Computers, Materials & Continua
DOI: 10.32604/cmc.2022.024819
Article

Tech Science Press

Transforming Hand Drawn Wireframes into Front-End Code with Deep Learning

Saman Riaz[1], Ali Arshad[2], Shahab S. Band[3,*] and Amir Mosavi[4]

[1]Department of Computer Science, National University of Technology, Islamabad, 44000, Pakistan
[2]Department of Computer Science, Institute of Space Technology, Islamabad, 44000, Pakistan
[3]Future Technology Research Center, National Yunlin University of Science and Technology, Douliu, 64002, Yunlin, Taiwan
[4]Faculty of Civil Engineering, Technische Universitat Dresden, Dresden 01069, Germany
*Corresponding Author: Shahab S. Band. Email: shamshirbands@yuntech.edu.tw
Received: 01 November 2021; Accepted: 25 January 2022

Abstract: The way towards generating a website front end involves a designer settling on an idea for what kind of layout they want the website to have, then proceeding to plan and implement each aspect one by one until they have converted what they initially laid out into its Html front end form, this process can take a considerable time, especially considering the first draft of the design is traditionally never the final one. This process can take up a large amount of resource real estate, and as we have laid out in this paper, by using a Model consisting of various Neural Networks trained on a custom dataset. It can be automated into assisting designers, allowing them to focus on the other more complicated parts of the system they are designing by quickly generating what would rather be straightforward busywork. Over the past 20 years, the boom in how much the internet is used and the sheer volume of pages on it demands a high level of work and time to create them. For the efficiency of the process, we proposed a multi-model-based architecture on image captioning, consisting of Convolutional neural network (CNN) and Long short-term memory (LSTM) models. Our proposed approach trained on our custom-made database can be automated into assisting designers, allowing them to focus on the other more complicated part of the system. We trained our model in several batches over a custom-made dataset consisting of over 6300 files and were finally able to achieve a Bilingual Evaluation Understudy (BLEU) score for a batch of 50 hand-drawn images at 87.86%

Keywords: Deep learning; wireframes; front-end; low fidelity; high fidelity; design process; html; computer vision; dsl

1 Introduction

Nowadays, transforming hand-drawn wireframes into front-end code is created by a designer into computer code is a typical customized software, websites, and mobile apps [1–3], which serves as a

rough idea of how we want the final structure to be. A key issue with this is that because sites are built of dozens if not thousands of different pages, having to create and design a sketch then converting that into functioning code can be extremely costly and time-consuming, as designers need to go through one by one and implement each required aspect of the final structure [1,4]. All this time adds up and takes up a significant amount of development time for the projects, time that could be better used elsewhere, such as the more technical aspects of the work.

In our proposed work, the process of converting the wireframes to code involves passing the design to a developer/UI designer, having them implement the boilerplate graphical user interface (GUI) [1,5,6], and then reiterate this product until they reach a stage that they or the client are satisfied with.

Deep learning changed front-end development by increasing prototypes speed and lowering the barrier for building software. In this paper, we tackle the issue and aim to facilitate the needs of designers by saving them a significant amount of time. Using deep learning techniques [2–3,7–9] would enable the automation of the creation process by simply letting the designers take their sketches and manipulate them using the model into HTML code. Using Neural Networks, we can employ them to learn the underlying patterns and context of elements on a sketch and convert them into front-end Html code [1,4], which they could then edit, customize or implement according to their needs. The objectives of our system are such that:

- Saves time: Removing the need for a large amount of busy work frees the designers to work on other more complicated or involved portions of their work as now the time to sketch to code conversation does not need to be painstakingly done over and over.
- Focus: Designing front-ends can sometimes be tedious and repetitive work, preventing developers from focusing on actual core logic. This system will help them focus on actual problems.
- Flexibility: This will allow us to quickly transition between different designs to test out what works best with the desired application, spending less time and resources with each iteration.
- Accuracy: With the implementation of our proposed technique, the system can produce an accurate extrapolation of the given sketches into their proper code form with an average of 87.86% accuracy.

The contribution of our proposed system, we found the best approach is through the usage of multi-Model deep learning [2–3,7–9] and computer vision [7–9] techniques to train a system to handle the vast number of variations in a hand-drawn wireframe. We believe that through a rigorous training process and implementation of these concepts, we can create a model capable of servicing various users' needs and generating a front end with a high amount of accuracy.

There were some major challenges in the creation of such a system:

- Dataset Disparity for a deep learning model.
- A large amount of time and resources are required in the creation of a new custom dataset.
- Hardware Limitations due to costly training times.
- Dividing the issue into sub problems, namely computer vision and language modeling problems.
- Getting each element to be recognized by the model, and correctly translated.

2 Backgrounds

2.1 Sketches

Sketching is a Hand-drawn mockup or User interface design shown in Fig. 1; it is the basis of a user interface before prototyping [10,11], and coding. Sketching [12] comes after the idea of a website or any application before building screen mockups in a website wireframe tool. Sketching refers to the

act of working through all the possible ways you could make your idea into an interface reality [12]. At the end of the UI sketching process, you should know that you have figured out the absolute best way to bring your product to the screen. However, you choose to sketch your initial User Interface ideas; keeping it simple and not spending too much time on it is essential.

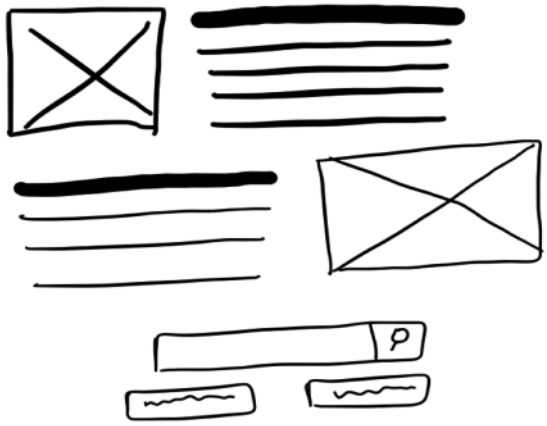
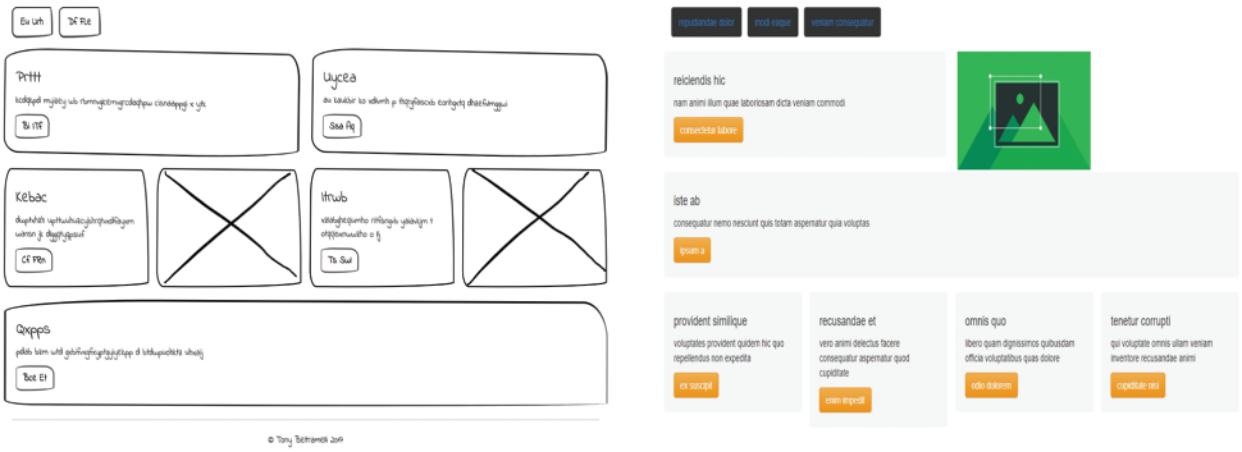


