

Fake Image Detection

Combating Image Forgery with Real-Time Digital Analysis

Daniel [REDACTED]

[REDACTED]

Computer Science Capstone

July 2020

Table of Contents

Project Proposal	4
Problem	4
Benefits	4
Description	4
Data	5
Objectives	5
Hypothesis	5
Project Methodology	5
Funding Requirements.....	6
Stakeholder Impact.....	6
Ethical and Legal Considerations	6
Expertise and Experience.....	6
Executive Summary	7
Problem Statement.....	7
Customers	7
Existing Systems	7
Available Data	8
Methodology	8
Requirements Gathering.....	8
Design	9
Implementation	9
Verification	9
Deployment	10
Maintenance	10
Deliverables	10
Implementation Plan	10
Validation	11
Environment and Funding Requirements	11
Timeline	12
Product Attributes	13
Descriptive Methods.....	13

Error-level Analysis (ELA)	13
Principle Component Analysis (PCA)	16
Luminance Gradient (LG)	17
Predictive Method	18
Security Features	21
Decision Support.....	21
Confusion Matrix.....	21
Accuracy Assessment	22
Documentation	23
Business Vision	23
Assessment of the Hypothesis	23
Data Preparation.....	24
Data Visualization and Interaction	25
Application Testing	28
Image Pre-processing.....	28
Convolutional Neural Network	29
GUI Testing Program	31
Quick Start Guide	32
Downloading	32
Prerequisites	32
Installation	33
Training	34
Testing Tool	34
API	35
Table of Figures	36
References	37

Project Proposal

Problem

Social media allows users to upload any photo they want, either as a profile image or to a news feed, depending on the site in question. Other than simple obscenity checks, there is no verification or detection that an image may be altered (Citraoën, 2017). Altered images can have significant negative effects if the viewer of the image cannot know if it is fake or otherwise altered. For a dating website, this can be demonstrated in customer dissatisfaction when a profile photo does not accurately reflect the person in question. For a social media networking platform, this can sow distrust between the users of the platform and the platform owners, or worse, sow distrust between other users and governments if fake imagery is used to spread propaganda. To address these issues, a predictive tool can be written to better inform viewers that a said image is authentic.

Benefits

This application includes a Dynamic-link library (DLL) which provides an Application Programming Interface (API) designed as a middleware solution to easily integrate the image detection system into any website or application. Once integrated, a simple visual interface can be displayed to the user indicating the probability that the image being viewed is fake or has otherwise been altered in some way.

Description

The application and middleware API library both utilize the .NET Core framework, the keras Python library, and OpenCV. The .NET Core framework enables rapid development with cross-platform runtime support. The keras framework is responsible for the model, which performs the categorization and

prediction of images. OpenCV is responsible for pre-processing the images for input into the keras framework.

Data

The initial data used to build the model will be obtained from the 140k Real and Fake Faces notebook on Kaggle (xhlulu, 2020). It is a combined dataset of real images obtained from Flickr by NVIDIA, as well as a set of fake images generated by Bojan Tunguz.

The data contains a total of 140k images. This is split evenly between real (70k) and fake (70k) categories. Each category is further split into a training set (40k), validation set (20k), and testing set (10k).

Objectives

The goal of this project is to provide an interface for any social media website or platform operator to integrate into their platform to detect falsified images. This can be used to ensure a dating website's user profiles have images that accurately portray the user, as well as for social networks such as Facebook or Twitter, to help fight the spread of fake news and propaganda imagery.

Hypothesis

A predictive model can identify fake or altered images.

Project Methodology

The project will be developed using the Waterfall methodology. This methodology is suited to this project as the development will consist of a single person, the project is small, and the requirements are well defined and unlikely to change during the project development cycle. Once the initial project is completed, further development can proceed using either waterfall or another method, as best determined by the project manager.

Funding Requirements

The estimated funding for this project comes out to \$20,000 in development costs (200 hours × \$100/hour), and an additional \$2,300 per machine for the additional hardware (GPU) to train and run the model self-hosted or \$180/month per server for the GPU hardware for cloud-hosted. This pricing is on top of any current or additional expenses for the base system, configuration, and maintenance required to run this project.

Stakeholder Impact

Facebook is one of the most popular social media networks in use. Ensuring that the images users of the network see are real, legitimate images and not falsified increases the confidence the users have in the social network. Confidence in the network increases user retention, which increases revenue from increased ad views and clicks.

Ethical and Legal Considerations

The data used in this project is anonymized and does not contain any private, personally identifiable information (PII). Future data fed into the system will be equally anonymized and will be untraceable to the original user posting said image.

Care must be taken with this project, as it could be misused to train image generators to create images that can fool the detection algorithm.

Expertise and Experience

The developer has 15 years of combined experience with software development and photographic manipulation. This overlapping knowledge is well suited to this project for both development and generating test images.

Executive Summary

Problem Statement

Fake images on social media services such as Facebook are typically accompanied by fake news articles. Fake news and images increase divisiveness, reduce trust in the service, and reduce overall user engagement with the service. This reflects in reduced ad interaction, and overall reduced ad revenue for the social network (Sloane, 2018). A second concern is that a platform that popularizes and allows fake news and images to proliferate will come under government scrutiny in cases such as manipulating an image of a world leader in a place they were never at (Fowler, 2018). Combating this problem, and ensuring social network users see not only relevant, but real content, will keep user trust and engagement high, increase ad revenue, and increase trust from lack of government scrutiny.

Customers

This application is intended to be a middleware solution, applied to an existing social media platform. The model and dataset required to train it can be configured independently from the main website. After it is set up, the API calls into the application can be made without significant changes to the existing system. Each image uploaded to the platform can be processed through the network and the probability returned can be saved along with the image. Each retrieval thereafter can include the probability of it being a fake, and a warning can be displayed to the user.

Existing Systems

Instagram is the only social media platform with fake image detection. Facebook has fake news detection, but not fake image detection. This system can be deployed to augment the existing fake news detection and remain competitive.

Available Data

The data used to train this model is published on the Kaggle website, under the title 140k Real and Fake Faces (xhlulu, 2020). This dataset is a combination of images from two different sources, which corresponds to the two-classification categorized used: real and fake.

The real data was sourced from the Flickr photo-sharing website by NVIDIA. This dataset was then resized to 256x256 by the user xhlulu. The fake images were generated by Bojan Tunguz, using the Nvidia StyleGAN network (NVIDIA, 2019), and resized to 256x256 by xhlulu.

This dataset is specifically human faces, which is ideal for detecting user profile images. For other types of image detection, alternate data sets can be compiled and used instead of, or in addition to, this one.

Methodology

This product will utilize the Waterfall development process.

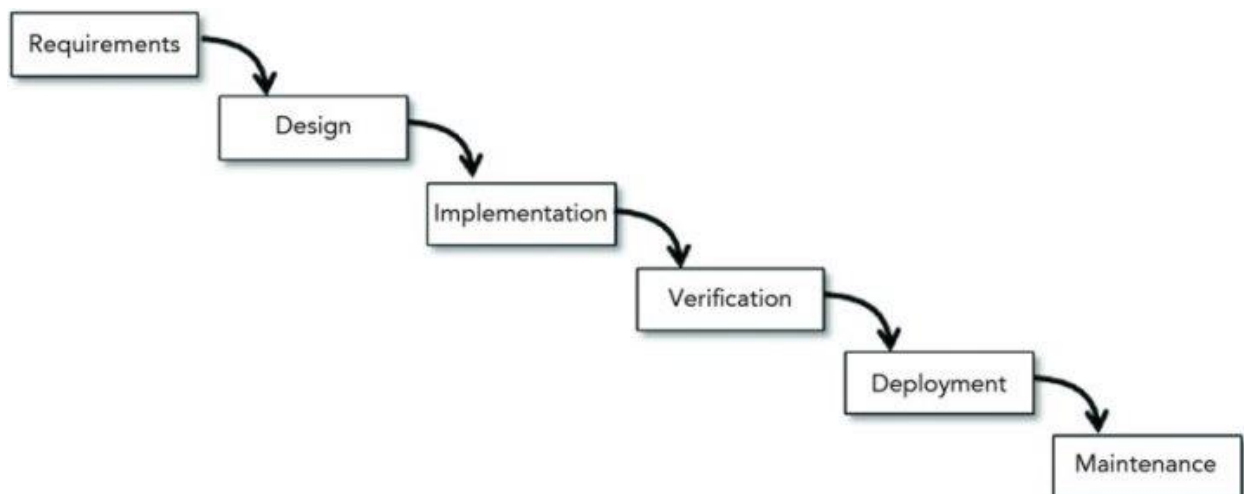


Figure 1: Waterfall model (Stephens, 2015)

Requirements Gathering

Kicking off the project will commence with a meeting containing all stakeholders to elicit and analyze the requirements and scope of the project. The requirements and scope must be completely finalized

before continuing to the next step. No further changes should happen after this phase is complete. All requirements should be documented for future use.

Design

During this research phase, an investigation will be done to determine if there are suitable tools and libraries available to accelerate the process, or if any such tools will require development themselves.

This phase will also contain the design of the application project itself. This will include any APIs that are necessary to satisfy the requirements specified in the previous phase, as well as planning for future extension of scope and requirements. As with the previous phase, the design should be finalized and frozen before progressing to the next phase.

Implementation

This phase is where the development will happen, and the system will be built according to the design written in the last phase. This phase can take advantage of parallel development if multiple developers are available. The high-level application product features will include the machine learning network model, various image pre-processing algorithms to support the aforementioned model, an API interface for integrating as middleware into an existing web or platform solution, and a graphical user interface (GUI) program for continuously testing and validating the model is predicting accurately. The application coding should be complete before moving on to the verification phase.

Verification

Thorough testing of the final application will be performed during this phase. This will include a GUI tool, mentioned above, to analyze an image and the network and provide visual feedback to the user the selected algorithm, the probability of the best match (real or fake) of the said algorithm, as well as the probability of the network as a whole. Additional verification will be performed using the feedback generated during the training process to track the accuracy and loss of the model. Finally, the

application system as a whole will be tested against the requirements, to ensure it will meet expectations.

