ニューラルネットワークの特徴マップの解析と活用
(Analysis and Applications of Feature Map of Deep Neural Networks)

井手　秀徳

学位取得年月　2021 年 3 月

あらまし

脳の視覚野を模倣して作られた畳み込みニューラルネットワーク (CNN) は, 画像分類の分野などで注目されている手法である. 階層構造を持つ CNN による情報処理の本質は, 画像などの入力データに内在する重要な情報 (特徴) を, データからの学習によって獲得することである. 本質に近い特徴を獲得するためによく試みられるアプローチは, CNN の層を増やす (深くする), 変数の数を増やすなどである. しかし, それらは, 学習に用いられるデータ (学習サンプル) を CNN が過度に学習してしまうことで逆に性能が悪化する, 値の発散により学習が失敗するといった多くのリスクも伴う. 他にも, CNN の実行に要する莫大な計算コストも問題になる. 本論文では, CNN が獲得すべき特徴について議論し, その獲得を促進することで学習の安定化や分類精度向上を試みるだけでなく, 獲得した特徴を活用することで, CNN の計算コスト削減も目指す.

第 2 章では, CNN の中間層が獲得すべき特徴について議論し, その獲得を促進し, 学習を安定させる正則化手法を提案する. CNN は, 複数の層からなる中間層と, 中間層で得られた特徴を用いて分類問題を解く出力層 (分類器) で構成されている. CNN の各層, 特に, 入力に近い層は脳の視覚野に近い機能を持つため, 学習時に視覚野の形成を促進できれば, より良い特徴を獲得できると示唆されている. スパースコーディングを用いると V1 視覚野の受容野を自己組織化できることから, 本章では, 中間層で獲得する特徴に対してスパース制約を課すスパース正則化を提案した. スパース正則化を様々な CNN に適用することで, 学習が安定し, 分類精度も向上することを実験的に確認した.

第 3 章では, CNN の分類器にとって好ましい特徴について議論し, その獲得を促進する正則化手法を提案する. K クラスの分類問題を解く分類器は, K 個のニューロンで構成され, ニューロンごとに分類すべきクラス (ターゲットクラス) が割り当てられる. サンプルが CNN に入力されると, ソフトマックス関数を介して, そのサンプルがターゲットクラスに属する事後確率をニューロンごとに出力し, その確率が最も高いニューロン (クラス) を分類結果とすることで, 分類器は K クラス分類問題を解いている. 分類器全体では K クラス分類問題を解くのに対し, 各ニューロンはターゲットクラスかそれ以外 (非ターゲットクラス) かの 2 クラス分類問題を解く. このとき, 各ニューロンにとって好ましい特徴は, ターゲットクラス, もしくは非ターゲットクラスの情報を含んだ 2 つのガウス分布 (分散は等しく, 平均は異なる) である. 本章では, そのような特徴の獲得を促進するために, 各ニューロンに対する正則化として判別基準を適用する判別正則化を提案した. 判別正則化を適用すると, そのような特徴が獲得されやすくなり, 分類精度も向上することを確認した.

第 4 章では, CNN が獲得した特徴マップを活用することで, その計算コストを削減する新たな Pruning 手法を提案する. 様々な Pruning 手法が提案されているが, その本質は, CNN が獲得した特徴の, 分類問題に対する貢献度合い (重要度) を評価し, 重要度が低い特徴, ならびに, 関連する変数の剪定 (Pruning) である. これは, Pruning で活用すべき情報は獲得された特徴であることを示唆している. 各特徴の分類に対する重要度 (Pruning Score) は, 正解が割り当てられた学習サンプルを用いて経験的分類損失を計算し, それを特徴マップに関してテイラー展開することで近似できる. しかし, 経験的分類損失は一部の極端なサンプルに大きく影響されるため, Pruning Score が信頼できない可能性がある. 本章では, テイラー級数を代数的に変換して導出した関数を用いる Pruning 手法を提案した. これにより, 一部の極端なサンプルによる悪影響を減らしつつ, 特徴の有無が分類損失に与える影響を直接評価 (予測) することができる. また, その延長として, Pruning Score を正規化する新たな正規化手法も提案した. 他にも, Pruning を補助変数の連続最適化問題に置き換え, それを安定して解く Pruning 手法も提案した.

目次

1 序論

1.1 研究の背景と位置づけ

人間の脳は, 経験を積むことで, 状況を理解し判断を下すという知的な活動を行うための不可欠な情報処理装置である. そのような知的活動を行うシステムを人工的に作り出すことが, 情報処理技術の究極的な目的の一つである. 深層学習 (Deep Learning) に代表される現在の情報処理技術は, 画像認識の分野において, その知的な活動の初歩的な部分を実現しつつある. その中心的役割を果たしている技術の一つは, 脳の視覚野を模倣して作られた畳み込みニューラルネットワーク (CNN) [1, 2] である.

CNN は, ILSVRC 2012 という画像分類の精度を競うコンテストにて, 従来手法より高い分類性能を達成したことで, 大きく注目された [3]. CNN が行なっている情報処理の本質は, 画像などの入力データに内在する重要な情報を捉えた情報表現 (特徴) を, データからの学習によって獲得することである. 脳の視覚野が行う情報処理と同様に, CNN も階層構造をもち, 入力に近い層では輪郭などの単純な特徴を, 出力に近い層では意味情報などの複雑な特徴を獲得している [4]. ここで得られる特徴がデータに内在する本質的な情報に近いものであるほど, 画像分類などのタスクにおいて, CNN は高い性能を示すと考えられる. より本質に近い, 良い特徴を入力データから獲得するためにしばしば試みられるアプローチは, CNN の層を増やす (深くする), 変数の数を増やすなどである [3, 5].

しかし, 層が深く, 変数の数が多い CNN は, 多くのリスクを伴う. よく知られているのは, 学習に用いられるデータサンプル (学習サンプル) を CNN が過度に学習してしまう過学習である. CNN を含めたニューラルネットワークは, 十分な数の変数があれば任意の出力が得られるため [6], 学習サンプルの数に対して変数が多すぎる場合に, このような問題が引き起こされる. 極端な例では, 全ての学習データを CNN が暗記することで, 正解をランダムにした分類問題でさえも解くことさえできた [7, 8]. このような CNN は, 入力データから本質的な特徴を取り出してはいないため, 学習サンプルに含まれない未知のデータをうまく分類できない. また, 入力に近い層の出力の絶対値が大きくなると, 後ろの層に対してバイアスとして働くため, 層が深い CNN は, 値の発散による学習の失敗や, CNN の分類性能が低くなるといった結果に陥る場合がある (Bias Shift Problem) [9]. これらの問題への対策として, CNN の変数や特徴に対して制約を課す正則化手法が研究されている. 変数に対する正則化手法は, 変数の値そのものに L2 正則化を適用することで, 変数の数や値の大きさを制限する重み減衰 (Weight Decay) がよく用いられる. それに対し, 厳密には正則化手法ではないが, 特徴の一部の値をランダムに 0 にする Dropout という手法が提案され, 広く用いられている [10]. 他にも, 特徴のサンプル平均を 0, 分散を 1 に標準化 (正規化) する Batch Normalization がよく用いられ, 学習の安定化だけでなく, 分類精度向上にも大きく寄与している [11]. このように, CNN の学習を安定化させ, 過学習も回避するには, 正則化手法, 特に特徴に対して制約をかける手法が必要となる.

CNN が抱えている問題は, 他にもある. CNN の学習が安定し, 高い分類精度が得られたとしても, CNN の計算コストが大きすぎるという問題である. 例えば, 一枚の画像に写っている物体を分類するのに CNN が要する計算コストは, 代表的な CNN の一つである AlexNet（変数の数は 6,100 万）が 7.2 億 Floating-Point Operations (FLOPs) であるのに対し, 分類精度を数パーセント向上させた VGG-16（変数の数は 13,800 万）では, 150 億 FLOPs に跳ね上がっている [12]. その巨大な計算コストは, 携帯電話などの計算リソースが限られたエッジデバイスに CNN を用いる際の障害となっている [13]. したがって, CNN の高い分類性能を維持しつつその計算コストを削減することは, CNN を用いる分野にて注目を集めている.

CNN の計算コストを削減するアプローチは, 大別して 2 つに分けることができる. 一つは, MobileNet に代表される, CNN の構造を 1 から再設計し, その後 CNN を学習するアプローチである [14, 15]. このアプローチは, ネットワークの再設計をヒューリスティックに行うことで高い分類性能と低い計算コストを達成しているが, web 上で公開されている既存の学習済み CNN を流用しづらいという点で難がある. それに対し, 高い分類精度を示す学習済み CNN を流用しやすいアプローチの一つに, Pruning 手法がある. Pruning 手法は, 分類などのタスクに対してあまり貢献していない (重要ではない) 特徴は存在してもしなくてもタスクにあまり影響しないという仮定のもと, 一定の基準・関数に基づいて, そのような特徴と関連する変数を学習済み CNN から剪定 (Pruning) し, CNN の計算コストを削減する手法である. 様々な Pruning 手法 [12] が提案されているものの, それらが本質的に行っているのは, CNN が獲得した特徴のタスクに対する貢献度合い (重要度) を評価し, 重要度が低い特徴, ならびに, それと関連する変数の Pruning である. これは, Pruning 手法が活用すべき情報が, 学習によって CNN が獲得した特徴であることを示唆している.

本論文では, CNN が学習によって獲得する特徴を解析するだけでなく, CNN が獲得すべき特徴や, 獲得した特徴の活用について論じる.

1.2 論文の構成

本論文は, CNN が獲得すべき特徴について論じ, その獲得を促進させる正則化手法を提案した第 2, 3 章と, CNN が獲得した特徴の活用方法について論じた第 4 章, 本研究の結論である第 5 章からなる.

第 2 章では, まず, CNN を含めたニューラルネットワークの概要を説明する. 本論文で扱う CNN は, 階層構造を持ち, 複数の層からなる中間層と, 中間層で得られた特徴を用いて分類問題を解く出力層 (分類器) で構成されている. つづいて, CNN の中間層に対して制約を加える従来の正則化手法が, どのように CNN の学習を安定化させるのかについて論じる. また, CNN の中間層でどのような特徴が獲得されるのかについても論じる. CNN は, 脳と同様に, 入力に近い層ではエッジや輪郭に関する情報を特徴として獲得するのに対し（V1 と呼ばれる脳の一次視覚野と同様の機能）, 出力に近い層では高次元の意味情報を認識する（V4 や V5 と同様の機能）と言われている [16]. これは, CNN の学習において, 視覚野の形成を促進することができれば, より良い特徴が獲得できることを示唆している. スパースコーディングを用いると V1 の受容野を自己組織化できることを Olshausen ら [17, 18] が示したことから, CNN が学習する特徴に対して同様のスパース制約を持ち込むことで, CNN の学習の安定化だけでなく, より良い特徴も獲得できると考えられる. そこで, CNN の中間層で得られる特徴に対して直接スパース制約を課すスパース正則化を提案し, 受容野形成の促進と Bias Shift Problem 改善の両方を目指す. 中間層で得られる特徴に対してスパース正則化を適用することで, 学習が安定するだけでなく, 分類精度も向上することを様々な実験で確認する. また, 学習サンプルの偏りなど, CNN の学習がうまくいかない条件であっても, スパース正則化はその悪影響を受けづらいことも確認する.

第 3 章では, まず, 判別基準を定義し, 多クラス問題を解く CNN の分類器にとって理想的な特徴について議論するためのツールを用意する. 判別基準は, 2 つの分布がどのくらい分離されているかを評価できる基準である [19]. つづいて, 分類器を構成する各ニューロンの挙動を解析し, 分類器にとって好ましい特徴について議論する. K クラスの分類問題を解く分類器は, K 個のニューロンで構成され, 各ニューロンごとに分類すべきクラス (ターゲットクラス) が割り当てられている. サンプルが CNN に入力されると, ソフトマックス関数を介して, そのサンプルがターゲットクラスに属する事後確率を分類器のニューロンごとに出力し, 最も事後確率が高いニューロン (クラス) を分類結果とすることで, 分類器は K クラス分類問題を解いている. このことから, 分類器全体では K クラス分類問題を解いているが, 実際には, 分類器の各ニューロンごとに, ターゲッ

トクラスかそれ以外 (非ターゲットクラス) かの 2 クラス分類問題を解いていることを示す. その後, 分類器の各ニューロンにとって好ましい特徴は, ターゲットクラス, もしくは非ターゲットクラスの情報を含んだ 2 つのガウス分布 (分散は等しく, 平均は異なる) であることを, 数式的・実験的に示す. つづいて, そのような特徴の獲得を促進し, CNN の分類精度を改善するために, 分類器の各ニューロンに対して判別基準を正則化として適用する判別正則化を提案する. 様々な実験にて, 判別正則化を導入することで, 分類器の各ニューロンに入力される 2 つの分布が, よりガウス分布に近くなり, さらに, 重なりがあった 2 つの分布も大きく分離されたことで, CNN の分類精度も向上することを確認した. また, 判別正則化を用いることで, 分類器に入力される特徴が, よりクラス毎に分離される (判別的である) ことも確認した. 他にも, 分類器の各ニューロンは 2 クラス分類問題を解いていることから, 分類器の活性化関数として, ソフトマックス関数よりもシグモイド関数の方が適している場合が多いことも, 実験的に確認した.

第 4 章では, まず, これまで研究されてきた Pruning 手法を, 変数に注目した手法と特徴に注目した手法の 2 つにグループ分けし, その違いについて議論する. なかでも, 特定の変数や特徴の有無により発生する分類の誤差 (分類誤差) の変動に注目し, 分類問題などのタスクに対する特徴の貢献度を直接評価する基準 (**スコアリング関数**) や, それを用いる Prunig 手法に焦点を当てる [20, 21, 22, 23]. 次に, 分類誤差を用いて重要度を評価する Pruning 手法が抱える問題点について論じる. 一般的に, 分類誤差そのものを知ることは困難なので, 重要度の評価 (Pruning Score) の計算に用いられる分類誤差は, 正解が割り当てられた学習サンプルを用いて計算される経験的分類損失で近似される. しかし, 分類が困難な学習サンプルや, 極端な損失の値をとるものが存在すると, 経験的分類損失はそれらに大きく影響される可能性があるため, Pruning Score が信頼できない可能性がある. そこで, そのような問題を改善した, よりロバストなスコアリング関数を用いて Pruning Score を計算する Pruning 手法を提案する. まず, Pruning による分類損失の悪化を数式で表現するために, 分類損失を評価する損失関数を, 特徴に関してテイラー展開する. テイラー展開により, 従来手法 [20, 21] のように, 特徴の有無が分類損失に与える影響を直接評価 (予測) することができる. 次に, 分類損失のテイラー級数を代数的に変換し, 極端な値をとるサンプルに対してロバストなスコアリング関数を導出する. その過程で, 提案するスコアリング関数や, 類似する関数 [22, 23] の関連性を数式的に示す. 学習サンプル毎に Pruning Score を計算し, その結果を合計する従来手法とは対照的に, 提案するスコアリング関数では, 個々のサンプルや層の深さに影響されることなく公平に Pruning Score を計算しているため, より安定した Pruning が可能となる. また, その延長として, Pruning Score を正規化する新たな正規化手法も提案し, その有効性も実験的に確認する. 他にも, 各特徴の有無による分類損失の変動を近似 (予測) することで重要度を求めるのではなく, 補助変数の連続最適化問題に置き換え, それを安定して解くことで重要度を求める新たな Pruning 手法も提案する. Pruning の問題を補助変数の離散最適化問題に置き換えた手法がすでに提案され, 優れた結果を示しているが [24], 離散最適化問題を解くため, 最適化が不安定であること, また, その手法を単に連続最適化問題に置き換えても結果が芳しくないことを示す. その後, それらの問題を解決するために, 特徴を圧縮した空間上で補助変数の最適化問題を解く新たな Pruning 手法を提案し, その有効性を実験的に確認する.

最後に, 第 5 章にて本研究の貢献を要約し, 今後の課題に触れることで, 本論文の結びとする.

2　スパース正則化の適用

2.1　はじめに

近年, 画像分類をはじめとしたコンピュータービジョンの分野で, ニューラルネットワークが盛んに研究されている [25, 5, 26, 27, 28]. 脳の情報処理システムの中枢である脳神経系を模倣したニューラルネットワークは, 入力に応じて値を出力（発火）するノードを用いてニューロンをモデル化し, シナプスを模した重み変数で各ニューロンを結合したネットワークである. 学習サンプルを用いて各変数を最適化 (学習) することで, ネットワークは画像分類などの任意のタスクを行うことができる. 初期の頃は比較的小規模なネットワーク [2, 29] が用いられていたが, ネットワークの分類精度を向上させるために, より大規模で複雑なネットワークが数多く提案されている [25, 5, 26, 27, 28].

