ADVERSARIAL ATTACKS ON IMAGE CAPTCHAS

A Project Report

submitted by

Adhithya S.
Janaki Keerthi
Jayasoorya Jithendra
Jessal V. A.

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM
April 2019

CERTIFICATE

This is to certify that the report entitled **ADVERSARIAL ATTACKS ON IMAGE CAPTCHAS**, submitted by **Adhithya S. (TVE15CS003), Janaki Keerthi (TVE15CS026), Jayasoorya Jithendra (TVE15CS028), Jessal V. A. (TVE15CS029)**, to College of Engineering Trivandrum, for the award of the degree of **Bachelor of Technology** is a bona fide record of the work carried out by them under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Rajasree R.
Project Guide
Assistant Professor
Dept. of Computer Science and Engineering
CET, 695 016

Place: Trivandrum

Date:

ACKNOWLEDGEMENTS

Firstly, we would like to thank the Almighty for giving us the wisdom and grace for presenting our project design.

With a profound sense of gratitude, we would like to express our heartfelt thanks to our guide **Prof. Rajasree R.**, Assistant Professor, Department of Computer Science and Engineering for her expert guidance, co-operation and immense encouragement in pursuing this project presentation.

We are very much thankful to **Dr. Salim A.**, Head of Department and **Prof. Vipin Vasu A. V.**, Assistant Professor of Department of Computer Science and Engineering, our guide for providing necessary facilities and his sincere co-operation.

Our sincere thanks is extended to all the teachers of the Department of CSE and to all our friends for their help and support.

ABSTRACT

KEYWORDS: Adversarial Attack, CAPTCHA, Deep Learning, Convolutional Neural Network, Fast Gradient Sign Method, Transfer Learning, MobilenNet.

An adversarial attack is a way of modifying an image slightly such that the changes are almost imperceptible to the human eye. The modified image thus obtained is called an adversarial image. When it is submitted to a classifier it is misclassified, while the original one is rightly classified. The real-life applications of such attacks can be very serious – for example, a traffic sign can be modified so as to be misinterpreted by an autonomous vehicle, and cause an accident.

Earlier, adversarial attacks were designed to degrade the performance of models or to cause machine learning models to produce particular outputs that are chosen ahead of time by the attacker. This project aims to introduce an adversarial attack that instead reprogram the target model to perform a task chosen by the attacker. Most of the successful attacks are gradient-based methods where the image is subjected to modification in the direction of the gradient of the loss function with respect to the input image. This vulnerability of DNNs to adversarial noise is exploited here constructively inorder to prevent bots from automating captcha tests. The **Fast Gradient Sign Method (FGSM)** has been adopted, which computes an adversarial image (the captcha) by adding a pixel-wide perturbation of magnitude in the direction of the gradient.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AI	Artificial Intelligence
FGSM	Fast Gradient Sign Method
DNNs	Deep Neural Networks
CNN	Convolutional Neural Networks
DDoS	Distributed Denial of Service
CDN	Content Delivery Networks

CHAPTER 1

INTRODUCTION

In the recent times, excellent results have been achieved in different real-world applications like autonomous cars, face recognition and medical image analysis, to name a few. These breakthroughs are not only due to the advances in Deep Neural Networks (DNNs), but also the availability of huge amount of data and computational power. Autonomous vehicles have become so reliable that they no longer need human drivers inside as a backup. Medical systems are now better than human experts in detecting cancer metastases. Even facial recognition software is able to surpass human capabilities. In spite of all these impressive advancements, the research community has recently found that Deep Neural Nets are susceptible to adversarial attacks.

1.1 Problem Description

The three main requirements of the Internet are privacy, security and equality. Nowadays as the artificial intelligence bots are becoming more and more intelligent, they are able to automate tests that help to differentiate a human from a bot. This will lead to the bots getting faster request reply from the web server, thereby giving some users advantage over the others. Bots will start performing DDoS (Distributed Denial of Service) attacks easily on Web Servers as CDN (Content Delivery Networks) use CAPTCHA to differentiate humans from bots. Since the AI bots are showing exponential improvement in CAPTCHA solving and facial recognition, it is high time that a solution be devised and implemented to deal with the issue.

1.2 Motivation

Deep Neural Nets are susceptible to adversarial noise. The aim of the project is to make use of this vulnerability in a constructive manner in order to improve computer security. Adversarial examples for CAPTCHA applications are very difficult for Deep Learning algorithms while easy for humans (as the adversarial noise tends to be small and does not affect human perception of image content). Machine learning models misclassify examples that are only slightly different from correctly classified examples drawn from the data distribution. As a matter of fact, it is found that a variety of models with different architectures and trained on different subsets of the training data misclassify the same adversarial example.

1.3 Overview

In the second chapter, the cause of adversarial attacks and how the Deep Neural Networks are vulnerable to them have been explained in detail. Also the FGSM algorithm and its types are discussed. The third chapter contains a detailed account of the architecture of the model used for the project. The main focus is on transfer learning and the MobileNet V2 architecture. The fourth chapter discusses the requirements and constraints - the environment, the interface, the users etc. Git log has been included in the fifth chapter. Accuracy status report as to the testing and training processes during the several phases has been included in the sixth chapter.

