A Software Engineering Project
On

Flamingo - Deep learning For Art

as the partial fulfillment of

Semester -VI study

for the degree of

Bachelor of Science (Computer Science)
(Affiliated to Veer Narmad South Gujarat University)

during the academic year 2020-2021.

Developed By:

1) Kush Gabani

Guided By:

Dr. Yesha Mehta

Sarvajanik Education Society

Shree Ramkrishna Institute of Computer Education and Applied Sciences

Behind P.T Science College, M.T.B. College Campus, Athwalines, Surat-395 001.

Sarvajanik Education Society
Shree Ramkrishna Institute of Computer Education and Applied Sciences
Behind P.T. Science College, M.T.B College Campus, Athwalines, Surat-395 001.

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE

This is to certify that

	Student name	*Exam no*
1)	Kush Gabani	20104

has successfully completed his software engineering project work entitled

Flamingo – Deep Learning For Art

as the partial fulfillment of

Semester -VI study
for the degree of

Bachelor of Science (Computer Science)
(Affiliated to Veer Narmad South Gujarat University)
during the academic year 2020-2021.

Internal Project Guide

HOD
Dept. of Computer Science

Date : 17/08/20201

Place : Surat

University Examination:

Examination No. : _____

Signature of External Examiners : _____

_____

_____

INDEX

SR.NO	CONTENTS	Page. No
1.	**Introduction**	
	• Project Profile	1
2.	**System Introduction**	
	• System Definition, Objective and scope	2
	• Hardware & Software Requirements	3
	• Overview of Deep Learning Concepts	4
3.	**Requirement Analysis & Modeling**	
	• Expected working of system	12
	• Dataset	13
	• Model Architectures	13
4.	**Design**	
	• System Flowchart	21
	• Form & Report Designs (Screenshots) Layouts	22
5.	**Coding**	
	• Training Methodology	26
	• Inference Methodology	29
	• Code Snippets	30
6.	**Testing**	57
7.	**Tools & Technologies**	58
8.	**System Limitations and Dependency Constraints**	59
9.	**Future Enhancement & Opportunities**	61
10.	**Bibliography & References**	62

INTRODUCTION

Project Profile

Project Title	Flamingo – Deep Learning for Art
Project Definition	A deep learning application to demonstrate the use of Machine Learning In the field of deep learning
Technology Used	Flask, Python (TensorFlow, Keras), HTML, CSS
Front-end	HTML, CSS, Tensorboard (Visualization)
Back-end	Flask, TensorFlow and Keras
Internal Guide	Yesha ma’am
Documentation Generation Tool	Google Docs
Created By	Kush Gabani

SYSTEM INTRODUCTION

System Definition

- Deep Learning models packaged in a web application that focuses to assist artists and transfer artistic styles to generate art and lead to aesthetically pleasing art. The application comprises of two modules that deals with two distinct types of art – Music Generation and Fast Neural Style Transfer.
- The music generation module is responsible to generate a music for any arbitrary number of seconds taking the persona of three (Beethoven, Mozart, and Schubert) of the best musicians of all time into account.
- The second module is Fast Neural Style Transfer (FSNT) where two images are selected i.e. the content image and the style image. Content and structure is extracted from the content image and the features of the style image are combined a single photo that will portray how the content would look if a style filter of that image was to be applied on it.

System Objective

- The project primarily aims at demonstrating how deep learning can be applied in the field of art like Music and Paintings and help assist artists to create better.
- The music generation module should be able to generate new music patterns of any arbitrary number of seconds taking into account the persona of the artist selected (Beethoven, Mozart or Schubert).
- Fast Neural Style Transfer module must be capable of generating new image that combines the structure of one content image with the stylistic features of a style image.

System Scope

- The underlying LSTM model for the music generation module can be improved in such a way that it would be able to end a piece of music smoothly instead of abruptly stopping.
- The model architecture can also be modified into an encoder-decoder LSTM that has proved to be more effective in generating new music patterns.
- The FNST as of now can only generate content images from a set of different style images but this can be modified by training a new model that will be capable of generating images from any arbitrary style images.

Hardware & Software Requirements

- HARDWARE REQUIREMENTS
  - Training Phase for Deep Neural Networks (Google Colab)

PROCESSOR	Xeon Processors @ 2.3GHz
RAM	12GB DDR5
SSD	64GB
GPU	Tesla K80 2496 CUDA Cores
RUNTIME ENVIRONMENT	5 hrs. active runtimes

  - UI and Deployment Phase

PROCESSOR	Intel i3 @ 2.3 GHz
RAM	4GB DDR4
SSD	64GB
GPU	-
RUNTIME ENVIRONMENT	-

- SOFTWARE REQUIREMENTS

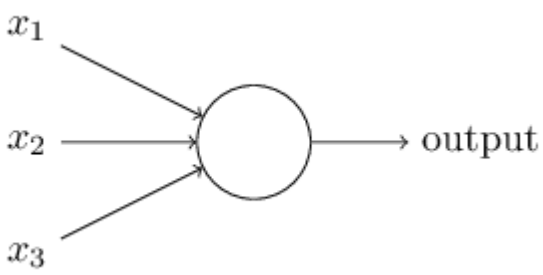
Operating System	Windows 7 / MacOS Catalina
Training Environment	Google Colab with GPU
Front-end Tools	HTML, CSS
Back-end Tools	Flask, Python > 3.8, TensorFlow 2

Overview of Deep Learning Concepts

NEURAL NETWORKS

Neural Networks, also known as ANNs (Artificial Neural Networks) are a subset of machine learning and at the core of deep learning. As it can be seen, its name and structure are inspired by a human brain. A simple neural network comprises of node layers containing an input layer, one or more hidden layers, and an output layer. Each node layer can have n number of neurons where n is an arbitrary positive integer. Each node connects to another node in the next layer and has a weight associated to it. If the output or the value store in a node/neuron exceeds a given threshold, that neuron is activated, sending data to the next layer of the network. If it does not exceed the threshold, no data is passed along the next layer of the network.

Let's start with the simplest neural network, a perceptron. Perceptrons were developed in the 1950s and 1960s by a Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. A perceptron takes several binary inputs, say x1, x2... So on, and produces a single binary output.



In general, a perceptron can have more or fewer inputs than these. Rosenblatt proposed a simple rule to compute the output. He introduced weights, w1, w2... and so on; weights are real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j (w_j x_j)$ is less than or greater than some threshold value. Just like the weights, the threshold is a real number which is a parameter of the neuron. The algebraic definition of the output would be:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

This is all there is for a working of the basic perceptron model.

ACTIVATIONS FUNCTIONS

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node in a layer of the neural network. Sometimes they are also referred to as transfer functions. An activation function is applied to the neuron after the weighted sum of that node has been calculated. The choice of activation functions has a profound impact on their

capabilities and the performance of a network. In addition, each neuron in a layer can also have distinct activation functions. Many activation functions are non-linear in nature and may be referred to as the nonlinearity in the layer. There are various types of activation functions available which can be applied to a node. A hidden layer in a network is the layer that receives input from another layer and provides output to another layer. It does not directly contact input data or produce outputs for a model. Typically, a differentiable non-linear activation function is used in the hidden layer of a network. This allows the model to learn more complex functions than a network trained using a linear activation function. Some of the commonly used activation functions are: sigmoid, ReLU, and tanh.

- Sigmoid
  these activation functions are also referred to as logistic function. It takes any real value as an input and outputs in the range 0 to 1. The larger the input, the closer the output will be 1, whereas smaller the input, the closer the output will be to 0.

$$S(x) = \frac{1}{1 + e^{-x}}$$

- ReLU
  ReLU stands for Rectified Linear Unit. It is perhaps the most common activation function used for hidden layers. It is common because it is both simple to implement and effective at overcoming the limitations of other previously popular activation functions, such as sigmoid or tanh

$$R(x) = \max(0, z)$$

- Tanh
  The hyperbolic tangent activation function is also referred to as simply as the tanh function. It is very similar to the sigmoid function and even has the same S shape. The function takes any real value as input and outputs values in the range -1 to 1. The larger the input, the closer the output value will be to 1, whereas he smaller the input, the closer the output will be -1.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Softmax
  It is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. The most common of the Softmax function is applying it at the output layer of the model. This will give us the probability of a class being the correct guess.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

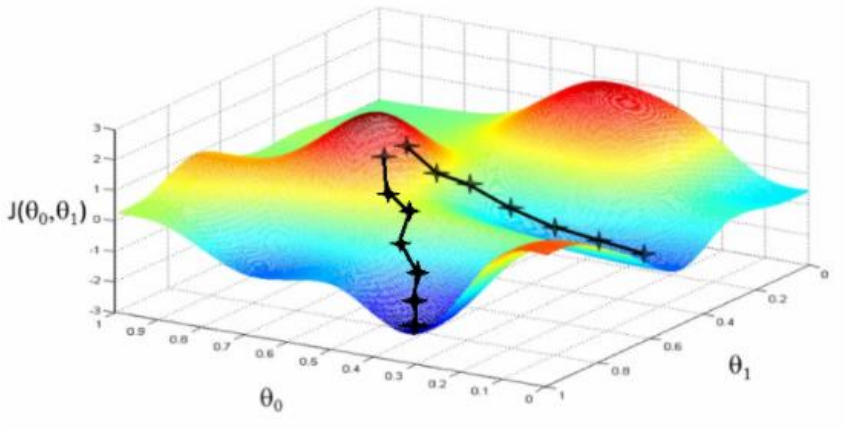
UNIVERSAL APPROXIMATION THEOREM

One of the striking facts about neural networks is that they can compute any function at all. No matter what the function, there is guaranteed to be a neural network so that for every possible input, x, the value f(x) is output from the network. This result holds even if the function has many inputs and outputs. This tells us that neural networks have a kind of universality. The universality theorem holds even if we restrict our networks to have just one single intermediate between the input and the output layer. This theorem is well known by the people who use the neural networks. The theorem is as follows: “In the mathematical theory of artificial neural networks, the universal approximation theorem states that a feed forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of Rn, under mild assumptions on the activation function.”

A neural network consists of weights, biases and non-linearity like sigmoid function. For a linear function, a network can contain linear activations like y = x. But, most functions in the world are non-linear hence we require non-linear activation functions. The values flow through the weights, get added to the bias and are activated by the activation functions.

GRADIENT DESCENT

Gradient descent is an optimization algorithm for minimizing the cost. To think of it intuitively, while climbing down a hill you should take small steps and walk down instead of jumping down at once. Therefore, what we do is, if we start from a point x, we move down a little i.e. delta, and update our position to x-delta and we keep doing the same till we reach the bottom.



BACKPROPAGATION ALGORITHM

It is probably the most fundamental building block in a neural network. It was first introduced in 1960s and almost 30 years later popularized by Rumelhart, Hinton, and Williams in a paper called “Learning representations by back-propagating errors”. The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backwards pass while adjusting the model’s parameters. While training, backpropagation computes the gradient of the loss function with respect to the weights of the network for a single input-output example, does so efficiently, unlike a naïve direct computation of the gradient with respect to each weight individually. This makes it feasible to use gradient methods for training neural networks, updating weights to minimize loss; It works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backwards from the last layer to avoid redundant calculations of intermediate terms in the chain rule.

DEEP LEARNING

It is a subfield of machine learning that is concerned with algorithms inspired by the structure and function of a human brain called artificial neural networks. Deep learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutional neural networks etc. have been applied to fields including computer vision, speech recognition, natural language processing etc. The “deep” in the term refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but with that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimizes implementation while retaining theoretical universality under mild conditions. The layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability.

EPOCH, DROPOUT AND BATCH NORMALIZATION

An epoch is defined as a single training iteration of all batches in both forward and back propagation. This means 1 epoch is a single forward and backward pass of the entire input data. The number of epochs you would use to train your network is to be chosen by us. It’s highly likely that more number of epochs would show higher training accuracy of the network, however, it would also take

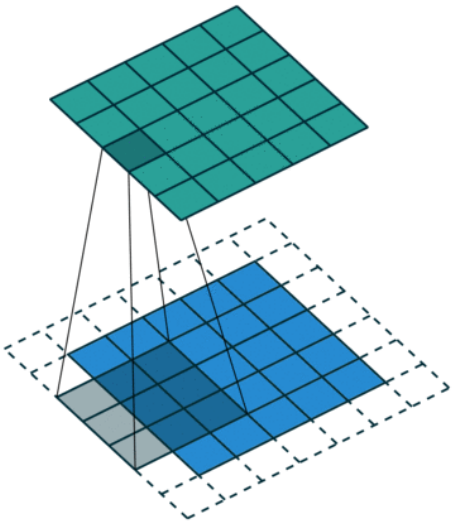
longer for the network to converge. Also you must take care that if the number of epochs are too high, the network might be over fit.
Dropout is a regularization technique which prevents over-fitting of the network. During training, a certain number of neurons in the hidden layer is randomly dropped. This means that the training happens on several architectures of the neural network on different combinations of the neurons.
Batch Normalization is a technique applied to ensure that the distribution of data is the same as the next layer hoped to get. When we are training the neural network, the weights are changed after each step of gradient descent. This changes how the shape of data is sent to the next layer.

CONVOLUTIONAL NEURAL NETWORK
It is a type of deep neural network which can take in an input image, assign learnable weights and biases to various aspects in the image and be able to differentiate one from another. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods, filters are hand-engineered, with enough training, ConvNet has the ability to learn these filters or characteristics. An image is nothing but a matrix of pixel values. In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images. A ConvNet is able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture performed a better fitting to the image dataset due to the reduction in the number of parameters involves and reusability of weights.

- The kernel (Convolution layer)
  A kernel is a small window of a fixed size that is moved on an image and the portion of the image that is covered by the kernel performs some operation. This operation is known as a convolution operation. It is an element-wise multiplication of each element in the kernel and the portion of the image and then summing them. The kernel moves to the right with a certain stride value till it parses the complete width. Then, it hops down to the beginning of the next row. In the case of images with multiple channels, the kernel has the same depth as that of the input image. The primary objective of the convolution operation is to extract the high-level features such as edges, from the input image. Conventionally, the first ConvLayer is responsible for capturing the low level features such as edges, color, gradient orientation etc. With added layers, the architecture adapts to the high level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would. There are two types of results to the operation – one in which the convolved feature is reduced in dimensionality as compared to the input,

and the older in which the dimensionality is either increased or remains the same. This is done by applying valid padding in case of the former, or same padding in the case of the latter.

- Pooling layer
  Similar to the convolutional layer, the pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. There are two types of pooling: max pooling and average pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel. On the other hand, average pooling returns the average of all the values from the portion of the image covered by the kernel.



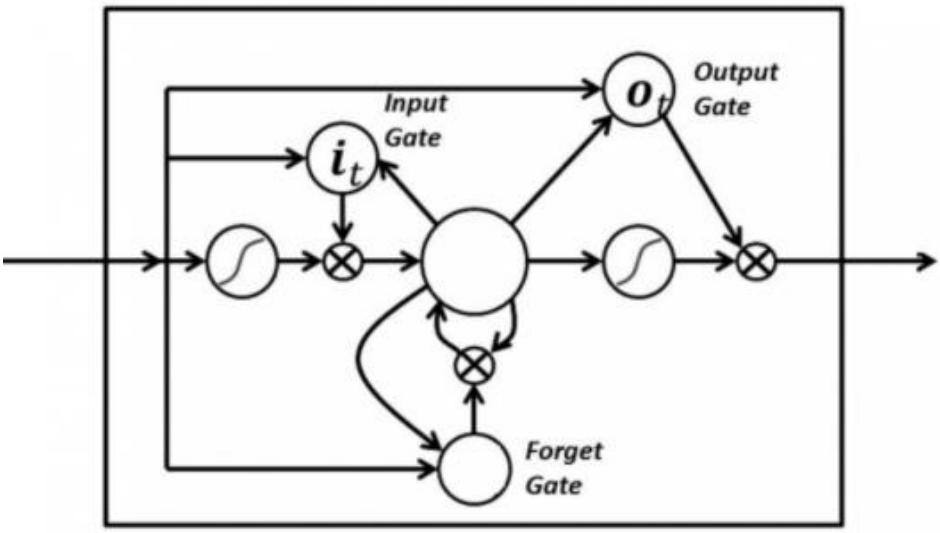
RECURRENT NEURAL NETWORKS

Recurrent Neural Networks are a class of neural networks that are helpful in modeling sequence data. Derived from feed-forward networks, RNNs exhibit similar behaviour to how human brains function. They are a class how neural networks produce predictive results in sequential data that other algorithms can’t. Because of their internal memory, RNNs can remember important things about the input they received, which allows them to be very precise in predicting the next value in the sequence. This is why they’re preferred algorithm for sequential data like time series, audio, speech, text, financial data etc. In an RNN, the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously. A usual RNN has a short term memory. In combination with an LSTM they also have a long term memory. An RNN is able to remember those characters because

of its internal memory. It produces output, copies that output and loops it back into the network.

LONG SHORT TERM MEMORY

LSTM networks are an extension for recurrent neural networks, which basically extends the memory of an RNN. Therefore, it is well suited to learn from important experiences that have very long time lags in between. In an LSTM you have three gates: input gate, output gate, forget gate. These gates determine whether or not to let new input in, delete the information because it isn't important, or let it impact the output at the current time step.



NEURAL STYLE TRANSFER

Neural Style Transfer is an optimization technique used to take three images, a content image, style reference image and an input image you want to style. And blend them together such that the input image is transformed to look like the content image but "painted" in the style of the style image. The principle of neural style transfer is to define two distance functions, one that describes how different content of two images are and another that describes the difference between two images in terms of their style. We try to transform the input image to minimize the content distance with the content image and its style distance with the style image.

