Machine Learning for Generating Code from Images

*Chandana Nikam[1], *Rahul Keshervani[2], *Shravani Shah[3], *Dr. Jagannath Aghav[4]

College of Engineering Pune

[1]*nikamca16.comp@coep.ac.in*, [2]*keshervanirav16.comp@coep.ac.in*, [3]*shahsp16.comp@coep.ac.in*, [4]*jva.comp@coep.ac.in*

Abstract

Generating code from images is a challenging task. The technical challenges include learning from images data, automation of iterative processes, and obtaining accuracy in ubiquitous images and users aiming the design of complex software systems. In this paper, we have proposed a method that generates code from images by applying techniques of machine learning. The input image is a screenshot taken from the user interface drawn. We have used classical computer vision techniques for images like canny edge detection, contour box detection, dilation and erosion, so as to attain highest accuracy in identifying the component for coding. We have used deep learning techniques of EAST text detection and convolutional neural networks for automating the iterative process between designer and web developer.

Keywords: *Computer Vision, Convolutional Neural Networks, Machine Learning, Deep Learning.*

1. Introduction

The creation of an application involves several steps. One of the essential steps in the development of an application is designing the Graphical User Interface(GUI). The GUI development process involves two teams, design team and software development team. Initially, a tentative design is sketched on paper or whiteboard by the design team. Then using tools like Photoshop, the tentative design is converted to a mock-up image.The mock-up is then sent to the developer team, which then converts it into a basic GUI code and adds more functionalities as required. The above process is time-consuming and inefficient. This process is deployed iteratively considering client feedback. Unnecessary efforts are spent by the teams in this iterative process. This process can be simplified by automating the conversion of GUI mock-ups to code. Other alternatives like GUI builder can be used, but they require the graphical designer to have knowledge about the source code hierarchies. Hence these alternatives are not popular in designers.

In recent years, there have been a few attempts to convert the mock-ups to code like pix2code, sketch2code, REMAUI, ReDraw. In this paper, we have tried different techniques to improvise current state-of-the-art models to get better accuracy. The conversion of mock-ups to applications will help designers as well as developers to

[1] Chandana Nikam,
[2] Rahul Kesharvani,
[3] Shravani Shah,
[4] Dr. Jagannath Aghav

complete the development process more efficiently. This reduces the time and efforts of a designer as well as a developer in the development process. The basic idea is shown in Figure.1.

Figure 1. Conversion of image mock-up into React Native Code

2. Synthetic Dataset Generation

The user inputs a mock-up GUI image of an application to our program. Looking at this image, the model must detect all the components present in the image. This model was trained on synthetically generated data. This dataset consists of several images of the components and their corresponding labels i.e. text-input, text, image or button, which are required for supervised learning. In order to generate this dataset, we have automated the process of writing code to generate individual components such as text-inputs, buttons, images, and text with varying attributes such as dimensions and colour. The process of taking screenshots of the generated components is also automated and it puts cropped images of all these components in the training dataset. These images were labeled according to the name of the component. The dataset generated by this method was used for the training of our model. The process of dataset generation is shown in Figure. 2.

Figure 2. Dataset Generation

3. Proposed Methodology

GUI Designers have their work pipeline defined as deciding on a design, sketching it and giving the sketch to a developer to develop a prototype code. This last step of converting the mock-up to code is carried out iteratively during the development of an application.

To simplify the process of converting the GUI design to the respective React Native code, we built a model using computer vision and machine learning. The GUI structure consists of basic components such as button, textbox, etc. After compiling the generated React Native code, we get a prototype app screen layout for the given GUI design. The whole proposed method is shown in Figure. 3. A multi-step process is followed to convert

the images provided by the user into a functional code. The process is broadly divided into three stages:

