Acknowledgment

First of all, we praise Allah for everything; for empowering us to go through all the hard times and for giving us patience and strength to tackle every and each problem we faced till we reached our goal and finished Tagit.

We would like to thank our parents for their supportive attitude towards us and encouragement. Without their patience and support we could never achieve any goal in our lives.

Also, we would like to express our sincere gratitude to our supervisor Dr. Khaled Soradi for his patience, motivation, and immense knowledge in both technical and personal skills. His guidance helped us during all the project phases; research, design and development. We could not have imagined having a better mentor for our project.

We would also like to express our special appreciation to all the TAs, who have helped us over the past years and helped us a lot during refining our graduation project ideas and even while working on the graduation project especially, TA.Yahia Zakaria for his guiding notes.

Special thanks to our friends and colleagues for their kind advice. Finally, we want to thank everybody who helped in a way or another in letting this project come to life.

Abstract

During the past few years, revolutionary changes occurred in everyday life as the mobile devices became essential. Then smartphones, which are capable of not only receiving and placing phone calls, but storing data, taking pictures as well. Operating systems on the smartphones were a groundbreaking addition as they allowed having applications that serve and help humans on a daily basis.

The more people possessing smartphones, the more the mobile application development expands. Hence the need for developers to fill such expansions in. The time and effort taken by a frontend developer to convert a given design into the corresponding code need to be saved. Also more focus needs to be put towards working on the applications' functionalities and logic.

Deep learning has helped enhancing the quality of human lives, serving purposes of daily needs. A field with such evolution would be beneficial in serving developers themselves. Repetitive and tedious tasks can be done by machines rather than developer. Given that Android is the most common operating system for smartphones, the focus was drawn to Android developers.

In this work, an approach is presented to help the developer taking any screenshot for android design, hand-drawn image or PSD file. The corresponding XML files and Java code is generated with basic functionality which are ready to run on Android Studio. The hierarchy of the XML is generated in a weighted layout form which makes the application compatible with any screen size. The colors of the background and the text appearing in the design are handled to end with a layout which is as near as possible to the given design.

Contacts

Supervisor

Dr.Khaled Soradi	khsoradi@eng.cu.edu.eg	01277377726

Team Members

Abd El-Rahman Sobhy	aabdosobhy@gmail.com	01064382458
Fatema Khalid	fatemakhalid96@gmail.com	01144108102
Feryal Hisham	feryalhisham19@gmail.com	01120819748
Heba Ahmed	heba.ahmed.ali2019@gmail.com	01270908334

List of figures

Figure 1. Salaries for Android developers..13
Figure 2. The result of the survey conducted showing the percentage of Android developers...14
Figure 3. A pie chart showing the amount of consumed by developers to write XML...............14
Figure 4. A pie chart showing the number of people who are in favor of using our tool............15
Figure 5. A Pie chart showing the percentage of people in favor of using the sketch mode.......16
Figure 6. Our time plan 18
Figure 7. An abstract representation of a NN...19
Figure 8. An abstract architecture of a NN with two hidden layers 20
Figure 9. The performance of Deep Learning with respect to   Big Data 20
Figure 10. An image classification CCN architecture..21
Figure 11. Example of a Filter Applied to a Two-Dimensional Input to create a feature Map...23
Figure 12. The ReLU activation function. 24
Figure 13. A figure demonstrating the effect of applying max pool 25
Figure 14. The categorical cross entropy equations 27
Figure 15. Gradient descent algorithm 27
Figure 16. SGD without momentum 28
Figure 17. SGD with momentum 28
Figure 18. HOG algorithm 31
Figure 19. An illustration for LBP encoding..32
Figure 20. CIE color space..33
Figure 21. Different Delta E ranges and their indications..33
Figure 22. Starting new activities using Intent objects...35
Figure 23. Android ImageButton 35
Figure 24. Android CheckBox 35
Figure 25. Android Button 35
Figure 26. Android EditText 35
Figure 27. Android Switch...36
Figure 28. Android RadioGroup 36
Figure 29. An example of a ListView..37
Figure 30. An example of an ActionBar 37
Figure 31. An example of DSL 39
Figure 32. An abstract representation of RNN...40
Figure 33. UI2code architecture..41
Figure 34. pix2code architecture..42
Figure 35. The Block Diagram of our system 44
Figure 36. Sequence diagram of the user interactions with the system.......................................45
Figure 37. An example of overlapping boxes 57
Figure 38. The component correctly classified as Switch..57
Figure 39. A sample of an input design used for testing..57
Figure 40. The results of the boxes drawn around the contours...57
Figure 41. A sample of the components classified correctly..58
Figure 42. Another sample of a design and the extracted boxes 58
Figure 43. An example of the TextViews boxes that needs to be merged 58

Figure 44. A sample of a design with a ListView 59
Figure 45. Android main directory 60
Figure 46. A ListView rendered in Android Studo 61
Figure 47. The app with the radio buttons rendered on Android Studio 62
Figure 48. A sample of the Java code showing the activities switching 63
Figure 49. Screenshots of the applications run on our model, ... 64
Figure 50. A sample of a hand-drawn design containing all the components we could predict . 65
Figure 51. A screenshot taken from our mobile after running 65
Figure 52. A sample of a hand-drawn design containing a ListView and ActionBar 65
Figure 53. A screenshot taken from the app. 65
Figure 54. The welcome window 71
Figure 55. Create a new project 72
Figure 56. Main screen 74
Figure 57 Add screenshots, hand-drawn or psd 75
Figure 58. Main screen after adding images 76
Figure 59. Edit images details 77
Figure 60. Please wait screen 78
Figure 61. After Generating XML screen 79
Figure 62. Update Components step 80
Figure 63. Connect Component window 81
Figure 64 Finish Window 82

List of tables

Table 1. Model results with SGD optimizer....................55
Table 2. Model results with Adam optimizer....................55
Table 3. Model results when adding only four features compared to when adding the total 18 features....................56

Table of Contents

Acknowledgment 0
Abstract 1
Contacts 2
  Supervisor 2
  Team Members 2
List of figures 3
List of tables 5
Table of Contents 6
Table of Acronyms 9
1. Introduction 10
  1.1. Problem definition 10
  1.2. Project Idea 10
  1.3. Motivation and Justification 10
    1.3.1. Motivation 10
    1.3.2. Educational value 11
      1.3.2.1. Technical 11
      1.3.2.2. Non-Technical 11
  1.4. Domain of Application 11
  1.5. Summary of Approach 11
  1.6. Document overview 12
2. Market Survey 13
  2.1. Intended Customers 13
  2.2. Current Market 13
  2.3. Market Survey Statistics 13
  2.4. Competition 16
  2.5. Feasibility Study 17
    2.5.1. Description of Products & Services 17
    2.5.2. Technological Feasibility 17
      2.5.2.1. Software Requirements 17
      2.5.2.2. Hardware Requirements 17
    2.5.3. Economic Feasibility 17

2.5.4. Schedule Feasibility 18
2.5.5. Availability 18
2.5.5.1. Marketing Strategy 18
2.5.6. Legal Feasibility 18
3. Necessary Background 19
3.1. Technical Details related to the project 19
3.1.1. Artificial Neural Network 19
3.1.2. Deep Learning 20
3.1.3. Convolutional Neural Networks (Conv Nets) 21
3.1.4. The Neural Network training process 26
3.1.5. Image Processing Techniques 29
3.1.6. Android Development Concepts 34
3.2. Literature Review 38
4. Design Specifications 43
4.1. System overview 43
4.2. Block Diagram 44
4.2.1. User Interface 45
4.2.2. Handcrafted Feature Extraction 45
4.2.3. Image Classification Training 47
4.2.4. Component Extraction 49
4.2.5. Components Prediction 50
4.2.6. Hand-drawing Components Recognition 50
4.2.7. Hierarchy and Code Generation 51
4.3. Enhancements 53
5. System Evaluation 54
5.1. Hardware and Data Used 54
5.2. Results and Discussion 54
6. Tools Used 66
6.1. Overview 66
6.2. System Tools and Libraries 66
6.3. Programming Languages 69
6.4. Hardware tools 69
7. Conclusion and Future work 70

7.1. Conclusion 70
7.2. Future Work 70
References 71
Appendices 71
Appendix A: User Manual 71
Appendix B: Additional Trials 82
Arabic Summary 83

Table of Acronyms

Acronym	Word
ANN	Artificial Neural Network
CNN / ConvNet	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTMs	LongShort-Term Memory
MaxPool	Maximum pooling layer
Conv Layer	Convolutional Layer
ReLU	Rectified Linear Unit layer
FC Layer	Fully-Connected Layer
SGD	Stochastic Gradient Descent
Adam	Adaptive Moment Estimation
OCR	Optical Character Recognition
HOG	Histogram of Oriented Gradients
LBP	Local Binary Patterns
UI	User Interface
UX	User Expreience
GUI	Graphical User Interface
XML	Extensible Markup Language
PSD (psd)	Photoshop Data File
IDE	Integrated Development Environment
App	Application
CLI	Command Line Interface

1. Introduction

This chapter briefly presents our project idea, our motivation to proceed through turning the idea into a real project and values we gained.

1.1. Problem definition

Front-end developers spend a lot of time and effort to convert a GUI application design image (design mockup) into XML. The UI designer uses a design software like Photoshop. The output of such program is an image or a psd file. The Android front-end developer needs to recognize the components (also called widgets) of the image. The widgets need to be laid out with suitable weights to give to be rendered as an Android application similar to the design. In addition, the developer also needs to decide the attributes of each widget such as color, text, width, height, etc. This part of their job is tedious.

In addition, at the very first levels of making an Android app, the designer usually uses pen and paper to sketch the design. This sketch allows the customer and managers to have an initial insight of what the design should look like.

Unlike implementing the interface components functionality, converting a GUI design to XML does not require problem solving skills. Also, most of the time is dedicated to make the interface interactive, i.e. handling user clicks and input, handling bugs and integrating with back-end developers.

Converting a GUI design screenshot or a sketch to an Android rendered XML is a repetitive and tedious task. The existing tools are not fully automated and requires the developer interaction. It is only recently that the machine learning community has started to provide alternative solutions to aid the front-end developers in this task. For now, there are few projects that tackle this challenge, yet the results are not satisfying. Of course, no solution can completely replace the front-end developer.

1.2. Project Idea

Tagit, is an application that generates XML from a GUI design. It is an automated and integrated solution that takes as an input a screenshot, psd or hand drawn sketch. With Tagit, the user can get a complete Android project where the XML files are connected to their corresponding Java classes ready for implementing the logic.

1.3. Motivation and Justification

1.3.1. Motivation

The rapid evolution in Computer Vision and Deep Learning technologies grabbed our interest to contribute with our project. Converting GUI to code is an essential step in any application development process. A task which consumes time that could be saved. Automating this task using Computer Vision and Deep Learning is a novel research.

Front-end developers need to deal with complex designs, where the components hierarchy is nested, which is one of the main challenges. Being developers ourselves made us aware of how helpful it would be to have an automated solution.

Moreover, there are several ways to write XML and it is expected to appear the same way when rendered on different Android mobile screens. Taking into consideration different screen sizes is also another challenge.

1.3.2. Educational value

Working on Tagit developed our technical and non-technical skills.

1.3.2.1. Technical

On the technical level we learned more about:

- Deep learning and Convolutional Neural Networks.
- Image processing.
- Mobile development.
- Machine learning tools and platforms.
- Python development.
- Extracting information from scientific papers and following technology updates.

1.3.2.2. Non-Technical

- Time management and working under pressure.
- Presentation skills.
- Technical writing.
- Working within a team on a large scale project.
- Using a version control system such as github.
- Work division among members and integration.

1.4. Domain of Application

Tagit targets a wide market with increasing demand. Android has the biggest market share by being sold 87.8% of total business. As a Google service we can be sure Android will have improvements to continue satisfying the demands of modern communications.1 The targeted market is the Android development market. Tagit helps android developers to reduce the time consumed in front-end development. The application accelerates the process of converting the mobile application GUI design as a screenshot or a sketch to an Android XML. Hence, the developer's productivity elevates reflecting on the profit.

1.5. Summary of Approach

Since Tagit is an integrated solution that can be used for converting a screenshot, psd, hand drawn sketch, there are several approaches used including computer vision, image processing, text extraction and mobile development.

The project consists of two main parts:

[1] Gartner Says Chinese Smartphone Vendors Were Only Vendors in the Global Top Five to Increase Sales in the Third Quarter of 2016. (n.d.). Retrieved from https://www.gartner.com/en/newsroom/press-releases/2016-11-17-gartner-says-chinese-smartphone-vendors-were-only-vendors-in-the-global-top-five-to-increase-sales-in-the-third-quarter-of-2016

- Components Classification: uses CNN to classify an unlabeled widget into one of the Android widgets.

- Code Generation: uses information extracted from the image and the user options chosen from Tagit interface. This information together with the components classification part is used to generate widgets attributes and the needed Java code. After this step, the output files are ready to be rendered and run as an Android application. The Java code does not implement any logic; it only complements the XML so that it renders properly on an Android mobile.

1.6. Document overview

In this document we will talk about Tagit in detail. This document is organized as follows:

- Chapter 1: Introduction to the project, and the problem it tackles.
- Chapter 2: The intended users, market survey and competitive analysis.
- Chapter 3: Necessary background, technology approaches used in the project, and the related research paper.
- Chapter 4: The system architecture, its building blocks in detail and the interaction between them precisely.
- Chapter 5: Our software experimental results and methodology to evaluate the performance and ensure quality.
- Chapter 6: Tools we used to develop the project
- Chapter 7: The conclusion, and future work.
- References: Technical references and papers used in this project.
- Appendix A: User Manual
- Appendix B: Additional approaches we used but the results were not satisfying.

2. Market Survey

This chapter briefly discusses the market, customers, competitors and marketing plan.

2.1. Intended Customers

Tagit is targeting an expansive market with increasing demand. It has great use in the continuously-growing field of Android development. Throughout the process of creating an Android App, developers are hired to work from the concept and design phases to the software architecture planning, coding and testing phases.

The application's Intended Customer is the Android Developer. For serving the costumer's demand, the application decreases the time required for the work in turning the concept and design into a software architecture. This will save the user more time for developing applications' logic and functionalities for the intended application.

2.2. Current Market

In the year 2016, users of Google Play spent an estimated $24.8 billion spent on Android Apps. Google Play apps installs grew by 6.4% translated into 29.4 billion.

An Android Developer earns an average of $97,645, ranging from $79,092 to earning more than $131,952. Compensation is derived from 8K profiles, including base salary, equity and bonus.

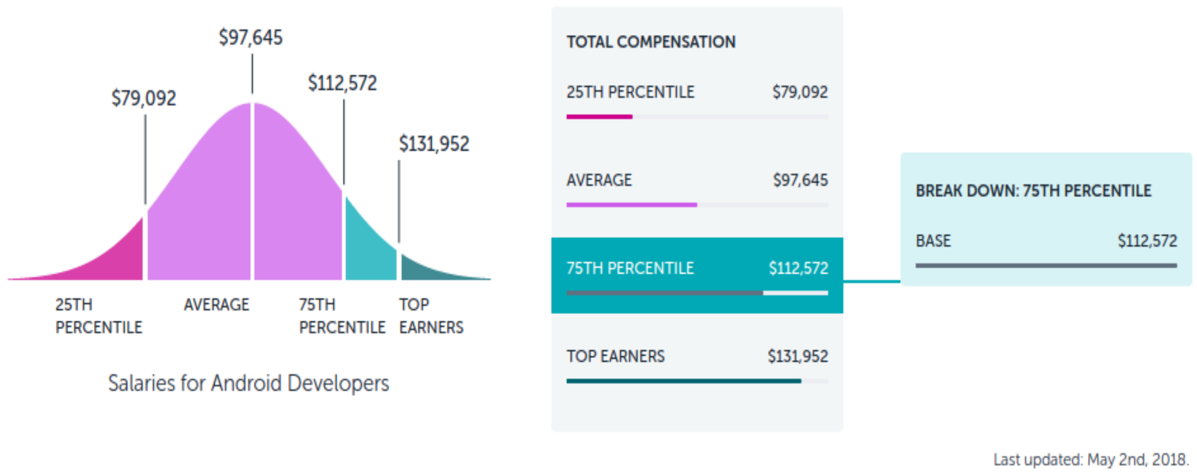


Figure 1. Salaries for Android developers

2.3. Market Survey Statistics

We held an online market survey on people from different backgrounds and the results of it came up as following:

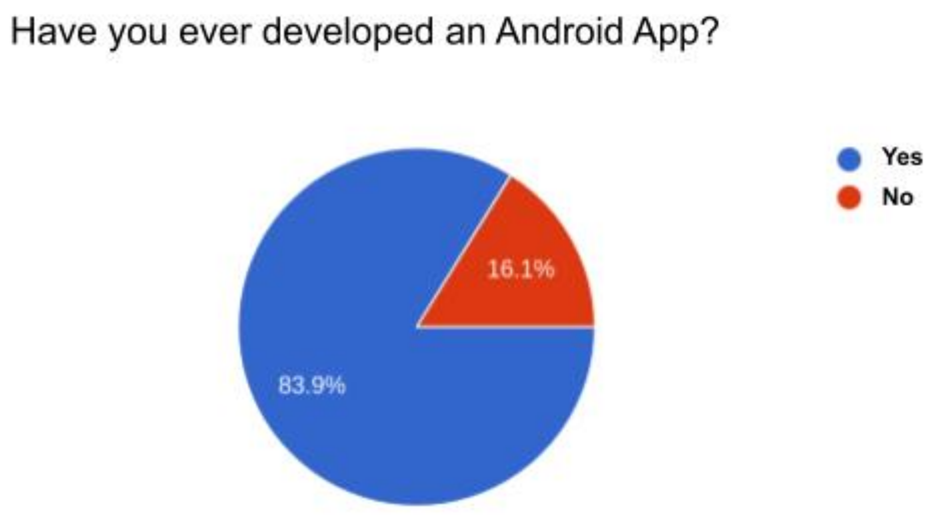


Figure 2. The result of the survey conducted showing the percentage of Android developers

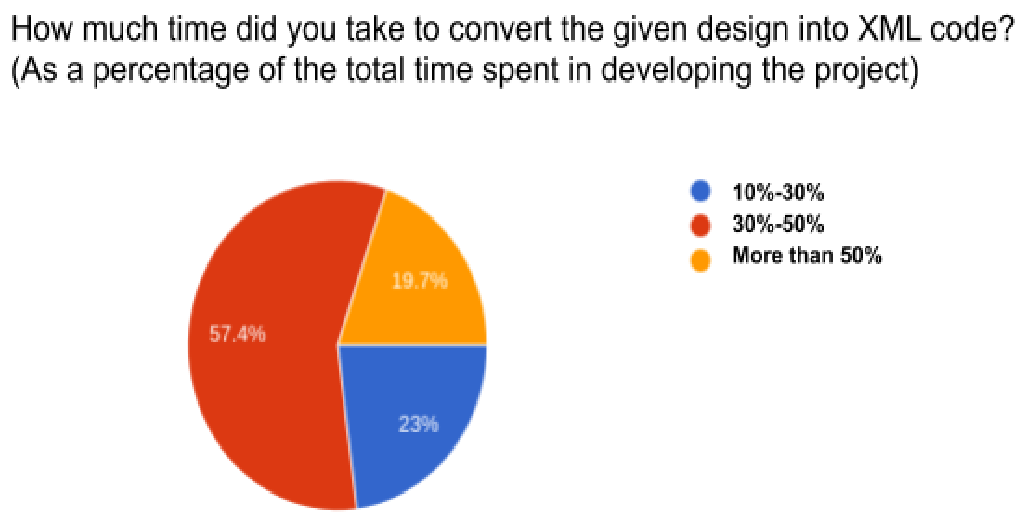


Figure 3. A pie chart showing the amount of consumed by developers to write XML

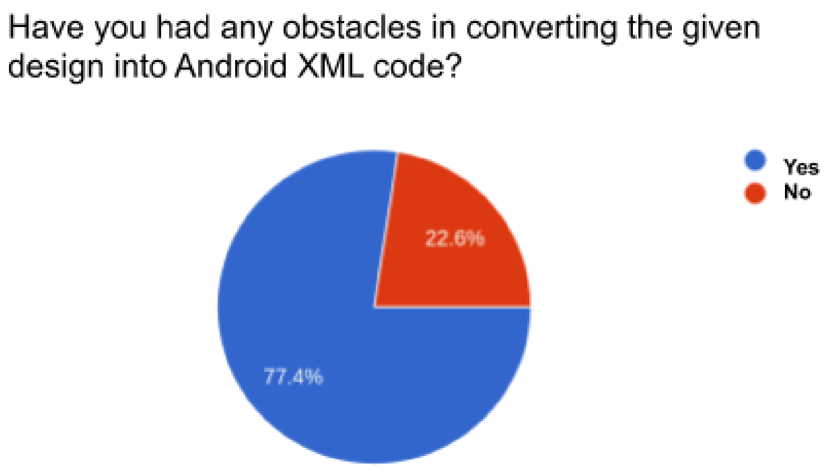


Figure 4. A pie chart showing the percentage of people who faced difficulties writing XML

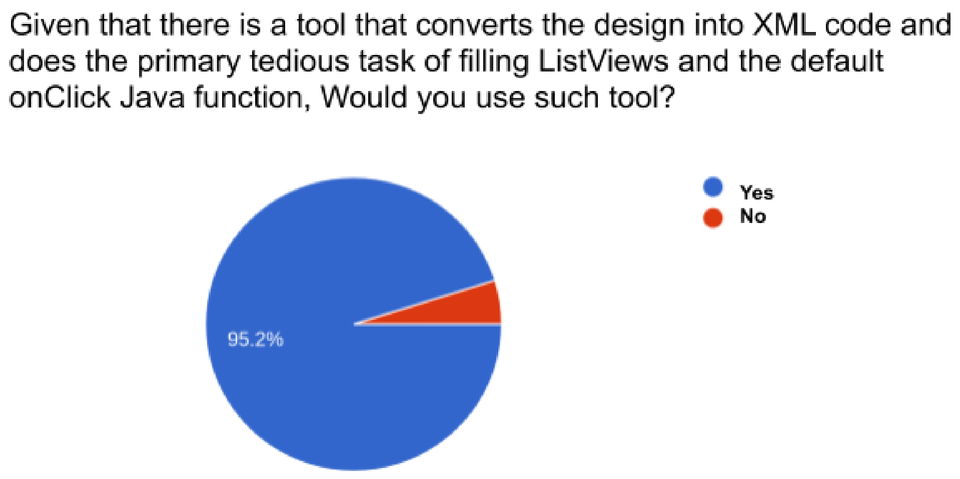


Figure 4. A pie chart showing the number of people who are in favor of using our tool

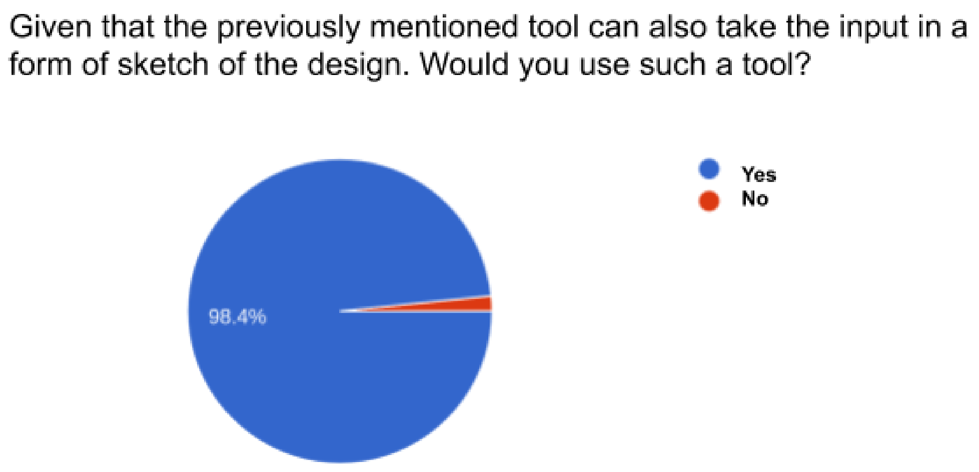


Figure 5. A Pie chart showing the percentage of people in favor of using the sketch mode

2.4. Competition

As we researched through the available applications and projects that convert a GUI design of an application into code, we found the following.

2.4.1. Pix2code

Pix2code is a deep learning model capable of turning sketches of graphical user interfaces into actual code.

