MOBILE ML INTERFACE DESIGNER

[1] Mrs. K.V.Jhansi Rani, M.Tech, Assistant Professor, Department of CSE, Eluru College of Engineering And Technology, Duggirala, Andhra Pradesh-534004.
[2] N.V.V.N.S.Pardhu, B.Tech, Department of CSE, Eluru College of Engineering And Technology, Duggirala, Andhra Pradesh-534004.
[3] P.Akhil, B.Tech, Department of CSE, Eluru College of Engineering And Technology, Duggirala, Andhra Pradesh-534004.
[4] P.Chandrasekhar, B.Tech, Department of CSE, Eluru College of Engineering And Technology, Duggirala, Andhra Pradesh-534004.
[5] M.Yamini, B.Tech, Department of CSE, Eluru College of Engineering And Technology, Duggirala, Andhra Pradesh-534004.

Abstract: This project presents an application for predicting JSON code from GUI screens using machine learning. We upload the RICO dataset after it can preprocess images by resizing and normalizing pixel values. The dataset is then shuffled and split into 80% training and 20% testing sets. A Convolutional Neural Network (CNN) ,Long Short-Term Memory (LSTM) algorithms are trained and achieving 95% accuracy. A training graph visualizes CNN ,LSTM progress. Users upload GUI screens to get code prediction, with results presented in JSON format.

1. INTRODUCTION

Most modern user-facing software applications are GUI-centric, and rely on attractive user interfaces (UI) and intuitive user experiences (UX) to attract customers, facilitate the effective completion of computing tasks, and engage users. Software with cumbersome or aesthetically displeasing UIs are far less likely to succeed, particularly as companies look to differentiate their applications from competitors with similar functionality. This phenomena can be readily observed in mobile application marketplaces such as the App Store , or Google Play where many competing applications (also known as apps) offering similar functionality (e.g., task managers, weather apps) largely distinguish themselves via UI/UX Thus, an important step in developing any GUI-based application is drafting and prototyping design mock-ups, which facilitates the instantiation and experimentation of UIs in order to evaluate or prove-out abstract design concepts. In industrial settings with larger teams, this process is typically carried out by dedicated designers who hold domain specific expertise in crafting attractive, intuitive GUIs using image editing software such as Photoshop or Sketch These teams are often responsible for expressing a coherent design language across the many facets of a company's digital presence, including websites, software applications and digital marketing materials. Some components of this design process also tend to carry over to smaller independent development teams who practice design or prototyping processes by creating wireframes or mock-ups to judge design ideas before committing to spending development resources implementing them. After these initial design drafts are created it is critical that they are faithfully translated into code in order for the end-user to experience the design and user interface in its intended form.

2. LITERATURE SURVEY

Shuffling and splitting datasets are crucial steps in machine learning model training. Techniques such as random shuffling and stratified splitting can ensure data randomness and balance in train-test splits. Furthermore, dataset normalization, including mean normalization and feature scaling, can enhance model convergence and performance by LeCun et al 1998. Preprocessing the dataset involves tasks such as resizing, normalization, and cleaning of data. Techniques like image resizing and normalization can be performed using popular libraries such as OpenCV or Pillow in Python . Moreover, data cleaning methods like noise removal and outlier detection may be applied to enhance dataset quality by Rousseeuw et al 1999. Visualization of CNN training progress can aid in understanding the model's learning behavior. Tools like Matplotlib in Python can be used to plot training accuracy and loss graphs during the training process Hunter 2007. Visual inspection of these graphs helps in monitoring model convergence and identifying overfitting or underfitting issues. Convolutional Neural Networks (CNNs) are widely used for image classification tasks. Training CNNs involves forward and backward passes, utilizing optimization algorithms like stochastic gradient descent (SGD) or its variants. Evaluation metrics such as accuracy, precision, and confusion matrices are employed to assess the model's performance by Sokolova and Lapalme 2009. Transfer learning, where a pre-trained CNN model is fine-tuned for a specific task, can significantly reduce training time and resource requirements. Pre-trained models like VGG, ResNet, or MobileNet can be adapted to new tasks by retraining only the final layers or a few selected layers by Pan and Yang 2010. Transfer learning has shown promising results in various image-related tasks. Effective hyperparameter tuning is essential for optimizing the performance of CNN models. Techniques such as grid search, random search, or Bayesian optimization can be employed to find the best combination of hyperparameters such as learning rate, batch size, and optimizer settings by Bergstra and Bengio 2012. 3 Predicting JSON code from images involves utilizing CNNs for image recognition tasks. Techniques such as transfer learning, where pre-trained CNN models are fine-tuned for specific tasks, can be effective Yosinski et al 2014. Additionally, frameworks like TensorFlow or PyTorch provide easy-to-use APIs for deploying CNN models and making predictions. To upload the RICO dataset, various methods and tools can be employed. The use of Python libraries like requests or urllib for downloading data from URLs and zipfile for extracting files can be effective by Jones et al 2017. Additionally, cloud storage services like Google Drive or Dropbox APIs can also be utilized for seamless dataset uploading. Data augmentation is crucial for increasing the diversity of the training dataset, especially in image classification tasks. Techniques such as rotation, flipping, zooming, and cropping can be employed to generate additional training samples by Shorten and Khoshgoftaar 2019. Augmenting the dataset helps prevent overfitting and improves the generalization of the model. Graphical User Interfaces (GUIs) play a vital role in deploying machine learning models to end-users. Frameworks like Tkinter in Python or Electron in JavaScript can be utilized to develop user-friendly interfaces for model prediction applications by Sparks et al 2020. GUIs simplify the interaction between users and machine learning models, making them more accessible and usable.

