Bilkent University

Department of Computer Engineering

Senior Design Project

Autoshop

Final Report

Efe Acer
Hikmet Demir
Mehmet Mert Duman
Talha Murathan Göktaş
Burak Yaşar

Supervisor	: Fazlı Can
Jury Members	: Cevdet Aykanat, Ercüment Çiçek
Innovation Expert	: Duygu Gözde Tümer
Website	: https://beastunited.github.io/

Final Report
May 27, 2020
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction

With social media becoming prominent as a byproduct of the technology era, we began to see a very intense online photo traffic. To be more specific, over 300 million photos are uploaded to Facebook daily and there are around 40 billion photos shared on Instagram since its creation [1]. The direct implication of these numbers is that people love to take and share photos. They spend hours trying to capture effectual moments or just to look good. To make it easier for people to be happy with their photos; a new industry, “Photoshop” had emerged.

Photoshop itself, is a demanding task. People train themselves to become good photoshoppers and spend heaps of time in front of the screen to make realistic photo edits. The departure point of our project, “Autoshop”, is to make this arduous task accessible to everyone, even the people knowing the absolute minimum of Photoshop.

The nature of innovation behind Autoshop is to create a new and powerful photo editing tool using state of the art Computer Science techniques. Autoshop will help people add objects to and remove objects from their photos without requiring any photoshop knowledge. Autoshop makes advanced technologies accessible in multiple platforms, aiming to dilate its user profile.

Imagine yourself missing a friend gathering, you would probably say “I wish I was there.”. Autoshop helps you right away at these moments. Using it, you will be able to effortlessly add yourself into the moment. You may argue that this is possible using tools such as Adobe’s Photoshop, however, you would have to work quite a bit to do that. Autoshop, as its name suggests, automizes the process and makes your computer or your mobile phone your personal photoshopper.

In this report, we intend to present the final status, architecture and design of Autoshop. First, the requirement details and refined analysis of our project will be presented. Later, the final architecture and design details of Autoshop will be given to indicate its last status quickly. Then, in the section named Development/Implementation Details, whole development process will be provided in detail. After that, Testing Details of the project will be given. Then in the next section called Maintenance Plan and Details will be provided. The report will be continued with other project elements which are Consideration of Various Factors, Ethics and Professional Responsibilities, Judgements and Impacts to Various Contexts, Teamwork and Peer Contribution, Project Plan Observed and Objectives Met and New Knowledge Acquired and Learning Strategies Used. Afterward, our discussion on the conclusion and future work will be given. Finally, user manual of Autoshop will be given which includes navigations and walkthroughs on the user interface of Autoshop.

2 Requirements Details

2.1 Functional Requirements

In this section, functional requirements will be discussed. Functional software requirements help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform. [2]. As part of functional requirements, we will discuss data resources requirements and user-specific requirements.

2.1.1 System Functionality

The system should:

- ask the user's permission to access the local machine's resources (photo gallery, camera, etc.).
- display a gallery of user photos from which the user can make selections.
- provide an option to receive camera input.
- receive a background image from the user as an input, the background image refers to the image to be edited.
- receive additional images, which are the image(s) to add on the background image, as a user input.
- get the position of the additional images over the background image by drag and drop.
- get the size of the additional images by resizing.
- get the borders of the additional images by cropping and scanning; if the user does not crop or scan, then use smart-cropping methods to extract the object of interest.
- get the borders of the background image by cropping.
- get the scaling factor of the background image by rescaling.
- display the images to edit, in other words, the background image and the additional images on top of it, in a frame.
- communicate with a server, which runs a pre-trained neural network. In the desktop application, the server will be the localhost itself since it can run GPU-heavy tasks. However, in the mobile application, the server will be a remote machine that is capable of running the neural network efficiently.
- send the background images, the additional images; the positions, sizes, and borders of those images to the server.

- use the pre-trained neural network to adjust the photographic properties (opacity, texture, brightness, color, etc.) of the additional images according to the background image, and generate a photoshopped output.
- receive the photoshopped output from the server.
- display the photoshopped image in another frame. As the system receives input, in the mobile application this will require some time to establish server communication and perform data transfers.
- provide an option that allows the user to save the photoshopped output to the local machine's photo gallery.
- provide an option that allows the user to share the photoshopped output using the various social media and messaging platforms (e.g. Whatsapp) in the local machine.

2.1.2 User Functionality

The user will be able to:

- allow or deny permission for the system to use the local machine's resources.
- select or take a background photo and upload it to the system.
- select or take additional images and upload them to the system.
- drag and drop the additional images over the background image to properly position them.
- rotate the additional images.
- resize the additional images.
- crop or scan the additional images.
- see the original image and the photoshopped output.
- save the photoshopped output to his/her photo gallery.
- share the photoshopped output directly in his/her preferred social media or messaging platform.

2.2 Nonfunctional Requirements

In this section, nonfunctional requirements will be discussed. There are many definitions of nonfunctional requirements. "A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system." [2]. Because of this, we will spend a lot effort to make them as good as possible. Below you can find information about Autoshop's non-functional requirements such as extensibility, reliability, usability, accessibility, portability, and efficiency.

2.2.1 Extensibility

The system should:

- be easy to maintain, in other words, open to updates.
- be available on multiple platforms (Android and iOS).

2.2.2 Reliability

The system should:

- ensure that the user data (the additional images and the background image) is not stored in the server-side unless the user gives permission.
- roll-back in case of a failure in the server communication. To be clearer, delete any data that is not completely processed.
- be resistant to adversarial cyber-attacks on the server-side. Methods such as hashing, signing and encrypting can be used to ensure data confidentiality and integrity.
- generate a realistic, hence reliable, output. For this, NVIDIA's libraries such as FastPhotoStyle and Image Inpainting will be integrated into the system [3].

2.2.3 Usability

The system should:

- be self-explanatory and user-friendly.
- be clear in terms of display and language when prompting.
- present a neat and well-organized user interface with themes that the user can select.
- require the absolute minimum photoshop knowledge from users.

2.2.4 Accessibility

The system should:

- be downloadable for free via GitHub.

2.2.5 Efficiency

The system should:

- not lag when communicating with the server. In case of a lag, there should be a time-out event, possibly 5 seconds (subject to change), that leads to server roll-back.
- not delay much in the mobile version when receiving touch-input from the user. A long delay is conventionally considered as 2-3 frames.

- not delay much in the desktop version when receiving keyboard and mouse input. A long delay is usually considered as 100 milliseconds.
- not wait longer than a second (subject to change) for the neural network to generate the photoshopped output.

3 Final Architecture and Design Details

We have made the rational choice to use Flutter for Android development. This is particularly because Flutter provides the ease of developing for multiple platform using the same codebase that at the end compiles to native code. Moreover, Flutter and the Dart language provide good abstractions over their native counterparts in Android and iOS which are easy to understand and work with. Additionally, Flutter's Widget classes make it relatively simpler to write UI code than using xml files.

3.1 Overview

We will describe the architecture and design overview under two main topics as Mobile Application and Server. In the following image, our system can be seen.

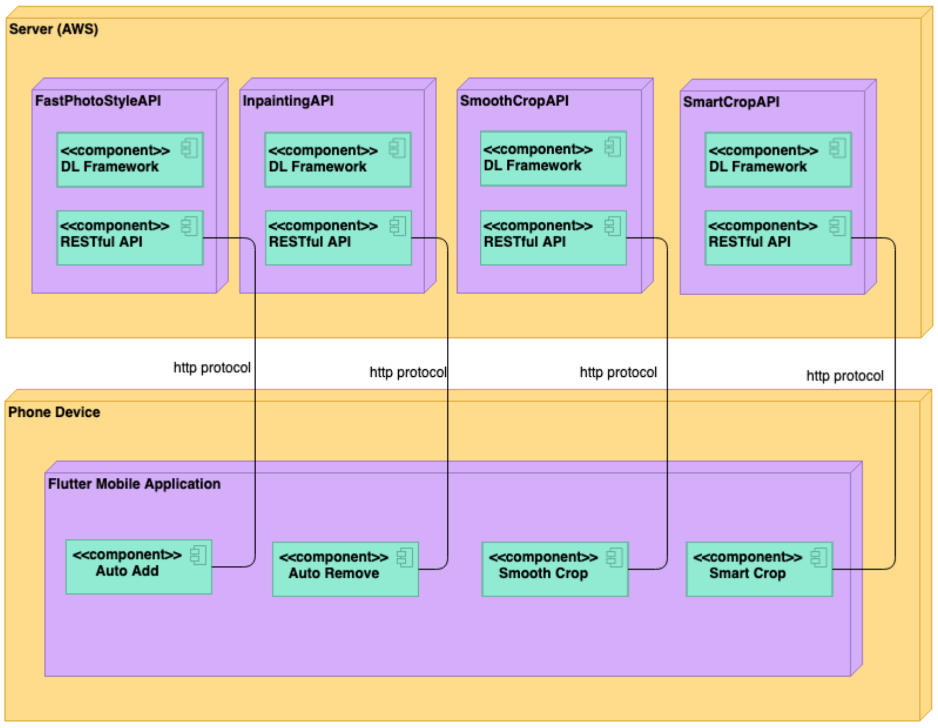

Server (AWS)
FastPhotoStyleAPI
<<component>> DL Framework
<<component>> RESTful API
InpaintingAPI
<<component>> DL Framework
<<component>> RESTful API
SmoothCropAPI
<<component>> DL Framework
<<component>> RESTful API
SmartCropAPI
<<component>> DL Framework
<<component>> RESTful API
http protocol
http protocol
http protocol
http protocol
Phone Device
Flutter Mobile Application
<<component>> Auto Add
<<component>> Auto Remove
<<component>> Smooth Crop
<<component>> Smart Crop

Figure 1: Deployment Diagram

3.1.1 Mobile Application

Flutter is an app SDK for building high-fidelity, high-performance apps for iOS and Android from a single codebase. Apps are written in Dart language in Flutter and it looks familiar if the one has used a language like Java or JavaScript, that' why we decided to continue with Flutter while designing the architecture of Autoshop. The components of Flutter such as 2D rendering engine, ready-made widgets, and development tools help us design, build and debug our application neat and swiftly.

Unlike other frameworks that have separate views, view controllers, layouts, and other properties, Flutter has a consistent, unified object model: the *widget*. A widget is a basic building block of UI of the Flutter applications and each of them is an immutable declaration of this UI. While developing an application, the one will be commonly an author of new widgets which are subclasses of either *StatelessWidget* or *StatefulWidget*, depending on whether her widget manages any state [4].

The main job of a a widget is to implement a *build()* function, that describes the widget in terms of other, lower-level widgets. Whenever it is asked to build, the widget should return a new tree of widgets regardless of what the widget previously returned. The framework does the heavy lifting of comparing the previous build with the current build and determining what modifications need to be made to the user interface [4].

3.1.2 Server

We decided to leverage AWS for our server structure. AWS provides secure and reliable server structures with variety of choices for different purposes. It can be expensive then it rivals yet they provide better functionality in simpler terms. We chose to rent a p2.xlarge server this server has 61 Gib Memory, 2496 Parallel Processing Cores and Nvidia Tesla K80 gpu which has 12 Gib GPU Memory. This GPU was essential for us to run our Deep learning models in parallel and fast enough in the app.

We are running four main server-side restful API for our project

- FastPhotoStyle API
- Inpainting API
- SmartCrop API
- SmoothCrop API

Flutter and AWS are communicating over certain ports with HTTP protocols, details of the server and the backend application will be given later.

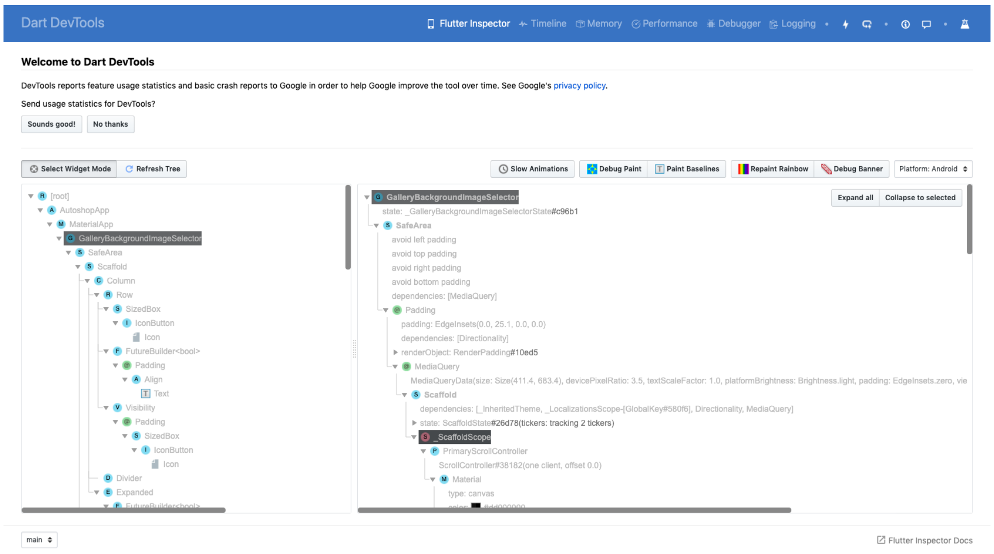
3.2 Application Architecture & Design Choices

Earlier we have mentioned that there are two mainly widget types as *Stateless* and *Stateful*. Stateless widgets receive arguments from their parent widget, which they store in final member variables. When a widget is asked to *build()*, it uses these stored values to derive new arguments for the widgets it creates.

In order to build more complex experiences for instance, to react in more interesting ways to user input applications typically carry some state. Flutter uses *StatefulWidgets* to capture this idea. *StatefulWidgets* are special widgets that know how to generate State objects, which are then used to hold state [5].

As mentioned earlier in section 3.1.1 Flutter uses dart language and therefore we developed our application with dart on the client side. In our application we have used both stateful and stateless widgets. In our flutter project, all the dart files that consist our source code are located inside lib directory. Inside that directory we have two packages as pages and services. The files under services directory is written to facilitate the development process, they are basically the helper classes and methods to operate common behaviour or practice that we use when needed in the files under pages directory. These files under pages directory maps with each page of the Autohop app and inside these pages stateful widgets are prefferred to handle user interaction therefore state changes.