しかし, そのような大規模なネットワークは, 出力値が発散する, 学習が収束しない, 学習サンプルに最適化されすぎて未知のデータに対する分類精度がかえって悪くなる (過学習) といったリスクが増加することも経験的に知られている [7, 8]. そのようなリスクを回避する手段の一つとして, ネットワークの変数などに制約を加える正則化手法が研究されており, 分類精度向上だけでなく学習の安定化にも大きく寄与している.

本章では, まず, 2.2 章にて, 本論文で中心的に扱うニューラルネットワークについて説明し, 各種議論・考察で必要となる事項のまとめを行う. 2.3 章では, 各種正則化手法がニューラルネットワークに与える効果を, 数式的・実験的に考察する. ここでは, ニューラルネットワークにて広く用いられている, 重みに対する L2 正則化 (Weight Decay), Batch Normalization, 及び活性化関数の働きについて考察する. 本来, Batch Normalization や活性化関数は正則化手法ではないが, 限定的ながら, ニューロンの出力 (反応) に対する正則化手法とみなすことができることも示す. 2.4 章では, ニューロンそのものに対してスパース制約を設ける新たなスパース正則化を提案し, その優位性を示す. この正則化は, Olshausen らが脳の視覚野の受容野を再現するために用いた正則化項を, ニューラルネットワークに応用したものである. 従来の正則化手法の多くはネットワークの各ニューロンをつなぐ重み変数に対して制約を設けているのに対し, 提案手法はニューロンそのものに制約を設けている点がそれらと異なる. 2.5 章では, 複数のデータセット・ネットワークを用いて, 従来手法とスパース正則化を比較する. 2.6 章では, 従来手法ではネットワークの学習がうまくいかない条件であっても, 提案手法ならば学習がうまくいくことも示す. 本章は, 過去に発表した研究 [30, 31] の論点を整理・追加し, より詳細に検証実験を行なったものである.

2.2　ニューラルネットワーク

2.2.1　パーセプトロン

20 世紀初頭, 生物の脳は, シナプス (Synaps) を介して結合される多数のニューロン（Neuron, 神経細胞）で構成されていることが明らかになった [16]. その発見をもとに, McCulloch らは, ニューロンを単純なノードとしてモデル化し, 複数のノードを接続することで, 任意の計算が可能であることを示した [6]. また, ノードを用いてチューリングマシンを模倣できることも示された [32, 33]. 脳が行う情報処理の特徴の一つである高い学習能力を再現するために, Hebb は, ノード間の接続を変化させる学習モデル (Hebb 則) を示した [34]. 彼の学習モデルは, 各ニューロンを繋ぐシナプスの結合の変化が学習において大きな役割を果たしているという考えを基にしている.

こうした研究を踏まえ, Rosenblatt らは, パーセプトロン (Perceptron) を提案した [35]. これは, 脳の認

知機能を司るニューロンをモデル化したもので, パターン認識を行う. パーセプトロンをはじめとする脳の情報処理を模倣したモデルや, 脳の情報処理そのものに対する研究は, 1950 年代から 60 年代にかけて盛んに研究された. パーセプトロンは, 重み結合 $\boldsymbol{w} = (w_1, w_2, \ldots, , w_D)^T \in \mathbb{R}^D$ をシナプス, ニューロンへの入力を $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)^T \in \mathbb{R}^D$ とすると

$$y = \sigma\left(\sum_d^D w_d x_d\right) \tag{2.1}$$

$$= \sigma\left(\boldsymbol{w}^T \boldsymbol{x}\right) \tag{2.2}$$

や図 2.1 のように表される. σ は, 活性化関数 (Activation Function) と呼ばれる, ニューロンの発火を再現した関数であり, 主に非線形関数が用いられる. パーセプトロンでは, 閾値関数（ヘビサイド関数）

$$\sigma(x) = \begin{cases} 1 & x \geq \theta \\ 0 & x < \theta \end{cases} \tag{2.3}$$

が活性化関数として用いられる. パーセプトロンは, 入力 $\boldsymbol{x}$ と重み $\boldsymbol{w}$ の内積が閾値 θ を越えると 0 以外の値

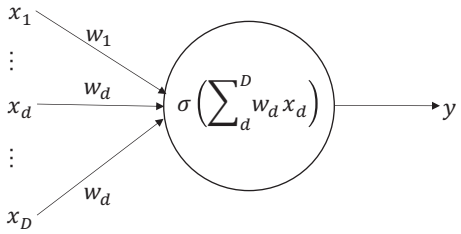


図 2.1: パーセプトロン.

を出力 (発火し) し, 閾値未満ならば 0 を出力する (発火しない). このような二値を出力するパーセプトロンは, 入力された $\boldsymbol{x}$ を 2 つのクラスに分類する 2 クラス分類問題を解く関数として使用できる.

パーセプトロンを用いて 2 クラス分類問題を解くには, 重み $\boldsymbol{w}$ と閾値 θ を最適化（学習）し, 入力サンプル $\boldsymbol{x}$ に対して正しい値を出力できるようにする必要がある. Rosenblatt らは, パーセプトロンの学習アルゴリズムとして誤り訂正学習 (Error-Correction Learning) を提案した. 誤り訂正学習は, 学習に使われる N 個のサンプル（学習サンプル）$\{(\boldsymbol{x}_n) | n = 1, \ldots, N\}$ を用意し, その中からランダムに選んだ学習サンプルの一つ $\boldsymbol{x}_n$ を入力した時のパーセプトロンの出力 y_n が, 目的とする出力 (教師信号) $t_n \in \{0, 1\}$ と異なっていた時だけ, 重み $\boldsymbol{w}$ を更新する. その時の更新式は

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \mu(t_n - y_n)\boldsymbol{x}_n = \boldsymbol{w} + \mu\delta_n \boldsymbol{x}_n \tag{2.4}$$

である. ここで, $\delta_n = t_n - y_n$ は, パーセプトロンの出力が教師信号と異なると値が ± 1 に, 等しければ 0 になることから, 誤差と呼ばれている. μ は重みの更新幅を決めるハイパーパラメータ (学習係数, Learning Rate) である. 式 (2.4) のように, 学習サンプルを一つ入力するごとに重みなどの変数を更新する学習方法は, オンライン学習 (Online Learning) と呼ばれている. 学習方法は, 他にも, N 個の学習サンプルを一度に全て入力し, その平均 $\mathrm{E}[\delta_n \boldsymbol{x}_n]$ を用いて変数を更新するバッチ学習 (Batch Learning), 学習サンプルを複数のサブセット

(ミニバッチ) に分割し, ミニバッチ毎に計算した平均を用いて変数を更新するミニバッチ学習 (Mini-Batch Learning) などがある.

誤り訂正学習は, 線形分離可能な分類問題ならばパーセプトロンの学習が有限時間内に収束するものの, いくつかの問題がある. 最も大きな問題は, 分類問題が線形分離可能ではない場合, 学習を無限に行っても収束しない可能性があることである. Minsky らは, パーセプトロン単体では, 排他的論理和などのいくつかの問題を解くことができないことを示した [36]. この結果により, パーセプトロンだけでなく, パーセプトロンを多数組み合わせたニューラルネットワークでも非線形問題を解くことはできないという誤った解釈が広まったことで, ニューラルネットワークの研究が下火となった. 他にも, たとえ線形分離可能な問題であっても, 学習によって得られるパーセプトロンの重みは, その初期値によって異なる (初期値依存) という問題もある. これは, 学習によって得られる解が一意に定まらない, 初期値によっては学習に要する時間が非常に多くなる, といった結果を招く.

2.2.2 多層ニューラルネットワーク

パーセプトロンは脳のニューロンをノードとしてモデル化したものであるが, 実際の脳は, 膨大な数のニューロンを相互に結合することで, 高度な情報処理を行なっている. それを模倣し, パーセプトロンをニューロンとして相互に結合したネットワークは, ニューラルネットワークと呼ばれている. なかでも, 多層パーセプトロン (Multi-Layer Perceptron, MLP), あるいは多層ニューラルネットワーク (Multi-Layer Neural Network, NN) と呼ばれるニューラルネットワークは, 複数のパーセプトロンで構成される層を階層的に結合し, 非線形問題を含めたより複雑な情報処理を行えるようにしたネットワークである. 図 2.2 に, 入力層 (Input Layer), 中間層 (Intermediate Layer, Hidden Layer), 出力層 (Output Layer) の三層からなる多層ニューラルネットワークを示した. ネットワークは, サンプル $\boldsymbol{x}^{(1)} \in \mathbb{R}^J$ が入力されると, 中間層と出力層の各ニューロンの出力を

$$x_d^{(2)} = \sigma\left(\sum_j^J x_j^{(1)} w_{j,d}^{(1)}\right) \tag{2.5}$$

$$y_k = f\left(\sum_d^D x_d^{(2)} w_{d,k}^{(2)}\right) \tag{2.6}$$

と計算する. ここで, $w_{j,d}^{(1)}$ は入力サンプルの j 番目の要素 $x_j^{(1)}$ と中間層の d 番目のニューロン $x_d^{(2)}$ を結ぶ結合の重みであり, $w_{d,k}^{(2)}$ は中間層の d 番目のニューロン $x_d^{(2)}$ と出力層の k 番目のニューロン y_k を結ぶ結合の重みである. また, σ と f は, それぞれ中間層と出力層の活性化関数である. このように, サンプルが入力されると, 入力層から中間層へ, そして出力層へと情報 (特徴) が一方向へ伝搬されるネットワークは, 順伝搬ニューラルネットワーク (フィードフォワードニューラルネットワーク, Feed-Forward Neural Network) と呼ばれる.

パーセプトロンと同様に, 分類問題を解くなどの情報処理をネットワークにさせるには, 各ニューロンを結ぶ重みの値を最適なものにする必要がある. パーセプトロンで用いられた誤り訂正学習は, 出力層と直接結合している重みしか最適化できないため, ネットワークの中間層の重みを最適化する学習アルゴリズムとして, 誤差逆伝播法 (Back-Propagation) が提案された [37]. 1980 年代に提案されたこの学習アルゴリズムは, 今でもニューラルネットワークの主要な学習アルゴリズムとして用いられている.

誤差逆伝播法は, 学習サンプルをネットワークに入力したときの出力と教師信号との誤差を評価する関数 $\mathcal{L}$

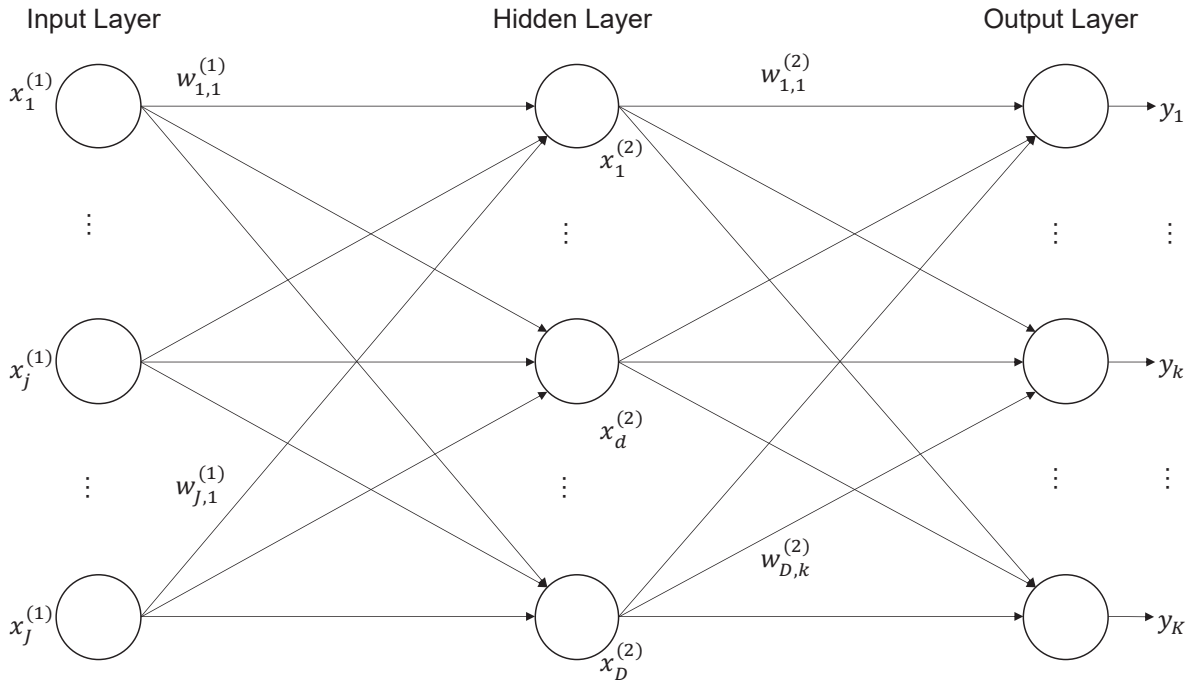


図 2.2: Feedforward Neural Network.

を損失関数とし, 重みに関する損失関数の偏微分を用いて, 各重みを

$$w_{d,k}^{(2)} \leftarrow w_{d,k}^{(2)} - \mu \frac{\partial \mathcal{L}}{\partial w_{d,k}^{(2)}} \tag{2.7}$$

$$w_{j,d}^{(1)} \leftarrow w_{j,d}^{(1)} - \mu \frac{\partial \mathcal{L}}{\partial w_{j,d}^{(1)}} \tag{2.8}$$

と更新することで, 損失関数の値, つまり誤差を最小化する. 偏微分はそれぞれ

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{d,k}^{(2)}} &= \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial w_{d,k}^{(2)}} \\ &= \delta_k \frac{\partial y_k}{\partial w_{d,k}^{(2)}} \end{aligned} \tag{2.9}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{j,d}^{(1)}} &= \sum_k^K \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial w_{j,d}^{(1)}} \\ &= \sum_k^K \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial x_d^{(2)}} \frac{\partial x_d^{(2)}}{\partial w_{j,d}^{(1)}} \\ &= \sum_k^K \delta_k \frac{\partial y_k}{\partial x_d^{(2)}} \frac{\partial x_d^{(2)}}{\partial w_{j,d}^{(1)}} \end{aligned} \tag{2.10}$$

である. 損失関数 $\mathcal{L}$ がネットワークの出力 $\boldsymbol{y} \in \mathbb{R}^K$ と教師信号 $\boldsymbol{t} \in \mathbb{R}^K$ との二乗誤差

$$\mathcal{L} = \frac{1}{2}\sum_{k}^{K}(t_k - y_k)^2 \tag{2.11}$$

であるとき, δ_k は

$$\delta_k = t_k - y_k \tag{2.12}$$

となる. δ_k は誤差であり, 式 (2.9), 式 (2.10) で示したように, 出力層から入力層に向けて, 入力とは逆方向に誤差が伝搬されている. これが, 誤差逆伝播法という名前の由来である.

式 (2.8) のように, 勾配を用いて変数を更新する学習アルゴリズムは, 勾配降下法 (Gradient Decent) と呼ばれる. なかでも, ミニバッチ学習のように, N 個の学習サンプルの中からランダムに m 個のサンプルを選び, そのサブセット (ミニバッチ) ごとに変数を更新する学習アルゴリズムは, 確率的勾配降下法 (Stocastic Gradient Decent, SGD) と呼ばれている. 式 (2.8) をミニバッチ毎のサンプル平均に拡張した

$$w_{d,k}^{(2)} \leftarrow w_{d,k}^{(2)} - \mu E_n \left[\frac{\partial \mathcal{L}\left(\boldsymbol{x}_n\right)}{\partial w_{d,k}^{(2)}}\right] \tag{2.13}$$

は, その最も基本的な手法であり, SGD という言葉は上式そのものをさすことも多い. そのため, 本論文では, 単に SGD とだけ表記してある場合, 上式のことを指す.

また, 誤差逆伝播法は損失関数 $\mathcal{L}$ の偏微分を計算する必要があることから, 活性化関数 σ は微分可能であることが必須条件である. この頃によく用いられていたのは, シグモイド関数 (Sigmoid Function)

$$\frac{1}{1 + \exp\left(-x\right)} \tag{2.14}$$

である.

勾配降下法を用いた学習アルゴリズムは 1960 年代にも提案されていたが [38], あまり実用的ではないと考えられていた. なぜならば, 学習アルゴリズムに求められるのは, 誤差を最小化する最適な解 (大域的最適解) に重みが収束することであるが, 勾配降下法による最適化では, 誤差を極小化するだけで, 最小にはならない解 (局所的最適解, local optima) に収束してしまい, 学習がうまくいかないと考えられたからである. 実際には学習がうまくいくことが Rumelhart らによって実験的に示されたこともあり, 1980 年代には, 勾配降下法やニューラルネットワークの研究が盛んに行われた.

