LUND UNIVERSITY

SCHOOL OF ECONOMICS AND MANAGEMENT

Classifying Google reCAPTCHA v2

A study using transfer learning models and evaluating their robustness against adversarial perturbations

Authors:
Arvid Björklund
Marius Uogelė

Lund University School of Economics and Management
Master's Degree Thesis 15 HE Credits, Spring 2023
Subject: DABN01 – Master Essay I
Program: Master of Science in Data Analytics and Business Economics
Supervisor: Farrukh Javed

Abstract

This thesis seeks to examine the suitability and robustness of transfer learning models in creating an efficient reCAPTCHA v2 classifier, and further evaluates their performance against various adversarial attacks. Three models - DenseNet201, EfficientNetV2, and InceptionV3 - were trained and assessed, highlighting the applicability of transfer learning techniques in the classification of reCAPTCHA v2 challenges. Despite variation in performance metrics, all models achieved satisfactory results, with DenseNet201 outperforming others in validation and test accuracy, and InceptionV3 demonstrating shortest training time.

The paper additionally showed varying levels of robustness for several different adversarial attacks among the models, with EfficientNetV2 proving to be the most resilient. This variability, despite identical top layers, points to the underlying base architecture as a significant determinant of a model's robustness against adversarial attacks. Consequently, the study supports a comprehensive evaluation for model selection, considering not only the performance of metrics but also the underlying model’s architectural properties that may affect its robustness.

Finally, this work indicates the potential of transfer learning models for image-based CAPTCHA challenge classification and stresses the need for further research focusing on enhancing the adversarial robustness of these models. The findings of this study contribute to the expanding body of research on transfer learning, showcasing its potential applications in the domain of image-based CAPTCHA systems.

Keywords: reCAPTCHA, transfer learning, adversarial perturbations, convolutional neural network

Table of contents

Table of figures

Table of tables

1 Introduction

1.1 Background

The use of Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) has become a common practice for increasing online security by, for example, preventing automated web-scraping, credential stuffing and denial of service attacks (Zhang et al. 2022; Guerar et al. 2022). Like in other similar scenarios within the online security industry, passing through the imposed type of CAPTCHA used by the target website is sort of a cat-and-mouse game, as both CAPTCHA developers and researchers of automated bots have to continuously adapt and evolve their algorithms and strategies to yield effective results.

Among various CAPTCHA systems, two prevalent types are those that rely on text and images. Google's reCAPTCHA v1, a text-based solution, enjoyed substantial popularity. This system served a dual purpose, it not only fortified websites against automated bot attacks but also contributed to the digitization of books, as noted by Guerar et al. (2022). Succeeding this version, Google introduced reCAPTCHA v2, an image-based challenge that adopts a more sophisticated approach. Instead of merely presenting challenges to solve, this version studies the user's behavior for a more nuanced security strategy, where risk-based analysis is performed to distinguish between bots and humans (Guerar et al. 2022).

With the advancements in image recognition technologies, such as convolutional neural networks (CNN), machine learning models have become increasingly skilled at solving image classification problems. Consequently, the effectiveness of reCAPTCHA v2 is facing significant challenges as new, more efficient and accurate image classification models can be implemented in automated CAPTCHA solvers. Despite its limitations, reCAPTCHA v2 remains in widespread use across websites worldwide (Zhang et al. 2022), therefore our research will concentrate on this specific type of CAPTCHA.

One way of empowering machine learning models for image classification tasks is transfer learning, particularly when faced with a limited dataset and challenges in obtaining new data (Zhuang et al. 2021). Leveraging knowledge gained from pre-trained models can be an effective approach to kickstart generalization capabilities of a deep learning model in image recognition (Lindholm et al. 2022). Transfer learning's effectiveness has been demonstrated in various domains (Zhuang et al. 2021), suggesting that it might be a promising solution for classification of reCAPTCHA[1] challenge images. By employing pre-existing models, we could potentially improve the model's capacity to recognize and differentiate between diverse reCAPTCHA images, even if those specific image classes were not part of the original training set.

Given the extensive prevalence of reCAPTCHA on the internet, one possible defense for the CAPTCHA is adversarial perturbations (Szegedy et al. 2014). These are specific changes to the pixel values of input images, which leads the CNN to misclassify the input image. The key attribute of adversarial perturbations is that the adversarial image is not necessarily separable from the original image to the human eye. In the case of image CAPTCHAs, this means that a human user could still be able to solve the problem, while an automated solver would not. Furthermore, the use of adversarial examples to improve CAPTCHAs has been proposed by researchers in the past (Osadchy et al. 2017; Hitaj et al. 2020).

1.2 Purpose

Our research focuses on exploring how well prior knowledge can be leveraged on the reCAPTCHA classification through transfer learning. Furthermore, we aim to evaluate another dimension of model robustness, specifically the robustness against adversarial perturbations of the images.

1.3 Research questions

In this thesis, we seek answers to the following questions:

- Can transfer learning models accurately classify multi-class Google reCAPTCHA v2?
- How robust are these models against adversarial perturbations?

[1] For the rest of this paper, we will interchangeably refer to reCAPTCHA v2 as reCAPTCHA.

1.4 Delimitations

Based on the problem at hand and the proposed research methods, this paper is limited to only exploring image-based CAPTCHA, namely Google's reCAPTCHA v2. Furthermore, the paper only focuses on one part of a wholly integrated CAPTCHA solver solution i.e. the image classification. The decision to not create a complete integrated solution was mainly based on the time factor, given that we had a limited time frame for completing our research. Moreover, a fully integrated solution introduces more "moving parts" to the experiment, which were not aligned with the research goals.

Although there exists a multitude of transfer learning models, in this thesis three specific transfer models were evaluated. These were: InceptionV3, DenseNet201 and EfficientNetV2. We limited the number of models to be evaluated due to limitations in time and computational power. Similarly, there are different ways of generating adversarial perturbed images, out of which two were chosen following the same limitations mentioned before.

1.5 Contribution

The topic of CAPTCHAs and how to solve them has been part of research for many years, but the methods and specific tasks have varied. Similar to us, some published papers have utilized deep learning models to solve CAPTCHA. However, they were aiming to solve text-based CAPTCHAs rather than the image-based CAPTCHA explored in our research (Wang et al, 2019; Zi et al. 2020). Some of these models had architectural similarities to the selected transfer learning models of our research, leading us to explore the capabilities of these architectures on a different CAPTCHA challenge.

On the other hand, former research has been conducted to solve the Google reCAPTCHA v2. However, these have not utilized the same models selected for our research, but rather other machine learning models for image recognition (Mikolov et al. 2013; Hossen et al. 2021; Weng et al. 2019; Bergström & Larsson, 2022). Therefore, we contribute with an alternative strategy of tackling the reCAPTCHA v2 challenge.

Moreover, adversarial perturbations are a well-defined topic in the deep learning field. Since the discovery of adversarial examples by Szegedy et al. (2014), other researchers have built upon the findings of Szegedy et al. (2014) in order to gain a better understanding of how adversarial examples work and can be generated. Over the years, different methods for generating adversarial examples have been developed, increasing the level of sophistication behind the methods (Goodfellow, Shlens & Szegedy, 2015; Kuraking, Goodfellow & Bengio, 2017; Carlini & Wagner, 2017). Following the development of adversarial examples, researchers have sought ways of making models more robust towards these examples. For example, Shi et al. (2022) explored different solutions to make network architecture more robust towards adversarial examples.

However, these two topics of adversarial examples have not been explored in conjunction nor on the specific problem presented in this paper. Therefore, we provide an assessment of the efficiency of leveraging transfer learning on the image CAPTCHA classification problem as well as shedding light on possible defenses against automated CAPTCHA solvers utilizing CNNs.

1.6 Contents

The paper's structure is as follows. Chapter 2 contains an overview of the theory and previous research in the field. Chapter 3 explains the methodology such as data handling, model building and optimization. Chapter 4 presents the results achieved and an analysis of said results. Chapter 5 gives the conclusions drawn from our research. Chapter 6 provides a discussion of our research and suggestions for further research.

2 Theory

2.1 CAPTCHA as a website protection tool against automated access

CAPTCHAs are designed to differentiate between human and automated access by presenting users with challenges that are simple for humans to solve but difficult for bots to pass through. The word *public* in the acronym stands for the idea that the code and the data utilized by a CAPTCHA ought to be accessible to everyone (von Ahn, Blum & Langford, 2004). This is because creating a solution that can successfully pass the tests generated by the CAPTCHA algorithm should remain challenging, even for those who possess a complete understanding of its functionality (von Ahn, Blum, & Langford, 2004).

The work by von Ahn et al. (2003) presented ideas that built foundations for a new web security paradigm. Namely, they introduced text-based challenges as promising candidates for CAPTCHA problems, as they focus on so-called *hard AI problems*. These problems include distorted and obscured text that humans can recognize with high accuracy, but computers struggle to process and analyze (von Ahn et al. 2003). In their further research, the authors expand on the topic by presenting image-based and sound-based CAPTCHAs (von Ahn, Blum & Langford, 2004).

Modern image-based CAPTCHAs utilize images to present challenges to the user and can take several forms such as: selecting all images containing a specific object, solving a puzzle or identifying a specific part of an image. The images presented are usually distorted, noisy, overlapped or blended together, with hard-to-define backgrounds for AI algorithms, leveraging the human ability to interpret and understand the visualization presented with the intent of exploiting image recognition limitations of non-human users.

In 2008, reCAPTCHA was introduced by von Ahn et al. (2008) as an extension and improvement of the original CAPTCHA concept. According to the authors, this updated version served two purposes: it provided enhanced security measures for websites as well as contributed to the digitization of printed texts. This new type of CAPTCHA presented the end-user with two distorted words instead of one, namely the control word and unknown target word. The latter

were selected from scanned texts that optical character recognition (OCR) algorithms failed to decipher, thus increasing the level of difficulty for automated systems trying to bypass it (von Ahn et al. 2008). Additional security techniques included advanced character distortion techniques, added background noise and random lines or arcs to the images. The authors stated that these improvements had made reCAPTCHA more robust and secure compared to the previous CAPTCHA version, whilst also contributing to the digitization of printed material to a great extent.

Recognizing the potential benefits of this new innovative technology Google LLC, already an internet giant, acquired reCAPTCHA in 2009 (Gelles, 2009). The new acquisition within the company's portfolio was aimed to bolster the security of its own products and services, such as Gmail and Google Search, as well as to improve its efforts of digitizing books and newspapers (Gelles, 2009). This in turn helped the company to expand access to information globally (Gelles, 2009). This marked the inception of the first version (v1) of Google reCAPTCHA.

Currently, Google offers two versions of reCAPTCHA - v2 and v3, that are collectively used by 5 million websites (Google, n.d.). Launched in 2014, reCAPTCHA v2 is a classic version of Google's reCAPTCHA that uses a "I'm not a robot" checkbox and interactive selection-based image challenges to distinguish between bots and humans (Google, n.d.). An example of this reCAPTCHA challenge is presented in figure 1. It also offers an audio challenge for users who are visually impaired or have difficulty solving visual puzzles. reCAPTCHA v3 is a newer version of Google's reCAPTCHA that uses a behavioral analysis engine to determine whether a user is human or not (Google, n.d.).

