課題研究報告書

Adversarial Noise for CAPTCHA Solver Protection

井上 寛章

主指導教員 青木 利晃

北陸先端科学技術大学院大学
先端科学技術研究科
(情報科学)

令和6年9月

Abstract

The purpose of this study is to analyze the effectiveness, limitations, and problems of CAPTCHA images by providing a means of “defense” against “attacks” that attempt to use machine learning to recognize them. CAPTCHA stands for “Completely Automated Public Turing test to tell Computers and Humans Apart”. It is a turing test to distinguish between computers and humans, and is used to prevent spam attacks on Web sites.There are several types of CAPTCHAs, but this study focuses on “Pix,” which selects an object with a specified content from among photos.

Advances in image recognition technology have made it easier for computers to break through CAPTCHA tests. Therefore, it is desirable to have a method to make only image recognition models misrecognize images without affecting human vision, perception, and recognition systems. As a defense method, we tested several methods that are likely to inhibit the image recognition model, such as Adversarial Noise, Adversarial Patch, and searching for the weakest class.

And these experiments showed that the method of mixing multiple object classes is effective in inhibiting image recognition models. (178 words)

目 次

第1章　序論

1.1 研究背景と課題

CAPTCHA は「Completely Automated Public Turing test to tell Computers and Humans Apart」の頭文字で、コンピューターと人間を区別する公開チューリングテストである。

これは、Web サイトへのスパム攻撃を防ぐためなどに利用されている。画像認識 CAPTCHA テストの場合、ユーザーは分割された画像の中で指定された対象物（例：車、道路標識）が含まれた画像を選択することが求められる。この手法が用いられている理由は、人間は日常生活の中で様々な状況や背景の中で物を見分けることに慣れているためである。

しかしながら、画像認識技術の進歩によりコンピューターでの CAPTCHA テストの突破が容易になってきている。これに対して、難解な画像を利用して画像認識モデル（Bot, Solver）からの攻撃を防いでいるが人間でさえも判別が難しくなり本末転倒な状態が見受けられる（機械が人に合わせるのではなく、人が機械に合わせている状態）。

このため、人間の視覚・知覚・認識系には影響を与えず画像認識モデルにのみ誤認識させる手法が望まれる。その手法の一つとして Adversarial Noise（以降 AN）に注目する。AN は、人の目には微小なノイズや単なる幾何学パターンとしか見えないが、画像認識モデルには実際とは別のクラスに識別させる効果がある。画像認識モデルの認識を防ぐ限界のキャリアノイズ量は定量的には計測されていない。本研究では、これらを定量的に与える事を試みる。また、本 AN は特定の Solver をアタックする場合を示す。

CAPTCHA にはいくつか種類があり、代表的に次の 3 つがある。1) 指定され得た内容の写真を正しく選択読み取れたかで判別する『Pix』[1] [2]、2) 文字を重ねたり歪めたりしてそれを正しく読み取れるかで判別する『Gimpy』[3] [4]、3) 流れる音声を聞き取って判別する『Sounds』[5] が存在する。『Gimpy』は、OCR の技術向上によりその対策として人でも判別が難しい難易度になってしまったことや答える際にユーザーが文字を入力する手間が発生する。そして、『Sounds』は流れてくる音を聞くため聞き逃した際のやり直しや聞き取った内容を文字入力する手間が発生する。これらの観点から『Pix』の方が安全性や利便性が優れているため本研究テーマは『Pix』を取り上げる。

1.2　本研究の目的

本研究の目的は、CAPTCHA 画像を機械学習により認識させようとする”攻撃”に対して”防衛”する手段を与え、その効果と限界及び問題点を解析することにある。

1.3　本研研究の重要性と先行研究

CAPTCHA ”Pix”の画像認識精度の下限を下回る AN の特性と人の視覚・知覚・認識特性に基づくパターン認識精度の下限を下回るそれ（AN の特性）とはトレードオフの関係にある。

先行研究ではノイズに対する検知限を計量し、これを下回りながら AN の効果を得る事を検討している [6]。

本研究の新規性・重要性は人の検知限ではなく許容限を実験により与え、人が許容できる範囲内で機械認識器には破れない AN の量を決定して CAPTCHA を実現することにある。

CAPTCHA 画像は人が鑑賞するものではなく、プロテクトを解除するために一時的に使用するものであるから人の視覚・知覚・認識系の許容限までノイズ量を増やせる。このため、CAPTCHA Solver を破る CAPTCHA を実現する自由度（ノイズ量の選択範囲）を大きく取れる。

1.4 Solving CAPTCHA

CAPTCHA を自動的に解読しようとする攻撃手法として次の 2 つを取りあげる。

1.4.1 CAPTCHA Solver

以下のようなに CAPTCHA Solver を提供しているサービスが存在する。[7]

- 2Captcha[8]
- BestCaptchaSolver[9]
- AntiCaptcha[10]

1.4.2 ReCAPTCHABreaker

自動化されたシステムを使用して、デバイス上で Google ReCAPTCHA を人間に近い精度で破る最初のライブラリの 1 つである [11]。ViT と同じ（clip-vit-base-patch32 を利用して、学習データは CLIP や作者独自のデータセットを利用している）。

これらの攻撃手法には、以下のような認識モデルが用いられている。

1.4.3 ViT による Object Classification

概要

Transformer[A.3] が登場して以降、自然言語処理分野で BERT や GPT などの Transformer を元に高い性能のモデルが作成された。その一方で、コンピュータビジョン分野においては、Transformer ベースのモデルである ViT[12] が登場した。CNN は、画像のエッジやテクスチャ等の局所的な部分を捉えることが得意だが、遠く離れた物体同士の関連性や全体の構造を捉えることが得意ではない。そこで、ViT は Self-Attention を用いることで、CNN の欠点であった画像全体の関連性を捉えることに成功した認識モデルである。

アーキテクチャ

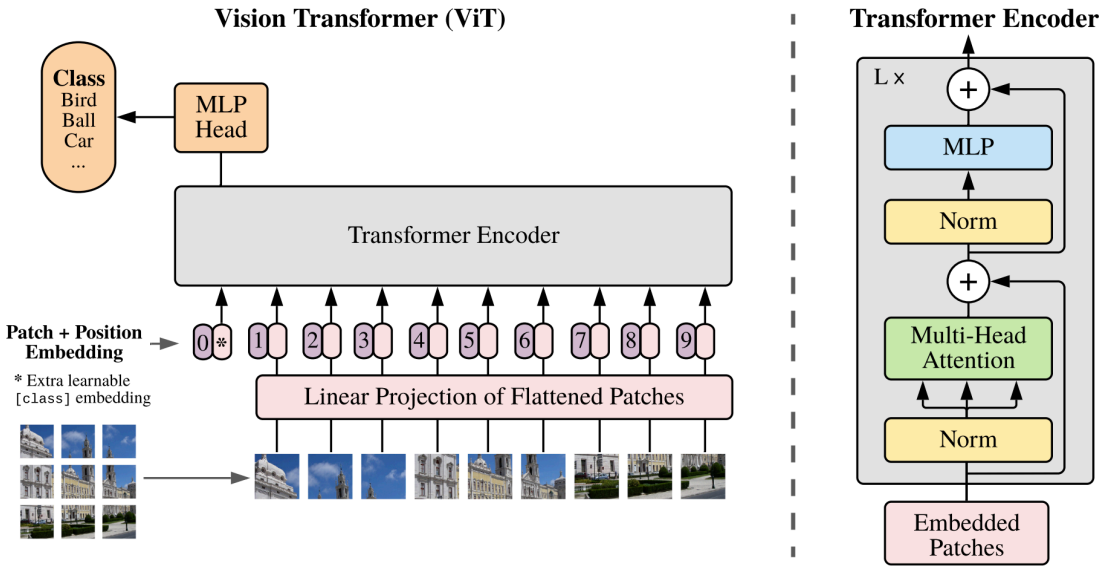


Fig. 1.1: ViT のアーキテクチャ

原理詳細

処理内容

- input：入力画像
- Input Layer：入力画像がパッチ (Patch) に分割され、クラストークンおよび各パッチに対応するベクトルが出力される
- Transformer Encoder：クラストークンが出力される。
- MLP Head：クラス分類器
- output：入力画像に対するラベル

