



School of Information Technology and
Engineering at the
ADA University



School of Engineering and Applied
Science at the
George Washington University

LOW QUALITY POSITION-AGNOSTIC IMAGE RECOGNITION USING SEQUENCE MODELS

A Thesis

Presented to the Graduate Program of Computer Science and Data Analytics
of the School of Information Technology and Engineering
ADA University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science and Data Analytics
ADA University

By
Narmin Jamalova
April 2022




THESIS ACCEPTANCE

This Thesis by: Narmin Jamalova

Entitled: *Low Quality Position-Agnostic Image Recognition Using Sequence Models*

has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

Dr. Jamaladdin Hasanov		28.04.2022
(Adviser)		(Date)
Dr. Abzatdin Adamov		28.04.2022
(Program Director)		(Date)
Dr. Sencer Yeralan		28.04.2022
(Dean)		(Date)

ABSTRACT

Information contained in printed or digitized chemical structures is required to research and develop new chemical products. Currently, few automatic recognition and translation systems of structural formulas exist in the industry leading to a lot of manual effort spent on their identification and analysis. Many older printed publications remain on paper due to the amount of effort required to accurately translate them to a computer-friendly format. Machine learning solutions are in development to address this problem yet such issues as drawing style variations and low-quality are not well-accounted for in existing research. This leads to unstable model predictions of incoming data from older sources. Hence, the purpose of this study is to develop a low-quality and position-agnostic chemical structure recognition model to address the problem. The study analyzes and develops several feature extraction methods, custom augmentations that preserve correct textual orientations, applications of random noise to translate images to sequences using LSTM networks with attention. The results show that InceptionV3 extraction method performs significantly better than Autoencoders due to its depth and several differently scaled filters. The baseline image-to-sequence model achieves a minimum Levenshtein score of circa 19 characters on the validation set, which constitutes approximately a 10% error rate. Custom augmentations and lowering of image quality do not significantly impact the score, which can be due to text ordering, random placement of noise and model overfitting on the original dataset.

Keywords: *image-to-sequence, encoder, decoder, LSTM, CNN, InceptionV3, Autoencoder, augmentations, Levenshtein, InChI, chemical structures*

TABLE OF CONTENTS

LIST OF FIGURES	5
LIST OF TABLES.....	6
LIST OF ABBREVIATIONS	7
1 INTRODUCTION.....	8
1.1 PROBLEM DEFINITION	9
1.2 OBJECTIVE OF THE STUDY	10
1.3 SIGNIFICANCE OF THE PROBLEM	11
1.4 REVIEW OF SIGNIFICANT RESEARCH	11
1.4.1 ABOUT THE CHEMICAL IDENTIFIERS	11
1.4.2 TRADITIONAL OPTICAL RECOGNITION OF CHEMICAL STRUCTURES.....	13
1.4.3 NOVEL OPTICAL RECOGNITION OF CHEMICAL STRUCTURES.....	14
1.4.4 IMAGE AUGMENTATION TECHNIQUES	16
1.4.5 SUMMARY.....	20
1.5 ASSUMPTIONS & LIMITATIONS.....	21
2 METHODOLOGY.....	22
2.1 DATASET	22
2.2 WORKFLOW	23
2.3 IMAGE AUGMENTATIONS.....	24
2.4 FEATURE EXTRACTION METHODS	26
2.5 IMAGE-TO-SEQUENCE ARCHITECTURE	30
2.6 SOFT ATTENTION	32
2.7 LEVENSHTAIN DISTANCE.....	33
2.8 NOISE INJECTION	34
3 RESULTS & ANALYSIS	37
3.1 VISUAL RESULTS OF CUSTOM IMAGE AUGMENTATION.....	37
3.2 FEATURE EXTRACTION.....	40
3.3 BASELINE MODEL.....	41
3.4. MODEL WITH CUSTOM AUGMENTATIONS	43
3.5 MODEL ON LOW-QUALITY IMAGES.....	45
3.6 TRAINING ON ORIGINAL VS CUSTOM-AUGMENTED IMAGES	47
4 CONCLUSION & FUTURE WORK	48
5 BIBLIOGRAPHY	51

LIST OF FIGURES

FIGURE 1. IMG2MOL ARCHITECTURE	15
FIGURE 2. IMG2SMILES ARCHITECTURE [16]	18
FIGURE 3. SMART AUGMENTATION ARCHITECTURE	19
FIGURE 4. SAMPLE TRAINING IMAGES	23
FIGURE 5. IMAGE AUGMENTATION WORKFLOW	26
FIGURE 6. PRE-ANOMALY CHEMICAL ELEMENT IDENTIFICATION	26
FIGURE 7. INCEPTIONV3 ARCHITECTURE	28
FIGURE 8. AUTOENCODER ARCHITECTURE	29
FIGURE 9. AUTOENCODER STRUCTURE	30
FIGURE 10. ENCODER-DECODER LSTM IMAGE CAPTIONING MODEL	32
FIGURE 11. ENCODER-DECODER LSTM IMAGE CAPTIONING MODEL WITH ATTENTION	33
FIGURE 12. PROGRAM WORKFLOW CUSTOMIZED BASED ON [27]	35
FIGURE 13. PROGRAM CLASS DIAGRAM BASED ON [27]	36
FIGURE 14. SAMPLE AUGMENTATIONS OF A MOLECULAR STRUCTURE IMAGE (1)	38
FIGURE 15. SAMPLE AUGMENTATIONS OF A MOLECULAR STRUCTURE IMAGE (2)	39
FIGURE 16. PRE- (LEFT) AND POST- (RIGHT) ANOMALY DETECTION IMAGES	40
FIGURE 17. BASIC LSTM IMAGE CAPTIONING TRAINING LOSSES USING INCEPTIONV3 VS AUTOENCODER	41
FIGURE 18. TRAINING LOSS OF A BASELINE MODEL	42
FIGURE 19. AVERAGE LEVENSHTTEIN DISTANCE OF BASELINE MODEL	42
FIGURE 20. TRAINING LOSS OF MODEL WITH CUSTOM AUGMENTATIONS VS BASELINE	44
FIGURE 21. AVERAGE LEVENSHTTEIN DISTANCE OF MODEL WITH CUSTOM AUGMENTATIONS VS BASELINE	44
FIGURE 22. TRAINING LOSS OF MODEL ON LOW-QUALITY IMAGES	46
FIGURE 23. AVERAGE LEVENSHTTEIN DISTANCE OF MODEL ON LOW-QUALITY IMAGES VS BASELINE	46

LIST OF TABLES

TABLE 1. DATA SUBSET ASPECT RATIO STATISTICS

22

LIST OF ABBREVIATIONS

Abbreviation	Explanation
BMS	Bristol Myers Squibb
CAS RN	Chemical Abstract Service Registry Number
CeDAR	Center for Data Analytics Research
CLiDE	Chemical Literature Data Extraction
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
InChI	International Chemical Element Identifier
IUPAC	International Union of Pure & Applied Chemistry
LSTM	Long Short-Term Memory
MNIST	Modified National Institute of Standards & Technology
NIST	National Institute of Standards & Technology
NLP	Natural Language Processing
OCR	Optical Chemical Recognition
OROCs	Optical Recognition of Chemical Graphics
OSRA	Optical Structure Recognition Application
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SMILES	Simplified Molecular-Input Line-Entry System
SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machine
TIFF	Tag Image File Format
TNT	Transformer-in-Transformer
TPU	Tensor Processing Unit
UNII	Unique Ingredient Identifier

1 INTRODUCTION

Chemical product research and development, as well as chemistry education, necessitate the examination of existing printed or digitized chemical structures. These are graphical representations of molecules where a group of atoms is connected via chemical bonds. Atoms and their connections are often depicted through graphs, with nodes acting as chemical elements and edges representing the bonds in-between. Chemical literature contains a whole range of such images containing visual elements (i.e., graph) and accompanying text (i.e., chemical element names). In effect, such illustrations are indicative of the underlying structural formulas, which are line-connected symbols of atoms. Single, double, and triple lines are used to characterize single, double, and triple bonds between atoms.

One of the most pressing problems in this sphere now is the lack of any automated recognition of such structural formulas and their subsequent translations to chemical identifiers. This is particularly true for older printed materials which are not even digitized yet contain important information about molecules. Furthermore, there is no universal database for storing, processing, and accessing this vast amount of chemical structural data for further analysis. The absence of such automated workflow slows down the chemical research and development process and leads to missed opportunities in applying novel tools such as machine learning to extract relevant insights.

To solve the problem of chemical structure recognition, image processing, classification and sequence modeling are all used. One of the unique characteristics when it comes to handling chemical structure data, particularly those contained in older publications, is low-quality of images, the presence of noise in the form of salt and pepper, broken image parts and incomplete textual elements. Moreover, different chemists can draw structural formulas in several ways, whereby the starting point can differ and thus contribute to non-identical illustrations of the same formula. Hence, image processing itself is of much importance to design effective machine learning solutions for image recognition and its translation to meaningful text. Specifically, any designed solution must be able to do the following:

- Effectively recognize low quality images containing chemical structures to be able to model the final output correctly.
- Be able to handle salt and pepper noise, broken image components and incomplete depictions of textual elements.
- Be able to assign the same final output to images drawn in different ways yet representing the same underlying formulas.

This problem is, in effect, an image-to-sequence modeling framework where the final output is a string that uniquely identifies the chemical structure. There are several identifiers in the industry, including InChI (the International Chemical Identifier), SMILES (Simplified Molecular-Input Line-Entry System), CAS RN (Chemical Abstract Service Registry Number) and UNII (Unique

Ingredient Identifier). The listed indicators encode information about molecules in a standard way helping to ease the process of searching for and storing such data.

This project targets the development of an end-to-end product capable of performing the following image processing techniques for automatic chemical structure recognition and translation:

- Extracting features from low quality images with minimum information loss as compared to high quality equivalents.
- Extracting features from images containing noise and broken components with minimum information loss as compared to images with no noise or broken parts.
- Transforming a given image in several ways to approximate the different approaches that can be taken to draw the same formula.

The effects of the above-listed techniques are then tested by applying image-to-sequence models and observing the resulting accuracies.

1.1 Problem Definition

This project aims to address the below-listed concerns expressed in the feedback received from prominent chemists and machine learning experts in literature reviews and via interviews.

- There are no standard databases as of the current timeframe that give enough flexibility to researchers in the chemical structure search and analysis process.
- There is a massive pool of chemical structure data available in earlier printed materials that is not digitized yet.
- The information that is already available digitally is often of low quality and can contain externalities, such as noise or image part breaks.
- Too much manual effort is being spent on analyzing chemical structures and assigning identifiers to the illustrative material.
- Too much manual effort is being spent on maintaining traditional rule-based algorithms that identify chemical structures with no complete documentation of such developed solutions and little understanding of the background process.
- There exist too many ways of encoding the same structural formula graphically and there is no guarantee that the same string identifier is going to be assigned to such varying representations.

1.2 Objective of the Study

The objective of this research is to create a more effective solution to automatically recognize and label molecular structure images, those that are of low-quality and can be positioned in several differing ways, as a requirement to expand the existing knowledge base and increase the efficiency and quality of scientific processes.

Specifically, the goals of the project include the following:

- *Creating a position-agnostic chemical structure recognition and translation system*

Typically, a chemical structure formula can be depicted in several ways. This means that any chemist constructing a given graph can choose to place atoms into different places on an image canvas, hence creating a visually differing representation. Although the representation is visually different, the underlying chemical identifier is necessarily the same. As of today, there are no such systems, to our knowledge and based on our research, that have accounted for this variety of approaches to this problem.

- *Creating a low-quality chemical structure recognition and translation system*

Digitizing older printed materials can lead to the presence of salt and pepper noise on an image, together with erased graphical parts and/or chemical element names, which appear as annotations next to each edge of the graph. There is a need to morphologically transform such images to fill in the broken components and remove pixels that do not carry useful information.

- *Creating an automated machine-learning based chemical structure recognition and translation system*

This means no replications of any rule-based systems for the purpose of assigning identifiers to structural formulas. The aim is to reduce the amount of manual effort required to maintain and update such rule-based systems and validate their outputted results.

The final model output can be any of the previously listed chemical identifiers - however, for the purposes of this research and in keeping with the latest developments, the InChI identifier is chosen. The said indicator uniquely captures the information about a molecular structure with no allowed duplications and helps record this information in a consistent way. This encoding is used to describe a given molecule through information layers including atoms and their connectivities, electronic charges and stereochemistry. A three-step process is applied to ensure the uniqueness of the InChI string:

- normalization (removal of any non-contributing components),
- canonicalization (tokenizing each atom by assigning it a unique label),
- serialization (developing a unique string of mixed textual and numeric characters).

1.3 Significance of the Problem

This problem is of significance since it addresses the concerns commonly faced within the chemical industry (commercial and non-commercial settings, such as in education). For instance, for the drug development process to succeed it is critical to ensure all the simple, complex, and literature-known structural formulas are fully captured in a consistent storage and can be swiftly extracted for useful insights. Collecting, analyzing, and utilizing such published data is usually the first step in designing a new chemical product.

However, for quite some time this collection and analysis process has mostly been manual which created unnecessary bottlenecks by requiring human effort to convert such data to computer-friendly format and further analyze them for classification purposes by utilizing the latest domain knowledge and interpreting style variations. Any written rule-based algorithms, although initially effective, have become too difficult to maintain and update, as well as too hard to understand and time-consuming to verify. At the same time, it is nearly impossible to know for sure whether all the different scenarios have been considered when specifying the classification rules.

Hence, comes the necessity to develop an image-to-sequence machine learning model to automatically recognize and translate such structures. Yet the models currently in development fail to account for low quality and positional variations in the images leading to unstable model predictions for new data coming from especially older publications which are now being digitized. Therefore, to enable stable model predictions image processing must be taken a step further and become more optimized to account for such externalities.

1.4 Review of Significant Research

1.4.1 About the Chemical Identifiers

As already listed, there are several chemical identifiers in use today, including InChI, SMILES, CAS RN and UNII, amongst others. This section describes each of those in greater detail.

The InChI standard was created by the joint effort of NIST (The United States National Institute of Standards and Technology), the InChI Trust and IUPAC (International Union of Pure & Applied Chemistry). The following must-have attributes had to be considered in its development [1]:

- Each identifier must be unique in a sense that it can only have a one-to-one relationship with its object.
- Each identifier must contain valuable information in a hierarchical structure that can be analyzed relatively easily to extract useful insights.
- Each identifier should not be too long and must be humanly and computer-readable in a sense that it cannot contain non-alphanumeric characters.
- Each identifier must be style-invariant in a sense that it should be the same for a given structural formula regardless of the way it is visually represented.

