Automated Music Synthesis for Album

November 30, 2017

Indian Institute Of Information Technology,Allahabad

Members :

IIT2014101 - Yerram Reddy Mahendra Reddy

IIT2014143 - Saketh Tadimeti

IIT2014153 - Sameer Sk

Mentor :

Dr.Satish Kumar Singh

Candidate's Declaration

We hereby declare that the work presented in this project entitled "**Automated Music Synthesis for Album**", submitted in the fulfillment of the completion of the semester VII of Bachelor of Technology (B.Tech) program, in Information Technology at Indian Institute of Information Technology, Allahabad, is an authentic record of our original work carried out under the guidance of **Dr.Satish Kumar Singh**.We have adequately cited and referenced the original sources and have adhered to all principles of academic honesty and integrity. Due acknowledgements have been made in the text of the project to all other material used.

Place:Allahabad	IIT2014101-Y Mahendra Reddy
Date: November 30,2017	IIT2014143-Saketh Tadimeti
	IIT2014153-Sameer Sk

Certificate from Supervisor

This is to certify that the statement made by the candidates is correct to the best of my knowledge and belief. The project titled "Automated Music Synthesis for Album" is a record of candidates' work carried out by them under my guidance and supervision. I do hereby recommend that it should be accepted in the fulfilment of the requirements of the 7th semester mini project at IIIT Allahabad.

Place:Allahabad	**Dr. Satish Kumar Singh**
Date: November 30,2017	Assistant Professor,IIIT-Allahabad

Committee for Evaluation of Project

Dr.U S Tiwary	Dr. Satish Kumar Singh	Dr. T.Pant

Table of Contents

ABSTRACT :

In this work, we transferred style of each image and Recurrent Neural Network (RNN) is used for generating automated music.These two processes will be merged to create a video with the auto-generated music played in the background.

INTRODUCTION :

In the past decade deep learning has taken-off . It is used in every application where the human intelligence is required i.e., computer vision and natural language processing and robotics. And of course it outperformed traditional methods in all applications.

Our application is a combination of computer vision and sequence learning. So we have also used deep learning techniques to shape our application.

MOTIVATION :

The use of deep learning networks has escalated in recent times.Our motivation is to understand and implement Recurrent Neural networks to solve the problem of sequence generation

PROBLEM STATEMENT :

We are given a album of photos (example, all of your google photos). Images are transferred into different styles.The aim is to create an automated system to make video snippets using the above transferred images while automatically generated music is played in the background.

LITERATURE SURVEY :

1.A Neural Algorithm of Artistic Style by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge : In this paper, authors present how to combine content of one image with style of other image to form a new styled content image. Drawback of this paper is that , it can not be deploy in real time because it take hour's on GPU's to mix content and style image.

2.Perceptual Losses for Real-Time Style Transfer and Super-Resolution by justin johnson,Alexandre Alahi,Fei-Fei Li : In this paper, authors present a practical (speed) algorithm to style transfer , if we fix the style image then we can transfer the style of input image on CPU in few milliseconds. Drawback of this paper is that noise in mixed image.

3.Instance Normalization: The Missing Ingredient for Fast Stylization by Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky : In this paper, authors improved the performance of justin johnson's paper. These people used instance normalization to improve gradients and it leads to improved mixed image over justin johnson's paper.

Music Generation :

1. **Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling - Authors(Hasim Sak, Andrew Senior, Francoise Beaufays) :** In this research paper, the authors explained how long short-term memory RNN can be modelled to be more powerful than feed forward Neural networks in case of sequential data like speech and music.
2. **GRUV: Algorithmic Music Generation using Recurrent Neural Networks - Authors (Aran Nayebi,Matt Vitelli) :** This paper gives us an insight on the comparative performance of LSTM based RNN , GRU based RNN

TECHNICAL APPROACH :

This section describe the methods we intend to apply to solve this problem.

We are approaching the solution with 3 phases.

Phase-1: Style Transfer

Content loss :

It is just euclidean distance between 2 vectors but without square root also called as $L2\ norm$ distance.

We can also use $L1\ norm\ distance$ i.e., absolute distance between 2 vectors.

But people usually prefer $L2\ norm$ ove $L1\ norm$ because it is easy prove lot's of theorem's with elegantly.

```
def content_loss(x,y) :
        # x : nXn array
        # y : nXn array
        tmp = 0
        for i in xrange(n) :
                for j in xrange(n) :
                        tmp += (x[i][j]-y[i][j])**2
         return tmp
```

Style loss :

This one is a bit of trick. First let's understand gram matrix.

gram matrix : it captures second order statistics about which features, in that feature map tend to activate together at different spatial volumes/positions. It is proportional to covariance matrix, since gram matrix calculation is computationally efficient than covariance matrix, people often use gram matrix instead of covariance matrix.

We get C x H x W image features after we pass image through CNN then we take outer product of (C-dim vector-1 , C-dim vector-2) pair to get CxC matrix and then we take all possible pair of C-dim vectors to get C x C matrix and average over all these to get graham matrix. We can efficiently calculate by

Let flatten this, C x H x W → C x HW and call it as F

Graham matrix = (F x F.T) where F.T is transpose of F.

```
def graham_matrix(x) :
    # x : numpy array of shape (C,H,W)
    x = x.reshape((C,HW)) # x : shape(C,HW)
    return np.dot(x,x.T) # shape(C,C)
```

Then style loss is *L2 norm distance* between graham matrices of style image and mixed style image at choosen layers of VGG network.

```
def style_loss(style_image, mixed_image) :
    # style_image : shape(C,H,W)
    # mixed image : shape(C,H,W)
    S = graham_matrix(style_image) # S : shape(C,C)
    M = graham_matrix(mixed_image) # M : shape(C,C)
    tmp = 0
    for i in xrange(C) :
        for j in xrange(C) :
            tmp += (S[i][j]-M[i][j])**2
    tmp = tmp/CONSTANT
    return tmp
```

Total Variation Loss :

If we consider only style loss and content loss into loss function , then it usually seems to be quite noisy . So to capture spatial smoothness people usually added regularization term into loss function.

```
    def total_variation_loss(x) :
        # x : image which numpy array with shape(h,w,p)
        tmp1 = x[:h-1,:w-1,:] - x[1,:w-1,:]
        tmp2 = x[:h-1,:w-1,:] - x[:h-1,1,:]
        tmp3 = np.pow(tmp1+tmp2,1.25)
        tmp = np.sum(tmp3)
        return tmp
```

