Research Article

AUTOMATED WEB DESIGN AND CODE GENERATION USING DEEP LEARNING

Vishaal Saravanan[1], Mohamed Arshad S S[1], N. Sabiyath Fatima[1] and P.Mahalakshmi [1&2]

[1]Department of Computer Science and Engineering B.S Abdur Rahman Crescent Institute of Science & Technology, Chennai.
[2]Department of Computer Science and Engineering, SRM Institute of Science and Technology, Kattankulathur.

Article History:Received:11 november 2020; Accepted: 27 December 2020; Published online: 05 April 2021

ABSTRACT : Excited by ground-breaking progress in automatic code generation, machine translation, and computer vision, further simplify web design workflow by making it easier and productive. A Model architecture is proposed for the generation of static web templates from hand-drawn images. The model pipeline uses the word-embedding technique succeeded by long short-term memory (LSTM) for code snippet prediction. Also, canny edge detection algorithm fitted with VGG19 convolutional neural net (CNN) and attention-based LSTM for web template generation. Extracted features are concatenated, and a terminal LSTM with a SoftMax function is called for final prediction. The proposed model is validated with a benchmark based on the BLUE score, and performance improvement is compared with the existing image generation algorithms.

Keywords: Attention mechanism, Word embedding, CNN, Code generation, Template generation

1.INTRODUCTION

Ongoing demand for websites ranging from Blogs, E-commerce, Product pages has increased considerably in recent years due to the more open access to the internet. A good website design adds to increased net revenue and customer churn rate in most cases. Nowadays, websites are available almost in every field, from health, shopping, games, marketing, and many more—some functional websites for advertising and product marketing and others to provide services.

According to recent statistics, even in heavily populated countries like India, only 5.15% of the population have basic computer knowledge. Even though services like E-ticket bookings, automated billings, and Digital Payments have reached several areas in India, with all the infrastructure, the computer literacy rate should be at least 35% to 40% more than the current figure. After much younger populations are moving into digital trends, many small-scale businesses get forced to move online to survive. The Involvement of modern-day machine learning algorithms could help web designers and the common public convert their hand-drawn website templates in paper to HTML code and static websites.

Creating appealing experiences for users is an essential goal for businesses of *all* sizes, and it's a long process driven by cycles of prototyping, designing, and user testing. Large enterprises like Google have the resources to dedicate entire workflow to the design process, which could take several weeks and include various stakeholders. Still, in most cases, small businesses don't have these resources, and their user interfaces may suffer the consequence.

Online sources confirm that companies like Airbnb are already showing significant advances in sketching interfaces into websites. So, it is evident that Automatic Code Generation (ACG) will change the current market space. Since most design workflow today follows a cycle of, Product managers performing requirements check to formulate a list of user specifications. Followed by designers developing prototypes ranging from storyboards to high fidelity designs and programmers for implementing those designs into code and finally ship the product to users. Following such a workflow might eventually involve cost and time weights engaged in each step; thus, advancements in ACG's work cycle will significantly reduce labor cost, resource consumption and promote quicker delivery.

Currently, the biggest hindrance to automating front-end development is the lack of computing power. However, gaining significance in the recurrent neural network fields like Long Short-Term Memories (LSTM) and the areas of Convolutional Neural Nets (CNN) made these ACG strategies possible. And by adding an attention mechanism over the existing model with human experts governing the model's training process, neural models can create more natural and industry-ready website blocks. Shortly, the most significant impact will come from

*Corresponding author: Vishaal Saravanan
Department of Computer Science and Engineering B.S Abdur Rahman Crescent Institute of Science & Technology, Chennai

building a scalable way to synthesize data, by which even modifying fonts, words, resizing components, and transitions would be possible.

2.RELATED WORKS

Y. Miao et.al., [1], explicitly state that the graphical user interface (GUI) should have a large amount of source code for a better user experience (UX). To address this, the authors come up with a search program called SUISE, which is used to overcome the difficulty while designing a graphical user interface of user specification. However, this research fails to expose complex interfaces that tend to be tightly integrated with the rest of the application. Since real-world applications often depend on external packages or external systems forming complex GUI's. Mahalakshmi and Sabiyath Fatima [25] illustrated the wide representation of concepts and the technique that has been carried out in processing the textual information. In addition, the authors also mentioned the need for the automated process to extract the features in processing the textual information.

T. A. Nguyen et.al., [2] discussed developing mobile user standard interfaces for android platforms using automatic code generation. Despite its favorable outcome, the research still doesn't offer a solution for adaptability with variable screen sizes and support for cross platforms like IOS and web applications. This makes the study vulnerable to users in need of cross-platform support.