多層ニューラルネットワークは, 十分な数のニューロンで構成される中間層が一層あれば任意の出力が得られることが証明されているが [6], 実用面では, ニューロンの数を増やす代わりに, 層の数を増やし, 深くしたほうが良いことが経験的に知られている. また, 脳の視覚野でも, 複数の層で情報処理を行なっていることが示唆されている [39]. しかし, 多層ニューラルネットワークの層を深くすると, 局所的最適解の問題が深刻となり, さらに, 入力に近い層の勾配が消失し, 学習がうまくいいかなくなる勾配消失問題 (Vanishing Gradient Problem) も発生するようになった. これらの問題により, 多層ニューラルネットワークは, 一部の例外を除いて, 層が深くなると学習は困難であると認識された. さらに, 上記の問題が存在しないサポートベクターマシン [40] の登場も追い打ちとなったことで, ニューラルネットワークの研究は再び下火となった.

2.2.3 畳み込みニューラルネットワーク

ニューラルネットワークの層を深くしても学習がうまくいく畳み込みニューラルネットワーク (Convolutional Neural Network, CNN) が高い分類性能を示したことにより, ニューラルネットワークの研究は再び活

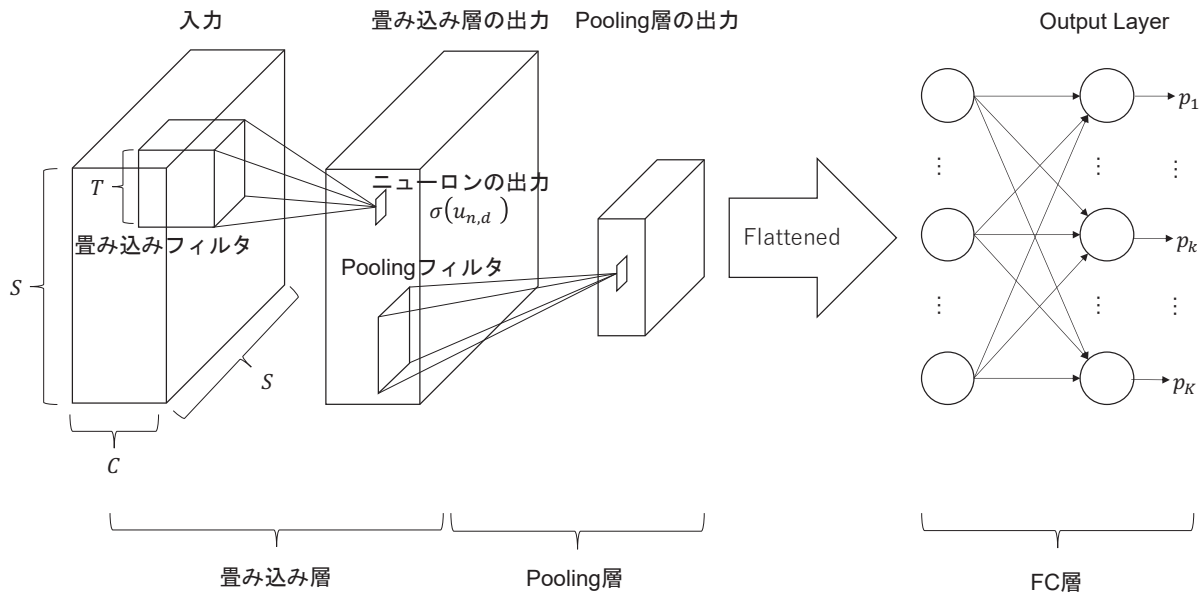


図 2.3: CNN.

発になり, 画像認識や画像生成などの分野で標準的な手法の一つとなった [3, 5, 41, 42, 43]. CNN のルーツは, 脳の視覚野を模した多層ニューラルネットワークであるネオコグ二トロン (Neocognitron) [1] である. ネオコグ二トロンの学習アルゴリズムとして誤差逆伝播法を導入した LeNet5 [29, 2] が 1980 年代に提案され, これが CNN の原型となった. その後, ニューラルネットワークの研究が下火なったことでしばらく埋もれていたが, 2012 年に, AlexNet [3] と呼ばれる大規模な CNN を Krizhevsky らが実装し, ILSVRC 2012 という画像分類の精度を競うコンテストにて従来手法より高い分類性能を達成したことで, CNN が大きく注目された.

従来の多層ニューラルネットワークと CNN が大きく異なる点は, 図 2.3 に示すように, ネットワークの中間層が畳み込み層や Pooling 層で構成されている点である. 畳み込み層 (Convolution Layer) は, CNN を構成する最も重要な層の一つである. 図 2.3 に示すように, l 層目の畳み込み層の入力 $\boldsymbol{X}^{(l)} \in \mathbb{R}^{B\times S\times S}$ は, 縦横のサイズが $S\times S$, 枚数 (チャネル数) が B の画像, もしくは, 前の層から得られた画像形式の特徴 (特徴マップ) である. チャネル数 B は, 入力画像が白黒画像ならば 1 チャネル, RGB ならば 3 チャネルであり, 中間層の特徴マップならば, ネットワークの設計者が任意に設定する. 特徴マップ $\boldsymbol{X}^{(l)}$ が畳み込み層に入力されると, 重み変数を持つ畳み込みフィルタ $\boldsymbol{W}^{(l)} \in \mathbb{R}^{C\times T\times T}$ は, $\boldsymbol{X}^{(l)}$ 上をスライドされ, サイズ $T\times T$ の小領域ごとに重み付き線形変換

$$u_{i,j}^{(c,l)} = \sum_{b}^{B}\left[\sum_{(p,q)\in\mathcal{P}_{i,j}} x_{p,q}^{(b,l)} w_{p-i,q-j}^{(b,l)}\right] \tag{2.15}$$

を出力する. ここで, C は出力される特徴マップのチャネル数である. $x_{p,q}^{(b,l)}$ は, 入力された特徴マップのうち, b チャネル目の座標 (p,q) のピクセル値である. $\mathcal{P}_{i,j}$ は, 出力される画像中の座標 (i,j) を左上の頂点とするサイズ $T\times T$ の小領域のインデックス集合

$$\mathcal{P}_{i,j} = \{(i+i', j+j') | i' = 1,\ldots,T, j' = 0,\ldots,T\} \tag{2.16}$$

である. 従来の多層ニューラルネットワークでは, 入力画像, あるいは特徴マップの要素ごとに重み変数を

与えていたが, 畳み込みフィルタは, 式 (2.15) で示したように, 小領域ごとに重み変数を再利用する重み共有 (Weight Sharing) を導入している．これは脳の視覚野を模した結果であるが, これにより, 学習すべき変数の数が大幅に削減され, 深い層のネットワークの学習が可能となった.

畳み込み層から最終的に出力される c 番目のチャネルの特徴マップ (次の層の入力) $\boldsymbol{X}^{(c,l+1)}$ は, $u_{i,j}^{(c,l)}$ を要素とする $\boldsymbol{U}^{(c,l)}$ を活性化関数 σ に入力して得られる

$$\boldsymbol{X}^{(c,l+1)} = \sigma(\boldsymbol{U}^{(c,l)}) \tag{2.17}$$

である．表記をシンプルにするために, 特段の理由がない限り, 各行列を $\boldsymbol{x}^{(b,l)} = \mathtt{vec}(\boldsymbol{X}^{(b,l)}) \in \mathbb{R}^{S^2\times 1}$, $\boldsymbol{w}^{(c,l)} = \mathtt{vec}(\boldsymbol{W}^{(c,l)}) \in \mathbb{R}^{T^2\times 1}$, $\boldsymbol{u}^{(c,l)} = \mathtt{vec}(\boldsymbol{U}^{(c,l)})$ のようにベクトル表記する．畳み込みフィルタは脳の視覚野を模倣したものであるため, フィルタの出力値 $x_{i,j}^{(l,c)}$ はニューロン (Neuron) と呼ばれ, フィルタに入力される $T \times T$ の小領域は, あるニューロンに刺激を与える小領域である受容野 (Receptive Field) と呼ばれることが多い.

Pooling 層 (Pooling Layer) も, CNN を構成する重要な層の一つである．Pooling 層では，特徴マップを重複無しの小領域に分割し, それぞれの小領域を入力としてスカラー値を出力することで, 特徴マップのダウンサンプリングを行う．Pooling 層を用いる主な利点は, 画像中の物体の位置の些細な違いに対して不変になる点や, 特徴マップの次元数や学習すべき変数の数を減らすことができる点である.

Pooling は, いくつもの亜種が提案されている．最もよく使われているものの一つが Max Pooling である．Max Pooling は, 特徴マップを分割した各小領域の最大値

$$x_{p,q}^{(b,l+1)} = \max_{(p,q)\in\mathcal{P}_{i,j}} x_{p,q}^{(b,l)} \tag{2.18}$$

を出力値とする手法である．Max Pooling は, 特徴マップの不変性や次元圧縮のために, 主に中間層で用いられる．それに対し, 特徴マップの各小領域の平均値を出力する Average Pooling

$$x_{p,q}^{(b,l+1)} = \frac{1}{|\mathcal{P}_{i,j}|} \sum_{(p,q)\in\mathcal{P}_{i,j}} x_{p,q}^{(b,l)} \tag{2.19}$$

は, 昔は中間層によく用いられていたが [2, 29], 最近では, 特徴マップの情報を集約するために中間層の一番最後の層に組み込まれる場合も多い．式 (2.18), 式 (2.19) のように, Pooling 層には活性化関数は導入されず, 学習によって変化する変数も存在しない.

入力画像を K クラスのうちのどれかに分類するために, 分類器 (Classifier) として CNN の出力層によく用いられるのは, 隣接する層のニューロンすべてと結合する全結合層 (Fully-Connected Layer, FC 層) である．これは, 従来のフィードフォワードニューラルネットワークの出力層などで用いられていたものと同じである．FC 層の入力 $\boldsymbol{x}$ は, Convolution 層と Pooling 層が画像の局所領域から獲得した情報の集まりであるため, そのような局所領域情報 (特徴) を統合することで, 入力画像が属するクラスを予測する K クラス分類問題を解く．情報の統合は, 中間層から出力された特徴マップをベクトル化した $\boldsymbol{x}$ を入力とし, 重み $\boldsymbol{W}$ を用いた線形変換

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^K \tag{2.20}$$

を計算することで行われるその後, 活性化関数にソフトマックス関数を用いて, 入力データが k 番目のクラスに属する事後確率

$$p_k = \frac{\exp y_k}{\sum_k^K \exp y_k} \tag{2.21}$$

が推定される．クラス分類を行う際には, 最大の事後確率をとるニューロンのインデックス $\hat{k} = \arg\max_k p_k$ を予測クラスとする．

CNN の学習アルゴリズムは, 多層ニューラルネットワークと同様に, クラス分類誤差を最小化する重み変数を確率的勾配降下法によって求める誤差逆伝播法が一般的である．多くの場合, K クラス分類問題では, 以下のように, n 個目の学習サンプルを入力した時の事後確率 $\boldsymbol{p}_n = (p_{n,1}, \ldots, p_{n,K})^T$ と, 目標とする出力 (教師信号) $\boldsymbol{t}_n = (t_{n,1}, \ldots, t_{n,K})^T$ の交差誤差 (Cross Entropy) のサンプル平均

$$\mathcal{L} = -\frac{1}{N} \sum_{n}^{N} \sum_{k}^{K} t_{n,k} \log p_{n,k} \tag{2.22}$$

を, 分類誤差を評価する損失関数とする．ここで, N は学習サンプルの総数である．$t_{n,k}$ と $p_{n,k}$ は, それぞれ, n 番目の学習サンプルがクラス k に属する確率とそのサンプルの教師信号 $\boldsymbol{t}_n \in \mathbb{R}^K$ の k 番目の要素である．教師信号 $\boldsymbol{t}_n$ は, 学習サンプルが属するクラスを one-hot ベクトルで表現したもので, 学習サンプルが属するクラスが k ならば $t_{n,k} = 1$, そうでなければ $t_{n,k} = 0$ となる．$p_{n,k}$ がソフトマックス関数の出力である時, 式 (2.22) は特別にソフトマックスクロスエントロピー (Softmax Cross Entorpy) と呼ばれる．

CNN の学習では, 学習サンプルの一部であるミニバッチごとに損失関数を計算し, その都度変数の更新を行う確率的勾配効果法が一般的である．これは, PC のメモリ使用量削減のためでもあるが, 全学習サンプルを用いるより学習が早く進み, さらに, 学習後の CNN の分類精度が高くなりやすいことが経験的に知られているからでもある [44].

2.3 従来の正則化手法

ニューラルネットワークの分類精度が低い, あるいは学習がうまくいかなくなる理由の一つに, Bias Shift Problem がある．Bias Shift Problem とは，ニューラルネットワークのあるニューロンの出力がそれより後ろの層に対してバイアスとして働くことで, 分類精度の悪化や, 値の発散による学習がうまくいかないといった現象を引き起こす問題である [9]．この問題に対して, 各ニューロンの出力の平均を 0 に近づけることで, それらの影響は緩和され, 学習に良い影響を与えることが知られている．LeCun ら [45] や Wiesler ら [46] は，各層に入力される特徴を白色化し, 特徴のサンプル平均を一定にすることで, ニューラルネットワークの学習が高速化することを示した．しかし，白色化は, 全ての学習サンプルの共分散行列を計算し, その固有値問題を解く必要があるため, 計算コストがかかりすぎる．

ここでは, より簡易な方法で分類精度を改善する正則化手法について論じる．

2.3.1 Batch Normalization

ニューラルネットワークの特徴の一つに，ある層の変数が学習で変化するとそれ以後の層の入力の分布が変化する, Internal Covariate Shift と呼ばれる現象がある [11]．この現象により, 各層の入力の分布が学習中に変化するので，学習係数を大きくすることができず, 結果的に学習速度が遅くなる．

したがって，ニューラルネットワークの学習を改善するには，Bias Shift Problem だけでなく, Internal Covariate Shift による悪影響も緩和する必要がある．例えば，LeCun ら [45] や Wiesler ら [46] は，各層の入力を白色化することで学習途中における各層の入力の分布を一定にし, 学習を加速させた．しかし, 引き換えに大きな計算コストが必要となった．

Ioffe ら [11] は，全ての学習サンプルを用いて白色化するのではなく, 一部の学習サンプル (ミニバッチ)

を用いて各層の入力を白色化する Batch Normalization を提案し, 白色化の計算コストを大きく削減した. Batch Normalization の計算に用いられる, あるニューロンの入力値 $u_{n,d}$[*1]のサンプル平均 μ_d は, 学習サンプルからランダムに N 個抽出したサンプルセット (ミニバッチ, Mini-Batch) を用いて

$$\mu_d = \frac{1}{N}\sum_{n=1}^{N} u_{n,d} \tag{2.23}$$

と計算される. 同様に, そのニューロンの入力の分散を

$$\sigma_d^2 = \frac{1}{N}\sum_{n=1}^{N}(u_{n,d} - \mu_d)^2 \tag{2.24}$$

とすると, 入力の正規化は

$$\hat{u}_{n,d} = \frac{u_{n,d} - \mu_d}{\sqrt{\sigma_d^2 + \epsilon}} \tag{2.25}$$

となる. ϵ は, 分母が 0 になるのを防ぐために用意される定数である. この正規化は, 統計学で使われる標準化である. これにより, そのニューロンへの入力は, ミニバッチ毎に平均 0, 分散 1 となる. 標準化された入力 $\{\hat{u}_n | n = 1, \ldots, N\}$ を用いると, 任意の平均 β, 分散 γ^2 の入力

$$u_{n,d} \leftarrow \gamma\hat{u}_{n,d} + \beta \tag{2.26}$$

を得ることができる. γ と β は学習によって決定される. 一般的に, γ と β はフィルタ毎 (チャネル毎) に共通であるため, 本論文の実験で Batch Normalization を用いる場合も, チャネル毎に γ と β を用意する. 式 2.26 から, Batch Normalization を用いることで, ミニバッチ毎の各ニューロンの出力の平均・分散は, サンプルの差異によって影響を受けないことを示している. おそらく, これが学習の改善に役立っている.

2.3.2 活性化関数

ニューラルネットワークの分類精度向上や学習の安定化のために, ニューロンの発火を模倣した活性化関数の改良を試みた研究も数多く存在する. 歴史的には, 多層パーセプトロンやボルツマンマシン [47] などの初期の頃のニューラルネットワークでは, 活性化関数としてシグモイド関数 (Sigmoid Function)

$$\sigma\left(u_{n,d}\right) = \frac{1}{1 + \exp\left(-u_{n,d}\right)} \tag{2.27}$$

がよく用いられてきた. シグモイド関数や tanh 関数 などの標準的な非線型活性化関数は, その出力値が大きな値を持つ時に, その勾配はほぼゼロになることが知られている. これにより, ネットワークの学習がある程度まで進むと, 確率的勾配降下法による変数の更新量は非常に小さくなり, 学習が進まなくなる. この問題は勾配消失問題 (Vanishing Gradient Problem) として知られている.

活性化関数による勾配消失問題を改良するために, Nair ら [48] は次のように定義された Rectified Linear Units (ReLU)

$$\sigma_{ReLU}(u_{n,d}) = max(0, u_{n,d}) \tag{2.28}$$

[*1] 実際には $u_{n,d}^{(c,l)}$ で表されるが, 表記をシンプルにするために, 本章ではチャネル c と層 l のインデックスを省略する. その他の記号も同様である.