Below you can see sample widget tree/hierarchy from Autohop.



Welcome to Dart DevTools

DevTools reports feature usage statistics and basic crash reports to Google in order to help Google improve the tool over time. See Google's privacy policy.

Send usage statistics for DevTools?

```
▼ [root]
  ▼ AutoshopApp
    ▼ MaterialApp
      ▼ GalleryBackgroundImageSelector
        ▼ SafeArea
          ▼ Scaffold
            ▼ Column
              ▼ Row
                ▼ SizedBox
                  ▼ IconButton
                      Icon
                ▼ FutureBuilder<bool>
                  ▼ Padding
                    ▼ Align
                        Text
                ▼ Visibility
                  ▼ Padding
                    ▼ SizedBox
                      ▼ IconButton
                          Icon
              Divider
              ▼ Expanded
```

```
▼ GalleryBackgroundImageSelector
    state: _GalleryBackgroundImageSelectorState#c96b1
  ▼ SafeArea
      avoid left padding
      avoid top padding
      avoid right padding
      avoid bottom padding
      dependencies: [MediaQuery]
    ▼ Padding
        padding: EdgeInsets(0.0, 25.1, 0.0, 0.0)
        dependencies: [Directionality]
      ▶ renderObject: RenderPadding#10ed5
      ▼ MediaQuery
          MediaQueryData(size: Size(411.4, 683.4), devicePixelRatio: 3.5, textScaleFactor: 1.0, platformBrightness: Brightness.light, padding: EdgeInsets.zero, vie
        ▼ Scaffold
            dependencies: [_InheritedTheme, _LocalizationsScope-[GlobalKey#580f6], Directionality, MediaQuery]
          ▶ state: ScaffoldState#26d78(tickers: tracking 2 tickers)
          ▼ _ScaffoldScope
            ▼ PrimaryScrollController
                ScrollController#38182(one client, offset 0.0)
              ▼ Material
                  type: canvas
```

Figure 2: Sample Widget Tree

3.3 Server Architecture & Design Choices

Reliability and distribution are the main goals in our design. We wanted to make our servers as reliable as possible and distribute different jobs to independent applications, so that a possible crash occurring in one of them will not be affecting the others. That is why we separated the server work into the following:

3.3.1 FastPhotoStyle API

NVIDIA researchers proposed a neural network that is capable of transferring the style of a base image to another in their paper "A Closed-form Solution to Photorealistic Image Stylization" [3], they call the network FastPhotoStyle which in the basic terms performs style transfer by deep feature transformations. We implemented an API that allows our server to run the neural network and interact with it according to the front-end needs.

3.3.2 Inpainting API

PartialConv is yet another neural network prepared by NVIDIA, that is capable of performing image inpainting which is basically to remove parts of images without distrupting the overall look of the image. The paper, "Image Inpainting for Irregular Holes Using Partial Convolutions" provides detailed information about the network architecture [6]. Again, we implemented our own API that interacts with the PartialConv network according to our needs.

3.3.3 SmartCrop API

Smart crop feature makes use of the state-of-the-art object and edge detection technologies to extract objects from the given scene. With this feature, our users are spared the hassle of picking out the edges of the objects they want to use in their photos. SmartCrop API provides this functionality by gathering algorithms and neural networks that perform edge detection, mask generation, dilation, erosion, and contour generation.

3.3.4 SmoothCrop API

Smooth crop allows our users to crop their images to their will. Simply dragging over the screen allows them to define the boundaries of the image segment that will be cropped. We made use of several computer vision technologies to allow for a smooth and satisfying experience for our users using this feature, and gather this functionality in the SmoothCrop API.

4 Development/Implementation Details

4.1 Mobile Application

Mobile application is implemented as a Flutter app where Flutter is an UI software development kit made by Google[7]. This method is chosen as it is optimized for creating fast and beautiful UI's practically. Because the models needed in *AutoAdd* and *AutoRemove* were planned to be ran only in the server, an UI SDK is adequate. Flutter apps are written in Dart language which has optimized features for UI developing such as *async-await* functions enabling event-driven design.

In the implementation stage of Autoshop, 3 members of team were selected to be responsiple of mobile application side. These members started by generating a empty Flutter app than creating the UI pages of Autoshop as proposed in Analysis Report. Then these pages are connected by UI elements such as buttons, after that the OS communication required events such as image saving to and retrieving from gallery, saving data to persistent and temprorary storage is implemented. In final stage the communication with the server side implemented and the Autoshop app became fully functional.

During the implementation, the team used github to coordinate their works by merging and branching. For the sake of consistency in code variable naming style etc. was decided on and commit was reviewed by another team member before merging. The main method for testing *Autoshop* while implementation stage was the Android Emulator. With the help of *hot-reload* feature of Flutter, it was possible to integrate the changes on the code to the emulator instantly, thus making the testing on Emulator fast and efficient.

4.2 Server

Server side of the mobile application mostly implemented in python with using various deep learning and web frameworks. We needed a powerful server in our app cause of the huge computational needs of the deep learning frameworks that we are using. This servers are backed with high capacity memory and powerful gpu (Nvidia Tesla K80 in our case)

We started with researching possible deep learning implementations to use in our applications. We needed three main deep learning application: Image inpainting for auto remove, Photo Style Transfer for auto add, Object detection & Edge detection for smart crop. We read and tried different papers and implementations of this papers. We also cross check the performance of the models (interms of memory efficiency, quality of the results and time taken for running)

As a result, we decided to use DeepFill with Contextual Attention and Gated Convolution[8] for inpainting (Auto Remove) purposes, A Closed-form Solution to Photorealistic Image Stylization[9] for Photo stylization (Auto Add), Mask R-CNN for Object Detection and Segmentation for object detection[10] and Holistically-Nested Edge Detection for edge detection[11]. All of these libraries have different dependencies and making them work simultaneously was a real challenge

We created separete virtual environments for each of these deep learning frameworks and installed necessary side packages and cuda versions. We decided to use http protocols for communicating between server and mobile application. As a result of these, we created restful api with flask package in the server. We hosted each of the api from a different port in the server and specificed the paths for get and post requests. For each of the request that come our server, we create universally unique identifier(uuid) and save the images with the uuid names. This creates extra security layer in order to not get exposed other images. Also, the folder structure that we are using internally in the server and the paths we are exposing the outer world are not consistent and not connected.

5 Testing Details

Testing is crucial to ensure that an application runs as intended. For this purpose, we integrated certain testing patterns to the development of Autoshop. These patterns are listed below:

5.1 Continuous Integration

In software engineering, the term Continuous Integration refers to the practice of merging all developers' working copies to a shared mainline several times a day. We applied this practice when developing Autoshop. The key for success in Continuous Integration is to make sure that the project copy in the shared mainline is error-free.

We used GitHub to store the shared mainline and used Travis CI to test it. Travis CI is basically a hosted continuous integration service used to build and test software projects hosted at GitHub. Travis CI triggers the test-code every time a team member pushes a working copy, and the shared mainline only accepts the working copy if there are no failing assertions in the test-code.

We wrote test-code regularly and aimed at a certain line-coverage, which means that a certain amount of the code lines are executed by our test-code. Consequently, we ensured a significant amount of the code lines are running without any failed assertions. Line-coverage can be relatively low for mobile apps since testing the UI-code is redundant. Having tested the server communication codes and app-logic, we are confident that

Autoshop runs as planned.

5.2 Automated Code Review

We employed certain coding conventions; such as variable and function naming, scoping and the use of certain design patterns. These conventions significantly speed up the development cycles and ensured that the team members review each others code with ease.

Code Climate is used as the Automated Code Review service to ensure that the style conventions determined by the team are followed. Code Climate was paired with GitHub at the beginning of the project and helped us follow the convention guidelines through out the development.

5.3 Testing the UI

Testing the User Interface was one of our primary concerns. For this reason we formed a user group from our friends and families, who acted as if they were the product owners. They were, of course, non-developers and they regularly tested the UI by navigating arbitrarily in the app; then they reported the bugs they have found and gave feedback on the appearance of the app. This significantly helped us to detect and solve UI bugs fast, and design a better interface.

5.4 Server Performance and Response

Since Autoshop operated on images and images can vary a lot in size and type, we needed to test the performance of our server in different image types and sizes. We had an image directory that is specifically created for testing, which includes high resolution images encoded with different schemes. We programmed the servers to respond properly to all images in this directory in an acceptable time. The server code utilized down-sampling and type-converting to accomplish this.

6 Maintenance Plan and Details

Maintenance of Autoshop is compulsory since the app is heavily server dependent. Also, the code-base needs to be adjusted to keep up with the upcoming versions of the plug-ins. The following tells about the maintenance plan:

6.1 Server Maintenance and Optimization

The models for photo styling and image inpainting requires certain versions of external `Python` libraries to be imported. We need to check for updates in these external libraries and the models itself on a regular basis, and the improving changes in the server code.

We are using the AWS' rental servers, so we need to deposit the rent and configure the server once at the end of each rent term.

Depending on the feedback we are going to receive from the users, we may prefer to switch to an upgraded server that can give a faster response to the requests.

6.2 Issue Tracking

We are strongly for Open-source software (OSS), which is the general name for computer software in which source code is released under a license where the copyright holder grants users the rights to study, change and distribute the source code.

Once we open-up our GitHub repository to public, we will be constantly reviewing the issues posted by the GitHub community and work with them to come up with fixes. This will hopefully make Autoshop much better as the Linus's law states "Given enough eyeballs, all bugs are shallow."

6.3 Google Play Store and other App Stores

Our aim is to upload Autoshop to Google Play Store after we are done with the demo-runs and receive feedback from our valuable colleagues and professors. After Google Play Store, we will proceed with uploading Autoshop to other App Stores such as Apple's. To be able to upload the app, we will need to write user agreements and send the app to the authorities for review.

6.4 Releasing New Versions

After the app becomes available in the App Stores, we will check for the ratings and comments of our users regularly; and release new versions accordingly. These new versions will involve changes to the UI that will hopefully make the user experience better and result in higher ratings.

7 Other Project Elements

7.1 Consideration of Various Factors

There were limiting factors in our project that required an extensive discourse. Below are some that factors limited our design and brief discussions on the potential effects of these factors:

Public Health

There is no physical illness that can be caused by our application, at least we cannot think of one right now. However, our application may trigger certain psychological problems. Those psychological problems are closely related with social media. Research show that social media stimulates a wide range of mental problems that are closely related with anxiety and inadequacy [12]. People are observed to be comparing themselves with macro-influencers and feel as if they cannot measure up with the "ideal" look depicted by them. One use of *Autoshop* is to modify a personal photo such that the photo becomes aligned with the social norms of beauty. This use case may potentially be a primer to an upcoming use of social media where everyone tries so hard to comply with social norms of beauty, which in turn causes self-dissatisfaction and originate even more mental problems. To prevent this to some extend, we made the design choice where the "autoshopping" tutorial used in the application is periodically changed and the tutorial photos are designed to be inclusive as possible.

Public Welfare

Since our application is planned to be free to download and contain no in-app purchases, it has no effect on public welfare. Anyone owning a mobile phone can download the application for free and downloading the application, obviously, will not affect who owns a mobile phone and who does not. Thus, a discussion on public welfare is not applicable to our context.

Global Factors

In our discussion, global factors refers to rules, regulations and norms that nearly every country come to a mutual agreement on. Such factors include well-defined standards like General Data Privacy Regulation (GDPR) [13], Google Play Terms of Service [14] and other explicitly written or implicit agreements between the developers, users and authorities. We inevitably limited ourselves to these factors. Since these factors, in some sense, reflect the public-opinion, following them is essential for us to build an inclusive application.

Social Factors

There is no indubitable definition of a social factor, but factors such as age, religious orientation, family history, race and ethnicity, education, and economic status are some examples of social factors. We cannot identify a clear link between these social factors and our application. We can only indicate that cultural and ethnic biases present in the training datasets of the models may play a part in the outputs of styling and inpainting models and thus introduce a bias.

In the table below, we indicate the significance, or equivalently the level of effect, on a scale of 0 (none) to 10 (maximum) of each factor we have mentioned above:

	Effect Level	Effect
Public Health	7	Psychological Problems of Anxiety and Inadequacy among Users
Public Welfare	0	Not Applicable
Global Factors	9	An Exclusive Application that is Far from Global Standards
Social Factors	4	Cultural/Ethnic Biases Incorporated into the Models

Table 1: Factors that can effect analysis and design

7.2 Ethics and Professional Responsibilities

This section covers the purpose of next section which is Judgements and Impacts to Various Contexts and it is included in this section 7.2 as it was in the Analysis Report.

We begin our discussion on ethics and professional responsibilities with the potential **Global Impact** of our project. The ultimate goal of *Autoshop* is to make the arduous task of photoshop easier and more accessible. For this reason, the application is available to everyone. The application does not restrict itself to the use of a certain group of people, country or any social class. As a direct implication of the unrestricted accessibility of *Autoshop* and its goal of making photoshopping easier; we expect the global impact of the application to make photoshopping a regular, effortless task that anyone in the world can perform anywhere, anytime.

Our discussion proceeds with the **Economic** impact and constraints of our project.

The photoshop market, namely the graphics market, is currently dominated by Adobe Photoshop, which owns a market share of 57.29% [15]. To estimate the total value of the Graphics market, we look at the value of Adobe. Adobe is reportedly a 95 billion dollar [16] company, so one can estimate the total value of the photoshop market as 170 billion dollars assuming that each percent of the market share possesses the same constant value. *Autoshop* begun its journey as a undergraduate capstone project, hence it will not claim any market share in the near future. In other words, the application is totally free to download. In some future time, we may put adds on our application that may potentially generate some amount of profit which in turn lets *Autoshop* to claim a small amount of the market share.

Besides the economic impact mentioned above, there are also economic constraints that affected the development phase. These constraints were basically costs. Some of these costs that we have experienced were:

- AWS Previous Generation GPU Instance Deployment Server Costs:
  \$0.90 per hour $\xrightarrow{\text{100 hours of testing}}$ \$90
- GitHub Registration Fee: Free Student Account
- Jira Registration Fee: Free up to 10 people

We did afford these costs, so none of them restricted us in any aspect. However, we were not able to rent a super-computer like server so we limited the app in certain areas, such as model performance and portability.