Input Layer

1. パッチに分割

   - 入力画像 x をパッチ x_p に分割し flatten することで、各パッチをベクトルに変換する（※ベクトルの各要素は RGB の値を 0〜255 の範囲で表す整数値）

$$x \in \mathbb{R}^{H \times W \times C} \tag{1.1}$$

$$x_p \in \mathbb{R}^{N_p \times (P^2 * C)} \tag{1.2}$$

2. 埋め込み (Embedding)

   - １層の線形層で埋め込みを行う。埋め込んだ後のベクトルの長さを D とすると、

$$[x_p^1 E; x_p^2 E; ...; x_p^{N_p} E] \in \mathbb{R}^{N_p \times D} \tag{1.3}$$

   $E \in \mathbb{R}^{N_p \times (P^2 * C) \times D}$：線形層の重み

   $x_p^i \in \mathbb{R}^{N_p \times (P^2 * C)}$：$i$ 個目のパッチのベクトル

   $x_p^i E \in \mathbb{R}^D$：i 個目のパッチのベクトルを埋め込んだ長さのベクトル

   ;：パッチ方向での結合

3. クラストークン (Class Token)

   - 画像全体の情報を凝縮したベクトル。

$$[x_{class}; x_p^1 E; x_p^2 E; ...; x_p^{N_p} E] \in \mathbb{R}^{(N_p+1) \times D} \tag{1.4}$$

   $x_{class} \in \mathbb{R}^D$：長さ D のクラストークン

4. 位置埋め込み (Positional Embedding)

   - Self-Attention だけでは、パッチの位置情報を学ぶことができないので位置情報 (E_{pos}) を付与する。最終的な Encoder への入力を z_0 とすると、以下の数式で表現できる。

$$z_0 = [x_{class}; x_p^1 E; x_p^2 E; ...; x_p^{N_p} E] + E_{pos} \in \mathbb{R}^{(N_p+1) \times D} \tag{1.5}$$

Transformer Encoder

- LayerNorm（Layer Normalization）、MHSA（Multi-Head Self-Attention）、MLP の 3 つで構成されている
- Layer Normalization は、正則化手法の 1 つ。

$$LN(a)_i = \gamma_i \frac{a_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta_i \tag{1.6}$$

$a \in \mathbb{R}$：あるデータ
μ：平均
σ：標準偏差
ϵ：微小な定数
β_i, γ_i：a の i 番目の要素に対する学習可能なパラメータ

- Multi-Head Self-Attention は、Transformer の箇所を参照
- MLP は、2 層の線形層を用いて構成されている。正確には、Liner → GELU → Dropout → Liner → Dropout の構成をしている。

したがって、Encoder Block を数式で以下のように表現できる

$$z'_l = MHSA(LN(z_{l-1})) + z_{l-1} \tag{1.7}$$

$$z' = MLP(LN(z'_l)) + z'_l \tag{1.8}$$

MLP Head

クラス分類を行う分類器。LayerNorm および線形層の 2 つので構成されている。

$$y = LN(z_L^0)W^y \tag{1.9}$$

クラストークン：$z_L^0 \in \mathbb{R}^D$
線形層の重み：$W^y \in \mathbb{R}^{D \times M}$
M：クラス数

1.4.4 YOLO による Object Detection

概要

YOLO[13] は、Joseph Redmon が“You Only Look Once: Unified, Real-Time Object Detection”で発表した認識モデルである。このモデルは、CNN を用いた完全畳み込みネットワーク（Fully Convolutional Network）で構成されている。

アーキテクチャ

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig. 1.2: Darknet-53

53 層の CNN から構成されているため、Darknet-53 と呼ばれている。

原理詳細

処理内容

- input：入力画像
- Input Layer：入力画像がパッチ (Patch) に分割され、クラストークンおよび各パッチに対応するベクトルが出力される
- アンカーを使用した検出: 事前定義されたアンカーボックスを使用して物体のバウンディングボックスを予測する。各グリッドセルは、複数のアンカーを使用して異なるサイズとアスペクト比の物体を捉える。
- マルチスケール検出: 3 つの異なるスケールで予測を行い、小さな物体から大きな物体まで幅広く検出する。
- output：入力画像に対するラベル, 位置

バウンディングボックス

各グリッドセルでアンカーボックスを使用して物体のバウンディングボックスを予測する。バウンディングボックスの座標予測は、以下のように表される。

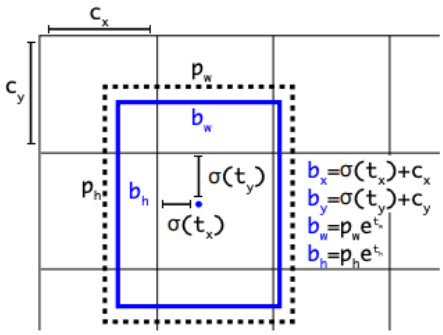


Fig. 1.3: バウンディングボックス

$$b_x = \sigma(t_x) + c_x \tag{1.10}$$

$$b_y = \sigma(t_y) + c_y \tag{1.11}$$

$$b_w = p_w e^{t_w} \tag{1.12}$$

$$b_h = p_h e^{t_h} \tag{1.13}$$

(b_x, b_y)、b_w、b_h：バウンディングボックスの中心座標、幅、高さ
t_x、t_y、t_w、t_h：ネットワークの出力
c_x、c_y：グリッドセルのオフセット
p_w、p_h：アンカーボックスの幅、高さ

損失関数

損失関数は、以下の 3 つで構成されている。
1. 位置の損失 (Localization loss)：

$$L_{loc} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(b_{x_i} - \hat{b}_{x_i})^2 + (b_{y_i} - \hat{b}_{y_i})^2 + (b_{w_i} - \hat{b}_{w_i})^2 + (b_{h_i} - \hat{b}_{h_i})^2 \right] \tag{1.14}$$

2. 信頼度の損失 (Confidence loss)：

$$L_{conf} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \mathbb{1}_{ij}^{noobj} (\hat{C}_i - 0)^2 \right] \tag{1.15}$$

3. クラス予測の損失 (Class prediction loss)：

$$L_{class} = \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \tag{1.16}$$

最終的な損失関数 L は以下のように定義される：

$$L = L_{loc} + L_{conf} + L_{class} \tag{1.17}$$

1.4.5 DETR による Object Detection

概要

DETR (End-to-End Object Detection with Transformers)[14] は、Facebook（現：Meta）の研究チームが公開し、初めて Transformer を採用した物体検出モデルである。

アーキテクチャ

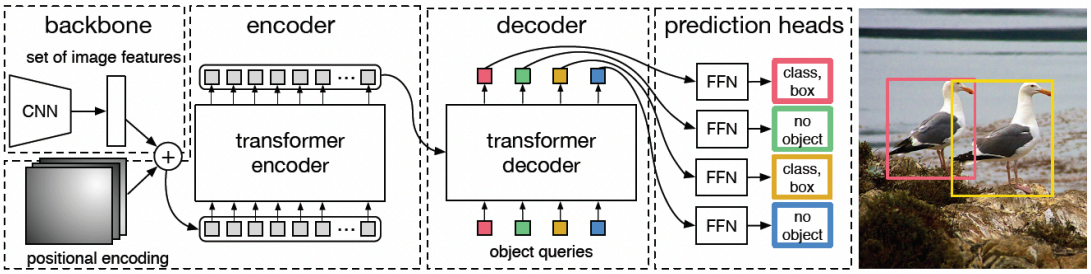


Fig. 1.4: DETR

原理詳細

処理内容

- input: 入力画像
- CNN Backbone: 画像の特徴を抽出するための CNN
- Transformer Encoder: CNN から得られた特徴マップをエンコードするトランスフォーマー
- Transformer Decoder: クエリを用いて物体の存在と位置をデコードするトランスフォーマー
- Prediction Heads: デコーダからの出力を分類と境界ボックスの予測に変換するヘッド
- output： 入力画像に対するラベル, 位置

CNN Backbone

- 入力と特徴抽出
  - 入力画像 $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ を CNN に通して特徴マップ $\mathbf{f} \in \mathbb{R}^{\frac{H}{32} \times \frac{W}{32} \times d}$ を得る。d は特徴マップのチャネル数。

$$\mathbf{f} = \text{CNN}(\mathbf{x}) \tag{1.18}$$

- 位置エンコーディング
  - $\mathbf{p} \in \mathbb{R}^{\frac{H}{32} \times \frac{W}{32} \times d}$ を特徴マップに加算

$$\mathbf{z} = \mathbf{f} + \mathbf{p} \tag{1.19}$$

Transformer Encoder

- 位置エンコーディング付き特徴マップ $\mathbf{z}$ をトランスフォーマーエンコーダに入力

$$\mathbf{h}_e = \text{TransformerEncoder}(\mathbf{z}) \tag{1.20}$$

Transformer Decoder

- 固定数のオブジェクトクエリ q をトランスフォーマーデコーダに入力

$$\mathbf{h}_d = \text{TransformerDecoder}(\mathbf{q}, \mathbf{h}_e) \tag{1.21}$$