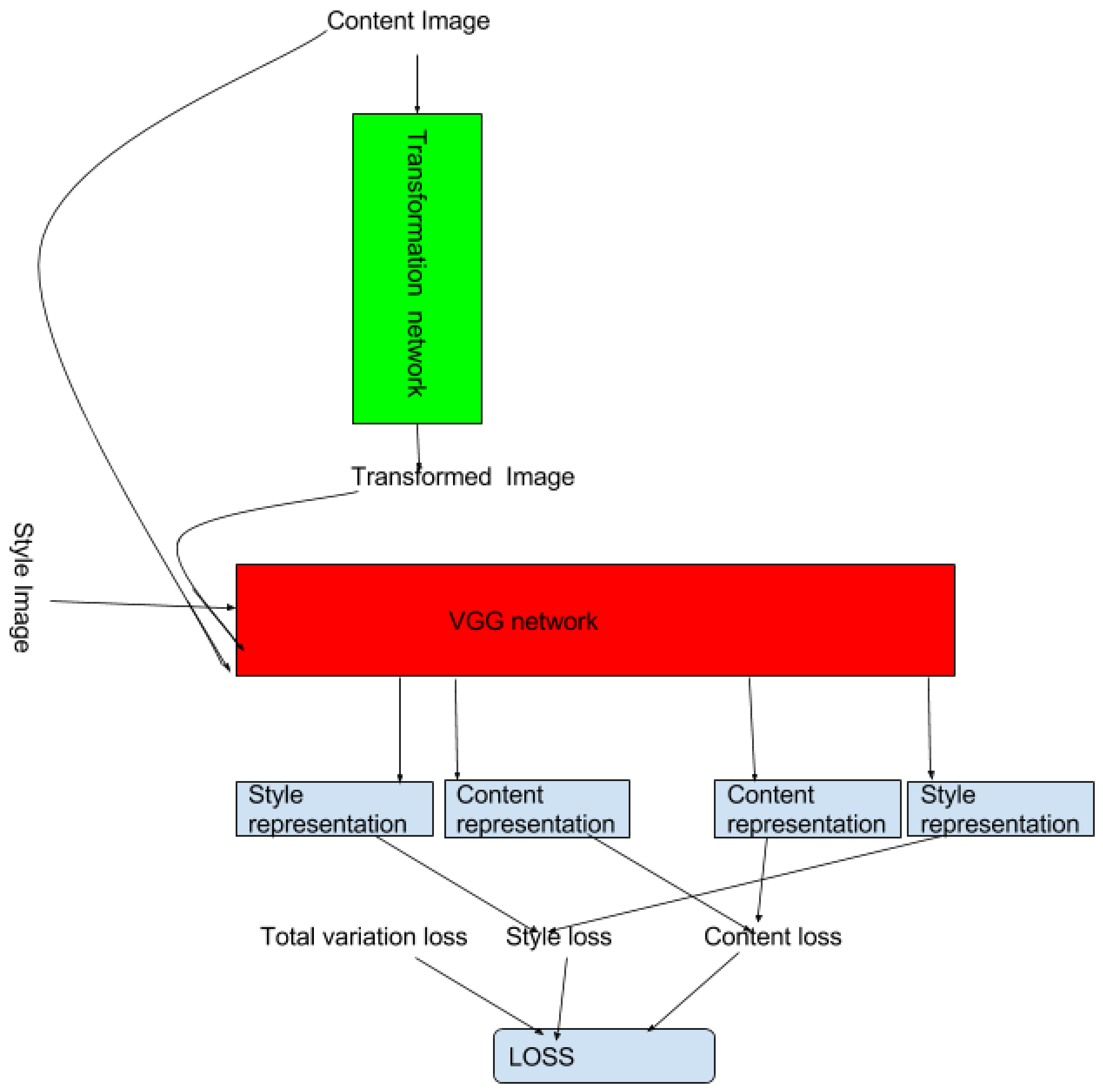
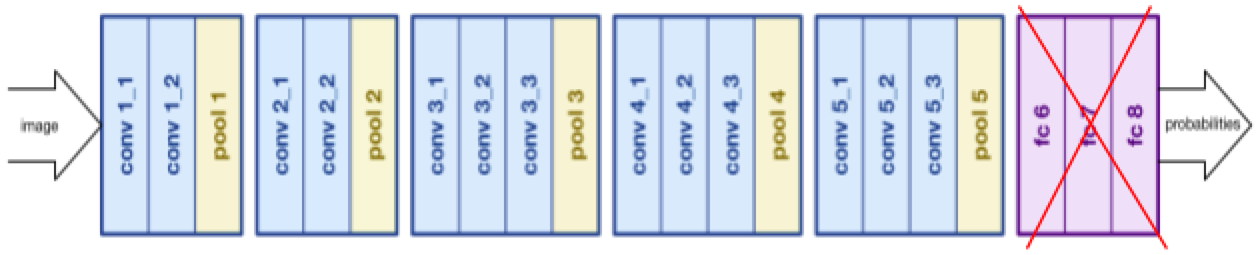


Fig : Architecture of style transfer in real time

VGG network :

VGG network is an outcome of ImageNet challenge in 2014 by oxford university team which capture's second position in competition. VGG network known for it's structural beauty which can be generalized to many networks/problems. So, people usually prefer VGG structure for their networks at first sight.

Let's discuss VGG network in brief .

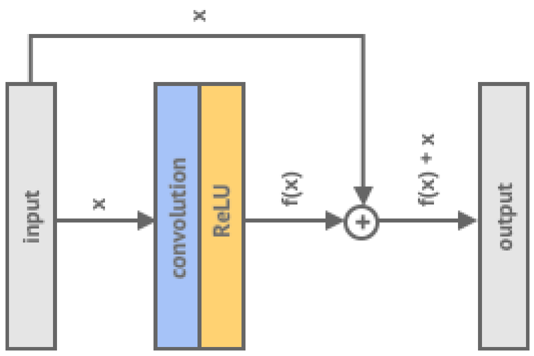


In our work we use pre trained VGG network i.e., we use the trained weights from ImageNet dataset . We don't consider last connected layers at all.