1. Preprocessing of image
2. Classification of components
3. React Native code generation

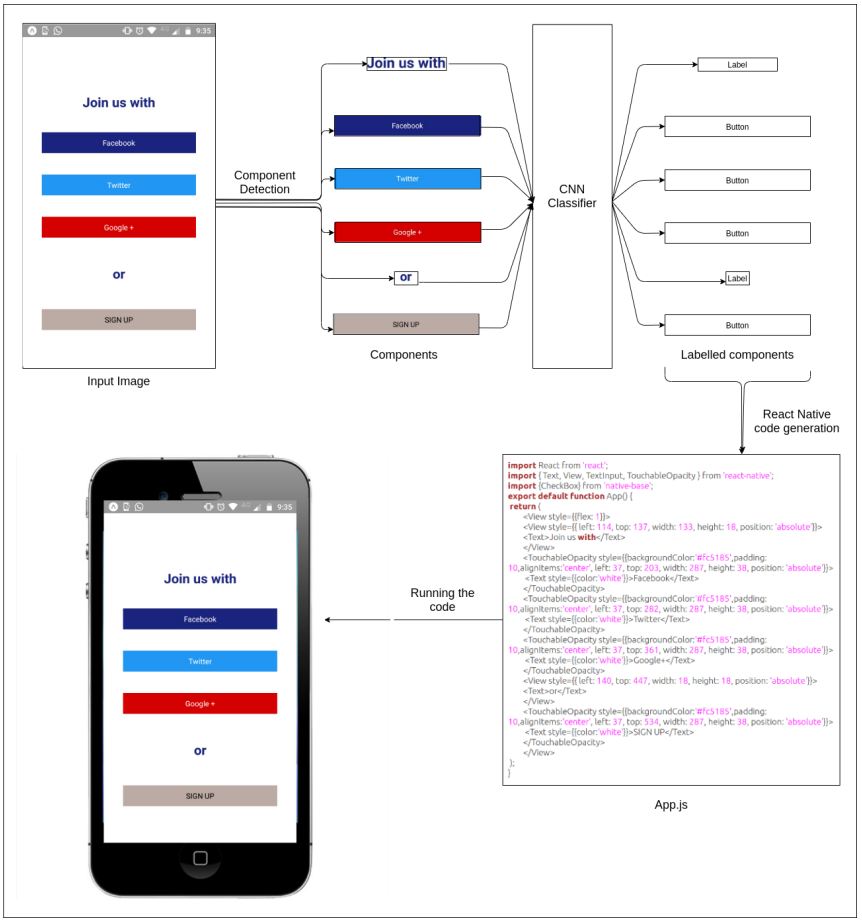

Figure 3. Proposed Methodology

The image is initially preprocessed. During preprocessing, the components are separated and we get images of individual components. Then the individual component images are passed through EAST and a convolutional neural network for classification and are labelled. Now, the co-ordinates as well as the dimensions of the components are known, and they are passed to the template code generator which creates the React Native code for the same and the user interface is created and can be opened in an android or on a web browser. Figure. 4. shows the complete flow of our proposed solution.