Prediction Heads

- デコーダの出力 h_d を使って、クラス予測 c と境界ボックス予測 b を行う

$$\mathbf{c} = \text{Linear}_c(\mathbf{h}_d) \tag{1.22}$$

$$\mathbf{b} = \text{Linear}_b(\mathbf{h}_d) \tag{1.23}$$

損失関数

- 1. 分類損失（交差エントロピー損失）：

$$L_{cls} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{1.24}$$

- 2. 境界ボックス損失（L1 損失と IoU 損失の組み合わせ）：

$$L_{bbox} = \sum_{i=1}^{N} \|b_i - \hat{b}_i\|_1 + 1 - \text{IoU}(b_i, \hat{b}_i) \tag{1.25}$$

- 最終的な損失関数 L は以下のように定義される：

$$L = \lambda_{cls} L_{cls} + \lambda_{bbox} L_{bbox} \tag{1.26}$$

1.5 Adversarial Examples

Adversarial examples とは、機械学習モデルが誤った予測をするように設計された入力データのことである。これらの例は、人間には無害または意味があるように見えるが、モデルに対しては誤った結果を導く。

1.5.1 Adversarial Noise

概要

Adversarial Noise（敵対的ノイズ）[15] は、機械学習モデル、ディープラーニングモデルに対する攻撃手法の一つである。このノイズは、入力データに対して、わずかなノイズを加えることでモデルの正しい認識を阻害することを目的としている。これは、人間の目にはほとんど見えないほど小さな変化だが、モデルには大きな影響を与えることができる。

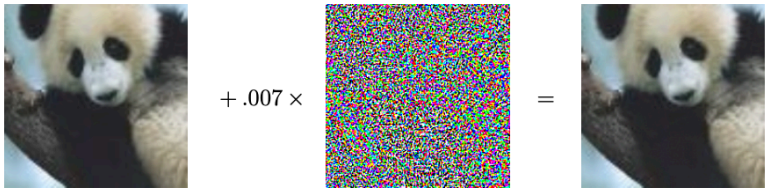


Fig. 1.5: Adversarial Noise

原理

FGSM (Fast Gradient Sign Method)

FGSM は、損失関数の勾配を利用して摂動を計算する手法であり、その摂動の計算は以下になる。

$$\eta = \epsilon \cdot \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y)) \tag{1.27}$$

ϵ：摂動の大きさを制御するスカラー（小さな値）
$\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y)$ ：入力 $\mathbf{x}$ に対する損失関数の勾配

sign(·) ：符号関数
摂動を元の入力に加えることで、Adversarial Example を生成します。

$$\mathbf{x}_{adv} = \mathbf{x} + \eta \tag{1.28}$$

PGD（Projected Gradient Descent）

PGD[16] は前述の FGSM の拡張であり、複数のステップを通じて摂動を生成することで、より強力な Adversarial Noise を作成する手法である。PGD は、反復的に勾配に基づいて摂動を更新し、その都度元の入力に投影することで摂動の大きさを制約する。

1. 初期摂動 η_0 を設定する（通常はランダム）
2. 次の更新式を繰り返す

$$\eta_{t+1} = \eta_t + \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} J(\mathbf{x} + \eta_t, y)) \tag{1.29}$$

3. 更新された摂動を元の入力に投影する

$$\eta_{t+1} = \text{Proj}_{\epsilon}(\eta_{t+1}) \tag{1.30}$$

   - Projϵ(·)：摂動が ϵ の範囲内に収まるようにする投影関数

4. 指定された回数（ステップ数）繰り返す
5. 最終的に得られる Adversarial Noise は次の式で表される

$$\mathbf{x}_{adv} = \mathbf{x} + \eta_T \tag{1.31}$$

   - T：全ステップ数

1.5.2 Adversarial Patch

概要

Adversarial Patch（敵対的パッチ）[17] は、機械学習モデル、ディープラーニングモデルに対する攻撃手法の一つである。このパッチは、入力データに加えることで、モデルの正しい認識を阻害することを目的としている。

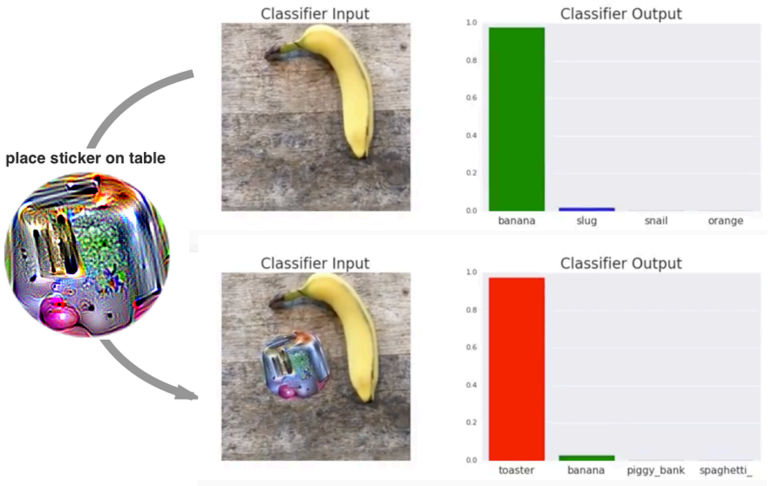


Fig. 1.6: Adversarial Patch

原理

目的関数

$$\hat{p} = \arg\max_{p} \mathbb{E}_{x\sim X, t\sim T, l\sim L} \left[\log \Pr(\hat{y}|A(p, x, l, t))\right] \tag{1.32}$$

- $\hat{p}$：学習されたパッチ
- X：画像のトレーニングセット
- T：パッチの変換に関する分布
- L：画像内の位置に関する分布

第2章　検討手法

2.1　検討手法の背景

YOLO[1.4.4], ViT[1.4.3], DETR[1.4.5] の各モデルにはそれぞれ固有の弱点があるが、共通する弱点もいくつか存在する。[18][19][14]

1. 小さなオブジェクトの検出
2. 複雑なシーンの処理（複雑な背景やオクルージョンのあるシーン。オクルージョン：遮蔽。）

これらを踏まえて、以降の方法でアプローチすることにした。

2.2　フレームワーク

2.2.1　概要図

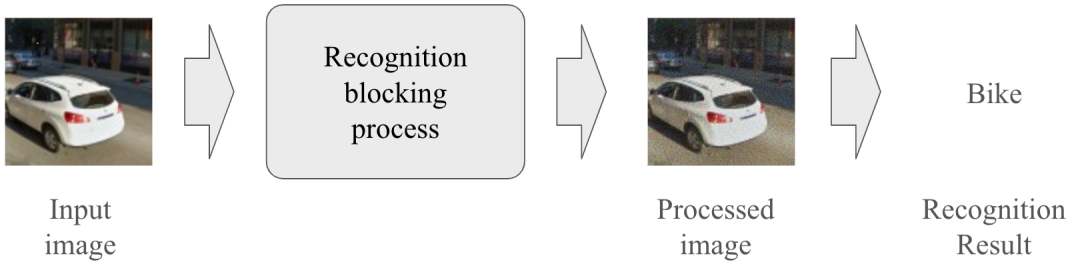


Fig. 2.1: Framework の概要図

2.2.2　概要図詳細

1. Input image: 認識を阻害させたい画像を入力
2. Recognition blocking process: 認識を阻害させる処理を入力画像に対して実行. 今回提案する「直接的攻撃１」～「強調的攻撃」.
3. Processed image: 認識阻害処理をされた画像を出力
4. Recognition Result: 認識モデルが真のクラスとは別のクラスと認識すると、認識阻害効果があることを示す.