Transformation network : This is the main network that we are going to train for style transfer. This network takes an content image and gives output as transformed image. This transformed image is mixture of content image and style image on which we trained our network.

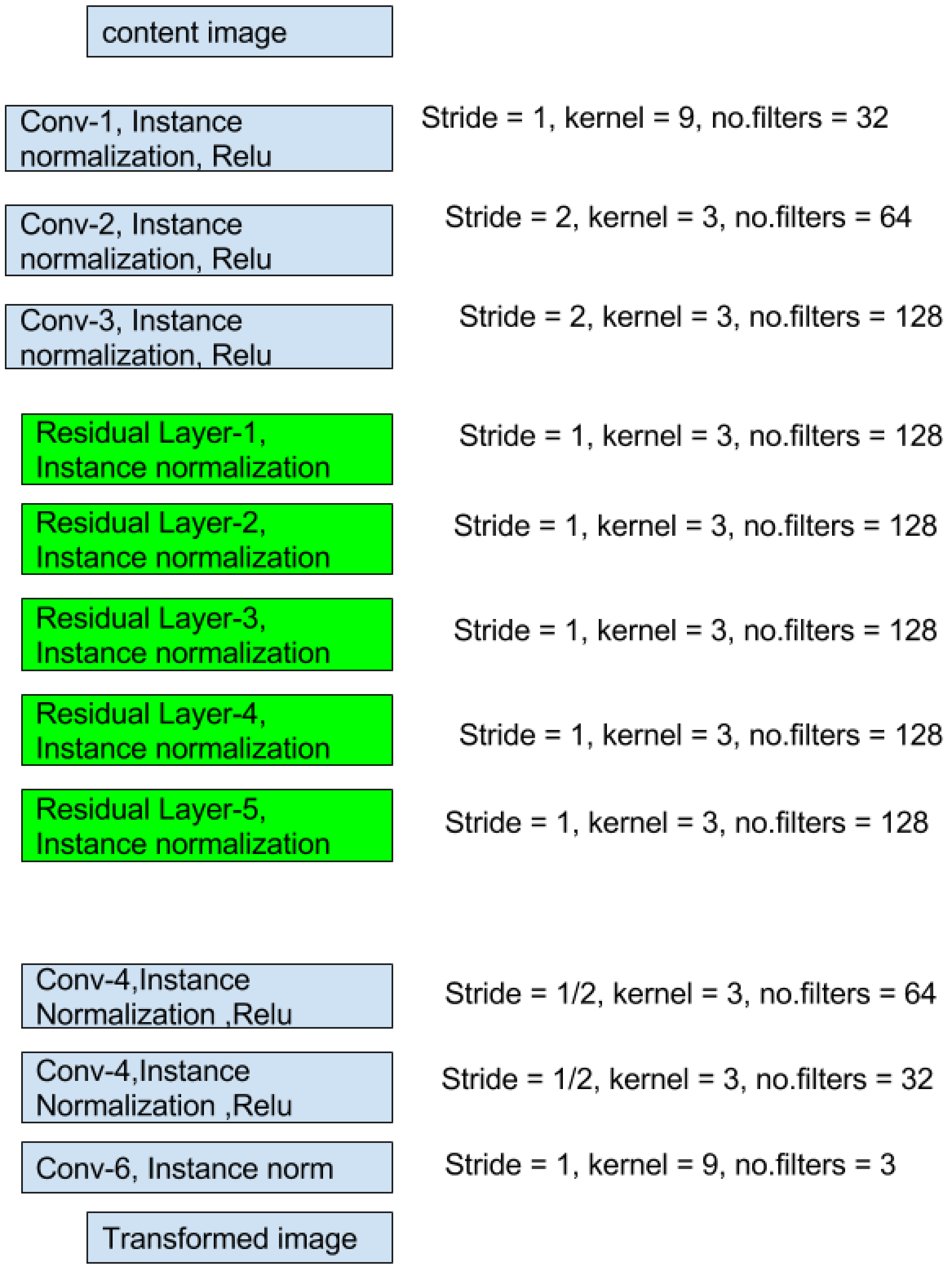
Convolution Layer : It is similar to fully connected layer but rejecting lots of weights (connections) i.e., output pixel is affected by surrounding pixels in input image. So we don't consider all connection with each pixel in input image to generate output pixel at particular position.

Residual Layer : Unlike Conv layers, it take consists of one conv layer and input passes through conv layer and it adds conv layer output with input and returns it. There are some motivation behind this, i.e., we trying to learn difference between input and output , so makes easier to train for certain tasks. But out layer does not learn how to transform input into new output , we are just adjusting input image.



Up Sampling Layer : This layer will take input image and increases it's size (resolution) by some factor as output. For example, conv-layer with stride ½ means we move kernel by half pixel that will lead to double the resolution but in practice we cannot move kernel by half pixel , so we pad the input with zeros and then moves the kernel window by 1 pixel.

Instance Normalization : After the each layer we get output image features which is normalized to zero mean and unit variance to capture spatial smoothness .


content image
Conv-1, Instance normalization, Relu
Stride = 1, kernel = 9, no.filters = 32
Conv-2, Instance normalization, Relu
Stride = 2, kernel = 3, no.filters = 64
Conv-3, Instance normalization, Relu
Stride = 2, kernel = 3, no.filters = 128
Residual Layer-1, Instance normalization
Stride = 1, kernel = 3, no.filters = 128
Residual Layer-2, Instance normalization
Stride = 1, kernel = 3, no.filters = 128
Residual Layer-3, Instance normalization
Stride = 1, kernel = 3, no.filters = 128
Residual Layer-4, Instance normalization
Stride = 1, kernel = 3, no.filters = 128
Residual Layer-5, Instance normalization
Stride = 1, kernel = 3, no.filters = 128
Conv-4,Instance Normalization ,Relu
Stride = 1/2, kernel = 3, no.filters = 64
Conv-4,Instance Normalization ,Relu
Stride = 1/2, kernel = 3, no.filters = 32
Conv-6, Instance norm
Stride = 1, kernel = 9, no.filters = 3
Transformed image

At initial layers, we used stride-2 that leads to half the size of input , so kernel filter can capture more of the content image without increasing kernel size . This is desired because the style transfer is applying the style to entire content image in a coherent way.