- Content Loss:
  We pass the network both the desired content image and our base input image. This will return the intermediate layer outputs from the model. Then we simply take the Euclidean distance between the two intermediate representations of those images. We perform backpropagation in the usual way such that we minimize this content loss. We thus change the initial

image until it generates a similar response in a certain layer as the original content image.

$$L^l_{content}(p, x) = \sum_{i,j}(F^l_{ij}(x) - P^l_{ij}(p))^2$$

- Style Loss:
  Computing style loss is a bit more involved, but follows the same principle this time feeding out network the base input image and the style image. However, instead of comparing the raw intermediate outputs of the base input image and the style image, we instead compare the gram matrices of the two outputs. To generate a style for our base input image, we perform gradient descent from the content image to transform it into image that matches the style representation of the original image. We do so by minimizing the mean squared distance between the feature correlation map of the style image and the input image. The contribution of each layer to the total style loss is described by:

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G^l_{ij} - A^l_{ij})^2$$

$$L_{style}(a, x) = \sum_{l \in L} w_l E_l$$

, where $(w_l = \frac{1}{\|L\|})$

FAST NEURAL STYLE TRANSFER (BY JC JOHNSON)

Image transformation is a process where an input image is transformed into an output image. Typically, feed-forward convolutional neural networks are trained using a per-pixel loss between the output and ground-truth images. Parallel work has shown that high-quality images can be generated by defining and optimizing perceptual loss functions based on high-level features extracted from pre-trained networks. Combining the benefits of both approaches, JC Johnson proposed the use of perceptual loss functions for training these feed-forward networks for image transformation tasks. Compared to the optimization-based method, this feed-forward network a.k.a. Transformer Net, gives similar qualitative results but is three orders of magnitude faster.

REQUIREMENT ANALYSIS & MODELING

Expected Working of the System

- Home Page
  - The home page is the entry to the web application. A client is directed here when they spin up a local host server at port 5555.
  - The home page will define the routes to the two modules of the project i.e. Music Generation and Fast Neural Style Transfer. When clicked on the either of the two buttons, the client is directed to their respective module.
  - Other than the two routes, client will also be to refer to the datasets being used, this documentation itself and the project's code base that is made public on Github on https://www.github.com/KushGabani

- Music Generation Page
  - This page is where the Deep Neural Network to generate new music patterns reside. The model has been deployed and made available through a flask API.
  - The client must select one of the three artist whose persona shall be reflected in the generate music pattern.
  - After selecting an artist, the client then needs to input how long the generated piece of music be in seconds.
  - Once, the user submits their choices and input the trained model is then given the input is submitted to the backend where the LSTM models generates a new music pattern
  - Once the new music is generated, the client is then directed to a success page where they can play that music.

- Style Transfer Page
  - This is the webpage where the user can perform Fast Neural Style Transfer. The underlying deep learning has been made available with Flask.
  - The user must upload a content image on which the style needs to be applied by the means of drag and drop or the basic input UI.
  - After uploading the content image on the server, the user must select the name of the painting whose style needs to be applied on the content image.
  - The client is then directed to the success page, once the model produces a new image by combining the content and style image

- Success Page
    - The results of any of the two modules are displayed on this page and the client is redirected automatically once they click the submit button on the modules.

Dataset

- Music Generation

  The music generation dataset is a collection of musical pieces created by three of the best artists of all time. Each of the three artists contains an average of 20-30 music pieces of varied length.

ARTISTS	**NUMBER OF MUSIC**
Beethoven	29
Mozart	21
Schubert	29
Total	**79**

- Fast Neural Style Transfer

  This module builds a transformer net that is trained on the COCO dataset (2014). This 13GB of data that contains almost 80,000 images in a real-life context. This dataset is commonly used for object detection and object classification of day to day objects and which is why this proved to be the correct dataset as the content image will contain objects found in day to day life.

PROPERTIES	**VALUES**
No. of images	80,000
Size of the images	Variable for each image

Model Architecture

Fast Neural Style Transfer

The inference model for Fast Neural Style Transfer is named as a transformer net. The transformer net does not contain any Pooling layers. Instead, they reduce the image size using solely striding. The Transformer Net is a stack of a series of Convolutional Layers followed by a stack of Transposed Convolutional Layers. Instance Normalization is proved to produce better quality features and images. In instance normalization, mean and variance are calculated for each individual channel for each individual sample across both spatial

Dimensions. The second half of the transformer net comprises of transposed convolutional layers which expands and creates the image from a latent space created in the middle of the model.

When we look closely at images generated by neural networks that directly use transposed convolutional layers, we see a strange unwanted checkerboard pattern of artifacts and in result the image appears low resolution. When we try to generate images, we generally build them up from low resolution and high level descriptors. This allows the network to describe the rough image and then fill in the details. When applying deconvolutions to an image, the checkerboard effect is clearly visible if the kernel size is not divisible by the stride which easily causes an uneven overlap. Typically, models like these use several layers of deconvolutions they sometimes cancel out the artifacts but often it compounds them, creating the effect at scale. To avoid these artifacts, the common method is to choose the kernel size in such a way that it is divisible by the stride avoiding the overlap issue. However, it is still easier for deconvolutions operations to fall into creating artifacts. So another approach is to separate out upsampling to a higher resolution from convolution to computer features. For example, resizing an image using nearest neighbour method and then do a convolution operation on the resulting image. This approach is quite easy to code and approach.

As we go deeper into the model, we use residual blocks of Instance Normalized Convolution layers. A residual block is a stack of layers in such a way that the output of a layer is taken and added to another layer deeper in the block. The non-linearity is then applied after adding it together with the output of the corresponding layer in the main path. This by-pass connection is known as a skip connection. The residual blocks create an identity mapping to activations earlier in the network to thwart the performance degradation problem associated with deep neural architectures. The skip connections help to address the problem of vanishing and exploding gradients. ReLU activation is applied after each Instance Normalized Convolution operation except the last layer of the Transformer Net.

As the model comprises solely of convolution operations, an image of any size can be passed through, to generate a new image of the selected style. There are a total of 8 style images to select from. Thus, there are 8 different transformer nets i.e. one transformer net is trained for each style image to produce a new image whose stylistic features will be the same as the style image. The architecture can be abstractly viewed as:

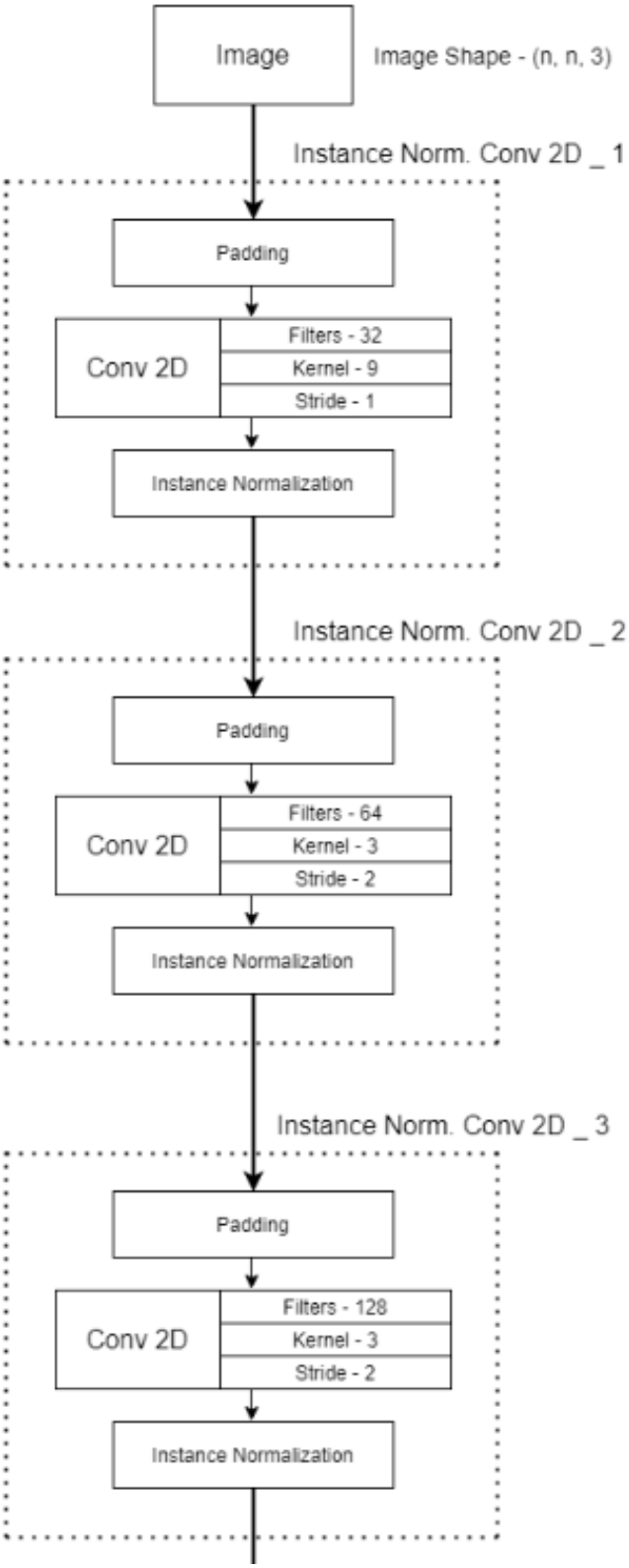
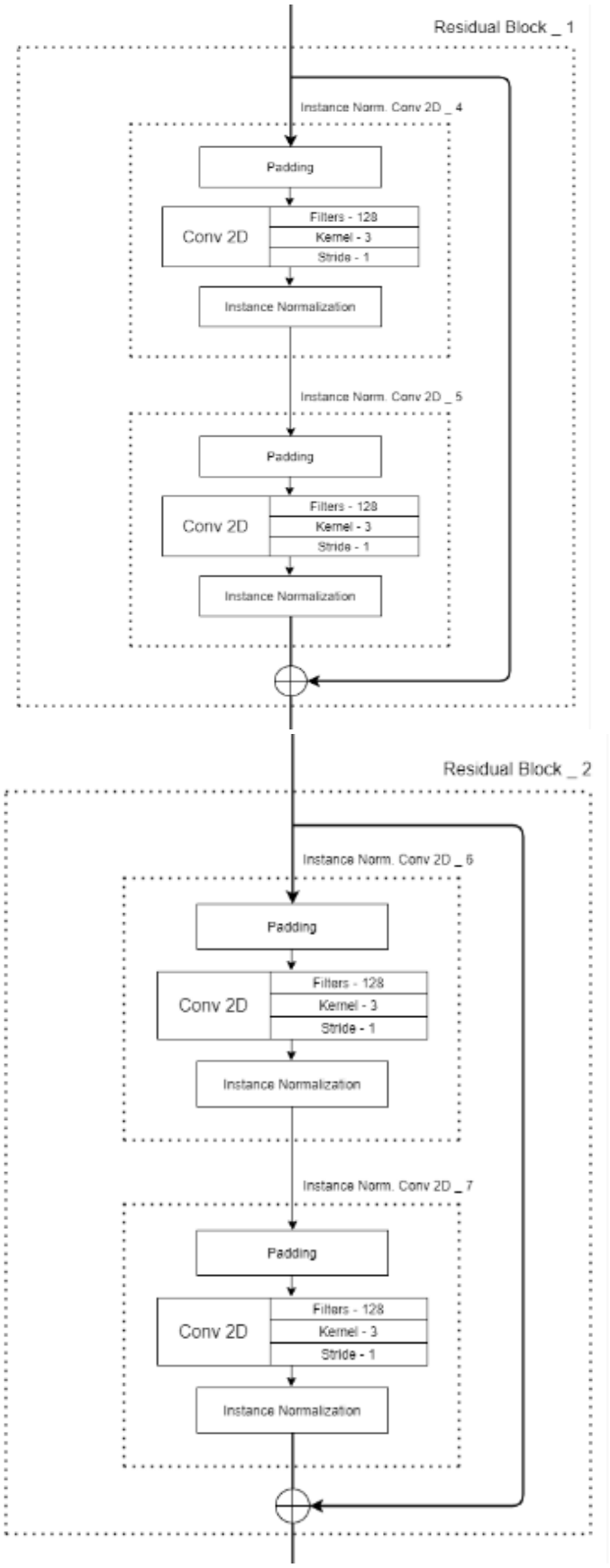
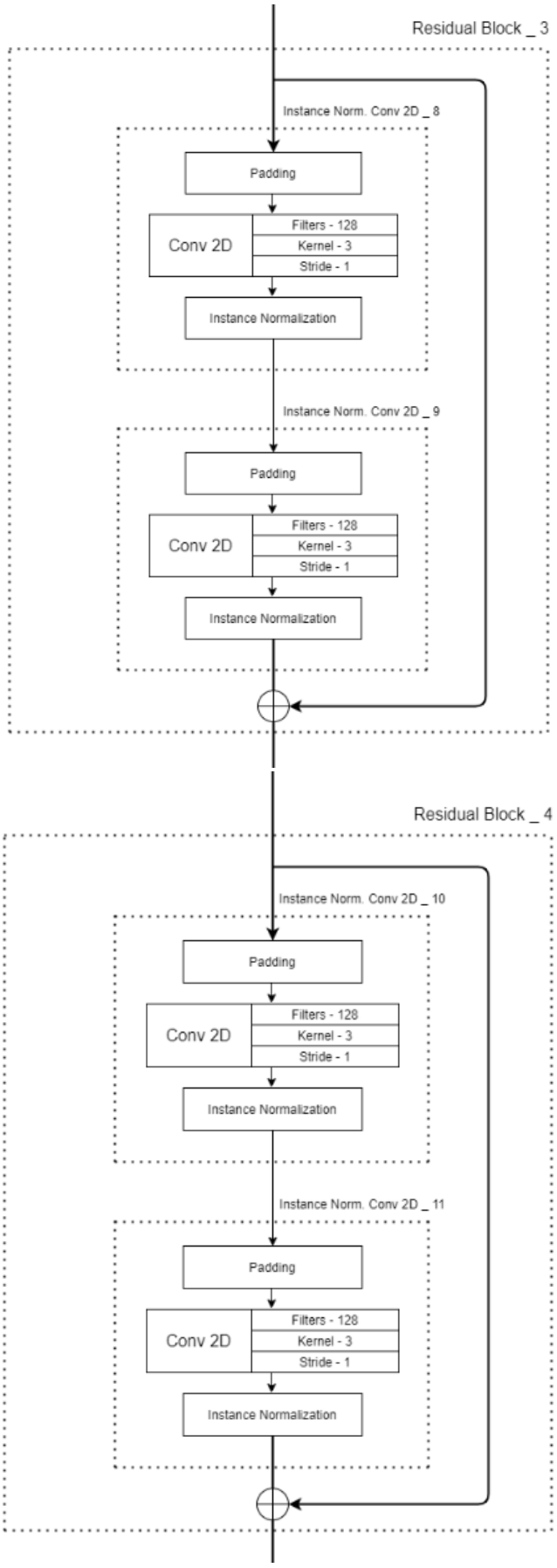
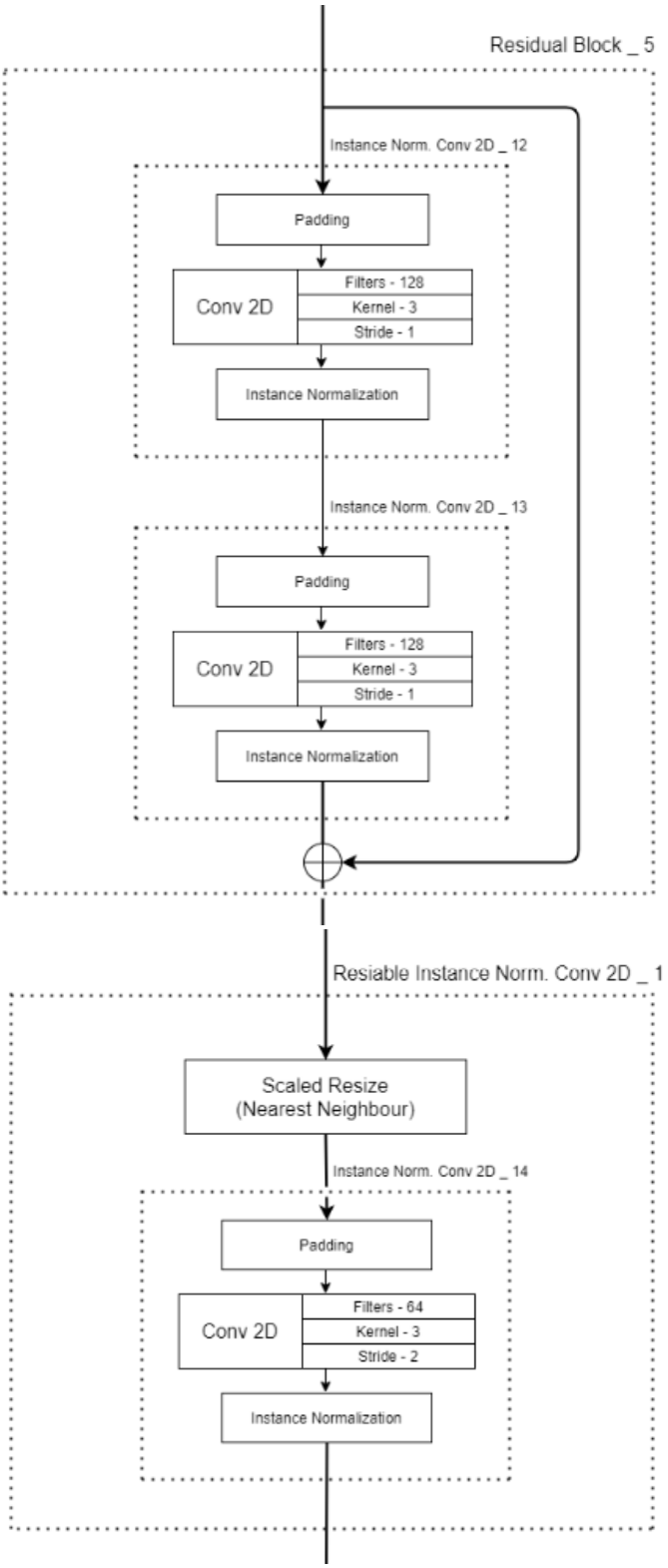
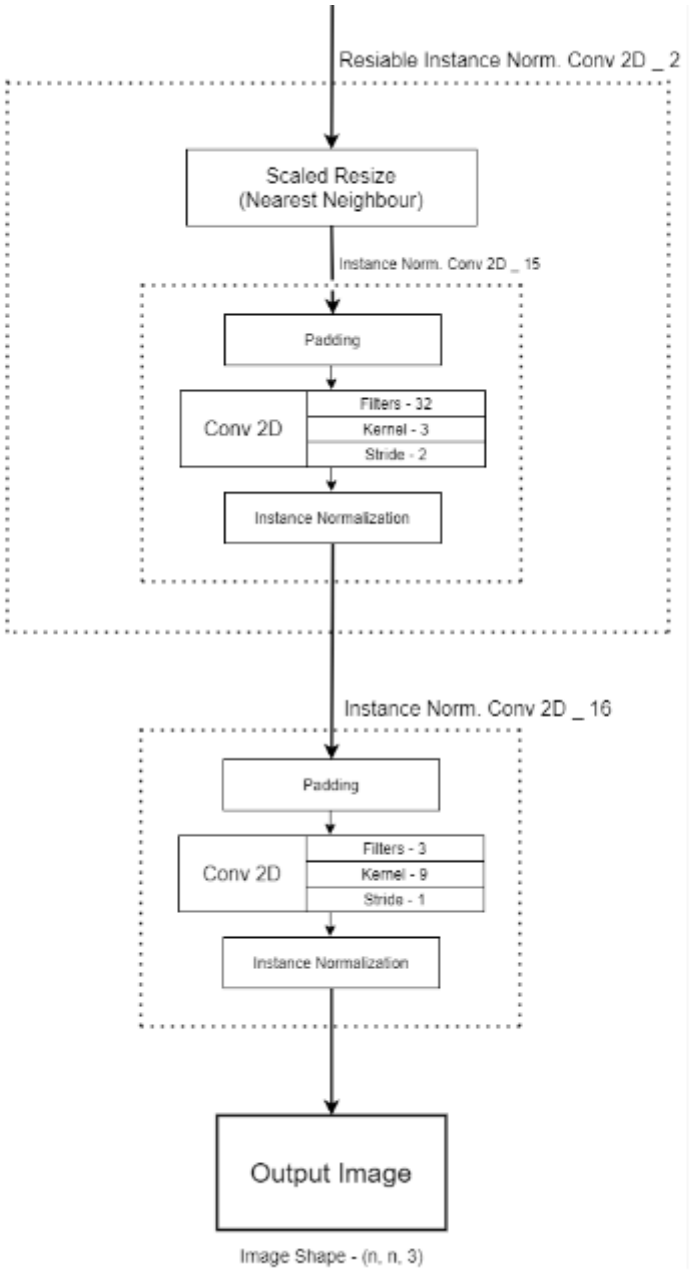