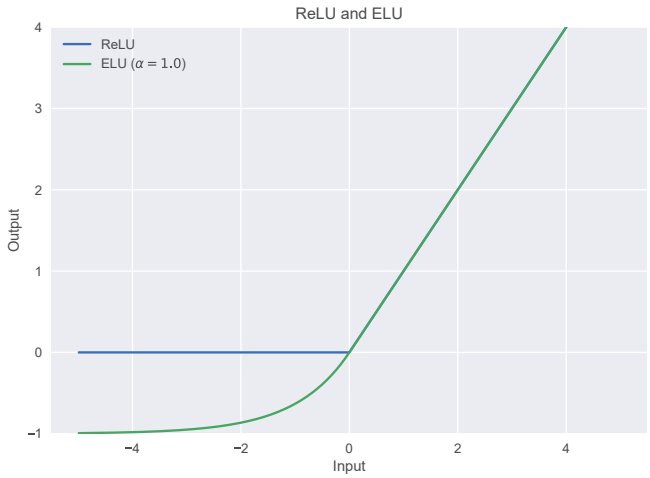


図 2.4: ReLU と ELU のグラフ．ReLU が青線, ELU が緑線で表示．

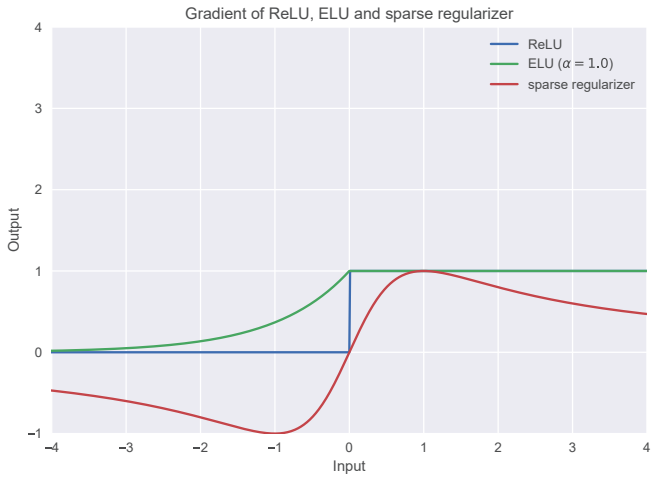


図 2.5: 活性化関数の入力に関する ReLU と ELU の導関数．ReLU は青線, ELU は緑線で表示．また, 活性化関数の入力に対するスパース正則化 $S(h_k)$ の導関数は赤線で表示．

を提案した．ReLU のグラフを図 2.4 に青で示す．また, ニューロンの入力 $u_{n,d}$ に関する ReLU の勾配は, 次のように求められる．

$$\frac{\partial \sigma_{ReLU}(u_{n,d})}{\partial u_{n,d}} = \begin{cases} 1 & (u_{n,d} > 0) \\ 0 & (u_{n,d} \leq 0) \end{cases} \tag{2.29}$$

ReLU の勾配のグラフは, 図 2.5 に青で示す．式 (2.29), 図 2.5 から, 入力が正の値であるときに勾配が一定になり, 消失しなくなっていることがわかる．これは, ReLU 活性化関数を用いることで, 勾配消失問題を回避できることを意味している．その後, Glorot ら [49] は, ReLU 活性化関数をネットワークに導入することで, 様々なネットワークの学習を高速化できること実験的に示した．現在では, CNN の標準的な活性化関数の一つ

として, ReLU 活性化関数が広く用いられている.

また, 各ニューロンの出力の平均を 0 に近づけることでネットワークの学習速度が改善される [50, 45] ことが知られているため, 活性関数を改良することで, 各ニューロンの出力の平均を 0 に近づける研究もいくつか行われている. Raiko ら [51] は, Fisher 情報行列の非対角要素を小さくするために, 各ニューロンの出力を小さくする方法を提案した. Desjardins らは, 各ニューロンの出力に対する白色化により, ニューロンの出力の平均が 0 に近くなることを暗に示した [52].

それに対し, Clevert ら [9] は, ReLU に負の出力を持たせた Exponential Linear Units (ELU) 活性化関数を提案した. ELU の定義は

$$\sigma_{ELU}(u_{n,d}) = \begin{cases} u_{n,d} & (u_{n,d} > 0) \\ \alpha(\exp(u_{n,d}) - 1) & (u_{n,d} \leq 0) \end{cases} \tag{2.30}$$

である. ここで, α はニューロンが負の値を出力するときの大きさを制御するハイパーパラメータである. ELU のグラフも図 2.4 に緑色で示している. 入力 $u_{n,d}$ に関する ELU の微分は, 次のように求めらる.

$$\frac{\partial \sigma_{ELU}(u_{n,d})}{\partial u_{n,d}} = \begin{cases} 1 & (u_{n,d} > 0) \\ f_{ELU}(u_{n,d}) + \alpha & (u_{n,d} \leq 0) \end{cases} \tag{2.31}$$

ELU の勾配のグラフを図 2.5 に緑で示す. ReLU と ELU を比較すると, 関数の入力が正の時は, どちらも勾配が消失しないため, 勾配消失問題を回避するのに有効であることがわかる. 一方, 関数の入力が負のとき, ReLU は 0 を出力し, 勾配も消失するのに対し, ELU は負の値を出力し, その勾配は消失しない. このことから, ReLU と ELU は, どちらも勾配消失問題を回避するのに有効であるが, ELU の方が, 勾配消失問題に対してより有効であることがわかる.

一方, Bias Shift Problem による悪影響を緩和する観点からも, ReLU に比べ, ELU の方が有効である. まず ReLU について考えてみる. ReLU の出力は常に 0 以上なため, その出力の平均値も 0 以上である. これは, ReLU を用いたネットワークでは Bias Shift Problem の影響を受けることを意味する. それに対し, 図 2.4 に示すように, ELU は負の値も出力するため, 出力の平均値が 0 に近くなる. これは, ELU によって, Bias Shift Problem の影響が和らげられることを意味する. これが, ELU がネットワークの学習を向上させる理由の一つであると考えられる.

2.3.3 受容野の単純型細胞

ニューラルネットワークは, 脳神経系を模倣しているため, 脳に関する知見が役立つことが多い. 事実, CNN の階層構造が脳の視覚野と似ている [39] だけでなく, 学習済みの CNN が獲得した特徴と, 神経細胞の挙動の間に類似性があることが報告されている [53, 54]. そこで, 脳に関する知見の一部も紹介しておく.

脳に関してよく知られている知見の一つは, 脳の視覚野を構成する各ニューロンは, 視野中の小さな領域（受容野）から刺激を与えられ, 個々のニューロンごとに特定の刺激に対して反応することである [55]. 受容野は, 視野全体をカバーするように敷き詰められ, 特定の視覚的な刺激に対して反応する局所フィルタとして機能する. なかでも, 視覚野の一部である一次視覚野 (V1) を構成する単純細胞 (Simple Cell) の大部分は, 位置や傾きなどに反応することが知られている [55, 56]. また, 単純細胞は大きさに対しても不変である. これはガボールウェーブレットの基底関数とよく似ているため, 単純細胞はガボールフィルタで近似できることもよく知られている [57].

視覚野のニューロンの働きを理解するため, 画像認識の分野では, 多くの研究者が画像の教師なし学習を行ってきた. Olshausenn らは, 画像から小パッチ (局所領域) を取り出し, 0 以外の値を取る変数が少ない内

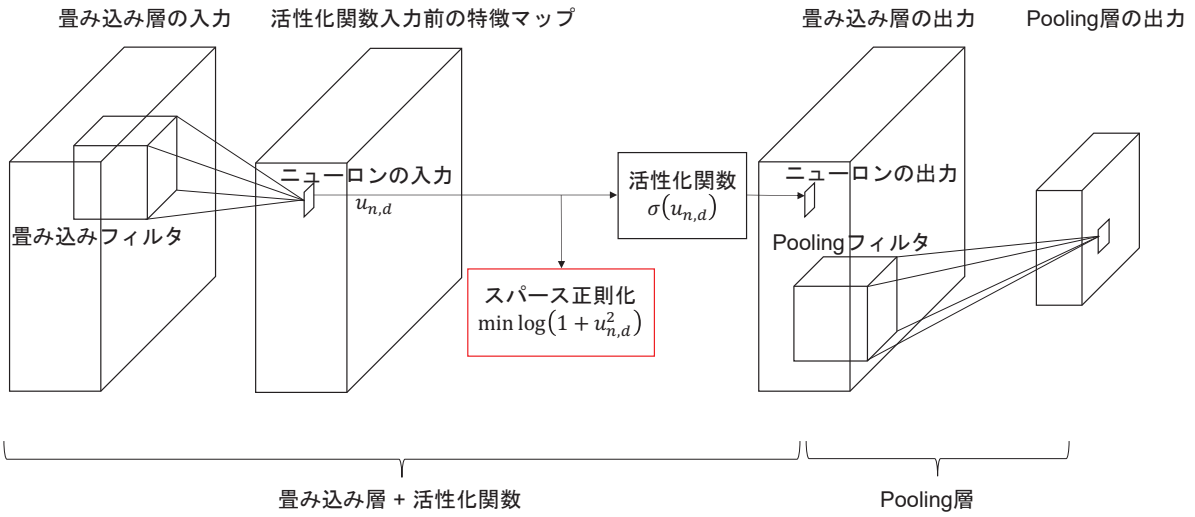


図 2.6: スパース正則化. 各ニューロンの入力 $u_{n,d}$ (活性化関数に入力する前の特徴マップ) にスパース正則化を適用する.

部表現 (スパースな表現) に変換する線形スパースモデリングを行うことで, V1 視覚野の受容野とよく似た受容野が得られることを示した [17, 18]. 彼らは, 画像の内部表現を得るために, 以下の式

$$\mathcal{Q} = \mathcal{R} + \lambda \mathcal{S} \tag{2.32}$$

の最適化問題を解いた. $\mathcal{R}$ は変換された内部表現から元の局所領域への再現度合いである再構成誤差, $\mathcal{S}$ は内部表現をスパースにする制約であるスパース正則化, λ はスパースさをコントロールするハイパーパラメータである. 彼らは, スパース正則化として log や exp などを用いることで一次視覚野の受容野を再現できると示しただけでなく, 局所領域を基底関数の線形和で表現することで, 高レベルの視覚処理に効果的な表現が得られることも示した.

彼らが行なった実験は, 線形な Auto Encoder の各ニューロンの出力にスパース正則化を適用した実験であるとみなすこともできる. Auto Encoder は, 入力画像を低次元の内部表現に圧縮し, その後再構成するニューラルネットワークのことである [58] . これは, スパース正則化を線形なニューラルネットワークに導入することで, ニューラルネットワークでも一次視覚野の受容野の形成を促進することができることを示唆している.

2.4 スパース正則化

ニューラルネットワークで用いられる正則化手法の多くは, 変数の数や, その値が不必要に増加することを防ぐことで, 分類精度などの性能向上や学習の安定化を目指している. 例えば, ネットワークの重みに対する正則化は, 重みの L2 ノルムを正則化 (重み減衰) として用いることで達成される. これは, 学習サンプルを学習し過ぎて, 学習サンプルに含まれない未知のサンプルに対する分類性能が低下する過学習を防ぐ効果がある. それに対し, Batch Normalization や ELU 活性化関数は, あくまでも, 特徴のサンプル平均を 0 に近づけることで, 学習の安定化を図っている. また, Olshausen らは, ネットワークを構成するニューロンの出力にスパース制約を導入することで, 受容野の自己組織化を達成した [17].

本節では, CNN のニューロンの入力, すなわち, ReLU 活性化関数の入力にスパース制約を導入すること

で, CNN の学習の安定化や, 分類性能を向上させるスパース正則化を提案する．ReLU の入力にスパース制約を導入することで, ニューロンの出力値が必要以上に増加することがなくなり, 各ニューロンの出力の平均も 0 に近くなる．その結果, Bias Shift Problem による悪影響も緩和され, 学習の安定化や分類性能の向上に繋がる．

あるニューロンに対するスパース制約は, n 番目のサンプルを入力した時の各ニューロンの入力（活性化関数に入力される特徴マップ $\boldsymbol{u}_n \in \mathbb{R}^D$ の各要素）を用いて

$$\mathcal{S}(\boldsymbol{u}_n) = \frac{1}{D}\sum_{d}^{D} \log\left(1 + u_{n,d}^2\right) \tag{2.33}$$

とする．これを図で表したのが, 図 2.6 である．これは, Olshausen ら [17] が受容野の自己組織化のために提案したスパース正則化の一つである．彼らがあくまでも線形な Auto Encoder における受容野の自己組織化のための正則化として $\mathcal{S}$ を用いたのに対し, 本研究では CNN の分類精度向上, 学習の安定化のための正則化として用いる．CNN の学習は, N 個の学習サンプルを用意し, 損失関数

$$E = \mathcal{L} + \frac{\lambda}{N}\sum_{n}^{N} \mathcal{S}(\boldsymbol{u}_n) \tag{2.34}$$

を最小化することで行われる．ここで, $\mathcal{L}$ は経験的分類誤差（ソフトマックスクロスエントロピー）, λ は各ニューロンのスパース性を制御するためのハイパーパラメータ, N はサンプル数である．スパース正則化を異なるチャネル・層のニューロンに対して適用する場合, 損失関数は

$$E = \mathcal{L} + \frac{\lambda}{N}\sum_{n}^{N,C,L} \mathcal{S}(\boldsymbol{u}_n^{(c,l)}) \tag{2.35}$$

となる．C はチャネル数, L は層の数である．式 (2.35) を最小化することで, ネットワークの経験的分類誤差と各ニューロンのスパース性が改善され, 結果としてネットワークの分類精度も改善される．

勾配降下法を用いて学習されるネットワークに対するスパース正則化の効果を理解するには, 正則化項の微分について考える必要がある．あるサンプルを入力した時のニューロンの入力 $u_{n,d}$ に関する正則化項 $\mathcal{S}(u_{n,d})$ の微分は

$$\frac{\partial \mathcal{S}(u_{n,d})}{\partial u_{n,d}} = \frac{2u_{n,d}}{1 + u_{n,d}^2}. \tag{2.36}$$

である．正則化項 $\mathcal{S}(u_{n,d})$ の微分のグラフは図 2.5 の赤線で示している．図 2.5 から, ニューロンの入力が正の値の時の微分は正の値であり, 負が入力された場合には負の値になることがわかる．勾配降下法を用いてネットワークの変数の学習を行う場合, 変数の値は勾配の符号と逆方向に更新されるため, スパース正則化を導入することでニューロンの出力の不必要な増加を防ぐことが期待できるだけでなく, ニューロンの負の入力値も 0 に近づけることができる．これにより, ネットワークの分類精度も向上させることができる．Batch Normalization を CNN に適用した場合でも同様の効果があるが, Batch Normalization は学習サンプルの分散を計算する必要があるため, サンプルの偏りが著しい場合には, 学習がうまくいかなくなることがある．それに対し, 本節で提案するスパース正則化にはそのような恐れはない．

つづいて, ELU 活性化関数とスパース正則化を比較し, その違いについて議論する．式 (2.30) で示した ELU は, 0 以上の値を出力する ReLU 活性化関数を改良し, 負の値も出力できるようにした活性化関数である．図 2.5 の青線と緑線で示すように, 負の値が入力されたときの ReLU の微分は 0 であるのに対し, ELU の

微分は正である．これは, ELU に負の値が入力されると, ニューロンの入出力値がますます大きな負の値となるように変数を学習することを意味する．ELU が負の値を出力することでニューロンの出力のサンプル平均を 0 に近づけることができるが, 正の出力値を 0 に近づける効果は無く, その値は不必要に大きいままである．そのため, Bias Shift Problem の改善には未だ不十分であると考えられる．それに対し, ReLU の入力に対するスパース正則化は, ReLU の出力の平均を 0 に近づけるだけでなく, ReLU の入出力の不必要に大きな値を小さくする効果もあるため, Bias Shift Problem を改善し, 分類精度も向上させることができる．

これまで ReLU (ニューロン) の入力にスパース正則化を適用することを前提に議論してきたが, 出力に対してスパース正則化を導入することもできる．これも, ReLU の入力に対してスパース正則化を導入した場合と同様の効果がある．しかし, ReLU は負の入力に対する出力とその微分を 0 にするので, ニューロンの負の入力の値を小さくする効果は失われてしまう．

