Deceiving Deep Neural Networks Using Generated Adversarial Input

Tom Brayshaw

Abstract

In the past five years, advancements in deep learning and artificial neural networks have been significant and noteworthy. However, there are still many parts of machine learning methods that we as humans cannot comprehend, which generates risk in the way that these methods are being implemented globally for a plethora of use cases. This project will apply the principles of transfer learning on Google's Inception V3 model to evaluate its capability of facial recognition, with few adaptations, to provide insight into its inner mechanisms. Then we will curate a white-box adversarial attack on the same model to demonstrate how it is susceptible to adversarial input, highlighting the vulnerabilities of the Inception model and abstracting this to Deep Neural Networks as a whole. The project concludes in the hope that it will inspire future research into methods focusing on defending models from adversarial attacks.

Table of Contents

1. Introduction

Deep Neural Networks (DNNs) are effective learning models, able to achieve state-of-the-art pattern recognition performance in many areas of research, such as computer vision, speech recognition, bioinformatics and autonomous systems. This project demonstrates the powerful concept of Transfer Learning on a pre-trained Deep Convolutional Neural Network, adapting Googles' Inception V3 model to successfully attempt a facial recognition task, rather than image classification which is what it was designed for. Having proved this concept indicates that the Inception model is impressively versatile and provides insight into the function of each layer of the network. If the models' entire function can be adapted by only altering the final layers, it signifies that low-level inter-class differences such as colour and shapes, are recognised at the initial layers.

Unfortunately, DNNs have been demonstrated to be susceptible to adversarial input through adversarial attacks. Adversarial input represents a legitimate input, except with added intentional perturbations, and the purpose of causing a misclassification at inference time. Perturbations to the input are constrained to such a small magnitude that it is usually imperceptible to humans yet causes an otherwise powerful classification model to confidently yield erroneous output. In this project, an adversarial network will be implemented to deceive the Inception model into making intentional misclassifications. Having proved the vague function of the initial layers in the network, it becomes apparent that adversarial input uncovers the network to its deepest level, causing the Inception model to even disregard low-level differences it has learned, to force an expected misclassification.

Motivation for this project derives from the fact that DNNs are currently being deployed in every domain imaginable as Machine Learning methods become more pervasive. A vulnerability to adversarial attacks, of which a resolution has not yet been found, exposes a large security flaw in every domain the DNNs are being utilised. Such a successful, publicly available and widely used model such as Inception V3 can be deceived with an almost perfect success rate, as demonstrated in this project. It is necessary that the community recognise these vulnerabilities, as current understanding of the extent of the vulnerabilities is limited.

2. Literature Review

Convolutional Neural Networks are an architecture originally defined in 1980 by Kunihiko Fukushima as a ‘Neocognitron’ [1], introducing the two basic layer types: Convolution and Downsampling. These networks are defined by the characteristic that they have one or more layers of Convolution units. A convolution unit receives its input from multiple units from the previous layer, creating a sense of ‘proximity’ causing the input units to share their weights. The benefit of convolution units can be found in the fact they are considered ‘many-to-one’ mappings, therefore reducing the total number of units in the network and the number of parameters to learn, consequently reducing the chance of overfitting the model. Additionally, the fact that the units takes many inputs forms a ‘neighbourhood’ of information in the model. This is important because if the network can consider the context of an input, it will more accurately recreate the target input. For example, this is highly useful in image and video recognition (where CNNs are typically utilised) as the neighbouring inputs (pixels/frames) tend to carry related information. A few years following Fukushima’s definition Alex Waibel introduced the ‘Time Delay Neural Network’, which was the first convolutional network trained by gradient descent using the backpropagation algorithm [2], which is still used to train these networks to this day.

The first realistic use of convolutional networks for image recognition was displayed in 1989, when Yann LeCun developed a system to recognise handwritten numbers by using backpropagation to learn the convolution kernel coefficients automatically [3], exposing convolutional networks as an approach to broader image recognition problems. After this there was significant (but not ground-breaking) research made on the topic, though progress stalled as hardware requirements grew and it was not until recently that hardware has been capable enough to handle such complex computations. Additionally, large datasets were non-existent until the advancement in Big Data and consequently, significant progress has been made in the last decade. In 2006 came the first GPU implementation of a CNN which performed 4 times faster than an equivalent CPU implementation [4]. Superhuman CNN performance was achieved for the first time in 2011 by Dan Ciresan et al. to win an image recognition contest, and he continued to use this approach to significantly improve on the best

performance for most of the prominent multiple image datasets available at the time [5].

In 2012, Krizhevsky, Sutskever and Hinton trained a deep convolutional neural network on a larger dataset than had ever been used before [6]. This research was vital in improving the performance of certain machine learning methods as prior to this, datasets being used were relatively small (tens of thousands of images). Krizhevsky et al. utilised recently released datasets that contained over 15 million images. The shortcomings of smaller datasets had been widely recognised and objects in realistic settings had significantly more variability, therefore it is considered especially notable that they were able to successfully train on such a dataset and have the model (known as 'AlexNet') perform exceedingly well. Their neural network had 60 million parameters, 650,000 neurons, 5 convolutional layers with some followed by max-pooling layers, 3 fully connected layers, a final 1000-way *Softmax* classification layer and employed the effective *Dropout* regularisation method in the fully-connected layers to reduce overfitting. These methods have seemed to serve as a baseline for CNN implementations since.

Following from the success of AlexNet, Matthew Zeiler and Rob Fergus won the ILSVRC 2013 with their fine-tuned model based on AlexNet. Named 'ZF Net', it achieved a top-5 error rate of 11.2% [15]. Zeiler & Fergus' model included a few minor modifications, including smaller filters in the first layer with a decreased stride value, helping to retain a significant amount of original pixel information that was previously being ignored.

In 2014, Google were the first to develop a network that significantly deviated from the CNN 'standard' that was forming. Szegedy et al. developed GoogLeNet (codenamed 'Inception') which was a 22-layer CNN that did not work by stacking convolution and pooling layer on top of each other sequentially, rather with the use of 'Inception modules' [8]. In a traditional CNN, you would have to make a choice of whether to perform a convolution or pooling operation, whereas these 'modules' allowed you to perform all of these operations in parallel. In GoogLeNet there were 9 Inception 'modules' used, with 152 total layers, and only required 4 million parameters (AlexNet required 60 million).

Szegedy et al. developed the Inception V3 model in 2015, which is the most recent iteration of the 'Inception' models [7]. It improves on Inception V1 by adding batch normalisation to create Inception V2, and then further improves on this by adding substantial factorisation ideas. Inception V3 achieves superior classification performance than most other models, achieving a top-5 error rate of 3.46%, compared to AlexNet's 15.3%, ZF Net's 11.2%, Inception V1's 6.7% and Inception V2's 4.9%.

Superior performance has been achieved more recently, for example the Inception-ResNet-v2 model achieved 3.08% top-5 error rate on the same dataset [9]. They did this by combining two of the most recent ideas: *Residual Connections* [10] and the latest revised version of the Inception architecture (Inception V3). Utilisation of residual connection methods improve training speed greatly.

Schroff et al. released FaceNet in 2015 [11], which is a model designed with the intention of improving the accuracy of a convolutional neural network model on both the Labeled Faces in the Wild dataset and the YouTube Faces Dataset. FaceNet succeeds with an accuracy of 99.63% and 95.12% ± 0.39%, compared to the previous best which was 97.35% and 93.2%. By applying end-to-end training for an embedded space instead of 3D modelling and alignment, vectors in such space are more straightforwardly classified by utilising classical algorithms such as k-Nearest-Neighbour, as they become measurable by Euclidean distance. Additionally, they managed to make a smaller architecture (Inception) perform almost as well as a larger architecture, while significantly reducing the number of parameters/FLOPS. Schroff et al. were critical of previous facial recognition approaches, criticising the use of a classification layer being trained on a set of known identities, followed by an intermediate bottleneck layer that would represent the model's ability to generalise recognition to new data. The authors highlighted the downsides of this approach as being indirect and inefficient, as it is based on the hope that the generalisation to new data performs well, and that it introduces a multitude of unnecessary dimensions via the bottleneck layer. Research has been made to offset the dimension problem, for example Yi Sun et al. in 2014 employed a complex system of multiple stages that works to combine the output of a model with Principal Component Analysis for dimensionality reduction [13], however, this represents a linear transformation which can be learnt in one layer of the network, so it does not carry significant architectural

change. FaceNet attempts to solve the proposed downfalls by directly training its output to be a compact 123-dimension embedding, utilising a triplet loss function based on Large Margin Nearest Neighbor [14].

The breakthrough theory of Adversarial Networks can be traced to Goodfellow et al in 2014, who proposed a framework for estimating generative models via an adversarial process [12]. The general idea behind this was that there would be two networks, a *discriminative* network to estimate the probability that the sample came from the training data, and a *generative* network which would train to maximise the probability that the first network would make a mistake. This corresponds to a minimax game. In this system, both networks are defined by multilayer perceptrons that can be trained with *backpropagation*, which is important because it removes the need for Markov chains or unrolled approximate inference networks during either training or generation of samples. This paper is monumental in its effect on neural networks, as it laid the foundation for much future research.

Literature on the topic of attacks using adversarial networks, in recent years, have grouped threat models into two types; *black-box* and *white-box* attacks. An application of a black-box attack by Papernot et al [20] demonstrated the ability of an adversarial attack to control a remotely hosted Deep Neural Network, with zero knowledge of its' training data, parameters, or internal mechanisms. White-box attacks constitute most research in this area, where information about the discriminator network is known, as they typically provide more insight.

Numerous defence mechanisms have been proposed to nullify the effect of adversarial attacks and can be considered in three different approaches. First, the theory of modifying the classifiers' training procedure to reduce the magnitude of gradients, primarily exemplified by the *Defensive Distillation* method defined by Papernot et al [22]. Defensive Distillation reported to reduce the success rate of an adversarial models' ability to generate a successful adversarial sample from 95% to less than 0.5%, using a variant of the distillation method proposed by Hinton et al [23]. Methods introduced here make it difficult to generate adversarial samples and prove to be effective for white-box attacks, though Carlini & Wagner demonstrated that it failed to protect against black-box attacks transferred from other networks, as they established three new attack algorithms which proved successful on a distilled network with 100%

probability. Consequently, this defence approach is not complete and does not defend adequately.

Another approach is to make the classifier more robust through modifying the training data, for example through the *Adversarial Training* process [24]. In this approach, adversarial samples are generated through specified attack models and added to the training set, which provides an extremely robust classification model when the attacker uses the same adversarial model. When the attacker attempts a different attack strategy, however, it does not perform well. Additionally, it only makes the model robust to white-box attacks due to *Gradient Masking* [25, 20].

The final defence approach to consider is attempting to remove *adversarial noise* from the input samples [26]. A representative example of this approach is MagNet [27], a framework that aims to improve robustness by training a collection of auto-encoders, known as a ‘reformer’ network, to move adversarial input closer to the manifold of legitimate input. It is a particularly strong defence against ‘grey-box’ attacks, where the attacker has all the knowledge about the model and defence except the parameters, but performance falls short for strictly white and black box attacks.

With these current defence approaches considered, their limitations are evident as they are only effective at reducing the success of either white or black box attacks, but not both. Further, some defence methods are designed to invalidate certain specific attack models, and consequently are not effective against new attack strategies. A gap in knowledge becomes apparent here, where a defence method that defends adequately against all types of attacks for all attack models is required, a gap which is not yet satisfied.

3. Methodology

This section of the report will outline the details of the Convolutional Neural Network that will be in use for this project, describe the concept and reasoning of generalising it to new data, and outline the theory powering the application of an adversarial network to deceive it.

3.1. Model

The Inception V3 model [7] developed by researchers at Google will be the Convolutional Neural Network of choice for this project, as it is a state-of-the-art model, where successful attempts at deceiving it would be substantial in exemplifying the effectiveness of an adversarial network. Deconstruction and evaluation of the architecture of such a network would also provide beneficial insight into the mechanisms that such models employ to achieve optimal results. The Inception-ResNet-v2 model described in [9] achieves slightly better performance and therefore would act as a better benchmark, but the performance difference is marginal and the training speed benefits it offers are of little concern in this project, whilst it is also far inferior regarding documentation and maintenance with respect to Inception V3. This is the reasoning for selecting Inception V3 (referred to as 'Inception' for the remainder of the document) over Inception-ResNet-v2.

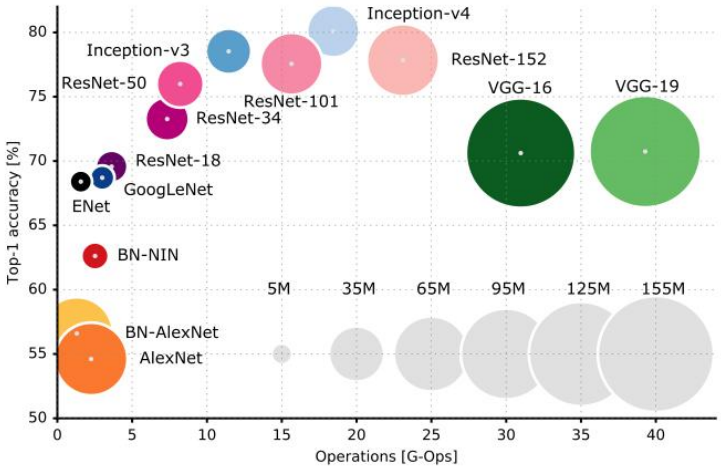


Figure 1: A graph showing the model Accuracy against total Operations. A legacy showing the number of parameters can be found in the bottom right. A small circle in the top left of the graph would represent the best value.

Inception recognised that the salient parts of input images can have significantly large variation in size, which makes it challenging to choose the appropriate kernel size for the convolution operations. Locally distributed information would translate to a smaller kernel size, and information that is distributed more globally would require a larger kernel size. It also recognised that complex, deep models are unfortunately susceptible to overfitting, whilst being more challenging to pass gradient updates through, and that the traditional method of stacking convolution and pooling operations becomes obscenely computationally expensive.

The implemented solution for these issues was based on the concept of making the network “wider” instead of “deeper”, by introducing filters with multiple sizes operating on the same level. This was developed in the form of Inception *modules*, which perform *max pooling* as well as *convolution* on 3 sizes of filters (1x1, 3x3, 5x5), and the outputs are concatenated to be sent to the next network layer. To offset the computational expense this would introduce, the number of input channels are limited through adding an extra 1x1 convolution immediately before the 3x3 and 5x5 convolutions, and immediately after the max pooling layer, representing a dimension reduction on a more “naïve” implementation without the additional 1x1 convolutions.

Inception has 9 *modules* stacked linearly, forming a 22-layer network (27 including pooling layers) with global average pooling following the final *module*. While the *modules* are successful in making the network wider, it is still incredibly deep. With any deep classifier it is susceptible to the vanishing gradient problem, which describes how as more layers use certain activation functions (such as sigmoid) the loss function gradients approach zero, which could potentially halt the ability to train the model. To solve this, Inception contains two auxiliary classifiers which apply a *softmax loss function* to the outputs of two *modules* and computes an *auxiliary loss*, where the total loss function is then a weighted sum of the real loss and the auxiliary loss. This is ignored during inference, though, as it is only for training purposes. However, it was noted that the auxiliary classifiers were not of much use until accuracies were saturating near the end of the training process, so in V3 *batch normalisation* was added to these classifiers.

Upgrades made to the Inception architecture in V2 sought to improve computational efficiency with smart factorisation methods. They achieved this by splitting larger convolutions into multiple smaller ones, for example a 5x5 convolution can be represented by two 3x3 convolutions, except the 5x5 convolution is almost 3 times more expensive. Further, they factorise convolutions of $n \times n$ to a combination of $n \times 1$ and $1 \times n$ convolutions, for example, a 3x3 convolution is equivalent to a 1x3 convolution followed by a 3x1 convolution, which proved to be 33% cheaper than a single 3x3 convolution. Combining these two ideas, a 5x5 convolution is equivalent to: $1 \times 3 \rightarrow 3 \times 1 \rightarrow 1 \times 3 \rightarrow 3 \times 1$. Additionally, V2 intended to reduce *representational bottleneck*, which occurs when information is lost through *dimensionality reduction*. They managed this by expanding the filter banks in the Inception *module*, making them "wider", by performing convolution operations in parallel. In V3, factorisation for 7x7 convolutions was implemented as three consecutive 3x3 convolutions.

