**Chapter 3**

**Introduction**

In this chapter, we discuss the data collection, listing of data properties and snapshot of collected dataset, data manipulation and pre-processing, model selection and brief description of corresponding models, algorithms of model building and training and, generate uncertainty in from predicted data, discuss tactics of CA, visualize uncertainty in terms of CA in web panels and show examples of uses of CA in real world charts.

**3.1 Data**

Authentic data is the most important part in the data visualization research. Without having an authentic dataset research cannot be started properly and without following a smart data preparation strategy such as cleaning, validating, and consolidating raw data in the right way, research cannot succeed in the long run.

**3.1.1 Data Collection**Data collection is the process of gathering, measuring, and analyzing accurate information on variables of interest, in an established systematic manner that enables one to pursue one to conduct research in right direction. Due to the global impact of pandemic, different individuals, organizations, or governments are storing data in their own way. After going through different repositories, we found that the complete and authentic data is bundled in ourworldindata.org at csv format. The following table shows the list of fields/properties of each record where many of them are not relevant to our research. For example: date, location, new\_cases, total\_cases are some of the useful attributes bolded in the following table.

|  |  |  |
| --- | --- | --- |
| **iso\_code** | continent | **location** |
| **date** | **total\_cases** | **new\_cases** |
| new\_cases\_smoothed | **total\_deaths** | **new\_deaths** |
| new\_deaths\_smoothed | total\_cases\_per\_million | new\_cases\_per\_million |
| new\_cases\_smoothed\_per\_million | population\_density | new\_deaths\_per\_million |
| new\_deaths\_smoothed\_per\_million | stringency\_index | **population** |
| new\_cases\_smoothed\_per\_million | median\_age | aged\_65\_older |
| aged\_70\_older | gdp\_per\_capita | extreme\_poverty |
| cardiovasc\_death\_rate | diabetes\_prevalence | female\_smokers |
| male\_smokers | handwashing\_facilities | hospital\_beds\_per\_thousand |
| life\_expectancy | human\_development\_index |  |

Table-1: COVID Data property list

**3.1.2 Sample Data**

**A picture containing text, appliance

Description automatically generated**

Table-2: screenshot of sample data

In the above Table-2, it shows only a snapshot of whole dataset where there are hundreds of thousands of records for Covid data for more than 237 countries and territories. Though there are numerous fields in the data, we only needed few of them as listed in previous section. The dataset is collected as a excel file which includes daily occurances and/or counts of all properties. The total\_\* fields like total\_cases, total\_deaths, etc are cumulative and so every day that is updated with previous day’s counts. Data is ordered by date and name of the country correspondingly. If there is no value in a cell for certain date for a country, that cell is kept empty, so that is needed to handle during data processing.

**3.2 Machine Learning Algorithms**

Although we have not done anything novel in machine learning domain, it is necessary to briefly introduce the salient algorithms that were used in our research to process the available data and generate the uncertainties of predictions since uncertainty representation is our prime concern.

**3.2.1 Predictive/Forecasting Models**A time series forecasting model comprises a sequence of data points captured, using time as the input parameter. It uses the historical data to develop a numerical metric and predicts values for the next duration, for instance, data for the next few weeks using that metric.

Forecasting Algorithms

Training Data

New Data  
(Optional)

Predictions

After Training

Data with Uncertainty

Calculate Uncertainty

Figure-3: Predictive modeling workflow to generate uncertainty

**3.2.2 Time Series Analysis vs Forecasting**

Sometimes ambiguity arises between time series analysis with time series forecasting when working with temporal data. As per Shmueli el al. [31] in time series analysis, a time series is modeled to determine its components in terms of seasonal patterns, trends, and relation to external factors. In contrast, time series forecasting uses the information in a time series (perhaps with additional information) to forecast future values of that series. The COVID-19 dataset is maintained on a global basis, so it is more trustworthy and with time series forecasting models can be considered as suitable for our research to get the predicted results and hence generate our required uncertainty data to represent chromatic aberration in visualization area.

