令和 2 年度茨城大学工学部情報工学科卒業研究論文

悪天候時の自動車の物体検出の領域適応

所属　情報工学科

著者　内間けんじ (17T4015S)

指導教員　新納浩幸教授

令和 3 年 2 月 5 日 (金)

令和 2 年度茨城大学工学部情報工学科卒業研究論文

悪天候時の自動車の物体検出の領域適応

著者

内間けんじ (17T4015S)

指導教員

新納浩幸教授

論文要旨

近年物体検出の技術は様々の分野に応用されている。例として自動運転で使われる自動車や人の検知・認識などが挙げられる。このような自動車の物体検出を行うためには物体検出器の学習を行う必要があり、そのために自動車データセット用意する必要がある。その後用意したデータセットで学習した物体検出モデルで推論を行うことで自動車の位置を予測することができるようになる。しかしこういった物体検出の手法には大量のデータセットが要求される。対象領域を検出するするのに必要なデータセットが少ない場合はデータを作成する必要があるが、通常データセットの自作にはアノテーションの作業等で大きなコストがかかってしまう。

具体的には、自動車の物体検出においてアノテーションされた好天候時のデータセットは多く存在しているが、悪天候時のデータセットが少ない場合が挙げられる。こういった場合で悪天候時の限定された領域での物体検出の目的とした時、本来ならば訓練データとして対象領域対象領域である悪天候時データセットを用意したいがそれが困難である。

そこで本稿では CycleGAN を用いて入力画像から対象領域の特徴を保持している画像生成を行い、作成した偽のデータセットで物体検出が可能かどうかの検証を行った。

悪天候時の自動車の物体検出を行うために、に CycleGAN によりアノテーションされた好天時領域の画像から悪天候時領域の偽画像を生成し、好天候時の画像のアノテーションを偽の画像データセットに対応させることで物体検出器の学習を行い偽画像データによる物体検出ができるかを確かめる。

目次

第 1 章

序論

ある画像について物体検出を行うためには、あらかじめある画像と同じ領域の特徴を保持する大量のアノテーションされた画像データセットを用意して物体検出モデルの学習を行う必要がある。アノテーションデータとは、画像のどこに何が写っているかの記録を保持しているデータを指す。

物体検出したい目的に対して利用可能なデータセットが存在する場合は問題ないが、目的に合致するデータセットが存在しない場合は初めから自作する必要がある。そういった場合で大量のアノテーションデータを手作業で用意するのには多くの時間と労力が要求され大変なコストがかかることになる。例として本研究では悪天候時の自動車の物体検出において悪天候時の画像データセットをあらかじめ用意することができない場面を想定している。

このような状況でも自動車の物体検出を行うために、CycleGAN によってアノテーションされた好天時のデータから悪天候時の偽画像データを生成し、そのアノテーションデータを悪天候時の画像データセットに対して再利用することで自動車の物体検出モデルの学習を行い実際に物体検出ができるかを検証する。

第 2 章

関連研究

CycleGAN を用いた関連研究である [3] では、アノテーションされた昼間の画像データセットから偽の夜間データを生成して夜間領域の画像における自動車の検出システムの学習手法を提案している。この研究では CycleGAN を用いて昼間の画像から夜間の偽画像を生成し、それらに昼画像のアノテーションデータを対応させて物体検出器の学習行っている。実験では昼間データのみでの学習、偽の夜間データのみでの学習、昼間データと偽の夜間データの複合データセットによる学習、夜間データのみでの学習、夜間データと昼間データの複合データセットによる学習を行い 5 種類のモデルを作成し、それらのモデルの評価を比べていた。

検出器で使用されていたモデルは Faster-RCNN であった。また、モデルの評価指標には mAP を採用していた。

結論では、昼間の画像データのみで学習を行ったモデルよりも昼間データと夜間データの複合データを学習を行ったモデルの方が優れていることが述べられていた。これは昼間データで学習したモデルと昼間と偽の夜間の複合データで学習したモデルにも同じような結果が得られていた。

図 2.1: 生成された偽の夜中の画像

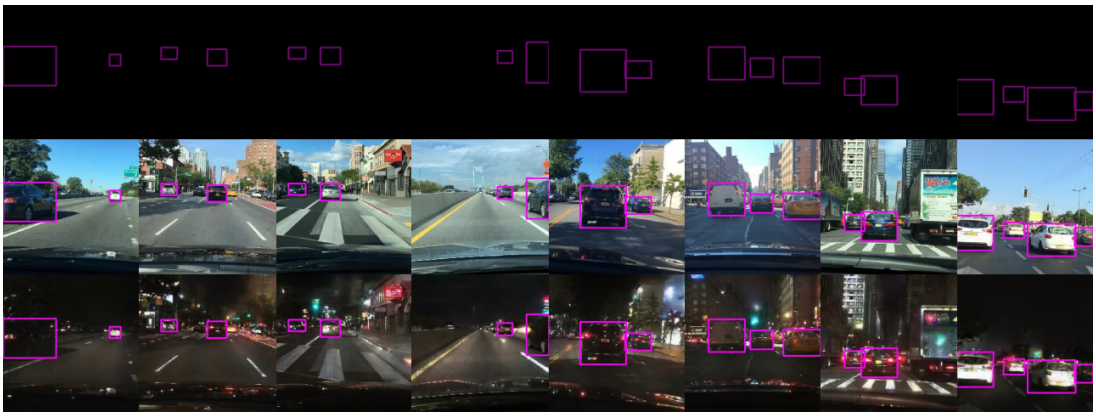

図 2.2: 日中データから変換した偽の夜中データとアノテーション

第 3 章

物体検出

物体検出とは入力された画像に対して、物体がどのような位置に存在しているかを、それがどのような物体であるかを推定するための手法である。物体検出を行うためには検出器のモデルを学習させる必要があり、それにはアノテーションされた大量のデータセットが必要になる。データセットは画像データとそれに対する物体の位置・種類の情報を含むアノテーションデータからなる。物体の位置はバウンディングボックスによって区別されている。アノテーションデータには物体の周りを囲む箱の座標が記されており、たいていの場合は四角い領域の対角の 2 点座標、あるいは領域中心の座標と領域の縦・横の長さで構成される。

3.1 CNN

CNN(Convolutional Neural Network) とは主に画像の分野においてよく使われるネットワークである。畳み込みニューラルネットワークとも呼ばれ、畳み込み層とプーリング層が交互に重なる構造をしている。通常のニューラルネットワークとして良く用いられる全結合層では入力データを単純に並べてから学習を行うが、それでは全ての数値が一次元的に配置されてしまうため同じように捉えられてしまう。それに対して CNN ではデータを入力する際に三次元的な情報が保持されながら学習は行われるため、画像データの位置情報が失われることがない。

畳み込み層では入力された画像にたいしてあらかじめ設定されたフィルターに従って画像の変換を行う。フィルターは入力画像よりもサイズが小さく、それらを入力画像に対しスライドするように移動する。その際フィルターが重なった場所で値を掛け合わせ

て合算した値を出力した二次元の特徴ベクトルを抽出する。

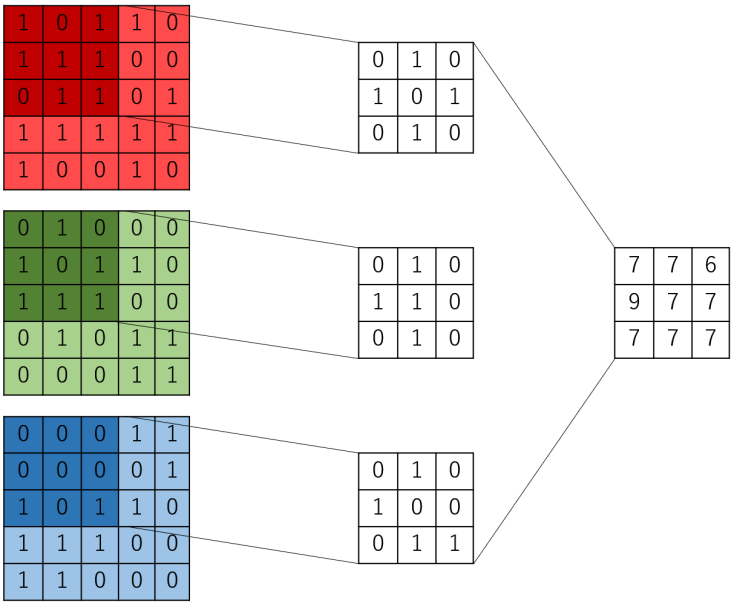

図 3.1: 畳み込み層とフィルタの例

図 3..1 の例では入力 (5,5,3) に対して (3,3) のフィルターがチャンネルずつに用意されている。入力に対してフィルターを枡掛けしていくと、始めは 3+3+1=7 の値が得られることがわかる。その後特徴ベクトルに 7 の値を記録してフィルターは一つとなりにずれて再度升掛けの演算を行っていく。最終的には (3,3) の特徴ベクトルが得られていることが分かる。