Figure 1: Representation of a user interface designed for a website

2.2 Wireframes

A wireframe [13,14] is a simple representation of a products design. They represent a way to get ideas down quickly. They help designers focus on the more functional aspect before moving on to the finer details. They are generally two types of wireframes: Low-fidelity wireframes [15] and High-fidelity wireframes [16]. A low-fidelity wireframe is a simple representation shown in Fig. 2a. It can be a drawing on a whiteboard, a rough sketch drawn on pieces of paper, just as long as it can properly communicate the purpose of what we want to create. High-fidelity wireframes are more complete representations of the end product, which is shown in Fig. 2b. They help to communicate how aesthetic features support essential functionalities.



(a) (b)

Figure 2: Representation of low fidelity wireframes (a) and high fidelity wireframes (b)

2.3 Frontend

Front-end design consists of user experience elements on the web page or application, including menus, pages, buttons, icons, links, graphics, etc. The front-end is a combination of the following technologies such as HTML [17], CSS [18], and Javascript [19].

3 Literature Review

In this section of the paper, we explore the current studies and research done in the field of sketch to code conversion.

Beltramelli proposed pix2code [4], which managed to implement, which managed to convert high-fidelity wireframes into code. They trained their model with stochastic gradient descent [20] to simultaneously learn to model sequences and Spatio-temporal visual features [21] to generate variable-length strings of tokens from a single GUI image as input. It was designed to process the input data; the Author model learns from the pixel values of the input image alone. Pix2code converted screenshots to code, but it cannot still convert hand-drawn wireframes into code. At the same time, our proposed method converted hand-drawn sketches into code.

Alexander Robinson proposed Sketch2code [1] by which attempts to solve the problem of translating wireframe sketches into code using deep learning [22] that showed considerable success over classical techniques when applied to other domains, particularly in vision problems.

DeepCoder [23], a system created to generate computer programs by leveraging statistical predictions to augment traditional search techniques done by MatejBalog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. In another work by Gaunt et al., the generation of source code is enabled by learning the relationships between input-output examples via differentiable interpreters. Program generation like this is a very active research field but program generation from taking visual inputs is still an unexplored area that our research paper attempts to explore.

Tuan Anh Nguyen and ChristophCsallner on Reverse Engineering Mobile Application User Interfaces (REMAUI) [24]. REMAUI identifies user interface elements such as images, texts, containers, and lists, via computer vision [25] and optical character recognition (OCR) [26] techniques. This is a good example of how these types applications take a wireframe or high-fidelity image and translate them into code. These applications with structure also solve all the associated styles with it, but it faces the problem of adding new examples, which is challenging and time-consuming. The problem is made worse by the inclusion of style in the designs, leading to a significantly higher variation than the digital drawings in the low fidelity designs [1].

SILK [27] by James A. Landay (Human-Computer Interaction Institute School of Computer Science Carnegie Mellon University). SILK is an interactive tool that allows designers to sketch and an electronic pad quickly. It then retains the "sketchy" look of the components. The system facilitates quick prototyping of interface ideas through the use of gestures for sketch creation and storyboards.

It uses a combination of computer vision [7–9] techniques to classify shapes drawn on a digital surface into predefined components such as corner detection and line detection, classify shapes and application elements such as buttons, text, textfield, box, etc. The main advantage of SILK [27] over paper sketches is that it allows the storyboards to come alive and permits the designer or test subjects to exercise the interface in this early, sketchy state.

While the concept of turning wireframes into code is not new, as with the advancement of machine learning, more work has been put into it, but it still runs into some complex challenges. Thus, our

primary analysis focused on two key papers, Sketch2code [1] and Pix2code [4], each concerned with a specific conversion aspect.

We have focused on two main research areas: the first one in turning high fidelity wireframes such as digital mockups into code and secondly converting low fidelity drawings into code. Both these studies have one thing in common: the translation of wireframes designs into a coded form. The significant difference is the fidelity of the designs being translated.

From our examination of the current work done, we saw that there was a limited number of elements considered in the research, which we believe could be improved upon, along with this the primary dataset contained GUI images and their associated code, so we decided to make alterations within the primary dataset to make it look more hand-drawn. To this end, we created a model with various optimizations to ensure better accuracy and training times, along with an increased amount of considered elements in its conversion from wireframes into code.

4 Techniques

4.1 Computer Vision Techniques

To build a system that could detect the various aspects of our sketches, such as the lines and edges, we needed to use computer vision techniques [7–9]. Computer vision is how a computer can understand images or videos; it relies on acquiring, processing, and analyzing digital images and using information supplied to it, to create a context in which to break down said images.

4.1.1 Edge Detection

To make our system work, we need to consider how we go about detecting the various elements in our wireframe [13] sketches. To this end, when we draw a wireframe symbol consisting of straight or sometimes rounded edges, the CNN will detect these edges effectively given enough sample size.

There are various techniques commonly used to detect edges:

- The Sobel operator [28] computes the gradient map of an image and then yields edges by thresholding the gradient map.
- The Canny operator [29] extends the Sobel operators, which adds Gaussian smoothing as a preprocessing step and uses the bi-threshold to get the edges.
- Richer Convolutional Features [30] is a state-of-the-art deep learning approach using convolutional networks.

4.1.2 Segmentation

The sketches that one would create many types of elements, for example, buttons, image boxes, radio buttons, checkboxes, text boxes, etc. To make sure these various elements are detected adequately to be converted into code, we need a system to segment [31] each part of the sketch and classify them for our system to convert them to their correct forms.

The approach we researched consisted of using a U-Net Model [32] that is trained to segment a hand-drawn image by color-based segmentation [33]. It derives its name from how its architecture, when visualized, seems to lay out what it is. It derives its name from how its architecture, when visualized, seems to lay out what it is can be said as the shape U.

4.1.3 Data Augmentation

One of the key hurdles for our work was the lack of available data; to this end, we needed to employ the concept of data augmentation [34]. Data augmentation is the technique of increasing the amount of data available by slightly modifying different copies of the existing data and turning them into new data.

