**IMAGES TO SMILES CONVERSION**

Abhinav Garg (MT22002)

Ankita Mahato (MT22013)

Riya Garg (MT22058)

Saksham Nautiyal (MT22061)

Thanmayee Matha (MT22084)

Komaljyot Kaur (MT22105)

INTRODUCTION:

We need to convert the Images of Chemical Compounds into the SMILES representation of the compound.

**SMILES:**

SMILES (Simplified Molecular Input Line Entry System) is a string-based notation system used to represent the chemical composition of molecules. It is a compact and human readable way of describing the structure of a molecule using ASCII characters. In SMILES notation, atoms and bonds are represented by specific characters and symbols.

SMILES notation provides a concise and portable representation of molecular structures, allowing chemists and researchers to easily communicate and exchange information about the chemical compounds. It is widely used in chemical informatics, computational chemistry, and molecular modelling applications. SMILES strings can be generated from molecular structure data and can also be used to reconstruct the molecular structure from the string representation.

Expected Outcome:

There exists and existing model DECIMER which converts the Chemical Compound Images to SMILES representation.

We are trying to develop an improved version of this DECIMER model along with an interactive web interface which will let the user to enter the chemical compound of their interest, and generate its corresponding SMILE accordingly.

BASE MODEL:

* DECIMER is a deep learning model designed for the prediction of chemical properties and molecular representations solely from 2D chemical structures.
* DECIMER Image Transformer is developed using Deep Learning for Chemical Image Recognition using Efficient-Net V2 + Transformer.
* DECIMER 2.2 (Deep lEarning for Chemical ImagE Recognition) is a new version of Decimer whereas the original implementation of DECIMER using GPU takes a longer training time when we use a bigger dataset of more than 1 million images. To overcome these longer training times, many implement the training script to work on multiple GPUs.
* The DECIMER now uses EfficientNet-V2 for Image feature extraction and a transformer model for predicting the SMILES.
* EfficientNet-V2 is an Image Classification Model.
* EfficientNet V2 is an improved version of EfficientNet architecture, which is a family of Convolutional Neural Network (CNN) models designed to achieve high accuracy while being computationally efficient. It builds upon the success of its predecessor, EfficientNet by incorporating new techniques and architectural advancements to further enhance its performance.
* The Transformer is a neural network architecture was originally proposed for natural language processing(NLP) tasks, particularly for machine translation, but has since been successfully applied to various other sequence based tasks as well. The key idea behind the transformer architecture is the use of self attention mechanisms to capture dependencies and relationships between different elements in a sequence. Transformers don’t rely on sequential processing or fixed receptive fields. Instead, they can attend to all elements in the input sequence simultaneously, allowing for more parallel computation.
* The combination of EfficientNet V2 and Transformer are used for chemical image recognition which converts the Images to SMILES.
* It can be done by adapting the hybrid architecture to the specific task of identifying and classifying chemical compounds or molecular structures from images.
* After decoding/ preprocessing the input dataset of images, we can use the EfficientNet V2 as the backbone network to extract features from chemical images. The efficientnet V2 is typically pre trained on large scale dataset like ImageNet, which helps it learn general visual representations. These pre-trained weights can be used as an initialization for the EfficientNet V2.
* On the top of EfficientNet V2, we integrate Transformer layers to capture long range dependencies and sequential patterns within the extracted features. This is achieved by adapting the self attention mechanism of the Transformer to the extracted feature maps.

DATASET GENERATION:

We need to give images as input, hence we need a dataset of chemical compounds.

Since we could not get the images of the Chemical compounds directly,

We have downloaded smiles dataset from ChemBL database. The database contains around 1million smiles compositions along with other information regarding the compounds.

We took a portion of the database 10,000 smiles and converted them to images using rdkit library.

The code used to generate images from the smiles is added in the server in the following path

"/home/arul/img\_2\_smiles/decimer/scrapimages.py"



The images are available in "/home/arul/img\_2\_smiles/decimer/gen\_images" directory.

Hence, the required dataset is generated.

OTHER EXISTING MODELS:

1. <https://github.com/spacemanidol/Image2Smiles>

The above GitHub link takes an image X and produces a caption in SMILES string. The repo consists of three main portions, the data generation code, a RNN based captioning system and a transformer based captioning system.