Deployment

This phase is completed when the application is integrated with the main site or platform and is functioning properly for users of the platform.

Maintenance

Continued evaluation of the application will be performed regularly to ensure it is still functioning adequately, and that the model is continuously being improved. Any defects encountered, or new features to be implemented, will occur during this phase.

Deliverables

The completed application will contain the following primary deliverables: A command-line executable for pre-processing the data and training the model, a GUI application for testing the model and algorithm processing, and the API middleware library for integrating into an existing site or platform.

The secondary deliverables are dependencies of the primaries: the .NET Core 3.1 runtime with the keras.net and OpenCVSharp packages, and if not already present, the Python 3.7 interpreter and runtime with the keras, plaidml-keras (for AMD), and TensorFlow packages.

Implementation Plan

The implementation of the application will be as follows:

1. Gather a balanced dataset of human faces containing real, unedited images, as well as fake or altered images.
2. Utilize descriptive methods to reduce the dataset variables.

- a. The algorithms selected for this step include Error-Level Analysis, Principle Component Analysis, and Luminance Gradient (Krawetz, 2008). Additional methods can be added in the future with minimal development effort.
3. Utilize non-descriptive methods to build predictive models. Each descriptive method used will map to a separate predictive model.
4. Implement an API middleware solution, which will be the interface between the descriptive and prescriptive methods, and the user interface. This interface will ultimately be used in the target website or platform.
5. Build a GUI program using the above API for providing visual feedback to aid in testing and debugging the descriptive methods and predictive models.
6. Deploy the model and API middleware solution into the target platform.
7. Perform acceptance testing and validation of the model.

Validation

The application will be validated by the GUI program dashboard, which will provide prediction results of a supplied image. The training executable will output logs that document the training and validation loss and accuracy, which will be used to fine-tune the model. The overall target accuracy is no less than 80%.

Environment and Funding Requirements

The equipment and environment funding will depend on whether the application will be deployed to a self-hosted platform, or hosted on a third-party cloud platform. All costs will assume existing infrastructure, and will only list costs that are associated with this project specifically.

All prices are in USD.

Self-hosted hardware	NVIDIA T4 x3 @ \$2,300	\$6,900
Cloud-hosted hardware	AWS w/GPU Add-on @ \$180/month	\$540/month
Development costs	200 man-hours @ \$100/hour	\$20,000
Maintenance	\$100/hour	Varies
Total		Self-hosted: \$26,900 up-front Cloud-hosted: \$20,000 up-front + \$540/month Maintenance: \$100/hour

Timeline

This is the projected timeline of the project. The dates may adjust accordingly if there are any obstacles or boosts.

Phase	Begin	End	Duration	Human Resources
Requirements	6/29/2020	6/29/2020	1 day	2-3
Design	6/30/2020	7/3/2020	4 days	1-2
Implementation	7/6/2020	7/24/2020	15 days	1-3
Testing	7/27/2020	7/30/2020	4 days	2-4
Deployment	7/31/2020	7/31/2020	1 day	1
Total			25 days	

Product Attributes

Descriptive Methods

Several descriptive methods were used to better eliminate unwanted data and to ensure the models are training on the correct information. Each method has strengths and weaknesses, so the use of multiple methods better ensures accurate results.

Error-level Analysis (ELA)

This involves taking an image at a known quality level, and resaving it at a lower quality, then calculating the difference between them (Hacker Factor, 2020). This difference yields the error level potential of the image.

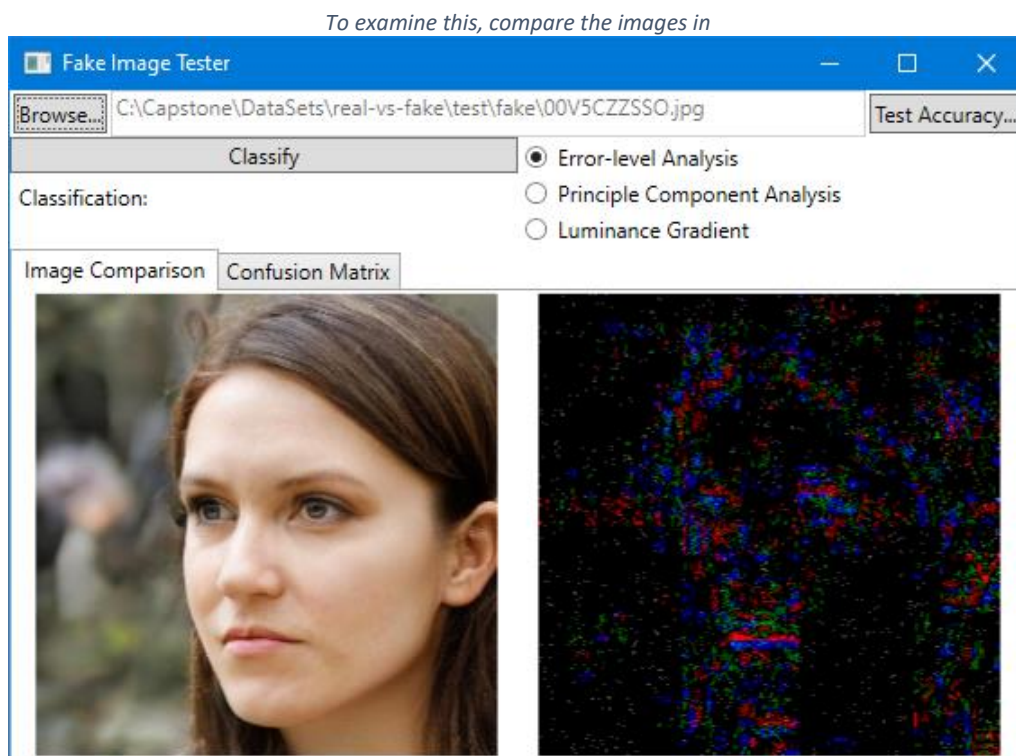


Figure 2: Computer generated face showing uneven error potential

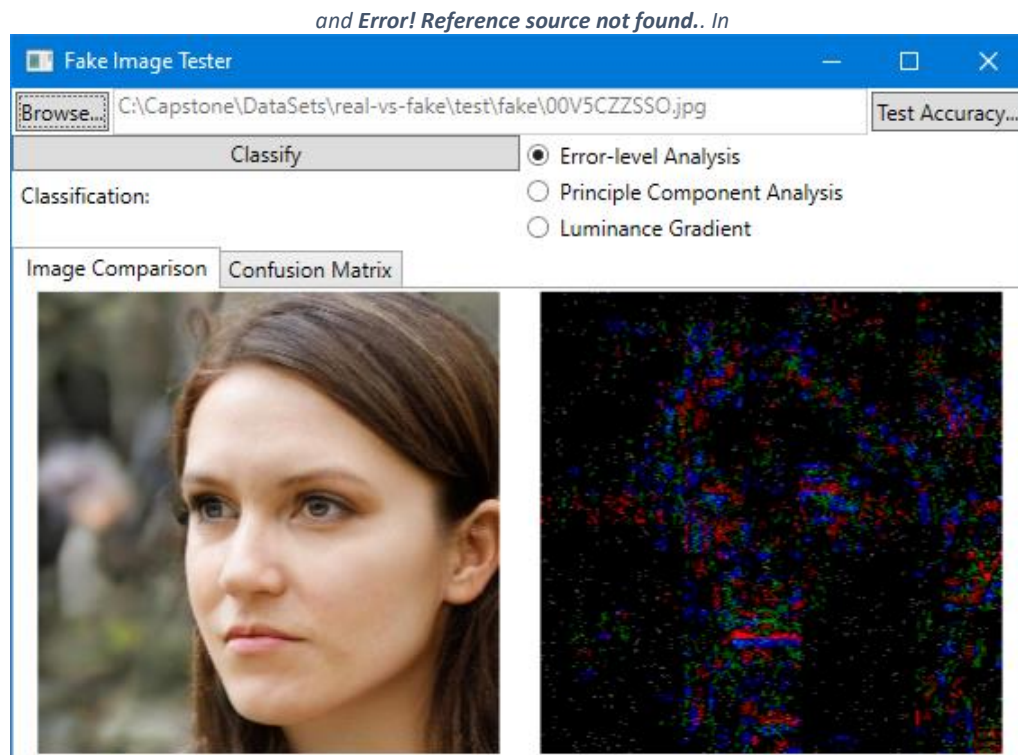


Figure 2: Computer generated face showing uneven error potential

, the woman in the picture is a fabrication, generated by a computer. The ELA indicates the image doesn't have the same error level potential throughout, as can be seen particularly around the hairline, eyes, and chin. Compare this to **Error! Reference source not found.**, which has a noticeably lower error potential. The image is softer and doesn't have as many bright spots, which are suspect areas. The exception is an extra bright region encompassing the woman's earrings and teeth, which are highlighted due to the camera flash catching those spots. Strong contrast in an image can reduce the reliability of this particular algorithm.