3. EXISTING SYSTEM

In industrial mobile app development, mock-up artifacts typically come in the form of high fidelity images (with or without meta-data) created by designers using software such as Photoshop or Sketch. In this scenario, depending on design and development workflows, metadata containing information about the constituent parts of the mock-up images can be exported and parsed from these artifacts. Independent developers may also use screenshots of existing apps to prototype their own apps. In this scenario, in addition to screenshots of running applications, runtime GUI-information (such as the html DOM-tree of a web app or the GUI hierarchy of a mobile app) can be extracted to further aid in the prototyping process. However, this is typically not possible in the context of mock-up driven development (which our approach aims to support), as executable apps do not exist.

DISADVANTAGES:

• Limited Interactivity: Mock-up artifacts lack interactive elements, making it difficult to assess user interaction and flow accurately.

• Time-Consuming Process: Creating high-fidelity images using specialized software is time-consuming and can slow down development.

• Dependency on Designers: Reliance on designers to create mock-ups can lead to delays and communication issues.

• Limited Realism: Mock-ups often lack realism, making it hard to visualize the final product accurately.

• Difficulty in Prototyping Dynamic Elements: Extracting runtime GUI-information for prototyping dynamic elements is challenging and may not fully capture app behavior.

4. PROPOSED SYSTEM

In industrial mobile app development, mock-up artifacts typically come in the form of high fidelity images (with or without meta-data) created by designers using software such as Photoshop or Sketch. In this scenario, depending on design and development workflows, metadata containing information about the constituent parts of the mock-up images can be exported and parsed from these artifacts. Independent developers may also use screenshots of existing apps to prototype their own apps. In this scenario, in addition to screenshots of running applications, runtime GUI-information (such as the html DOM-tree of a web app or the GUI hierarchy of a mobile app) can be extracted to further aid in the prototyping process. However, this is typically not possible in the context of mock-up driven development (which our approach aims to support), as executable apps do not exist.

ADVANTAGES:

• Interactive Prototypes: Get more realistic user experiences with interactive elements, thanks to runtime GUI-information.

• Faster Development: Speed up prototyping with real-time feedback, making adjustments quicker and easier.

• Better Collaboration: Align prototypes closely with design visions, fostering collaboration between designers and developers.

• Realistic Previews: Provide stakeholders with more accurate previews of the final product.

• Dynamic Features: Add dynamic elements like animations for more engaging prototypes.

SYSTEM ARCHITECTURE9+

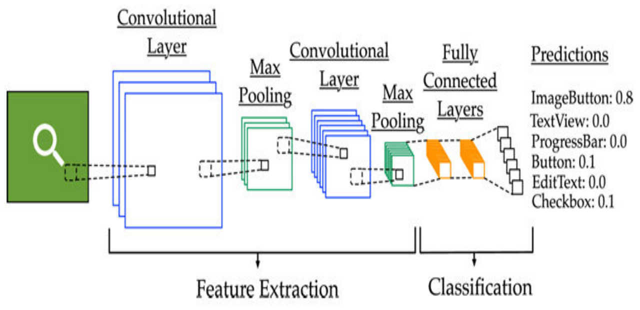


Fig1: System Architecture

5. UML DIAGRAMS

1. CLASS DIAGRAM

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application. Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages. It is also known as a structural diagram. Class diagram contains • Classes • Interfaces • Dependency, generalization and association.