Our methods of data augmentation involved manipulating our current hand-drawn images by Altering the scale, Flipping and Rotating, Changing Brightness and Contrast and Adding Noise. Through this augmentation, we were able to achieve a certain amount of variation within the dataset, mainly used to counter overfitting problems that arise from the use of our synthesized dataset.

4.2 Scripting Techniques

To increase the sample size of the primary dataset, we finally decided to employ a custom script that would directly alter the CSS elements present in the GUI images. We tried to introduce various human elements and variations to the images like skewed lines, fonts that resemble human writing, rounded off edges, clear white background with the elements drawn in black, etc. We were successfully able to replicate a hand-drawn feeling to the primary dataset, though produced synthetically.

4.3 Deep Learning Techniques

Machine learning serves as the basis for our entire system to work. It is a greatly emerging and powerful tool used to give computers the ability to learn and adapt to serve purposes without explicitly programming each function. It achieves this by being fed a large amount of data and "teaching" it how to analyze and use that data to then work on newer information for a specific task. We chose machine learning as it is excellent for classification and detection tasks, along with helping to automate busywork significantly. In this section, we talk about the essential techniques we explored and used in our approach.

4.3.1 Convolutional Neural Networks

Convolutional Neural Networks [30,35] are deep learning [3] algorithms that take input images and assign a level of weight and basis to them. Using those weights can better differentiate one image from another.

CNN's use raw pixel data as input when incorporated into image-based projects, and they consist of input, output, and hidden layers shown in Fig. 3, and these hidden layers consist of Convolutional layers, Pooling layers, and Fully connected layers. The way we have incorporated CNNs in our work is an Encoder. This encoder used as means through which the model encodes, and understands the elements present in the hand-drawn image. With this, we can classify various types of elements present in a Wireframe [13] Image.

4.3.2 Recurrent Neural Networks

RNNs are a robust form of neural network that uses their internal state (memory) to process a variable length of sequences of inputs; this makes them great for applications involving handwriting recognition. There are two of the types of RNNs that are made use of are Long short-term memory (LSTM) [36] and Gated recurrent units (GRUs) are a gated mechanism in RNNs; they lack an output gate and thus have fewer overall parameters than an LSTM. They serve well as encoders and decoders [37], as they provide context to the images in the form of a sequence of tokens. In our system we used GRU as a decoder.

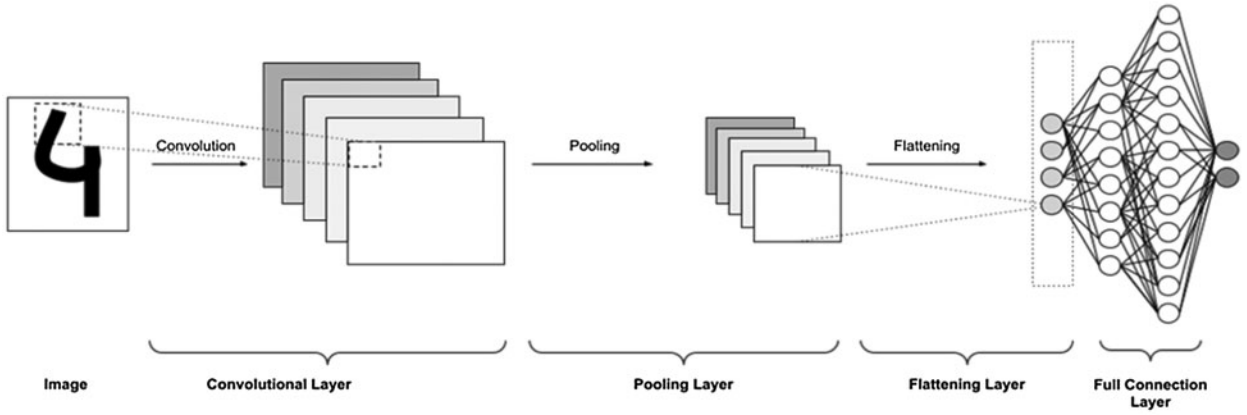


Figure 3: The architecture of Convolutional Neural Network

4.3.3 Fine Tuning

All image tensors [38] were compressed into simple images and were converted into low-resolution images to save memory. These low-resolution images were then converted again into image tensors for further resource-saving. Although the model has been relatively optimized since it started working, it is costly to Pre-Process, so to evaluate it, we had to reduce the sample size. In addition to this, to train such a complicated Model with our Improvements on the total sample size, we used a Cloud Server [32] (Google Cloud Services) to train it.

5 Methodology

To conduct an applied research study and develop methods that convert a Hand Drawn Wireframe into a working HTML code with a Hand Drawn Wireframe into a working HTML code with a Hand Drawn Wireframe into a working HTML code with Deep Learning. We used a primary dataset acquired from the research conducted by Tony Beltramelli [39], which contained over 1750 Images with relation to their appropriate code files. We used this primary dataset [39] as a basis for our dataset that we later improved upon, and trained over a series of Neural Networks [40] based on an Image Captioning Architecture. With the help of this approach, we were able to convert a given Hand Drawn Image of a Wireframe [1,15,16] into its HTML [13] file.

5.1 Dataset

Our approach required a larger dataset consisting of hand-drawn wireframes and their associated code.

We used Tony Beltramelli's research dataset [39] as our primary dataset, which contained over 1750 GUI Images and their associated Code Files. However, since we required Hand Drawn Images, we had to take specific steps to achieve a modified Dataset that our approach used effectively. Our modified dataset contains Images that are limited to look as close as possible to a hand-drawn image shown in Fig. 4 and has a sample size of 6300. We also altered the images and the code files with the addition of newer elements which is shown in Fig. 5.