After that we used residual layer which is good for tasks like style transfer because transforming input image into mixture of content and style images which is also somewhat similar to input image.

After that we used up sampling layers because to restore the input image resolution because we have applied stride> 1 to input image that leads to reduction in resolution.

Activation Layers - ReLU :

Activation layers are very important as they introduce the nonlinearity to the model and allows the model to map much more complex nonlinear functions. The intuition behind the activation layers is simple. If there were no activation layers, then the final output from the model would be linear function of the input features. This is because each of the layers of the model is resulted from the previous layer by multiplying with a set of weights. So every layer of the model is a linear function of the previous layer and therefore the final layer would be a linear function of the input features.

This means that the model can now only model the functions which are linear functions of the input features and cannot model any more complex relations. Therefore certain changes have to be made for the model to be able to complex nonlinear functions. The introduction of an activation layer known as Relu does the trick. The Relu (Rectified linear unit) is a ramp function which can be defined as max (0,x). So if the input is greater than zero, the function returns the same value and zero if the input is less than zero. In this way, a nonlinearity is introduced. The activation layer is applied after every convolutional and the fully connected layers.

Squashing Layers

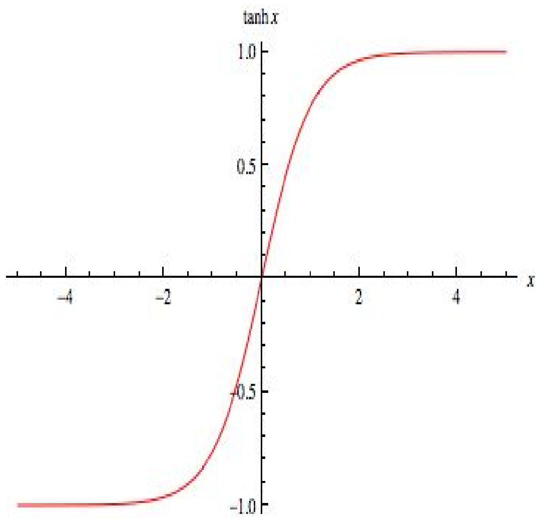
The large number of multiplications and the computations performed will result in a wide range of values which may vary from very small to very huge. Hence, it is important to squash these values to a certain range so that they can be used for computation in the next layer.

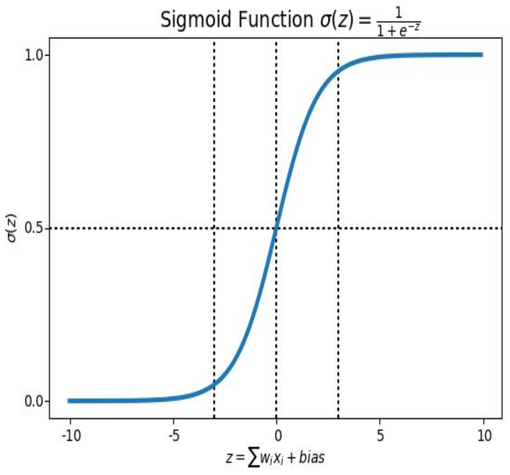
Sigmoid Function

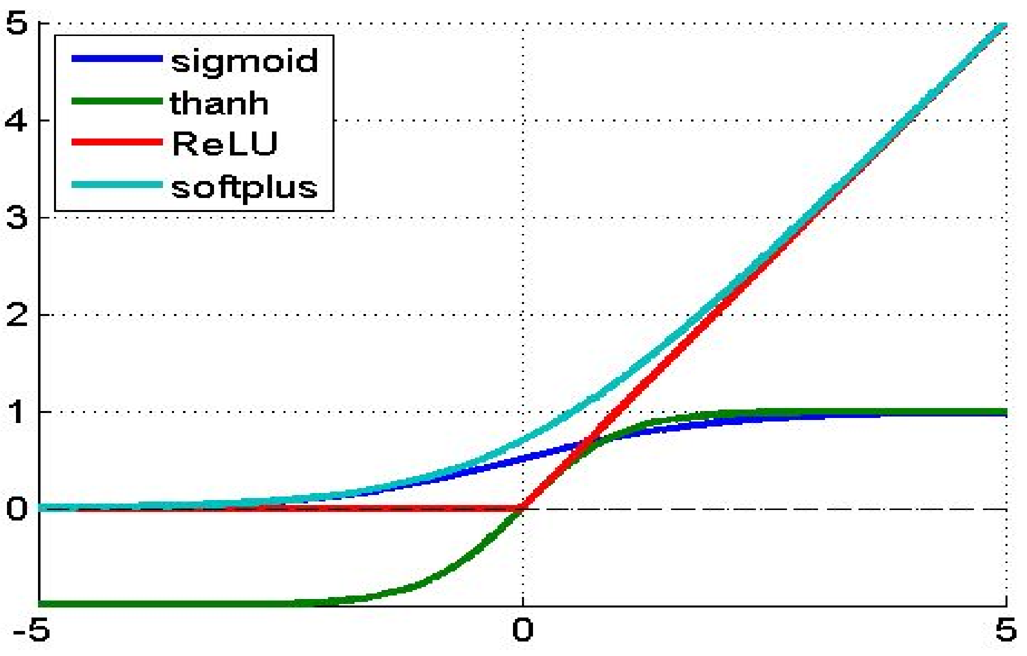
A Sigmoid layer squashes the input to a range varying from 0 to 1. Therefore all probabilistic functions are modelled using the Sigmoid function. This functions is function is shown in the below figure.

Tanh Function

A Tanh layer squashes the input to range from -1 to 1. It acts as an alternative to the sigmoid function but it can also give negative values. Tanh can provide stronger gradients compared to sigmoid layers because it is centered around zero.







