



**Ain Shams University**  
**Faculty of Computer & Information Sciences**  
**Computer Science Department**

# Automatic Colorization Using Convolution Neural Networks

**July 2021**



**Ain Shams University**  
**Faculty of Computer & Information Sciences**  
**Computer Science Department**

# Automatic Colorization Using Convolution Neural Networks

**By:**

*Name*

*Department*

Eman Ossama Mohamed

Computer Science

Basem Maher Ramadan

Computer Science

Poula Atef Nashed

Computer Science

Tawfek Hesham Tawfek

Computer Science

Mohamed Samir Asaad

Computer Science

**Under Supervision of:**

**Dr. Maryam Al-Berry**

Scientific Computing Department,  
Faculty of Computer and Information Sciences,  
Ain Shams University

**TA. Marwah Helaly**

Computer Science Department,  
Faculty of Computer and Information Sciences,  
Ain Shams University

# Acknowledgement

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We are highly indebted to “Dr. Maryam Al-Berry” & “TA. Marwah Helaly” for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would also like to express our gratitude towards our parents & friends for their kind co-operation and encouragement which help us in completion of this project. We would like to express our special gratitude and thanks to our college “Faculty of Computer & Information Sciences, Ain Sham University” for giving us such attention, time & facilitate the way to gain such a knowledge.

Our thanks and appreciations also go to ourselves in developing the project and people who have willingly helped us out with their abilities.

# Abstract

In this project, we review some of the most recent approaches to colorize gray-scale images using deep learning methods. The study is based on the best-known deep learning algorithm, the CNN (Convolutional neural network). Inspired by these, we propose a model which combines a deep Convolutional Neural Network trained from scratch with high-level features extracted from VGG16 pre-trained model. Thanks to its fully convolutional architecture, our encoder-decoder model can process images of any size and aspect ratio. The model that developed taking the input in gray scale and predict the color of image based on the dataset that trained on it. The color space used in this work is Lab Color space the model takes the L channel as the input and the ab channels as the output. The ImageNet dataset used, and random selected image have been used to construct a mini dataset of images that contains 10,000 images are divided into 80% training and 20% testing. Other than presenting the training results, we assess the “Public acceptance” of the generated images by means of a user study. The proposed method has been tested and evaluated on samples images with accuracy= 82%. Finally, we present a carousel of applications on different types of images, such as historical photographs.

# Table of Contents

Acknowledgement .....	i
Abstract.....	ii
List of Figures .....	iv
List of Abbreviations .....	v
1- Introduction .....	1
1.1 Motivation.....	1
1.2 Problem Definition.....	1
1.3 Objective .....	1
1.4 Time Plan.....	2
1.5 Document Organization .....	3
2- Background .....	4
3- Analysis and Design.....	8
3.1 System Overview.....	8
3.1.1 System Architecture.....	8
3.1.2 System Users.....	9
3.2 System Analysis & Design .....	11
3.2.1 Use Case Diagram .....	11
3.2.2 Class Diagram .....	12
3.2.3 Sequence Diagram .....	12
3.2.4 Database Diagram.....	15
4- Implementation and Testing.....	16
5- User Manual.....	36
6- Conclusion and Future Work .....	49
6.1 Conclusion.....	49
6.2 Future Work .....	50
References .....	51

# List of Figures

Figure 1- Time Plane Gantt Chart.....	2
Figure 3.1.1- Three Tier architecture.....	8
Figure 3.2.1- Use Case Diagram.....	11
Figure 3.2.2- Class Diagram .....	12
Figure 3.2.3.1- Sequence Diagram for Registration .....	13
Figure 3.2.3.2- Sequence Diagram for Colorization.....	14
Figure 3.2.4- Database Diagram .....	15
Figure 4.1.4.1- VGG16 .....	21
Figure 4.1.4.2- Model1 .....	22
Figure 4.1.4.3- RELU function .....	22
Figure 4.1.4.4- Tanh function .....	23
Figure 4.1.4.5- MSE loss function.....	23
Figure 4.1.4.6- Inception Block1 .....	24
Figure 4.1.4.7- Residual Block1 .....	25
Figure 4.1.4.8- InceptionResNetV2 .....	25
Figure 4.1.4.9- Second Model Architecture.....	26

# List of Abbreviations

CNNs	Convolution Neural Networks
Conv Layer	Convolution Layers
DL	Deep Learning
FC	Fully Connected Layers
IDE	Integrated Development Environment
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LAB	Color Space
MaxPool	Max Pooling Layers
ML	Machine Learning
MSE	Mean Square Error Loss Function
NLP	Natural Language Processing
PC	Personal Computer
RELU	Rectified Linear Unit Activation Function
ResNet	Residual Neural Networks
SVM	Support Vector Machine
Tanh	Hyperbolic Tangent Sigmoid Function





# **1- Introduction**

## **1.1 Motivation**

Colorization is fundamentally an ill-posed problem mainly due to the loss of information across dimensions when a color image is converted to grayscale version. There is something uniquely and powerfully satisfying about the simple act of adding color to black and white image. Whether as a way of rekindling old, dormant memories or expressing artistic creativity, people continue to be fascinated by colorization. From remastering classic black and white films to the enduring popularity of coloring books for all ages, to the surprising enthusiasm for various automatic colorization bots online, this topic continues to fascinate the public. It normally requires human assistance to achieve artifact free color images. In this project an attempt has been made to come with methods to colorize images without human assistance.

## **1.2 Problem Definition**

Apply automatic colorization for gray-scaled images without any human assistance (input). The model tries to color different objects in the image as close as possible to the ground truth throughout set of steps implemented. It helps and takes place in many fields like: - movies coloring, health care, weather prediction, ...etc.

## **1.3 Objective**

The objectives of this research are:

- Build a deep learning model to color gray scale images without human assistance.
- Build a classification model instead of a regression model to give an efficient testing rate.
- Train model on a large dataset to capture salient information from images.
- Use Autoencoders & Pre-trained model to improve accuracy.

## 1.4 Time Plan

- 1- Surveying existing studies and papers about different deep learning models.
- 2- Choose feasible dataset “test – train – validation” parts and decide which tools will be used.
- 3- Specify the functional and non-functional requirements.
- 4- Build deep learning model “Implementation phase”.
- 5- Test final model.
- 6- During each phase, the steps and output results will be documented.

Project Activities	Start Date	End Date
Survey	1/11/2020	30/11/2020
Requirement Specifications	1/12/2020	8/12/2020
Project Analysis		
Use Case	9/12/2020	16/12/2020
ER Diagrams	17/12/2020	24/12/2020
Sequence Diagram	25/12/2020	31/12/2020
Project Design		
Project Architecture	1/1/2021	15/1/2021
User Interface Design	16/1/2021	31/1/2021
Project Implementation	1/2/2021	1/5/2021
Project Testing		
Modules Testing	2/5/2021	1/6/2021
Modules Integration	2/6/2021	30/6/2021
Project Documentation	1/11/2020	30/6/2021

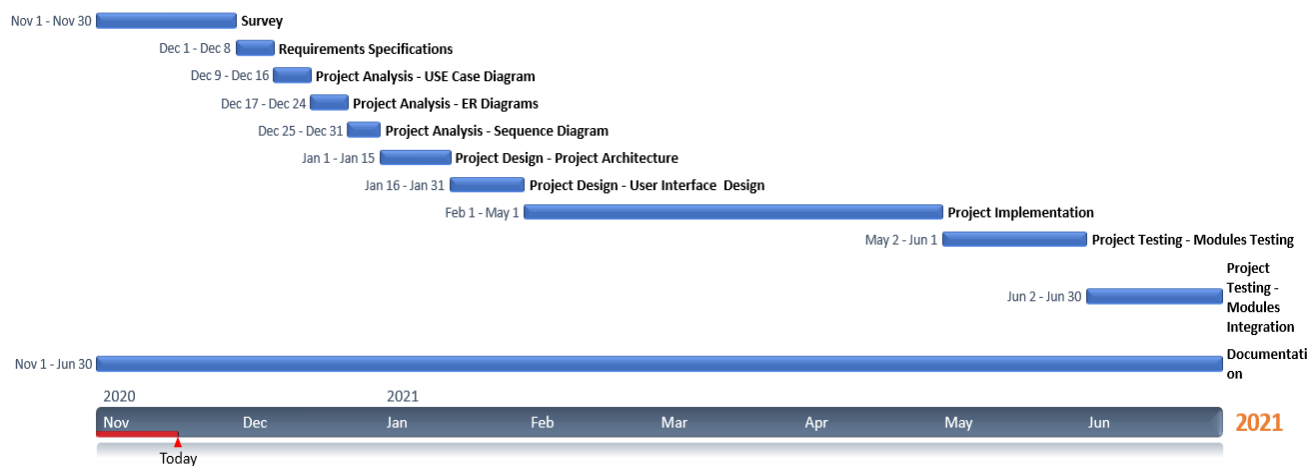


Figure 1 Time Plan Gantt Chart

## 1.5 Document Organization

### *Chapter 2: Background*

In this chapter, we describe the following:

- A detailed description of the field of the project.
- All the scientific background related to the project.
- A survey of the work done in the field.
- Description of existing similar systems.

### *Chapter 3: Analysis & Design*

In this chapter, we describe the following:

- System overview through some diagrams “3 Tier Architecture”.
- System architecture.
- A detailed description for user functional requirements & non-functional requirements.

### *Chapter 4: Implementation & Testing*

In this chapter, we discuss the following:

Part one: Implementation:

- Tools & Challenges.
- Detailed description for the pre-processing phase.
- Detailed description for different architectures for autoencoders & decoders.
- Detailed descriptions for the pre-trained models used.

Part two: Testing:

- Detailed description for the testing phase.
- List of all trials to improve the model’s generalization.

### *Chapter 5: User Manual*

In this chapter, we describe the following:

- A detailed description using screenshots from the interfaces for the user to learn how to deal and interact with the system whether in the desktop application or in the mobile application.

### *Chapter 6: Conclusion & Future Work*

In this chapter, we describe the following:

- Provide a summary of our results obtained from several trial and what can be done in the future to improve the accuracy and performance of the model and what additional functions could be added.

## **2- Background**

### **2.1 Description of Project Field**

This project is not targeting a special field, as it may be helpful in many fields and tracks as mentioned before in the introduction. But the main goal is to change an image's color space or channels from one "Gray-Scaled" to "RGB" one by the use of "LAB" color space. This can be done through different technologies as ML algorithms or DL models.

### **2.2 Scientific Background**

The scientific background in the project is totally based on deep learning models. Specially, CNNs which have an instrumental role in many fields as object recognition, image segmentation, motion detection, ...etc. we decided to use CNNs as it must shorten the way in many requirements as feature extraction. The role of first layers in CNNs to extract primitive features as edges and lines the deeper layers the finer and higher features are extracted. We need a high level of features to recognize objects to be colored by the most suitable color to be realistic and closer to the ground truth as much as possible.

### **2.3 Survey of Work Done**

Though CNNs was not entirely new technology, it had taken the world by storm since 2012. CNNs now developed which are utilized in different digital applications were first observed in living organisms. In the 1950s & 1960s, Hubel & Wiesel have worked on cats and monkeys which showed that their visual cortexes contained neurons that separately reciprocate to small areas of the visual field [1]. The visual stimulus in visual space affects the firing of a single neuron provided their eyes are not moving. This is its receptive field. Intercepting receptive regions have been observed in the neighboring cells. Each hemisphere in the cortex represents the contralateral visual field. All these formed basics to the first ever documented commercial use of CNNs which dates to 1998, with LeNet-5. LeNet-5 was designed by LeCun et al.

Years of researching CNNs, made it possible to recognize text character based on 32x32 pixel images. Since then, CNNs have dominated this area of computer vision and surpassed results obtainable by other machine learning methods. But the large-scale application of CNNs were not possible until a decade later.

In 2012, when the CNN based model entry of “Alex Krizhevsky et al” and their AlexNet in ImageNet Large Scale Visual Recognition Challenge won that year image classification challenge by a significant margin, it took most of the computer vision research community by surprise and lit a huge amount of interest. This event made CNN a staple in the computer vision field and many others. Countless problems, such as image recognition, facial recognition, video sequence tracking, automatic image segmentation, handwriting to text conversion, natural language processing have been made possible to solve easily with the help of CNN [4].

CNNs have proven to be functioning as automatic data encoders. They can learn very complex mappings of inputs to outputs from huge amounts of data. The 16 layers deep VGG-16 and the 22 layers deep GoogLeNet models which were introduced in the year 2014, have excelled the projected human error in the image classification task on the ImageNet dataset.

In 2015, A year later, Microsoft Research of Asia has introduced an alternative architectural approach to traditional convolutional networks, called residual networks (shortened to ResNets), achieving state-of-the-art accuracy on ImageNet classification [3]. This architecture had allowed the team to significantly increase the depth of their networks, the best performing model consisted of 152 layers. This research aims to project a methodology to convert a continuous series of grayscale images(frames) of a video and then assemble them together to form a colored video. To make the process faster and easier a separate and huge dataset is allocated through which the learning efficiency of CNNs also increase.

The following table summarizes the survey:

<b>Paper Name</b>	<b>Dataset</b>	<b>Method</b>	<b>Results</b>
Image Colorization with CNN Stanford 2020	MIT CVCL Urban & Natural Scene	CNN with regression-based model	Unsaturated images
Real Time User Colorization California 2017	ImageNet dataset	CNN with classification & user sparse input	Deals with user incorrect input

Image Colorization Russian 2017	ImageNet dataset	Classification model with CNN & ResNet layers	Accuracy 75.7%
Gray-Scaled Image Colorization using ML Techniques 2015	Subset of the colored images made available by Jegou, Douze and Schmid	Classification using SVM	SVM is better than Linear Logistic Regression in colorization
Colorful Image Colorization California 2017	ImageNet dataset	Classification & Segmentation with CNN & VGG16	Indistinguishable from real color photos

## 2.4 Description of Existing Similar Projects (Related Work)

Our project was inspired by a CNN based system which automatically colorizing images. This system relies on several layers of ImageNet-trained from VGG16, which will be integrating with a system which is an autoencoder which has residual connections which helps in merging intermediate outputs that are generated by the encoding part of the network comprising the VGG16 layers, with those generated by the latter decoding part of the network. The inspiration behind the residual connections is the ResNet system built by “He et al” that won the 2015 ImageNet challenge. As the connections are used to link upstream network edges with downstream network edges, they allow swift of gradients with the help of the system, so that training convergence time gets reduced and it also enables more reliably in training more deeper networks. Veritably, the system reports on a much larger scale decreases training loss on each training iteration with the help of his most recent system when compared to an earlier variant that could not utilize residual connections.