**3.2.3 Concerns of Forecasting**Time series forecasting is an important area of machine learning. It is important because there are so many prediction problems that involve real life issues like time component. In forecasting it is very important to understand the goal of the problem and the nature of the available data. For instance, the volume of data, time horizons (short, medium or long term), frequency of update etc. plays an important role in forecasting. Sometimes time series data requires cleaning, scaling and even transformation, for example: if there are gaps/missing data, if there are outliers or corrupt data then those need to be addressed. Depending on the frequency, a time series can be of yearly (e.g., annual budget), quarterly (e.g., profit), monthly (e.g., cash flow), weekly (e.g., sales quantity), daily (e.g., weather forecast), hourly (e.g., stock market price), minutes (e.g., calls in a call canter) and even seconds wise (e.g., web traffic). Being the covid pandemic world-wide concerns for whole humanity, we use the daily forecast mechanism to our research. To compare the results side by side we have created prediction for 200 days from every models.

**3.2.4 Example of Forecasting**

Histogram

Description automatically generated

Figure-4: Example of daily covid forecasting for 200 days

The above Figure-4 shows the daily forecasting of number new cases for United States based on previous statistics. So, in the blackish line in left shows the actual occurrences and the reddish line at right shows the predicted number of cases and greyed background surrounding the predicted line represents the ranges of model prediction, that means the model can predict a value between the lower and upper value for a certain day and that grey area represents the area of uncertainty.

**3.3 MLP**  
A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). It is a neural network connecting multiple layers in a directed graph, which means that the signal passes through the nodes only in one direction. It can be used for time series forecasting by taking multiple observations at prior time steps, called lag observations, and using them as input features and predicting one or more-time steps from those observations. The training dataset is therefore a list of samples, where each sample has some number of observations from days prior to the time being forecasted, and the forecast is the next days in the sequence.

Diagram

Description automatically generated

Figure-5: Basic Architecture of MLP network [ref. 33]

We use the rectified linear activation function on the hidden layer as it performs well and a linear activation function on the output layer because we are predicting a continuous value. We use root squared error as loss function and the ‘adam’ optimizer for training the network.

The following steps shows the algorithm used to setup our MLP model:

---------------------------------------------------------------------------------------------------------------------

1. Take an instance of ‘Sequential’ Model from Keras deep learning library.
2. Add a Dense layer to the model with stating number of inputs (24), number of nodes (500), number of epochs (100) and batch size (100), rectified linear activation function (relu).
3. Add another Dense layer with number of outputs (1), since we predict a continuous value.
4. Compile the model with mean square error (mse) loss function and ‘adam’ optimizer.
5. Fit the model with training data set for number of epochs (100) and batch size (100).
6. Make an ensemble of models by following the steps 1 to 5.
7. Get prediction ‘yhat’ for each time step (day) from all the models of the ensemble.
8. Calculate the ranges (lower level, mean and upper level) of each prediction.
9. Calculate uncertainty of the model for each day by using the set of yhats using the uncertainty calculating formula explained in 3.7.

---------------------------------------------------------------------------------------------------------------------

Algorithm-1: MLP Model

**3.4 CNN**Convolutional Neural Networks are a type of deep neural network developed for computer vision; for instance, two-dimensional image data, although they can be used for one-dimensional data such as sequences of text and time series forecasting. When operating on one-dimensional data, the CNN reads across a sequence of lag observations and learns to extract features that are relevant for making a prediction.

Diagram

Description automatically generated

Figure-6: Basic Architecture of CNN network [ref. 34]

We define a CNN with two convolutional layers, one max-pooling layer, one flatten layer, and a dense layer from the input sequences. 1D convolution (Conv1D) layer (e.g., temporal convolution creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. They have a configurable number of filters, kernel size, pool size and rectified linear activation function is used as loss function. The number of filters determines the number of parallel fields on which the weighted inputs are read and projected. A max pooling layer is used after convolutional layers to distill the weighted input features into those that are most salient, reducing the input size by 1/2. The pooled inputs are flattened to generate a long vector before being interpreted and used to make the prediction.