S. Natarajan and C. Csallner, [3], developed a tool called P2A to convert pixel into user interface applications by using computer vision techniques and REMAUI to identify visual elements and structure them. The advantage is that it supports the inter-screen transition and in-screen animation. But their approach doesn't address the need for scalability of the proposed system to large scale interfaces.

T.Beltramelli, [4], the author, generates code using GUI images with deep learning to create a training model to output the code for a single image. This is indeed a great approach towards automatic code generation; nevertheless, the model is trained on a smaller set of parameters and data points, making the model vulnerable to underfitting. Proposed a approach has used a much larger set of data sources and parameter loss functions to drive the accuracy and presenting the system as more stable.

K.P.Moran et.al., [5] In this paper for the transformation process of mock-ups of GUI into code is approached by three steps. To improvise this model, attention mechanism have been implemented for better supporting the detection of components.

S. Barnett, et.al.,[6] introduce RAPPT tool, which generates the structure of the mobile application based on the description mentioned in a DLS (Domain Specific Language). The main limitation of RAPPT is that it cannot be used after the code has been modified. Thus, the scalability of the DSL and the robustness of the underlying meta-model are compromised.

M. Book et.al.,[7] study that web-based applications' presentation and application logic hide the dialog logic control in most cases. It has been presented to avoid dialog control logic reimplementation. The notation's weakness may currently be the fine granularity of grained more acceptable actions to adapt quickly to different interaction patterns required to reuse them flexibly in dialog graphs on other presentation channels. It is thus making the proposed model suspectable to overfit the given sample.

V. Lelli et al. [8] limit interactions with GUI to a maximum of four to reduce errors. Considering such a decision boundary has shown promising results. Still, this experiment shows that if current GUI testing tools have demonstrated for finding GUI errors, they also fail to detect several GUI faults making them less adaptable to a real-world environment.

A.Antonacopoulos et al.[9] discusses ICDAR page segmentation to evaluate the latest and old page segmentation methods using real datasets and objective performance analysis. Despite showing promising results, numerous studies already reveal that implementations like Fraunhofer segmentation have a clear advantage in improving after learning real-life datasets.

Bo Dai et al.[10] studies image captioning by use of Conditional Generative Adversarial Network (CGAN) with policy gradient. Such an approach would also be useful for web mockup generations. This model's terminal results are much better than the existing systems, but it has some failure cases like the frequent inclusion of incorrect details. Also, since CGAN's require adequate processing power and time, we cannot deploy this method at a larger scale in real-time.

J. Canny, [11] the approaches are mainly based on two criteria: detection and localization in mathematical forms. For better detection, the two measures were compared together with multiple datasets. Such detectors are used in edge detection in images by varying intensities of pixels. With the maximum number of sensors, the result is efficient in the edge detection's computational process. For two dimensional images, the computational approach is yet to improve.

Prasang Gupta and Swayambodha Mohapatra [12], provide a pipeline for converting the code from the hand-drawn images despite the model's training consumes a lot of time and also being computationally expensive, it was proven to achieve the desired result. But in Gupta’s work a approach overcome computational expensiveness by using the attention mechanism to highlight the essential components, thus reducing the overall load of the distribution significantly.

S. A. Agrawal et al. in [13] claims that their approach can be extended to multiple platforms. Still, no specific method or algorithm is mentioned here for an automated system instead focuses only on the design process. Deep learning concepts like CNN, LSTM can significantly improve GUI and quality of code, but their study fails to address them.
João Carlos Silva Ferreira [14] In this approach, the detection of containers in hand-drawn mockups did not seem to be structured, also not seem feasible to be implemented due to some small miss alignments or larger gaps in the final mockup.
BatuhanAs [15] studied automating the code generalization process from hand-drawn mockups with minimum time and labor cost. But the quality of the final template images didn't meet today's design standards.
Rajat Kanti Bhattacharjee et al Jha [16] explains generating intricate handloom designs. They had explored multiple methods, but no specific algorithm is mentioned here to convert the hand-drawn mask to user design. Instead, the survey focus only on the dataset and design process, which can significantly improve GUI and quality of code; by changing the dataset supplied to our current model, our model pipeline can be scalable to such user-specific use cases.