2.3 単一の防衛手法

2.3.1 ①Mixed Adversarial Noise

概要

Mixed Adversarial Noise を用いて認識の阻害可否の実験を行った

原理

$$minimum\ Mixed\ Adversarial\ Noise = Constant\ *\ Mixed\ Adversarial\ Noise \quad (2.1)$$

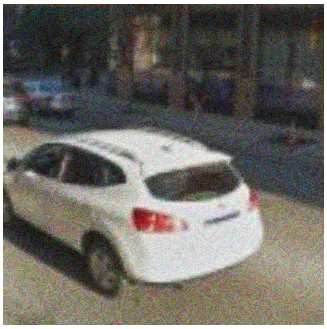
$$Mixed\ Adversarial\ Noise = 0.5*(DETR\ Adversarial\ Noise)+0.5*(yolov3\ Adversarial\ Noise) \quad (2.2)$$

$$Noise\ =\ (Added\ Adversarial\ Noise\ Image)\ -\ (Original\ Image) \quad (2.3)$$

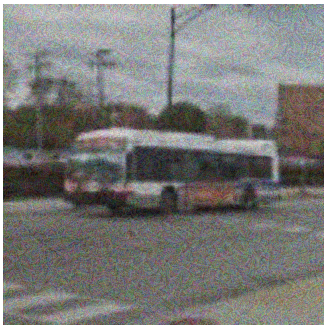
補足説明

- *DETR Adversarial Noise*：DETR を阻害する下限の *Adversarial Noise*
- *yolov*3 *Adversarial Noise*：yolov3 を阻害する下限の *Adversarial Noise*

例



(a) Car



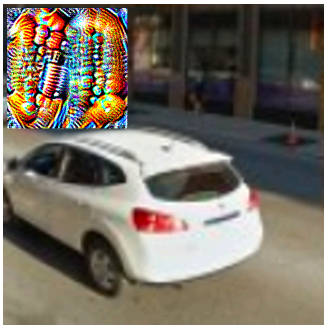
(b) Bus

2.3.2 ②Adversarial Patch

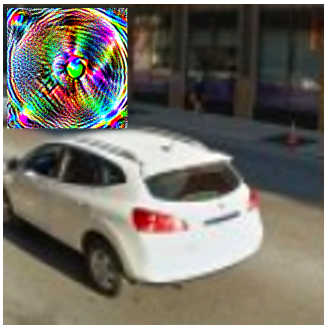
概要

Adversarial Patch を用いて認識の阻害可否の実験を行った

例



(a) Patch A



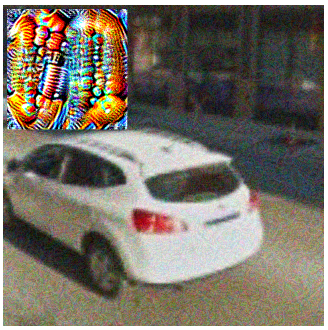
(b) Patch B

2.3.3 ③Mixed Adversarial Noise + Adversarial Patch

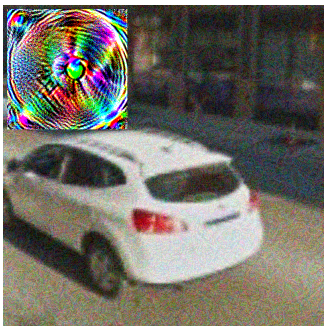
概要

Mixed Adversarial Noise と Adversarial Patch の組み合わせを行い、認識の阻害可否の実験を行った

例



(a) Noise + Patch A



(b) Noise + Patch B

2.3.4 ④攻撃に強い object class を選択

概要

CAPTCHA でよく利用されているクラスで苦手なクラスがあるかの確認を実施した

Table 2.1: CAPTCHA でよく利用されているクラス

No.	class name	クラス名
1	Bicycle	自転車
2	Bridge	橋
3	Bus	バス
4	Car	車
5	Chimney	煙突
6	Crosswalk	横断歩道
7	Hydrant	消火栓
8	Motorcycle	バイク
9	Mountain	山
10	Palm	ヤシ
11	Traffic Light	信号機

例

(a) Bicycle

(b) Traffic Light

2.3.5 ⑤複数の object class を混合

概要

CAPTCHA で利用されている物体が複数入っていて、目的の物体は小さく写っているような画像を用いる。これにより、目的以外の物体に認識を集中させて目的の物体に対する認識を阻害させる。

Table 2.2: 対象物のクラス名と状況

class name	状況
Bicycle	車に囲まれて見えづらい状態など
Car	自転車などが前面に出ていて車が主体で写っていない状態など
Hydrant	車が側にあったり、遠くに写っている状態など
Traffic Light	遠くに写っていたり、建物と重なって写っている状態など

例

(a) Car

(b) Traffic Light

2.3.6 ⑥CAPTCHA に使用されていない object class を選択

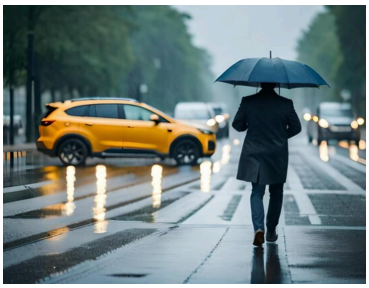
概要

CAPTCHA で利用されていない物体（※マイナーすぎるものは良くないので、最低限一般的に誰もが知ってはいる物体）を用いて認識の阻害可否の実験を行った。

Table 2.3: マイナーな対象物

No.	class name	クラス名
1	Cane	杖
2	Faucet	蛇口
3	Smart Phone	スマートフォン
4	Umbrella	傘

例



(a) Umbrella

(b) Faucet

2.4　複数の防衛手法の組み合わせ

『2.3 単一の防衛手法』で示した手法を組み合わせる

2.4.1　⑦：① + ④

概要

「①Mixed Adversarial Noise」と「④攻撃に強い object class を選択」を組み合わせた実験

例

実験手法としては、上記の方法が考えられるが、Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

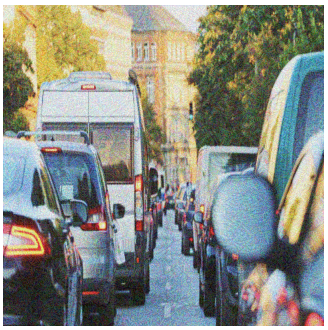
2.4.2　⑧：① + ⑤

概要

「①Mixed Adversarial Noise」と「⑤複数の object class を混合」を組み合わせた実験

例

(a) Car



(b) Traffic Light

2.4.3 ⑨：① + ⑥

概要

「①Mixed Adversarial Noise」と「⑥CAPTCHA に使用されていない object class を選択」を組み合わせた実験

例

実験手法としては、上記の方法が考えられるが、Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

2.4.4 ⑩：② + ④

概要

「②Adversarial Patch」と「④攻撃に強い object class を選択」を組み合わせた実験

例

(a) Bicycle　　**(b)** Traffic Light

2.4.5 ⑪：② + ⑤

概要

「②Adversarial Patch」と「⑤複数の object class を混合」を組み合わせた実験

例

(a) Car

(b) Traffic Light

2.4.6 ⑫：② + ⑥

概要

「②Adversarial Patch」と「⑥CAPTCHA に使用されていない object class を選択」を組み合わせた実験

例

(a) Umbrella

(b) Faucet

2.4.7 ⑬：③ + ④

概要

「③Mixed Adversarial Noise + Adversarial Patch」と「④攻撃に強い object class を選択」を組み合わせた実験

例

実験手法としては、上記の方法が考えられるが、Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

2.4.8 ⑭：③ + ⑤

概要

「③Mixed Adversarial Noise + Adversarial Patch」と「⑤複数の object class を混合」を組み合わせた実験

例

実験手法としては、上記の方法が考えられるが、Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

2.4.9 ⑮：③ + ⑥

概要

「③Mixed Adversarial Noise + Adversarial Patch」と「⑥CAPTCHA に使用されていない object class を選択」を組み合わせた実験

例

実験手法としては、上記の方法が考えられるが、Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

2.5 強調的攻撃

概要

Adversarial Noise でターゲット以外の物体を強調し、ターゲットへの認識を逸らす試み。今回は自転車を強調させて、車を認識させないようにする実験を行った。

2.6 評価指標：PSNR

概要

PSNR（Peak Signal-to-Noise Ratio）は、元画像と処理後の画像の 2 つの画像間の画像の劣化の度合いを示す評価指標である。この指標は値（単位：dB）が高いほど品質が良く、元画像に近いことを示す。一般的に「30dB 以上：高画質」、「20 30dB：中画質」、「20dB 未満：低画質」とされている。Adversarial Noise, Adversarial Patch を CAPTCHA 画像に付与する量を定量的に評価する。

式

$$PSNR = 10\log_{10}\frac{MAX_I^2}{MSE} \tag{2.4}$$

MSE: 平均二乗誤差
MAX_I: 画像のピクセルの最大値

式

$$MES = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(I_{original}(i,j) - I_{processed}(i,j))^2 \tag{2.5}$$