これらの操作に加えて畳み込み演算を行う際に別のパラメータを設定することがある。

畳み込み層でフィルターを入力した画像よりはみ出るように使用したい場合、入力画像の外周を別の値で埋めることをパディングという。特に入力画像の外周を 0 で囲むように拡張することをゼロパディングといい、これによって入力画像と同じサイズの出力画像を得ることができる。

入力画像上でフィルターをスライドさせる場合は通常 1 マスずつずれるように移動する。しかしあらかじめストライドの値を 1 より大きく設定することによって移動幅を変えることができる。これによって出力されるサイズを 1/ストライドの値にすることができるため、出力画像を小さくするときに使用される。

プーリング層では畳み込み層で出力された特徴マップの特定の領域から局所的な値を取り出して特徴マップを出力する。例として指定された領域内から最大となる値を出力

していく MaxPooling や領域内の平均の値を演算して取り出し AveragePooling が存在する。プーリング層によって値を取り出すことによって画像の特徴的な部分を保持するようにデータを小さくすることができる。

代表的な CNN として 2012 年に開発された Alexnet [1] や 2014 年の VGG などが挙げられる。

3.2　R-CNN

R-CNN は物体検出モデルの一つである。CNN では画像全体から特徴量を抽出するが、R-CNN ではあらかじめ領域を提案して抽出しリサイズした後に CNN への入力画像としている。領域の提案は selective search などで行われる。その後領域候補にたいして CNN で特徴量を計算しそれぞれの領域になにが映っているかを SVM によって分類する。

selective search は画像中の式特徴量を計算して画像中で類似している領域を pixel レベルで分割してバウンディングボックスを作成する手法である。これによって画像を 2000 個程度の領域へと分割している。

CNN 部分では転移学習を行って AlexNet や VGG などといった学習済のネットワークを使用することができる。

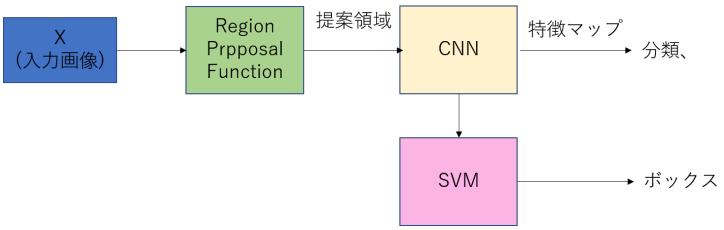

図 3.2: R-CNN の構造

3.3　Faster R-CNN

Faster R-CNN は R-CNN をベースに考案された物体検出モデルである。R-CNN やその系統である Fast R-CNN では画像中の領域提案に先述した selective search や他の手法を利用していた。しかしそれら外部のアルゴリズムを使うとコストが大きくなってしまい計算速度が遅くなることが問題であった。そこで Faster R-CNN ではそういった

アルゴリズムを使用する代わりにネットワーク内に RPN(領域提案ネットワーク) を追加することで End-to-End で学習を実施できるようになり高速化を実現をしている。

RPN は学習済のモデルよって出力された特徴マップに対して物体の候補領域とスコアを出力するネットワークである。特徴マップ上で Anchor を定義する。それらを中心に slaiding window という小さな領域で特徴マップ上を走査し複数の Anchor Boxe s を作成する。その後各 Anchor Boxe s についてそれが物体であるか背景であるかと物体である場合 Truth とどれだけ離れているかの誤差評価して学習を行う。物体か背景かの判断には IOU 指標を用いる。IOU とは 2 領域の共通面積と 2 領域の合計面積の商であり、この値が 1 に近いほど正解に近いと判断できる。っこのとき IOU が 0.3 未満である場合は背景と判断し 0.7 より多きければ物体であると判断する。正解データとのずれに関しては Box の中心座標と縦横の長さの誤差から測る。

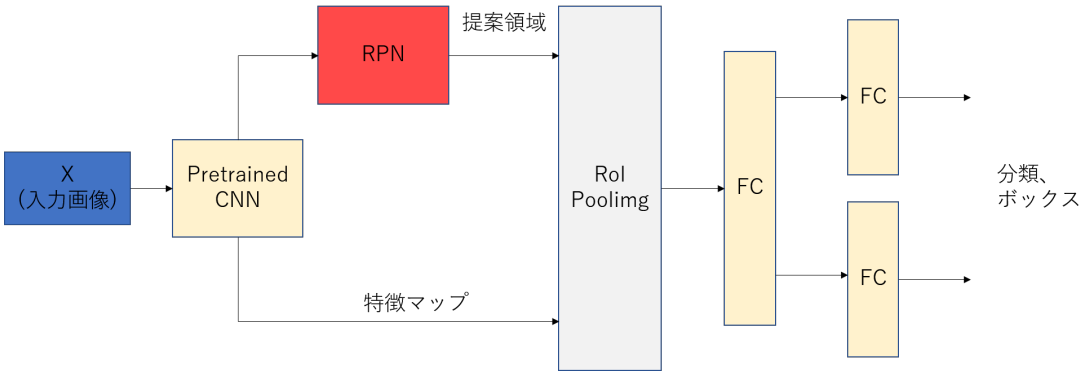

図 3.3: Faster R-CNN の構造

第 4 章

生成モデル

4.1 GAN

GAN(Generative Adversarial Nets) [**?**] とは生成モデルを訓練させるための手法であり、敵対的生成ネットワークとも呼ばれ 2014 年に考案された。ここでの生成モデルとは、訓練データを学習しそれに近いデータを生成するためのモデルである。つまり GAN の目的は訓練データのサンプル $\mathrm{xp}_{data}(x)$ の分布より $p_{data}(x) = p_g(z)$ となる分布を生成するモデルを得ることである。

GAN では生成モデルを得るために同時に識別モデルの学習も行うことで生成モデルの精度を高めている。

ここで GAN の二つのネットワークについて生成モデルを Generator、識別モデルを Discriminator と呼ぶ。GAN では生成モデル G と識別モデル D を競わせるように学習を行うことで互いに精度を高め合うようにできている。この二つのネットワークは互いに競い合うように訓練が行われるため敵対性とも呼ばれている。Generator のよって生成された画像を Discriminator が識別を行うが、この際 Discriminator は画像が生成されたデータか用意されたデータを正確に測ろうとし、Generator は Discriminator を欺こうと学習を行う。

G と D に対して GAN の目的関数を次のように表す。

$$\min_G \max_D V(D,G) = E_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}\big[\log D(\boldsymbol{x})\big] + E_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\big[\log(1 - D(G(\boldsymbol{z})))\big] \tag{4.1}$$

識別モデル D は入力されたデータが訓練データのものか生成モデルが生成したデータなのかの判別を行う。

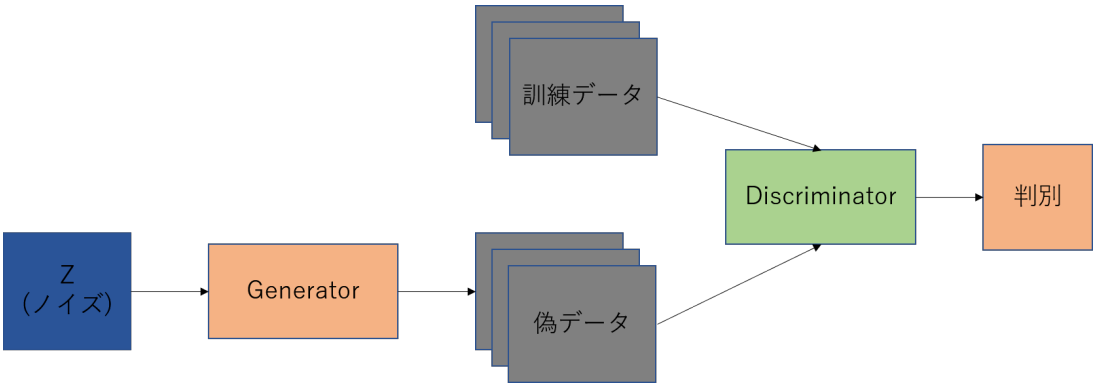

図 4.1: Faster R-CNN の構造

識別モデルの損失関数は次のように表せる。

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\boldsymbol{x}^{(i)}) + \log(1 - D(G(\boldsymbol{z}^{(i)}))) \right] \tag{4.2}$$