3.PROPOSED METHODOLOGY

3.1 System Architecture

The proposed model architecture Fig.1 for CNN and the LSTM mechanism inside the proposed model. Publicly available datasets of pix2code and image captioning datasets of Flickr8k are separated into test and training sets in the ratio of 1:5, respectively. More on datasets gets discussed in a later section of the proposed research. For training, LSTM1 takes inputs of HTML code contexts (one hot encoded tokens corresponding to DSL code) of their equivalent template images supplied to the CNN network with attention-based LSTM. Finally, features are concatenated and fed into a decoder containing LSTM layers and a SoftMax function for producing code snippets and GUI images. Similarly, during sampling, the input connection is updated for specific predictions to carry the current predicted token. The final distribution of DSL tokens is compiled to the desired target language using conventional compiler design methods.

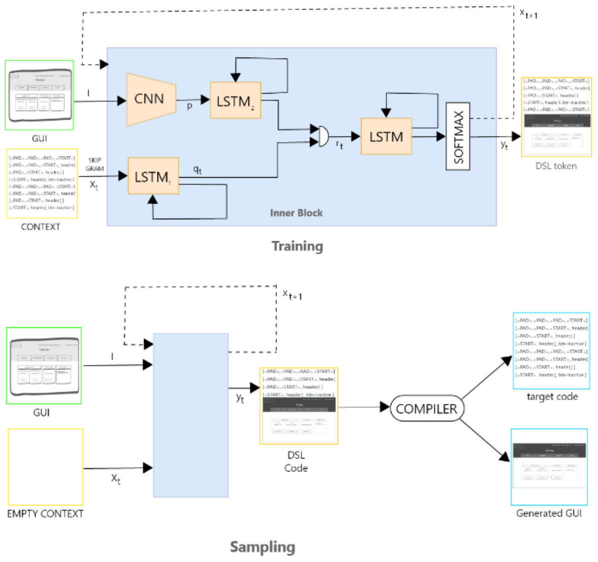


Fig.1 CNN and LSTM mechanism for training and sampling

3.2IMAGE PREPROCESSING

Before passing on images as inputs to CNN, the images are preprocessed by applying the Canny Edge Detection Algorithm to make the model more robust and reliable for prediction.

3.2.1 Canny Edge Detection

To understand the canny edge detection algorithm, it is necessary to understand edge and Edge detection's terminologies. Edges are an image discontinuity formed by object boundaries, and it also associates with color discontinuity, image intensity discontinuity, and image density discontinuity. Four types of edges include step, line, ramp, and roof (convex and concave). In contrast, edge detection is the process of finding meaning full transition in an image. These are points where the sharp changes in the brightness occur typically form the border between different objects. A machine can easily detect these points based on intensity differences and suppresses the noise while detecting the edges flawlessly. The canny algorithm takes six steps to detect the edges:
Step 1. Conversion to the GrayScale: Convert the input RGB image to a Gray Scale image.

Step 2. Gaussian Blur**:** It is an operator, which helps in removing the noise in the input image. The sigma value has to be set approximately for better results.

Step 3. Intensity Gradient Calculation**:** This process is to find the edges' gradient and every pixel's direction, vertically and horizontally. The sobel filter [18] is used over the input image. with the use of any gradient operator, the final result should be the gradient magnitude of the picture. The gradient approximation with

$$\boldsymbol{Edge\ Detection} = \sqrt{G_x^2} + \sqrt{G_y^2} \quad \textbf{(1)}$$

The formula for finding edge direction

$$\boldsymbol{A\ ngle}(\boldsymbol{\Theta}) = \tan^{-1}(G_x/G_y) \quad \textbf{(2)}$$

Where G_x and G_y are the gradient magnitudes in the principal x and y axis respectively. The gradient should always be perpendicular to the edges, and it should represent two diagonal directions, horizontal and vertical directions.

Step4. Non-Maximum Suppression**:** The gradient magnitude operators discussed in the previous stage typically obtain thick edges. But the final image is expected to have thin edges. Hence, the process of non-maximum suppression shall enable us to derive thin edges from thicker ones. The picture gets fully scanned to remove unnecessary pixels that are not associated with the edges. For this, every pixel is checked if it is in the direction of the gradient.

Step 5. Thresholding**:**The thresholding technique is used to set the maximum and minimum edge intensities for a range of pixels to be evaluated as an edge. A high threshold value is any pixel with a value above 0.8 is seen as a more robust edge, and a value less than 0.2 is not considered an edge. Weak edges are the ones having edge values between 0.2 and 0.8.

Step 6. Edge Tracking**:**To understand which of the weaker edges are solid edges. Evaluate the weak edges connected to strong edges as our actual edges and retain them, and we can omit the rest of the edges. The preprocessed image vector is passed in as an input to CNN and LSTM for ACG and web template generation.