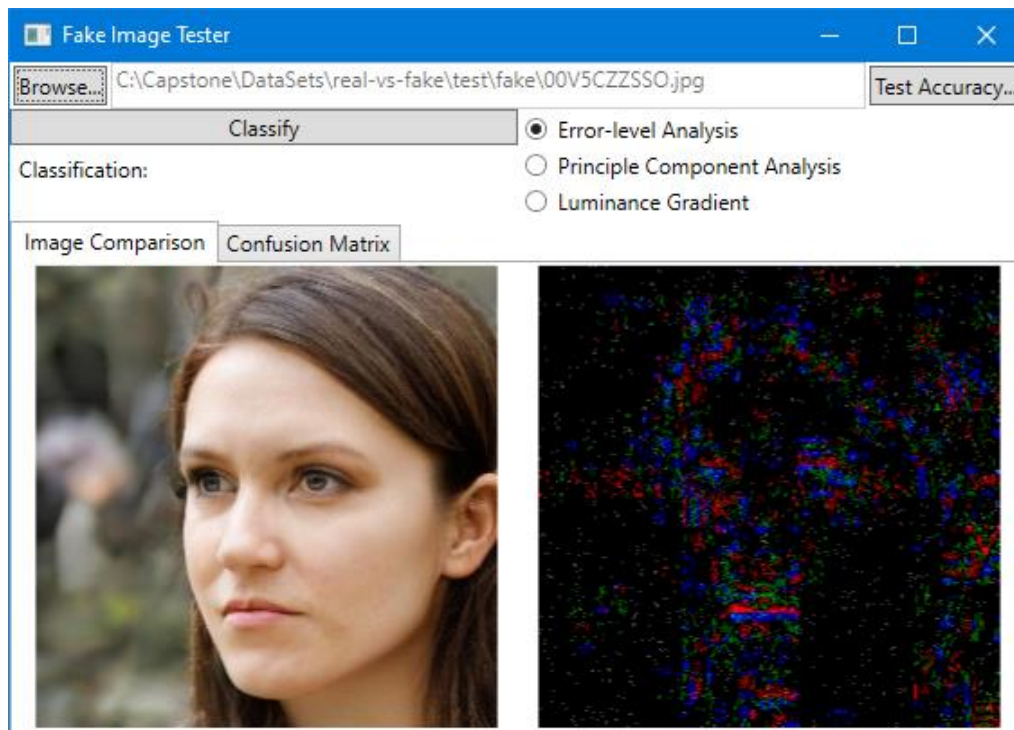


Figure 2: Computer generated face showing uneven error potential

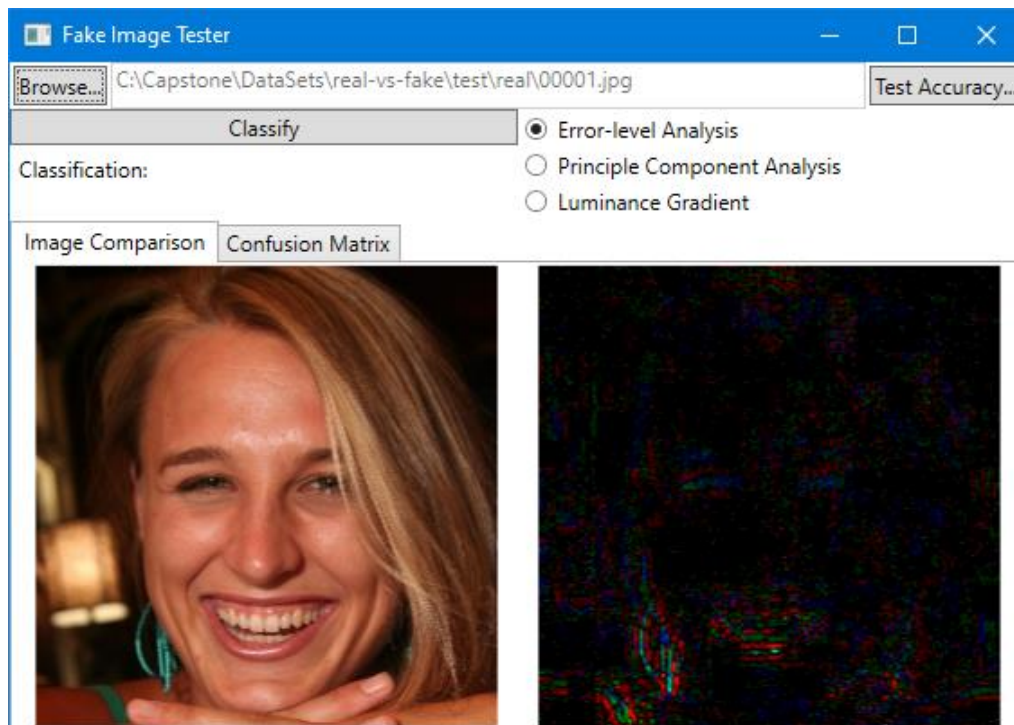


Figure 3: A real human face, showing an even error potential.

Principle Component Analysis (PCA)

This method is used for clustering data points. “Each principal component defines a plane across the data set. The first principal component (PC1) identifies a plane with the widest variance across the data. In effect, the average distance from every point to the PC1 plane is maximized” (Krawetz, 2008, p. 13).

*The PCA is useful for detecting, among other things, compression artifacts (blocky rectangular regions) in JPEG images. This is demonstrated in Figure 4. The left image is the PC2 display of the same real face referenced in **Error! Reference source not found.**, showing a consistent artifact pattern throughout the image (in this case, very little). Compare to the right-hand computer-generated face from*

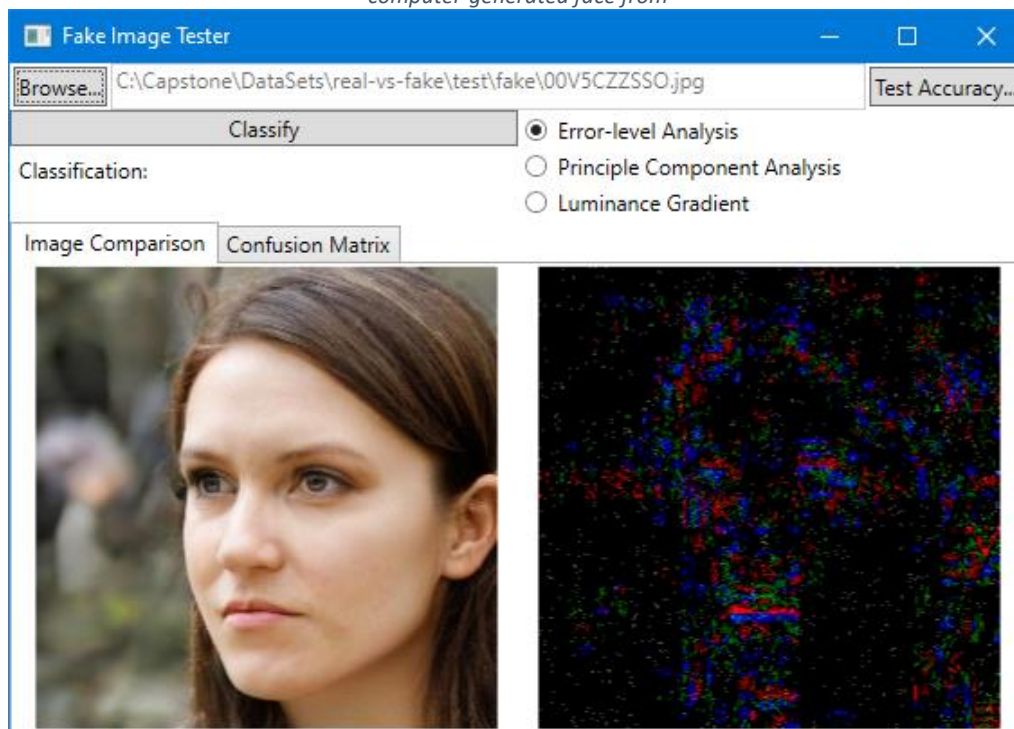


Figure 2: Computer generated face showing uneven error potential

, which shows varying levels of compression; high-compression alongside the hair, eyebrows, and cheeks, but low compression around the nose, mouth, and forehead.



Figure 4: PC2 Comparison of a real face (left), and a computer-generated face (right).

Luminance Gradient (LG)

This method is primarily used to identify the direction of light sources. However, the basis of this method is also used for edge detection, and as such, it can also be used to detect many different kinds of photographic alterations, or computer-generated images, that other methods cannot catch.

Real images will have an even noise pattern, while an edited or fake image will contain different noise patterns and harsh edges.

See the comparison in Figure 5. In the real image on the left, the noise pattern is fairly consistent throughout the image and there are very few sharp lines. On the right side, however, several abnormalities stand out; the noise on the hair is uneven and doesn't match the face or background, nor does the background noise level match the face. Also, the red outline along the subject's right cheek and chin (image left) are sharp compared to the same location in the left image. The final obvious marker is how clearly outlined the pupil of the eyes are.

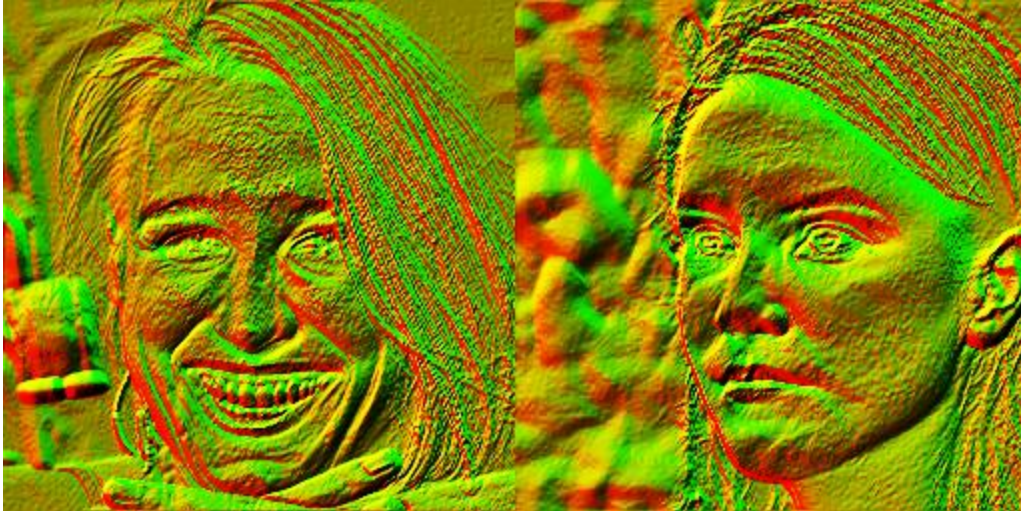


Figure 5: Luminance gradient comparison of a real face (left), and a computer-generated face (right).