最後に, 有名な正則化法の一つである, 重み変数 $\boldsymbol{w}$ に対する L2 正則化 (重み減衰)

$$\mathcal{E} = \mathcal{L} + \|\boldsymbol{w}\|^2 \tag{2.37}$$

をスパース正則化と比較する．式 (2.37) の第二項が重み減衰である．ニューロンの入力に対するスパース正則化と重み減衰の大きな違いは, 重みの更新時にニューロンの反応を考慮するかどうかである．重み減衰を適用すると, 重み w_i に関する損失関数の勾配は

$$\frac{\partial E}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial w_i} + \frac{\partial \|\boldsymbol{w}\|^2}{\partial w_i} \tag{2.38}$$

$$= \frac{\partial \mathcal{L}}{\partial w_i} + 2w_i \tag{2.39}$$

となるため, 重み w_i の更新にニューロンの反応そのものは考慮されない．それに対し, ニューロンの入力 $\boldsymbol{u}_n$ に対してスパース正則化を適用した時の勾配は

$$\frac{\partial E}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial w_i} + \frac{\mathcal{S}\left(\boldsymbol{u}_n\right)}{\partial u_{n,j}} \frac{\partial u_{n,j}}{\partial w_i} \tag{2.40}$$

$$= \frac{\partial \mathcal{L}}{\partial w_i} + \frac{\partial \log\left(1 + \|\boldsymbol{u}_n\|^2\right)}{\partial u_{n,j}} \frac{\partial u_{n,j}}{\partial w_i} \tag{2.41}$$

$$= \frac{\partial \mathcal{L}}{\partial w_i} + \frac{2u_{n,j}}{1 + \|\boldsymbol{u}_n\|^2} x_{n,k} \tag{2.42}$$

である．$u_{n,j}$ は重み w_i と繋がっている l 層のニューロンの入力, $x_{n,k}$ はそのニューロンと重み w_i で結ばれた $l-1$ 層のニューロンの出力である．重み w_i だけで無く, j 番目のニューロンの入力やその前の層のニューロンの出力も考慮する点が, 重み減衰とは異なる．

一般的に, ネットワークの学習が進むにつれ, 特定のサンプルが入力されたときに値を出力する (反応する) 重要なニューロンと, サンプルにかかわらず値が 0 になりやすい (反応しない) ためあまり重要度ではないニューロンが出現しやすいと言われている [59]．スパース正則化は, ニューロンの重要度に応じて勾配の大きさを調整することができるため, 重要なニューロンの学習を重点的に行えると考えられる．

また, すでに提案したニューロンに対するスパース正則化の拡張版として, チャネル単位 (Channel-Wise) でのスパース正則化を示す．式 (2.33) のスパース正則化は, ニューロン単位でのスパース制約のみを考慮し, 他のニューロンとの関係は無視しているのに対し, Channel-Wise なスパース正則化は, あるチャネルに含まれるニューロン全体のスパース制約を考慮する．それを定式化したものが

$$\mathcal{S}\left(\boldsymbol{u}_n^{(c,l)}\right) = \sum_{c}^{C} \log\left(1 + \|\boldsymbol{u}_n^{(c,l)}\|^2\right) \tag{2.43}$$

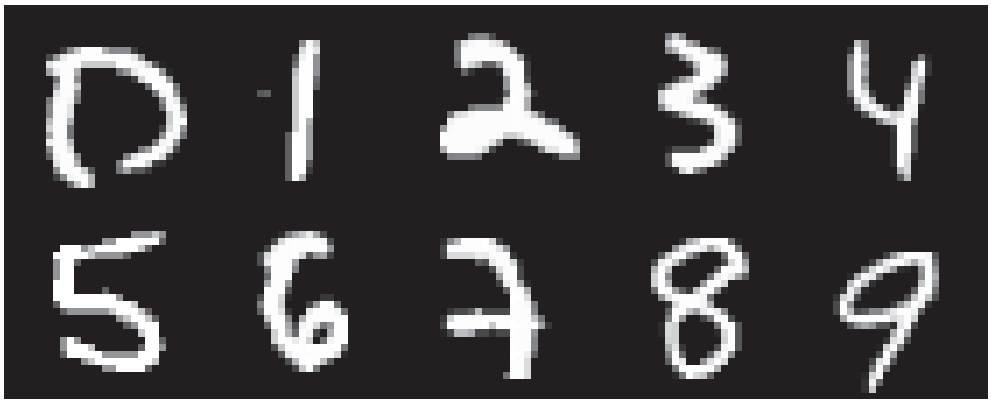


図 2.7: MNIST のサンプル．クラス数は, 数字の 0 から 9 までの 10 クラス．

図 2.8: CIFAR-10 のサンプル．クラス数は, 飛行機, 自動車, 鳥, 猫, 鹿, 犬, カエル, 馬, 船, トラックの 10 クラス．

である．これまでのニューロンに対するスパース正則化はニューロンごとに log を計算していたのに対し, Channel-Wise なスパース正則化は, あるチャネルに含まれるニューロン全体を用いて log を計算する．

2.5　スパース正則化の有効性を検証

本節では, いくつかの画像分類用データセットと CNN を用いて, 2.4 章で提案したスパース正則化の有効性を評価する．使用するデータセットは, 画像分類タスクにおいて有名な MNIST [60] と CIFAR-10/100 [61] である．MNIST [60] は, 図 2.7 に示す通り, 0 から 9 までの 10 種類の数字 (クラス) からなる手書き数字認識用のデータセットである．データセットは 6 万枚の学習用画像と 1 万枚のテスト画像からで構成され, 全て白黒画像である．画像のサイズは 28×28 ピクセルである．CIFAR-10/100 [61] は, 図 2.8 に示すような物体分類用のデータセットで, それぞれ 10 クラス, 100 クラスの RGB カラー画像が含まれている．どちらも 5 万枚の学習用画像と 1 万枚のテスト画像からなり, 画像サイズは 32×32 ピクセルである．各画像のピクセル値は, 全て 0 から 1 の値に正規化してある．

スパース正則化は, 主にネットワークの活性化関数や Pooling 層の入力に適用する．その際, スパース正則化を導入する層の違いによる影響も評価する．CNN の各種変数の初期値を 3 回変えてネットワークの分類性

表 2.1: LeNet-5 like CNN.

Layer	Parameters
Conv1	32 filters, 5×5, pad $= 0$, ReLU
Pool1	Max-pooling, 2×2, pad $= 0$
Conv2	64 filters, 5×5, pad $= 0$, ReLU
Pool2	Max-Pooling, 2×2, pad $= 0$
FC1	256 fully-connected filters, ReLU
FC2	10 or 100 fully-connected filters
output	Softmax

能を評価し, その平均と標準偏差を実験結果として報告する.

2.5.1 ニューロンに対するスパース正則化の有効性

まず, スパース正則化の有効性を評価するために, LeNet5 [2] を模した層の浅い CNN (LeNet-5 like (表 2.1)) を用いて, 基礎的な実験を行う. ネットワークは, ミニバッチのサイズを 100 とし, 確率的勾配降下法 (SGD) を用いて学習される. データセットが CIFAR-10 のときの学習回数は 1,000 epoch, CIFAR-100 のときの学習回数は 2,000 epoch である. CIFAR-10/100 の学習画像を用いてネットワークを学習し, その後, テスト画像をネットワークに入力した結果を表 2.2, 2.3 に示す.

表に示した分類精度 (入力) は, スパース正則化や Batch Normormalization をニューロンの入力 (ReLU の入力) に適用した時の分類精度, 分類精度 (出力) は, 比較として, それらをニューロンの出力に適用した時の分類精度である. スパース正則化なしの行は, 上から順に, 表 2.1 に示したシンプルなネットワーク (Baseline), 重み減衰を Baseline に導入したネットワーク (Weight Decay), Baseline の ReLU を ELU に置き換えたネットワーク (ELU), Baseline に Batch Normalization を適用したネットワーク (Batch Norm) である. スパース正則化ありの各行は, スパース正則化を導入した層を示している. 例えば, Conv & Pool は, 全ての畳み込み層の入力もしくは出力と, Pooling 層の入力にスパース正則化が適用された時の分類精度を示している. それに対し, Conv1 & Pool2 は, 一層目の畳み込み層と二層目の Pooling 層にスパース正則化が適用された時の分類精度を意味する. また, スパース正則化と Batch Normalization の相性を検証するために, スパース正則化 (Conv & Pool & FC) と Batch Normalization の組み合わせ (スパース正則化 + Batch Norm) も検証した. この時の分類精度 (入力) は, 畳み込みフィルタ, スパース正則化, Batch Normalization, ReLU の順に畳み込み層を構築したときのものである. それに対し, 分類精度 (出力) は, 畳み込みフィルタ, Batch Normalization, スパース正則化, ReLU の順に畳み込み層を構築したときのものである. Channel-Wise は, 全ての層に対して, チャネル単位のスパース正則化を適用したときの分類精度を意味している. 実験では, ニューロンのスパース制約の強さをコントロールするハイパーパラメータを, 層ごとに $\lambda/$(ニューロン数) とする. 学習係数は, いくつか試した中で, 最も分類精度が高かったときの値を用いる.

表 2.2 から, スパース正則化をニューロンに適用することで分類精度向上することがわかる. 特に, 全ての層の入力にスパース正則化を適用した Conv & Pool & FC と, Conv1 & Pool2 & FC が高い分類精度を示した. これらの条件では, Baseline や Weight Decay と比べて 6% 以上，Batch Norm と比べて約 0.2% ほど, 分類精度が高くなった. しかし, FC 層だけにスパース正則化を適用した時 (FC) は, あまり効果がなかった. これらの実験結果は, 畳み込み層や Pooling 層などの, CNN の中間層にスパース正則化を適用すべきであることを

表 2.2: CIFAR-10 の分類精度 (テスト画像, %). ネットワークは LeNet-5 like (表 2.1).

スパース正則化	ネットワーク	λ	学習係数	分類精度 (入力)	分類精度 (出力)
なし	Baseline	-	0.1	69.80 ±0.25	-
	Weight Decay	5e-5	0.01	68.36 ±0.25	-
	ELU	-	0.1	70.10 ±0.12	-
	Batch Norm	-	0.1	75.76 ±0.34	75.72 ±0.31
あり	Conv	1.0	0.001	74.22 ±0.56	72.36 ±0.35
	Pool	1.0	0.001	74.16 ±0.14	-
	FC	1.0	0.001	71.33 ±1.42	69.42 ±0.12
	Conv & Pool	1.0	0.001	**75.83** ±0.19	74.52 ±0.44
	Conv & FC	1.0	0.001	75.10 ±0.87	74.49 ±0.17
	Pool & FC	1.0	0.001	75.27 ±0.37	**76.03** ±0.56
	Conv1 & Pool1 & FC	1.0	0.001	75.30 ±0.44	74.63 ±0.47
	Conv1 & Pool2 & FC	1.0	0.001	**75.85** ±0.33	75.07 ±0.24
	Conv2 & Pool1 & FC	1.0	0.001	75.39 ±0.18	74.53 ±0.34
	Conv2 & Pool2 & FC	1.0	0.001	75.12 ±0.20	73.99 ±0.36
	Conv & Pool & FC	1.0	0.001	**75.93** ±0.34	75.50 ±0.26
	スパース正則化 + Batch Norm	1.0	0.001	69.58 ±0.25	69.90 ±0.16
	Channel-Wise	1.0	0.001	74.42 ±0.53	73.92 ±0.03

示している. これは, Bias Shift Problem と関係があるからである. Bias Shift Problem では, ある層の出力が 0 からずれるとそれより後ろの層に対してバイアスとして働くことで, 分類精度の悪化や学習の不安定化につながる. FC 層などの出力に近い層に対してスパース正則化を適用するよりも, 入力に近い層にスパース正則化を適用した方が, 後ろ層に対して働くバイアスの効果を軽減できるできるため, このような結果となった. 他にも, Channel-Wise なスパース正則化も分類精度を向上させることができた. それらに対し, スパース正則化 + Batch Norm は, 分類精度が低くなった. これは, この実験において, スパース正則化と Batch Norm の相性があまりよくなかったからと思われる.

また, ニューロンの入力と出力, どちらかにスパース正則化を適用したあとの分類精度を比較すると, ニューロンの入力に適用した方が高い分類精度を得られやすいことがわかった. これは, ReLU 活性化関数の出力にスパース正則化を適用すると負の入力に対する勾配が常に 0 になり, スパース正則化の効果が弱くなるのに対し, 入力に適用すると勾配が 0 以外の値を持つ可能性があるからである. このことは, スパース正則化は ReLU の入力に適用すべきであることを示唆している.

CIFAR-100 を用いた実験でも, 同様の傾向を示している (表 2.3). 表 2.3 から, CIFAR-100 では, スパース正則化 (Conv & Pool & FC) の効果はより顕著で, Baseline や Weight Decay と比べて 12% 以上, Batch Norm と比べても約 5% 分類精度が高い. それに対し, FC 層のみにスパース正則化を適用しても分類精度があまり改善しなかったことから, やはりスパース正則化はネットワークの中間層に適用すべきであることを示している.

ネットワークの学習の挙動を理解するために, 学習の推移を表した学習曲線を図 2.9 に示す. 比較のために,

表 2.3: CIFAR-100 の分類精度 (テスト画像, %). ネットワークは LeNet-5 like (表 2.1).

スパース正則化	ネットワーク	λ	学習係数	分類精度 (入力)	分類精度 (出力)
なし	Baseline	-	0.01	33.39 ±0.62	-
	Weight Decay	5e-4	0.01	34.01 ±0.27	-
	ELU	-	0.1	35.04 ±0.30	-
	Batch Norm	-	0.1	41.01 ±0.16	37.95 ±0.17
あり	Conv	1.0	0.001	**43.16** ±0.06	38.47 ±0.29
	Poo	1.0	0.001	**41.79** ±0.10	-
	FC	1.0	0.001	34.30 ±4.77	27.17 ±0.57
	Conv & Pool	1.0	0.001	**45.28** ±1.71	**42.09** ±1.26
	Conv & FC	1.0	0.001	**43.49** ±1.80	41.00 ±0.29
	Pool & FC	1.0	0.001	**45.04** ±0.17	**44.82** ±0.05
	Conv1 & Pool1 & FC	1.0	0.001	**42.60** ±0.35	37.11 ±0.56
	Conv1 & Pool2 & FC	1.0	0.001	**45.44** ±0.26	**44.39** ±0.41
	Conv2 & Pool1 & FC	1.0	0.001	**44.56** ±0.15	**41.69** ±0.20
	Conv2 & Pool2 & FC	1.0	0.001	**44.82** ±0.18	39.62 ±0.25
	Conv & Pool & FC	1.0	0.001	**46.72** ±0.95	**45.81** ±0.31
	スパース正則化 + Batch Norm	1.0	0.001	36.80 ±0.63	**41.11** ±0.40
	Channel-Wise	1.0	0.001	**41.41** ±0.30	30.45 ±0.11

Baseline, Batch Norm, 全ての層にスパース正則化を導入したネットワーク (Conv & Pool & FC) を載せた. Conv & Pool & FC の (Input) は, スパース正則化を入力に適用した結果, (Output) は出力に適用した結果である. これらのグラフから, スパース正則化は, Batch Norm より学習に時間がかかるものの, 分類精度が改善されることがわかる.