The data used is the collection of different SMILES molecules. These smiles strings were joined, shuffled and then images were created from each of the strings.

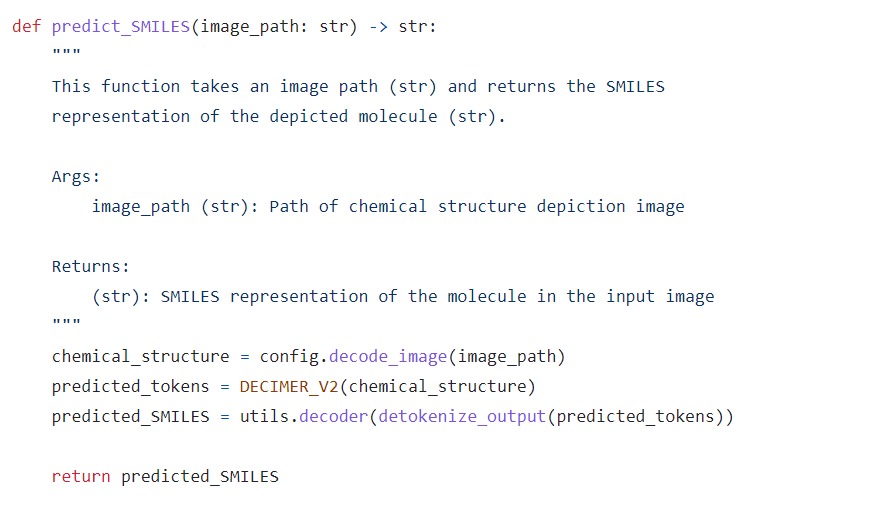
The Validation and Evaluation sets consists of 5,000 image and smiles captions and the training corpus consists of ~1m smile images. Transformer Model was used for training the model.

1. <https://mathpix.com/image-to-smiles>

Mathpix has image to SMILES converter which supports converting printed and handwritten chemical diagrams into canonical SMILES code. It works on both printed and handwritten chemical diagrams.

HOW THE BASE MODEL WORKS:

Given, the input image the DECIMER model predicts the smiles for a chemical compound using the following function “predict\_SMILES”.



The predict\_smiles method which converts the given image into smiles, does the following:

Step – 1:

Takes image as input, then decoding and pre-processing of the images in the following process:

1. Removes the transparent layer from the PNG image with an alpha channel
2. Increase the contrast of the inputted image
3. Converts the given image into black and white
4. Resize the image without lose of information from the image.
5. Converts the image into grayscale and removes the white space around the borders.
6. Add white padding so that the image has a square shape with the width/height of the longest side of the original image.
7. Decode the PNG image into a tensor (primary data structure used in TensorFlow for representing data). The resulting tensor will have the shape corresponding to the dimensions of the image, representing an RGB image.
8. Then resize the tensor to a target size of (512,512) pixels using the gaussian method with antialiasing enabled (helps to reduce aliasing artifacts and smoothens the edges of the resized image, used during the down sampling).
9. Now this resized image should be converted into a format that can be used by the EfficientNet model. The method “preprocess\_input()” is provided by the EfficientNet TensorFlow-Keras implementation which is used for pre-processing the image data according to the requirements of the EfficientNet Models. It performs the preprocessing steps such as NORMALIZATION and CHANNEL REORDERING to ensure that the input image is in appropriate format expected by the EfficientNet model.

Step – 2:

The preprocessed image is given as input for the EfficientNet – V2 + Transformer Model by giving the respective parameters and we get output in the form of tokens predicted.

Step – 3:

After the tokens are generated from the decimer model for the given image in the required format, we detokenize them and extract the SMILES string from that.

After that we decode all the special symbols and then we have the final SMILES string of the given input image.

IMPLEMENTATION:

Code to generate the smiles from images is in the following path "/home/arul/img\_2\_smiles/decimer/test.py".

A corresponding csv file which contains the file name of the image and the respective smiles output is generated.

The csv file is available in the following path:"/home/arul/img\_2\_smiles/decimer/img\_smiles\_map.csv".

TESTING with some random 100 images:

To test if the model is predicting the correct SMILES format or not, we have given around 100 SMILES,

(The csv file which contains the initial SMILE - Img file name mapping is in the path "/home/arul/img\_2\_smiles/decimer/sm\_img\_map.csv".)

converted them to Images,

(The test images are in the following directory "/home/arul/img\_2\_smiles/decimer/test\_img".)

have given images as input to the code and generated the smiles.