Gradient Descent :

Let's consider $f(x,y)$ as scalar valued function which is showed in below figure.
$\nabla f(x,y)$ *is* gradient of $f(x,y)$.

$$\nabla f(x_0, y_0) = \begin{bmatrix} \frac{\partial f}{\partial x}(x_0, y_0) \\ \\ \frac{\partial f}{\partial y}(x_0, y_0) \end{bmatrix}.$$

$x1, y1 \;=\; [x0\; y0] - \; \alpha * \nabla f(x,y)$, where α *is learning rate.*

Think it as function as hill terrain, you are standing at random point in input space of this function on hill terrain. Now your goal is move to lowest point on hill terrain from your location, multivariable calculus says go in the opposite direction of your gradient vector by small distance. This is called Gradient Descent .

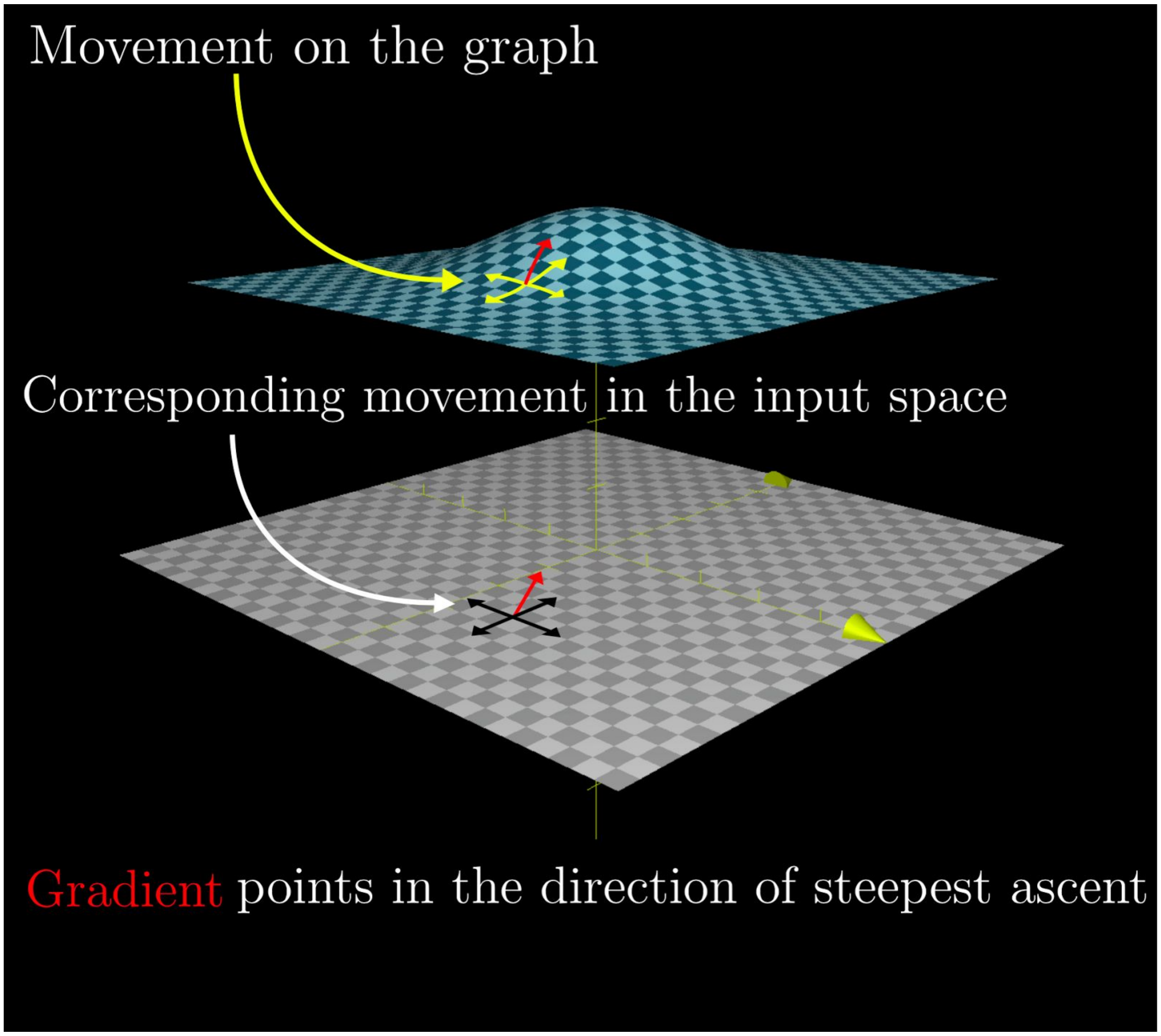
Gradient Ascent :

Let's consider $f(x,y)$ as scalar valued function which is showed in below figure.
$\nabla f(x,y)\ is$ gradient of $f(x,y)$.

It is similar to gradient descent but here we maximising the scalar output of the function , so called as maximization problem.

$x1, y1 \;=\; [x0\ y0] + \ \alpha * \nabla f(x,y)$, where $\alpha\ is\ learning\ rate.$

Think it as function as hill terrain, you are standing at random point in input space of this function on hill terrain. Now your goal is move to highest point on hill terrain from your location , multivariable calculus says go in the direction of your gradient vector by small distance. This is called *Gradient Ascent* . Below figure explains gradient ascent visually.


Movement on the graph
Corresponding movement in the input space
Gradient points in the direction of steepest ascent

LOSS FUNCTION :

$$Loss(content_{image,},\ weights\ of\ transformation\ network) = style\ loss + content\ loss + total\ variation\ loss.$$

VGG network weights are constants those are pre-trained weights on ImageNet dataset and style image is also constant, so these are not trainable. Usually $content_{image}$ is varying every time , so we won't consider it as trainable.

