Automatic Flutter Generation using Sketch UI Mockups

Name: Prince Hodonou & Aakash Madabhushi
Class: CS 577 - Deep Learning
Professor: Gady Agam
Date: 5/4/21

Abstract

In modern software engineering, applications are getting more and more complex. Client demands to build and deploy apps due quickly are increasing tremendously. Yet with this increase of demand in software applications, the development process is stagnate. Development process is usually a two-fold problem: 1. Developing a sketch mockup 2. Implementing that sketch mockup along with its respective logic. However, implementation of the sketch mockup takes an unnecessary amount of time for developers to do correctly. As this problem is challenging and time consuming, we present Widget Detector which automates the process of converting sketch mockups to Flutter UI components.

Widget Detector employs deep learning to detect common user interface components from hand drawn sketches. It then generates the necessary Flutter widget for that specific component. The approach achieves an average widget component classification of 97% and an average bounding box regression of 60%. Widget Detector can be used to speed up user interface generation or even as a teaching tool for creating user interfaces..

Keywords:
Object Detection, Automatic Code Generation, Graphical user interfaces, Mobile applications, Machine Learning, Deep Learning

1 Introduction

User interfaces enable users to interact with the application and for them to operate its capabilities. As the development of UI involves starting with the creative process of drawing sketches, they are an effectively useful way to showcase your ideas with stakeholders in a quick and inexpensive way.

Object detection is a computer vision technique that allows us to identify and locate objects within the image. In this technique, this can be used to track and count objects in an image while labelling them using bounding boxes. This allows us at once to classify different types of objects found while also locating instances of them within the image. There are two main types of object detection: Single class object detection and Multiclass/multiple object detection with bounding box regression. The first type is simpler than the latter. In the first type, the goal is to classify an object that is the main focus of an image. On the other hand, the second task is to classify multiple objects in an image and also locate where they are in the image by drawing a bounding box around each object. Initially we chose to go with the latter. The goal was to draw a bounding box around all user interface components and classify them. Because this proved to be rather difficult in the time we had for the project, we chose a task that falls between both types of computer vision techniques. We thought it would be reasonable to create a program that accepts a hand drawn UI sketch, predict the most confident user interface component and draw a bounding box around it. Consequently, code will be generated of the most confident widget in that sketch.

2 Problem Statement

It is extremely time consuming for a developer to implement hand drawn sketches from scratch. This bottleneck can be partially automated with the use of deep learning by classifying and predicting user interface components and generating appropriate code for that component.

3 Proposed Solution

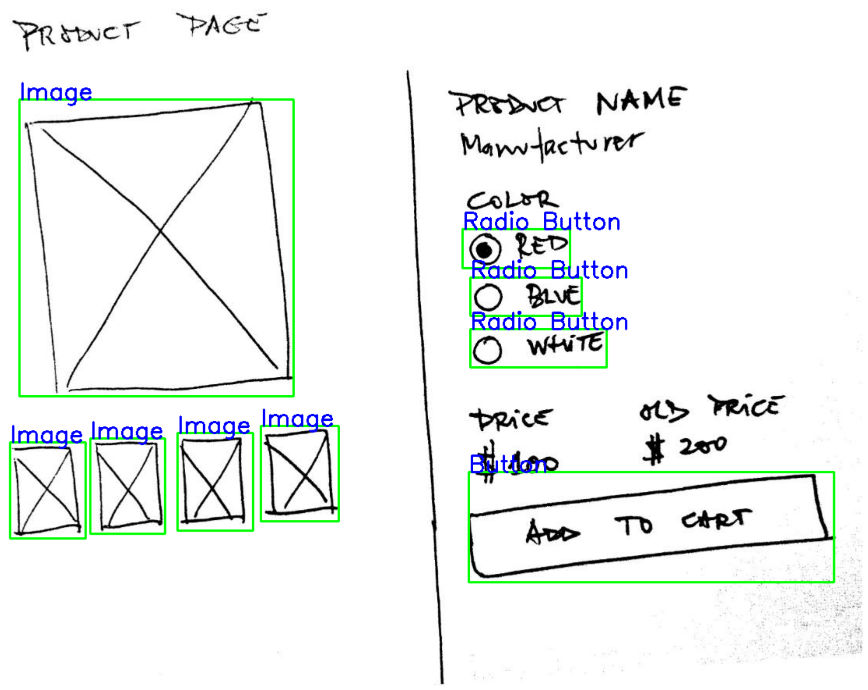
Implemented a two branch neural network, one for bounding box regression and one for widget label classification. We utilized transfer learning through the VGG16 pretrained network on ImageNet and added our two branch network on top of it.