To dive into further we need to briefly introduce some of the basic terms used in this model:

**Conv1D:**

A convolution layer transforms the input image in order to extract features from it. This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs

**MaxPooling1D:**

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

The following steps shows the algorithm used to setup our CNN model:

---------------------------------------------------------------------------------------------------------------------

1. Take an instance of ‘Sequential’ Model from Keras deep learning library.
2. Add a Conv1D layer to the model defining the number of filters (24), kernel size (500), input shape (100), rectified linear activation function (relu).
3. Add another Conv1D layer with same settings but without input shape.
4. Add another MaxPooling1D layer with pool size of 2.
5. Flatten (reshape) the result of previous step into single dimension before interpreted by the next layer.
6. Add a Dense layer with number of outputs (1), since we predict a continuous value.
7. Compile the model with mean square error (mse) loss function and ‘adam’ optimizer.
8. Fit the model with training data set for number of epochs (100) and batch size (100).
9. Create an ensemble of 6 models by following the steps 1 to 8.
10. Get prediction ‘yhat’ for each time step (day) from all the models of the ensemble.
11. Calculate the ranges (lower level, mean and upper level) of each prediction.
12. Calculate uncertainty of the model for each day by using the set of yhats using the uncertainty calculating formula explained in 3.7.

---------------------------------------------------------------------------------------------------------------------

Algorithm-1: CNN Model

**3.5 LSTM**

The LSTM neural network is a member of RNN and it can be used for univariate time series forecasting. It uses an output of the network from a prior step as an input in attempt to automatically learn across sequence data. The LSTM has an internal memory allowing it to accumulate internal state as it reads across the steps of a given input sequence.

**Diagram

Description automatically generated**

Figure-7: Basic Architecture of LSTM network (ref. 55)

For this model we define a LSTM layer from inputs and subsequently two dense layers. Like other models, rrectified linear activation function is used in LSTM layer and in one of dense layer. A simple grid search of model hyperparameters was performed with the predefined configuration.

The following steps shows the algorithm used to setup our LSTM model:

---------------------------------------------------------------------------------------------------------------------

1. Take an instance of ‘Sequential’ Model from Keras deep learning library.
2. Add an LSTM layer to the model defining the number of nodes (24), input shape (100), rectified linear activation function (relu).
3. Add a Dense layer for 24 input nodes and ‘relu’ activation function.
4. As we predict single value output, add a Dense output layer of 1 node.
5. Compile the model with mean square error (mse) loss function and ‘adam’ optimizer.
6. Fit the model with training dataset for number of epochs (100) and batch size (100).
7. Create an ensemble of 6 models by following the steps 1 to 8.
8. Get prediction ‘yhat’ for each time step (day) from all the models of the ensemble.
9. Calculate the ranges (lower level, mean and upper level) of each prediction.
10. Calculate uncertainty of the model for each day by using the set of yhats using the uncertainty calculating formula explained in 3.7.

---------------------------------------------------------------------------------------------------------------------

Algorithm-1: LSTM Model

**3.6 ARIMA**  
An Autoregressive Integrated Moving Average (ARIMA), is a statistical analysis model that uses time series data to either better understand the data set or to predict future trends. A statistical model is autoregressive if it predicts future values based on past values. It is a very popular technique for time series modeling. It describes the correlation between data points and considers the difference of the values. ARIMA models work better with the following assumptions –

1. The data series is stationary, which means that the mean and variance should not vary with time. A series can be made stationary by using log transformation or differencing the series.
2. The data provided as input must be a univariate series since it uses the past values to predict the future values.

The model has three major components which come from its name – AR (autoregressive term), I (Integrated term) and MA (moving average term). Let us try to briefly explain each of these components –

* AR term refers to predicting the next value using the prior values of dataset. The AR term is defined by the parameter ‘p’ in ARIMA.
* Integrated(I) term represents the number of times the differencing operation is performed on series to make it stationary (i.e., data values are replaced by the difference between the data values and the previous values). Test like ADF can be used to determine whether the series is stationary and help in identifying the d value. Differencing is only needed if the series is non-stationary otherwise, no differencing is needed, and in that case d=0.
* MA term is used to define number of prior/lagged forecast errors used to predict the future values. The parameter ‘q’ in ARIMA represents the MA term.