3.3Convolutional Neural Network (CNN)

CNN was original research published by Geoffrey E. Hinton et al. [19] in 2012. It has created a remarkable impact and pioneered research for more advanced computer vision algorithms. In the essence of simplicity, CNN is a deep learning algorithm that can take in an input image, attach importance (learnable weights and biases) to multiple objects in the image, and distinguish one from the other by series of convolution, pooling, and flattening operations. Finally, the flattened layer gets inputted to a fully connected Artificial Neural Network for further classification. A CNN architecture called Very Deep Convolutional Net [20,21] (VGG16) trained on ImageNetis used to reduce the computing time and GPU load on our machine.

3.4Long Short-Term Memory (LSTM)

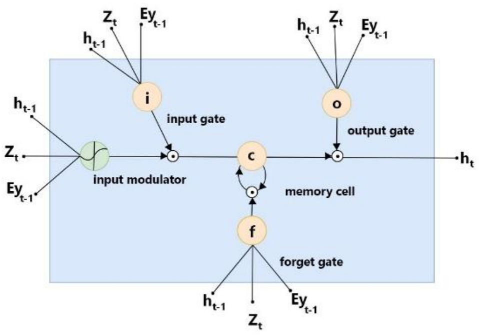
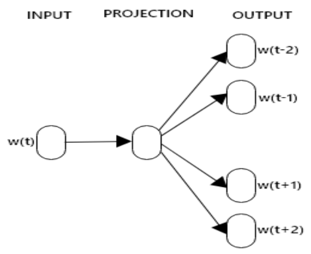
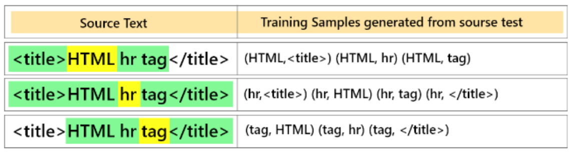


Fig.2 LSTM cell with projections learned from weighted feature vector. Learning weights by input gate, modulating the memory contribution by input modulator, erasing the memory block by forget gate, and controlling memory emission by output gate.

Most deep learning algorithms widely used recurrent neural networks to remember and pass down the forward network's retained information. Still, they suffered from short memory, which gave rise to modern-day LSTM's. They have an internal mechanism containing forget, cell, input, and output gate combined with series of activation functions for retaining relevant information for a long time, as shown in Fig.2.

Inputs for LSTM1 are the code blocks for the respective hand-drawn images from the dataset. Each code block is encoded with a word embedding technique called skip-gram. The goal of skip-gram based architecture is to learn the weights of the word vectors (amount of correlation between each HTML tags) that we are trying to predict.Fig.3 explains skip-gram architecture and shows the example of skip-gram in HTML tagging. Where weight of the tags w(t) is projected into number of multiple independent skip labels.





Source Text	Training Samples generated from sourse test
<title>HTML hr tag</title>	(HTML,<title>) (HTML, hr) (HTML, tag)
<title>HTML hr tag</title>	(hr,<title>) (hr, HTML) (hr, tag) (hr, </title>)
<title>HTML hr tag</title>	(tag, HTML) (tag, hr) (tag, </title>)

Fig.3 Skip-Gram model architecture and example explaining skip-gram HTML tagiing

The input vector dimensions will be 1xV, where V is the number of statementsor HTML tags in the vocabulary i.e.,one-hot representation of the word. Each hidden layer will have the dimension VxE, where E is the word embedding size and its hyper-parameter. The hidden layer's output would be of the dimension 1xE, which will get feeded into a SoftMax [22]. Similarly, the output layer will be of 1xV dimension, where each vector will be representing the probability score of the target word at that position.

3.5Attention Mechanism:

Features extracted from the CNN are passed on to a newer LSTM2 node as an input to keep track of contour variation and enables the model to learn the contents of HTML elements automatically. As the model identifies each feature vector from the CNN output, the model's attention mechanism gradually shifts and focuses on the different HTML components on the same image, as shown in the Fig 4.