As per the results, the system carries out extremely well to make it realistic. We nonetheless discern that in many cases, the images which are produced by the system are mainly sepia-toned and muted in color. Image colorization is formulated as a problem of regression but the training goal to be minimized is a sum of Euclidean distances between each pixel's that are blurred color channel values in the target image as well as predicted image. While regression does seem to be properly suited to the task as it shows continuous nature of color spaces, practically, an approach based on classification may work better. To understand why, consider a pixel which exists in a flower petal across multiple images that are identical, save for the color of the flower petals. The taken pixel can take various tones or colors of red, yellow, blue, and many more. With a system which is regression-based that uses an  $\ell_2$  loss function, the value of the predicted pixel minimizes the loss for this pixel is the mean pixel value.

Accordingly, the predicted pixel ends up being an unattractive, subdued mixture of the possible colors or tones. Taking this scenario into consideration, we hypothesize that a system which is regression-based would tend to generate images which are desaturated and impure in color tonality, especially for the objects that which take on many colors in the real world and is the reason behind lack of punchiness in color in the images colorized by the system. So, if we consider the system as a classification-based would tend to generate a saturated image.

## 3- Analysis and Design

### 3.1 System Overview

#### 3.1.1 System Architecture 3 Tier Architecture

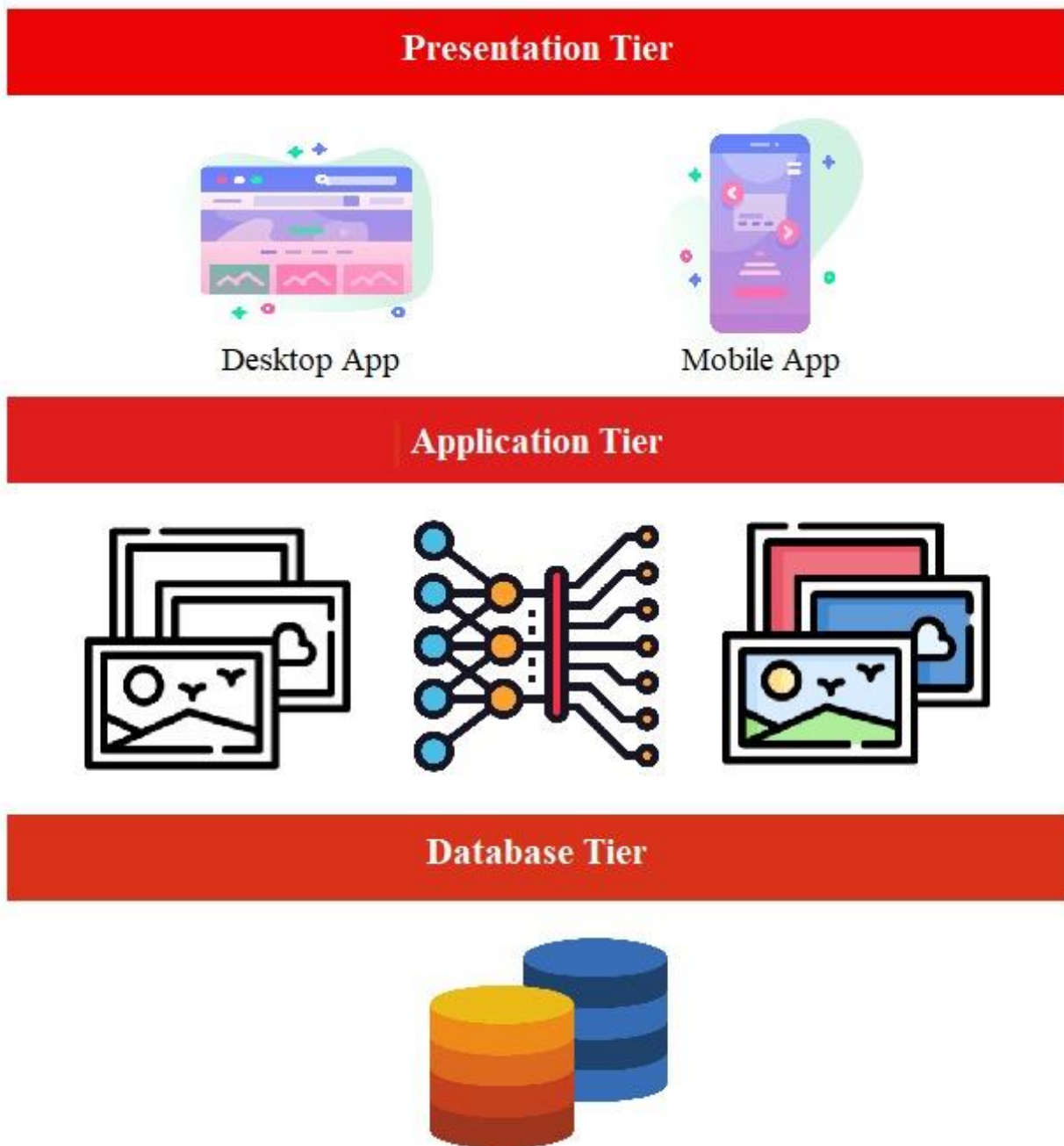


Figure 3.1.1 Three Tier Architecture 1



- Presentation Tier:

It is the front Tier in the system architecture which includes the user interface and responsible for any interaction between the user and the system.

The user can interact with the system by selecting a gray-scaled image to be colorized.

- Application Tier:

It is the tier contains the main functions responsible for the colorization process. It takes the gray-scaled images from the presentation tier and interact with the below tier which is the database tier and pass the image to the deep learning model to return the colored image. Both other tiers cannot interact with each other without application tier.

- Database tier:

It is the tier responsible for storing and retrieving data when needed. We store the weights of our fine-tuned model to help in a faster testing process.

### 3.1.2 System Users

#### *A. Intended Users:*

The system's target audience is any person who has gray-scaled image wants to be colorized.

#### *B. User Characteristics*

Our system is designed to be user friendly to allow any user to use the system easily without and pre-requisite knowledge.

## Functional Requirements:

### User Requirements:

- 1- User can register or login.
- 2- User can browse his desktop storage or phone storage.
- 3- User can select image to be colorized.
- 4- User can view his history of colored images.

### System Requirements:

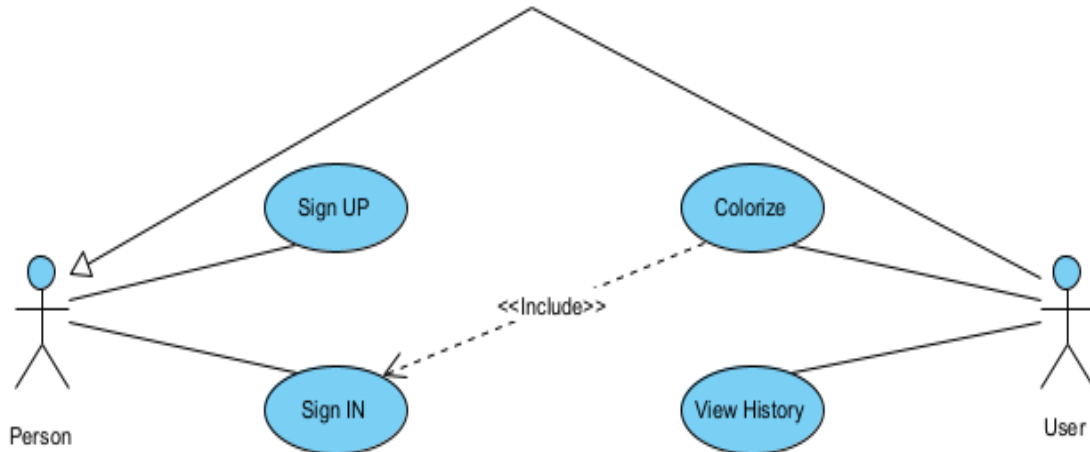
- 1- System asks to select a gray-scaled image.
- 2- System applies pre-processing steps on the desired image.

## Non-Functional Requirements:

- Security & Authentication:  
System verifies each user logged in through username and password.
- Availability:  
The system is available all the time for all users.
- Performance:  
A fast response to the user is granted.
- Portability:  
System is portable which means easy of transfer from an environment to another.
- Scalability:  
The system can provide its services to many users at any time.
- Reliability:  
The system performs in a consistent way.

## 3.2 System Analysis & Design

### 3.2.1 Use Case Diagram

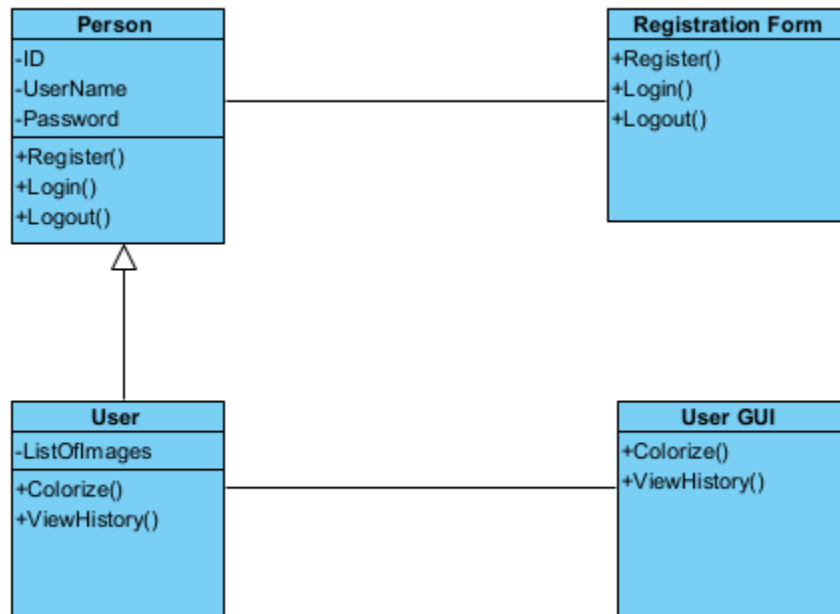


*Figure 3.2.1 Use Case 1*

#### **Use case notations' description:**

- Actor person initiates two use cases which are sign up and sign in to be able to log to the system.
- Actor user initiates two other use cases which are the main functions in the system colorize & view history. This actor is inherited from the person actor so he must be logged in the system to gain the functions of being user.
- Colorize function must include sign in function to make sure that the user is signed in before applying the function.

### 3.2.2 Class Diagram

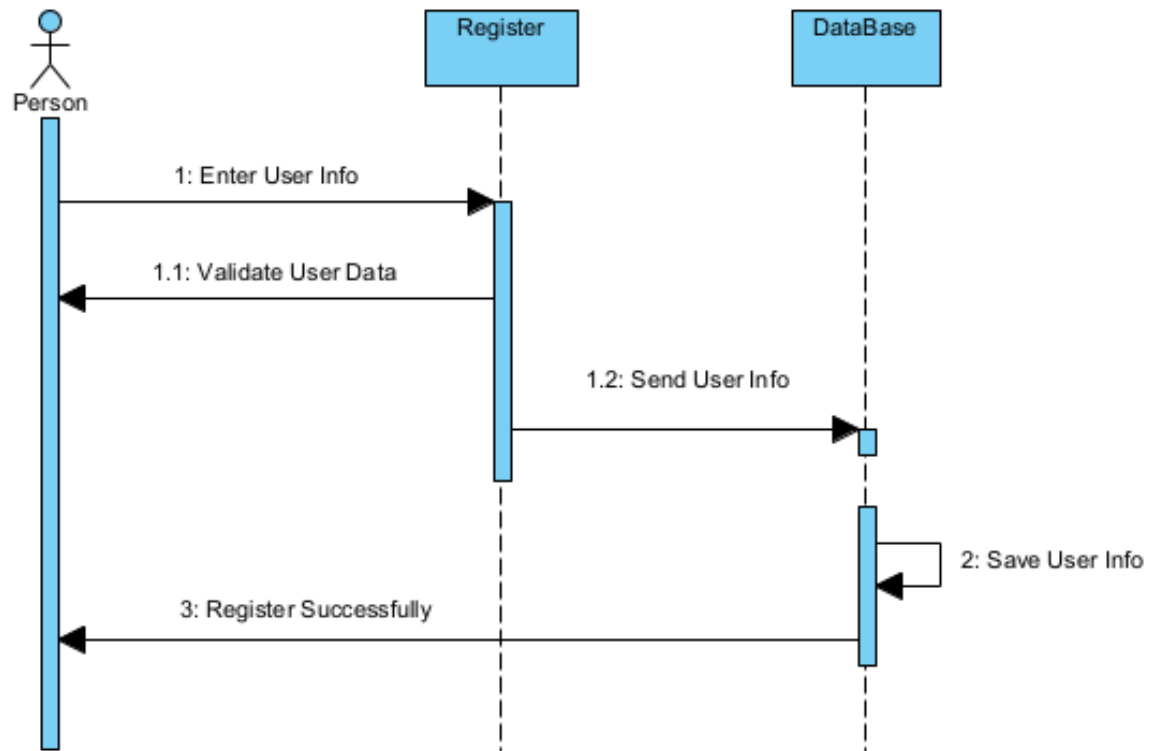


*Figure 3.2.2 Class Diagram 1*

This class diagram represents the data requirements of the system. It contains 2 Entities classes & 2 Boundary classes.