2.4.1.1. Type:

- CLI

2.4.1.2. Features:

- Output can be in IOS, Android or HTML.

2.4.1.3. Limitations:

- Pix2code is only a research project.
- Pix2code won't work for specific use cases.
- Doesn’t work on neither PSD nor hand-drawn sketches

2.4.2. Sketch2Code

Sketch2Code is a Microsoft product that has a trained model to recognize hand-drawn web design elements like a textbox or button.

2.4.2.1. Type:

- Web App

2.4.2.2. Features:

- Understands the handwritten text and the structure

2.4.2.3. Limitations:

- Application can’t recognize neither text nor components colors.
- Doesn’t work on neither screenshots made by designers nor PSD files.

2.4.3. UI2code

UI2code is a tool to convert the UI design image into the skeleton code with deep learning methods. It is a Neural Machine Translator to Bootstrap Mobile GUI

2.4.3.1. Type:

- CLI

2.4.3.2. Features:

- Obtains a GUI skeleton from a UI design image.
- Well-Documented code and research paper.

2.4.3.3. Limitations:

- User needs to fill in some detailed attributes such as color, text.
- Doesn’t support designs as PSD files nor hand-drawn wireframe as user input.

2.5. Feasibility Study

This section briefly describes our technical and financial feasibility studies.

2.5.1. Description of Products & Services

The product provides the user with the service of transforming a graphical user interface created by a designer in the form of PSD file, screenshot or hand-drawn sketch. The output form is a computer code in a form of Android XML accompanied with its corresponding Java code required. The output can be built and run on a mobile device.

2.5.2. Technological Feasibility

2.5.2.1. Software Requirements

- Operating system: Ubuntu 16.04 LTS, Ubuntu 18.04.2 LTS, Windows 10 with Python 3.6
- NVidia CUDA, cudNN
- Tensorflow deep learning library
- Keras deep learning library
- OpenCV-python library
- Numpy and, Scipy.io libraries
- Pytesseract library
- Colormath library
- PyQt5 and,Qdarkstyle libraries
- Screeninfo and, Imagesize libraries
- Psd-tools3 library
- Google-cloud-vision API
- Skimage and, scikit-image libraries

2.5.2.2. Hardware Requirements

- RAM: 12.6 GB
- GPU: 1xTesla K80, having 2496 CUDA cores, compute 3.7, 12GB GDDR5 VRAM.
- CPU: 1xsingle core hyper threaded i.e. (1 core, 2 threads) Xeon Processors @2.3Ghz (No Turbo Boost), 45MB Cache.

2.5.3. Economic Feasibility

2.5.3.1. Cost

- Android Studio/ SDK Native applications: Free.
- Google Cloud Vision API: Offers up to 1000requests/month for Free.
- Google Colaboratory machines satisfying the Hardware Requirements: Offers 12-hours sessions for Free.
- Tensorflow / Keras Training Deep Learning models : Free.
- All libraries required in Software Requirements: Free

2.5.3.2. Benefit

- It bridges the gap between UI/UX designers and front-end developers.
- It saves the Android developer's time to develop more functionalities for an App rather than building GUI structure.
- It widens deep learning technology usage in Egyptian market.

2.5.4. Schedule Feasibility

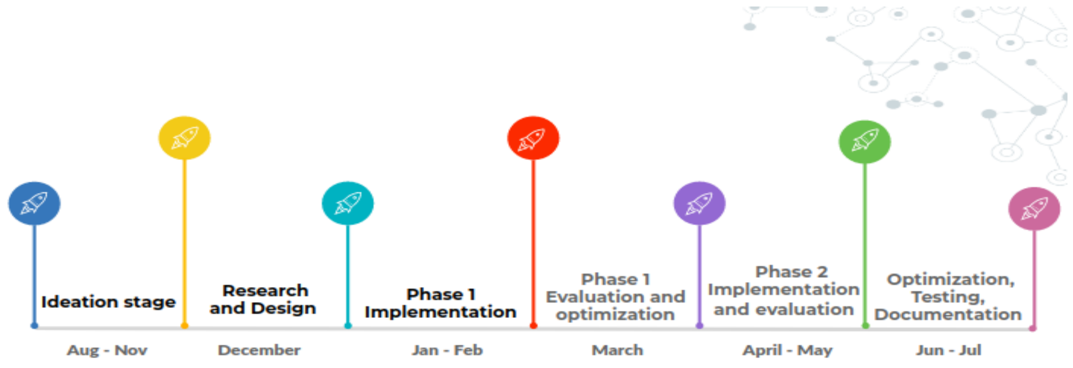

Ideation stage
Aug - Nov
Research and Design
December
Phase 1 Implementation
Jan - Feb
Phase 1 Evaluation and optimization
March
Phase 2 Implementation and evaluation
April - May
Optimization, Testing, Documentation
Jun - Jul

Figure 6. Our time plan

2.5.5. Availability

2.5.5.1. Marketing Strategy

Tagit is available for everyone to use for free as an open-source project. It will always be available for free.

2.5.6. Legal Feasibility

Our project doesn't conflict with any intellectual property. Our project uses open source tools and implements concepts from papers published in international scientific conferences or journals.

3. Necessary Background

This chapter introduces the technical concepts that are necessary for designing and implementing our project and a summary of related research papers.

3.1. Technical Details related to the project

3.1.1. Artificial Neural Network

An ANN is a computational model inspired by networks of biological neurons, wherein the neurons compute output values from inputs. It learns from its past experience and errors in a non-linear parallel processing manner. The neuron is the basic calculating entities which computes from a number of inputs and deliver one output comparing with threshold value and turned on (fired). The computational processing is done by internal structural arrangement consists of hidden layers which utilizes the back propagation and feed forward algorithms to deliver output close to accuracy.

The back propagation and feed forward algorithms are the main steps of the learning process, scientifically known as the training process. During the training process the model is exposed to new, unfamiliar data step by step. At each step, the model makes predictions using the initial weights it assumed, feed forward, and gets feedback about how accurate its generated predictions were. This feedback is used to correct the errors made in prediction. The feedback is provided in terms of an error according to some measure, for example, distance from the correct solution. The corrections that are made due to the feedback are adjustments to some parameters or most commonly known as weights. So, basically the learning process is an iterative predict-and-adjust process. This process of tuning the weights is called backpropagation algorithm.

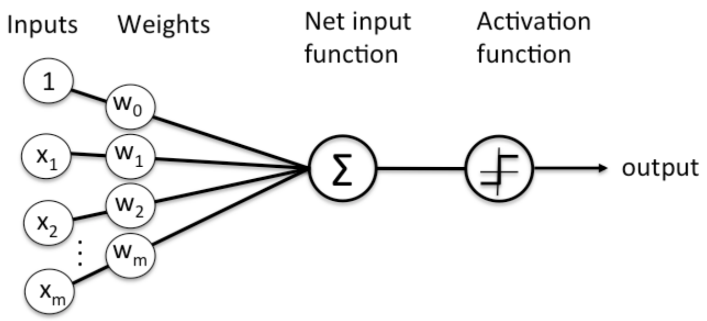


Figure 7. An abstract representation of a NN

3.1.2. Deep Learning

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data inspired by the structure and function of the human brain called artificial neural networks. A neural network having more than one hidden layer is generally referred to as a Deep Neural Network. The massive data available made it possible for deep learning to evolve at a faster pace.[2]

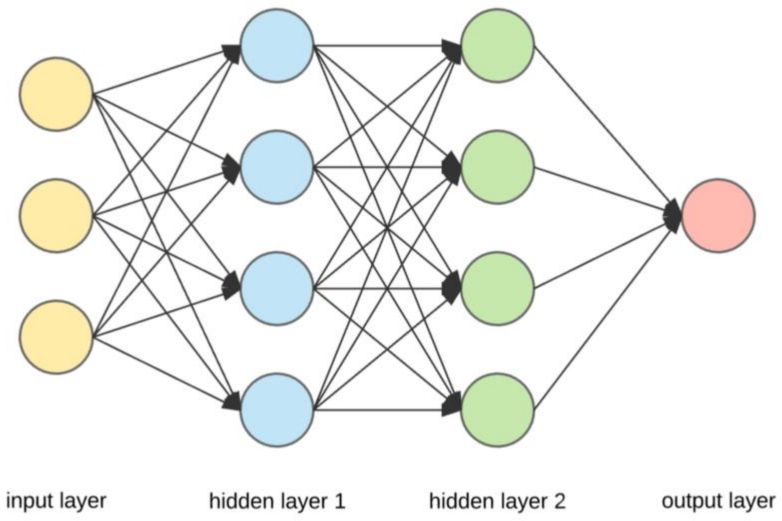


Figure 8. An abstract architecture of a NN with two hidden layers

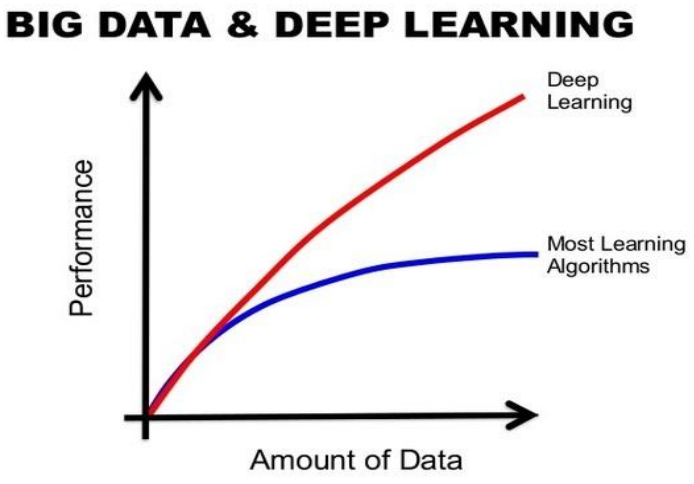


Figure 9. The performance of Deep Learning with respect to Big Data

[2] https://www.researchgate.net/publication/303741861: _A_Review_of_Deep_Machine_Learning

3.1.3. Convolutional Neural Networks (Conv Nets)

Traditional architectures for solving computer vision problems and the degree of success they enjoyed have been heavily reliant on hand-crafted features. However, of late, deep learning techniques have offered a compelling alternative that of automatically learning problem-specific features. Convolutional Neural Networks are considered one form of deep networks widely used in computer vision.[3]Convolutional Neural Networks are very similar to Artificial Neural Networks. The main difference between Conv Nets and ANNs, is that Conv Nets are made specially for images. The idea of applying the convolutional operation to image data is a common technique used in computer vision. As Dr.Terrence Sejnowski, a computational neuroscientist, described "If you look at the architecture of the CNNs, it's not just lots of units, they're connected in a fundamental way that mirrors the brain. One part of the brain that's best studied in the visual system and fundamental work in the visual cortex show that there are simple and complex cells."[4]

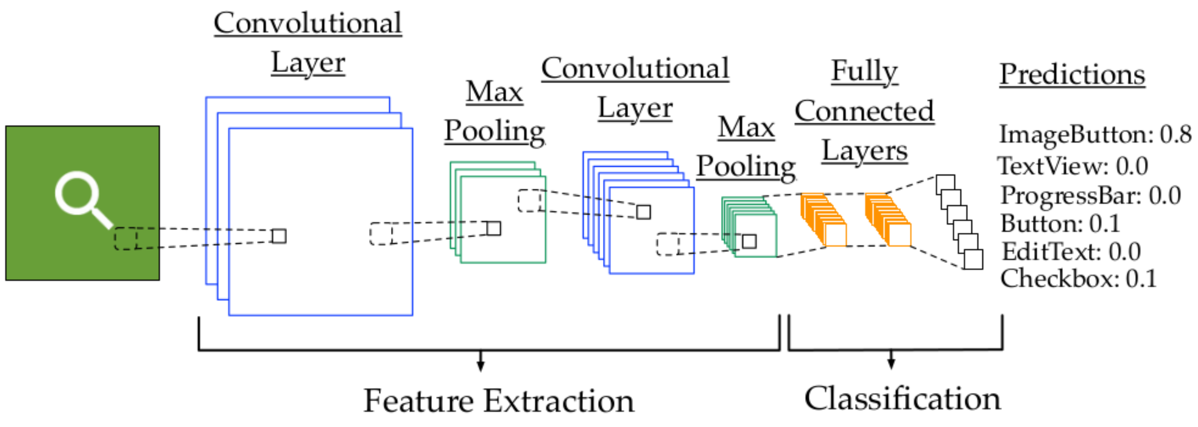


Figure 10. An image classification CCN architecture

3.1.3.1. Architecture Overview

In general, Neural Networks receive an input-a single vector- and passes it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the output layer and in classification settings it represents the class scores.

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In

[3] Chapter 2 - An Introduction to Deep Convolutional Neural Nets for Computer Vision. https://doi.org/10.1016/B978-0-12-810408-8.00003-1

[4] https://www.theverge.com/2018/10/16/17985168/deep-learning-revolution-terrence-sejnowski-artificial-intelligence-technology

particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in three dimensions: width, height, depth.

3.1.3.2. Conv Nets Layers[5]

- Input Layer

The Input layer holds the raw pixel values of the image, for example an image of width 150, height 150, and depth of three which are the color channels R,G,B. Usually, the third dimension (depth) is normalized by dividing each pixel value by 255, since the value of a single color channel ranges from 0 to 255.

- Convolution Layer

The Conv layer extracts features from the input image by computing the output of neurons that are connected to local regions in the input. It applies a dot product operation between a small square of the image matrix and a filter or kernel. In other words, it is multiplying the entries in the filter with the original pixel values of the image (computing element wise multiplications). The size of the filter is usually a 3*3 or 5*5 matrix, in addition to a third dimension equal to the third dimension of the image matrix. The size of the filter matrix is a hyperparameter that we tune to find better results.

As the filter slides or convolves over the width and height of the input volume we will produce a 2-dimensional activation map also known as feature map that gives the responses of that filter at every spatial position. We must specify the stride with which the filters slide. When the stride is 1 then the filters move one pixel at a time. When the stride is 2 then the filters jump 2 pixels at a time. The later will produce smaller output volumes spatially.

Sometimes it is convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes. Most commonly it is used to exactly preserve the spatial size of the input volume so the output width and height are the same as that of the input.

[5] CS231 in Convolutional Neural Networks, Stanford University

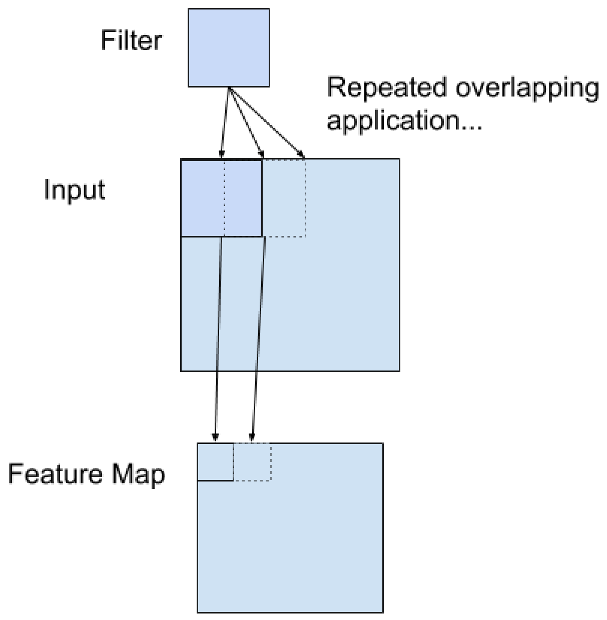


Figure 11. Example of a Filter Applied to a Two-Dimensional Input to create a feature Map

Different filters can perform operations such as edge detection, blur and sharpen. The network will learn the filters that activate when they see some type of visual feature such as an edge of some orientation on the first layer, or eventually entire patterns and shapes on higher layers of the network. So, the values of the filter matrix are the learnable weights.

Now, each CONV layer has an entire set of filters and each of them will produce a separate 2-dimensional activation map. The depth of the output volume of each layer is a hyperparameter which corresponds to the number of filters we would like to use, each learning to look for something different in the input. The activation maps generated are stacked along the depth dimension to produce the output volume.

Several Conv layers are stacked so they are not only applied to input data, e.g. raw pixel values, but they can also be applied to the output of other layers. The filters that operate on the output of the first layers may extract more complex features that are combinations of lower-level features.

- ReLU Layer

After each Conv layer, it is a convention to apply a nonlinear layer (activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the Conv layers, just element wise multiplication and summations. This allows the neural network to learn more complex features. These functions are embedded within Conv. In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster because of the computational efficiency.

ReLU layer applies an elementwise activation function, such as the max (0,x) thresholding at zero. This leaves the size of the volume unchanged.

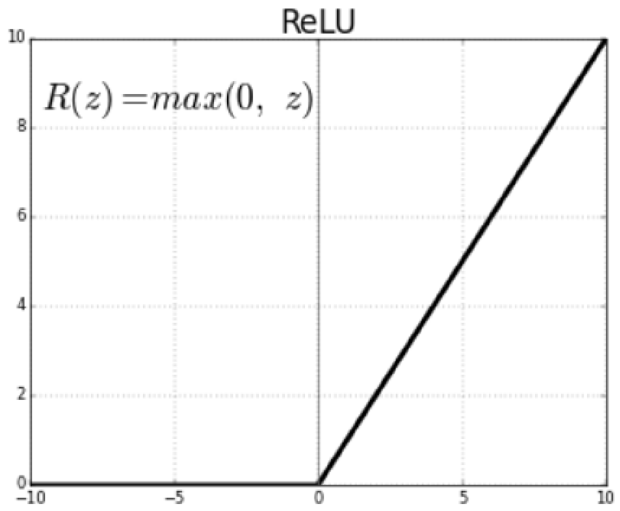


Figure 12. The ReLU activation function.

- Pool Layer

A pooling layer is also referred to as a down sampling layer. It is the layer that lies between two successive Conv layers. It performs a down sampling along the spatial dimensions (width, height). This subsampling or down sampling operation reduces the dimensionality of each output map but retains the important information. Accordingly, the amount of parameters and computation in the network are reduced.

In this category, there are several options of pooling, with max pooling being the most popular. This basically takes a filter, normally of size 2*2 and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub-region that the filter convolves around.

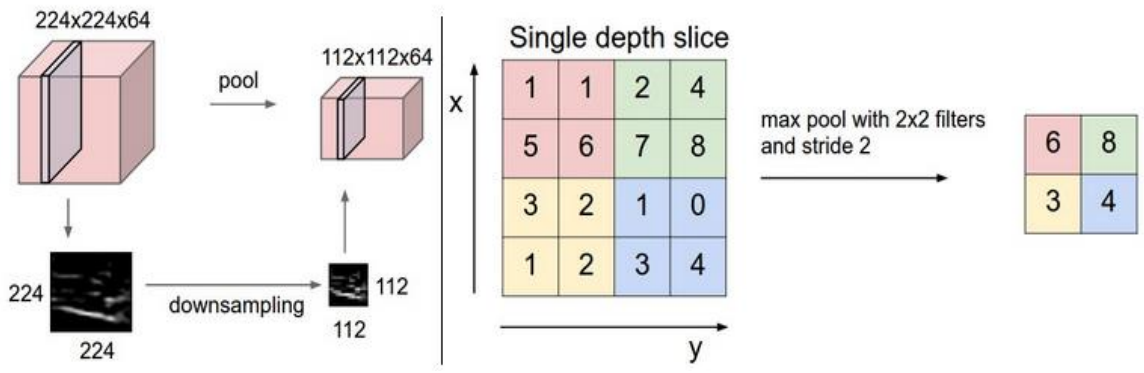


Figure 13. A figure demonstrating the effect of applying max pool

In our case we started with an input of width and height 150*150, after applying the first max pooling layer with filter size 2*2 the output volume is 75*75*3, the depth is not affected by the subsampling operation.

- Fully-Connected Layer

The output matrix from the previous Conv, ReLU and pooling layers is flattened into a vector and fed into a fully connected layer like neural network. In a FC layer neurons (units or nodes) have full connections to all activations in the previous layer, as in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. The FC layer is also known as dense layer. The dense layers are responsible for computing the class scores, resulting in an N dimensional vector at the final output dense layer, where N is the number of classes that the model has to choose from.

- Softmax Layer

Softmax is another commonly known activation function. It is used in the output layer to represent categorical values. It is a mathematical function that maps input (feature value) to its probability distribution.

$$Soft\max(x)_J = \frac{e^{X_J}}{\sum_{k=1}^{K} e^{X_k}} \quad for J = 1,\ldots\ldots,K$$

X_j is a feature input, k is the total number of features and J is the index of “X” feature.

- Dropout Layer

Deep learning neural networks are likely to quickly overfit a training dataset with few examples. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. One approach to reduce overfitting is to fit all possible different neural networks on the same dataset and to average the predictions from each model. This is not feasible in practice.

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or dropped out. This has the effect of making the layer look like a layer with a different number of nodes and connectivity to the prior layer. As a result, each update to a layer during training is performed with a different view of the configured layer.[6]

3.1.4. The Neural Network training process

Supervised neural networks rely on learning from data examples. A training dataset is used to train (fit) a neural network model. The goal in training a Neural Network is to find the point of least error as fast as possible. The training process starts with initial neuron weights and then iteratively updating those weights using the backpropagation algorithm. A validation dataset is the sample of data used to provide an evaluation of a model fit on the training dataset, Machine learning engineers use this data to fine-tune the model hyperparameters such learning rate, number of epochs, batch size,etc.[7] The learning rate controls the portion of error that is considered for updating the model weights. The number of epochs defines the number of times that the learning algorithm will work through the entire training dataset. The batch size defines the number of samples to work through before updating the internal model parameters.[8] Finally, The test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained.

To fit a model, we need to specify a way to measure such error and the optimization algorithm for updating the weights.

3.1.4.1. Loss Function

The choice of an error function, conventionally called a loss function, is crucial. The loss function that is used to estimate the loss of the model, so that the weights can be updated to reduce the loss on the next evaluation

The choice of loss function must match the framing of the specific predictive modeling problem, such as classification or regression. Further, for the classification model there are several loss functions to choose from depending on the configuration of the output layer.

Categorical Cross-Entropy loss, also called Softmax Loss.It is a Softmax activation plus a Cross-Entropy loss. This loss is used when a CNN is trained to output a probability over **C** classes for each image. It is used for multi-class classification.[9]

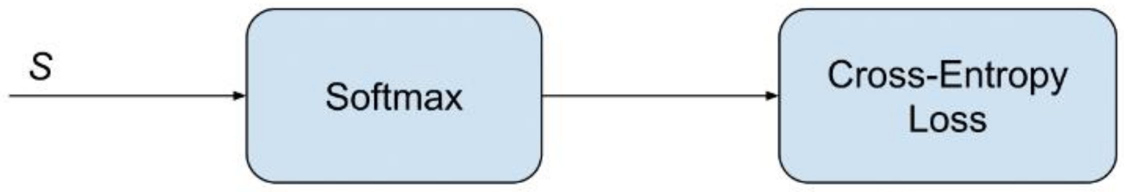
---

[6] A Gentle Introduction to Dropout for Regularizing Deep Neural Networks By Jason Brownlee on December 3, 2018 in Better Deep Learning

7 Tarang Shah - Blog Train, Validation and Test Sets.

[8] What is the Difference Between a Batch and an Epoch in a Neural Network? By Jason Brownlee on July 20, 2018 in Deep Learning

[9] https://gombru.github.io/2018/05/23/cross_entropy_loss/



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

Figure 14. The categorical cross entropy equations

3.1.4.2. *Optimization Algorithms*[10]

- Stochastic Gradient Descent

Gradient descent is a famous algorithm for gradient based optimization. Gradient descent is a way to minimize an objective function parameterized by a model's parameters. This is achieved by updating the parameters in the opposite direction of the gradient of the objective function with respect to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.