- $I_{original}$：元画像のピクセル値
- $I_{processed}$：処理後の画像のピクセル値
- M：画像の縦のピクセル値
- N：画像の横のピクセル値

第3章　実験結果

3.1 単一の防衛手法

3.1.1 ①Mixed Adversarial Noise

結果概要

YOLO と DETR には認識を阻害できたが、CaptchaBreaker は認識を阻害できなかった

Table 3.1: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	○	○	×
Bus	○	○	×
Bicycle	○	○	×
Hydrant	○	○	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.1.2 ②Adversarial Patch

結果概要

YOLO は認識を阻害できたが、DETR と CaptchaBreaker は認識を阻害できなかった

Table 3.2: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	○	×	×
Bus	○	×	×
Bicycle	○	×	×
Hydrant	○	×	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.1.3 ③Mixed Adversarial Noise + Adversarial Patch

結果概要

YOLO と DETR には認識を阻害できたが、CaptchaBreaker は認識を阻害できなかった

Table 3.3: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	○	○	×
Bus	○	○	×
Bicycle	○	○	×
Hydrant	○	○	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.1.4 ④攻撃に強い object class を選択

結果概要

各モデルともにCaptchaで良く利用されているオブジェクトクラスで苦手としているクラスは特になかった

Table 3.4: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Bicycle	×	×	×
Bridge	×	×	×
Bus	×	×	×
Car	×	×	×
Chimney	×	×	×
Crosswalk	×	×	×
Hydrant	×	×	×
Motorcycle	×	×	×
Mountain	×	×	×
Palm	×	×	×
Traffic Light	×	×	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.1.5 ⑤複数の object class を混合

結果概要

各モデルにに対して認識を阻害させることができた

Table 3.5: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	○	○	○
Bicycle	○	○	○
Hydrant	○	○	○
Traffic Light	○	○	○

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.1.6 ⑥CAPTCHA に使用されていない object class を選択

結果概要

マイナーな対象物でも苦手としているクラスはなかった。また、YOLO と DETR にはこれらのクラスに対応していなかったため実験を実施していない。

Table 3.6: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Cane	−	−	×
Faucet	−	−	×
Smart Phone	−	−	×
Umbrella	−	−	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.2　複数の防衛手法の組み合わせ

3.2.1　⑦：① + ④

結果概要

Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

3.2.2　⑧：① + ⑤

結果概要

目的以外の物体に認識を集中させて、目的の物体に対する認識を阻害させることができた。しかし、これは「⑤複数の object class を混合」単体でも認識の阻害ができていたため自明な結果である

Table 3.7: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	○	○	○
Bicycle	○	○	○
Hydrant	○	○	○
Traffic Light	○	○	○

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.2.3 ⑨：① + ⑥

結果概要

Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

3.2.4 ⑩：② + ④

結果概要

YOLO は認識を阻害できたが、DETR と CaptchaBreaker は認識を阻害できなかった（苦手なクラスは特に見られなかった）

Table 3.8: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Bicycle	○	×	×
Bridge	○	×	×
Bus	○	×	×
Car	○	×	×
Chimney	○	×	×
Crosswalk	○	×	×
Hydrant	○	×	×
Motorcycle	○	×	×
Mountain	○	×	×
Palm	○	×	×
Traffic Light	○	×	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.2.5 ⑪：② + ⑤

結果概要

目的以外の物体に認識を集中させて目的の物体に対する認識を阻害させることができた。しかし、これは『⑤複数の object class を混合』単体でも認識の阻害ができていたため自明な結果である。

Table 3.9: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	○	○	○
Bicycle	○	○	○
Hydrant	○	○	○
Traffic Light	○	○	○

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.2.6 ⑫：② + ⑥

結果概要

マイナーな対象物でも苦手としているクラスはなかった。また、YOLO と DETR にはこれらのクラスに対応していなかったため実験を実施していない。

Table 3.10: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Cane	−	−	×
Faucet	−	−	×
Smart Phone	−	−	×
Umbrella	−	−	×
Window	−	−	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

3.2.7 ⑬：③ + ④

結果概要

Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

3.2.8 ⑭：③ + ⑤

結果概要

Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない。

3.2.9 ⑮：③ + ⑥

結果概要

Mixed Adversarial Noise を作成する時間がかかってしまう都合上、難しいため実験を実施していない

3.3 強調的攻撃

結果概要

強調的攻撃で認識を阻害することはできなかった。

Table 3.11: 各モデルにおける実験結果

	YOLO	DETR	CaptchaBreaker
Car	×	×	×

- ○：認識を阻害できている
- ×：認識を阻害できていない

第4章　考察と結論, 今後の展望

4.1 考察と結論

4.1.1 考察

共通で効果のあった手法

- ⑤複数の object class を混合
  - これまでの手法の中で一番効果があった。これは、検討手法の複雑なシーンの処理に該当する。

4.1.2 結論

これらの実験結果から、CAPTCHA 画像は画像認識モデルを誤認識させるには、本実験で行った『⑤複数の object class を混合』のような複雑なシーンの画像が必要であることがわかった。

また、これらの結果から当初の目的の CAPTCHA Breaker に対する阻害実験というだけでなく、視点を変えると認識器の弱点を探索するという実験であったとも言える。

4.2 今後の展望

複雑なシーンの画像が必要であることがわかったが、どの程度までの複雑性が許容されるか（モデルと人間の境界）までは究明できていない。そのため、今後はこの境界を求めて行くことが焦点になる。

謝辞

本研究は, 北陸先端科学技術大学院大学 先端科学技術研究科 青木 利晃 教授、小谷 一孔 教授、SIRITANAWAN, Prarinya 助教授のご指導のもとで行われました.

参考文献

[1] “Pix.” [Online]. Available: http://www.captcha.net/captchas/pix/

[2] “What is recaptcha?” [Online]. Available: https://www.google.com/recaptcha/about/

[3] “Gimpy.” [Online]. Available: http://www.captcha.net/captchas/gimpy/

[4] “Using a captcha to prevent bots from using your asp.net web razor) site.” [Online]. Available: https://learn.microsoft.com/en-us/aspnet/web-pages/overview/security/using-a-catpcha-to-prevent-automated-programs-bots-from-using-your-aspnet-web-site

[5] “Sounds.” [Online]. Available: http://www.captcha.net/captchas/sounds/

[6] G. F. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. J. Goodfellow, and J. Sohl-Dickstein, “Adversarial examples that fool both human and computer vision,” *CoRR*, vol. abs/1802.08195, 2018. [Online]. Available: http://arxiv.org/abs/1802.08195

[7] R. Jin, L. Huang, J. Duan, W. Zhao, Y. Liao, and P. Zhou, “How secure is your website? a comprehensive investigation on captcha providers and solving services,” 2023.

[8] “2captcha,” https://2captcha.com/ja.

[9] “Best captcha solver,” https://bestcaptchasolver.com/.

[10] “Anticaptcha,” https://anti-captcha.com/ja.

[11] Anonymous, “ReCaptchaBreaker: Breaking Google’s Recaptcha Image Challenge with near human performance,” 12 2022. [Online]. Available: https://github.com/Hackear2041/ReCaptchaBreaker

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[13] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[14] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[16] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2019.

[17] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2018. [Online]. Available: https://arxiv.org/abs/1712.09665

[18] H. Ouyang, "Deyo: Detr with yolo for end-to-end object detection," 2024.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[20] K. FUKUSHIMA, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, pp. 193–202, 1980. [Online]. Available: https://cir.nii.ac.jp/crid/1573387448925217280

[21] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *CoRR*, vol. abs/1511.08458, 2015. [Online]. Available: http://arxiv.org/abs/1511.08458

[22] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2018.