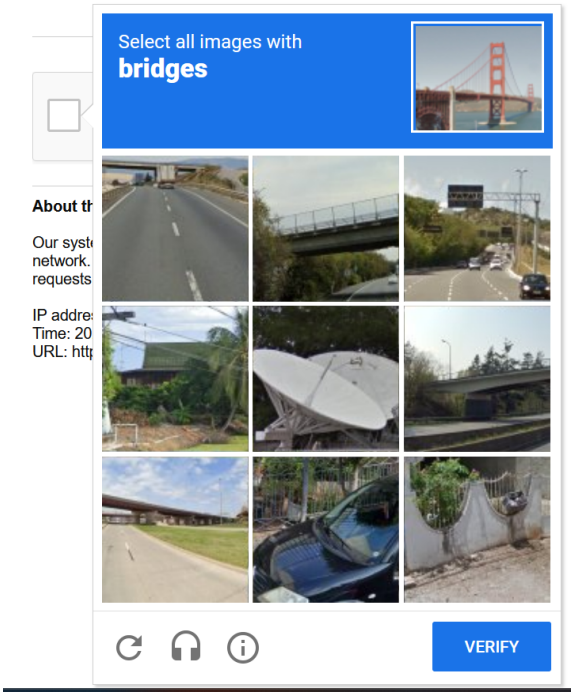

Figure 1. An example of reCAPTCHA v2 selection-based bridge image challenge

2.2 Model frameworks

Given the specific problem of classifying Google reCAPTCHA it is necessary to understand the theoretical foundations of the methods used to solve the task at hand. As a result of reCAPTCHA being image based we chose to use deep learning as our theoretical framework, as it is a common approach for image recognition tasks (LeCun, Bengio & Hinton, 2015). In this section, we provide a theoretical background of the fundamental deep learning concepts used in our research such as convolutional neural networks, activation functions and transfer learning.

2.2.1 Convolutional neural network

One of the foundational methods for image classification in deep learning are the convolutional neural networks, first proposed by LeCun (1989). CNNs have since become an effective and popular way of solving tasks when the input data has a grid structure, image recognition being one such case (LeCun, Kavukcuoglu & Farabet, 2010; Goodfellow, Courville & Bengio, 2016; Lindholm et al. 2022). The main idea of a CNN is that the network utilizes a specific mathematical operation called a convolution (Goodfellow, Courville & Bengio, 2016). As Goodfellow, Courville and Bengio (2016, p. 330) state: “Convolutional neural networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers”.

A convolution in the neural network context takes an input (such as an image pixel grid) and kernel, which moves over the whole input space producing an output usually referred to as a feature map (Goodfellow, Courville & Bengio, 2016). This procedure is visualized in figure 2.

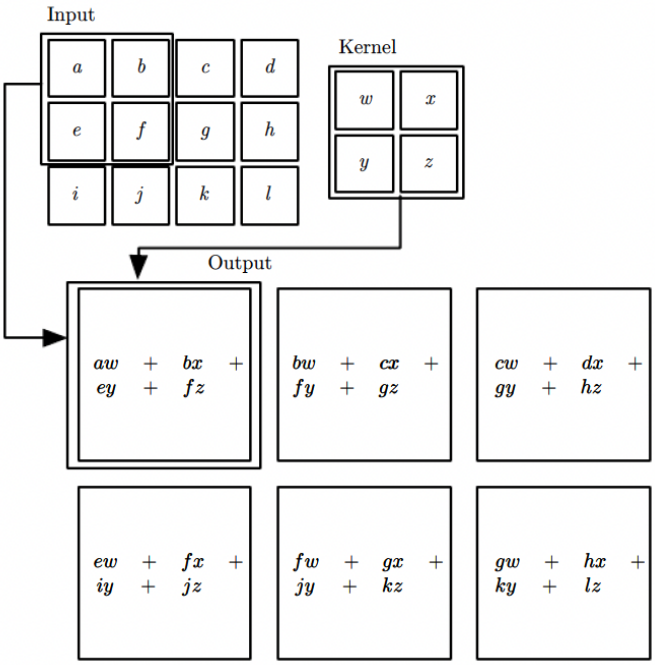

Figure 2. Visualization of convolution (Goodfellow, Courville & Bengio, 2016)

This convolution is then typically employed as the first stage in a multi-stage layer, where firstly several convolutions are performed to produce linear activations (Goodfellow, Courville & Bengio, 2016). These activations are then fed through a nonlinear activation function and in the third and final stage, a pooling function is used to change the output of the layer (Goodfellow, Courville & Bengio, 2016). The pooling function serves to change the output at a certain position depending on the nearby outputs, in the image example the pixel value could at one position be changed depending on the pixels surrounding it (Goodfellow, Courville & Bengio, 2016).

There are several different pooling functions, these include: max pooling, average pooling or L^2 norm pooling (Goodfellow, Courville & Bengio, 2016). These pooling functions simplify a rectangular neighborhood by condensing it into one value. How this is achieved differs between the methods. Max pooling returns the maximum value, average pooling the average value and L^2 pooling the L^2 norm value (Goodfellow, Courville & Bengio, 2016). Pooling is an effective operation since it makes the output approximately invariant to translations of the inputs, meaning that by changing the input by a small amount most of the pooled outputs remain unchanged (Goodfellow, Courville & Bengio, 2016).

CNNs have previously been proven valuable for image recognition tasks, such as medical image analysis (Ghaderzadeh et al. 2022) as well as traffic sign and facial recognition (LeCun, Bengio & Hinton, 2015). Given the overall success of CNNs researchers have tried to classify CAPTCHA using CNNs. Wang et al. (2019) created an original CNN called DFCR based on the DenseNet architecture. Using this model, they classified three different types of text CAPTCHA and achieved test set accuracies between 99.60-99.96% (Wang et al. 2019). Similarly, Zi et al. (2019) created a model to solve different text CAPTCHA by implementing an architecture based on a CNN in conjunction with a long short-term memory network and attention mechanism. Using said model, the authors managed to achieve accuracies between 58.7-96.7%, depending on the degree of sophistication employed by the specific text CAPTCHA (Zi et al. 2019). In summary, CNNs are a versatile tool for solving different image recognition problems and it seems prudent to explore how well they can perform on our problem.

2.2.2 Activation functions

To enable a CNN model to learn complex patterns and relationships within data, activation functions are employed to introduce non-linearity into the network. Certain activation functions transform input features into output parameters within hidden layers, effectively capturing vital features and patterns for tasks such as image classification. Meanwhile, other activation functions focus on generating the model's final predictions by converting the output of the last hidden layer into an appropriate and interpretable format, such as probabilities for classification tasks. Common form for an activation function:

$$h = g(W^T x + c),$$

where W is the weight of a linear transformation and c is the bias term (Goodfellow, Bengio & Courville, 2016).

One of the most widely used activation functions within the hidden layers is the rectified linear unit, or ReLU (Géron, 2019). This function is recommended for use within most deep learning models due to its computational efficiency, which is especially relevant in convolutional neural networks for image classification (Mastromichalakis, 2020; Boob, Dey & Lan, 2020). A mathematical formula of ReLU is given below:

$$g(z) = max(0, z),$$

where z is the input to the function (Goodfellow, Bengio & Courville, 2016).

The ReLU function takes an input value (z) and returns the maximum of 0 and the input value itself (Goodfellow, Bengio & Courville, 2016). In other words, if the input is positive, the function returns the input value, and if the input is negative or zero, the function returns 0. Graphical representation of this function is presented below.

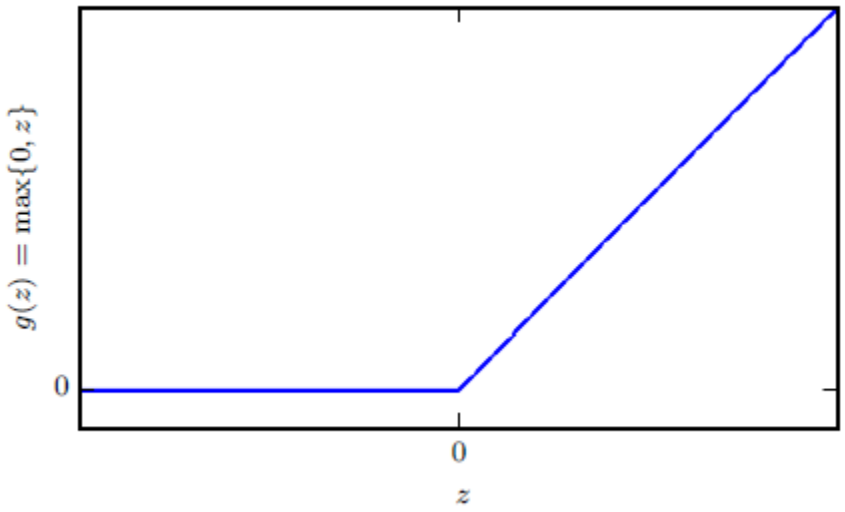

Figure 3. Graphical representation of ReLU activation function (Goodfellow, Bengio & Courville, 2016)

According to Goodfellow, Bengio, and Courville (2016), applying this activation function to a linear transformation's output leads to a nonlinear transformation. Despite this, the function remains close to being linear, as it consists of two linear segments within a piecewise linear structure. According to the authors, rectified linear units, being nearly linear, preserve several properties that make linear models both easy to optimize using gradient-based methods and highly generalizable.

Géron (2019) highlights that this activation function is somewhat quicker to compute compared to other activation functions. Additionally, gradient descent experiences fewer instances of getting stuck on plateaus, primarily due to its non-saturation for large input values. In contrast, both the logistic function and the hyperbolic tangent function saturate at 1.

Unfortunately, the ReLU activation function has its shortcomings. It encounters an issue referred to as dying ReLUs, wherein some neurons cease to function during training and only output 0 (Géron, 2019). Consequently, these neurons become incapable of learning through gradient-based methods (Goodfellow, Bengio & Courville, 2016). Hence, multiple iterations of this activation function has been suggested in the research, trying to address its drawbacks (Goodfellow, Bengio & Courville, 2016; Géron, 2019; Mastromichalakis, 2020). Despite the limitations, the ReLU activation function was used for the purposes of our research due to its simplicity and computational efficiency.

For multiclass classification, softmax activation function is typically the preferred choice (Géron, 2019). It can be perceived as an extension of the sigmoid function, which represents a probability distribution over a binary variable. The Softmax function, however, is adept at reflecting a probability distribution over multiple classes. Consequently, it is frequently employed as the final layer in a classifier, indicating the probability distribution across a number of distinct classes (Goodfellow, Bengio & Courville, 2016). Formally, the softmax function is given by:

$$softmax(z)_i = \frac{exp(z_i)}{\sum j\ exp(z_j)},$$

where z is the input vector containing n elements, an exponent of each input vector value z_i in the numerator and a normalization term in the denominator, ensuring that all the output values add up to 1 (Goodfellow, Bengio & Courville, 2016). A visual representation of the softmax function is shown in figure 4.