Image
Image Shape - (n, n, 3)
Instance Norm. Conv 2D _ 1
Padding
Conv 2D
Filters - 32
Kernel - 9
Stride - 1
Instance Normalization
Instance Norm. Conv 2D _ 2
Padding
Conv 2D
Filters - 64
Kernel - 3
Stride - 2
Instance Normalization
Instance Norm. Conv 2D _ 3
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 2
Instance Normalization


Residual Block _ 1
Instance Norm. Conv 2D _ 4
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Instance Norm. Conv 2D _ 5
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Residual Block _ 2
Instance Norm. Conv 2D _ 6
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Instance Norm. Conv 2D _ 7
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization

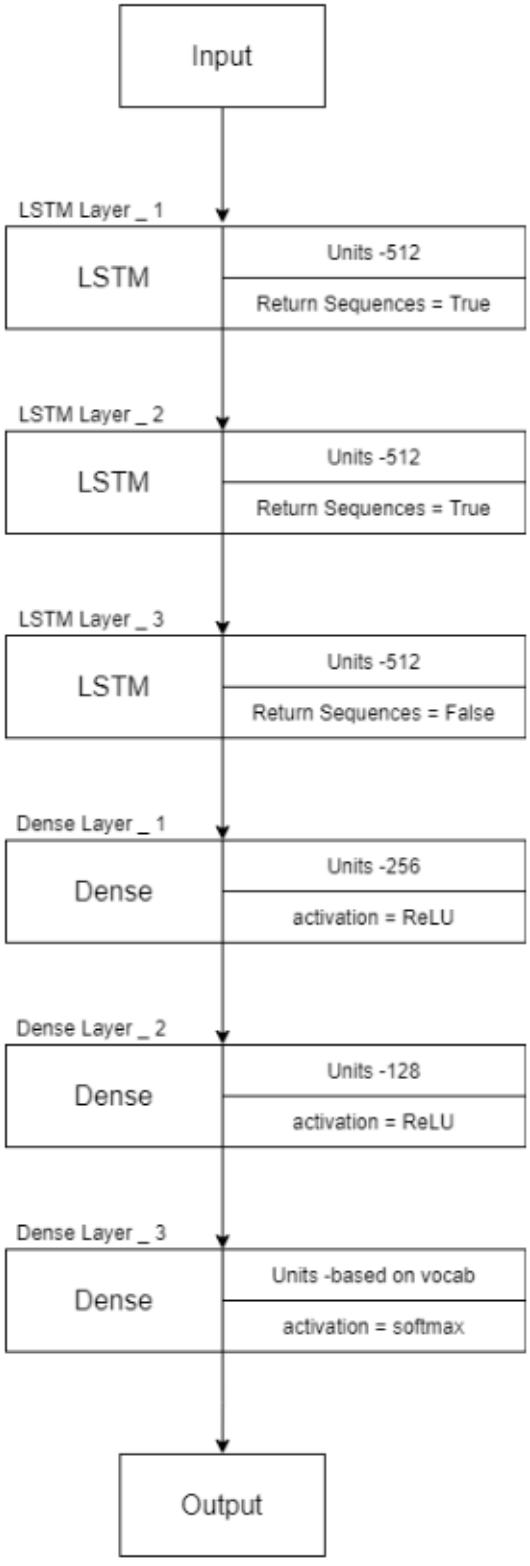

Residual Block _ 3
Instance Norm. Conv 2D _ 8
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Instance Norm. Conv 2D _ 9
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Residual Block _ 4
Instance Norm. Conv 2D _ 10
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Instance Norm. Conv 2D _ 11
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization


Residual Block _ 5
Instance Norm. Conv 2D _ 12
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Instance Norm. Conv 2D _ 13
Padding
Conv 2D
Filters - 128
Kernel - 3
Stride - 1
Instance Normalization
Resiable Instance Norm. Conv 2D _ 1
Scaled Resize
(Nearest Neighbour)
Instance Norm. Conv 2D _ 14
Padding
Conv 2D
Filters - 64
Kernel - 3
Stride - 2
Instance Normalization



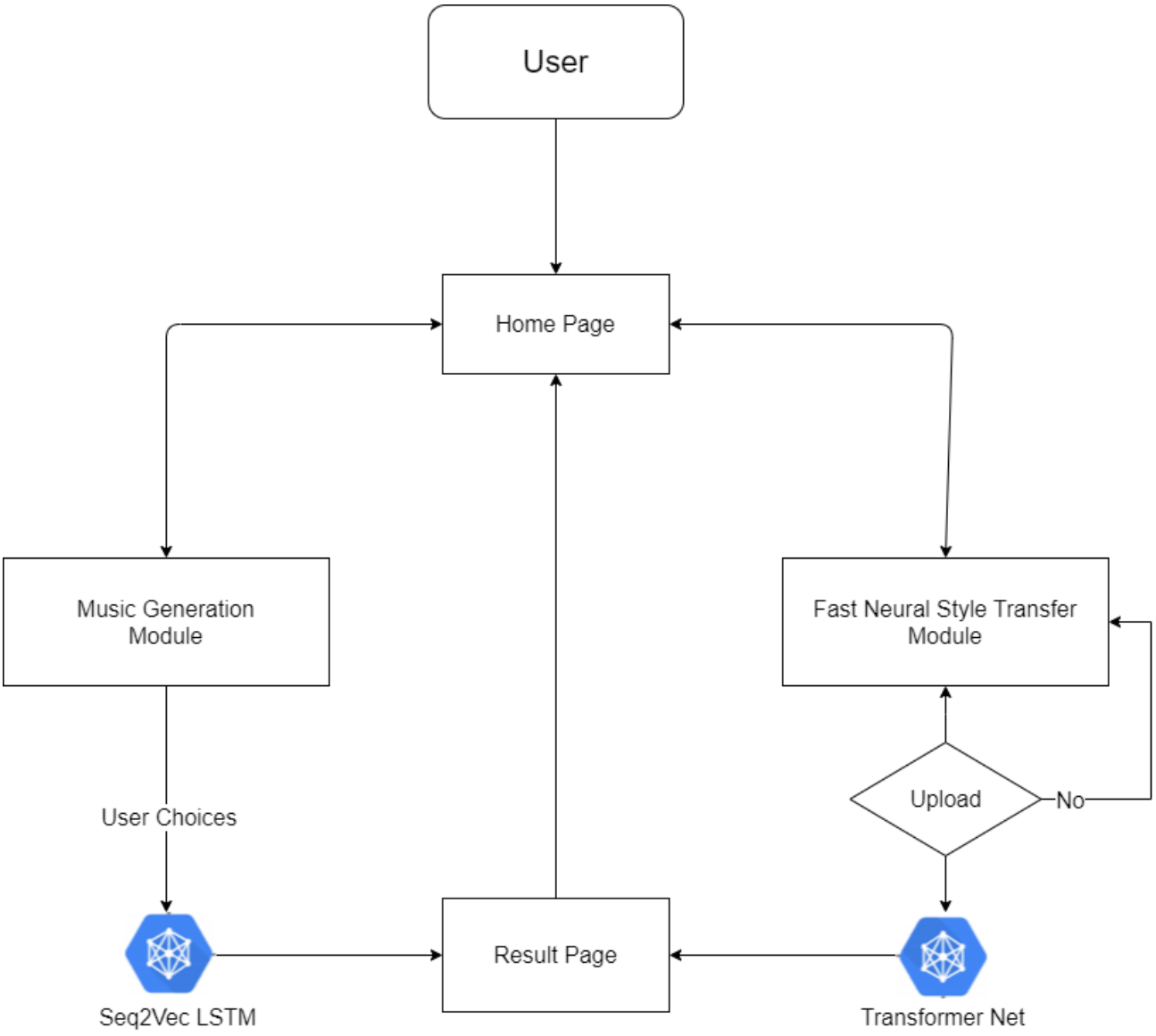
Music Generation using LSTMs

The music generation model comprises a block of 3 LSTM layers followed by 3 fully-connected dense layers. The first two LSTM returns sequences too as they need to be fed to the following layer. A sequence length of 100 notes is chosen as an input for the model and the final layer predicts the probability of notes and chords that might be played next. Softmax is chosen as the activation function for the last layer as it outputs a probability distribution.


Input
LSTM Layer _ 1
LSTM
Units -512
Return Sequences = True
LSTM Layer _ 2
LSTM
Units -512
Return Sequences = True
LSTM Layer _ 3
LSTM
Units -512
Return Sequences = False
Dense Layer _ 1
Dense
Units -256
activation = ReLU
Dense Layer _ 2
Dense
Units -128
activation = ReLU
Dense Layer _ 3
Dense
Units -based on vocab
activation = softmax
Output