Another impact that *Autoshop* has is **Environmental**. There exists a significant correlation between the use of machine learning models and electricity consumption. As data dependent fields such as deep learning and machine learning became more prominent, the need to store and process the data increased. There is a drastic incline in the data centers built, amount of time allocated servers run, and consequently the electricity consumed. According to International Energy Agency's data almost 1% of the global final demand of electricity is the global data center electricity demand [17]. Since *Autoshop* uses NVIDIA's pre-trained generative models the training time required for the servers was relatively low. However, there was some amount of testing for the additional helper models we have constructed from scratch. *Autoshop* is an application that is completely data dependent, so storage is also a must. Because of the aforementioned model training and data storage requirements there is an essential but small negative environmental impact of the application.

The last impact we will touch on is the **Societal Impact** of *Autoshop*. The biggest issue regarding ethics is the issue of deep fakes. Deep fake refers to manipulated photos or videos that yield fabricated images, motions and sounds that appear to be real [18]. The term deep fake combines "deep learning" and "fake", as the name implies, the fake digital representations are powered by sophisticated learning algorithms. Such an algorithm is Generative Adverserial Networks (GANs), which are primarily used to make preliminary fake images look more realistic. The preparation process of a deep fake directly aligns with the photoshopping process in *Autoshop*. To be clearer, the application operates on a preliminary input photo where the user indicates which objects to add or remove, and then proceeds by making this input more believable by benefiting from NVIDIA's GANs and other helper models. As a consequence, *Autoshop* imposes a serious risk of becoming a deep fake generator. The danger, or the ethical issue, here is that deep fakes can be used to make people believe things that are not actually real. For instance, using a deep fake one may decrease the number of audiences in the photo of a crowded concert and depict the concert as if it was unsuccessful, or one can use deep fakes to alternate the facial reactions of people, and so on.

Another significant societal issue that we should certainly safeguard is data privacy. *Autoshop* is designed to conform to the General Data Privacy Regulation (GDPR) [13]. Hence the application never keeps user data in the remote servers without explicitly asking for permission. The input photo that the user had prepared on his/her local machine is transferred to the processing servers, after the processing is done the application deletes the photo if the user had denied permission. In case of an interrupt in the server communication, the servers roll-back to the initial state where no input photo is received. This way we ensure that no data is kept without permission.

7.3 Judgements and Impacts to Various Contexts

The numbers in the table below are out of 10:

	Impact Level	Impact
Impact in Global Context	5	Attempted to make Photoshopping easier and more accessible.
Impact in Economic Context	3	Costs restricted model performance as expected.
Impact in Environmental Context	2	Server's electricity consumption when training the models is small but not insignificant.
Impact in Societal Context	8	Deep fakes are highly risky in the societal grounds. Safeguarding data is crucial and the responsibility completely belongs to the team.

Table 2: Impacts of this judgement

7.4 Teamwork and Peer Contribution

Efe Acer: Took part in the front-end work, contributed mainly to the interface of the Merge, Remove, Manual Crop and Smart Crop pages. Helped design the interface of the application. Added service code that helps all pages. Worked on the export functionality and implemented painter classes that draw user-given masks and other custom shapes.

Hikmet Demir: Took part in the back-end work, integrated the NVIDIA models to the system, made sure they were working as intended without unexpected errors. Added optimizations to speed up server response. Helped front-end side to prepare server communication code.

Mehmet Mert Duman: Took part in both the front-end and back-end work. Integrated segmentation models to the system for Smart Crop, and prepared additional algorithms to generate masks from raw images. Primarily contributed to the interfaces of Background and Asset Image Selector, Smooth and Smart Crop pages. Added service code to the project that served as helpers to all pages, helped with the design.

Talha Murathan Göktaş: Took part in the front-end work. Mainly contributed to the Remove, Smooth Crop and Asset Image Selector pages. Added UI animations. Worked on the image exporting feature. Helped write the server communication code by writing algorithms to convert between different types of images.

Burak Yaşar: Took part in the front-end work. Mainly contributed to the Remove, Merge and Smooth Crop pages. Connected the UI pages together to form the navigation tree. Helped design the interface and added service code that helps all pages. Implemented models that can store and draw user-given image masks. Worked on the server communication codes.

Note: All group members helped with bug-solving, code refactoring and report preparation. Although there was a primary focus for each member, everyone helped each other and took part in each other's work.

7.5 Project Plan Observed and Objectives Met

In this section, what we have done based on the Requirements section (which is second part of the report) will be presented.

7.5.1 Functional Objectives

In this section, functional requirements that is met will be listed. As part of functional requirements, we will discuss system related requirements and user-specific requirements.

7.5.1.1 System Functionality

Now the system is able to:

- ask the user's permission to access the local machine's resources (photo gallery, camera, etc.).
- display a gallery of user photos from which the user can make selections.
- provide an option to receive camera input.
- receive a background image from the user as an input, the background image refers to the image to be edited.
- receive an additional image, which is the image to add on the background image, as a user input.
- get the position of the additional images over the background image by drag and drop.
- get the size of the additional images by resizing.
- get the borders of the additional images by cropping and scanning; if the user does not crop or scan, then use smart-cropping methods to extract the object of interest.

- get the borders of the background image by cropping.
- display the images to edit, in other words, the background image and the additional images on top of it, in a frame.
- communicate with a server, which runs a pre-trained neural network. However, in the mobile application, the server will be a remote machine that is capable of running the neural network efficiently.
- send the background images, the additional images; the positions, sizes, and borders of those images to the server.
- use the pre-trained neural network to adjust the photographic properties (opacity, texture, brightness, color, etc.) of the additional images according to the background image, and generate a photoshopped output.
- receive the photoshopped output from the server.
- display the photoshopped image in another frame. As the system receives input, in the mobile application this will require some time to establish server communication and perform data transfers.
- provide an option that allows the user to save the photoshopped output to the local machine's photo gallery.
- provide an option that allows the user to share the photoshopped output using the various social media and messaging platforms (e.g. Whatsapp) in the local machine.

7.5.1.2 User Functionality

Now the user is able to:

- allow or deny permission for the system to use the local machine's resources.
- select or take a background photo and upload it to the system.
- select or take additional images and upload them to the system.
- drag and drop the additional images over the background image to properly position them.
- rotate the additional images.
- resize the additional images.
- crop or scan the additional images.
- see the photoshopped output.
- save the photoshopped output to his/her photo gallery.
- share the photoshopped output directly in his/her preferred social media or messaging platform.

7.5.2 Non-Functional Objectives

In this section, non – functional requirements will be discussed. Because of this, we spent a lot effort to make them as good as possible. Below you can find information about Autoshop's non-functional objectives that are met such as extensibility, reliability, usability, accessibility, and efficiency.

7.5.2.1 Extensibility

Now the system:

- is easy to maintain, in other words, open to updates.
- can be available on multiple platforms (Android and iOS).

7.5.2.2 Reliability

Now the system:

- ensures that the user data (the additional images and the background image) is not stored in the server-side unless the user gives permission.
- can roll-back in case of a failure in the server communication. To be clearer, delete any data that is not completely processed.
- is resistant to adversarial cyber-attacks on the server-side. Methods such as hashing, signing and encrypting can be used to ensure data confidentiality and integrity. This is provided by the security of Amazon Web Services.
- generates a realistic, hence reliable, output. For this, NVIDIA's libraries such as FastPhotoStyle and Image Inpainting will be integrated into the system [3].

7.5.2.3 Usability

Now the system:

- is self-explanatory and user-friendly.
- is clear in terms of display and language when prompting.
- presents a neat and well-organized user interface with themes that the user can select.
- requires the absolute minimum photoshop knowledge from users.

7.5.2.4 Accessibility

Now the system:

- is downloadable for free via GitHub.

7.5.2.5 Efficiency

Now the system:

- does not lag too much when communicating with the server.
- does not delay much in the mobile version when receiving touch-input from the user. A long delay is conventionally considered as 2-3 frames.
- does not wait longer than three second for the neural network to generate the photoshopped output.

7.6 New Knowledge Acquired and Learning Strategies Used

Strictly speaking, our past knowledge was inadequate to build *Autoshop* right away. Thus, we acquired some knowledge, developed new skills and plan appropriate learning strategies to do so. Some topics that we centered upon are:

- Advanced Image Processing
- Deep Learning
- Flutter Development
- Application Design
- Software Development Planning

Automating the process of photoshopping required us to understand the state of the art machine learning models, which may not be covered in Bilkent's courses. Hence, we gained in-depth knowledge on deep learning. As photoshopping deals with images, we naturally needed to learn about advanced image processing algorithms. As a consequence of our choice of deployment platform we gained a good understanding of android development and application design. Finally, to meet the deliverables and deadlines we have established a solid understanding of software development planning.

Below are some methods that we have employed to learn more on the topics above:

- Literature Review
- Online Learning
- Hands-on Experience

Literature review was beneficial for us to discover cutting edge deep learning architectures and their working principles, same for advanced image processing algorithms.

For us, literature review included a wide range of research from surfing in Stack Overflow to reading ACM papers. Online learning was necessary, since there are a lot of online tutorials that teach how to write Android programs. Watching these tutorials accelerated the development process a lot. Lastly, hands-on experience was the key for us since the best teacher for us happened to be our own mistakes.

8 Conclusion and Future Work

8.1 Conclusion

After all, we succeeded to create the app that we have designed at the beginning of the year. Autoshop is up and working, it can add objects to images by adjusting the style of the added object and it can seemlessly remove objects from images. It is free and open-source, anyone can access the code, download it and use it via GitHub.

We are happy with our progress as well as the app's progress. We improved our design, coding, team-communication, and mobile development skills a lot. We still need to get better at many aspects and we certainly will. We welcome any feedback and advice related to us, and Autoshop.

8.2 Future Work

Our main goal, as stated in the maintenance section, is to make Autoshop downloadable in certain App Stores. Thus, the next step for us will be to change the app according to the feedback we are going to receive from our professors and colleagues. After that, we will write user agreements and send our app to be reviewed by the App Store authorities. Then, we will keep releasing new modifications/features together with the Open Source community to move the app to higher rankings in the stores.

We will apply the experience and skills we have learned by developing Autoshop to any other project that is waiting for us; and we will, hopefully and if possible, work together and build new apps in the future.

9 User Manual

In this part of the report, the user manual of the Autoshop will be presented. Each picture is provided with an explanation to better specify how it addresses the usage in that page.

Logo Page

The user is welcomed with the *Logo Page* having the logo of *Autoshop* and production remarks noted at the bottom of the page as shown in Figure 3. User cannot interact with the application at this page while the application is initializing the network with the servers.

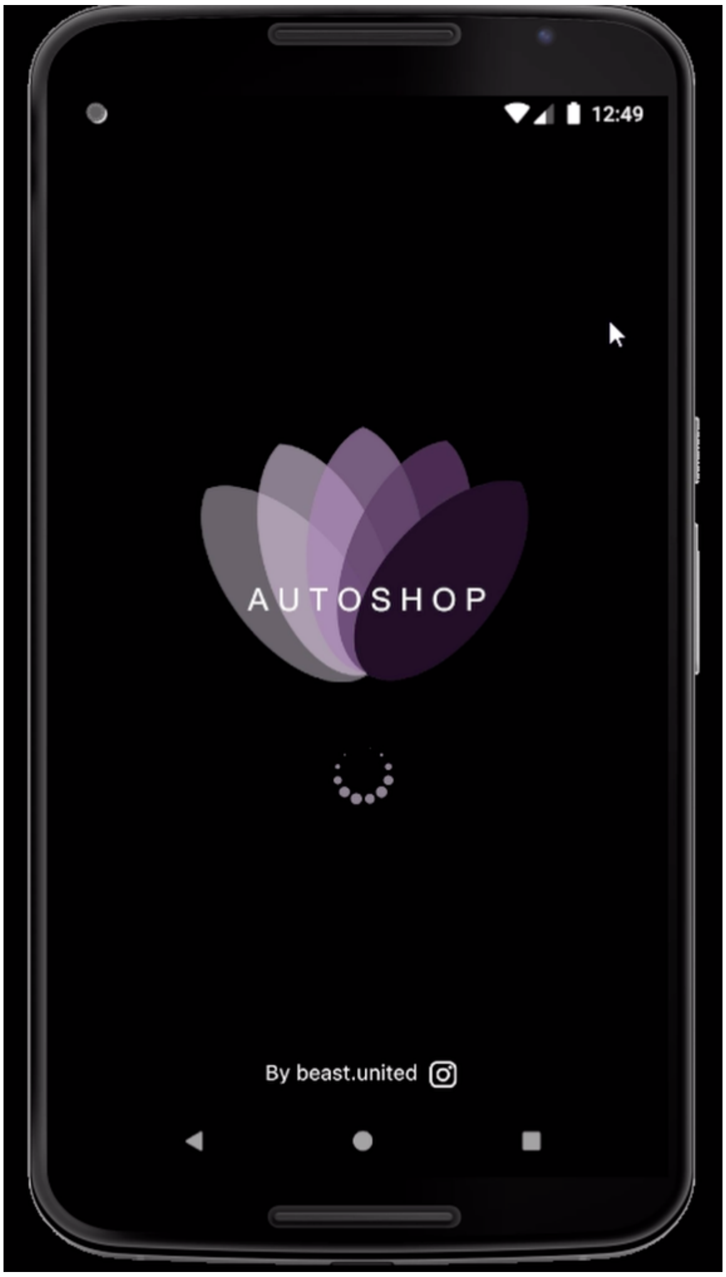


Figure 3: Logo Page

Background Photo Selection Page

After the *Logo Page*, the user is directed to *Background Photo Selection Page* for choosing a background photo that is required for both *Auto Remove* and *Auto Add* functionalities of *Autoshop.* At this page, the photos from the local storage, which is basically the gallery application of the device, are shown in Autoshop in grid view where user can choose the background image by tapping on them. When the selection is made it is indicated by showing little blue tick on the bottom-right of the selected image. Another option to select a background image is that users can take a photo after tapping the camera icon, which is located between *information* icon and *settings* icon on the bottom-right of the page, as shown in Figure 4. Users can tap *information* icon to see the tutorial about the usage of the Autoshop and to be directed to *Settings Page* a user can tap *settings* icon.