Such structural-based identifiers as InChI are based on an atom model. This means that any molecule can be expressed as its constituent atoms connected by pairwise bonds. Such information is encoded into the standard identifier using textual and numeric symbols. An atom can be of a hydrogen or non-hydrogen type. Each atom has its own chemical element, isotopic mass, charge, radical state, and bonds with other atoms. This data is fed into the identifier in a layered format. For example, for “*InChI=1S/C2H60/c-1-2-3/h3H,2H2,1H3*”, the layers are as such:

- “*C2H60*” following the “*1S/*” prefix is the chemical formula of the molecule.
- “*/c-1-2-3*” starting with the “*/c*” prefix is the carbon or the skeletal connections layer which describes which atoms connect with each other by bonds.
- “*/h3H,2H2,1H3*” starting with the “*/h*” prefix is the hydrogen layer which describes how many hydrogen atoms connect to each of the other atoms.

The three sub-layers described above form the main layer of InChI. There may also be a charge layer consisting of a charge sub-layer following the “*/q*” prefix and a protonation/deprotonation sub-layer following the “*/p*” prefix. Stereochemical layer follows sub-layers starting with any of “*/b*”, “*/t*”, “*/m*” and “*/s*” prefixes and indicates information about the atoms’ spatial arrangement. Additionally, there can also be an isotopic layer. [1]

The InChI standard also implements normalization to ensure that the same identifier is generated regardless of style differences. This is performed by correcting molecule representations to ensure they comply with standard chemical rules (for instance, removing radicals, fixing bonds and protonation patterns). Canonicalization is applied iteratively to avoid issues caused by possible duplications in atom representations. To provide flexibility and compactness, a hashed version of the InChI standard - the InChIKey - was also created [2]. The InChIKey is always of a fixed length which is shorter than the standard InChI and is thus easier to work with on a computer. The entire implementation is through the InChI software package, the source code of which is openly distributed.

SMILES is a line notation that is both human- and computer-readable. It is usually considered to be more user-friendly than InChI in terms of deriving direct meaning from a given identifier string. Computationally, the string is generated by symbols found during the depth-first search in a tree. Atoms map to their usual chemical symbols typically surrounded by square brackets, except for organic elements (such as *O*, *C*, for instance). Bonds are represented by either of “-”, “=”, “#” or “*” characters to indicate single, double, triple, or aromatic bonds, respectively. Hydrogen atoms are not typically entered into a SMILES string as the underlying software is able to derive hydrogen connections automatically [3]. Branch symbols are placed within parentheses while atom charges are kept inside brackets. Ring structures are determined with numbers being placed before the opening and after closing ring atoms. The carbon-to-carbon bonds are usually omitted, as well, due to the ability to derive them implicitly. For instance, “*C1=CC=CC=C1*” is a SMILES string for

benzene. Using the rules, one can observe that there is a ring present in the molecule, with double bonds in-between some atoms.

One of the major shortcomings of this standard is that it allows for duplications and is non-unique. The same molecule can have more than one SMILES representation due to several varying algorithms in place, developed at different intervals by various organizations. At the same time, the source code is not publicly available and so it is difficult to judge which algorithm should prevail over others [4].

CAS RN was developed by the Chemical Abstract Service to map to every open-source chemical structure. Unlike InChI and SMILES, this identifier is simply a unique index in a common repository and does not contain any valuable information about the molecule itself. However, it is still useful to ease the search and store processes in databases and search engines and can handle up to one billion chemical substances [6]. At the same time, it makes it relatively easy to combine and index all open-source information about any molecule in one place.

UNII was created by the Global Substance Registration System of the Food and Drug Administration (FDA) and is openly available for usage. Information about the underlying molecular structure is used to produce the UNII for a chemical substance. UNII is always of a fixed length. The shortcoming of this identifier is in its inability to capture broader information about a given molecule, and the possibility of duplication [7].

1.4.2 Traditional Optical Recognition of Chemical Structures

Conventional molecular structure recognition methods have historically been rule-based. Rule-based approaches rely on pre-written chemical “laws” to classify a given substance as one molecule or another. OROCS, Kekule, ChemOCR, OSRA and CLiDE, described below, are examples of such methods.

OROCS (Optical Recognition of Chemical Graphics) was released by IBM in the 1990s. An image of a chemical substance at 300 dots per inch resolution is fed into the system. Nine steps must be followed to arrive at a connection table like an adjacency matrix for a chemical structure graph. Separation of connected elements is performed on the image using polygon-shaped bounding boxes. A connected components algorithm is utilized for the purpose of separating graph elements from atom names [8]. Boxes that appear to be a part of the same connected component are merged into one element. Vectorization (i.e., forming connections between elements), segmentation and image clean-up is then applied as the last preprocessing step for OCR (Optical Chemical Recognition). [9]

Kekule method, also developed in the 1990s, scans the image containing a chemical substance and segments the actual graphical part into a *TIFF* format. After that, image thinning, vectorization and smoothing are applied. Vectorization, in this context, means a coordinate-like representation of image elements in a vector. Any dashed or strict lines appearing on an image signal the presence of connected components, i.e., edges between atoms. These lines are looked for using such methods as Hough transforms or slope analysis. These edges are analyzed for width and bond orders, which are indicative of extra molecule information. Character recognition is then performed using a multilayered perceptron with adjustments made for rotational variations. A 96% accuracy score is achieved for the character recognition task. The recognized characters, if conflicting, are kept for

further reference and refinement (for instance, a poorly recognized “5” can be either a number or letter “S”). A graph is constructed as a combination of all the gathered data. Each identified character is treated as a node while vectorized information is used to generate edges. Kekule also applies a post-processing step whereby the resulting graph is checked for correctness by the user and any errors get fixed. [9, 10]

CLiDE (Chemical Literature Data Extraction), developed in the 1990s and further extended as a commercial tool, applies polygon-shaped structures onto a binarized image to determine the outer contours of elements, and categorize them as character, graphic or dash symbols. Height and size are used as governing factors to distinguish between chemical element classifications. If atoms can be lined up using a straight line, the line, and a single point for such a line are preserved. Only important chemical structures are saved for further usage, and the textual pieces are then gathered using relative positions. A connection table is constructed once textual components have been resolved and graphical elements have been identified. Atom and super atom labels are identified using an internal database. OCR is performed with a neural network. [9, 11]

ChemOCR combines a pattern recognition algorithm with a support vector machine (SVM) and is based on Java. This tool accepts either an image or a PDF file as input. The input is transformed into individual bitmap pictures if it is in a PDF format. A vaporizing unit is applied to erase parts of the image that are not related to chemical structures. The images are further processed, including conventional augmentations (scaling, rotation, sharpening, erosion, and dilation), extraction of connected components, and OCR detection of molecular characters. Following the vectorization of the images, the reconstruction of chemical structures begins with an expert system analyzing and annotating related components, followed by the application of rules to cluster those components together. [12]

OSRA method is another rule-based approach which utilizes such preprocessing techniques as image binarization and scaling at different resolutions: 72, 150 and 300 dots per inch. Segmentation using rectangular structures is performed. A noise factor is computed after the segmentation for each identified segment and if it crosses a predefined threshold, then smoothing is performed. Thinning is applied to make sure that all lines have a width of one pixel. A vectorized form of the image is produced using the Potrace algorithm which finds image control points - atoms. Bonds are detected as vectors connecting the determined control points. [13]

1.4.3 Novel Optical Recognition of Chemical Structures

More recent machine-learning based solutions have been developed for the problem of optical chemical structure recognition. Such methods include Img2Mol, ChemGrapher, Image2SMILES and Deep TNT, amongst others.

Img2Mol architecture is essentially an encoder-decoder network provided in Figure 1. The way it works is as follows:

- An input image is fed into a convolutional neural network (CNN) to extract feature vectors. Training images are generated from such datasets as PubChem and ChEMBL25, using such tools as RDKit, Indigo and OEChem TK to produce canonical SMILES. All images are

resized to a fixed 224x224 resolution. Augmentation techniques are applied to vary bond thickness, style representations, etc.

- The generated feature vectors are fed into a 512-dimensional embedding layer called CDDD. The pretrained decoder then produces a canonical SMILES output string from the CDDD-generated output.
- The model performance was evaluated on a GPU hardware to enable fast training.

The results showcase the model's robustness as compared to other publicly available alternatives. The authors manage to prove that increasing image resolution size does not bring about any performance benefits. To make model predictions more stable, researchers also add in augmentation methods such as blurring images and oversampling on higher resolution images. The achieved accuracy scores range between 45% and 88% on various tested datasets, which outperforms the results of baseline comparative models. [14]

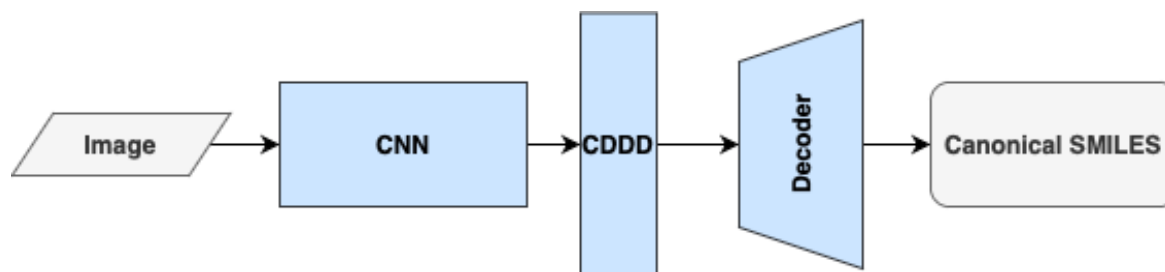


Figure 1. Img2Mol Architecture

ChemGrapher is a deep learning tool that predicts the graph structure of a provided input image directly. The workflow of this method starts with a segmentation task which learns the positions of atoms, types of bonds and charges in the binarized image. The segmentation network consists of eight 3x3 convolutional layers that are succeeded by ReLU (Rectified Linear Unit) functions and a linear layer. Dilation is periodically applied. The next step is to classify the segmented image parts into atoms, bonds, and charge classes, which is performed by employing three separate classification networks: the atom prediction, bond prediction and charge prediction models. All three models all have five 3x3 convolutional layers with periodic dilation application, a max pooling, and linear layers. The number of training epochs varies between 2 and 5. The resulting graph construction error rate ranges between 5% to 16% depending on the complexity of the image. Interestingly, errors happen more often when nitrogen dioxide is present in the picture. The shortcoming of this method is in its requirement to have a labeled pixel wise dataset which may not work for real world images. [15]

Image2SMILES is another technique used to optically recognize chemical structures using transformer networks. Due to the low amount of real-world data available for usage and to account for style variations, the authors had to generate data by applying augmentations to such datasets, as

PubChem. An auto-drawing tool by RDKit was utilized to randomly rotate images, adjust font, whitespaces, and thickness. At the same time, images are purposefully polluted with noise, random cuts or overlaps to simulate real-world low-quality pictures. All images are then resized to a fixed 384x384 resolution. The modeling framework consists of:

- a ResNet-50 CNN with the final output of size 512x48x48 to obtain image feature maps, and
- an encoder-less transformer with a multi-head attention to output the final SMILES string.

Despite having been trained for two weeks, the number of epochs the model was trained for is five, which does not promise a converging result. Still, the results are quite good with a 99% accuracy on the training set and a similar accuracy score on an internal validation set. It is worth noting, however, that the validation set is not representative of real-world data due to being synthetically generated. The more realistic score on an external validation dataset was estimated at only 90%. [16] A schema of the Img2SMILES architecture adopted from [16] is provided in Figure 2.

Image captioning based on Deep TNT (Transformer-in-Transformer) is a more recent tool implementing an encoder-decoder architecture with optional attention model. This paper predicts the InChI string for a provided input image from the Bristol-Myers Squibb synthetic dataset, achieving a Levenshtein distance score in the range from 0.24 to 2.5 surpassing competitors' scores. On an image processing level, the following techniques are applied:

- Images are denoised and smart-cropped.
- Images are padded to maintain a fixed aspect ratio.
- Images are blurred and randomly rotated to simulate examples from the test set.

The researchers propose an extended version of a basic transformer-in-transformer to achieve better results. Specifically, three transformer blocks are utilized to process an image at different patch sizes. An internal transformer block processes pixel-level (4x4) features of small patch embeddings, a middle transformer block processes small patch (16x16) embeddings, and an external transformer block handles large patch (32x32) embeddings. Positional encoding is used for pixel-level and small patch embeddings. Training is performed for two resolutions: 224x224 for less complex images and 384x384 for more complicated ones. Noisy labels are added to continue predicting a given string despite previously incorrect predictions. Beam search is applied at the inference stage. The shortfalls of this method lie in the inability to correctly predict some stereo-chemical layers and error-proneness around “+/-” signs. [17]

1.4.4 Image Augmentation Techniques

To deal with the issue of having little data or data that do not encompass all possible variations, several techniques are applied in research, data augmentation being one of them. These techniques can be both simple, such as automatic flips and rotations, and complex, such as smart and customized augmentations. Their purpose is to reduce overfitting and increase model accuracy.

Augmentations can take on different forms [18]:

- *Basic manipulations* are image flips and rotations that transform a given image geometrically. An image may be flipped horizontally, vertically or across both axes, and rotated by any angle. This application, however, is not straightforward and should consider dataset context to disable distortions. For example, rotating an image of a number by 180 degrees may not be the best technique for such a dataset since numbers can get mixed up, such as 6 and 9.
- *Color space manipulations* apply color isolations, brightness and hue changes, grayscale binarizations and other methods to alter the appearance of a given image. Such transformations create examples that resemble real-world capturing objects more closely, such as broken or noisy cameras, lighting conditions, etc.
- *Cropping manipulations* can help remove parts of the image that do not provide any valuable data.
- *Noise manipulations* can help control for noise to resemble the real world more closely. For instance, adding Gaussian or salt and pepper noise can help models learn to filter out these patterns in image classification tasks.
- *Advanced filters* can transform the image appearance through the usage of kernels. For example, a kernel filter to sharpen an image and increase its contrast can help produce clearer representations. Randomly erasing certain parts of the image can also help the classification model to focus on various parts of it, not just on some area deemed to be more important.
- *Nonlinear augmentation techniques* generate transformations that are not obvious and often impossible to produce manually. For example, a mix of two images can produce a very strange albeit useful output for training. Although it may make little sense to the human eye, the low-level features such as edges get grabbed more formally using such augmentations.
- *Machine learning based techniques* such as SMOTE, GANs amongst others. These are generative models that learn input image features to create and synthesize new images. For instance, SMOTE (Synthetic Minority Oversampling Technique) finds clusters of features in an image array (i.e., nearest neighbors), and a point is selected randomly in a feature space between a given input and its neighbor. Such an averaging technique forms a synthetic image.