3.2. Generalising the model to new data

It is important to prove that the Inception V3 model is an incredibly capable CNN in order to understand the magnitude of the concept of deceiving it. This will be done by evaluating the performance of the model on data it has not seen before, for a task it has not been designed for. Inception was trained on the *ImageNet* database which consists of 1.2 million images, however, it is limited to classifying to only 1000 different classifications on which it was trained. To classify new objects for new tasks, it is necessary to train the parameters of at least one network layer. In theory, the earlier layers in the network are good at recognising low-level features like shapes and colours, leaving the final layers to distinguish class-specific features. It is not feasible to train the entire network on a new dataset for this task, so only the last layer will be trained, and it is important to note that this means the results will not be optimal.

First, a dataset must be collected that will be presented to the model as new data. It should contain enough labelled images in every class so that it gives the model ample opportunity to learn every classification. *ImageNet* consists of around 1200 images per class, which is not feasible with limited resources, so a smaller dataset is required. An appropriate dataset is *Labelled Faces in the Wild* (LFW) [16], a collection of 13,233

images of 5749 different individuals gathered from the internet. It is apparent that 4069 individuals have only one image in this dataset, which would be impossible for Inception to learn their classification. Considering this, it is necessary to take the top 10 individuals with the most images as the dataset to be trained on. This is an attempt to balance a fast training time with being fair to the model by giving it enough opportunity to learn.

It is not within the scope of this project to implement image *pre-processing* techniques; however, it is useful to obtain more accurate results. Therefore, a version of LFW courtesy of Huang et al. [17] will be utilised. In this dataset, all images have been pre-processed through the *deep funnelling* algorithm, which is an image alignment technique that aims to reduce intra-class variability due to factors such as lighting, pose, background, and perspective transformation, by detecting facial feature points on the image and transforming these points to a canonical configuration, so to align the image. Performance is improved by allowing the model to learn more about inter-class differences, such as facial features and facial landmark positions.

Transfer Learning is a powerful concept in machine learning, which describes how a model that proves adept at one task is often adept at similar tasks, and this principle is demonstrated here by the fact that Inception has been trained on 1.2 million images, none of which contained a single face or were of a class related to “human”, yet with just adapting a single layer (of 100+ total) it is able to differentiate between 10 different human individuals.

3.3. Adversarial Networks

Neural network models typically represent a *discriminative* algorithm, which is described by aiming to achieve the highest possible accuracy in classifying a certain input. The model will learn the *conditional probability distribution* $p(y|x)$, or the decision boundary between target classes, displayed in examples such as logistic regression or support vector machines.

Alternatively, a model can represent a *generative* algorithm which instead aims to understand the input in such a way that it can generate similar input (with correct labelling). To do this it instead learns the *joint probability distribution*:

$$P(x, y) = p(y|x) \cdot p(x)$$

where $p(x)$ is what the model needs to learn. Generative behaviour is exemplified in the Naïve Bayes classifier.

Generative Adversarial Networks (GANs) belong to the set of generative models, therefore they are designed to generate content as an output. The main intention of a GAN is to predict features given a label, the antithesis of the intention of a discriminative model. GAN architecture consists of two neural networks; a *Generator* (G) which generates new data points from a random normal or uniform distribution, and a *Discriminator* (D) which identifies the fake data produced by G from real data. These two networks will compete in a zero-sum game, as G generates data attempting to fool D. If G is unsuccessful, it will learn to instead produce similar data and attempt again. This process is repeated until *Nash equilibrium* is reached, where G can be considered stable and its outputs consistently successful, completing the adversarial training process.

Input to G is a sample of noise from normal or uniform distribution of random values, and conceptually, this input represents latent features of the images generated (low level image features such as colour, shape, texture). It is difficult to comprehend the semantic meaning of this input as we have no control over it, it is only useful for the training process. The *Generator*, G, is an implementation of Deep Convolutional Generative Adversarial Network architecture which performs multiple transposed convolutions to *upsample* the input and generate a new image.

4. Implementation

Implementation of the two previously outlined concepts will require two different programs to be developed. The first will train Inception to classify new data and new classes, whereas the second will focus on training an adversarial network to successfully deceive the Inception model. These will both be developed as *Python* scripts, relying heavily on the *TensorFlow* library for retrieval of the Inception model along with useful training functions. Other libraries used frequently include *numpy*, *Pillow*, and *matplotlib*. These scripts will be able to be run as-is, provided the user attempting to run them has followed the *Installation Instructions* outlined in Appendix E1.

4.1. Classifying New Data

To begin trying to get Inception to generalise to a new task, the Inception model must first be retrieved. However, the model itself is not sufficient as we require some functionality and capability from the start, so we must restore an Inception *checkpoint* which provides a pre-trained model to work with. Here we will be using an Inception model released on 28th August 2016.

4.1.1. Data Wrangling

Inception has been trained on the *ImageNet* dataset, which does not contain a single image of a human. To ensure the new data is truly novel, it is appropriate to select a dataset that consists of only human faces, as a 'human face' is a concept that Inception is guaranteed to not be familiar with. Therefore, we will use the *Labelled Faces in the Wild* (LFW) dataset [16], more specifically the version of the dataset that has been pre-processed by the *Deep Funnelling* [17] algorithm to increase the general accuracy of the model and aid in achieving the objective, as using these pre-processed images will help Inception differentiate people based on their facial features rather than the lighting or background in the image. The LFW dataset contains over 13,000 images of 5749 people and it is not necessary for the model to learn all examples, especially as

many of them have only a single image. Using too many images would severely inflate the *bias* of the neural network, so it would not be able to learn the underlying differences between classes. Therefore, we will use the top 10 individuals with the most images in the dataset:

1. George W. Bush, 530 images.
2. Colin Powell, 236 images.
3. Tony Blair, 144 images.
4. Donald Rumsfeld, 121 images.
5. Gerhard Schroeder, 109 images.
6. Ariel Sharon, 77 images.
7. Hugo Chavez, 71 images.
8. Junichiro Koizumi, 60 images.
9. Jean Chretien, 55 images.
10. John Ashcroft, 53 images.

This presents two issues; *limited samples* (some classes have ~50 images, is it enough for the model to learn?) and *uneven class distribution*. We can say the classes have uneven distribution because more than half of the images are images of either George Bush or Colin Powell, so it is a valid concern that the model might learn to identify these two as a “safe guess”, incorrectly labelling individuals when the network is unsure of a classification. This represents the model having *high recall* with respect to George Bush or Colin Powell, but *low precision*. Typically, it is necessary to adjust the architecture to accompany the precision/recall trade-off, keeping in mind whether false positives/negatives should be allowed, however, as we are not intending to achieve record results we can work with a non-optimal dataset. The common values for splitting the dataset will be used; 70% training data, 25% testing data, and 5% validation data.

Images of the individuals need to be translated into arrays before the sets are split, which is achieved with the *imread* function from the *matplotlib* library:

```
image_arrays = []
image_labels = []
root_image_directory = "images/faces/"
for label, person in class_mapping.items():
    for directory in os.listdir(root_image_directory):
        if directory == person:
            image_directory = root_image_directory + directory
            break

    for image in os.listdir(image_directory):
        image = plt.imread(os.path.join(image_directory, image))
        image_arrays.append(image)
        image_labels.append(label)
image_arrays = np.array(image_arrays)
image_labels = np.array(image_labels)
```

This gives us an array of shape *(1456, 250, 250, 3)*. We now need to split these 1456 images into their respective sets:

```
X = np.array(image_arrays)
X_raw_test = []
X_raw_valid = []
X_raw_train = []
y = np.array(image_labels)
y_raw_test = []
y_raw_valid = []
y_raw_train = []

random_indices = np.random.permutation(len(X))
X = X[random_indices]
y = y[random_indices]

for image, label in zip(X, y):
    test_length = math.floor(test * class_images[label])
    valid_length = math.floor(validation * class_images[label])

    if Counter(y_raw_test)[label] < test_length:
        X_raw_test.append(image)
        y_raw_test.append(label)
    elif Counter(y_raw_valid)[label] < valid_length:
        X_raw_valid.append(image)
        y_raw_valid.append(label)
    else:
        X_raw_train.append(image)
        y_raw_train.append(label)

X_train = np.array(X_raw_train, dtype=np.float32)
X_valid = np.array(X_raw_valid, dtype=np.float32)
X_test = np.array(X_raw_test, dtype=np.float32)
y_train = np.array(y_raw_train, dtype=np.int32)
y_valid = np.array(y_raw_valid, dtype=np.int32)
y_test = np.array(y_raw_test, dtype=np.int32)
```

4.1.2. Pre-processing

Now we have three arrays; a training set (1027 images), testing set (361 images), and validation set (68 images). It is important to note that the array with the class labels (correct names for each image) is put through the same process.

Now we must *pre-process* the training set array, as Inception requires a different image size as input than what is provided by the LFW dataset. Images are read as arrays by specifying a value between 0 and 255 for each pixel (x and y location) for each colour channel (RGB). Inception takes images of size 299x299 whilst LFW images are 255x255. Pixel values will also need to be normalised from 0-255 to 0-1. The following code will execute for each image in the training set, when it should be used:

```
def preprocess(image):
    image = imresize(image, (299, 299)) / 255
    return image
```

4.1.3. Inception Layer

For this to work, it is necessary to define a new output layer as well as train at least one of the layers specifically for the new data. This is powered by a rather hopeful concept; that the previous layers in training are useful for differentiating low-level features such as objects and colours, rather than high-level features which would be class-specific. We will train the penultimate layer in the assumption that the weights learned from ImageNet are also useful in the facial recognition task.

In the TensorFlow slim library there is an *inception_v3* function, which will be used here. It returns *logits* (unscaled output) and *endpoints* (a dictionary containing the outputs of each layer). Using the endpoints dictionary, we can locate the penultimate layer, which is the layer immediately before the classifications are made:

```
tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, 299, 299, 3], name='X')
is_training = tf.placeholder_with_default(False, [])

with slim.arg_scope(inception.inception_v3_arg_scope()):
    logits, end_points = inception.inception_v3(X, num_classes=1001,
is_training=is_training)
```

Here, the *PreLogits* layer is the array of unscaled outputs with a shape of *(None, 1, 1, 2048)*, created by applying *dropout* after *average pooling* the previous layers with an *8x8 kernel*. We need this to become a 2048 neuron fully-connected layer by removing the two dimensions of size 1, which is achieved by utilising the TensorFlow provided *squeeze* function.

```
prelogits = tf.squeeze(end_points['PreLogits'], axis=[1, 2])
```

Now a new output layer needs to be defined that takes this new layer as input. The probabilities for each class can be found by applying a *softmax* activation function on the logits provided by the previous layer. Then the implementation follows a path predictable for a classification neural network; use the *average cross entropy* as a loss function, followed by an optimisation operation. In this case we will use *Adam Optimisation* [19], as it seemed to provide significantly faster optimisation than typical gradient descent in this case. It is an extension to *stochastic gradient descent* which maintains per-parameter learning rates, adapted based on the average recent magnitudes of gradients and the average of the second magnitudes (the uncentered variance), realising a performance improvement on sparse gradients. Essentially, it combines the *Adaptive Gradient* and *Root Mean Square Propagation* algorithms, with added flair.

```
people_logits = tf.layers.dense(prelogits, n_outputs, name="people_logits")
probability = tf.nn.softmax(people_logits, name='probability')

y = tf.placeholder(tf.int32, None)

loss =
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=people_logits,
labels=y))
train_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
scope="people_logits")
training_op = tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss,
var_list=train_vars)

correct = tf.nn.in_top_k(predictions=people_logits, targets=y, k=1)
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

It is not possible to pass all the training or test images through at the same time due to limited memory, so it is necessary to process these images in *batches*:

```
def batch(X, y, start_index=0, batch_size=4):
    stop_index = start_index + batch_size
    prepared_images = []
    labels = []
    for index in range(start_index, stop_index):
        prepared_images.append(preprocess(X[index]))
        labels.append(y[index])

    X_batch = np.stack(prepared_images)
    y_batch = np.array(labels, dtype=np.int32)
    return X_batch, y_batch
```

This will split the training and test data at runtime, also allowing the implementation of *early stopping* here as we can process the whole validation set at once (as it only consists of 68 samples, which can fit in memory). In this case early stopping is implemented by testing the network on the validation data each *epoch*, to evaluate the *average cross entropy*. If this value does not decrease for a certain number of epochs, training is stopped, and the optimal model is retained by saving the model when a better loss value is achieved.

4.1.4. Performance Evaluation

Once the network is trained for the facial recognition task, we can measure its performance against the testing data to determine whether it has performed adequately. Again, batch processing will need to be used to avoid memory issues:

```
eval_batch_size = 32
n_iterations = len(X_test) // eval_batch_size
with tf.Session() as sess:
    saver.restore(sess, unaugmented_training_path)

    start_index = 0
    test_acc = {}

    t0 = time.time()
    for i in range(n_iterations):
        X_test_batch, y_test_batch = batch(X_test, y_test, start_index,
batch_size=eval_batch_size)
        test_acc[i] = accuracy.eval({X: X_test_batch, y: y_test_batch})
        start_index += eval_batch_size
        print('Iteration: {} Batch Testing Accuracy: {:.2f}%'.format(i + 1,
test_acc[i] * 100))
```

```
 1 Validation Loss: 2.5502 Accuracy: 0.455882
 2 Validation Loss: 1.5946 Accuracy: 0.529412
 3 Validation Loss: 1.2524 Accuracy: 0.544118
 4 Validation Loss: 0.9393 Accuracy: 0.705882
 5 Validation Loss: 0.8756 Accuracy: 0.735294
 6 Validation Loss: 0.8405 Accuracy: 0.735294
 7 Validation Loss: 0.8566 Accuracy: 0.676471
 8 Validation Loss: 0.8462 Accuracy: 0.705882
 9 Validation Loss: 0.8386 Accuracy: 0.720588
10 Validation Loss: 0.9092 Accuracy: 0.676471
11 Validation Loss: 0.7431 Accuracy: 0.750000
12 Validation Loss: 0.7564 Accuracy: 0.838235
13 Validation Loss: 0.7857 Accuracy: 0.735294
14 Validation Loss: 0.7889 Accuracy: 0.720588
15 Validation Loss: 0.7945 Accuracy: 0.735294
16 Validation Loss: 0.7990 Accuracy: 0.691176
17 Validation Loss: 0.8206 Accuracy: 0.676471
18 Validation Loss: 0.9208 Accuracy: 0.720588
19 Validation Loss: 1.0273 Accuracy: 0.720588
```

Figure 2: Training progress as console output from running the script shows the Loss and Accuracy for each training epoch.

In this case an *early stopping* parameter of 8 was used, so it would run for a maximum of 8 more epochs if loss is not improving. The model reached optimal loss at epoch 11, so this version of the model is saved and will be restored for testing, and training stops at epoch 19. Loss can be improved by removing early stopping, but a value of 8 appeared to balance well between good loss and short training time. In this instance training took 55 minutes.