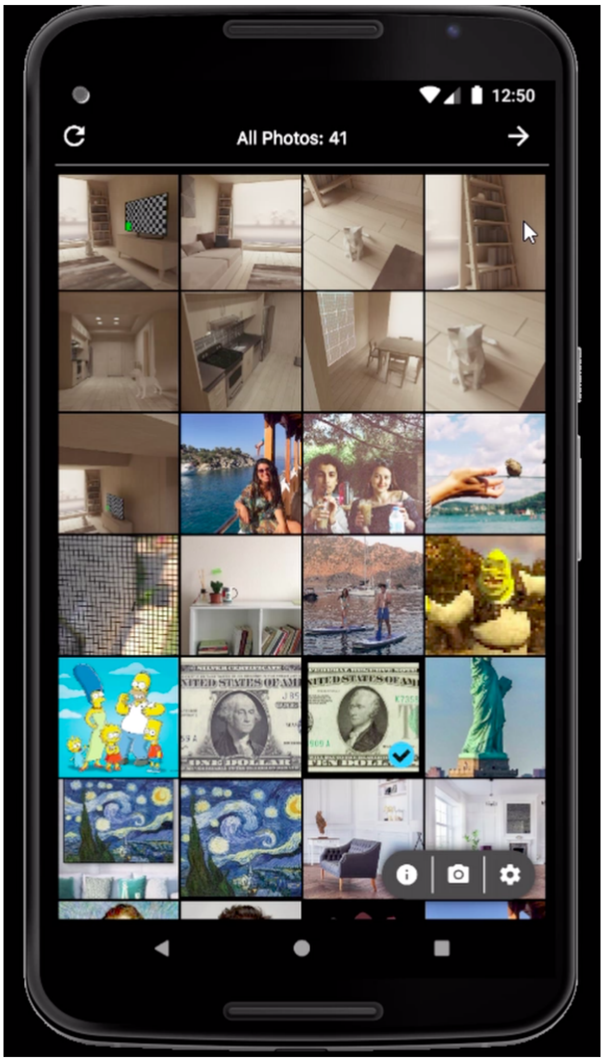


Figure 4: Background Photo Selection Page

Add or Remove Page

After selecting the background image user is asked for the *Autoshop* functionality that she proceeds as shown in Figure 5. If she chooses *Add Image*, she is directed to *Add Additional Image Page* or if she chooses *Remove* she is directed to *Auto Remove Page.* Also the user can go to the previous page, which is *Background Photo Selection Page*, by tapping the *left arrow* icon on the top-left corner.

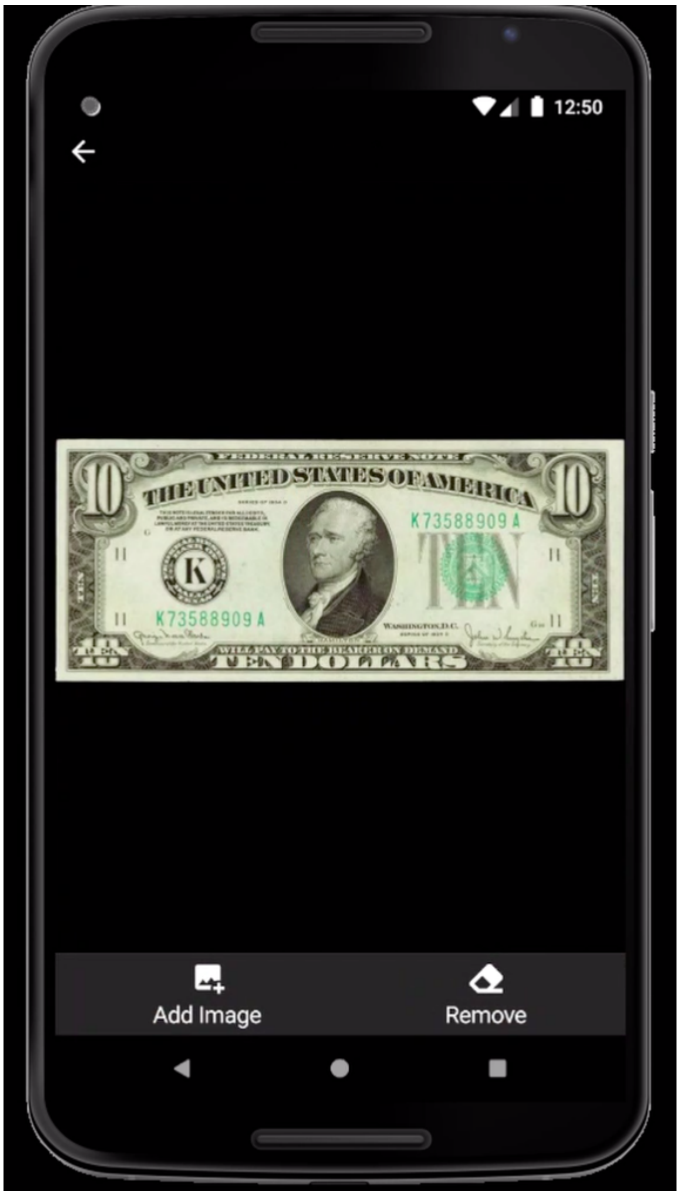


Figure 5: Add or Remove Page

Add Additional Image Page

User is directed to this page if she chooses *Add Image* option in *Add or Remove Page* as she needs an additional image. User can choose a photo from displayed photos which are same as the photos of the gallery application. Again, when the selection is made it is indicated by showing little blue tick on the bottom-right of the selected image. Another alternative is that she can click the *Assets* icon on the bottom-right of the page and next to the camera icon, to import from previously saved assets. Also she can take a photo as an additional image by using camera when she clicks the *Camera* icon on the bottom-right of the page next to the assets icon. If the user clicks the *right arrow* icon on the top-right corner she is directed to *Crop Selection Page* to crop the desired part of the photo. Also the user can go to the previous page with tapping the *left arrow* icon on the top-left corner.

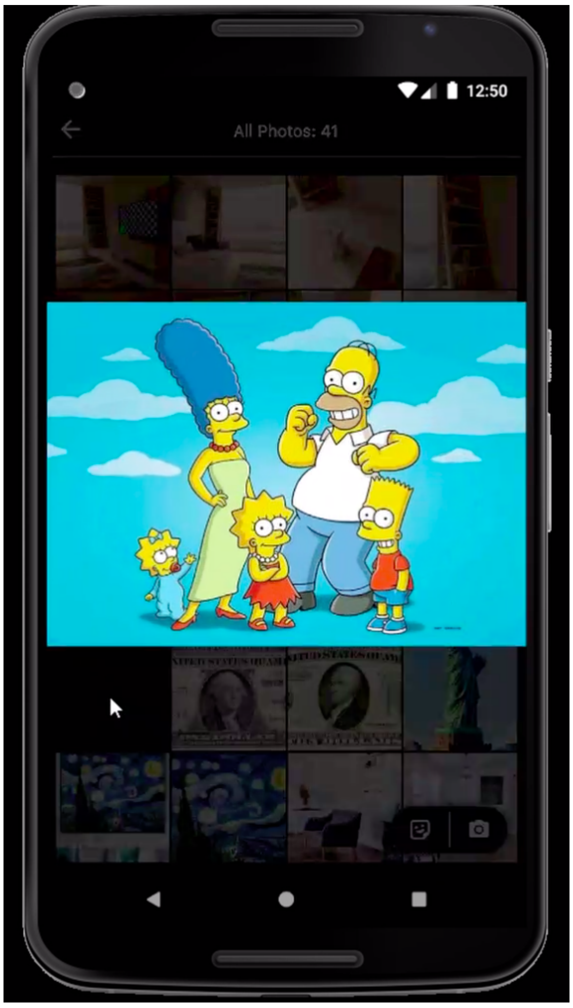


Figure 6: Add Additional Image Page

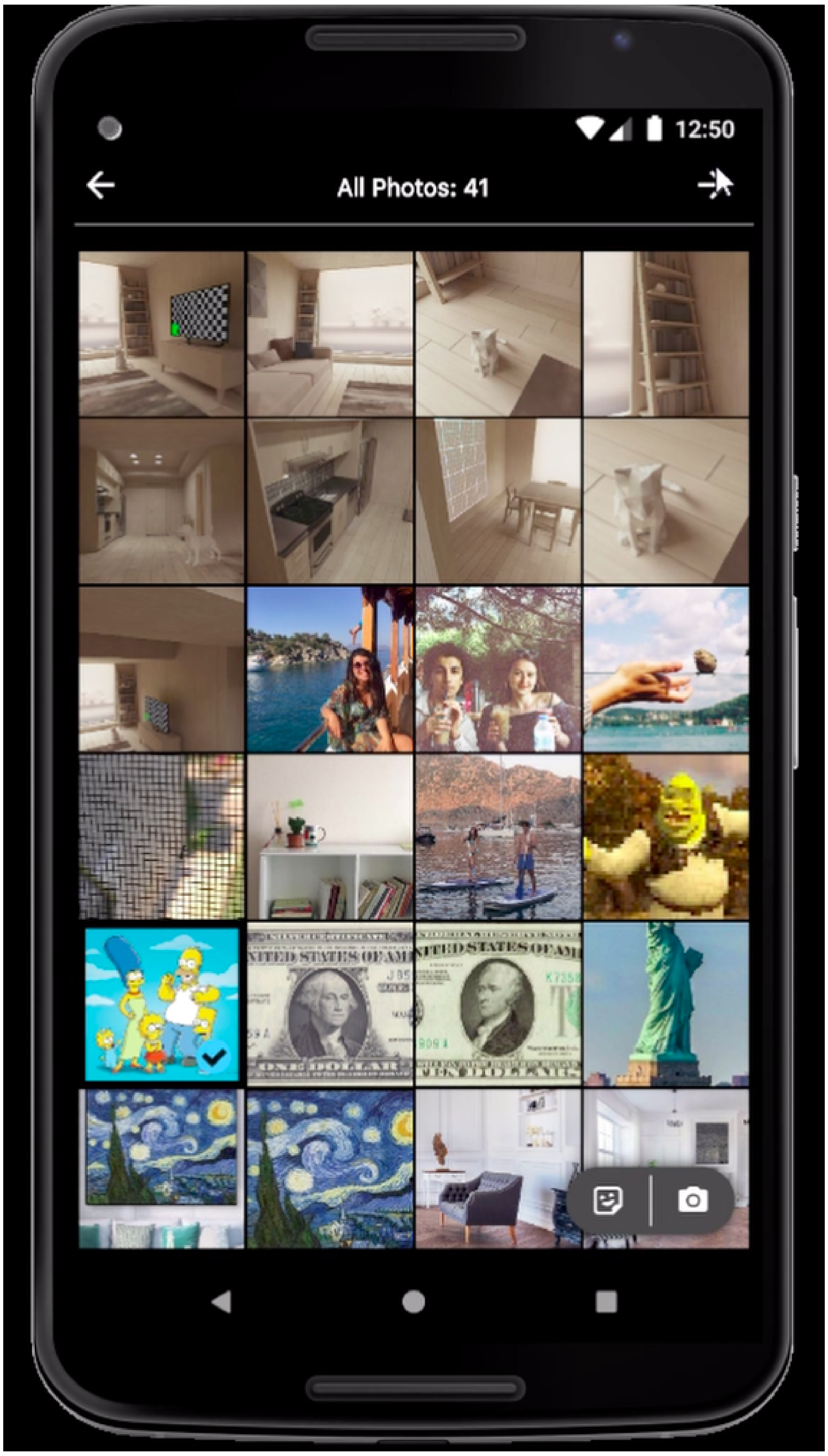


Figure 7: Add Additional Image Page

Crop Selection Page

User is directed to this page after she clicks the *right arrow* icon on the previous page. In this page, additional image which is selected by the user is displayed and the user chooses a method to crop the image using the *Manual Crop*, *Smooth Crop* or *Smart Crop* buttons on the bottom of the page as shown in Figure 8. User is directed to the corresponding page based on her crop method selection. She can also go to the previous page by tapping *left arrow* on the top-left corner of the page.

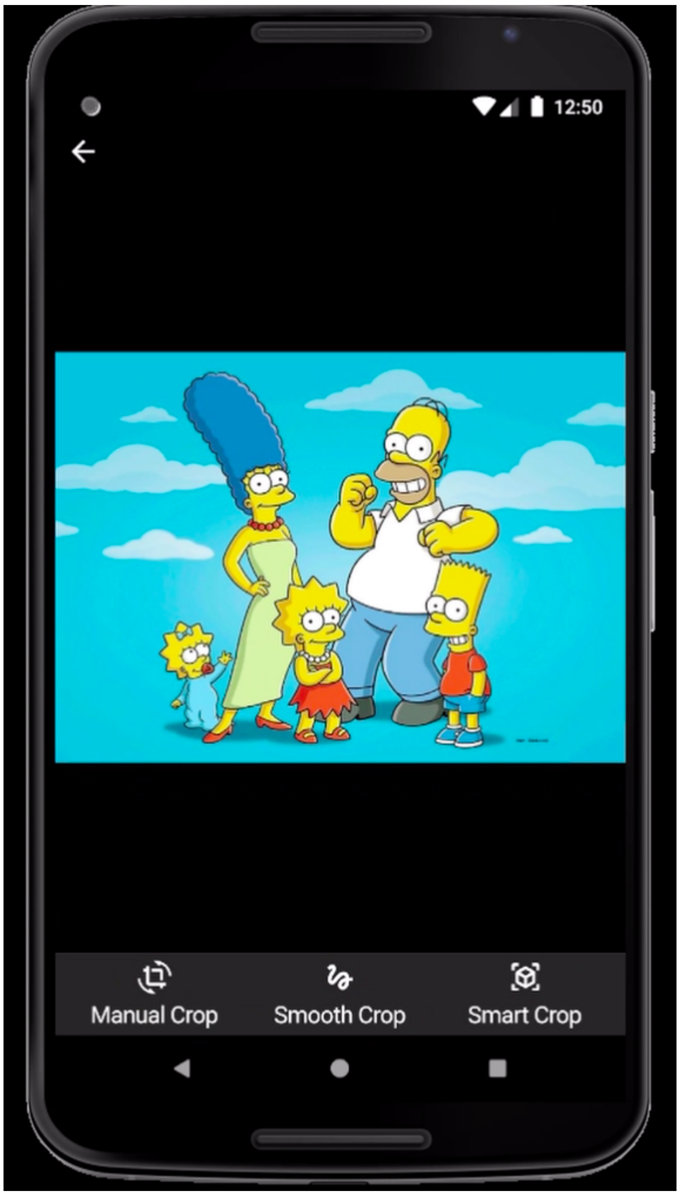


Figure 8: Crop Selection Page

Manual Crop Page

User is directed to this page if she chooses *Manual Crop* option in *Crop Selection Page*. *Manual Crop* requires user to select a rectangular region as shown in Figure 9. On the bottom of this page there are 5 buttons which are *Right*, *Left*, *Mirror*, *Reset* and *Save* respectively. User can tap *Right* button to rotate the image to right by 90 degress, or tap *Left* button to rotate left similarly or tap the *Mirror* button to mirror the image, or tap the *Reset* button to reset to initial position. After she decides on the rectangular region she taps the *right arrow* icon on the top-right corner to be directed to the *Merge Page*. Also the user can go to the previous page by tapping the *left arrow* icon on the top-left corner.

