as we required the largest contour, but only the contours we were able to extract are the fingers individually. Therefore this idea was soon discarded. The link for that is provided below:
https://m.9gag.com/gag/ae53W3O?utm_campaign=link_post&utm_medium=social&utm_source=Instagram_Story

B)Next we tried to find the center of the hand using skeletonization implemented through distance transform and it was successfully achieved.
But we were not able to proceed further with this method .

C)We shifted to deep learning and tried various architectures like le-net5, resnet100,vgg19. Vgg16 , resnet 150,mobilenet,alexnet with continuously changing their parameters, but all these efforts resulted in very poor accuracy.

In above three parts, dataset is not segmented so we can't get good accuracy.

D)Then we modified the dataset and used skeletonized one and trained on resnet50 which resulted into 98.04% accuracy.

The Working Flow Diagram is Given below:

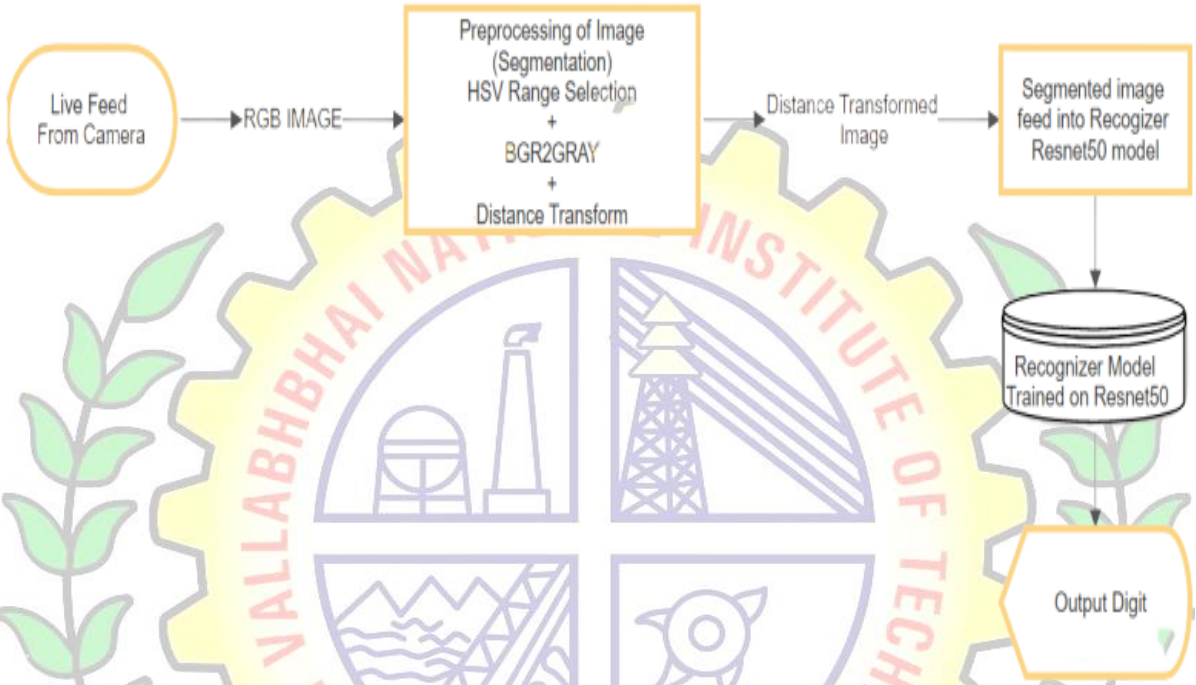


Figure 10 *Flow chart for Digit Recognition*

Motion in Space

Behind this term, Motion in Space idea is to track the path of object in space.
We track the object path by tracking the centroid of image.so, we have to see how we find the centroid of image. We done it by moments of image.

Image moment

In image processing, computer vision and related fields, an image moment is a certain particular weighted average (moment) of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property or interpretation.

Image moments are useful to describe objects after segmentation. Simple properties of the image which are found via image moments include area (or total intensity), its centroid, and information about its orientation.

Raw Moments

For a 2D continuous function *f(x,y)* the moment (sometimes called "raw moment") of order *(p+q)* is defined as

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y)\, dx\, dy$$

for *p*,*q* = 0,1,2,... Adapting this to scalar (greyscale) image with pixel intensities *I*(*x*,*y*), raw image moments M_{ij} are calculated by

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

In some cases, this may be calculated by considering the image as a probability density function, *i.e.*, by dividing the above by

$$\sum_x \sum_y I(x,y)$$

A uniqueness theorem (Hu [1962]) states that if *f*(*x*,*y*) is piecewise continuous and has nonzero values only in a finite part of the *xy* plane, moments of all orders exist, and the moment sequence (M_{pq}) is uniquely determined by *f*(*x*,*y*). Conversely, (M_{pq}) uniquely determines *f*(*x*,*y*). In practice, the image is summarized with functions of a few lower order moments.