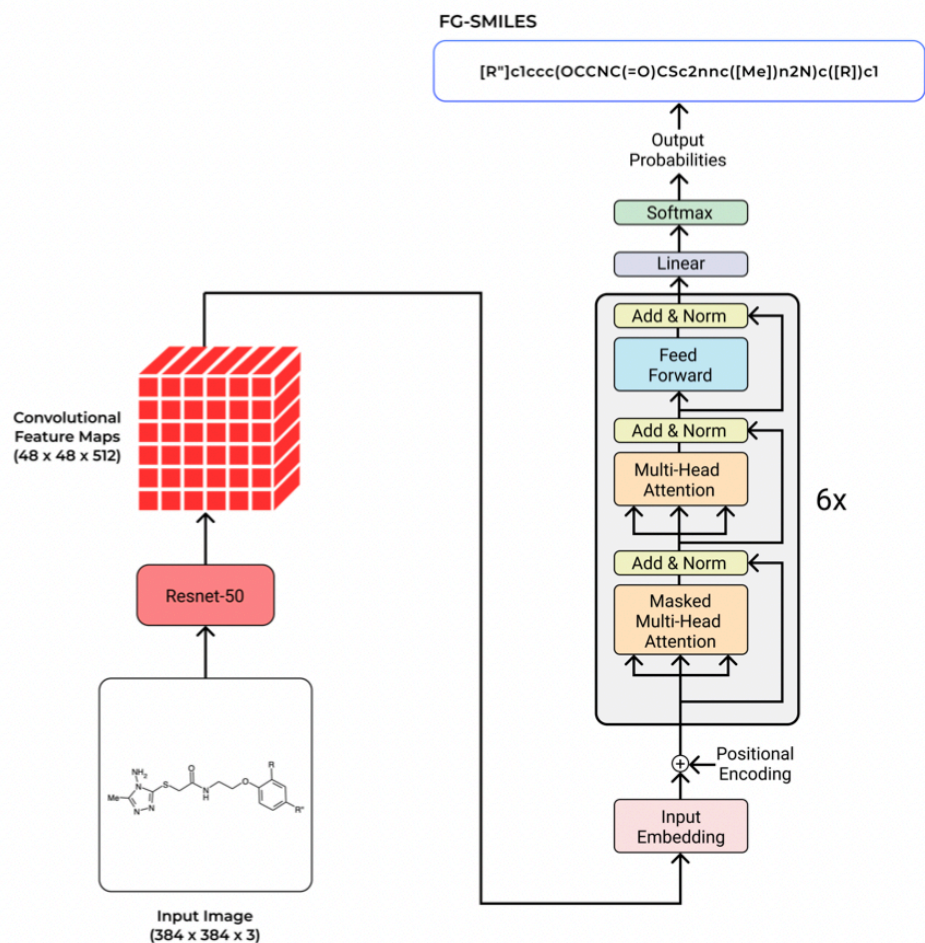


Figure 2. Img2SMILES Architecture [16]

In [19] the authors claim that model performance can be improved by applying smart augmentation techniques. Their work showcases that generating image transformations through the application of a deep learning network produces better and more consistent results than blindly employing a set of random augmentations, such as flips and rotations. The goal is to discover the best augmentation approach for a given set of inputs. The network teaches itself to combine samples into one class which is then used to supplement the dataset for training. The effect of this is that non-standard augmentations can be come up with which would have been very hard to define as a manual preprocessing step. The workflow of the process is such that two deep learning networks are in operation at the same time - one focused on model training (*B*), and the other dedicated to generating more sample data (*A*). *A*'s performance is largely driven by the results coming out of the first

network. Backpropagation of errors from B to A takes place to ensure that B results with the best accuracy score given A 's inputs. A is then forced to generate the best sample data for training. The framework of this method is depicted in Figure 3.

Network A is a fully connected convolutional network, which accepts a k -channel image. This image is a combination of k samples of the same class. A loss function between a randomly selected image from the same class and the generated k -channel image is computed to measure the network's accuracy. Network A 's output, together with the target image, are fed into network B which has two convolutional layers together with batch normalization and max pooling in-between them. Dropout is used to prevent the network from overfitting. This network's loss function is the categorical cross-entropy between output and target. Depending on the result of this loss function, the information feeds back into network A as a backpropagation step. The authors propose another variation of network A as having multiple networks per class so that the results are more specific.

The authors compare the results of their generated samples with manual augmentations which are flips, rotations and blurring. The training gets run for 1000 epochs. The results are such that network B achieves an accuracy of circa 88% without applying smart augmentation. The accuracy score rises to above 90% by applying the smart augmentation techniques. Traditional non-smart augmentation techniques, however, report almost similar results which may be due to convergence. Both methods are advised to be used for more benefit.

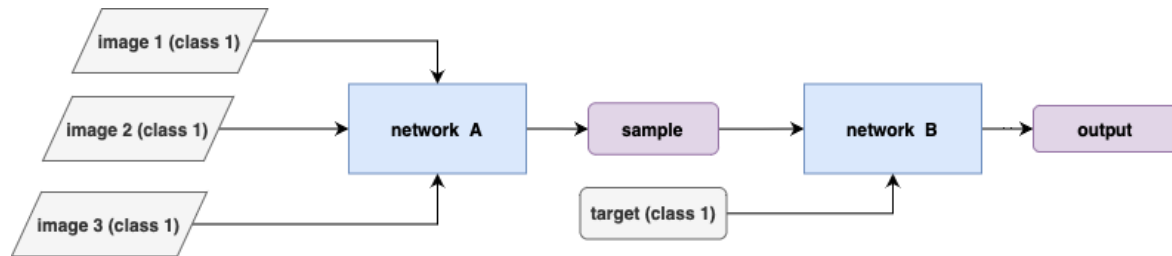


Figure 3. Smart Augmentation Architecture

Similar research compared the performance of smart augmentation with that of using generative adversarial networks (GANs) to produce a synthetic set of images on MNIST dataset. The authors concatenate two images of the same class and use a CNN to produce one synthetic output having the same height and width. A loss function gets calculated between the resultant image and a randomly selected target from the same class. Once a batch of such images gets generated, it gets fed into a classification network which consists of several convolutional, max pooling and batch normalization layers. The classification loss result is then combined with the loss generated by the augmentation network to govern the backpropagation process. The results indicate that such a smart augmentation technique does not provide any significant benefit to MNIST dataset classification since the images are simple anyway, and hence their combination does not bring about any extra advantages. [20]

The problem of training on low quality images has long been standing and several researchers have focused on making image recognition models quality-agnostic. In [source] the authors propose an additional training loss to rebuild high quality images into lower quality ones using decoded low

quality image features. This latter decoder is called to be invertible since it downgrades the original input image. The entire framework is built up as follows:

- A ResNet-50 classifier is used for the image classification task.
- An i-RevNet300 is used as an invertible decoder connected to the classifier via a 3x3 convolutional layer. This gets trained first with high quality images.
- The inverse decoder is moved to the classifier-decoder structure and its parameters get frozen.
- A new classifier is then trained using the inverse decoder's frozen parameters, hence utilizing both high- and low-quality image features.

More recently, the Bristol-Myers Squibb pharmaceutical company conducted a chemical image-to-sequence translation competition on Kaggle [21], whereby some of the winning solutions have built upon the following augmentation techniques:

- Increasing image resolutions progressively by first, training on 224x224 images, moving on to 384x384 and 448x448.
- Random scale, shift, and rotation augmentations to create noisy cut-out subsets and image variations.
- Adding salt and pepper noise for consistency between test and training images.
- Replacing atom symbols with incorrect noisy ones to force the model to learn about correct string sequence predictions.
- Image cropping to remove unnecessary information-less image parts.

1.4.5 Summary

To summarize, the optical chemical structure recognition problem aims at automatically detecting and translating molecular structure images to unique chemical identifiers, such as InChI or SMILES strings. This image-to-sequence task has conventionally been rule-based such that any developed solutions relied heavily on expert knowledge and complicated algorithms for finding connected components, vectorizing images, applying atom and edge detection techniques. To reduce the dependence on chemical rules, simplify and automate this problem, machine learning based models are in development. Such novel techniques as encoder-decoder models and segmentation networks are implemented as part of the latest advances. Images are either extracted for features or segmented to learn positional elements, the outputs of which are then fed into decoder networks responsible for

rendering a descriptive string for a given image. Several researchers have found it beneficial to test model performance on varying image resolutions and to transform pictures in different ways by blurring, randomly shifting, rotating, or thickening pictorial objects. Recent image-to-sequence competitive entries also build upon image transformation techniques, including image resizing, scaling, rotational and flipping variations, noise injections, to synthetically create real-world examples of data. In general, a review of image augmentation methods across several industries suggests that smarter transformations and quality-agnostic modeling can be more powerful in computer vision tasks depending on the data context.

1.5 Assumptions & Limitations

The following assumptions and limitations apply throughout this study:

- Dataset size is limited to approximately 300,000 images of resolutions less than 300x300 due to resource limitations and to speed up code execution.
- Resources are limited to Kaggle’s Cloud TPU (Tensor Processing Unit) 20-hour/week quota and GPU (Graphics Processing Unit) hardware availability accessible at the ADA University’s CeDAR (Center for Data Analytics Research) premises.
- Basic image augmentation techniques are deemed to be insufficient for a quality transformation which preserves graph meaning and encompasses all possible drawing style variations. Hence, a custom image transformer is created for this purpose.
- The custom image transformer only approximates the actual molecule drawing style variations but does not guarantee to encompass all the possibilities. Nonetheless, it provides enough coverage of style variances for modeling purposes.
- The mean and standard deviation of an average bounding box surrounding a chemical element is estimated based on a sample of 500 images. This is deemed to be a good enough coverage for such an evaluation, which is necessary to remove anomalous atom detections. A more detailed description of anomaly detection technique is provided in the next section.

2 METHODOLOGY

This section describes the strategies, tools and techniques used to address the problem statement of this paper. The first module is dedicated to the dataset description. The second module details a step-by-step workflow and code implementation. The third module describes a custom image augmentation strategy. Modules four to seven focus on the image-to-sequence modeling framework. The penultimate module briefly describes noise injection techniques used for lower-quality model training. The final module provides insight into how the image-to-sequence task can be broken down for more granular training.

2.1 Dataset

The dataset used for this study is a set of molecular structure images synthetically generated and shared by the Bristol-Myers Squibb (BMS) pharmaceutical company on Kaggle [source]. The total size of the dataset is about 4 million. Images are all grayscale. As compared to training images, test images contain salt and pepper noise, as well as more corruptions in the form of erased or unclear image parts. Aspect ratios higher than 3:1 are found in about 1.4% of training images and 0.7% of test images. Such unusual image sizes as 1955x72 are observed. The maximum sizes of images in the training and test sets are 1723x1537 and 1838x1578, respectively. The minimum sizes are 117x98 and 93x123 in the same order. Each training set image is mapped to a unique InChI label provided in an accompanying comma-delimited file. The maximum length of the InChI string is about 200 characters.

To speed up training, as well as due to time and resource constraints, a sample of 289,494 images was chosen from the training set for the purposes of this study. The criterion for choosing this subset is a resolution size not exceeding 300x300. The mean aspect ratio encountered across the sample is 0.77. Information about the standard deviation, minimum and maximum aspect ratios observed in this sample is provided in Table 1. Examples of sample images are shown in Figure 4. The chosen subset is further split into two folds for cross-validation purposes.

Table 1. Data Subset Aspect Ratio Statistics

minimum	maximum	standard deviation	mean
0.57	1.0	0.11	0.77

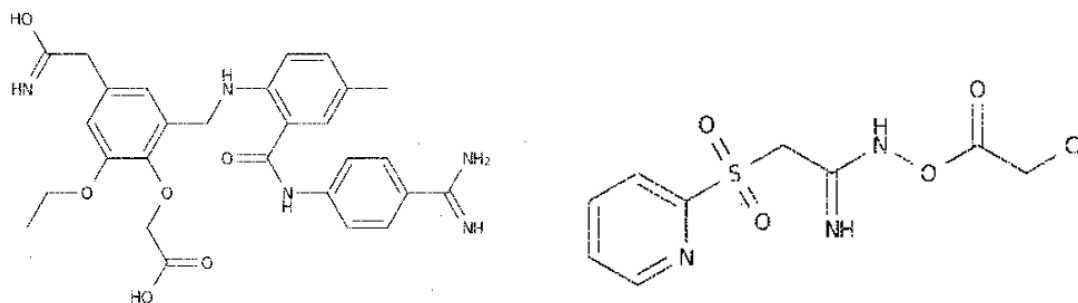


Figure 4. Sample Training Images

2.2 Workflow

The following step-by-step actions are taken to meet the objectives of this study:

1. Extraction of feature vectors from images using InceptionV3 or Autoencoder models.
2. Training and validation of a baseline LSTM image captioning model.
3. Training and validation of an LSTM image captioning model on custom-augmented images.
4. Training and validation of an LSTM image captioning model on low-quality images.
5. Comparative analysis of training results on original versus custom-augmented images only.

The following classes and functions are implemented in the code base to address the workflow plan:

- *Tokenizer*: creates an integer-to-string and string-to-integer mappings.
- *CFG*: sets configurations for modeling and training, such as the number of epochs, encoder size, cross validation folds, and learning rates.
- *get_score*: computes the average Levenshtein distance metric between true labels and predictions.
- *TrainDataset*: defines methods to get the next training image or transformation together with its label.
- *TestDataset*: defines methods to get the next testing image or transformation.
- *get_transforms*: applies image transformations such as resizing, normalizing and conversion to tensors.
- *Encoder*: encodes an input three-channel image into a smaller learned representation using a pretrained CNN.
- *Attention*: defines the attention network to compute weighted encodings.
- *DecoderWithAttention*: applies an LSTM network with self-attention.
- *train_loop*: performs encoder-decoder model training for a specified number of epochs and folds.

The implementation of the code base is in Python 3, using *Keras* and *PyTorch* frameworks, with applications from *OpenCV*, *skimage* and *imutils* packages. The training takes place on Kaggle’s Cloud GPU and TPU resources, as well as physical GPU hardware at CeDAR. Levenshtein distance is used as an evaluation metric.

2.3 Image Augmentations

To account for style variations that different chemists can employ to draw a chemical structure formula, a custom augmenter is necessitated. Practically, a chemist, given a chemical formula, can choose a random starting point and strategy to draw a molecular graph. Despite the chosen graph construction approach, the underlying chemical formula is the same. Hence, any machine learning model should be resilient to style variations and be able to predict the same output identifier regardless of the position of chemical elements on the graph. This resilience can be built up by introducing synthetically created positional variations per image in the data subset.

A custom image transformer workflow built specifically for this study is provided in Figure 5 and it follows the steps described below.

- A given input image array is normalized using a Min-Max normalization type for the purpose of relative pixel intensity value consistency across all images. The array values are rescaled to map with the interval between 0 and 255 to enable conversion to 8-bit integers.
- Thresholding is then applied to the normalized grayscale image so that the output result is a binary array where pixels are either 0 or 255 depending on the input threshold value which is set at 180. This means that for pixels less than 180, their value falls back to 0 (black), and for pixel values exceeding 180, they are set to 255 (white). The output is a binary image with black background and white objects.
- A bitwise *AND* operation is then performed on the original and thresholded images to extract only atoms and edges, and discard areas on the image that do not convey any valuable information about the chemical structure. The image is then inverted back to have a white background and black foreground objects.
- To increase the size of foreground objects and join discontinued image parts, a dilation morphological operation is applied iteratively using a cross-shaped 1x1 kernel nine times.
- A Laplacian edge detection algorithm is employed on the dilated image to detect edges. This algorithm computes a second-order derivative across both 2D coordinate axes as a determiner of the spike in image pixel frequency which signals the beginning or end of an edge.
- All extreme outer image contours are then retrieved and stored. Extreme contour points are those corresponding to the left, right, top and bottom of the detected edges.