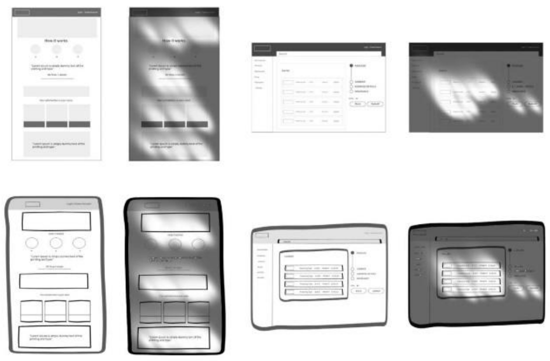


Fig.4 Web images with no attention mechanism on the left and on the right with hard and soft attention applied

This idea of adding an attention mechanism to the LSTM architecture was inspired by works of Kelvin Xu et al., Mahalakshmi & Sabiyath Fatima [23,20] on image captioning. The LSTM architecture is implemented based on the results of [24,20], defining simple affine parameters with the transformation as,

$$\begin{pmatrix} i_t \\ f_t \\ o_t, g_t \end{pmatrix} = \begin{pmatrix} 6 \\ 6 \\ 6, tanh \end{pmatrix} T_{D+m+n,n} \begin{pmatrix} Ey_t - 1 \\ h_t - 1 \\ \hat{z}_t \end{pmatrix} \textbf{(3)}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \mathrm{g_t} \qquad \textbf{(4)}$$

$$h_t = o_t \odot \tanh(\mathrm{c_t}) \qquad \textbf{(5)}$$

Here, i_t, f_t, c_t, o_t, h_t, m, n is the input, forget, memory, LSTM output states, hidden state of the LSTM, embedding vector and dimensionality associated with LSTM respectively. The vector $z \in \mathbb{R}^D$ is the context vector, for capturing the hand drawn data for the given input location. $\odot$ and 6 be the element wise multiplication and logistic sigmoid activation function respectively. The probability that model focuses on right places (hard stochastic attention mechanism) for each location i, is given by positive weight $\alpha_{i.}$ α_i is computed by our attention model f_{att} in the hidden state h_{t-1}. Thus, calculation of weights, and context vector $\hat{z}_t$is done by

$$e_{ti} = f_{att}(a_i, h_t - 1) \textbf{(6)}$$

$$\alpha_{ti} = exp(e_{ti})\textbf{(7)}$$
$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{L} exp(e_{tk})}\textbf{(8)}$$

$$\hat{z}t = \phi(\{ai, \{\alpha_{ti}\}) \qquad \textbf{(9)}$$

where φ, is a function returning a single vector containing corresponding weights. Further as suggested for optimizing the final equation, the dataset is sampled using Monte Carlo distribution with exponential decay involved. Similarly, by taking context vector $\hat{z}_t$ directly we formulate final deterministic equation for soft visual attention by considering soft annotation $\alpha_{t,i}$ vector attention

$$\sum_{i=1}^{L} \alpha_{t,i} a_i \textbf{(10)}$$

3.5 Encoder

While the inputted image passing through canny edge detection algorithms is fed into the CNN attached with attention mechanism in LSTM's used for object detection. The other set of dependent LSTMs is provided with inputs from the word embedding for Automatic Code Generation (ACG) is stacked with One-hot encoders for the final output. This one-hot encoding arrangement establishes a relationship in terms of digits and HTML tags. Furthermore, a time-distributed layer is applied to form high-level features with inputs from both the LSTM's independently. Finally, A concatenated layer is introduced to create a final set of image-HTML features.

3.6 Decoder:

Outputs from the encoder are passed on as inputs to the following decoder, where a dense layer stacked with a newer set of LSTMs is used for the final prediction of the tags and mockups. Initially, the LSTM layer has the sequence set to false. Instead of returning the length of the input sequence, the model predicts one feature at a time. In the proposed case, it's the next tag and corresponding image feature vector that contains the information for the final output. Finally, a SoftMax activation outputs the last set of labels and output image.
By Fig.5, Encoder and Decoder pipeline are visualized in simpler intermittent steps.

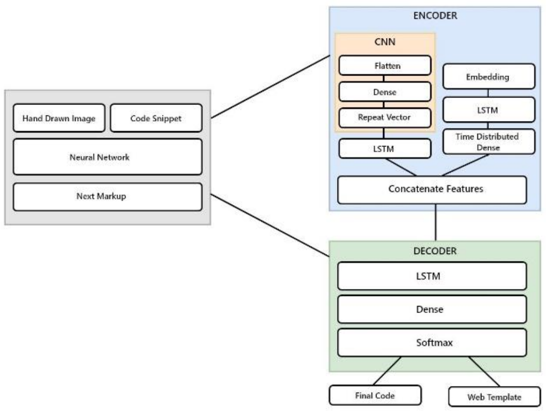


Fig.5 Intermittent phases in encoder and decoder pipeline

4. IMPLEMENTATION:

The pix2code dataset available at [26] is used with some modification to make the web images look like hand-drawn web templates using open-source tools like PIL, PGMAGICK, OPENCV, SCIKIT-IMAGE, and SIMPLE CV. Simple adjustments like border-radius, border thickness, grayscale, drop shadows, font styles, shifts, skews, and image rotations were added to address the anomalies and mimic hand-drawn images. Also, LSTM was initially trained and allocated weights based on the randomly sampled flickr8k from Kaggle [27] dataset for its attention mechanism. And pre-trained CNN with images available on ImageNet [28] is used in our research.