- Entity class person: - stores the data for anyone uses the system that could be a user due to inheritance relationship “Generalization”.
- Entity class user: - stores the data for the user to save the history-colored images.
- Boundary class registration form: - controls the interactions between the person wants to log into the system and the system to help him register or login to the system.
- Boundary class user GUI: controls the interactions between the user and the system to help users colorize their gray-scaled images and view their history.

### 3.2.3.1 Sequence Diagram for Registration



*Figure 3.2.3.1 Sequence Diagram 1*

This sequence diagram is based on the class diagram. It describes the process of user registration. This process is done through the interaction between different objects type. The first object always deals with the user boundary class interface to take the user input then proceed in the next processes.

### 3.2.3.2 Sequence Diagram for Colorization

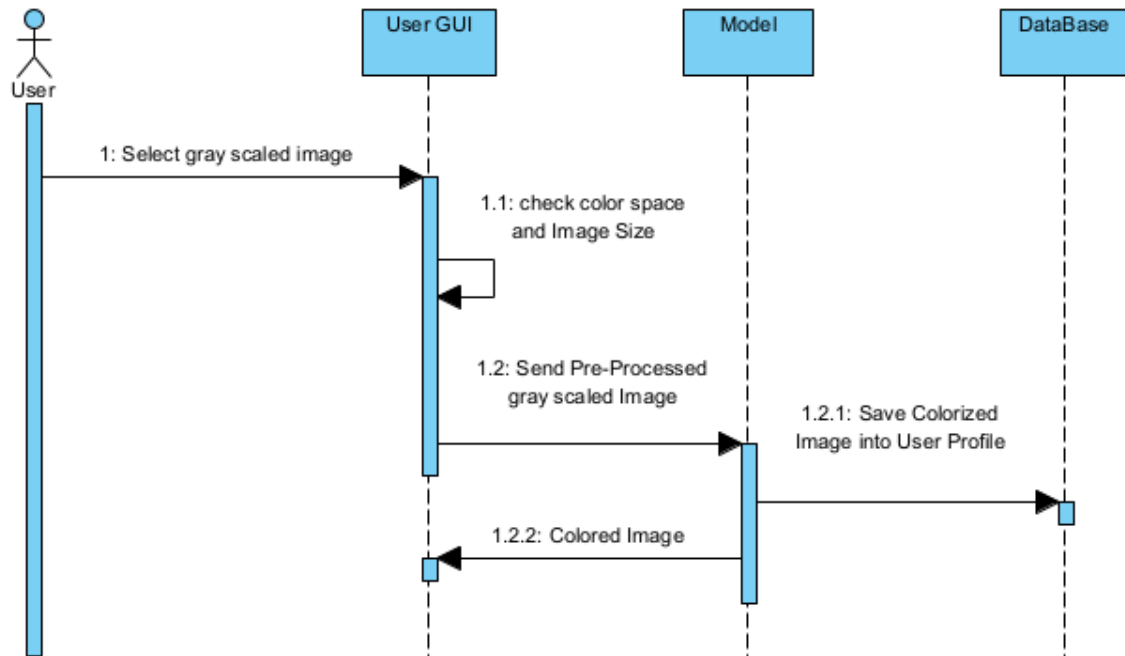
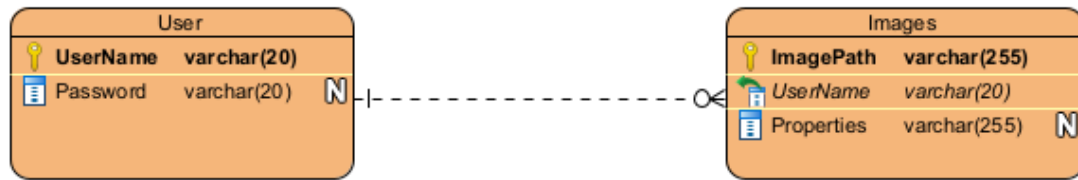


Figure 3.2.3.2 Sequence Diagram 2

This sequence diagram is based on the class diagram. It describes the process of user colorization for a gray-scaled image. This process is done through the interaction between different objects type. The first object always deals with the user boundary class interface to take the user input in this case it is the user image then proceed in the next processes and pass the image to the model to return the colored image to the user at the end.

### 3.2.4 Database Diagram



*Figure 3.2.4 Database Diagram 1*

Our database contains only 2 tables:

- User table: username is a primary key to uniquely discernment users, password to allow system authentication & list of colorized images as a history.
- Images table: image path is the primary key to uniquely discernment images, username as a foreign key to provide the relation between both tables “1 to many” that each one user can have many images in his profile.
- In the mobile application: Firebase database is used while in desktop application a file system is used.

## **4- Implementation and Testing**

### **Part One: Implementation**

#### **4.1.1 Dataset Description, Tools & Challenges**

##### **Tools & Development Environment:**

**1- Python:**

is a general-purpose programming language, so it can be used for many things. Python is used for web development, AI, machine learning, operating systems, mobile application development, and video games.

**2- PyCharm IDE:**

is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

**3- Google Collaboratory:**

a web IDE for python, to enable Machine Learning with storage on the cloud — this internal tool had a quiet public release in late 2017 and is set to make a huge difference in the world of machine learning, artificial intelligence, and data science work.

**4- Keras Library:**

is a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.



5- TensorFlow:

It is an open-source artificial intelligence library, using data flow graphs to build models. It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

6- Tkinter:

is a standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications.

7- Java:

Java is a programming language and computing platform first released by Sun Microsystems in 1995. Java is fast, secure, and reliable.

8- Android Studio IDE:

Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug.





## Challenges:

- 1- Hard disk storage to store huge data to train the model
- 2- High computational power needed in GPUs to deal with images
- 3- To solve the problem of GPUs. We used Google CO-LAB so, we faced problem with internet connection & speed and limited GPU size.

## Dataset Description:

- Different classes from ImageNet dataset with almost 10,000 images used to fine-tune the model.
- Classes contains images with different objects, animals & natural scenes.

### 4.1.2 Pre-Processing Phase

Pre-processing is a crucial step that helps enhance the quality of data to promote the extraction of meaningful insights from the data. Data preprocessing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. In simple words, data preprocessing in Machine Learning is a data mining technique that transforms raw data into an understandable and readable format.

The main goal of this phase is to provide the model with data more consistent and accurate to give high accuracy results. So, we used different pre-processing techniques, such as:

#### 1- Data augmentation:

At first, we faced the problem of poor dataset due to the small size. It was around 500 images only so, we needed to apply data augmentation which means apply some geometric transformations as translation, rotation, scaling, flipping, cropping, etc, to increase the size of the dataset.

## 2- Feature Scaling or Standardization:

It is a step of data Pre-Processing which is applied to independent variables or features of data. It basically helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

## 3- Splitting the dataset:

Every dataset for Machine Learning model or Deep Learning model must be split into two separate sets – training set and test set. Training set denotes the subset of a dataset that is used for training the machine learning model. Here, you are already aware of the output. A test set, on the other hand, is the subset of the dataset that is used for testing the machine learning model. The ML model uses the test set to predict outcomes. Usually, the dataset is split into 70:30 ratio or 80:20 ratio. This means that you either take 70% or 80% of the data for training the model while leaving out the rest 30% or 20%. The splitting process varies according to the shape and size of the dataset in question.

We worked out with all of these different techniques, but we had a dataset large enough, so we removed out data augmentation and split data with ratio 80:20 which means 8,000 images for training & 2,000 images for testing.

## 4.1.3 Different Architecture for Autoencoders & Decoders

### Main idea behind using Autoencoders:

Autoencoders are neural networks that aim to copy their inputs to outputs. Suppose data is represented as  $x$ . Decoder: - a function  $g$  that reconstruct the input from the latent space representation.

### Layers of Autoencoders:

A deep autoencoder is composed of two symmetrical deep-belief networks having four to five shallow layers. One of the networks represents the encoding half of the net and the second network makes up the decoding half.

### Autoencoders Components:

An autoencoder consists of 3 components: encoder, code, and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code. The encoder and decoder are completely connected to form a feed forwarding mesh—the code act as a single layer that acts as per its own dimension.

### Autoencoders Types:

- Denoising autoencoder
- Sparse Autoencoder
- Deep Autoencoder
- Contractive Autoencoder
- Undercomplete Autoencoder
- Convolutional Autoencoder
- Variational Autoencoder

## 4.1.4 Description of Pre-Trained Model

In this part we will discuss 2 different models. The first one based on the idea of VGG16 & the second one based on the idea of inception-ResNet-V2.

Before, we discuss the models, we need to know the main layers and their functions:

- Convolution Layers:  
converts all the pixels in its receptive field into a single value. For example, if you would apply a convolution to an image, you will be decreasing the image size as well as bringing all the information in the field together into a single pixel. The final output of the convolutional layer is a vector.
- Pooling Layers:  
is another building block of a CNN. Pooling. Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling.

- **Fully Connected Layers:**  
are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are full connected layers which compile the data extracted by previous layers to form the final output. As The output from the convolutional layers represents high-level features in the data. While that output could be flattened and connected to the output layer, adding a fully connected layer is a (usually) cheap way of learning non-linear combinations of these features.

### First Model:

VGG16 was one of the famous models submitted in ILSVRC in ImageNet 2014 with test accuracy 92.7%.

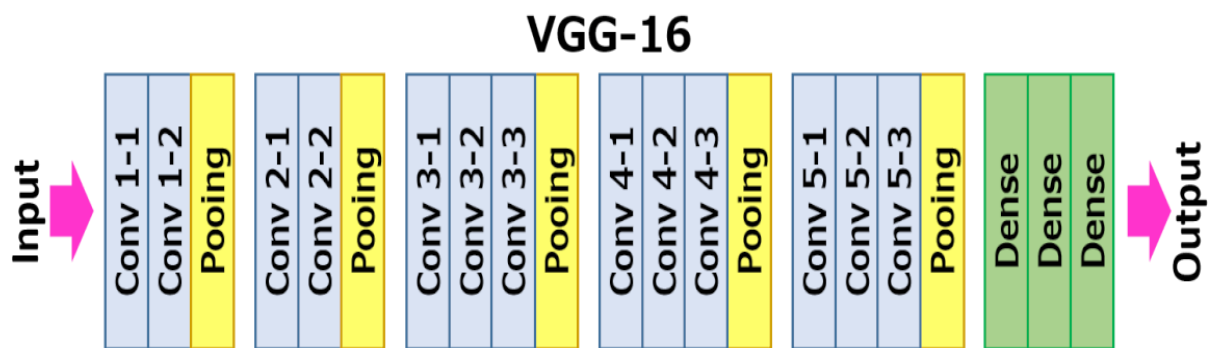


Figure 4.1.4.1 VGG16 1

As we see in Figure 4.1.4 the architecture consists of 16 different layers. The first block of layers uses Conv. Layers for feature extraction. The next block of layers is responsible for classification as it is the main goal for VGG16 it is a classification model. The output block is responsible for mapping the input to 1000 classes.

As illustrated, VGG16 is a classification model. But our goal is to colorize the image, so we need to convert this architecture to satisfy our goal and that is what we are going to discuss.

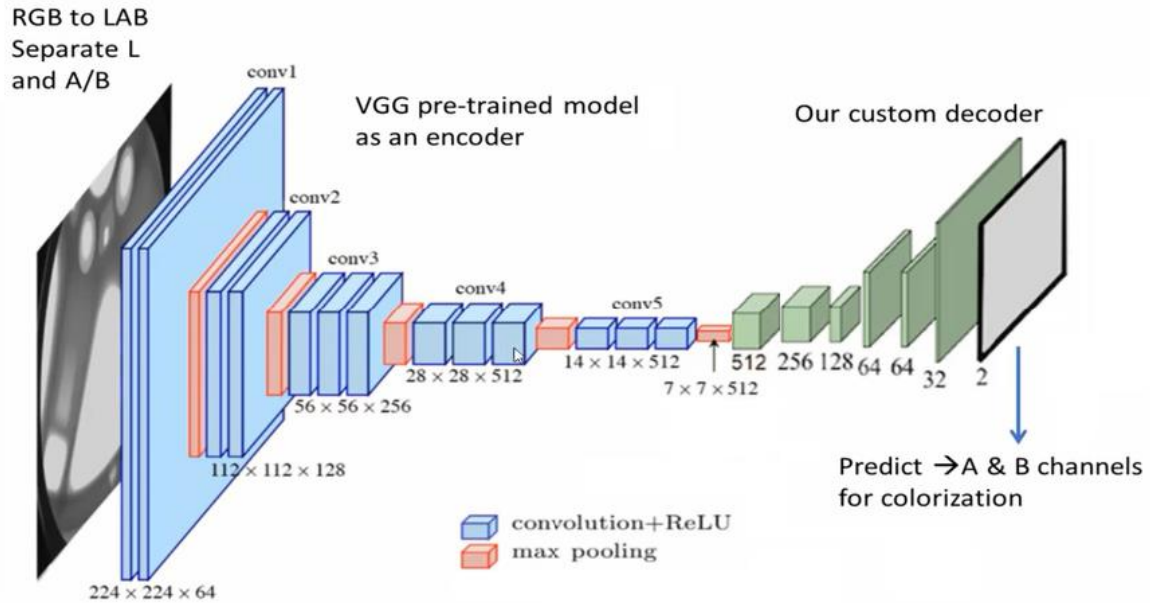


Figure 4.1.4.2 Model 1

In this architecture we tried to make the best use of all layers we mentioned before.

We use 19 layers. The first part act as an encoder and the second one act as a decoder to map the output to the input size again. We convert the image to LAB color space. “L” stands for lightness, “AB” stands for color channels. Then separate the L channel from the input image it will be added to the output image later but after predicting the “AB” channels from the model’s final layer.