CHAPTER 2

RELATED WORKS

Basically, CAPTCHAS are automatically constructed problems which are quite difficult to be solved by AI algorithms, but easy for humans. However, large number of CAPTCHA designs have become ineffective because of the fast growth in AI. Particularly, recent advancements in the field of deep learning reduced the gap between machine and human ability to solve problems that were usually used in CAPTCHAS in the past. As a matter of fact, several breakthroughs in AI led some researchers to claim that deep learning would gradually lead to the "end" of CAPTCHAS [1].

Despite all this, DNNs still have some shortcomings with regard to human capability [2]. To be specific, they are susceptible to small perturbations in the input, that cannot be perceived by humans but still cause misclassification. Such perturbations, can be specifically crafted for a particular input that forces misclassification by the machine learning model. Though initially this phenomenon was discovered in the context of DNNs, it was observed in other classifiers also, such as decision trees, KNN etc. Moreover, it was proved that adversarial examples designed to be misclassified by one model are often also misclassified by different other machine learning models [3]. This transferability allows adversarial examples to be used in misclassification attacks on machine learning systems, even without having access to the underlying model [4]. Consequently, adversarial attacks pose serious security threats for a number of machine learning based solutions like facial recognition, biometric authentication, voice commands and spam filters. However, this project aims to make use of adversarial examples in a constructive way so as to improve computer security.

CHAPTER 3

METHODOLOGY

3.1 Cause of Adversarial Attacks

Modern neural networks manipulate data in the form of 32-bit floating point numbers. Whereas modern hardware manipulate images as 8 bit values. Thus adversarial attacks manipulate the data such that the main 8 bits are unchanged, by modifying the remaining 24 bits. Neural Networks perform linear transformation to the matrices. Additions of dropout, pre-training, and model averaging do not improve the vulnerability of model to adversarial examples.

Convolutional neural networks approximate perceptual distance as euclidean distance. This resemblance is clearly flawed if images that have an immeasurably small perceptual distance correspond to completely different classes in the network's representation.

3.2 The Linear Explanation of Adversarial Examples

The precision of the features is limited. It is not rational for the classifier to respond differently to an input x than to an adversarial input $x \; = x + \eta$ if every element of the perturbation η is smaller than the precision of the features. Precision of the features is about (1/256) ≈ (0.03). Thus for $\eta \leq (0.03)$ the adversarial input should act similiar to the original input. Thus effectively the transformation becomes:

$$W^T * \tilde{x} = W^T * x + W^T * \eta$$

3.3 Fast Gradient Sign Method

FGSM computes an adversarial image by adding a pixel-wide perturbation of magnitude in the direction of the gradient. This perturbation is computed with a single step, thus is very efficient in terms of computation time.

$$X^{adv} = x + \varepsilon.sign(\nabla_x J(x, y_{true}))$$

where,

X : is the clean input

X^{adv} : is the perturbed adversarial example.

J : is the classifier's loss function.

y_{true} : is the true label for the input x.

3.3.1 Targeted Fast Gradient Sign Method

Similar to the FGSM, in this method a gradient step is computed, but in this case in the direction of the negative gradient with respect to the target class:

$$X^{adv} = x - \varepsilon.sign(\nabla_x J(x, y_{target}))$$

where y_{target} is the target label for the adversarial attack.

3.3.2 Iterative Fast Gradient Sign Method

The iterative methods take T gradient steps of magnitude $\alpha = \varepsilon/T$ instead of a single step t:

$$X_0^{adv} = X$$

$$X_{t+1}^{adv} = X_t^{adv} + \alpha.sign(\nabla_x J(x, y_{true}))$$

3.3.3 Comparison of various Fast Gradient Sign Methods

Both one-shot methods (FGSM and T-FGSM) have lower success rates when compared to the iterative methods (I-FGSM) in white box attacks. When it comes to black box attacks the basic single-shot methods turn out to be more effective. The most likely explanation for this is that the iterative methods tend to overfit to a particular model.

CHAPTER 4

ARCHITECTURE / STRUCTURE OF THE PROJECT

4.1 Phases

The main phases of the project are as follows:

1. **CAPTCHA Generation**
   A CAPTCHA generator is used to give an input CAPTCHA to the system. The CAPTCHA consists of 6 characters in text format.
2. **Image Modification**
   This is where adversarial noise is added to the image using FGSM. The modification will be imperciptible to human eye.
3. **CNN Model prediction**
   Modified image is given to a Convolutional Neural Network model and prediction is made. Predicted value is compared with the original value and performance of system is evaluated.
4. **Output**
   Output is a modified CAPTCHA image, which is indistinguishable to human eye, but wrongly predicted by the CNN model.

A python script was used to generate the CAPTCHAs. The script can be used to generate CAPTCHAs consisting of lowercase alphabets or uppercase alphabets or digits. It also allows us to generate CAPTCHAs of different lengths. For this project, the CAPTCHA length was restricted to 6 lowercase characters. The generated CAPTCHAs were then subjected to modification using the FGSM and given to the CNN model.