4 Implementation Details

Datasets Used:

We used a dataset, “Sketch2code”, from kaggle (https://www.kaggle.com/biniamad/sketch2code) which comprises of total 900 training set and 224 testing set of images(.png) and annotations(.xml). The annotations of the image consists of the folder, filename, path, size, object name and labeled bounding box coordinates(xmin,ymin,xmax,ymax).

Data Pre-processing & Post-processing:



During this phase, we will start converting widget class labels using the LabelBinarizer which assigns a unique value to each label in a categorical feature and the bounding box will be assigned binary value (0,1) since it's easier for the model to be able to predict values in a fixed range for each image.

- Next, we split the csv file into the image_name, label, x1, y1, x2, y2, w and h.
- After which, we would need to normalize the bounding box coordinates (x1,y1,x2,y2) before storing them to the list
- Then we would call keras to load the images and convert each of them to array before storing them to the list.
- Using the VGG16 model preprocess_input() function, this changes the images to the format that is compatible with the model.
- The label binarizer is used to fit and transform the array of labels to binary labels.

Model Architecture:

The model architecture is as follows:

1. VGG16 trained on ImageNet
    a. Weights from VGG16 will not be updated
    b. Regression Branch for widget bounding
        i. Dense layer with 128 units with activation relu
        ii. Dense layer with 64 units with activation relu
        iii. Dense layer with 32 units with activation relu
        iv. Dense layer with 16 units with activation relu
        v. Dense layer with 8 units with activation relu

vi. Dense output layer with 4 units with activation sigmoid. Each neuron represents a bounding box point representing:
   1. x1 ~ starting x value
   2. y1 ~ starting y value
   3. x2 ~ ending x value
   4. y2 ~ ending y value