識別モデル D は訓練時、事前分布 $\mathrm{p}_z(\boldsymbol{z})$ サンプリングを行いデータ $\left\{ \boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)} \right\}$ を得る。次に生成データを得た後、訓練データセットから観測データ $\left\{ \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)} \right\}$ を抽出する。そしれ先の損失関数より勾配を計算しパラメータを更新している。

生成モデルの損失関数は以下のように表せる。

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[\log(1 - D(G(\boldsymbol{z}^{(i)}))) \right] \tag{4.3}$$

生成モデル G は訓練時、識別モデルと同様に事前分布 $\mathrm{p}_z(\boldsymbol{z})$ でサンプリングを行いデータ $\left\{ \boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)} \right\}$ を得て、先の損失関数で勾配計算をしてパラメータの更新を行っている。

4.2　Conditional GAN

CGAN(Conditional Generative Adversarial Nets) は GAN が発表されてすぐ年内に提案された生成モデルである。CGAN では GAN の Generator や Discriminator の概念や構造がベースになっている。GAN との違いは条件付が行われているところにある。CGAN の目的関数は次のように表される。

$$\min_G \max_D V(D, G) = E_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})} \left[\log D(\boldsymbol{x}|\boldsymbol{y}) \right] + E_{\boldsymbol{z} \sim p_z(\boldsymbol{z})} \left[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y}))) \right] \tag{4.4}$$

目的関数からわかるように、CGAN は GAN の目的関数に条件ベクトル y を加えたものになっていることが分かる。これらは Discreminator と Generator に画像の入力と同

時に教師データのラベルを入力して条件付けしているためである。通常の GAN では入力された画像データのみで訓練されるため、訓練データに近い分布をランダムで出力するのみにとどまる。しかし CGAN では入力データであるラベルの情報を入力するころで生成器がどういった画像を生成するかと識別機が何について正否の判別をするかをあらかじめ操作することができる。

CGAN は条件付けを行ったのみでそれ以降の学習では通常の GAN とほとんど同様である。

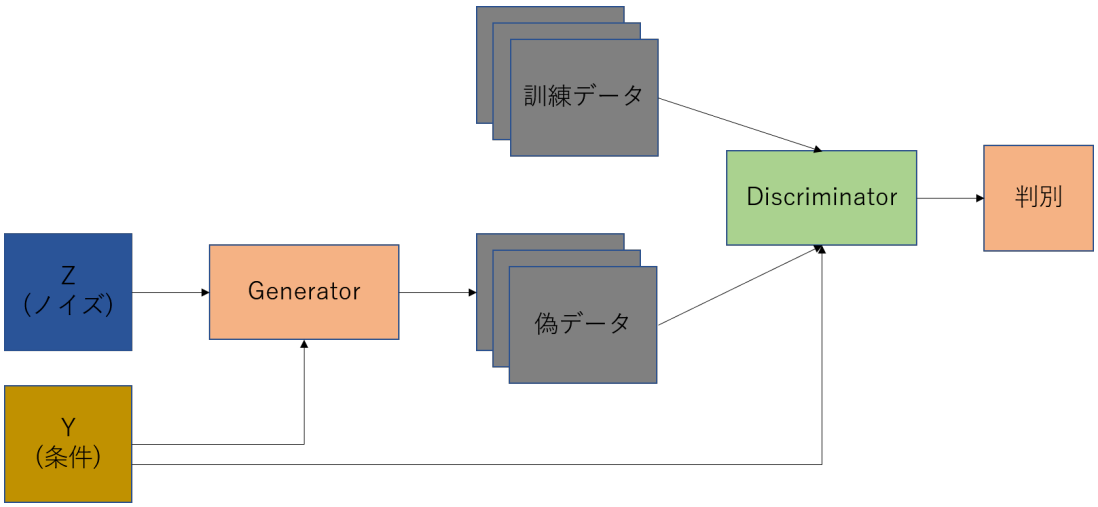

図 4.2: CGAN の構造

4.3　DCGAN

DCGAN [7] は 2015 年に通常の GAN よりも精度を向上させるものとして提案された手法である。オリジナルの GAN と DCGAN との主な違いはネットワークに全結合層を使っているか CNN を使っているかどうかの違いである。

DCGAN のアーキテクチャでも GAN と同様に Generator と Discriminator が存在しており、それらのネットワークでは CNN 層が使われている。

DCGAN では GAN の学習が安定しない問題に対し batch normalization が行われている。各々の層でデータ分布を正規化することによって学習速度向上を促したり過学習を防いでいる。また Discriminator の活性化関数には ReLU の代わりに Leaky ReLU が導入されている。ReLU は入力が 0 未満の場合出力の値を 0 とするため、f(x) = $\max(0, x)$ と表す。対して $LeakyReLU$ では 0 未満場合でも係数を掛けることによって

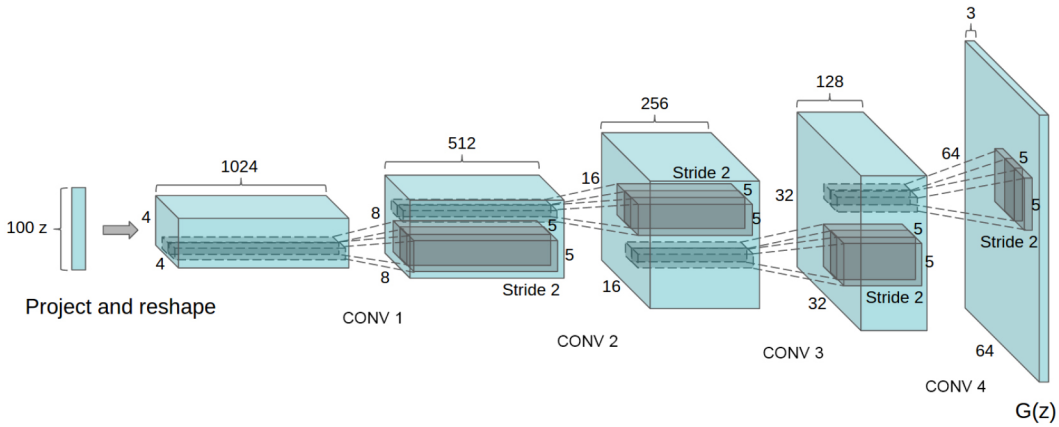

図 4.3: DCGAN

マイナスとして値が保存されるようにしている。これは $f(x) = \max(ax, x)$ と表せる。これによって計算した際勾配が 0 になり誤差逆伝搬による学習が終了してしまうことを回避している。

4.4　pix2pix

pix2pix [?] は 2018 年に提案された生成モデルの手法であり、そのベースとして先に述べた DCGAN、CGAN が用いられている。pix2pix という名前は「画素から画素へ (from pixel to pixel)」という意味ある。これは、通常の GAN では訓練時に 1 枚の画像データを仕様して学習を行うのに対して、pix2pix では 2 枚のペアになっている画像を用いて学習を行うことから名付けられている。ペアとなっている画像の一方は条件画像となっており、これはつまり CGAN で扱っていた条件ベクトル y の代わりに使われているということが分かる。

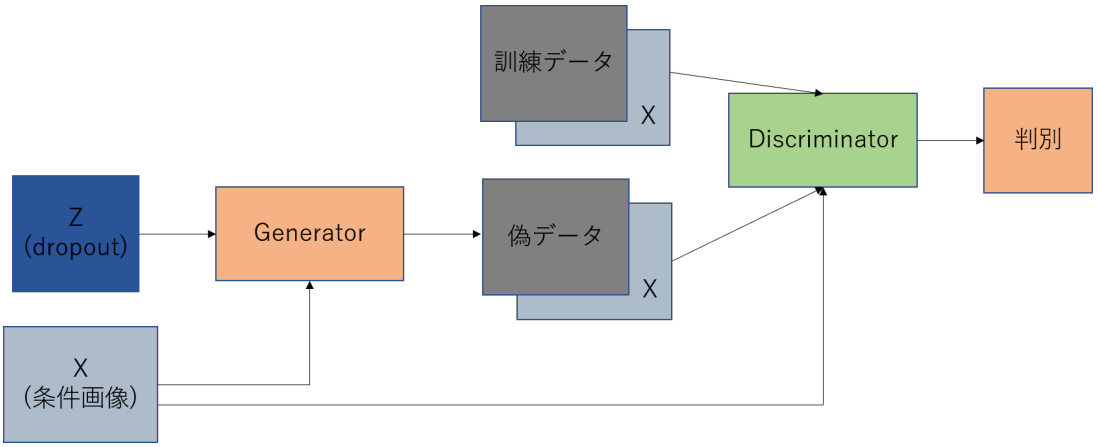

図 4.4: pix2pix の構造

pix2pix の Generator には U-Net [9] が使用されている。U-Net とは 2015 年に考案された Semantic Segmentation の手法のために用いられるネットワークである。Semantic Segmentation とは対象画像中の画素の情報を元に物体をピクセルレベルでカテゴリ分類する手法のことである。U-Net は全結合層を持たない左右対称の Encoder–Decoder 構造になっており、始めにエンコードで畳み込みを行った後デコードを行うが、その時にエンコード時に得たト特徴マップの一部を再使用している。エンコーダーの各層で出力された特徴マップをデコーダーの一部に連結するスキップ接続により物体の位置情報を保持することができる。

pix2pix では Generator に対するノイズ Z の入力を生成期の中間層にて dropout を行う事によって内的に与えている。dropout とは特定の層の出力の値を 0 に大体することである。これによって外的なノイズ Z の量を調整しなくても汎用的にノイズを与えることができるようになる。