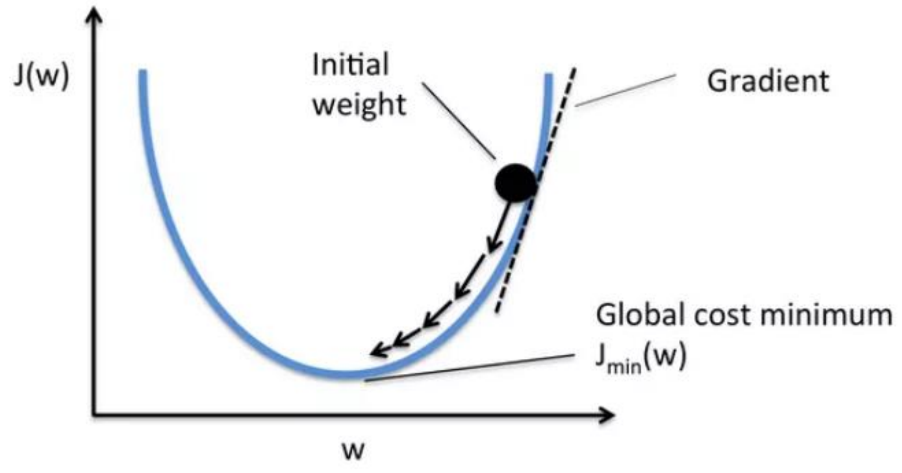


Figure 15. Gradient descent algorithm

10 An overview of gradient descent optimization algorithms: https://arxiv.org/abs/1609.04747

Stochastic Gradient Descent is one of the gradient descent variants. It performs a parameter update for each training example. Batch gradient descent performs redundant computations for large datasets, as it re-computes gradients for similar examples before each parameter update. SGD avoids this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

$$\theta = \theta - \eta \cdot \nabla\theta J(\theta; x(i); y(i)).$$

The above equation demonstrates how the weights (parameters θ) are updated, where $\eta$is the learning rate, $x(i)$is a training example with a label. $J(\theta; x(i); y(i))$is the objective function, in terms ofθ, $x(i)$, $y(i)$

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.

Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$vt = \gamma vt - 1 + \eta\nabla\theta J(\theta)$$
$$\theta = \theta - vt$$

Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity if there is air resistance, i.e. $\gamma<1$). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same direction and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

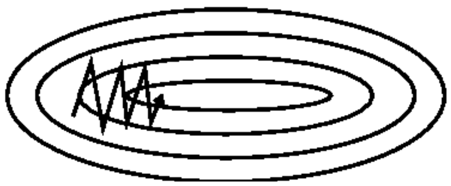


Figure 16. SGD without momentum

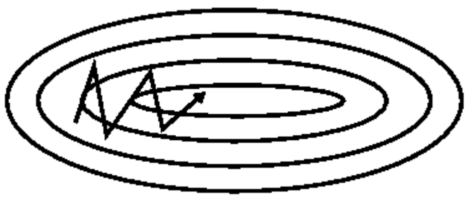


Figure 17. SGD with momentum

- Adaptive Moment Estimation (Adam)

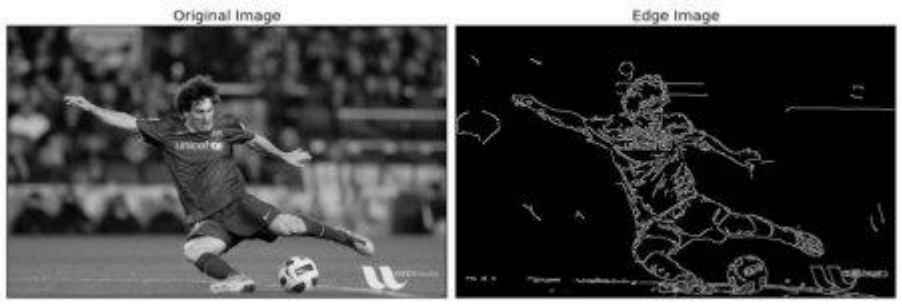
  Adaptive Moment Estimation is an algorithm for gradient-based optimization. It adapts the learning rate to the parameters, performing smaller updates. In addition, it stores an exponentially decaying average of past squared gradients v_t. Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum.

3.1.5. Image Processing Techniques

This section introduces some image processing concepts and techniques. These techniques are referred to later in the upcoming chapters.

3.1.5.1. Canny Edge Detection

Canny edge detection is a popular edge detection algorithm. It is a multi-stage algorithm. Since edge detection is susceptible to noise in the image, the first step is to remove the noise in the image with a 5x5 Gaussian filter which performs image blurring (smoothing). Then, the smoothened image is filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, canny algorithm finds the edge gradient and direction for each pixel. After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. In the last step, hysteresis thresholding, stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. The final result is a binary image with strong edges.[11]



3.1.5.2. Contours

11 OpenCv Canny Edge Detection: https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html

A contour is simply a curve joining all the continuous points (along the boundary), having the same color or intensity. For better accuracy, binary images are used. So before finding contours, canny edge detection is applied. The contours are a useful tool for shape analysis and object detection and recognition. The contour area and perimeter are used as features for the object. The area and perimeter are also used to calculate more features such as the circularity of the shape. In addition, the contour shape is approximated to another shape with less number of vertices. The number of vertices gives an indication of the shape type whether it is most likely a triangle, rectangle, pentagon, circle, etc.

3.1.5.3. Connected Components Labeling:

Connected Component Labeling (CCL) is a basic algorithm in image processing and an essential step in nearly every application dealing with object detection. Connected component labeling works by scanning an image, pixel-by-pixel (from top to bottom and left to right) in order to identify connected pixel regions, *i.e.* regions of adjacent pixels which share the same set of intensity values *V*.

3.1.5.4. Histogram of Oriented Gradients

HOG is a feature descriptor (a feature vector that describes an image) used to detect objects. The HOG descriptor technique counts occurrences of gradient orientation in localized portions of an image (detection window). To calculate a HOG descriptor, first the horizontal and vertical gradients are calculated. Then, the image is divided into small connected regions called cells. For each cell an 8-bin histogram of gradient directions or edge orientations is computed for the pixels within the cell. The number of bins (the number of orientations) is not fixed, it can change from one implementation to another. The next step is to discretize each cell into angular bins according to the gradient orientation. Each cell's pixel contributes weighted gradient to its corresponding angular bin. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.[12]

[12] Histogram Of Oriented Gradients (hog) Descriptor Admin - https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor

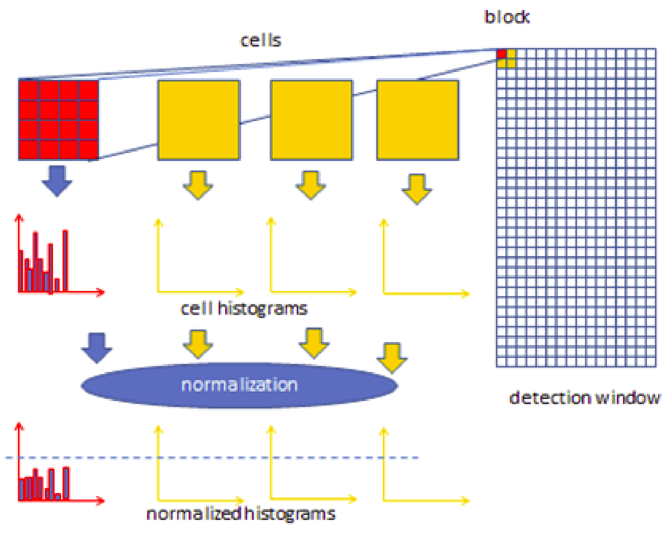


Figure 18. HOG algorithm

3.1.5.5. Local Binary Pattern

Local Binary Patterns are one of the texture descriptors. The LBP operator replaces the value of the pixels of an image with decimal numbers, which are called LBPs or LBP codes that encode the local structure around each pixel. Each central pixel is compared with its eight neighbors; the neighbors having smaller value than that of the central pixel will have the bit 0, and the other neighbors having a value equal to or greater than that of the central pixel will have the bit 1. For each given central pixel, a binary number is generated that is obtained by concatenating all these binary bits in a clockwise manner, which starts from one of its top-left neighbors. The resulting decimal value of the generated binary number replaces the central pixel value. The histogram of LBP labels (the frequency of occurrence of each code) calculated over a region or an image can be used as a texture descriptor of that image. The LBP histogram consists of 256 bins if 8 neighbors are considered. Another option is considering only 4 neighbors.[13]

[13] A Comparative Study of Image Segmentation Algorithms and Descriptors for Building Detection

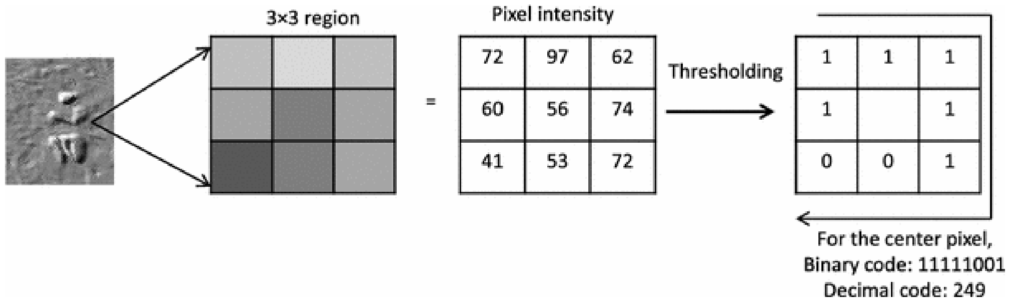


Figure 19. An illustration for LBP encoding

3.1.5.6. *Hough Line Transform*

Hough Transform is a popular technique to detect any shape, if you can represent that shape in mathematical form. It can detect the shape even if it is broken or distorted a little bit.

A line can be represented as $y = mx + c$ or in parametric form, as $\rho = x\ cos\theta + y\ sin\theta$where $\rho$is the perpendicular distance from origin to the line, and $\theta$is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise. Hough line transform algorithms detects the lines with different slopes m. The input image to the algorithm is a binary image resulted from canny edge detection.

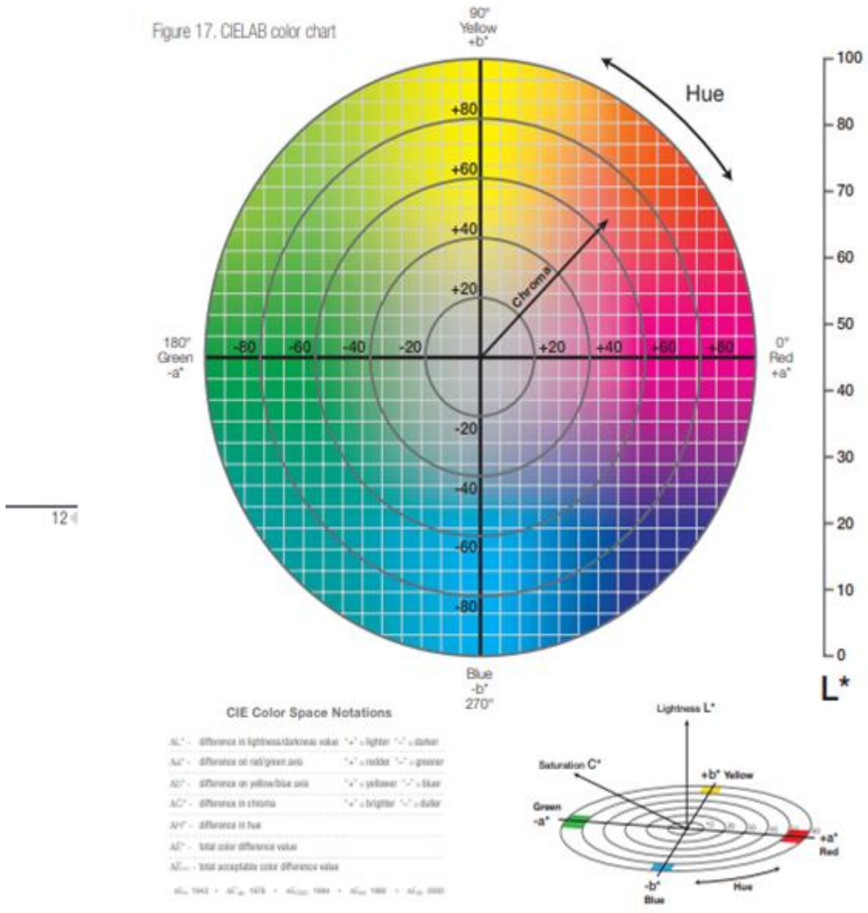
3.1.5.7. *Color Spaces*

In this section, two color spaces are presented which are RGB color space and CIELab color space.

RGB stands for red, green and blue. A pixel in an RGB image holds a value for each one of the three color channels. Any visible color ,in a digital image of this format, is a mixture of the three channels with different weights.

Lab stands for L: Lightness, a: Red/Green Value, b: Blue/Yellow Value. The CIE Delta E is the measure of change in visual perception of two given colors. It is a metric for understanding how the human eye perceives color difference. On a typical scale, the Delta E value will range from 0 to 100.[14]

[14] http://zschuessler.github.io/DeltaE/learn/
https://www.xrite.com/blog/lab-color-space



CIE Color Space Notations

ΔL* -	difference in lightness/darkness value	"+" = lighter "-" = darker
Δa* -	difference on red/green axis	"+" = redder "-" = greener
Δb* -	difference on yellow/blue axis	"+" = yellower "-" = bluer
ΔC* -	difference in chroma	"+" = brighter "-" = duller
ΔH* -	difference in hue	
ΔE* -	total color difference value	
ΔE -	total acceptable color difference value	

[illegible] 1942 • [illegible] 1976 • [illegible] 1984 • [illegible] 1988 • [illegible] 2000

Figure 20. CIE color space

Delta E	Perception
<= 1.0	Not perceptible by human eyes.
1 - 2	Perceptible through close observation.
2 - 10	Perceptible at a glance.
11 - 49	Colors are more similar than opposite
100	Colors are exact opposite

Figure 21. Different Delta E ranges and their indications

3.1.6. Android Development Concepts[15]

In this section we discuss some Android development concepts in order to give a complete illustration of our project.

3.1.6.1. Activities

An activity is the entry point for interacting with the user. It represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Similar to the main() function when working with C, C++ or Java programming language, one activity in an application is specified as the main activity, which is the first screen to appear when the user launches the app.

For an app to be able to switch between activities, activities and their attributes are declared, in the manifest file. Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest presents essential information about the application to the Android system, which permissions the application must have, etc. In addition, the manifest includes the Java package name for the application. The package name serves as a unique identifier for the application. The manifest also describes the activities of the application, mentioning which one starts first on launching the app. The manifest has a similar structure to one below.

```
<manifest ... >
  <application ... >
     <activity android:name=".ExampleActivity" />
     ...
  </application ... >
  ...
</manifest >
```

Starting a new instance of an Activity requires passing an Intent to startActivity() function. An Intent is a messaging object you can use to request an action from another app component. The Intent describes the activity to start and carries any necessary data.

15 Android Developers

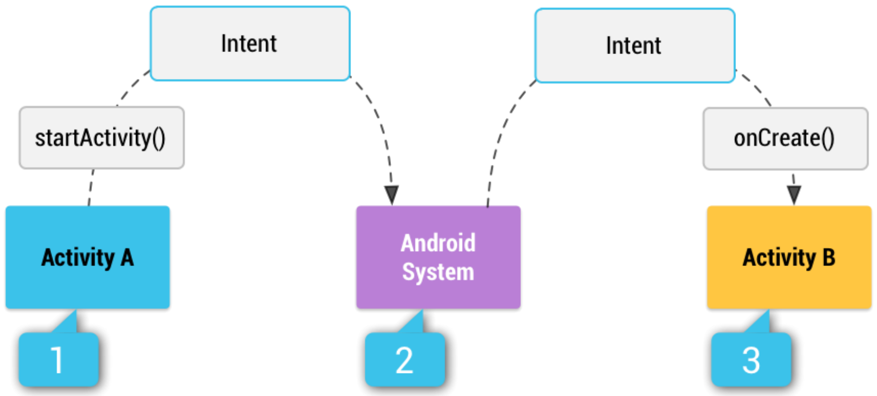


Figure 22. Starting new activities using Intent objects

3.1.6.2. Android Application Interface

- Widgets:

Widgets are the UI controls that allow you to build the graphical user interface for your app. Android also provides a variety of pre-built UI structured layouts. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and ViewGroup objects.

Android provides a wide variety of controls you can use in your UI. The most widely used widgets are Buttons,ImageButtons, TextViews, EditTexts, ImageViews, Switches, SeekBars, CheckBoxes, RadioButtons, etc. A TextView is used to display text to the user. EditText is a subclass of TextView that includes editing capabilities. You should use Checkboxes when presenting users with a group of selectable options that are not mutually exclusive. On the other side, a RadioGroup is used to group together one or more RadioButtons such that only one option could be chosen.

Figure 26. Android EditText

Figure 24. Android CheckBox

Figure 25. Android Button

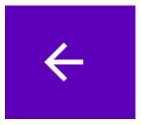


Figure 23. Android ImageButton

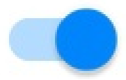


Figure 27. Android Switch

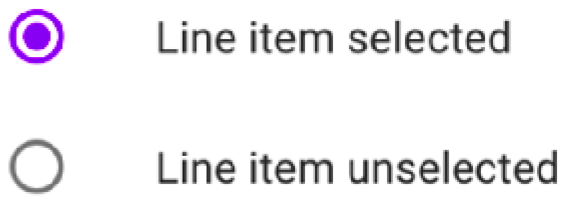


Figure 28. Android RadioGroup

To customize the widgets according to your preferences, Android XML is provided with a rich set of attributes some are general for all widgets or views such as width, height, background color, padding (spacings inside the view). Others are specific to some views including text attribute for TextViews and Buttons, source image for ImageViews and ImageButtons, onClick Listener for handling the user click events of the Buttons and ImageButtons.

- Containers and Layouts:

Widgets need to be grouped in a layout. Android provides a very flexible way to display layouts using a wide variety of layouts. To avoid complexity in building a layout hierarchy a LinearLayout is the one we choose in our approach in generating Android XML. A LinearLayout is a ViewGroup that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute. LinearLayout also supports assigning a weight to individual children with the android:layout_weight attribute. This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen.

To show a vertical list of scrollable items a ListView is a suggested way to do so. In the ListView the data is dynamically populated using the Java code. In other words, the data contents could be changed according to a database for example, or based on user input. This makes the ListView flexible for changing rather than a fixed (static) view. In the XML, only the type of views or widgets are specified but the data content is handled in the corresponding Java classes. The ListView could be a list of TexViews only, ImageViews with TextViews, ImageButtons with TextViews, CheckBoxes with TextViews, or more nested views.

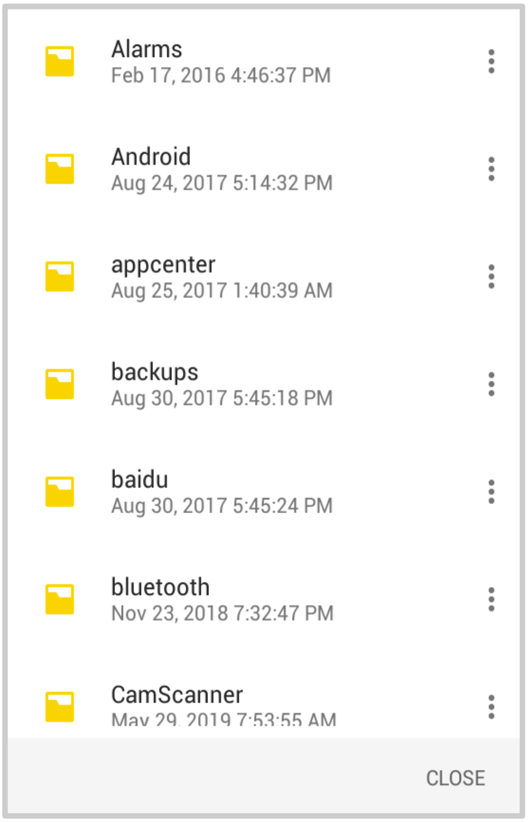

Alarms
Feb 17, 2016 4:46:37 PM
Android
Aug 24, 2017 5:14:32 PM
appcenter
Aug 25, 2017 1:40:39 AM
backups
Aug 30, 2017 5:45:18 PM
baidu
Aug 30, 2017 5:45:24 PM
bluetooth
Nov 23, 2018 7:32:47 PM
CamScanner
May 29, 2019 7:53:55 AM

Figure 29. An example of a ListView

- ActionBar

    By default any android application has an action bar that appears at the top of the screen containing the name of the application. Developers can customize this action bar and add more functionality.

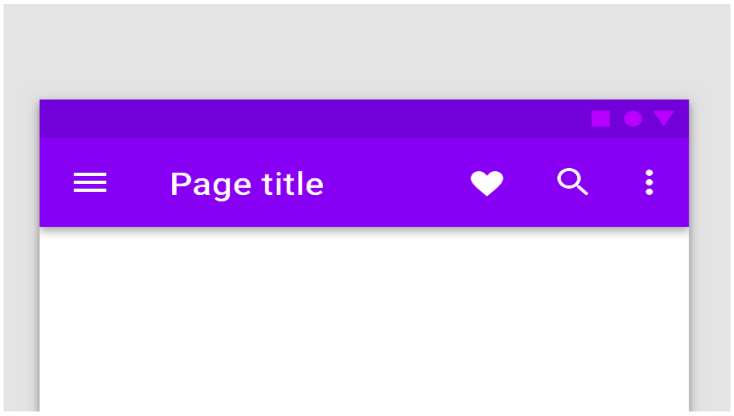


Figure 30. An example of an ActionBar

3.2. Literature Review

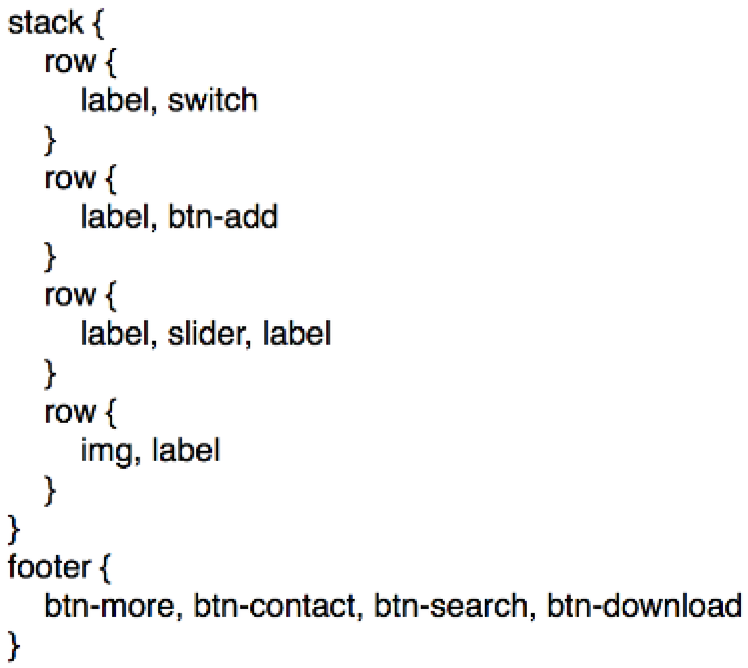
The automatic generation of software applications using machine learning techniques is a relatively new field of research. Researchers model this problem with different point of views resulting in having different approaches. This section presents different approaches researchers took to generate code from GUI.

A number of research have addressed the problem as an image captioning problem. A very similar research formulates the problem as neural machine translation where the input is an image, unlike normal machine translation tasks where both source and target languages are text data. These methods use the significant capabilities of deep learning and computer vision. The output of these approaches is the GUI skeleton in Domain Specific Language. DSLs are computer languages (e.g. markup languages, programming languages, modeling languages) that are designed for a specialized domain but are typically more restrictive than full-featured computer languages. So, the search space is limited to the some GUI components names, hierarchy keywords (e.g. stack, row) and opening and closing braces (‘{‘ and ’}’).[16]

In the machine translation approach, a neural machine translation model integrates a vision CNN, an RNN encoder and an RNN decoder in a unified framework. “Given an input UI image, the CNN extracts a diverse set of image features through a sequence of convolution and pooling operations. The RNN encoder then encodes the spatial layout information of these image features to a summary vector C, which is then used by the RNN decoder to generate the GUI skeleton in token sequence representation.”[17] Each token is represented as a one-hot

[16] pix2code: Generating Code from a Graphical UserInterface Screenshot: https://arxiv.org/pdf/1705.07962.pdf

[17] From UI Design Image to GUI Skeleton: A Neural MachineTranslator to Bootstrap Mobile GUI Implementation: https://chunyang-chen.github.io/publication/ui2code.pdf



```
stack {
   row {
      label, switch
   }
   row {
      label, btn-add
   }
   row {
      label, slider, label
   }
   row {
      img, label
   }
}
footer {
   btn-more, btn-contact, btn-search, btn-download
}
```

Figure 31. An example of DSL

vector. One-hot encoding maps categorical data to a binary representation, where ‘1’ is placed in the bit representing this token.