Implementation details :

```
for epoch in xrange(number of epochs) :
    for content_image in dataset :
        '''
        1. calculate loss function, it gives you loss value
        2. calculate partial derivatives of loss function w.r.t every parameter
        3. Update parameter according to gradient descent algo
        '''
#save your parameters.
```

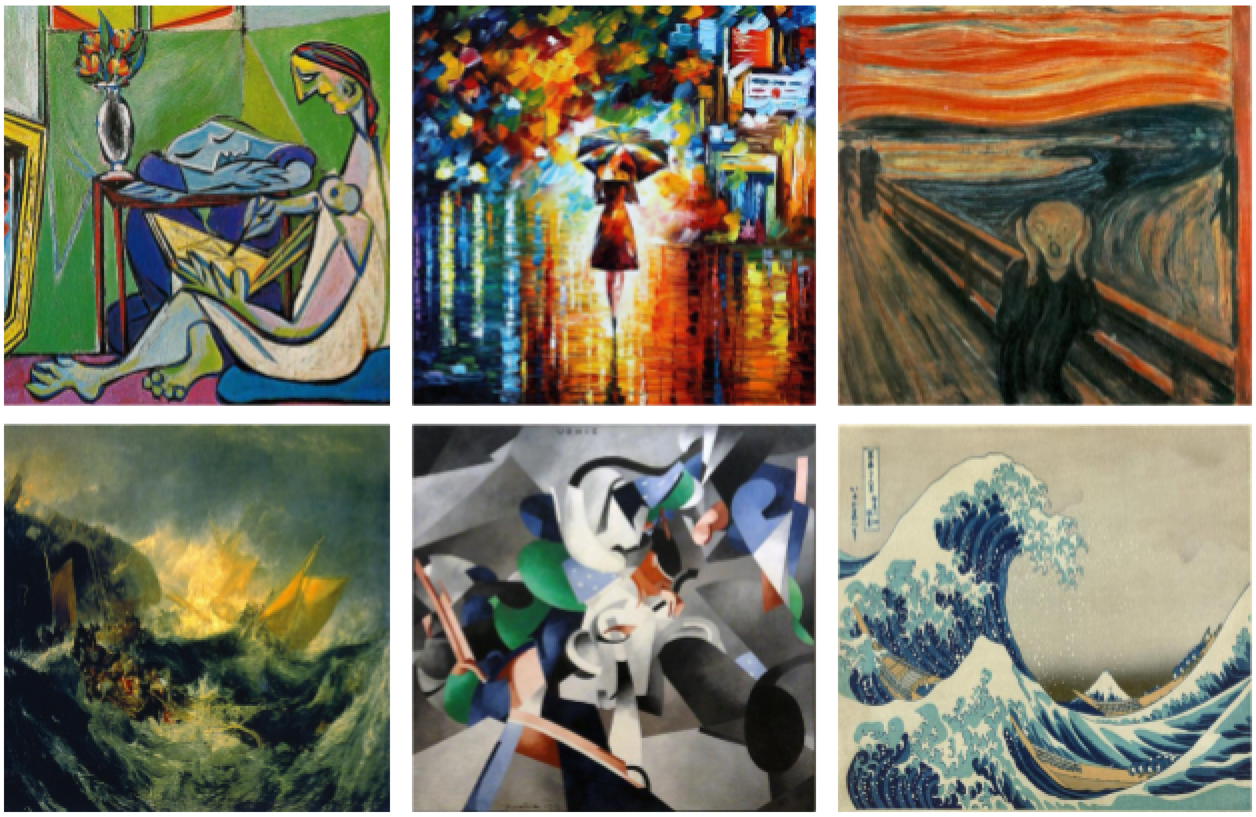
DATASET :

We have used MSCOCO-2014 dataset for our training purpose.

Brief description of dataset :

1. It contains 80,000 images
2. Each image is color image(RGB)
3. Each image size is big (648, 450, 3).

STYLE IMAGES are famous paintings. We have trained the network separately on 6 style images. Below are style images.



Phase-2: Music Synthesis

This phase is independent of phase-1. In this phase we will create a model to generate thematic music automatically. More formally, given the set of music files , our model learns how to generate music.

Music generation is a creative process by musicians. Artistic drawing is also a creative process by humans.Artificial Intelligence is all about thinking and learning on it's own like humans.But there was lot's of AI generated artistic drawing.Generating music is a difficult & open challenge problem because no one yet published some good results.

They are some results from Google Magenta, Deepmind but they all are in it’s starting phases of generating compelling music.

Methodology :

Dataset :

We are given set of mp3 music files for training the model.

Output :

Every time we invoke the model, it will generate some compelling music which is not exactly the training music dataset but it captures the patterns from training music.

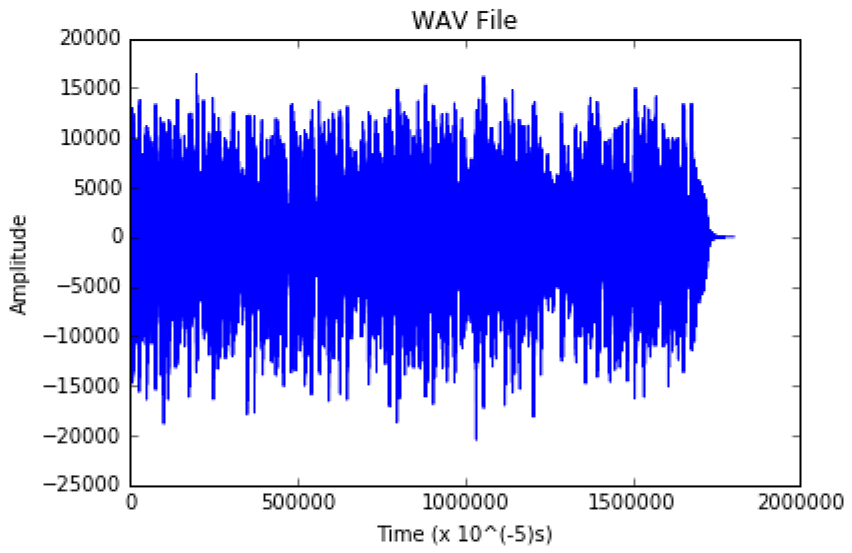
Preprocessing the dataset :

1. Converting mp3 to wav :

Training dataset contains audio format (mp3) which is stereo sounds. So, we convert it into monaural sound which is sufficient for work. In stereo sound , it uses more than one channel to code i.e., we hear sounds in both ears are different whereas in monaural sound it uses only one channel to code i.e., we hear same thing in both ears. Monaural sound saves lot of memory space and time to train our model.

Next this monaural sound is something that can not be used for training our model, so we converted it into wav format which can be used for our purpose.

Audio is continuous signal, so we divide it into discrete signal with some sampling frequency. Humans can only hear audio with frequency less than 20,000 Hz. Even if there are some signals with frequency greater than 20,000 Hz , but humans cannot hear those i.e., we can neglect those ones. Nyquist's Sampling Theorem says that sampling must be 2 times than 20,000 Hz, so we sampled the audio at 44,100 Hz which is considered to be standard.