- To separate contours corresponding to chemical element names (atoms) from those representing edges (bonds), a character-only retrieval is implemented. For each detected contour, the minimum enclosing circle and rectangle are found. The coordinates of the enclosing rectangle are saved separately for later usage. The contour is then deemed to represent an atom element name only if the circle area is within predefined boundaries and rectangle side difference is less than a fixed threshold. These boundary values are determined empirically. The resulting chemical element contours are then drawn on the original image. Figure 6 shows one example of identified chemical element coordinates.
- The previous step results in misidentification of some image components as chemical elements. For instance, on Figure 6 there are a few boxes around graph edges that are mistakenly counted as chemical elements. To circumvent this and remove false positives, an average box area and standard deviation are computed on a sample of 500 images. The resultant average box size is 108, while the standard deviation is 69.
- Those boxes the size of which exceeds the calculated mean within three standard deviations are removed as anomalous. So, at the end only non-anomalous coordinates are saved for the last processing step.
- Next, basic image transformations are applied with a modification to preserve the textual orientation of an identified chemical element character. To preserve correct text appearance a given image is
 - flipped horizontally but the coordinate-bounded chemical element areas are then re-flipped back horizontally,
 - flipped vertically but the coordinate-bounded chemical element areas are then re-flipped back vertically,
 - flipped diagonally but the coordinate-bounded chemical element areas are then re-flipped back diagonally,
 - rotated clockwise but the coordinate-bounded chemical element areas are then rotated counterclockwise,
 - rotated counterclockwise but the coordinate-bounded chemical element areas are then rotated clockwise.

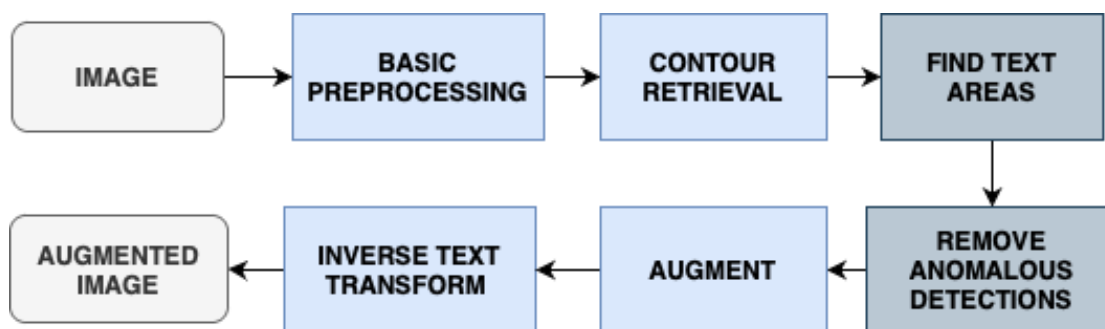


Figure 5. Image Augmentation Workflow

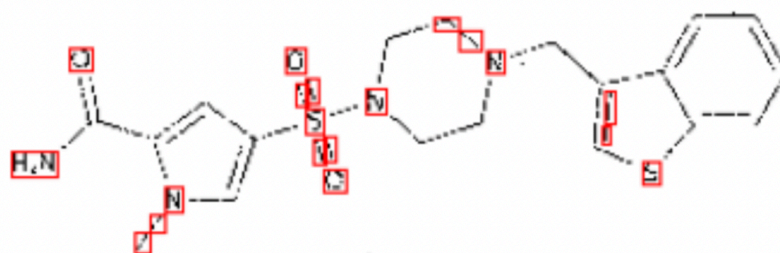


Figure 6. Pre-Anomaly Chemical Element Identification

2.4 Feature Extraction Methods

To convey important data patterns to the recognition model, extraction of image features, such as edges and regions of interest, is a must. In traditional rule-based approaches edge and connected components detection type algorithms were used for this reason. Nowadays, deep learning methods can achieve the same goal and extract even more complex data patterns. Two feature extraction deep learning methods are tested in this study - InceptionV3 and Autoencoder models. Both are examples of transfer learning the aim of which is to capitalize on the weights of pre-trained convolutional models to adapt to the specific problem at hand. In essence, transfer learning comes down to changing and training just the last fully connected layer of a convolutional neural network on a custom dataset. Transfer learning works because images in any classification task share general features with one another, such as having edges around objects, and hence, just the final classification layer can be trained specifically on the custom dataset to add to the already existing model knowledge.

The inception network consists of several modules [22, 23]. The first module learns image features by applying and concatenating the results of four parallel filters: a 1x1 convolutional layer, a 3x3 convolutional layer, a 5x5 convolutional layer, and a 3x3 max pooling layer. The main idea is for the model to learn data at different scales which are then stacked up to match image dimensions. Padding is used in the max pooling layer to ensure correct image shapes. These operations help

reduce the computational costs of image processing and different kernel sizes ensure that all valuable information locations are accounted for. Bigger filters capture information, which is larger in size, while smaller filters capture information less in size. Images of any channel sizes are shrunk to much smaller intermediate volumes consisting of fewer channels, enabling a reduction in the number of multiplications needed to apply convolutional filters. The inception network gets wider instead of deeper.

The second module is dimension reduction which reduces the number of unnecessary parameters and helps avoid overfitting, as well as saves computational resources. This is implemented by adding extra 1x1 convolutions to the first module, right before 3x3 and 5x5 convolutional layers so that the number of channels can be limited. Additionally, another 1x1 convolutional layer is added after the 3x3 max pooling layer. These additional small filters reduce the number of channels entering larger filters by summarizing valuable data. In essence, they are linear projections of a stack of feature maps. Reduction of channels implies a decrease in the number of parameters the model needs to learn.

The latest development of this architecture known as InceptionV3 [23] includes factorization into smaller convolutions. The 5x5 convolution is replaced with two 3x3 kernels. This helps further reduce the number of computed parameters by about 28%. So, if one 5x5 filter returned 25 parameters, two 3x3 kernels result in only 18 parameters. Another update of InceptionV3 is factorization into asymmetric convolutions. The 3x3 convolutions are broken down into 1x3 and 3x1 filters. So, if one 3x3 filter outputs 9 parameters, the updated break-down results in only 6 parameters, which is a reduction of 33%. The updated InceptionV3 architecture is pre-trained on ImageNet dataset for 1000 classes and has three blocks (*A*, *B* and *C*) with different sized kernels all of which apply the newer factorization techniques, described above. The layer architecture of this model is such that an input 3-channel image of size 299x299 is fed into the network and is propagated through:

- a set of three convolutional 3x3 layers where the first layer has 32 such filters with 2 strides, the second layer has 32 such filters with 1 stride and the final one has 64 filters with 1 stride;
- a max pooling layer of 3x3 window size with 2 strides;
- 80 1x1 convolutional filters with 1 stride;
- 192 3x3 convolutional filters with 1 stride;
- a max pooling layer of 3x3 window size with 2 strides;
- inception block *A* which gets run 3 times and is then reduced by applying 1x1 kernels;
- inception block *B* which gets run 4 times and is then reduced by applying 1x1 kernels;
- inception block *C* which gets run 2 times;
- average pooling layers before the main and auxiliary classifiers;
- batch-normalized fully connected layer with ReLU activation function outputting 2048 features;
- a softmax classification layer outputting probability values for each of the 1000 classes.

Figure 7 illustrates the architecture of InceptionV3. In this paper, the last two layers of this network are removed to be able to use the extracted features directly from the learned feature maps.

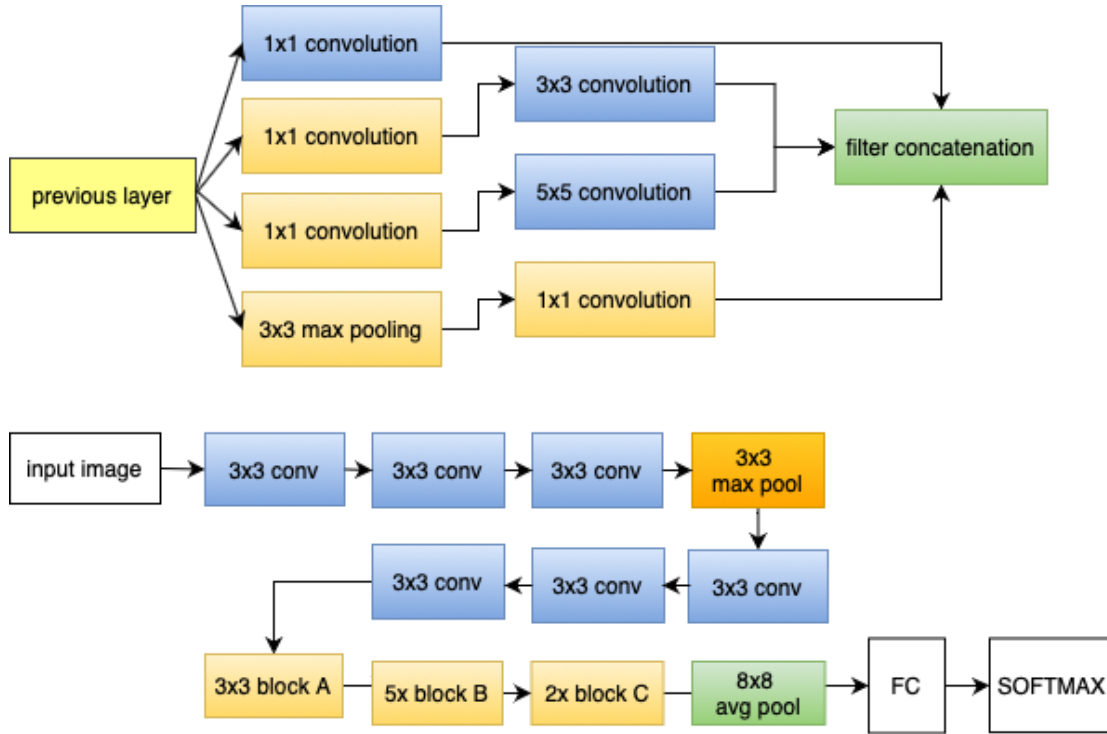


Figure 7. InceptionV3 Architecture

Autoencoders [24] learn simpler representations of image data in an unsupervised way. Their architecture consists of three parts: an encoder, a bottleneck, and a decoder. The output represents a reconstructed input and hence, has the same dimensions. The objective is to learn a representation which will minimize the reconstruction loss and ensure a lower-dimensional view of the original data. The architecture of an autoencoder is provided in Figure 8.

The role of the encoder network is to compress data into a latent space which only focuses on the most important attributes. The compressed result typically looks garbled and nothing like the original input. A necessary condition for learning a good representation is that the data should have dependencies across its different dimensions. If dimensions are independent, learning the latent space becomes almost impossible. Unlike principal component analysis which can learn only linear relationships in data, autoencoders can also learn complex non-linear patterns due to their usage of non-linear activation functions. Moreover, due to the presence of multiple layers, they can learn more efficiently and accurately. The bottleneck layer represents the learned latent space. Decoder networks decompress representations back into the original domain. The reconstruction is lossy which means that the output will lose some data originally present in the input image. Backpropagation is used to train the autoencoders to optimize the learning process. A reconstruction

error (see Formula 1) which is usually simply a mean squared error between the original and reconstructed image is minimized. A regularization term is added to ensure that overfitting on the input data is avoided. Four hyperparameters are set for model training: the code size (bottleneck layer dimension), number of layers, loss function choice (mean squared error or binary cross entropy if input values are in the range from 0 to 1), and number of nodes per layer.

$$E(X_{original}, X_{reconstructed}) = ||X_{original} - X_{reconstructed}||^2 + regularization \quad (1)$$

Autoencoders can have multiple layers before arriving at the bottleneck. The pre-bottleneck layers are focused on learning features with increasingly lower dimensionality, whereas in the layers following the bottleneck features are upscaled back to fit the original image shape. One flavor of the autoencoders is a deep convolutional autoencoder [25], where the encoder has convolution layers followed by a leaky ReLU activation function and batch normalization, and the decoder performs deconvolutions succeeded by a leaky ReLU and batch normalization. The convolution operator is defined as per Formula 2, which is a commutative operation performed on the integral of two functions after one is reversed and shifted.

$$f(t) \times g(t) = \int_{-inf}^{inf} f(\tau)g(t - \tau)d\tau \quad (2)$$

In this study, an autoencoder accepting an input image of size 256x256 and outputting 2048 features is tested. The structure of the implemented Autoencoder is provided in Figure 9.

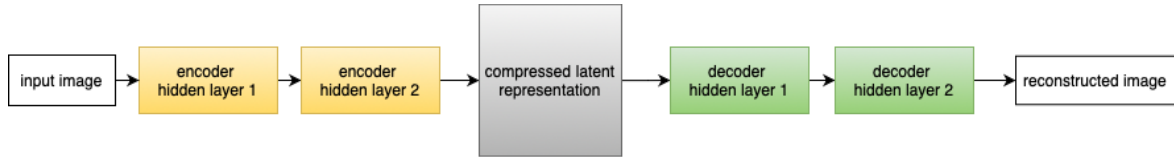


Figure 8. Autoencoder Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 256, 256, 1)]	0
conv2d (Conv2D)	(None, 256, 256, 8)	80
max_pooling2d (MaxPooling2D)	(None, 128, 128, 8)	0
conv2d_1 (Conv2D)	(None, 128, 128, 16)	1168
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_2 (Conv2D)	(None, 64, 64, 8)	1160
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 8)	0
conv2d_3 (Conv2D)	(None, 32, 32, 8)	584
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 8)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 2048)	4196352
Total params: 4,199,344		
Trainable params: 4,199,344		
Non-trainable params: 0		

Figure 9. Autoencoder Structure

2.5 Image-to-Sequence Architecture

The objective of an image captioning model is to produce a description given an image. The input to such a model is an image matrix while the output is a sequence which is a set of variables that has some defined ordering. Sentences are sequences because one word must come after another to generate a meaning. This problem falls into the space of vector-to-sequence architectures with encoders transforming inputs to vectors, and decoders translating the vectors into a sequence. Since the input must be in a vector format, the image can be either flattened to produce a sparse one-dimensional array or passed through a CNN for a dense vector representation. The obtained vector

is passed onto the decoder, which is typically an RNN. At the same time, tokenized words must also be input to the decoder so it can learn how to generate sequences. So, a set of word-embedding vectors that incorporate the meaning of any given word enters the decoder in each iteration. The full encoder-decoder architecture is provided in Figure 10.

One of the most widespread RNNs in practice is LSTM (Long Short-Term Memory) network [26]. It was designed with such goals in mind, as (1) the presence of skip connections so that the current step is affected by previous steps, (2) the replacement of one-length connections with longer ones, (3) the choice of what to remember or forget throughout training, and (4) preventing exploding or vanishing gradient issues. In a typical LSTM, every hidden unit is called an LSTM cell, and connections between them are called cell states. Each LSTM cell has a cell-state vector, and the next cell can choose to either read from it, write to it, or reset it using the gating mechanism. There are three gates in an LSTM:

- the input gate which controls cell updates,
- the forget gate which controls memory settings, and
- the output gate which controls the visibility of current cell information.