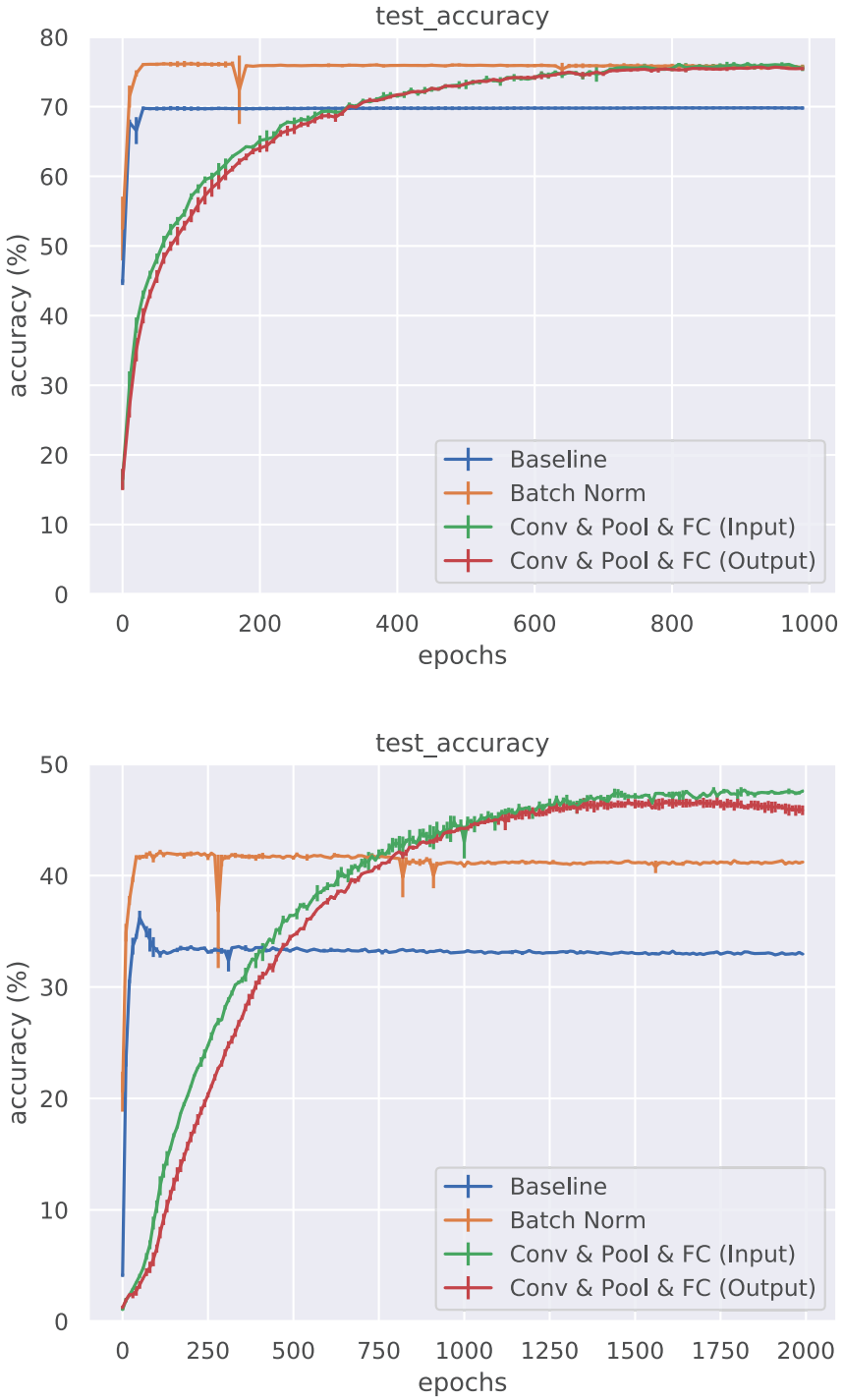


図 2.9: LeNet-5 like (表 2.1) の学習曲線 (テスト画像). 上側が CIFAR-10, 下側が CIFAR-100 のグラフ.

表 2.4: CIFAR-10 のテスト画像を入力した時の各層のニューロンの出力値の平均. ネットワークは LeNet-5 like (表 2.1).

スパース正則化	ネットワーク	Conv1	Pool1	Conv2	Pool2	FC
なし	Baseline	0.24	0.35	0.40	0.85	1.02
	Weight Decay	0.64	0.80	1.43	2.41	2.16
	ELU	-0.03	0.13	0.07	0.86	0.69
	Batch Norm	0.21	0.36	0.27	0.57	0.49
あり	Conv	0.06	0.10	0.09	0.18	0.49
	Pool	0.05	0.09	0.05	0.11	0.44
	FC	0.18	0.24	0.25	0.39	0.12
	Conv & Pool	0.026	0.05	0.03	0.07	0.30
	Conv & FC	0.04	0.07	0.05	0.10	0.09
	Pool & FC	0.04	0.08	0.04	0.09	0.12
	Conv1 & Pool1 & FC	0.03	0.06	0.16	0.24	0.10
	Conv1 & Pool2 & FC	0.04	0.08	0.03	0.06	0.09
	Conv2 & Pool1 & FC	0.03	0.05	0.05	0.10	0.10
	Conv2 & Pool2 & FC	0.07	0.12	0.04	0.09	0.13
	Conv & Pool & FC	**0.03**	0.05	**0.03**	**0.06**	**0.09**
	スパース正則化 Batch Norm	0.01	**0.02**	0.03	0.08	0.14
	Channel-Wise	0.10	0.15	0.06	0.12	0.14

2.5.2 スパース正則化の挙動の解析

表 2.4, 表 2.5 は, それぞれ CIFAR-10/100 のテスト画像をネットワークに入力した時の, 各層のニューロンの出力値 (反応) の平均である. スパース正則化を各層の入力に適用することで, 適用した層のニューロンの出力値の平均は, 大きく下がっている. 更に, Conv 層や Pooling 層にスパース正則化を適用することで, 適用した層の出力値だけでなく, 出力に近い FC 層の出力値の平均も大きく下がっている. それに対し, FC 層のみにスパース正則化を適用すると, それ以前の層の出力値の平均は大きくなった. これらの結果から, 中間層の畳み込み層や Pooling 層にスパース正則化を導入することで, ある層の出力が後ろの層に対してバイアスとして働く Bias Shift Problem を緩和できるということがわかる.

表 2.5: CIFAR-100 のテスト画像を入力した時の各層のニューロンの出力値の平均. ネットワークは LeNet-5 like (表 2.1).

スパース正則化	ネットワーク	Conv1	Pool1	Conv2	Pool2	FC
なし	Baseline	0.95	1.13	2.0	3.3	4.68
	Weight Decay	1.85	2.09	2.84	4.91	4.07
	ELU	-0.07	0.10	-0.10	0.64	1.47
	Batch Norm	0.26	0.42	0.32	0.70	1.39
あり	Conv	0.09	0.16	0.26	0.50	2.10
	Pool	0.09	0.16	0.16	0.33	2.27
	FC	0.62	0.73	1.32	1.85	0.32
	Conv & Pool	0.04	0.08	0.08	0.16	1.09
	Conv & FC	0.06	0.11	0.12	0.23	0.26
	Pool & FC	0.06	0.09	0.07	0.15	0.27
	Conv1 & Pool1 & FC	0.071	0.13	0.78	1.122	0.42
	Conv1 & Pool2 & FC	0.09	0.15	0.10	0.20	0.37
	Conv2 & Pool1 & FC	0.07	0.11	0.16	0.31	0.37
	Conv2 & Pool2 & FC	0.15	0.22	0.09	0.189	0.34
	Conv & Pool & FC	0.04	0.07	**0.06**	**0.12**	**0.23**
	スパース正則化 + Batch Norm	**0.01**	**0.03**	**0.06**	0.18	0.36
	Channel-Wise	0.31	0.42	0.32	0.55	0.41

2.5.3 異なるデータセットを用いた比較

つづいて, 異なるデータセットでもスパース正則化が効果的であることを, MNIST を用いて検証する. ネットワークは, 引き続き LeNet-5 like (表 2.1) を用いる. スパース正則化を様々な層の入力に適用し, その分類精度を表 2.6 に示す. このときも, ニューロンのスパース制約の強さをコントロールするハイパーパラメータを, 層ごとに λ/(ニューロン数) とする.

ここでも, これまでと同様の結果を示し, 全ての層にスパース正則化を適用した時 (Conv & Pool & FC) に高い分類精度を示し, Baseline や Weight Decay と比べて約 0.13%, Batch Norm と比べて 0.04 %, 分類精度が高かった. CIFAR10/100 のときと比べて分類精度があまり改善しなかったのは, Baseline ですら分類精度がほぼ 100% であるため, 改善の余地がほとんどなかったからである.

これらの結果から, 異なるデータセットでもスパース正則化は効果的であり, さらに全ての層の入力に適用すべきであることを検証できた.

表 2.6: MNIST の分類精度 (テスト画像, %). ネットワークは LeNet-5 like (表 2.1).

スパース正則化	ネットワーク	λ	学習係数	分類精度
なし	Baseline	-	0.1	99.18 ±0.03
	Weight Decay	5e-6	0.1	99.18 ±0.02
	ELU	-	0.1	99.13 ±0.02
	Batch Norm	-	0.1	99.27 ±0.04
あり	Conv	0.1	0.01	99.24 ±0.02
	Pool	0.1	0.01	**99.37** ±0.05
	FC	0.1	0.01	**93.34** ±0.05
	Pool & FC	0.1	0.01	99.24 ±0.00
	Conv & Pool	0.1	0.01	99.26 ±0.07
	Conv & FC	0.1	0.01	99.27 ±0.05
	Pool & FC	0.1	0.01	99.24 ±0.00
	Conv & Pool & FC	0.1	0.01	**99.31** ±0.03
	スパース正則化 + Batch Norm	0.1	0.01	**99.28** ±0.02
	Channel-Wise	0.1	0.01	**99.39** ±0.02

2.5.4 層の深いネットワークにおける比較

ここでは, ネットワークを構成する層が LeNet-5 like よりも深い, VGG-13 を模した CNN (VGG-13 like (表 2.8)) におけるスパース正則化の効果を検証する. このネットワークも, LeNet5-like と同様の条件で 1,000 epoch 学習される. 本実験では, 先ほどよりも条件を絞り, 正則化無しのネットワーク (Baseline), 重み減衰を用いた (Weight Decay), ReLU の代わりに ELU を用いた (ELU), 正則化として Batch Normalization を用いたネットワーク (Batch Norm), スパース正則化をニューロンの入力に用いた提案手法 (Conv & Pool & FC), Batch Normalization の入力にスパース正則化 (Conv & Pool & FC) を組み合わせたネットワーク (スパース正則化 + Batch Norm), 逆に Batch Normalization の出力にスパース正則化を組み合わせたネット

表 2.7: MNIST のテスト画像を入力した時の各層のニューロンの出力値の平均. ネットワークは LeNet-5 like (表 2.1).

スパース正則化	ネットワーク	Conv1	Pool1	Conv2	Pool2	FC
なし	Baseline	0.32	0.48	0.83	1.74	2.28
	Weight Decay	0.34	0.49	0.84	1.72	1.83
	ELU	0.14	0.33	0.35	1.56	2.04
	Batch Norm	0.32	0.52	0.46	1.01	1.12
あり	Conv	0.03	0.06	0.06	0.13	0.47
	Pool	0.02	0.03	0.02	0.06	0.38
	FC	0.15	0.24	0.16	0.29	0.10
	Conv & Pool	0.01	0.03	0.03	0.07	0.37
	Conv & FC	0.02	0.05	0.04	0.09	0.10
	Pool & FC	0.02	0.03	0.02	**0.04**	0.10
	Conv & Pool & FC	0.014	0.03	0.02	0.05	0.10
	スパース正則化 + Batch Norm	**0.00**	**0.01**	**0.012**	**0.04**	**0.09**
	Channel-Wise	0.04	0.07	0.03	0.06	0.10

ワーク (Batch Norm + スパース正則化), Channel-Wise なスパース正則化 (Channel-Wise) の 8 つを比較する. データセットには CIFAR-10/100 を用い, テスト画像の分類精度を表 2.9, 表 2.10 に示す. このときも, ニューロンのスパース制約の強さをコントロールするハイパーパラメータを, 層ごとに λ/(ニューロン数) とする.

これらの結果から, スパース正則化を導入することで, Baseline に比べて分類精度が向上することがわかる. 分類精度は, CIFAR-10 のとき, Baseline や Weight Decay に比べ約 1.3 % 向上した. CIFAR-100 のときは, さらに効果的で, 7 % 以上分類精度を改善した. 他の正則化手法も分類精度を大幅に改善しているため, スパース正則化は, 他の手法と同様に, 過学習の改善に効果があることがわかる. また, VGG-13 like では, Batch Normalization の入力にスパース正則化を導入したスパース正則化 + Batch Norm の分類精度が最も高くなった. それに対し, Batch Normalization の出力にスパース正則化を導入した場合の分類精度は, Batch Norm より悪化した. これらのことから, スパース正則化は Batch Normalization の入力に対して用いられるのが好ましいということを示唆している. これは, おそらく, スパース正則化を Batch Normalization の出力に適用すると, 正規化の効果を打ち消されてしまうのに対し, 入力に適用するならばその効果も残ったままであるからだと考えられる.

2.6 ミニバッチに含まれるサンプルに偏りがある場合の比較

これまでは, ミニバッチに含まれるサンプルのクラスに大きな偏りがない状態で実験を行なってきた. しかし, サンプルに偏りがある場合, ネットワークの学習がうまくいかない場合がある. ここでは, ミニバッチに含まれるクラスに偏りがある場合のスパース正則化の有効性を検証する.

実験では, 各ミニバッチに含まれるクラス数が, 多くても 4 クラスまでになるように設定する. ミニバッチの偏り以外は, VGG-13 like を用いた実験 (2.5.4 章) と同じ条件で実験を行う. 実験結果を表 2.13 に示す.

表 2.8: VGG-13 like CNN.

Layer	Parameters
Conv1	64 filters, 3×3, pad $= 1$, ReLU
Conv2	64 filters, 3×3, pad $= 1$, ReLU
Pool1	Max-pooling, 2×2, pad $= 0$
Conv3	128 filters, 3×3, pad $= 1$, ReLU
Conv4	128 filters, 3×3, pad $= 1$, ReLU
Pool2	Max-pooling, 2×2, pad $= 0$
Conv5	256 filters, 3×3, pad $= 1$, ReLU
Conv6	256 filters, 3×3, pad $= 1$, ReLU
Pool3	Max-pooling, 2×2, pad $= 0$
FC1	1024 fully-connected filters, ReLU, DropOut ($p = 0.5$)
FC2	10 or 100 fully-connected filters
output	Softmax

表 2.9: CIFAR10 の分類精度 (テスト画像, %). ネットワークは VGG-13 like (表 2.8).

スパース正則化	ネットワーク	λ	学習係数	分類精度
なし	Baseline	-	0.01	88.48 $\pm$0.24
	Weight Decay	5e-6	0.01	88.53 $\pm$0.03
	ELU	-	0.01	88.51 $\pm$0.19
	Batch Norm	-	0.01	89.42 $\pm$0.11
あり	Conv & Pool & FC	0.1	0.01	88.79 $\pm$0.07
	スパース正則化 + Batch Norm	0.1	0.01	**89.80** $\pm$0.06
	Batch Norm + スパース正則化	0.1	0.01	89.28 $\pm$0.07
	Channel-Wise	0.1	0.01	88.78 $\pm$0.16

表に示した”クラス数”の列は, 学習時の各ミニバッチに含まれるクラス数を, 指定した数に制限したときの分類精度を意味している. 例えば, ”クラス数 1”は, ミニバッチに含まれるサンプルのクラス数は 1, ”クラス数 4”は, ミニバッチに含まれるサンプルのクラス数は 4 であることを意味している. クラス数を制限すると, Batch Norm を導入した CNN の分類精度は大きく悪化し, Baseline すら下回ったのに対し, スパース正則を導入した CNN では, 分類精度はあまり悪化せず, Baseline を上回った. 特に, クラス数を 1 に制限したときには, その差が顕著である.

これらの実験結果から, Batch Normalization などは, ミニバッチに含まれるサンプルが偏ると分類精度が大きく悪化するリスクがあるのに対し, スパース正則化は, そのようなリスクは低いことがわかる.

表 2.10: CIFAR100 の分類精度 (テスト画像, %). ネットワークは VGG-13 like (表 2.8).

スパース正則化	ネットワーク	λ	学習係数	分類精度
なし	Baseline	-	0.01	59.06 ±0.52
	Weight Decay	5e-6	0.01	59.13 ±0.56
	ELU	-	0.01	65.32 ±0.14
	Batch Norm	-	0.01	66.61 ±0.07
あり	Conv & Pool & FC	1.0	0.01	66.14 ±0.13
	スパース正則化 + Batch Norm	0.1	0.01	**67.47** ±0.20
	Batch Norm + スパース正則化	0.1	0.01	**67.13** ±0.30
	Channel-Wise	0.1	0.01	63.58 ±0.38

表 2.11: CIFAR-10 のテスト画像を入力した時の各層のニューロンの出力値の平均. ネットワークは VGG-13 like (表 2.8).

スパース正則化	ネットワーク	Conv1	Conv3	Conv6	FC1
なし	Baseline	0.10	0.30	0.28	0.31
	Weight Decay	0.10	0.31	0.28	0.31
	ELU	0.01	0.13	0.18	0.89
	Batch Norm	0.26	0.15	0.21	0.37
あり	Conv & Pool & FC	**0.05**	**0.05**	**0.04**	**0.14**
	スパース正則化 + Batch Norm	0.08	0.07	0.11	0.26
	Batch Norm + スパース正則化	0.15	0.11	0.17	0.25
	Channel-Wise	0.08	0.16	0.11	0.18

2.7 本章のまとめ

本章では, 畳み込みニューラルネットワーク (CNN) の ReLU 活性化関数の入力 (ニューロンの入力) に対してスパース正則化を適用することで, CNN の分類精度が改善されることを確認した．これは，ReLU 活性化関数の入力に対してスパース正則化を適用すると ReLU の入力が 0 に近づき，各ニューロンの出力が必要以上に大きくならない, スパースなネットワークになったからである．他にも, Batch Normalization の入力に対してスパース正則化を適用することで, Batch Normalization 以上の分類精度を達成できる場合があることも示した．また, 学習サンプルが極端に偏り, ネットワークの学習がうまくいかない条件下でも, スパース正則化ならばその悪影響が少ないことも示した.

これらの結果は, CNN の分類精度を改善する新たな正則化手法の有効性を示しただけでなく, 未だ多くの部分が未知である CNN を理解するのにも役立つと考えられる.

表 2.12: CIFAR-100 のテスト画像を入力した時の各層のニューロンの出力値の平均. ネットワークは VGG-13 like (表 2.8).