**3.6.1 Auto ARIMA**Although ARIMA is a very powerful model for forecasting time series data, the data preparation and parameter tuning processes end up being really time consuming. Before implementing ARIMA, it needs to make the series stationary, and determine the values of p and q as stated earlier. Auto ARIMA makes this complicated task simple for us as it eliminates those time-consuming tasks. Below are the steps you should follow for implementing auto ARIMA:

---------------------------------------------------------------------------------------------------------------------

1. Load data: Collect data from the source repository and load into a data table.
2. Preprocess data: As the prerequisite of the model input is to be univariate, drop other columns from the data table and make sure all empty values with NULL so that system does not break during runtime.
3. Fit Auto ARIMA: Fit the model on the univariate series of data
4. Predict values: Make predictions on the validation set by using the prior values.
5. Calculate Series: Calculate series by using the forecasted results in earlier step.
6. Find the lower and upper bound of the series which will be used to calculate the uncertainties of the prediction.

---------------------------------------------------------------------------------------------------------------------Algorithm: ARIMA Model

**3.7 Uncertainty Data Generation**

Uncertainties are calculated from the ranges of predicted values for every time step (day) during the specified 200 days of forecasting period. That means we have a lower bound, mean and upper bound of the predictions for each time step. So the difference between upper and lower limit is the grey area of model prediction. Then find the maximum difference to set out the domain of the difference. Finally, divide each difference by maximum difference and multiply by a scaling factor to keep the maximum result in single digit. Here is given the steps to find the uncertainties using the machine learning models:

1. Read data from filesystem (excel file) to Data-Frame
2. Select Fields for which we need to generate uncertainty data
3. Create Machine Learning model for MLP/CNN/LSTM
4. Split data into training and test set
5. Train model with training set
6. Use model to get predicted or forecasted results
7. Find uncertainties or prediction error from model
8. Continue step 3 to 7 for each field and each model
9. Store uncertainty data as json in filesystem

Algorithm-1: calculate uncertainty using predictive models

**3.7.1 Uncertainty Data Scaling**

We have shown top-level algorithm in the above section to generate uncertainty data. Since the uncertainty values are larger to accommodate in display, so it needed to scale in certain level. The following pseudo code is used to scale the uncertainty data.

1. country\_avg\_error = pred\_errors\_of\_all\_dates/number\_of\_days
2. max\_error = find\_max\_error(all\_country\_avg\_errors)
3. scaling\_factor = 7
4. country\_uncertainty = country\_avg\_error \* scaling\_factor / max\_error;

Algorithm-2: data scaling

**3.7.2 Snapshot of uncertainty data**

Since the pandemic affected all the countries of the world and there are more than 200 countries, so we have trained the models for top 100 countries which were infected severely. Based on that setup, we have sorted the countries by obtained uncertainties in both ascending and descending orders. The following two tables shows the top 10 uncertainty attaining countries and the bottom one shows the lowest 10 uncertainty attaining countries.

**3.7.3 Top 10 uncertainty countries using MLP model**

|  |  |  |  |
| --- | --- | --- | --- |
| **Country** | **Actual Count** | **Predicted Count** | **Uncertainty** |
| United States | 14,851,118 | 15,652,300 | 7.00 |
| India | 15,693,425 | 7,409,636 | 4.28 |
| Brazil | 7,219,982 | 7,409,636 | 3.64 |
| Kazakhstan | 667,009 | 651,009 | 2.43 |
| France | 2,088,610 | 2,307,005 | 2.15 |
| Peru | 432,034 | 546,901 | 1.28 |
| Germany | 1,700,161 | 1,599,684 | 1.21 |
| Spain | 1,542,012 | 1,510,467 | 1.07 |
| Turkey | 3,645,288 | 3,389,016 | 1.03 |
| Argentina | 2,352,216 | 2,450,255 | 1.02 |

Table-3: Top uncertainty countries in the ordered list

**3.7.4 Lowest 10 uncertainty countries using MLP model**

|  |  |  |  |
| --- | --- | --- | --- |
| **Country** | **Actual Count** | **Predicted Count** | **Uncertainty** |
| Qatar | 36,256 | 36,796 | 0.013 |
| Albania | 62,292 | 65,515 | 0.016 |
| Estonia | 90,950 | 89,900 | 0.017 |
| Egypt | 118,376 | 124,175 | 0.019 |
| Moldova | 103,270 | 101,832 | 0.019 |
| Australia | 161,819 | 147,134 | 0.021 |
| Algeria | 86,238 | 82,121 | 0.022 |
| Singapore | 178,151 | 175,400 | 0.025 |
| North Macedonia | 57,447 | 57,420 | 0.037 |
| South Korea | 277,584 | 274,766 | 0.037 |

Table-3: Lowest uncertainty countries in the ordered list

From the above two tables, it is clearly noticeable that uncertainty is completely independent on the number of cases (Actual Count). For example: United States has lower number of cases than India but achieved higher uncertainty than India. Again, Kazakhstan and France exhibit same behavior and if we examine other countries then surely, we will get more.