Examples

Simple image properties derived *via* raw moments include:

Area (for binary images) or sum of grey level (for greytone images):M_{00}

Centroid:

$$\{\bar{x},\ \bar{y}\} = \left\{\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right\}$$

For finding center of image we do the same segmentation as used in digit recognition. Now we have distance transformed image on which we find moments. By formula given above we find the centre which is given below:

Figure 11 *Source image with it's centre*

Figure 12 *Distance Transform of ROI*

Hence, We successfully find the centre of given ROI(Region of Interest) image.
Now we defined one algorithm which can track the path of this centre.

For this some pre steps are taken:
We defined one blank image with fix camera frame size.
The raw images are segmented as above.

Step1) Live feed from camera.
Step2) Segmentation of frame and finding centre of it.
Loop: Timer start:
Step3) Plot centre of it on blank image.
Step4) Repeat loop and plot centre on same blank image till threshold count reached.
Step5) At threshold count, timer will stop and initialized at 0.
Step6) Blank image is modified into path of centre travelling.
Step7) Save modified image and Start again.
The output of proposed algorithm is given below:

Figure 13 *Output of centre plotting on blank image*

DETECTING MOTIONS IN SPACE

Following the principles of object path tracking we traced the center and considering the distance and the direction of travelling of center through the first and last frames center coordinate differences we find whether it was indicating right,left ,up or down direction. Then using the contours property we calculated the area and its difference given whether it was front or back movement.

So, We proposed this algorithm:
Let (cX1,cY1) be the position of center in last frame and (cX0,cY0) be the position of center in last frame

```
x=abs(cX1-cX0)
y=abs(cY1-cY0)
if(x>y):
        if(cX1>cX0):
                move="Left"
        else:
                move="Right"
if(x<y):
        if(cY1>cY0):
                move="Down"
        else:
                move="Up"
```

Hence, by basic distance formula we can conclude the direction of movement.

Example:
(cX1,cY1)=(1,5)
(cX0,cY0)=(100,7)
So, x=abs(cX1-cX0)=99
y=abs(cY1-cY0)=2
here, x>y:
and cX1<cX0
so answer is : **Right** Direction

Further we add on two more things that is depth In or Out.
We find the contour Area of first and last image and then compare it and find the depth direction like in or out.
Let suppose first frame contour area ia A0 and last frame contour Area is A1.
If A1>A0 ; then it is outward direction
else it is inward direction

Example:
A1=500 sq.unit
A0=100 sq.unit
Here, A1>A0 so direction is **outward**

Alphabet Recognition

Sign Language Recognition

This method is as same as used in digit recognition but we got accuracy on only 09 classes which are A,G,L,O,Q,V,W,Y and Z.
The dataset is also trained on resnet50 architecture.
We got 98.93% test accuracy which is Quite good on practical result also.

The description of accuracy is given below:

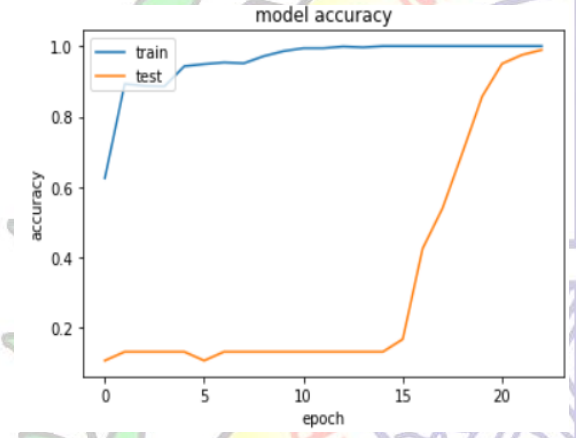


Figure 14 *Model Accuracy*

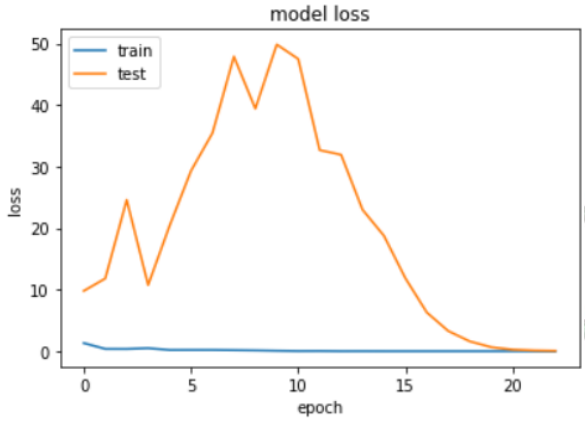


Figure 15 *Model Loss*

Alphabet Tracing with Motion in Space

As we seen the motion in space we used it in the Alphabet Recognition. We trace the movement of hand and feed the final traced plotted path in our model recognizer.
It is somewhat same as Handwritten Digit Recognition also.