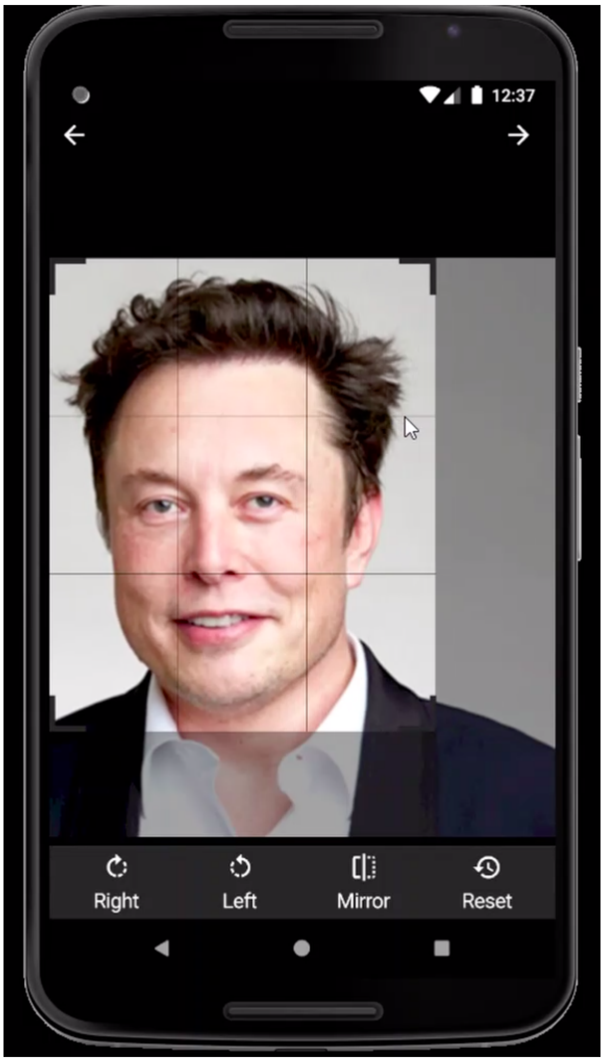


Figure 9: Manual Crop Page

Smooth Crop Image

User is directed to this page if she chooses *Smooth Crop* option in *Crop Selection Page.* For *Smooth Crop* user is required to draw the rough edges of the object. User can draw the boundaries of an object by hand. User can reset the drawing whenever she wants by tapping *reset* button on the bottom of the page as shown in Figure 10. If the drawing is finished user can click the *right arrow* icon on the top-right corner and the application then processes this drawing input to return a cropped area and user is directed to *Merge Page* along with this cropped area on the background image which is selected by the user beforehand. Also the user can go to the previous page with tapping the *left arrow* icon on the top-left corner.

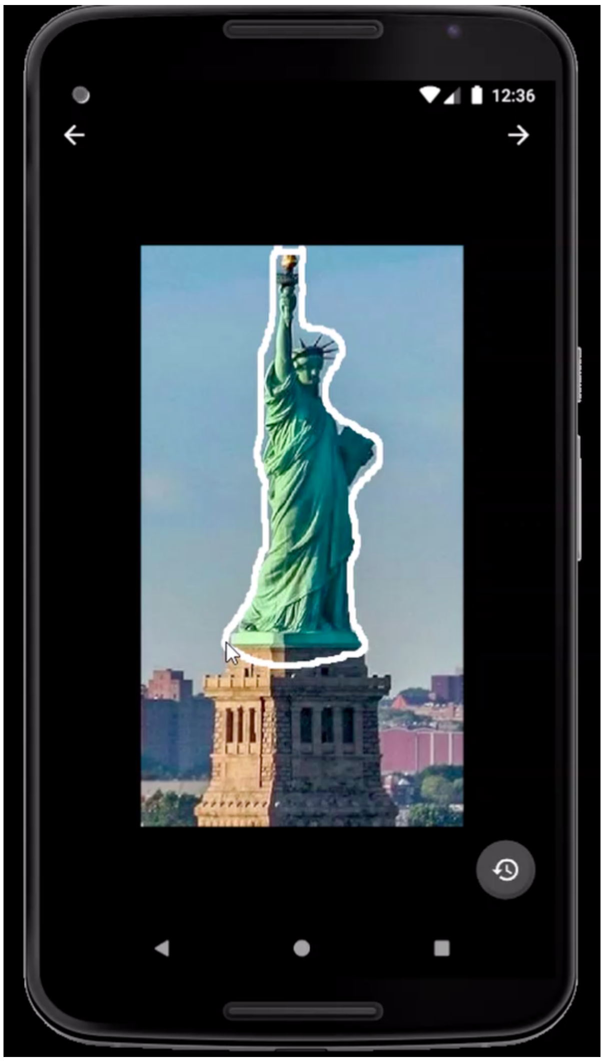


Figure 10: Smooth Crop Page

Smart Crop Image

User is directed to this page if she chooses *Smart Crop* option in the *Crop Selection Page*. *Smart Crop* requires a rectangular region from user that includes the desired object. This rectangular region can be accessed by tapping *Open Detector* button which is located at the bottom as shown in Figure 11. If she taps the *Smart Crop* button this input inside the detector will be processed by *Autoshop* server to detect the object inside the region and it will be cut from its edges then the result is taken and user is directed to the *Merge Page* along with this result of the *Smart Crop*. Also the user can go to the previous page by tapping the *left arrow* icon on the top-left corner.

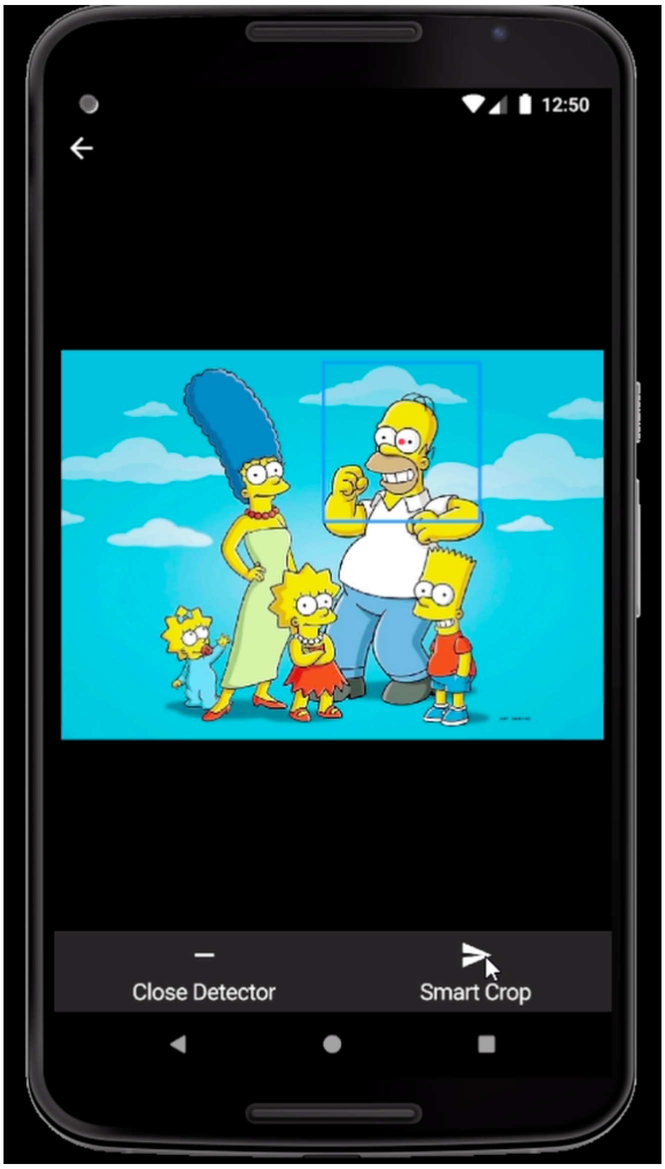


Figure 11: Smart Crop Page

Choose from Assets Page

User is directed to this page if she taps *Choose from Assets* button on the *Add Additional Image Page.* The user can access her previously created assets as shown in Figure 14 and is able to use them without further cropping. If she presses long enough on an asset, remove icon appears on that asset and if she tap this remove icon this asset will be deleted. The user can go to the previous page by tapping the *left arrow* icon on the top-left corner.