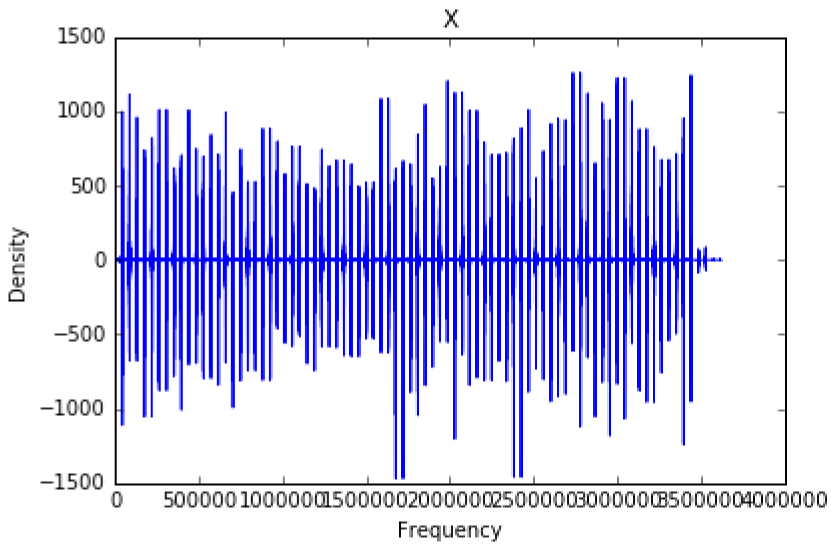


2. Converting to np tensors :

Using wav(library) , we can read .wav files into np array. This array length is M.

Dividing the np array into small block of size 44,100. For end block we padded with zeros to make the it to be 44,100 size. Now each block is a discrete waveform in time domain. Now data shape will be (N,44,100) where M = N*44100.

Then we converted this discrete wave in time domain into frequency domain using Fast Fourier Transform. FFT(Fast Fourier Transform) outputs real part and imaginary part , so our discrete waveform(each block) is of size 2*44,100. Now the data shape will be (N,2*44,100).



3. Normalising the data:

We normalize the data values into [-1 1]. This gives speed to train our model using Gradient Descent.

4. **Getting the Output label:**
   We shift the input sequence by one unit.

Loss Function :

We use *L2 norm distance* between labels and predicted ones.

Optimizer :

We use RMSprop(Root Mean Square) for optimising the loss function.This gives boost for our training process and it is also recommended optimizer for training RNNs.

Model :

We used static RNN of length 10 because our sequence length is 10 with LSTM as RNN basic unit of size 512. First is Dense layer which maps input feature(88200) into hidden dimension(512). Second is LSTM cell maps 512-->512 dimensions. Third one is again Dense layer which is maps 512-->88200 dimensions.

```
1.model = Sequential()
2.model.add(TimeDistributed(Dense(num_hidden_dimensions),
input_shape = (10, 88200)))

3.model.add(LSTM(input_dim = num_hidden_dimensions,
output_dim = num_hidden_dimensions, return_sequences = True))
4.model.add(TimeDistributed(Dense(88200)))
5.model.compile(loss = 'mean_squared_error', optimizer = 'rmsprop')
```

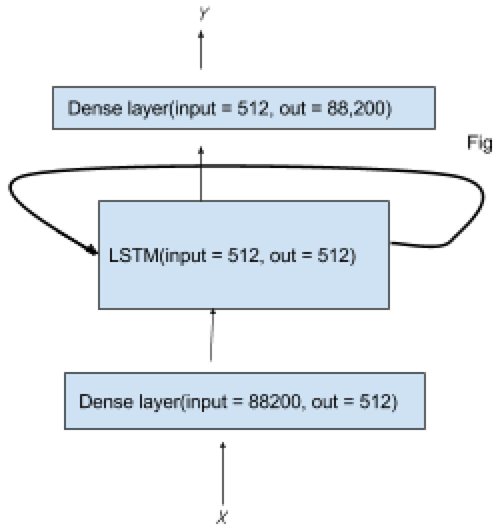
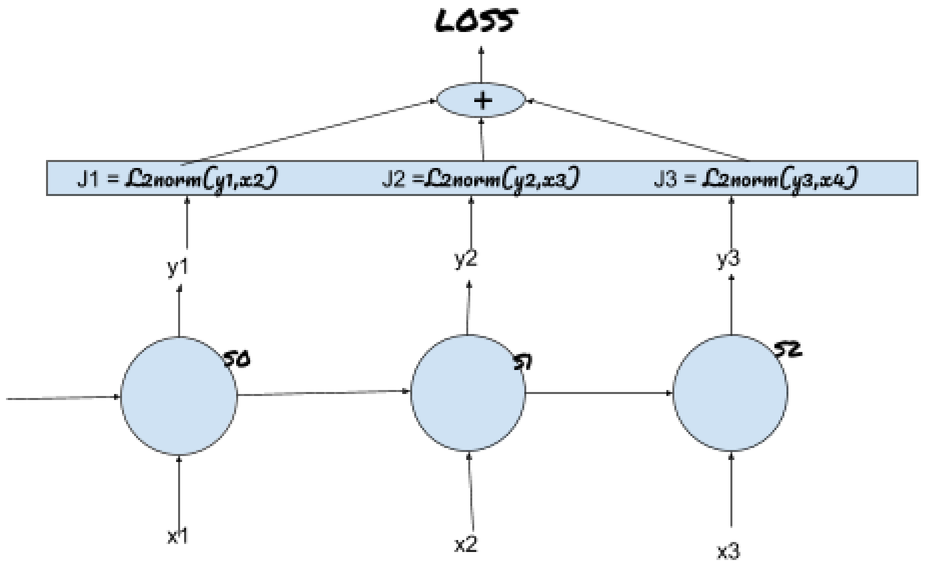

Y
Dense layer(input = 512, out = 88,200)
LSTM(input = 512, out = 512)
Dense layer(input = 88200, out = 512)
X

Fig : RNN model



Circles represent the computation in above graph

Fig : Expanded RNN unit

Save the Model weights :

Save the model weights after every 500 epochs for future use.