Each gate takes the hidden state and the current data as inputs, concatenates the two and applies a sigmoid activation. Sigmoid activations make the LSTM model differentiable. The cell state is then modified by applying new candidate values with *tanh* activations to prevent gradient computational problems.

Since the decoder needs tokens as part of its input, the labels must pass through a tokenization mechanism. In this study, the tokenization of an InChI label is performed according to the following rules:

- The chemical formula part is split into letters and numbers, such that *C13* becomes *C* and *13* as an instance.
- The characters following the carbon layer indicated by the */c* sign are parsed in such a way that any auxiliary symbols, such as slashes or brackets, also form separate tokens. So, */c1-9(2)8-15* becomes */c, 1, -, 9, (, 2,), 8, -, 15*.
- The parsed tokens become part of a vocabulary which is a string-to-integer representation.
- For backward compatibility, an integer-to-string vocabulary is also maintained.

Any label is then turned into a string of characters in the following way: (1) the *<start>* token is appended to the beginning of a string, (2) adding integers mapping to the character in place, and (3) placing the *<end>* token to the end of a string.

The workflow of the program and its class diagrams are displayed in Figures 12 and 13, with the adoption and customization of one of the best Kaggle solutions from [27].

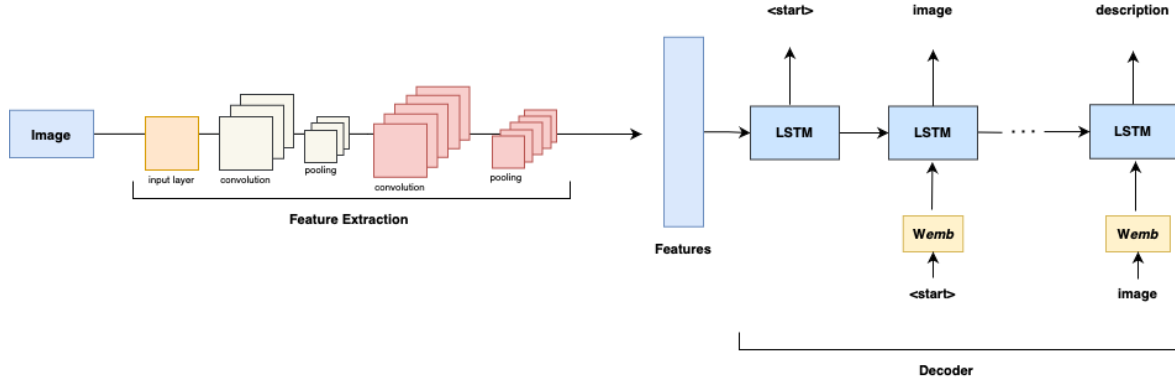


Figure 10. Encoder-Decoder LSTM Image Captioning Model

2.6 Soft Attention

Visual attention is integrated into the image captioning model. The attention model [28] dynamically looks at the feature map generated by the encoder and only takes those parts that are relevant at every stage for the decoder to generate a sequence. The output of the attention model is a context vector which then enters the decoder as an input.

Three inputs enter the attention model – Query (Q), Key (K) and Value (V). Query is the token for which attention score is being computed, and Value is a set of words to which attention is applied. A dot product between Key and Query is computed to produce a word similarity score which is indicative of the probability that a given word appears at a certain index in the sequence. Softmax activation is applied to the computed score to return it as a probability (see Formula 3). The computed scores are used as weights that get applied to the vector which is then fed into the decoder.

$$\text{softmax}\left(\frac{QK^T + \text{mask}}{\sqrt{\text{embedding size}}}\right) * V \quad (3)$$

The attention model architecture is provided in Figure 11.

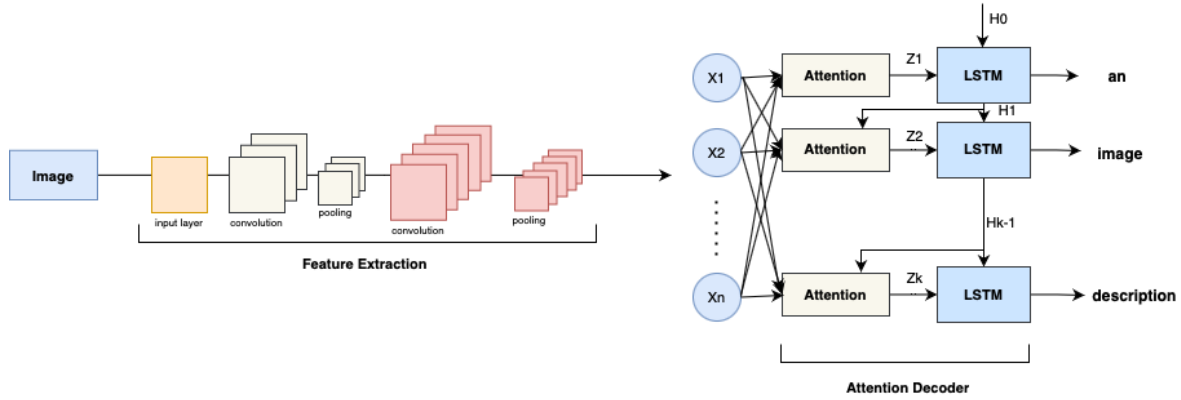


Figure 11. Encoder-Decoder LSTM Image Captioning Model with Attention

2.7 Levenshtein Distance

The Levenshtein distance metric [29] is an indicator of the minimum edit distance between two strings. There are three operations that can be performed to transform one string into another: insertion, deletion, and replacement. The operation that is less costly to implement is the minimum distance.

Practically, there are four actions that may be required when matching two strings:

- *Do nothing*. This is in the case of total match between strings.
- *Deletion*. Deleting unnecessary characters to create a match.
- *Insertion*. Inserting necessary characters to create a match.
- *Replacement*. Replacing certain characters to create a match.

The following properties apply to a Levenshtein distance calculation:

- The minimum edit distance is zero only in the case the two strings are equal.
- The edit distance is at most the length of the longer string.
- The edit distance is at least the difference of the sizes of two strings.

The recursive algorithm for its computation is provided in Algorithm 1.

```
if a.size() == 0 then:
    return b.size()

if b.size() == 0 then:
    return a.size()

check = a[0] == b[0]

return minimum(
    dist(a.substr(1), b) + 1,           // deletion
    dist(b.substr(1), a) + 1,           // insertion
    dist(a.substr(1), b.substr(1)) + check // substitution
)
```

2.8 Noise Injection

To produce lower-quality images to train on, *skimage* library is used. In particular, the following sample code is implemented to add salt and pepper noise to an original image:

```
random_noise(image, mode = 's&p', amount = 0.3)
```

The above returns a floating-point noisy image array.

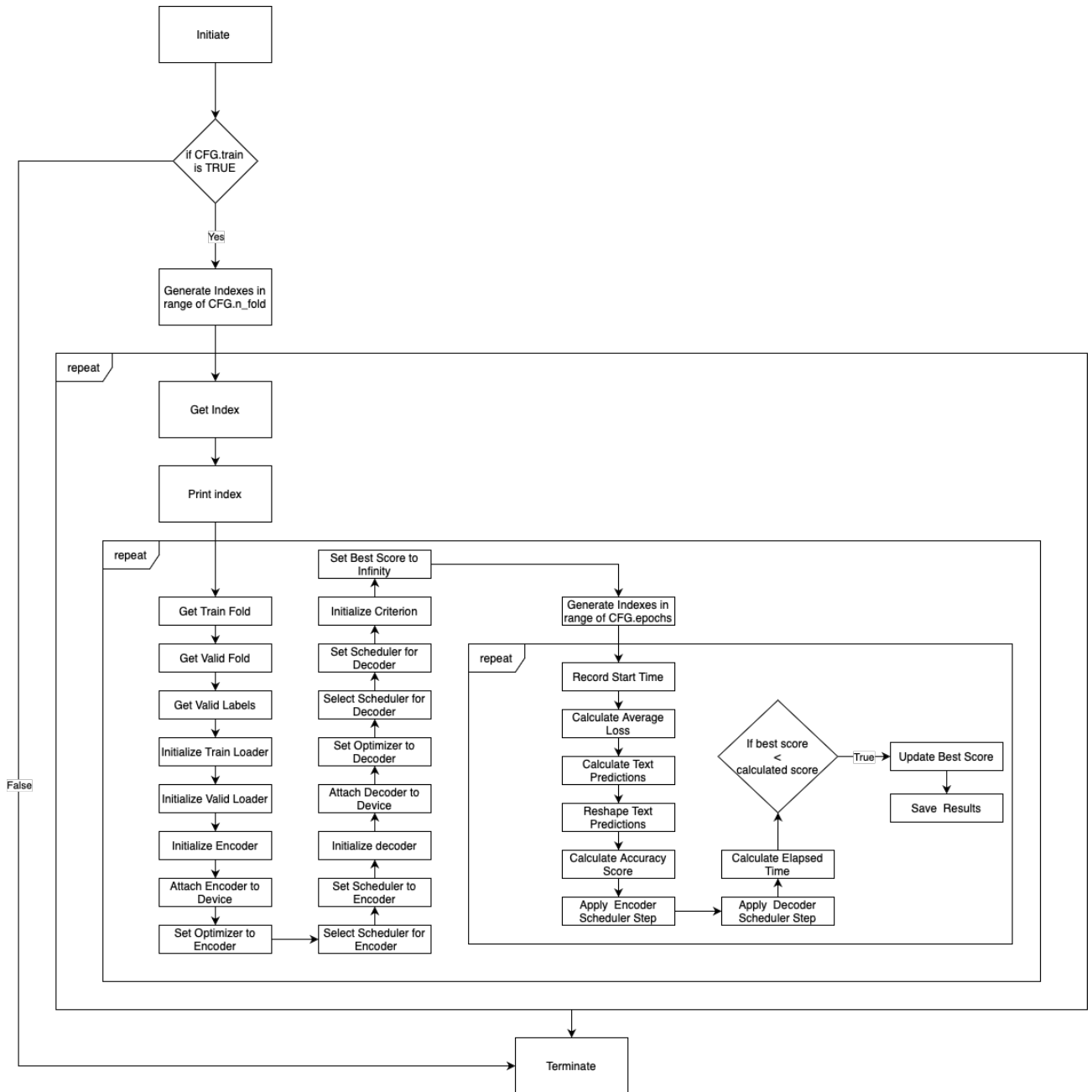


Figure 12. Program Workflow Customized Based on [27]

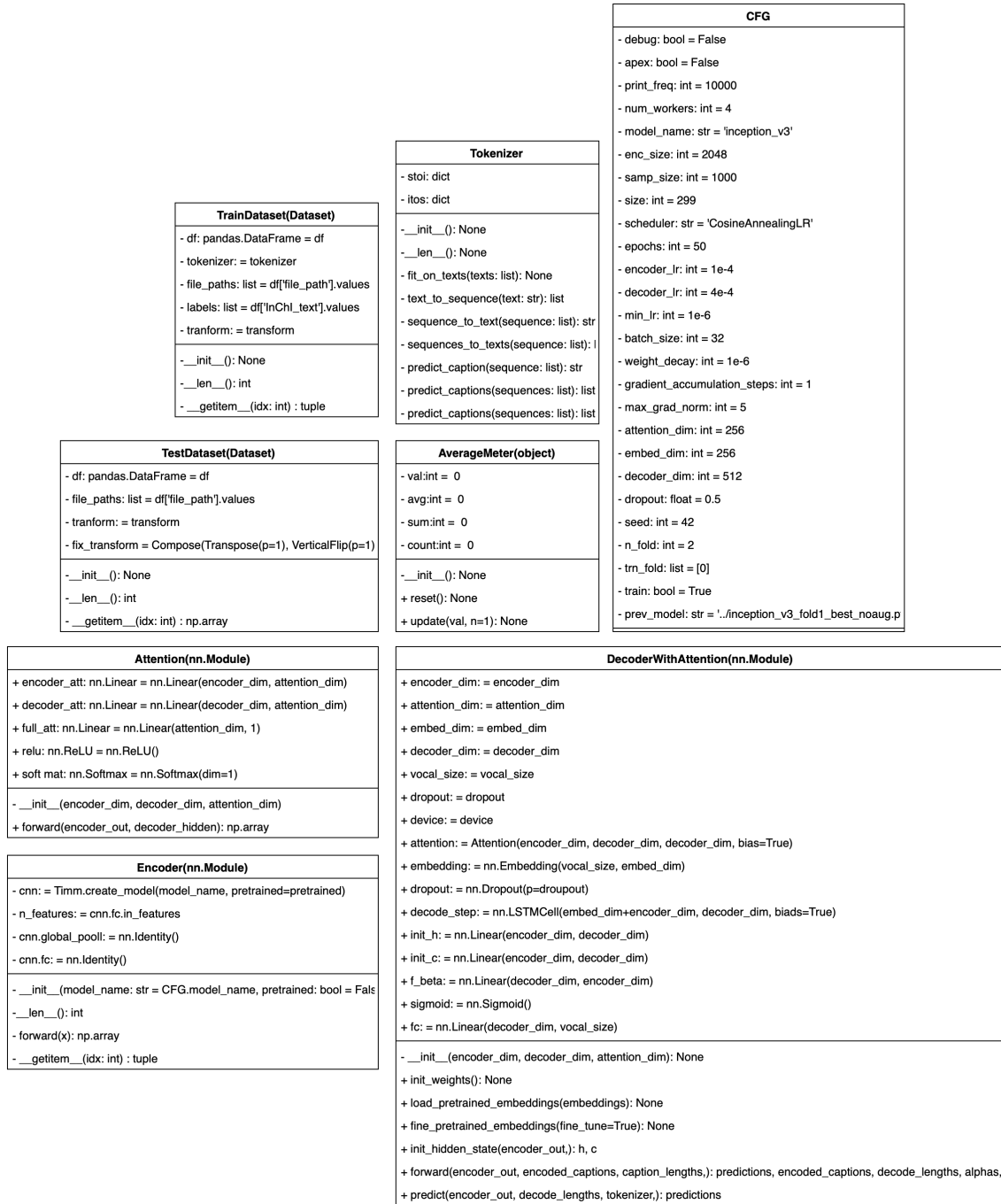


Figure 13. Program Class Diagram Based on [27]

3 RESULTS & ANALYSIS

This section details the results of the study and presents a comprehensive analysis of investigations made. The first subsection presents the visual results of custom image augmentation techniques. The second subsection analyzes the effects of InceptionV3 versus Autoencoder implementations. The third to fifth subsections present the results of a baseline model, and its equivalents with custom augmentations and lower-quality images. The sixth subsection analyzes model performance when trained on original versus custom-augmented datasets.