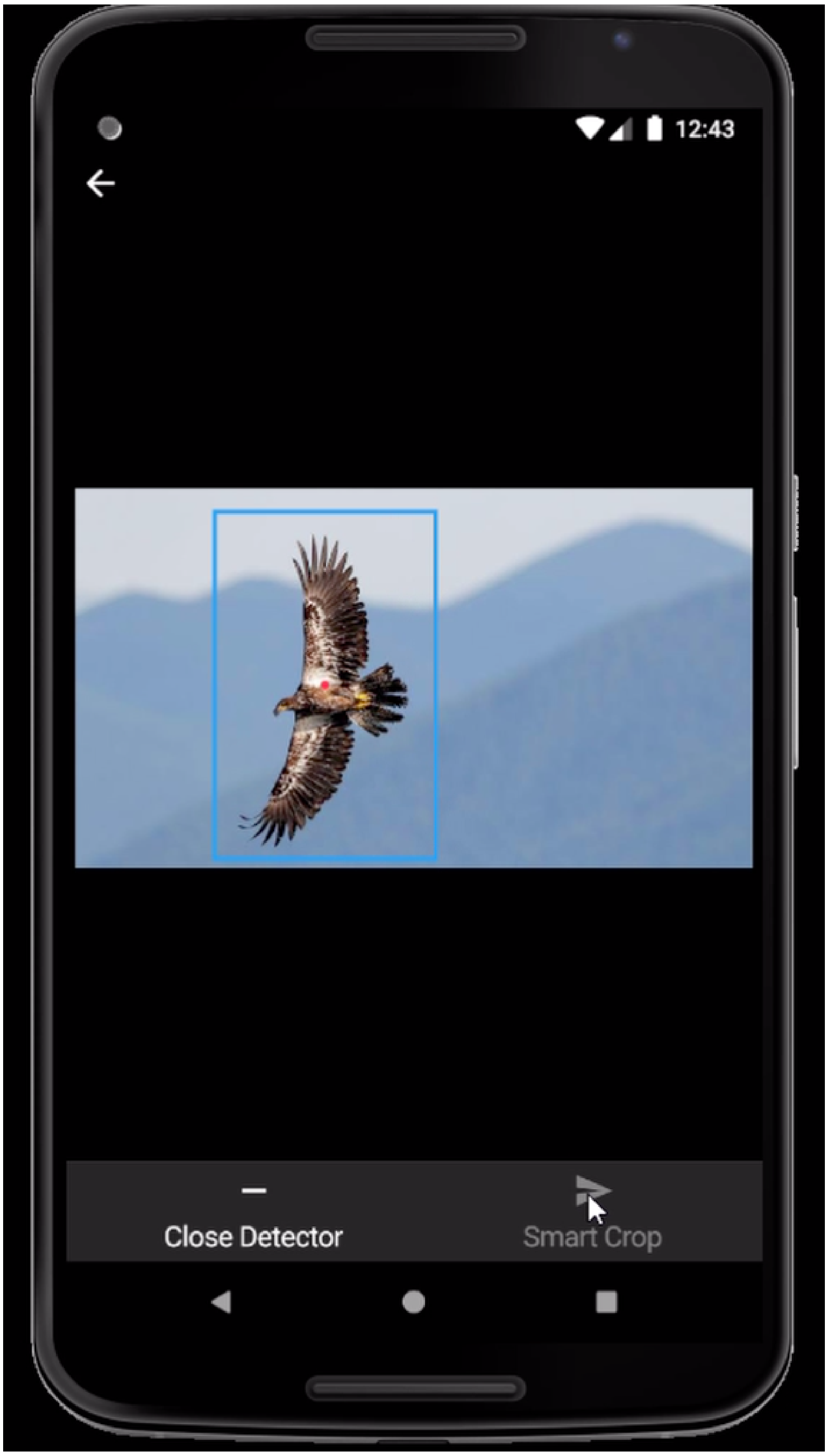


Figure 12: Asset to be Saved with Smart Crop

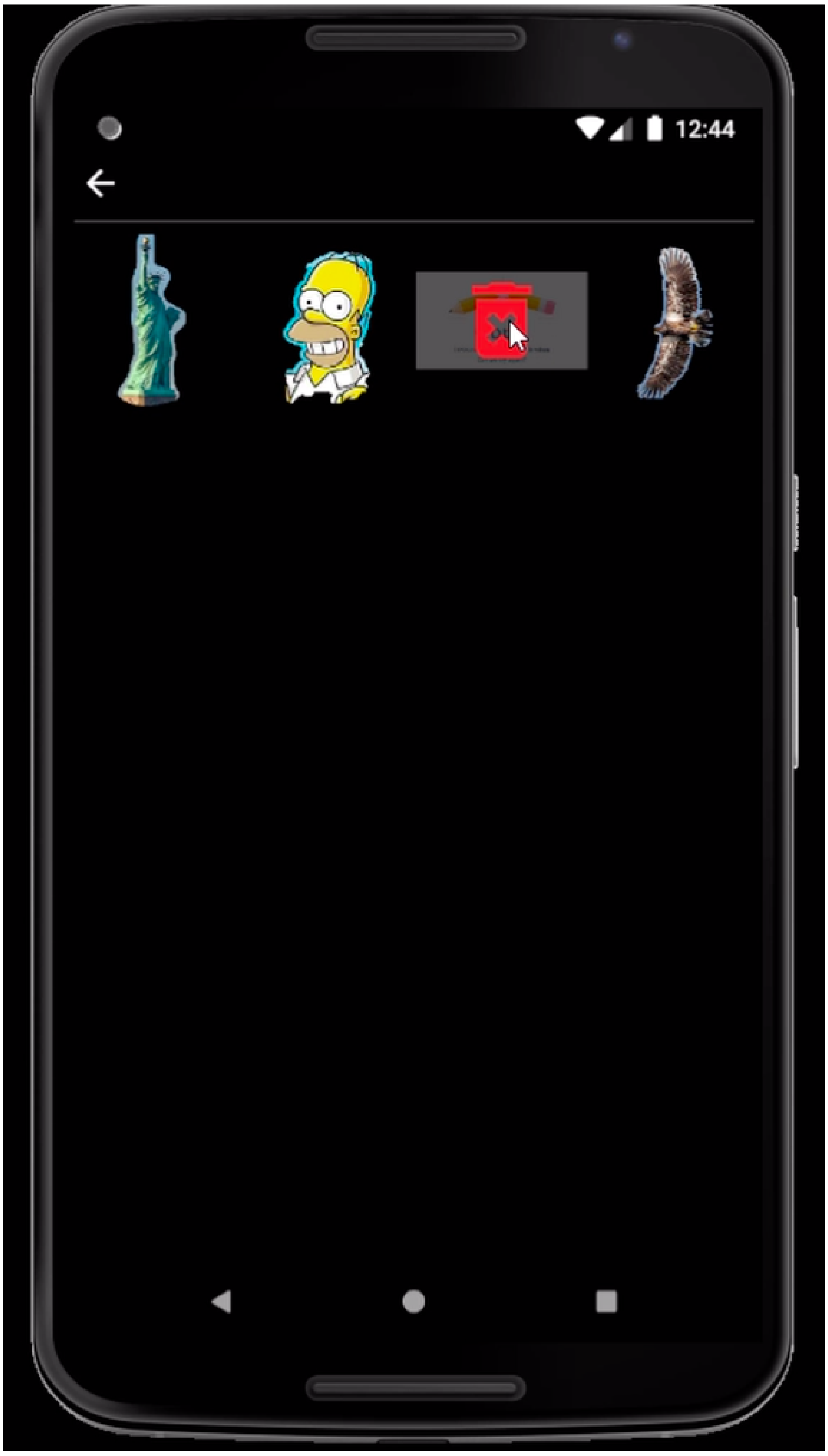


Figure 13: Asset Deletion

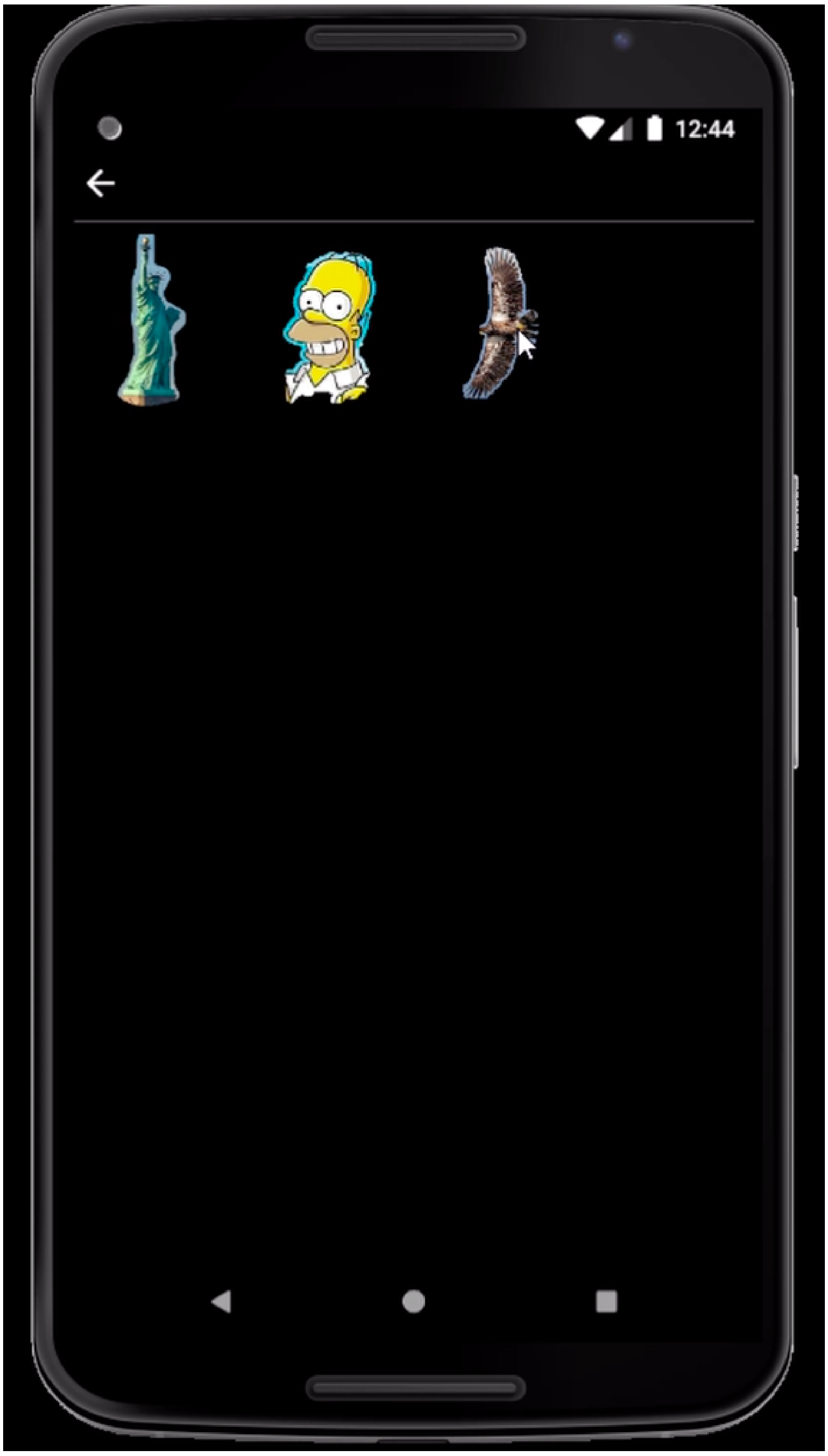


Figure 14: After deletion Choose From Assets Page

Merge Page

After having the *Background Image* and *Additional Image* user is directed to this page to decide on the scales of *Additional Image* also the relative position of that image, using tap gestures. Additional image comes from one of the crop pages or the assets and this additional image can be located anywhere in the background image by drag and drop. Also, user can enlarge, shrink or rotate this additional image by using pinch gesture with two finger. As shown in Figure 18, the user will tap *Apply Auto Add* button on the bottom to get the final image. The user can go to the previous page by tapping the *left arrow* icon on the top-left corner.