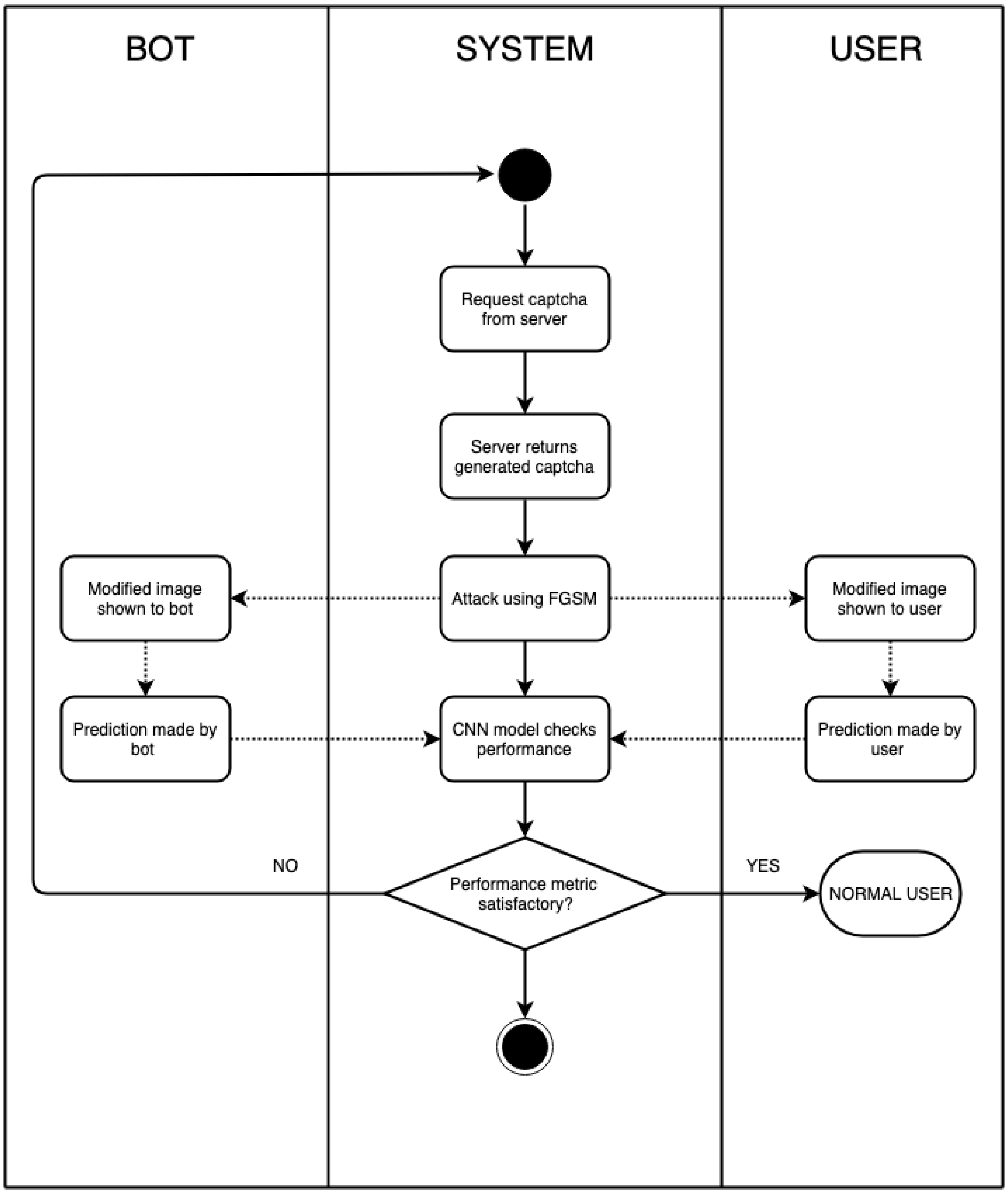

BOT
SYSTEM
USER
Request captcha from server
Server returns generated captcha
Modified image shown to bot
Attack using FGSM
Modified image shown to user
Prediction made by bot
CNN model checks performance
Prediction made by user
NO
YES
Performance metric satisfactory?
NORMAL USER