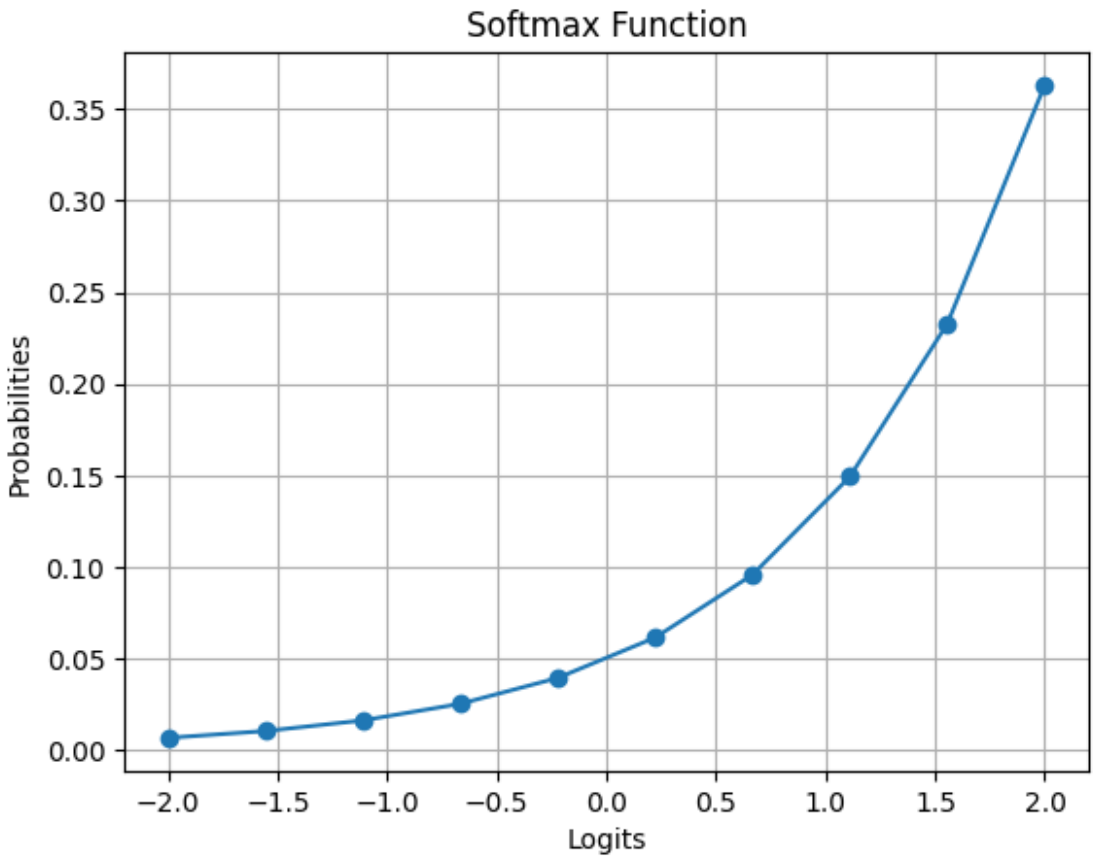

Figure 4. Graphical representation of the softmax activation function

The softmax function graph in figure 4 provides a visual representation of how the function translates a vector of inputs, which can be both positive and negative, into corresponding probabilities. Input logits are displayed on the x-axis, whereas the output probabilities are given on the y-axis. The plot highlights an important aspect of the softmax function: it favors higher-value elements of the input vector and suppresses the lower-value ones, effectively pushing the highest probability towards 1 (Géron, 2019). Such a distribution of probabilities is particularly beneficial in multi-class classification tasks, where the aim is to assign the highest probability to the correct class. Hence, we chose this function as the activation function of the output layer in our models.

2.2.3 Transfer learning

Transfer learning is a machine learning method with a use case for situations when the amount of data and or computing power is limited. In transfer learning, one can use knowledge from a model trained on a task using a specific dataset and apply the same model to solve another, at least slightly related, task (Lindholm et al. 2022; Goodfellow, Courville & Bengio, 2018).

In practice, this amounts to using a pre-trained model, usually trained on a very large data set, as a "base model" on top of which additional layers can be added for classification. Usually, the pre-trained weights of the base model are freezed, meaning that they are no longer updated in the training process. This in turn leads to the base model retaining the knowledge gained during pre-training. The additional layers added on top can be chosen freely and contain the weights which will be trained on the data. After the model has been trained sufficiently, the whole or parts of the base model can be unfrozen and trained, in order to tune the pre-trained model to the specific problem at hand.

Many models used for transfer learning are readily available through Python packages such as Keras[2]. These models have usually been created by researchers in order to perform well on large scale image data sets such as ImageNet. The trained models can then be downloaded and used as described in the workflow above.

Transfer learning has proven to be successful in previous research, since knowledge gained in previous image recognition problems can be leveraged on new use cases. Some of these use cases involve medical image recognition (Ghaderzadeh et al. 2022) and in their survey of transfer learning, Zhuang et al. (2021) noted several studies applying transfer learning in the medical, bioinformatics, transportation and communication fields.

[2] Chollet, F. & others (2015). Keras. Available online: https://keras.io/

2.3 Adversarial perturbations

The idea of adversarial perturbations was first mentioned by Szegedy et al. (2014, p. 2) who described them as: "imperceptible non-random perturbation to a test image". The authors then coined the term "adversarial examples", meaning images which have been altered by perturbations which maximize the prediction error of a model. This might seem contradictory, as state-of-the-art deep neural networks are usually trained to generalize well on image recognition, i.e., they should be robust to small changes to the input image (Szegedy et al. 2014). However, Szegedy et al. (2014) found that the so-called "smoothness assumption", meaning that small changes to an image do not change the underlying class, of the kernels utilized in kernel methods does not hold. So by using an optimization procedure, the authors were able to find the adversarial examples with small perturbations which caused the model to misclassify the image.

Following this discovery, the concept of adversarial examples, how to make them more effective and how to generate them quickly became a research topic of its own. Goodfellow, Shlens and Szegedy (2015) suggested a method of generating adversarial examples called the Fast Gradient Sign Method (FGSM). This method takes advantage of the linear nature of neural networks, such as the ReLU activation function which was designed to behave linearly in order to make model optimization easier (Goodfellow, Shlens & Szegedy, 2015). The FGSM can mathematically be described as:

$$\eta = \epsilon * sign(\nabla_x J(\theta, x, y)),$$

where θ are the model parameters, y the targets of x (in image recognition the labels of images) and J(θ,x,y) the cost function of the neural network (Goodfellow, Shlens & Szegedy, 2015). Cost function in this context simply refers to the average of the loss function over the entire data set. Some common loss functions for classification include cross entropy or logistic loss (Lindholm et al. 2022). In the context of multiclass classification, a common loss function is the categorical cross entropy loss also known as softmax loss (Wen et al. 2016).

This mathematical model is then applied to the pixel values of the input image with ϵ controlling the magnitude of the perturbation, i.e. $x + \epsilon * sign(\nabla_x J(\theta, x, y))$. Furthermore, the authors found that their suggested method reliably made several models misclassify the perturbed inputs.

Building upon the FGSM, Kurakin, Goodfellow and Bengio (2017) extended what they called "the fast method" (FGSM) by implementing it iteratively. This amounts to applying the FGSM several times using a small step size and then clipping pixel values of intermediate results after every step (Kurakin, Goodfellow & Bengio, 2017). By doing so, the authors ensure that the pixel values do not stray to far from the original image keeping it in the "ϵ-neighborhood".

Mathematically this basic iterative method[3] can be described as:

$$X_0^{adv} = X,\ X_{N+1}^{adv} = Clip_{X,\epsilon}\left\{X_N^{adv} + \alpha\, sign(\nabla_x J(X_N^{adv}, y_{true}))\right\},$$

where X is the input image pixel values and α corresponds to ϵ in the FGSM (Kurakin, Goodfellow & Bengio, 2017).

As can be seen in the equation above, the method amounts to applying the FGSM iteratively for a predefined number of iterations. Kuraking, Goodfellow and Bengio (2017) found that the adversarial images generated by the "fast" method were more robust than ones generated by the "iterative methods" in both tried cases. The experiments included both classifying photos of adversarial images (physical transformation) and images which had been transformed artificially (change of contrast and brightness, Gaussian blur, Gaussian noise, and JPEG encoding). The authors concluded that adversarial examples generated by the fast methods were more robust to transformations compared to the iterative method and pondered that it was a result of iterative methods producing more subtle perturbations than their fast counterparts.

[3] In this paper, the method will be referred to as projected gradient descent (PGD)

2.4 Attempts to break reCAPTCHA v2

Since the deployment of reCAPTCHA v2 in 2014, there have been several attempts to come up with effective techniques to bypass this specific image-based CAPTCHA and demonstrate the potential weaknesses of the system.

A group of researchers at Columbia university designed a complete, deep learning based, reCAPTCHA solver that achieved 70.78% accuracy on the image reCAPTCHA challenges (Sivakorn, Polakis & Keromytis, 2016). The system processes images from the challenge through several modules, which leverage Google Reverse Image Search (GRIS), image annotation, tag classifier built on Word2Vec model (Mikolov et al. 2013) and best guess results to compare images against a hint provided by the reCAPTCHA challenge. Afterwards, these results are processed and merged, weighing the information based on confidence. Additionally, Sivakorn, Polakis & Keromytis (2016) pointed out that for the automatic solution to be useful, the task has to be completed in under 55 seconds or else the whole process would have to be repeated again, highlighting speed as an additional feature for the automated solver to be effective in practice.

In a different paper, another fully automated object detection-based system was proposed that breaks updated and even more advanced Google reCAPTCHA v2 challenges with a success rate of 83.25% in, on average, 19.93 seconds (Hossen et al. 2021). The presented automated CAPTCHA breaker system was made up of two parts - a browser automation module and a solver module. The first unit of the system was used to automate multiple tasks within the browser, such as locating the checkbox of the reCAPTCHA, starting the challenge, collecting the images as well as confirming the solution after it is solved. The second module uses the YOLOv3 (Redmon & Farhadi, 2018) algorithm as the base object detector to identify objects in the image of the challenge, providing class name, confidence score and bounding box coordinates (Hossen et al. 2021).

Weng et al. (2019) have conducted an extensive research, evaluating popular CAPTCHA schemes, including Google's reCAPTCHA v2. Through their analysis, the authors reveal that these security systems can be successfully bypassed using advanced vision techniques, namely image classification by CNN and objection detection models. The success rate achieved by the researchers on reCAPTCHA v2 was 79% and 5 seconds on average to solve the challenge.

A recent paper written by Bergström & Larsson (2022) presented another solver to break two different reCAPTCHA v2 challenge types. Their proposed system was trained on the YOLOv5 (with pre-trained weights on the COCO dataset) algorithm and reached a success rate of 23.8% and 78.5% for 3x3 fading grid and 4x4 grid challenges, respectively. The overall mean accuracy of the tests was 47.2%. Although multiple reasons are presented as to why the results for different types of challenges differ to such a great extent, the authors highlight that the aim of this study was not to develop the most effective solver for Google's reCAPTCHA v2. Furthermore, the authors have also concluded that, in general, the car class demonstrated greater robustness to the solver as compared to traffic lights. However, the classes: fire hydrants, buses, bicycles and motorcycles, showed relatively similar results of success rates.