(The csv file which is later generated after giving the input images which contains the img file name and smile mapping is in the path "/home/arul/img\_2\_smiles/decimer/img\_sm\_map.csv".)

We have observed a different representation of the SMILES so could not exactly check if they are correct or not because a molecule can be expressed in different SMILES representation.

So in order to check whether the output SMILE composition is correct or not, we have generated them back into images and observed that the same image which we have given as input is regenerated.

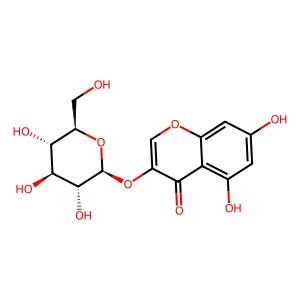
The images generated later are in the directory “/home/arul/img\_2\_smiles/decimer/testsmile”.

**RESULTS:**

**Example-1:**

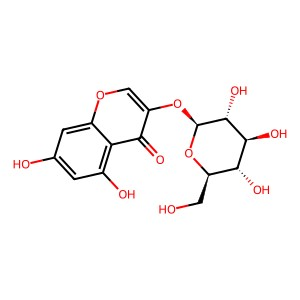
Initial SMILE: O=c1c(O[C@@H]2O[C@H](CO)[C@@H](O)[C@H](O)[C@H]2O)coc2cc(O)cc(O)c12

Generated image by rdkit for the above smile.



Decimer Output for the given image: C1=C(C=C2C(=C1O)C(=O)C(=CO2)O[C@H]3[C@@H]([C@H]([C@@H]([C@H](O3)CO)O)O)O)O

Generating the smile back to image to check if the output is correct or not.

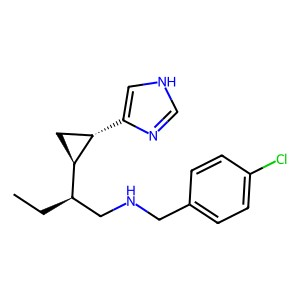


We can say that both the images are the same but just oriented differently.

**Example-2:**

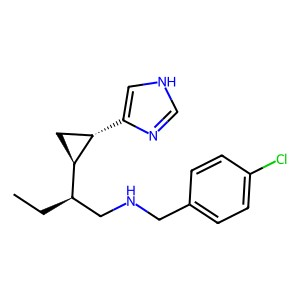
Initial SMILE: CC[C@H](CNCC1=CC=C(C=C1)Cl)[C@H]2C[C@@H]2C3=CNC=N3

Generated image by rdkit for the above smile.



Decimer output for image: CC[C@H](CNCc1ccc(Cl)cc1)[C@H]1C[C@@H]1c1c[nH]cn1

Generating the smile back to image to check if the output is correct or not.

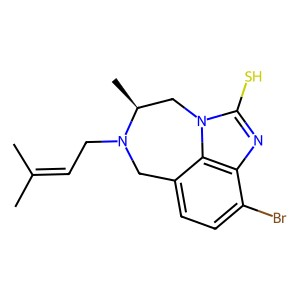


We can say that both the images are the same.

**Example-3:**

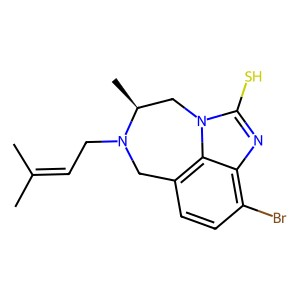
Initial SMILE: C[C@H]1CN2C3=C(CN1CC=C(C)C)C=CC(=C3N=C2S)Br

Generated image by rdkit for the above smile.



Decimer Output for the image: CC(C)=CCN1Cc2ccc(Br)c3nc(S)n(c23)C[C@@H]1C

Generating the smile back to image to check if the output is correct or not.

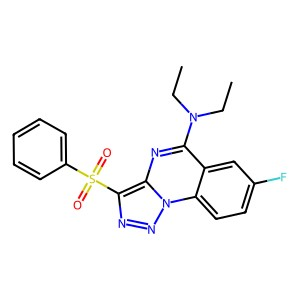


We can say that both the images are the same.