Figure 4.1: Activity Diagram

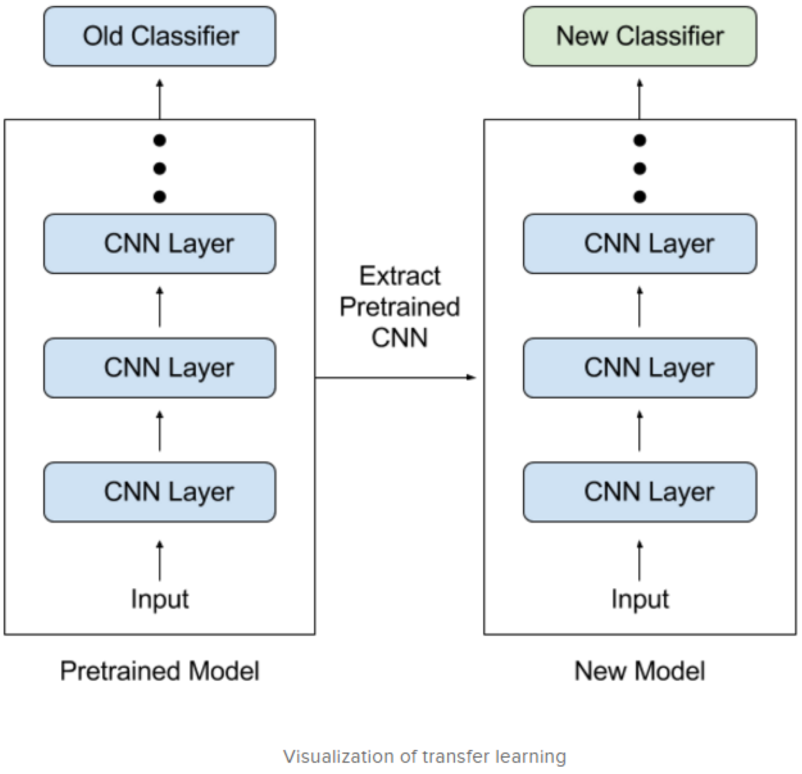


Figure 4.2: Transfer Learning

4.2 Transfer Learning and the MobileNet model

Transfer learning is a popular approach in deep learning where pre-trained models are used as the starting point. Instead of training a deep network from scratch for the task, it allows to take a network trained on a different domain for a different source task and adapt it for the target domain and target task. For this, the MobileNet V-2 architecture was used.

The core layer of MobileNet is depthwise separable filters, named as Depthwise Separable Convolution. The network structure is another factor to boost the performance. Finally, the width and resolution can be tuned to trade off between latency and accuracy.

MobileNetV2 improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. Depthwise separable convolutions which is a form of factorized convolutions which factorize a stan-

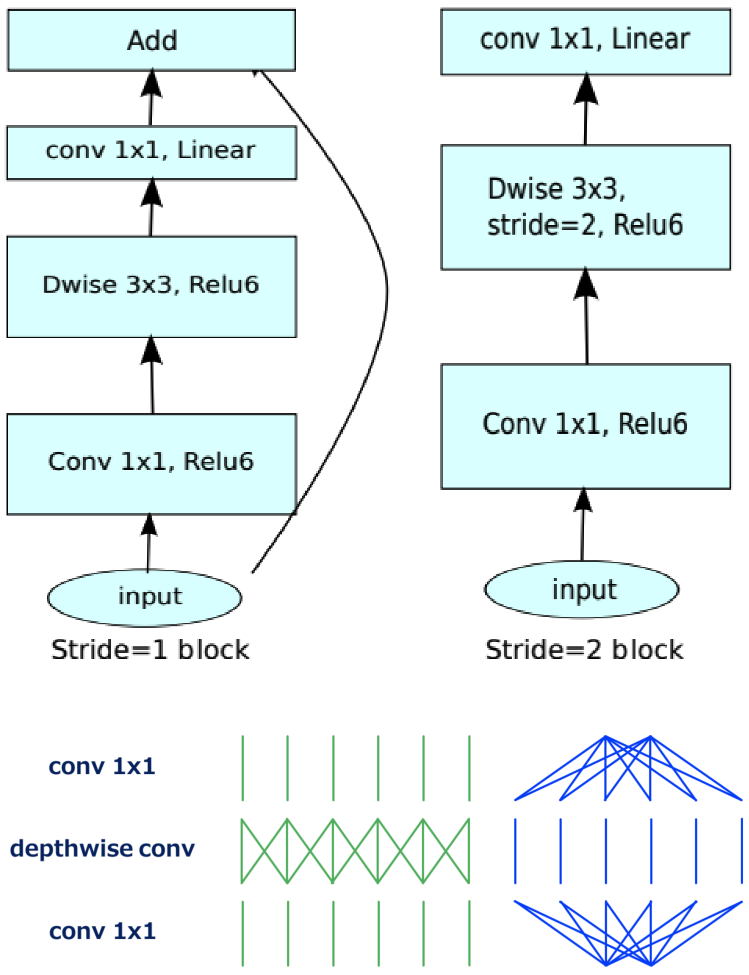


Figure 4.3: MobileNet V2 architecture

dard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. In MobileNet, the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution.

MobileNet-V2 utilizes a module architecture similar to the residual unit with bottleneck architecture of ResNet; the modified version of the residual unit where convolution3×3 is replaced by depthwise convolution. Contrary to the standard bottleneck architecture, the first convolution 1×1 increases the channel dimension, then depthwise convolution is performed, and finally the last convolution 1×1 decreases the channel dimension.

CHAPTER 5

FUNCTIONAL REQUIREMENTS COMPLIANCE REPORT

5.1 Requirements and Constraints

The adversarial CAPTCHA is given to the CNN model and it will predict a corresponding output. Performance assessment can be done by comparing the predicted value and original value of CAPTCHA.

1. **Input**
   CAPTCHA generator is used to give an input CAPTCHA to the system. Each CAPTCHA consists of six characters in lowercase.

2. **Image Modifier**
   Image modifier will add noise to the given CAPTCHA. Noise is generated using FGSM algorithm. The modification must be imperceptible to human eye.