4.1 Module Flow

Step 1: The pix2code dataset is split into a set of test and training with the ratio of 1:5.

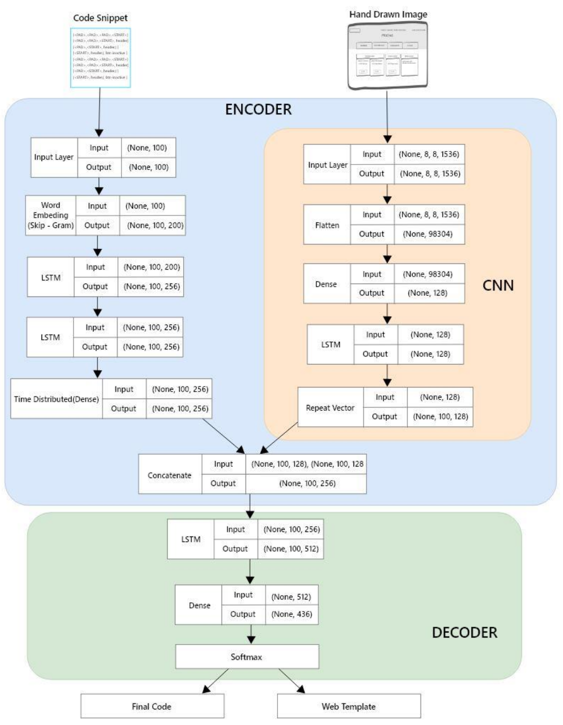

Code Snippet
Hand Drawn Image
ENCODER
Input Layer | Input (None, 100) | Output (None, 100)
Word Embeding (Skip - Gram) | Input (None, 100) | Output (None, 100, 200)
LSTM | Input (None, 100, 200) | Output (None, 100, 256)
LSTM | Input (None, 100, 256) | Output (None, 100, 256)
Time Distributed(Dense) | Input (None, 100, 256) | Output (None, 100, 256)
Input Layer | Input (None, 8, 8, 1536) | Output (None, 8, 8, 1536)
Flatten | Input (None, 8, 8, 1536) | Output (None, 98304)
Dense | Input (None, 98304) | Output (None, 128)
CNN
LSTM | Input (None, 128) | Output (None, 128)
Repeat Vector | Input (None, 128) | Output (None, 100, 128)
Concatenate | Input (None, 100, 128), (None, 100, 128 | Output (None, 100, 256)
LSTM | Input (None, 100, 256) | Output (None, 100, 512)
Dense | Input (None, 512) | Output (None, 436)
DECODER
Softmax
Final Code
Web Template

Fig.6 Proposed Model Pipeline.

Step 2: Then the model enters into the first stage of the pipeline, where the Canny Edge Detection (CDE)algorithm is applied to detect the edges, which involves conversion to grayscale, gaussian operation, intensity gradient calculation, followed by non-maximum suppression with thresholding and edge tracking.
Step 3: Then output feature vectors are inputted to the VGG16 CNN to extract image features,
Step 4: Feature vectors are then fed into the LSTM node fixed with an attention mechanism to focus on only template contours.
Step 4.1: Simultaneously, an independent LSTM network with word embedding technique skip-gram, learns the markup tags for the images supplied.
Step 5: A concatenated layer binds the features of the markups and images into a single flattened layer as required as input by the decoder mechanism.
Step 6: The decoder mechanism, which consists of the LSTM layer, dense layer, and the SoftMax function, makes the final prediction for the generated image and markup tags. A complete model-flow in Fig.6 illustrates each step's input and output dimensions in the model's pipeline.

5.RESULT AND ANALYSIS:

Comparing the interpretability of the generated web template and the markup tags is a complex process because:

1. Though HTML tags can be compared with existing ones or execute them for errors in a browser or an IDE manually, it wouldn't be the ideal error checking convention.
2. The design's adaption is narrowed down to an individual user perspective, making the process more complicated.
3. It’s not possible to compare the ensemble of ensemble models versus a single model. Though many methods are proven to yield more performance by ensembling, we restrict our results to single model performance.

Results for the proposed architecture are listed in the Tab below. The result is reported in terms of the BLUE metric. Tab1 holds the value for the mean blue score(mean), standard deviation(std), a maximum blue score of the sample(max), a minimum blue score of the sample(mini), distribution of the 25%,50% and 75% blue score population respectively.