**Example-4:**

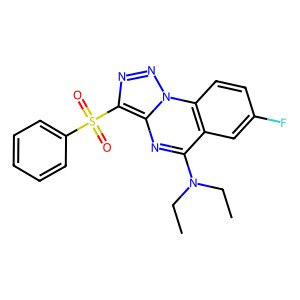
Initial SMILE: CCN(CC)C1=NC2=C(N=NN2C3=C1C=C(C=C3)F)S(=O)(=O)C4=CC=CC=C4

Generated image by rdkit for the above smile.



Decimer output for image: CCN(CC)c1nc2c(S(=O)(=O)c3ccccc3)nnn2c2ccc(F)cc12

Generating the smile back to image to check if the output is correct or not.

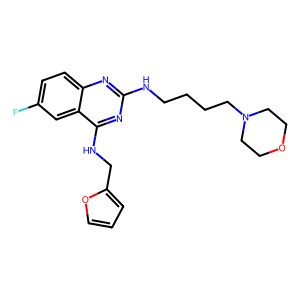


We can say that both the images are the same but just oriented differently.

**Example-5:**

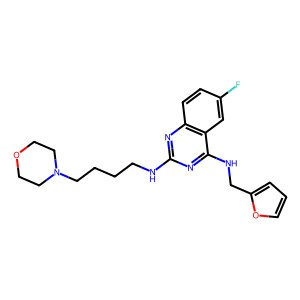
Initial SMILE: C1COCCN1CCCCNC2=NC3=C(C=C(C=C3)F)C(=N2)NCC4=CC=CO4

Generated image by rdkit for the above smile.



Decimer Output for image: Fc1ccc2nc(NCCCCN3CCOCC3)nc(NCc3ccco3)c2c1

Generating the smile back to image to check if the output is correct or not.

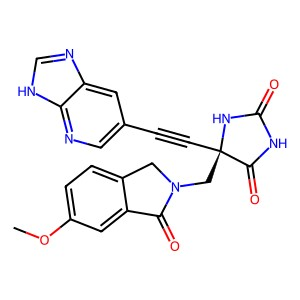


We can say that both the images are the same but just oriented differently.

**Example- 6:**

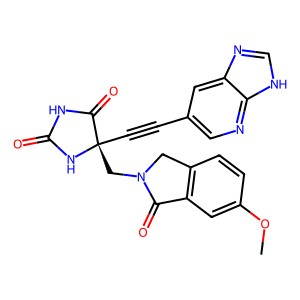
Initial SMILE: COC1=CC2=C(CN(C2=O)C[C@@]3(C(=O)NC(=O)N3)C#CC4=CN=C5C(=C4)N=CN5)C=C1

Generated image by rdkit for the above smile.



Decimer Output for image: COc1ccc2c(c1)C(=O)N(C[C@@]1(C#Cc3cnc4[nH]cnc4c3)NC(=O)NC1=O)C2

Generating the smile back to image to check if the output is correct or not.

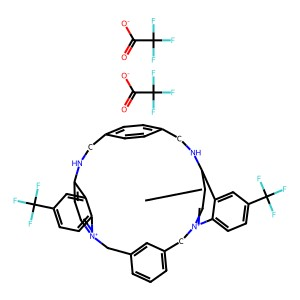


We can say that both the images are the same but just oriented differently.

**Example – 7:**

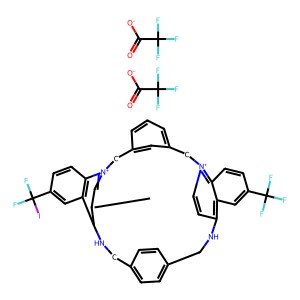
Initial SMILE: C1C2=CC=C(CNC3=C4C=C(C=CC4=[N+](CC5=CC=CC(=C5)C[N+]6=CC=C(N1)C7=C6C=CC(=C7)C(F)(F)I)C=C3)C(F)(F)F)C=C2.C(=O)(C(F)(F)F)[O-].C(=O)(C(F)(F)F)[O-]

Generated image by rdkit for the above smile.



Decimer Output for image: FC(F)(F)c1ccc2c(c1)c1cc[n+]2Cc2cccc(c2)C[n+]2ccc(c3cc(C(F)(F)F)ccc32)NCc2ccc(cc2)CN1.O=C([O-])C(F)(F)F.O=C([O-])C(F)(F)F

Generating the smile back to image to check if the output is correct or not.