```
Iteration: 1 Batch Testing Accuracy: 50.00%
Iteration: 2 Batch Testing Accuracy: 68.75%
Iteration: 3 Batch Testing Accuracy: 62.50%
Iteration: 4 Batch Testing Accuracy: 78.12%
Iteration: 5 Batch Testing Accuracy: 68.75%
Iteration: 6 Batch Testing Accuracy: 62.50%
Iteration: 7 Batch Testing Accuracy: 65.62%
Iteration: 8 Batch Testing Accuracy: 75.00%
Iteration: 9 Batch Testing Accuracy: 59.38%
Iteration: 10 Batch Testing Accuracy: 68.75%
Iteration: 11 Batch Testing Accuracy: 78.12%

Final Testing Accuracy: 67.0455% on 361 instances.
```

Figure 3: Testing progress as console output from running the script shows the accuracy of the model's classifications for each batch. Final testing accuracy displays the average accuracy over all test batches.

The results of testing showed that the average top-1 accuracy of the model over all batches was *67%*. This might outperform most humans; however, it is not particularly impressive for a facial recognition system. Nevertheless, it successfully proves the fact that Inception can be altered to perform well at a task other than the one it was designed for, as it was able to learn the inter-class differences with training.

4.2. Adversarial Process

Just as in the generalisation script, it is necessary to retrieve a checkpoint of the Inception model that will act as the *discriminator*, in this case the 28th August 2016 checkpoint. For the adversarial process, however, the *ImageNet classes* are also required. These labels will be represented in a JSON file, of which the contents of that file can be found in Appendix E4 and are required to be loaded into memory at runtime.

4.2.1. Scaling the Input

Once the Inception model is restored and ready, an input image needs to be selected. The program has been written to retrieve an image from a specified web address to aid demonstration of the flexibility of the adversarial process (being able to alter any input image reduces potential bias). It is required to be at least 299x299 pixels as this is the shape of the *input tenso*r for Inception, so if it is larger it will be resized and then cropped to the centre 299x299 pixels with this code:

```
img = PIL.Image.open(img_path)
wide = img.height < img.width

nw = 299 if not wide else int((img.width*299) / img.height)
nh = 299 if wide else int((img.height*299) / img.width)

img = img.resize((nw, nh)).crop(((nw/2)-149, 0, (nw/2)+150, 299)) \
    if wide else img.resize((nw, nh)).crop((0, (nh/2)-150, 299, (nh/2)+149))
img = (np.asarray(img) / 255.0).astype(np.float32)
```

4.2.2. Benchmark Classification

As a benchmark, the original scaled image will be classified by Inception to demonstrate the model's ability to correctly classifying the image prior to any adversarial perturbation. In the running example in this section, an image of a basketball will be used.

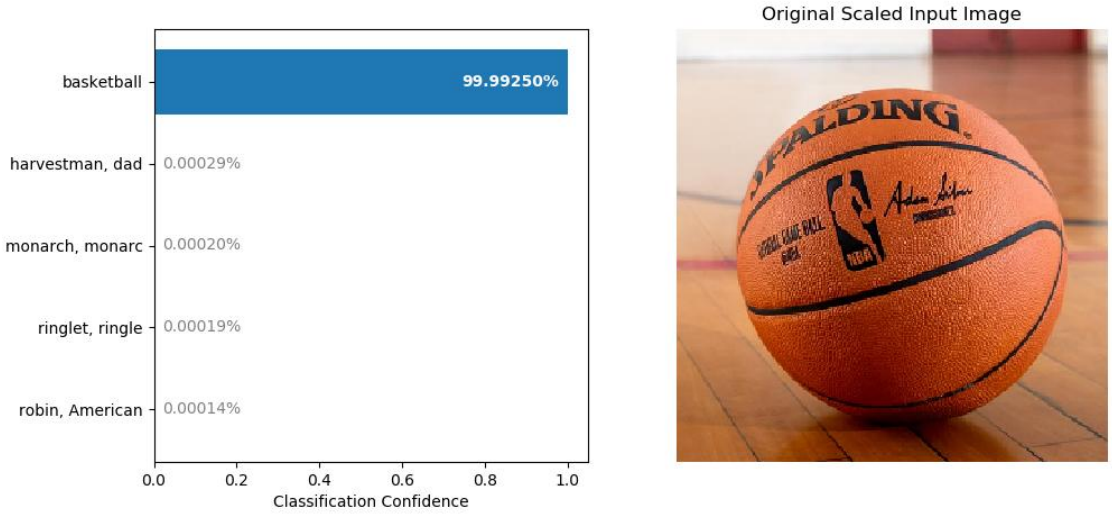


Figure 4: The 299x299 scaled input image (right) and the top 5 classifications with confidence percentage (left).

Inception was able to correctly classify the image as a Basketball with over *99.99%* confidence.

4.2.3. Creating Adversarial Noise

The next phase is to begin the adversarial process by using the same input image and adding *noise* to it, until it can fool Inception into thinking it is an image of another target class. In the program the target misclassification is randomised at the start, as another demonstration of flexibility and bias reduction. On this run it had selected *hourglass* as the target misclassification.

Inception outputs a probability distribution over a set of labels, given the vectors of an input image $\boldsymbol{x}$. The adversarial network, on the other hand, is seeking to generate an image $\dot{\boldsymbol{x}}$ where $logP(\dot{\boldsymbol{y}}|\dot{\boldsymbol{x}})$ is maximised, given a target misclassification $\dot{\boldsymbol{y}}$**.** It is also necessary to constrain to an $\boldsymbol{\ell\infty}$ sequence space with radius $\boldsymbol{\varepsilon}$**,** to ensure that $\dot{\boldsymbol{x}}$ does not change significantly enough from $\boldsymbol{x}$ to be immediately noticeable by the human eye. This requires that $|\boldsymbol{x} - \dot{\boldsymbol{x}}|\infty \leq \boldsymbol{\varepsilon}$.

This represents a constrained optimisation problem, which can be solved using *backpropagation*, and particularly *projected gradient descent* techniques. The algorithm implemented in this project can be defined efficiently:

1. Initialise: $\dot{x} \leftarrow x$
2. Repeat until convergence:

$$\dot{x} \leftarrow project(\dot{x} + a \cdot \nabla logP(\dot{y}|\dot{x}), x - \varepsilon, x + \varepsilon)$$

The gradient descent part of this algorithm is implemented in the optimisation step of the network. It minimizes the output from the loss function, in this case a *softmax cross entropy* computation between logits and labels, which in turn maximises the log probability of the target misclassification.

```
input_placeholder = tf.placeholder(tf.float32, (299, 299, 3))

assign_input_to_placeholder = tf.assign(image, input_placeholder)

learning_rate_placeholder = tf.placeholder(tf.float32, ())
classification_label_placeholder = tf.placeholder(tf.int32, ())

labels = tf.one_hot(classification_label_placeholder, 1000)

loss = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=[labels])

optim_op =
tf.train.GradientDescentOptimizer(learning_rate_placeholder).minimize(loss,
var_list=[input_image])
```

Projection is necessary to constrain the alteration to the adversarial input to within the previously defined ε limits:

```
epsilon_ph = tf.placeholder(tf.float32, ())
lower = input_placeholder - epsilon_ph
upper = input_placeholder + epsilon_ph

projected = tf.clip_by_value(tf.clip_by_value(image, lower, upper), 0, 1)

with tf.control_dependencies([projected]):
    projection_op = tf.assign(image, projected)
```

At runtime, there are two *hyperparameters* that need to be considered. First, the value of ε should be very small as it defines the upper and lower bounds of any changes to the adversarial input image in the projection step. In this implementation ε has been set to 0.008 as it seemed to be the optimal value for fast convergence and high stability. The second *hyperparameter*, the *learning rate*, is set to 0.1 which represents a typical rate. A more advanced implementation could use a momentum algorithm to optimise this hyperparameter, but this would provide little benefit in this project.

Now that we have defined the optimisation and projection operations, we can run them in the following sequence:

```
last_loss = 10

for i in range(steps):
    if(last_loss > 0.01):
        _, loss_value = sess.run(
            [optim_op, loss],
            feed_dict={learning_rate_placeholder: lr,
                       classification_label_placeholder: target_class})

        last_loss = loss_value

        grad = sess.run(projection_op, feed_dict={input_placeholder: img,
epsilon_ph: epsilon})

        print(f"Step {i+1}, Loss = {loss_value}")
    else:
        print(f"Stopping early - minimal loss reached at step {i}\n")
        break
```

Early Stopping is implemented here so if the loss returned is lower than a threshold value, it is considered strong enough output to trick the discriminator already, so any additional steps will have significantly diminishing returns. In this implementation, the maximum number of *steps* is set to 50 (relatively low) and the acceptable loss threshold is set to 0.01. A loss value lower than 0.01 typically represents a classification confidence of above 99%.

```
Learning Rate: 0.1
Epsilon: 0.008
Maximum Steps: 50
Target Class: 604 - hourglass

Step 1, Loss = [17.714827]
Step 2, Loss = [9.007754]
Step 3, Loss = [7.1812725]
Step 4, Loss = [6.573827]
Step 5, Loss = [5.980139]
Step 6, Loss = [5.3637724]
Step 7, Loss = [4.9654655]
Step 8, Loss = [4.4576445]
Step 9, Loss = [4.3241205]
Step 10, Loss = [3.4613123]
Step 11, Loss = [3.557744]
Step 12, Loss = [1.673381]
Step 13, Loss = [2.5659466]
Step 14, Loss = [3.0033193]
Step 15, Loss = [1.3781822]
Step 16, Loss = [1.2728398]
Step 17, Loss = [0.22616917]
Step 18, Loss = [0.09253006]
Step 19, Loss = [0.01014623]
Step 20, Loss = [0.0015947]
Stopping early - minimal loss (classification confidence above 99%) reached at step 20
```

Figure 5: Console output from training the adversarial network shows the loss value for each iteration of projected gradient descent. Training stops at step 20 because a very small loss value is achieved.

It is possible to visualise the adversarial noise $\boldsymbol{Z}$ by normalising the difference in array values of the original input image $\boldsymbol{x}$ and the generated adversarial image $\dot{\boldsymbol{x}}$, defined by the following formula:

$$Z = \frac{|x - \dot{x}| - min(|x - \dot{x}|)}{max(|x - \dot{x}|) - min(|x - \dot{x}|)}$$

Figure 6: The adversarial noise visualised as a 255x255 image, achieved by normalising the difference between the original and adversarial images. Each pixel has 3 values, each one associated with a colour channel (RGB).

To the human eye, the output of the adversarial network $\dot{x}$ does not appear to change the original image at all. Though when fed into the Inception classifier, it confidently classifies it as the target classification selected earlier ('*hourglass*').

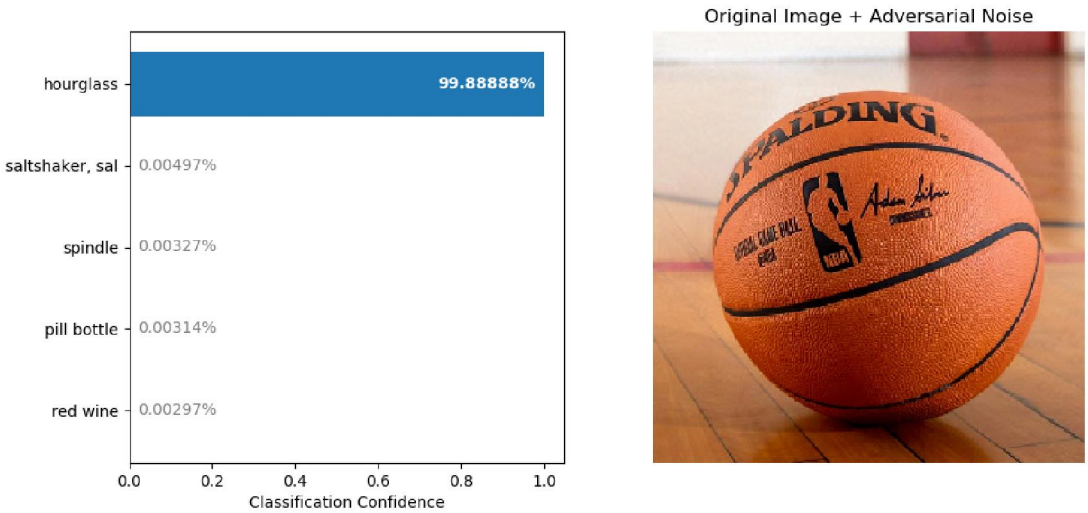


Figure 7: The image produced from the adversarial network, consisting of the original image with small pixel perturbations, intended to make Inception classify it as an 'hourglass' (right) and Inceptions' top 5 classifications of the adversarial image (left).

4.2.4. Evaluating Adversarial Success

This shows that it is possible to harness a generative adversarial network to trick the Inception model into misclassifying *any* input image to *any* target class (that it has trained on). However, this adversarial process is not sufficiently robust because, under the circumstance that the input image is transformed in any way through scale/zoom or rotation, Inception will correctly classify the image, meaning the adversarial network has failed. Fortunately, this downfall can be eliminated by training the Inception model on transformed examples, hence to train a robust adversarial network. In this case we will be exploring the adversarial network robustness to transformations by rotation.

4.2.5. Robust Adversarial Network

It is first necessary to demonstrate that Inception can correctly classify the adversarial image if it is rotated slightly, which means the network is not generating adversarial output with rotation invariance.

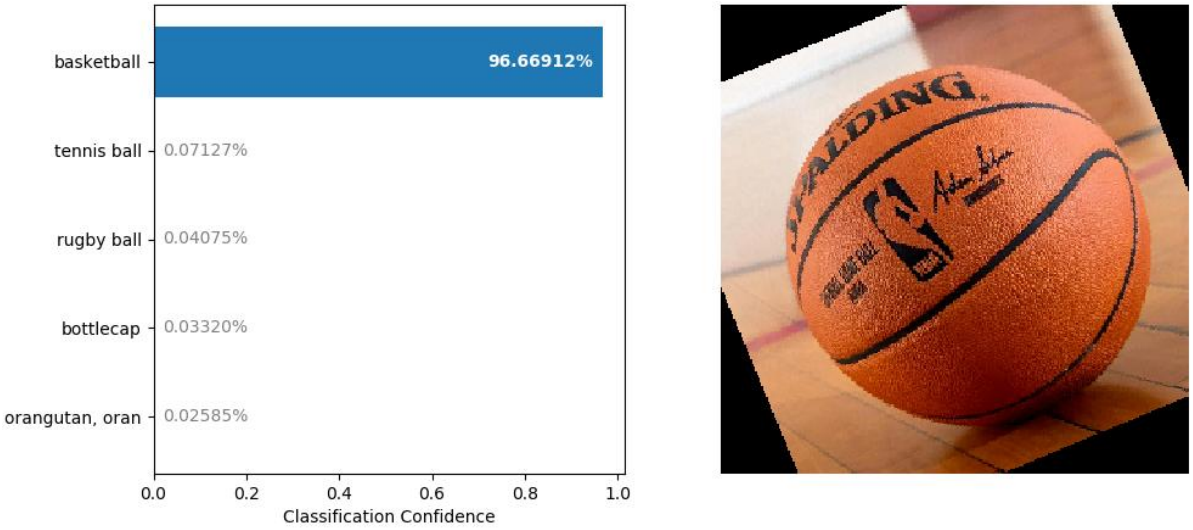


Figure 8: The adversarial image generated in the last section rotated by pi/8 (right) and Inceptions' top 5 classifications of the rotated adversarial image (left).

Here we can see that when the same adversarial image that was previously classified as '*hourglass*' is rotated by a factor of $\frac{\pi}{8}$, Inception correctly classifies the image as '*basketball*'.

Next, we will train Inception on a specified number of samples of the basketball image rotated with random values between a certain range, which should be enough for it to confidently classify the image. The optimisation step is then updated to use gradient descent optimisation whilst minimising the average loss produced by the softmax cross entropy loss function. The number of samples we will use in this case is 30 and rotate between the range of $-\frac{\pi}{4}$ and $\frac{\pi}{4}$.

```
samples = 30
avg_loss = 0

for i in range(samples):
    r = tf.contrib.image.rotate(image, tf.random_uniform((), minval=-np.pi/4,
maxval=np.pi/4))

    rlogits, _ = inception(r, reuse=True)
    avg_loss += tf.nn.softmax_cross_entropy_with_logits_v2(logits=rlogits,
labels=labels) / samples

optim_op =
tf.train.GradientDescentOptimizer(learning_rate_placeholder).minimize(avg_loss,
var_list=[image])
```

The implementation of the final step is almost identical to the previous adversarial process, except the hyperparameters are slightly different and the loss function is the output of the average loss operation from the previous training step. Now we will use an $\boldsymbol{\varepsilon}$ value of 0.03 and a learning rate of 0.2 for even faster convergence, as this process can take a lot of time depending on the number of samples chosen.