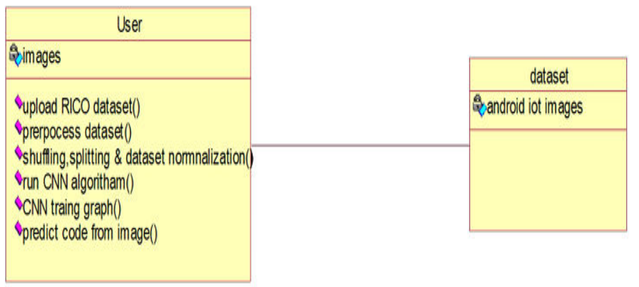


Fig 5.1 shows the class diagram of the project

2. USECASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted

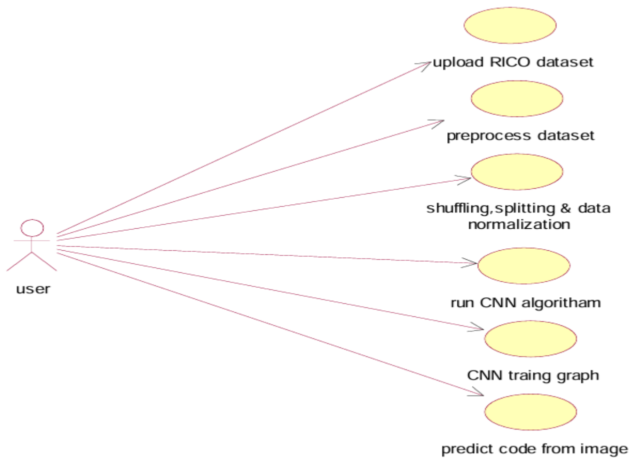


Fig 5.2 shows the Use case Diagram

3. SEQUENCE DIAGRAM:

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. Sequence diagrams are used to formalize the behavior of the system and to visualize the communication among objects. These are useful for identifying additional objects that participate in the use cases. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

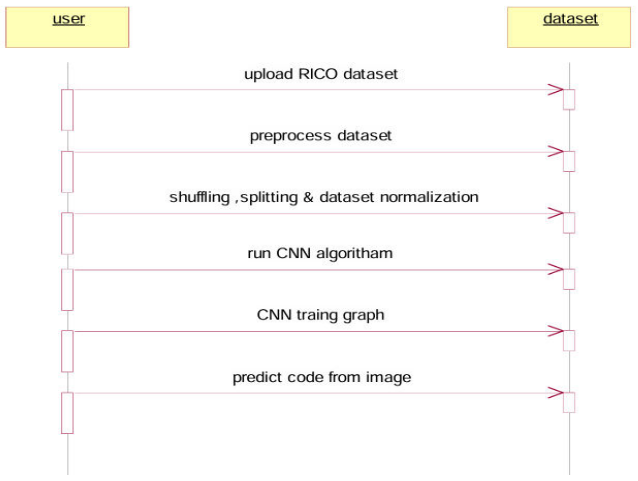


Fig 5.3 Shows the Sequence Diagram

6. RESULTS

6.1 Output Screens

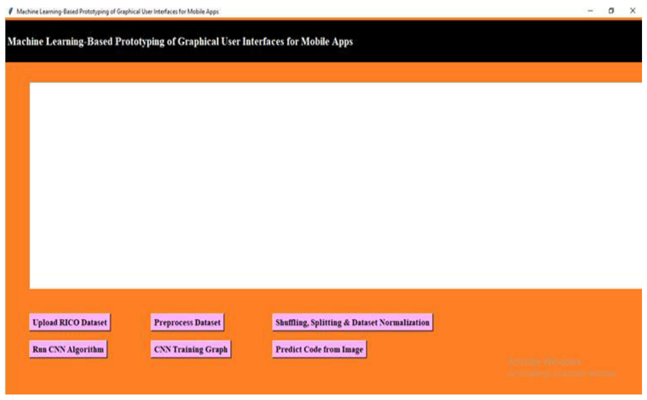

Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps
Upload RICO Dataset
Preprocess Dataset
Shuffling, Splitting & Dataset Normalization
Run CNN Algorithm
CNN Training Graph
Predict Code from Image