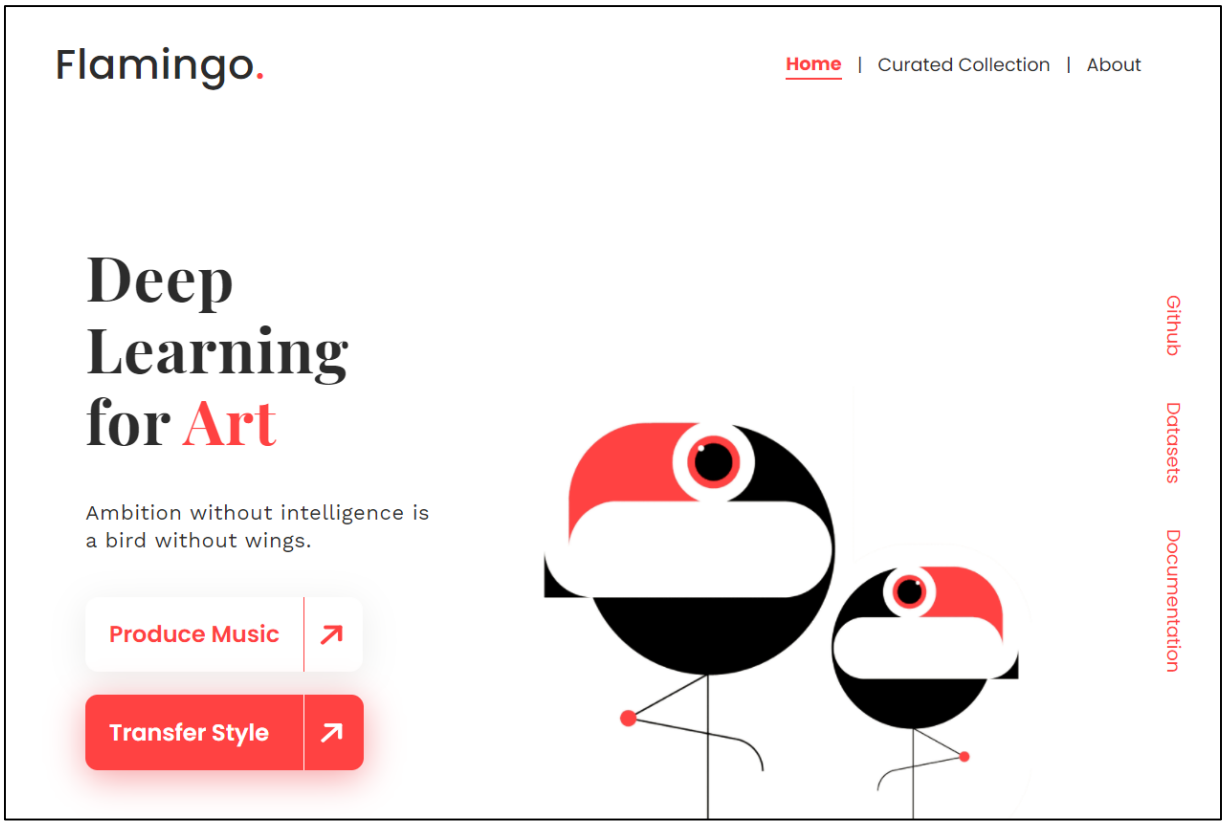
DESIGN

System Flowchart

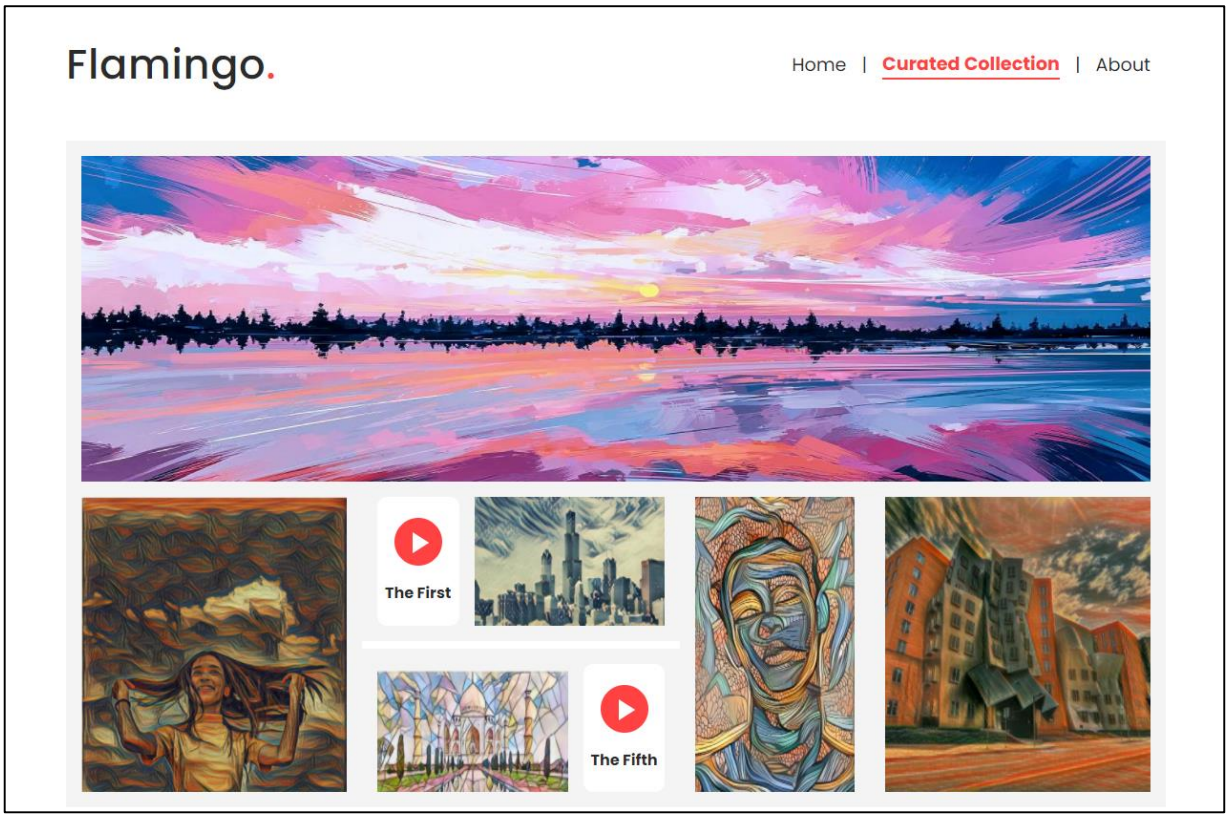


Form & Report Design Layouts

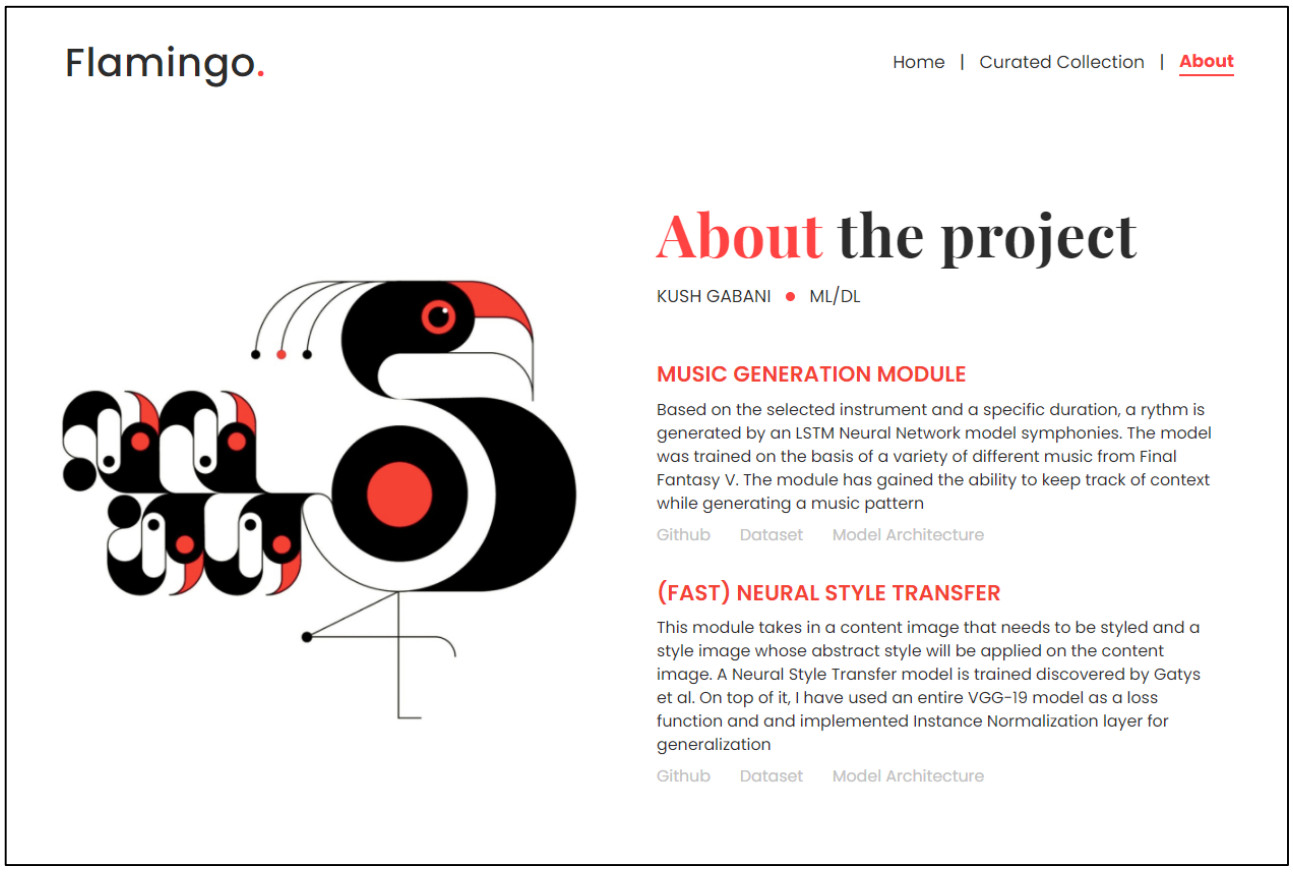
- Home Page



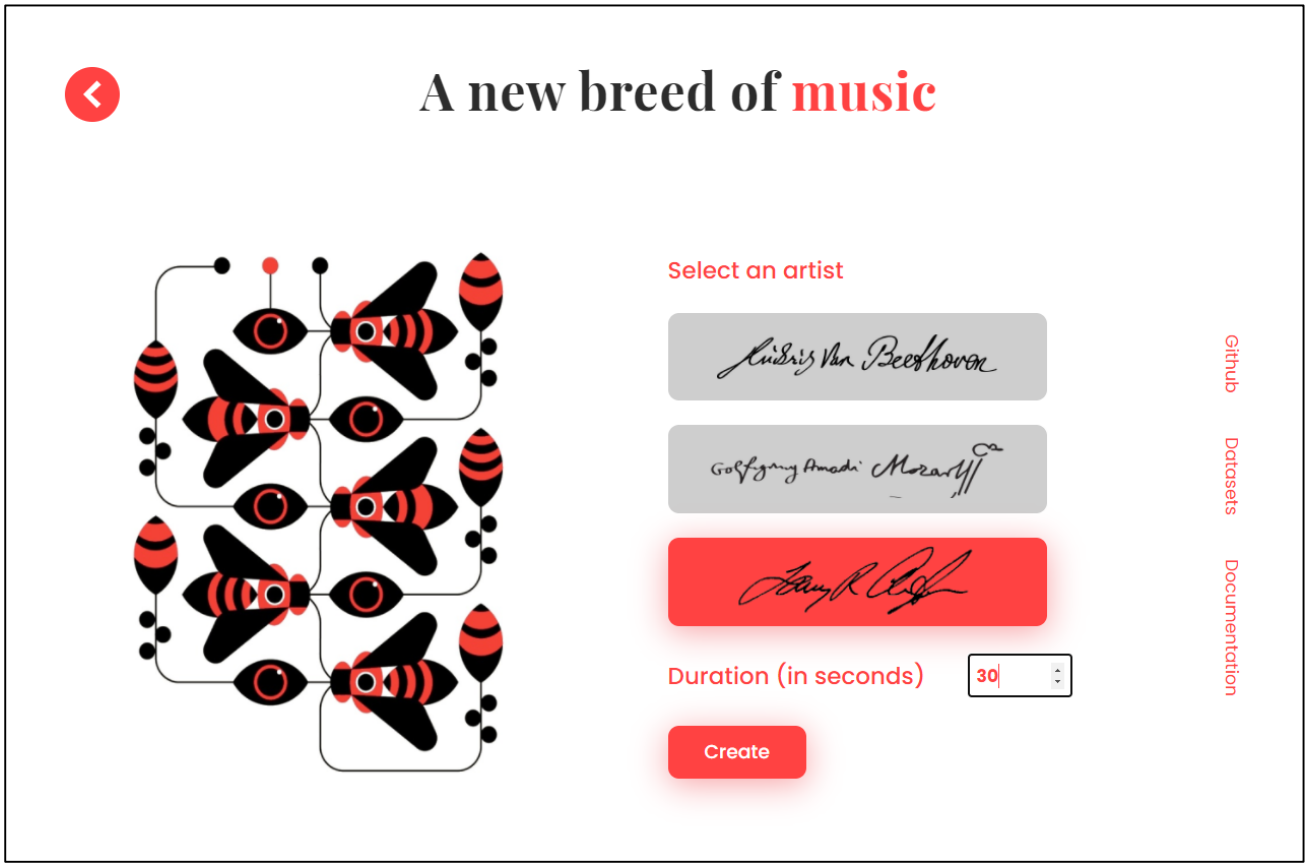
- Curated Collection Page



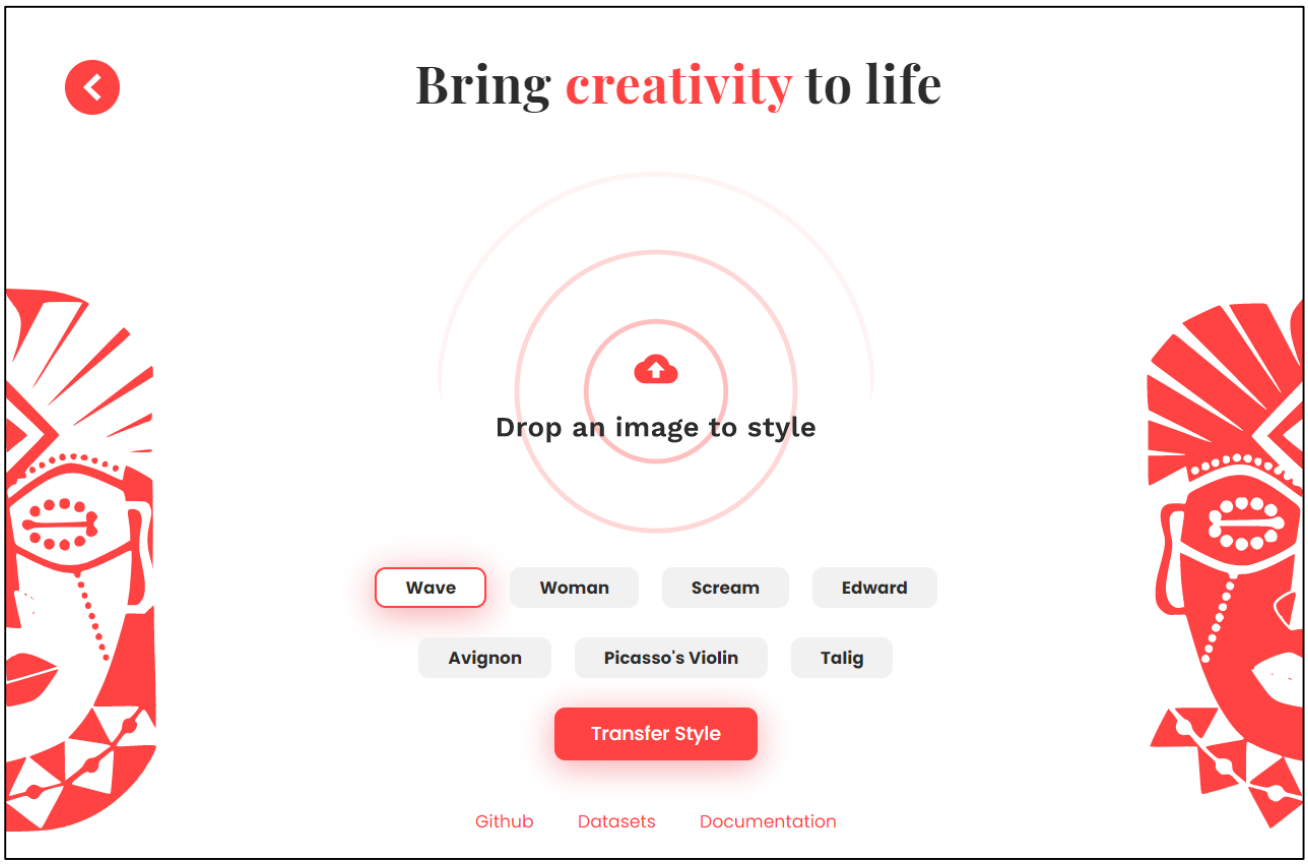
- About page



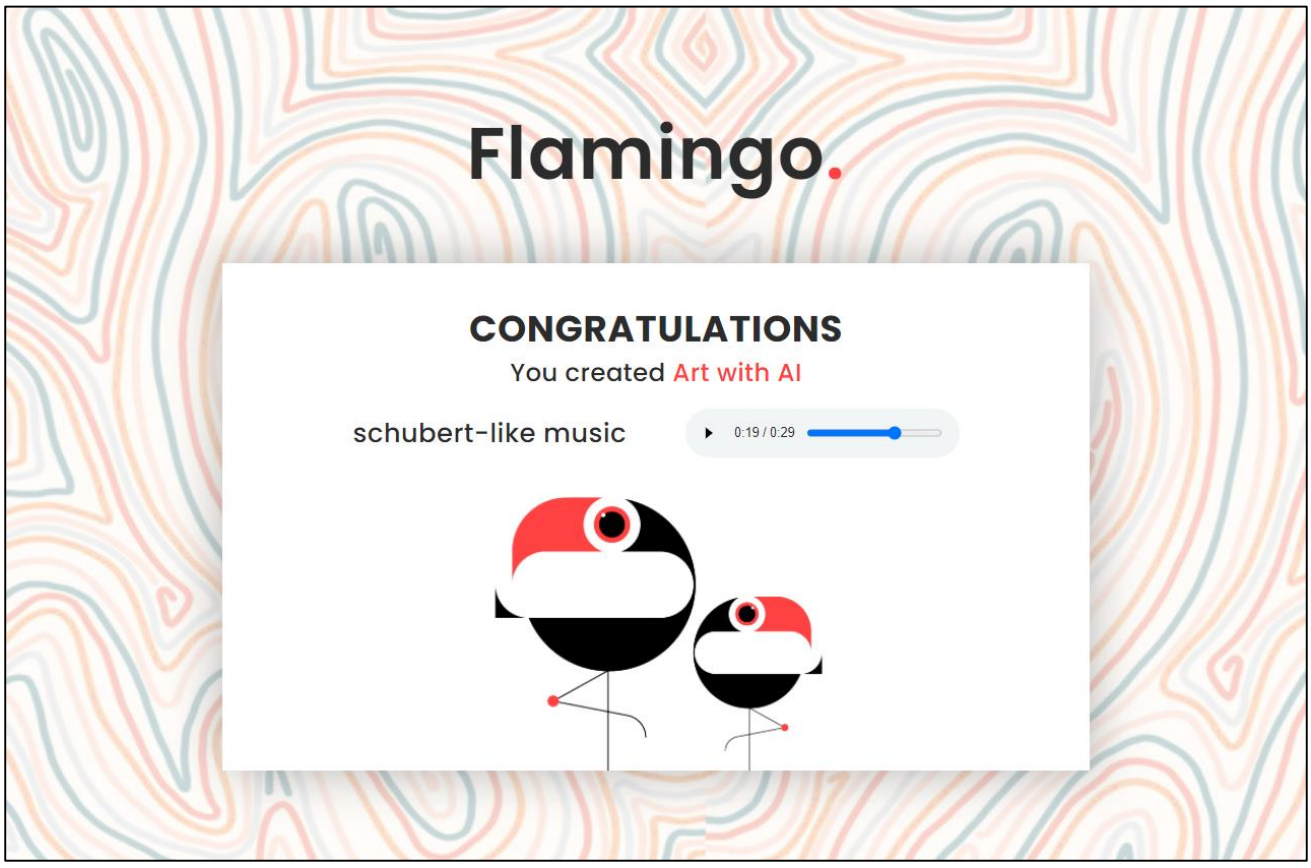
- Music Transfer Module



- Style Transfer Module



- Success Page (Music Generation)



- Success Page (Style Transfer)

CODING

Training Methodology

FAST NEURAL STYLE TRANSFER MODULE

Our system consists of two primary components: an image transformation network (Transformer Net) f_W and a loss network φ that is used to define several loss functions $l_1 \ldots, l_k$. The Transformer Net is a deep residual convolutional neural network parameterized by weights W; it transforms input images x into output images y' via the mapping $y' = f_W(x)$. Each loss function computes a scalar value $l_1(y', y_i)$ measuring the difference between the output image y' and a target image y_i. The transformer net is trained using stochastic gradient descent to minimize a weighted combination of loss functions:

$$W^* = \arg\min_W \mathbf{E}_{x,\{y_i\}} \left[\sum_{i=1} \lambda_i \ell_i(f_W(x), y_i) \right]$$

To address the shortcomings of per-pixel losses and allow our loss functions to better measure perceptual and semantic differences between images, we use CNNs that are pre-trained for image classification as they have already learned to encode the perceptual and semantic information. We, therefore, make use of a network φ pre-trained for image classification as a fixed loss network in order to define our loss functions. This pre-trained model is VGG-19. Our deep convolutional networks is thus trained using loss functions that are also deep convolutional networks. We use the loss network φ to define a feature reconstruction loss l^{φ}_{feat} and style reconstruction loss l^{φ}_{style} that measures the difference between the content and target y_c and a style target y_s . For style transfer, the content image is the input image x and the output image y' should combine the content of $x = y_c$ with the style ofy_s; we train one network per style target.

We define two perceptual loss functions that measure high-level perceptual and semantic differences between images. They make use of a VGG-19 as a loss-network φ, meaning that these perceptual loss functions are themselves deep convolutional networks. In all the training experiments, the VGG-19 that is pre-trained on ImageNet is used.