```
epsilon = 0.03
lr = 0.2
steps = 50

sess.run(assign_input_to_placeholder, feed_dict={input_placeholder: img})

last_loss = 10
for i in range(steps):
    if (last_loss > 0.01):
        _, loss_value = sess.run([optim_op, avg_loss],
            feed_dict={learning_rate_placeholder: lr,
classification_label_placeholder: target_class})

        sess.run(projection_op, feed_dict={input_placeholder: img, epsilon_ph:
epsilon})

        last_loss = loss_value
        print(f"Step {i + 1}, Loss = {loss_value}")
    else:
        print(f"Stopping early - minimal loss (classification confidence above 99%)
reached at step {i}")
        break
```

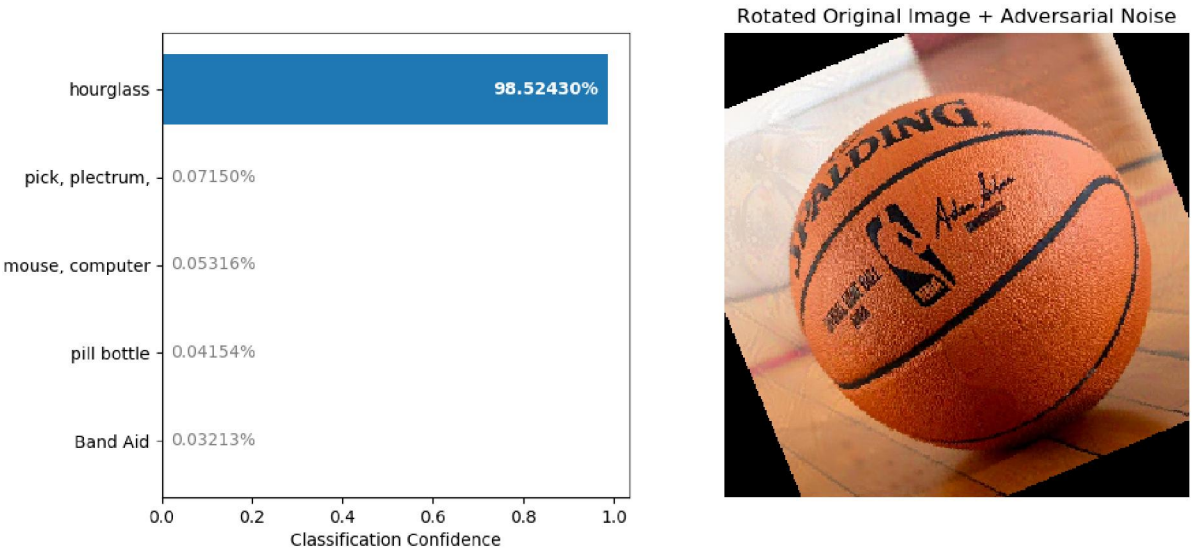


Figure 9: The new adversarial image rotated by pi/8 (right) and Inceptions' top 5 classifications (left).

We can see here that the adversarial network, when trained to be rotation invariant, outputs an adversarial image able to trick Inception with rotation-invariance. To visualise the strength of this approach we can plot the probability of adversarial success (for a number of values in a certain range) of the robust model against the previous, more "naïve" model.

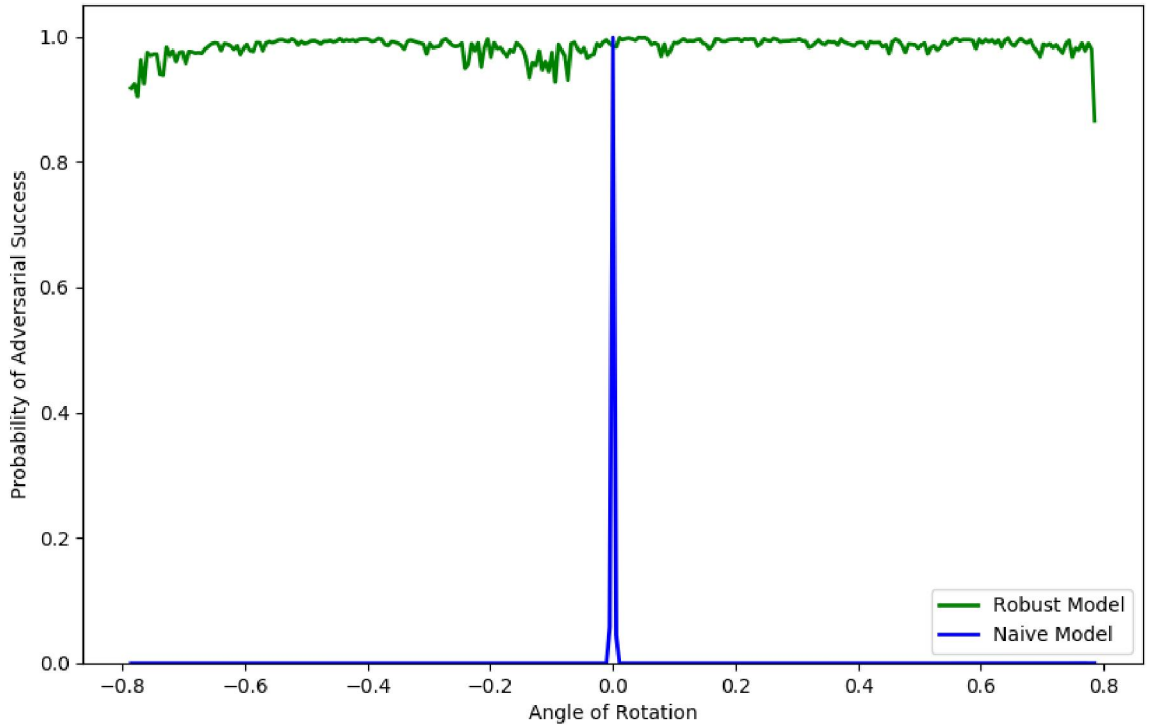


Figure 10: Graph showing the probability of adversarial success against the angle of rotation of an image, for the robust and naïve models.

Evidently, the robust model is more successful. This approach can be undertaken for other transformations, such as larger rotations or zoom/scale factors.

5. Evaluation

Inception proves itself as remarkably capable at image classification. Through testing it consistently correctly classified even tricky sample images. Following is an example of such testing that was performed.

Actual Class	Predicted Class	Confidence (%)
Guacamole	Guacamole	98.05155
Prison	Prison	96.44824
Great white shark	Great white shark	67.81483
Komondor	Komondor	96.08900
Chain link fence	Chain link fence	99.46285
Digital clock	Digital clock	98.78860
Harp	Harp	99.99970
Table lamp	Table lamp	86.25525
Planetarium	Stage	44.24981
Wreck	Wreck	99.92675
Trifle	Trifle	98.42546
Sombrero	Maraca	87.89772
African elephant	African elephant	53.53317
Polecat	Polecat	91.77765
Chambered nautilus	Chambered nautilus	99.59608

Figure 11: A table showing the results of 15 benchmark Inception classification tests. (From left to right) The actual class of the input, the classification given by Inception, Inceptions' confidence in its classification.

Testing with 15 images from randomly selected classes, Inception displays an impressive top-1 classification accuracy of 86.67%, with 13 out of 15 correct classifications, and a median confidence of 96.45% (87.89% mean confidence). Even in difficult situations it was successful, for example in the African Elephant attempt, an intentionally tricky image was used but it managed to distinguish the difference between three species of elephant (African, Tusker, Indian) to correctly classify the image with majority confidence (over 50%). However, its' sometimes strange and nuanced behaviour is also apparent here. For example, one of the images supplies was of a man wearing a sombrero and holding maracas. It was intended that Inception would classify it as 'sombrero', but instead it classified it as 'maraca', with a significant difference of 87.9% to 10.9%. If both objects are present in the image, why does it so confidently disregard one? This exposes the rudimentary nature of CNNs as we know them, and their inability to classify multiple object in the same image with high confidence. Furthermore, in the 'Planetarium' classification attempt, 'Planetarium' was not even in the top 5 classifications (they were 'stage', 'home theatre', 'cinema', 'theatre curtain' and 'projector'). To a human, the image was clearly a planetarium by virtue of the large projection of a star system. It is feasible that the model instead

recognised the arrangement of seated people in the image, highlighting another issue; that common information in images of different classes can cause the model to severely misclassify.

Regarding adapting Inception for facial recognition, achieving the optimal accuracy in this task is not in the interest of this project, although it is important to consider that obtaining a significantly better accuracy is possible. Considering that the accuracy of a model positively correlates with the number of training samples it is given (to a limit), we can safely assume that providing more labelled training images will increase the accuracy of the model. Rather than hand pick images, pre-process and label them, a more efficient approach is *data augmentation*. This defines methods of rotating, shifting and flipping images that are already in the training set, so they are considered novel data by the model in training. Data augmentation methods can significantly increase the model accuracy, as well as offset the effects of having limited samples and uneven class distribution.

Aligned Input Image	Actual Class	Inception Classification	Inception Confidence (%)
	George W Bush	George W Bush	49.73
	Gerhard Schroeder	Gerhard Schroeder	54.97
	Colin Powell	Colin Powell	51.42

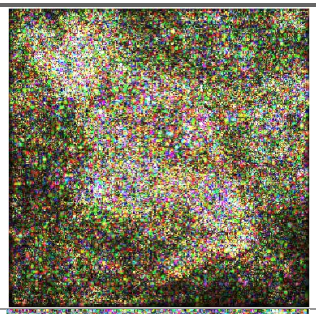
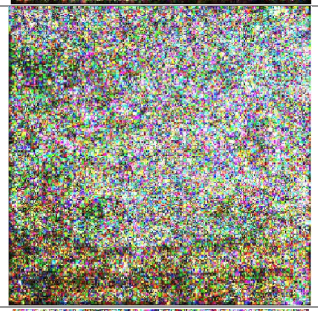
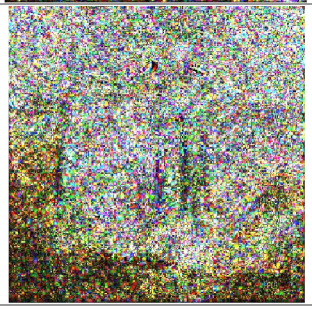
Figure 12: Three example tests of using Inception for facial recognition on the LFW dataset. (left to right) The image input to the model, the actual class of the input image, the classification made by Inception, Inceptions' confidence in its classification.

As stated in the previous chapter, Inception was able to recognise the individuals in the test set with 67% top-1 accuracy. In aid of visualising this, separate tests on 10 random images in the test set were made and are shown in Appendix D1, in which a top-1 accuracy of 80% was achieved (100% top-5) with a median confidence score of 84.61% (mean confidence of 70.64%). These tests echoed the expectations and concerns previously mentioned, for example, George W Bush was selected randomly for 5 out of 10 tests, appeared in all top-5 classifications, and was the most confident classification "by default" if the model was unconfident of a classification. This was expected as a large portion of the dataset was comprised of George W Bush images, and the solution is to achieve an even class distribution by utilising methods such as data augmentation.

Considering the accuracy and versatility of the Inception model, it was deceived by the adversarial network rather trivially. Appendix D2 outlines the adversarial tests that were undertaken immediately following from the original classification attempts at the beginning of this section. Inception made the expected misclassification in 100% of cases (15 out of 15) with a median confidence of 99.1% (mean confidence 92.9%). These confidence values can still be improved, as early stopping and step limiting were implemented here so optimal confidence may not have been reached if the loss was still converging.

However, this also exposed some concerns in the approach. For example, in the attempt to misclassify '*hand blower*', the classes '*sombrero*', '*poncho*' and '*maraca*' were also in the top 5 classifications and were all present in the original image. This may sound expected, but it represents unusual behaviour as this was not the case for any of the other tests, in which the top 5 classifications consisted of classes semantically similar to the target misclassification. A reason for this could be that it converged in only 9 steps, not giving enough time for the model to gain confidence in similar classes. Regardless, this could potentially pose a weakness in adversarial networks. If a model can be developed to ensure that the dominant classification is semantically similar to the next most confident classifications, could it defend itself from adversarial input?

Figure 13: Three examples of testing adversarial success. (L to R) The original scaled input image, the actual class of the original image, the adversarial noise added to the original image by the adversarial network, the adversarial image output by the adversarial network, the target class used as the adversarial networks target for Inception to classify the image as, Inceptions' classification of the adversarial image output by the adversarial network.

Original Image (scaled)	Original Image Classification (Confidence)	Adversarial Noise	Adversarial Image (Original Image + Noise)	Adversarial Target Classification	Inception Classification of Adversarial Image
	Mongoose (86.20%)			German short-haired pointer	German short-haired pointer (99.09%)
	Fountain (91.32%)			Oyster catcher	Oyster catcher (99.17%)
	Toaster (99.77%)			Sea anemone	Sea anemone (99.93%)

6. Conclusion

In conclusion, the ability to generalise Inception so it can successfully perform a new task is a robust demonstration of the transfer learning principles of machine learning. This provided some insight into the mechanisms that the model uses to classify images, notably that low-level inter-class differences such as colours and shapes are handled in the initial layers of the network. Generated adversarial examples were able to make Inception disregard it's prior learning to the deepest level, as it was now unable to recognise the true shapes and colours, but rather classified the input as whichever target class was expected by the adversarial network.

Successful applications of adversarial attacks on the Inception model present significant security concerns. Inception is a state-of-the-art model, publicly available and widely implemented with derivative models spawning from its' concept, though it still suffers from adversarial attacks the same any Deep Neural Network. Defence mechanisms have been theorised, but none satisfy the robustness necessary to defend against all types of adversarial attack, which makes this a pervasive issue to be heavily considered by researchers, consumers and developers alike.

6.1. Professional, Ethical, Social, Security and Legal Issues

With machine learning methods making a significant resurgence in today's consumer markets, such vulnerabilities pose security risks not only for the consumer, but the technology provider also. Deep Neural Networks are used or will be used in almost every commercial domain. Consider autonomous vehicles, where the repercussions are severe if a misclassification is made, and even more severe if an intentional misclassification is forced by a malicious attacker. Similarly, in the bioinformatics domain, malicious involvement with a classification could cause a misdiagnosis with unintentional consequences. Furthermore, DNNs are often tasked with the protection of critical personal information as they are implemented in many anti-malware applications. This is a more significant concern as it is possible for such an adversarial attack to be realised in the very near future, and it stands to reason this is an enormous security flaw with any anti-malware software using DNNs if an attacker can manipulate adversarial examples in such a way.