損失関数は Discrimintor は通常の GAN と同じように表されるが、Generator は通常の GAN に L1 損失を加えたものになっている。L1 損失及び Generator の損失関数は次の通りである。

$$L_{\boldsymbol{L1}} = E_{x,y,z}\left[||\boldsymbol{y} - \boldsymbol{G}(\boldsymbol{x}, \boldsymbol{z})||\right] \tag{4.5}$$

$$G* = arg\min_{G}\max_{D} L_{\boldsymbol{cGAN}}(\boldsymbol{D}, \boldsymbol{G}) + \lambda\ L_{\boldsymbol{L1}(\boldsymbol{G})} \tag{4.6}$$

また Discrinminator では PatchGAN により画像全体をそのまま俯瞰して使うのでは

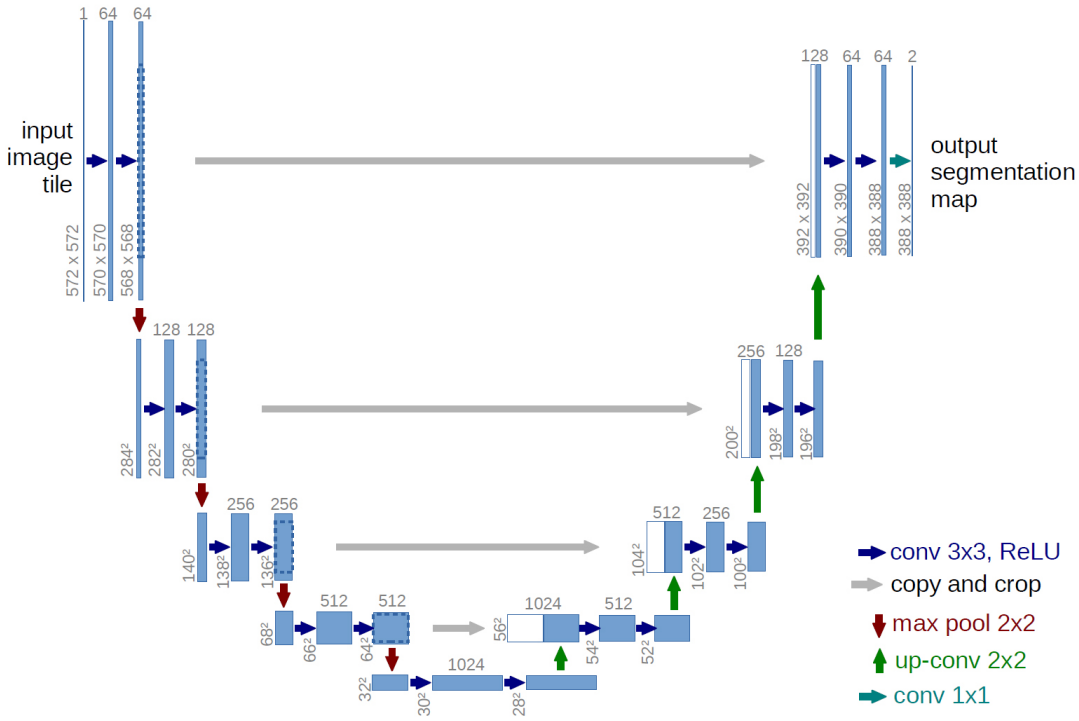

図 4.5: U-net

なく、入力画像をパッチ単位で分割してから学習識別を行っている。

4.5　CycleGAN

CycleGAN は 2017 年に提案された生成モデルの手法である。CycleGAN の入力画像は pix2pix 同様に image-to-image translation であり二枚の画像を用意して画像を生成する。pix2pix は画像は正解か不正解か分かるように画像がペアになっている必要のある教師あり学習である。しかし大量のペアになっている画像のデータセットを作成するのには時間がかかりそのコストが高いことが pix2pix の課題であった。一方 CycleGAN では画像がペアになっている必要がなく入力された画像を対象の画像へと変換することができる。すなわち教師なし学習を行う。CycleGAN は対象領域から目的領域への領域適応を行うことによって教師なし学習を実現している。つまり二つの異なる領域の画像から対応関係を導き学習を行っている。

CycleGAN は構造として通常の GAN の Discriminator と Generator を複数組み合わせた構造になっている。CycleGAN ではドメイン X の画像 x に対するドメイン Y の画像 y について、X から Y の写像を学習するだけでなく Y から X へ

の写像も学習を行う。それぞれの写像 G:X $\rightarrow$ Y と写像 $F : Y \rightarrow X$ と表す。これらの写像に対してドメイン Y の画像 y であるかドメイン X である画像 x から生成した画像 $G(x) = Y'$を識別する識別器 D_Y、一方ドメイン X の画像 x であるかドメイン Y である画像 y から生成した画像 $F(y) = X'$を識別する識別器 D_Xが存在する。生成器 G と識別器 D_Yについて敵耐性損失は次のようになる。

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)}[\log D_Y(y)] + E_{x \sim p_{data}(x)}[\log 1 - D_Y(G(x))] \quad (4.7)$$

この式の目的は G の最小化と D_Yの最大化 $\min_G \max_{D_Y} \mathcal{L}_{GAN}(G, D_Y, X, Y)$ である。どうように生成期 F と識別器 D_Xについて敵耐性損失は次のようになる。

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = E_{x \sim p_{data}(x)}[\log D_X(x)] + E_{y \sim p_{data}(y)}[\log 1 - D_X(F(y))] \quad (4.8)$$

同様にに $\min_F \max_{D_X} \mathcal{L}_{GAN}(F, D_X, X, Y)$ を求められる。

通常の GAN のように生成器と識別器を各々学習してもうまく相互に影響を与えることができない。そこで CycleGAN では次のようなサイクル一貫性損失が考えられている。サイクル一貫性損失では変換した画像を逆変換した時の一貫性を保つために学習を行う。

$$\mathcal{L}_{cyc}(G, F) = E_{x \sim p_{data}(x)}||F(G(x)) - x||_1 + E_{y \sim p_{data}(y)}||G(F(y)) - y||_1 \quad (4.9)$$

第 5 章

提案手法

関連研究である [3] では CycleGAN を用いて昼間のデータセットと夜中のデータセットから学習して偽のデータセットを生成している。生成したデータセットによって自動車の物体検出の学習をしデータセットごとにモデルを評価している。

本実験ではそれらの方法とアイデアを踏襲して、自動車の物体検出について好天候時のデータセットと悪天候時データセットを利用してデータ拡張を行い物体検出が可能かどうかを実験する。

好天候時のデータと悪天候時のデータセットについては BDD100K のものを採用する。BDD100K は前方車載カメラによって撮られた大量の自動車画像の及びそれらをアノテーションした情報を含むデータセットである。BDD100K のデータセットは様々な状況設定によってカテゴライズされており、悪天候時や好天候時、夜中や昼間の時間帯、高速道路や一般道路などで分けることができる。またアノテーション情報については物体のバウンディングボックスだけではなく物体のセグメンテーション情報や走行エリアのセグメンテーションなども含んでいる。BDD100K のデータセットは下記の場所で公開されている。が、ユーザー登録が必要であるが、無料で手に入れることができる。
https://bdd-data.berkeley.edu/portal.html

CycleGAN のプログラムに好天候時のデータセットと悪天候時の双方を入力することで、それぞれのドメインに適応した生成器を作成することができる。そこで悪天候時の領域に適応させる画像を生成する Generator を用いて偽の悪天候時に画像データセットを作成する。これらのデータセットには本来アノテーション情報がないが、CycleGAN の生成器でつくった偽の画像は入力した画像の形状を保持している。したがって入力画像のアノテーションデータをそのまま生成した画像に対応させることができる。