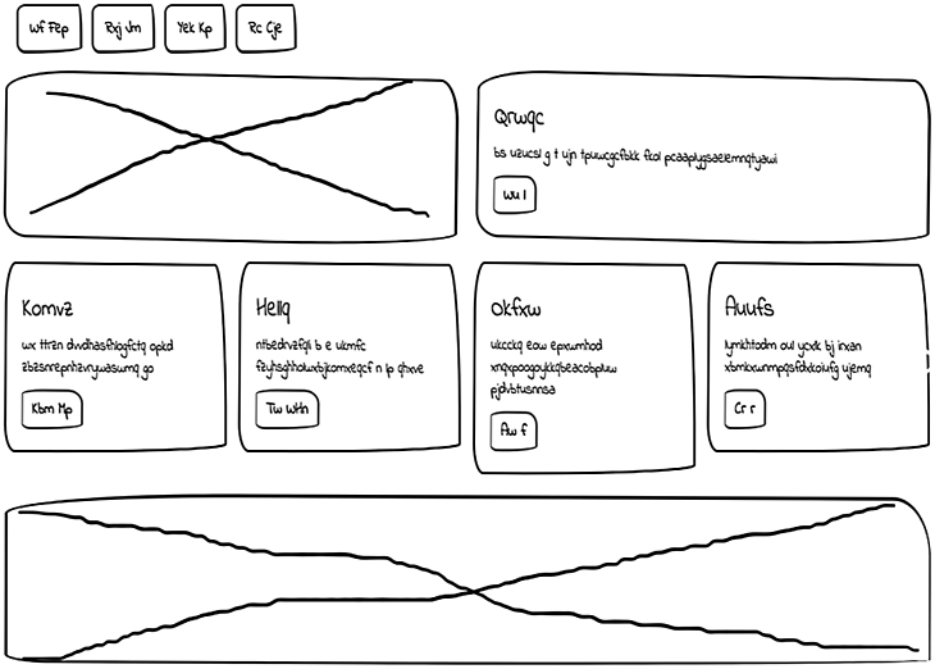


Figure 4: Wireframe image from our dataset

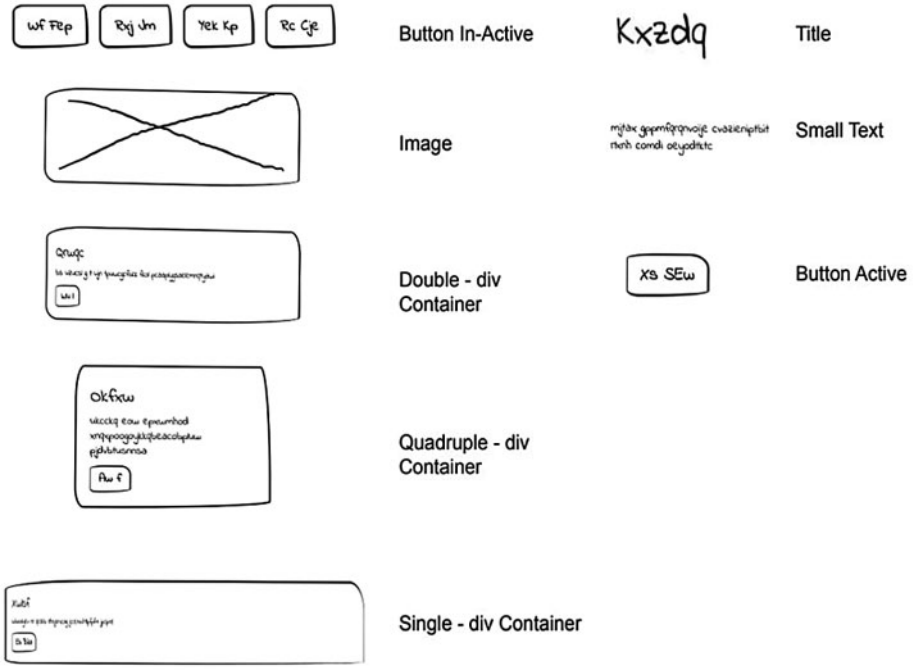


Figure 5: Elements key for our wireframe images

5.2 Framework

Our expected input image was a white background with elements drawn in a darker color fed into the Framework. The front end was intended to make it easier to feed the input from the User. The image was then processed into the Model, where Model tries to learn elements present in the image and the context that each element has, including its sequence and structure. Our proposed Framework shows in Fig. 6 how a wireframe image is passed into our Deep Learning model and how it is eventually processed into our required output. The output of the Model was a sequence of DSL [1,4] tokens shown in Fig. 6, which is Twitter Bootstrap [41], the reason being the limit on the sequence of GRU [5,42] tokens. These DSL [1,4] tokens were then fed into a custom-made compiler that was finally converted into an HTML [13] file, the expected output from our Framework.

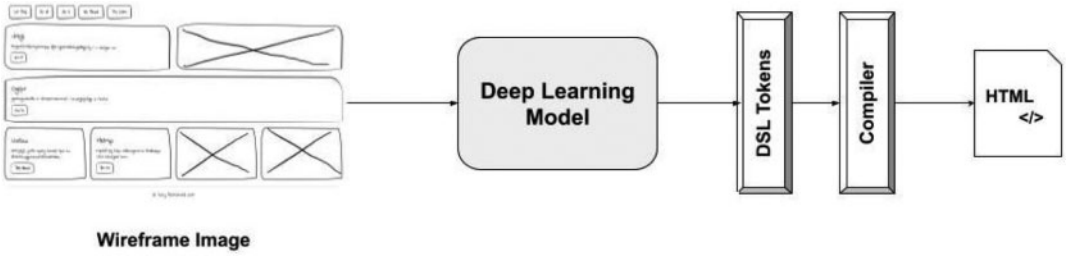


Figure 6: Flowchart of our proposed approach

5.3 Technical Approach

Our proposed image captioning architecture based model shown in Fig. 7. Every image is first encoded with a CNN-based encoder [5] model, and every code part that is first converted into one-hot encoding [43] is then encoded with a language model composed of stacks of GRU [44] layers. These two encoders are then concatenated and are fed into another stack of GRU layers which behave as a decoder [5]. It performs token-level modeling by using one-hot encoded vectors [45], eliminating any need for word embedding techniques such as word2vec [20], resulting in costly computations. After which, a softmax layer [43] is used to output tokens based on the DSL [1,4] vocabulary size. Those tokens are then fed into the custom bootstrap HTML compiler, which outputs HTML code. It is created with the help of TensorFlow and Keras [46].

A rough image along with its associated code is used to train the model. After the model is trained, it gives us the output in the form of tokens that are then passed into the compiler, which generates its pure HTML and CSS shown in Fig. 8.

6 Experiment

6.1 Experiment Setup

As stated before, we decided to use Tony Beltramelli's research dataset [39], which consisted of over 1750 GUI images and their associated code files. However, since our input requirement was a hand-drawn image, we required low-fidelity wireframes and associated code files. Since such a convenient dataset was not available in quality and quantity, we decided to alter and make modifications within the primary dataset, to give it a feel as close as possible to a hand-drawn image. We researched various techniques for making this possible; however, we ended up relying on a script that directly alters the CSS properties of the GUI image front ends. The images were altered in a way to produce a hand-drawn image shown in Fig. 9.