Recurrent neural networks are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. It is able to memorize parts of the inputs and use them to make accurate predictions. These networks are at the heart of speech recognition, translation and more.[18]

[18] CS 230 - Deep Learning: Recurrent Neural Networks cheatsheet
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
WildML

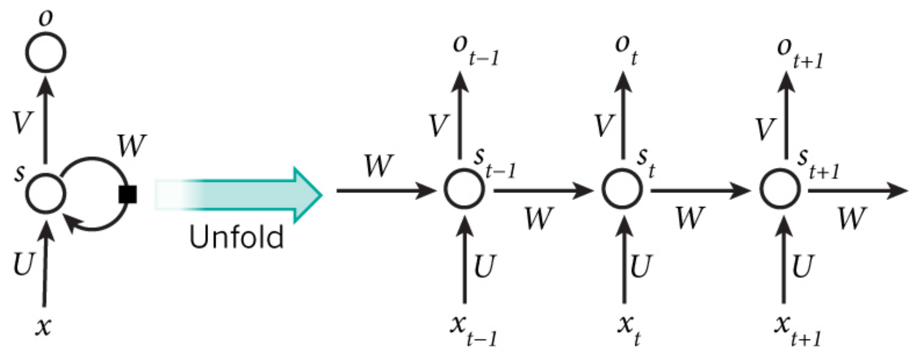


Figure 32. An abstract representation of RNN

Unfolding the RNN, simply means showing the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 tokens, the network would be unrolled into a 5-layer neural network, one layer for each token. x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence. s_t is the hidden state at time step t. It is the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes. o_t is the output at step t. To predict the next word in a sentence it would be a vector of probabilities across the prespecified vocabulary. $o_t = \mathrm{softmax}(Vs_t)$.[19]

The most commonly used type of RNNs are LSTMs, which are much better at capturing long-term dependencies. The LSTM architecture can solve general sequence to sequence problems.The idea is to use one LSTM to read the input sequence, one timestep at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector. A useful property of the LSTM is that it learns to map an input sentence of variable length into a fixed-dimensional vector representation.[20]

[19] WildML Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs by Denny Britz

[20] Sequence to Sequence Learning with Neural Networks: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

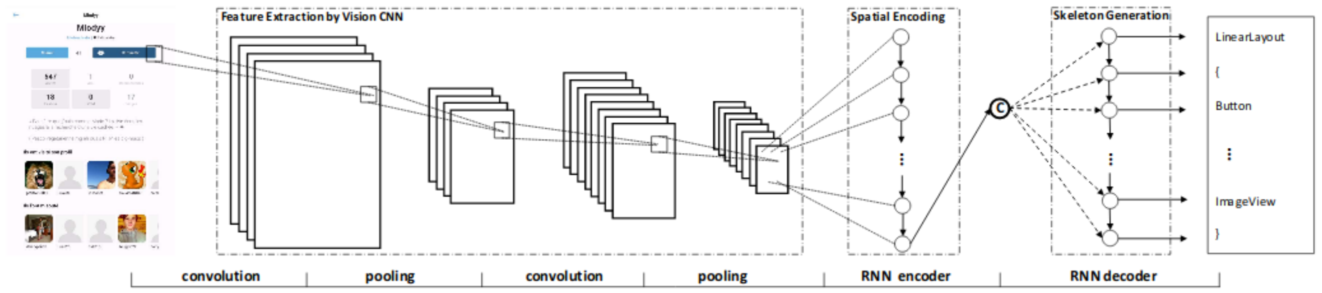


Figure 33. UI2code architecture

Although the neural machine translator proposed is composed of three neural networks (a vision CNN, a spatial layout RNN encoder, and a GUI skeleton generation RNN decoder), these networks are jointly trained end-to-end with one loss function. The RNN encoder and decoder backpropagate the error to the input, which makes end-to-end training possible. At training time, the sum of log probabilities is optimized over the whole training set using stochastic gradient descent. The training data consists of pairs of UI images and corresponding GUI skeletons. The dataset is collected from real Android applications available on Play store.

Similarly, the image captioning approach proposed a model consisting of a CNN and an RNN. The CNN maps the raw input image to a learned representation, in other words, a feature vector. The CNN-based vision model encodes the input image *I* into a vectorial representation *p* (feature). The input token is encoded by an LSTM-based language model into an intermediary representation q_t allowing the model to focus more on certain tokens and less on others. This first language model is implemented as a stack of two LSTM layers. The vision-encoded vector *p* and the language-encoded vector q_t are concatenated into a single feature vector r_t which is then fed into a second LSTM-based model decoding the representations learned by both the vision model and the language model. The decoder thus learns to model the relationship between objects present in the input GUI image and the associated tokens present in the DSL code. The decoder is also implemented as a stack of two LSTM layers each. This approach also takes advantage of its architecture to train the model end-to-end using gradient descent optimization.

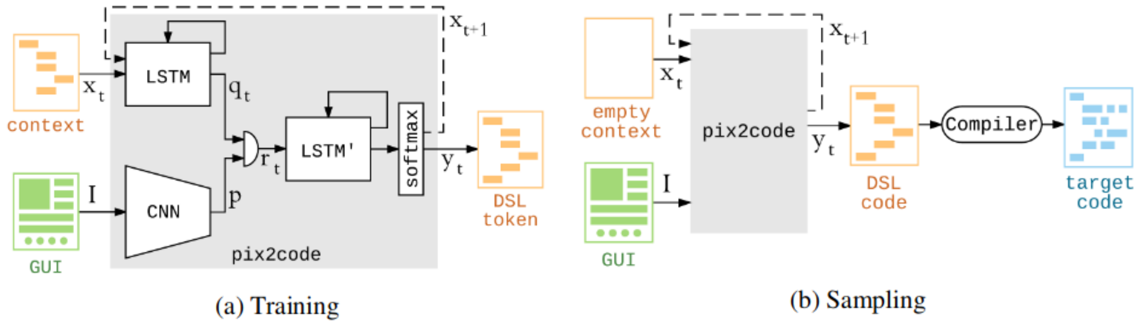
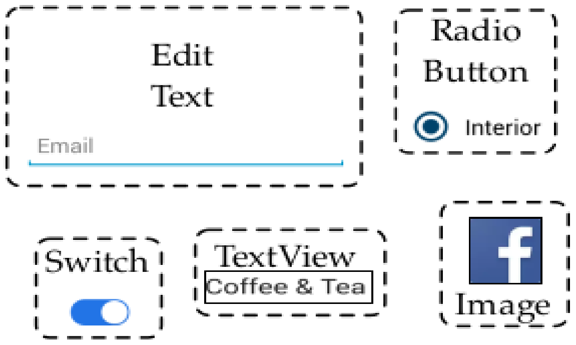


Figure 34. pix2code architecture

On the other side, a recent approach proposed using only a CNN. The dataset collected for training consists of labeled UI components images. The trained model can then be applied to mock-up artifacts to classify detected components. In order to construct a suitable GUI-hierarchies (e.g. proper groupings of GUI-components in GUI-containers), a KNN-based algorithm is used to assemble a realistic nested hierarchy of GUI-components and GUI-containers. In combination with computer vision techniques, a prototype application that closely resembles the mock-up GUI is generated.[21] This approach is adopted in our project, with some modifications and extensions as discussed in the next chapter.



[21] Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. #, NO. #, 2018

4. Design Specifications

This chapter discusses our system architecture and modules in detail.

4.1. System overview

The user enters one or more images exported from Photoshop as PNG or JPEG or screenshot for any application. Also, the user can enter hand-drawn images or PSD files. The following will be provided for any input type.

4.1.1. XML Generation.

Our app generates the XML files for those images with action bar and list view options.

4.1.2. Java Generation.

Our app generates the Java classes corresponding to each XML file with the basic functionalities like the onClick functions for buttons and the list views dynamic items filling.

4.1.3. Updating components.

After we show the output to the user we give him/her the option to update any of the components output results. We take those updates and update XML files and Java classes for him/her accordingly.

4.1.4. Connecting Activities.

Finally, we give the user the option to connect activities by choosing a button on any activity and the activity which he/she wants this button to switch to. We update the java classes with that information and the manifest XML file.

4.2. Block Diagram

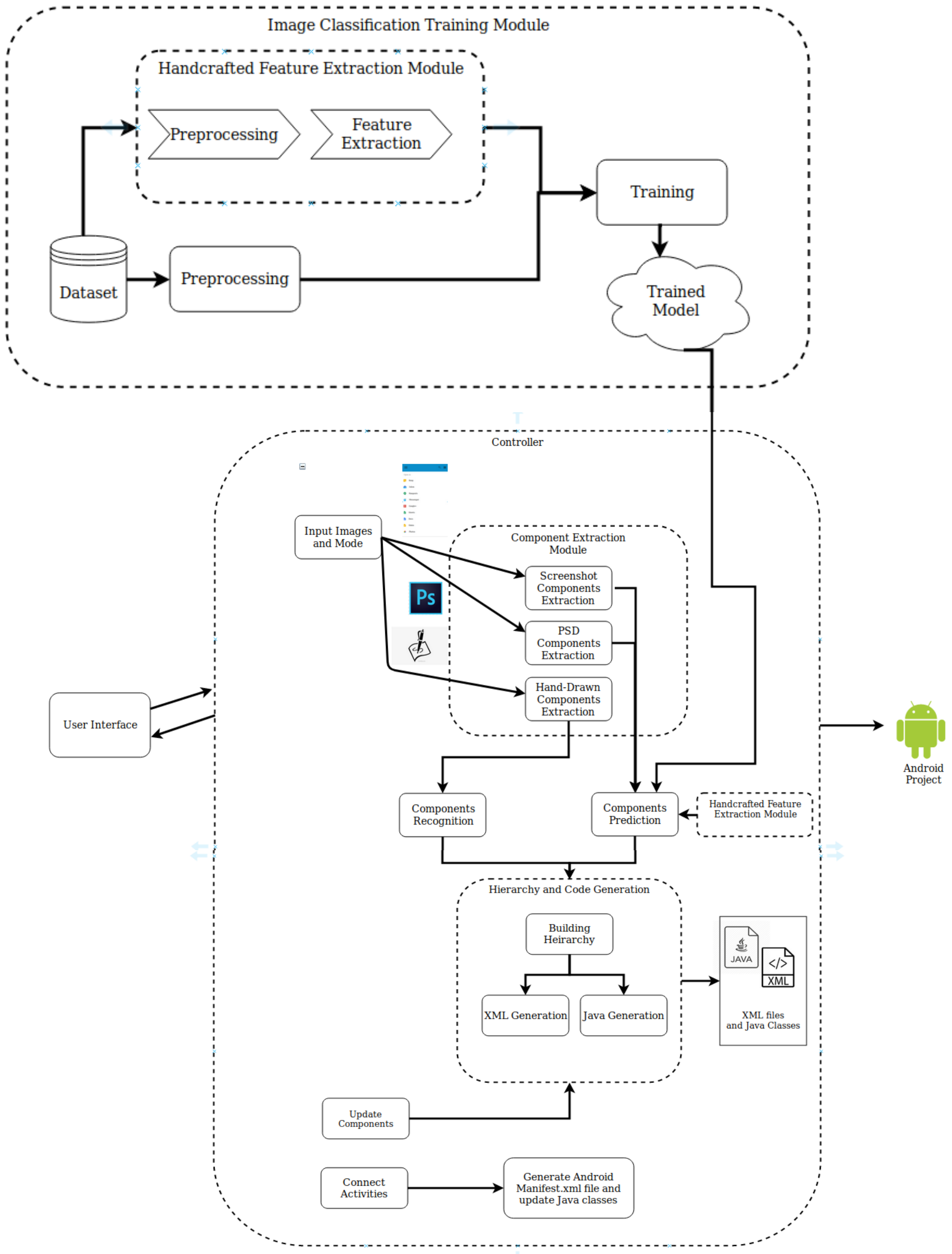


Image Classification Training Module
Handcrafted Feature Extraction Module
Preprocessing
Feature Extraction
Training
Dataset
Preprocessing
Trained Model
Controller
Input Images and Mode
Component Extraction Module
Screenshot Components Extraction
PSD Components Extraction
Hand-Drawn Components Extraction
User Interface
Android Project
Components Recognition
Components Prediction
Handcrafted Feature Extraction Module
Hierarchy and Code Generation
Building Heirarchy
XML Generation
Java Generation
JAVA
XML
XML files and Java Classes
Update Components
Connect Activities
Generate Android Manifest.xml file and update Java classes

Figure 35. The Block Diagram of our system

4.2.1. User Interface

- The user interface opens the main window to the user.
- The user selects the intended project directory.
- The user uploads the screenshot images, hand-drawn or PSD files.
- After the user runs the code, we generate the XML files and Java classes to him in the project directory. We show him/her the extracted components with the corresponding XML on the screen.
- Then the user can update components and connect activities through the user interface as described in the system overview.

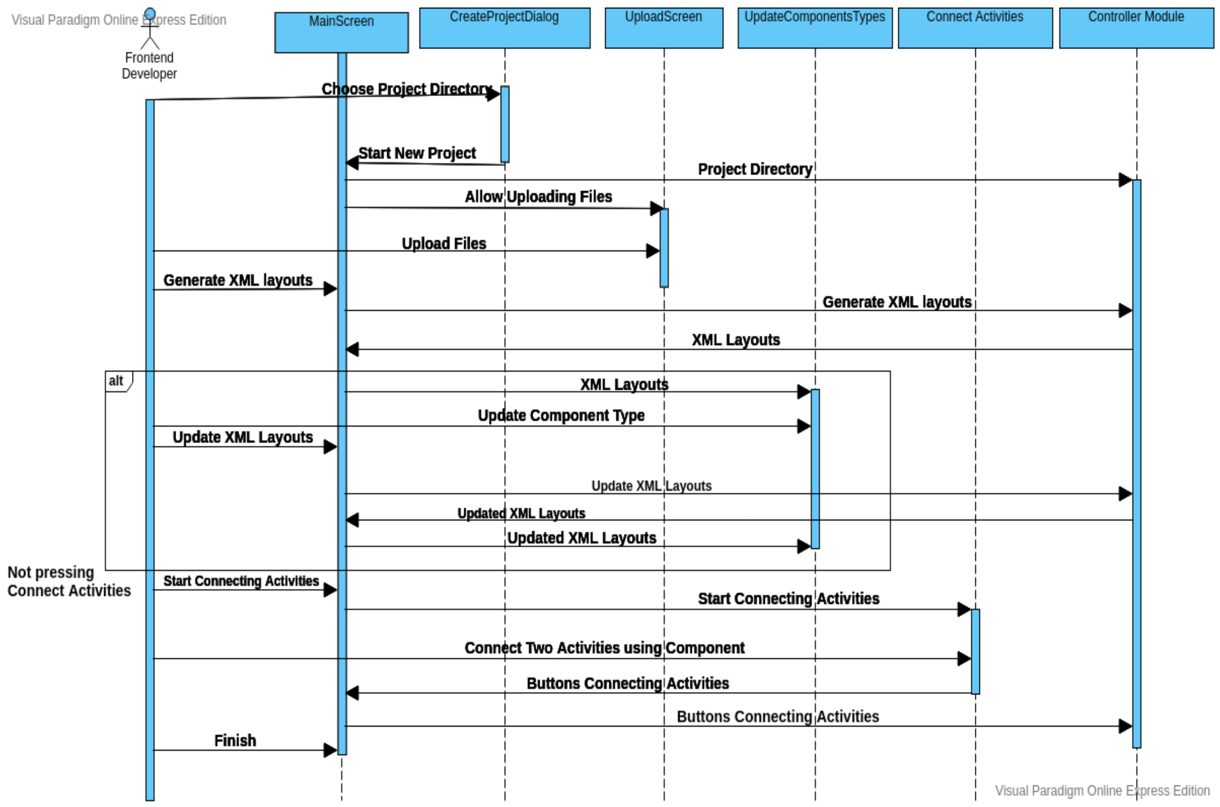


Figure 36. Sequence diagram of the user interactions with the system.

4.2.2. Handcrafted Feature Extraction

Our dataset consists of labeled UI components images where each image is one of those types:

- Button
- ImageView
- TextView
- Switch
- SeekBar

- CheckBox
- RadioButton
- EditText
- ImageButton
- CheckedTextView

We extract 18 features from each image:

- Normalized histogram of 256 bins for the LBP.
- Normalized hog histogram for eight edges directions.
- Five bins gray-scale normalized histogram.
- The number of lines in the image using the Hough transform. We standardized this number by subtracting the mean and dividing by the standard deviation which is calculated on the dataset.
- The maximum horizontal line length divided by the width of the image. This feature is deduced from the characteristics of the EditText and the SeekBar which have a long horizontal line.
- The number of lines with slope 45 degrees divided by the total number of lines. This feature is deduced from the characteristics of the checkbox mark.
- A Boolean variable that indicates if the image contains text or not by using the tesseract library for text extraction. This feature is deduced from the characteristics of the Buttons and the TextViews.
- The number of different colors appears in the image. We standardized this value by subtracting the mean and dividing by the standard deviation which is calculated on the dataset.
- The RGB of the background color which is the most common color in the image.
- The biggest contour area divided by the approximated box area.
- The biggest contour parameter divided by the approximated box parameter.
- The circularity of the biggest contour which is equal to 2*pi(contour area/(contour parameter*contour parameter)). This may help the model to learn that the RadioButton is a circle.
- The aspect ratio which is equal approximated box width divided by the approximated box height which helps to distinguish between the square and the rectangle.
- The number of vertices which approximates the biggest contour. We standardized this value by subtracting the mean and dividing by the standard deviation which is calculated on the dataset.
- A Boolean variable that indicates if the biggest contour is a square. This feature is deduced from the characteristics of the Checkbox and also most ImageViews.
- The number of edges in the image which is also standardized.
- The resizing ratio for the image width calculated by dividing the minimum between the width of the original image and the resizing

width -which is 150 in our case- by the other. We add a positive sign if the width of the original image is a bigger or negative sign if it is smaller.

- The resizing ratio for the image height calculates in the same way as the width.

4.2.3. Image Classification Training

This section focuses on the training process including building our model, layers, and parameters we used and the dataset the model trained on.

We first build our CNN model which takes the image after resizing as input. Our CNN model consists of the following layers.

- Keras Conv layer with filter size 7*7, Number of filters 64, stride 2*2, padding same and ReLU activation function.
- Keras Conv layer with filter size 7*7, Number of filters 64, stride 2*2, padding same and relReLU activation function.
- MaxPool layer with filter size 3*3.
- Keras Conv layer with filter size 3*3, Number of filters 96, padding same and ReLU activation function.
- MaxPool layer with filter size 2*2.
- Flatten layer.
- Dropout layer with dropout factor 0.5.
- Dense layer with 1024 hidden units.
- Dropout layer with dropout factor 0.5.
- Dense layer with 1024 hidden units.
- Output Dense layer with 10 hidden units -our vocabulary size- and we use the softmax activation function.

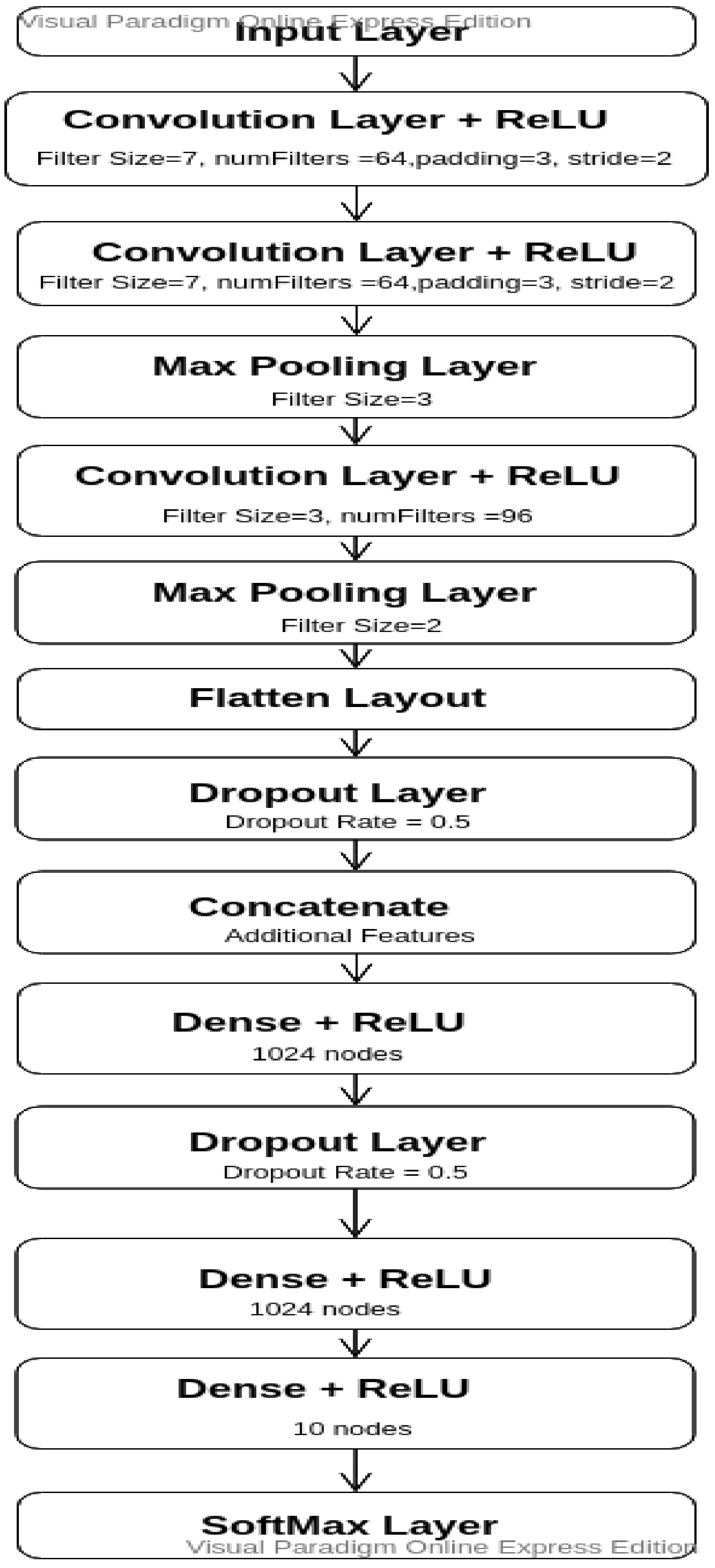
We also tried the AlexNet architecture but we got better results for the previous architecture which we adopted from the paper.

We trained our model with different image sizes which are 64*64, 150*150 and 224*224. We got the best results on 150*150 image size.

We also tried Adam optimizer and SGD with a decaying learning rate. We got the best results on SGD with 100 epochs where the learning rate decays from 0.001 in the first 50 epochs to 0.00001 in the next 25 epochs then to 0.000001 in the last 25 epochs.

We use momentum factor 0.9 and the batch size is 64. We train our model on 245000 images where 0.2 of data is used for validation in each epoch. We use categorical cross-entropy loss function and the metric we use is the categorical accuracy. After we trained our selected model with the mentioned selected parameters, we added the handcrafted extracted features to the model and retrained it.

Our feature vector is of length 286 which consists of the concatenated features described in the previous section. We concatenate the feature vector with the output of the Flatten layer after the Dropout. We got better results after adding features and the next figure shows our final model.


Visual Paradigm Online Express Edition
Input Layer
Convolution Layer + ReLU
Filter Size=7, numFilters =64,padding=3, stride=2
Convolution Layer + ReLU
Filter Size=7, numFilters =64,padding=3, stride=2
Max Pooling Layer
Filter Size=3
Convolution Layer + ReLU
Filter Size=3, numFilters =96
Max Pooling Layer
Filter Size=2
Flatten Layout
Dropout Layer
Dropout Rate = 0.5
Concatenate
Additional Features
Dense + ReLU
1024 nodes
Dropout Layer
Dropout Rate = 0.5
Dense + ReLU
1024 nodes
Dense + ReLU
10 nodes
SoftMax Layer
Visual Paradigm Online Express Edition

4.2.4. Component Extraction

This section discusses the component extraction process. This section is subdivided into three sections according to the type of input. The input could be a screenshot image in a JPG or PNG format, a PSD file where each layer corresponds to a component or a hand-drawn image in a JPG or PNG format.