3. **CNN Model**
   Modified image is given to a CNN model and prediction is made. Predicted value is compared with the original value and performance of system is evaluated.

4. **Output**
   Output is a modified CAPTCHA image, which is indistinguishable to human eye, but wrongly predicted by the CNN model.

5.1.1 Design Constraints

1. **Physical Environment**
   The model requires to be trained in the cloud - a programming environment which supports TensorFlow and GPU training. Training was done on Google Colaboratory.

2. **Interface**
   The user will be given a web application to interact with. It will provide the user with a random CAPTCHA generated real time. The user will be given a text box to give the response to the displayed CAPTCHA.

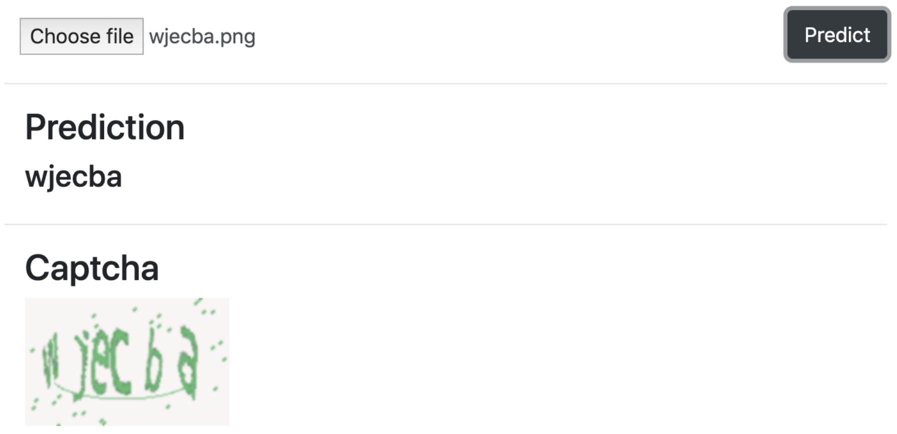
3. **Users**
   The system can be used by an individual or organization who wants to secure their image from image recognition. The user will only have to provide the data or dataset on which attack has to be performed on.

5.1.2 Process Constraints

1. CNN model used for the process has to be trained to recognize the characters from an image CAPTCHA close to optimal bayes error.
2. Training a CNN model is a difficult task as it requires a lot of computational power which can be harnessed with the help of Cloud GPU .

5.1.3 Quality Constraints

1. Training requires high computational power
2. The probability of the randomly guessing each character of the CAPTCHA is $(\frac{1}{26})^6$
3. This comes out to be about 1% probability.
4. The attack will be successful if the model's accuracy can be reduced to atleast 4%.


Choose file wjecba.png

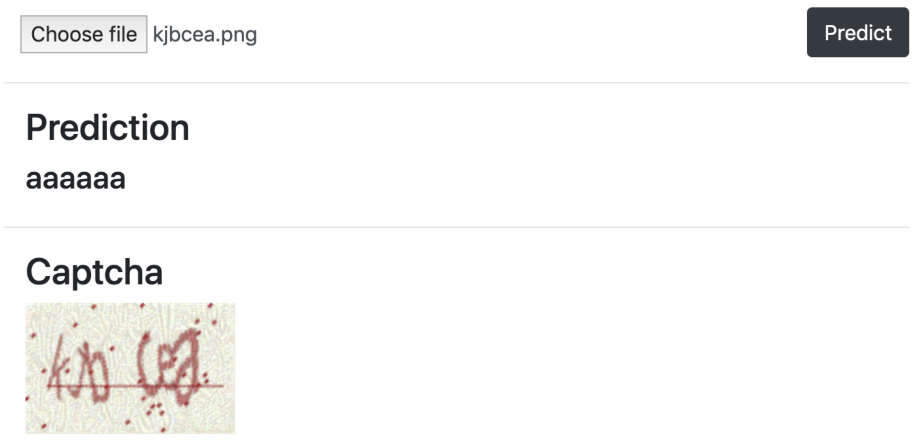
Predict

Prediction

wjecba

Captcha

Figure 5.1: System implemented as a Web Application


Choose file kjbcea.png

Predict

Prediction

aaaaaa

Captcha

Figure 5.2: System implemented as a Web Application

CHAPTER 6

TESTING/TRAINING/DATASET ACCURACY STATUS REPORT

The generated dataset consisted of around 10 million CAPTCHAs. The training and testing status has been discussed below in detail.

6.1 Training done on FloydHub and Google Colab

Initially 10gb dataset was uploaded on FloydHub and training was performed. It was successful but resultant accuracy was only about 25% . Inorder to improve accuracy, we decided to use a pretrained model (Transfer Learning). Training was then performed on Google Colaboratory by making use of the MobileNet V2 model. In Google Colab, the dataset was compressed using tfrecord where 10gb data can be compressed to approximately 1gb training tfrecord. The training performed on Google Colab was successful and reached upto 99% accuracy for each of the 6 characters.