c. Classification Branch for widget classification

i. Dense layer with 512 units with activation relu

ii. Dropout of 0.5

iii. Dense layer with 512 units with activation relu

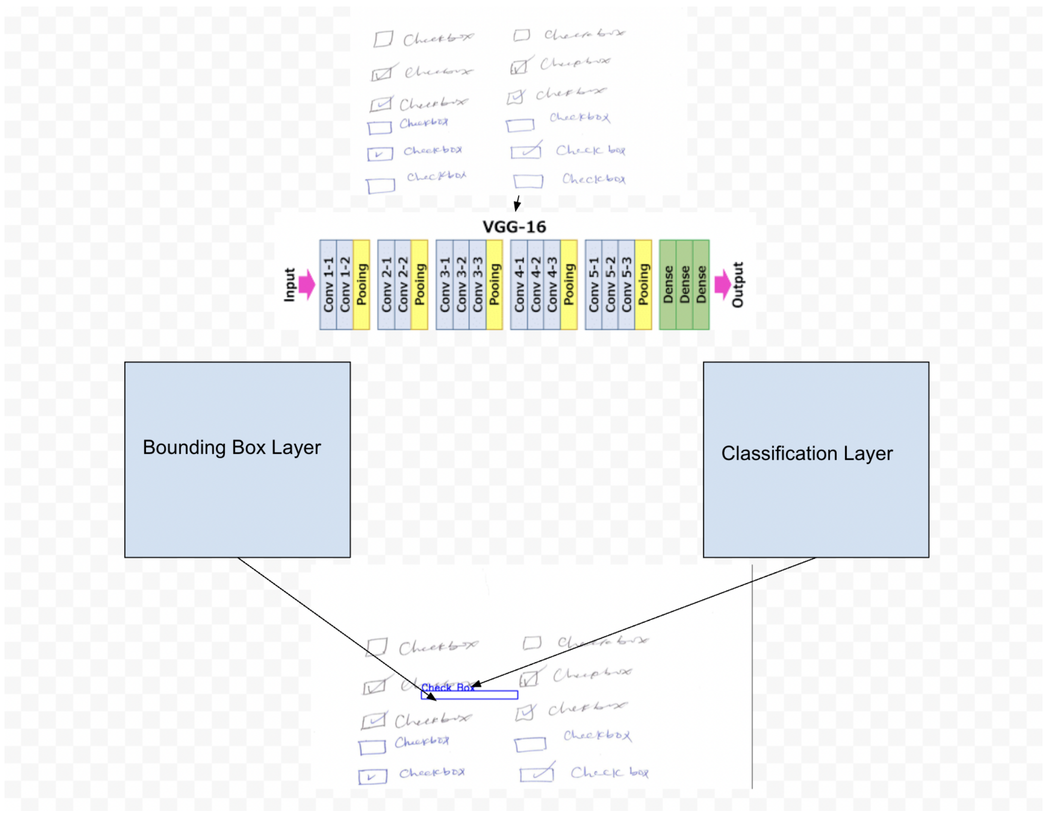
iv. Dropout of 0.5

v. Dense layer with 14 units. Each unit represents each classification label. The model chooses the label with the best confidence. And it outputs the label along with the regression box

d. Other hyperparameters

i. For the bounding box branch, we used the mean squared error loss function since it's a regression problem

ii. For the Widget label classification, we used categorical_crossentropy.



We used the VGG16 pre-trained convolutional neural network model since it is useful for many applications that have to do image classification. This network architecture comprises only 3x3 convolutional layers which are stacked on top of each other, leading to an increase in the depth as the number of layers increases. Max pooling layers are used in the model in order to reduce the volume size of the image and highlight the most important feature in the patch of the feature map. The VGG16 model is used as a transfer learning and we used the VGG single dimensional vector output feature maps to connect with the bounding box localization head and the classification head.

6 Results and Discussions

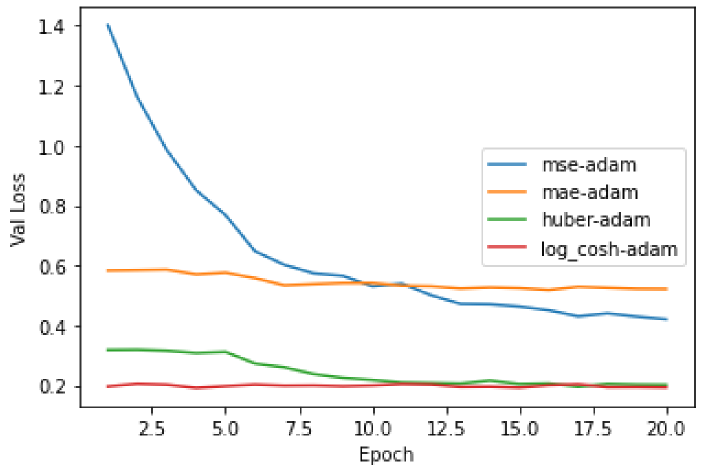


Figure 1

As figure 1 shows, we have tested our model on four different loss functions. Although mse-adam decreases faster, log_cosh-adam stays at a constant loss function. Huber-adam loss function is similar to log_cosh-adam in terms of output. Mae-adam is not efficient for this task as the loss function is constantly high. We used these loss functions as suggested by different papers on loss functions useful for object detection.

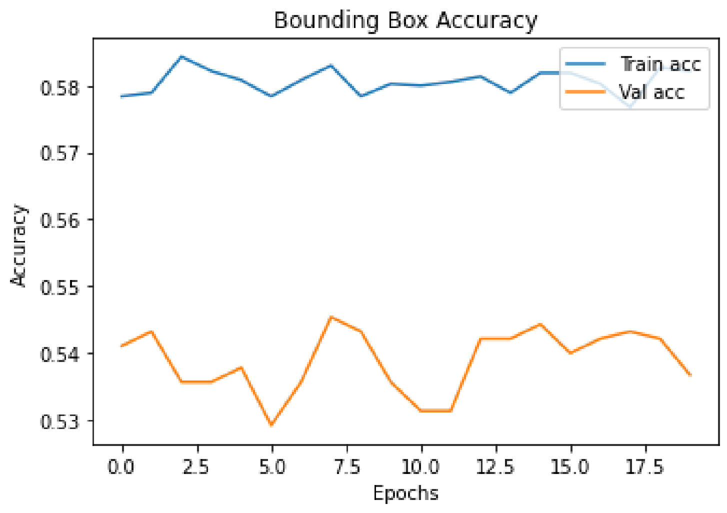


Figure 2

The accuracy for the bounding box fluctuates quite a bit. The reason for this may be because of our dataset pulled from Kaggle. Upon close examination of the data, we realized that some of the ground truth labels were wrong. For example, Checkboxes were labeled as Select components in the training set.