3.1 Visual Results of Custom Image Augmentation

The sample results of custom image augmentation mechanism are provided in Figures 14 and 15. As observed, although the chemical element characters were identified correctly, some atom names still got incorrectly rotated or flipped. This is especially true for atoms with names exceeding the length of one character since no information was passed on to the algorithm regarding the ordering of a chemical element's components, and the entire name was recognized as one instead of several letters.

For instance, the following chemical element names have problems in correct textual ordering in the sample images: "OH" and "Br". One way of solving this can be through NLP, however, the observed results after the application of such libraries, as *pytesseract*, are not satisfactory. At the current research stage, therefore, this is accepted as a margin of error to handle in future work that would require more advanced object detection techniques.

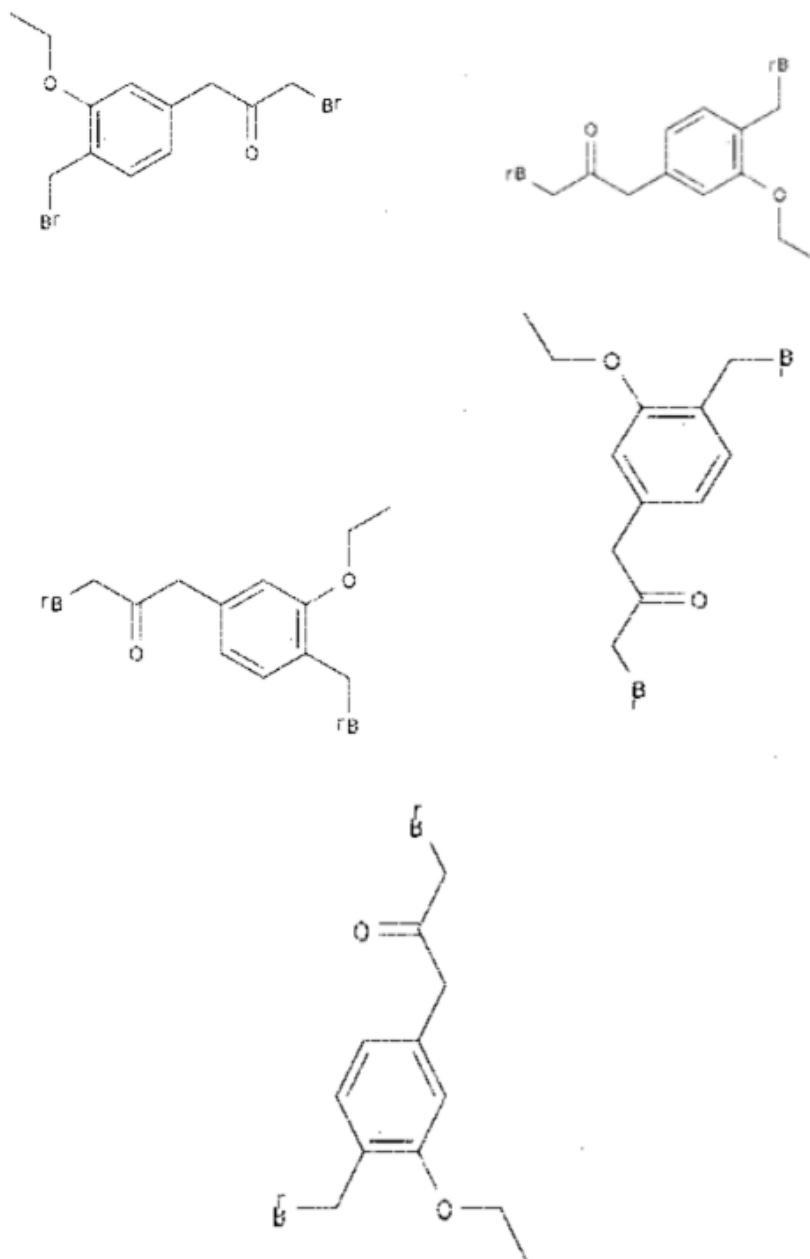


Figure 14. Sample Augmentations of a Molecular Structure Image (1)

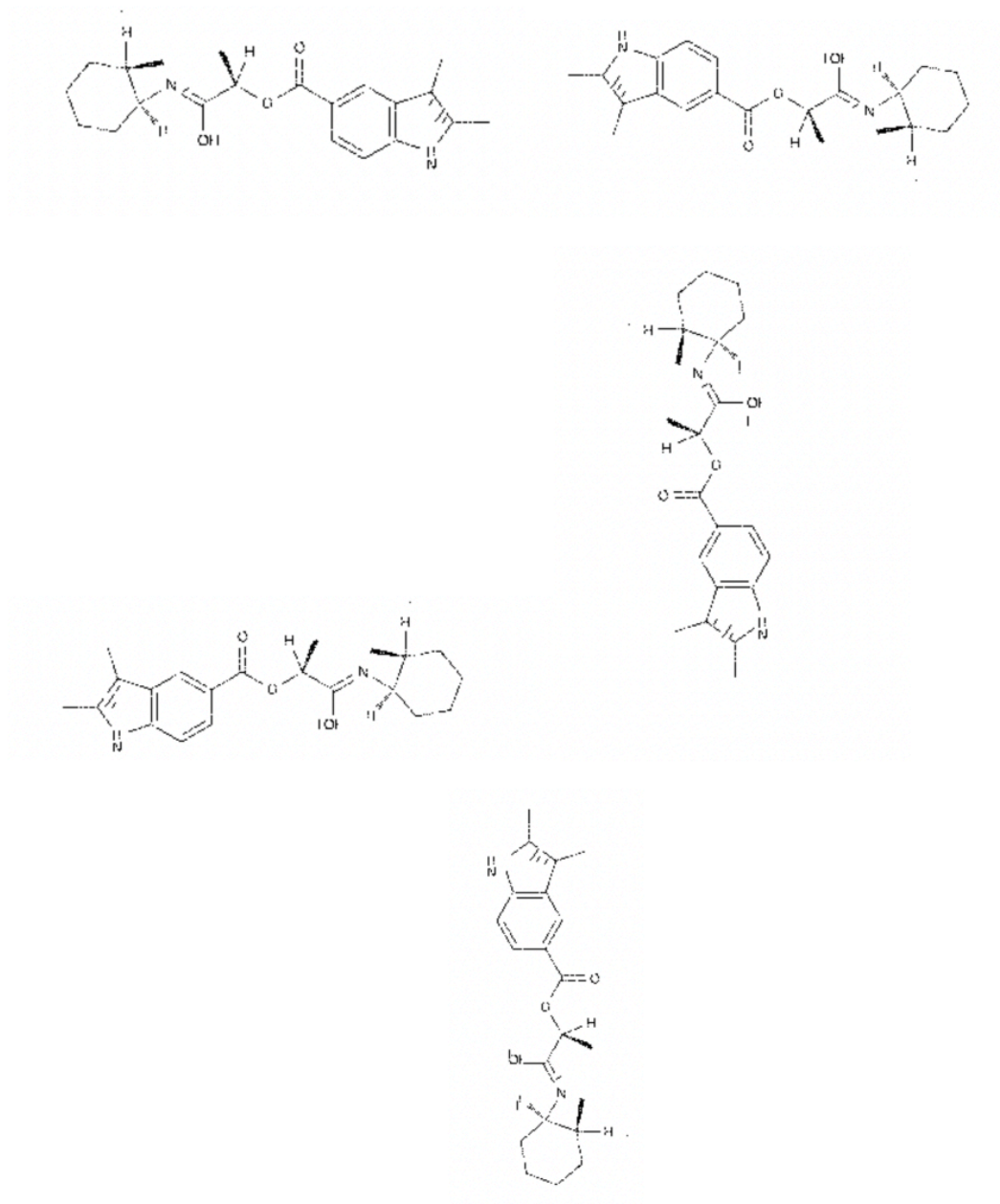


Figure 15. Sample Augmentations of a Molecular Structure Image (2)

Figure 16 can be used to compare the results of pre- and post-anomaly object detection techniques. As observed, although boxes which are originally misclassified as chemical elements are declassified after the application of anomaly detection, a few misclassifications remain, and sometimes the objects are not even detected.

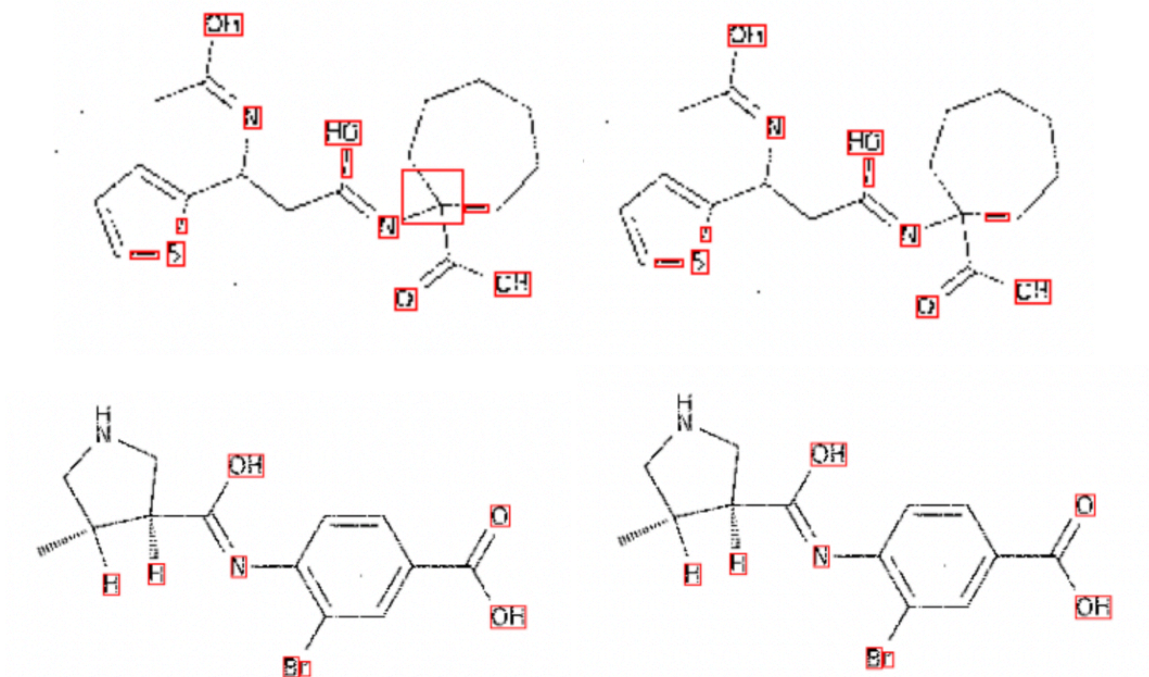


Figure 16. Pre- (left) and post- (right) Anomaly Detection Images

3.2 Feature Extraction

Both InceptionV3 and Autoencoder models are tested in terms of model performance depending on the generated feature vectors. For the purposes of this analysis, a sample of 50,000 original and augmented images is selected from the training set and the resultant feature vectors are saved in a *TFRecord* format for their subsequent entry directly into a basic LSTM decoder network without attention. The training takes place for about 50 epochs with 5 steps per epoch and the batch size is set to 32. The loss function used is the categorical cross-entropy which quantifies the deltas between probability distributions and is used in multi-class classification problems. The resultant training losses are plotted in Figure 17.

In the first few epochs, the Autoencoder-outputted features result in a much higher training loss than the InceptionV3-outputted features. Between the 6th and 20th epochs, the results of both converge, however, the Autoencoder option starts overfitting more than the InceptionV3 post the 20th epoch, with final convergence achieved somewhere between the 45th and 50th epochs. This result

could be because InceptionV3 outputs a more useful feature map which learns valuable image representations at different sizes by using several convolutional filters as compared to Autoencoder which simply compresses the input using standard filter sizes, which nonetheless bring to approximately the same result as the model learns over time. The key point is that the InceptionV3 is much deeper from the very beginning and hence, able to learn important features at different scales throughout its network. Autoencoders, however, keep decreasing the dimensionality of the image as they get deeper and hence, can lose some of the more useful features at different points in their structure.

Due to these results, the InceptionV3 CNN is used as an encoder in all subsequent analyses.

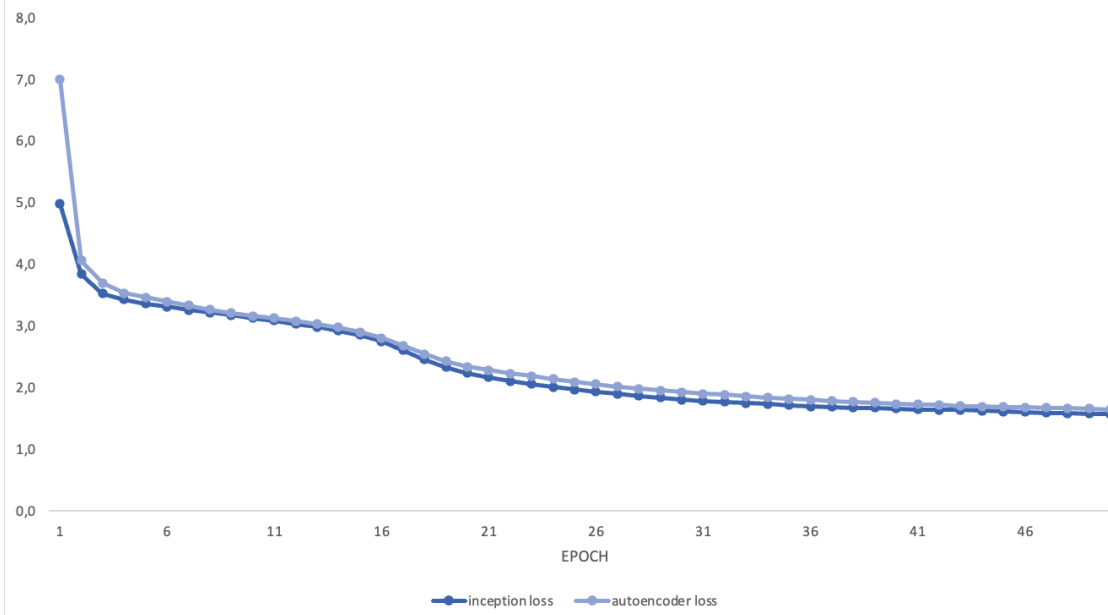


Figure 17. Basic LSTM Image Captioning Training Losses Using InceptionV3 vs Autoencoder

3.3 Baseline Model

The baseline model is an encoder-decoder architecture, whereby the encoder is InceptionV3, and the decoder is LSTM network with attention. The model gets trained for 50 epochs on a sample of 300,000 training images of size less than 300x300, and in two cross-validation folds. The batch size is set to 32, the encoding output size is 2048, the decoder dimension is 512. The input size is 299x299, and Cosine Annealing Learning Rate Scheduler is applied. Adam optimizer is used to fine-tune weights by speeding up gradient descent calculations. The cross-entropy training losses over epochs are plotted in Figure 18, while the resultant average Levenshtein distances on the validation set are displayed in Figure 19.