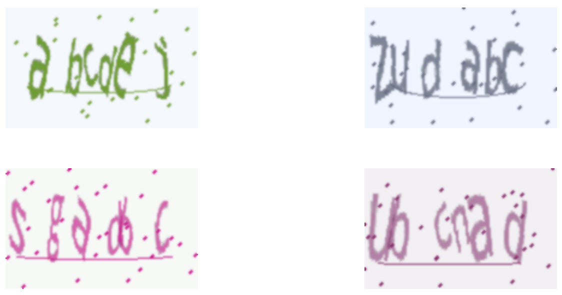


Figure 6.1: Generated CAPTCHAs before the attack

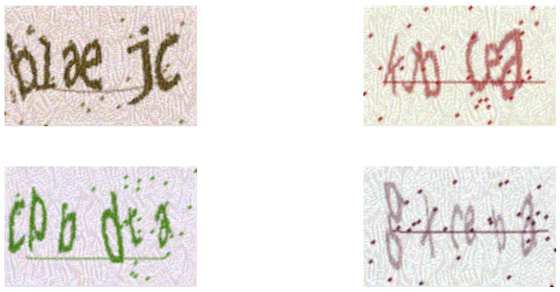
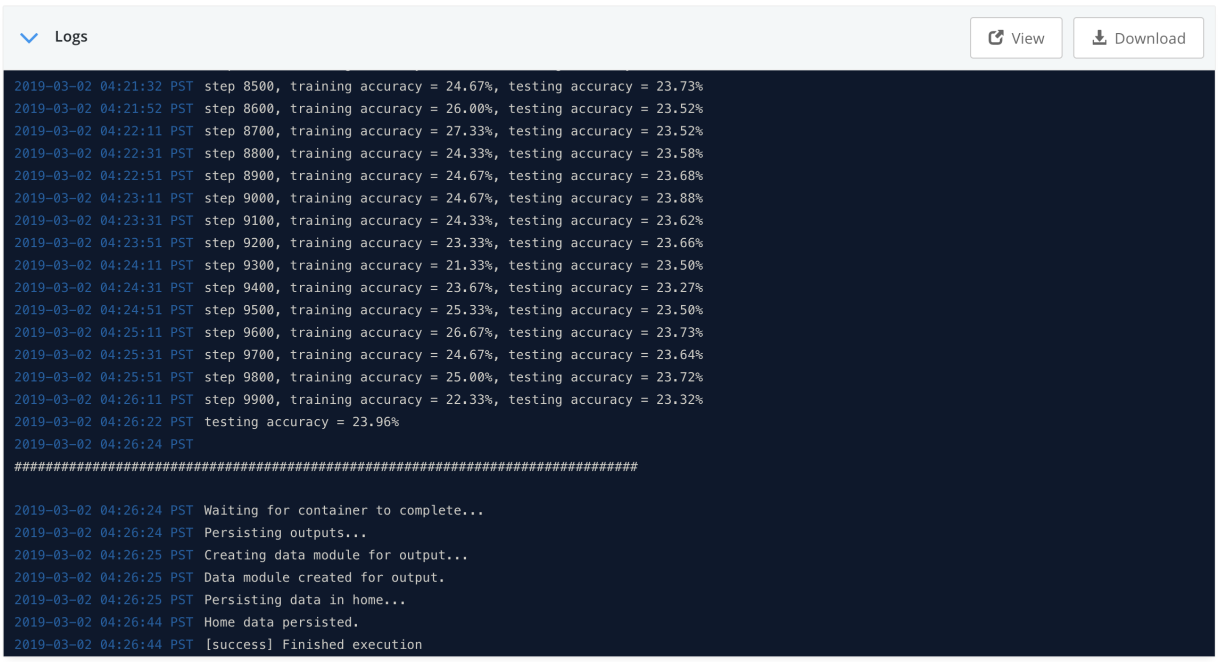