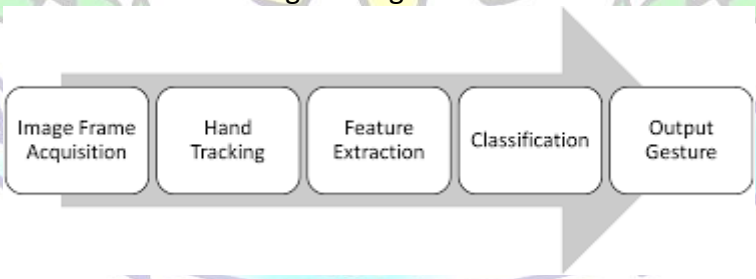


Figure 16 *Flow Diagram*

This time the model architecture is changed. We used simple architecture:

(Conv2D+maxPool) * 3

+

Fully Connected Layer

We got accuracy for only 10 classes which are B,C,L,N,O,P,S,U,V and Z. The practical accuracy for 10 clasees are quite good as nearly 90%.

Text to Speech

A text-to-speech synthesis system typically consists of multiple stages, such as a text analysis frontend, an acoustic model and an audio synthesis module. Building these components often requires extensive domain expertise and may contain brittle design choices. In this paper, we present Tacotron, an end-to-end generative text-to-speech model that synthesizes speech directly from characters. Given pairs, the model can be trained completely from scratch with random initialization. We present several key techniques to make the sequence-tosequence framework perform well for this challenging task. Tacotron achieves a 3.82 subjective 5-scale mean opinion score on US English, outperforming a production parametric system in terms of naturalness. In addition, since Tacotron generates speech at the frame level, it's substantially faster than sample-level autoregressive methods.

Introduction

Modern text-to-speech (TTS) pipelines are complex (Taylor, 2009). For example, it is common for statistical parametric TTS to have a text frontend extracting various linguistic features, a duration model, an acoustic feature prediction model and a complex signal-processing-based vocoder (Zen et al., 2009; Agiomyrgiannakis, 2015). These components are based on extensive domain expertise and are laborious to design. They are also trained independently, so errors from each component may compound. The complexity of modern TTS designs thus leads to substantial engineering efforts when building a new system.

There are thus many advantages of an integrated end-to-end TTS system that can be trained on pairs with minimal human annotation. First, such a system alleviates the need for laborious feature engineering, which may involve heuristics and brittle design choices. Second, it more easily allows for rich conditioning on various attributes, such as speaker or language, or high-level features like sentiment. This is because conditioning can occur at the very beginning of the model rather than only on certain components. Similarly, adaptation to new data might also be easier. Finally, a single model is likely to be more robust than a multi-stage model where each component's errors can compound. These advantages imply that an end-to-end model could allow us to train on huge amounts of rich, expressive yet often noisy data found in the real world.

TTS is a large-scale inverse problem: a highly compressed source (text) is "decompressed" into audio. Since the same text can correspond to different pronunciations or speaking styles, this is a particularly difficult learning task for an end-to-end model: it must cope with large variations at the signal level for a given input. Moreover, unlike end-to-end speech recognition (Chan et al., 2016) or machine translation (Wu et al., 2016), TTS outputs are continuous, and output sequences are usually much longer than those of the input. These

attributes cause prediction errors to accumulate quickly. In this paper, we propose Tacotron, an end-to-end generative TTS model based on the sequence-to-sequence (seq2seq) (Sutskever et al., 2014) with attention paradigm (Bahdanau et al., 2014). Our model takes characters as input and outputs raw spectrogram, using several techniques to improve the capability of a vanilla seq2seq model. Given pairs, Tacotron can be trained completely from scratch with random initialization. It does not require phoneme-level alignment, so it can easily scale to using large amounts of acoustic data with transcripts. With a simple waveform synthesis technique, Tacotron produces a 3.82 mean opinion score (MOS) on an US English eval set, outperforming a production parametric system in terms of naturalness