### Activation functions:

RELU activation function is used in all hidden layers to provide the model with some non-linear features and to squash the result in a certain range  $[0,1]$  to facilitate the classification process at the output layer later.

$$v = b + \sum_i w_i x_i$$

$$y = \begin{cases} v & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

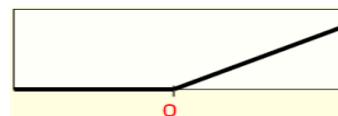


Figure 4.1.4.3 RELU function 1

Tanh activation function is used in the output layer to squash the output values in range of [-1,1].

$$\phi(v) = \tanh(av) = \frac{1 - \exp(-av)}{1 + \exp(-av)}$$

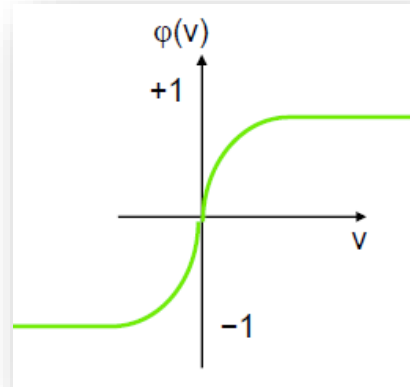


Figure 4.1.4.4 Tanh function 1

### **Loss Function:**

MSE loss function was used to calculate the loss error in the output layer.

$$E(n) = \frac{1}{m} \sum_{p=1}^m \frac{1}{2} (d^p - y^p)^2$$

Figure 4.1.4.5 MSE loss function 1

After building the model's architecture. We decided to start the training phase but through a transfer learning and fine-tuning:

Transfer learning is when a model developed for one task is reused for a model on a second task. Fine tuning is one approach to transfer learning, and it is very popular in computer vision and NLP. The most common example given is when a model is trained on ImageNet is fine-tuned on a second task.

We use the weight of a pre-trained model to initialize our model's weights to grantee a faster convergence.

## Second Model:

In CNNs we need to focus on computational cost & performance, and it is known that inception blocks in CNNs focus on computational cost while, residual blocks in CNNs focus on computational performance that is why we decided to work on a model that merge both blocks together. So, training of residual connections accelerate training of inception network as inception tends to replace the filter concatenation stage of the inception architecture with residual connectors and that is the idea of Inception-ResNet-V2. We grantee that we reap all benefits of the Res approach and retain its computational efficiency. Inception-ResNet-V2 depends on Inception-V3 and ResNet-V2, V5. It provides a high level of feature extraction.

### The Inception Block:

An Inception Module is an image model block that aims to approximate an optimal local sparse structure in a CNN. Put simply, it allows for us to use multiple types of filter size, instead of being restricted to a single filter size, in a single image block, which we then concatenate and pass onto the next layer.

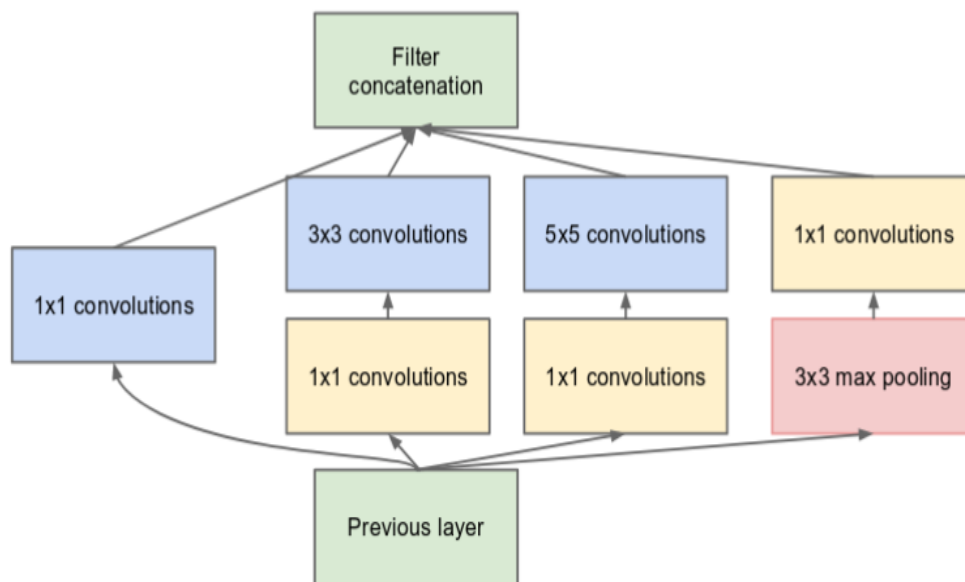


Figure 4.1.4.6 Inception Block 1



## The Residual Block:

They are basically a special case of highway networks without any gates in their skip connections. Essentially, residual blocks allow the flow of memory (or information) from initial layers to last layers.

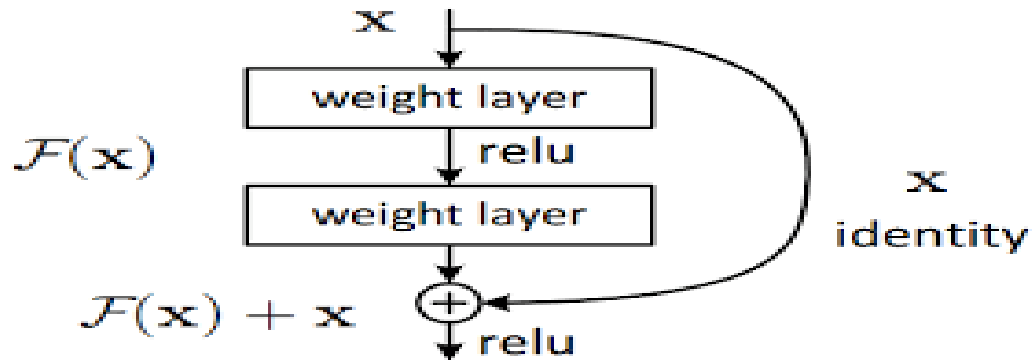


Figure 4.1.4.7 Residual Block 1

## The architecture of Inception-ResNet-V2:

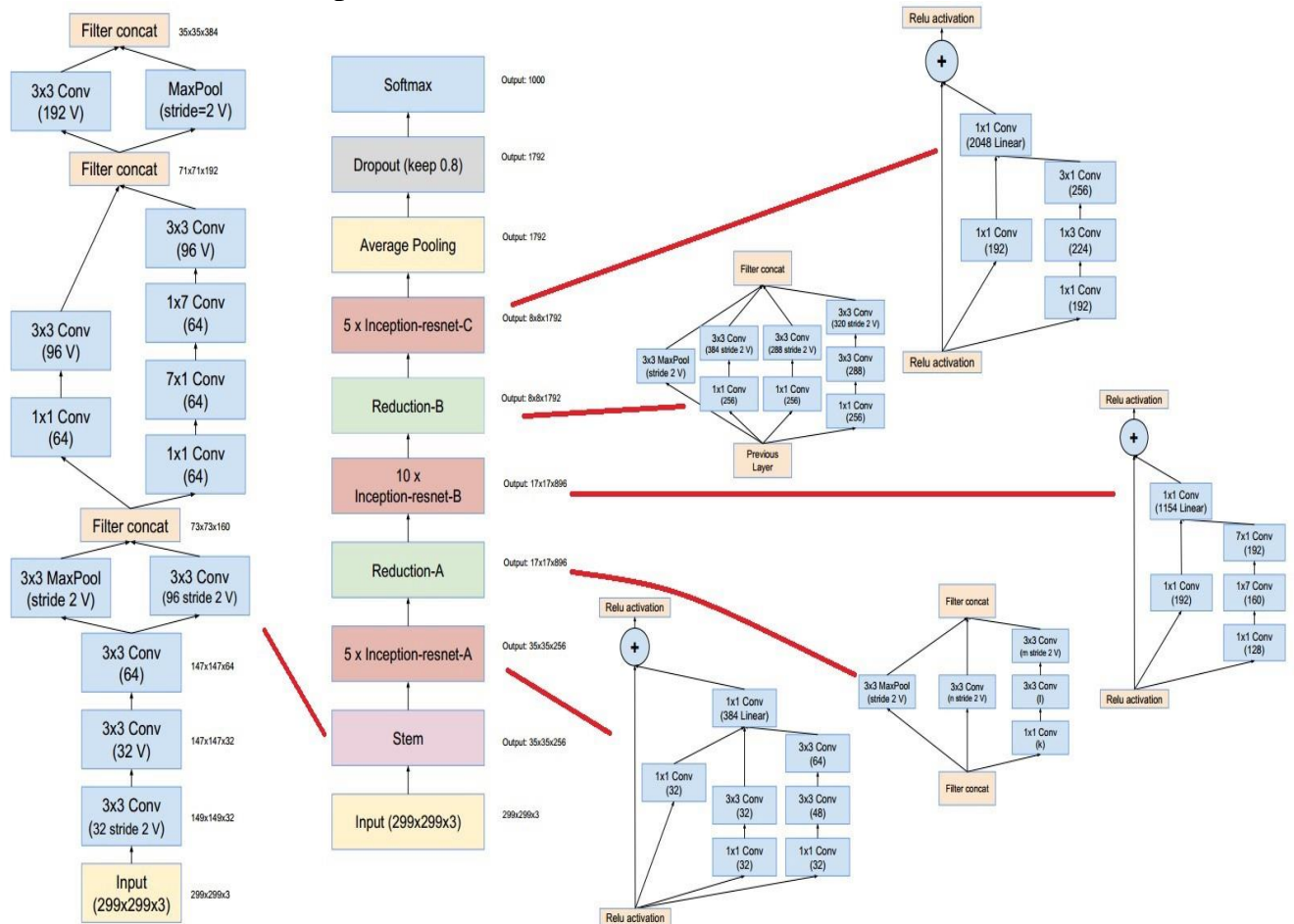


Figure 4.1.4.8 InceptionResNetV2 1

In the previous figure 4.1.4 InceptionResNetV2 it consists of 5 main blocks:

- Reduction A: reduces the dimensions from  $35*35$  to  $17*17$
- Reduction B: reduces the dimensions from  $17*17$  to  $8*8$
- Inception Res A: outputs dimensions  $35*35$
- Inception Res B: outputs dimensions  $17*17$
- Inception Res C: outputs dimensions  $17*17$

### Second model architecture:

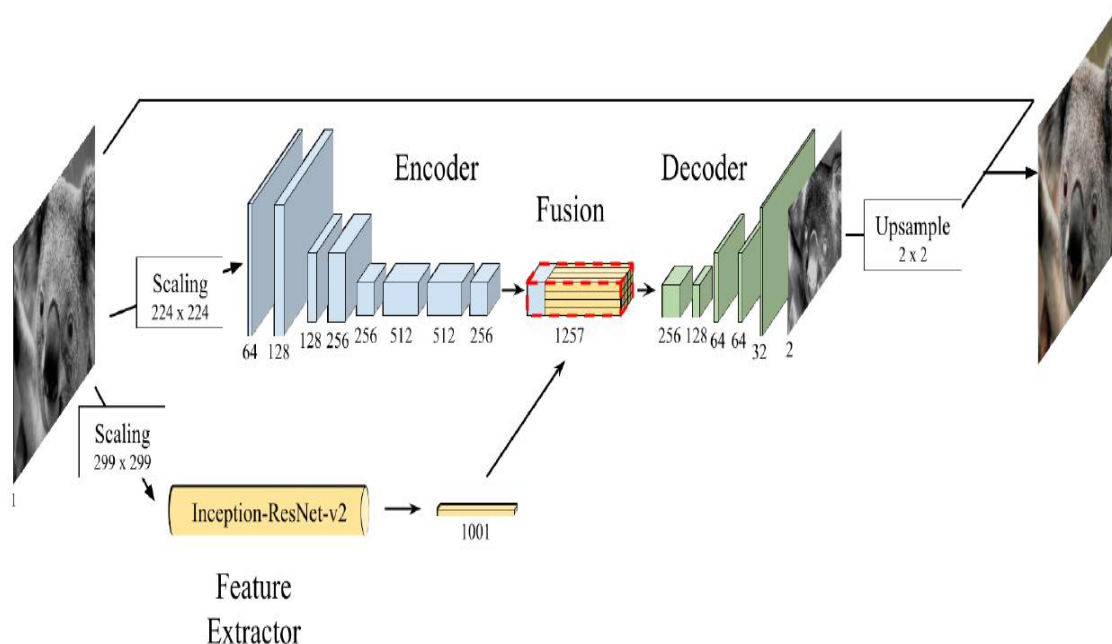


Figure 4.1.4.9 Second Model Architecture 1

Here, the model architecture consists of 3 main layers:

- Encoder: first 8 layers responsible for features extraction.
- Fusion Layer: responsible for merging the encoder output & Inception-ResNet-V2 output.
- Decoder: estimate the output and map its dimension again to match the input dimension.

This model is robust to input image size to add more flexibility. Each inception block is followed by filter expansion layer  $1*1$  Conv. layer without an activation function to scale up the dimension to match the depth of input and needed to compensate dimensionality reduction in inception blocks.

## Part Two: Testing

### 4.2.1 Description of Testing Phase

After applying fine tuning to the model, we wanted to check the generalization error. Actually, the testing phase was divided into separate steps according to the implementation phase:

1. Unit testing:

Is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected.

So, we apply unit testing on the autoencoder firstly to check that the encoder and decoder works correctly, to check the feature map values & to check that the output size matches up the input size.

Then, we apply unit testing to the model using VGG16 weights only to see at which degree the output is close to the ground truth.

2. Integration testing:

Is a type of software testing, which is performed on software to determine the flow between two or more modules by combining them. Integration testing makes sure that the interactions between different components of the software is completed smoothly without any complication.

To apply integration testing we merge different components together to check final results. We tested the output of the model by merging the VGG16 model with different types of encoders were mentioned previously.

3. System testing:

is the testing of a complete and fully integrated software product. It is the final test to verify that the product to be delivered meets the specifications mentioned in the requirement document. It should investigate both functional and non-functional requirements.




Here, in this phase we merge the model with the GUI to test the user input is passing to the model correctly and to check the output displayed to the user. We checked that the system colorization process is done to the most accuracy can be reached on the dataset we work on & authentication works as required in the non-functional requirements.