In conclusion, recent research has successfully exposed the vulnerabilities of Google's reCAPTCHA v2 to automated attacks, particularly in the face of advancements in image-recognition algorithms. These findings underscore the need for continuous improvements in security measures and the development of more robust mechanisms to stay ahead of increasingly sophisticated attempts to bypass such systems. As the field of artificial intelligence and machine learning continues to evolve, it is important that future research is directed towards identifying and addressing potential weaknesses in order to maintain the integrity of online security systems like reCAPTCHA.

3 Methodology

3.1 Data

We used a publicly available reCAPTCHA dataset from a Github repository[4]. It contains 11,775 individual images, which have been segmented from reCAPTCHA v2 challenges. The dataset was originally divided into 12 classes, where each class represents a different reCAPTCHA image object, such as a bicycle or a car. A glance into the images of this dataset is shown in figure 5.

Figure 5. Sample images from the reCAPTCHA dataset

Figure 5 showcases a selection of five images, each assigned to a distinct class. Reading from left to right, these classes are *Bicycle, Bridge, Bus, Ca*r, and *Crosswalk*. The images are not limited to their class-defined subjects, thereby adding complexity to their classification. For instance, the *Bicycle* and *Bridge* classes incorporate cars within their frames, and the *Crosswalk* class features additional elements like traffic lights and cars. This collocation of objects, similar to collinearity of feature variables, poses a substantial challenge. It transforms a seemingly simple task of classification into a more sophisticated exercise of object detection for the classifier.

3.2 Preprocessing of data

The first part of preprocessing was the removal of certain classes. Since some of the classes had less than 1% of the overall data size (less than 117 images), we removed them. These classes were: *Chimney, Motorcycle* and *Mountain*. Additionally, the class Other included a mixture of

[4] Dataset available online: https://github.com/folfcoder/recaptcha-dataset/tree/main/Large

different images without labels, meaning it was not useful for classification and hence removed. After these removals the remaining classes were *Car, Bridge, Bicycle, Traffic light, Palm*, *Hydrant*, *Bus* and *Crosswalk*. An outlook of the class distribution of the dataset after this step is presented in figure 6.

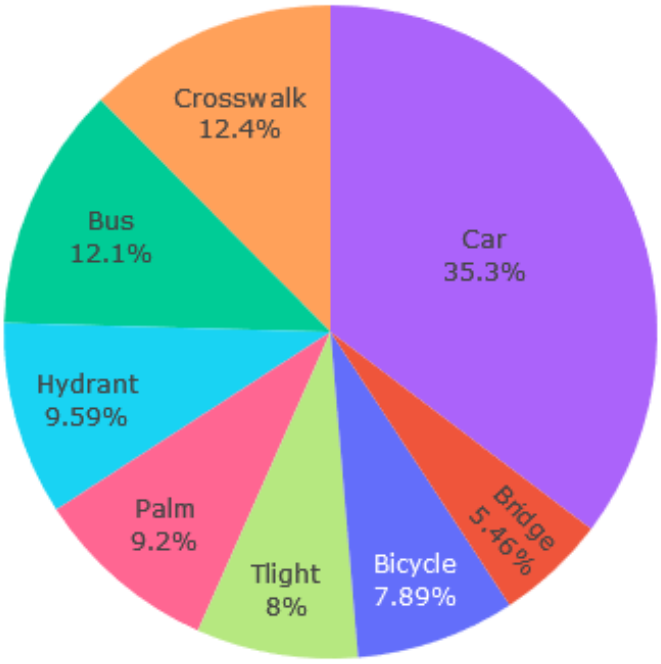

Figure 6. Distribution of the classes after removing unsuitable ones

The imbalance in the dataset is immediately evident from figure 6. As a first step of dealing with this, we divide the dataset into training, validation, and test sets. Splitting it in this particular way serves several purposes:

1. Apply rebalancing measures only on training data.
2. Evaluate overfitting during training with validation data.
3. Evaluate the performance of the trained models with test data.
4. Keep imbalanced ratios of the original data within validation and test data sets to reflect the distribution of real-life data.

In order to balance out the training data, we employed traditional data augmentation techniques to create supplementary images for the *Bridge*, *Bicycle*, *Traffic Light*, *Palm*, and *Hydrant* classes. We established the average between *Crosswalk* and *Bus* class sizes, resulting in 1000 images as

our benchmark and generated the necessary additional images to balance the underrepresented classes. This was accomplished by using the ImageDataGenerator function from the TensorFlow[5]-Keras framework, which incorporates a variety of random transformations. These include rotation, zoom, width and height shifts, horizontal flipping, shear intensity adjustments, and the filling of missing pixels with the closest available pixel values.

Furthermore, we downsized the *Car* class, as it was overrepresented, to mitigate any potential negative effects on the models' training. This was done by randomly removing observations in the *Car* class until the desired number of observations was attained. Finally, the training dataset was made up of around 8000 images following the distribution in figure 7. Validation and test data contained 1011 and 1019 images respectively, following the distribution shown in figure 6.

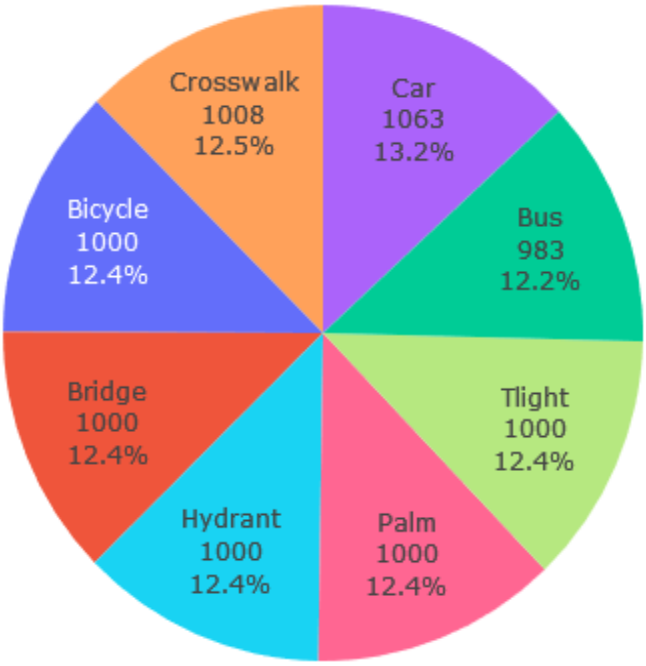

Figure 7. Class distribution of the training dataset after pre-processing

Research has demonstrated the efficacy of several sophisticated data augmentation techniques, such as SMILES enumeration (Bjerrum, 2017), GANs (Perez & Wang, 2017; Mikołajczyk & Grochowski, 2018), and quintuplet sampling with triple-header loss (Huang et al. 2016). However, classical image transformation techniques have been used as one of the most popular

[5] Abadi, M. & others (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

approaches for balancing dataset size and enhancing overall robustness, particularly in medical image classification models (Kwasigroch, Mikołajczyk & Grochowski., 2017; Ghaderzadeh et al. 2022). By generating new and diverse instances for the underrepresented class, these techniques facilitate the equalization of class distribution within the dataset. This not only aids in preventing overfitting to the majority class but also bolsters the model's generalization capabilities.

In addition to the data augmentation, each model required certain pre-processing of the data in order to receive the correct inputs. This included rescaling the pixel values of every image to the range -1 to 1 for the InceptionV3 and EfficientNetV2 models. For the DenseNet model, pixel values were scaled to have values between 0 and 1, and each channel was normalized to the ImageNet database. These pre-processing procedures were performed by either adding a Rescaling layer to the model, or by passing the inputs through specific pre-processing layers provided in the TensorFlow package.

3.3 Transfer Learning

3.3.1 InceptionV3

The first transfer learning model chosen was the InceptionV3 model (Szegedy et al. 2015). This model is an iteration upon an earlier version of the Inception model, which is a continuation of the work done on one of the most famous CNNs, AlexNet (Szegedy et al. 2015). When creating the model, the authors set up four guiding principles which were aimed towards achieving higher computational efficiency. This included avoiding representational bottlenecks, balancing width and depth of the network and reducing dimensionality of input representations (Szegedy et al. 2015).

Szegedy et al. (2015) found that substantial gains in computational efficiency could be made by factoring large convolutions into smaller ones. The authors mention that convolutions with large filters (kernels) such as 5x5 or 7x7 are disproportionately expensive computationally, which can be mitigated by replacing these larger filters with more layers of smaller size. For example, the output of a 5x5 filter can be represented by two layers of 3x3 convolution which results in a reduced parameter count (Szegedy et al. 2015). Furthermore, the authors found that using asymmetric filters is even more efficient than reducing symmetric filter size, for example a 3x3

convolution can be replaced by a 3x1 convolution followed by a 1x3 convolution, resulting in a solution that is 33% cheaper keeping input and output filters equal. Szegedy et al. (2015) found that these asymmetric convolutions work well on medium grid sizes, meaning that this factorization performs well in the later layers of the network.

Additionally, the InceptionV3 model uses auxiliary classifiers which were originally proposed to combat the vanishing gradient problem in deep networks and to promote more stable learning and better convergence (Szegedy et al. 2015). On the contrary, Szegedy et al. (2015) found that auxiliary classifiers did not result in improved convergence in the early stages of training; however, near the end of training the network with auxiliary classifiers started to outperform the network without the branch. Furthermore, the authors found that removing the lower branch auxiliary classifier did not negatively affect the network, leading them to believe that the auxiliary classifier performs like a regularizer.

As mentioned before, CNNs commonly use some form of pooling operation in order to decrease the size of feature maps after convolutions. However, Szegedy et al. (2015) suggested a new type of grid size reduction layer which does not create any representational bottleneck while performing the necessary grid size reduction on the feature maps. The solution was to use two parallel stride 2 blocks, one pooling layer (average or maximum pooling) and the activation (Szegedy et al. 2015). This so-called Inception module simultaneously reduces the grid size while also expanding the filter banks, resulting in a computationally cheaper operation which avoids representational bottlenecks (Szegedy et al. 2015).

The full InceptionV3 architecture is based on the different architectural ideas described in this section and is shown in figure 8. Importantly, note that for our purposes, the so-called “final part” in figure 8 is not used, since we are building our own classifier on top of the InceptionV3, meaning that the InceptionV3 base model is used as a feature extractor or object detector.