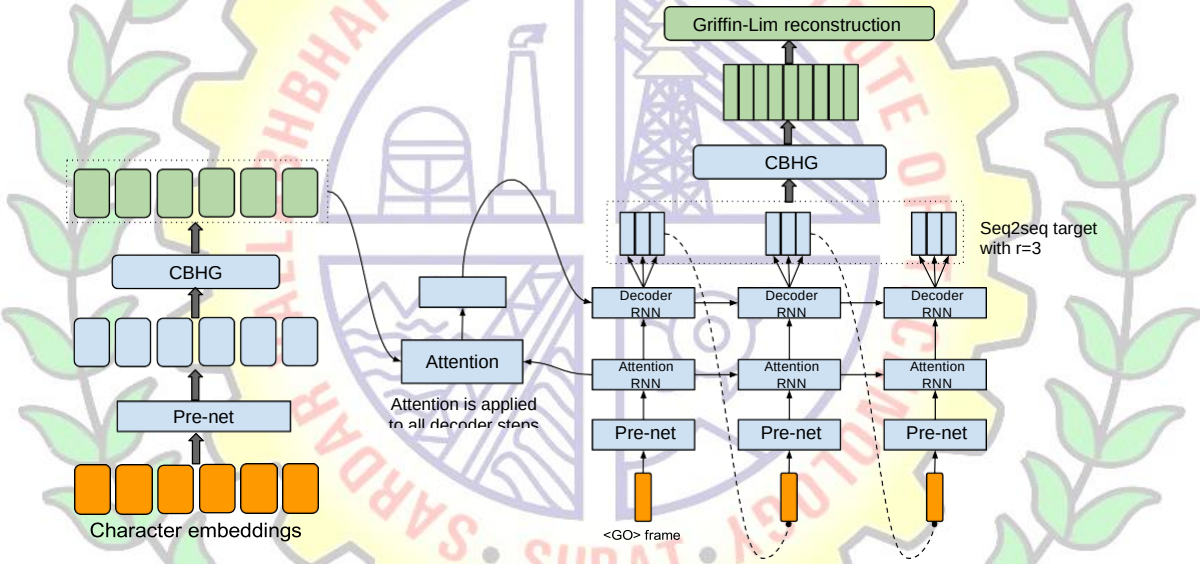


Figure 17 *Model architecture. The model takes characters as input and outputs the corresponding raw spectrogram, which is then fed to the Griffin-Lim reconstruction algorithm to synthesize speech*

Related Work

WaveNet (van den Oord et al., 2016) is a powerful generative model of audio. It works well for TTS, but is slow due to its sample-level autoregressive nature. It also requires conditioning on linguistic features from an existing TTS frontend, and thus is not end-to-end: it only replaces the vocoder and acoustic model. Another recently-developed neural model is DeepVoice (Arik et al., 2017), which replaces every component in a typical TTS pipeline by a corresponding neural network. However, each component is independently trained, and it's nontrivial to change the system to train in an end-to-end fashion.

To our knowledge, Wang et al. (2016) is the earliest work touching end-to-end TTS using seq2seq with attention. However, it requires a pre-trained hidden Markov model (HMM)

aligner to help the seq2seq model learn the alignment. It's hard to tell how much alignment is learned by the seq2seq per se. Second, a few tricks are used to get the model trained, which the authors note hurts prosody. Third, it predicts vocoder parameters hence needs a vocoder. Furthermore, the model is trained on phoneme inputs and the experimental results seem to be somewhat limited.

Char2Wav (Sotelo et al., 2017) is an independently-developed end-to-end model that can be trained on characters. However, Char2Wav still predicts vocoder parameters before using a SampleRNN neural vocoder (Mehri et al., 2016), whereas Tacotron directly predicts raw spectrogram. Also, their seq2seq and SampleRNN models need to be separately pre-trained, but our model can be trained from scratch. Finally, we made several key modifications to the vanilla seq2seq paradigm. As shown later, a vanilla seq2seq model does not work well for character-level inputs.