Legal and ethical issues are derivative of the security concerns, in the sense that they would spawn in the case of a successful adversarial attack in the public domain. If this happened, it would be incredibly difficult to place any legal responsibility on any one party. It is evident that providers and consumers alike are equally unaware of such a vulnerability, and whilst it can be considered unethical to provide a service with such a vulnerability without warning, it can be assumed that the consumer has responsibility of enlightening themselves with the possible associated risks of the service before using it.

In the event of a successfully robust defence method appearing, technology that is already in the market will need to be rebuilt with these defence methods in mind. If not, then this presents even more legal and ethical issues for the provider of the technology. Therefore, it is important for providers to be aware of the issue, and if they are willing to still provide the technology, they must be made aware of such defence methods with immediate effect. If a successful attack were to happen after the fact, this would be entirely the legal responsibility of the provider and could carry substantial consequences.

Professional issues can be considered as any of the previously mentioned issues where the provider is in the role of responsibility. Professionals must be aware of the risks of certain methods as they are developing and releasing them. It is conceivable that the risks of using DNNs without defending has somewhat encumbered the progress of machine learning in the consumer markets, but with good reason, it is a professional's duty to provide services that are at least slightly robust to their perceived vulnerabilities.

6.2. Further Research

Though there have been several attempts into defending from adversarial attacks already, it is important that research continues as more robust approaches are appearing. It is important to ask; if adversarial input can render a DNN almost entirely ineffective, can we still consider DNNs to function as intended? A robust method of defending a model from adversarial input must appear, ideally before consumer markets are flooded with deep neural network implementations.

7. References

[1] Fukushima, Kunihiko. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." *Biological cybernetics* 36, no. 4 (1980): 193-202.

[2] Waibel, Alexander, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. "Phoneme recognition using time-delay neural networks." *Backpropagation: Theory, Architectures and Applications* (1995): 35-61.

[3] LeCun, Yann, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. "Backpropagation applied to handwritten zip code recognition." *Neural computation* 1, no. 4 (1989): 541-551.

[4] Chellapilla, Kumar, Sidd Puri, and Patrice Simard. "High performance convolutional neural networks for document processing." In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[5] Cireşan, Dan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification." *arXiv preprint arXiv:1202.2745* (2012).

[6] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[7] Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818-2826. 2016.

[8] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.

[9] Szegedy, Christian, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning." In *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[10] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

[11] Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815-823. 2015.

[12] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *Advances in neural information processing systems*, pp. 2672-2680. 2014.

[13] Sun, Yi, Xiaogang Wang, and Xiaoou Tang. "Deeply learned face representations are sparse, selective, and robust." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2892-2900. 2015.

[14] Weinberger, Kilian Q., and Lawrence K. Saul. "Distance metric learning for large margin nearest neighbor classification." *Journal of Machine Learning Research* 10, no. Feb (2009): 207-244.

[15] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In *European conference on computer vision*, pp. 818-833. Springer, Cham, 2014.

[16] Huang, Gary B., Marwan Mattar, Tamara Berg, and Eric Learned-Miller. "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments." In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*. 2008.

[17] Huang, Gary, Marwan Mattar, Honglak Lee, and Erik G. Learned-Miller. "Learning to align from scratch." In *Advances in neural information processing systems*, pp. 764-772. 2012.

[18] Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).

[19] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*(2014).

[20] Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning." In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 506-519. ACM, 2017.

[21] Carlini, Nicholas, and David Wagner. "Towards evaluating the robustness of neural networks." In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39-57. IEEE, 2017.

[22] Papernot, Nicolas, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. "Distillation as a defense to adversarial perturbations against deep neural networks." In *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582-597. IEEE, 2016.

[23] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." *arXiv preprint arXiv:1503.02531* (2015).

[24] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

[25] Papernot, Nicolas, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. "Towards the science of security and privacy in machine learning." *arXiv preprint arXiv:1611.03814*(2016).

[26] Hendrycks, Dan, and Kevin Gimpel. "Early methods for detecting adversarial images." *arXiv preprint arXiv:1608.00530* (2016).

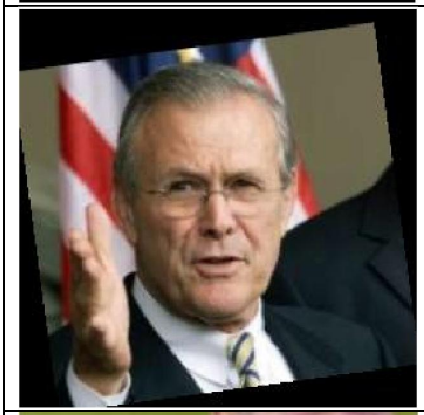
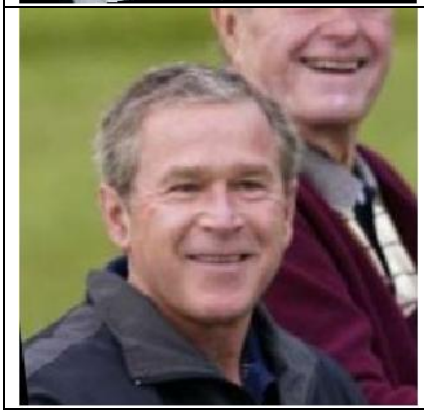
[27] Meng, Dongyu, and Hao Chen. "Magnet: a two-pronged defense against adversarial examples." In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135-147. ACM, 2017.

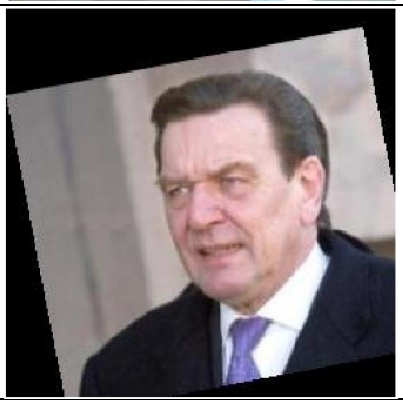
8. Appendices

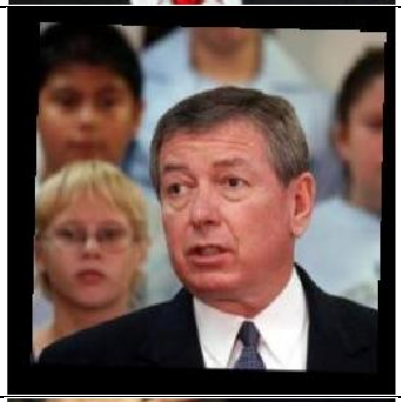
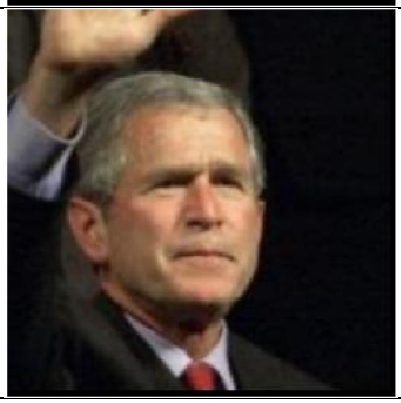
D. Testing

D1. Facial Recognition Tests

Figure 14: Ten test examples of Inception being used for facial recognition. (L to R) The input image from the LFW deep funnelled dataset, the actual class of the image (who the person is), Inceptions' top 5 classifications for the image (who Inception thinks the image is).

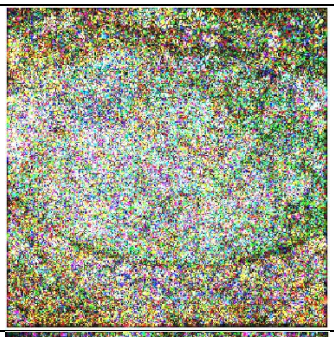
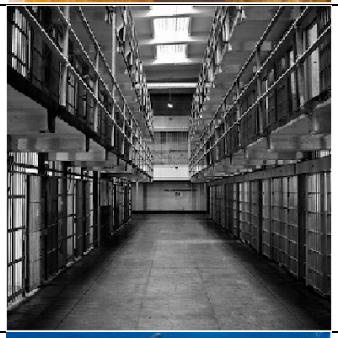
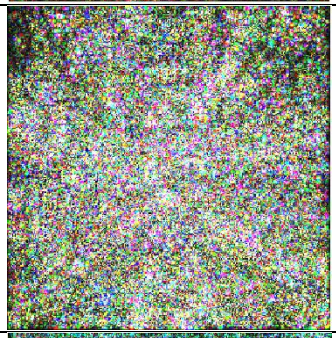
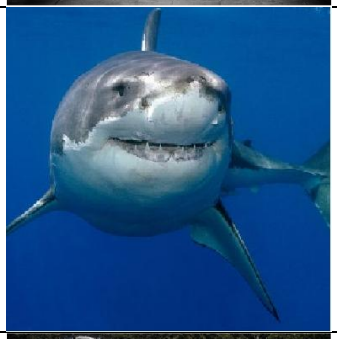
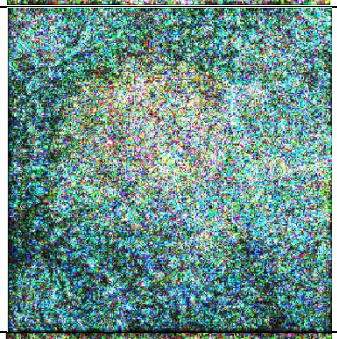
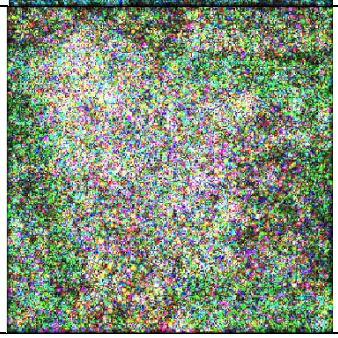
Face Image	Actual Class	Classification (Top-5)
	Hugo Chavez	**Hugo Chavez (94.65%)** Gerhard Schroeder (1.58%) George W Bush (1.26%) Tony Blair (0.74%) Colin Powell (0.70%)
	Donald Rumsfeld	**Donald Rumsfeld (63.63%)** Gerhard Schroeder (13.72%) Colin Powell (7.2%) George W Bush (6.32%) Ariel Sharon (2.88%)
	George W Bush	**George W Bush (90.87%)** Tony Blair (6.41%) Hugo Chavez (2.28%) Jean Chretien (0.29%) Gerhard Schroeder (0.14%)

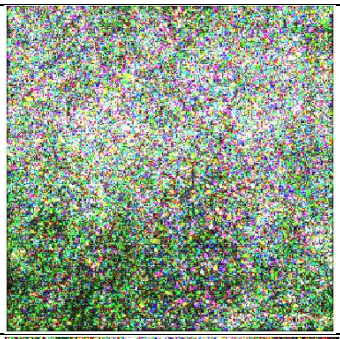
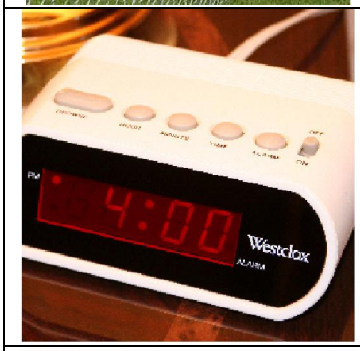
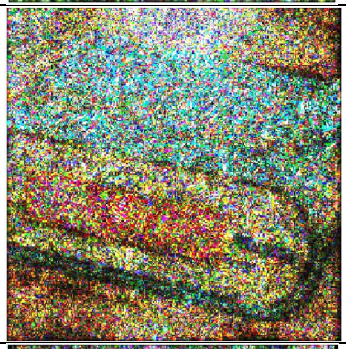
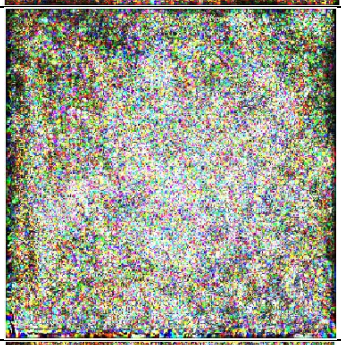
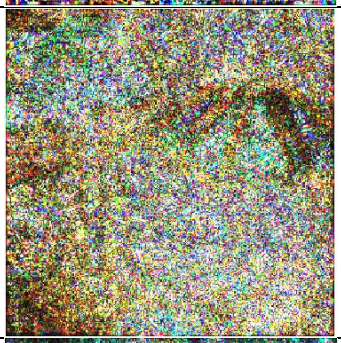
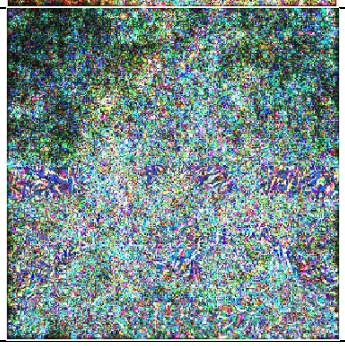
	Tony Blair	**Tony Blair (78.35%)** George W Bush (14.58%) Gerhard Schroeder (2.14%) John Ashcroft (1.71%) Colin Powell (1.31%)
	George W Bush	**George W Bush (95.09%)** Tony Blair (1.75%) Gerhard Schroeder (0.92%) Colin Powell (0.69%) Donald Rumsfeld (0.60%)
	George W Bush	**George W Bush (95.68%)** Tony Blair (2.54%) Hugo Chavez (0.82%) Gerhard Schroeder (0.54%) Colin Powell (0.17%)
	Gerhard Schroeder	George W Bush (33.29%) Colin Powell (29.03%) **Gerhard Schroeder (23.56%)** Tony Blair (9.00%) Jean Chretien (1.65%)

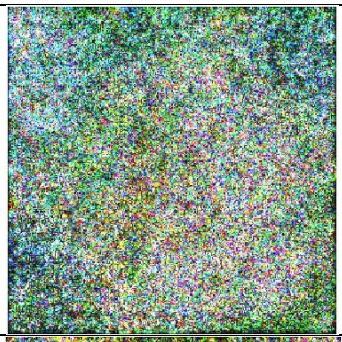
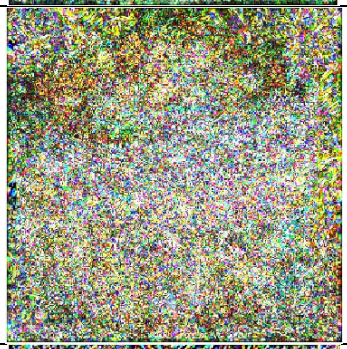
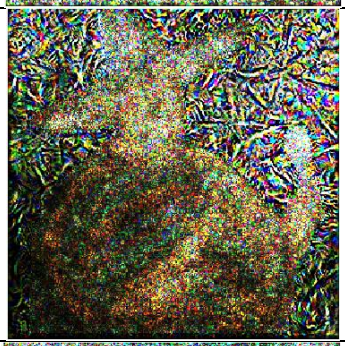
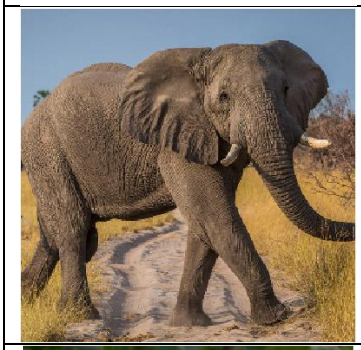
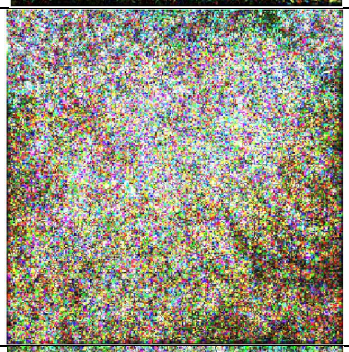
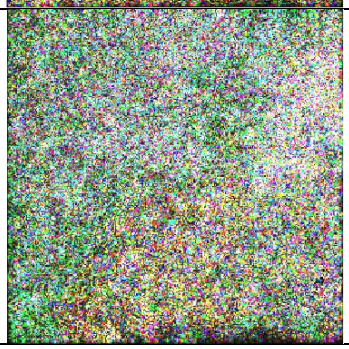
	George W Bush	**George W Bush (65.68%)** Tony Blair (15.83%) Gerhard Schroeder (5.81%) Jean Chretien (5.46%) Colin Powell (2.74%)
	John Ashcroft	George W Bush (54.23%) Tony Blair (21.55%) **John Ashcroft (7.30%)** Colin Powell (7.16%) Junichiro Koizumi (7.04%)
	George W Bush	**George W Bush (91.59%)** Tony Blair (2.22%) Colin Powell (1.84%) Ariel Sharon (1.25%) Hugo Chavez (0.79%)

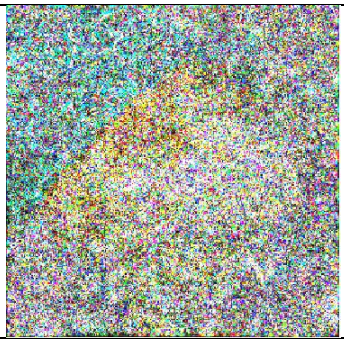
D2. Adversarial Tests

Figure 15: Fifteen test examples of tricking Inception with adversarial input. (L to R) The input image, the adversarial noise added to input by the adversarial network, the target classification to determine adversarial success, Inceptions' classification of the adversarial image, Inceptions' confidence in its classification.

Original Input (scaled)	Adversarial Noise	Target Classification	Inception Classification	Confidence (%)
		Stupa, tope	Stupa, tope	93.33
		Miniature poodle	Miniature poodle	98.03
		Damselfly	Damselfly	99.13
		Hoop skirt, crinoline	Hoop skirt, crinoline	95.60

		Espresso maker	Espresso maker	96.40
		Bernese mountain dog	Bernese mountain dog	99.13
		Bighorn, bighorn sheep	Bighorn, bighorn sheep	16.79
		Pomeranian	Pomeranian	98.68
		Bikini, two-piece	Bikini, two-piece	99.27

		Safe	Safe	99.40
		Long-horned beetle	Long-horned beetle	99.75
		Hand blower, blow dryer	Hand blower, blow dryer	99.45
		Wire-haired fox terrier	Wire-haired fox terrier	99.15
		Viaduct	Viaduct	99.30

		Revolver, six-gun	Revolver, six-gun	99.87

E. Source Code

E1. Installation Instructions

This project was developed using:
Python 3.7.2, pip 10.0.1, TensorFlow 1.12.0, SciPy 1.2.0, Pillow 5.4.1, matplotlib 3.0.2, numpy 1.14.5 and other dependencies that come with these versions.

The following commands should be run in the Python virtual environment terminal:

pip install tensorflow
pip install scipy
pip install pillow
pip install matplotlib

Note: TensorFlow will only work on 64-bit versions of Python, therefore will only work on 64-bit operating systems.

E2. Generalisation Script