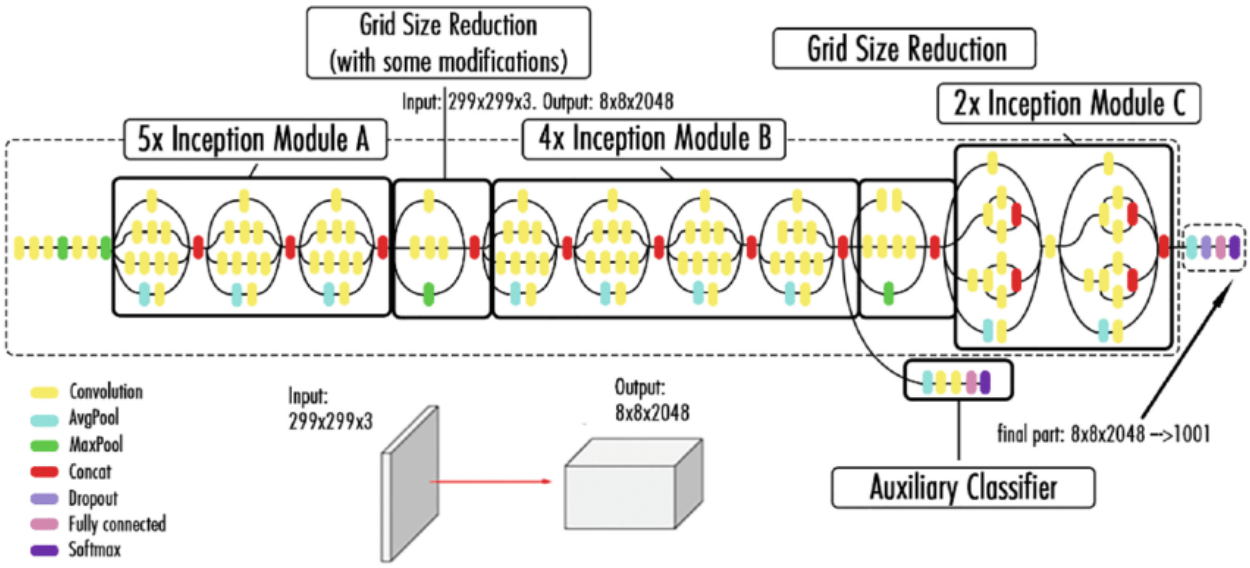

Figure 8. Architecture of InceptionV3 model (Zorgui et al. 2020)

3.3.2 DenseNet201

The second model used in our research is the DenseNet model proposed by Huang et al. (2017). Huang et al. (2017) designed the model in order to combat the vanishing gradient problem, using a key idea suggested by previous researchers. This idea involves creating short paths from early to later layers, retaining information from previous layers further on in the architecture (Huang et al. 2017). Huang et al. (2017) proposed an architecture where maximum information flow between layers is retained, by connecting all layers directly with each other, shown in figure 9. The authors did so by sending the feature map of one layer as the input to subsequent layers and coined their approach Dense Convolutional Network (DenseNet).

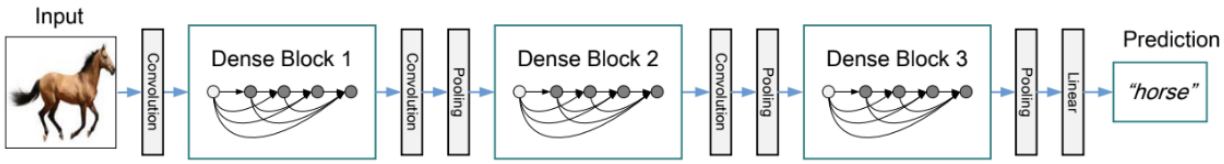

Figure 9. Architectural idea of DenseNet model (Huang et al. 2017)

In practice, this means that the ℓth layer of the network has ℓ inputs (feature maps of all preceding convolutional blocks). The ℓth layers feature maps are then passed on to all L-ℓ

subsequent layers leading to $\frac{L(L+l)}{2}$ connections in a L-layer network (Huang et al. 2017). As Huang et al. (2017) state, this actually requires fewer parameters than standard neural networks, since the DenseNet layers are kept narrow, only contributing a small amount of feature maps to the knowledge of the entire network and keeping other feature maps unchanged. The authors mean that this leads to a network which distinguishes between preserved information and information added to the network. Huang et al. (2017) also note that this results in parameter efficiency as well as an improved information and gradient flow in the network, making them easier to train. Moreover, the authors mention the regularizing effect that the dense connections introduce.

Mathematically, this way of connecting the layers can be described as:

$$x_l = H_l\left(\left[x_o, x_1, \ldots, x_{l-1}\right]\right),$$

where $\left[x_o, x_1, \ldots, x_{l-1}\right]$ is the concatenation of feature maps produced in layers 0 to l-1. H_l is in this case a composite function comprising three operations: batch normalization, rectified linear unit and a 3x3 convolution (Huang et al. 2017).

Huang et al. (2017) noted that the DenseNet model is similar to the ResNet family of models, another type of neural network, which employs the idea of connected layers through summation instead of concatenation of previous layers' feature maps. The authors found that this seemingly small difference led to very different behaviors. They noted a few advantages achieved through the DenseNet architecture such as: model compactness, implicit deep supervision, stochastic versus deterministic connection and feature reuse.

In our research, we employed the DenseNet201 variation of the model. There are a multitude of different versions of the model architecture with slight alterations to what layers are included, such as bottleneck- and compression layers for DenseNet-BC variants (Huang et al. 2017). The DenseNet201 differs from other variants as it includes more parameters in the network, however it is not the variant with the most parameters. DenseNet201 was chosen as a base model since the number of parameters included in the network was similar to that of the other base models. We believed this would increase our ability to fairly compare the different models.

3.3.3 EfficientNetV2

The third and last model we chose to evaluate was EfficientNetV2 created by Tan and Le (2021). This model was a continuation of the authors previous EfficientNet model, in order to improve training speed and parameter efficiency (Tan & Le, 2021). The authors evaluated their old EfficientNet model and found that gains could be made by employing neural architecture search (NAS) and scaling, which lead to the EfficientNetV2. EfficientNetV2's training was also effectivized by introducing a method of progressive learning, which amounts to scaling up image size as it moves deeper in the network (Tan & Le, 2021).

When reviewing the older model, Tan and Le (2021, p. 3) found three critical results: "training with very large image sizes is slow … depthwise convolutions are slow in early layers but effective in later stages … equally scaling up every stage is sub-optimal". This led the researchers to use smaller image sizes during training and larger sizes for inference, replacing a network building block called MBConv with a more recent fused-MBConv and a non-uniform scaling strategy for adding layers to the network and scaling of images (Tan & Le, 2021).

These changes were subsequently used for setting up the search space which involved design choices such as convolutional operation types, number of layers, kernel sizes and expansion ratio (Tan & Le, 2021). Furthermore, other redundant design choices were removed from the search space. Tan and Le (2021) then employed their training-aware NAS and sampled up to 1000 models based on the EfficientNet backbone which then gave them their baseline searched model called EfficientNetV2-S. A summary of the EfficientNetV2-S architecture can be found in table 1. Finally, Tan and Le (2021) scaled this model using certain restrictions in order to increase model accuracy.

Stage	Operator	Stride	Number of channels	Number of layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Table 1. Architecture of EfficientNetV2-S

With this architecture, Tan and Le (2021) were able to outperform many state-of-the art image recognition models using fewer parameters and decreasing training time. Furthermore, the authors conducted experiments on transfer learning data sets where they also managed to outperform previous CNNs and Vision Transformers, suggesting that their models generalize well for other applications (Tan & Le, 2021).

To conclude, we chose to use EfficientNetV2 as it is a more recently proposed model compared to the other models, meaning it is at the forefront of CNN architectures. Given its results and efficiency we believe that it is a good candidate for transfer learning evaluation.

3.3.4 Comparison

Comparing the three different model architectures, one can easily note that they are all extensions of the idea of a CNN, trying to improve upon prior iterations and making them more computationally efficient. Both the Inception, EfficientNet and DenseNet architectures try to effectivize computations by reducing the number of parameters in the network. This is a common idea in the field of neural networks as it is seen as one crucial way of improving computational efficiency. However, they differ in how they go about this problem. The Inception model fractions the convolutions performed in the layers while the DenseNet uses connectivity between the layers. On the other hand, EfficientNetV2 employs a new set of convolutional building blocks and is optimized by a new search method, meaning that it was built using a more exploratory way of finding efficient networks. However, only the DenseNet introduces an idea of preserving prior knowledge gained by the network, an idea not shared in the Inception or EfficientNet philosophy.

3.3.5 Classifying model

The three above mentioned transfer learning models were used as a base, on top of which a classifying model was added. In order to make comparisons between the different base models, the classifying model was kept the same in all three instances. The top model was kept fairly simple with the following set up: one global pooling layer, one fully connected dense layer, a drop-out layer, a fully connected dense layer and then one final dense layer (output layer). The two first dense layers had 416 and 208 hidden units respectively and both utilized the ReLU activation function. The exact amount of hidden units for the two first dense layers was decided using hyperparameter tuning. The final dense layer had 8 hidden units corresponding to the number of classes and used the softmax activation function. The class with the highest probability from the softmax output is chosen as the prediction of the image. A summary of the architecture can be found in figure 10.

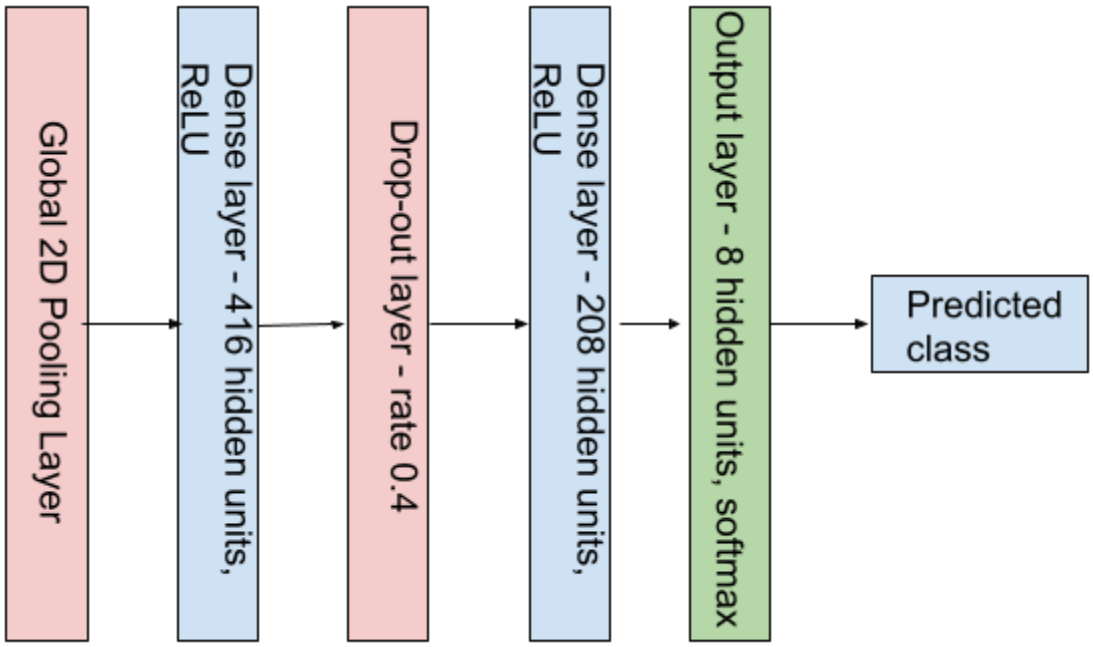

Figure 10. Schematic of top classifying model

3.4 Hyperparameter tuning

For the hyperparameter tuning of the models, we have opted to employ the random search method (Bergstra & Bengio, 2012), as it can offer distinct advantages, particularly when faced with limited computing power and time constraints.