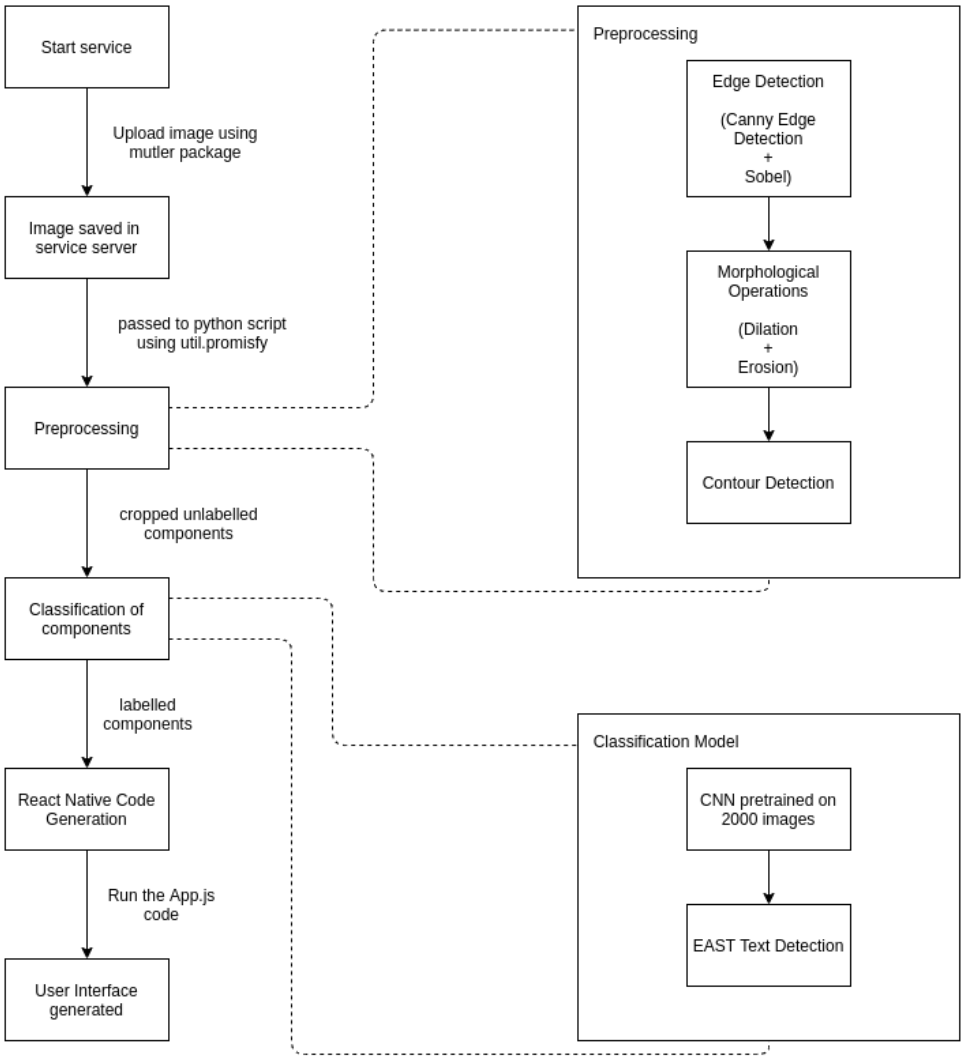

Figure 4. Complete flow of our proposed solution

4. Preprocessing of Image

The preprocessing stage uses various classical computer vision techniques. The input image is first converted to a binary image. For detecting the edges of each component in the binary image, Canny edge detection and Sobel operator are applied. Vertical and horizontal morphological operations are applied to the resultant image. The summation of this, gives contours from which the box detection is done. Using the coordinates obtained by box detection, components are extracted from the original image. Figure. 5. shows the steps carried out during the preprocessing stage upon receiving the input image.