```
import tensorflow as tf
import numpy as np
import os
import matplotlib.pyplot as plt
import math
import random
import time
from datetime import datetime
from tensorflow.contrib.slim.nets import inception
from scipy.misc import imresize
from collections import Counter
from PIL import Image
from models.research.slim.datasets import dataset_utils
slim = tf.contrib.slim

train = 0.70
validation = 0.05
test = 0.25

def split_data(X, y, train=train, test=test, validation=validation):
    X = np.array(X)
    X_raw_test = []
    X_raw_valid = []
    X_raw_train = []
    y = np.array(y)
    y_raw_test = []
    y_raw_valid = []
    y_raw_train = []

    random_indices = np.random.permutation(len(X))
    X = X[random_indices]
    y = y[random_indices]

    for image, label in zip(X, y):
        test_length = math.floor(test * class_images[label])
        valid_length = math.floor(validation * class_images[label])
```

```
        if Counter(y_raw_test)[label] < test_length:
            X_raw_test.append(image)
            y_raw_test.append(label)
        elif Counter(y_raw_valid)[label] < valid_length:
            X_raw_valid.append(image)
            y_raw_valid.append(label)
        else:
            X_raw_train.append(image)
            y_raw_train.append(label)

    return np.array(X_raw_train, dtype=np.float32), \
           np.array(X_raw_valid, dtype=np.float32), \
           np.array(X_raw_test, dtype=np.float32), \
           np.array(y_raw_train, dtype=np.int32), \
           np.array(y_raw_valid, dtype=np.int32), \
           np.array(y_raw_test, dtype=np.int32)

def plot_color_image(image):
    plt.figure(figsize=(4,4))
    plt.imshow(image.astype(np.uint8), interpolation='nearest')
    plt.axis('off')

def preprocess(image):
    image = imresize(image, (299, 299)) / 255
    return image

def batch(X, y, start_index=0, batch_size=4):
    stop_index = start_index + batch_size
    prepared_images = []
    labels = []

    for index in range(start_index, stop_index):
        prepared_images.append(preprocess(X[index]))
        labels.append(y[index])

    X_batch = np.stack(prepared_images)
    y_batch = np.array(labels, dtype=np.int32)

    return X_batch, y_batch

if not tf.gfile.Exists("models/cnn"):
    tf.gfile.MakeDirs("models/cnn")

if not os.path.exists("models/cnn/inception_v3.ckpt"):

dataset_utils.download_and_uncompress_tarball("http://download.tensorflow.org/model
s/inception_v3_2016_08_28.tar.gz", "models/cnn")

if not os.path.exists("tmp/faces"):
    os.makedirs("tmp/faces")

if not os.path.exists("tmp/faces/lfw-deepfunneled.tgz"):
    dataset_utils.download_and_uncompress_tarball("http://vis-
www.cs.umass.edu/lfw/lfw-deepfunneled.tgz", "tmp/faces")

class_mapping = {}
class_images = {}
dir = enumerate(os.listdir("images/faces/"))

for index, directory in dir:
    a = directory.split(" ")
    class_mapping[index] = a[0]
    path = os.path.join("images/faces/", a[0])
    class_images[index] = len([f for f in os.listdir(path)])
print(class_mapping)

image_arrays = []
image_labels = []
```

```
root_image_directory = "images/faces/"
for label, person in class_mapping.items():
    for directory in os.listdir(root_image_directory):
        if directory == person:
            image_directory = root_image_directory + directory
            break

    for image in os.listdir(image_directory):
        image = plt.imread(os.path.join(image_directory, image))
        image_arrays.append(image)
        image_labels.append(label)
image_arrays = np.array(image_arrays)
image_labels = np.array(image_labels)


X = np.array(image_arrays)
X_raw_test = []
X_raw_valid = []
X_raw_train = []
y = np.array(image_labels)
y_raw_test = []
y_raw_valid = []
y_raw_train = []

random_indices = np.random.permutation(len(X))
X = X[random_indices]
y = y[random_indices]

for image, label in zip(X, y):
    test_length = math.floor(test * class_images[label])
    valid_length = math.floor(validation * class_images[label])

    if Counter(y_raw_test)[label] < test_length:
        X_raw_test.append(image)
        y_raw_test.append(label)
    elif Counter(y_raw_valid)[label] < valid_length:
        X_raw_valid.append(image)
        y_raw_valid.append(label)
    else:
        X_raw_train.append(image)
        y_raw_train.append(label)

X_train = np.array(X_raw_train, dtype=np.float32)
X_valid = np.array(X_raw_valid, dtype=np.float32)
X_test = np.array(X_raw_test, dtype=np.float32)
y_train = np.array(y_raw_train, dtype=np.int32)
y_valid = np.array(y_raw_valid, dtype=np.int32)
y_test = np.array(y_raw_test, dtype=np.int32)

for class_number, person in class_mapping.items():
    print(f"{class_number}. {person}: {class_images[class_number]}")


tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, 299, 299, 3], name='X')
is_training = tf.placeholder_with_default(False, [])

with slim.arg_scope(inception.inception_v3_arg_scope()):
    logits, end_points = inception.inception_v3(X, num_classes=1001,
is_training=is_training)

inception_saver = tf.train.Saver()

prelogits = tf.squeeze(end_points['PreLogits'], axis=[1, 2])
n_outputs = len(class_mapping)

people_logits = tf.layers.dense(prelogits, n_outputs, name="people_logits")
```

```
probability = tf.nn.softmax(people_logits, name='probability')

y = tf.placeholder(tf.int32, None)

loss =
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=people_logits,
labels=y))
train_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
scope="people_logits")
training_op = tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss,
var_list=train_vars)

correct = tf.nn.in_top_k(predictions=people_logits, targets=y, k=1)
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
saver = tf.train.Saver()

X_valid, y_valid = batch(X_valid, y_valid, 0, len(X_valid))

with tf.name_scope("tensorboard"):
    valid_acc_summary = tf.summary.scalar(name='valid_acc', tensor=accuracy)
    valid_loss_summary = tf.summary.scalar(name='valid_loss', tensor=loss)
    train_acc_summary = tf.summary.scalar(name='train_acc', tensor=accuracy)
    valid_merged_summary = tf.summary.merge(inputs=[valid_acc_summary,
valid_loss_summary])

    n_epochs = 100
    batch_size = 50
    max_checks_without_progress = 10
    checks_without_progress = 0
    best_loss = np.float("inf")
    show_progress = 1
    n_iterations_per_epoch = len(X_train) // batch_size
    now = datetime.now().strftime("%Y%m%d_%H%M%S")
    model_dir = f"{now}_unaugmented"
    logdir = "tensorboard/faces/" + model_dir
    file_writer = tf.summary.FileWriter(logdir=logdir,
graph=tf.get_default_graph())
    inception_v3_checkpoint_path = "models/cnn/inception_v3.ckpt"
    unaugmented_training_path = "models/cnn/inception_v3_faces_unaugmented.ckpt"

with tf.Session() as sess:
    init.run()
    inception_saver.restore(sess, inception_v3_checkpoint_path)

    t0 = time.time()
    for epoch in range(n_epochs):
        start_index = 0

        for iteration in range(n_iterations_per_epoch):
            X_batch, y_batch = batch(X_train, y_train, start_index, batch_size)
            sess.run(training_op, {X: X_batch, y: y_batch})
            start_index += batch_size

        if epoch % show_progress == 0:
            train_summary = sess.run(train_acc_summary, {X: X_batch, y: y_batch})
            file_writer.add_summary(train_summary, (epoch + 1))
            valid_loss, valid_acc, valid_summary = sess.run([loss, accuracy,
valid_merged_summary], {X: X_valid, y: y_valid})
            file_writer.add_summary(valid_summary, (epoch + 1))
            print(f"Epoch: {epoch+1} Loss: {valid_loss} Accuracy: {valid_acc}")

        if valid_loss < best_loss:
            best_loss = valid_loss
            checks_without_progess = 0
            save_path = saver.save(sess, unaugmented_training_path)
        else:
```

```
            checks_without_progress += 1

        if checks_without_progress > max_checks_without_progress:
            print(f"Stopping Early. Loss has not improved in
{max_checks_without_progress} epochs.")
            break

    t1 = time.time()

print('Training Time: {:.2f} minutes'.format((t1 - t0) / 60))

eval_batch_size = 32
n_iterations = len(X_test) // eval_batch_size
with tf.Session() as sess:
    saver.restore(sess, unaugmented_training_path)

    start_index = 0
    test_accuracy = {}

    t0 = time.time()
    for i in range(n_iterations):
        X_test_batch, y_test_batch = batch(X_test, y_test, start_index,
batch_size=eval_batch_size)
        test_accuracy[i] = accuracy.eval({X: X_test_batch, y: y_test_batch})
        start_index += eval_batch_size
        print('Iteration: {} Batch Testing Accuracy: {:.2f}%'.format(i + 1,
test_accuracy[i] * 100))

t1 = time.time()

print('\nFinal Testing Accuracy: {:.4f}% on {}
instances.'.format(np.mean(list(test_accuracy.values())) * 100, len(X_test)))
print('Total evaluation time: {:.4f} seconds'.format((t1 - t0)))

def classify_image(index, images=X_test, labels=y_test):
    image_array = images[index]
    label = class_mapping[labels[index]]

    prepared_image = preprocess(image_array)
    prepared_image = np.reshape(prepared_image, newshape=(-1, 299, 299, 3))

    predictions = sess.run(probability, {X: prepared_image})

    predictions = [(i, prediction) for i, prediction in enumerate(predictions[0])]
    predictions = sorted(predictions, key=lambda x: x[1], reverse=True)
    print('\nCorrect Answer: {}'.format(label))
    print('\nPredictions:')
    for prediction in predictions:
        class_label = prediction[0]
        probability_value = prediction[1]
        label = class_mapping[class_label]
        print("{:26}: {:.2f}%".format(label, probability_value * 100))

    plot_color_image(image_array)
    return predictions

with tf.Session() as sess:
    init.run()
    saver.restore(sess, "models/cnn/inception_v3_faces_unaugmented.ckpt")
    for i in range(10):
        classify_image(random.randint(1, 360))
```

E3. Adversarial Script

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import tensorflow.contrib.slim as slim
import tensorflow.contrib.slim.nets as nets
import tempfile
from urllib.request import urlretrieve
import tarfile
import os
import json
import PIL
from random import randint

numTopProbs = 5
target_class = randint(0, 999)

def inception(image, reuse):
    img_expanded = tf.expand_dims(image, 0)
    img_subtracted = tf.subtract(img_expanded, 0.5)
    img_preprocessed = tf.multiply(img_subtracted, 2.0)
    arg_scope = nets.inception.inception_v3_arg_scope(weight_decay=0.0)
    with slim.arg_scope(arg_scope):
        logits, _ = nets.inception.inception_v3(img_preprocessed, 1001,
is_training=False, reuse=reuse)
        logits = logits[:,1:]
        probs = tf.nn.softmax(logits)
    return logits, probs

def classify(img, title):
    fig, (ax2, ax1) = plt.subplots(1, 2, figsize=(11, 5))
    p = sess.run(probs, feed_dict={image: img})[0]
    ax1.title.set_text(title)
    ax1.imshow(img)
    ax1.axis("off")

    topk = list(p.argsort()[-numTopProbs:][::-1])
    topprobs = p[topk]

    barlist = ax2.barh(range(numTopProbs), topprobs)
    ax2.set_yticks(range(numTopProbs))
    ax2.set_yticklabels([imagenet_labels[i][:15] for i in topk])
    ax2.invert_yaxis()
    ax2.set_xlabel("Classification Confidence")

    labels = []
    for bar in barlist:
        width = bar.get_width()
        if (width < 0.5):
            x = width + 0.02
            weight = 'normal'
            colour = 'grey'
            align = 'left'
        else:
            x = width * 0.98
            weight = 'bold'
            colour = 'white'
            align = 'right'

        y = bar.get_y() + (bar.get_height() / 2)
        label = ax2.text(x, y, "{0:.5%}".format(width), horizontalalignment=align,
                         verticalalignment='center', color=colour, weight=weight,
clip_on=True)
        labels.append(label)
```

```
    print(f"Classification: {imagenet_labels[topk[0]]}")
    print("Confidence: {0:.10%}\n".format(topprobs[0]))

    plt.show()

def adv_plot_noise(diff):
    min = diff.min()
    max = diff.max()

    norm = (diff - min) / (max - min)

    plt.imshow(norm)
    plt.title("Adversarial Noise")
    plt.axis("off")
    plt.show()

imageUrl = "http://fergusontalon.com/wp-content/uploads/2018/12/image1-9.jpg"
labels_resource = 'imagenet_labels.json'
inceptionUrl =
'http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz'

data_dir = 'models/cnn'
checkpointPath = 'models/cnn/inception_v3.ckpt'

print("Creating Tensorflow session...")
sess = tf.InteractiveSession()
print("Tensorflow session created.\n")
image = tf.Variable(tf.zeros((299,299,3)))

print("Retrieving logits and probabilities from Inception model...")
logits, probs = inception(image, reuse=False)
print("Logits and probabilities retrieved.\n")

if (os.path.exists(checkpointPath) == False):
    print(f"Inception v3 checkpoint does not exist in path: {checkpointPath}")
    print(f"Retrieving Inception v3 from URL: {inceptionUrl}")
    inception_tarball, _ = urlretrieve(inceptionUrl)
    tarfile.open(inception_tarball, 'r:gz').extractall(data_dir)
    print(f"Inception model extracted to path: {data_dir}\n")

print("Restoring Inception variables...")
vars = [var for var in tf.global_variables() if
var.name.startswith('InceptionV3/')]
print("Inception variables restored.\n")
saver = tf.train.Saver(vars)
print("Restoring Inception checkpoint...")
saver.restore(sess, checkpointPath)
print("Inception checkpoint restored.\n")

print(f"Loading ImageNet labels from resource: {labels_resource}")
with open(labels_resource) as f:
    imagenet_labels = json.load(f)
print("ImageNet labels loaded.\n")

print(f"Loading input image from URL: {imageUrl}")

img_path, _ = urlretrieve(imageUrl)
print("Image loaded.")

img = PIL.Image.open(img_path)
wide = img.width > img.height
new_w = 299 if not wide else int(img.width * 299 / img.height)
new_h = 299 if wide else int(img.height * 299 / img.width)
print(f"Image is {img.width}x{img.height}... resizing to {new_w}x{new_h}.\n")
img = img.resize((new_w, new_h))
img = img.crop((np.floor(new_w/2 - 149), 0, np.floor(new_w/2 + 150), 299)) \
    if wide else img.crop((0, np.floor(new_h/2 - 149), 299, np.floor(new_h/2 +
150)))
```

```
img = (np.asarray(img) / 255.0).astype(np.float32)

print("Classifying image...")
classify(img, "Original Scaled Input Image")


print("Starting Adversarial process...\n")
lr = 0.1
epsilon = 8e-3
steps = 50
print(f"Learning Rate: {lr}")
print(f"Epsilon: {epsilon}")
print(f"Maximum Steps: {steps}")
print(f"Target Class: {target_class} - {imagenet_labels[target_class]}\n")



input_placeholder = tf.placeholder(tf.float32, (299, 299, 3))
input_image = image
assign_input_to_placeholder = tf.assign(input_image, input_placeholder)

learning_rate_placeholder = tf.placeholder(tf.float32, ())
classification_label_placeholder = tf.placeholder(tf.int32, ())

labels = tf.one_hot(classification_label_placeholder, 1000)
loss = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=[labels])
optim_step =
tf.train.GradientDescentOptimizer(learning_rate_placeholder).minimize(loss,
var_list=[input_image])

epsilon_ph = tf.placeholder(tf.float32, ())
below = input_placeholder - epsilon_ph
above = input_placeholder + epsilon_ph

projected = tf.clip_by_value(tf.clip_by_value(input_image, below, above), 0, 1)
with tf.control_dependencies([projected]):
    project_step = tf.assign(input_image, projected)

sess.run(assign_input_to_placeholder, feed_dict={input_placeholder: img})

last_loss = 10

for i in range(steps):
    if(last_loss > 0.01):
        _, loss_value = sess.run(
            [optim_step, loss],
            feed_dict={learning_rate_placeholder: lr,
                       classification_label_placeholder: target_class})

        last_loss = loss_value

        grad = sess.run(project_step, feed_dict={input_placeholder: img,
epsilon_ph: epsilon})

        print(f"Step {i+1}, Loss = {loss_value}")
    else:
        print(f"Stopping early - minimal loss (classification confidence above 99%)
reached at step {i}\n")
        break

adv = input_image.eval()
diff = np.abs(np.subtract(img, adv))
adv_plot_noise(diff)

print("\nClassifying perturbed image with adversarial noise...")
print(f"ADVERSARIAL ATTACK HAS SUCCEEDED IF CLASSIFICATION IS:
{imagenet_labels[target_class]}")
```

```
classify(adv, "Original Image + Adversarial Noise")



ex_angle = np.pi/8

print("Rotating input image...")
rotation = tf.placeholder(tf.float32, ())
rotated_image = tf.contrib.image.rotate(image, rotation)
rotated_example = rotated_image.eval(feed_dict={image: adv, rotation: ex_angle})
print("Classifying rotated input image...")
classify(rotated_example, "Rotated Adversarial Image")

num_samples = 30
average_loss = 0

print("Training model on rotated input images...")
for i in range(num_samples):
    rotated = tf.contrib.image.rotate(image, tf.random_uniform((), minval=-np.pi/4,
maxval=np.pi/4))
    rotated_logits, _ = inception(rotated, reuse=True)
    average_loss +=
tf.nn.softmax_cross_entropy_with_logits_v2(logits=rotated_logits, labels=labels) /
num_samples
    print(f"Rotated sample {i+1}/{num_samples} trained.")

optim_step =
tf.train.GradientDescentOptimizer(learning_rate_placeholder).minimize(average_loss,
var_list=[input_image])
print("Model trained and optimisation step updated.\n")



epsilon = 3e-2
lr = 0.2
steps = 50

print("Starting adversarial process for rotated samples...")

sess.run(assign_input_to_placeholder, feed_dict={input_placeholder: img})

last_loss = 10
for i in range(steps):
    if (last_loss > 0.01):
        _, loss_value = sess.run(
            [optim_step, average_loss],
            feed_dict={learning_rate_placeholder: lr,
classification_label_placeholder: target_class})
        sess.run(project_step, feed_dict={input_placeholder: img, epsilon_ph:
epsilon})

        last_loss = loss_value
        print(f"Step {i + 1}, Loss = {loss_value}")
    else:
        print(f"Stopping early - minimal loss (classification confidence above 99%)
reached at step {i}")
        break

adv_robust = input_image.eval()

print("\nClassifying rotated image with adversarial noise...")
print(f"ADVERSARIAL ATTACK HAS SUCCEEDED IF CLASSIFICATION IS:
{imagenet_labels[target_class]}")
rotated_example = rotated_image.eval(feed_dict={image: adv_robust, rotation:
ex_angle})
```

```
classify(rotated_example, "Rotated Original Image + Adversarial Noise")


print("\nBeginning robustness evaluation...")
angles = np.linspace(-np.pi / 4, np.pi / 4, 301)

probs_naive = []
probs_robust = []
for angle in angles:
    print(f"Completing rotation of {angle}")
    rotated = rotated_image.eval(feed_dict={image: adv_robust, rotation: angle})
    probs_robust.append(probs.eval(feed_dict={image: rotated})[0][target_class])

    rotated = rotated_image.eval(feed_dict={image: adv, rotation: angle})
    probs_naive.append(probs.eval(feed_dict={image: rotated})[0][target_class])

print("\nPlotting robustness evaluation...")
print("Plot should show that the robust adversarial model performs well with
rotation invariance.")
robust_line, = plt.plot(angles, probs_robust, color='g', linewidth=2, label='Robust
Model')
naive_line, = plt.plot(angles, probs_naive, color='b', linewidth=2, label='Naive
Model')
plt.ylim([0, 1.05])
plt.xlabel('Angle of Rotation')
plt.ylabel('Probability of Adversarial Success')
plt.legend(handles=[robust_line, naive_line], loc='lower right')
plt.show()
```