4.2.4.1. Screenshots Component Extraction

- Boxes Extraction

  We preprocess our image by

1. Converting the image to gray-scale.
2. Blurring our image using Gaussian blur with kernel size 3*3.
3. Detecting the edges using Canny edge detection with low threshold 30 and high threshold 60 and with kernel size 3*3.
4. dilating the edges using kernel size 3*3.

Then, we extract the contours in the preprocessed image and approximate a box on each contour. Each box is considered as a component which we will pass to the model to predict. Many of these boxes are small parts from a real bigger component, so we need to make additional filtration after prediction as will be described later.

- Text Extraction

  In this phase, we extract the text in each of the extracted boxes in the previous phase using the tesseract library.

4.2.4.2. PSD Component Extraction

- Boxes Extraction

  We use PSD-tools library to extract the layers and corresponding boxes from the PSD file where each layer corresponds to a component.

- Text Extraction

  We extract the text in each of the extracted boxes using tesseract library as we do for the Screenshots.

4.2.4.3. Hand-drawing Component Extraction

- Boxes Extraction

  We preprocess our image by

1. Converting the image to gray-scale.
2. Detecting the edges by applying grey dilation on the gray-scale image and subtract the original gray-scale image from the dilation result.
3. Transforming the image into a binary form using thresholding techniques.

Then, we extract the connected component contours and approximate a box on each contour.

- Text Extraction

We extract the text in the full image by using Google API Vision which provides us with the text boxes in the image and the corresponding text inside. Then, we merge the text boxes on the same horizontal line if the distance between them is smaller than a certain threshold. Finally, we concatenate the text boxes with our extracted boxes and all this boxes will be used for the components recognition purpose in the next phase.

4.2.5. Components Prediction

4.2.5.1. Screenshots Component Prediction

- Prediction

We pass each of these boxes to the model after extracting the handcrafted features described above to get the model prediction.

- Filtration

In this phase, we filter the overlapping boxes to get the main components. To do this we first packet the overlapping boxes. Then, we sort each packet of boxes on the area so the biggest box area will be at the head -the first element in the packet -. We recursively check the head of the packet if we found that it is any component other than ImageView we return and consider this as one of the main components in the image and neglect the rest of the packet -the undesired inner overlapping boxes-.

If it is an ImageView we need to check that the ratio between its area to the image area is smaller than some threshold. We also need to check that the summation of the area of inner boxes in the same packet which predicted as ImageView is bigger than the summation of the area of the inner boxes that is not predicted as ImageView. This condition is to prevent some big boxes which include many components inside from being considered as ImageView.

If this condition is satisfied, we consider this component as ImageView and we return from the function and neglect the rest of this packet. However, if it is not satisfied we recursively repeat the packeting process on the rest of the boxes in the packet until one of the two conditions is satisfied. Finally, we only have the desired components after filtration.

4.2.5.2. PSD Component Prediction

We pass each of the boxes to the model after extracting the handcrafted features described above to get the model prediction. No filtration is needed here as each box corresponds to a layer -a component-.

4.2.6. Hand-drawing Components Recognition

We use image processing techniques in this phase to recognize the components. We First exclude the biggest box area in the image and consider it as the application border. Then we packet the rest of overlapping boxes and sort each packet on the area.

We always check the first element of the packet. If it is a text box we get from the Google API Vision, so we consider it as a TextView and neglect the rest of the boxes in the packet which are just letters boxes inside the text. If the first box in the packet is not a text box so this component is considered as Button if we found another text box inside this packet. If the packet does not contain a text box so it can be ImageView, CheckBox, EditText or RadioButton.

We recognize the RadioButton by getting the circularity of the biggest contour inside the box and if we found that it is a circle, we consider it as a RadioButton. To distinguish between the ImageView, the CheckBox, and the EditText, we first calculating the ratio between the height and the width of the box. If this ratio is smaller than a certain threshold, So this component is considered as EditText. However, if its area is smaller than a certain threshold we consider it as a CheckBox otherwise it is an ImageView.

4.2.7. Hierarchy and Code Generation

4.2.7.1. Building the Hierarchy

In this phase, we start by considering each component as a leaf node with the following attributes.

- The type which is one of the Android widget types.
- Identifier (id).
- x attribute which is the x-coordinate of the upper left corner of the box.
- y attribute which is the y-coordinate of the upper left corner of the box.
- Width.
- Height.
- Background color.
- Text color.
- Weight.
- Children list.

Then, we build the hierarchy by first packeting the nodes with the same horizontal range. We sort each packet on the x attribute and consider this sorted packet of nodes as the children list for a horizontal LinearLayout parent node. Each node inside the horizontal LinearLayout takes a weight. This weight is calculated according to the distance between the consecutive nodes' x attribute in the same packet with respect to the image width.

We then packet the children nodes of each horizontal LinearLayout vertically. If we find vertical packets we consider each packet as the children list for a vertical LinearLayout node. This vertical LinearLayout node replaces its children list in the parent horizontal LinearLayout node children list. We also sort the nodes inside the vertical LinearLayout on the y attribute. Each node inside the vertical LinearLayout takes a weight. This weight is calculated according to the distance between the consecutive nodes' y attribute in the same packet with respect to the image height. We again packet the children nodes of each vertical LinearLayout node horizontally and if we find horizontal packets we consider each packet as the children list for a horizontal

LinearLayout node. This horizontal LinearLayout node replaces its children list in the parent vertical LinearLayout node children list. Then, we assign weights to horizontal LinearLayout children nodes using the same way described above.

Finally, we consider all the horizontal LinearLayout parent nodes as the children list for a vertical LinearLayout node which is considered as the root node. We assign weights to each node in its children list depending on the x attribute as described above. For more illustration, this is an example for a complex nested hierarchy which we can generate.

```
<Vertical LinearLayout>
  <Horizontal LinearLayout>
    <Vertical LinearLayout>
      <Horizontal LinearLayout>
             <LeafNode>
             <LeafNode>
      <Horizontal LinearLayout>
    <Vertical LinearLayout>
  <Horizontal LinearLayout>
<Vertical LinearLayout>
```

In any horizontal LinearLayout, we merge the TextView nodes if the distance between them is smaller than a certain threshold. We determine the color of the background for the leaf nodes by getting the most major color in the node box (x, y, width, height). We also need to determine the text color in the Button and TextView. To do this we get the second most major color with a CIE deltaE higher than a certain threshold. We need this delta because the most common color has different shades in the image pixels, So we use the CIE deltaE to get the second most common different color.

To get the background color of the LinearLayout nodes, we subtract the inner children nodes from the background. We get the most common color of the background only after subtraction.

4.2.7.2. XML Generation

This module takes the root node which is built in the previous phase and recursively print the corresponding XML of each node in the XML file. If we find more than one RadioButton horizontally or vertically (giving the priority for the horizontal RadioButtons), we group them in a RadioGroup. If the user selects to generate Actionbar we take the first horizontal LinearLayout child node of the root and print it in a separate XML file for the Actionbar.

If the user selects the dynamic ListView option we detect the ListView by checking repeated horizontal LinearLayout pattern. We also print the types of the widgets contained in the ListView in a separate XML file.

4.2.7.3. Java Generation

This module takes the root node and recursively finds all the Buttons and ImageButtons to print the onClick functions for them.

If the user selects to generate Actionbar we print the needed Java code for this purpose. We print the needed Java code to fill the data of the ListView items in the Java classes if the user selects the dynamic ListView option. We also print the needed Java code for the RadioGroup to restrict clicking on one RadioButton only.

4.2.7.4. Components Update

We take the updated components from our User Interface. We update the nodes corresponding to these components in our hierarchy. We then repeat the process of XML and Java generation.

4.2.7.5. Activities Connection

We take the list of buttons with the containing activity and the intended activity from the User Interface. We print the Java code needed to handle the connection inside the onClick function of the corresponding button.We also print the needed XML inside the Manifest XML file.

4.3. Enhancements

We use multiprocessing to parallelize our code as much as possible. We parallelize the phase of features and text extraction on the extracted boxes. We create a number of processes and make each one work on a part of the extracted boxes.

We also parallelize the phase of XML and Java generation by creating a process for each image entered by the user to work on.

We do not include the phase of model prediction in the parallelization as the model loading takes time, so loading the model in each process is an overhead.

5. System Evaluation

This chapter demonstrates how the system modules were tested, including how the model was validated.

5.1. Hardware and Data Used

This section presents the hardware and data used for model training and validation.

5.1.1. Hardware Used

Google Colaboratory (Colab) provides a Tesla K80 GPU, having 2496 CUDA cores, 12GB (11.439GB Usable) RAM.[22] We used Colab to train and validate the model on the training and validation set respectively. In addition to a test set for an unbiased evaluation. The prediction is done our standard machines.

5.1.2. Data Used

The dataset we used for training, validation, test set is collected by the ReDraw project team. As a part of implementing ReDraw tool, the research team collected a large dataset of mobile application GUI data containing screenshots for over fourteen thousand (14k) screens and over a hundred and ninety thousand (190k) UI-components. The ReDraw dataset of mobile GUI data contains a set of labeled GUI-components cropped from larger screenshots that

are ready for processing by machine learning classifiers. Further, dataset was cleaned and more data was augmented to ensure balanced distributions of UI components images.[23] The dataset was categorized into training, validation and test sets. The training dataset contains over two-hundred and forty thousand (240k) training examples of labeled UI components images. The test set and validation set together contain about twenty-nine thousand (29k) labeled UI components images. We merged both sets and used them as a test set. While in the training process, the training dataset is split into training set and validation set by a ratio eight to two.

5.2. Results and Discussion

5.2.1. Image Classification Model Results

This section shows the results of the image classification model. We conducted several trials, changing the input image size and the optimizer algorithm used in training. A comparison of these modifications is shown. In addition, the results of adding the handcrafted features explained in the previous chapter.

[22]Google Colaboratory https://colab.research.google.com/drive/151805XTDg--dgHb3-AXJCpnWaqRhop_2#scrollTo=vEWe-FHNDY3E

[23] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Redraw online appendix https://www.android-dev-tools.com/redraw"

Table 1. Model results with SGD optimizer

	Image Size	
	150 pixel *150 pixel	224 pixel *224 pixel
Training set Accuracy	98%	97%
Validation set Accuracy	91%	90%
Test set Accuracy	85.5%	85%

Table 2. Model results with Adam optimizer

	Image Size	
	150 pixel *150 pixel	224 pixel *224 pixel
Training set Accuracy	98%	97%
Validation set Accuracy	90%	89%
Test set Accuracy	85%	83%

Based on the above results, we decided to proceed with the model that is trained with SGD optimizer. Firstly, four features were added for each training example and the training process was repeated giving the results shown in table 3. Secondly, more features were added summing up to a total of eighteen features that were mentioned in the previous chapter. An increase in the accuracy was achieved as shown in the following table.

Table 3. Model results when adding only four features compared to when adding the total 18 features

	Number of Features	
	4 features	18 features
Training set Accuracy	98%	98%
Validation set Accuracy	91.5%	92%
Test set Accuracy	86%	86%

The AlexNet implementation[24] we tried on our dataset resulted in the following results.

	AlexNet Results
Training set Accuracy	90%
Validation set Accuracy	85.5%

5.2.2. Components Extraction Results

This section demonstrates the results of the component extraction module. Different methods and techniques are used in this module according to the type of input image. As mentioned before, the input design image is one of three types, screenshot, hand-drawn image, or psd file. The results are shown for each mode.

5.2.2.1. Screenshot mode

The screenshot is an image with JPEG or PNG format. First, boxes are cut from the contours. Then, the overlapping boxes are packeted and explored to find only the boxes representing the UI components. The following figures demonstrate the results of the screenshot component extraction module and the components classification results.

[24] Amir-saniyan/alexnet Amir-Saniyan - https://github.com/amir-saniyan/AlexNet

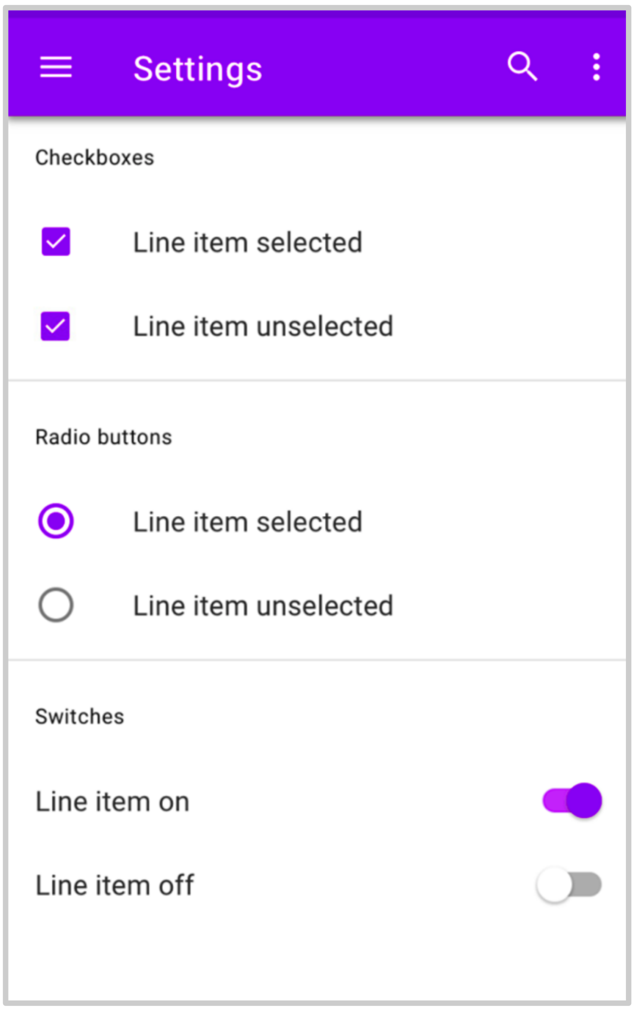


Figure 39. A sample of an input design used for testing

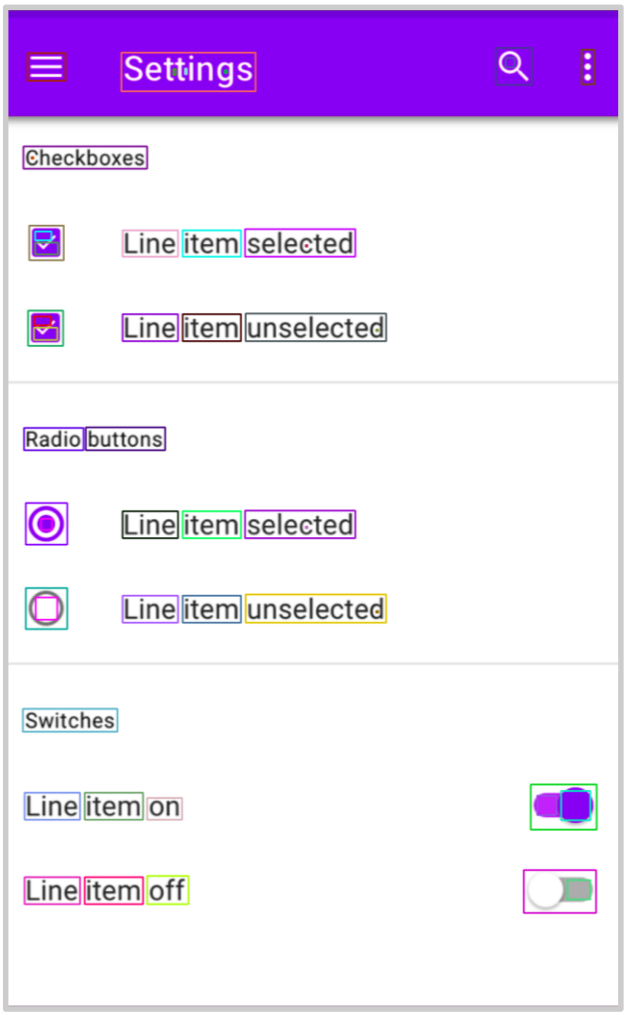


Figure 40. The results of the boxes drawn around the contours

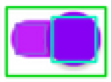


Figure 37. An example of overlapping boxes

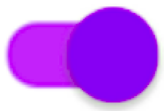


Figure 38. The component correctly classified as Switch

Another problem we needed to fix is merging the unmerged text. The following figures illustrate this problem. Each box is classified as a TextView, however, they are actually one TextView. We were able to merge theses TextViews into one TextView.

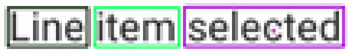
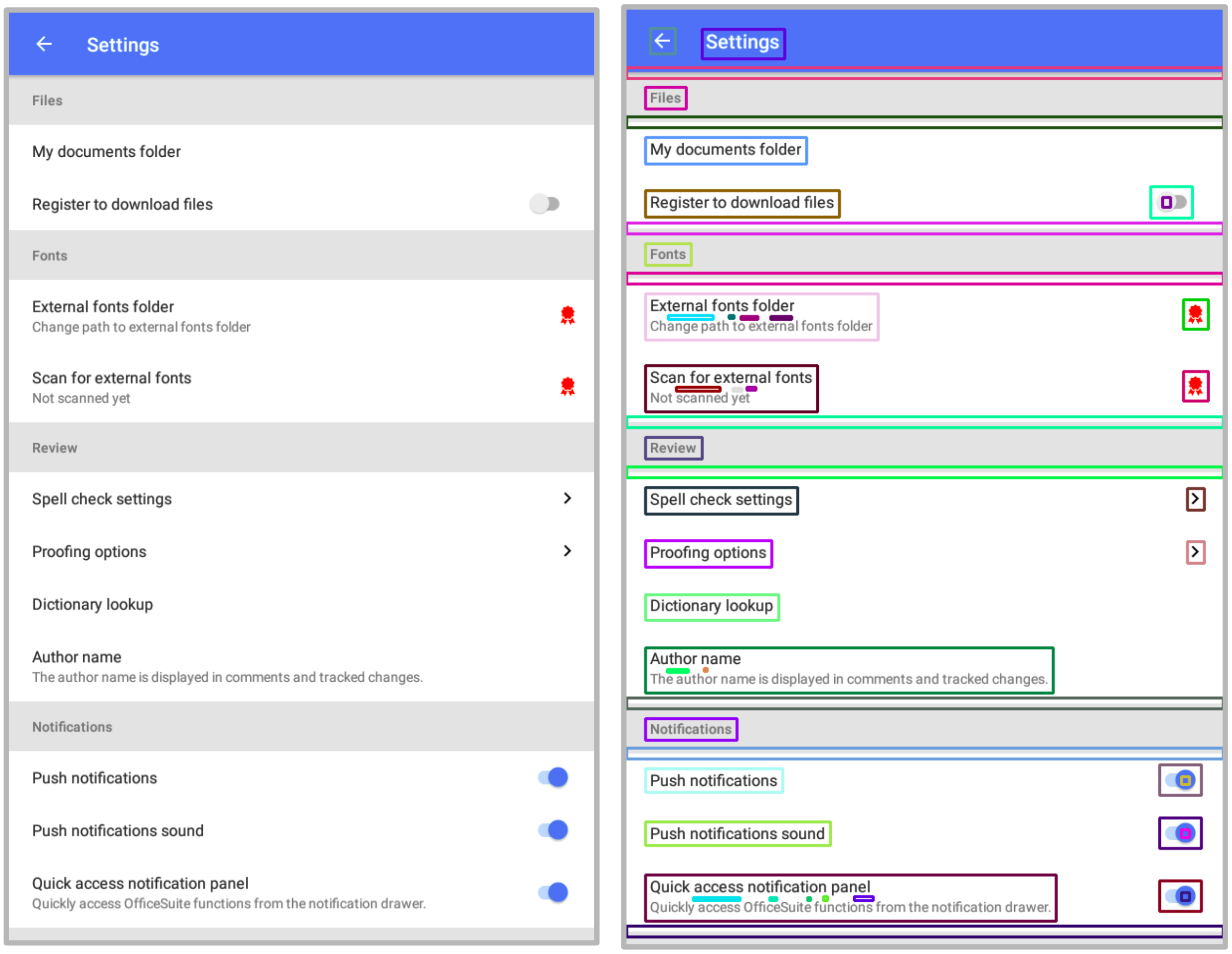


Figure 43. An example of the TextViews boxes that needs to be merged


Settings

Files

My documents folder

Register to download files

Fonts

External fonts folder
Change path to external fonts folder

Scan for external fonts
Not scanned yet

Review

Spell check settings

Proofing options

Dictionary lookup

Author name
The author name is displayed in comments and tracked changes.

Notifications

Push notifications

Push notifications sound

Quick access notification panel
Quickly access OfficeSuite functions from the notification drawer.

Figure 42. Another sample of a design and the extracted boxes

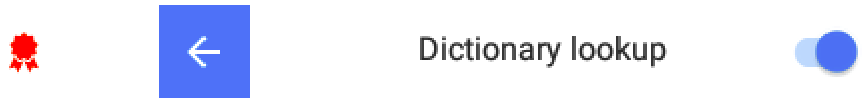
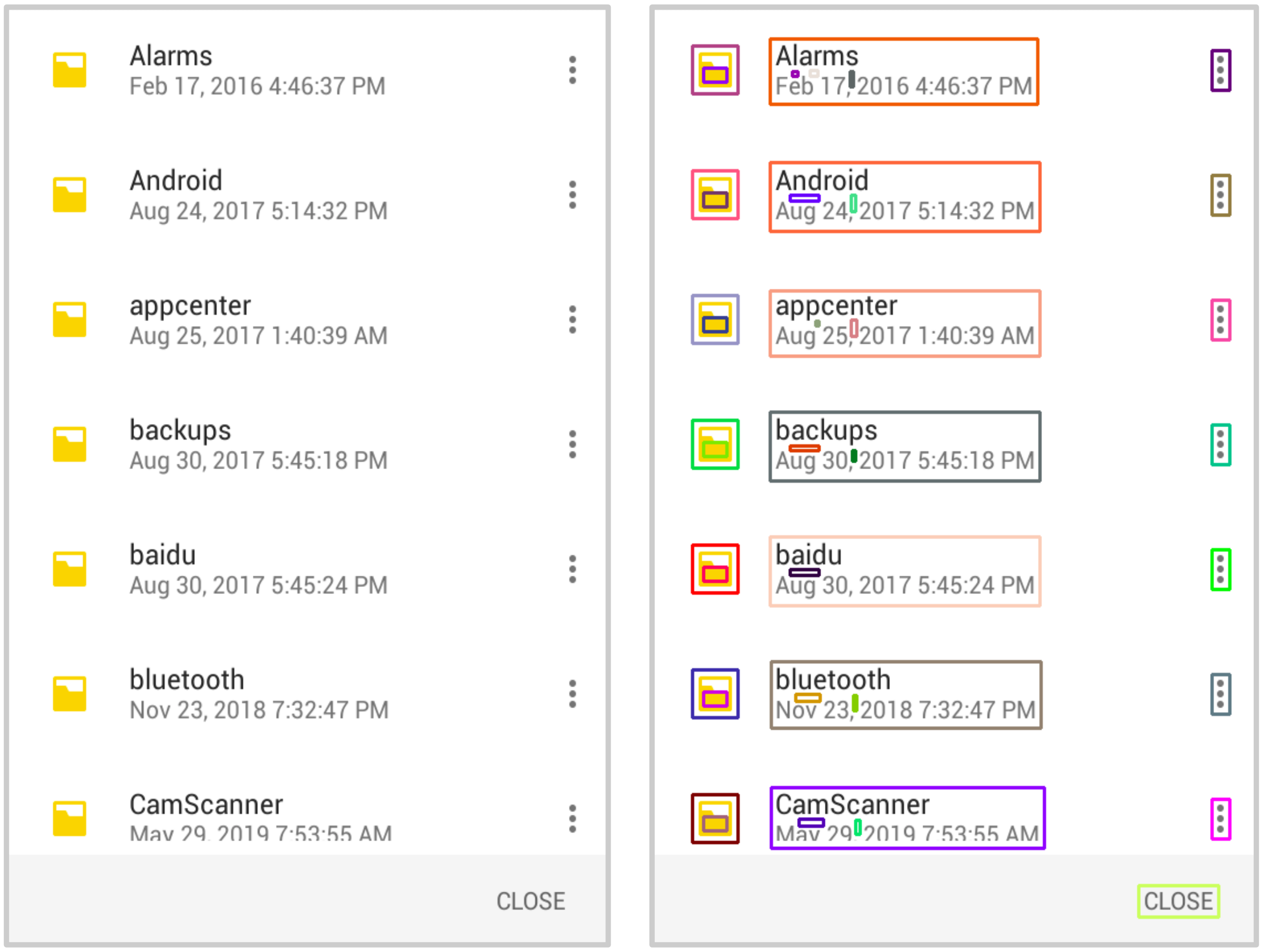


Figure 41. A sample of the components classified correctly



Alarms
Feb 17, 2016 4:46:37 PM

Android
Aug 24, 2017 5:14:32 PM

appcenter
Aug 25, 2017 1:40:39 AM

backups
Aug 30, 2017 5:45:18 PM

baidu
Aug 30, 2017 5:45:24 PM

bluetooth
Nov 23, 2018 7:32:47 PM

CamScanner
May 29, 2019 7:53:55 AM

CLOSE

Alarms
Feb 17, 2016 4:46:37 PM

Android
Aug 24, 2017 5:14:32 PM

appcenter
Aug 25, 2017 1:40:39 AM

backups
Aug 30, 2017 5:45:18 PM

baidu
Aug 30, 2017 5:45:24 PM

bluetooth
Nov 23, 2018 7:32:47 PM

CamScanner
May 29, 2019 7:53:55 AM

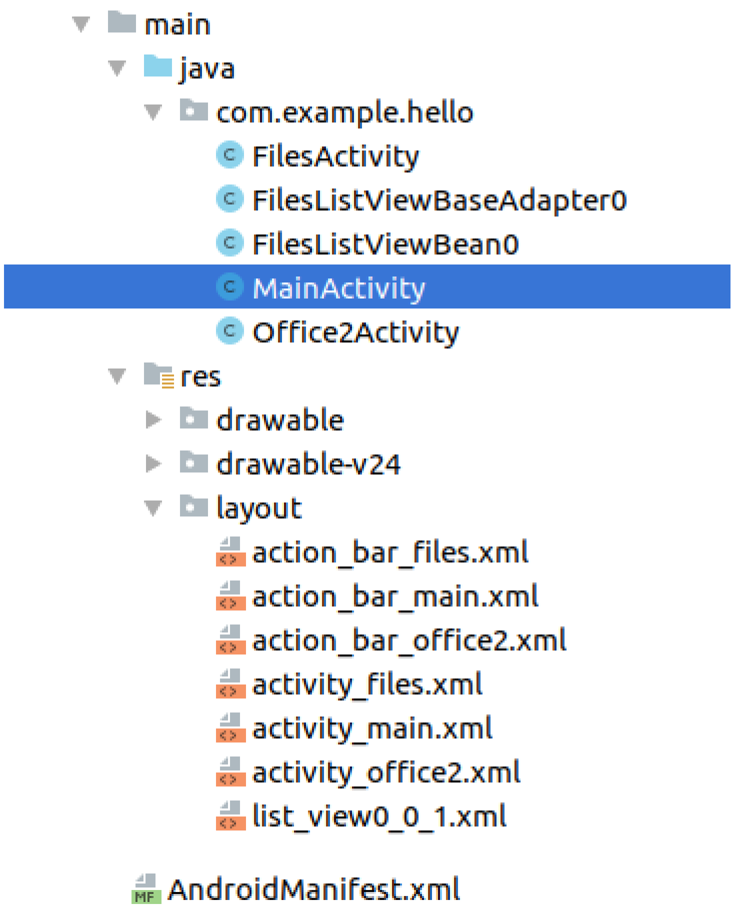
CLOSE

Figure 44. A sample of a design with a ListView

5.2.3. XML and Java Generation Results

This section illustrates the XML and Java classes generation results. In addition, an example of the generated manifest.xml file which allows the user to switch between activities.