FEATURE RECONSTRUCTION LOSS:

Rather than encouraging the pixels of the output image to exactly match the pixels of the target image, we instead encourage them to have similar feature representations of the j[th] layer of the loss network. Finding an image y' that

minimizes the feature reconstruction loss for early layers tends to produce images that are visually indistinguishable from y. As we reconstruct from higher layers, image content, and overall spatial structure are preserved but color, texture, and the exact shape are not. Let $\varphi_j(x)$ be the activations of the j[th] layer of the loss network when processing the image$x$; if j is a convolutional layer then $\varphi_j(x)$ will be a feature map of shape $\mathrm{C}_j \times \mathrm{H}_j \times W_j$. The feature construction loss is the Euclidean distance between feature representations:

$$\ell_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

STYLE RECONSTRUCTION LOSS

The feature reconstruction loss penalizes the output image y' when it deviates in content from the target y. We also wish to penalize the differences in style: colours, textures, common patterns etc. To achieve this effect, Gatys et al. proposed a style reconstruction loss. As above, Let $\varphi_j(x)$ be the activations of the j[th] layer of the loss network when processing the image$x$; if j is a convolutional layer then $\varphi_j(x)$ will be a feature map of shape $\mathrm{C}_j \times \mathrm{H}_j \times W_j$. We define a Gram Matrix $G_j^{\varphi}(x)$ to be the $\mathrm{C}_j \times C_j$ matrix whose elements are given by:

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}$$

The goal of style transfer is to generate an image y' that combines the content of a target content image y_c with the help of the target style imagey_s. We train one image transformation network per style target for several hand-picked style targets. This entire process can be summarize in an equation:

$$\hat{y} = \arg\min_y \lambda_c \ell_{feat}^{\phi,j}(y, y_c) + \lambda_s \ell_{style}^{\phi,J}(y, y_s) + \lambda_{TV} \ell_{TV}(y)$$

TRAINING PROCESS

We train style transfer networks on the MS-COCO (Common objects in context) dataset. The dataset has 80,000 training images which are resized them into 256 x 256. We train it in a batch size of 2 until each image in the dataset has been put through the model at least once.
The following are the training details in a tabular format:

HYPERPARAMETERS AND CONFIGS	VALUE
Batch Size	2
Content Weight (cw)	6×10^{0}
Style Weight (sw)	2×10^{-3}
Total Variation Weight (tv)	6×10^{2}
Total Loss	cw * content loss + sw * style loss + tv * total variation loss
Epochs	2
Optimizer	Adam
Learning Rate	1×10^{-3}
Layers of VGG-19 used for content	block4_conv2

Layers of VGG-19 used for style	{
	block1_conv1,
	block2_conv1,
	block3_conv1,
	block4_conv1,
	block5_conv1,
	}

MUSIC GENERATION MODULE

The dataset contains of music file in a midi format. Midi is a technical standard that describes a communication protocol, digital interface, and electrical connectors that connectors that connect a wide variety of electronic, musical instruments, computers and related audio devices. In the Music Generation Module, a Seq2Vector LSTM model is trained. A sequence-to-vector model is a neural network that takes a sequence of numbers as an input and outputs a fix sized vector. A module named music21 is used to read these midi files into a list of Chord and Note objects. All the unique chords and notes are tokenized, and a sliding frames with a sequence length of 100 are created as training data. Suppose, a training row is 0-100 and its label is 101. Then the next training row will be 1-100 and its label will be 102.

HYPERPARAMETERS AND CONFIGS	VALUE

Epochs	100
Batch Size	128
Sequence Length	100
Optimizer	Adam
Learning Rate	3×10^{-4}
Loss Function	Categorical Cross entropy

Inference Methodology

FAST NEURAL STYLE TRANSFER MODULE

After training, the weights of the Transformer Net are saved. We save the weights of the model and not the model architecture because if we save the architecture, the images are then needed to be resized to 256 x 256 x 3. As our transformer net is a deep ConvNets, we just save the weights of the filters which can be applied on image of arbitrary size. Thus allowing us to input and transform images of any resolution. During inference, the image is uploaded by the user by which the respective style transfer model architecture is built and loaded with their pre-trained weights. Along with the uploaded image, the user also has to select the style photo which needs to be applied on the content image. The content image is read into a numpy array and is put through the trained transformer net. The Transformer Net takes the content image and proceeds to the feed-forward mechanism to generate an output. The image is passed through several residual convolution layers after which the output image is returned back. This output image is our content image that is styled on the basis of the style image. This is then saved on the disk and viewed to the user.

MUSIC GENERATION MODULE

In the music module, we take two inputs: the name of the artist whose music style needs to be generated and its duration in seconds. This input data is first collected by a form. The duration is first mapped from seconds to the number of notes that needs to be generated. Then the pre-processed data of the respective artist and their pitch range is loaded from disk. Along with these, the said “vocabulary” is also returned. Then the model architecture is built and the respective trained weights of the artist is loaded from disk. During inference, a random note sequence from the artist is selected which is taken by the model as a context. The LSTM model outputs a vector of probabilities of note that might be played. The note with the highest probability is selected and is mapped from integer back to a string. Using music21 library, we build a sequence of these output notes and convert them from text to midi audio files. The generated midi audio file is saved to the disk which is then viewed to the user in the output screen.

Coding Snippets

Fast Style Transfer.ipynb – The source code that is used to build and train a fast style transfer network.

```
%tensorflow_version 2.x
import IPython.display as display
import time
import os
import shutil
import cv2
import PIL.Image

import tensorflow as tf
import numpy as np

from keras.utils.vis_utils import plot_model

try:
    shutil.rmtree("./sample_data")
except:
    pass

def tensor_to_image(tensor):
    tensor = np.array(tensor, dtype=np.uint8)
    if np.ndim(tensor) > 3:
        assert tensor.shape[0] == 1
        tensor = tensor[0]
    return PIL.Image.fromarray(tensor)

def load_img(path, max_dim=None, resize=True):
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)

    if resize:
        new_shape = tf.cast([256, 256], tf.int32)
        img = tf.image.resize(img, new_shape)

    if max_dim:
        shape = tf.cast(tf.shape(img)[:-1], tf.float32)
        long_dim = max(shape)
        scale = max_dim / long_dim
        new_shape = tf.cast(shape * scale, tf.int32)
        img = tf.image.resize(img, new_shape)

    img = img[tf.newaxis, :]

    return img

def resolve_video(network, path, result):
    cap = cv2.VideoCapture(path)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWrite(result, fourcc, 30.0, (640, 640))

    while cap.isOpened():
        ret, frame = cap.read()

        print('Transferring Video.......')
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RBG)
```

```
        frame = tf.cast(frame[tf.newaxis, ...], tf.float32) / 255.0

        prediction = network(frame)

        prediction = clip_0_1(prediction)
        prediction = np.array(prediction).astype(np.uint8).squeeze()
        prediction = cv2.cvtColor(prediction, cv2.COLOR_BG2BGR)

        if cv2.waitKey(1) & 0xFF == org('q'):
            break

    cap.release()
    out.release()
    cv2.destroyALLWindow()

def create_folder(dirname):
    if not os.path.exists(dirname):
        os.mkdir(dirname)
        print('Directory ', dirname, ' created')
    else:
        print('Directory', dirname, ' already exists')

def clip_0_1(image):
    return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=255.0)

class InstanceNormalization(tf.keras.layers.Layer):
    def __init__(self, epsilon=1e-3):
        super(InstanceNormalization, self).__init__()
        self.epsilon = epsilon

    def build(self, input_shape):
        self.beta = tf.Variable(tf.zeros([input_shape[3]]))
        self.gamma = tf.Variable(tf.ones([input_shape[3]]))

    def call(self, inputs):
        mean, var = tf.nn.moments(inputs, axes=[1, 2], keepdims=True)
        x = tf.divide(tf.subtract(inputs, mean), tf.sqrt(tf.add(var, self.epsilon
)))

        return self.gamma * x + self.beta

class InstanceNormConv2D(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(InstanceNormConv2D, self).__init__()
        pad = kernel // 2
        self.padding = tf.constant([[0, 0], [pad, pad], [pad, pad], [0, 0]])
        self.conv = tf.keras.layers.Conv2D(filters, kernel, stride, use_bias=Fals
e, padding='valid')
        self.instance_norm = InstanceNormalization()

    def call(self, inputs, relu=True):
        x = tf.pad(inputs, self.padding, mode='REFLECT')
        x = self.conv(x)
        x = self.instance_norm(x)

        if relu:
            x = tf.keras.layers.Activation("relu")(x)
        return x--

class ResizableConv2D(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
```

```
        super(ResizableConv2D, self).__init__()
        self.conv = InstanceNormConv2D(filters, kernel, stride)
        self.stride = stride

    def call(self, inputs):
        height = inputs.shape[1] * self.stride * 2
        width = inputs.shape[2] * self.stride * 2
        x = tf.image.resize(inputs, [height, width], method=tf.image.ResizeMethod
.NEAREST_NEIGHBOR)
        x = self.conv(x)
        return x

class InstanceNormConv2DTranspose(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(InstanceNormConv2DTranspose, self).__init__()
        self.conv_transpose = tf.keras.layers.Conv2DTranspose(filters, kernel, st
ride, padding="same")
        self.instance_norm = InstanceNormalization()

    def call(self, inputs):
        x = self.conv_transpose(inputs)
        x = self.instance_norm(x)
        return tf.keras.layers.Activation("relu")

class Residual(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(Residual, self).__init__()
        self.conv1 = InstanceNormConv2D(filters, kernel, stride)
        self.conv2 = InstanceNormConv2D(filters, kernel, stride)

    def call(self, inputs):
        x = self.conv1(inputs)
        return inputs + self.conv2(x, relu=False)

class TransformerNet(tf.keras.models.Model):
    def __init__(self):
        super(TransformerNet, self).__init__()
        self.conv1 = InstanceNormConv2D(filters=32, kernel=9, stride=1)
        self.conv2 = InstanceNormConv2D(filters=64, kernel=3, stride=2)
        self.conv3 = InstanceNormConv2D(filters=128, kernel=3, stride=2)

        self.res1 = Residual(filters=128, kernel=3, stride=1)
        self.res2 = Residual(filters=128, kernel=3, stride=1)
        self.res3 = Residual(filters=128, kernel=3, stride=1)
        self.res4 = Residual(filters=128, kernel=3, stride=1)
        self.res5 = Residual(filters=128, kernel=3, stride=1)

        self.resize_conv1 = ResizableConv2D(filters=64, kernel=3, stride=2)
        self.resize_conv2 = ResizableConv2D(filters=32, kernel=3, stride=2)
        self.conv4 = InstanceNormConv2D(filters=3, kernel=9, stride=1)

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.res1(x)
        x = self.res2(x)
        x = self.res3(x)
        x = self.res4(x)
        x = self.res5(x)
        x = self.resize_conv1(x)
```

```
        x = self.resize_conv2(x)
        x = self.conv4(x, relu=False)

        return (tf.keras.activations.tanh(x) * 150 + 255.0 / 2)

CONTENT_WEIGHT = 6e0
STYLE_WEIGHT = 2e-3
TV_WEIGHT = 6e2
LEARNING_RATE = 1e-3
NUM_EPOCHS = 2
BATCH_SIZE = 2

DATASET_PATH = "./drive/MyDrive/COCO-dataset/train2014"
MODEL_PATH = "./drive/MyDrive/Fast-Style-Transfer/models/scream/model/"
WEIGHTS_PATH = "./drive/MyDrive/Fast-Style-Transfer/models/scream/weights/"
STYLE_IMAGE = "./drive/MyDrive/Fast-Style-Transfer/images/style/the_scream.jpg"
RESULT_NAME = "./drive/MyDrive/Fast-Style-Transfer/Result.jpg"

def vgg_layers(layer_names):
    vgg = tf.keras.applications.vgg19.VGG19(include_top=False, weights='imagenet'
)
    vgg.trainable = False

    outputs = [vgg.get_layer(layer).output for layer in layer_names]
    model = tf.keras.Model([vgg.input], outputs)
    return model

def gram_matrix(feature_map, normalize=True):
    gram = tf.linalg.einsum('bijc,bijd->bcd', feature_map, feature_map)
    if normalize:
        input_shape = tf.shape(feature_map)
        gram /= tf.cast(input_shape[1] * input_shape[2], tf.float32)

    return gram

def style_loss(style_outputs, style_target):
    loss = tf.add_n([tf.reduce_mean((style_outputs[name] - style_target[name]) **
 2) for name in style_outputs.keys()])
    return loss

def content_loss(content_outputs, content_target):
    loss = tf.add_n([tf.reduce_mean((content_outputs[name] - content_target[name]
) ** 2) for name in content_outputs.keys()])
    return loss

def total_variation_loss(img):
    x = img[:, :, 1:, :] - img[:, :, :-1, :]
    y = img[:, 1:, :, :] - img[:, :-1, :, :]

    return tf.reduce_mean(tf.square(x)) + tf.reduce_mean(tf.square(y))

class VGG19LossModel(tf.keras.models.Model):
    def __init__(self, style_layers, content_layers):
        super(VGG19LossModel, self).__init__()
        self.vgg = vgg_layers(style_layers + content_layers)
        self.style_layers = style_layers
        self.content_layers = content_layers
        self.num_style_layers = len(style_layers)
        self.num_content_layers = len(content_layers)
        self.vgg.trainable = False
```

```
    def call(self, inputs):
        preprocessed_input = tf.keras.applications.vgg19.preprocess_input(inputs)
        outputs = self.vgg(preprocessed_input)
        style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                          outputs[self.num_style_layers:])


        style_outputs = [gram_matrix(feature_map) for feature_map in style_output
s]

        style_dict = {
            style_name: value for style_name, value in zip(self.style_layers, sty
le_outputs)
        }

        content_dict = {
            content_name: value for content_name, value in zip(self.content_layer
s, content_outputs)
        }

        return {
            'content' : content_dict,
            'style': style_dict
        }

train_dataset = tf.data.Dataset.list_files(DATASET_PATH + '/*.jpg')
train_dataset = train_dataset.map(load_img, num_parallel_calls = tf.data.experime
ntal.AUTOTUNE)
train_dataset = train_dataset.shuffle(1024)
train_dataset = train_dataset.batch(BATCH_SIZE, drop_remainder=True)
train_dataset = train_dataset.prefetch(tf.data.experimental.AUTOTUNE)

# THE WAVE.jpg

content_layers = ['block4_conv2']
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

transformer_net = TransformerNet()
transformer_net.load_weights(WEIGHTS_PATH)
extractor = VGG19LossModel(style_layers, content_layers)

style_image = load_img(STYLE_IMAGE, resize = False)
input_shape = (BATCH_SIZE, 256, 256, 3)

X_batch = np.zeros(input_shape, dtype=np.float32)

style_target = extractor(style_image * 255.0)['style']

optimizer = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)

loss_metric = tf.keras.metrics.Mean()
sloss_metric = tf.keras.metrics.Mean()
closs_metric = tf.keras.metrics.Mean()
tvloss_metric = tf.keras.metrics.Mean()

@tf.function()
```

```
def train_step(X_batch):
    with tf.GradientTape() as tape:
        content_target = extractor(X_batch * 255.0)['content']
        image = transformer_net(X_batch)
        outputs = extractor(image)

        sloss = STYLE_WEIGHT * style_loss(outputs['style'], style_target)
        closs = CONTENT_WEIGHT * content_loss(outputs['content'], content_target)
        tvloss = TV_WEIGHT * total_variation_loss(image)
        loss = sloss + closs + tvloss

    gradients = tape.gradient(loss, transformer_net.trainable_variables)
    optimizer.apply_gradients(zip(gradients, transformer_net.trainable_variables)
)

    loss_metric(loss)
    sloss_metric(sloss)
    closs_metric(closs)
    tvloss_metric(tvloss)

start = time.time()
for epoch in range(1, NUM_EPOCHS + 1):
    print(f'Epoch {epoch}/{NUM_EPOCHS}')
    iteration = 0
    progress_bar = tf.keras.utils.Progbar(None, interval=1, unit_name='iteration'
)
    for image in train_dataset:
        for j, img_p in enumerate(image):
            X_batch[j] = img_p

        iteration += 1
        train_step(X_batch)
        progress_bar.add(iteration,
                         [('loss', loss_metric.result()),
                          ('style_loss', sloss_metric.result()),
                          ('content_loss', closs_metric.result()),
                          ('tv_loss', tvloss_metric.result())])

        if iteration % 3000 == 0:
            transformer_net.save_weights(WEIGHTS_PATH, save_format='tf')
            print(f'\n\nstep {iteration}: loss = {loss_metric.result()}')
            print(f'style_loss={sloss_metric.result()}, content_loss={closs_metri
c.result()}, tv_loss={tvloss_metric.result()}')

end = time.time()
print("Time: {:.1f}".format(end - start))

transformer_net.save_weights(WEIGHTS_PATH, save_format='tf')
print("Model Trained!")

model = TransformerNet()
model.load_weights("./drive/MyDrive/Fast-Style-Transfer/models/wave/weights/")

image_type = ('jpg', 'jpeg', 'png')
CONTENT_PATH = "./drive/MyDrive/Fast-Style-Transfer/images/content/chicago.jpg"

image = load_img(CONTENT_PATH, resize=False)
image.shape
if CONTENT_PATH[-3:] in image_type:
  image = load_img(CONTENT_PATH, resize = False)
  image = model(image)
```

```
  image = clip_0_1(image)
  output = tensor_to_image(image)
```