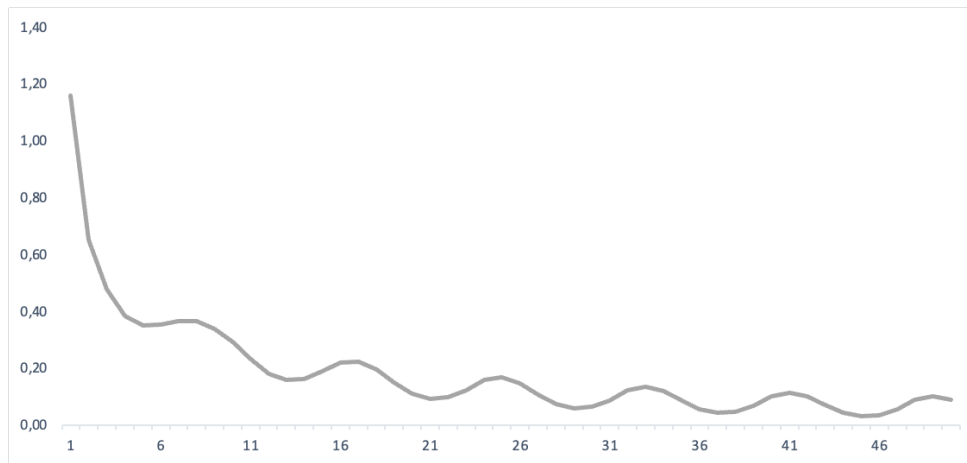


Figure 18. Training Loss of a Baseline Model

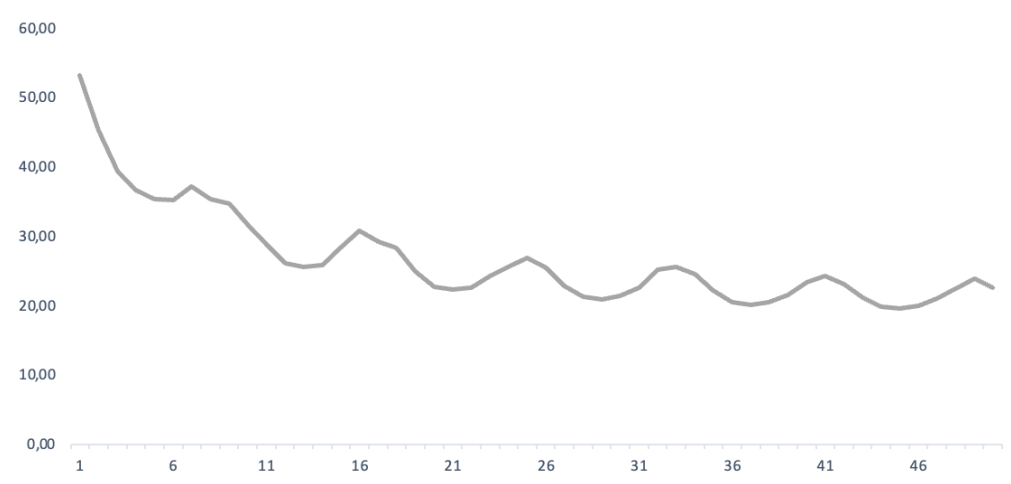


Figure 19. Average Levenshtein Distance of Baseline Model

The minimum achieved Levenshtein distance is 19,5 which takes place in the 45th epoch, and the training loss is also minimum in that epoch. As can be observed, the model learns iteratively and tends to overfit at some intervals after which the learning rate is adjusted and the model starts learning better until the next learning rate reset. This explains the wavy shape of the line on both plots. The training takes place for approximately 24 hours on a GPU and increasing the number of epochs beyond 50 can possibly generate even better results. Nonetheless, the achieved score is quite comparable with some of the most popular Kaggle solutions to the BMS competition. All the subsequent analyses are benchmarked to these outputs, and hence their comparative performance is of more importance than looking for ways to further improve the baseline score.

3.4. Model with Custom Augmentations

Custom augmentations applied on original images result in training losses and Levenshtein distances as per Figures 20 and 21, respectively. The model gets trained using the same parameters as the baseline with a modification made to add custom transformations to PyTorch's *get_transforms()* method in real-time as the model learns.

As can be observed, the resulting training loss is slightly higher than the baseline, although they converge post 11th epoch. The computed average Levenshtein distance is higher across all epochs than its baseline equivalent, with a minimum score of 20,8 achieved in the 45th epoch. This could be due to the presence of extra noise in the form of incorrectly rotated or flipped chemical elements, as detailed previously. The elements containing more than one letter have a significant problem in correct ordering transformations and could contribute to additional image corruptions and a lower model performance. Improving the augmentation methods in such a way that letters are separately identified and transformed in a correct order can significantly contribute to a better score since the results are not too far off anyway.

Considering a maximum InChI length of 200 characters, a minimum edit distance of 20 constitutes only a 10% error rate, which is still quite high yet good enough for a custom augmentation method. Another reason why the error rate is not getting too high is because the same augmentation techniques get applied to validation set images, and hence, the incorrect textual orders are observed in some of them as well, resulting in a training that approximates the validation set.

The obtained results are also quite consistent with the reviewed literature base. Several studies suggest that smart augmentation techniques do not bring any extra benefit if the context to which they are applied is quite simple. Possibly the encoder network is strong enough to extract the needed features, such as straight lines, for the decoder to work in a stable manner. The images indeed are not too complex in terms of lightning or orientational differences, which can lead to the obtained results.

Another reason for the observation could lie in the baseline model overfitting. The baseline model could have learned useless features and memorized the data. When the size of the dataset is increased artificially by applying more transformations, the learning process becomes even more difficult. Most likely, the model itself needs some further fine-tuning to compare the results on a more detailed level. Applying different encoder or decoder models, as well as optimizers can help in this instance. Additionally, real-world examples of several flipped or rotated chemical molecules are missing, and this study is forced to apply the custom augmentation technique to the train set, as well, which is not desirable.

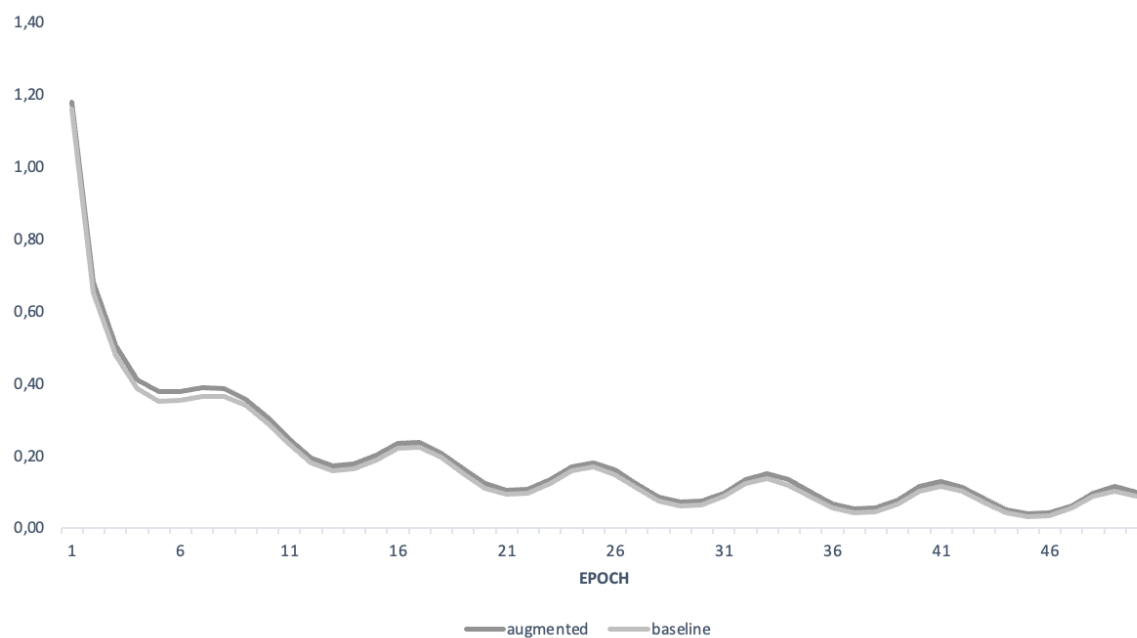


Figure 20. Training Loss of Model with Custom Augmentations vs Baseline

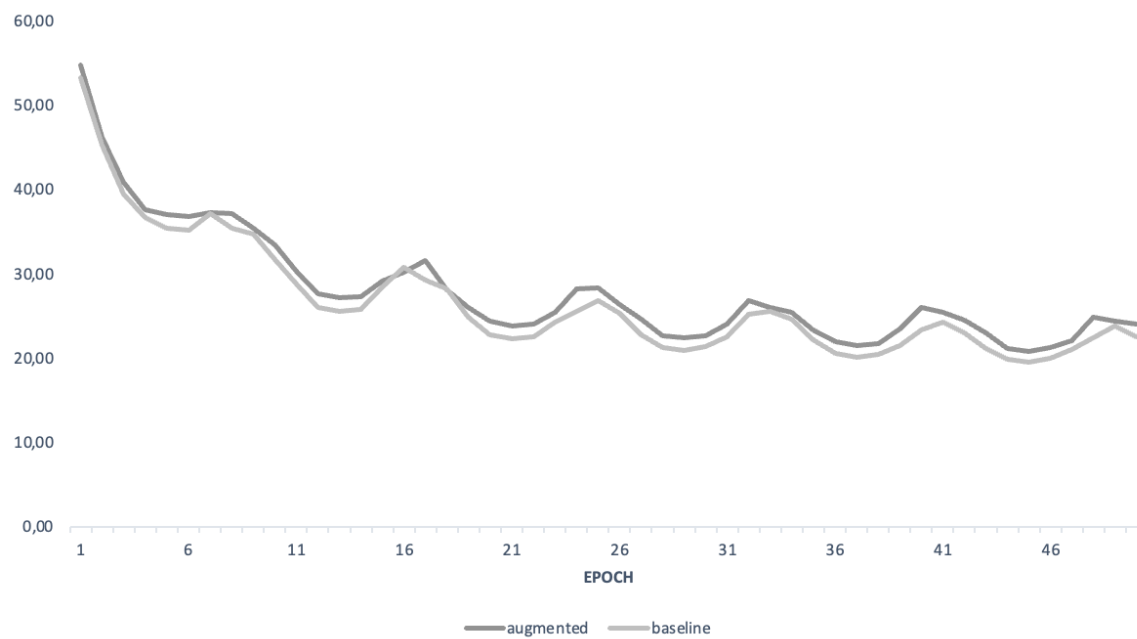


Figure 21. Average Levenshtein Distance of Model with Custom Augmentations vs Baseline

3.5 Model on Low-Quality Images

The addition of salt and pepper noise to original images produces a lower quality model output. The model gets trained for a total of 50 epochs using the same configurations as the baseline model. Salt and pepper noise is added through PyTorch's *get_transforms()* functionality in real-time.

As can be observed, the training loss performance is nearly the same as the baseline. However, when adding salt and pepper noise, the Levenshtein distance performance is sporadic – either better or worse than the baseline at different training intervals. No convergence takes place. This could be due to salt and pepper noise on training images either getting close to validation set images or far apart, depending on the quality of the latter. If the validation image is of high quality, the performance suffers. However, if the validation image is poorly represented such that it has a lot of noise in it and broken parts, then the performance improves. The results are as per Figures 22 and 23, with minimum Levenshtein distance achieved at the 29th epoch.

One of the reasons for the obtained result could lie in a lack of real-world noisy test data and its forceful approximation for the purposes of this study. The achieved validation scores behave in such a varying way due to the presence or absence of noise, and the effect of that noise on the validation set images.

The model does not perform too poorly, however, especially if compared with the model trained using custom augmentations. This is because garbled training images start more closely resembling validation set data which also has discontinuities and/or unclear letters, etc. In fact, the effect of noise is quite random as we do not control where on the image the noise appears. More advanced targeted lower-quality representations can contribute better to the final output.

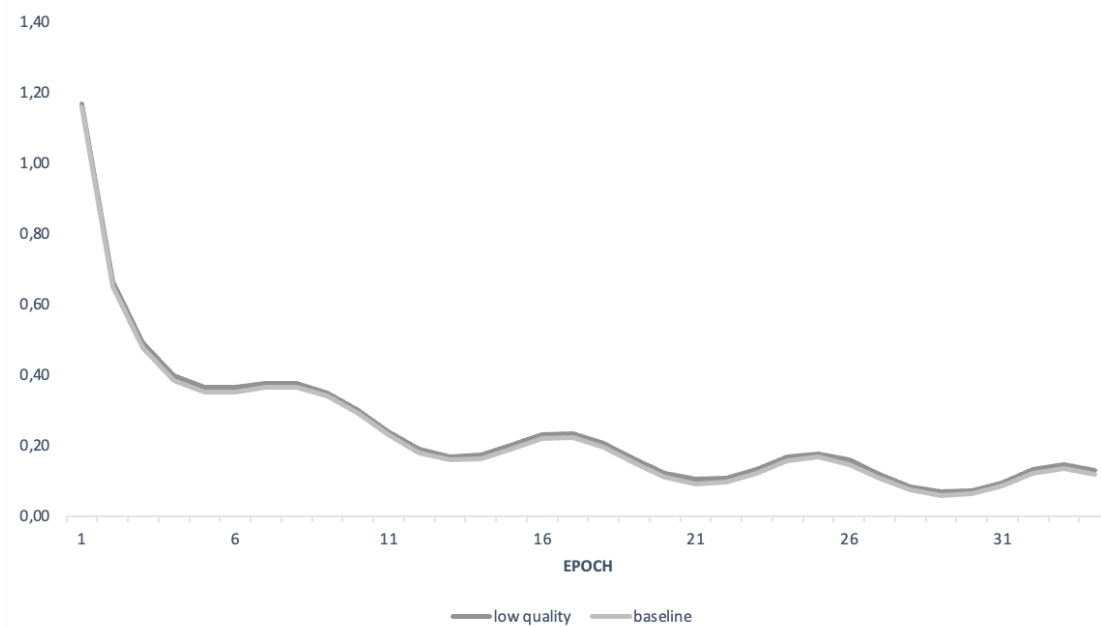


Figure 22. Training Loss of Model on Low-Quality Images

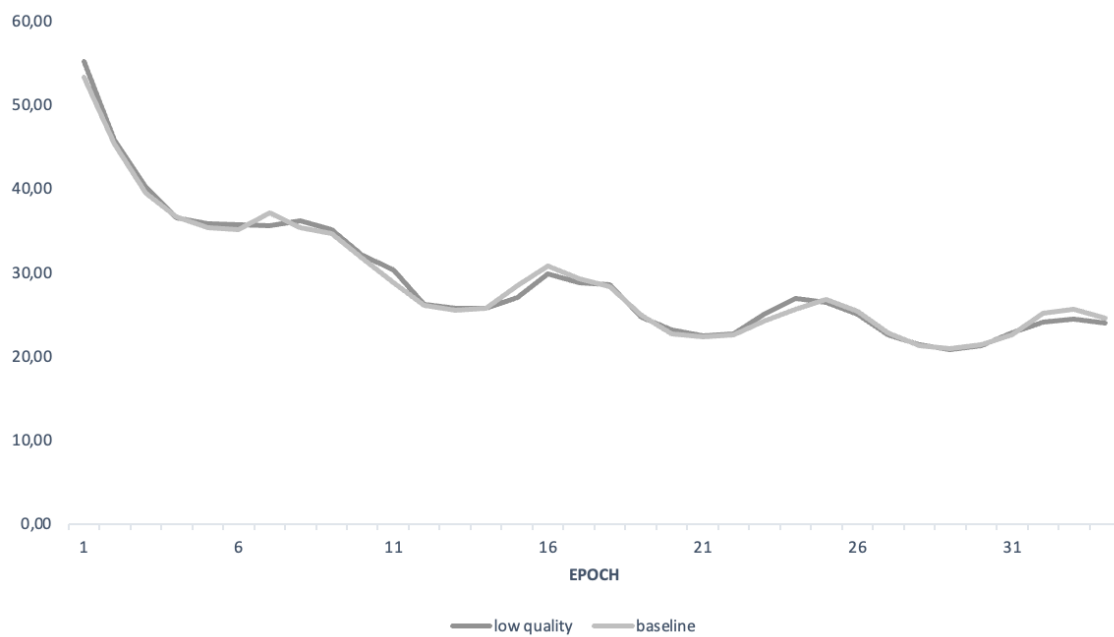


Figure 23. Average Levenshtein Distance of Model on Low-Quality Images vs Baseline

3.6 Training on Original vs Custom-Augmented Images

An experiment is performed to test whether training on augmented images results in approximately the same accuracy as training on original images. This can help determine if the augmentation process is at all effective and produces images like the original copies. Hence, two models are instantiated and trained:

- a basic attention-less LSTM image captioning model trained on 50,000 original images and tested on the augmented dataset (model 1), and
- a basic attention-less LSTM image captioning model trained on 50,000 augmented images and tested on the original dataset (model 2).