**3.7.5 Uncertainty Comparison among Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Country** | **MLP** | **CNN** | **LSTM** | **ARIMA** |
| United States | 7.00 | 7.00 | 3.44 | 7 |
| India | 4.28 | 0.61 | 7.00 | 3.52 |
| Brazil | 3.64 | 0.51 | 3.24 | 1.27 |
| Kazakhstan | 2.43 | 0.42 | 0.35 | 0.17 |
| France | 2.15 | 0.31 | 0.81 | 0.56 |
| Peru | 1.28 | 0.23 | 0.28 | 0.22 |
| Germany | 1.21 | 0.19 | 0.50 | 0.51 |
| Spain | 1.07 | 0.19 | 0.67 | 0.33 |
| Turkey | 1.03 | 0.19 | 1.21 | 0.30 |
| Argentina | 1.02 | 0.14 | 1.08 | 0.25 |

From the above comparison table of three different machine learning models, we notice that the uncertainties greatly vary for each country based on the model. There is no country which has identical uncertainty values for all three models. Though the dataset used in each of the models in similar approach, the variation appears due to their internal mechanism of the model algorithms. Since the model superiority examination is not our goal, we are not going to discuss further about it. We use the uncertainty data whatever we obtained from model prediction and uncertainty calculation methods.

**3.8 Architecture of CA Representation**

As we have seen lateral chromatic aberration example in Figure-1 (Chapter 1) where all lights with different wavelengths does not focus to the same convergent point because lights having shorter wavelength refract more than the lights with longer wavelength. Inspiring from that analogy, we can consider a circle represents the predicted number of new cases for a country in a specific day. But since there is associated uncertainty of the prediction, a single circle will not be sufficient to represent bivariate (number of cases and uncertainty) distribution. That’s why we need a little change on the arrangement where instead of single circle if we use three different circles of RGB color channels, apply lateral shifting of the center of the circle by the amount of uncertainty and blend them together then the resultant outcome would be a perfect representation of CA. The following Figure-8 shows such a geometric arrangement of unit radius circle.

**Diagram, engineering drawing

Description automatically generated**

(x, y)

Figure-8: Geometric concept

To draw a circle representing aberration as per above explanation if we draw 3 circles, let’s call them 3 chromatic circles. Then we can conclude the technique with the following algorithm –

---------------------------------------------------------------------------------------------------------------------

1. Let’s consider the center of the target circle at (x, y).
2. Radius (radial offset) of the circle is ‘r’ represents uncertainty.
3. Draw first chromatic circle with color (R, 255, 255) with a shifted location of (x, y + r) where ‘r’ denotes red color channel.
4. Draw second chromatic circle with color (255, G, 255) with a shifted location of (, ) where ‘G’ denotes green color channel.
5. Draw second chromatic circle with color (255, 255, B) with a shifted location of (, ) where ‘B’ denotes blue color channel.
6. Apply css ‘mix-blend-mode’ to ‘darken’ to blend all three circles to get the resultant CA appearance.

---------------------------------------------------------------------------------------------------------------------Algorithm: CA Construction Formula

**3. 10 Example of CA**By using the above formula explained in section 3.9, a resultant aberration is presented with the uncertainty for the country India (IND) in Figure-9 below. Center dark-grey area represents the predicted number of new cases, and the bright colorful edge represents the amount of uncertainty in that prediction.