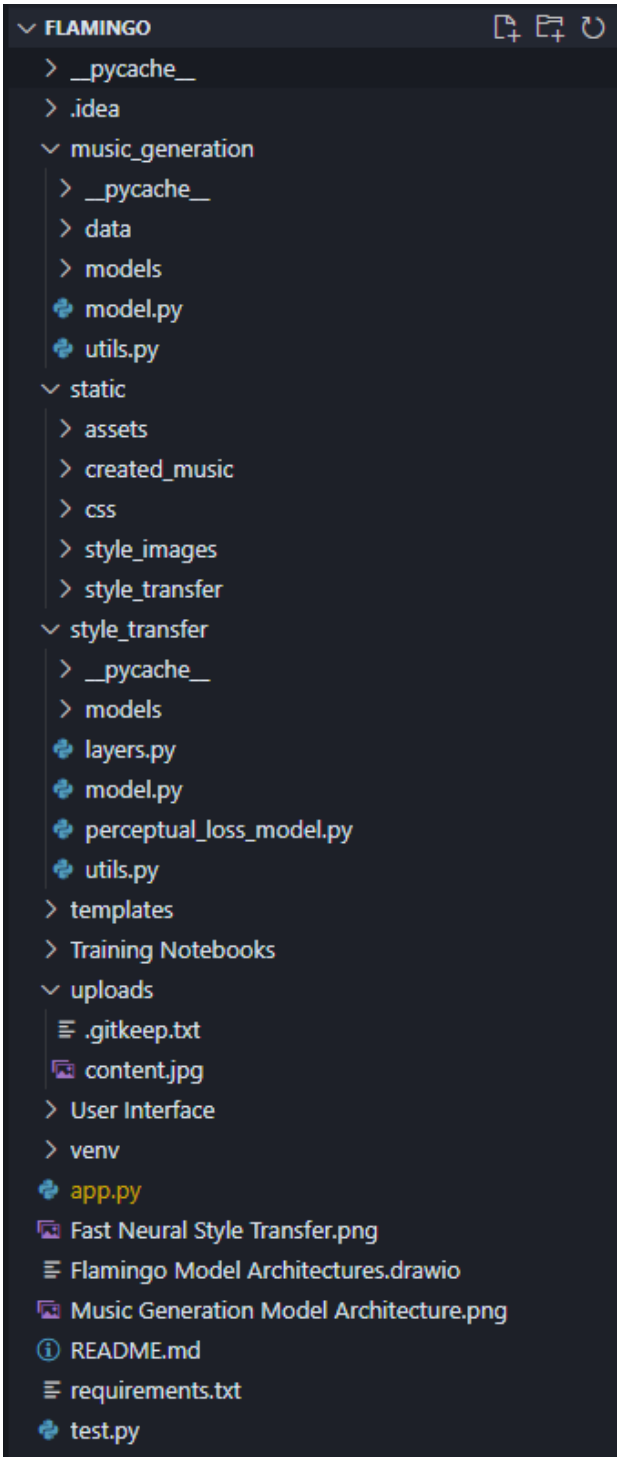
Music Generation Using LSTMs.ipynb – The source code that is used to build and train a Seq2Vec LSTM to generate Music.

```
%tensorflow_version 2.x
%load_ext tensorboard
import shutil
import music21
import glob

import tensorflow as tf
from tensorflow.keras import layers
import numpy as np

try:
    shutil.rmtree("./sample_data")
except:
    pass

notes = []

def load_songs(path):
    for file in glob.glob(path):
        midi = music21.converter.parse(file)
        notes_to_parse = None

        parts = music21.instrument.partitionByInstrument(midi)

        if parts:
            notes_to_parse = parts.parts[0].recurse()
        else:
            notes_to_parse = midi.flat.notes

        for element in notes_to_parse:
            if isinstance(element, music21.note.Note):
                notes.append(str(element.pitch))
            elif isinstance(element, music21.chord.Chord):
                notes.append('.'.join(str(n) for n in element.normalOrder))

load_songs("./classical_music/schubert/*.mid")

sequence_length = 100
pitchnames = sorted(set(item for item in notes))

notes_to_int = dict((note, number) for number, note in enumerate(pitchnames))
notes_to_int

x = []
y = []
for i in range(0, len(notes) - sequence_length, 1):
    seq_in = notes[i:i + sequence_length]
    seq_out = notes[i + sequence_length]
    x.append([notes_to_int[char] for char in seq_in])
    y.append(notes_to_int[seq_out])

n_patterns = len(x)
```

```
n_vocab = len(pitchnames)
x = np.expand_dims(x, axis = 2)
x = x / float(n_vocab)
y = tf.keras.utils.to_categorical(y)

np.savez_compressed("./drive/MyDrive/Music-
Generation/schubert_processed.npz", x, y)
np.savez_compressed("./drive/MyDrive/Music-
Generation/schubert_pitchnames.npz", pitchnames)
# data = np.load("/content/drive/MyDrive/Music-Generation/beeth_processed.npz")
# x, y = data['arr_0'], data['arr_1']
# n_patterns = len(x)

# data = np.load("/content/drive/MyDrive/Music-Generation/beeth_pitchnames.npz")
# pitchnames = data['arr_0']
# n_vocab = len(pitchnames)

model = tf.keras.Sequential()
model.add(layers.LSTM(512, input_shape=(x.shape[1], x.shape[2]), return_sequences
=True, recurrent_dropout=0))
model.add(layers.LSTM(512, return_sequences=True, recurrent_dropout=0))
model.add(layers.LSTM(512))
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(n_vocab, activation="softmax"))

model.summary()

# model.load_weights("./drive/MyDrive/Music-Generation/model/Classical-
Model/Mozart/w-175.h5")
model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=3e-4))

def generate_sequence():
    start = np.random.randint(0, len(x) - 1)
    int_to_note = dict((number, note) for number, note in enumerate(pitchnames))

    pattern = x[start]
    prediction_output = []

    for ni in range(150):
        prediction_input = np.reshape(pattern, (1, len(pattern), 1))
        prediction_input = prediction_input / float(n_vocab)

        prediction = model.predict(prediction_input, verbose=0)

        index = np.argmax(prediction)
        result = int_to_note[index]
        prediction_output.append(result)

        pattern = np.append(pattern, index)
        pattern = pattern[1:len(pattern)]
    return prediction_output

def generate_notes(predictions):
    offset = 0
    output_notes = []

    for pattern in predictions:
```

```
        # Pattern is chord
        if pattern.isdigit() or '.' in pattern:
            notes_in_chord = pattern.split(".")
            notes= []

            for current_note in notes_in_chord:
                new_note = music21.note.Note(int(current_note))
                new_note.storedInstrument = music21.instrument.Violin()
                notes.append(new_note)
            new_chord = music21.chord.Chord(notes)
            new_chord.offset = offset
            output_notes.append(new_chord)

        # Pattern is a note
        else:
            new_note = music21.note.Note(pattern)
            new_note.offset = offset
            new_note.storedInstrument = music21.instrument.Violin()
            output_notes.append(new_note)

        offset += 0.5
    return output_notes

def save_midi(notes, artist, id=1):
    midi_stream = music21.stream.Stream(notes)
    midi_stream.write("midi", fp=f'./drive/MyDrive/Music-Generation/Classical-
Output/schubert/{artist}-{id}.mid')

log_writer = tf.summary.create_file_writer("./drive/MyDrive/Music-
Generation/logs/Classical-Model/schubert_model_unreg")
EPOCHS = 100
CURRENT_EPOCH = 1
for epoch in range(CURRENT_EPOCH, EPOCHS + 1):
    print(f"{epoch}/{EPOCHS}")
    history = model.fit(x, y,
                        epochs=1,
                        batch_size=128)

    with log_writer.as_default():
        tf.summary.scalar('loss', data=history.history['loss'][0], step=epoch)

    if epoch % 25 == 0:
        model.save(f"./drive/MyDrive/Music-Generation/model/Classical-
Model/Schubert/unreg_m-{epoch}.h5", save_format='h5')
        model.save_weights(f"./drive/MyDrive/Music-Generation/model/Classical-
Model/Schubert/unreg_w-{epoch}.h5", save_format='h5')

        prediction_output = generate_sequence()
        notes = generate_notes(prediction_output)
        save_midi(notes, "Schubert", epoch)

prediction_output = generate_sequence()
notes = generate_notes(prediction_output)
save_midi(notes, "schubert", "1")
```

Project Directory Structure



./app.py – Flask API source code used in the backend for the interface.

```
import os
import warnings
from flask_dropzone import Dropzone
from flask import Flask, url_for, render_template, request
from music_generation import utils as music_utils
from music_generation.model import MusicGenerationModel
from style_transfer import utils as nst_utils
from style_transfer.model import TransformerNet
warnings.filterwarnings("ignore")

basedir = os.path.abspath(os.path.dirname(__file__))
```

```
app = Flask(__name__)

app.config.update(
    UPLOADED_PATH=os.path.join(basedir, 'uploads'),
    DROPZONE_ALLOWED_FILE_TYPE='image',
    DROPZONE_MAX_FILES=1,
)

dropzone = Dropzone(app)


def generate_music(num_notes, artist):
    print("-----------------process: generating music")
    x, pitchnames, n_vocab = music_utils.get_data(artist)
    mgModel = MusicGenerationModel(artist, n_vocab, (x.shape[1], x.shape[2]))
    mgModel.init_model_architecture()
    model = mgModel.load_model_weights()
    prediction_output = music_utils.generate_sequence(
        model, num_notes, x, pitchnames, n_vocab)
    music = music_utils.generate_notes(prediction_output)
    filename = music_utils.save_midi(music, artist)
    print("-----------------process: end")

    return filename


def transfer_style(content_image, style_image):
    print("-----------------process: getting images")
    nstModel = TransformerNet()
    output_filepath = nst_utils.neural_style_transfer(
        nstModel, content_image, style_image)
    print("-----------------process: end")
    return output_filepath


@app.route("/", methods=['GET'])
def home():
    return render_template("home.html")


@app.route("/about", methods=['GET'])
def about():
    return render_template("about.html")


@app.route("/music-generation", methods=['GET'])
def music_generation():
    return render_template("music_generation.html")


@app.route("/style-transfer", methods=['GET'])
def style_transfer():
    return render_template("style_transfer.html")


@app.route("/upload-content", methods=['POST'])
def upload_content_image():
    print(request.files)
```

```
    for key, f in request.files.items():
        if key.startswith('file'):
            f.save(os.path.join(app.config['UPLOADED_PATH'], "content.jpg"))
    return ""


@app.route("/curated-collection", methods=['GET'])
def collection():
    return render_template("curated_collection.html")


@app.route("/handle-music-form", methods=['GET', 'POST'])
def success1():
    if request.method == 'POST':
        try:
            duration = int(request.form.get("duration"))
            artist = request.form.get("artist")
            num_notes = int((duration * 150) / 38)
            print(f"duration: {duration}, type: {type(duration)}")
            print(f"artist: {artist}, type: {type(artist)}")
            print(f"num_notes: {num_notes}, type: {type(num_notes)}")
            filename = generate_music(num_notes, artist)
            return render_template("success.html", artist=artist, audio_src=filename, is
_music_module="true")
        except:
            return "<div style='width: 100%; height: 100%; display:flex; justify-
content:center; align-
items: center'><h1 style='color:#ff4242;'>You didn't provide the needed information</h1>
</div>"
    else:
        return render_template("success.html", artist="Schubert", audio_src='created_mus
ic/schubert.mid', is_music_module="true")


@app.route("/handle-transfer-form", methods=['GET', 'POST'])
def success2():
    if request.method == 'POST':
        # try:
        style_filepath = f"static/style_images/{request.form.get('style-image')}.jpg"
        filename = transfer_style("uploads/content.jpg", style_filepath)
        return render_template("success.html", image_src=filename, is_music_module="fals
e")
        # except:
        #     return "<div style='width: 100%; height: 100%; display:flex; justify-
content:center; align-
items: center'><h1 style='color:#ff4242;'>You didn't provide the needed information</h1>
</div>"
    else:
        return render_template("success.html", image_src='style_transfer/output.png', is
_music_module="false")


if __name__ == "__main__":
    app.run(debug=True)
```

./music-generation/model.py – Source code to build the LSTM model before loading its weight.

```
import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense, BatchNormalization, Dropout
from tensorflow.keras import Sequential
from tensorflow.python.keras.engine.training import Model

class MusicGenerationModel:
    def __init__(self, artist, n_vocab, input_shape):
        self.model = None
        self.artist = artist
        self.vocab = n_vocab
        self.input_shape = input_shape

    def init_model_architecture(self):
        print("-----------------compiling model architecture")
        model = Sequential()
        model.add(LSTM(512, input_shape=self.input_shape, return_sequences=True, recurre
nt_dropout=0))
        model.add(LSTM(512, return_sequences=True, recurrent_dropout=0))
        model.add(LSTM(512))
        if self.artist == "schubert":
            model.add(BatchNormalization())
            model.add(Dropout(0.3))

        model.add(Dense(256, activation="relu"))

        if self.artist == "schubert":
            model.add(BatchNormalization())
            model.add(Dropout(0.3))

        if self.artist != "schubert":
            model.add(Dense(128, activation="relu"))
        model.add(Dense(self.vocab, activation="softmax"))
        self.model = model

    def load_model_weights(self):
        print("-----------------loading model weights")
        if self.model is not None:
            self.model.load_weights(f"music_generation/models/{self.artist}.h5")
        else:
            print("Could not load model weights")
            self.model = None
        return self.model

    def print_model_summary(self):
        if self.model is not None:
            print(self.model.summary())
        else:
            print("Model does not exist")
```

./music-generation/utils.py – Utility function needed to initialize model and get predictions

```
import os
from datetime import datetime

import music21
import numpy as np
```

```
def get_data(artist):
    print("-----------------Loading compressed dataset")
    data = np.load(f"music_generation/data/{artist}_processed.npz")
    x = data['arr_0']

    print("-----------------Loading personal artist pitches")
    data = np.load(f"music_generation/data/{artist}_pitchnames.npz")
    pitchnames = data['arr_0']
    n_vocab = len(pitchnames)
    return x, pitchnames, n_vocab

def generate_sequence(model, notes, x, pitchnames, n_vocab):
    start = np.random.randint(0, len(x) - 1)
    int_to_note = dict((number, note) for number, note in enumerate(pitchnames))

    pattern = x[start]
    prediction_output = []
    print("-----------------generating new music patterns")
    for ni in range(notes):
        prediction_input = np.reshape(pattern, (1, len(pattern), 1))
        prediction_input = prediction_input / float(n_vocab)

        prediction = model.predict(prediction_input, verbose=0)

        index = np.argmax(prediction)
        result = int_to_note[index]
        prediction_output.append(result)

        pattern = np.append(pattern, index)
        pattern = pattern[1:len(pattern)]
    return prediction_output

def generate_notes(predictions):
    offset = 0
    output_notes = []

    print("-----------------converting patterns to notes and chords")
    for pattern in predictions:
        # If the generated pattern is a chord, then we'll have to split the array of gen
erated
        # notes, convert them into sounds and again put it back together in a Chord obje
ct
        if pattern.isdigit() or '.' in pattern:
            notes_in_chord = pattern.split(".")
            notes= []

            for current_note in notes_in_chord:
                new_note = music21.note.Note(int(current_note))
                new_note.storedInstrument = music21.instrument.Piano()
                notes.append(new_note)
            new_chord = music21.chord.Chord(notes)
            new_chord.offset = offset
            output_notes.append(new_chord)

        # If the generated pattern is a note, then store them in a Note object
        else:
            new_note = music21.note.Note(pattern)
            new_note.offset = offset
```

```
            new_note.storedInstrument = music21.instrument.Piano()
            output_notes.append(new_note)

        offset += 0.5
    return output_notes

def save_midi(music, artist):
    print("-----------------Saving generated music")
    timestamp = datetime.now().strftime("%d-%m-%Y-%H-%M-%S")
    midi_stream = music21.stream.Stream(music)
    filename = f'created_music/{artist}-{timestamp}.mid'
    midi_stream.write("midi", fp=f'static/{filename}')
    print(f"-----------------Generate music saved: static/{filename}")

    return filename
```

./style_transfer/model.py – Model Architecture for Fast NST.

```
import tensorflow as tf
from .layers import InstanceNormConv2D, Residual, ResizableConv2D


class TransformerNet(tf.keras.models.Model):
    def __init__(self):
        super(TransformerNet, self).__init__()
        print("-----------------building model architecture")
        self.conv1 = InstanceNormConv2D(filters=32, kernel=9, stride=1)
        self.conv2 = InstanceNormConv2D(filters=64, kernel=3, stride=2)
        self.conv3 = InstanceNormConv2D(filters=128, kernel=3, stride=2)

        self.res1 = Residual(filters=128, kernel=3, stride=1)
        self.res2 = Residual(filters=128, kernel=3, stride=1)
        self.res3 = Residual(filters=128, kernel=3, stride=1)
        self.res4 = Residual(filters=128, kernel=3, stride=1)
        self.res5 = Residual(filters=128, kernel=3, stride=1)

        self.resize_conv1 = ResizableConv2D(filters=64, kernel=3, stride=2)
        self.resize_conv2 = ResizableConv2D(filters=32, kernel=3, stride=2)
        self.conv4 = InstanceNormConv2D(filters=3, kernel=9, stride=1)

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.res1(x)
        x = self.res2(x)
        x = self.res3(x)
        x = self.res4(x)
        x = self.res5(x)
        x = self.resize_conv1(x)
        x = self.resize_conv2(x)
        x = self.conv4(x, relu=False)

        return (tf.keras.activations.tanh(x) * 150 + 255.0 / 2)
```