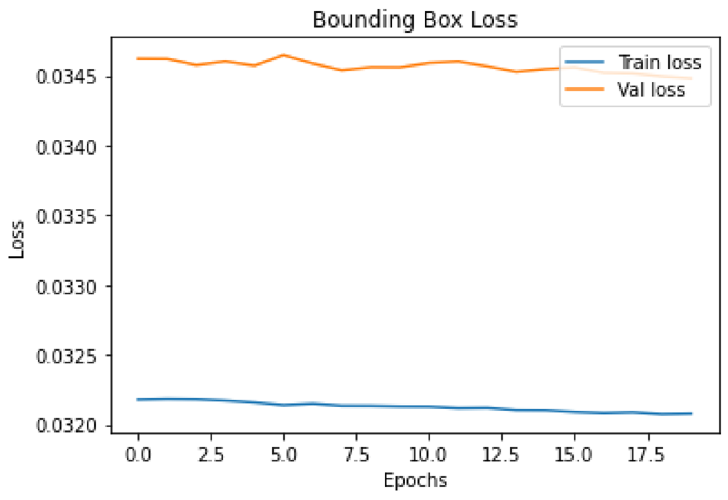


Figure 3

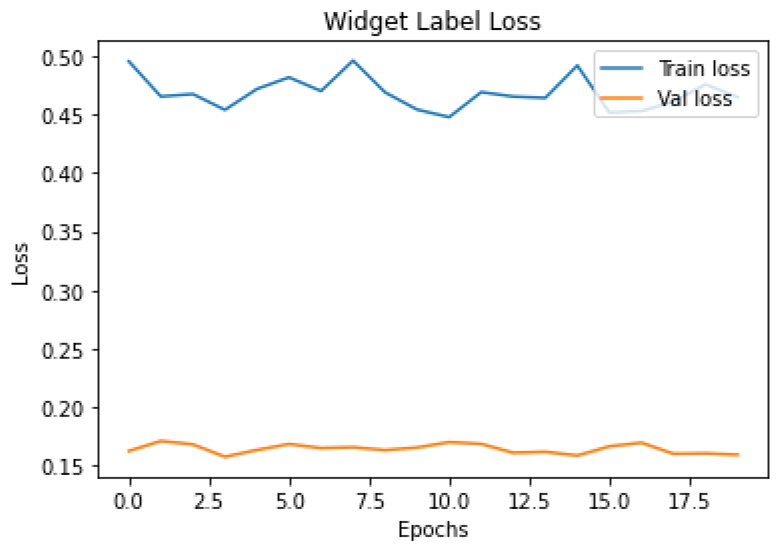


Figure 4

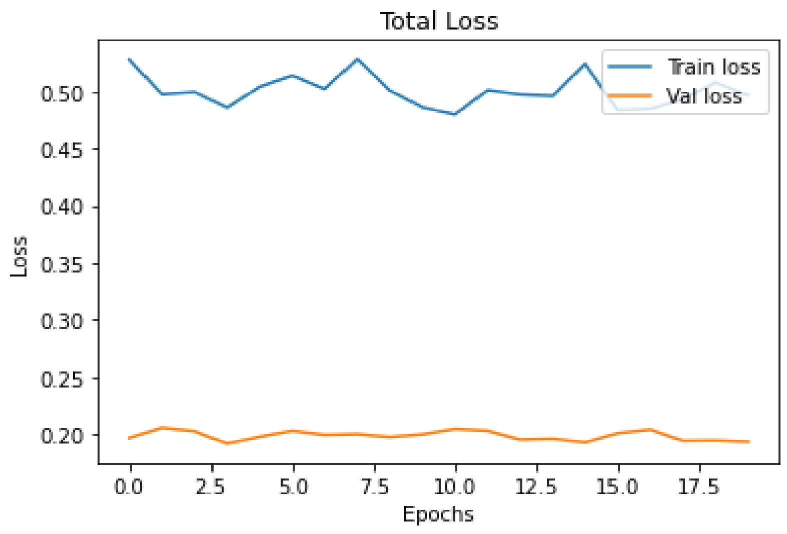


Figure 5

In both figure 3, 4, and 5 loss seemed to stay around the same area. However in Figure 6, loss decreased significantly. This is because we used different loss functions. The loss in 3,4,5 started relatively low and stayed relatively low. However, the loss in figure 6 started relatively high and decreased dramatically.