4. Acceptance testing:

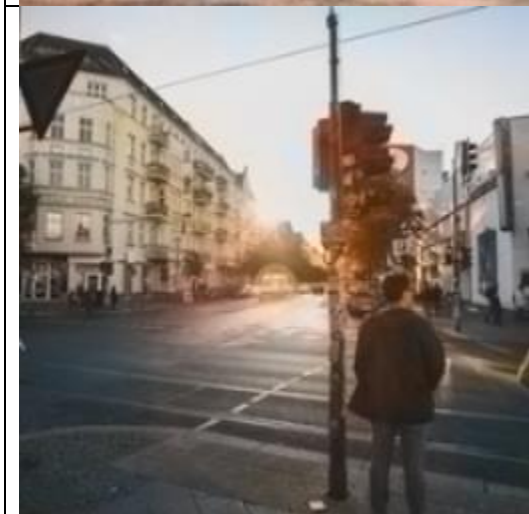
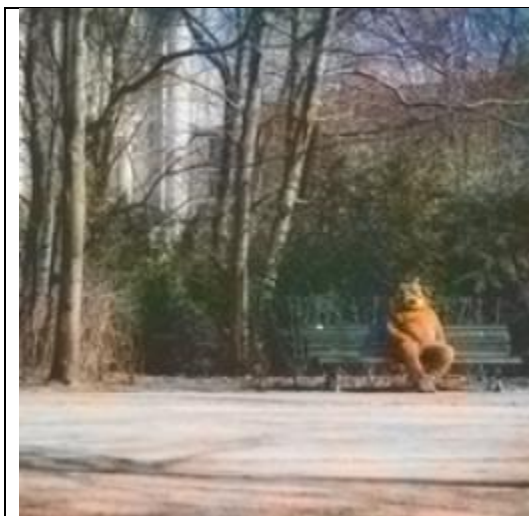
Is a type of testing done by users, customers, or other authorized entities to determine software needs and business processes. It is the most important phase of testing as this decides whether the client approves the software or not.

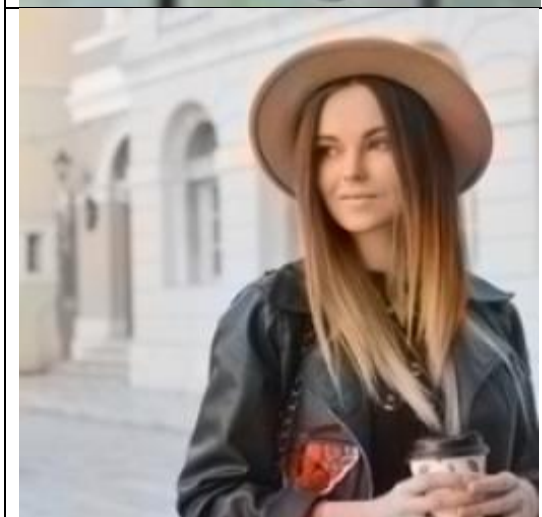
In this phase we tried to deploy the application to different users to check whether they are satisfied by the output or not to get their feedbacks in case of any change or update is required.

**Final results:**

Model's Output	Ground Truth
	
	
	








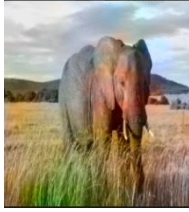






These results were obtained from the first architecture with hyperparameters: -

- Number of epochs:1000
- Batch size: 100
- Optimizer used: Adam
- Accuracy: 82%

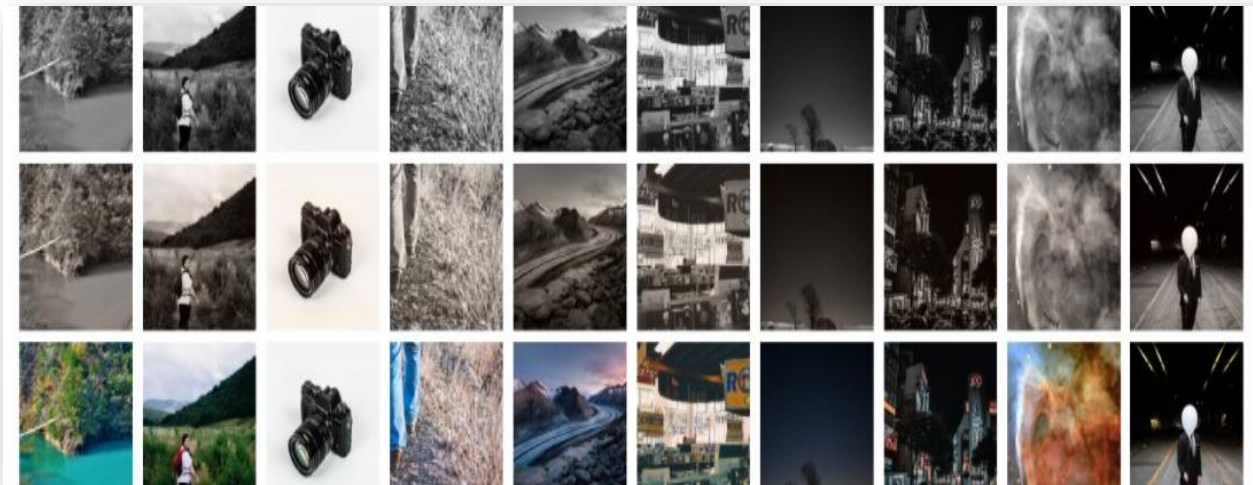
This is the best accuracy achieved by the model and as we have seen it works well with different objects and images contain natural scenes, humans, objects, animals, or even historical images. By improving the dataset, the accuracy will increase, and the colorization process will be closer to the ground truth.

Tested results on animals' images with average accuracy 79.05%:

<b>Model Output</b>				
<b>Ground Truth</b>				



## Results for second architecture:



Due to the high computational resources needed to train this model the size of dataset trained on this model was reduced from 10,000 to 3,000 image. So, the accuracy was not very good as the first model, but it can be improved by increasing the dataset size in case of available resources.





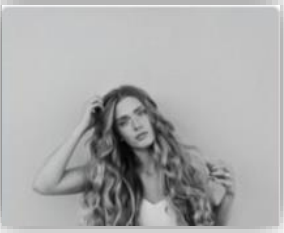











Hyperparameters for this model to obtain the results before were:

- Number of epochs: 100
- Batch size: 100
- Learning rate: 0.001
- Accuracy: 55.7%

The results here were not as expected but this is due to the lack of resources. So, the images were unsaturated.








### 4.2.2 List of Trials to Improve Generalization

First Trial	Second Trial	Third Trial	Ground Truth
			
			
			
			







### First Trial Hyperparameters (Underfitting Case):

- Number of epochs: 100.
- Batch size: 100.
- Optimizer: Adam.
- Accuracy: 54.73%.
- Other Results:

<b>Model Output</b>			
<b>Ground Truth</b>			

### Second Trial Hyperparameters:

- Number of epochs: 400.
- Batch size: 100.
- Optimizer: Adam.
- Accuracy: 69.01%.
- Other Results:

<b>Model Output</b>			
<b>Ground Truth</b>			

### Third Trial Hyperparameters:

- Number of epochs: 1000.
- Batch size: 100.
- Optimizer: Adam.
- Accuracy: 84.73%

<b>Model Output</b>			
<b>Ground Truth</b>			

As we see the more epochs the model is trained on the better accuracy we get, it is granted that the more classes included in the dataset the better colorization is done.

### 4.2.3 UI Design and Wireframes

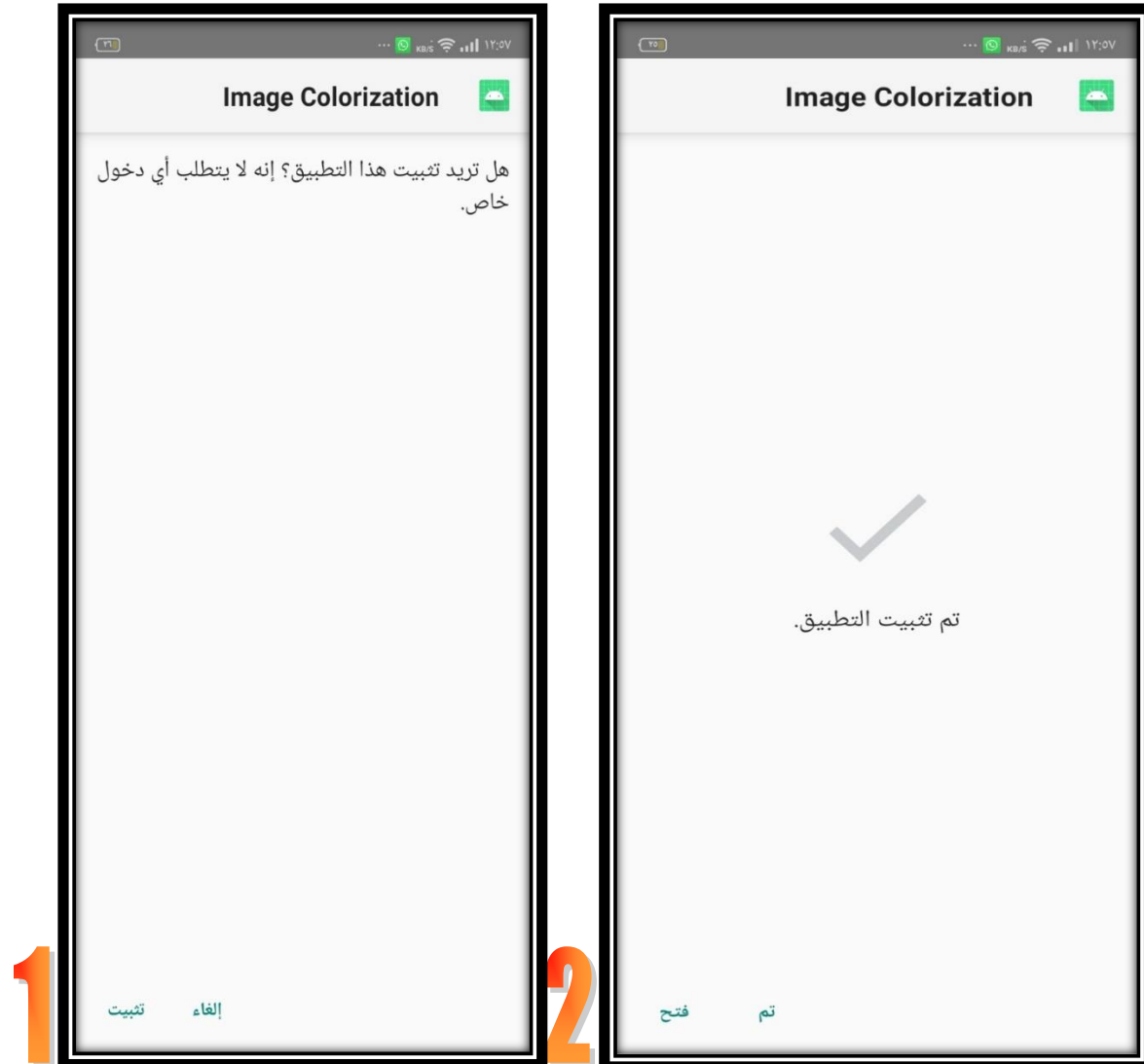
#### GUI for Desktop Application & Mobile (Android) Application:

Both are designed to be user friendly using Tkinter library in desktop application and using Java in mobile application. In both applications the user has to register first to have account in the system. Then, the user is able to browse his storage on his PC or mobile device to select a gray-scaled image to be colorized by the application. After, selecting the image the output will be displayed to the user and saved in his history in case of he wants to navigate in his history at any time.

## 5- User Manual

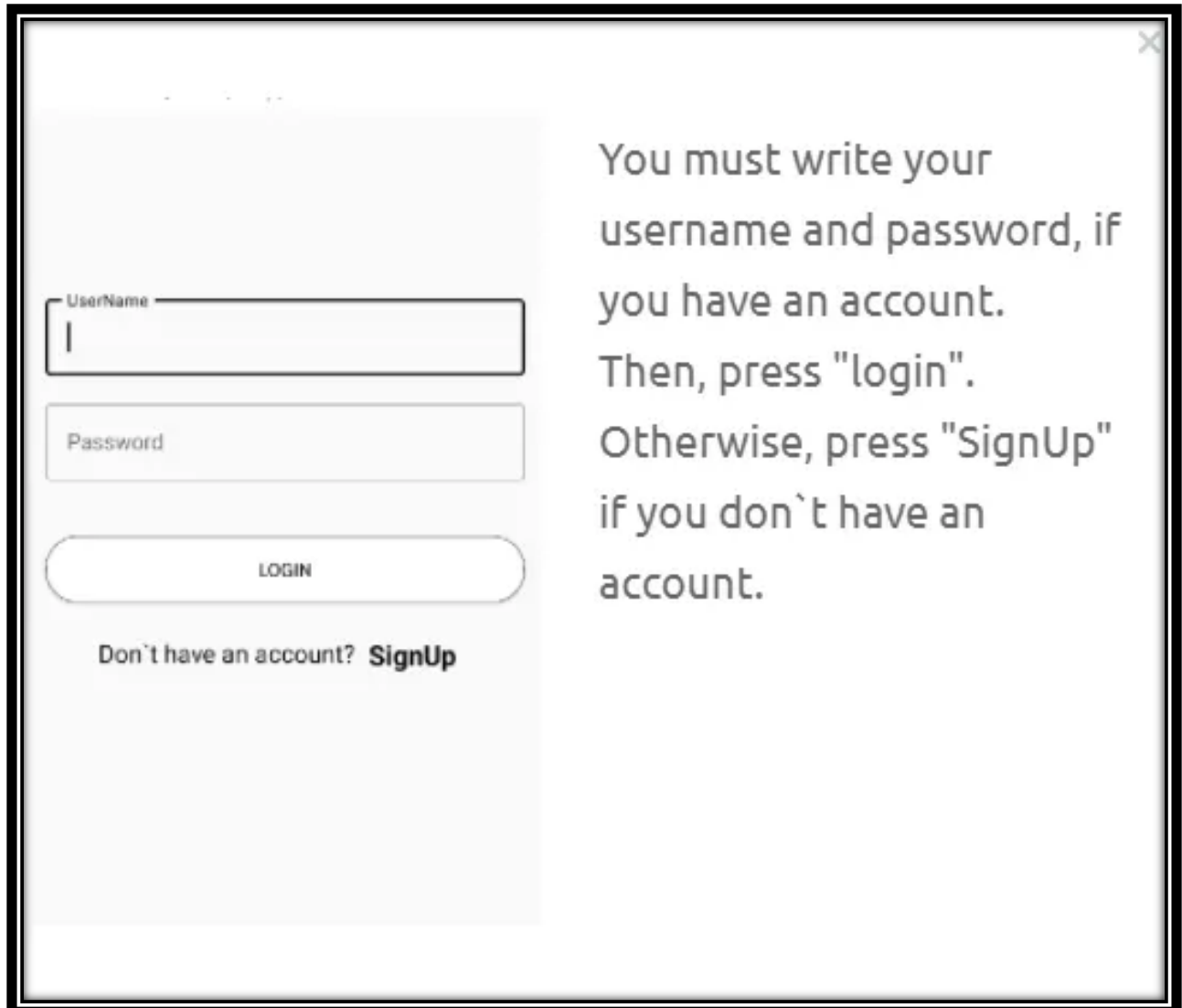
### Installation Guide for The Mobile Application:

User should press next to finish setup after downloading the APK.