Model Architecture

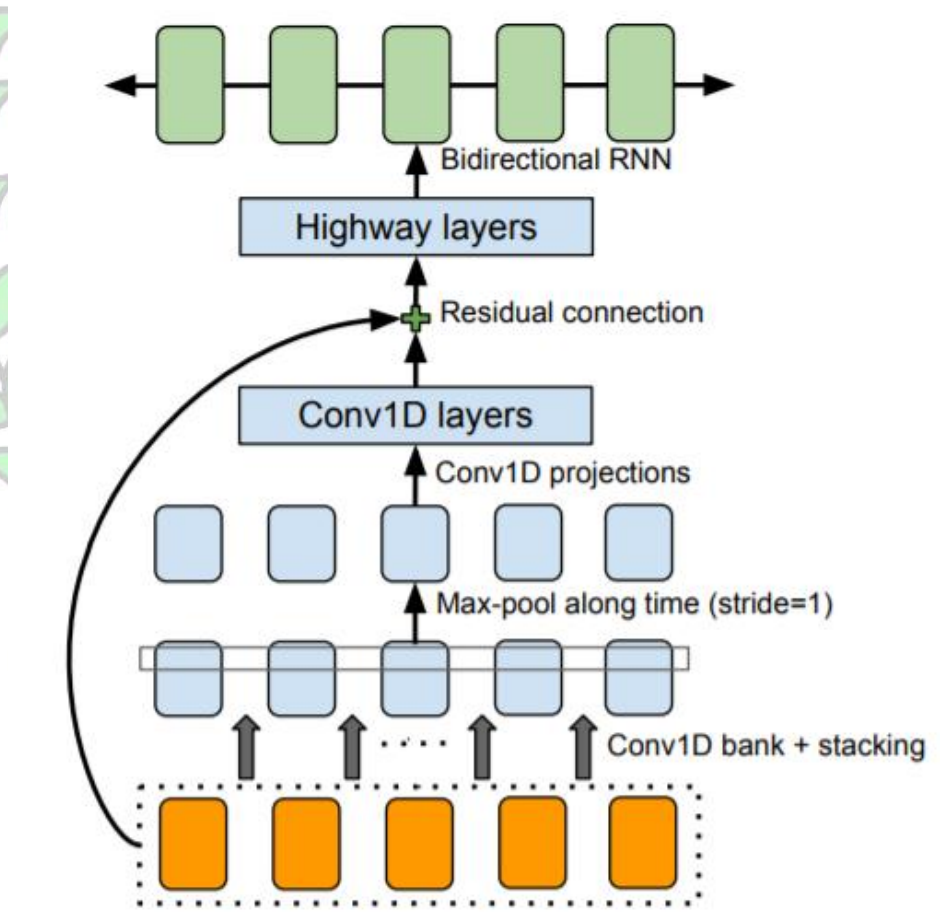


Figure 18 *The CBHG (1-D convolution bank + highway network + bidirectional GRU) module adapted from Lee et al. (2016)*

The backbone of Tacotron is a seq2seq model with attention (Bahdanau et al., 2014; Vinyals et al., 2015). Figure 1 depicts the model, which includes an encoder, an attention-based decoder, and a post-processing net. At a high-level, our model takes characters as input and produces spectrogram frames, which are then converted to waveforms. We describe these components above.

CBHG module

We first describe a building block dubbed CBHG, illustrated in Figure 2. CBHG consists of a bank of 1-D convolutional filters, followed by highway networks (Srivastava et al., 2015) and a bidirectional gated recurrent unit (GRU) (Chung et al., 2014) recurrent neural net (RNN). CBHG is a powerful module for extracting representations from sequences. The input sequence is first convolved with K sets of 1-D convolutional filters, where the k-th set contains Ck filters of width k (i.e. k = 1, 2, . . . , K). These filters explicitly model local and contextual information (akin to modeling unigrams, bigrams, up to K-grams). The convolution outputs are stacked together and further max pooled along time to increase local invariances. Note that we use a stride of 1 to preserve the original time resolution. We further pass the processed sequence to a few fixed-width 1-D convolutions, whose outputs are added with the original input sequence via residual connections (He et al., 2016). Batch normalization (Ioffe & Szegedy, 2015) is used for all convolutional layers. The convolution outputs are fed into a multi-layer highway network to extract high-level features. Finally, we stack a bidirectional GRU RNN on top to extract sequential features from both forward and backward context. CBHG is inspired from work in machine translation (Lee et al., 2016), where the main differences from Lee et al. (2016) include using non-causal convolutions, batch normalization, residual connections, and stride=1 max pooling. We found that these modifications improved generalization.

Encoder

Spectral analysis	*pre-emphasis*: 0.97; *frame length*: 50 ms; *frame shift*: 12.5 ms; *window type*: Hann
Character embedding	256-D
Encoder CBHG	*Conv1D bank*: *K*=16, conv-*k*-128-ReLU *Max pooling*: stride=1, width=2 *Conv1D projections*: conv-3-128-ReLU → conv-3-128-Linear *Highway net*: 4 layers of FC-128-ReLU *Bidirectional GRU*: 128 cells
Encoder pre-net	FC-256-ReLU → Dropout(0.5) → FC-128-ReLU → Dropout(0.5)
Decoder pre-net	FC-256-ReLU → Dropout(0.5)→ FC-128-ReLU → Dropout(0.5)

Decoder RNN	2-layer residual GRU (256 cells)
Attention RNN	1-layer GRU (256 cells)
Post-processing net CBHG	*Conv1D bank*: K=8, conv-k-128-ReLU *Max pooling*: stride=1, width=2 *Conv1D projections*: conv-3-256-ReLU → conv-3-80-Linear *Highway net*: 4 layers of FC-128-ReLU *Bidirectional GRU*: 128 cells
Reduction factor (r)	2

Figure 19 *Hyper-parameters and network architectures. "conv-k-c-ReLU" denotes 1-D convolution with width k and c output channels with ReLU activation. FC stands for fully-connected.*

The goal of the encoder is to extract robust sequential representations of text. The input to the encoder is a character sequence, where each character is represented as a one-hot vector and embedded into a continuous vector.We then apply a set of non-linear transformations, collectively called a "pre-net", to each embedding. We use a bottleneck layer with dropout as the pre-net in this work, which helps convergence and improves generalization. A CBHG module transforms the prenet outputs into the final encoder representation used by the attention module. We found that this CBHG-based encoder not only reduces overfitting, but also makes fewer mispronunciations than a standard multi-layer RNN encoder.