5.1 Bilingual Evaluation Understudy (BLUE):

It's the ideal strategy for comparing machine-generated images with text generation (in our case, markup tags). It compares the generated labels with the existing labels to create a precision metric. A BLEU score of 0 means that the image generated is irrelevant, and a BLEU score of 1 means that the image is perfectly recreated. Under different evaluation environments blue scores are compared and findings are reported in Table.1

.

	BLUE Score		
Baseline Model (In absence of Canny Edge Detection, Word Embeddingand Attention Mechanism,)		Training set	Test set
	Mean	0.612	0.524
	Std	0.242	0.267
	Min	0.464	0.156
	25%	0.501	0.454
	50%	0.525	0.535
	75%	0.701	0.539
	Max	0.869	0.679
Model1(With VGG16 CNN, Canny Detection, Word Embedding)		Training set	Test set
	Mean	0.834	0.606
	Std	0.081	0.408
	Min	0.772	0.184
	25%	0.799	0.419
	50%	0.852	0.697
	75%	0.888	0.815
	Max	0.919	0.972
Proposed Model(With VGG16 CNN, Canny Detection, Word Embedding and Attention mechanism)		Training set	Test set
	Mean	0.95	0.89
	Std	0.25	0.38
	Min	0.64	0.44
	25%	0.87	0.53
	50%	1.26	0.52
	75%	1.17	1.22
	Max	1.19	1.88

Table.1 Experimental results based on the BLUE score.

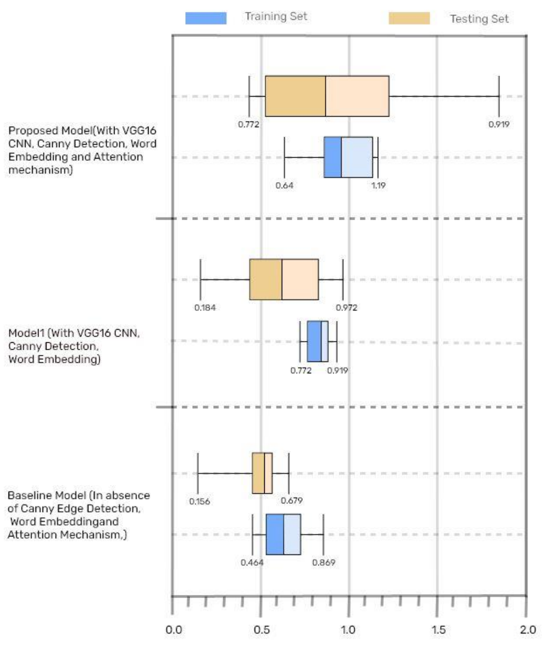


Fig.7 BLUE score comparison.

The generated web templates are examined pictorially to understand the critical difference in the model outputs. One such finding is given in Fig.7.Furthermore, the proposed model, on average at its best-case scenario, confined anywhere between 0.89(+0.04 to -0.04) or 89% in the mean BLUE score study in the test set and between 0.96(+0.04 to -0.04) or 96 % in its training set, respectively. The model constantly outperformed the existing approach to hand-drawn image conversion by a minimum difference of over 20% in the training set and 30% with the baseline model's test set. Similarly, proposed pipeline showed almost a 9% positive change in the training set and around 20% change in the test set with the modified baseline model. This result can be visualized from the fig.7