[23] M. Yani, M. B. I. S, Si., and M. C. S. S.T., "Application of transfer learning using convolutional neural network method for early detection of terry ' s nail," *Journal of Physics: Conference Series*, vol. 1201, no. 1, p. 012052, may 2019. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1201/1/012052

[24] Dharmaraj, "Zero-padding in convolutional neural networks," https://medium.com/@draj0718/zero-padding-in-convolutional-neural-networks-bf1410438e99, 2021.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

付録A　補足事項: その他関連技術

A.1 CNN

A.1.1 概要

CNN（Convolutional Neural Network、畳み込みニューラルネットワーク）は、畳み込み層と呼ばれるパターン認識を行う層を持ったニューラルネットワークである。畳み込み層は、データの局所性を表現する層である。そして、層が進むにつれて異なるスケールでの特徴を抽出する。最初の層では、物体のエッジなどの低レベルな情報を抽出し、層が深くなることでより抽象的な特徴を抽出する。[20][21]

A.1.2 アーキテクチャ

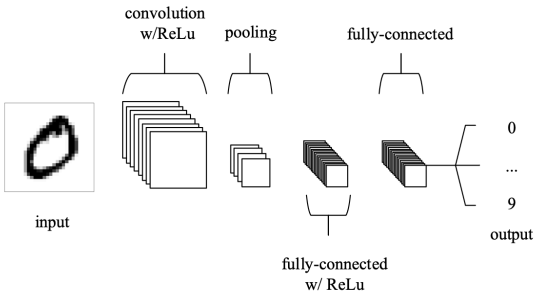


Fig. A.1: CNN architecture[21]

- 畳み込み層、プーリング層、全結合層で構成されている

A.1.3 原理詳細

畳み込み (Convolution)

画像等の入力データに対して、フィルタ（カーネル）を適用することで特徴マップを生成する。入力データに対して、フィルタのウィンドウをスライドさせながらドット積を計算する。畳み込みは、以下の式で表される。

$$(I * K)(x, y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x + i, y + j) \cdot K(i, j) \tag{A.1}$$

- I：入力画像
- K: フィルタ
- m, n：フィルタの行列数

Fig. A.2: カーネル [22]

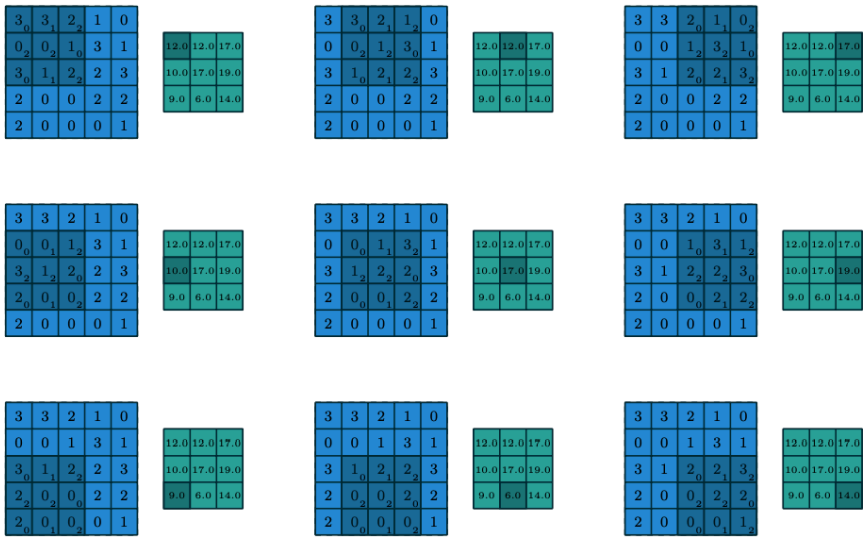


Fig. A.3: 畳み込みの演算概略図 [22]

プーリング (Pooling)

Pooling にはMAX Pooling（左図）と Average Pooling（右図）がある。MAX Pooling は局所範囲内の最大値、Average Pooling は局所範囲内の平均値を集約して特徴マップを作成する（局所範囲：図で色分けされている領域）。

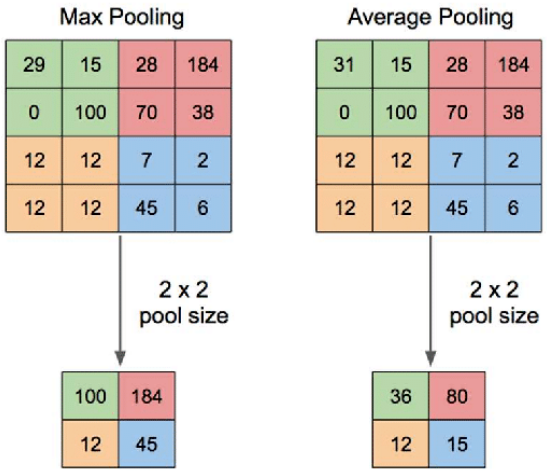


Fig. A.4: Max Pooling と Average Pooling[23]

パディング (Padding)

パディングの方法にはいくつか種類があるが、一般的にはゼロパディングが使用されている。入力データの周囲にゼロを追加する操作である。これにより、畳み込み後の出力サイズを調整し、エッジ部分の情報を保護することができる。

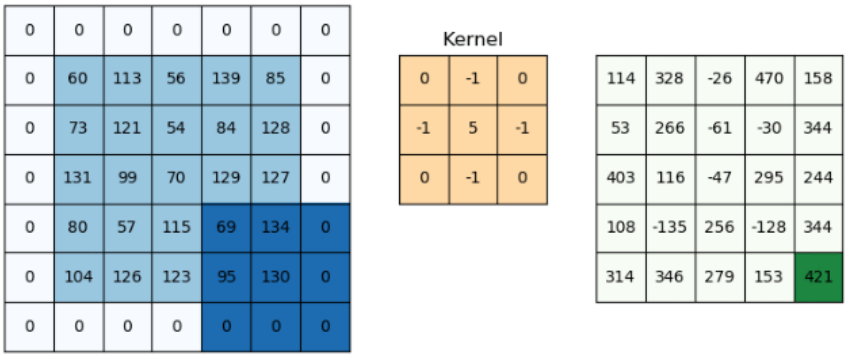


Fig. A.5: ゼロパディングの演算概略図 [24]

A.2 ResNet

A.2.1 概要

CNN は、層数を増やすことでより高度な学習ができるようになったが、その層が増えることで勾配消失が起こり学習が進まなくなる課題が出てきた。そして、この課題を解決するために残差ブロックを導入して層の深さが深くなっても学習することができるようにしたモデルである。[25]

A.2.2 アーキテクチャ

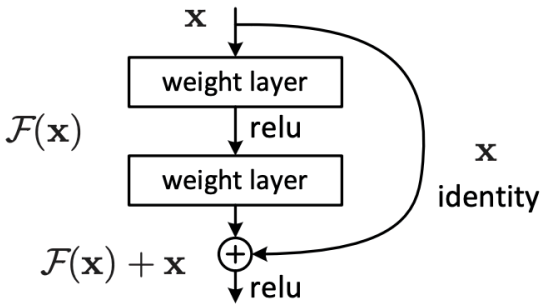


Fig. A.6: ResNet architecture

A.2.3 原理詳細

ResNet は「残差ブロック」の導入により、残差学習 (residual learning) でより深い学習ができるモデルである。
残差ブロックは、入力をそのまま次の層に伝播させる「ショートカット接続（Shortcut Connection）」を含む構造で、バックプロパゲーション時に勾配消失を防ぐ役割がある。

$$H(x) = F(x) + x$$

$F(x)$: 層を通った出力
x: 入力

A.3 Transformer

A.3.1 概要

Transformer[19]は、それまでのRNN（Recurrent Neural Networks）やLSTM（Long Short-Term Memory）などのシーケンシャルな処理を行うモデルとは異なり、データの全体を一度に処理できることが特徴である。これにより、計算の効率化が図られ、より長いシーケンスのデータを扱うことが可能となった。

A.3.2 アーキテクチャ

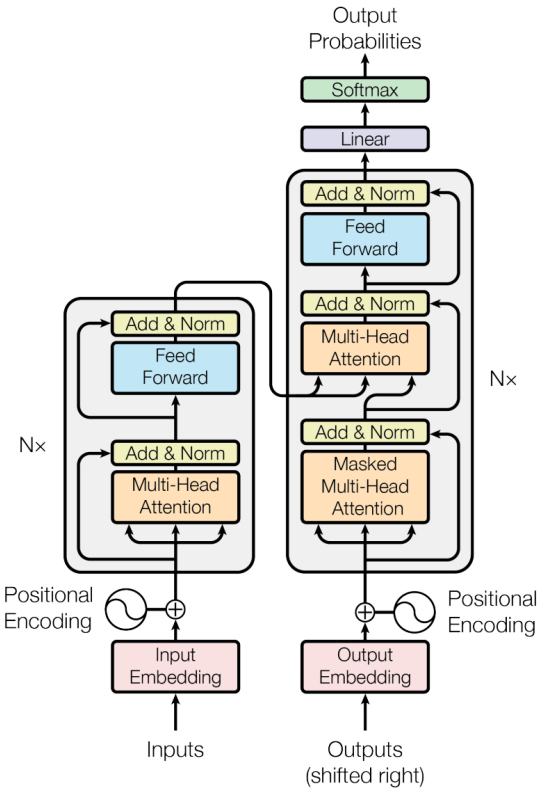


Fig. A.7: Transformer

A.3.3 原理詳細

処理内容

Transformer は、大きく分けてエンコーダー（Encoder）とデコーダー（Decoder）の 2 つの部分から構成されている。エンコーダーの各層の出力が次のエンコーダー層の入力として使用され、最終的なエンコーダーの出力がデコーダーの入力として渡される。

$$Output = Decoder(Encoder(X_{input})) \tag{A.2}$$

また、エンコーダーは N 層の積み重ねによって構成されており、各層は以下の 2 つの主要なサブレイヤーから構成されている。

- マルチヘッド自己注意機構（Multi-Head Self-Attention Mechanism）
- 位置ごとのフィードフォワードネットワーク（Position-wise Feed-Forward Neural Network）