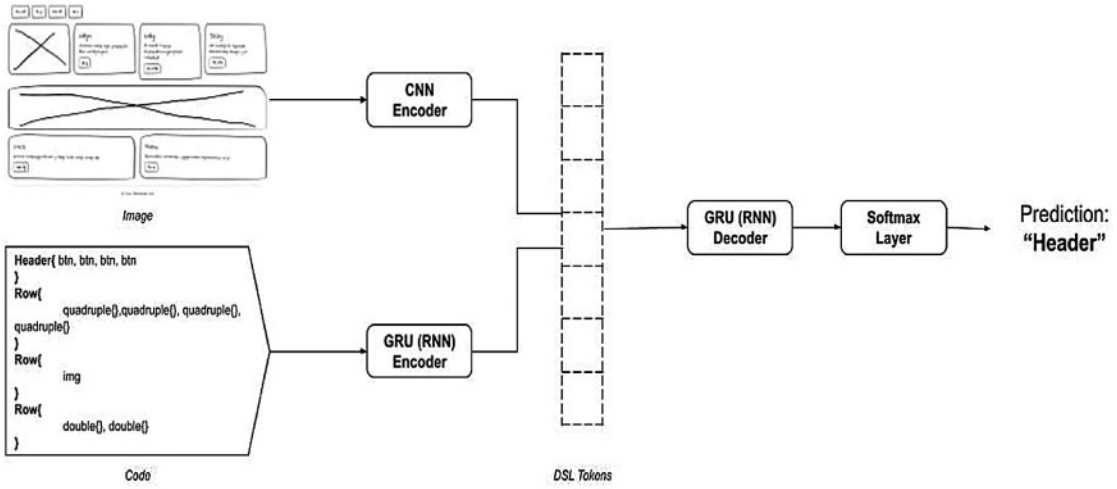


Figure 7: Architecture of our proposed approach

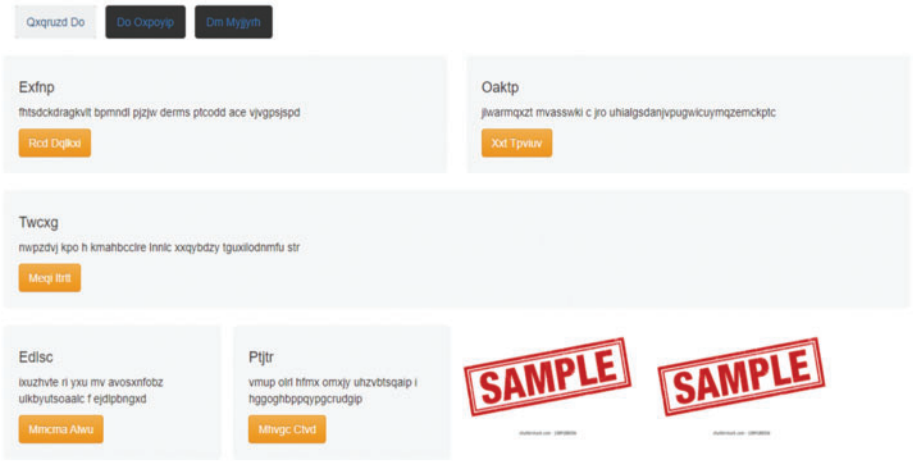


Figure 8: HTML/CSS

The very first step we did was to pre-process our image data. Load images from an npz file extension by converting them to a png format, using the PIL library. After loading the images, we resized them from their former size of 1380×2400 to 256×256. This preprocessing step aimed to reduce the amount of processing power needed to process UINT8 image tensors. The image tensors were converted to npz and fed into the model. The next step was to pre-process the language files of the dataset. A Vocabulary is made, which consists of all the elements that are present within an image. We updated this vocabulary by the addition of a newer element for classification.

A total of 18 classes were used to predict the labels that are mentioned in Tab. 1, using Keras we have a detailed summary of the model.

Total parameters shown in Tab. 2 are 139,723,686; apart from this, all libraries used in building and evaluating this project is as numpy, lorem_text, nltk, argparse, keras.models, keras.callbacks, keras.layers, keras.optimizers, spacy.

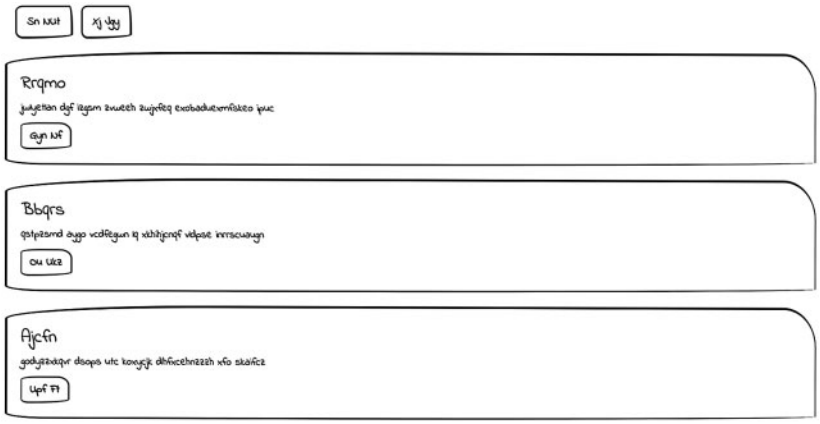


Figure 9: Our hand-drawn image

Table 1: Vocabulary

Ref no.	Token	Description
1	<START>	starting tag for the token.
2	<END>	ending tag for the tokens.
3	{	opening braces
4	}	closing braces
5	Btn	button tag
6	btn-active	active button tag
7	btn-inactive	inactive button tag
8	btn-orange	orange coloured button tag
9	btn-green	green coloured button tag
10	btn-red	red coloured button tag
11	Row	tag donating row section
12	Single	div container that covers the full row
13	Double	div container that covers half the row
14	quadruple	div container that covers quarter of the row
15	Header	header tag
16	Text	text tag
17	small-title	smaller font size title tag
18	Img	image tag

Table 2: Model parameters

Ref no.	Layer	Output shape	Parameter
1	Embedding	(None, 48, 50)	900
2	Input Layer	(None, 256, 256, 3)	0
3	GRU	(None, 48, 128)	68736
4	Sequential	(None, 48, 1024)	135414288
5	GRU_2	(None, 48, 128)	98688
6	Concatenate	(None, 48, 1152)	0
7	GRU_3	(None, 48, 512)	2557440
8	GRU_4	(None, 512)	1574400
9	Dense	(None, 18)	9234

6.2 Performance Measure

To measure and evaluate the performance of our model, we used the following methods:

1. **BLEU Score:** To evaluate and analyze the performance of our proposed method we use to evaluate it using a batch of images on the BLEU, or the Bilingual Evaluation Understudy metric, it is a score for comparing a candidate translation of the text to one or more reference translations, where 1.0 would represent a perfect score whilst a 0.0 would indicate a poor score. It is mathematically expressed as:

$$Unigram\ precision\ \ (P) = \frac{m}{w_t} \tag{1}$$

which is the N-gram precision bounded above by the highest count of n-grams in any reference sentence.

$$Brevity\ penalty\ \ (p) = \begin{cases} 1 & if\ c > r \\ e\left(1 - \frac{r}{c}\right) & if\ c \leq r \end{cases} \tag{2}$$

a penalty added for short translations.

$$BLEU = p.e \sum_{n=1}^{N} \left(\frac{1}{N} * \log Pn\right) \tag{3}$$

consists of the brevity penalty and the geometric mean of N-gram precisions.

2. **Spacy's document similarity:** Spacy's similarity refers to the semantic similarity between words, phrases or even between two whole documents. The core logic behind this is to create two representative vectors of two items; using either universal vector created from pre-trained models like word2vec, glov, fasttext, bartetc, spacy similarity is a cosine similarity between the internal vectors for the document spacy creates. We selected five random documents with each predicted and actual translation, and by using spacy's similarity, we obtained results in Tab. 3.