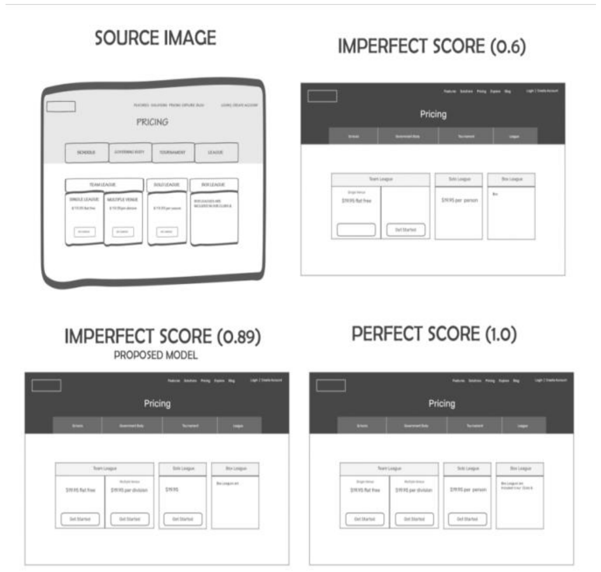


Fig 7. Different blue scores evaluation for the same web GUI

6. CONCLUSION AND FUTURE WORKS:

This research proposed new model architecture for converting the hand-drawn image to a web template with HTML code generation. The model uses attention-based LSTM's, VGG19 Convolutional Neural Nets, and web embedding vector techniques in the proposed model flow to validate the claim. With the contour detection supplemented by canny edge detection, the model can output more real-world web templates with accurate HTML code snippets. The proposed pipeline attains a BLUE score close to most human-made web templates spotted online, with a little trade-off between human resources distribution, product development time, the investment involved, and information representation. In most cases, the model passes all the above-mentioned use cases contrasting to the traditional workflow but misses out on representing all the information in some cases.

The proposed system wouldn’t run with the range of ensemble models; running such architecture has shown a significant performance increase in current research work in similar domains. With General Adversarial Nets (GAN's) [9] gaining momentum in recent years, it can be deployed to provide newer variations of web templates and illustrations, making the system more reachable and adaptable. Moreover, GAN's can also be useful for generating more unique data samples in case of underfitting.

References

Y. M. a. Q. x. S. P. Reiss, "Seeking the user interface," in IEEE Conference paper, 2018.
C. C. T.A Nguyen, "Reverse Engineering Mobile Application User Interfaces with REMAUI," in IEEE Conference paper, 2015.
C. C. S. Natarajan, "P2A: A Tool for Converting Pixels to Animated Mobile Application User Interfaces," in IEEE Conference paper, 2018.
T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in IEEE Conference paper, 2017.
C. B.-C. M. C. R. B. a. D. P. K. P. Moran, "Machine learning-based prototyping of graphical user interfaces for mobile apps," in IEEE Conference paper, 2018.
R. V. a. J. G. S. Barnett, "Bootstrapping mobile app development," in IEEE Conference paper, 2015.
V. G. M. Book, "Modeling web-based dialog flows for automatic dialog control," in IEEE Conference paper, 2004.
A. B. a. B. B. V. Lelli, "Classifying and qualifying GUI defects," in IEEE Conference paper, 2015.
S. P. D. B. a. C. P. A. Antonacopoulos, "ICDAR 2009 page segmentation competition," in IEEE Conference paper, 2009.
D. L. R. U. a. S. F. Bo Dai, "Towards Diverse and Natural Image Descriptions via Conditional GAN," in IEEE Conference paper, 2017.
J. Canny, "A computational approach to edge detection," in IEEE Conference paper, 1986.

Prasang Gupta and Swayambodha Mohapatra, "HTML Atomic UI Elements Extraction from Hand-Drawn Website Images using Mask-RCNN and novel Multi-Pass Inference Technique”, 2020.

S. A. Agrawal, Swapnil Suryawanshi, Vikas Arsude, Nilesh Maid, "Artificial Intelligence based Automated HTML Code Generation Tool using Design Mockups”, 2020.

João Carlos Silva Ferreira., " Live Web Prototypes from Hand-Drawn Mockups " in IEEE Conference paper, 2019.

BatuhanAs¸ıro˘glu., " Automatic HTML Code Generation from Mock-up Images Using Machine Learning Techniques" in IEEE Conference paper, 2019.

Rajat Kanti Bhattacharjee, Amrit Jha, “Handloom design generation using generative networks”in IEEE Conference paper, 2020.

Wen Zhou “WebVR Human-centered Indoor Layout Design Framework using a Convolutional Neural Network and Deep Q-learning. " in IEEE Conference paper, 2019.

Vincent, Olufunke & Folorunso, Olusegun. (2009). A Descriptive Algorithm for Sobel Image Edge Detection.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

Mahalakshmi, P., Fatima, N.S. Ensembling of text and images using Deep Convolutional Neural Networks for Intelligent Information Retrieval. *Wireless Personal Communication* (2021)

Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

Martins, André & Astudillo, Ramon. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification.

Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: neural image caption generation with visual attention. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15). JMLR.org, 2048–2057.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, September 2014

P.Mahalakshmi and N.Sabiyath Fatima, “An art of review on Conceptual based Information Retrieval”, Webology Journal, volume 18, issue no. 1, pp. 51-61.

Tony Beltramelli. 2018. Pix2code: Generating Code from a Graphical User Interface Screenshot. In <i>Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems</i> (<i>EICS '18</i>). Association for Computing Machinery, New York, NY, USA, Article 3, 1–6. DOI:https://doi.org/10.1145/3220134.3220135

J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14). MIT Press, Cambridge, MA, USA, 2672–2680.