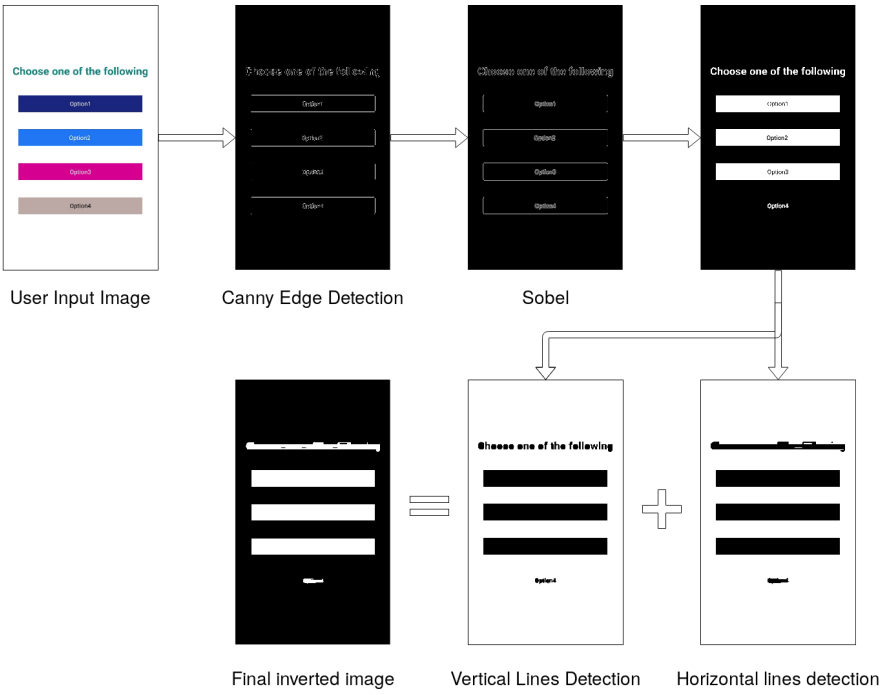

Figure 5. Preprocessing Stage

4.1. Edge detection operations

The initial step is to convert the image into its binary form as it helps reduce the dimensionality of the image. Then the Canny edge detection is applied to the binary image. Canny filter from skimage library of python, has three adjustable parameters:

1. Width of Gaussian filter (known as sigma)
2. Low threshold of the hysteresis.
3. High threshold of the hysteresis.

A reference paper [18] guided us while experimenting on the optimal values for the parameters. It was found that the best suited values for sigma, low threshold and high threshold are 2.0, 0.1, 0.3 respectively.

Sobel operator is applied to the resultant image to emphasize more on the edges. The sobel operator significantly affects the detection of contours, as it intensifies the corresponding image gradient by the values in the near 3 x 3 region.

4.3. Morphological operations

The resultant image obtained from the Sobel operator is dilated, which helps to avoid fragmented region detection as it groups the nearby elements together. Then the image is inverted. The morphological operation of erosion is applied on the inverted image. The erosion operation is applied vertically and then horizontally, by taking a vertical and a horizontal kernel respectively. The size of the horizontal and vertical kernel is the same as the width and height of the image respectively. The anchor(center point) of the kernel is placed on each pixel in the image and the intensity value of each pixel is calculated by looking at the neighbouring pixel intensity values and selecting the maximum value. Erosion helps in forming the contours for all the components within the image.The

primary objective of applying morphological operations is to merge the nearby contours and thus generate larger regions, which are less fragmented.

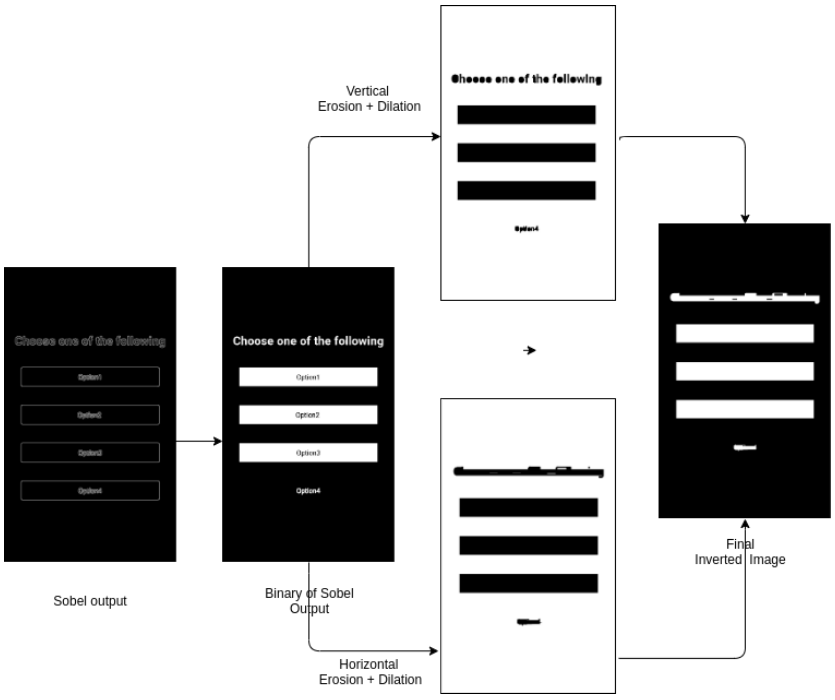

Figure 6. Erosion and Dilation

4.3. Contour detection and Component extraction

Contour detection is the final step of preprocessing, which eventually results in getting individual images of each component. The resultant image from previous preprocessing steps is used for contour detection. The detected contours are sorted and bound to get only the external boundaries of components. Using the coordinates of the components, the components are cropped from the original input image as shown in Figure. 7. The coordinates are stored in a file for later use in the code generation stage.