Fig 6.1 Home Page

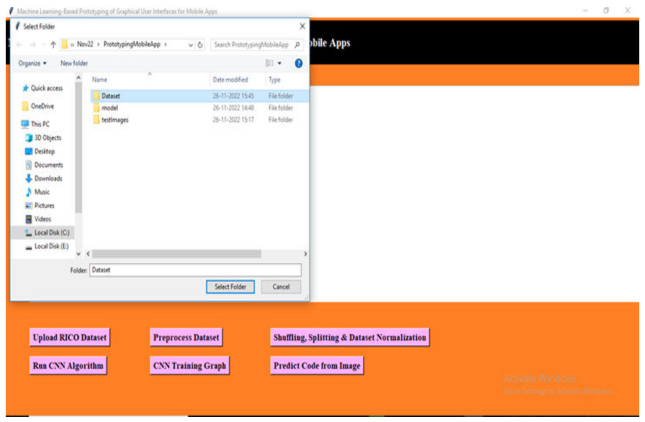
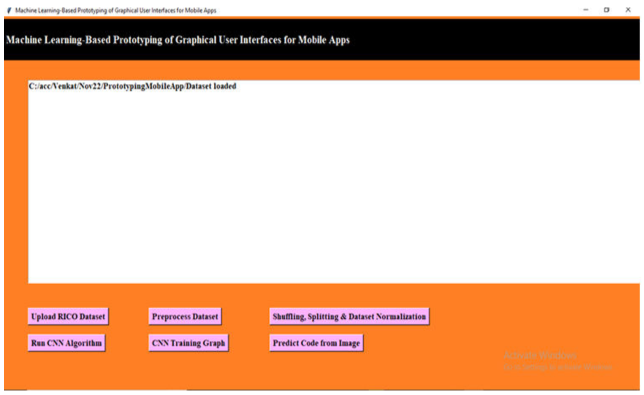
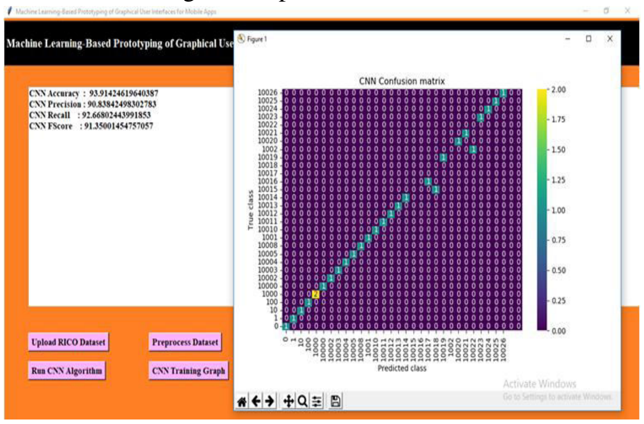


Fig 6.2 Upload the Dataset


Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps
Upload RICO Dataset
Preprocess Dataset
Shuffling, Splitting & Dataset Normalization
Run CNN Algorithm
CNN Training Graph
Predict Code from Image

```
C:/acc/Venkat/Nov22/PrototypingMobileApp/Dataset loaded
```

Fig 6.3.Preprocess the Dataset



```
CNN Accuracy  :  93.91424619640387
CNN Precision : 90.83842498302783
CNN Recall    : 92.66802443991853
CNN FScore    : 91.35001454757057
```