Predictive Method

The method used here is a 22-layer very deep convolutional neural network, based on the VGG algorithm by Simonyan & Zisserman (2015). Additional layers were added to reduce the parameters of the network and help it focus on the details we wanted to detect.

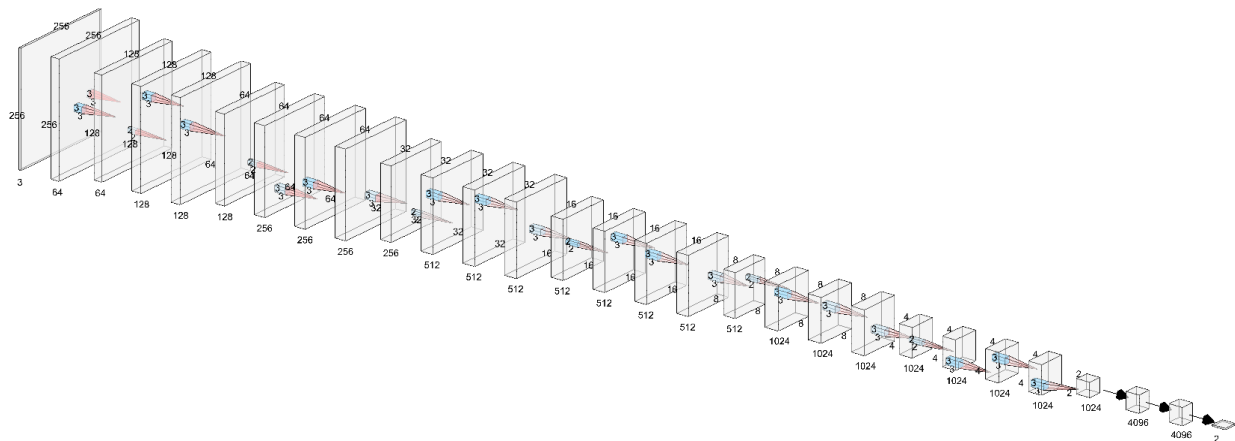


Figure 6: VGG21 convolutional neural network used.

As shown in the figure above, the convolutional neural network is composed of 18 convolutional layers, 3 dense activation layers, and 7 max-pooling layers. The total number of parameters (weights) of the network is 100 million.

A convolutional neural network was chosen as a “CNN does not require hand-crafted feature extraction” (Yamashita, Nishio, & Do, 2018). Due to the sheer volume of data posted on social networks and the internet at large every day, manual intervention and network adjustments for each image is an impossible task. Instead, the network itself does the heavy lifting.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 256, 256, 64)	1792
conv2d_2 (Conv2D)	(None, 256, 256, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 64)	0
conv2d_3 (Conv2D)	(None, 128, 128, 128)	73856
conv2d_4 (Conv2D)	(None, 128, 128, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 128)	0
conv2d_5 (Conv2D)	(None, 64, 64, 256)	295168
conv2d_6 (Conv2D)	(None, 64, 64, 256)	590080
conv2d_7 (Conv2D)	(None, 64, 64, 256)	590080
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 256)	0
conv2d_8 (Conv2D)	(None, 32, 32, 512)	1180160
conv2d_9 (Conv2D)	(None, 32, 32, 512)	2359808
conv2d_10 (Conv2D)	(None, 32, 32, 512)	2359808
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 512)	0
conv2d_11 (Conv2D)	(None, 16, 16, 512)	2359808
conv2d_12 (Conv2D)	(None, 16, 16, 512)	2359808
conv2d_13 (Conv2D)	(None, 16, 16, 512)	2359808
max_pooling2d_5 (MaxPooling2D)	(None, 8, 8, 512)	0
conv2d_14 (Conv2D)	(None, 8, 8, 1024)	4719616
conv2d_15 (Conv2D)	(None, 8, 8, 1024)	9438208
conv2d_16 (Conv2D)	(None, 8, 8, 1024)	9438208
max_pooling2d_6 (MaxPooling2D)	(None, 4, 4, 1024)	0
conv2d_17 (Conv2D)	(None, 4, 4, 1024)	9438208
conv2d_18 (Conv2D)	(None, 4, 4, 1024)	9438208
conv2d_19 (Conv2D)	(None, 4, 4, 1024)	9438208
max_pooling2d_7 (MaxPooling2D)	(None, 2, 2, 1024)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dense_2 (Dense)	(None, 4096)	16781312
dense_3 (Dense)	(None, 2)	8194
Total params: 100,196,162		
Trainable params: 100,196,162		
Non-trainable params: 0		

Figure 7: The VGG network in detail, showing the weights of each layer.

Security Features

This application is not directly accessible to users and is only available to the backend developers. It also does not need to access any RDBMS or store sensitive data. As such, additional security precautions are not necessary and are not provided.

Decision Support

The application enables automatic detection of images to determine whether the specified image is real, or has been altered or faked. The confusion matrices described below indicate the matching accuracy of each descriptive algorithm, as well as the accuracy of the entire network as a whole.

The convolutional neural network breaks down each image into increasingly smaller samples; this is the feedforward stage. The samples are reprocessed through the network to build the original image; this is backpropagation. The parameters derived during the backpropagation are the weights and are saved in the network. Future images fed into the network will have those weights applied, and then the network will classify the image into one of the two categories specified in this application: fake, or real. The actual results of the classification are a set of probabilities of the image matching each class in the network. For example, if the prediction result is the set of [0.4, 0.6] for the classes [fake, real], respectively, then the image has a 40% chance of being fake and a 60% chance of being a real image. To aid in decision making, this probability is rounded, and the result becomes [0, 1] for the previous example. The result then is the image is determined to be real.

Confusion Matrix

Here are the confusion matrices for each network that represents the three descriptive algorithms, as well as combined. The data set used is a total of 20k images, 10k real, and 10k fake (the combined matrix indicates 60k images, 3x each of 20k). These images were not used in either the training or validation datasets. As far as the application is concerned, it has never seen any of these images before.

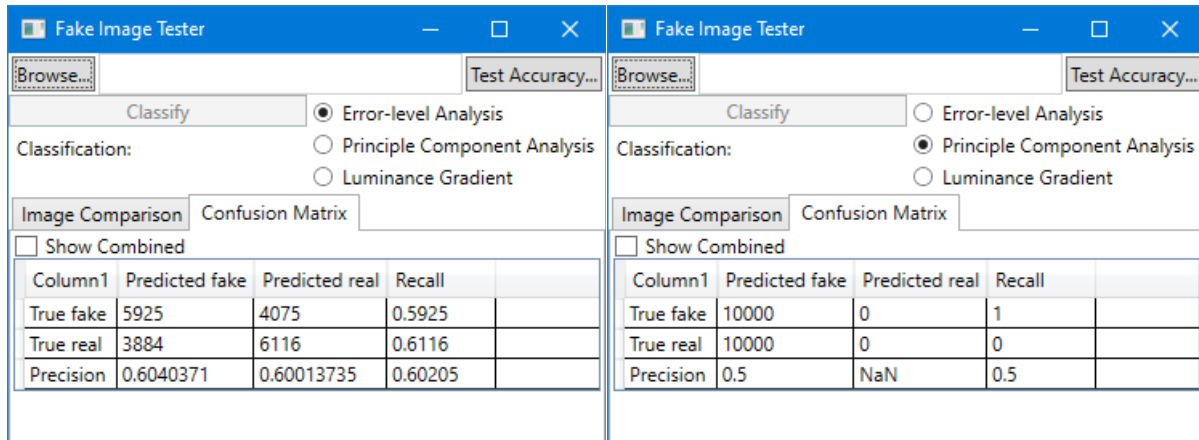


Figure 8: ELA Confusion Matrix, 60% accuracy

Figure 9: PCA Confusion Matrix, 50% accuracy

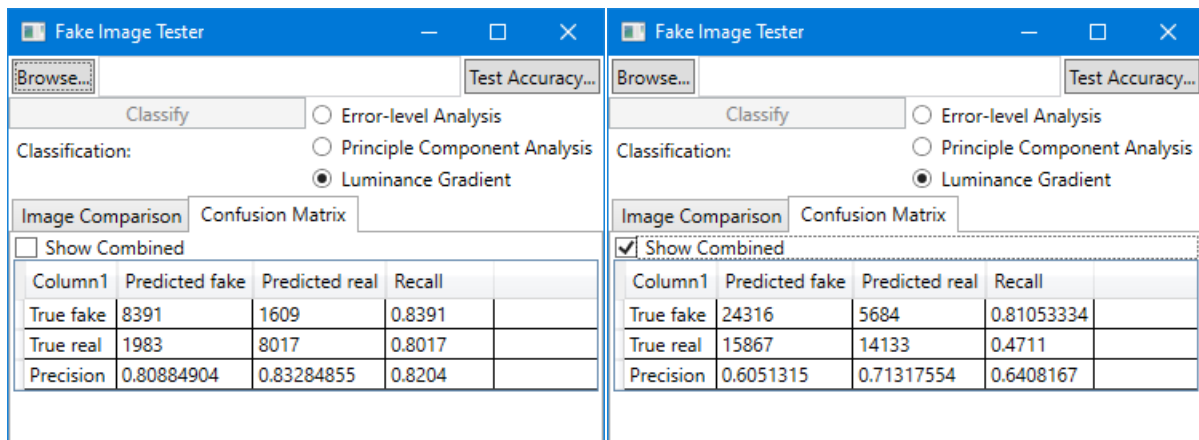


Figure 10: LG Confusion Matrix, 82% accuracy

Figure 11: Combined Confusion Matrix, 64% accuracy

The application supports returning a single “yes” or “no” value, as well as the actual probabilities of the class matches if more detailed processing is required before a decision is reached.

Accuracy Assessment

As indicated in the confusion matrices (Figure 8-Figure 11), the ELA algorithm model provides an accuracy rating of 60% (lower right cell of the matrix), while the LG algorithm provides a usable 82% accuracy. However, the PCA model was unable to train to an accuracy better than 50%. Excluding that specific algorithm from the application will boost overall accuracy from 64% to 71%.