Table 3: Spacy's document similarity

Ref no.	Similarity
1	0.9198421346903709
2	0.9986393236624259
3	0.9981439951832412
4	0.9921174511226027
5	0.9940782151415509

6.3 Results and Analysis

As shown in Tabs. 2–5 and their associated graphs in Figs. 10–12, the training process for the model on our altered dataset consisting of over 6300 files shows the gradual decrease in the loss and validation loss values after each batch was trained. We trained our images over four batches to accommodate the complex pre-processing required before our training process. We observed from the results we collected that there were spikes amidst the validation loss values and have concluded that it was caused mainly because the dataset was synthesized to produce a hand-drawn feel and was not hand-drawn apart from the addition of the newer element.

Table 4: Loss and validation loss

	Batch 1		Batch 2	
Epoch	Loss (B1)	Validation loss (B1)	Loss (B2)	Validation loss (B2)
0	1.810527562	1.019041462	0.1393571585	0.1262309174
1	0.7119945834	0.523254671	0.1314871309	0.1204248799
2	0.4200088239	0.3024036242	0.1253236435	0.1284791624
3	0.2752406637	0.2435374795	0.1172689563	0.1129708358
4	0.2346654019	0.2117428654	0.1110115367	0.1218733783
5	0.1964809008	0.2054587587	0.1072947877	0.1095917118
6	0.1793122481	0.171840534	0.100063439	0.1405449001
7	0.1700920767	0.1855002354	0.09747299706	0.1391996284
8	0.1651249885	0.1623811646	0.09498966053	0.1033399156
9	0.1565822626	0.1825149895	0.08927372812	0.1195550857

Table 5: Third and fourth batch epoch, loss, validation loss

	Batch 3		Batch 4	
Epoch	*Loss (B3)*	*Validation loss (B3)*	*Loss (B4)*	*Validation loss (B4)*
0	0.2714013859	0.1409717719	0.1201789499	0.1167939797
1	0.1282625072	0.2286272529	0.1085395017	0.1304072287

(Continued)

Table 5: Continued

	Batch 3		Batch 4	
Epoch	*Loss (B3)*	*Validation loss (B3)*	*Loss (B4)*	*Validation loss (B4)*
2	0.1155536701	0.1328662336	0.1000736975	0.1105456484
3	0.1003801154	0.1922926778	0.0921522197	0.1258771133
4	0.09802151081	0.1574381596	0.08878851081	0.1056628559
5	0.09681803485	0.1260038112	0.08384299183	0.1078268196
6	0.08836132215	0.1588779345	-	-
7	0.0836164189	0.116681695	-	-
8	0.08256914456	0.1166688687	-	-
9	0.07968492299	0.1336399116	-	-

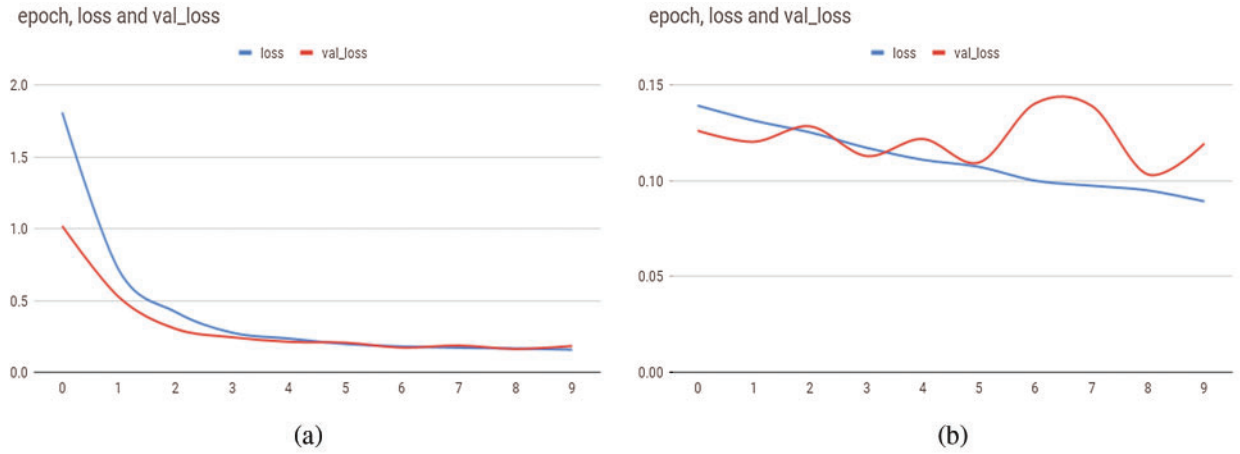


Figure 10: Graph of loss and validation loss values for the batch (1st, 2nd) *vs.* epoch size

In Figs. 10a and 11a, we can see that our model's loss score and validation score are moving side by side, but as we progress further in Figs. 10b and 11b, the validation score becomes increasingly jagged. By observing the graph and its lines, it is clear that the model is overfitted. Although the newer element we introduced, i.e., hand-drawn images, into the dataset, all the other elements were synthesized to produce the effect of a handmade drawing. This caused our model to learn the synthesized part of the dataset to the extent that it caused negative impact on overall results.

However, some of the negative impacts were reduced using overfitting reduction techniques, namely by adding Gaussian noise and increasing the dropout of nodes. The model incorporated with these techniques was trained again. We can see in Fig. 13, and Tab. 6 do somewhat mitigate the negative effects of a synthesized dataset, but ultimately it reduces accuracy.

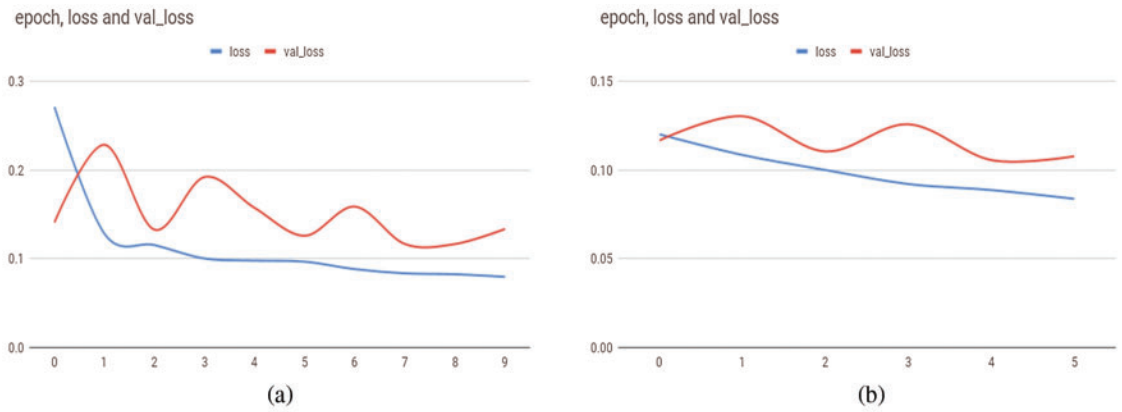


Figure 11: Graph of loss and validation loss values for the batch (3rd, 4th) *vs.* epoch size