We can say that both the images are the same but just oriented differently.

OBSERVATION:

Like this we have checked for all the 100 images and found out the structure we get is same.

A csv file containing the file names of the images which are input to the program and output of the program is generated as is available in the path

“/home/arul/img\_2\_smiles/decimer/comparison\_images.csv”

Input images: "/home/arul/img\_2\_smiles/decimer/test\_img".

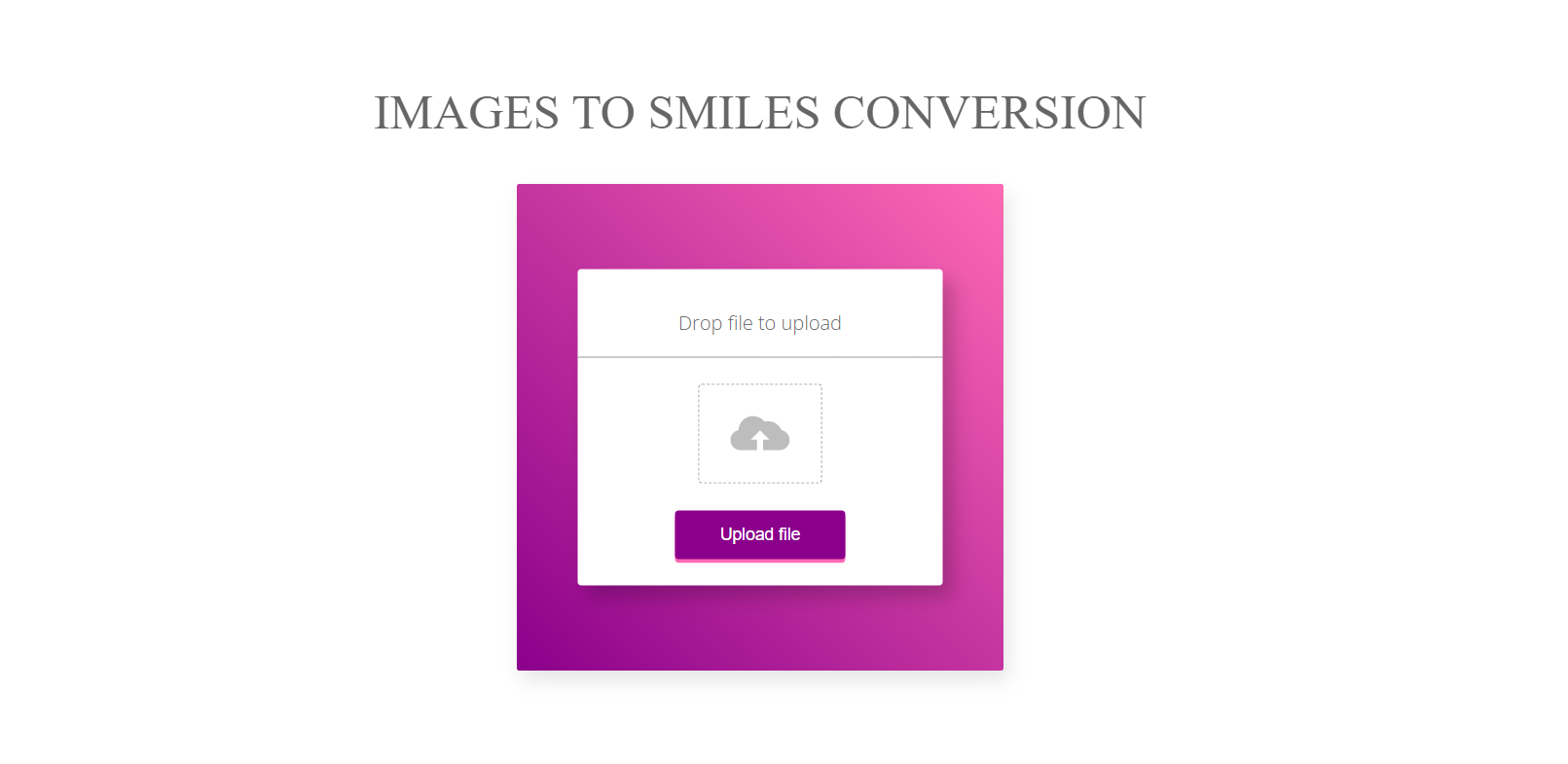
Output images: “/home/arul/img\_2\_smiles/decimer/testsmile”.