スパース正則化	ネットワーク	Conv1	Conv3	Conv6	FC1
なし	Baseline	0.10	0.35	0.66	0.38
	Weight Decay	0.10	0.37	0.69	0.39
	ELU	0.01	0.07	0.19	0.91
	Batch Norm	0.27	0.15	0.12	0.37
あり	Conv & Pool & FC	**0.04**	**0.03**	**0.05**	**0.21**
	スパース正則化 + Batch Norm	0.08	0.09	0.09	0.36
	Batch Norm + スパース正則化	0.17	0.12	0.11	0.36
	Channel-Wise	0.07	0.13	0.17	0.22

表 2.13: 学習時の各ミニバッチに含まれるクラス数に偏りがある場合の分類精度 (CIFAR-10 のテスト画像, %). VGG-13 like を 1, 000 epochs 学習した. 正則化は活性化関数の入力に適用.

スパース正則化	ネットワーク	λ	学習係数	クラス数 1	クラス数 2	クラス数 4
なし	Baseline	-	0.01	87.20 $\pm$0.30	87.61 $\pm$0.01	88.04 $\pm$0.08
	Batch Norm	-	0.01	14.78 $\pm$1.7	65.53 $\pm$5.34	87.97 $\pm$0.89
あり	Conv & Pool & FC	0.1	0.01	**87.72** $\pm$0.10	**88.20** $\pm$0.16	**88.41** $\pm$0.30

3　判別正則化の適用

3.1　はじめに

CNN を含めたニューラルネットワークは, 脳の情報処理を模倣した手法で, 画像分類などにおいて多くの研究が行われてきた [25, 5, 26, 27, 28]. ネットワークの分類精度は, データサンプルをネットワークに入力し, その中間層から得られた特徴によって大きく左右されるため, 多くの研究は, 中間層を工夫することで, 分類精度の向上を試みている [25, 5, 26, 27, 28]. それに対し, 入力されたサンプルを実際に分類し, 分類結果を出力する出力層 (分類器) については, 中間層に対するものほど関心が向けられてこなかった. そのため, 分類器の振る舞いや, 分類器にとって好ましい特徴については未知な部分がある.

本章では, まず, 3.2 章にて判別基準を定義し, 分類器にとっての特徴の好ましさを評価するツールを準備する. 次に, 3.3 章ではニューラルネットワークの分類器を構成する各ニューロンの挙動を, 数式的・実験的に解析し, 分類器にとって好ましい特徴について議論する. ここでは, ソフトマックス関数を用いて多クラス分類問題を解く分類器に注目し, ある特定のクラス (ターゲットクラス) とそれ以外のクラス (非ターゲットクラス) を分類する 2 クラス分類問題を, 分類器の各ニューロンがそれぞれ解いていることを示す. また, 分類器にとって理想的な入力 (中間層から出力される特徴) は, 分類器の各ニューロンの入力が 2 つのガウス分布 (平均はクラス毎に異なり, 分散は全てのクラスで等しい) であることを数式的に導出する. さらに分類器の各ニューロンに入力される特徴がそれに近い分布であること, ただし, 各分布の一部が重なっており, 分類精度の悪化を招いていることを実験的にも確認する. 次に, 分類器にとってより好ましい特徴を中間層から得るために, かつ, 各分布の重なりを減らすために, 学習中の分類器の各ニューロンに判別基準を適用する判別正則化を提案する. ニューラルネットワークに判別基準を適用した研究は数多く存在するが, それらの手法は出力層のニューロン全体から得られる特徴ベクトルに対して判別基準を用いるのに対し, 提案手法は分類器の各ニューロン (特徴ベクトルの各要素) に対して個別に判別基準を用いるという点で, 従来のものとは異なる [62, 63]. 3.4 章では, ソフトマックス関数とシグモイド関数, それぞれを分類器の活性化関数に用いた時の違いについても議論する. シグモイド関数を多クラス問題に拡張したものがソフトマックス関数であるが, 分類器の各ニューロンから見てどのような違いがあるかを議論する. 3.5 章では, いくつかのデータセットとネットワークを用いて, 判別正則化の有効性を実験的に検証する. 3.6 章では, ソフトマックス関数とシグモイド関数の違いによるネットワークへの影響も, 実験的に検証する.

本章は, 過去に発表した研究 [64] の論点を整理・追加したものである.

3.2　判別基準

判別基準は, 複数クラスから得られる特徴がどのくらい分離されているか (分離度) を評価するために使用されるツールである. Fisher は, この判別基準を用いて線形判別分析 (LDA) を定義した [19]. それ以来, 判別規準は多くの応用分野で利用されてきた. 例えば, 大津は, 大津の二値化として知られる, 画像中の領域を二値化するアルゴリズムの中で, そのしきい値を選択する基準として用いた [65]. 他にも, 顔認識のための特徴点の検出にも判別規準が用いられた [66].

判別基準や線形判別分析をニューラルネットワークに応用した研究も数多く行われている. Osman らは, ニューラルネットワークの線形出力と教師データの平均二乗誤差を最小化することが, 判別基準を最小化することと等価であることを示した [67]. Chen らは, 判別基準とニューラルネットワークの組み合わせを提案し

た [68]. 彼らは, ニューラルネットワークを用いて決定木を設計し, この決定木の中間層に判別規準を採用することで, 大規模で複雑なタスクを単純なサブタスクに分割できることを, 実験的に示した. 人物識別を行うために, 判別規準をニューラルネットワークの出力ベクトルに対して適用し, それを損失関数に組み込む手法も提案された [62]. Wu らは, ニューラルネットワークを用いて, 入力データから特徴ベクトルを生成し, LDA と組み合わせたサポートベクターマシン (SVM) を用いて人物識別を行った [69]. 一方, 栗田らは, 学習したニューラルネットワークの出力を事後確率の推定値とし, 推定された事後確率を用いて非線形判別分析を行った [63]. また, Abe らは, 本章の提案手法 [64] を発展させ, 判別基準をネットワークの中間層に導入した [70].

ここからは, K クラス分類問題をもとに, 判別基準を定式化する. まず, クラスラベル k を持つ 1 次元特徴の学習サンプルを $\{x_{k,n}|n=1,\ldots,N_k; k\in\{1,\ldots,K\}\}$ とする. N_k はクラス k に属する学習サンプルの総数であり, 全学習サンプル数 N との関係は $N=\sum_{k=1}^{K} N_k$ である. すると, それぞれのクラスに属する特徴の分散 (クラス内分散) σ_W^2 は

$$\sigma_W^2 = \sum_{k}^{K}\left(\frac{1}{N_k}\sum_{n}^{N_k}(x_{k,n}-\mu_k)^2\right) \tag{3.1}$$

で与えられる. μ_k はクラス k に属する特徴の平均であり

$$\mu_k = \frac{1}{N_k}\sum_{n}^{N_k} x_{k,n} \tag{3.2}$$

である. 同様に, 全ての特徴の分散 (全分散) σ_T^2 は

$$\sigma_T^2 = \frac{1}{N}\sum_{k}^{K}\sum_{n}^{N_k}(x_{k,n}-\mu)^2 \tag{3.3}$$

で与えられる. μ はすべての特徴の平均であり

$$\mu = \frac{1}{N}\sum_{k}^{K}\sum_{n}^{N_k} x_{k,n} \tag{3.4}$$

で与えられる. 式 3.1, 3.3 から, 1 次元特徴の判別基準は

$$\gamma = \frac{\sigma_W^2}{\sigma_T^2} \tag{3.5}$$

の形に書ける. また, σ_T^2 と σ_W^2 の関係は, クラス間分散 σ_B^2 を用いて

$$\sigma_T^2 = \sigma_W^2 + \sigma_B^2 \tag{3.6}$$

と表すことができる. ここで, クラス間分散 σ_B^2 は

$$\sigma_B^2 = \sum_{k}\frac{N_k}{N}(\mu_k-\mu)^2 \tag{3.7}$$

と定義される. 本章で扱う判別基準 γ と他の一般的な判別基準 $\frac{\sigma_B^2}{\sigma_T^2}$ や $\frac{\sigma_B^2}{\sigma_W^2}$ との関係性は

$$\gamma = \frac{\sigma_W^2}{\sigma_T^2} \tag{3.8}$$

$$= -\frac{\sigma_B^2}{\sigma_T^2} \tag{3.9}$$

$$= \frac{1}{1 + \frac{\sigma_B^2}{\sigma_W^2}} \tag{3.10}$$

となることから, 判別基準 γ の最小化は, 他の標準的な判別基準の最大化と等価である.

3.3 判別正則化の適用

多クラス分類問題を解くニューラルネットワークは, ネットワークに入力されたサンプルから特徴を生成する中間層と, 中間層から出力された特徴を用いて分類を行う出力層 (分類器) で構成される. そのため, ネットワークの分類精度を向上させるには, 中間層の改良だけでなく, 分類器の改良も必要不可欠である.

本節では, K クラス分類問題を解くニューラルネットワークの分類器の挙動を解析し, 分類器にとって理想的な特徴とは何かについて議論したのち, その生成を加速する判別正則化を提案する. まず, 分類器を構成する K 個のニューロンのうち, その一つに入力される特徴 y を用いて, K クラス分類問題を考えてみる. このとき, k 番目のクラス C_k の事前確率は, $P(C_k)$ で与えられる. クラス C_k の条件付き確率分布 $p(y|C_k)$ が, 平均 μ_k と分散 σ^2 のガウス分布で与えられると仮定すると

$$p(y|C_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y-\mu_k)^2}{2\sigma^2}\right\} \tag{3.11}$$

と表すことができる. すべてのクラスの分散が σ^2 であると仮定すると, クラス C_k の事後確率は

$$\begin{aligned} P(C_k|y) &= \frac{P(C_k)p(y|C_k)}{p(y)} \\ &= \frac{P(C_k)p(y|C_k)}{\sum_i^K P(C_i)p(y|C_i)} \\ &= \frac{\exp\left\{(y-\mu_k)^2\right\}}{\sum_i^K \exp\{(y-\mu_i)^2 + \log\frac{P(C_i)}{P(C_k)}\}} \\ &= \frac{\exp\left\{(y-\mu_k)^2\right\}}{\sum_i^K \alpha_i \exp\{(y-\mu_i)^2\}} \qquad (3.12) \\ &= \frac{1}{1 + \sum_{i,i\neq k}^K \beta_i \exp(\beta_i)} \qquad (3.13) \end{aligned}$$

となる. ここで, $\alpha_i = \exp\left(\log\frac{P(C_i)}{P(C_k)}\right)$, $\beta_i = 2\mu_i y - \mu_i^2$ である. 式 (3.12) は, クラスごとに異なる平均 μ_k とクラス全体で同じ分散 σ^2 を持つガウス分布が分類器の各ニューロンに与えられるならば, 重み付きソフトマックス関数が分類器の活性化関数として理想的であることを意味している. そのためか, K クラス分類問題に用いられるネットワークの多くは, 分類器にソフトマックス関数を導入している. また, 式 (3.13) から, K クラス問題を解くためにソフトマックス関数を用いた分類器は, 重み付きシグモイド関数のような関数を用いて, 自身に割り当てられた予測すべきクラス (ターゲットクラス) とそれ以外の非ターゲットクラスを分類す

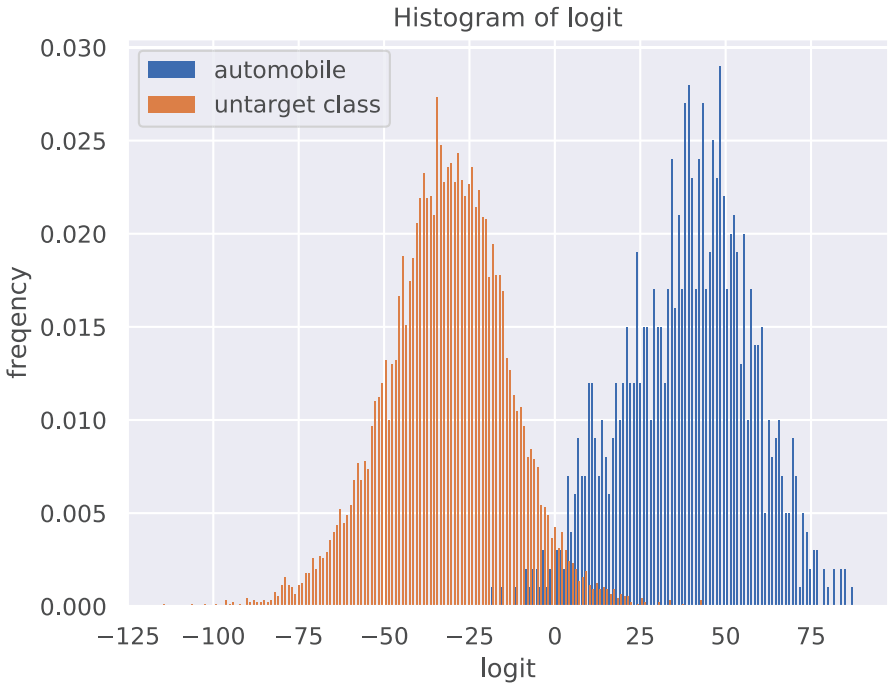


図 3.1: 分類器のあるニューロン (自動車クラス) のロジットのヒストグラム. グラフの横軸はロジットの値, 縦軸は頻度数, ビン数は 200 である. 赤い分布はターゲットクラス (自動車クラス) のサンプルから得られたヒストグラム, 青い分布は非ターゲットクラスのサンプルから得られたヒストグラムである.

る 2 クラス分類問題を, ニューロン毎に解いていることがわかる. それに対し, ネットワークの中間層は, 分類器の各ニューロンにとって理想的な特徴であるガウス分布を生成していると考えられる.

ネットワークが実際にそのような働きをしているか確認するために, 学習後の CNN を用いて, 分類器のあるニューロンの入力 (ロジット) の分布 (ヒストグラム) を図 3.1 に可視化した. このネットワークは, CIFAR-10 のデータセットを用いて学習された. 実験の詳細は 3.5 章に, 可視化の詳細は 3.5.2 章に記載した. 赤色の分布は, ターゲットクラス (この場合は自動車クラス) に属する入力サンプルから得られたロジットのヒストグラムであり, 青色はそれ以外のクラス (非ターゲットクラス) に属するサンプルから得られたロジットのヒストグラムである. 図 3.1 から, 各分布は, それぞれ平均が異なり, かつ, 分散が等しいガウス分布に近いことがわかる.

CNN の中間層は, 実際に, 2 つのガウス分布に近いものを出力していることを, 実験的に確認できた. また, 分類器の各ニューロンは, 2 つのガウス分布を用いて 2 クラス分類問題を解いていることも確認できた. しかし, 2 つの分布の一部は重なっているため, 分類が困難なデータサンプルが存在する.

この 2 つのガウス分布の生成を促進し, さらに, 各分布をより分離させるために, 分類器の各ニューロンの入力に対する制約として判別基準を用いる判別正則化を提案する. 3.2 章で触れたように, 判別基準は, 各特徴のクラス間での分離度を評価するために用いられる基準である [19][65]. 分類器のニューロン毎 (ロジットの要素毎) に判別規準 γ_k の値を最小化するようにネットワークを学習することで, 分類器の各ニューロンは, ターゲットクラスとそれ以外のクラスの 2 つの分布を分離させることができ, 結果として分類精度を向上させるこ

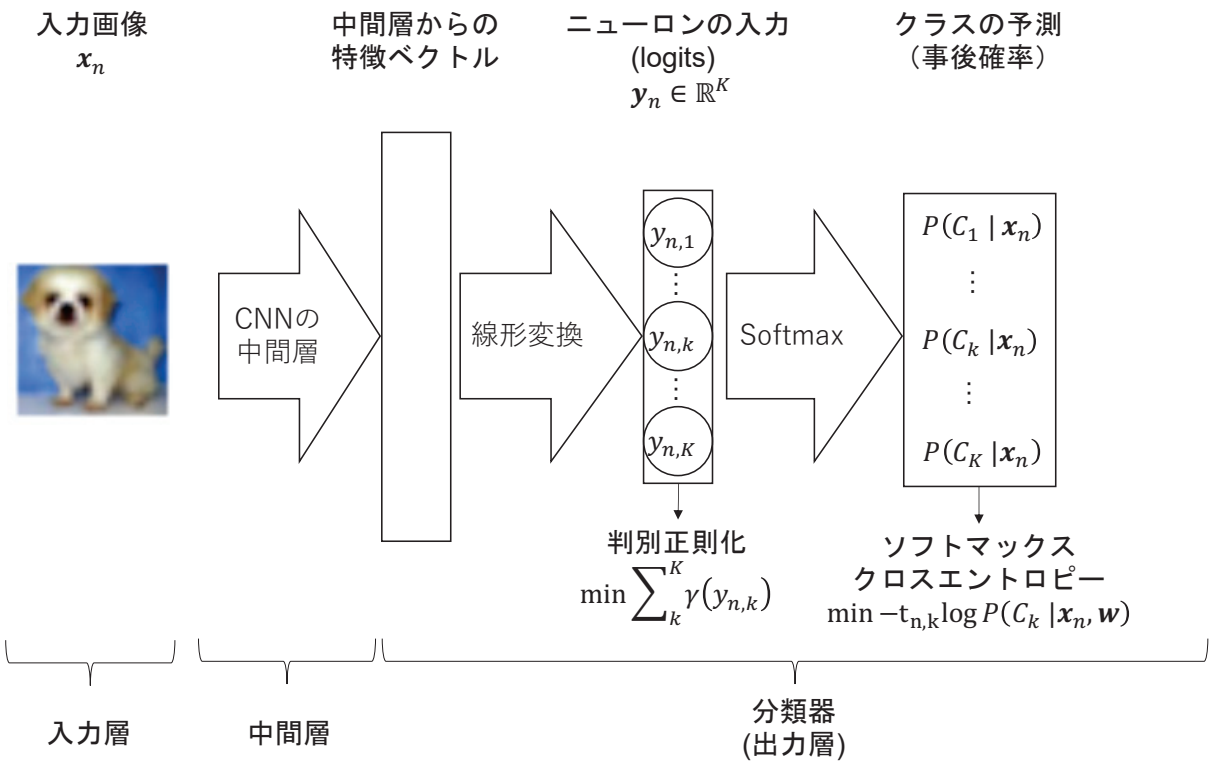


図 3.2: 判別正則化を導入したネットワーク.