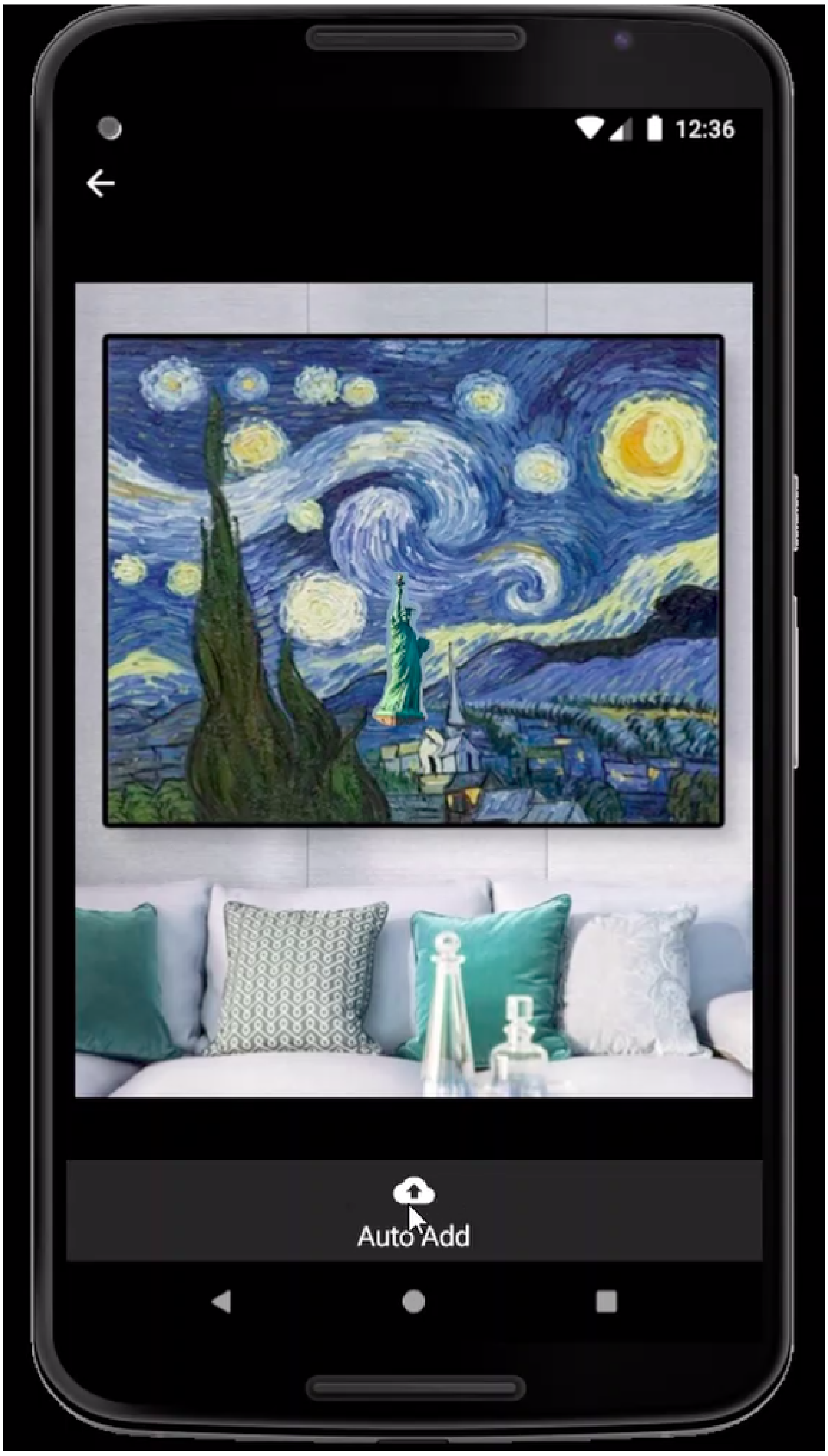


Figure 15: Merge Page with Smooth Crop

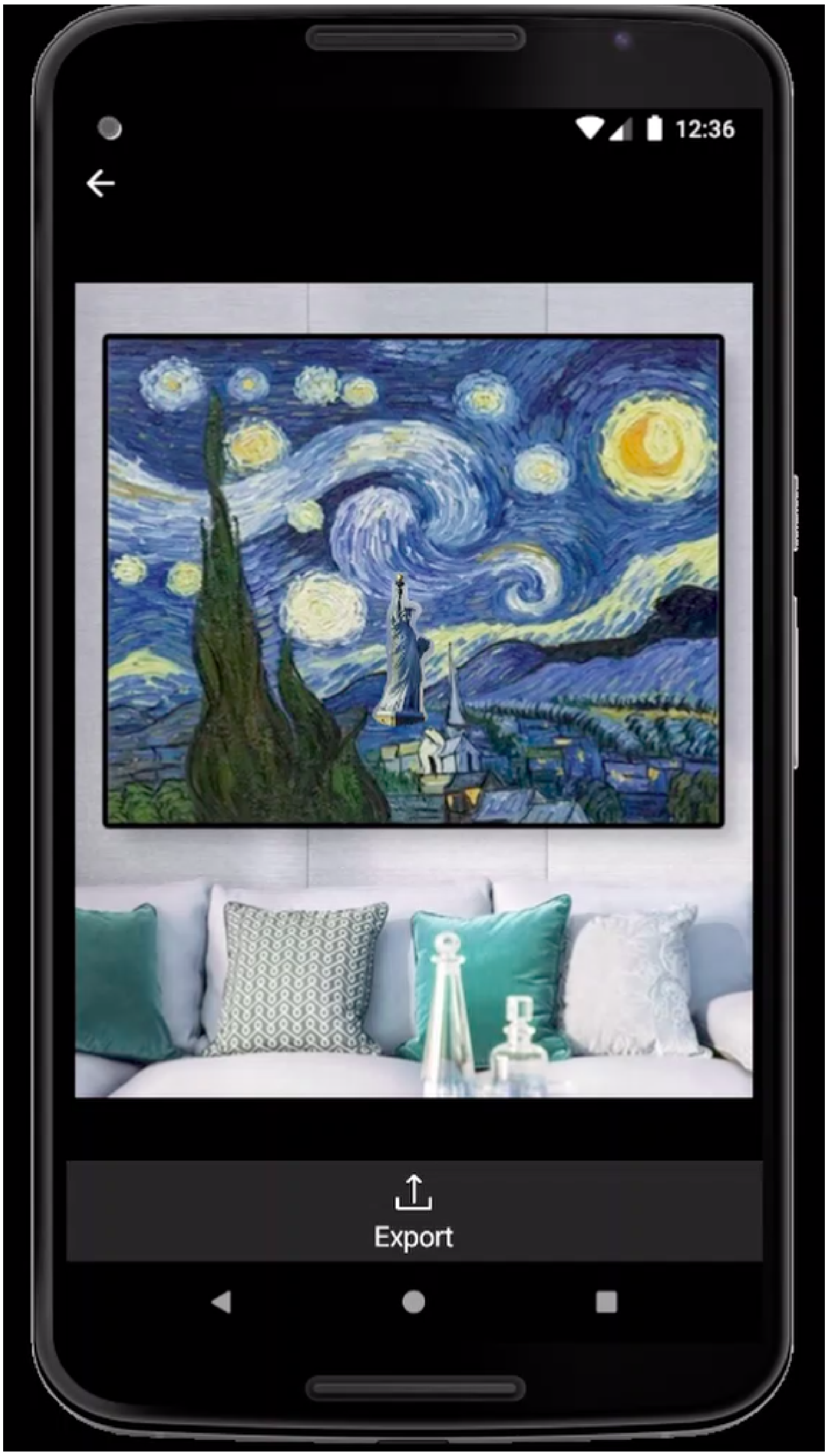


Figure 16: Result after Auto Add

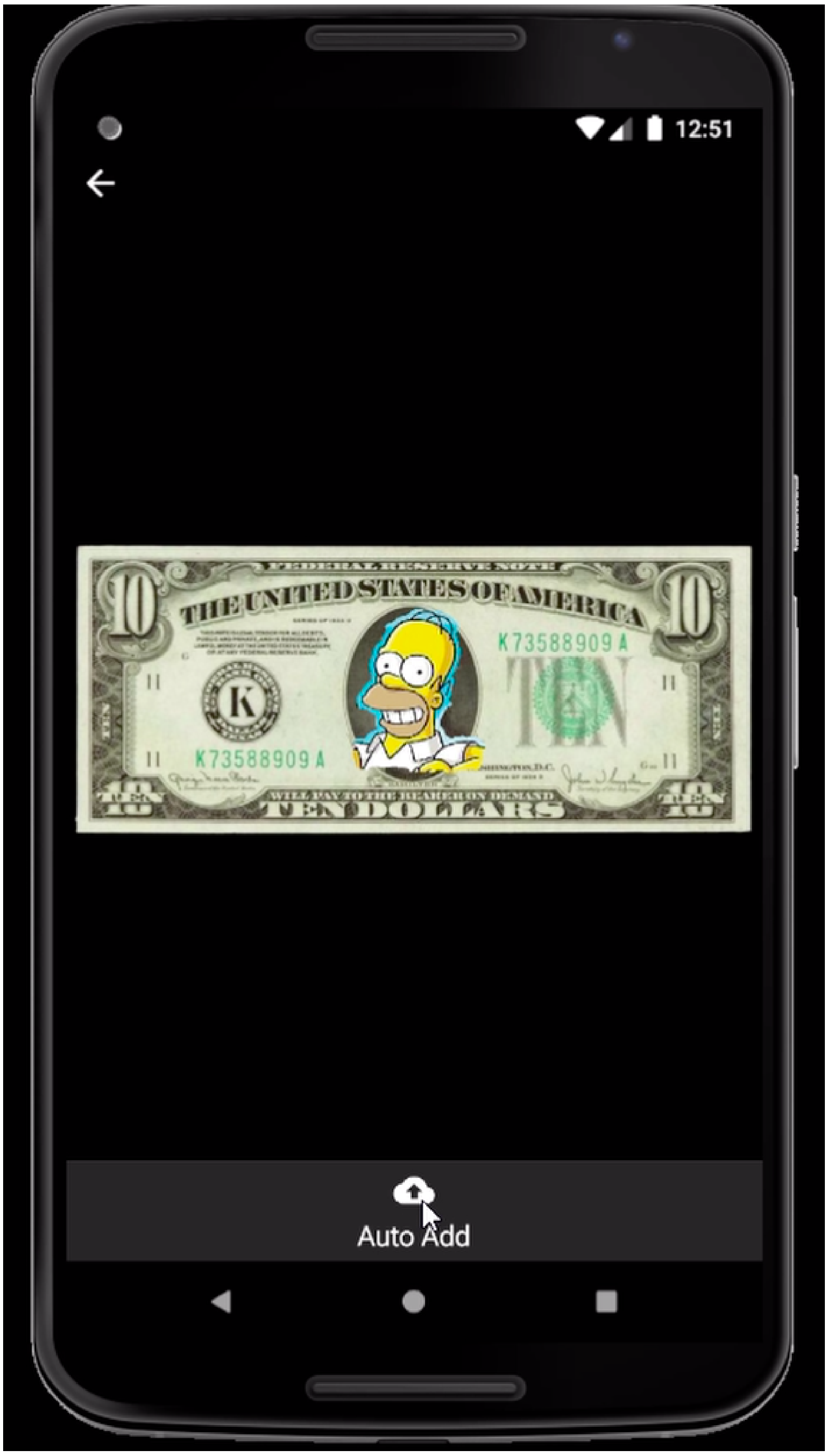


Figure 17: Merge Page after Smart Crop

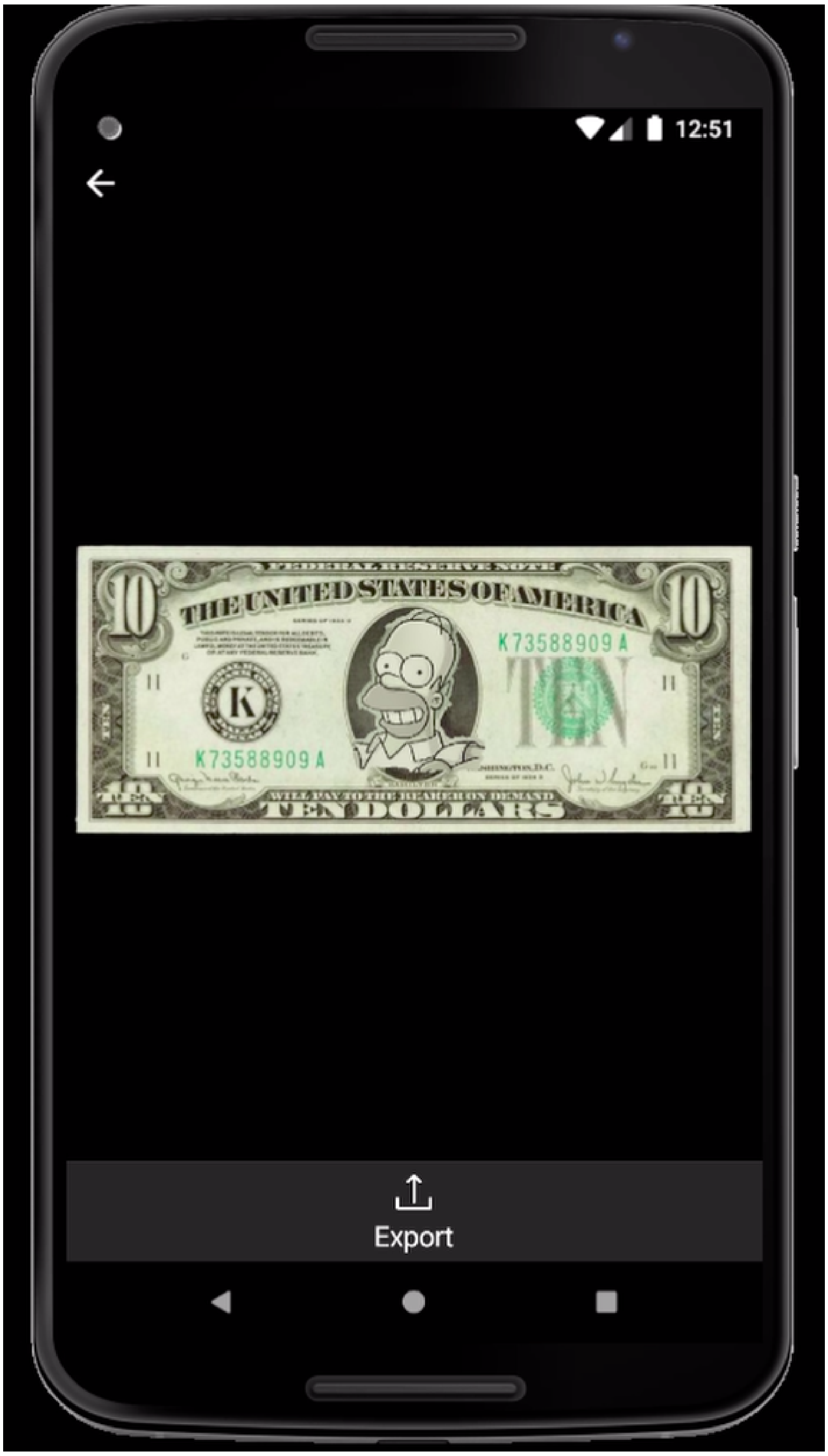


Figure 18: Result after Auto Add

Auto Remove Page

User is directed to this page if she chooses *Remove* option in *Add or Remove Page.* The user is required to scan (draw/brush a white mask on) the object to be removed with the brush at this page. The user can change brush size by the slider located at the bottom as shown in Figure 21. After scanning the object with the brush, user taps the *Apply Auto Remove* button to get the final image, then the user is directed to the *Export Page.* User can go to the previous page by tapping the *left arrow* icon on the top-left corner.