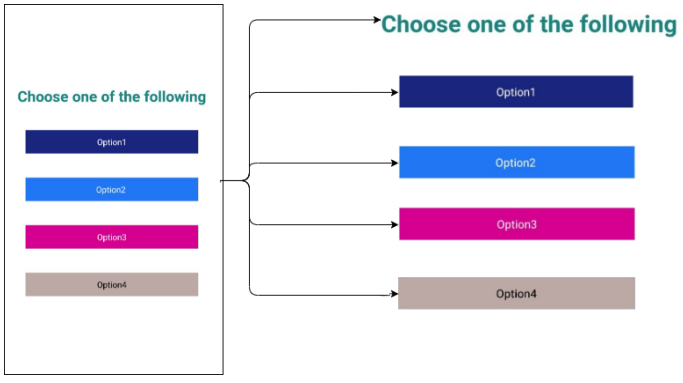

Figure 7. Getting individual components by using contour detection

5. Classification of Components

For classification, a combination of Convolutional Neural Network(CNN) and EAST text detection is used. The CNN is trained on images of 4 components i.e. text, textinput, button and image. Each cropped component from the input image is passed to CNN and EAST text detector for classification.

5.1. Convolutional Neural Network

The CNN is trained using 2000 images of the 4 above mentioned components. The CNN uses 5 layers of convolution and max-pooling and then fully connected layers. The best suited values for CNN were:

1. Learning Rate: 3e-3
2. Dropout Rate: 0.8
3. Five convolution and max-pooling layers and one fully connected layer
4. Number of nodes from first layer to last : 34, 64, 128, 64, 32 and fully connected layer with 1024 nodes

A confidence array for each component by CNN.. This array is then passed to EAST Text Detector.

5.2. EAST Text Detection

The EAST Text Detector is an OpenCV based Text detector which recognizes individual text words and draws boxes around each of them, as shown in Figure. 8.

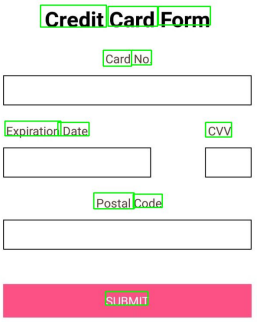

Figure 8. EAST Text Detection

In our proposed method, the EAST text detector is used to detect if the text is not present in a component. This gives practically good results in improving CNN's classification of text input components. A large positive confidence value is added to the text-input item of CNN's confidence array, if no text is detected in the component. A large negative value is added to the text-input item of CNN's confidence array, if the text is detected. The classified components with their co-ordinate data are then passed to the React Native code generator.

6. React Native code generation

The React Native Code Generator takes component type, its x-coordinate from the left, its y-coordinate from the top, its width and its height, for each component in the image.

This data is then used to generate template code. Every component has a template code which is placed on the screen exactly where it was in the GUI layout. This is done using the coordinates passed to the Generator.

While doing so, the absolute positioning of the components in terms of pixels is converted into relative positioning of the components in terms of the percentage of the screen size. This is beneficial in the case where the size of the screen varies. For example, when it is viewed in mobiles of different screen sizes or on a desktop. The size and positioning of the components varies according to the screen size.

7. Putting it all together in a service

To demonstrate our proposed solution, we created an end-to-end service, which executes the whole python script passing the image uploaded by the user and the code generated is saved in the local computer, the path of which is displayed.