Decoder

We use a content-based tanh attention decoder (see e.g. Vinyals et al. (2015)), where a stateful recurrent layer produces the attention query at each decoder time step. We concatenate the context vector and the attention RNN cell output to form the input to the decoder RNNs. We use a stack of GRUs with vertical residual connections (Wu et al., 2016) for the decoder. We found the residual connections speed up convergence. The decoder target is an important design choice. While we could directly predict raw spectrogram, it's a highly redundant representation for the purpose of learning alignment between speech signal and text (which is really the motivation of using seq2seq for this task). Because of this redundancy, we use a different target for seq2seq decoding and waveform synthesis. The seq2seq target can be highly compressed as long as it provides sufficient intelligibility and prosody information for an inversion process, which could be fixed or trained. We use 80-band mel-scale spectrogram as the target, though fewer bands or more concise targets such as cepstrum could be used. We use a post-processing network (discussed below) to convert from the seq2seq target to waveform.

We use a simple fully-connected output layer to predict the decoder targets. An important trick we discovered was predicting multiple, non-overlapping output frames at each decoder

step. Predicting r frames at once divides the total number of decoder steps by r, which reduces model size, training time, and inference time. More importantly, we found this trick to substantially increase convergence speed, as measured by a much faster (and more stable) alignment learned from attention. This is likely because neighboring speech frames are correlated and each character usually corresponds to multiple frames. Emitting one frame at a time forces the model to attend to the same input token for multiple timesteps; emitting multiple frames allows the attention to move forward early in training. A similar trick is also used in Zen et al. (2016) but mainly to speed up inference.

The first decoder step is conditioned on an all-zero frame, which represents a frame. In inference, at decoder step t, the last frame of the r predictions is fed as input to the decoder at step t + 1. Note that feeding the last prediction is an ad-hoc choice here – we could use all r predictions. During training, we always feed every r-th ground truth frame to the decoder. The input frame is passed to a pre-net as is done in the encoder. Since we do not use techniques such as scheduled sampling (Bengio et al., 2015) (we found it to hurt audio quality), the dropout in the pre-net is critical for the model to generalize, as it provides a noise source to resolve the multiple modalities in the output distribution.

POST-PROCESSING NET AND WAVEFORM SYNTHESIS

As mentioned above, the post-processing net's task is to convert the seq2seq target to a target that can be synthesized into waveforms. Since we use Griffin-Lim as the synthesizer, the post-processing net learns to predict spectral magnitude sampled on a linear-frequency scale. Another motivation of the post-processing net is that it can see the full decoded sequence. In contrast to seq2seq, which always runs from left to right, it has both forward and backward information to correct the prediction error for each individual frame. In this work, we use a CBHG module for the post-processing net, though a simpler architecture likely works as well. The concept of a post-processing network is highly general. It could be used to predict alternative targets such as vocoder parameters, or as a WaveNet-like neural vocoder (van den Oord et al., 2016; Mehri et al., 2016; Arik et al., 2017) that synthesizes waveform samples directly.

We use the Griffin-Lim algorithm (Griffin & Lim, 1984) to synthesize waveform from the predicted spectrogram. We found that raising the predicted magnitudes by a power of 1.2 before feeding to Griffin-Lim reduces artifacts, likely due to its harmonic enhancement effect. We observed that Griffin-Lim converges after 50 iterations (in fact, about 30 iterations seems to be enough), which is reasonably fast. We implemented Griffin-Lim in TensorFlow (Abadi et al., 2016) hence it's also part of the model. While Griffin-Lim is differentiable (it does not have trainable weights), we do not impose any loss on it in this work. We emphasize that our choice of Griffin-Lim is for simplicity; while it already yields

strong results, developing a fast and high-quality trainable spectrogram to waveform inverter is ongoing work.

Model Details

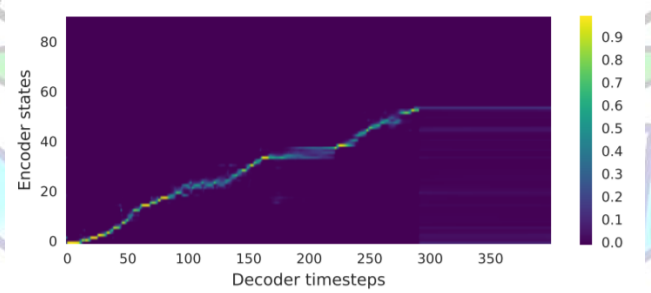
Table 1 lists the hyper-parameters and network architectures. We use log magnitude spectrogram with Hann windowing, 50 ms frame length, 12.5 ms frame shift, and 2048-point Fourier transform. We also found pre-emphasis (0.97) to be helpful. We use 24 kHz sampling rate for all experiments.

We use r = 2 (output layer reduction factor) for the MOS results in this paper, though larger r values (e.g. r = 5) also work well. We use the Adam optimizer (Kingma & Ba, 2015) with learning rate decay, which starts from 0.001 and is reduced to 0.0005, 0.0003, and 0.0001 after 500K, 1M and 2M global steps, respectively. We use a simple `1 loss for both seq2seq decoder (mel-scale spectrogram) and post-processing net (linear-scale spectrogram). The two losses have equal weights.