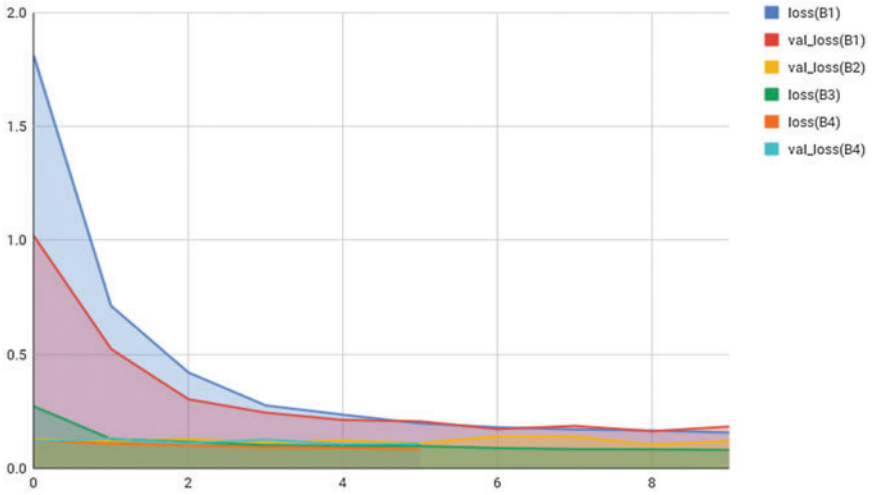


Figure 12: Graph of loss and validation loss for all the batches *vs.* epoch size

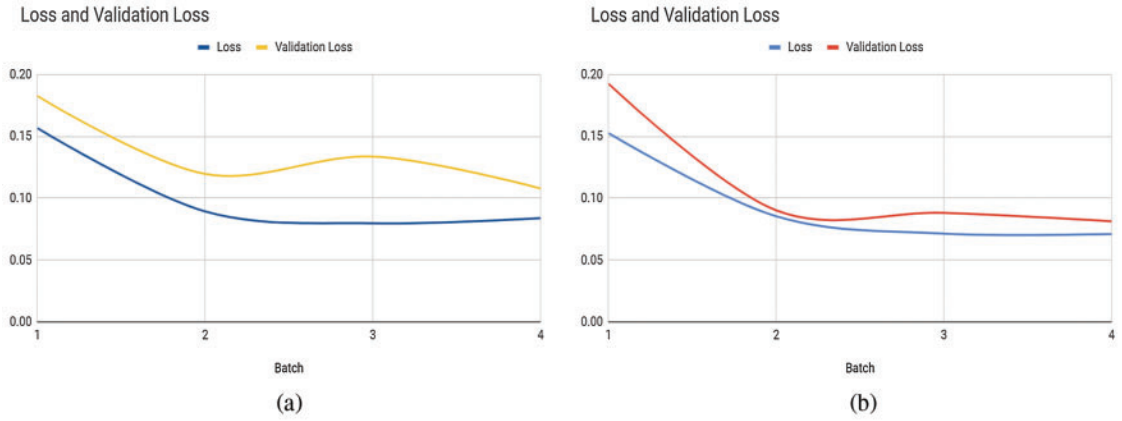


Figure 13: Graph of final loss and validation loss values of each batch in M1 and M2

Table 6: Comparison of loss values between original model (M1) and our modified model (M2)

	M1		M2	
Batch	*Loss*	*Validation loss*	*Loss*	*Validation loss*
1	0.1565822626	0.1825149895	0.1525533622	0.1924324221
2	0.08927372812	0.1195550857	0.0853459232	0.0901312341
3	0.07968492299	0.1336399116	0.0713423424	0.0881313123
4	0.08384299183	0.1078268196	0.0710324202	0.0813042943

In the end, we obtained an average BLEU score of 0.8785 and received a favorable outcome from spacy's document similarities. However, these results can be further refined by using a dataset of actual hand-drawn images instead of synthesized images and increasing the quantity of the overall dataset with more significant variation. The model parameters can then be readjusted to coincide with the dataset to produce a better result.

7 Conclusions

Our proposed method of Transforming Hand Drawn Wireframes into Front-End Code based on multi-model Deep Learning techniques can successfully translate hand-drawn images based on an established vocabulary with high accuracy and less time if the user adheres to will result in a clean output for an Html front-end page. We enhanced the previous studies on the subject by increasing the sample size of the preliminary dataset and added a newer element for classification and an analysis of its performance. Various optimizations were made in the model to process the complicated data with more ease and greater accuracy than previous implementations.

Our study into the subject of image conversion and multi-model deep learning techniques also made us aware of the limitations that a project like this has, namely that, to further increase the accuracy of this or a model of similar nature, a far greater dataset of quality and quantity will be required, along with training for much greater variations and elements by using data based on actual drawings that contain noise as well, because as our results concluded there are bound to be limitations to a synthetically produced image. With the work done in this paper multi-model approach, enough groundwork has been established for further extension of this work as deep learning, and computer vision are both emerging fields.

Author Contributions: S.R. Conceptualization and Methodology; A.A. Software, Writing—review & editing; S.S. funding acquisition; and A.M. Editing.

Acknowledgement: We would like to thank Sir Syed CASE Institute of Technology for their resources and help throughout both the development of this project, as well as our time gaining the knowledge and tools we would need to succeed in the professional world and Tony Beltramelli for providing us with the base dataset and his precious work on his paper on which we extended our research.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Robinson, "Sketch2code: Generating a website from a paper mockup," *ArXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1905.13750.

[2] M. Sonka, V. Hlavac and R. Boyle, "Image processing, analysis, and machine vision," *Thomso*, pp. 507–542, 1993.

[3] X. Zhang and S. Xu, "Research on image processing technology of computer vision algorithm," in *2020 Int. Conf. on Computer Vision, Image and Deep Learning (CVIDL)*, Chongqing, China, pp. 122–124, 2020.

[4] T. Beltramelli, "Pix2code: Generating code from a graphical user interface screenshot," *ArXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1705.07962.

[5] K. N. Haque, M. A. Yousuf and R. Rana, "Image denoising and restoration with CNN-LSTM encoder decoder with direction attention," *ArXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1801.05141.

[6] J. Ferreira, A. Restivo and H. S. Ferreira, "Automatic generation of synthetic website wireframe datasets from source code," M.S. Dissertation, University in Porto, Portugal, 2020.

[7] A. Garcia, "A review on deep learning techniques applied to semantic segmentation," *ArXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1704.06857.

[8] J. Han, D. Zhang, G. Cheng, N. Liu and D. Xu, "Advanced deep-learning techniques for salient and category-specific object detection: A survey," In *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 84–100, 2018.