Generating Music :

Initially , give some random waveform from training data, from it will generate next waveform and give this to next LSTM cell as input , like this keep on generating next waveforms , at end concatenate all waveforms into one. This big waveform is a music.

Phase-3: Making video

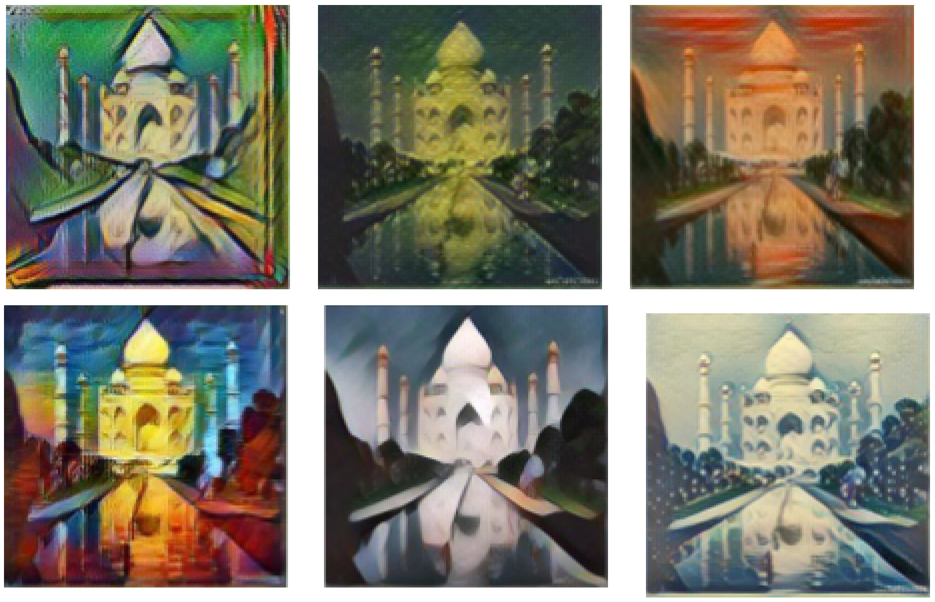
This phase is the most easily of all. Now we have a bunch stylized image and audio, so we can make the video by sliding the each image about 1 sec with background audio.

This can be achieved by ***ffmpeg*** in ubuntu.

Results :

For style transfer, results will be like the below one.

Original Image


Stylized Images

For music generation , since we can't put it on pdf, so we are providing google drive link which can be viewed by only IIIT mail.

Sample-1:https://drive.google.com/a/iiita.ac.in/file/d/1Ud3W7JZxATmNSHigJ-nP_-cipm8n1K3Y/view?usp=sharing

Sample-2 :https://drive.google.com/open?id=1y5nloL3cIBWsKe5Uo4ZhFQimR4ZtwDWi

Similarly for video, below is the google drive link which can viewed by only IIIT mail.

Sample : https://drive.google.com/open?id=1SGG7c2Q8t2x5xVze4Ft_rhLvbSUDM2sP

Software Requirements :

- **Python interpreter:**

  The source code is written entirely in python and therefore a python interpreter is required preferably version 2.7. It also works on python 3 but minor syntax changes are required.

- **Tensorflow:**

  Tensorflow is an open source deep learning framework developed by Google. Tensorflow library builds a computation graph and takes care of the forward and backward propagation of gradients. The model is implemented as a computation graph in tensorflow. The model consisting of both CNN and the RNN modules are both implemented in the form of computation graphs. Tensorflow also provides various gradient descent optimizers to train the model and adjust the weights.

- **Python Scientific Computing Libraries:**

  1.Numpy
  2.Matplotlib
  3.Scikit-Image

These libraries are necessary for different purposes. Numpy library is used to perform numerical computations efficiently. Matplotlib is a library which allows us to plot various graphs. Scikit-Image library is used to perform certain image processing tasks on the image. The library pandas is used to read and manipulate the dataset. It is used to process the MS COCO training and validation data.

HARDWARE REQUIREMENTS :

- **RAM or Memory**

One of the major constraints imposed by the neural network model is the runtime memory required. The amount of runtime memory required by the model is around 1.5 gigabytes approximately. Hence the minimum system requirement is to have at least 4 gigabytes of runtime memory to ensure any resource allocation errors do not occur. It is advisable not to perform training on normal machines due to two factors. The first being the lack of enough memory train the model in batches and the second is that the training can take an awfully long time on a normal CPU where operations are sequential. The testing and validation however can be performed on the local machines.

- **GPU (Graphics processing unit):**

A GPU is required to perform the training phase of the project. This is not a necessary requirement as training can be done on normal machines as well. However, it is not advisable to train on CPU because of its limitations. As the dataset is very huge, It would take a very long time to in order to complete the training process on a normal CPU because the matrix operations would be computed sequentially rather than in parallel which would make very slow. Also the data cannot be trained in batches due to the memory constraints. The ideal situation would be to train on a GPU in batches so that the training can be completed quickly in a few days or a week. This would however require a very powerful GPU with at least 8-16 gigabytes of memory so that the data can be trained in batches of 32 or 64. Training in smaller batches will once again double the training time required. It would be unadvisable to train on GPUs on normal machines as the batch size would have to be small due to memory constraints and hence it would result in the training time taking upto months. So theoretically even though A GPU is not absolutely necessary - for practical purposes, It is essential in order to perform different experiments in a short period of time and reach a satisfactory conclusion.

REFERENCES :

[1] A Neural Algorithm of Artistic Style by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge

[2] Perceptual Losses for Real-Time Style Transfer and Super-Resolution,Justin Johnson, Alexandre Alahi, Li Fei-Fei.

[3] Instance Normalization: The Missing Ingredient for Fast Stylization by Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky.

[4] Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling - Authors(Hasim Sak, Andrew Senior, Francoise Beaufays)

[5] GRUV: Algorithmic Music Generation using Recurrent Neural Networks - Authors (Aran Nayebi,Matt Vitelli)

[6] S. Hochreiter, J. Schmidhuber. Long short-term memory. Neural Computation, 1997.

[7] https://magenta.tensorflow.org/

[8] http://www.deeplearningbook.org/, Ian Goodfellow and Yoshua Bengio and Aaron Courville.