E4. ImageNet Classes JSON


```
[
  "tench, Tinca tinca",
  "goldfish, Carassius auratus",
  "great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias",
  "tiger shark, Galeocerdo cuvieri",
  "hammerhead, hammerhead shark",
  "electric ray, crampfish, numbfish, torpedo",
  "stingray",
  "cock",
  "hen",
  "ostrich, Struthio camelus",
  "brambling, Fringilla montifringilla",
  "goldfinch, Carduelis carduelis",
  "house finch, linnet, Carpodacus mexicanus",
  "junco, snowbird",
  "indigo bunting, indigo finch, indigo bird, Passerina cyanea",
  "robin, American robin, Turdus migratorius",
  "bulbul",
  "jay",
  "magpie",
  "chickadee",
  "water ouzel, dipper",
  "kite",
  "bald eagle, American eagle, Haliaeetus leucocephalus",
  "vulture",
  "great grey owl, great gray owl, Strix nebulosa",
  "European fire salamander, Salamandra salamandra",
  "common newt, Triturus vulgaris",
  "eft",
  "spotted salamander, Ambystoma maculatum",
  "axolotl, mud puppy, Ambystoma mexicanum",
  "bullfrog, Rana catesbeiana",
  "tree frog, tree-frog",
  "tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui",
  "loggerhead, loggerhead turtle, Caretta caretta",
  "leatherback turtle, leatherback, leathery turtle, Dermochelys coriacea",
  "mud turtle",
  "terrapin",
  "box turtle, box tortoise",
  "banded gecko",
  "common iguana, iguana, Iguana iguana",
  "American chameleon, anole, Anolis carolinensis",
  "whiptail, whiptail lizard",
  "agama",
  "frilled lizard, Chlamydosaurus kingi",
  "alligator lizard",
  "Gila monster, Heloderma suspectum",
  "green lizard, Lacerta viridis",
  "African chameleon, Chamaeleo chamaeleon",
  "Komodo dragon, Komodo lizard, dragon lizard, giant lizard, Varanus komodoensis",
  "African crocodile, Nile crocodile, Crocodylus niloticus",
  "American alligator, Alligator mississipiensis",
  "triceratops",
  "thunder snake, worm snake, Carphophis amoenus",
  "ringneck snake, ring-necked snake, ring snake",
  "hognose snake, puff adder, sand viper",
  "green snake, grass snake",
  "king snake, kingsnake",
  "garter snake, grass snake",
  "water snake",
  "vine snake",
  "night snake, Hypsiglena torquata",
  "boa constrictor, Constrictor constrictor",
  "rock python, rock snake, Python sebae",
  "Indian cobra, Naja naja",
  "green mamba",
  "sea snake",
  "horned viper, cerastes, sand viper, horned asp, Cerastes cornutus",
  "diamondback, diamondback rattlesnake, Crotalus adamanteus",
  "sidewinder, horned rattlesnake, Crotalus cerastes",
  "trilobite",
  "harvestman, daddy longlegs, Phalangium opilio",
  "scorpion",
  "black and gold garden spider, Argiope aurantia",
  "barn spider, Araneus cavaticus",
  "garden spider, Aranea diademata",
  "black widow, Latrodectus mactans",
  "tarantula",
  "wolf spider, hunting spider",
  "tick",
  "centipede",
  "black grouse",
  "ptarmigan",
  "ruffed grouse, partridge, Bonasa umbellus",
  "prairie chicken, prairie grouse, prairie fowl",
  "peacock",
  "quail",
  "partridge",
  "African grey, African gray, Psittacus erithacus",
  "macaw",
  "sulphur-crested cockatoo, Kakatoe galerita, Cacatua galerita",
  "lorikeet",
  "coucal",
  "bee eater",
  "hornbill",
  "hummingbird",
  "jacamar",
  "toucan",
  "drake",
  "red-breasted merganser, Mergus serrator",
  "goose",
  "black swan, Cygnus atratus",
  "tusker",
  "echidna, spiny anteater, anteater",
  "platypus, duckbill, duckbilled platypus, duck-billed platypus, Ornithorhynchus anatinus",
  "wallaby, brush kangaroo",
  "koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus",
  "wombat",
  "jellyfish",
  "sea anemone, anemone",
  "brain coral",
  "flatworm, platyhelminth",
  "nematode, nematode worm, roundworm",
  "conch",
  "snail",
  "slug",
  "sea slug, nudibranch",
  "chiton, coat-of-mail shell, sea cradle, polyplacophore",
  "chambered nautilus, pearly nautilus, nautilus",
  "Dungeness crab, Cancer magister",
  "rock crab, Cancer irroratus",
  "fiddler crab",
  "king crab, Alaska crab, Alaskan king crab, Alaska king crab, Paralithodes camtschatica",
  "American lobster, Northern lobster, Maine lobster, Homarus americanus",
  "spiny lobster, langouste, rock lobster, crawfish, crayfish, sea crawfish",
  "crayfish, crawfish, crawdad, crawdaddy",
  "hermit crab",
  "isopod",
  "white stork, Ciconia ciconia",
  "black stork, Ciconia nigra",
  "spoonbill",
  "flamingo",
  "little blue heron, Egretta caerulea",
  "American egret, great white heron, Egretta albus",
  "bittern",
  "crane",
  "limpkin, Aramus pictus",
  "European gallinule, Porphyrio porphyrio",
  "American coot, marsh hen, mud hen, water hen, Fulica americana",
  "bustard",
  "ruddy turnstone, Arenaria interpres",
  "red-backed sandpiper, dunlin, Erolia alpina",
  "redshank, Tringa totanus",
  "dowitcher",
  "oystercatcher, oyster catcher",
  "pelican",
  "king penguin, Aptenodytes patagonica",
  "albatross, mollymawk",
  "grey whale, gray whale, devilfish, Eschrichtius gibbosus, Eschrichtius robustus",
  "killer whale, killer, orca, grampus, sea wolf, Orcinus orca",
  "dugong, Dugong dugon",
  "sea lion",
  "Chihuahua",
  "Japanese spaniel",
  "Maltese dog, Maltese terrier, Maltese",
  "Pekinese, Pekingese, Peke",
  "Shih-Tzu",
  "Blenheim spaniel",
  "papillon",
  "toy terrier",
  "Rhodesian ridgeback",
  "Afghan hound, Afghan",
  "basset, basset hound",
  "beagle",
  "bloodhound, sleuthhound",
  "bluetick",
  "black-and-tan coonhound",
  "Walker hound, Walker foxhound",
  "English foxhound",
  "redbone",
  "borzoi, Russian wolfhound",
  "Irish wolfhound",
  "Italian greyhound",
  "whippet",
  "Ibizan hound, Ibizan Podenco",
  "Norwegian elkhound, elkhound",
  "otterhound, otter hound",
  "Saluki, gazelle hound",
  "Scottish deerhound, deerhound",
  "Weimaraner",
  "Staffordshire bullterrier, Staffordshire bull terrier",
  "American Staffordshire terrier, Staffordshire terrier, American pit bull terrier, pit bull terrier",
  "Bedlington terrier",
  "Border terrier",
  "Kerry blue terrier",
  "Irish terrier",
  "Norfolk terrier",
  "Norwich terrier",
  "Yorkshire terrier",
  "wire-haired fox terrier",
  "Lakeland terrier",
  "Sealyham terrier, Sealyham",
  "Airedale, Airedale terrier",
  "cairn, cairn terrier",
  "Australian terrier",
  "Dandie Dinmont, Dandie Dinmont terrier",
  "Boston bull, Boston terrier",
  "miniature schnauzer",
  "giant schnauzer",
  "standard schnauzer",
  "Scotch terrier, Scottish terrier, Scottie",
  "Tibetan terrier, chrysanthemum dog",
  "silky terrier, Sydney silky",
  "soft-coated wheaten terrier",
  "West Highland white terrier",
  "Lhasa, Lhasa apso",
  "flat-coated retriever",
  "curly-coated retriever",
  "golden retriever",
  "Labrador retriever",
  "Chesapeake Bay retriever",
  "German short-haired pointer",
  "vizsla, Hungarian pointer",
  "English setter",
  "Irish setter, red setter",
  "Gordon setter",
  "Brittany spaniel",
  "clumber, clumber spaniel",
  "English springer, English springer spaniel",
  "Welsh springer spaniel",
  "cocker spaniel, English cocker spaniel, cocker",
  "Sussex spaniel",
  "Irish water spaniel",
  "kuvasz",
  "schipperke",
  "groenendael",
  "malinois",
  "briard",
  "kelpie",
  "komondor",
  "Old English sheepdog, bobtail",
  "Shetland sheepdog, Shetland sheep dog, Shetland",
  "collie",
  "Border collie",
  "Bouvier des Flandres, Bouviers des Flandres",
  "Rottweiler",
  "German shepherd, German shepherd dog, German police dog, alsatian",
  "Doberman, Doberman pinscher",
  "miniature pinscher",
```

"Greater Swiss Mountain dog",
"Bernese mountain dog",
"Appenzeller",
"EntleBucher",
"boxer",
"bull mastiff",
"Tibetan mastiff",
"French bulldog",
"Great Dane",
"Saint Bernard, St Bernard",
"Eskimo dog, husky",
"malamute, malemute, Alaskan malamute",
"Siberian husky",
"dalmatian, coach dog, carriage dog",
"affenpinscher, monkey pinscher, monkey dog",
"basenji",
"pug, pug-dog",
"Leonberg",
"Newfoundland, Newfoundland dog",
"Great Pyrenees",
"Samoyed, Samoyede",
"Pomeranian",
"chow, chow chow",
"keeshond",
"Brabancon griffon",
"Pembroke, Pembroke Welsh corgi",
"Cardigan, Cardigan Welsh corgi",
"toy poodle",
"miniature poodle",
"standard poodle",
"Mexican hairless",
"timber wolf, grey wolf, gray wolf, Canis lupus",
"white wolf, Arctic wolf, Canis lupus tundrarum",
"red wolf, maned wolf, Canis rufus, Canis niger",
"coyote, prairie wolf, brush wolf, Canis latrans",
"dingo, warrigal, warragal, Canis dingo",
"dhole, Cuon alpinus",
"African hunting dog, hyena dog, Cape hunting dog, Lycaon pictus",
"hyena, hyaena",
"red fox, Vulpes vulpes",
"kit fox, Vulpes macrotis",
"Arctic fox, white fox, Alopex lagopus",
"grey fox, gray fox, Urocyon cinereoargenteus",
"tabby, tabby cat",
"tiger cat",
"Persian cat",
"Siamese cat, Siamese",
"Egyptian cat",
"cougar, puma, catamount, mountain lion, painter, panther, Felis concolor",
"lynx, catamount",
"leopard, Panthera pardus",
"snow leopard, ounce, Panthera uncia",
"jaguar, panther, Panthera onca, Felis onca",
"lion, king of beasts, Panthera leo",
"tiger, Panthera tigris",
"cheetah, chetah, Acinonyx jubatus",
"brown bear, bruin, Ursus arctos",
"American black bear, black bear, Ursus americanus, Euarctos americanus",
"ice bear, polar bear, Ursus Maritimus, Thalarctos maritimus",
"sloth bear, Melursus ursinus, Ursus ursinus",
"mongoose",
"meerkat, mierkat",
"tiger beetle",
"ladybug, ladybeetle, lady beetle, ladybird, ladybird beetle",
"ground beetle, carabid beetle",
"long-horned beetle, longicorn, longicorn beetle",
"leaf beetle, chrysomelid",
"dung beetle",
"rhinoceros beetle",
"weevil",
"fly",
"bee",
"ant, emmet, pismire",
"grasshopper, hopper",
"cricket",
"walking stick, walkingstick, stick insect",
"cockroach, roach",
"mantis, mantid",
"cicada, cicala",
"leafhopper",
"lacewing, lacewing fly",
"dragonfly, darning needle, devil's darning needle, sewing needle, snake feeder, snake doctor, mosquito hawk, skeeter hawk",
"damselfly",
"admiral",
"ringlet, ringlet butterfly",
"monarch, monarch butterfly, milkweed butterfly, Danaus plexippus",
"cabbage butterfly",
"sulphur butterfly, sulfur butterfly",
"lycaenid, lycaenid butterfly",
"starfish, sea star",
"sea urchin",
"sea cucumber, holothurian",
"wood rabbit, cottontail, cottontail rabbit",
"hare",
"Angora, Angora rabbit",
"hamster",
"porcupine, hedgehog",
"fox squirrel, eastern fox squirrel, Sciurus niger",
"marmot",
"beaver",
"guinea pig, Cavia cobaya",
"sorrel",
"zebra",
"hog, pig, grunter, squealer, Sus scrofa",
"wild boar, boar, Sus scrofa",
"warthog",
"hippopotamus, hippo, river horse, Hippopotamus amphibius",
"ox",
"water buffalo, water ox, Asiatic buffalo, Bubalus bubalis",
"bison",
"ram, tup",
"bighorn, bighorn sheep, cimarron, Rocky Mountain bighorn, Rocky Mountain sheep, Ovis canadensis",
"ibex, Capra ibex",
"hartebeest",
"impala, Aepyceros melampus",
"gazelle",
"Arabian camel, dromedary, Camelus dromedarius",
"llama",
"weasel",
"mink",
"polecat, fitch, foulmart, foumart, Mustela putorius",
"black-footed ferret, ferret, Mustela nigripes",
"otter",
"skunk, polecat, wood pussy",
"badger",
"armadillo",
"three-toed sloth, ai, Bradypus tridactylus",
"orangutan, orang, orangutang, Pongo pygmaeus",
"gorilla, Gorilla gorilla",
"chimpanzee, chimp, Pan troglodytes",
"gibbon, Hylobates lar",
"siamang, Hylobates syndactylus, Symphalangus syndactylus",
"guenon, guenon monkey",
"patas, hussar monkey, Erythrocebus patas",
"baboon",
"macaque",
"langur",
"colobus, colobus monkey",
"proboscis monkey, Nasalis larvatus",
"marmoset",
"capuchin, ringtail, Cebus capucinus",
"howler monkey, howler",
"titi, titi monkey",
"spider monkey, Ateles geoffroyi",
"squirrel monkey, Saimiri sciureus",
"Madagascar cat, ring-tailed lemur, Lemur catta",
"indri, indris, Indri indri, Indri brevicaudatus",
"Indian elephant, Elephas maximus",
"African elephant, Loxodonta africana",
"lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens",
"giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca",
"barracouta, snoek",
"eel",
"coho, cohoe, coho salmon, blue jack, silver salmon, Oncorhynchus kisutch",
"rock beauty, Holocanthus tricolor",
"anemone fish",
"sturgeon",
"gar, garfish, garpike, billfish, Lepisosteus osseus",
"lionfish",
"puffer, pufferfish, blowfish, globefish",
"abacus",
"abaya",
"academic gown, academic robe, judge's robe",
"accordion, piano accordion, squeeze box",
"acoustic guitar",
"aircraft carrier, carrier, flattop, attack aircraft carrier",
"airliner",
"airship, dirigible",
"altar",
"ambulance",
"amphibian, amphibious vehicle",
"analog clock",
"apiary, bee house",
"apron",
"ashcan, trash can, garbage can, wastebin, ash bin, ash-bin, ashbin, dustbin, trash barrel, trash bin",
"assault rifle, assault gun",
"backpack, back pack, knapsack, packsack, rucksack, haversack",
"bakery, bakeshop, bakehouse",
"balance beam, beam",
"balloon",
"ballpoint, ballpoint pen, ballpen, Biro",
"Band Aid",
"banjo",
"bannister, banister, balustrade, balusters, handrail",
"barbell",
"barber chair",
"barbershop",
"barn",
"barometer",
"barrel, cask",
"barrow, garden cart, lawn cart, wheelbarrow",
"baseball",
"basketball",
"bassinet",
"bassoon",
"bathing cap, swimming cap",
"bath towel",
"bathtub, bathing tub, bath, tub",
"beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon",
"beacon, lighthouse, beacon light, pharos",
"beaker",
"bearskin, busby, shako",
"beer bottle",
"beer glass",
"bell cote, bell cot",
"bib",
"bicycle-built-for-two, tandem bicycle, tandem",
"bikini, two-piece",
"binder, ring-binder",
"binoculars, field glasses, opera glasses",
"birdhouse",
"boathouse",
"bobsled, bobsleigh, bob",
"bolo tie, bolo, bola tie, bola",
"bonnet, poke bonnet",
"bookcase",
"bookshop, bookstore, bookstall",
"bottlecap",
"bow",
"bow tie, bow-tie, bowtie",
"brass, memorial tablet, plaque",
"brassiere, bra, bandeau",
"breakwater, groin, groyne, mole, bulwark, seawall, jetty",
"breastplate, aegis, egis",
"broom",
"bucket, pail",
"buckle",
"bulletproof vest",
"bullet train, bullet",
"butcher shop, meat market",
"cab, hack, taxi, taxicab",
"caldron, cauldron",
"candle, taper, wax light",
"cannon",
"canoe",
"can opener, tin opener",
"cardigan",
"car mirror",
"carousel, carrousel, merry-go-round, roundabout, whirligig",
"carpenter's kit, tool kit",
"carton",
"car wheel",
"cash machine, cash dispenser, automated teller machine, automatic teller machine, automated teller, automatic teller, ATM",
"cassette",
"cassette player",
"castle",
"catamaran",
"CD player",
"cello, violoncello",
"cellular telephone, cellular phone, cellphone, cell, mobile phone",
"chain",
"chainlink fence",