There is room for improvement, both with the image processing algorithms used, as well as the models.

Documentation

Business Vision

Social networking platforms provide valuable methods of communication between friends and family. They are also an important tool in the sharing of knowledge. However, it's important to ensure that the knowledge shared is trustworthy and reliable. If the users of the network see any information that is false and shared under the pretense of spreading misinformation, that negatively affects the reputation of the social network.

The goal of this application is to give social networks an additional tool to help combat the spread of false information on their platform. By using this application to supply a warning to users that see the false information, the users can be better informed and retain confidence that the social network is protecting knowledge.

Users that are informed about the information they are seeing will be more likely to remain and use the social networking platform more often. This increased usage will improve advertising revenue, and further grow the business.

Assessment of the Hypothesis

The assessment of the model, as described in the previous section, supports the hypothesis that a predictive model can be used to reliably detect fake or altered images.

While the specific model parameters provided in this application are not ideal, with further refinement this model's accuracy can be improved to reach 90% or above.

Data Preparation

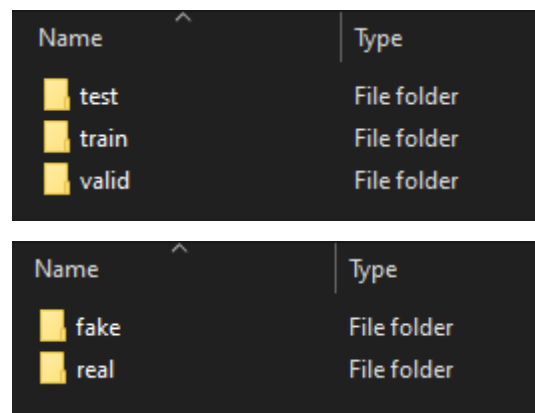
The source dataset used was a combination of two original datasets, all available on Kaggle.com. The author, xhlulu, made changes themselves, “In this dataset, I convenient[sic] combined both dataset[sic], resized all the images into 256px, and split the data into train, validation and test set” (2020).

The application itself comes with a training tool. This tool includes the ability to further pre-process the input dataset so the images can be supplied to their respective neural networks.

The pre-processing step can be executed manually during installation, or automatically when a model begins training.

The source dataset should be split into 2-3 groups: a training set, a validation set, and optionally, a testing set.

Inside each of those three folders, there should be two additional folders. These folders represent the model classes the application intends to match against, in this case, *fake* and *real*.

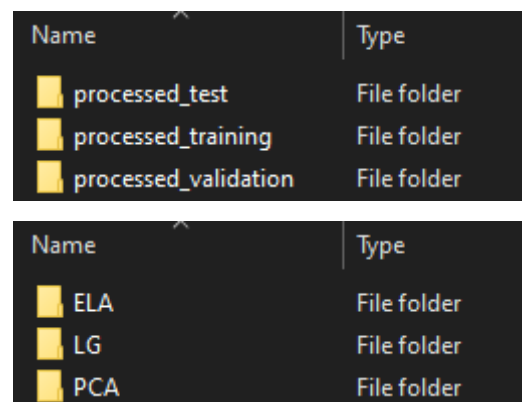


Name	Type
test	File folder
train	File folder
valid	File folder

Name	Type
fake	File folder
real	File folder

Figure 12: A properly prepared dataset.

Once the raw dataset has been prepared, the training application will pre-process the dataset. It does this by passing each image through each of the three descriptive algorithms available: ELA, PCA, and LG. The results are placed into a new location specified on the command-line at execution time. This output location will be split into folders that match the name of the algorithm used and each of those folders inside follows the same pattern above.



Name	Type
processed_test	File folder
processed_training	File folder
processed_validation	File folder

Name	Type
ELA	File folder
LG	File folder
PCA	File folder

Figure 13: Pre-processed dataset, ready for training.

Data Visualization and Interaction

In addition to the confusion matrices shown in a previous section, there are additional ways of viewing the dataset and model information.

The GUI program supplied with the application allows the user to visualize and examine many different aspects of the data. This includes seeing the post-processed data described in the last section, the confusion matrix for each model format, and the application in its entirety, as well as the respective classifications and probabilities of each algorithm plus the system as a whole for each image.

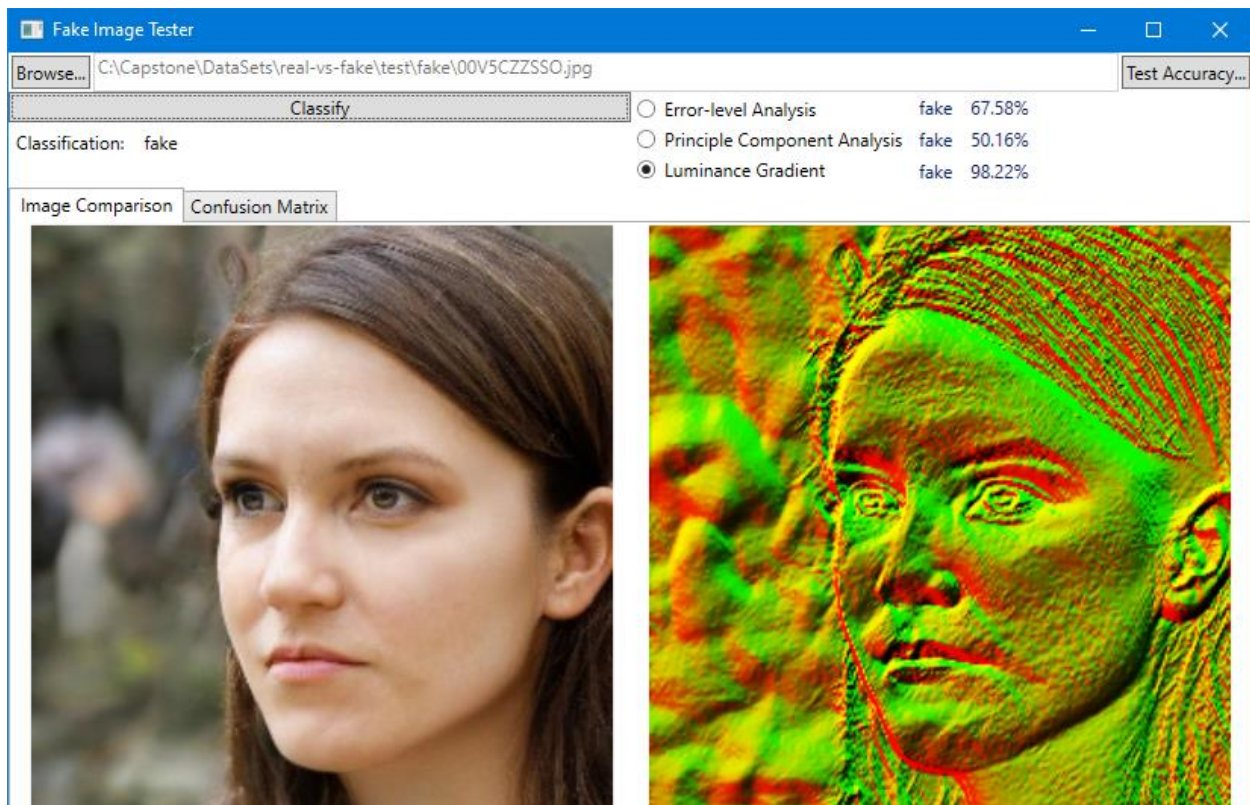


Figure 14: An image correctly detected as a fake.

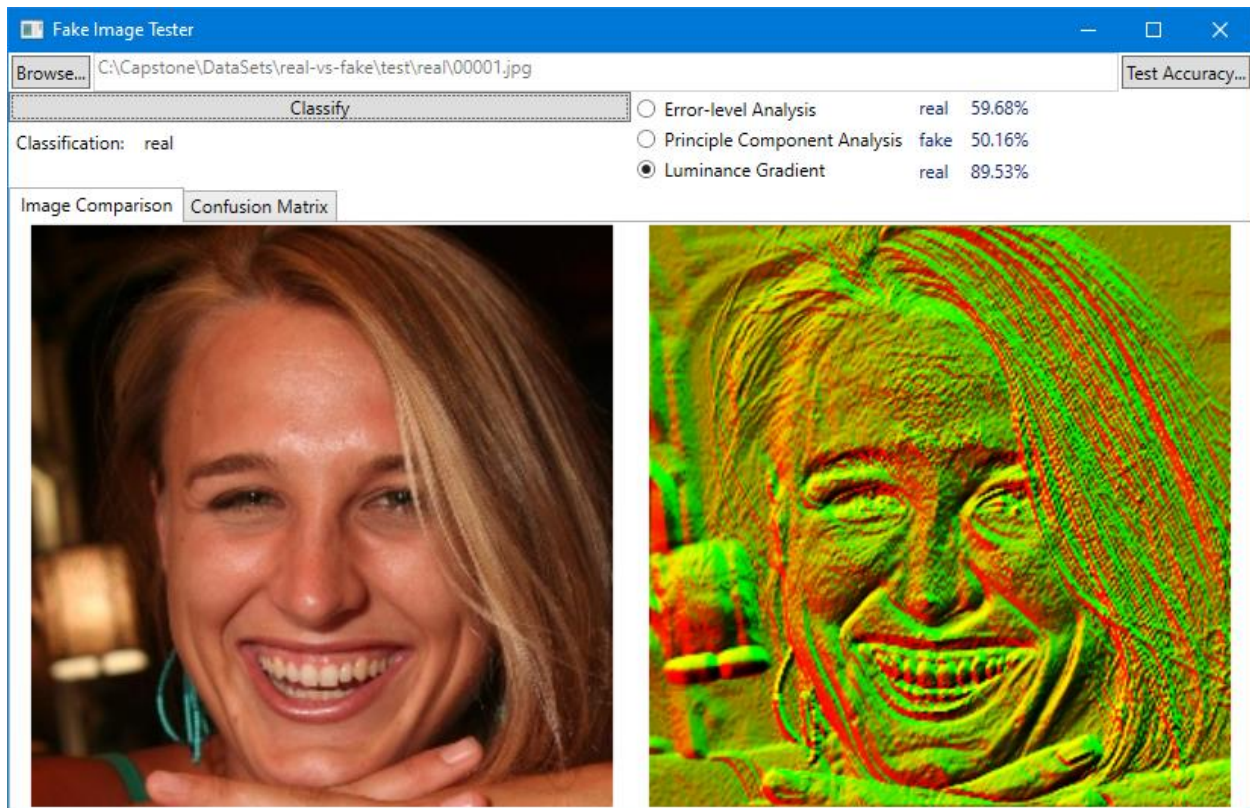


Figure 15: An image correctly detected as real.