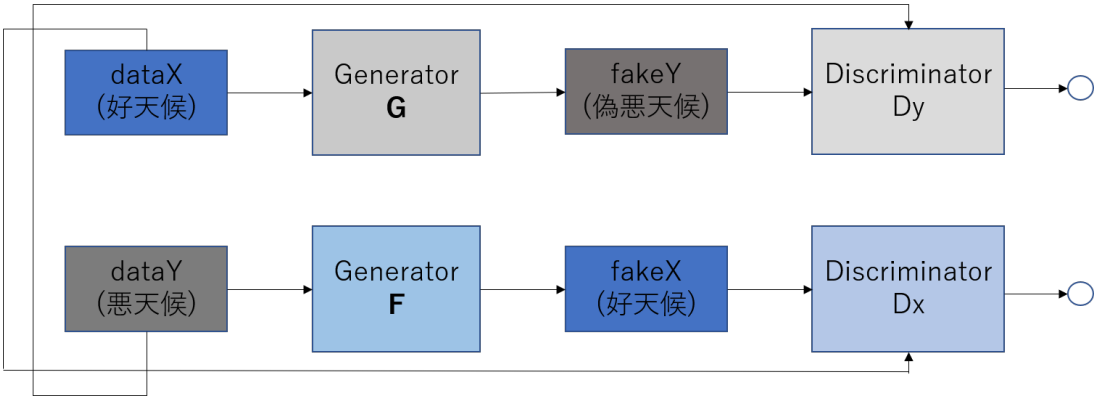

図 5.1: 本実験での CycleGAN の抽象図

生成した偽のデータと用意していたデータから実際に物体検出が可能かどうかを調べるために、それらのデータセットを使用して物体検出器の学習を行う。本実験では物体検出器に入力するデータを 3 種類に分けて学習を行い 3 つのモデルを作成した。入力するデータセットは、あらかじめ用意していた好天時のデータセット、CycleGAN で生成した偽の悪天候時のデータセット、好天時のデータと偽の悪天候時のデータを組み合わせたデータセットの 3 回に分けて学習を行わせる。

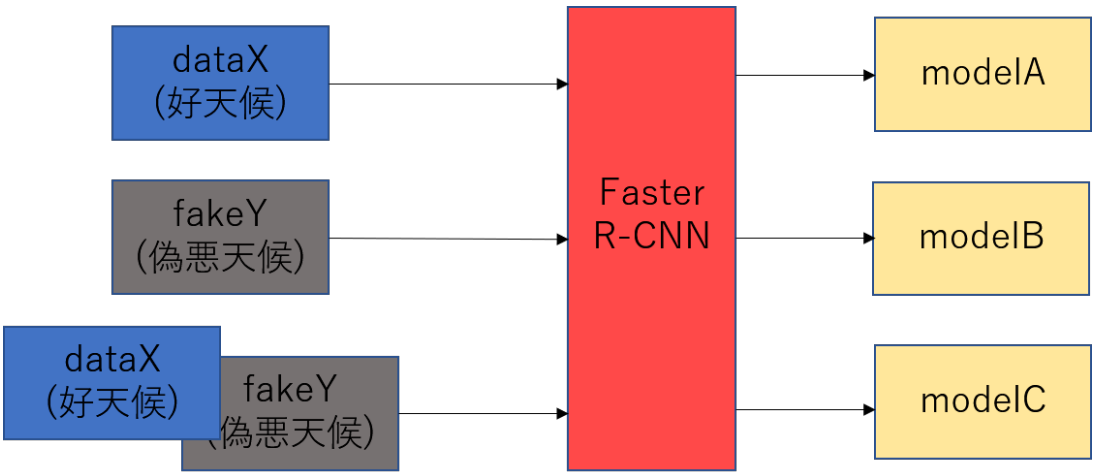

図 5.2: 提案手法の概要図

第 6 章

実験

6.1 実験方法

CycleGAN によって学習を行うにあたり CycleGAN 論文の原著 [5] である Jun-Yan Zhu 氏が Github 上で CycleGAN のプログラムを公開している。以下はその URL である。

https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

これらのプログラムを github 上からローカルの環境へと clone することで CycleGAN を利用することができる。今回のプログラムを動かしたマシンの実行環境は次の通りである。

・Ubuntu 18.04.1

・python 3.6.9

・pytorch 1.6.0

・GPU GeForce RTX 2060

・CUDA 9.1.

このプログラムに使うデータセットを用意する際、デフォルトの場合だと 4 つのファイルを用意する必要がある。それぞれ train 用のデータセット 2 種類と test 用のデータセットを用意し、特にプログラムに変更を書き換えない場合ファイル名を”trainA”、”trainB”、”testA”、”testB” にする。本実験の場合だとファイル”trainA に訓練用の好天時データセット、ファイル”trainB” に訓練用の悪天候時データセット、ファイル”testA” にテスト用の好天のデータセット、ファイル”testB” にテスト用の悪天候時データセットを用意した。今回は学習を行った後に好天時のデータセットから悪天

候時領域のデータをつくることが目的であるため、先に用意したファイル”testB” は実際には使用しなかった。

はじめの状態であれば入力画像と出力画像のサイズは 256*256 で固定されている。このサイズはオプションから値を変更することができ、それによって任意の画像サイズで CycleGAN でデータを扱うことができる。しかし、サイズを大きくしすぎてしまうと学習に膨大な時間がかかるため現実的ではない。したがって、BDD100K で提供されている画像のサイズは 1280*720 であるが、画像のサイズの方を 256*256 へとリサイズを行うデフォルトの設定で学習を行うことにした。

あらかじめ用意されている train.py をコマンドラインに入力する際使用するデータセット名と出力するときのデータセットの名前を引数として入力する。学習終了後に二つの生成モデルが出力されるため、用途に合うモデルを test.py のコマンドラインで指定して使用することによって必要な偽データを生成することができる。このとき生成されるデータ数は 50 となっているため、より多くのデータが欲しい場合はオプションの方から変更する。

次に CycleGAN によって生成したデータセットを用いて物体検出を行う。物体検出のプログラムは GoogleColaboratory 上で作成した。GoogleColaboratory は Google が提供している python の実行環境で、ブラウザ上で無料宇で GPU を使うことができる。また GoogleDrive 上のファイルを参照してプログラムを動かすことができるため、先に作成したデータセットをを GoogleDrive 上にアップロードしておけば物体検出の学習データとして使用する事ができる。

物体検出器を実装する際にネットワークについて”fasterrcnn$_r esnet50_f pn''$を採用した。'''$fasterrcnn_r esnet50_f pn''$は $torchvision$ によって用意されている学習済の $FasterR-CNN$ モデルである。

検出器で学習する際クラスラベルは次のように設定した。

['person', 'traffic light', 'train', 'traffic sign', 'rider', 'car', 'bike', 'motor', 'truck', 'bus']

6.2　実験データ

実験データとして使用するために BDD100K の画像データセットから好天時領域の画像 2000 枚と悪天候時領域の画像 2000 枚を入力する。ここで BDD100K の画像データはサイズが 1280*720 であるが、このまま CycleGAN に入力して学習をすると時間がかかりすぎてしまうため、画像のサイズを 256*256 へとリサイズする。また出力される画像サイズも 256*256 である。CycleGAN で生成する偽のデータ数はオプションで 200 枚に設定しておく。

物体検出器で学習をするとき、データセットを次のように用意する。

・好天時のデータセット (200 枚)

・悪天候時の偽のデータセット (200 枚)

・好天時のデータセット＋悪天候時のデータセット (400 枚)

以上 3 種類のデータセットより 3 つの検出モデルを作成する。また BDD100K で用意されているアノテーションデータは PasvalVOC 形式に変換した。この際アノテーション元の画像は 256*256 にリサイズされているため、アノテーション中のバウンディングボックスの座標や画像サイズをそのまま使うことはできない。したがって座標とサイズの値について x 方向では 256/1280 倍にし、y 軸方向では 256/720 倍に変更することでリサイズ後の画像に対応させた。

PascalVOC 形式に変更した後のアノテーションの構造は次のようになっている。

```
<annotation>
        <folder>○○</folder>
        <filename>○○</filename>
        <size>
                <width>○○</width>
                <height>○○</height>
                <depth>○○</depth>
        </size>
        <weather>○○</weather>
        <scene>○○</scene>
        <timeofday>○○</timeofday>
        <object>
                <name>○○</name>
                <occluded>○○</occluded>
                <truncated>○○</truncated>
                <trafficLightColor>○○
</trafficLightColor>
                <bndbox>
                        <xmin>○○</xmin>
                        <ymin>○○</ymin>
                        <xmax>○○</xmax>
                        <ymax>○○</ymax>
                </bndbox>
        </object>
        <object>
                        .
                        .
                        .
```