Goodfellow, Bengio and Courville (2016) argue that, unlike other optimization algorithms such as Grid Search or Bayesian Optimization, random search offers a more computationally efficient approach by sampling hyperparameters from a predefined distribution and is generally easier to implement and understand. Also, Bergstra & Bengio (2012) showed that random search optimization is even more efficient than grid search algorithms for hyperparameter optimization.
In our research, we will employ the Keras Tuner library, a component of the Keras framework, to conduct Random Search hyperparameter optimization. This technique involves selecting a specific number of hyperparameter combinations from a predetermined search space and evaluating them across a fixed number of trials.

The random search was employed to determine the amounts of hidden units in the two first dense layers of the top classifying model as well as determining the starting value of the learning rate. These searches were done separately, as the starting learning rate had the highest impact on the training process. Thus, by searching for the amount of hidden units with a fixed learning rate, we could determine the optimal setup of the architecture.

The starting point of the learning rate was then applied to our cosine annealing scheduler. This type of learning rate scheduler decays the learning rate as the model updates, following a cosine curve (Loshchilov & Hutter, 2017). After the learning rate has reached a certain value or a predefined amount of epochs have passed, the learning rate resets to the initial value. This combats the issue of selecting the learning rate, since the annealing procedure and restarts provides a range of learning rates which the model uses in training (Loshchilov & Hutter, 2017). The learning rate scheduler was then provided to the Adam optimizer used in all model training. All model training was done for 100 epochs with early stopping. The tolerance for the early stopping was set to 10 epochs of no decrease in validation loss.

A summary of all hyperparameter values can be found in table 2.

Hidden units first layer	416
Hidden units second layer	208
Drop-out rate	0.4
Amount of epochs	100
Initial learning rate	$5*10^{-5}$
Decay steps	1000

Table 2. Chosen hyperparameter values

3.5 Model evaluation

In order to evaluate the performance of the three respective models some key metrics were gathered. Firstly, the model accuracy on the three subsets of data: training, validation and testing. This is an overall accuracy measure, providing the percentage of overall correctly predicted inputs. We decided to use the accuracy metric, as it is a common metric used for evaluating deep learning models (Wang et al. 2019; Ghaderzadeh et al. 2022; Zi et al. 2020). In order to gain a deeper understanding of the class specific accuracies, we also calculated the confusion matrices of each individual model. The confusion matrix provides a table of all predicted labels compared with the true labels, giving insight into what classification was made for a wrong prediction.

Secondly, we used the loss values of the three subsets of data. The loss value corresponds to the value of the categorical cross entropy loss on the specific subset of data evaluated. As mentioned before, the categorical cross entropy (softmax) loss is a common loss function used when conducting multi-class classification, as it behaves well in conjunction with the softmax activation function in the output layer (Bishop, 2006).

This loss function amounts to calculating the cross entropy between the probabilities calculated using the softmax function and the true probabilities, mathematically described as:

$$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(x_n, w) \quad ,$$

where t_k corresponds to the one-hot encoded target variables (image class labels) and y_k to the network outputs for an input x_n and weights w (Bishop, 2006).

By evaluating the value of the loss, one can see if the model properly saturates in training. Moreover, monitoring the validation loss after each epoch provides a check for overfitting.

3.6 Cross validation

A common technique in machine learning to evaluate model results is cross-validation (Rogers & Girolami, 2016; Lindholm et al. 2022). In our research, we have employed k-fold cross validation, which involves partitioning the dataset into k equally sized subsets (folds) and iteratively training the model on k-1 folds while validating it on the remaining fold. This process is repeated k times, with each fold serving as the validation set once. The model's performance is then averaged across all iterations to provide a more stable and robust evaluation of its generalization ability (Rogers & Girolami, 2016)

We used the guidance provided by Lindholm et al. (2022, p. 70) as a benchmark for selecting the parameter k in our k-fold cross validation approach. They suggest that "If the training is computationally demanding (as for deep neural networks, for instance), it becomes a rather cumbersome procedure, and a choice like k = 10 might be more practically feasible." Based on this recommendation, we chose k = 5 as a more efficient and faster alternative to k=10, allowing for a more practical application given computational limitations.

To ensure that our validation and test splits are not affected by class imbalance, we have chosen to use a stratified k-fold cross validation method, a variant of the traditional k-fold cross validation. In cases where some classes have fewer samples than others, the standard k-fold cross validation can lead to biased results as some folds may have an insufficient number of samples from a particular class. By contrast, stratified k-fold cross validation guarantees that each fold has a representative class distribution (Géron, 2019), making it an effective technique for dealing with imbalanced datasets (Prusty, Patnaik & Dash, 2022).

However, stratified k-fold cross validation makes an assumption about the distribution of real-life data. By retaining the original imbalance in the validation and test sets, we implicitly assume that the real-world data also follows this imbalanced distribution. This assumption could be considered fair, given that if all classes were equally likely to appear in a reCAPTCHA challenge during image collection, the dataset should be balanced from the outset.

The choice of stratified k-fold cross-validation represents a compromise between developing an effective classifier and making assumptions about real-world data distribution. In our specific case, this trade-off is acceptable. We opted for stratified k-fold cross validation over regular cross validation to address the constraints arising from the small and imbalanced validation and test datasets. This validation technique allowed for more efficient data utilization, decreased performance estimate variance, and assisted in detecting and mitigating potential overfitting, ultimately enhancing our models' robustness.

The models were trained until validation loss no longer decreased after 10 consecutive epochs. Following this initial training, the base layer of each model was unfrozen and then trained for 5 more epochs using a lower learning rate. Finally the model was evaluated on the test data of the specific fold. This process was repeated for each fold and the final metrics were the averages of all 5 folds.

3.7 Implementation of adversarial perturbations

In order to evaluate the robustness of the models to adversarial examples, the package CleverHans was used (Papernot et al. 2018). The package was created by Papernot et al. (2018) in order to facilitate adversarial attacks on neural networks. The package includes a class of functions called attacks which takes a model, test data and hyperparameters (for specific methods) as arguments. The function then generates adversarial examples based on the model and a specific adversarial perturbation algorithm such as FGSM or PGD. Using these adversarial examples, one can predict the example's label using said model. The prediction can then be compared with the true label of the original image in order to determine model accuracy on the test data with adversarial perturbations. For the purposes of our research, we will utilize two different attacks based on FGSM and PGD. These correspond to the following functions from the package: fast_gradient_method and projected_gradient_descent.

4 Results

4.1 Classification accuracy

4.1.1 Model metrics

The results of the models were given by randomly selecting one set of training, validation and test data and fully training each model on this chosen data. These results are shown in table 3.

	Training accuracy	Training loss	Validation accuracy	Validation loss	Test accuracy	Test loss
InceptionV3	98.60%	0.0470	76.85%	0.9626	77.23%	1.0251
DenseNet201	98.85%	0.0448	89.71%	0.3935	86.16%	0.4512
EfficientNetV2	98.68%	0.0409	87.04%	0.5496	84.40%	0.6350

Table 3. Comparison of evaluation metrics of the models

All three models were able to saturate well in training, achieving high accuracy and training losses close to zero. However, validation and test accuracy were more divisive. DenseNet201 performed the best in terms of validation and test accuracy, outperforming EfficientNetV2 by around 2% units. InceptionV3 was the worst performing in this context, with an accuracy around 15% units lower than the other models. However, in terms of training speed InceptionV3 was the fastest out of all three models. Commonly between all three models, we were not able to reduce validation and test loss to zero. We believe that this could be due to the limited size of the validation and test sets which were a result of the limited amount of original data. Seeing as the validation and test sets had low amounts of observations, there might not have been a sufficient amount of variation in the data leading to a loss plateau.

In comparison to the previous attempts to break reCAPTCHA v2 that achieved success rates of 70.78% (Sivakorn, Polakis & Keromytis, 2016) and 83.25% (Hossen et al. 2021), our classifiers are showing some promising potential. Although a direct comparison is not feasible, given that our focus has been solely on the classification components while previous research encompassed the complete reCAPTCHA solver, our models' accuracies are close to the ones achieved in previous studies. This suggests that our three transfer learning models are able to accurately predict the reCAPTCHA images.

For a more in-depth view of the model results, a confusion matrix was made for all three models. The respective confusion matrices for each model follow below.

Predicted/ True	Bicycle	Bridge	Bus	Car	Crosswalk	Hydrant	Palm	Traffic light	Accuracy
Bicycle	68	1	1	2	4	1	1	2	85.00%
Bridge	0	52	1	3	0	0	0	0	92.86%
Bus	0	6	101	6	3	0	2	6	81.45%
Car	4	18	23	223	29	2	34	26	62.12%
Crosswalk	1	3	1	7	106	0	1	7	84.13%
Hydrant	0	0	0	0	0	95	2	1	96.94%
Palm	0	1	0	8	0	0	79	6	84.04%
Traffic light	0	2	4	7	2	1	3	63	76.83%

Table 4. Confusion matrix for InceptionV3

As seen in table 4, the InceptionV3 model struggled the most with predicting class *Car* and *Traffic light*, achieving class accuracies of 62.12% and 76.83% respectively. For images with true label *Car* the model would wrongly predict *Palm* in most cases, followed by *Crosswalk* and *Traffic light*. This could indicate that the *Car* images was the class with the most contamination of objects belonging to other classes, such as the ones most often wrongly predicted. Furthermore, it meant that not only was the InceptionV3 the model which had the lowest overall accuracy, it also performed poorly on the most challenging image class.

Predicted/ True	Bicycle	Bridge	Bus	Car	Crosswalk	Hydrant	Palm	Traffic light	Accuracy
Bicycle	75	0	1	1	1	2	0	0	93.75%
Bridge	0	47	3	1	1	0	4	0	83.93%
Bus	0	1	114	5	1	0	1	2	91.93%
Car	4	9	21	278	11	1	12	23	77.44%
Crosswalk	1	0	1	3	118	0	0	3	93.65%
Hydrant	0	0	0	0	0	98	0	0	100%
Palm	0	2	0	7	0	0	78	7	82.98%
Traffic light	0	3	0	6	0	0	3	70	85.37%

Table 5. Confusion matrix for DenseNet201

The DenseNet201 model on the other hand, performed better than the InceptionV3 model for individual class accuracies as well. As shown in table 5, the accuracy of class *Car* was 77.44%, an increase of 15% units compared to the InceptionV3. For most other classes, the DenseNet201 also improved on the accuracies achieved by the InceptionV3 model. This resulted in a higher overall accuracy, but most importantly DenseNet201 performed better on the more complex classes.