During the training process, information concerning the accuracy and loss functions are logged. Those values can be processed to produce a visual representation of the training process.

This visual diagram provides an easy to interpret guide for how well the network is training to the supplied dataset.

To determine if a network is training correctly, the Loss and Validation (Val) Loss values should decrease over time, and the Accuracy and Validation (Val) Accuracy should increase over time. If the Val Accuracy is higher than Accuracy, that means the network is overfitting to the training data, and will not function adequately with new data it hasn't seen before.

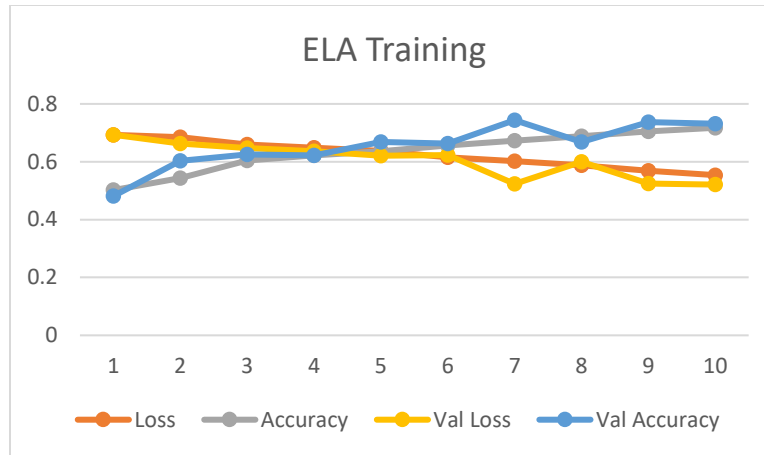


Figure 16: ELA model's loss and accuracy. The model functions, but needs work.

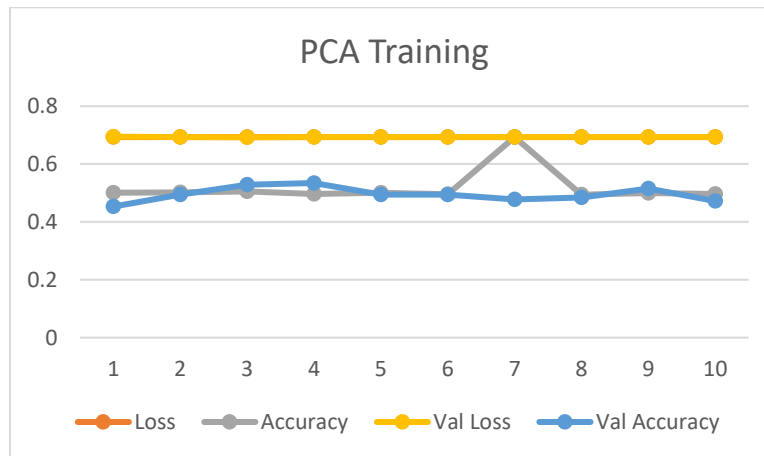


Figure 17: Chart of loss and accuracy for the PCA model. The model stagnates, is unable to progress.

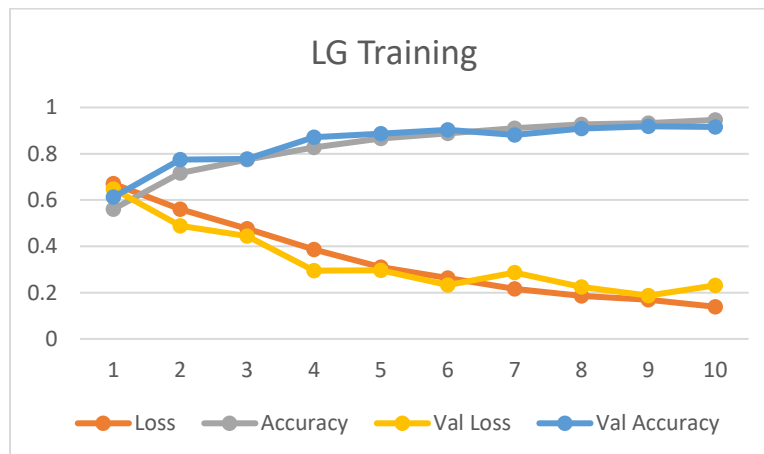


Figure 18: Chart of loss and accuracy for the LG model. The model is trained well and is ideal.

Application Testing

The development and testing of the application were done in two phases: The image pre-processing algorithms, and the neural networks themselves. The image pre-processing algorithms were first.

Image Pre-processing

Pre-processing development became with the core editing layer, and user-controllable options to determine the best fit for the algorithm. This code for this support is included in the training tool itself via a specific command-line option.

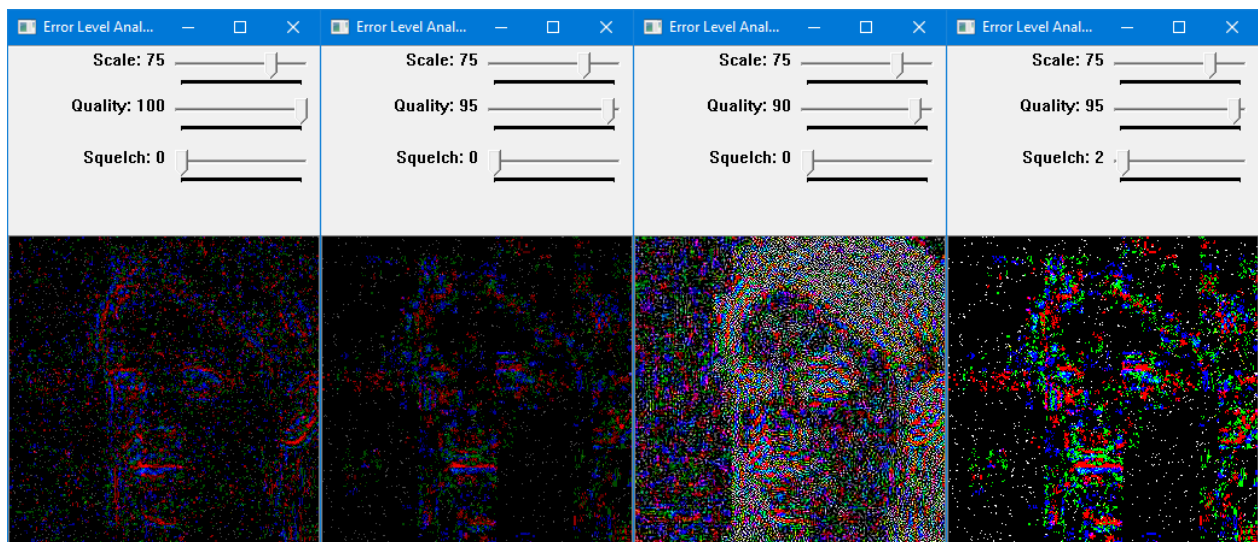


Figure 19: Manual testing of different ELA parameters to determine the best fit.

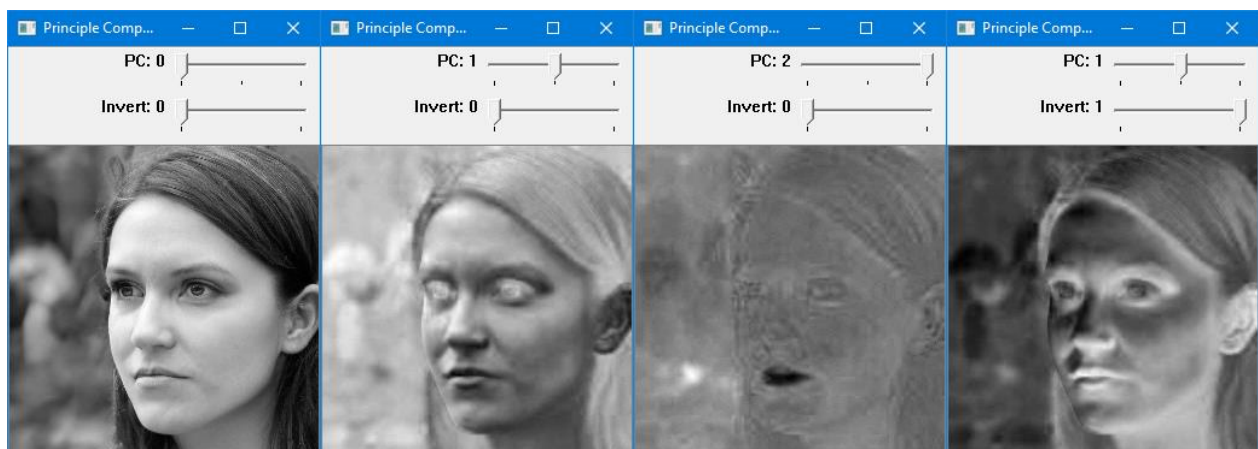


Figure 20: Manual testing of different PCA parameters. Note that the PC range here is 0-2, not 1-3 as previously mentioned.