We train using a batch size of 32, where all sequences are padded to a max length. It's a common practice to train sequence models with a loss mask, which masks loss on zero-padded frames. However, we found that models trained this way don't know when to stop emitting outputs, causing repeated sounds towards the end. One simple trick to get around this problem is to also reconstruct the zero-padded frames.

Experiments

We train Tacotron on an internal North American English dataset, which contains about 24.6 hours of speech data spoken by a professional female speaker. The phrases are text normalized, e.g. "16" is converted to "sixteen".



(a) Vanilla seq2seq + scheduled sampling

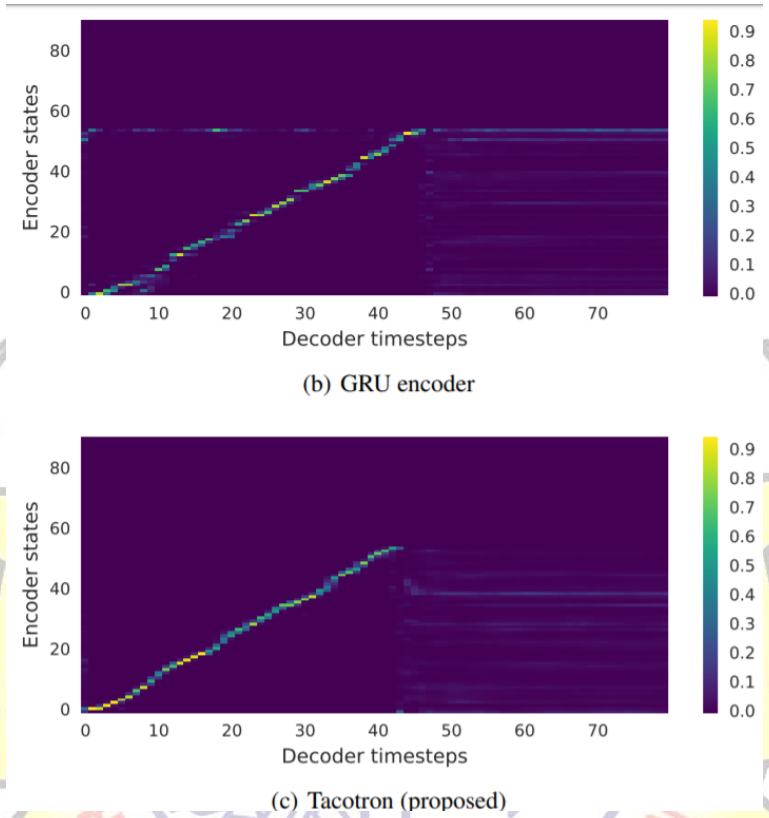


***Figure 20** : Attention alignments on a test phrase. The decoder length in Tacotron is shorter due to the use of the output reduction factor r=5.*

ABLATION ANALYSIS

We conduct a few ablation studies to understand the key components in our model. As is common for generative models, it's hard to compare models based on objective metrics, which often do not correlate well with perception (Theis et al., 2015). We mainly rely on visual comparisons instead. We strongly encourage readers to listen to the provided samples.

First, we compare with a vanilla seq2seq model. Both the encoder and decoder use 2 layers of residual RNNs, where each layer has 256 GRU cells (we tried LSTM and got similar results). No pre-net or post-processing net is used, and the decoder directly predicts linear-scale log magnitude spectrogram. We found that scheduled sampling (sampling rate 0.5) is required for this model to learn alignments and generalize. We show the learned attention

alignment in Figure 20. Figure 20(a) reveals that the vanilla seq2seq learns a poor alignment. One problem is that attention tends to get stuck for many frames before moving forward, which causes bad speech intelligibility in the synthesized signal. The naturalness and overall duration are destroyed as a result. In contrast, our model learns a clean and smooth alignment, as shown in Figure 20(c).

Second, we compare with a model with the CBHG encoder replaced by a 2-layer residual GRU encoder. The rest of the model, including the encoder pre-net, remain exactly the same. Comparing Figure 20(b) and 20(c), we can see that the alignment from the GRU encoder is noisier. Listening to synthesized signals, we found that noisy alignment often leads to mispronunciations. The CBHG encoder reduces overfitting and generalizes well to long and complex phrases.

Figures 21(a) and 21(b) demonstrate the benefit of using the post-processing net. We trained a model without the post-processing net while keeping all the other components untouched (except that the decoder RNN predicts linear-scale spectrogram). With more contextual information, the prediction from the post-processing net contains better resolved harmonics (e.g. higher harmonics between bins 100 and 400) and high frequency formant structure, which reduces synthesis artifacts.