WEBSITE:

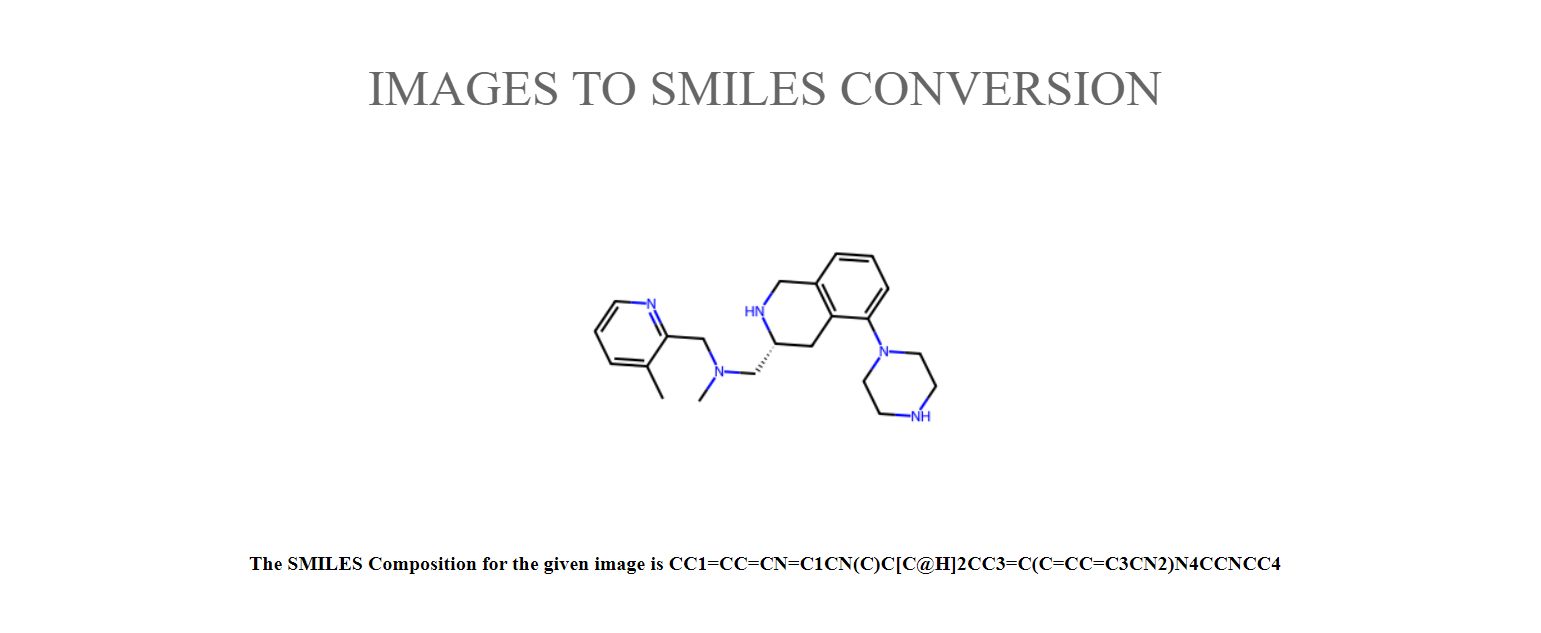
We have developed a user-friendly interface where the user can upload the image of any chemical compound, and get the corresponding SMILES composition of the provided chemical compound. We also save the image provided by the user in the local system to increase the database and help to train the model with the new dataset so that the model can improve its efficiency.

Since it was computationally expensive, we used a local server to work as remote server. When we run the program on local server it dynamically generates a new link every time. User can use website with that link if remote server is working in background.

The following is the sample of the UI where we can upload the images of the chemical compound, once we upload and click on the Upload File button, the algorithm which takes the input file and predicts the SMILES of the given image is executed in the backend.



Once, we click on the Upload File, the following screen will be displayed showing the image uploaded and the corresponding SMILES Composition generated by the Algorithm.



We have tested for the website for many images and it generates the correct SMILE composition of all the images uploaded.