The resultant metrics are quite comparable. The training loss is approximately 3.9% for model 1, with a validation accuracy of 42%. For model 2, the training loss is circa 3.4% with a 40% validation accuracy. Regardless of similar results, the augmented images approximate the real images they are based on not too well. This can be judged by the validation accuracy achieved on the original set with a model trained on augmentations. The expectation ideally would have been a much higher accuracy since all scenarios are accounted for when custom transformations are generated. However, likely due to the absence of real-world examples of several style variations per InChI label, the results are not satisfactory.

4 CONCLUSION & FUTURE WORK

This study is dedicated to the creation of a low-quality position-agnostic chemical structure image recognition system. The significance of this problem lies in the need to automate the traditionally manual identification and analysis of molecules in a way that automatically recognizes and translates such structures. However, one of the main difficulties when creating such a system lies in the need to account for real-world low-quality data which can contain discontinuities, unclear components, and noise, as well as drawing style variations. The latter issue results in positional variances of chemical components on a given image and hence, creates a difference when it comes to modeling this type of data. Therefore, to enable stable model predictions the traditionally simple image recognition problem is taken a step further to consider these aspects.

The task at hand is to translate molecular images provided by the BMS company to standard identifiers, such as the InChI labels. There are several identifiers in the industry apart from InChI (such as SMILES, etc.) – yet the InChI label is more suitable to this study due to its uniqueness and the ability to encompass all the valuable information about a molecule in a layered string representation.

Traditionally, the problem of OCR was rule-based such that chemical knowledge was used to create connection tables for chemical structure graphs. Different types of algorithms for finding connected components, detecting lines using transformations and slope analyses were utilized to read the graph data into a computer-friendly format. However, such rule-based approaches heavily rely on domain expertise and written-down scenarios which entails constant monitoring and manual updates in case of any changes. More recent machine-learning approaches automate this workflow without the need to rely on any rules, thus taking away the bottleneck in the system caused by inefficiency of the process. Encoder-decoder structures, attention models and deep transformers, amongst others, are used for image-to-sequence translations. To make model predictions more stable to real-world data, augmentations are added to decrease or increase the resolution of images, blur them, and create orientational variations. In most of the cases, however, the data is synthetically generated, and does not account for real-world noise and positional differences.

Overall, data augmentation can be quite beneficial in research settings because it can reduce overfitting and increase model accuracy. There are several augmentation strategies that can be applied to a given image, ranging from basic manipulations to non-linear transformations and machine-learning approaches. In the reviewed literature, authors sometimes utilize smart augmentation techniques to suit the data context. This entails the usage of neural networks to create otherwise unimaginable augmentations. Some research claims that such transformations help produce better results, while other authors maintain that no major difference is reported. This depends on the complexity and size of the dataset as well as its context. In any case, augmentation must be applied carefully because sometimes it may not even carry any meaning – for example, rotating the image clockwise with number 6 on it produces a semantically different representation (the number 9).

For the purposes of this research, a sample of 300,000 BMS images of size less than 300x300 is used. The modeling is performed on Kaggle's cloud platform and ADA University CeDAR premises. Custom augmentation techniques are developed to create synthetic style variations in molecular

structure images. Low-quality image representations are added by using random salt and pepper noise. Different types of feature extraction methods are tested, such as InceptionV3 and Autoencoder. The quality of the generated augmentations is tested and compared against original images. The model used is an LSTM image captioning network with attention trained for about 50 epochs.

The results of the custom augmentation methods are visually satisfactory except for the errors created when chemical element names consist of more than one letter, and hence ordering must be considered when performing transformations to preserve their textual orientation. However, oftentimes such atoms are recognized as one character and therefore, preserving a correct order becomes a challenging task. One way of solving this can be through natural language processing but the implementations using such packages as *pytesseract* proved to be unsuitable. At the same time, chemical element name coordinates are calculated to enable correct transformations that preserve text orientations, and this entails the usage of object detection tools. The performance of such tools is satisfactory but results in a few misidentifications caused by the reliance of anomaly detection mechanisms on predefined hardcoded rules.

In terms of feature extraction, an analysis on 50,000 original and augmented images was performed using 299x299 InceptionV3 and 256x256 Autoencoder networks as encoders, and a basic attention-less 512-dimensional LSTM decoder. The results indicate that InceptionV3 performs better than the Autoencoder until the 45th epoch after which the results start converging. The Autoencoder starts from a much higher loss than the InceptionV3 network due to a simpler compression technique without convolutional filter variations. InceptionV3 is much deeper and can learn important features at different scales because of several different convolutions.

The baseline model is an encoder-decoder architecture, where the encoder is InceptionV3, and the decoder is LSTM with attention. Cross-entropy training losses and validation-set Levenshtein distances are analyzed and compared. The losses and metrics follow a wavy pattern which is due to using the Cosine Annealing Learning Rate scheduler which resets the learning rate as soon as the model stops learning. The achieved minimum edit distance is 19.5 in the 45th epoch, and it is quite possible that the model is overfitting at some intervals as explained by spikes in its shape over time.

To compare the results of custom augmentations to the baseline, custom augmentations are added to the baseline model. The resultant average Levenshtein distance is higher across all epochs than the baseline, with a minimum of 20.8 achieved in the 45th epoch. This corresponds to a 10% error rate which is not too bad and can be indicative of a lack of real-world validation-set data that includes those different style variations in lieu of custom-created augmented data which adds extra corruptions to the original data. The obtained result is quite consistent with the reviewed literature base, as several studies report no significant benefit from using transformations in the context of non-complex synthetically generated images. Another reason for the obtained results could be due to baseline overfitting in which case adding more data only amplifies the overfitting effect.

Modeling on lower-quality images results in nearly the same performance as the baseline but the Levenshtein distance scores are sporadic. This could be due to randomness of the generated noise as no control exists over the place to which noise applies. Sometimes the noise is approximated and at times, the validation set data is of more quality than the input noisy data, leading to such variances.

Training on augmented images results in approximately the same accuracy scores as training on original images. This signals that augmentations nearly approximate original data, however, the results are not too high (circa 40% accuracy score) and indicate that more can be done to increase their quality.

To summarize, the obtained results are due to:

- the absence of real-world test data with style and noise variations,
- models that are not too complex in terms of generating more useful features,
- randomness in positions of generated noise,
- incorrect text ordering of lengthy chemical elements on augmented images.

Future work includes generating better low-quality images using GANs, applying advanced object detection techniques like YOLO and NLP to fix the text ordering of incorrectly transformed chemical element names, and finding real-world style-variable data as a test dataset.

5 BIBLIOGRAPHY

- [1] Heller, S.R., McNaught, A., Pletnev, I. et al. 2015. InChI, the IUPAC International Chemical Identifier. *J Cheminform* 7, 23. <https://doi.org/10.1186/s13321-015-0068-4>.
- [2] Paul J. Karol. 2018. The InChI Code. *Journal of Chemical Education* 2018 95 (6), 911-912. DOI: 10.1021/acs.jchemed.8b00090
- [3] U.S. Environmental Protection Agency. 2016. Retrieved from https://archive.epa.gov/med/med_archive_03/web/html/smiles.html
- [4] Robert Belford. 2019. Chemical Representations on Computer: Part III. University of Arkansas at Little Rock. Retrieved from [https://chem.libretexts.org/Courses/University_of_Arkansas_Little_Rock/ChemInformatics_\(2017\)%3A_Chem_4399_5399/2.3%3A_Chemical_Representations_on_Computer%3A_Part_III](https://chem.libretexts.org/Courses/University_of_Arkansas_Little_Rock/ChemInformatics_(2017)%3A_Chem_4399_5399/2.3%3A_Chemical_Representations_on_Computer%3A_Part_III).
- [5] CAS Registry. 2022. Retrieved from <https://www.cas.org/cas-data/cas-registry>.
- [6] U.S. Food & Drug Administration. 2022. FDA's Global Substance Registration System. Retrieved from <https://www.fda.gov/industry/fda-data-standards-advisory-board/fdas-global-substance-registration-system>.
- [7] Park, Jungkap, et al. 2009. Automated extraction of chemical structure information from digital raster images. *Chemistry Central Journal*, vol. 3, 5 Feb. 2009, p. 4. Gale Academic OneFile, link.gale.com/apps/doc/A194503504/AONE?u=googlescholar&sid=googleScholar&xid=6d1bcf35. Accessed 20 Apr. 2022.
- [8] Kohulan Rajan, Otto Brinkhaus, Cristoph Steinbeck, Achim Zielesny. 2020. A Review of Optical Chemical Structure Recognition Tools. *Journal of Cheminformatics* 12(1).
- [9] Joe R. McDaniel and Jason R. Balmuth. 1992. Kekule: OCR-optical chemical (structure) recognition. *Journal of Chemical Information and Computer Sciences* 1992 32 (4), 373-378. DOI: 10.1021/ci00008a018.
- [10] Aniko T. Valko and A. Peter Johnson. 2009. CLiDE Pro: The Latest Generation of CLiDE, a Tool for Optical Chemical Structure Recognition. *Journal of Chemical Information and Modeling* 2009 49 (4), 780-787. DOI: 10.1021/ci800449t.
- [11] Jacobs, Marc. 2011. Chemical Structure Reconstruction with chemoCR. NIST Special Publication.
- [12] Filippov, I. V., & Nicklaus, M. C. 2009. Optical structure recognition software to recover chemical information: OSRA, an open source solution. *Journal of chemical information and modeling*, 49(3), 740–743. <https://doi.org/10.1021/ci800067r>.
- [13] Clevert, D. A., Le, T., Winter, R., & Montanari, F. 2021. Img2Mol - accurate SMILES recognition from molecular graphical depictions. *Chemical science*, 12(42), 14174–14181. <https://doi.org/10.1039/d1sc01839f>.
- [14] Martijn Oldenhof, Adam Arany, Yves Moreau, and Jaak Simm. 2020. ChemGrapher: Optical Graph Recognition of Chemical Compounds by Deep Learning. *Journal of Chemical Information and Modeling* 2020 60 (10), 4506-4517. DOI: 10.1021/acs.jcim.0c00459.
- [15] Khokhlov I, Krasnov L, Fedorov M, Sosnin S. 2021. Image2SMILES: Transformer-based Molecular Optical Recognition Engine. *ChemRxiv*. Cambridge: Cambridge Open Engage.
- [16] Li, Yanchi, Guanyu Chen, and Xiang Li. 2022. Automated Recognition of Chemical Molecule Images Based on an Improved TNT Model. *Applied Sciences* 12, no. 2: 680. <https://doi.org/10.3390/app12020680>.
- [17] Shorten, C., Khoshgoftaar, T.M. 2019. A survey on Image Data Augmentation for Deep Learning. *J Big Data* 6, 60. <https://doi.org/10.1186/s40537-019-0197-0>.

- [18] Lemley, J., Bazrafkan, S., & Corcoran, P.M. 2017. Smart Augmentation Learning an Optimal Data Augmentation Strategy. *IEEE Access*, 5, 5858-5869.
- [19] Perez, Luis & Wang, Jason. 2017. The Effectiveness of Data Augmentation in Image Classification using Deep Learning.
- [20] I. Kim, S. Han, J. -W. Baek, S. -J. Park, J. -J. Han and J. Shin. 2021. Quality-Agnostic Image Recognition via Invertible Decoder. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 12252-12261, doi: 10.1109/CVPR46437.2021.01208.
- [21] Bristol Myers Squibb Molecular Translation Competition. Kaggle. 2021. Retrieved from <https://www.kaggle.com/competitions/bms-molecular-translation>.
- [22] Szegedy, Christian & Vanhoucke, Vincent & Ioffe, Sergey & Shlens, Jonathon & Wojna, Zbigniew. 2015. Rethinking the Inception Architecture for Computer Vision.
- [23] Szegedy, Christian & Liu, Wei & Jia, Yangqing & Sermanet, Pierre & Reed, Scott & Anguelov, Dragomir & Erhan, Dumitru & Vanhoucke, Vincent & Rabinovich, Andrew. 2015. Going deeper with convolutions. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 1-9. 10.1109/CVPR.2015.7298594.
- [24] Dor Bank, Noam Koenigstein, Raja Giryes. 2020. Autoencoders. Retrieved from <https://arxiv.org/abs/2003.05991>
- [25] Turchenko, Volodymyr & Chalmers, Eric & Luczak, Artur. 2017. A Deep Convolutional Auto-Encoder with Pooling - Unpooling Layers in Caffe. *International Journal of Computing*. 18.
- [26] Alex Sherstinsky. 2018. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. Elsevier "Physica D: Nonlinear Phenomena" journal, Volume 404, March 2020: Special Issue on Machine Learning and Dynamical Systems.
- [27] Y. Nakama. 2021. InChI / Resnet+ LSTM with attention / starter. Retrieved from <https://www.kaggle.com/code/yasufuminakama/inchi-resnet-lstm-with-attention-starter>.
- [28] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.
- [29] Michael Gilleland, Merriam Park Software. Levenshtein Distance, in Three Flavors. Retrieved from <https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm>.