Multi-Head Self-Attention Mechanism

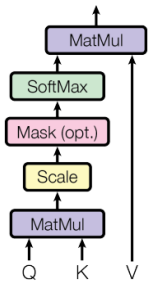


Fig. A.8: Self-Attention

Transformer の中心的な部分は自己注意機構である。自己注意は、入力シーケンス内の各トークンが他のトークンにどれだけ注意を払うべきかを学習する。

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{A.3}$$

- $Q = XW_Q$：Query

- $K = XW_K$：Key
- $V = XW_V$：Value
- d_k：キーの次元数

Multi-Head Attention

自己注意機構を複数のヘッドで並行して実行することで、異なる部分空間での注意を同時に学習する。

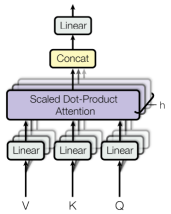


Fig. A.9: Multi-Head Attention

Multi-Head Attention は以下のように表される。

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \ldots, head_h)W^O \tag{A.4}$$

また、各 Head は以下のように計算される。

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{A.5}$$

Feed-Forward (Position-wise Feed-Forward)

各自己注意層の出力は、位置ごとのフィードフォワードネットワークに渡される。これは 2 層で構成されていて、以下の式で表される。

$$FNN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{A.6}$$

W_1：１層目の重み
b_1：１層目のバイアス
W_2：２層目の重み
b_2：２層目のバイアス

Multi-Head Attention（Masked）

デコーダーの自己注意機構は、未来のトークンに対する情報漏洩を防ぐためにマスクが施されている。

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}} + M)V \tag{A.7}$$

- $Q = XW_Q$：Query
- $K = XW_K$：Key
- $V = XW_V$：Value
- d_k：キーの次元数
- M：マスク行列

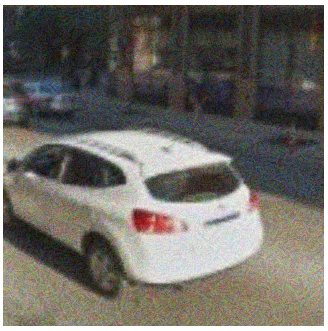
Encoder-Decoder Attention

エンコーダーの出力を用いてデコーダーの各層が入力シーケンスの情報を取り込む。

$$\text{Attention}(Q_{decoder}, K_{encoder}, V_{encoder}) \tag{A.8}$$

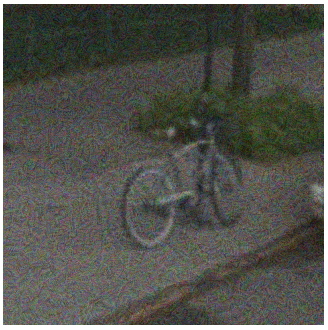
付録B　補足事項: 実験で使用した画像（抜粋）

B.1 ①Mixed Adversarial Noise

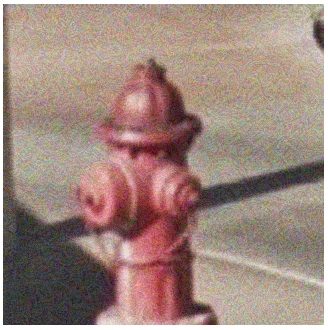


(a) Car

(b) Bus

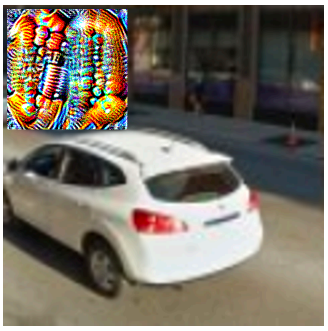


(c) Bycycle

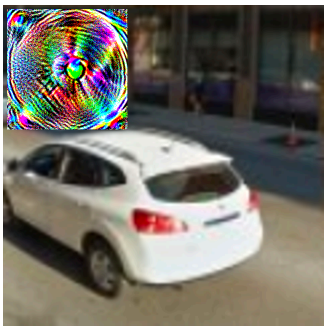


(d) Hydrant

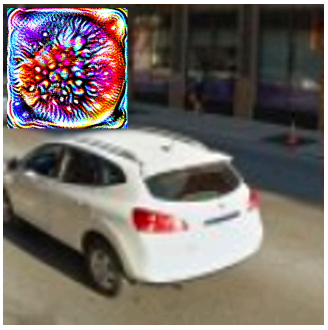
B.2 ②Adversarial Patch



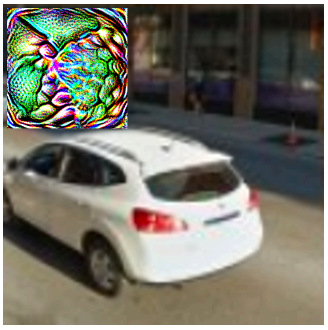
(a) Patch A



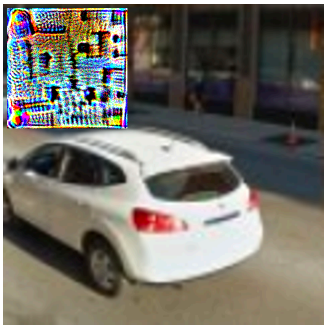
(b) Patch B



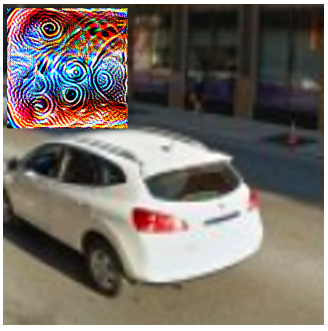
(c) Patch C



(d) Patch D



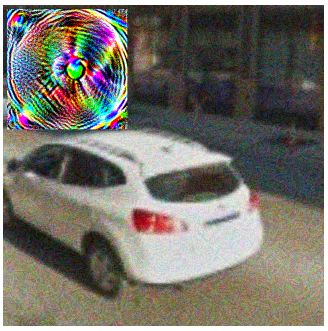
(e) Patch E



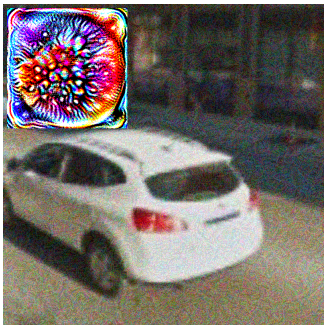
(f) Patch F

B.3 ③Mixed Adversarial Noise + Adversarial Patch

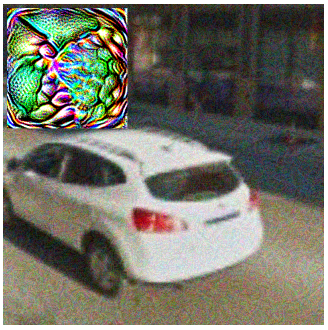
(a) Noise + Patch A



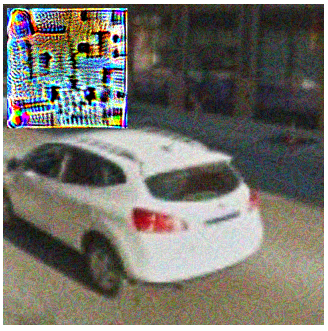
(b) Noise + Patch B



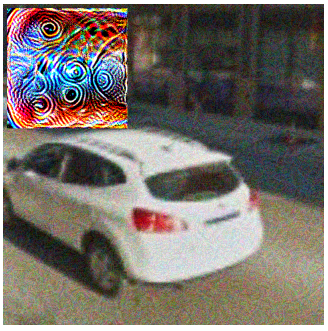
(c) Noise + Patch C



(d) Noise + Patch D



(e) Noise + Patch E



(f) Noise + Patch F

B.4 ④攻撃に強い object class を選択

(a) Bicycle

(b) Bridge

(c) Bus

(d) Car

(e) Chimney

(f) Crosswalk

(a) Hydrant

(b) Motorcycle

(c) Mountain

(d) Palm