図 6.1: PascalVOC 形式の構造

6.3　実験結果

CycleGAN によって偽データが生成された。次の図はうまく生成ができた画像の例である。図上部の好天時の画像をもとに悪天候時領域の画像を生成している。

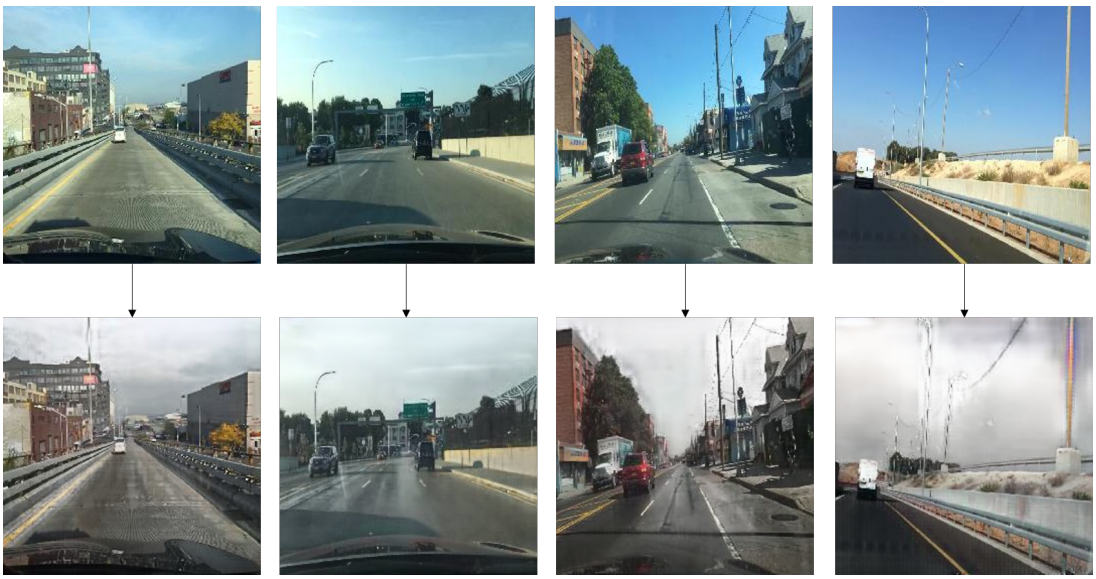

図 6.2: CycleGAN によって生成した画像

しかしすべての画像がうまく変換できたわけではない。下記の画像のように、画像に何らかの影響を与えているが悪天候とは言えないような画像が生成されている場合が見受けられた。