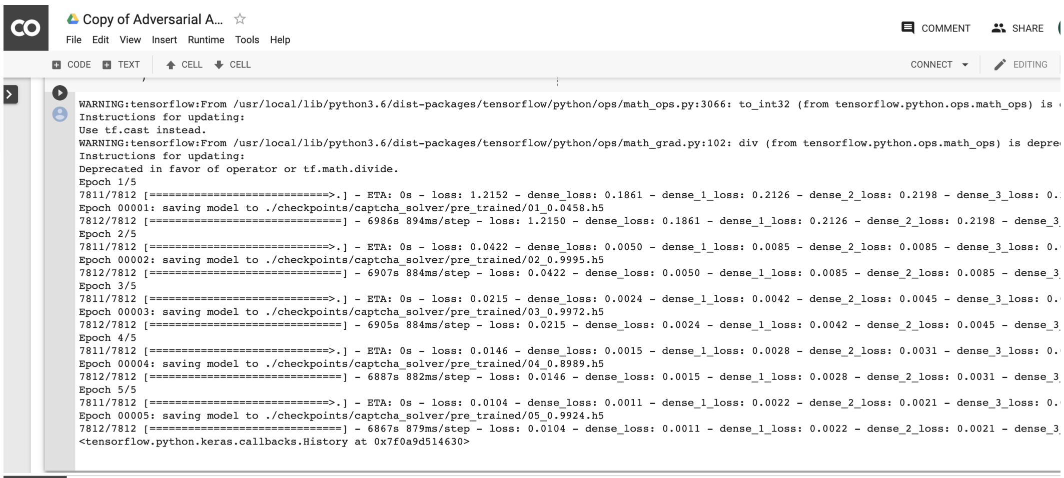
Figure 6.2: CAPTCHAs after the attack



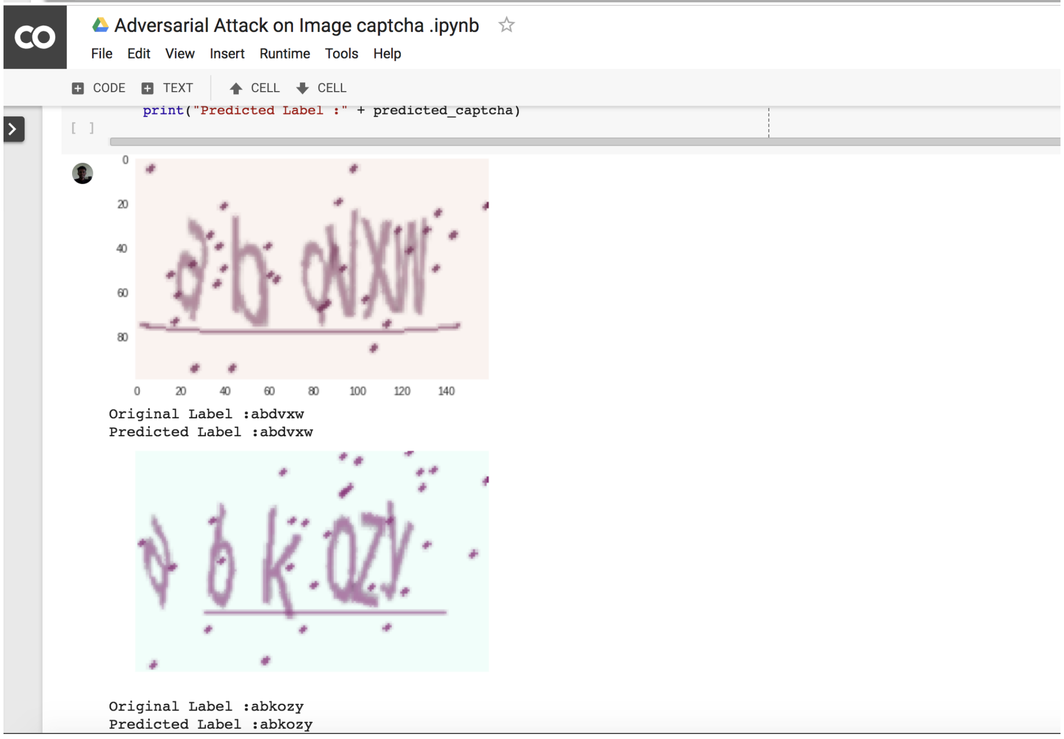
```
2019-03-02 04:21:32 PST step 8500, training accuracy = 24.67%, testing accuracy = 23.73%
2019-03-02 04:21:52 PST step 8600, training accuracy = 26.00%, testing accuracy = 23.52%
2019-03-02 04:22:11 PST step 8700, training accuracy = 27.33%, testing accuracy = 23.52%
2019-03-02 04:22:31 PST step 8800, training accuracy = 24.33%, testing accuracy = 23.58%
2019-03-02 04:22:51 PST step 8900, training accuracy = 24.67%, testing accuracy = 23.68%
2019-03-02 04:23:11 PST step 9000, training accuracy = 24.67%, testing accuracy = 23.88%
2019-03-02 04:23:31 PST step 9100, training accuracy = 24.33%, testing accuracy = 23.62%
2019-03-02 04:23:51 PST step 9200, training accuracy = 23.33%, testing accuracy = 23.66%
2019-03-02 04:24:11 PST step 9300, training accuracy = 21.33%, testing accuracy = 23.50%
2019-03-02 04:24:31 PST step 9400, training accuracy = 23.67%, testing accuracy = 23.27%
2019-03-02 04:24:51 PST step 9500, training accuracy = 25.33%, testing accuracy = 23.50%
2019-03-02 04:25:11 PST step 9600, training accuracy = 26.67%, testing accuracy = 23.73%
2019-03-02 04:25:31 PST step 9700, training accuracy = 24.67%, testing accuracy = 23.64%
2019-03-02 04:25:51 PST step 9800, training accuracy = 25.00%, testing accuracy = 23.72%
2019-03-02 04:26:11 PST step 9900, training accuracy = 22.33%, testing accuracy = 23.32%
2019-03-02 04:26:22 PST testing accuracy = 23.96%
2019-03-02 04:26:24 PST
################################################################################

2019-03-02 04:26:24 PST Waiting for container to complete...
2019-03-02 04:26:24 PST Persisting outputs...
2019-03-02 04:26:25 PST Creating data module for output...
2019-03-02 04:26:25 PST Data module created for output.
2019-03-02 04:26:25 PST Persisting data in home...
2019-03-02 04:26:44 PST Home data persisted.
2019-03-02 04:26:44 PST [success] Finished execution
```