./style_transfer/layers.py – Custom-built layer source code for the model

```
import tensorflow as tf
```

```
class InstanceNormalization(tf.keras.layers.Layer):
    def __init__(self, epsilon=1e-3):
        super(InstanceNormalization, self).__init__()
        self.epsilon = epsilon

    def build(self, input_shape):
        self.beta = tf.Variable(tf.zeros([input_shape[3]]))
        self.gamma = tf.Variable(tf.ones([input_shape[3]]))

    def call(self, inputs):
        mean, var = tf.nn.moments(inputs, axes=[1, 2], keepdims=True)
        x = tf.divide(tf.subtract(inputs, mean),
                      tf.sqrt(tf.add(var, self.epsilon)))

        return self.gamma * x + self.beta


class InstanceNormConv2D(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(InstanceNormConv2D, self).__init__()
        pad = kernel // 2
        self.padding = tf.constant([[0, 0], [pad, pad], [pad, pad], [0, 0]])
        self.conv = tf.keras.layers.Conv2D(
            filters, kernel, stride, use_bias=False, padding='valid')
        self.instance_norm = InstanceNormalization()

    def call(self, inputs, relu=True):
        x = tf.pad(inputs, self.padding, mode='REFLECT')
        x = self.conv(x)
        x = self.instance_norm(x)

        if relu:
            x = tf.keras.layers.Activation("relu")(x)
        return x


class Residual(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(Residual, self).__init__()
        self.conv1 = InstanceNormConv2D(filters, kernel, stride)
        self.conv2 = InstanceNormConv2D(filters, kernel, stride)

    def call(self, inputs):
        x = self.conv1(inputs)
        return inputs + self.conv2(x, relu=False)


class ResizableConv2D(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(ResizableConv2D, self).__init__()
        self.conv = InstanceNormConv2D(filters, kernel, stride)
        self.stride = stride

    def call(self, inputs):
        height = inputs.shape[1] * self.stride * 2
        width = inputs.shape[2] * self.stride * 2
        x = tf.image.resize(
            inputs, [height, width], method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
        x = self.conv(x)
```

```
        return x


class InstanceNormConv2DTranspose(tf.keras.layers.Layer):
    def __init__(self, filters, kernel, stride):
        super(InstanceNormConv2DTranspose, self).__init__()
        self.conv_transpose = tf.keras.layers.Conv2DTranspose(
            filters, kernel, stride, padding="same")
        self.instance_norm = InstanceNormalization()

    def call(self, inputs):
        x = self.conv_transpose(inputs)
        x = self.instance_norm(x)
        return tf.keras.layers.Activation("relu")
```

./style_transfer/perceptual_loss_model.py – initializing VGG-19 as a perceptual loss network

```
import tensorflow as tf
from .utils import vgg_layers, gram_matrix


class VGG19LossModel(tf.keras.models.Model):
    def __init__(self, style_layers, content_layers):
        super(VGG19LossModel, self).__init__()
        self.vgg = vgg_layers(style_layers + content_layers)
        self.style_layers = style_layers
        self.content_layers = content_layers
        self.num_style_layers = len(style_layers)
        self.num_content_layers = len(content_layers)
        self.vgg.trainable = False

    def call(self, inputs):
        preprocessed_input = tf.keras.applications.vgg19.preprocess_input(
            inputs)
        outputs = self.vgg(preprocessed_input)
        style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                          outputs[self.num_style_layers:])

        style_outputs = [gram_matrix(feature_map)
                         for feature_map in style_outputs]

        style_dict = {
            style_name: value for style_name, value in zip(self.style_layers, style_outp
uts)
        }

        content_dict = {
            content_name: value for content_name, value in zip(self.content_layers, cont
ent_outputs)
        }

        return {
            'content': content_dict,
            'style': style_dict
        }
```

./templates/base.html – Parent class for all .html files

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="preconnect" href="https://fonts.gstatic.com" />
    <link
      href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@400;500;600;7
00;800;900&family=Poppins:wght@300;400;500;600;700;800;900&family=Work+Sans:wght@400;500
;600;700&display=swap"
      rel="stylesheet"
    />
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scr
ipt>

    {% block title %}{% endblock %} {% block css %}{% endblock %}
  </head>

  <body>
    {% block content %}{% endblock %}
  </body>

  {% block scriptFooter %}{% endblock %}
</html>
```

./templates/home.html – User Interface for the Home page

```
{% extends "base.html" %} {% block title%}
<title>Flamingo - Home</title>
{% endblock %} {% block css%}
<link
  rel="stylesheet"
  href="{{ url_for('static', filename='css/home.css') }}"
/>
{% endblock %} {% block content %}
<div class="page-wrapper">
  <div class="nav-bar">
    <p class="site-title">Flamingo<span class="period">.</span></p>
    <div class="navigation-tabs">
      <div class="tabs active-tab">
        Home
        <hr color="#ff4242" />
      </div>
      <span class="separator">|</span>
      <a class="link" href="/curated-collection">
        <p class="tabs">Curated Collection</p>
      </a>
      <span class="separator">|</span>
      <a class="link" href="/about">
        <p class="tabs">About</p>
      </a>
    </div>
  </div>
```

```
  <div class="content-wrapper">
    <div class="content">
      <p class="hero-title">
        Deep Learning<br />for <span style="color: #ff4242">Art</span>
      </p>
      <p class="hero-subtitle">
        Ambition without intelligence is<br />a bird without wings.
      </p>
      <a href="/music-generation" class="link"
        ><div class="music-module">
          <p>Produce Music</p>
          <div class="divider-color"></div>
          <img
            src="{{url_for('static', filename='assets/open_link_color.svg')}}"
            width="15%"
          />
        </div>
      </a>
      <a href="/style-transfer" class="link"
        ><div class="style-module">
          <p>Transfer Style</p>
          <div class="divider"></div>
          <img
            src="{{url_for('static', filename='assets/open_link_white.svg')}}"
            style="color: white"
            width="15%"
          />
        </div>
      </a>
    </div>
    <div class="prop">
      <img
        src="{{url_for('static', filename='assets/home_prop.png')}}"
        width="80%"
      />
    </div>
    <div class="footer">
      <a
        href="https://github.com/KushGabani/flamingo"
        class="link"
        target="_blank"
      >
        <p>Github</p>
      </a>
      <p>Datasets</p>
      <p>Documentation</p>
    </div>
  </div>
</div>
{% endblock %}
```

./templates/about.html – User Interface for about page

```
{% extends "base.html" %} {% block title%}
<title>Flamingo - Music Generation</title>
{% endblock %} {% block css%}
<link
  rel="stylesheet"
```

```
  href="{{ url_for('static', filename='css/about.css') }}"
/>
{% endblock %} {% block content %}
<div class="page-wrapper">
  <div class="nav-bar">
    <a href="/" class="link">
      <p class="site-title">Flamingo<span class="period">.</span></p>
    </a>
    <div class="navigation-tabs">
      <a class="link" href="/"><p class="tabs">Home</p></a>
      <span class="separator">|</span>
      <a class="link" href="/curated-collection">
        <p class="tabs">Curated Collection</p>
      </a>
      <span class="separator">|</span>
      <div class="tabs active-tab">
        About
        <hr color="#ff4242" />
      </div>
    </div>
  </div>

  <div class="content-wrapper">
    <div class="prop">
      <img src="{{url_for('static', filename='assets/about_prop.png')}}" />
    </div>
    <div class="content">
      <div>
        <p class="page-title">
          <span style="color: #ff4242">About</span> the project
        </p>
        <div class="page-subtitle">
          <p class="subtitle">KUSH GABANI</p>
          <div
            style="
              width: 10px;
              height: 10px;
              background-color: #ff4242;
              border-radius: 50%;
              margin: 0rem 1rem;
            "
          ></div>
          <p class="subtitle">ML/DL</p>
        </div>
        <div class="module">
          <p class="module-title">MUSIC GENERATION MODULE</p>
          <p class="module-description">
            Based on the selected instrument and a specific duration, a rythm is
            generated by an LSTM Neural Network model symphonies. The model was
            trained on the basis of a variety of different music from Final
            Fantasy V. The module has gained the ability to keep track of
            context while generating a music pattern
          </p>
          <div class="module-footer">
            <p style="margin-right: 2rem">Github</p>
            <p style="margin-right: 2rem">Dataset</p>
            <p style="margin-right: 2rem">Model Architecture</p>
          </div>
        </div>
```

```
        <div class="module">
          <p class="module-title">(FAST) NEURAL STYLE TRANSFER</p>
          <p class="module-description">
            This module takes in a content image that needs to be styled and a
            style image whose abstract style will be applied on the content
            image. A Neural Style Transfer model is trained discovered by Gatys
            et al. On top of it, I have used an entire VGG-19 model as a loss
            function and and implemented Instance Normalization layer for
            generalization
          </p>
          <div class="module-footer">
            <p style="margin-right: 2rem">Github</p>
            <p style="margin-right: 2rem">Dataset</p>
            <p style="margin-right: 2rem">Model Architecture</p>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
{% endblock %}
```

./templates/curated_collection.html – User Interface to display training results

```
{% extends "base.html" %} {% block title%}
<title>Flamingo - Curated Collection</title>
{% endblock %} {% block css%}
<link
  rel="stylesheet"
  href="{{ url_for('static', filename='css/curated_collection.css') }}"
/>
{% endblock %} {% block content %}
<div class="page-wrapper">
  <div class="nav-bar">
    <a href="/" class="link">
      <p class="site-title">Flamingo<span class="period">.</span></p>
    </a>
    <div class="navigation-tabs">
      <a class="link" href="/"><p class="tabs">Home</p></a>
      <span class="separator">|</span>
      <div class="tabs active-tab">
        Curated Collection
        <hr color="#ff4242" />
      </div>
      <span class="separator">|</span>
      <a class="link" href="/about"><p class="tabs">About</p></a>
    </div>
  </div>

  <div class="content-wrapper">
    <div class="upper-section">
      <div class="c1">
        <img
          src="{{url_for('static', filename='assets/c1.png')}}"
          width="100%"
          height="100%"
        />
      </div>
```

```
</div>
<div class="lower-section">
  <div class="c2">
    <img
      src="{{url_for('static', filename='assets/c2.png')}}"
      width="100%"
      height="100%"
    />
  </div>
  <div class="lower-sub">
    <div class="sub1">
      <div class="audio1">
        <audio loop="false" id="m1">
          <source
            src="{{url_for('static', filename='assets/m1.mp3')}}"
            type="audio/mpeg"
          />
        </audio>
        <div class="audio">
          <img
            src="{{url_for('static', filename='assets/play.svg')}}"
            id="p1"
          />
          <h4 style="margin-top: 1rem">The First</h4>
        </div>
      </div>
      <div class="c3">
        <img
          src="{{url_for('static', filename='assets/c3.png')}}"
          width="100%"
          height="100%"
        />
      </div>
    </div>
    <div class="sub2">
      <div class="c4">
        <img
          src="{{url_for('static', filename='assets/c4.png')}}"
          width="100%"
          height="100%"
        />
      </div>
      <div class="audio2">
        <audio loop="false" id="m2">
          <source
            src="{{url_for('static', filename='assets/m2.mp3')}}"
            type="audio/mpeg"
          />
        </audio>
        <div class="audio">
          <img
            src="{{url_for('static', filename='assets/play.svg')}}"
            id="p2"
          />
          <h4 style="margin-top: 1rem">The Fifth</h4>
        </div>
      </div>
    </div>
  </div>
```

```
      <div class="c5">
        <img
          src="{{url_for('static', filename='assets/c5.png')}}"
          width="100%"
          height="100%"
        />
      </div>
      <div class="c6">
        <img
          src="{{url_for('static', filename='assets/c6.png')}}"
          width="100%"
          height="100%"
        />
      </div>
    </div>
  </div>
</div>
{% endblock %} {% block scriptFooter %}
<script>
  is_audio_1_playing = false;
  is_audio_2_playing = false;
  audio1 = document.getElementById("m1");
  audio2 = document.getElementById("m2");

  document.getElementById("p1").addEventListener("click", function () {
    if (is_audio_2_playing) {
      audio2.pause();
    }
    if (is_audio_1_playing) {
      audio1.pause();
      is_audio_1_playing = false;
    } else {
      is_audio_1_playing = true;
      audio1.play();
    }
  });

  document.getElementById("p2").addEventListener("click", function () {
    if (is_audio_1_playing) {
      audio1.pause();
    }
    if (is_audio_2_playing) {
      audio2.pause();
      is_audio_2_playing = false;
    } else {
      is_audio_2_playing = true;
      audio2.play();
    }
  });
</script>
{% endblock %}
```

./templates/music_generation.html – User Interface for music generation

```
{% extends "base.html" %} {% block title%}
<title>Flamingo - Music Generation</title>
{% endblock %} {% block css%}
```

```
<link
  rel="stylesheet"
  href="{{ url_for('static', filename='css/music_generation.css') }}"
/>
{% endblock %} {% block content %}
<div class="page-wrapper">
  <div class="top-bar">
    <a href="/">
      <div class="back">
        <img src="{{ url_for('static', filename='assets/back.svg') }}" />
      </div>
    </a>
    <div class="caption">
      A new breed of <span style="color: #ff4242">music</span>
    </div>
    <div></div>
  </div>

  <div class="content-wrapper">
    <div class="prop">
      <img
        src="{{ url_for('static', filename='assets/music_prop.png') }}"
        width="70%"
      />
    </div>
    <div class="content">
      <form method="POST" action="/handle-music-form">
        <div class="form">
        <div class="select-instrument">
            <p>Select an artist</p>
            <div class="options">
              <input type="radio" name="artist" value="beethoven" class="rd-
button" id="beethoven" />
              <label for="beethoven">
                  <div class="option-icon">
                      <img
                          src="{{ url_for('static', filename='assets/beethoven.png') }}"
                          width="80%"
                      />
                  </div>
              </label>

              <input type="radio" name="artist" value="mozart" class="rd-
button" id="mozart" />
              <label for="mozart">
                  <div class="option-icon">
                    <img
                        src="{{ url_for('static', filename='assets/mozart.png') }}"
                        width="90%"
                    />
                  </div>
              </label>

              <input type="radio" name="artist" value="schubert" class="rd-
button" id="schubert" />
              <label for="schubert">
                  <div class="option-icon">
                      <img
                          src="{{ url_for('static', filename='assets/schubert.png') }}"
```

```
                        width="70%"
                    />
                  </div>
                </label>
              </div>
            </div>
            <div class="duration-container">
                <p class"duration-label">Duration (in seconds)</p>
                <input type='number' name="duration" class="duration" autocomplete="off" />
            </div>
            <input type="submit" name="submit" class="submit" value="Create"/>
            </div>
          </form>
        </div>
        <div class="footer">
          <a href="https://github.com/KushGabani/Flamingo/tree/main/music_generation" target
="_blank" class="link">
            <p>Github</p>
          </a>
          <a href="https://drive.google.com/file/d/1epTcJbd6Q5QMt1OIjAT2EugbCMvoS1y9/view?us
p=sharing" target="_blank" class="link">
            <p>Datasets</p>
          </a>
          <p>Documentation</p>
        </div>
      </div>
    </div>{% endblock %}
```

./templates/style_transfer.html – User Interface for NST module

```
{% extends "base.html" %} {% block title%}
<title>Flamingo - Style Transfer</title>
{{ dropzone.load_css() }} {% endblock %} {% block css%}
<link
  rel="stylesheet"
  href="{{ url_for('static', filename='css/transfer-style.css') }}"
/>
{% endblock %} {% block content %}
<div class="page-wrapper">
    <a href="/"><div class="top-bar">
      <div class="back">
        <img src="{{ url_for('static', filename='assets/back.svg') }}" />
      </div>
    </a>
    <div class="caption">
      Bring <span style="color: #ff4242">creativity</span> to life
    </div>
    <div></div>
  </div>
  <div class="content-wrapper">
    <div class="dropzone-container">
    {{ dropzone.create('/upload-content') }}
    <form action="/handle-transfer-form" method="POST" class="form">
      <div class="options">
        <input type="radio" name="style-image" value="wave" class="rd-
button" id="wave" />
        <label for="wave">
          <div class="option-icon">
```

```
            <h3>Wave</h3>
          </div>
        </label>

        <input type="radio" name="style-image" value="adobe" class="rd-
button" id="adobe" />
        <label for="adobe">
          <div class="option-icon">
            <h3>Woman</h3>
          </div>
        </label>

        <input type="radio" name="style-image" value="scream" class="rd-
button" id="scream" />
        <label for="scream">
          <div class="option-icon">
            <h3>Scream</h3>
          </div>
        </label>

        <input type="radio" name="style-image" value="edward" class="rd-
button" id="edward" />
        <label for="edward">
          <div class="option-icon">
            <h3>Edward</h3>
          </div>
        </label>
      </div>
      <div class="options">
        <input type="radio" name="style-image" value="avignon" class="rd-
button" id="avignon" />
        <label for="avignon">
          <div class="option-icon">
            <h3>Avignon</h3>
          </div>
        </label>

        <input type="radio" name="style-image" value="violin" class="rd-
button" id="violin" />
        <label for="violin">
          <div class="option-icon">
            <h3>Picasso's Violin</h3>
          </div>
        </label>

        <input type="radio" name="style-image" value="talig" class="rd-
button" id="talig" />
        <label for="talig">
          <div class="option-icon">
            <h3>Talig</h3>
          </div>
        </label>
      </div>
      <button type="submit"  id="upload" class="transfer-button">Transfer Style</button>
    </form>
    {{ dropzone.load_js() }} {{ dropzone.config() }}
    </div>
  </div>
  <div class="footer">
```