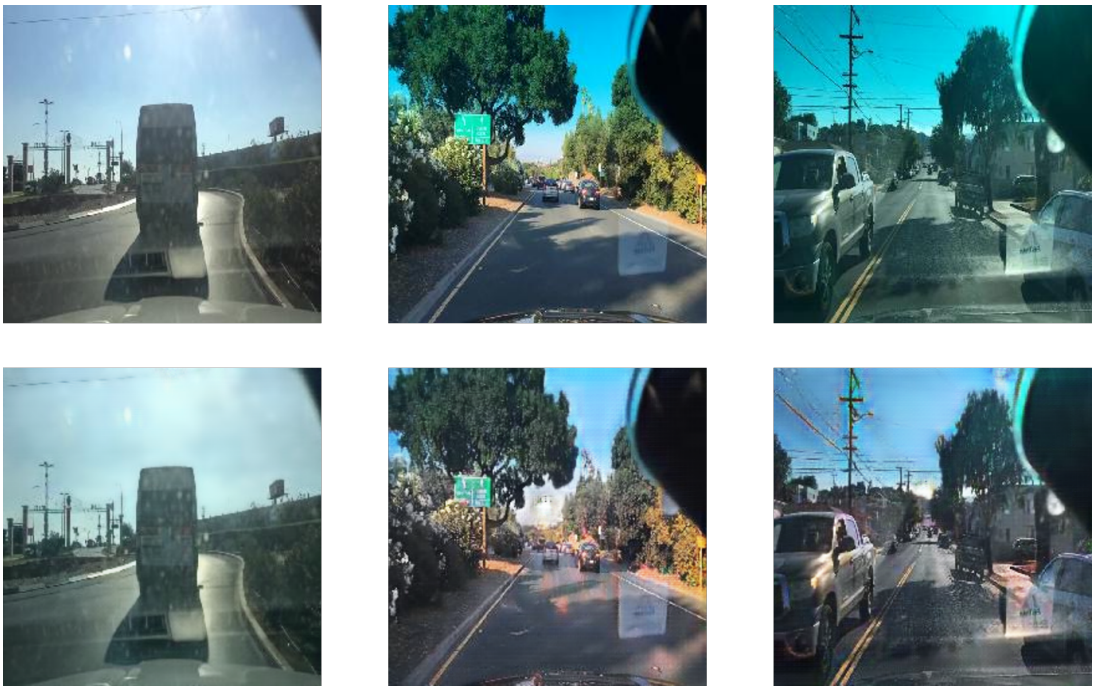

図 6.3: CycleGAN によってうまく生成できなかった画像

先述したそれぞれのモデルで物体検出器を学習した結果次のように物体が検出されることを確認した。

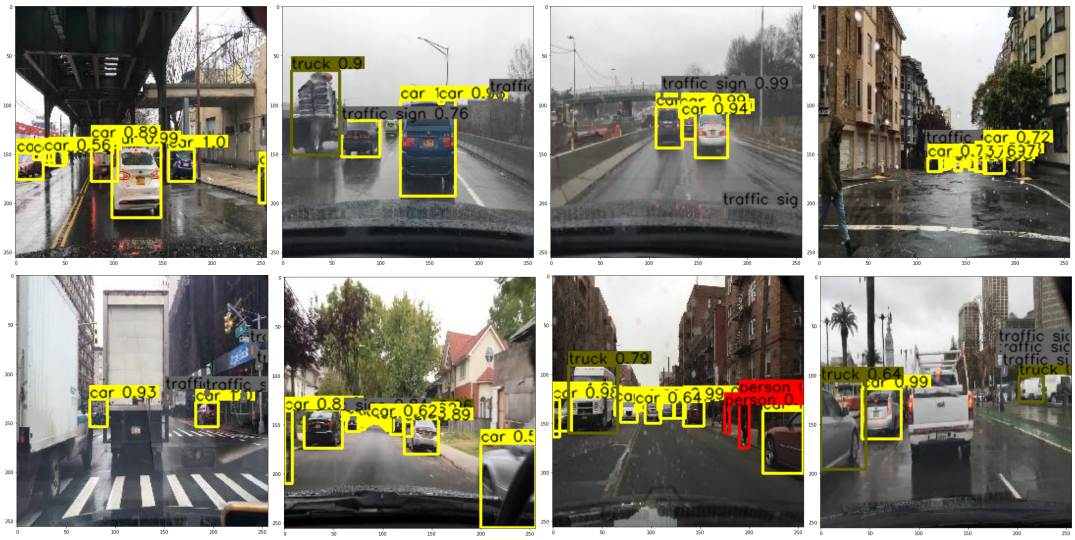

図 6.4: 好天時データによって訓練したモデル

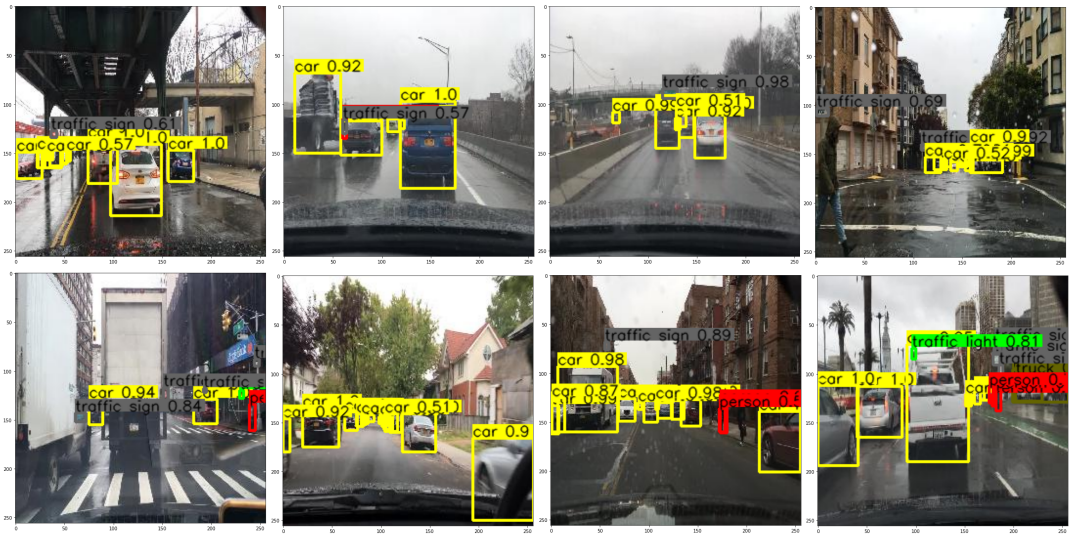

図 6.5: 偽の悪天候時データによって訓練したモデル

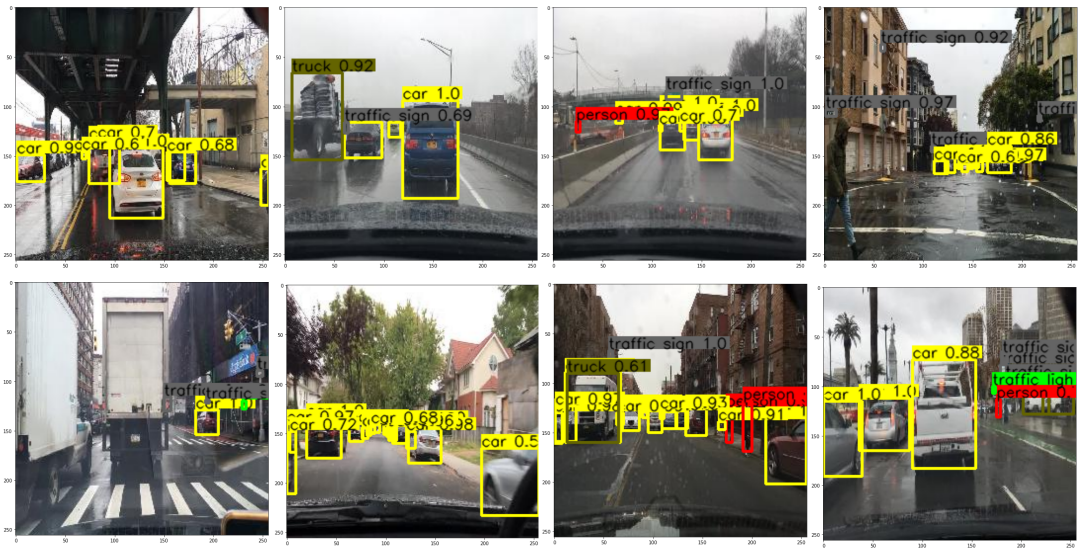

図 6.6: 好天時のデータ + 偽の悪天候時データによって訓練したモデル

第 7 章

考察

CycleGAN によって好天時の画像データから悪天候時の画像データを作成した際にすべての画像データがうまく対象領域の画像へと変換されているわけではなかった。それでも偽の画像データをもとに学習した結果では物体検出自体は行うことができた。しかし検出結果を見るとそれほど高い精度で検出できているわけではないと考えられる。今回の実験では 200 枚という少ないデータセットで物体検出器の学習を行ったが、より精度を高めるためにはより多くの画像データセットを用いて学習を行う必要があると考えられる。また、生成器で作成した画像データが完全に正確ではなかったことから、用意した偽の画像データセットから不完全なデータを取り除くことでも精度を上げることができると考えられる。

また精度を上げるために学習時の epock 数を増やすことで検出器の精度を高めることができると考えられる。

しかし画像のデータセットの量や epock 数を増やして精度向上には限界があり、データ数が大きくなりすぎたときの学習コストと見合わないことも考えられる。そのためデータ量とコストの釣り合いを考えながら学習を行う必要がある。

今回の実験では評価指標を利用せず結果の出力を物体の検出のみでとどめため、三つのモデルのうちどのモデルの精度が高いかということはわからなかった。したがって 3 つのモデルについて比較することは難しいため、これらのモデルの精度を詳しく測るための評価指標としては先行研究でも利用されていた mAP などがを取り入れる必要がある。今回の実験では評価指標を実装することができなかったが、より正確に精度を測定するためにも今後の実験では取り入れるべきである。

第 8 章

結論

本研究では CycleGAN によって好天候時のアノテーションデータを利用できる偽の悪天候時のデータセットの生成を行った。生成したデータセットをもとに物体検出を行うことはでき、データを拡張することに対しては有効であることが分かった。しかし今回学習した検出器のモデル毎の精度がどれほどまで高いかの判別をすることはできなかった。これは主に評価指標の実装ができなかったためである。また精度に関しても学習が不十分であったりデータ数が少ないなどのことから、改善の余地は多くあることが分かる。結論として実験により生成した偽データを利用すること自体は可能であることが分かったがどの程度まで友好的であったかを示すことはできなかった。

謝辞

本論文の最後に、日頃から多くのご指導をしていただいた新納浩幸教授に心からの感謝をいたします。また研究をするにあたり助言をいただいた新納研究室の皆様に多くの感謝をいたします。

参考文献

[1] Alex Krizhevsky Ilya Sutskever Geoffrey E. Hinton"'ImageNet Classification with Deep Convolutional Neural Networks'(2012)

[2] Karen Simonyan Andrew Zisserman"Very Deep Convolutional Networks for Large-Scale Image Recognition"(2015)

[3] Vinicius F. Arruda, Thiago M. Paixão, Rodrigo F. Berriel, Alberto F. De Souza, Claudine Badue, Nicu Sebe, Thiago Oliveira-Santos"Cross-Domain Car Detection Using Unsupervised Image-to-Image Translation: From Day to Night"(2019)

[4] Ian J. Goodfellow, Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio"Generative Adversarial Nets"(2014)

[5] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros"Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks"(2017)

[6] Mehdi Mirza, Simon Osindero "'Conditional Generative Adversarial Nets'(2014)

[7] Alec Radford, Luke Metz, Soumith Chintala "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"(2015)

[8] 'Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros'Image-to-Image Translation with Conditional Adversarial Networks"(2018)

[9] Olaf Ronneberger, Philipp Fischer, Thomas Brox"U-Net: Convolutional Networks for Biomedical Image Segmentation"(2015)

付録

A　プログラム

以下では実験中に使用したプログラムについて記す。次のプログラムは CycleGAN によって得られた画像を入力して物体検出を行ったものである。これらのプログラムは GoogleColab 上で動作していることを確認している。

下記のクラスは xml ファイルからラベル、バウンディングボックスの情報を取り出してリスト化を行っている。ここで扱う xml ファイルは BDD100K のデータを PascalVOC 形式に変更したものである。ここで値を取り出してリスト化をしているのでわざわざ BDD100K のデータセットで用意されていたアノテーションファイルを別の形式に変換する必要はないが、公開されているプログラムでは PascalVOC 形式のデータセットを扱っているものも多いので一度変換しておくと後々融通が利くようになる。

ソースコード A.1: xlmload

```
1  class xml2list(object):
2
3      def __init__(self, classes):
4          self.classes = classes
5
6      def __call__(self, xml_path):
7
8          ret = []
9          xml = ET.parse(xml_path).getroot()
10
11         boxes = []
12         labels = []
13         zz=0
14
```

```
15          for zz,obj in enumerate(xml.iter('object')):
16
17              label = obj.find('name').text
18
19              if label in self.classes :
20                  bndbox = obj.find('bndbox')
21                  xmin = int(bndbox.find('xmin').text)
22                  ymin = int(bndbox.find('ymin').text)
23                  xmax = int(bndbox.find('xmax').text)
24                  ymax = int(bndbox.find('ymax').text)
25                  boxes.append([xmin, ymin, xmax, ymax])
26                  labels.append(self.classes.index(label))
27              else:
28                  continue
29
30          num_objs = zz +1
31
32          anno = {'bboxes':boxes, 'labels':labels}
33
34          return anno,num_objs
```