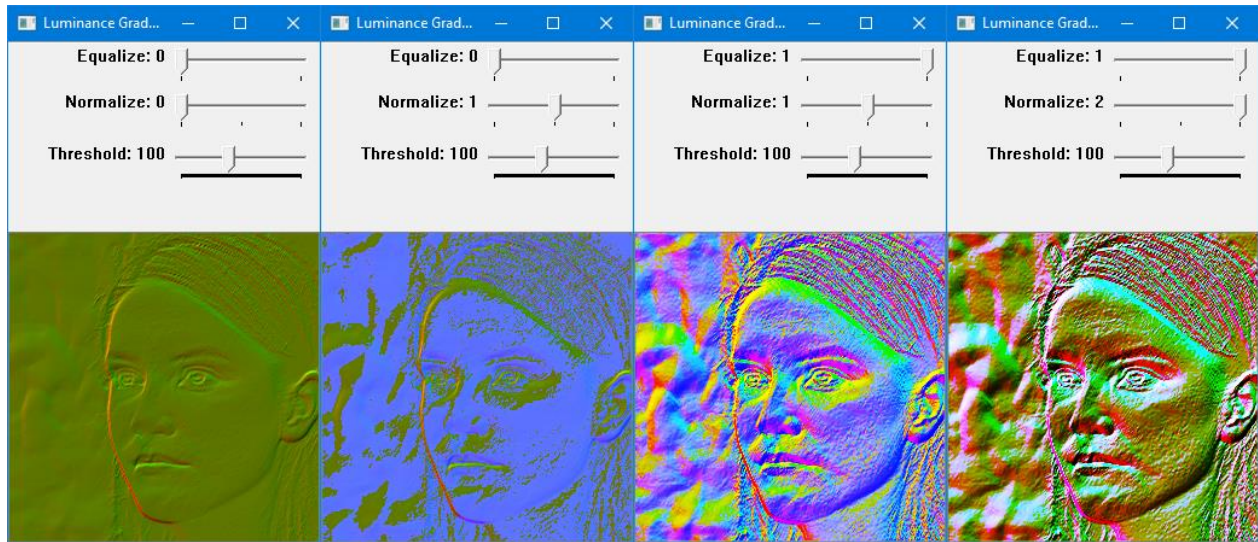
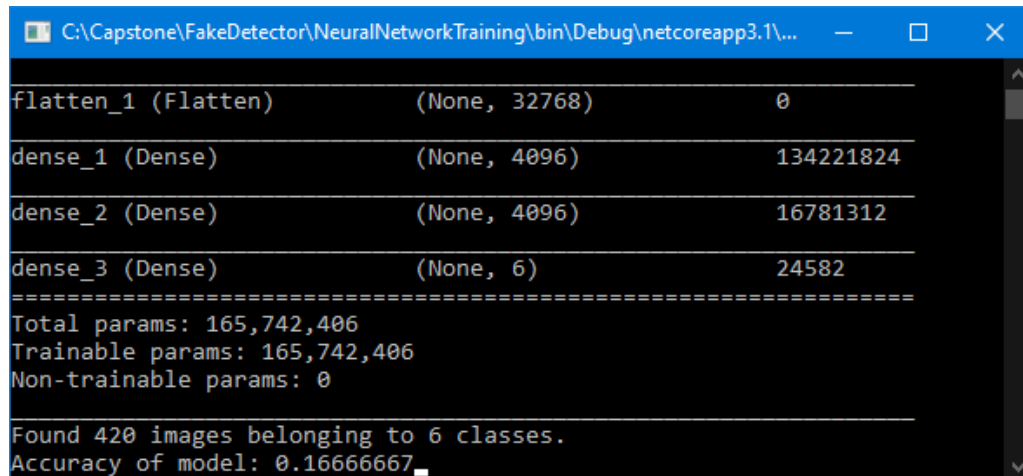


Figure 21: Manual testing of different LG parameters to determine the best fit.

Convolutional Neural Network

For the convolutional neural network, the first iteration was a single network model with all three image algorithms combined, for a total of 6 classes (fake and real for each). However, this had a very poor accuracy rate and was nearly always wrong. “However, we may take it as a general property of prediction models that they will tend to have greater difficulty differentiating object types that look *similar* and will tend to have less difficulty differentiating object types that look *dissimilar*” (Ben, 2018).

Separating each image algorithm into a separate network removes the misclassification problem. The downside, however, is that it requires more VRAM to use all the networks at runtime.



```

C:\Capstone\FakeDetector\NeuralNetworkTraining\bin\Debug\netcoreapp3.1\...
flatten_1 (Flatten)      (None, 32768)      0
dense_1 (Dense)          (None, 4096)      134221824
dense_2 (Dense)          (None, 4096)      16781312
dense_3 (Dense)          (None, 6)         24582
=====
Total params: 165,742,406
Trainable params: 165,742,406
Non-trainable params: 0
Found 420 images belonging to 6 classes.
Accuracy of model: 0.16666667_

```

Figure 22: Early iteration that was poorly trained.

Another issue encountered was a poor training rate. This was solved by lowering the *learning rate*.

```

var kernel_size = new Tuple<int, int>(3, 3);
var padding = "same";
var poolsize = new Tuple<int, int>(2, 2);
var strides = new Tuple<int, int>(2, 2);
var opt = new Adam(lr: 1e-5f);
Size = new Tuple<int, int>(height, width);

```

Figure 23: Learning rate changed from the default of 0.001 to 0.00001 (1e-5).

Due to the large dataset, the most frequent issue encountered while training the models was the time required to train. This involved experimenting with the steps per epoch, the number of epochs, and batch size. Several different batch sizes were attempted (32, 48, 64), but they increased memory usage. With the hard cap of 8GiB VRAM, only the batch size of 32 (the default) was usable. From there, the number of steps per epoch was maximized to ensure as much data as possible could be processed per epoch, and an epoch size of 10 was settled on to prevent overfitting.

GUI Testing Program

The GUI program included underwent several revisions. The first version only showed the original image and the image as processed through the image pre-processor. The next iteration was integrating the neural network into it, so the neural network classification can be run. The final major iteration was the addition of the confusion matrix visualization, to provide vital information on the performance of the network.

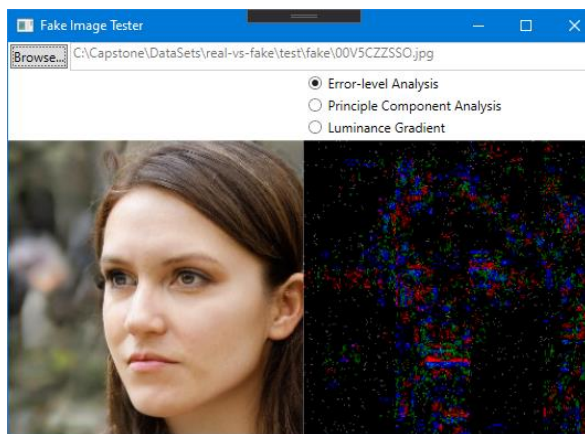


Figure 25: First iteration of GUI tool, only shows a side-by-side comparison with the image pre-processor.

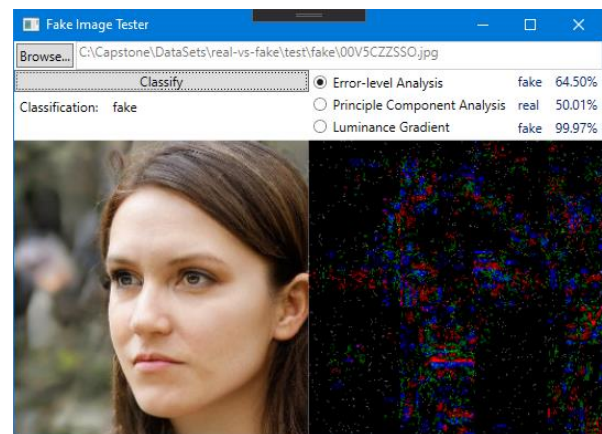


Figure 24: Second iteration, adding the neural network classification.

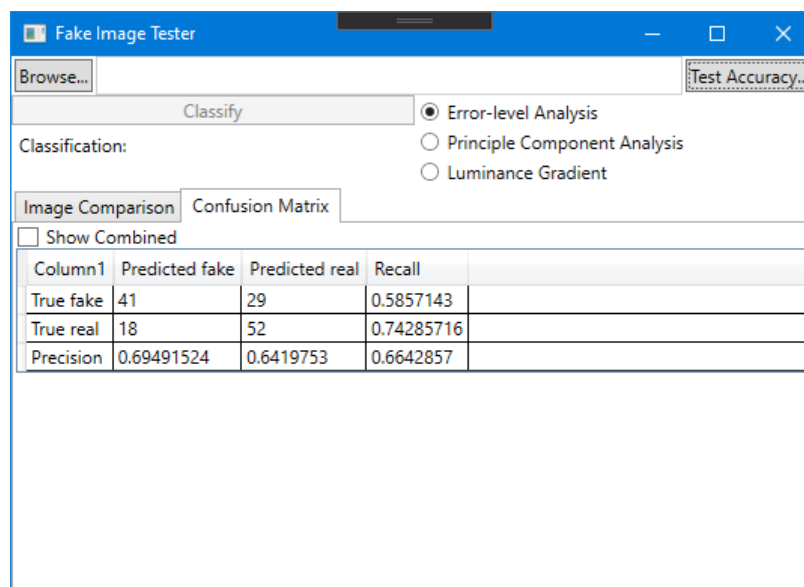


Figure 26: Final iteration, adding the confusion matrix.

Quick Start Guide

Downloading

There are several distribution methods available. Download your preferred one.

Source Code

FakeDetector-source.zip (47KiB)



Binary (Portable)

FakeDetector-portable.zip (74MiB)



Binary (self-contained, Windows x64)

FakeDetector-win-x64.zip (100MiB)



Pre-trained Models

models.7z (2.62GiB)

<https://drive.google.com/file/d/1oTfvuQEGt-w-zSE8PSOMjjStWRo6PBj0/view?usp=sharing>

Dataset

501529_939937_bundle_archive.zip (3.7GiB)

<https://www.kaggle.com/xhlulu/140k-real-and-fake-faces>

Prerequisites

Drivers

Download and install the latest drivers for your graphics card, plus additional ML libraries (cuCNN, etc.).