Predicted/ True	Bicycle	Bridge	Bus	Car	Crosswalk	Hydrant	Palm	Traffic light	Accuracy
Bicycle	70	1	2	3	1	2	1	0	87.50%
Bridge	0	54	1	1	0	0	0	0	96.43%
Bus	0	0	120	0	2	0	0	2	96.77%
Car	0	10	31	259	19	0	17	23	72.14%
Crosswalk	0	2	1	8	110	0	0	5	87.30%
Hydrant	0	0	0	0	0	98	0	0	100%
Palm	0	2	1	7	0	0	83	1	88.30%
Traffic light	0	4	3	2	4	0	3	66	80.49%

Table 6. Confusion matrix for EfficientNetV2

As can be seen in table 6, the EfficientNetV2 achieved an accuracy of 72.14% for the *Car* class. This meant an improvement over the InceptionV3 model, but a deterioration compared to the DenseNet201. When comparing the EfficientNetV2 with the DenseNet201, it did achieve a higher accuracy for classes such as *Bridge*, *Bus* and *Palm* leading to a similar overall test accuracy. This emphasizes the importance of comparing separate class accuracies if accuracy on the most complex image class is an imposed goal.

In summary, all three models performed the worst for the class *Car* and the best for the class *Hydrant*, while the other class accuracies varied more between the different models. This result is the same as Bergström and Larsson (2022) demonstrated, meaning that the *Car* class was the most robust image class to the models in our case as well. This is likely a result of images belonging to class *Car* including other objects such as bicycles, buses, traffic lights which are common in a street environment. Consequently, there is a possible incentive for reCAPTCHA creators to include images which contain many different objects in order to mislead automated solvers. Furthermore, this result showed that it is not always sufficient to only consider overall accuracy since an overall high accuracy model can be good at classifying the simpler images, whilst underperforming on more complex ones. A good reCAPTCHA solver should be able to classify all types of reCAPTCHA images with high accuracy.

4.1.2 Stratified K-fold cross validation

The robustness of the results found in table 3 were tested using 5-fold cross validation. As previously mentioned, we performed a stratified 5-fold cross validation, meaning that we fully trained the models on five different folds and finally averaged the loss and accuracy metrics attained on the three different subsets of data (training, validation, and test). These final metrics are presented in table 7.

	Training accuracy	Training loss	Validation accuracy	Validation loss	Test accuracy	Test loss
InceptionV3	92.57%	0.2268	75.53%	0.7872	75.25%	0.8042
DenseNet201	98.62%	0.0451	87.16%	0.4688	86.44%	0.5772
EfficientNetV2	99.49%	0.0264	87.14%	0.4386	86.49%	0.5130

Table 7. Results of stratified 5-fold cross validation

In table 7, training accuracy and loss refers to the average accuracy and loss on the training data over the five different folds. Similarly, validation accuracy and loss refer to average accuracy and loss calculated on the validation data. Likewise, the test accuracy and loss represent the average accuracy and loss evaluated over the test data. These averages are computed across all five folds. All accuracy metrics show the percentage of correct predictions of the labels and loss metrics are the loss values (softmax loss) over the respective subsets of data, averaged over the five folds.

The InceptionV3 model achieved the lowest results overall, having both lower accuracy and higher loss on all three subsets of data on average. Training accuracy was 92.57%, which while quite high is not really sufficient as it should approach 100%. Similarly, training loss was around 0.23, meaning that it did not fully saturate in which case loss should be around 0. Additionally, this training accuracy is around 7% units lower than what was achieved for the one run of full training presented in table 3. Moreover, validation and test accuracy were both around 75%, which is slightly lower (around 1-2% units) than the same accuracies for the InceptionV3 for the non-averaged fully trained model. This suggests that validation and test accuracies shown in table 3 are quite robust while the training accuracy is not as robust.

This could imply that the InceptionV3 model is more susceptible to small differences in the data and that the subset of the original data used for training the one run model could by chance be slightly less optimal.

On the other hand, for DenseNet201 the three accuracies achieved in the 5-fold cross validation did not differ majorly from the results obtained in the one run of full training of the model only being around 1% unit lower for all three metrics. This suggests that the results presented in table 3 are quite robust. Likewise, the results of EfficientNetV2 in the cross validation only differ about 1% unit compared to the accuracies of the subsets of the data for the one time fully trained model. Subsequently, this also suggests that the results are robust for this model.

Overall, the results achieved by the DenseNet201 and EfficientNetV2 seem to be slightly more trustworthy than the results achieved by the InceptionV3 model. Furthermore, the results were similar between the models trained only once and the cross validated models, implying overall robustness of the findings. Moreover, the fact that the average accuracy for the validation and test data does not approach 100% suggests that these sets may not include enough observations to achieve a sufficient level of variation.

4.2 Classification accuracy with adversarial examples

The results of the adversarial attacks on the three models as well as an evaluation of the model accuracy on the unaltered test data set are shown in table 8.

	Clean test accuracy	FGSM test accuracy	PGD test accuracy
InceptionV3	75.25%	65.28%	57.72%
DenseNet201	86.04%	80.10%	77.99%
EfficientNetV2	86.50%	81.30%	80.55%

Table 8. Test data accuracies for different adversarial attacks

In table 8, the columns respond to the accuracies achieved on test data for the different methods of generating adversarial examples. Column clean corresponds to test data without any adversarial examples introduced, column FGSM test data with adversarial examples generated

with fast gradient sign method and column PGD the test data with adversarial examples generated with projected gradient descent. Commonly, between all three models, FGSM lowered the accuracy of the models in contrast to the evaluation on “clean” data. However, PGD decreased accuracy on test data more than FGSM for all the models, which could be a result of the PGD method being more sophisticated than the FGSM.

In terms of robustness to adversarial examples, EfficientNetV2 was the most robust model followed by DenseNet201 and lastly InceptionV3. Seeing as the top classifying layers added by us were the same for all three models it would seem that the differing architectures of the base transfer learning models have an impact on the model’s robustness to adversarial attacks. It would seem that the difference in layers and architectural ideas between the different models plays a role in determining model robustness to adversarial examples. For example, the DenseNet’s dense blocks emphasizing prior knowledge and gradient throughput and the EfficientNet’s specific convolutional layers could have a positive impact on the overall model robustness. Moreover, Szegedy et al. (2014) suggest that adversarial robustness has a correlation with ease of optimization, which was the case for our experiments as InceptionV3 was the quickest to train and also the least robust to adversarial examples.

5 Conclusion

This paper was aimed to explore the extent to which transfer learning models could be leveraged to create a useful reCAPTCHA v2 classifier and assess their robustness against different adversarial attacks. Our investigation has produced significant findings, contributing to the existing research on transfer learning and its applicability to diverse domains, including image-based CAPTCHA systems.

In this study, DenseNet201, EfficientNetV2, and InceptionV3 were trained and validated, demonstrating that transfer learning can indeed be successfully applied to the classification of reCAPTCHA v2 challenges. Despite varying degrees of success, all three models trained satisfactorily, achieving high accuracy and low losses on training data. It was observed that the DenseNet201 model outperformed EfficientNetV2 and InceptionV3 in terms of validation and test accuracy. In contrast, InceptionV3 proved to be the fastest in training speed, offering an advantage in environments where training time is a critical factor.

Additionally, several classes in the dataset featured multiple objects within the frames of the images, which made accurate classification of these particular images more challenging. Nevertheless, DenseNet201 and EfficientNetV2 models were able to produce good results in this context. However, their performance was lower than for classes mainly containing a single object within the image frame. This observation suggests a potential improvement in the reCAPTCHA challenge, where only multi-object images are given in the challenge, thus decreasing the chances for the automated captcha solver to pass through.

When compared to prior attempts to break reCAPTCHA v2 in research, our classifiers, although focused solely on the classification component, showed promising potential, with accuracies nearing those achieved in full solver-based solutions. This finding confirms that transfer learning, a technique traditionally employed in other fields, can be effectively used in solving image-based CAPTCHA challenges as well.

In terms of resilience to adversarial examples, EfficientNetV2 emerged as the most robust model, with DenseNet201 and InceptionV3 following behind. This disparity in robustness among the models, despite having identical top classifying layers, suggests the significance of the underlying base architectures of the transfer learning models on their robustness. This further implies that the choice of model for specific needs should consider not only performance metrics but also the inherent architectural properties that influence adversarial robustness.

Our findings highlight the potential of transfer learning models in the field of image-based CAPTCHA challenge classification and stress the importance of model selection based on specific needs and constraints. It also underscores the impact of adversarial perturbations on these models, emphasizing the need for future research to focus on enhancing the adversarial robustness of these models.

6 Discussion and further research

Although our results show that transfer learning can be applied to solve the reCAPTCHA classification task, it is not necessarily the best way of solving the problem outside the realm of research. One glaring issue is the fact that CNNs need to be trained on the images they are supposed to solve, meaning that by introducing new images and classes to the reCAPTCHA challenges would put a halt to automatic solvers using CNNs. This could result in having to redo the gathering of training data and retraining the models. This could be limited by having highly generalizable models that are able to classify new images, but we still believe that accuracy and success rate of solvers would be negatively affected.

Our research also showed that the models had differing robustness towards adversarial attacks, which also asks questions about the plausibility of implementing adversarial perturbations as defense mechanisms for CNN reCAPTCHA solvers. Firstly, many adversarial examples are generated by using the trained model to, for example, extract the gradient for the FGSM. Since there are a multitude of different, high performing CNN models, it does not seem feasible to create adversarial examples to be used in the reCAPTCHA challenges to negatively impact all possible models. If there is a way to generate adversarial examples in a more general way, maybe it would also impact CNNs more broadly and therefore be more useful as a defense. Secondly, as our results show, the worst performing classifier was also the least robust against adversarial examples, begging the question: is it necessary to defend against an already worse performing model and would it even be utilized in automated solvers?

One of the main challenges of our research was the limited amount of data available. For example, validation and test loss plateaued during the training of all models, not being able to reach zero. We believe that the limited number of observations in these subsets of the data, especially for some classes which were majorly underrepresented, led to a low variation in the validation and test data. Since the variation was not high enough it was not possible to reduce validation and test loss further than what we were able to achieve. Furthermore, gathering more data to test reCAPTCHA is a time-consuming procedure, which requires certain technical knowledge of triggering, collecting and segmenting the reCAPTCHA challenges on websites in order to classify the images. Therefore, it would be interesting to see a study with less time constraints try to validate our results on a larger dataset.

Another issue was the problem of overfitting i.e., the trade-off between model complexity and model generalization. When applying transfer learning one must be especially aware of overfitting, owing to the fact that the pre-trained models are very complex. This high model complexity corresponds to large numbers of parameters, up to 20 million parameters in our case. Since many of the state-of-the-art models were created in order to solve a more complex task, classifying up to 1000 different classes, one must be careful when limiting the objective to fewer classes such as 8 in our case. To our end, we combatted overfitting by utilizing the validation data set, which can aid in detecting overfitting when training the model. Subsequently, we halted our training process if validation loss started to increase after firstly decreasing. Furthermore, we decreased the learning rate and only trained for a few epochs in the most impactful part of the training process, the fine tuning of the base models, in order to avoid overfitting.