[9] E. Nishani and B. Çiço, "Computer vision approaches based on deep learning and neural networks: Deep neural networks for video analysis of human pose estimation," in *2017 6th Mediterranean Conf. on Embedded Computing (MECO)*, Bar, Montenegro, pp. 1–4, 2017.

[10] P. Mayhew and C. Worsley, "Software prototyping: The management implications", P.h.D. dissertation, 2019.

[11] S. Yacoob, "Paving the way for software prototyping," [Online]. Available: https://www.justinmind.com/blog/ui-sketching/ seen June 2021.

[12] S. Sutipitakwong and P. Jamsri, "Pros and cons of tangible and digital wireframes," in *2020 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, pp. 1–5, 2020.

[13] "Refsnes Data, "Introduction to html," [Online]. Available: https://www.w3schools.com/html/ seen June 2021.

[14] "Winnie Nguyen, "Why is Low-fidelity wireframe important in product design?" Available: https://uxdesign.cc/why-low-fidelity-wireframe-curious-in-product-design-c7bea87bc23d seen June 2021.

[15] J. Erikson, "Smart UX: High fidelity wireframes," Available: https://usabilitygeek.com/smart-ux-high-fidelity-wireframes/ seen June 2021.

[16] Y. Lecun, Y. Bengio and G. Hint, "Deep learning review," Available: http://www.personal.psu.edu/lxx6/DeepLearning.pdf seen June 2021.

[17] Refsnes Data, "Introduction to css," Available: https://www.w3schools.com/css/ last seen June 2021.

[18] Introduction to javascript," Available: https://www.w3schools.com/js/ seen June 2021.

[19] Misko Hevery, "What is angularjs?" Available: https://docs.angularjs.org/guide/introduction seen June 2021.

[20] J. Alammar, "The illustratedword2vec," [Online]. Available: https://jalammar.github.io/illustrated-word2vec/ seen July 2021.

[21] J. Brownlee, "A gentle introduction to calculating the bleu score for text in python," *Deep Learning for Natural Language Processing*, 2017. [Online]. Available: https://machinelearningmastery.com/calculate-bleu-score-for-text-python/.

[22] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez *et al.,* "Deep learning vs. traditional computer vision", *Advances in Computer Vision. CVC 2019. Advances in Intelligent Systems and Computing*, vol. 943, Cham, Springer, 2020.
[23] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with remaui (t)," in *2015 30th Ieee/Acm Int. Conf. on Automated Software Engineering (Ase)*, Lincoln, NE, USA, pp. 248–259, 2015.
[24] G. Vamvakas, B. Gatos, N. Stamatopoulos and S. J. Perantonis, "A complete optical character recognition methodology for historical documents," in *2008 the Eighth IAPR International Workshop on Document Analysis Systems*, Nara, Japan, pp. 525–532, 2008.
[25] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin and D. Tarlow, "Deepcoder: Learning to write programs," *ArXiv*, 2017.
[26] J. A. Landay and B. A. Myers, "Sketching interfaces: Toward more human interface design," in *Computer*, vol. 34, no. 3, pp. 56–64, 2001.
[27] R. G. Zhou and D. Q. Liu, "Quantum image edge extraction based on improved sobel operator," *International Journal of Theoretical Physics,* vol. 58, pp. 2969–2985, 2019.
[28] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *ArXiv*, 2021.
[29] Y. Liu, M. M. Cheng, X. Hu, J. W. Bian, L. Zhang *et al.,* "Richer convolutional features for edge detection," *ArXiv*, 2013. [Online]. Available: https://arxiv.org/pdf/1612.02103.pdf.
[30] O. Ronneberger, P. Fischer and T. Brox "U-Net: Convolutional networks for biomedical image segmentation," *ArXiv*, 2015 [Online]. Available: https://arxiv.org/pdf/1505.04597.pdf.
[31] F. Shammala and W. Ashour, "Color based image segmentation using different versions of Kmean in two spaces," *Global Advanced Research Journal of Engineering, Technology and Innovation*, vol. 1, no. 9, pp. 30–41, 2013.
[32] R. Rana, J. Epps, R. Jurdak, X. Li, R. Goecke, M. Breretonk and J. Soar *et al.,* "Gated recurrent unit (gru) for emotion classification from noisy speech," *ArXiv*, 2016 [Online]. Available: https://arxiv.pdf/1org/801.06105.pdf.
[33] J. Brownlee, "Why one-hot encode data in machine learning?" *Data Preparation*, 2017. [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/ seen July 2021.
[34] T. Nayak and H. T. Ng, "Effective modeling of encoder-decoder architecture for joint entity and relation extraction," *Thirty-Fourth AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8528–8535, 2020.
[35] T. Beltramelli, "Github," [Online]. Available: https://github.com/tonybeltramelli/pix2code seen July 2021.
[36] Thomas Wood, "What is the softmax function?" Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer/seen July 2021.
[37] I. Sriram and A. K. Hosseini "Research agenda in cloud technologies," *ArXiv*, 2010. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1001/1001.3259.pdf.
[38] Y. Hu, A. Huber, J. Anumula and S. Liu, "Overcoming the vanishing gradient problem in plain recurrent networks," *ArXiv*, 2018 [Online]. Available: https://arxiv.org/abs/1801.06105.
[39] Twitter bootstrap, [Online]. Available: https://getbootstrap.com/2.0.2/ seen July 2021.
[40] R. Liu, J. Mao, "Research on improved canny edge detection algorithm," in *2018 2nd International Conference on Electronic Information Technology and Computer Engineering (EITCE 2018)*, Berlin, vol. 232, pp. 1–4, 2018.
[41] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM–a tutorial into long short-term memory recurrent neural networks," *ArXiv*, 2019. [Online]. Available: https://arxiv.org/pdf/1909.09586.pdf.
[42] S. Yuheng and Y. Hao "Image segmentation algorithms overview," *ArXiv*, 2017. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1707/1707.02051.pdf seen July 2021.
[43] L. Yang, Z. Ma, L. Zhu and L. Liu, "Research on the visualization of spatio-temporal data," in *6th Annual 2018 Int. Conf. on Geo-Spatial Knowledge and Intelligence*, Hubei, China, vol. 234, no. 1, pp. 012013. IOP Publishing, 14–16 December, 2019.

[44] G. Litjens, C. I. Sánchez, N. Timofeeva, M. Hermsen, I. Nagtegaal *et al.,* "Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis," *Scientific Reports*, vol. 6, no. 1, pp. 1–11, 2016.

[45] E. Bisong, "Tensorflow 2.0 and Keras," In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, 1st ed., vol. 1. New York: Apress, 2019.

[46] S. C. Wong, A. Gatt, V. Stamatescu and M. D. McDonnell, "Understanding data augmentation for classification: When to warp?" in *2016 Int. Conf. on Digital Image Computing: Techniques and Applications (DICTA)*, Gold Coast, QLD, Australia, pp. 1–6, 2016.