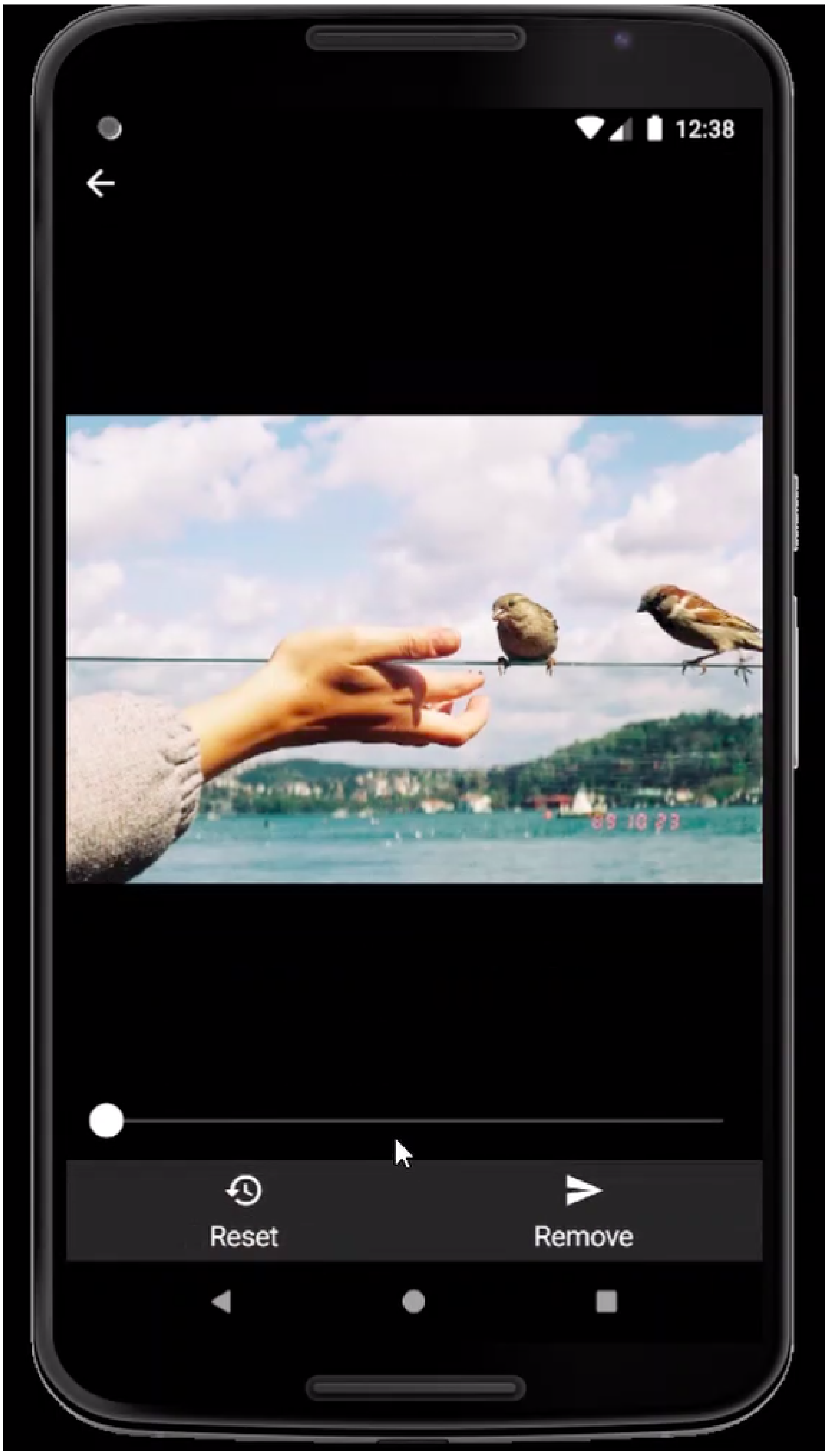


Figure 19: Auto Remove Page

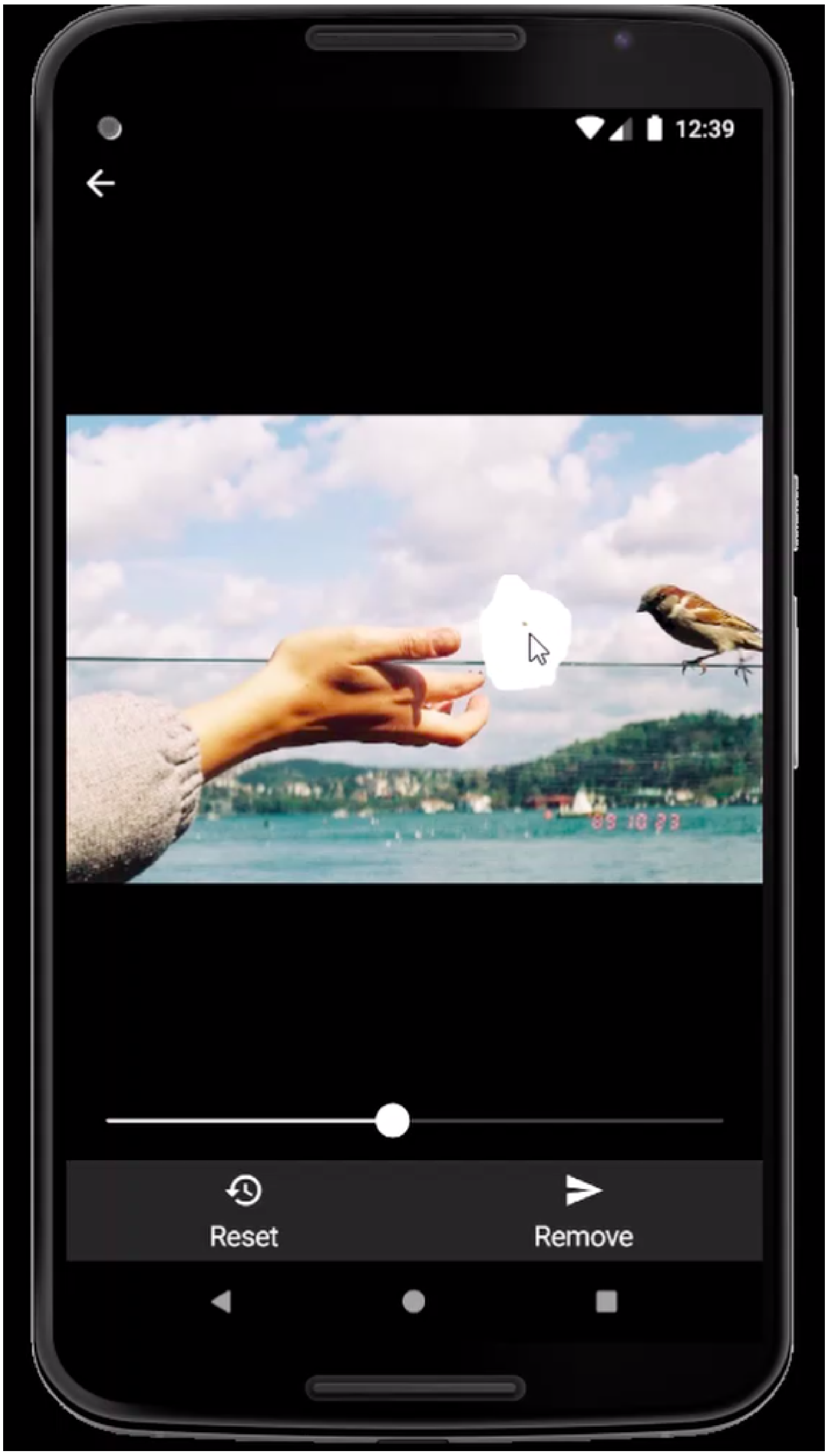


Figure 20: Brush in Auto Remove Page

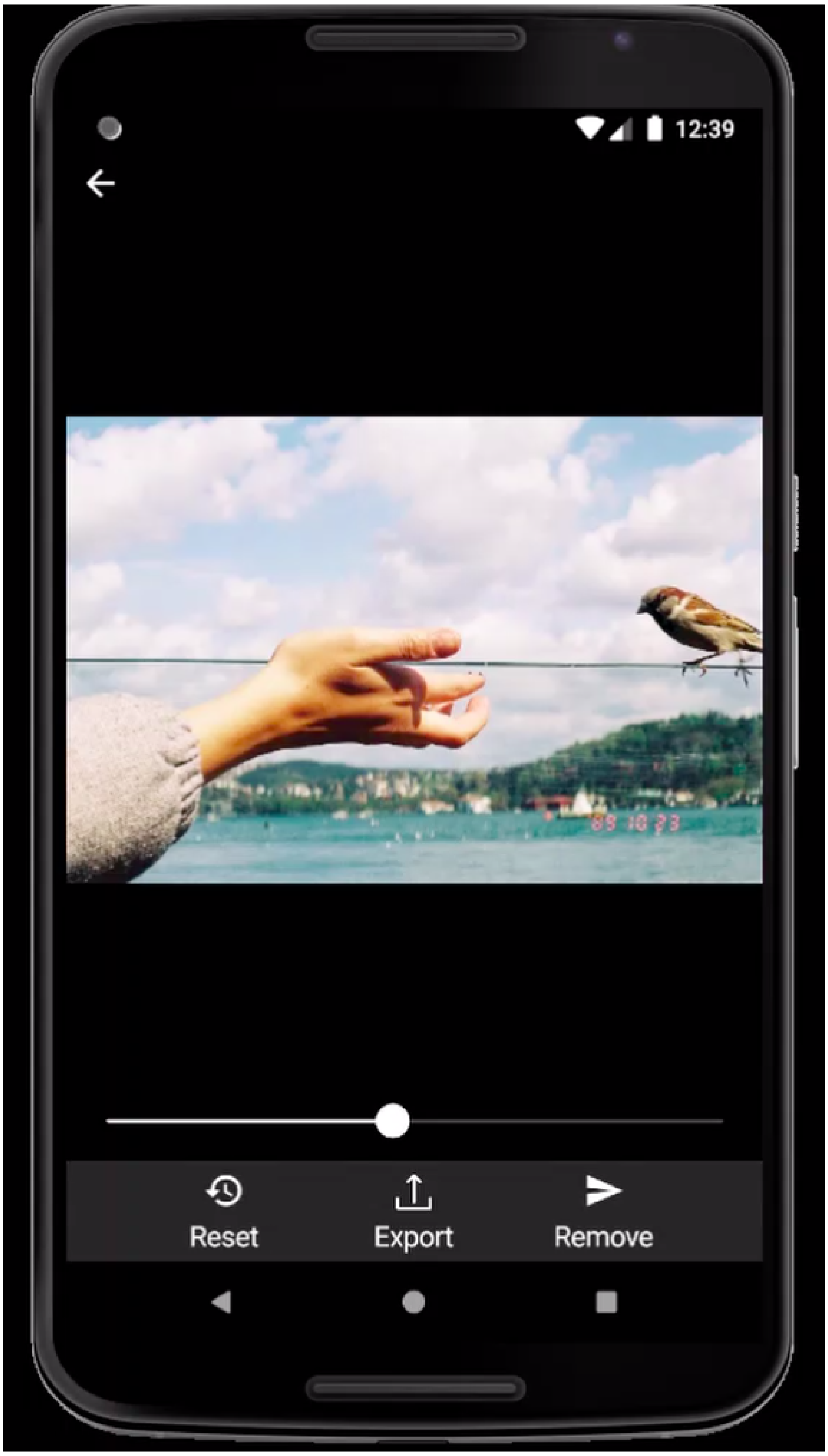


Figure 21: Result after Auto Remove

Export Button

With this button user can export the final output to social media platforms such as *Instagram*, *Whatsapp* with the buttons on the bottom of the page as shown in Figure 22. User can save the photo to gallery by tapping the *Save* icon. The user can go to the previous page by tapping the *left arrow* icon on the top-left corner.

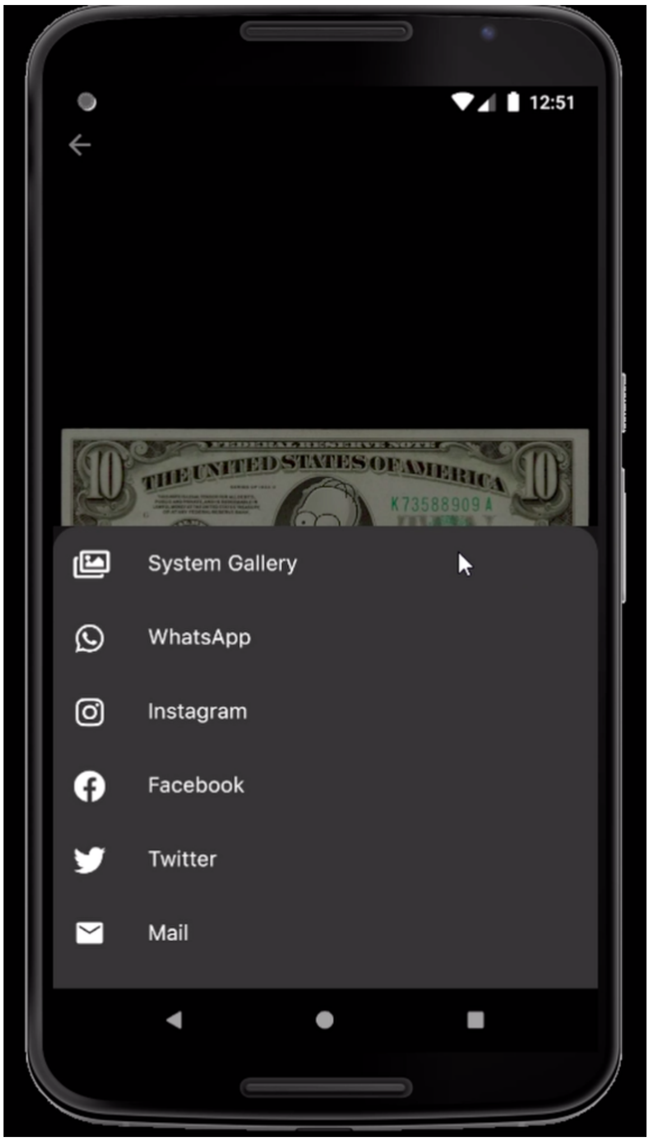


Figure 22: Export Page

10 Glossary

`FastPhotoStyle`	An NVIDIA library that is available in `Python`. Given a content photo and a style photo, the library can adjust the content photo according to the style of the style photo [3].
`ImageInpainting`	An NVIDIA library available in `Python`. The library can remove parts of the image and replace the removed parts with realistic artificial parts [6].
GPU	Graphics Processing Unit. A dedicated electronic circuitry designed to efficiently manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device [19]. GPUs are useful for deep learning applications since they are well-suited for frequently used operations such as matrix-multiplication and convolution.
Neural Networks	A set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns [20].
GAN	Generative Adversarial Networks. A model consisting of a generator and a discriminator, which aims to create new data instances that resemble your training data [21].
AWS	Amazon Web Services. Reliable, scalable, and inexpensive cloud computing services provided by Amazon. Remote servers can be rented through this service [22].
Agile Development	Agile software development refers to software development methodologies centered round the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams [23].
GDPR	General Data Privacy Regulation. A rule set concerning the protection of personnel data.
HTTP	Hypertext Transfer Protocol. HTTP is the protocol used to transfer data over the web. It is part of the Internet protocol suite and defines commands and services used for transmitting webpage data.
SDK	Software Development Kit. An SDK is a collection of software used for developing applications for a specific device or operating system.
UI	User Interface. The user interface (UI) is the point of human-computer interaction and communication in a device.

References

[1] D. W. Stout, Claire, J. Lyles, James, A. Sarkar, Barbora, Kyle, Betty, A. Brown, Robison, and et al., "Social media statistics: Top social networks by popularity," Jul 2019.

[2] "Functional requirements vs non functional requirements: Key differences." url=https://www.guru99.com/functional-vs-non-functional-requirements.html. [Accessed: 3- Nov- 2019].

[3] Nvidia, "Nvidia/fastphotostyle." `https://github.com/NVIDIA/FastPhotoStyle`, Feb 2019. [Accessed: 3- Nov- 2019].

[4] "Technical overview." `https://flutter.dev/docs/resources/technical-overview`. [Accessed: 27- May- 2020].

[5] "Technical overview." `https://flutter.dev/docs/development/ui/widgets-intro`. [Accessed: 27- May- 2020].

[6] "Remove unwanted objects." `https://online.theinpaint.com/`. [Accessed: 3- Nov- 2019].

[7] "Flutter." `https://flutter.dev/`. [Accessed: 26- May- 2020].

[8] "Deepfill." `https://github.com/JiahuiYu/generative_inpainting`.

[9] Y. Li, M.-Y. Liu, X. Li, M.-H. Yang, and J. Kautz, "A closed-form solution to photorealistic image stylization," 2018.

[10] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow." `https://github.com/matterport/Mask_RCNN`, 2017.

[11] S. "Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of IEEE International Conference on Computer Vision*, 2015.

[12] "Social media and psychology." `https://www.allpsychologyschools.com/psychology/social-media-psychology/`. [Accessed: 6- Nov- 2019].

[13] "General data privacy regulation." `https://eugdpr.org/`. [Accessed: 6- Nov- 2019].

[14] "Google play terms of service." `https://play.google.com/about/play-terms/index.html`. [Accessed: 6- Nov- 2019].

[15] “Graphics market share report.” https://www.datanyze.com/market-share/graphics. [Accessed: 3- Nov- 2019].

[16] “Adobe’s value.” https://producthabits.com/adobe-95-billion-saas-company/. [Accessed: 3- Nov- 2019].

[17] “Data center electricity consumption.” https://www.iea.org/tcep/buildings/datacentres/. [Accessed: 3- Nov- 2019].

[18] “Danger in deep fakes.” https://www.cnbc.com/2019/10/14/what-is-deepfake-and-how-it-might-be-dangerous.html. [Accessed: 3- Nov- 2019].

[19] “Gpu.” https://en.wikipedia.org/wiki/Graphics_processing_unit. [Accessed: 8- Nov- 2019].

[20] “Neural networks.” https://skymind.ai/wiki/neural-network. [Accessed: 8- Nov- 2019].

[21] “Generative adversarial netorks.” https://developers.google.com/machine-learning/gan. [Accessed: 8- Nov- 2019].

[22] “Amazon web services.” https://aws.amazon.com/. [Accessed: 8- Nov- 2019].

[23] “Agile development.” https://www.cprime.com/resources/what-is-agile-what-is-scrum/. [Accessed: 8- Nov- 2019].