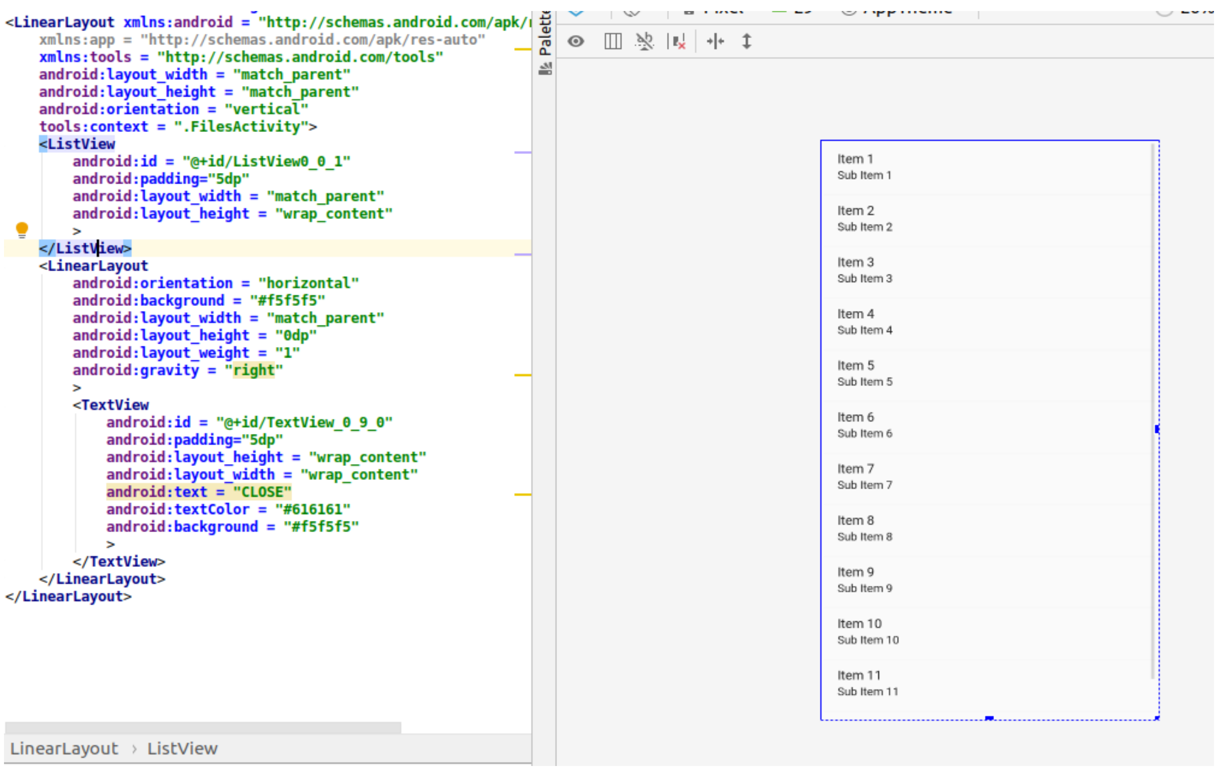
As shown in figure 45, the main directory in the Android studio projects contains the java directory containing the generated Java classes corresponding to each design image. The main directory also contains the resources folder (res folder) which contains the layout folder with all the generated XML files. In our case the design contains a ListView so, two extra Java classes are generated. These files are necessary for the ListView to appear when running the project on an Android mobile. Also, a separate XML file for the ListView is generated. Also the design has an ActionBar so, separate XML files are generated for rendering the ActionBar. The Android manifest file is also included in the main folder.



```
main
  java
    com.example.hello
      FilesActivity
      FilesListViewBaseAdapter0
      FilesListViewBean0
      MainActivity
      Office2Activity
  res
    drawable
    drawable-v24
    layout
      action_bar_files.xml
      action_bar_main.xml
      action_bar_office2.xml
      activity_files.xml
      activity_main.xml
      activity_office2.xml
      list_view0_0_1.xml
  AndroidManifest.xml
```

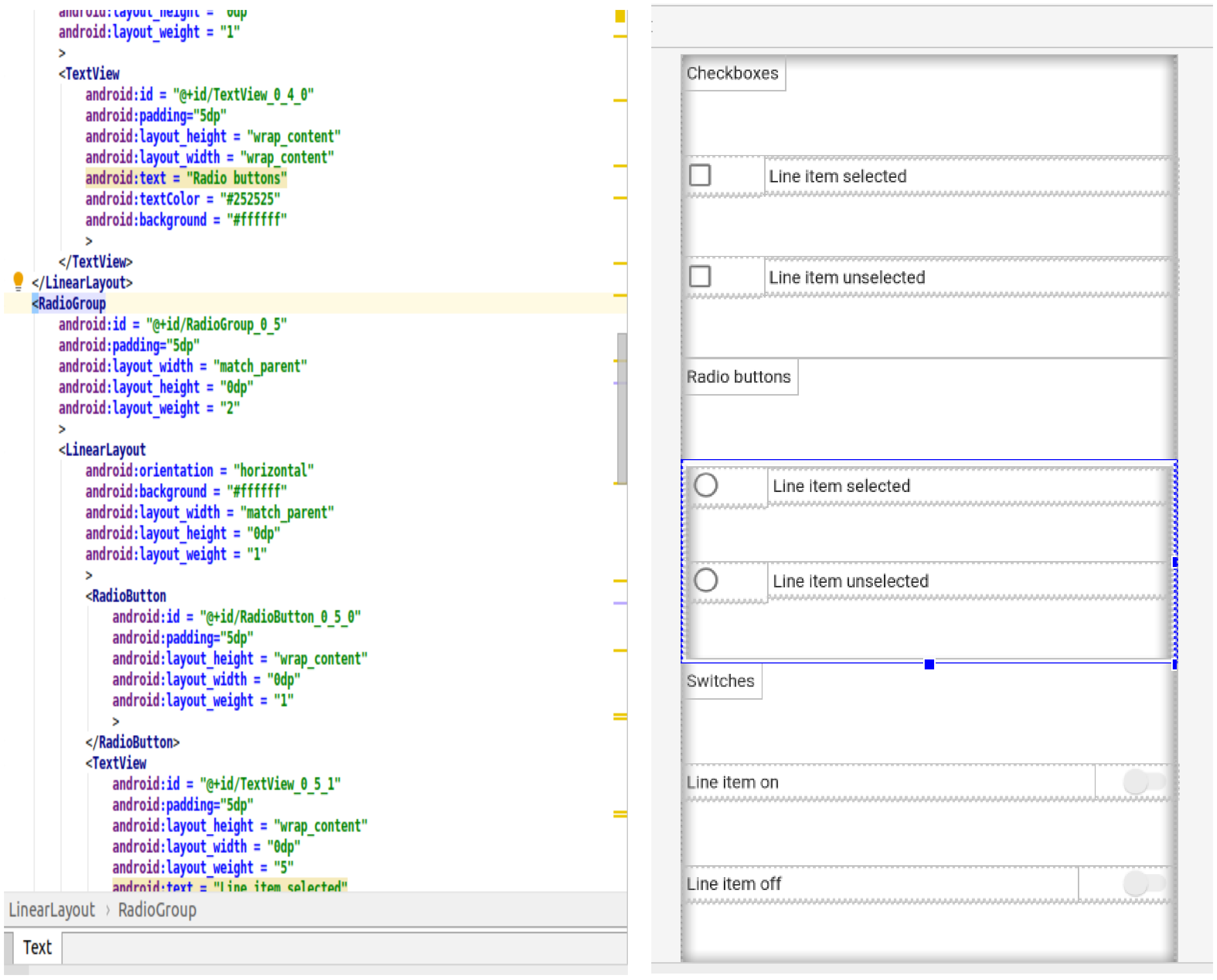
Figure 45. Android main directory

The following figures are taken from Android Studio, for the generated XML files of the previous designs. As shown in figure 46, the ListView is not rendered with the data contents. The project must be compiled and run on an Android mobile. Only then, the ListView items will appear. In figure 47, the generated XML for the design in figure 39 shows that the two RadioButtons are grouped in a RadioGroup. In figure 48, the Java code for the second sample design (shown in figure 40) is shown with the onClick function. The function starts the MainActivity again.



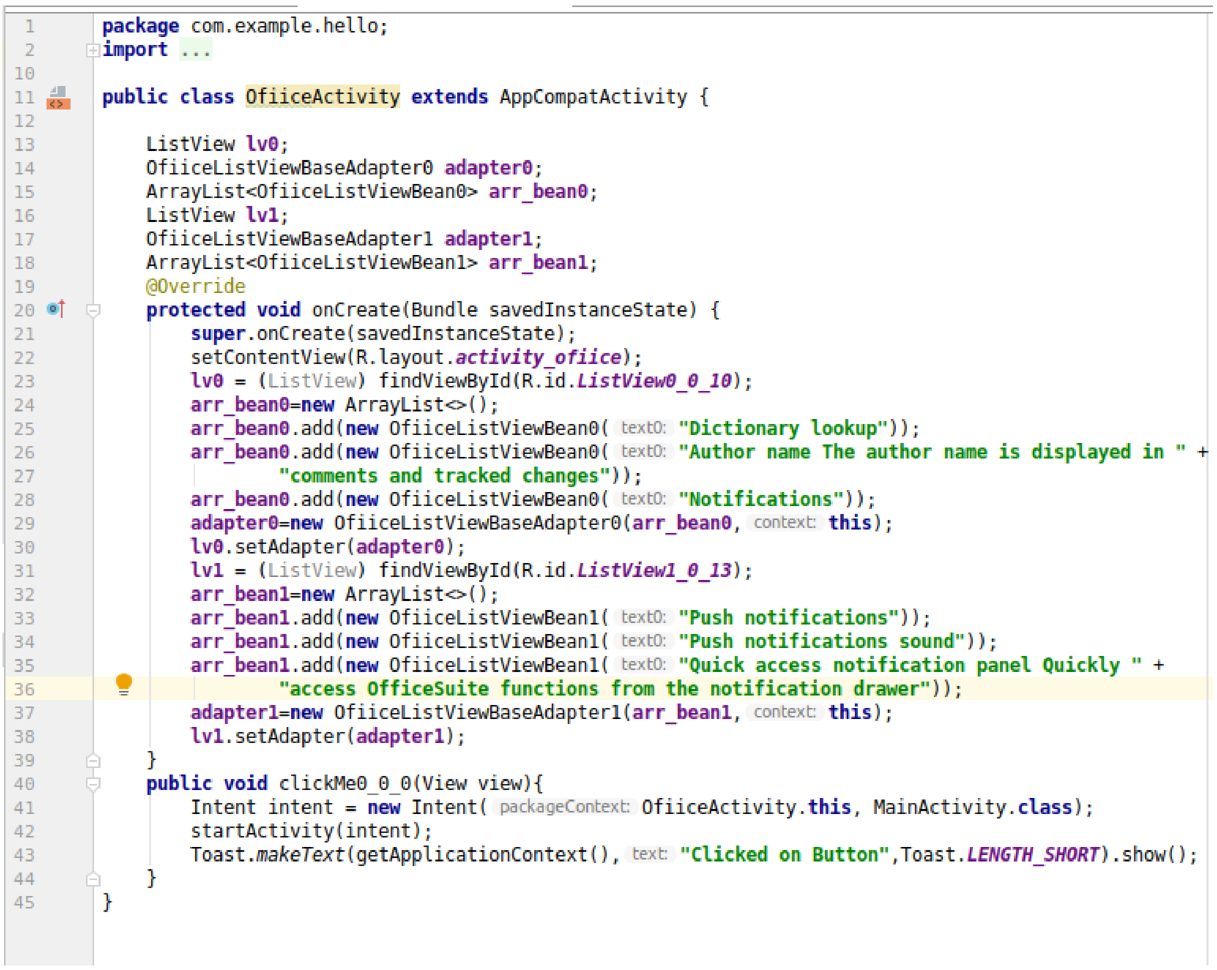
```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/
    xmlns:app = "http://schemas.android.com/apk/res-auto"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical"
    tools:context = ".FilesActivity">
    <ListView
        android:id = "@+id/ListView0_0_1"
        android:padding="5dp"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        >
    </ListView>
    <LinearLayout
        android:orientation = "horizontal"
        android:background = "#f5f5f5"
        android:layout_width = "match_parent"
        android:layout_height = "0dp"
        android:layout_weight = "1"
        android:gravity = "right"
        >
        <TextView
            android:id = "@+id/TextView_0_9_0"
            android:padding="5dp"
            android:layout_height = "wrap_content"
            android:layout_width = "wrap_content"
            android:text = "CLOSE"
            android:textColor = "#616161"
            android:background = "#f5f5f5"
            >
        </TextView>
    </LinearLayout>
</LinearLayout>
```

Figure 46. A ListView rendered in Android Studo



```
        android:layout_height = "0dp"
        android:layout_weight = "1"
        >
        <TextView
            android:id = "@+id/TextView_0_4_0"
            android:padding="5dp"
            android:layout_height = "wrap_content"
            android:layout_width = "wrap_content"
            android:text = "Radio buttons"
            android:textColor = "#252525"
            android:background = "#ffffff"
            >
        </TextView>
    </LinearLayout>
    <RadioGroup
        android:id = "@+id/RadioGroup_0_5"
        android:padding="5dp"
        android:layout_width = "match_parent"
        android:layout_height = "0dp"
        android:layout_weight = "2"
        >
        <LinearLayout
            android:orientation = "horizontal"
            android:background = "#ffffff"
            android:layout_width = "match_parent"
            android:layout_height = "0dp"
            android:layout_weight = "1"
            >
            <RadioButton
                android:id = "@+id/RadioButton_0_5_0"
                android:padding="5dp"
                android:layout_height = "wrap_content"
                android:layout_width = "0dp"
                android:layout_weight = "1"
                >
            </RadioButton>
            <TextView
                android:id = "@+id/TextView_0_5_1"
                android:padding="5dp"
                android:layout_height = "wrap_content"
                android:layout_width = "0dp"
                android:layout_weight = "5"
                android:text = "Line item selected"
```

Figure 47. The app with the radio buttons rendered on Android Studio



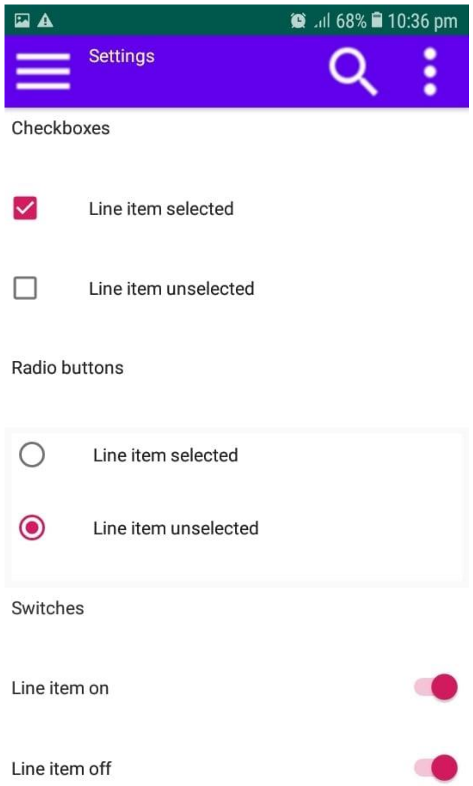
```java
package com.example.hello;
import ...

public class OfiiceActivity extends AppCompatActivity {

    ListView lv0;
    OfiiceListViewBaseAdapter0 adapter0;
    ArrayList<OfiiceListViewBean0> arr_bean0;
    ListView lv1;
    OfiiceListViewBaseAdapter1 adapter1;
    ArrayList<OfiiceListViewBean1> arr_bean1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ofiice);
        lv0 = (ListView) findViewById(R.id.ListView0_0_10);
        arr_bean0=new ArrayList<>();
        arr_bean0.add(new OfiiceListViewBean0( text0: "Dictionary lookup"));
        arr_bean0.add(new OfiiceListViewBean0( text0: "Author name The author name is displayed in " +
                "comments and tracked changes"));
        arr_bean0.add(new OfiiceListViewBean0( text0: "Notifications"));
        adapter0=new OfiiceListViewBaseAdapter0(arr_bean0, context: this);
        lv0.setAdapter(adapter0);
        lv1 = (ListView) findViewById(R.id.ListView1_0_13);
        arr_bean1=new ArrayList<>();
        arr_bean1.add(new OfiiceListViewBean1( text0: "Push notifications"));
        arr_bean1.add(new OfiiceListViewBean1( text0: "Push notifications sound"));
        arr_bean1.add(new OfiiceListViewBean1( text0: "Quick access notification panel Quickly " +
                "access OfficeSuite functions from the notification drawer"));
        adapter1=new OfiiceListViewBaseAdapter1(arr_bean1, context: this);
        lv1.setAdapter(adapter1);
    }
    public void clickMe0_0_0(View view){
        Intent intent = new Intent( packageContext: OfiiceActivity.this, MainActivity.class);
        startActivity(intent);
        Toast.makeText(getApplicationContext(), text: "Clicked on Button",Toast.LENGTH_SHORT).show();
    }
}
```

Figure 48. A sample of the Java code showing the activities switching

The following figures are screenshots taken from our mobile after running the Android project and switching between activities.



Settings

Checkboxes

Line item selected

Line item unselected

Radio buttons

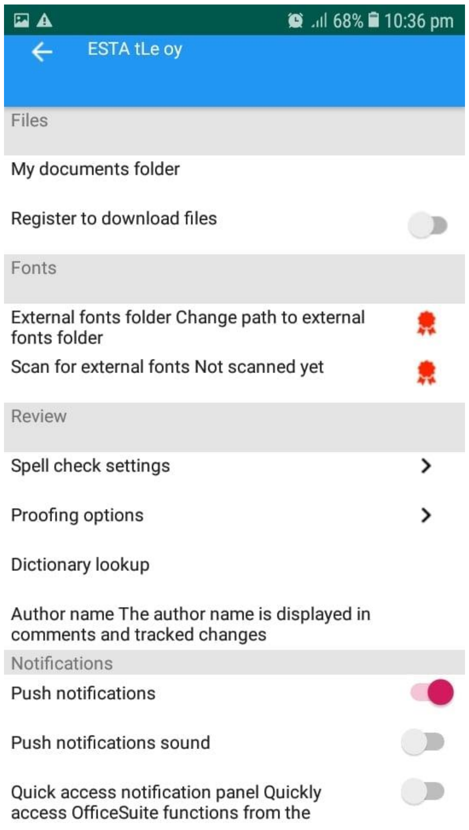
Line item selected

Line item unselected

Switches

Line item on

Line item off



ESTA tLe oy

Files

My documents folder

Register to download files

Fonts

External fonts folder Change path to external fonts folder

Scan for external fonts Not scanned yet

Review

Spell check settings

Proofing options

Dictionary lookup

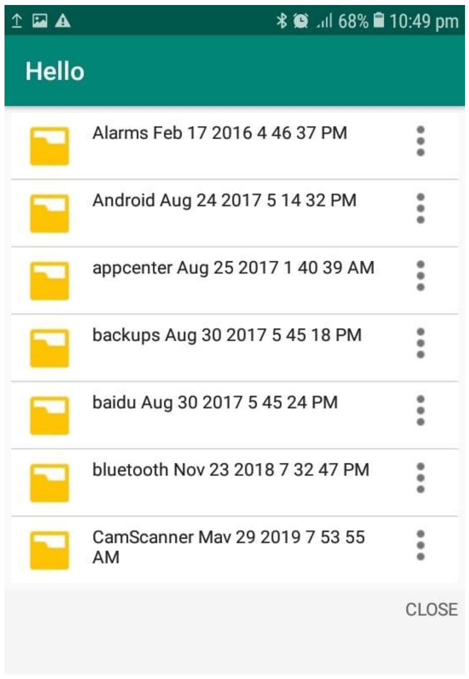
Author name The author name is displayed in comments and tracked changes

Notifications

Push notifications

Push notifications sound

Quick access notification panel Quickly access OfficeSuite functions from the



Hello

Alarms Feb 17 2016 4 46 37 PM

Android Aug 24 2017 5 14 32 PM

appcenter Aug 25 2017 1 40 39 AM

backups Aug 30 2017 5 45 18 PM

baidu Aug 30 2017 5 45 24 PM

bluetooth Nov 23 2018 7 32 47 PM

CamScanner Mav 29 2019 7 53 55 AM

CLOSE

Figure 49. Screenshots of the applications run on our model,

5.2.3.1. ***Hand-drawn mode***

The results of the hand-drawn mode is shown in the figures below.