.NET Core (Optional)

Download and install the .NET Core 3.1 Runtime from <https://dotnet.microsoft.com/download/dotnet-core/3.1>

Python

Download and install the latest version of Python 3.7 from the Python website. Latest as of this writing is 3.7.8: <https://www.python.org/downloads/release/python-378/>

Once installed, open a command prompt and use PIP to install the required modules.

For nVidia hardware, or CPU-only processing:

```
pip install keras==2.2.4 tensorflow tensorflow-cpu
```

For AMD hardware:

```
pip install keras==2.2.4 tensorflow tensorflow-cpu plaidml-keras  
plaidml-setup
```

Follow the prompts to select the correct driver.

Installation

- 1) If the host is 64-bit Windows, you have the option to use the self-contained binary (FakeDetector-win-x64.zip). This includes the .NET Core runtime and doesn't require the installation of the separate runtime listed in the previous section.
- 2) For all other installations, ensure the .NET Core runtime is installed for your platform.
- 3) Extract the appropriate ZIP file to a location of your choice.
- 4) If you wish to use the pre-trained models, extract models.7z to the same location as the previous step. Otherwise, proceed to the next section for training models.
- 5) If you wish to use the dataset used to train the pre-trained models, extract them to a location of your choosing. By default, the application expects them in C:\Capstone\DataSets. However, the location can be overridden on the command-line during training.

Training

This step is only required if you do not choose to use the pre-trained models.

The models were trained with the following commands (one for each image algorithm):

```
.\NeuralNetworkTraining.exe --steps 1000 --epochs 10 -a ELA
.\NeuralNetworkTraining.exe --steps 1000 --epochs 10 -a LG
.\NeuralNetworkTraining.exe --steps 1000 --epochs 10 -a PCA
```

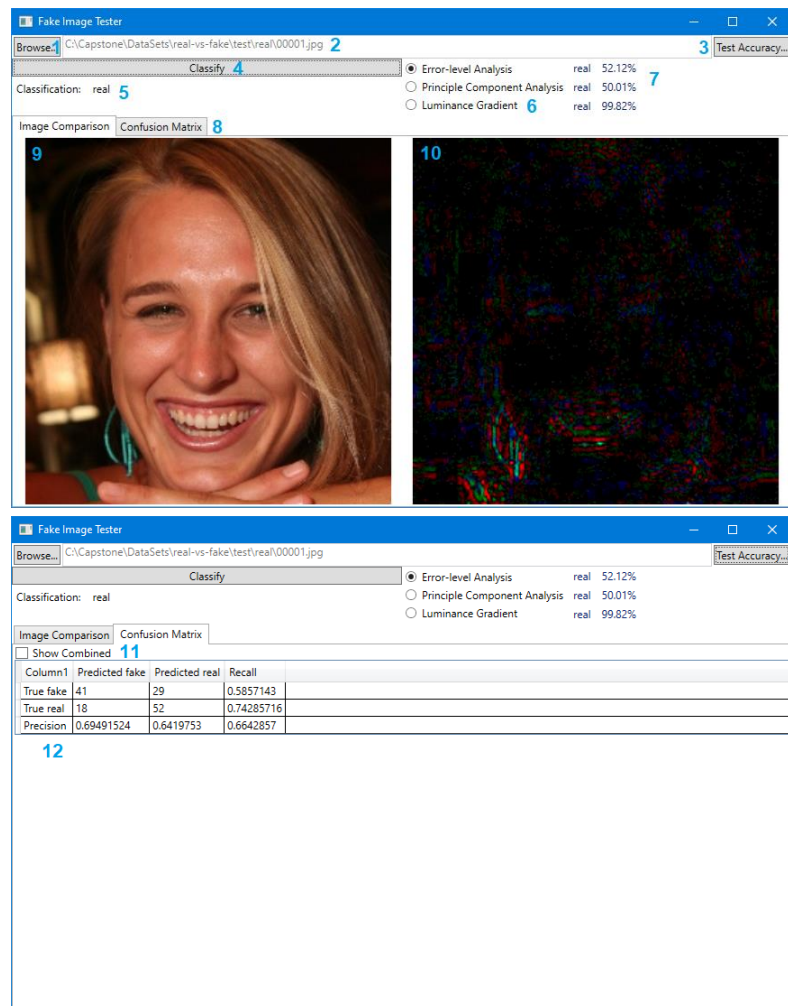
The training output can be redirected to a file if desired. Each model on an AMD RX480 with 8GiB VRAM took 22 hours.

Testing Tool

The model can be tested by running the FakeDetectorUI.exe program.

The labels are the following:

- 1) Load a single image
- 2) Loaded image filename
- 3) Select a dataset location for building a confusion matrix
- 4) Classify the single selected image
- 5) The classification of the image
- 6) Show a specific image processing algorithm
- 7) Class and probability of the image for each algorithm
- 8) Tab bar for changing views
- 9) Original image
- 10) Image after being processed through a selected algorithm
- 11) Show the confusion matrix for all algorithms combined
- 12) Confusion matrix



API

The main application interface for integrating into the existing social networking site or platform is a set of DLLs. These are Gwindalmir.ImageAnalysis.dll and Gwindalmir.NeuralNetwork.dll.

The API documentation is provided via XML files in the installation location.

A simple example to predict an individual image:

```
private (float, float, string) GetPrediction(AnalyzerAlgorithm algorithm)
{
    Mat image;

    using (var algo = AnalyzerFactory.Create(algorithm))
    {
        algo.LoadImage(@"processed_test\LG\real\57562.jpg");
        algo.AnalyzeImage();
        image = algo.GetResultImageSafe();
    }

    using (var nn = new NeuralNetworkModel("vgg16_LG.h5", 256, 156))
    using (var result = image)
    using (var pyresult = result.ToNDarray())
    using (var resized = Numpy.np.resize(pyresult, new
Numpy.Models.Shape(nn.Size.Item1, nn.Size.Item2, pyresult.shape.Dimensions[2])))
    {
        var (fake, real) = nn.Predict(resized);

        if (fake > real)
            return (fake, real, NeuralNetworkModel.Labels[0]);
        else if (fake < real)
            return (fake, real, NeuralNetworkModel.Labels[1]);
        else
            return (fake, real, "indeterminate");
    }
}
```

Further examples are available in the tests provided with the source code.

Table of Figures

Figure 1: Waterfall model (Stephens, 2015)	8
Figure 2: Computer generated face showing uneven error potential	15
Figure 3: A real human face, showing an even error potential.	15
Figure 4: PC2 Comparison of a real face (left), and a computer-generated face (right).	17
Figure 5: Luminance gradient comparison of a real face (left), and a computer-generated face (right)...	
18 Figure 6: VGG21 convolutional neural network used	18
Figure 7: The VGG network in detail, showing the weights of each layer.	20
Figure 8: ELA Confusion Matrix, 60% accuracy	22
Figure 9: PCA Confusion Matrix, 50% accuracy	22
Figure 10: LG Confusion Matrix, 82% accuracy	22
Figure 11: Combined Confusion Matrix, 64% accuracy	22
Figure 12: A properly prepared dataset	24
Figure 13: Pre-processed dataset, ready for training.	24
Figure 14: An image correctly detected as a fake	25
Figure 15: An image correctly detected as real.	26
Figure 16: ELA model's loss and accuracy. The model functions, but needs work	27
Figure 17: Chart of loss and accuracy for the PCA model. The model stagnates, is unable to progress....	27
Figure 18: Chart of loss and accuracy for the LG model. The model is trained well and is ideal.	27
Figure 19: Manual testing of different ELA parameters to determine the best fit	28
Figure 20: Manual testing of different PCA parameters. Note that the PC range here is 0-2, not 1-3 as previously mentioned.	29
Figure 21: Manual testing of different LG parameters to determine the best fit.	29
Figure 22: Early iteration that was poorly trained	30
Figure 23: Learning rate changed from the default of 0.001 to 0.00001 (1e-5)	30
Figure 25: Second iteration, adding the neural network classification.	31
Figure 24: First iteration of GUI tool, only shows a side-by-side comparison with the image preprocessor	31
Figure 26: Final iteration, adding the confusion matrix	31

References

- Ben. (2018, 12 19). *Do more object classes increase or decrease the accuracy of object detection*. Retrieved from StackExchange: <https://stats.stackexchange.com/q/349105>
- Citroën, L. (2017, March 17). *Sharing Fake News Can Hurt Your Reputation*. Retrieved from Entrepreneur: <https://www.entrepreneur.com/article/290087>
- Fowler, G. A. (2018, October 18). *I fell for Facebook fake news. Here's why millions of you did, too*. Retrieved from The Washington Post: <https://www.washingtonpost.com/technology/2018/10/18/i-fell-facebook-fake-news-heres-why-millions-you-did-too/>
- Hacker Factor. (2020, January 21). *Evaluating ELA*. Retrieved from FotoForensics: <http://www.fotoforensics.com/tutorial.php?tt=ela>
- Krawetz, N. (2008). A Picture's Worth... DC, United States of America. Retrieved from <https://www.blackhat.com/presentations/bh-dc-08/Krawetz/Whitepaper/bh-dc-08-krawetz-WP.pdf>
- NVIDIA. (2019, February 5). *StyleGAN — Official TensorFlow Implementation*. Retrieved from GitHub: <https://github.com/NVlabs/stylegan>
- Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*, (p. 14). doi:1409.1556
- Sloane, G. (2018, March 5). *Over Sharing*. Retrieved from AdAge: <https://adage.com/article/digital/over-sharing/312575>
- Stephens, R. (2015). *Beginning software engineering*. Indianapolis: Wrox.
- xhlulu. (2020). *140k Real and Fake Faces*. Retrieved from Kaggle: <https://www.kaggle.com/xhlulu/140k-real-and-fake-faces>
- Yamashita, R., Nishio, M., & Do, R. e. (2018). Convolutional neural networks: an overview and application in radiology. *Insights Imaging*. doi:10.1007/s13244-018-0639-9