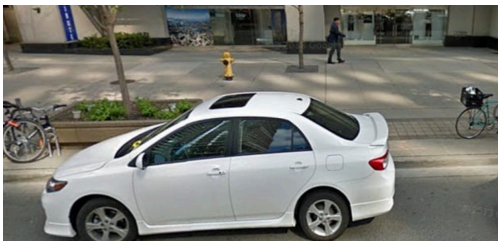
(e) Traffic Light

B.5 ⑤複数の object class を混合

(a) Car

(b) Traffic Light

(c) Bicycle



(d) Hydrant

B.6 ⑥CAPTCHA に使用されていない object class を選択

(a) Cane

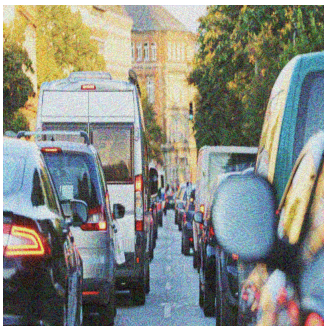
(b) Faucet

(c) Smart Phone

(d) Umbrella

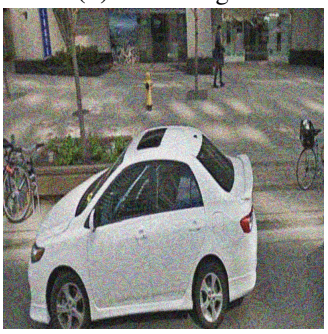
B.7 ⑧ : ① + ⑤

(a) Car



(b) Traffic Light

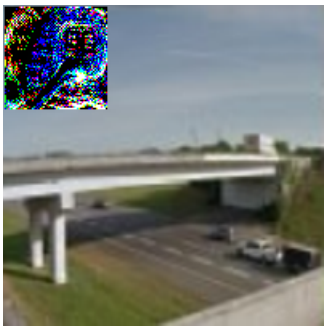
(c) Bicycle



(d) Hydrant

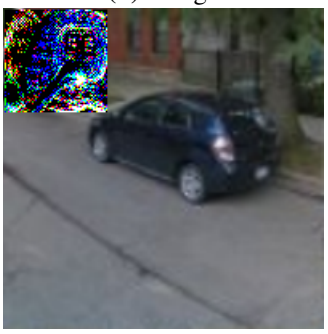
B.8 ⑩ : ② + ④

(a) Bicycle



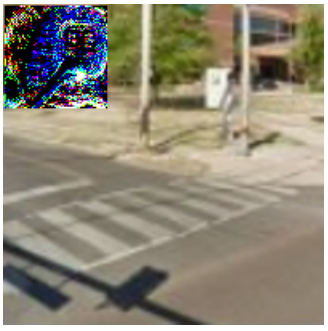
(b) Bridge

(c) Bus



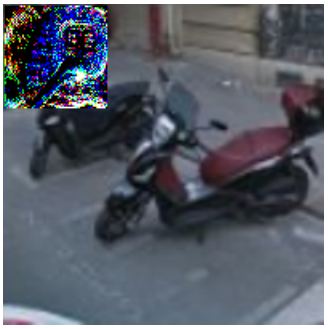
(d) Car

(e) Chimney

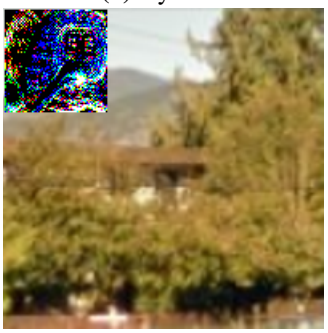


(f) Crosswalk

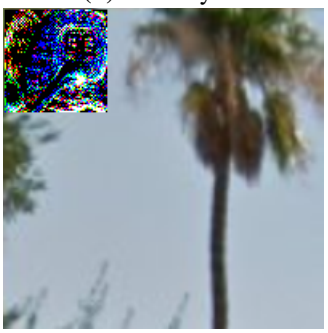
(a) Hydrant



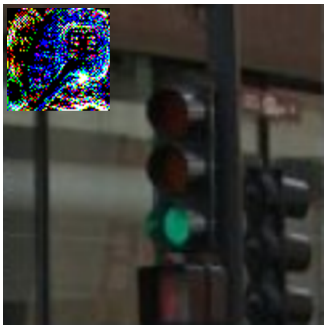
(b) Motorcycle



(c) Mountain



(d) Palm



(e) Traffic Light

B.9 ⑪ : ② + ⑤

(a) Car

(b) Traffic Light

(c) Bicycle

(d) Hydrant

B.10 ⑫ : ② + ⑥

(a) Cane

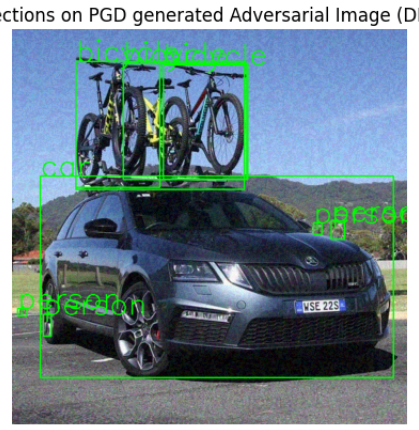
(b) Faucet

(c) Smart Phone

(d) Umbrella

B.11 強調的攻撃

(a) 強調前



(b) 強調後

付録C　補足事項: 実験結果の一覧

Table C.1: 実験結果の一覧①

手法分類	手法名	class name	YOLO	DETR	Captcha Breaker
単一の防衛手法	①Mixed Adversarial Noise	Car	○	○	×
		Bus	○	○	×
		Bicycle	○	○	×
		Hydrant	○	○	×
	②Adversarial Patch	Car	○	×	×
		Bus	○	×	×
		Bicycle	○	×	×
		Hydrant	○	×	×
	③Mixed Adversarial Noise + Adversarial Patch	Car	○	×	×
		Bus	○	×	×
		Bicycle	○	×	×
		Hydrant	○	×	×
	④攻撃に強い object class を選択	Bicycle	×	×	×
		Bridge	×	×	×
		Bus	×	×	×
		Car	×	×	×
		Chimney	×	×	×
		Crosswalk	×	×	×
		Hydrant	×	×	×
		Motorcycle	×	×	×
		Mountain	×	×	×
		Palm	×	×	×
		Traffic Light	×	×	×
	⑤複数の object class を混合	Car	○	○	○
		Bicycle	○	○	○
		Hydrant	○	○	○
		Traffic Light	○	○	○
	⑥CAPTCHA に使用されていない object class を選択	Cane	–	–	×
		Faucet	–	–	×
		Smart Phone	–	–	×
		Umbrella	–	–	×
		Window	–	–	×

○：認識を阻害できている, ×：認識を阻害できていない, –：実験未実施

Table C.2: 実験結果の一覧②

手法分類	手法名	class name	YOLO	DETR	Captcha Breaker
複数の防衛手法の組み合わせ	⑧：① + ⑤	Car	○	○	○
		Bicycle	○	○	○
		Hydrant	○	○	○
		Traffic Light	○	○	○
	⑩：② + ④	Bicycle	○	×	×
		Bridge	○	×	×
		Bus	○	×	×
		Car	○	×	×
		Chimney	○	×	×
		Crosswalk	○	×	×
		Hydrant	○	×	×
		Motorcycle	○	×	×
		Mountain	○	×	×
		Palm	○	×	×
		Traffic Light	○	×	×
	⑪：② + ⑤	Car	○	○	○
		Bicycle	○	○	○
		Hydrant	○	○	○
		Traffic Light	○	○	○
	⑫：$_2$ + ⑥	Cane	–	–	×
		Faucet	–	–	×
		Smart Phone	–	–	×
		Umbrella	–	–	×
		Window	–	–	×
強調的攻撃	強調的攻撃	Car	×	×	×

○：認識を阻害できている, ×：認識を阻害できていない, –：実験未実施