```
      <a href="https://github.com/KushGabani/Flamingo/tree/main/style_transfer" target="
_blank" class="link">
        <p>Github</p>
      </a>
      <a href="https://drive.google.com/drive/folders/1r3DpGudMGkOUqRCNapR9HdPqBEyMvHT8?
usp=sharing" target="_blank" class="link">
        <p>Datasets</p>
      </a>
      <p>Documentation</p>
    </div>
</div>
{% endblock %}
```

Testing

Testing Methods for Deep Learning Models isn't that simple. The quality of image generated by the fast neural style transfer model is purely subjective and there is no proper measurement tool or formula that can mimic a human behaviour. Similarly, in music generation module, the music generated by the model is purely subjective to a person. On the other hand, on the UI side there may be some testing needed.

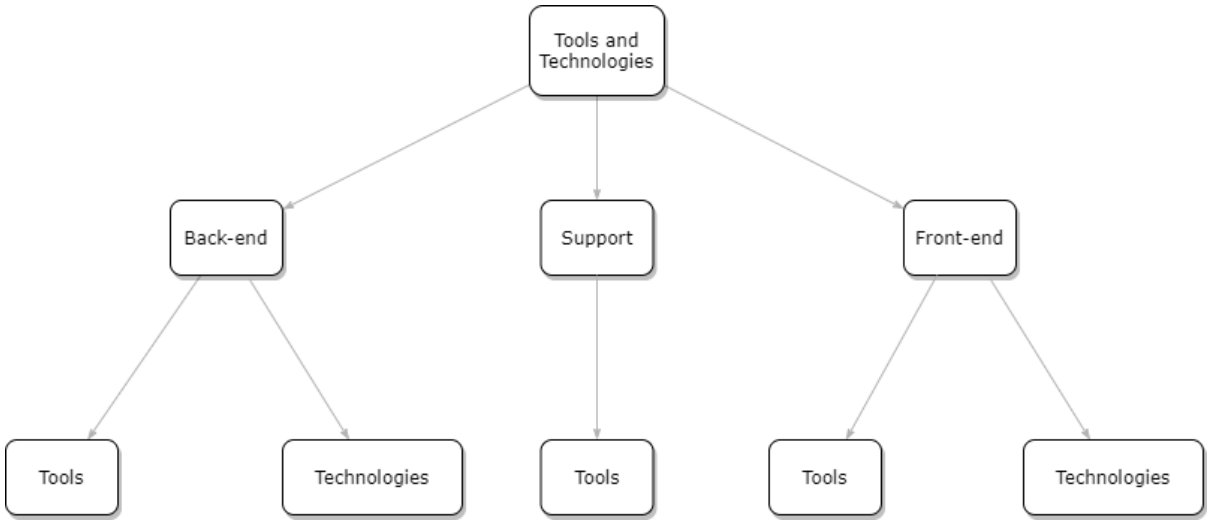
In the Fast NST module, the user is prompted to upload an image to style. The upload sub-module is built using dropzone.js and thus all the errors and exceptions are handled by the library already. Thus, the only qualifying factor for all these methods is the loss of the model after training. Following is the table containing the loss of the aforementioned models:

(The lower the loss, the better the model)

MODEL	**LOSS**
FAST NEURAL STYLE TRANSFER	
Model 1.1 (Scream)	3742698.50
Model 1.2 (Edward)	3265484.21
Model 1.3 (Wave)	4765165.03
Model 1.4 (Violin)	3956541.42
Model 1.5 (Talig)	4127354.65
Model 1.6 (Woman)	3965417.51
Model 1.7 (Avignon)	3513549.73
SEQUENCE-TO-VECTOR LSTM MODEL	
Model 2.1 (Beethoven)	0.08796
Model 2.2 (Mozart)	0.1598
Model 2.3 (Schubert)	0.07482

Tools & Technologies

The project is an amalgamation of various tools and technologies. Ranging from HTML, CSS to Python, Deep Learning frameworks and third-party libraries. We categorize the tools and technologies in the following hierarchy:



- Back-end
  - Tools
    - Google Colab (GPU Tesla K80)
    - COCO Dataset
    - NumPy, Pillow
  - Technologies
    - Python
    - Tensorflow, TFHub, TensorBoard
    - Keras
    - Music21
    - openCV
    - Flask
- Front-end
  - Tools
    - dropzone.js
  - Technologies
    - HTML, CSS
    - APIs
- Support
  - Tools
    - Git & Github
    - Draw.io
    - PyCharm, VSCode & Vim

System Limitations and Dependency Constraints

SYSTEM LIMITATIONS

System Limitations are defined as an ability of a system to perform its intended tasks under its current configuration. The project consists of two phase: the training phase and the inference phase. The inference phase is not very resource demanding and thus it can be run on any medium performance system. But the training is very time consuming.

The TransformerNet needed to be trained for 2 epochs with a batch size of 2 on 80,000 images. For each feed-forward pass, there was also a perceptual loss network that was 5 levels deep. The output of the image depends on the resolution of the image needed. To apply transformations on high-resolution images is very time consuming and highly depends on the configuration of the system. 1 epoch with these system constraints takes almost 4 hours and the quality of output is not fixed. The output varies all the time and thus if the output is not pleasing or meet the necessary requirements, the whole process needs to be repeated again. There are 7 NST models for each style image. Optimistically, **at least 56 hours** of training is needed. Apart from being time consuming, storage also needs to be accounted as the entire model and its weights are saved after every 25% completed. The COCO dataset itself consists of 80,000 images which roughly translates to 12GB-13GB of training data. GPU-enabled Machines with high configurations are needed to train these models. For convenience, all these models are trained on GPU sessions provided by Google Colab.

For the music generation model, the three different LSTM models are trained on dataset of 30 songs on average. The model is trained for 100 epochs and a batch size of 128. Each epoch takes roughly 1 minute 40 seconds. This translates to around 1.5 hours for each model. Summing this all up, training time of **around 4.5 hours is needed.** Though this module doesn't consume much storage, it is still operation heavy and time consuming.

DEPEDENCY CONSTRAINTS

The Tensorflow framework is used with the Keras API to build and train models. Tensorflow 2.0 must be installed on the system along with music21 and numpy. Other than these the following are the dependencies also needs to be installed:

```
astunparse==1.6.3
cachetools==4.2.2
certifi==2021.5.30
chardet==4.0.0
click==8.0.1
colorama==0.4.4
cycler==0.10.0
Flask==2.0.1
Flask-Dropzone==1.6.0
flatbuffers==1.12
gast==0.4.0
google-auth==1.31.0
google-auth-oauthlib==0.4.4
google-pasta==0.2.0
grpcio==1.34.1
h5py==3.1.0
idna==2.10
itsdangerous==2.0.1
Jinja2==3.0.1
joblib==1.0.1
keras-nightly==2.5.0.dev2021032900
Keras-Preprocessing==1.1.2
kiwisolver==1.3.1
Markdown==3.3.4
MarkupSafe==2.0.1
matplotlib==3.4.2
more-itertools==8.8.0
music21==6.7.1
numpy==1.19.5
oauthlib==3.1.1
opt-einsum==3.3.0
Pillow==8.2.0
protobuf==3.17.3
pyasn1==0.4.8
pyasn1-modules==0.2.8
pyparsing==2.4.7
python-dateutil==2.8.1
requests==2.25.1
requests-oauthlib==1.3.0
rsa==4.7.2
six==1.15.0
tensorboard==2.5.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.0
tensorflow-cpu==2.5.0
tensorflow-estimator==2.5.0
termcolor==1.1.0
typing-extensions==3.7.4.3
urllib3==1.26.5
webcolors==1.11.1
Werkzeug==2.0.1
wrapt==1.12.1
```

Future Enhancements and Opportunities

- The Fast Neural Style Transfer module currently needs to train a model for a specific style image. The model architecture can be changed and new constraints / loss functions can be defined resulting in a generalized model which can accept an arbitrary style image and transfer that on a content image.
- Other than this, the model architecture and the training process can also be modified to style a content image using multiple style images and combine them. The resulting image would have colours, textures and strokes from all the style images.
- A mechanism can be introduced to select a portion of the content image where a specific style of an image needs to be applied. This will result in an image that is completely modifiable by the user themselves.
- TensorBoard can be used for further visualization of the training process and the model architecture. By using this, we can also study the model in detail to find new insights.

- The Music Generation module as of now has different models for each artist. The model can be generalized to generate new patterns based on a mixed persona of all the artist
- The dataset is still very small compared to others like COCO dataset etc. The internet is filled with openly available midi files and audio files in other format can also be converted into midi files. This will increase the size of the dataset substantially and the model will be able to generalize it well.
- Data Augmentation is not yet applied on audio files due to complexities and technical limitations. Augmentations to pitch, and time between two notes can be varied by changing the distance between them. This would result in a more flexible model.
- An entirely new user interface can be created for the user from where they can control the speed of the notes, change the frequency of the occurring notes, and the duration of the song.
- Apart from changes related to this project, there is a proper need of ending a song. It is often difficult to train a model that will conclude a music just like artists do. This particular enhancement will result in a much more pleasant user experience and that’s how it will truly assist an artist to create new melodies.

Bibliography & References

[1] Neural algorithm of Artistic Style, Gatys et al. Sep 2, 2015 https://arxiv.org/abs/1508.06576

[2] Instance Normalization: The Missing Ingredient For Fast Stylization, Dmitry Ulyanov, Nov 6, 2017 https://arxiv.org/abs/1607.08022

[3] Perceptual Losses for Real-time Style Transfer and Super-Resolution, Justin Johnson, Alexandre Alahi, and Li Fei-Fei, Mar 27, 2016 https://arxiv.org/abs/1603.08155

[4] Common Objects in Context (COCO) Dataset, info@cocodataset.org, https://cocodataset.org/#download

[5] Chapter 8, Optimization for Deep Neural Networks, Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016 https://www.deeplearningbook.org

[6] Chapter 10, Sequence Modeling: Recurrent and Recursive Nets, Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016 https://www.deeplearningbook.org

[7] Chapter 20, Deep Generative Models, Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, 2016 https://www.deeplearningbook.org

[8] LSTM based Music Generation System, Sanidhya Mangal, Rahul Modak, Poorva Joshi, Aug 2, 2019 https://arxiv.org/abs/1908.01080

[9] The Effectiveness of Data Augmentation in Image Classification using Deep Learning, Jason Wang and Luis Perez, Stanford https://arxiv.org/pdf/1712.04621.pdf

[10] Convolutional Neural Network (CNN), Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/images/cnn

[11] Keras API Reference, Keras, Keras Special Interest Group, https://keras.io/api/

[12] Eager Execution, Tensorflow Documentation, Google, https://www.tensorflow.org/guide/eager

[13] Intro to Autoencoders, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/generative/autoencoder

[14] Convolutional Variational Autoencoders, Tensorflow Documentation, Google, < https://www.tensorflow.org/tutorials/generative/cvae>

[15] Deep Convolutional Generative Adversarial Networks, Tensorflow Documentation, Google, https://www.tensorflow.org

[16] Pix2Pix: Image-to-image translation with a conditional GAN, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/generative/dcgan

[17] Neural Style Transfer, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/generative/style_transfer

[18] Data Augmentation, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/images/data_augmentation

[19] Transfer Learning and Fine-Tuning, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/images/transfer_learning

[20] Tune hyperparameters with the Keras Tuner, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/keras/keras_tuner

[21] Integrated gradients, Tensorflow Documentation, Google, https://www.tensorflow.org/tutorials/interpretability/integrated_gradients

[22] Making new Layers and Models via subclassing, Tensorflow Documentation, Google, https://www.tensorflow.org/guide/keras/custom_layers_and_models

[23] Writing a training loop from scratch, Tensorflow Documentation, Google, https://www.tensorflow.org/guide/keras/writing_a_training_loop_from_scratch
[24] Recurrent Neural Networks (RNN) with Keras, Tensorflow Documentation, Google, https://www.tensorflow.org/guide/keras/rnn
[25] Optimize Tensorflow GPU performance with the Tensorflow Profiler, Tensorflow Documentation, Google, https://www.tensorflow.org/guide/gpu_performance_analysis
[26] Intro | Neural Style Transfer #1, The AI Epiphany, Mar 16, 2020, YouTube https://www.youtube.com/c/TheAIEpiphany
[27] Basic Theory | Neural Style Transfer #2, The AI Epiphany, Mar 24, 2020, YouTube https://www.youtube.com/c/TheAIEpiphany
[28] Optimization method | Neural Style Transfer #3, The AI Epiphany, Apr 5, 2020, YouTube https://www.youtube.com/c/TheAIEpiphany
[29] Advanced Theory | Neural Style Transfer #4, The AI Epiphany, Apr 18, 2020, YouTube https://www.youtube.com/c/TheAIEpiphany
[30] Feed-Forward method | Neural Style Transfer #5, The API Epiphany, Jun 3, 2020, YouTube https://www.youtube.com/c/TheAIEpiphany
[31] pytorch-neural-style-transfer, gordicaleksa, Github https://github.com/gordicaleksa/pytorch-neural-style-transfer
[32] pytorch-neural-style-transfer-johnson, gordicaleksa, Github https://github.com/gordicaleksa/pytorch-neural-style-transfer
[33] Neural-Style-Transfer, titu1994, Github https://github.com/titu1994/Neural-Style-Transfer
[34] Fast-style-transfer, lengstrom, Github https://github.com/lengstrom/fast-style-transfer
[35] Neural-style, jcjohnson, Github https://github.com/jcjohnson/neural-style
[36] Fast-neural-style, jcjohnson, Github https://github.com/jcjohnson/neural-style
[37] tf.keras.layers.LSTM, Tensorflow Documentation, Google, https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
[38] LSTM by example using Tensorflow, Rowel Atienza, towardsdatascience, Mar 17, 2017 https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537
[39] A Gentle Introduction to Convolutional layers in CNNs, Jason Brownlee, machinelearningmastery, Apr 17, 2019 https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
[40] A Gentle Introduction to Padding and Stride for Convolutional Neural Networks, Jason Brownlee, machinelearningmastery, Apr 19, 2019 https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/
[41] A Tour of Generative Adversarial Network Models, Jason Brownlee, machinelearningmastery, Jul 10, 2019 < https://machinelearningmastery.com/tour-of-generative-adversarial-network-models/>
[42] How to implement Pix2Pix GAN models from scratch with Keras, Jason Brownlee, Jul 30, 2019 https://machinelearningmastery.com/how-to-implement-pix2pix-gan-models-from-scratch-with-keras/
[43] The Promise of Recurrent Neural Networks for Time Series Forecasting, Jason Brownlee, machinelearningmastery, May 22, 2017 https://machinelearningmastery.com/promise-recurrent-neural-networks-time-series-forecasting/

[44] A Gentle Introduction to Long Short-Term Memory Networks by the Experts, Jason Brownlee, machinelearningmastery, May 24, 2017, https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/
[45] How to Handle Missing Timesteps in Sequence Prediction Problems with Python, Jason Brownlee, machinelearningmastery, Jun 21, 2017, https://machinelearningmastery.com/handle-missing-timesteps-sequence-prediction-problems-python/
[46] How to Diagnose Overfitting and Underfitting of LSTM Models, Jason Brownlee, machinelearningmastery, Sep 1, 2017, https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/
[47] A Gentle Introduction to Backpropagation Through Time, Jason Brownlee, machinelearningmastery, Jun 23, 2017 https://machinelearningmastery.com/gentle-introduction-backpropagation-time/
[48] How to Avoid Overfitting in Deep Learning Neural Networks, Jason Brownlee, machinelearningmastery, Dec 17, 2018, https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/
[49] Reduce Overfitting With Dropout Regularization in Keras, Jason Brownlee, machinelearningmastery, Dec 5, 2018 https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/
[50] Want to Generate your own Music using Deep Learning? Here's a Guide to do just that!, Aravind Pai, Analytics Vidya, Jan 21, 2020 https://www.analyticsvidhya.com/blog/2020/01/how-to-perform-automatic-music-generation/
[51] Generating Original Classical Music with an LSTM Neural Network and Attention, Alex Issa, Medium, May 22, 2019 https://medium.com/@alexissa122/generating-original-classical-music-with-an-lstm-neural-network-and-attention-abf03f9ddcb4
[52] Generate Piano Instrumental Music by using Deep Learning, Haryo Akbarianto Wibowo, towardsdatascience, Mar 25, 2019 https://towardsdatascience.com/generate-piano-instrumental-music-by-using-deep-learning-80ac35cdbd2e
[53] An Intuitive Guide To Deep Learning Architectures, Joyce Xu, towardsdatascience, Aug 15 2017 https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41
[54] Practical techniques for getting style transfer to work, Sahil Singla, towardsdatascience, Aug 31, 2017 https://towardsdatascience.com/practical-techniques-for-getting-style-transfer-to-work-19884a0d69eb
[55] Deconvolution and Checkerboard Artifacts, Augustus Odena, Vincent Dumoulin, Chris Olah, Google Brain and Université de Montréal, Oct 17, 2016 https://distill.pub/2016/deconv-checkerboard/
[56] Is the Deconvolution layer same as a convolution layer?, Wenzhe Shi and Jose Caballero, Sep 22, 2016, https://arxiv.org/abs/1609.07009
[57] Image Super-Resolution Using Deep Convolutional Networks, Chao Dong, Chen Change Loy, Dec 31, 2014 https://arxiv.org/abs/1501.00092