Fig 6.4 CNN Algorithm Accuracy

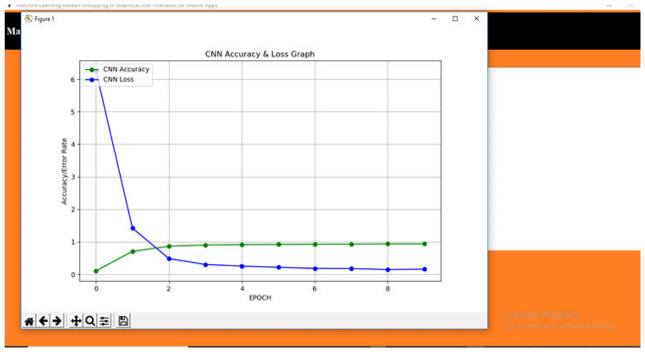


Fig 6.5 Performance Graph

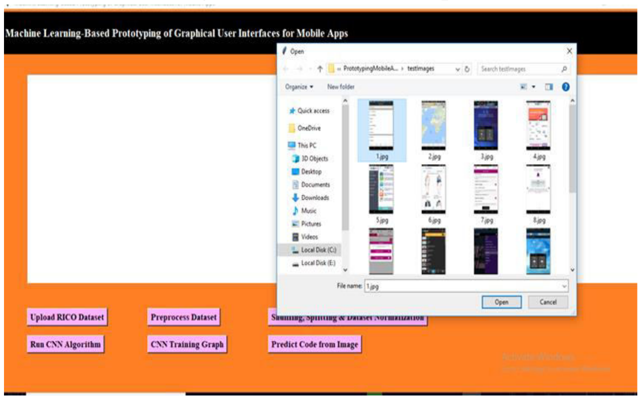
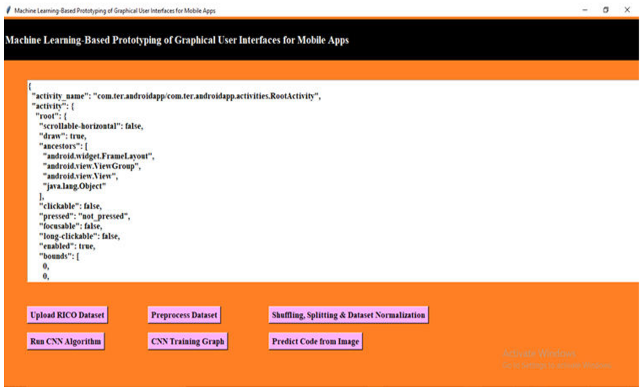


Fig 6.6 Upload the Test Image


Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps
Upload RICO Dataset
Preprocess Dataset
Shuffling, Splitting & Dataset Normalization
Run CNN Algorithm
CNN Training Graph
Predict Code from Image

```
{
 "activity_name": "com.ter.androidapp/com.ter.androidapp.activities.RootActivity",
 "activity": {
  "root": {
   "scrollable-horizontal": false,
   "draw": true,
   "ancestors": [
    "android.widget.FrameLayout",
    "android.view.ViewGroup",
    "android.view.View",
    "java.lang.Object"
   ],
   "clickable": false,
   "pressed": "not_pressed",
   "focusable": false,
   "long-clickable": false,
   "enabled": true,
   "bounds": [
    0,
    0,
```

. Fig 6.7 Prediction Result

7. CONCLUSION

In this paper we have presented a data-driven approach for automatically prototyping software GUIs, and an implementation of this approach in a tool called REDRAW for Android. A comprehensive evaluation of REDRAW demonstrates that it is capable of (i) accurately detecting and classifying GUI-components in a mock-up artifact. (ii) generating hierarchies that are similar to those that a developer would create. (iii) generating apps that are visually similar to mock-up artifacts. (iv) positively impacting industrial workflows. In the future, we are planning on exploring CNN architectures aimed at object detection to better support the detection task. Additionally, we are planning on working with industrial partners to integrate REDRAW, and our broader prototyping approach, into their workflows.

FUTURE SCOPE

The future of Automatic GUI generation with great promises, as major technology companies and developers have been attempting the application of machine learning in their own field. The following are some future work for GUI generation: • In order to further improve the accuracy of code semantic metric in Chapter 3, future research directions will consider building a simpler monitoring model through deep learning and multiple similarity metrics. The evaluation model 90 needs to be able to understand the importance of each similarity metric. An- other interesting direction is to use unsupervised learning algorithms on multiple source code translation datasets, and then use these representations as features of the supervised model. • In Chapter 4, the future work could be trying to create more elements to gen- erate additional web examples such as actual icons, image buttons, drop-down menus, forms, and bootstrap components. With the increasing performance of computer hardware, it is better to create a dataset that can be directly trained by HTML/CSS code than a DSL token sequence in the future.

A good way to generate more variants in hand-drawn sketch data might be to create a realistic hand-drawn website image using a Generative Adversarial Network (GAN).

8. REFERENCES

1. LeCun, Y. et al 1998 "Gradient-based learning applied to document recognition."

2. Rousseeuw, P. J. et al. 1999 "The Wessa Online Outlier Detection."

3. Hunter, J. D 2007 "Matplotlib: A 2D graphics environment." Computing In Science & Engineering, (3), 90-95.

4. Sokolova. M & Lapalme. G. 2009. "A systematic analysis of performance measures for classification tasks."

5. Pan. S. J & Yang. Q 2010 "A survey on transfer learning." IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

6. Bergstra. J & Bengio.Y 2012 "Random search for hyper-parameter optimization." Journal of Machine Learning Research, 13(Feb), 281-305.

7. Yosinski. J. et al 2014 "How transferable are features in deep neural networks?" Advances in Neural Information Processing Systems, 27.

8. Jones. E. et al 2017 "Requests: HTTP for Humans." Retrieved from https://docs.python requests.org/en/latest/.

9. Shorten. C & Khoshgoftaar. T. M. 2019 "A survey on Image Data Augmentation for Deep Learning." Journal of Big Data, 6(1), 60.

10. Sparks. E. R. et al 2020 "Electron: Cross-platform desktop application development."