## User Guide for The Mobile Application:

### First screen: Login Screen





If this is the first time for the user to register:

### Registration Form

Please, fill the following fields:

- 1- Enter your username.
- 2- Enter your password.
- 3- Confirm your password.

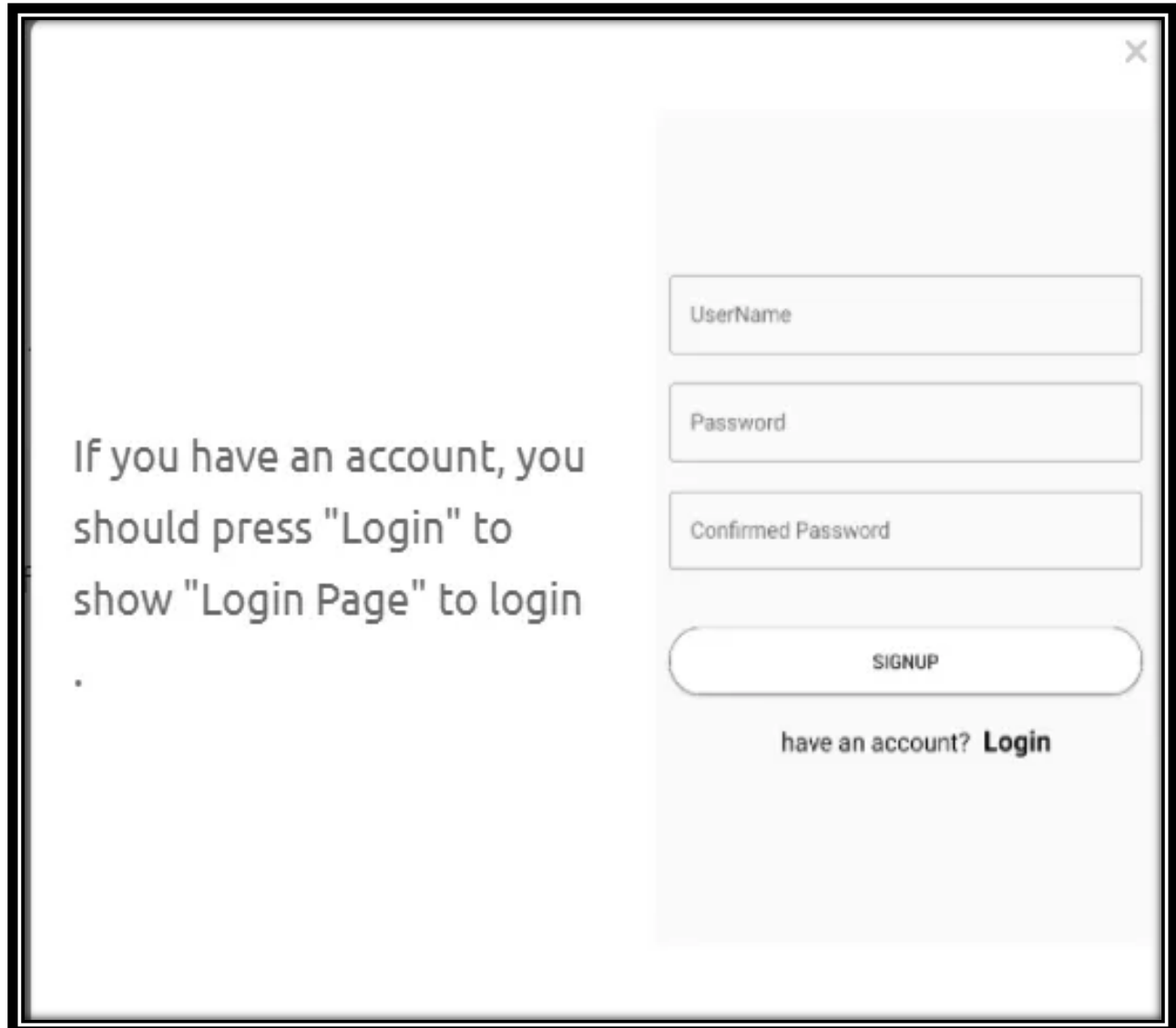
Then, press "SignUP".

**\*Note:** please make sure that your password length is not less than 8 characters and contains at least one special character.

SIGNUP

have an account? **Login**

In the case where the user is on the registration form & wants to login (if the user pressed “Signup” by mistake and wants to return back to the login page), they can just press “Login”.



The image shows a registration form interface. On the left side, there is a message: "If you have an account, you should press 'Login' to show 'Login Page' to login .". On the right side, there is a registration form with the following elements: a "UserName" input field, a "Password" input field, a "Confirmed Password" input field, a "SIGNUP" button, and a link that says "have an account? **Login**". The entire form is enclosed in a light gray box with a close button (X) in the top right corner.

If you have an account, you should press "Login" to show "Login Page" to login .

UserName

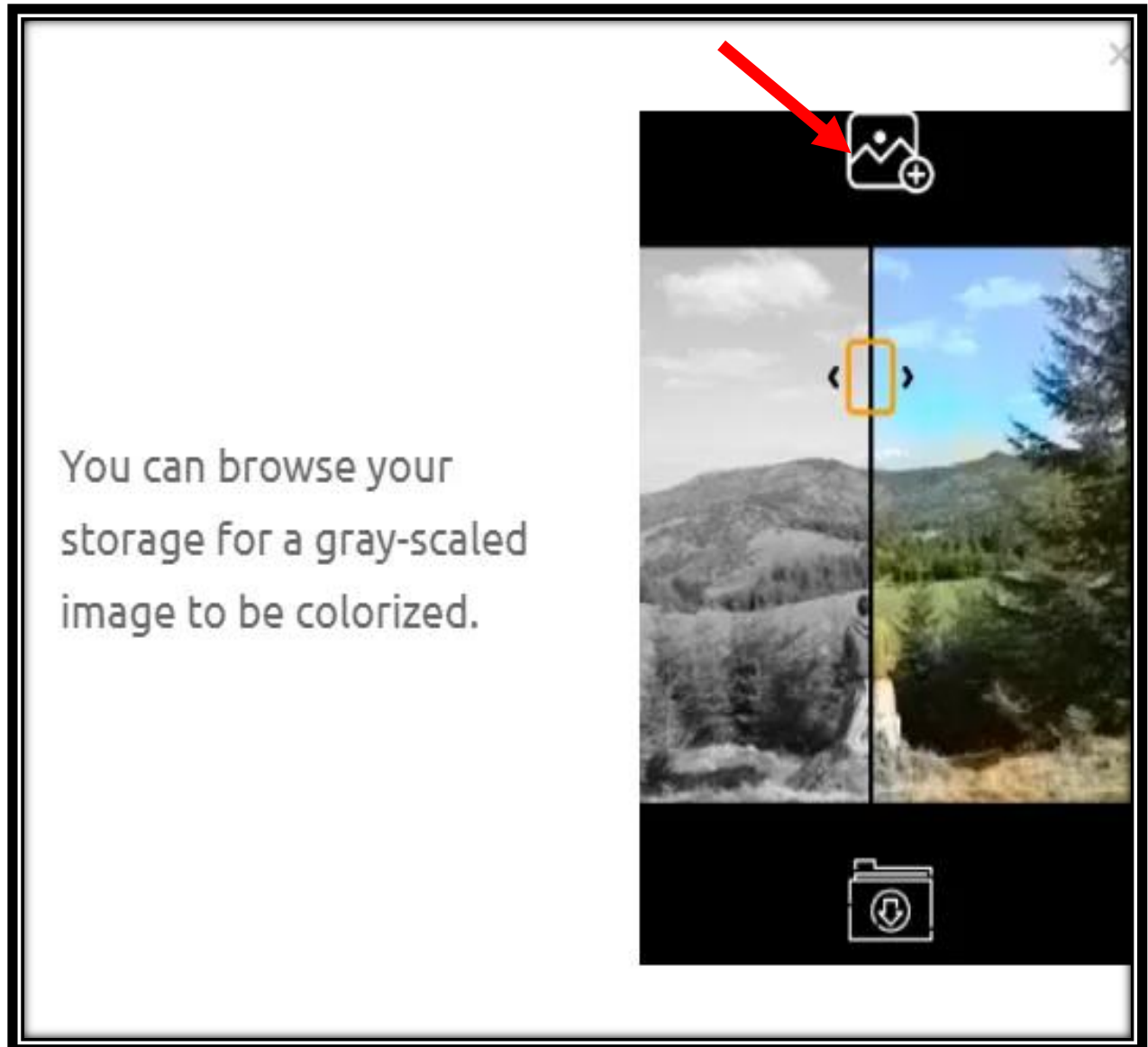
Password

Confirmed Password

SIGNUP

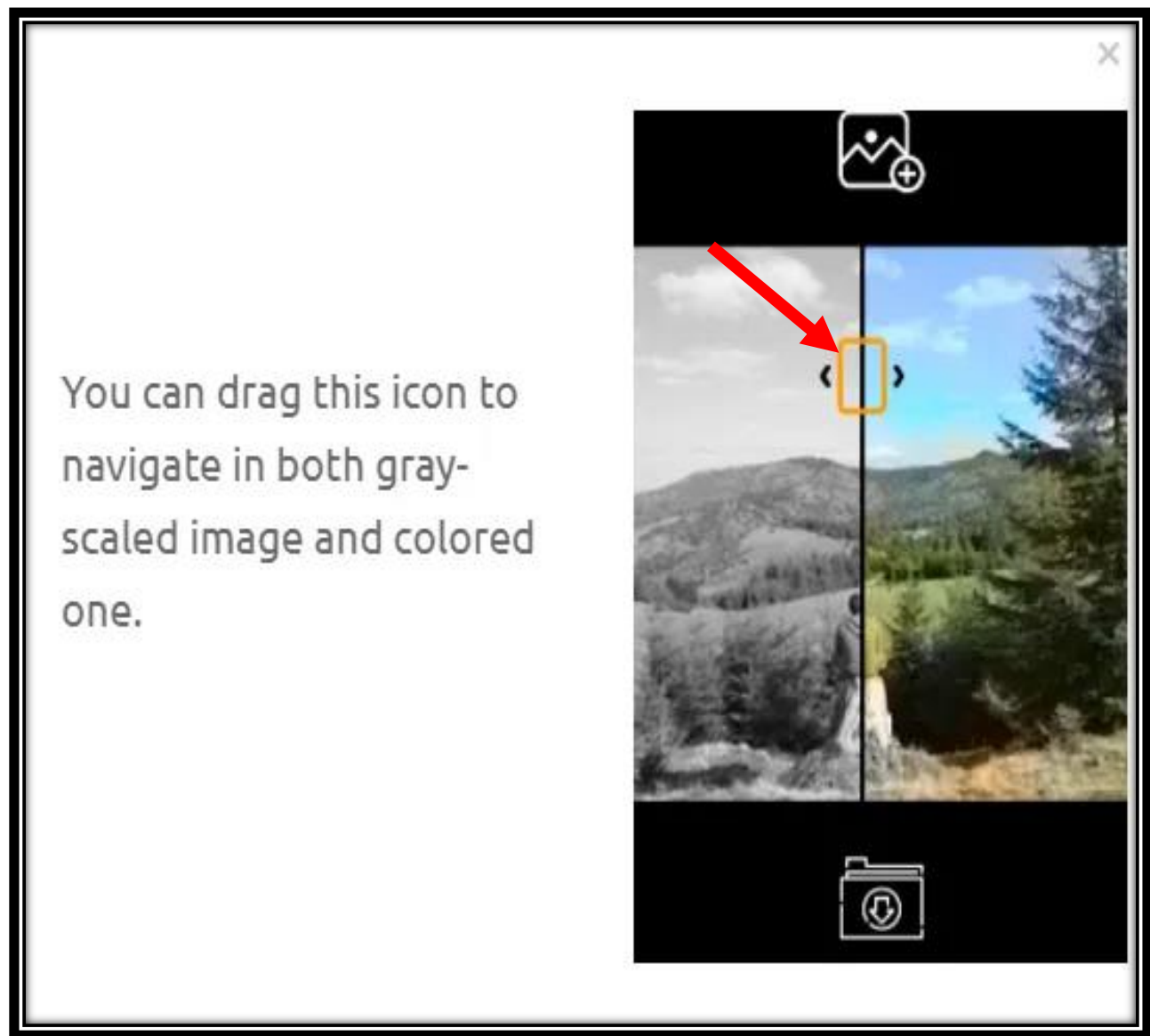
have an account? **Login**

After successful login, the user will see this screen; s/he is asked to browse his/her storage to select a gray-scaled image.