**A picture containing chart

Description automatically generated**

**Figure 9:** Example CA

**3.10 Texture Pattern Generation**

Texture is extremely common in modern web design, and it can be used in countless different ways in practical applications. Textures in web design can be very subtle, so that the visitor hardly notices, or they can be a central point of the design. In some cases, textures are used to emphasize or deemphasize certain parts of the design. Because of the versatility of textures, they can be used or generated in combination with many other design elements, such as typography, lighting, and colors. There is a subtle difference different between in patterns and textures. Patterns are visual element of geometric and mathematical structures that form consistent and repeated graphic shape on a surface. Visual activity across a surface is a texture when the structure forming the texture is based on irregular and random relationships over given areas. There are various kinds of textures and one of them is visual textures and patterns fall in that category. So, in our perspective we build our textures with the help of SVG patterns where everyday predictions are presented with patterns and the collective outcome for the whole duration will be textures.

A picture containing chart

Description automatically generated Background pattern

Description automatically generated

**Figure 10: Streamgraph Color Filled (left), Texture Filled (right)**

**3.10.1 Slicing plot**

In the above section, the streamgraph is shown in both color-filled version and texture-filled version. To better understand how the conversion is done the following Figure-11 gives a clear insight. We split the flow in horizontal direction and made a slice for every 3 days since horizontal axis represents the time in days. We have also tried by chopping the graph with other number of days like 2, 4, 5, 6, 7 and so on but 3 days gives best result among all. Because if we split it by 2 days then the width of the slice is too small to accommodate the content and if we use higher number of days then the shape of the plot distorted.

**A red and white logo

Description automatically generated with low confidence**

**Figure-11: Sliced Streamgraph**

**3.10.2 Pattern Algorithm**

---------------------------------------------------------------------------------------------------------------------

* 1. Repeat steps – to – for all the countries
  2. Define two sets of alternating colors to distinguish patterns side by side as follows:  
     {0: '#ff0000', 1: '#800000', 2: '#FF00FF'} -> Reddish colors  
     {0:'#008080 ', 1: '#0000FF', 2: '#000080'} -> Bluish colors
  3. Repeat step 3 for three aberration points.
  4. Repeat step 4 for three alternative colors.
  5. Repeat steps 6 to 12 for 10 uncertainty scales [0 to 9].
  6. Define a pattern by using <pattern> tag that defines a graphics object which can be redrawn at repeated x- and y- coordinate intervals to cover an area.
  7. Set an ‘id’ of the pattern element. This ‘id’ is used later during filling texture. We use the following convention to define an id:  
     pattern\_id = 'pat-' + country + '-' + aber\_indx + '-' + rgb\_indx + '-' + uncertainty\_scale

where,

country = name of the country of stream graph

aber\_indx = index of the aberration from [0, 1, 2] from step 3.

rgb\_indx = index of the color channel (0 for red, 1 for green, 2 for blue) from step 4.

uncertainty = uncertainty in the scale of 0 to 9 (normalized to meet the range) from step 5.

* 1. Set width and height of the pattern.
  2. Append a shape of the pattern such as ‘circle’, ‘rect’, ‘ellipse’ etc. In our case, we use ‘circle’.
  3. Set center (cx, cy) and radius (r) of the circle.
  4. Set attribute ‘patternUnits’ to **‘**userSpaceOnUse’ that defines the coordinate system for cx, cy, width, and height.
  5. Fill the pattern by setting a fill color.

----------------------------------------------------------------------------------------------------------------Algorithm: Pattern defining algorithm

**3.10.3 Texture Algorithm**

---------------------------------------------------------------------------------------------------------------------

1. Select the streamgraph container using d3 js.
2. Find the path of the streamgraph by from the value of ‘d’ property.
3. Divide the upper and lower segments of the path and save in two variables.
4. Determine the number of vertexes (coordinates) in each segment (they would be same).
5. Repeat steps 6 to 11 until all vertexes are traversed.
6. Take three vertexes from upper segment, let’s call it p1.
7. Take three vertexes from lower segment, let’s call it p2.
8. Build a new path string by joining p1 and p2 with standard rule of using M (moveto), L (lineto) and Z (closepath).
9. Append a new ‘path’ element into the container svg and set ‘d’ property by the path string.
10. Fill the path with the pattern id (generated by the previous algorithm) with the following syntax (value of fill attribute 'url(#pattern\_id')).
11. Add blend style property ‘mix-blend-mode’ to ‘darken’.

---------------------------------------------------------------------------------------------------------------------Algorithm: Texture generation algorithm