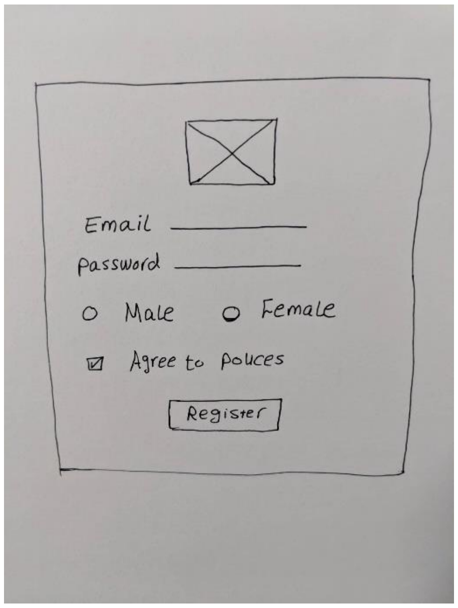


Figure 50. A sample of a hand-drawn design containing all the components we could predict

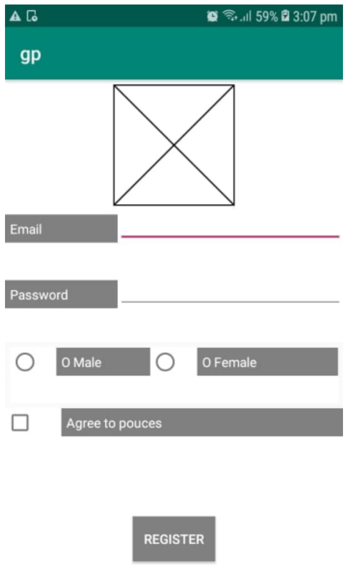
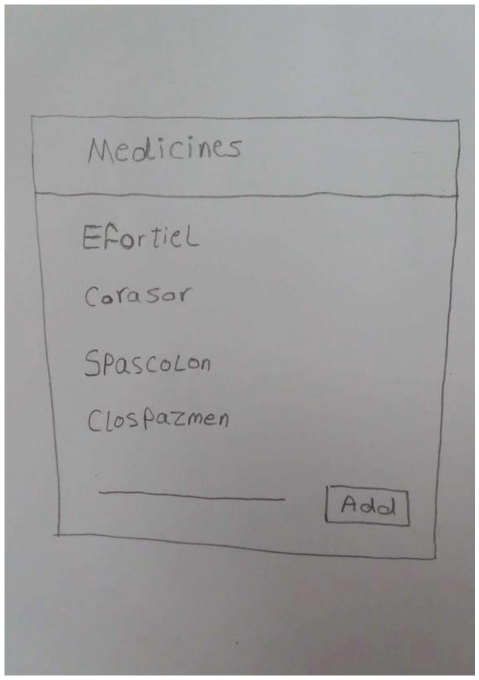


Figure 51. A screenshot taken from our mobile after running



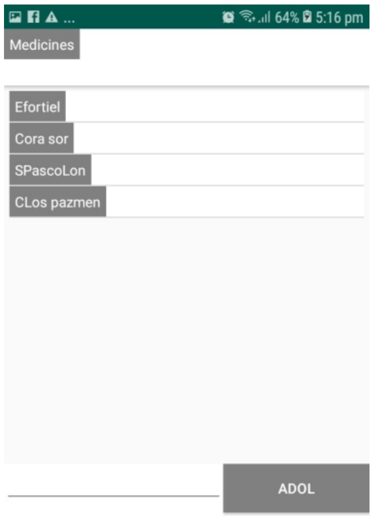
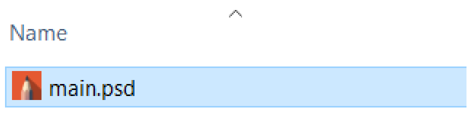
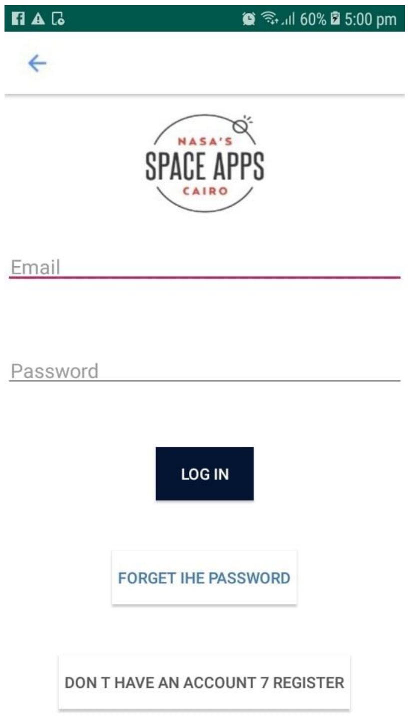


Figure 53. A screenshot taken from the app.

Figure 52. A sample of a hand-drawn design containing a ListView and ActionBar

5.2.3.2. PSD mode





6. Tools Used

In this chapter we present the tools and libraries we used to help us build our project.

6.1. Overview

This chapter presents the tools and programs that are used in the project. It explains briefly the software tools and their importance in the project.

6.2. System Tools and Libraries

6.2.1. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was designed for computational efficiency and with a strong focus on real-time applications.

We used OpenCV mainly in the pre-processing of the images of our datasets.

6.2.2. TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization.

6.2.3. Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is the key to doing good research. It is a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.

We used Keras for building, training and validating our NNs.

6.2.4. Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

6.2.5. Python-tesseract

Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and “read” the text embedded in images.
Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine. It is also useful as a stand-alone invocation script to tesseract, as it can read all image types supported by the Python Imaging Library, including jpeg, png, gif, bmp, tiff, and others, whereas tesseract-ocr by default only supports tiff and bmp.

We used python-tesseract to recognize the text in the components extracted from the design, e.g. text in TextView and Button components.

6.2.6. Python-colormath

Python-colormath is a simple Python module that spares the user from directly dealing with color math. Some features include:

- Support for a wide range of color spaces. A good chunk of the CIE spaces, RGB, HSL/HSV, and many more.
- Conversions between the various color spaces. For example, XYZ to sRGB, Spectral to XYZ, CIE Lab to Adobe RGB.
- Calculation of color difference. All CIE Delta E functions, plus CMC. Chromatic adaptations (changing illuminants). RGB to hex and vice-versa.

We used python-colormath to decide the colors of the background and the components using CIE Delta E functions.

6.2.7. Psd-tools

Psd-tools is a Python package for working with Adobe Photoshop PSD files. The library supports:

- Read and write of the low-level PSD/PSB file structure.
- Raw layer image export.
- We used it to extract PSD files' layers to be recognized and processed.

6.2.8. Google Cloud Vision API

Google Cloud's Vision API offers powerful pre-trained machine learning models through REST API. Assign labels to images and quickly classify them into millions of predefined categories. Read printed and handwritten text, and build valuable metadata into your image catalog. We used it to detect and recognize the hand written text in hand-drawn designs.

6.2.9. Android Studio

Android Studio is the official IDE for Android development, and includes everything you need to build Android apps.

We used it to render our XML, and compile our generated Java within an Android studio project to be build mobile devices.

6.2.10. PyQt5

PyQt is a Python bindings for the Qt cross-platform C++ framework.Qt classes employ a signal/slot mechanism for communicating between objects that is type safe but loosely coupled making it easy to create reusable software components. PyQt combines all the advantages of Qt and Python. A programmer has all the power of Qt, but is able to exploit it with the simplicity of Python.

We used PyQt to build our GUI structure and communicate between its classes.

6.3. Programming Languages

Python 3.6

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

We used python and numpy, Scipy libraries in the preprocessing of the images of datasets before the training phase, and used with opencv for feature extraction. Python is also used along with Tensorflow library for the testing and prediction phases of the deep learning models and computing final accuracy.

6.4. Hardware tools

Colaboratory Google

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

It allows the user to start working directly on a free Tesla K80 GPU using Keras, Tensorflow and PyTorch. This is provided in a free 12-hours sessions.

We used the previously mentioned sessions to train, validate and test our model. Also in calculating the mean and the standard deviation for some of the extracted features on the dataset for standardization purposes.

7. Conclusion and Future work

This chapter briefly concludes the work done developing the application and presents the plans for its future expansion.

7.1. *Conclusion*

As a program is always working on solutions to make peoples' lives easier and/ or more entertaining. However, Tagit aims to make those programmers' lives easier and more productive.

Our solution is open source technology-based which makes it affordable. Tagit has presented an approach to take any screenshot for android design, hand-drawn image or PSD file and generate the corresponding XML files and Java code with basic functionality which are ready to run on Android Studio. We generate the hierarchy of the XML in a weighted layout form which makes the application compatible with any screen size. We also handle the colors of the background and the text appearing in the design, so we finally have a layout which is as near as possible to the given design.

7.2. *Future Work*

We can collect a dataset for websites and IOS components and try to apply our approach on it.

References

[1] From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation
[2] Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps
[3] pix2code: Generating Code from a Graphical User Interface Screenshot

Appendices

Appendix A: User Manual

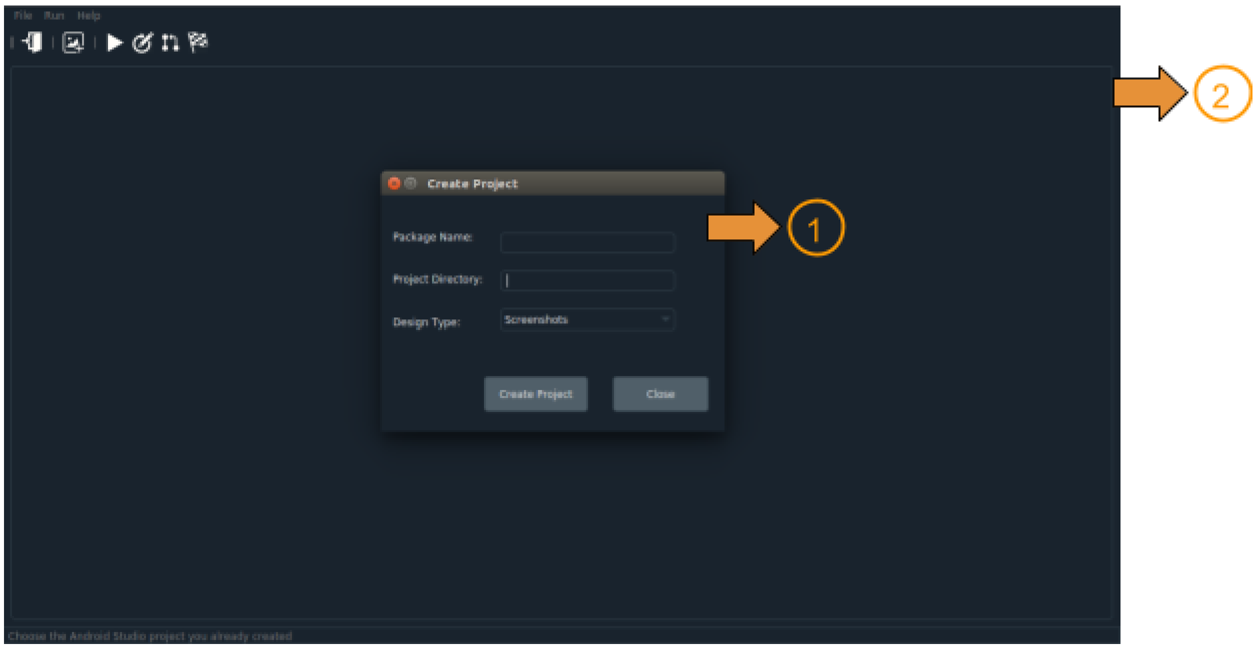
Overview

Create a new project.
Main screen -Upload Images.
Add Screenshots / Hand Drawn / PSD files.
Edit Images Details.
Generate XML files.
Main Screen -Update wrong-predicted components.
Main Screen -Connect buttons to existing activities.
Finish.

Screen & Actions Description

Create a new project:

After running Tagit application you will see two views as shown in figure 54. The first view is the Create a new project, the second view is the Main Screen.



Choose the Android Studio project you already created

Figure 54. The welcome window

To create a new Project: As shown in Fig 55

1. The Package Name: Choose the Android Studio Package name.
2. Project Directory: Choose the Project Directory that you previously created using Android Studio, This is the path for the created XML and Java files that our App will create.
3. Design Type: Choose from three Designs that we support:
    1. Screenshots:
       Create a project that will run on Images exported from Photoshop with extensions JPEG or PNG. In addition to images of your design with the same extensions JPEG or PNG.
    2. Hand Drawing: Create a project that will run on Hand drawing Images.
    3. PSD Files: Create a project that will run on PSD files.

4. Create Project:
   After editing the previously discussed parameters click Create Project button. The next step will be uploading the desired images to build your Application.

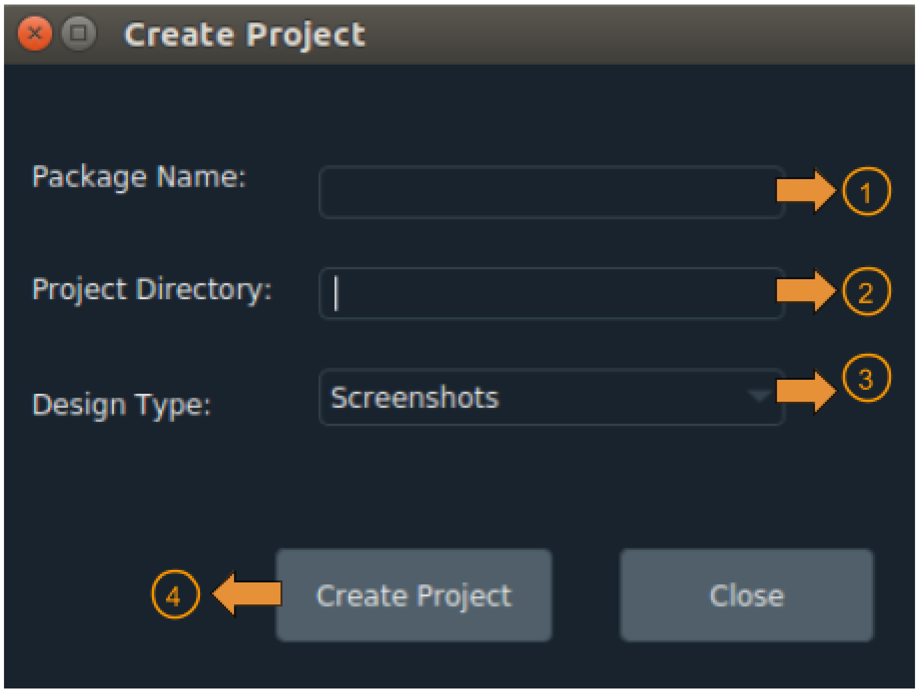


Figure 55. Create a new project

Main screen (Upload Images Screen):

As shown in Fig A.3 that shows the components of the Main Screen:

1. Add Images (Ctrl + A):
   Add images based on the Design you chose in the Create Project

(Screenshots, Hand Drawn, PSD files).

2. Generate XML and Java (shortcut: F5):
   After adding images and edit its details User can Generate XML and Java files to enter the next process (Update components).

3. Update components (shortcut: F6):
   (optional process) Update the wrong-predicted components to make sure that the components match the Application you desired.

4. Connect (shortcut:F7):
   (optional process) Connect Buttons or Image Buttons to other activity in order to add the functionality of that button in the java code and add the intended activity in the manifest file in your application. That will produce connected activities as a skeleton to start coding from.

5. Finish (shortcut: F8):
   Generate the connections if exists and your Android Application will be generated in the directory you set before in Create Project.

6. Images viewer:
   Images Viewer area that will display the images you will upload.

7. Status bar:
   A Status bar that shows messages for your progress of creating your Android Application.

8. Exit:
   That action will exit from the Application.

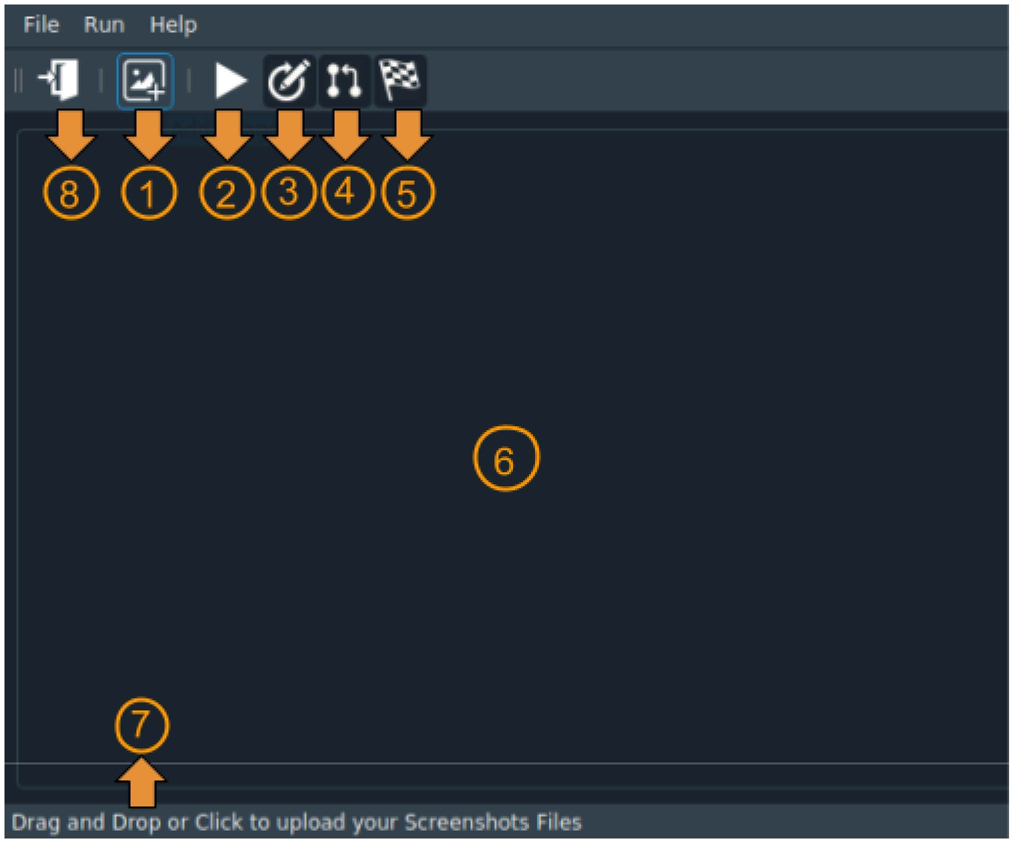


Figure 56. Main screen

Add Screenshots / Hand Drawn / PSD files

Add images based on the Design you chose in the Create Project (Screenshots, Hand Drawn, PSD files). You can achieve that process by 4 actions, as shown in figure 57:

1. Press Add Images button in the action bar.
2. Drag and drop Images in the Images viewer area.
3. From File -> Add Images and press Add Images.
4. Use shortcut Ctrl + A

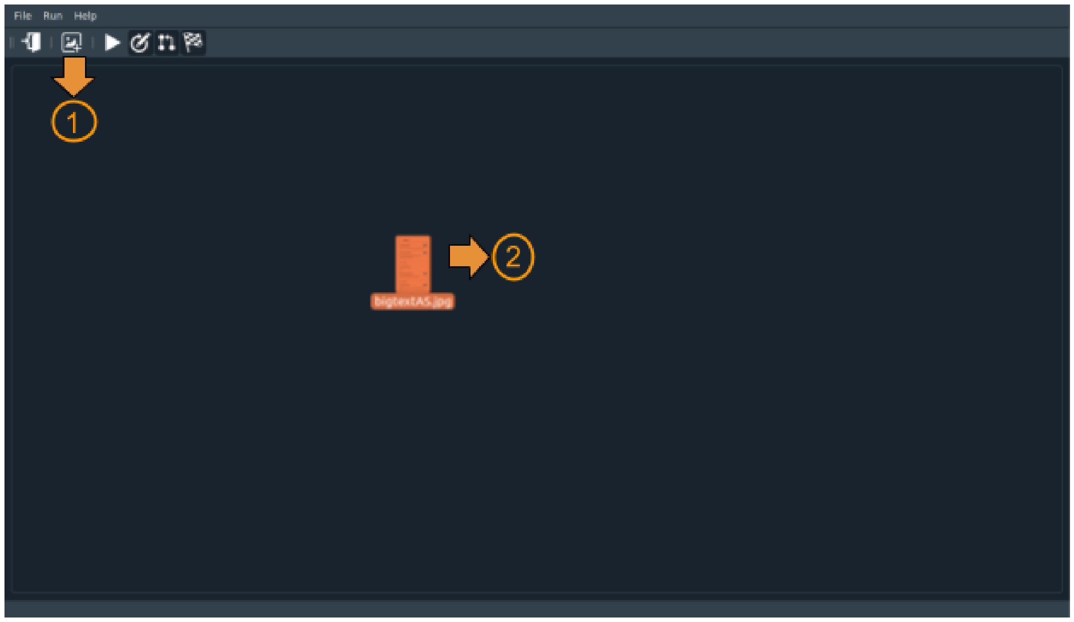


Figure 57 Add screenshots, hand-drawn or psd

Edit Images Details:

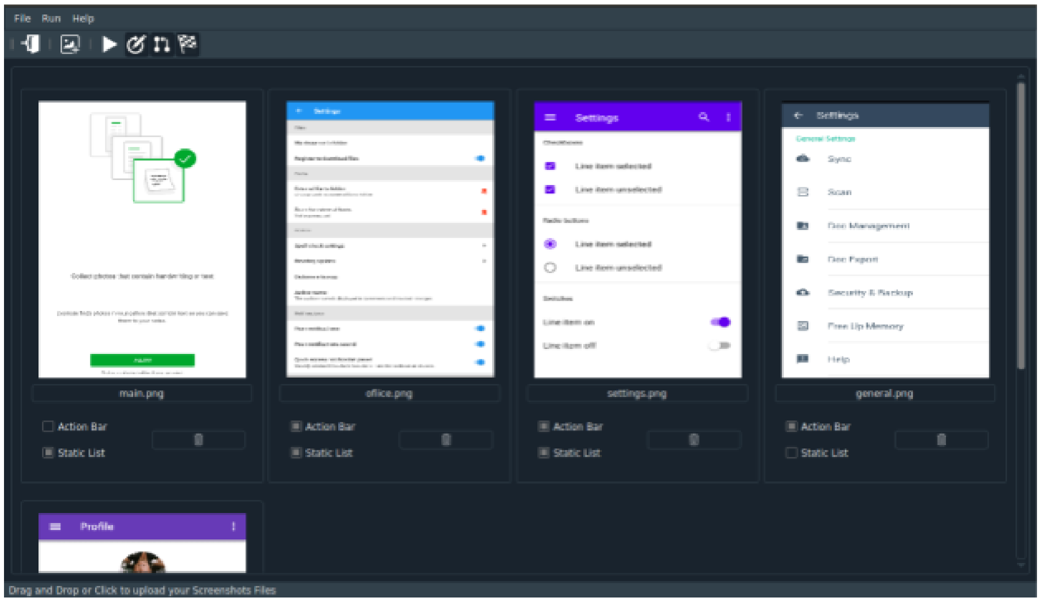
After adding images to the project the Main Screen will look like figure 58. Now it's time to edit images details to match the Android Application you desired. As shown in Fig A.6. that shows the image details with respect to the numbers that in Fig:

1. Image name:
   User can edit the Image name with the name he wants. User must use an image with the name main that will be the "main.extension" activity for the Android Application, or if there is no image with name "main" User can rename the image.
2. Action Bar:
   A checkbox user can check to Indicate That the Image has an action bar as shown in label 2 the image main has no action bar but at label 6 for the pointing image has.

3. Static List:
   A checkbox user can check if he/she doesn't want the Application to generate any dynamic ListViews -if exists- he can check the static list option. As shown in label 3 the User wants a static List in this activity but at label 7 the User wants a Dynamic ListView in this activity.