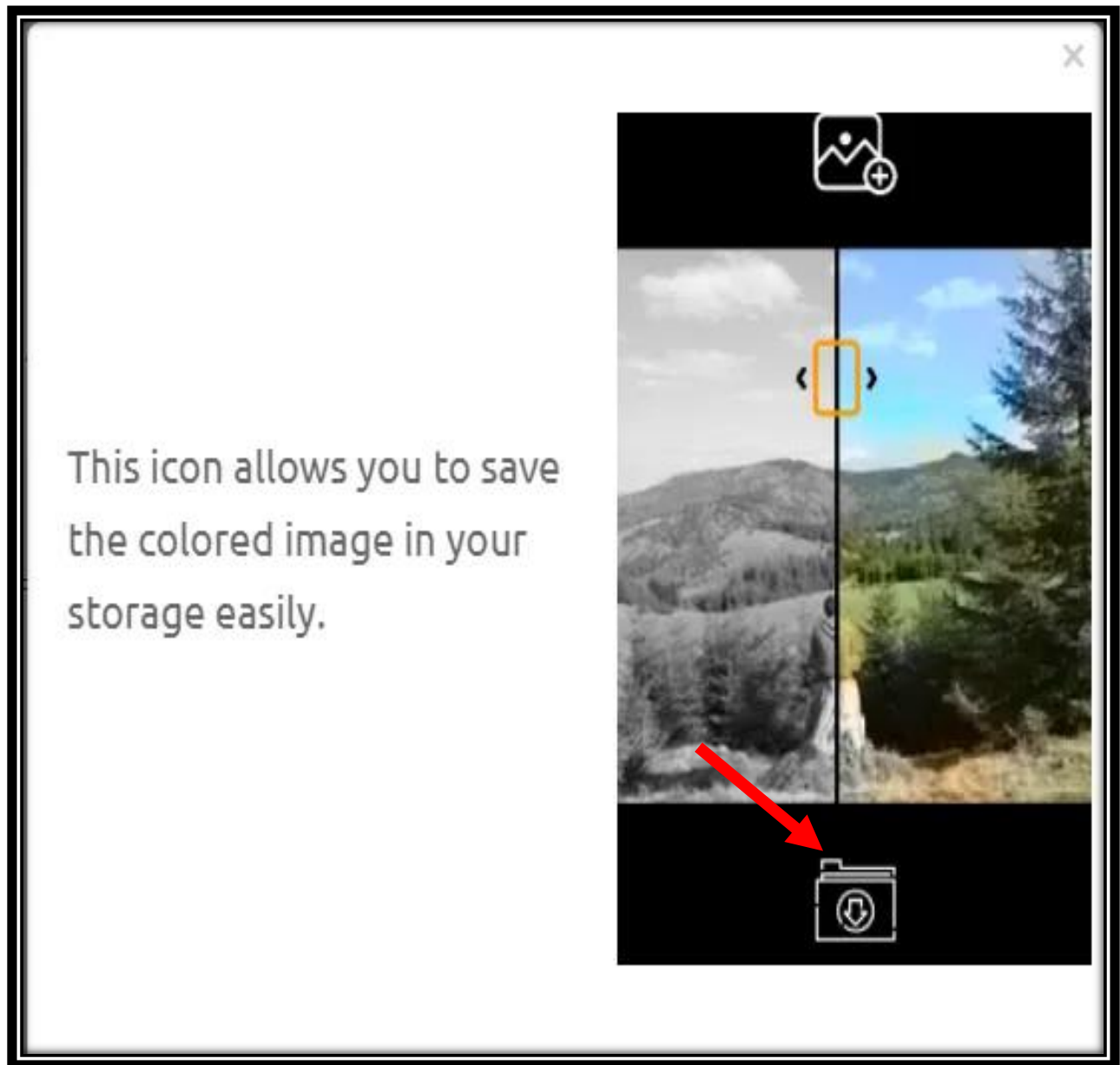




The user is allowed to navigate freely between both images the grayed-scaled image and the colored one using this slider.



The user can save the new colored image easily. If s/he wants to view the images later, s/he will be found in his/her storage.



## User Guide for The Desktop Application:

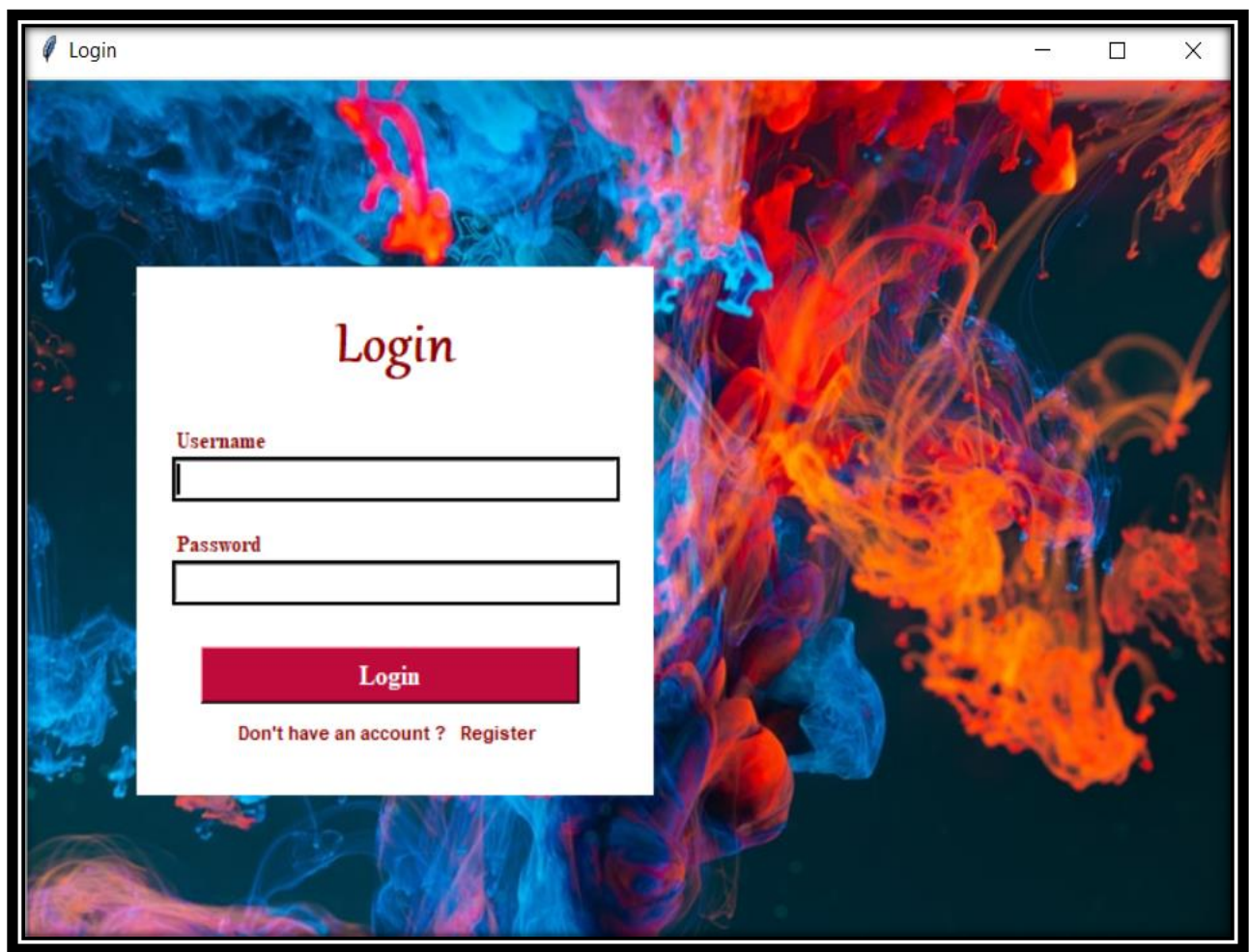
### First screen: Login Screen

Please, Fill the following fields to login successfully if you have an account.

1- Enter your username.

2- Enter your password.

Then, press “Login”.



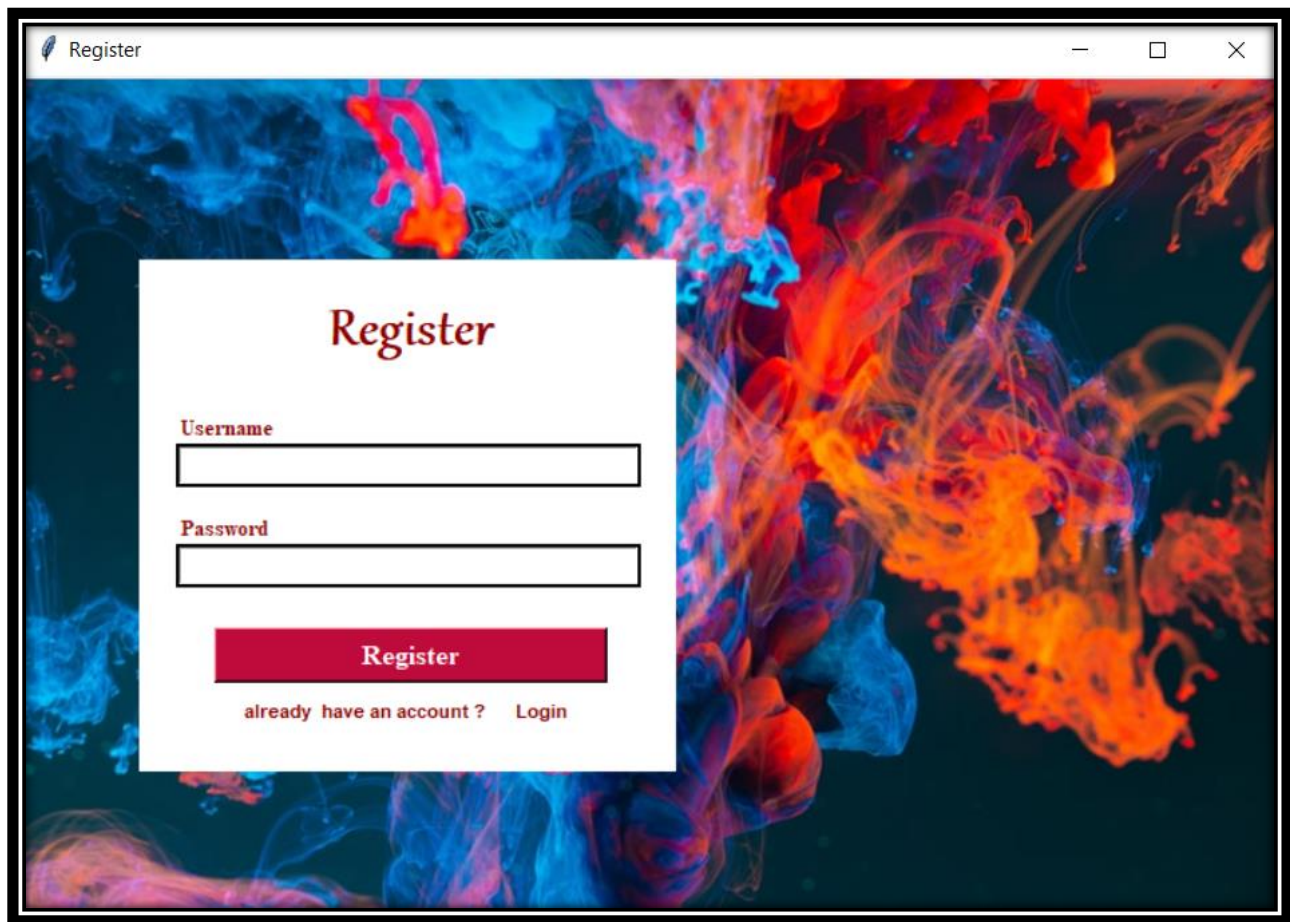
If this is the first time for the user to register:

In order to create an account please, Fill the following fields:

1- Enter a username.

2- Enter a password.

Then, press “Register”.