The second main challenge was the imbalance of classes in the original data set. We employed one technique, stratified k-fold cross validation, in order to have representative validation and test sets. We then augmented additional images for some of the classes within the training set in order to balance them out before training our models. We limited the amount of training observations to around 1000 images per class, as this threshold meant several classes had no need for added synthetic images, which we argued would be more representative of the original data. We also considered using the number of observations in the *Car* class, around 3000 images, as a benchmark for balancing the data. However, we refrained from this benchmark as we believed it would introduce too many augmented images in the training data set.

As some classes were very underrepresented to begin with (300-500 images) this would have led to synthetic observations making up a great majority of those classes in the training data. Such an approach could have resulted in a different set of problems, such as poorer generalization capabilities.

On the other hand, it could have been beneficial to add augmented data for all of the classes and increase the total amount of synthetic images added to the training data set, which could provide more robustness to the models. Additionally, there are other techniques to overcome imbalanced data sets, such as weighting the loss function proportionally to the ratio of the classes. Due to time constraints, we were unable to implement this in our models, but it is something we would suggest future researchers to explore in this specific topic.

References

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13, 281–305.

Bergström, O. & Larsson, J. (2022). Image Recognition for Solving Google's reCAPTCHA : An Investigation of how Different Aspects Affects the Security of Google's reCAPTCHA (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-320012

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer. Available online: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

Bjerrum, E.J. (2017). SMILES Enumeration as Data Augmentation for Neural Network Modeling of Molecules. *arXiv:1703.07076 [cs]*. [online] Available online: https://arxiv.org/abs/1703.07076.

Boob, D., Dey, S. S. & Lan, G. (2020). Complexity of training ReLU neural network. Discrete Optimization, p.100620. doi:https://doi.org/10.1016/j.disopt.2020.100620.

Carlini, N. & Wagner, D. (2017). Towards Evaluating the Robustness of Neural Networks, 2017 IEEE Symposium on Security and Privacy (SP). doi: 10.1109/SP.2017.49

Gelles, D. (2009). Google buys ReCaptcha to help with security, Financial Times, 17 September, Available online: https://www.ft.com/content/67dc8818-a301-11de-ba74-00144feabdc0 [Accessed 29 April 2023]

Géron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc.

Ghaderzadeh, M., Aria, M., Hosseini, A., Asadi, F., Bashash, D. & Abolghasemi, H. (2022). A fast and efficient CNN model for B-ALL diagnosis and its subtypes classification using peripheral blood smear images. Int J Intell Syst.; 37: 5113- 5133. doi:10.1002/int.22753 [Accessed 04-29-2023]

Goodfellow, I., Bengio, Y. & Courville, A. (2016). Deep Learning. Cambridge, Massachusetts: The Mit Press.

Goodfellow, I.J., Shlens, J. & Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples, Proceedings of the International Conference on Learning Representations (ICLR) 2015. doi:https://doi.org/10.48550/arXiv.1412.6572.

Google (n.d.). reCAPTCHA, Available online: https://www.google.com/recaptcha/about/ [Accessed 29 April 2023]

Guerar, M., Verderame, L., Migliardi, M., Palmieri, F. & Merlo, A. (2022). Gotta CAPTCHA 'Em All: A Survey of 20 Years of the Human-or-computer Dilemma. ACM Computing Surveys, 54(9), pp.1–33. doi:https://doi.org/10.1145/3477142.

Hitaj, D., Hitaj, B., Jajodia, S. & Mancini, L.V. (2020). Capture the Bot: Using Adversarial Examples to Improve CAPTCHA Robustness to Bot Attacks. IEEE Intelligent Systems, pp.1–1. doi:https://doi.org/10.1109/mis.2020.3036156

Hossen, I., Tu, Y., Rabby, F., Islam, N., Cao, H. & Hei, X. (2021). An Object Detection based Solver for Google's Image reCAPTCHA v2. *arXiv.org*. Available online: https://arxiv.org/pdf/2104.03366v1.pdf [Accessed 3 May 2023].

Huang, C., Li, Y., Loy C. C. & Tang, X. (2016). Learning Deep Representation for Imbalanced Classification, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 5375-5384, doi: 10.1109/CVPR.2016.580.

Huang, G., Liu, Z., van der Maaten, L. & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/CVPR.2017.243.

Kurakin, A., Goodfellow, I. & Bengio, S. (2017). Adversarial examples in the physical world, ICLR Workshop (2017). Available online: https://arxiv.org/abs/1607.02533.

Kwasigroch, A., Mikołajczyk, A. & Grochowski, M. (2017). Deep neural networks approach to skin lesions classification — A comparative analysis, 22nd International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje, Poland, 2017, pp. 1069-1074, doi: 10.1109/MMAR.2017.8046978.

LeCun, Y. (1989). Generalization and network design strategies. Technical Report CRG-TR-89-4, University of Toronto.

LeCun, Y., Kavukcuoglu, K. & Farabet, C. (2010). Convolutional networks and applications in vision. Proceedings of 2010 IEEE International Symposium on Circuits and Systems. doi:https://doi.org/10.1109/iscas.2010.5537907.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep Learning. Nature, 521(7553), pp.436–444. doi:https://doi.org/10.1038/nature14539.

Lindholm, A., Wahlström, N., Lindsten, F. & Schön, T. (2022). MACHINE LEARNING A First Course for Engineers and Scientists. Available online: https://smlbook.org/book/sml-book-draft-latest.pdf [Accessed 3 May 2023].

Loshchilov, I. & Hutter F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts, 5th International Conference on Learning Representations (ICLR) 2017. Available online: https://arxiv.org/abs/1608.03983.

Mastromichalakis, S. (2020). ALReLU: A different approach on Leaky ReLU activation function to improve Neural Networks Performance. arXiv preprint arXiv:2012.07564.

Mikołajczyk, A. & Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem, 2018 International Interdisciplinary PhD Workshop (IIPhDW). doi:10.1109/IIPHDW.2018.8388338.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv.org. Available online: https://arxiv.org/abs/1301.3781.

Osadchy, M., Hernandez-Castro, J., Gibson, S., Dunkelman, O. & Pérez-Cabo, D. (2017). No Bot Expects the DeepCAPTCHA! Introducing Immutable Adversarial Examples, With Applications to CAPTCHA Generation. IEEE Transactions on Information Forensics and Security, [online] 12(11), pp.2640–2653. doi:https://doi.org/10.1109/TIFS.2017.2718479.

Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., Matyasko, A., Behzadan, V., Hambardzumyan, K., Zhang, Z., Juang, Y.-L., Li, Z., Sheatsley, R., Garg, A., Uesato, J. & Gierke, W. (2018). Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv.* Available online: https://arxiv.org/abs/1610.00768 [Accessed 4 May 2023].

Perez, L. & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Available online: https://arxiv.org/abs/1712.04621. [Accessed 4 May 2023]

Prusty, S., Patnaik, S. & Dash, S.K. (2022). SKCV: Stratified K-fold cross-validation on ML classifiers for predicting cervical cancer. Frontiers in Nanotechnology, 4, p.972421.

Redmon, J. & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arxiv.org*. Available online: doi:https://doi.org/10.48550/arXiv.1804.02767.

Rogers, S. & Girolami, M. (2016). A First Course in Machine Learning (2nd ed.). Chapman and Hall/CRC. doi:https://doi.org/10.1201/9781315382159

Shi, X., Peng, Y., Chen, Q., Keenan, T., Thavikulwat, A.T., Lee, S., Tang, Y., Chew, E.Y., Summers, R.M. & Lu, Z. (2022). Robust convolutional neural networks against adversarial attacks on medical images. *Pattern Recognition*, [online] 132, p.108923. Doi: https://doi.org/10.1016/j.patcog.2022.108923.

Sivakorn, S., Polakis, J. & Keromytis, A. (2016). I'm not a human: Breaking the Google reCAPTCHA. Available online: https://www.blackhat.com/docs/asia-16/materials/asia-16-Sivakorn-Im-Not-a-Human-Breaking-the-Google-reCAPTCHA-wp.pdf.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. & Fergus, R. (2014). Intriguing properties of neural networks, Proceedings of the International Conference on Learning Representations (ICLR) 2014. doi:https://doi.org/10.48550/arXiv.1312.6199

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Available online: https://ieeexplore.ieee.org/document/7780677.

Tan, M. & Le, Q.V. (2021). EfficientNetV2: Smaller Models and Faster Training, Proceedings of the 38th International Conference on Machine Learning, in Proceedings of Machine Learning Research 139:10096-10106. Available online: https://proceedings.mlr.press/v139/tan21a.html.

von Ahn, L., Blum, M., Hopper, N.J. & Langford, J. (2003). CAPTCHA: Using Hard AI Problems for Security. *Lecture Notes in Computer Science*, pp.294–311. doi:https://doi.org/10.1007/3-540-39200-9_18.

von Ahn, L., Blum, M. & Langford, J. (2004). Telling humans and computers apart automatically. *Communications of the ACM*, 47(2), pp.56–60. doi:https://doi.org/10.1145/966389.966390.

von Ahn, L., Maurer, B., McMillen, C., Abraham, D. & Blum, M. (2008). reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895), pp.1465–1468. doi:https://doi.org/10.1126/science.1160379.

Wang, J., Qin, J., Xiang, X., Tan, Y. & Pan, N. (2019). CAPTCHA recognition based on deep convolutional neural network. Mathematical Biosciences and Engineering, 16(5), pp.5851–5861. doi:https://doi.org/10.3934/mbe.2019292.

Wen, Y., Zhang, K., Li, Z. & Qiao, Y. (2016). A Discriminative Feature Learning Approach for Deep Face Recognition. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9911. Springer, Cham. https://doi-org.ludwig.lub.lu.se/10.1007/978-3-319-46478-7_31

Weng, H., Zhao, B., Ji, S., Chen, J., Wang, T., He, Q. & Beyah, R. (2019). Towards understanding the security of modern image captchas and underground captcha-solving services. *Big Data Mining and Analytics*, 2(2), pp.118–144. doi:https://doi.org/10.26599/bdma.2019.9020001.

Zhang, J., Tsai, M.-Y., Kitchat, K., Sun, M.-T., Sakai, K., Ku, W.-S., Surasak, T. & Thaipisutikul, T. (2022). A secure annuli CAPTCHA system. 125, pp.103025–103025. doi:https://doi.org/10.1016/j.cose.2022.103025.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H. & He, Q. (2021). A Comprehensive Survey on Transfer Learning, *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43-76, Jan. 2021, doi: 10.1109/JPROC.2020.3004555.

Zi, Y., Gao, H., Cheng, Z. & Liu, Y. (2020). An End-to-End Attack on Text CAPTCHAs. IEEE Transactions on Information Forensics and Security, 15, pp.753–766. doi:https://doi.org/10.1109/tifs.2019.2928622.

Zorgui, S., Chaabene, S., Bouaziz, B., Batatia, H. & Chaari, L. (2020). A Convolutional Neural Network for Lentigo Diagnosis. In: Jmaiel, M., Mokhtari, M., Abdulrazak, B., Aloulou, H., Kallel, S. (eds) The Impact of Digital Technologies on Public Health in Developed and Developing Countries. ICOST 2020. Lecture Notes in Computer Science, vol 12157. Springer, Cham. doi:https://doi.org/10.1007/978-3-030-51517-1_8