4. Delete Button:

   User can delete the image he wants to delete to not be in the project anymore.



main.png: ☐ Action Bar, ☑ Static List

office.png: ☑ Action Bar, ☑ Static List

settings.png: ☑ Action Bar, ☑ Static List

general.png: ☑ Action Bar, ☐ Static List

Figure 58. Main screen after adding images

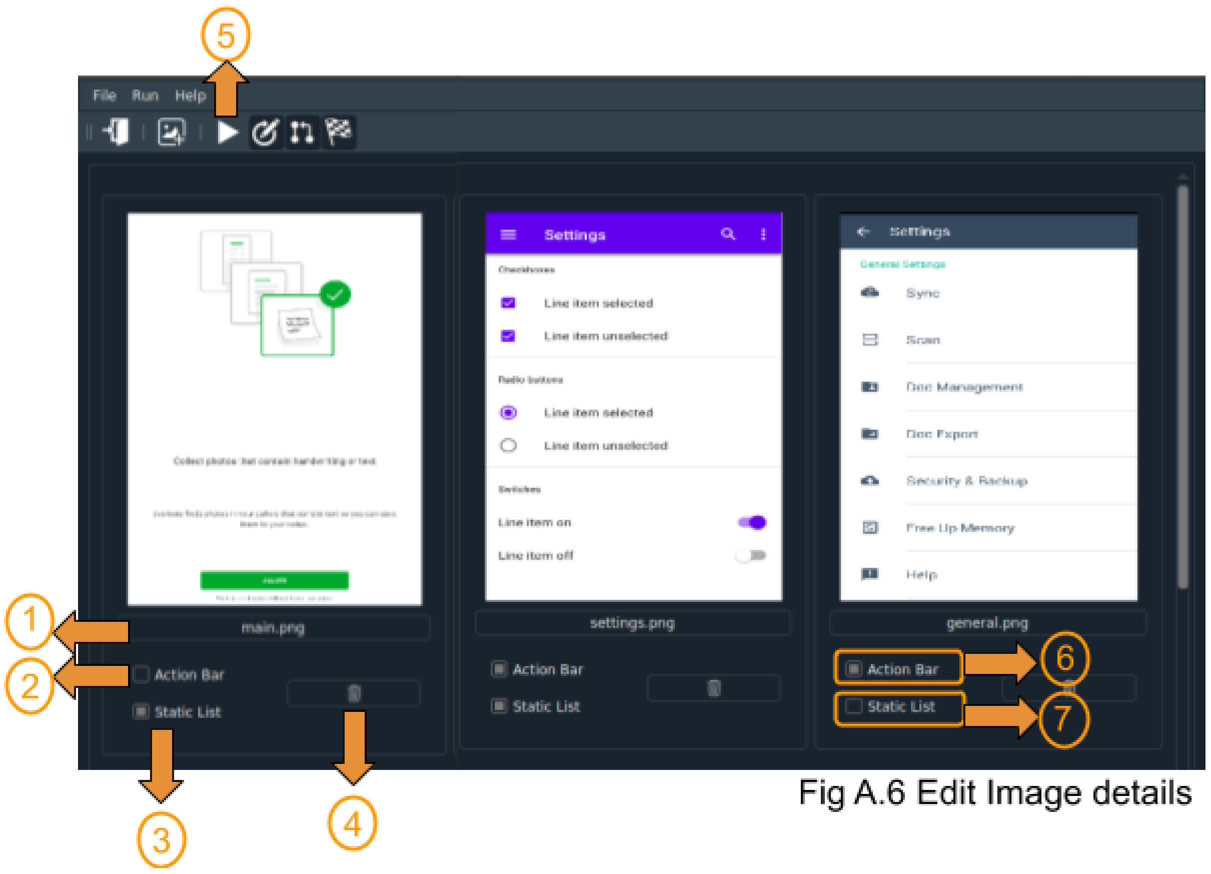


Figure 59. Edit images details

Generate XML files:

1. After Editing the Images and satisfy the constraints (images must have an image with name main, images must satisfy extensions described before), User can Generate the XML files with 3 different ways:

   1. Click on Generate XML icon in the toolbar (Fig A.6 label 5)
   2. From menu choose Run -> Generate XML.
   3. Use Short cut F5.

2. User can see "please wait" screen while generating XML and Java code as shown in Fig 60
3. After Generating XML the User will see a screen like Fig 61.
   The description of the screen as follows:
   1. Images List:
      A List view of the images that the User enters in the add images process

2. View button:
   A button for each image in the List of images. On click that button the image corresponding to that button is set to be the active image and User can make his/her changes as will be discussed later.
3. Active image:
   The Activity that User can see its components information and can change its component in the update state.
4. XML Taps:
   The Taps view that shows the XML files generated for the corresponding active image.
5. Component highlight:
   Every component in the Activity (active image) can be highlighted, once the user clicks on it it shows its information in two views the Component information and the XML Component information.
6. Component information:
   It shows what type that component is predicted and User can change its type if it is wrongly predicted.
7. XML Component information:
   The XML code that corresponds to the last clicked component that describes that will be part of the XML files in the Tap view.

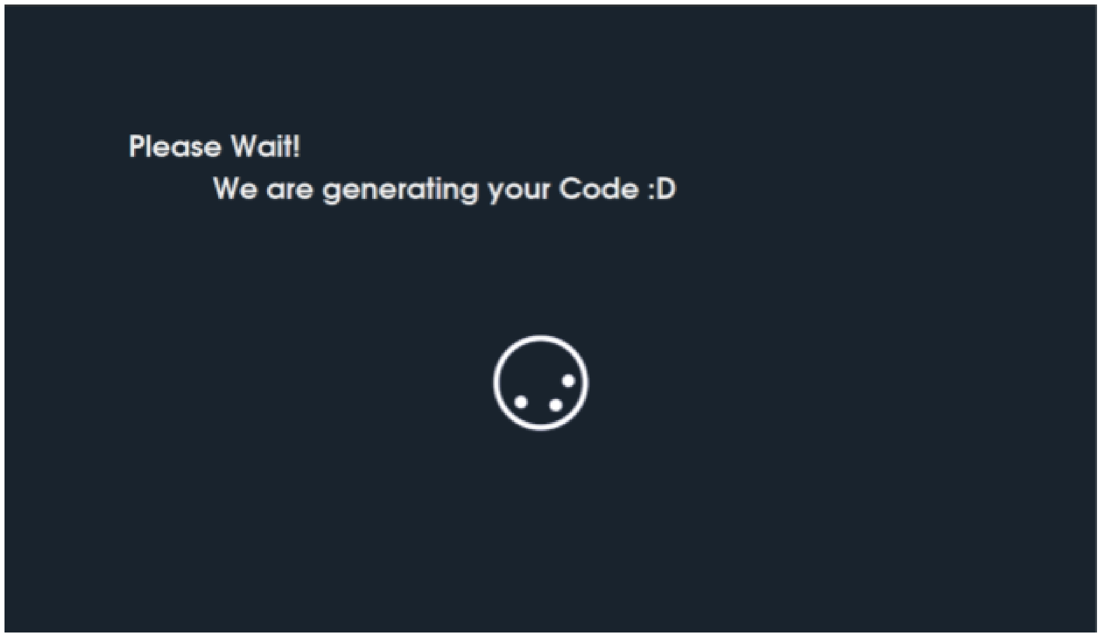
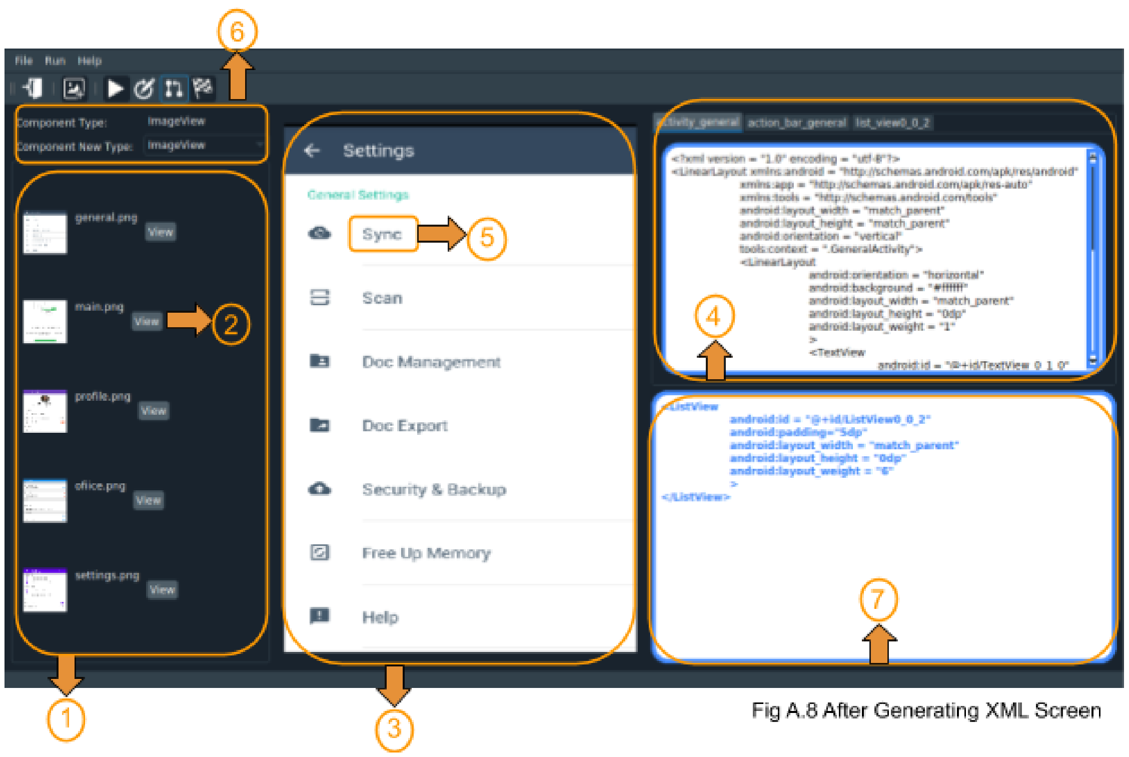

Please Wait!
We are generating your Code :D

Figure 60. Please wait screen



```
<?xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
          xmlns:app = "http://schemas.android.com/apk/res-auto"
          xmlns:tools = "http://schemas.android.com/tools"
          android:layout_width = "match_parent"
          android:layout_height = "match_parent"
          android:orientation = "vertical"
          tools:context = ".GeneralActivity">
          <LinearLayout
                    android:orientation = "horizontal"
                    android:background = "#ffffff"
                    android:layout_width = "match_parent"
                    android:layout_height = "0dp"
                    android:layout_weight = "1"
                    >
                    <TextView
                              android:id = "@+id/TextView_0_1_0"
```

```
<ListView
          android:id = "@+id/ListView0_0_2"
          android:padding="5dp"
          android:layout_width = "match_parent"
          android:layout_height = "0dp"
          android:layout_weight = "6"
          >
</ListView>
```

Figure 61. After Generating XML screen

Main Screen -Update wrong-predicted components:

User can update the wrongly predicted components in order to make his/her Application as desired. As shown in Fig A.9 here are the steps for updating a component(s):

1. Choose the Active Image:
   User has to click on the view button to set the activity -that holds the component the user wants to change- to be the active Image.

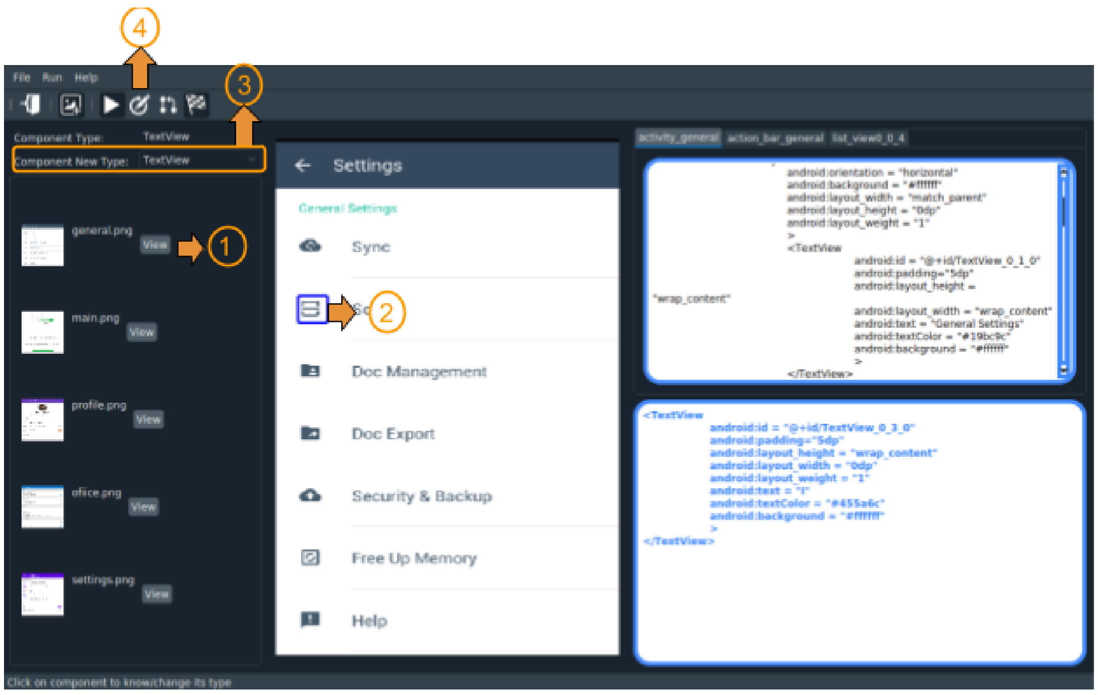
2. Choose the component:
   Choose the component that User want to change its type.

3. Change the component Type:
   By clicking on the combo box that at label 3 we can change the new type of the component and that will save the changes locally and needs to the final step (Update).

4. Update Components:
   Update Components that update the component(s) the user made from the previously clicked Update Components or for the first time after XML generation. That will change the Component(s) the user wants to change in the XML and Java codes and show the new results again by clicking on the components again. It is an optional stage as user can

skip that stage and move to the connection stage. User can run the Update Components with three different ways:

1. Click on Update Components icon in the toolbar (Fig 60 label 4)
2. From menu choose Run -> Update Components.
3. Use Shortcut F6



```
android:orientation = "horizontal"
android:background = "#ffffff"
android:layout_width = "match_parent"
android:layout_height = "0dp"
android:layout_weight = "1"
>
<TextView
        android:id = "@+id/TextView_0_1_0"
        android:padding="5dp"
        android:layout_height =
"wrap_content"
        android:layout_width = "wrap_content"
        android:text = "General Settings"
        android:textColor = "#19bc9c"
        android:background = "#ffffff"
        >
</TextView>
```

```
<TextView
        android:id = "@+id/TextView_0_3_0"
        android:padding="5dp"
        android:layout_height = "wrap_content"
        android:layout_width = "0dp"
        android:layout_weight = "1"
        android:text = "I"
        android:textColor = "#455a6c"
        android:background = "#ffffff"
        >
</TextView>
```

Figure 62. Update Components step

Main Screen -Connect buttons to existing activities

After finishing all updates components or skip. User can enter the connect state which is connecting the buttons to the existing activities. That will produce connected activities as a skeleton to start coding from. It is an optional state that the User can move to state finish without connecting any buttons.

1. User can enter the connect state with three different ways:
    1. Click on Connect Components icon in the toolbar (Fig 56 label 4)
    2. From menu choose Run -> Connect Components.

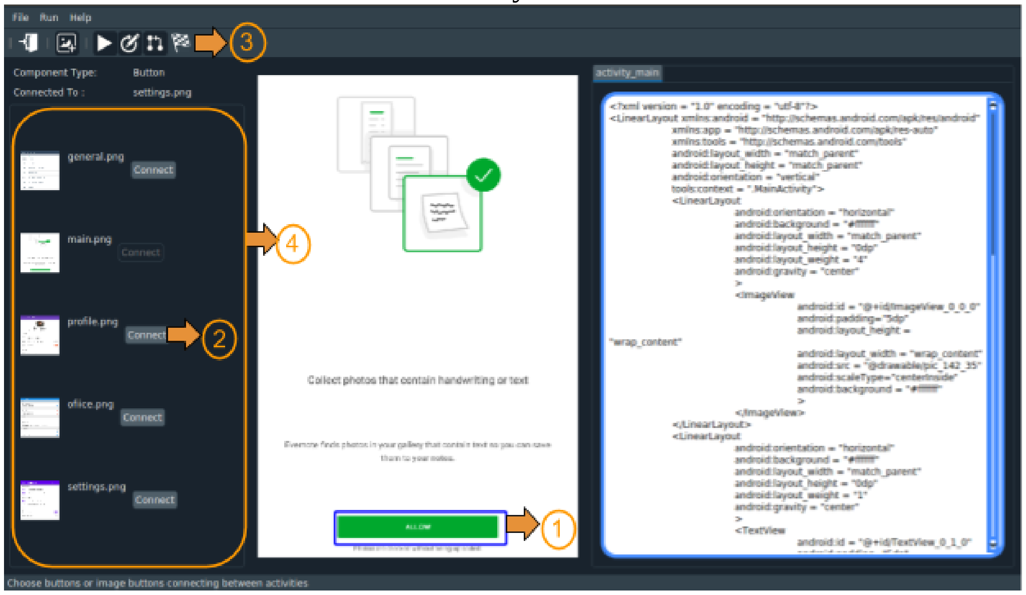
    3. Use Shortcut F7

2. As shown in Fig 63 here are the steps of connecting components to existing Activity:

    1. Click on the button or Image Button that User wants to connect with another activity.

    2. Choose the activity from the images list of label 4 and click on its corresponding connect button.

    3. All changes were saved locally and need the last Step (Finish) to save the actions made by the User.



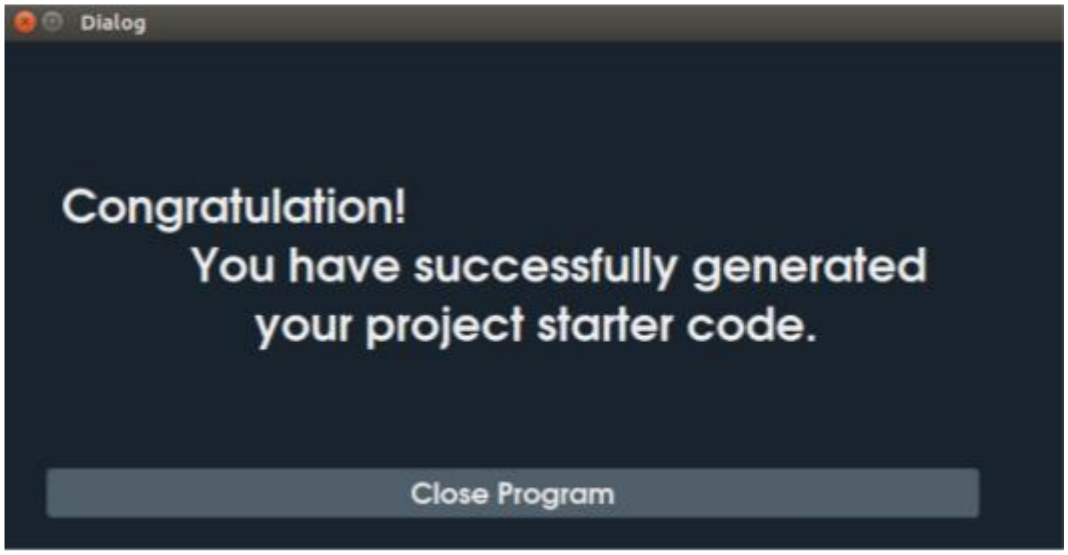
```
<?xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
              xmlns:app = "http://schemas.android.com/apk/res-auto"
              xmlns:tools = "http://schemas.android.com/tools"
              android:layout_width = "match_parent"
              android:layout_height = "match_parent"
              android:orientation = "vertical"
              tools:context = ".MainActivity">
              <LinearLayout
                        android:orientation = "horizontal"
                        android:background = "#ffffff"
                        android:layout_width = "match_parent"
                        android:layout_height = "0dp"
                        android:layout_weight = "4"
                        android:gravity = "center"
                        >
                        <ImageView
                                  android:id = "@+id/ImageView_0_0_0"
                                  android:padding="5dp"
                                  android:layout_height =
"wrap_content"
                                  android:layout_width = "wrap_content"
                                  android:src = "@drawable/pic_142_35"
                                  android:scaleType="centerInside"
                                  android:background = "#ffffff"
                                  >
                        </ImageView>
              </LinearLayout>
              <LinearLayout
                        android:orientation = "horizontal"
                        android:background = "#ffffff"
                        android:layout_width = "match_parent"
                        android:layout_height = "0dp"
                        android:layout_weight = "1"
                        android:gravity = "center"
                        >
                        <TextView
                                  android:id = "@+id/TextView_0_1_0"
```

Figure 63. Connect Component window

Finish

Finish will save the last changes of connecting components -if exists- and output the final XML and Java codes in the directory specified previously in Create a new project. So the user can run his/her Android Application from Android Studio without copying files. The output of this process will be as Fig A.11 and then the Application will close.
Finish Action can be achieved in three ways:

1. Click on Finish icon in the toolbar (Fig A.3 label 5)
2. From menu choose Run -> Finish
3. Use Shortcut F8


Dialog

Congratulation!
You have successfully generated
your project starter code.

Figure 64 Finish Window

Appendix B: Additional Trials

Appendix B: Additional Trials

This appendix discusses the trials we made to achieve certain goals but failed. This resulted in not using those methods.

A different approach using Neural Machine Translation.

We first tried the approach of UI2code which applies the CNN on the image, then the output is converted into a sequence of D-dimensional feature vectors (D is the depth of the feature vector). The length of this image-based sequence is W × H (the width and height of the CNN output). Then the RNN encoder-decoder model is applied to this image-based sequence to translate the image to a corresponding hierarchy structure. The result was not satisfying. We thought that it is better to train the model on the components extracted from the image instead of trying to translate the full image. So we adopted the approach described in this document. We made it simpler for the model to learn and we made additional filtration and built an algorithm to construct the hierarchy structure.

Hand-Drawn Text Extraction Trials

As for the hand-drawn mode we needed to extract the hand written text for multiple purposes (e.g. recognize text within a Button, TextView). To achieve this, we have tried:

1. Simple HTR:

   Handwritten Text Recognition (HTR) system implemented with TensorFlow and trained on the IAM (Handwriting Database contains forms of handwritten English text which can be used to train and test handwritten text recognizers[25]) off-line HTR dataset. This Neural Network model recognizes the text contained in the images of segmented words.[26]

   This pre-trained model input should have the input images with higher resolution than the images given by Tagit user. Hence the model couldn't recognize the text in the input images of Tagit.

2. Keras MNIST

   The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples[27], and a test set of 10,000 examples. Trains a simple ConvNet on the MNIST dataset[28].

   Using this pre-trained model to obtain the text in the user's hand drawn didn't hand in better results. The output wasn't accurate and some of the shapes were being detected as letters. For that reason we couldn't use this model.

Arabic Summary

إن الهدف من مشروعنا توفير الوقت والمجهود على مبرمجى تطبيقات الأندرويد ؛ فهم يقضون أوقاتا طويلة في تحويل التصميم المعطى إلى الكود المناسب. عن طريق مشروعنا يمكن للمبرمج إدخال صورة لأى تصميم مراد تحويله، او صورة مرسومة باليد للتصميم، او ملف مستخرج من الفوتوشوب وسنقوم بتوفير جميع ملفات الكود المطلوبة في صورة جاهزة للعمل على برنامج اندرويد استوديو. ولتحقيق ذلك الهدف فقد اتبعنا الطريقة القائمة على استخراج العناصر المكونة للصورة والتعرف علي كل منها باستخدام الموديل و فلترتها للحصول على المكونات الاساسية فى الصورة، ومن ثم وضعها بالشكل المناسب للحصول على الهيكل المكون للصورة. كما اننا نقوم بطباعة الكود اللازم لأداء الوظائف الأساسية بالتطبيق كالضغط على اى زر و ملىء اى قوائم وجدت فى التصميم. إننا نوفر ايضا للمستخدم إمكانية التعديل بعد الحصول على الناتج ، عن طريق الضغط على العنصر المكون المراد تحويله وتغيير نوعه ومن ثم الضغط على زر تعديل لنقوم بتعديل نوع العنصر المكون و إعادة إنتاج الكود من جديد.
و بإمكانية المستخدم أيضا ربط صفحات التطبيق المختلفة بالضغط على أى زر بأى صفحة واختيار الصفحة المراد الانتقال اليها وبعد اختيار جميع الانتقالات المطلوبة والضغط على زر إنهاء فإننا نقوم بطباعة الكود المطلوب لأداء تلك المهمة.

---

[25] IAM Handwriting Database. (n.d.). Retrieved from http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

[26] Githubharald. (2019, May 25). Githubharald/SimpleHTR. Retrieved from https://github.com/githubharald/SimpleHTR

[27] THE MNIST DATABASE. (n.d.). Retrieved from http://yann.lecun.com/exdb/mnist/

[28] EN10. (2018, March 12). EN10/KerasMNIST. Retrieved from https://github.com/EN10/KerasMNIST?fbclid=IwAR1nxaotY6BEHxAyZ5STgLQpnM943cWcJD0MsA3a5fMQmdu0H1Q4Xn7GWls