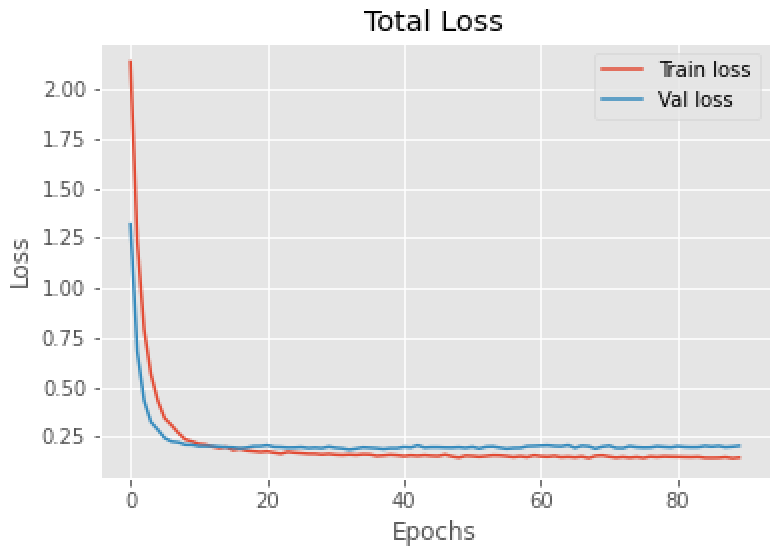


Figure 6

7 Conclusion

We have implemented automation of translating hand draw sketches to code with the use of deep learning by classifying and predicting user interface components and generating appropriate code for that component. If implemented fully, this can be used to greatly speed up the app development process. It can also be used as a teaching tool. To accomplish our task, we designed a two branch neural network, one for bounding box regression and one for widget label classification. We utilized transfer learning through the VGG16 pretrained network on ImageNet and added our two branch network on top of it. Our architecture proved to give promising results for the future with an average widget component classification of 97% and an average bounding box regression of 60%.

8 References

Automatic code generation from sketches of mobile applications in end-user development using Deep Learning - Daniel Baulé1 , Christiane Gresse von Wangenheim1 , Aldo von Wangenheim1 , Jean C. R. Hauck1 , Edson C. Vargas Júnior2 https://arxiv.org/pdf/2103.05704v1.pdf

pix2code: Generating Code from a Graphical User Interface Screenshot - Tony Beltramelli UIzard Technologies Copenhagen, Denmark tony@uizard.io
https://arxiv.org/pdf/1705.07962.pdf

sketch2code: Generating a website from a paper mockup - Alexander Robinson University of Bristol ar15247@bristol.ac.uk

YOLO v3 Object Detection with Keras -
https://towardsdatascience.com/yolo-v3-object-detection-with-keras-461d2cfccef6

You Only Look Once: Unified, Real-Time Object Detection -
https://arxiv.org/pdf/1506.02640.pdf

Bounding Box -
https://d2l.ai/chapter_computer-vision/bounding-box.html

Going Deeper with Convolutions -
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf

Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network -
https://arxiv.org/pdf/1910.08930.pdf

Sketch2code using Visual Attention & LSTM Decoder -
https://vincentk1991.github.io/sketch2code/

Sketch2code dataset -
https://www.kaggle.com/biniamad/sketch2code

Dataset generator for web sketch-to-code deep learning problems -

https://medium.com/@devtarek/dataset-generator-for-sketch-to-code-deep-learning-problems-be9317b7e9b2

How to Train an Object Detection Model with Keras -
https://machinelearningmastery.com/how-to-train-an-object-detection-model-with-keras/

Build your First Multi-Label Image Classification Model in Python -
https://www.analyticsvidhya.com/blog/2019/04/build-first-multi-label-image-classification-model-python/

https://www.youtube.com/channel/UCkzW5JSFwvKRjXABI-UTAkQ

Extract annotations from CVAT XML file into mask files in Python -
https://towardsdatascience.com/extract-annotations-from-cvat-xml-file-into-mask-files-in-python-bb69749c4dc9

ImageNet: VGGNet, ResNet, Inception, and Xception with Keras -
https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

Classification with Localization: Convert any Keras Classifier to a Detector -
https://learnopencv.com/classification-with-localization/