Figure 6.3: Training done on FloydHub



```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:102: div (from tensorflow.python.ops.math_ops) is depre
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Epoch 1/5
7811/7812 [============================>.] - ETA: 0s - loss: 1.2152 - dense_loss: 0.1861 - dense_1_loss: 0.2126 - dense_2_loss: 0.2198 - dense_3_loss: 0.
Epoch 00001: saving model to ./checkpoints/captcha_solver/pre_trained/01_0.0458.h5
7812/7812 [==============================] - 6986s 894ms/step - loss: 1.2150 - dense_loss: 0.1861 - dense_1_loss: 0.2126 - dense_2_loss: 0.2198 - dense_3
Epoch 2/5
7811/7812 [============================>.] - ETA: 0s - loss: 0.0422 - dense_loss: 0.0050 - dense_1_loss: 0.0085 - dense_2_loss: 0.0085 - dense_3_loss: 0.
Epoch 00002: saving model to ./checkpoints/captcha_solver/pre_trained/02_0.9995.h5
7812/7812 [==============================] - 6907s 884ms/step - loss: 0.0422 - dense_loss: 0.0050 - dense_1_loss: 0.0085 - dense_2_loss: 0.0085 - dense_3
Epoch 3/5
7811/7812 [============================>.] - ETA: 0s - loss: 0.0215 - dense_loss: 0.0024 - dense_1_loss: 0.0042 - dense_2_loss: 0.0045 - dense_3_loss: 0.
Epoch 00003: saving model to ./checkpoints/captcha_solver/pre_trained/03_0.9972.h5
7812/7812 [==============================] - 6905s 884ms/step - loss: 0.0215 - dense_loss: 0.0024 - dense_1_loss: 0.0042 - dense_2_loss: 0.0045 - dense_3
Epoch 4/5
7811/7812 [============================>.] - ETA: 0s - loss: 0.0146 - dense_loss: 0.0015 - dense_1_loss: 0.0028 - dense_2_loss: 0.0031 - dense_3_loss: 0.
Epoch 00004: saving model to ./checkpoints/captcha_solver/pre_trained/04_0.8989.h5
7812/7812 [==============================] - 6887s 882ms/step - loss: 0.0146 - dense_loss: 0.0015 - dense_1_loss: 0.0028 - dense_2_loss: 0.0031 - dense_3
Epoch 5/5
7811/7812 [============================>.] - ETA: 0s - loss: 0.0104 - dense_loss: 0.0011 - dense_1_loss: 0.0022 - dense_2_loss: 0.0021 - dense_3_loss: 0.
Epoch 00005: saving model to ./checkpoints/captcha_solver/pre_trained/05_0.9924.h5
7812/7812 [==============================] - 6867s 879ms/step - loss: 0.0104 - dense_loss: 0.0011 - dense_1_loss: 0.0022 - dense_2_loss: 0.0021 - dense_3
<tensorflow.python.keras.callbacks.History at 0x7f0a9d514630>
```



Adversarial Attack on Image captcha .ipynb
File Edit View Insert Runtime Tools Help
CODE
TEXT
CELL
CELL
0
20
40
60
80
0
20
40
60
80
100
120
140

```
print("Predicted Label :" + predicted_captcha)
```

```
Original Label :abdvxw
Predicted Label :abdvxw
```

```
Original Label :abkozy
Predicted Label :abkozy
```

Figure 6.4: Training done on Google Colab

Epoch	**dense_loss**	**dense_acc**	**val_dense_loss**	**val_dense_acc**
1	0.1701	0.9511	12.2047	0.0526
2	0.0035	0.9993	0.0024	0.9998
3	0.0020	0.9996	0.0083	0.9984
4	0.0013	0.9997	0.4325	0.8964
5	7.9859e-04	0.9998	0.0813	0.9826

Table 6.1: Accuracy status after training

Since the loss is decreasing and the accuracy is found to be increasing, it indicates that the model build is learning and working fine.

CHAPTER 7

CONCLUSION

In the world of existing advanced AI, there exists a security threat to CAPTCHAs as mentioned. This project implemented in the form of a web application, provides a simple and secure solution that deceives the deep learning tools and cannot be removed using preprocessing (however they are not robust to affine transformations). The same model can be used to secure social media pictures from being automatically tagged by bots. Additionally, adversarial networks can have a wide range of applications and is sure to lead to new research outcomes.

REFERENCES

[1] **Elsayed, G. F.**, **I. Goodfellow**, and **J. Sohl-Dickstein** (2018). Adversarial reprogramming of neural networks. *CoRR abs/1806.11146*.

[2] **Goodfellow, I. J.**, **J. Shlens**, and **C. Szegedy** (2015). Explaining and harnessing adversarial examples. *Conference paper at ICLR*.

[3] **Gu, S.** and **L. Rigazio** (2014). Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*.

[4] **Tramèr, F.**, **A. Kurakin**, **N. Papernot**, **I. Goodfellow**, **D. Boneh**, and **P. McDaniel** (2018). Ensemble adversarial training: Attacks and defenses. *Conference paper at ICLR*.