Register

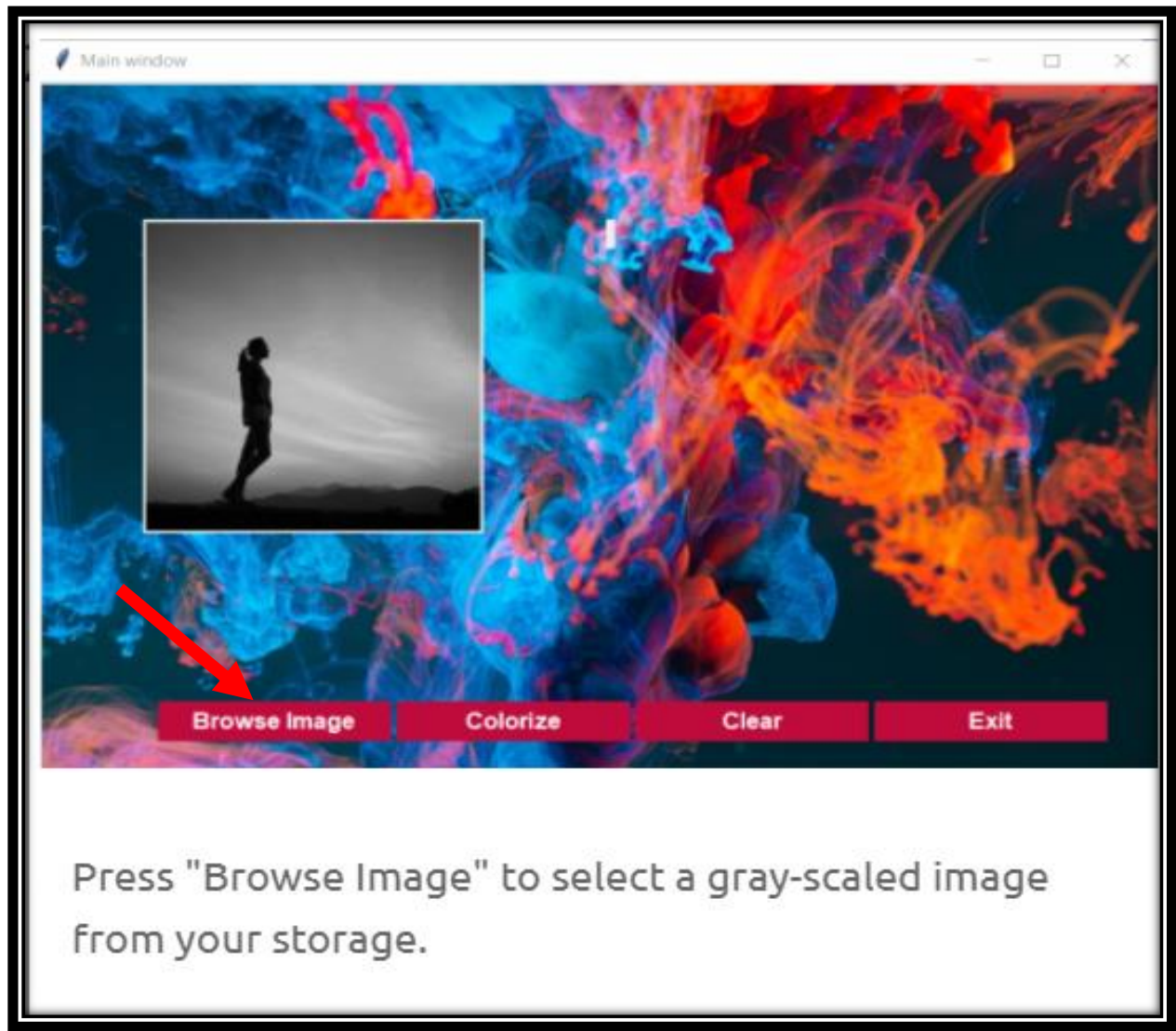
Username

Password

Register

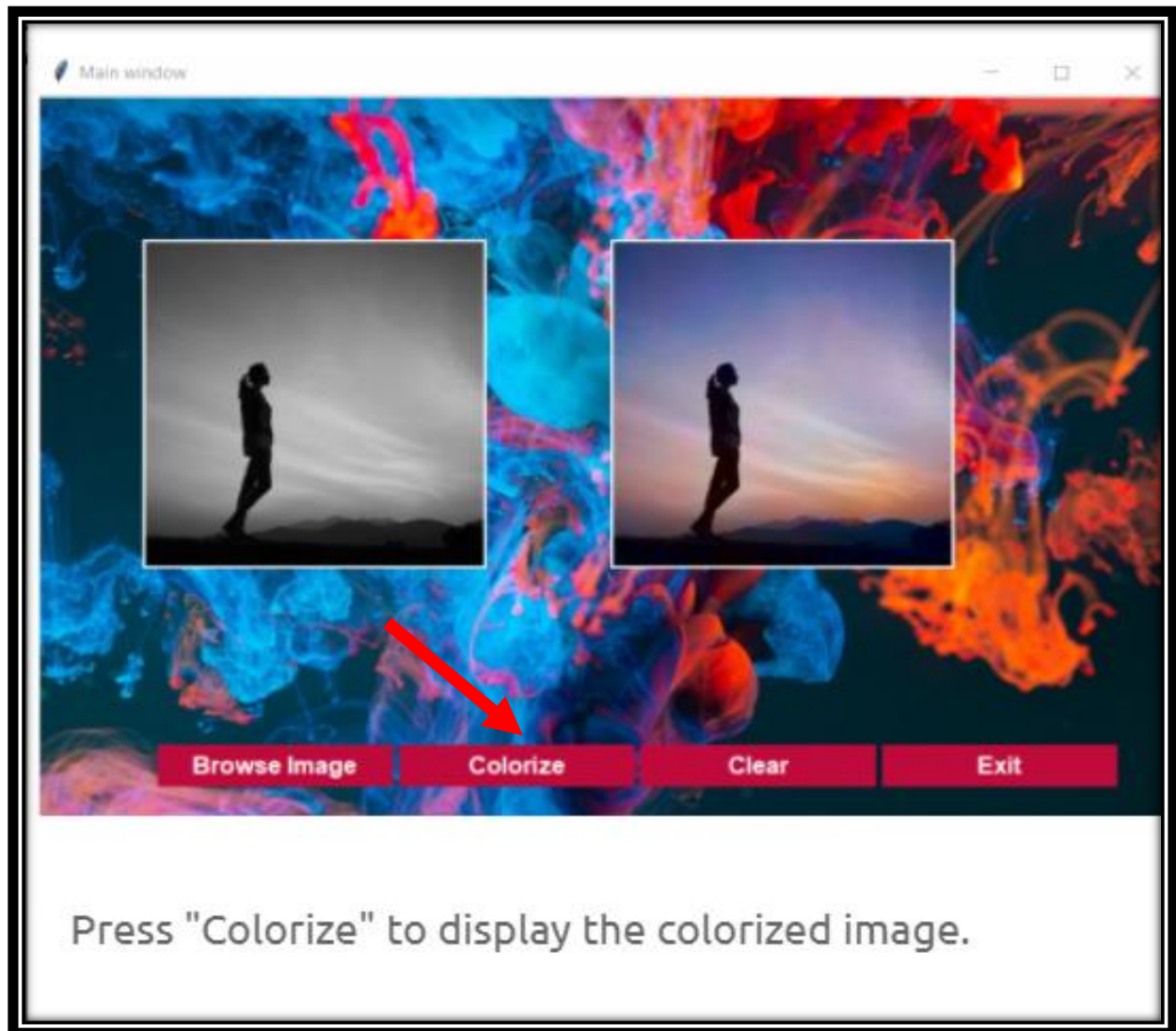
already have an account ? Login

After successful login, the user will see this screen; s/he is asked to browse his/her storage to select a gray-scaled image.

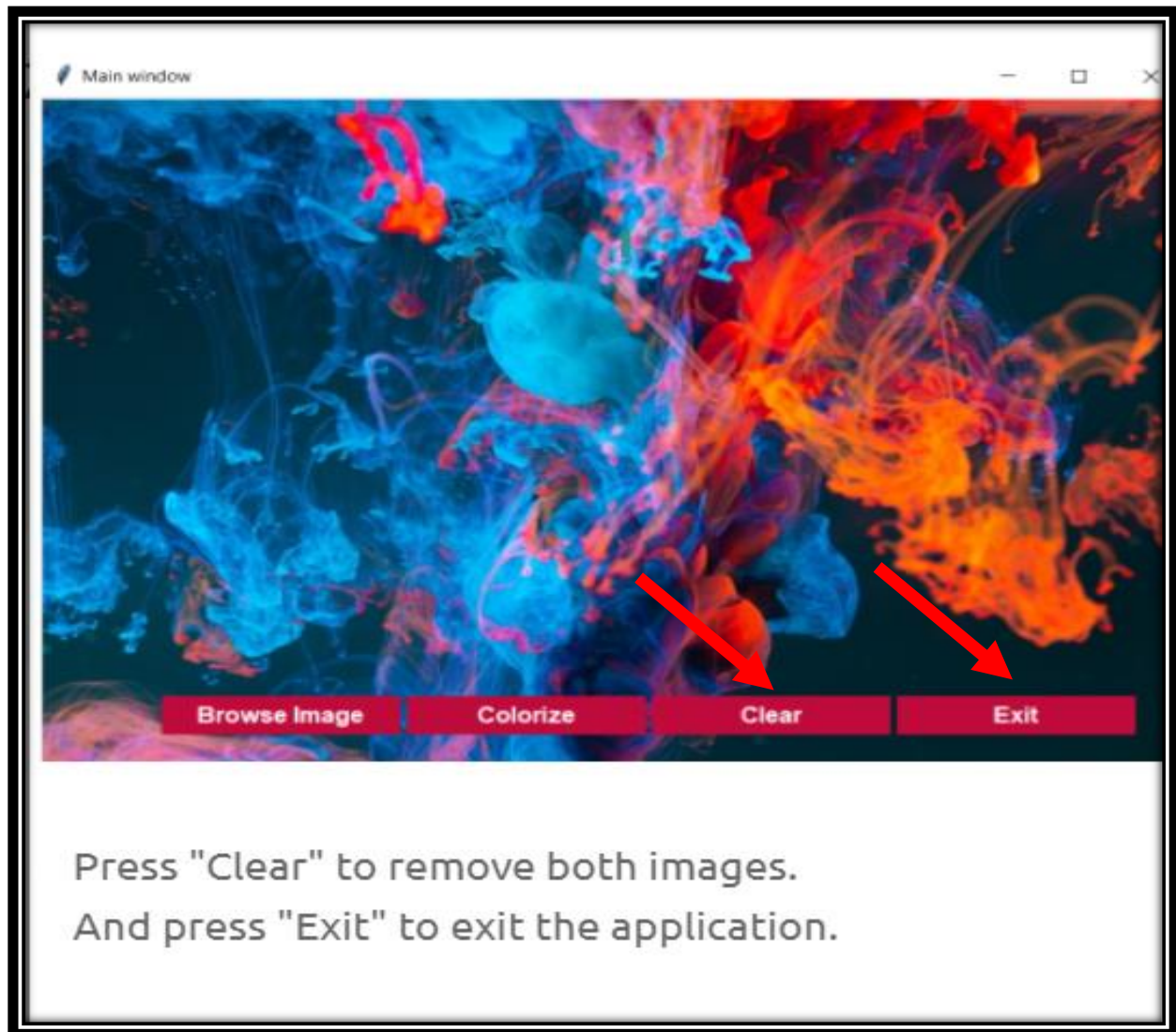




After, browsing an image please, press “Colorize” to display the colored image.



For removing both images and clear your canvas area. Please, press on “Clear”. If you want to exit the application press “Exit”.



In case of wrong user input. If s/he pressed on “Register” in login form and s/he has an account. s/he can navigate through both forms easily and vice versa.

Login	Register
<p>Username</p> <input type="text"/>	<p>Username</p> <input type="text"/>
<p>Password</p> <input type="password"/>	<p>Password</p> <input type="password"/>
<p>Login</p>	<p>Register</p>
<p>Don't have an account ? <a href="#">Register</a></p>	<p>already have an account ? <a href="#">Login</a></p>



## 6- Conclusion and Future Work

### 6.1 Conclusion

This project validates that an end-to-end deep learning architecture could be suitable for some image colorization tasks. We have presented a method of fully automatic colorization of unique grayscale images combining different CNN techniques. Using “LAB” color representation and the cross-entropy loss function. We have represented that method can produce a plausible and vibrant colorization of certain parts of images that has properties which may be applied to video sequences also. Our model does very well with the animals like cats and dogs because the dataset we chose. ImageNet consists large number of pictures of these animals. Even the outdoor scenes turnout very good with our model. The model also captures notion of sunset and paints it orange. The model produces plausible images even with the sketches.

Our approach can successfully color high-level image components such as the sky, the sea, or forests. Nevertheless, the performance in coloring small details is still to be improved. As we only used a reduced subset of ImageNet, only a small portion of the spectrum of possible subjects is represented, therefore, the performance on unseen images highly depends on their specific contents. To overcome this issue, our network should be trained over a larger training dataset. In this regard, a probabilistic approach in the spirit of seems more adequate. We believe that a better mapping between luminance and  $a^*b^*$  components could be achieved by an approach like variational autoencoders, which could also allow for image generation by sampling from a probability distribution.

## 6.2 Future Work

It could be interesting to continue training on the second architecture as it is expected to give better accuracy. With available resources and more classes included in the dataset, the colorization process will be smoother.

It could be interesting also to apply colorization techniques to video sequences, which could potentially re-master old documentaries. This, of course, would require adapting the network architecture to accommodate temporal coherence between subsequent frames. A betterment in the video colorization could still be made with the large datasets that are not available now. The model works very well with the image colorization and video colorization in most of the cases. Even though, the video colorization can still be brushed. We could also apply style transfer for images on a real time data from the user as another way of colorization to allow user to input their preferred style. It can be considered a type of sparse user input as the user is not forced to assign color to each pixel the model will predict the colors of unassigned pixels.

Overall, we believe that while image colorization might require some degree of human intervention it still has a huge potential in the future and could eventually reduce hours of supervised work

## References

1. Szegedy, C., Vanhoucke, V., Io\_e, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR abs/1512.00567 (2015)
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)
3. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)
4. Szegedy, C., Io\_e, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. CoRR abs/1602.07261 (2016)
5. Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification." ACM Transactions on Graphics (TOG) 35.4 (2016):110.
6. Cheng, Zezhou, Qingxiong Yang, and Bin Sheng. "Deep colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.
7. Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning representations for automatic colorization." European Conference on Computer Vision. Springer, Cham, 2016.
8. Li, Bo, et al. "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
9. Zhang, Richard, et al. "Real-time user-guided image colorization with learned deep priors." arXiv preprint arXiv:1705.02999 (2017).
10. Deshpande, Aditya, Jason Rock, and David Forsyth. "Learning large-scale automatic image colorization." Proceedings of the IEEE International Conference on Computer Vision. 2015.

11. Nixon, M.S., Aguado, A.S.: Feature Extraction & Image Processing for Computer Vision. Elsevier Ltd (2002)
12. Hoffman, G.: Cielab colorspace. Technical report, University of Applied Sciences, Emden (Germany), [http://docs-ho\\_mann.de/cielab03022003.pdf](http://docs-ho_mann.de/cielab03022003.pdf) (2003)
13. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.A.: Striving for simplicity: The all convolutional net. CoRR abs/1412.6806 (2014)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRRabs/1412.6980 (2014)
15. Chollet, F., et al.: Keras. <https://github.com/fchollet/keras> (2015)
16. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision, Springer (2014) 818-833