"chain mail, ring mail, mail, chain armor, chain armour, ring armor, ring armour",
"chain saw, chainsaw",
"chest",
"chiffonier, commode",
"chime, bell, gong",
"china cabinet, china closet",
"Christmas stocking",
"church, church building",
"cinema, movie theater, movie theatre, movie house, picture palace",
"cleaver, meat cleaver, chopper",
"cliff dwelling",
"cloak",
"clog, geta, patten, sabot",
"cocktail shaker",
"coffee mug",
"coffeepot",
"coil, spiral, volute, whorl, helix",
"combination lock",
"computer keyboard, keypad",
"confectionery, confectionary, candy store",
"container ship, containership, container vessel",
"convertible",
"corkscrew, bottle screw",
"cornet, horn, trumpet, trump",
"cowboy boot",
"cowboy hat, ten-gallon hat",
"cradle",
"crane",
"crash helmet",
"crate",
"crib, cot",
"Crock Pot",
"croquet ball",
"crutch",
"cuirass",
"dam, dike, dyke",
"desk",
"desktop computer",
"dial telephone, dial phone",
"diaper, nappy, napkin",
"digital clock",
"digital watch",
"dining table, board",
"dishrag, dishcloth",
"dishwasher, dish washer, dishwashing machine",
"disk brake, disc brake",
"dock, dockage, docking facility",
"dogsled, dog sled, dog sleigh",
"dome",
"doormat, welcome mat",
"drilling platform, offshore rig",
"drum, membranophone, tympan",
"drumstick",
"dumbbell",
"Dutch oven",
"electric fan, blower",
"electric guitar",
"electric locomotive",
"entertainment center",
"envelope",
"espresso maker",
"face powder",
"feather boa, boa",
"file, file cabinet, filing cabinet",
"fireboat",
"fire engine, fire truck",
"fire screen, fireguard",
"flagpole, flagstaff",
"flute, transverse flute",
"folding chair",
"football helmet",
"forklift",
"fountain",
"fountain pen",
"four-poster",
"freight car",
"French horn, horn",
"frying pan, frypan, skillet",
"fur coat",
"garbage truck, dustcart",
"gasmask, respirator, gas helmet",
"gas pump, gasoline pump, petrol pump, island dispenser",
"goblet",
"go-kart",
"golf ball",
"golfcart, golf cart",
"gondola",
"gong, tam-tam",
"gown",
"grand piano, grand",
"greenhouse, nursery, glasshouse",
"grille, radiator grille",
"grocery store, grocery, food market, market",
"guillotine",
"hair slide",
"hair spray",
"half track",
"hammer",
"hamper",
"hand blower, blow dryer, blow drier, hair dryer, hair drier",
"hand-held computer, hand-held microcomputer",
"handkerchief, hankie, hanky, hankey",
"hard disc, hard disk, fixed disk",
"harmonica, mouth organ, harp, mouth harp",
"harp",
"harvester, reaper",
"hatchet",
"holster",
"home theater, home theatre",
"honeycomb",
"hook, claw",
"hoopskirt, crinoline",
"horizontal bar, high bar",
"horse cart, horse-cart",
"hourglass",
"iPod",
"iron, smoothing iron",
"jack-o'-lantern",
"jean, blue jean, denim",
"jeep, landrover",
"jersey, T-shirt, tee shirt",
"jigsaw puzzle",
"jinrikisha, ricksha, rickshaw",
"joystick",
"kimono",
"knee pad",
"knot",
"lab coat, laboratory coat",
"ladle",
"lampshade, lamp shade",
"laptop, laptop computer",
"lawn mower, mower",
"lens cap, lens cover",
"letter opener, paper knife, paperknife",
"library",
"lifeboat",
"lighter, light, igniter, ignitor",
"limousine, limo",
"liner, ocean liner",
"lipstick, lip rouge",
"Loafer",
"lotion",
"loudspeaker, speaker, speaker unit, loudspeaker system, speaker system",
"loupe, jeweler's loupe",
"lumbermill, sawmill",
"magnetic compass",
"mailbag, postbag",
"mailbox, letter box",
"maillot",
"maillot, tank suit",
"manhole cover",
"maraca",
"marimba, xylophone",
"mask",
"matchstick",
"maypole",
"maze, labyrinth",
"measuring cup",
"medicine chest, medicine cabinet",
"megalith, megalithic structure",
"microphone, mike",
"microwave, microwave oven",
"military uniform",
"milk can",
"minibus",
"miniskirt, mini",
"minivan",
"missile",
"mitten",
"mixing bowl",
"mobile home, manufactured home",
"Model T",
"modem",
"monastery",
"monitor",
"moped",
"mortar",
"mortarboard",
"mosque",
"mosquito net",
"motor scooter, scooter",
"mountain bike, all-terrain bike, off-roader",
"mountain tent",
"mouse, computer mouse",
"mousetrap",
"moving van",
"muzzle",
"nail",
"neck brace",
"necklace",
"nipple",
"notebook, notebook computer",
"obelisk",
"oboe, hautboy, hautbois",
"ocarina, sweet potato",
"odometer, hodometer, mileometer, milometer",
"oil filter",
"organ, pipe organ",
"oscilloscope, scope, cathode-ray oscilloscope, CRO",
"overskirt",
"oxcart",
"oxygen mask",
"packet",
"paddle, boat paddle",
"paddlewheel, paddle wheel",
"padlock",
"paintbrush",
"pajama, pyjama, pj's, jammies",
"palace",
"panpipe, pandean pipe, syrinx",
"paper towel",
"parachute, chute",
"parallel bars, bars",
"park bench",
"parking meter",
"passenger car, coach, carriage",
"patio, terrace",
"pay-phone, pay-station",
"pedestal, plinth, footstall",
"pencil box, pencil case",
"pencil sharpener",
"perfume, essence",
"Petri dish",
"photocopier",
"pick, plectrum, plectron",
"pickelhaube",
"picket fence, paling",
"pickup, pickup truck",
"pier",
"piggy bank, penny bank",
"pill bottle",
"pillow",
"ping-pong ball",
"pinwheel",
"pirate, pirate ship",
"pitcher, ewer",
"plane, carpenter's plane, woodworking plane",
"planetarium",
"plastic bag",
"plate rack",
"plow, plough",
"plunger, plumber's helper",
"Polaroid camera, Polaroid Land camera",
"pole",
"police van, police wagon, paddy wagon, patrol wagon, wagon, black Maria",
"poncho",
"pool table, billiard table, snooker table",
"pop bottle, soda bottle",
"pot, flowerpot",
"potter's wheel",
"power drill",
"prayer rug, prayer mat",
"printer",
"prison, prison house",
"projectile, missile",
"projector",
"puck, hockey puck",
"punching bag, punch bag, punching ball, punchball",
"purse",
"quill, quill pen",
"quilt, comforter, comfort, puff",
"racer, race car, racing car",
"racket, racquet",
"radiator",
"radio, wireless",
"radio telescope, radio reflector",
"rain barrel",
"recreational vehicle, RV, R.V.",
"reel",
"reflex camera",
"refrigerator, icebox",
"remote control, remote",
"restaurant, eating house, eating place, eatery",
"revolver, six-gun, six-shooter",
"rifle",
"rocking chair, rocker",

```
  "rotisserie",
  "rubber eraser, rubber, pencil eraser",
  "rugby ball",
  "rule, ruler",
  "running shoe",
  "safe",
  "safety pin",
  "saltshaker, salt shaker",
  "sandal",
  "sarong",
  "sax, saxophone",
  "scabbard",
  "scale, weighing machine",
  "school bus",
  "schooner",
  "scoreboard",
  "screen, CRT screen",
  "screw",
  "screwdriver",
  "seat belt, seatbelt",
  "sewing machine",
  "shield, buckler",
  "shoe shop, shoe-shop, shoe store",
  "shoji",
  "shopping basket",
  "shopping cart",
  "shovel",
  "shower cap",
  "shower curtain",
  "ski",
  "ski mask",
  "sleeping bag",
  "slide rule, slipstick",
  "sliding door",
  "slot, one-armed bandit",
  "snorkel",
  "snowmobile",
  "snowplow, snowplough",
  "soap dispenser",
  "soccer ball",
  "sock",
  "solar dish, solar collector, solar furnace",
  "sombrero",
  "soup bowl",
  "space bar",
  "space heater",
  "space shuttle",
  "spatula",
  "speedboat",
  "spider web, spider's web",
  "spindle",
  "sports car, sport car",
  "spotlight, spot",
  "stage",
  "steam locomotive",
  "steel arch bridge",
  "steel drum",
  "stethoscope",
  "stole",
  "stone wall",
  "stopwatch, stop watch",
  "stove",
  "strainer",
  "streetcar, tram, tramcar, trolley, trolley car",
  "stretcher",
  "studio couch, day bed",
  "stupa, tope",
  "submarine, pigboat, sub, U-boat",
  "suit, suit of clothes",
  "sundial",
  "sunglass",
  "sunglasses, dark glasses, shades",
  "sunscreen, sunblock, sun blocker",
  "suspension bridge",
  "swab, swob, mop",
  "sweatshirt",
  "swimming trunks, bathing trunks",
  "swing",
  "switch, electric switch, electrical switch",
  "syringe",
  "table lamp",
  "tank, army tank, armored combat vehicle,
armoured combat vehicle",
  "tape player",
  "teapot",
  "teddy, teddy bear",
  "television, television system",
  "tennis ball",
  "thatch, thatched roof",
  "theater curtain, theatre curtain",
  "thimble",
  "thresher, thrasher, threshing machine",
  "throne",
  "tile roof",
  "toaster",
  "tobacco shop, tobacconist shop, tobacconist",
  "toilet seat",
  "torch",
  "totem pole",
  "tow truck, tow car, wrecker",
  "toyshop",
  "tractor",
  "trailer truck, tractor trailer, trucking rig, rig,
articulated lorry, semi",
  "tray",
  "trench coat",
  "tricycle, trike, velocipede",
  "trimaran",
  "tripod",
  "triumphal arch",
  "trolleybus, trolley coach, trackless trolley",
  "trombone",
  "tub, vat",
  "turnstile",
  "typewriter keyboard",
  "umbrella",
  "unicycle, monocycle",
  "upright, upright piano",
  "vacuum, vacuum cleaner",
  "vase",
  "vault",
  "velvet",
  "vending machine",
  "vestment",
  "viaduct",
  "violin, fiddle",
  "volleyball",
  "waffle iron",
  "wall clock",
  "wallet, billfold, notecase, pocketbook",
  "wardrobe, closet, press",
  "warplane, military plane",
  "washbasin, handbasin, washbowl, lavabo, wash-
hand basin",
  "washer, automatic washer, washing machine",
  "water bottle",
  "water jug",
  "water tower",
  "whiskey jug",
  "whistle",
  "wig",
  "window screen",
  "window shade",
  "Windsor tie",
  "wine bottle",
  "wing",
  "wok",
  "wooden spoon",
  "wool, woolen, woollen",
  "worm fence, snake fence, snake-rail fence,
Virginia fence",
  "wreck",
  "yawl",
  "yurt",
  "web site, website, internet site, site",
  "comic book",
  "crossword puzzle, crossword",
  "street sign",
  "traffic light, traffic signal, stoplight",
  "book jacket, dust cover, dust jacket, dust
wrapper",
  "menu",
  "plate",
  "guacamole",
  "consomme",
  "hot pot, hotpot",
  "trifle",
  "ice cream, icecream",
  "ice lolly, lolly, lollipop, popsicle",
  "French loaf",
  "bagel, beigel",
  "pretzel",
  "cheeseburger",
  "hotdog, hot dog, red hot",
  "mashed potato",
  "head cabbage",
  "broccoli",
  "cauliflower",
  "zucchini, courgette",
  "spaghetti squash",
  "acorn squash",
  "butternut squash",
  "cucumber, cuke",
  "artichoke, globe artichoke",
  "bell pepper",
  "cardoon",
  "mushroom",
  "Granny Smith",
  "strawberry",
  "orange",
  "lemon",
  "fig",
  "pineapple, ananas",
  "banana",
  "jackfruit, jak, jack",
  "custard apple",
  "pomegranate",
  "hay",
  "carbonara",
  "chocolate sauce, chocolate syrup",
  "dough",
  "meat loaf, meatloaf",
  "pizza, pizza pie",
  "potpie",
  "burrito",
  "red wine",
  "espresso",
  "cup",
  "eggnog",
  "alp",
  "bubble",
  "cliff, drop, drop-off",
  "coral reef",
  "geyser",
  "lakeside, lakeshore",
  "promontory, headland, head, foreland",
  "sandbar, sand bar",
  "seashore, coast, seacoast, sea-coast",
  "valley, vale",
  "volcano",
  "ballplayer, baseball player",
  "groom, bridegroom",
  "scuba diver",
  "rapeseed",
  "daisy",
  "yellow lady's slipper, yellow lady-slipper,
Cypripedium calceolus, Cypripedium parviflorum",
  "corn",
  "acorn",
  "hip, rose hip, rosehip",
  "buckeye, horse chestnut, conker",
  "coral fungus",
  "agaric",
  "gyromitra",
  "stinkhorn, carrion fungus",
  "earthstar",
  "hen-of-the-woods, hen of the woods, Polyporus
frondosus, Grifola frondosa",
  "bolete",
  "ear, spike, capitulum",
  "toilet tissue, toilet paper, bathroom tissue"
]
```