下記のクラスではデータセットの作成を行っている。入力したい画像の ID を先述のリストから読み込み image に格納している。ここでの scale は画像のサイズを指しており、入力する画像が 256*256 なのであらかじめ scale=256 としておく。ボックスやリストのデータを扱うためにデータをテンソルの形式に保存している。辞書型の変数である target 内について、"'boxes' はバウンディングボックスの座標、"labels" はラベル、"image$_i d''$は画像の名前、$''area''$はバウンディングボックスの内側の領域である。$''iscrowd''$について、この値が $True$ である場合インスタンスの評価が無視されるが、

$$\text{iscrowd} = \text{torch.zeros((records.shape[0],), dtype=torch.int64)}$$

によって 0 配列になっている。

ソースコード A.2: Mydataset

```
1 class MyDataset(torch.utils.data.Dataset):
2
3         def __init__(self,image_dir,xml_paths,scale,classes):
4
5             super().__init__()
```

```
6             self.image_dir = image_dir
7             self.xml_paths = xml_paths
8             self.image_ids = sorted(glob('{}/*'.format(xml_paths)))
9             self.scale=scale
10            self.classes=classes
11
12        def __getitem__(self, index):
13
14            transform = transforms.Compose([
15                                            transforms.ToTensor()
16            ])
17
18            image_id=self.image_ids[index].split("/")[-1].split(".")
                  [0]
19            image = Image.open(f"{self.image_dir}/{image_id}.jpg")
20
21
22            t_scale_tate=self.scale
23
24            ratio=t_scale_tate/image.size[1]
25
26            t_scale_yoko=image.size[0]*ratio
27            t_scale_yoko=int(t_scale_yoko)
28
29            image = image.resize((t_scale_yoko,t_scale_tate))
30            image = transform(image)
31
32            transform_anno = xml2list(self.classes)
33            path_xml=f'{self.xml_paths}/{image_id}.xml'
34
35
36            annotations,obje_num= transform_anno(path_xml)
37
38            boxes = torch.as_tensor(annotations['bboxes'], dtype=torch
                  .int64)
39            labels = torch.as_tensor(annotations['labels'], dtype=
                  torch.int64)
40
41            boxes=boxes*ratio
42
43            area = (boxes[:, 3]-boxes[:, 1]) * (boxes[:, 2]-boxes[:,
```

```
                0])
44              area = torch.as_tensor(area, dtype=torch.float32)
45
46              iscrowd = torch.zeros((obje_num,), dtype=torch.int64)
47
48              target = {}
49              target["boxes"] = boxes
50              target["labels"] = labels+1
51              target["image_id"] = torch.tensor([index])
52              target["area"] = area
53              target["iscrowd"] = iscrowd
54              return image, target,image_id
55
56          def __len__(self):
57
58              return len(self.image_ids)
```

下記の関数ではデータローダーの作成を行っている。データセットはすべてのデータを一律に保持しているものであるが、ここで作成するデータローダーはデータセットの中身をミニバッチ毎に固めたものである。10 行目で使用している関数”torch.utils.data.DataLoader” では 4 つの引数がある。第一引数では先に作成したデータセットをいれる。第二引数ではバッチサイズを入力する。バッチサイズは訓練あるいはテスト時にデータをいくつ使用するかを定めるものである。したがってバッチサイズはデータセット数で割り切れる値にする必要がある。本研究では $\mathrm{batch}_size = 100$ としてある。第三引数ではデータの参照順をランダムにするかどうかを決める。ここでは $shuffle = True$ となっているためデータの並びはランダムになる。第四引数の $collate_fn$ はあらかじめ定めた関数の戻し値のとおりに $batch$ 出力を行うということを示している。

ソースコード A.3: dataloader

```
1 def dataloader (data,dataset_class,batch_size,scale=256):
2     xml_paths=data[0]
3     image_dir1=data[1]
4     dataset = MyDataset(image_dir1,xml_paths,scale,dataset_class)
5
6     torch.manual_seed(0)
7     def collate_fn(batch):
```

```
8         return tuple(zip(*batch))
9
10    train_dataloader = torch.utils.data.DataLoader(dataset, batch_size
          =batch_size, shuffle=True,collate_fn=collate_fn)
11
12    return train_dataloader
```

モデルは torch が公開しているモデルである”fasterrcnn$_resnet50_fpn$”を使用する。6 行目の $model.roi_heads.box_predictor$ ではモデルでの分類器の出力数と座標の出力数を調整している。$print(list(model.children())[-1])$ を行いモデルの構造を見ると以下のようになっている。

ソースコード A.4: 出力結果

```
RoIHeads(
  (box_roi_pool): MultiScaleRoIAlign()
  (box_head): TwoMLPHead(
    (fc6): Linear(in_features=12544, out_features=1024, bias=True)
    (fc7): Linear(in_features=1024, out_features=1024, bias=True)
  )
  (box_predictor): FastRCNNPredictor(
    (cls_score): Linear(in_features=1024, out_features=91, bias=True)
    (bbox_pred): Linear(in_features=1024, out_features=364, bias=True
        )
  )
)
```

出力結果を見てわかる通り、クラス数 91、ボックスの座標数 364 での出力をする形になっている。これらを調整するために次のように model.roi$_heads.box_predictor$ を操作をする。

ソースコード A.5: model

```
def model ():

    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(
        pretrained=True)
    num_classes=len(dataset_class)+1
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
        num_classes)
```

```
7     return model
```

ここでもう一度モデルの構造を確認すると出力の値が調整されていることが分かる。

ソースコード A.6: 出力結果

```
1 RoIHeads(
2   (box_roi_pool): MultiScaleRoIAlign()
3   (box_head): TwoMLPHead(
4     (fc6): Linear(in_features=12544, out_features=1024, bias=True)
5     (fc7): Linear(in_features=1024, out_features=1024, bias=True)
6   )
7   (box_predictor): FastRCNNPredictor(
8     (cls_score): Linear(in_features=1024, out_features=11, bias=True)
9     (bbox_pred): Linear(in_features=1024, out_features=44, bias=True)
10  )
11 )
```

下記のプログラムでは物体検出での訓練を行う。epoch はあらかじめ 100 に設定している。

ソースコード A.7: train

```
1 model=model()
2 params = [p for p in model.parameters() if p.requires_grad]
3 optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9,
      weight_decay=0.0005)
4 epochs = 100
5
6 device = torch.device('cuda') if torch.cuda.is_available() else torch.
      device('cpu')
7 model.cuda()
8 model.train()
9
10 loss_list=[]
11 for epoch in range(epochs):
12     loss_epo=[]
13
14     for i, batch in enumerate(train_dataloader):
15
16         images, targets, image_ids = batc
17         print(image_ids)
```

```
18          images = list(image.to(device) for image in images)
19          targets = [{k: v.to(device) for k, v in t.items()} for t in
                targets]
20
21          loss_dict= model(images, targets)
22          losses = sum(loss for loss in loss_dict.values())
23          loss_value = losses.item()
24
25          optimizer.zero_grad()
26          losses.backward()
27          optimizer.step()
28
29          loss_epo.append(loss_value)
30
31          print(image_ids)
32          if (i+1) % 10== 0:
33            print(f"epoch␣#{epoch+1}␣Iteration␣#{i+1}␣loss:␣{loss_value}"
                  )
34
35      loss_list.append(np.mean(loss_epo))
36      torch.save(model, path+'/colab_frcnn-main/model.pt')
```

下記のプログラムではモデルのテストを行い openCV を使って物体検出の結果表示を行う。

ソースコード A.8: test

```
1
2 data_class=dataset_class
3 data_class.insert(0, "__background__")
4 classes = tuple(data_class)
5 model=torch.load(path+'/colab_frcnn-main/model_real100.pt')
6
7 model.to(device)
8
9 model.eval()
10 for imgfile in sorted(glob.glob(test_path+'/*')):
11
12      img = cv2.imread(imgfile)
13      img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
14      image_tensor = torchvision.transforms.functional.to_tensor(img)
15
```

```
16      with torch.no_grad():
17          prediction = model([image_tensor.to(device)])
18
19      for i,box in enumerate(prediction[0]['boxes']):
20          score = prediction[0]['scores'][i].cpu().numpy()
21          if score > 0.5:
22              score = round(float(score),2)
23              cat = prediction[0]['labels'][i].cpu().numpy()
24              txt = '{}␣{}'.format(classes[int(cat)], str(score))
25              font = cv2.FONT_HERSHEY_SIMPLEX
26              cat_size = cv2.getTextSize(txt, font, 0.5, 2)[0]
27              c = colors[int(cat)]
28              box=box.cpu().numpy().astype('int')
29              cv2.rectangle(img, (box[0], box[1]), (box[2], box[3]), c
                    , 2)
30              cv2.rectangle(img,(box[0], box[1] - cat_size[1] - 2),(box
                    [0] + cat_size[0], box[1] - 2), c, -1)
31              cv2.putText(img, txt, (box[0], box[1] - 2), font, 0.5,
                    (0, 0, 0), thickness=1, lineType=cv2.LINE_AA)
32
33
34      plt.figure(figsize=(15,10))
35      plt.imshow(img)
36  plt.show()
```