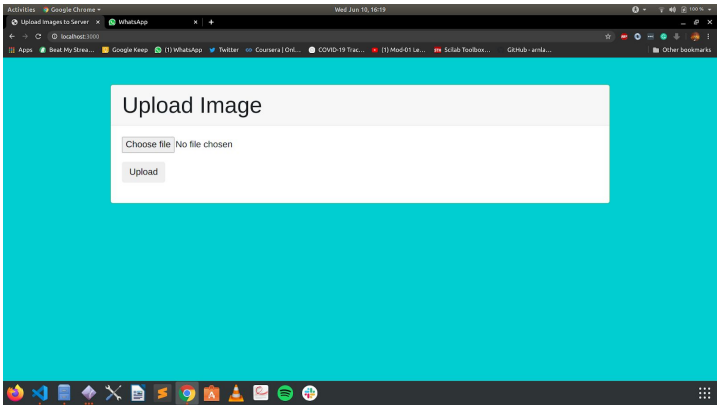

Figure 8. The service

The service is based on Node.js and Express.js, where the user can upload the GUI image of which the user wants React Native code. We used the multer package to upload images. We passed the image as an argument to util.promisfy, which executes the python script. The python script generates React Native code which is saved on the local computer in a file. The service displays the path to the location where the code is saved.

8. Results

We trained our CNN on 4 components with training accuracy of around 90%. It is difficult to measure testing accuracy of our model, as there is no automated way to compare the generated GUI with the input image.This is because while generating the template code, we have used random filler text in place of the real text and random colours for components like button, etc. Also, we need to check if the generated components are of correct size and are placed at proper position in the resultant GUI.

So to evaluate the results, instead of comparing the input image and the resultant GUI, we manually checked if the components are classified correctly or not. For this, 11 test images with multiple components were used. The following table shows the confusion matrix for the classification of components.

Table 1. Confusion Matrix

Input\Output	Text	Textinput	Image	Button	Accuracy
Text	24	-	2	-	24/26 = 92%
Textinput	-	7	-	12	7/19 = 36%
Image	-	-	6	-	6/6 = 100%
Button	2	1	-	11	11/14 = 78%

For the 11 test images, the total number of components was 65. Out of these 65 test components, 48 were classified correctly, which gave us an overall accuracy of 73%. Following are the major highlights of our classification model:

1. Text classification is excellent, this is due to the training dataset as well as good preprocessing that provides test components to CNN similar to the training dataset's text components.
2. Image classification by CNN is excellent(100%). All test images were processed and classified accurately.
3. Confusion between button and text is clear as both components have text in the middle.
4. Confusion between text-input and button is seen despite using EAST Text Detection.

The final generated GUIs were manually compared with the respective input images. It was found that they are visibly similar.

9. Conclusion

The importance of a good UI cannot be underestimated, it can mean the difference between success and failure of an application. Thus, the development of GUI is a crucial part of any application. The designing and developing team are involved in the iterative process of developing and upgrading the UI of the application based on client feedback. This task of designing and developing GUI is time consuming and tedious. Automation of converting the mockup images to code will make the former task easier and efficient.

In this paper, we proposed a novel approach to automate the conversion of mockup images to React Native code. Our work shows that computer vision techniques along with machine learning can be used to automate the above process.The results show that the proposed approach is practically feasible, while also highlighting that more work can be done to include corner cases. Our method proves that using synthetic dataset, real images can be classified by CNN and EAST text detection.The dataset can be easily scaled up for more components for practical use. Using the scaled up dataset and OCR to detect exact text in the user input image, will make this approach practically feasible and deployable.

Our work provides a solution to bridge the gap between developers and designers by automating the process of converting mock-up images to basic GUI code. This will let the designers experiment with the designs on their own, while the developers focus more on functionalities than appearance and alignments of the application.

References

*[1] Beltramelli, Tony. (**2018**). pix2code: Generating Code from a Graphical User Interface Screenshot. 1-6. 10.1145/3220134.3220135.*

*[2] Alex Robinson (**2019**), Sketch2code: Generating a website from a paper mockup.*

[3] K. P. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett and D. Poshyvanyk, "Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps," in IEEE.