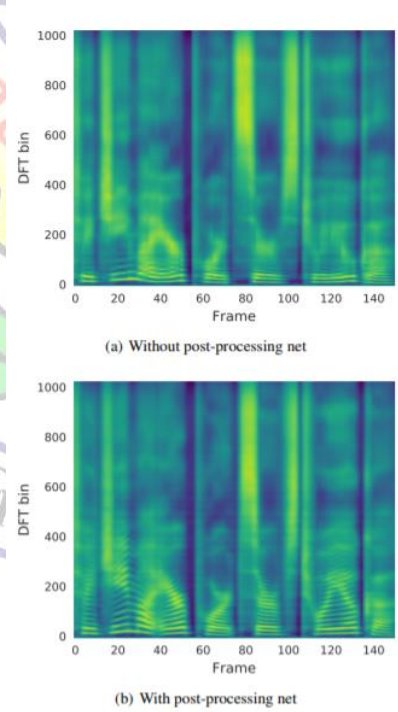


Figure 21 *Predicted spectrograms with and without using the post-processing net*

MEAN OPINION SCORE TESTS

We conduct mean opinion score tests, where the subjects were asked to rate the naturalness of the stimuli in a 5-point Likert scale score. The MOS tests were crowdsourced from native speakers. 100 unseen phrases were used for the tests and each phrase received 8 ratings. When computing MOS, we only include ratings where headphones were used. We compare our model with a parametric (based on LSTM (Zen et al., 2016)) and a concatenative system (Gonzalvo et al., 2016), both of which are in production. As shown in Table 2, Tacotron achieves an MOS of 3.82, which outperforms the parametric system. Given the strong baselines and the artifacts introduced by the Griffin-Lim synthesis, this represents a very promising result.

	mean opinion score
Tacotron	3.82 ± 0.085
Parametric	3.69 ± 0.109
Concatenative	4.09 ± 0.119

***Figure 22** : 5-scale mean opinion score evaluation*

Discussions

We have proposed Tacotron, an integrated end-to-end generative TTS model that takes a character sequence as input and outputs the corresponding spectrogram. With a very simple waveform synthesis module, it achieves a 3.82 MOS score on US English, outperforming a production parametric system in terms of naturalness. Tacotron is frame-based, so the inference is substantially faster than sample-level autoregressive methods. Unlike previous work, Tacotron does not need handengineered linguistic features or complex components such as an HMM aligner. It can be trained from scratch with random initialization. We perform simple text normalization, though recent advancements in learned text normalization (Sproat & Jaitly, 2016) may render this unnecessary in the future.

We have yet to investigate many aspects of our model; many early design decisions have gone unchanged. Our output layer, attention module, loss function, and Griffin-Lim-based waveform synthesizer are all ripe for improvement. For example, it's well known that Griffin-Lim outputs may have audible artifacts. We are currently working on fast and high-quality neural-network-based spectrogram inversion.

Conclusion

For TTS we used Pretrained model tacotron for speech synthesis.
We used LJspeech dataset for TTS. Firstly we preprocess the dataset like we want sequence to sequence modelling so we convert speech dataset into spectrogram sequence.then we mapped input to output. then we train tacotron on it.

Basically the tacotron is a mixture of GRU and MULTI RNN with an Adam optimizer and reduced mean loss.

After training the model we feed input in the model.now for speech synthesis the model output is spectrogram of its text we convert it to mp3 audio file with the help of text synthesization.

Hence we successfully synthesize speech from input text.

Reference: https://github.com/keithito/tacotron

Project GitHub Repository: ***https://github.com/shubhi0168/G.I.S.A.-Gesture-Interpreter-for-Specially-Abled-***

References

- https://github.com/keithito/tacotron
- https://m.9gag.com/gag/ae53W3O?utm_campaign=link_post&utm_medium=social&utm_source=Instagram_Story
- https://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm
- https://docs.opencv.org/2.4/index.html
- http://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm
- http://www.inf.u-szeged.hu/~palagyi/skel/skel.html
- https://www.tutorialspoint.com/python/time_time.htm
- https://www.geeksforgeeks.org/speech-recognition-in-python-using-google-speech-api/
- https://github.com/mozilla/TTS
- https://github.com/Kyubyong/tacotron
- https://www.coursera.org/learn/natural-language-processing-tensorflow/home/welcome
- https://colab.research.google.com/notebooks/intro.ipynb#recent=true
- https://github.com/stevenobadja/math_object_detection
- https://github.com/gary30404/neural-network-from-scratch-python/blob/master/network.py
- https://github.com/HYPJUDY/handwritten-digits-recognition
- https://github.com/cvqluu/TDNN/blob/master/tdnn.py
- https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470
- https://github.com/anujdutt9/Handwritten-Digit-Recognition-using-Deep-Learning
- https://morioh.com/p/cdb65d590bd5
- https://stackoverflow.com/questions/36102437/streaming-droidcam-video-to-opencv-python-in-anyway-possible
- https://github.com/jonasvdd/TDNN
- https://www.coursera.org/specializations/deep-learning
- http://keras.io/applications/
- https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
- https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624


SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY
SURAT
विज्ञानं सारथि नः स्यात ॥