*[4] Nguyen, Tuan & Csallner, Christoph. (**2015**). Reverse Engineering Mobile Application User Interfaces with REMAUI (T). 248-259. 10.1109/ASE.2015.32.*

*[5] Y. Liu, Q. Hu and K. Shu, "Improving pix2code based Bi-directional LSTM," **2018** IEEE International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), Shenyang, China, 2018, pp. 220-223. :https://ieeexplore.ieee.org/document/8720784.*

*[6] Davis, Richard & Saponas, T. & Shilman, Michael & Landay, James. (**2007**). SketchWizard: Wizard of Oz Prototyping of Pen-based User Interfaces. UIST:12 Proceedings of the Annual ACM Symposium on User Interface Software and Technology. 119-128. 10.1145/1294211.1294233.*

*[7] Ian J. Goodfellow and Jean Pouget-Abadie and Mehdi Mirza and Bing Xu (**2014**), Generative Adversarial Networks.*

*[8] Hochreiter, Sepp & Schmidhuber, Jürgen. (**1997**). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735. M. Young, The Technical Writer's Handbook, Mill Valley, CA: University Science, 1989.*

*[9] O'Shea, Keiron & Nash, Ryan. (**2015**). An Introduction to Convolutional Neural Networks. ArXiv e-prints.*

*[10] Shetty, Rakshith & Rohrbach, Marcus & Hendricks, Lisa & Fritz, Mario & Schiele, Bernt. (**2017**). Speaking the Same Language: Matching Machine to Human Captions by Adversarial Training.*

*[11] Dai, Bo & Fidler, Sanja & Urtasun, Raquel & Lin, Dahua. **(2017)**. Towards Diverse and Natural Image Descriptions via a Conditional GAN. 2989-2998. 10.1109/ICCV.2017.323.*

*[12] Xu, Kelvin & Ba, Jimmy & Kiros, Ryan & Cho, Kyunghyun & Courville, Aaron & Salakhutdinov, Ruslan & Zemel, Richard & Bengio, Y.. **(2015)**. Show, Attend and Tell:Neural Image Caption Generation with Visual Attention.*

*[13] Karpathy, Andrej & Li, Fei Fei. **(2015)**. Deep visual-semantic alignments for generating image descriptions. 3128-3137.*

*[14] Donahue, Jeff & Hendricks, Lisa & Guadarrama, Sergio & Rohrbach, Marcus & Venugopalan, Subhashini & Darrell, Trevor & Saenko, Kate. **(2015)**. Long-term recurrent convolutional networks for visual recognition and description. 2625-2634. 10.1109/CVPR.2015.7298878.*

*[15] Raman, Maini & Aggarwal, Himanshu. **(2009)**. Study and Comparison of Various Image Edge Detection Techniques. International Journal of Image Processing. 3.*

*[16] Bhardwaj, Saket & Mittal, Ajay. **(2012)**. A Survey on Various Edge Detector Techniques. Procedia Technology. 4. 220–226. 10.1016/j.protcy.2012.05.033.*

*[17] Canny, John. **(1986).** A Computational Approach To Edge Detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on. PAMI-8. 679 - 698. 10.1109/TPAMI.1986.4767851.*

*[18] Zhao, X & Wang, W & Wang, L. **(2011)**. Parameter optimal determination for canny edge detection. Imaging Science Journal, The. 59. 332-341. 10.1179/136821910X12867873897517.*

*[19] Zhou, X., Yao, C., Wen, H., et al. (**2017**) EAST: An Efficient and Accurate Scene Text Detector. In: Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). New York. pp.2642-2651.*

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

[21] H. Cho, M. Sung, and B. Jun. "Canny Text Detector: Fast and Robust Scene Text Localization Algorithm". In: 2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR). ***June 2016****, pp. 3566–3573. doi: 10.1109/CVPR.2016.388.*

[22] Sunil Kumar Katiyar and P. V. Arun. "Comparative analysis of common edge detection techniques in context of object extraction". In: CoRR abs/1405.6132 ***(2014)****. arXiv: 1405.6132. url: http://arxiv.org/abs/1405.6132.*

*[23] James A. Landay and Brad A. Myers. "Interactive Sketching for the Early Stages of User Interface Design". In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '****95.***

[24] T. Silva da Silva et al. "User-Centered Design and Agile Methods: A Systematic Review".In: ***2011*** *Agile Conference. Aug. 2011, pp. 77–86. doi: 10.1109/AGILE.2011.24.*

[25] J. A. Landay and B. A. Myers. "Sketching interfaces: toward more human interface design".In: Computer 34.3 ***(Mar. 2001)****, pp. 56–64. issn: 0018-9162. doi: 10.1109/2.910894.*