とができる. それを分類器全体に拡張すると, 判別正則化 G は

$$G = \sum_k^K \gamma_k = \sum_k^K \frac{\sigma_{W_k}^2}{\sigma_{T_k}^2} \tag{3.14}$$

となる. ニューロンごとのクラス内分散 $\sigma_{W_k}^2$ と 全分散 $\sigma_{T_k}^2$ は, それぞれ

$$\sigma_{W_k}^2 = \frac{1}{N}\sum_n^N \left\{ t_{n,k}(y_{n,k} - \mu_k)^2 + (1 - t_{n,k})(y_{n,k} - \hat{\mu}_k)^2 \right\} \tag{3.15}$$

$$\sigma_{T_k}^2 = \frac{1}{N}\sum_n^N (y_{n,k} - \mu_T)^2. \tag{3.16}$$

である. ここで, N は全てのサンプルの数, $y_{n,k}$ は n 個目のサンプルから得られるロジットの k 番目の要素 (k 番目のニューロン), $t_{n,k}$ はそのサンプルの教師ベクトルの k 番目の要素（サンプルがクラス k に属するなら 1, そうでないならば 0）である. また, クラス k に属するサンプルの数を N_k, それ以外のクラス (非ターゲットクラス) に属するサンプルの数を $\hat{N}_k$ とする. そのとき, クラス k のサンプルから得られる $y_{n,k}$ のサンプル平均 μ_k と, 非ターゲットクラスのサンプルから得られる $y_{n,k}$ のサンプル平均 $\hat{\mu}_k$ は, それぞれ $\mu_k = \frac{1}{N_k}\sum_n^N t_{n,k} y_{n,k}$, $\hat{\mu}_k = \frac{1}{\hat{N}_k}\sum_n^N (1 - t_{n,k}) y_{n,k}$ と表される. $y_{n,k}$ の全サンプル平均 μ_T は $\frac{1}{N}\sum_n^N y_{n,k}$ と定義される.

提案手法の損失関数は, 経験的分類損失としてのソフトマックスクロスエントロピー (Softmax Cross Entropy) $\mathcal{L}$ と, 判別基準 G の和で与えらる.

$$E = \mathcal{L} + \lambda G \tag{3.17}$$

ここで, λ は判別正則化の効果を制御するためのハイパーパラメータである. ソフトマックスクロスエントロピーは

$$\mathcal{L} = -\frac{1}{N}\sum_{n}^{N}\sum_{k}^{K} t_{n,k} \log \frac{\exp(y_{n,k})}{\sum_{i}^{K}\exp(y_{n,i})} \tag{3.18}$$

と定義される. これらを図で表したのが, 図 3.2 である.

判別基準を持つネットワークは, この損失関数を最小化するように重み変数 $\boldsymbol{w}$ を

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\frac{\partial E}{\partial \boldsymbol{w}} \tag{3.19}$$

と更新する. α は変数の更新幅を制御する学習係数である.

3.4 ソフトマックス関数とシグモイド関数の違い

分類問題を解くために, ニューラルネットワークの分類器は, 入力サンプル $\boldsymbol{x}$ が属するクラス C_k の事後確率 $P(C_k|\boldsymbol{x})$ を推定し, 最も事後確率が高いクラスを分類結果として出力する. 3.3 章で議論したように, 事後確率の推定に使われるのは, ソフトマックス関数 (Softmax Function) が一般的である. ソフトマックス関数は, 二値分類に使われるシグモイド関数 (Sigmoid Function) を, 多クラスへ一般化したものとみなすことができるが, ソフトマックス関数よりもシグモイド関数を事後確率の推定に用いた方が高い分類精度を示す場合があることを, Beyer らは実験的に示した [71].

ここでは, ソフトマックス関数とシグモイド関数の違いを議論する. 入力サンプル $\boldsymbol{x}$ が属するクラスがクラス C_k であるとき, ネットワークは, 事後確率 $P(C_k|\boldsymbol{x})$ の値が 1 になるように変数を学習し, 逆に, クラス C_k がターゲットクラスではないとき, $P(C_k|\boldsymbol{x})$ の値が 0 となるように学習する. このとき, シグモイド関数を用いて $P(C_k|\boldsymbol{x})$ を推定すると

$$P(C_k|\boldsymbol{x}) = \frac{1}{1+\exp(-y_k)} \tag{3.20}$$

となる. y_k は, $\boldsymbol{x}$ を入力した時に得られるロジットの k 番目の要素である. この式から, シグモイド関数でも, 入力サンプルを正しく分類するために, 分類器の各ニューロンにて, ターゲットクラスか非ターゲットクラスかを分類する 2 クラス分類問題を解いていることがわかる. これは, 3.3 章で議論した, 分類器の各ニューロンが解いている問題と一致する. それに対し, ソフトマックス関数を用いて事後確率 $P(C_k|\boldsymbol{x})$ を推定すると

$$\begin{aligned} P(C_k|\boldsymbol{x}) &= \frac{\exp(y_k)}{\sum_{i}^{K}\exp(y_i)} \\ &= \frac{1}{1+\exp(-y_k)\sum_{i,i\neq k}^{K}\exp(y_i)} \end{aligned} \tag{3.21}$$

となる. この式から, ソフトマックス関数は, 他のクラスとの絡みも考慮しながら, K クラス分類問題をニューロン単位で解いていることがわかる. これは各ニューロンが解いている問題との間にギャップがあることを意味する. これが, ソフトマックス関数とシグモイド関数の違いであると同時に, シグモイド関数の方がソフトマックス関数よりも高い分類精度を示す場合がある理由でもある.

表 3.1: LeNet-5 like CNN 1.

Layer	Parameters
Conv1	64 filters, 3×3, pad = 0, stride=2, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Pool1	Max-pooling, 3×3, pad = 0
Conv2	128 filters, 3×3, pad = 0, stride=2, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Pool2	Max-Pooling, 2×2, pad = 0
FC1	10 fully-connected filters
output	Softmax

表 3.2: LeNet-5 like CNN 2.

Layer	Parameters
Conv1	64 filters, 5×5, pad = 2, stride=1, Batch Norm, ReLU
Pool1	Max-pooling, 5×5, pad = 0
Conv2	128 filters, 3×3, pad = 2, stride=1, Batch Norm, ReLU
Pool2	Max-Pooling, 2×2, pad = 0
FC1	1024 fully-connected filters, ReLU
FC2	10 fully-connected filters
output	Softmax

3.5 判別正則化の有効性の検証

本節では, 有名なデータセットである MNIST [60], CIFAR-10/100 [61], およびいくつかの畳み込みニューラルネットワーク (CNN) を用いて, 判別正則化の有効性を評価する. MNIST は, 0 クラスの白黒画像からなる, 手書き文字認識用のデータセットである. それに対し, CIFAR-10 と CIFAR-100 は物体分類用のデータセットで, それぞれ 10 クラス分, 100 クラス分の RGB カラー画像が含まれている. 各データセットの詳細は, 2.5 章に記載してある.

提案する判別正則化の有効性を検証するため, 5 種類の CNN を実験に用いる. そのうちの 2 種類 (表 3.1, 表 3.2) は, LeNet5 [2] を模した層の浅い CNN で, MNIST をデータセットとして画像分類を行う. 一つ目の LeNet-5 like 1 (表 3.1) は, 中間層として畳み込み層と Pooling 層を 2 層づつ持ち, 二つ目の LeNet-5 like 2 (表 3.2) は, さらに FC 層が中間層に追加されている. 各畳み込み層には, Batch Normalization (Batch Norm) [11] を導入しいる . 活性化関数は ReLU [48] を用いる. 最適化手法として, LeNet-5 like 1 は Adam [72] を, LeNet-5 like 2 は確率的勾配降下法 (SGD) を用いる. Adam の各種変数は, それぞれ $\alpha = 0.001$, $\beta 1 = 0.9$, $\beta 2 = 0.999$, $eps = 10^{-8}$ とする. SGD の学習係数は, 実験毎にその都度表記する. 損失関数 (式 (3.17)) の経験的分類損失 $\mathcal{L}$ には, ソフトマックスクロスエントロピーを用いる. ミニバッチサイズはどちらも 100 である. それに対し, 次の 2 種類の CNN (表 3.3, 3.4) は, CIFAR-10/100 をデータセットとして画像分類を行う. 一つ目の Deep CNN (表 3.3) は, 畳み込み層が 9 層, Pooling 層が 3 層, 分類器が 1 層で構成され, 2 つ目の VGG-13 like (表 3.4) は, 6 層の畳み込み層, 1 層の FC 層と分類器で構成される. 活性化関数は, Deep CNN では ReLU 関数の亜種の一つである Leaky ReLU [73] を, VGG-13 llike では ReLU を用いる. 最適化

表 3.3: Deep CNN.

Layer	Parameters
Conv1	128 filters, 3×3, pad $= 1$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Conv2	128 filters, 3×3, pad $= 1$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Conv3	128 filters, 3×3, pad $= 1$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Pool1	Max-pooling, 2×2, pad $= 0$, DropOut ($p = 0.5$)
Conv4	256 filters, 3×3, pad $= 1$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Conv5	256 filters, 3×3, pad $= 1$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Conv6	256 filters, 3×3, pad $= 1$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Pool2	Max-pooling, 2×2, pad $= 0$, DropOut ($p = 0.5$)
Conv5	512 filters, 3×3, pad $= 0$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Conv6	256 filters, 3×3, pad $= 0$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Conv6	128 filters, 3×3, pad $= 0$, Batch Norm, Leaky ReLU ($\alpha = 0.1$)
Pool3	Average-Pooling, 2×2, pad $= 0$
FC2	10 or 100 fully-connected filters
output	Softmax

表 3.4: VGG-13 like.

Layer	Parameters
Conv1	64 filters, 3×3, pad $= 1$, Batch Norm, ReLU
Conv2	64 filters, 3×3, pad $= 1$, Batch Norm, ReLU
Pool1	Max-pooling, 2×2, pad $= 0$
Conv3	128 filters, 3×3, pad $= 1$, Batch Norm, ReLU
Conv4	128 filters, 3×3, pad $= 1$, Batch Norm, ReLU
Pool2	Max-pooling, 2×2, pad $= 0$
Conv5	256 filters, 3×3, pad $= 1$, Batch Norm, ReLU
Conv6	256 filters, 3×3, pad $= 1$, Batch Norm, ReLU
Pool3	Max-pooling, 2×2, pad $= 0$
FC1	1024 fully-connected filters, DropOut ($p = 0.5$)
FC2	10 or 100 fully-connected filters
output	Softmax

手法として, Deep CNN は Adam を, VGG-13 like は SGD を用いる. 最適化手法の各種変数, 損失関数, およびミニバッチサイズは, LeNet-5 like の場合と同様である. さらに, ResNet-18 を模した ResNet-18 like (表 3.5) も用いて, 比較実験を行う. この時のデータセット, 最適化手法, ミニバッチサイズは, どれも VGG-13 like と同様である. そのときの学習係数は, 0.1 を初期値とし, ネットワークを 80 epochs, 120 epoch 学習した時に, その都度, 学習係数を 10 で割る.

CNN の各種変数の初期値を 3 回変えてネットワークの分類精度を評価し, その平均と標準偏差を実験結果

表 3.5: ResNet-18 like. ( ) × 3 は, 積み重ねた Residual Block の数を表す. 各 Residual Block は, ショートカット層として 1 × 1 畳み込み層を有する. ダウンサンプリングは, 第 l の層である conv2_1 と conv3_1 のストライドを 2 にすることで行われる.

Layer	Parameters
Conv0	16 filters, 3 × 3, pad=1, BatchNorm, ReLU
Conv1_x	$\begin{pmatrix} \text{16 filters, } 3\times3\text{, pad=1, BatchNorm, ReLU} \\ \text{16 filters, } 3\times3\text{, pad=1, BatchNorm, ReLU} \end{pmatrix}\times3$
Conv2_x	$\begin{pmatrix} \text{32 filters, } 3\times3\text{, pad=1, BatchNorm, ReLU} \\ \text{32 filters, } 3\times3\text{, pad=1, BatchNorm, ReLU} \end{pmatrix}\times3$
Conv3_x	$\begin{pmatrix} \text{64 filters, } 3\times3\text{, pad=1, BatchNorm, ReLU} \\ \text{64 filters, } 3\times3\text{, pad=1, BatchNorm, ReLU} \end{pmatrix}\times3$
Pool	Average-pooling, 8 × 8, pad = 0
FC	10 or 100 fully-connected filters
output	Softmax

として報告する.

3.5.1 判別正則化を導入した CNN の分類精度

ここでは, 判別正則化を導入した CNN と, 導入しなかった CNN の分類精度を比較することで, 判別正則化の有効性を検証する. 各データセット・ネットワークにおける実験結果を, 表 3.6 に示した. 表中の"-"の行は, 判別正則化なしの Baseline, λ は判別正則化の係数である. MNIST を用いた実験では, 判別正則化を導入することで, LeNet-5 like 2 (表 3.2) では分類精度が 0.03 % 向上し, LeNet-5 like 1 (表 3.2) では, 正則化なしのものとほぼ同等の性能だった. CIFAR-10 を用いた実験でも, 判別正則化を Deep CNN, VGG-13 like, ResNet-18 like に導入することで, それぞれ約 2 %, 約 0.1 %, 約 0.1 % 向上した. CIFAR-100 では判別正則化の効果がより顕著で, Deep CNN, VGG-13 like, ResNet-18 like にて, それぞれ約 5 %, 約 0.2 %, 約 3 % 向上した. CIFAR-100 に対する効果が最も大きい理由は，MNIST と CIFAR-10 の分類精度は 100 % に近いので改善の余地があまり無いないのに対し，CIFAR-100 の分類精度は 100 % には程遠く，改善の余地が大きいからである. また, SGD を最適化手法として用いる VGG-13 like よりも, Adam を用いる Deep CNN のほうが判別正則化による分類精度改善が著しいことから, 判別正則化と Adam の相性が良いかもしれないことを示唆している.

図 3.3 の学習曲線は，学習途中の Deep CNN の分類精度の推移を示している. 入力データは CIFAR-10 のテストサンプルである. このグラフから, 学習速度判別正則化を用いることで, CNN の学習速度はさほど変わらないにも関わらず, 分類精度が向上することがわかる.

3.5.2 ロジットの分布

判別正則化の効果をさらに調べるために, 学習した CNN の分類器の各ニューロンの入力 (ロジット) の分布 (ヒストグラム) を可視化し, それらを比較する. 図 3.4 と図 3.5 は, それぞれ, Deep CNN に CIFAR-10 の学習サンプル, もしくはテストサンプルを入力した時に, 車クラスに対応するニューロンから得られるロジットのヒストグラムである. それぞれの図中の上側のヒストグラムは, 判別正則化なしで学習した Deep CNN から

表 3.6: 分類精度 (テスト画像, %). λ は判別正則化の係数. ”-”の行は判別正則化なしの結果.

データセット	ネットワーク	λ	学習係数	epochs	分類精度
MNIST	LeNet-5 like 1 (表 3.1)	-	0.001	300	**98.52** ±0.03
		1.0	0.001	300	98.51 ±0.02
	LeNet-5 like 2 (表 3.2)	-	0.01	300	99.29 ±0.06
		1.0	0.01	300	**99.32** ±0.01
CIFAR-10	Deep CNN (表 3.3)	-	0.01	1,000	90.20 ±0.27
		1.0	0.01	1,000	**92.42** ±0.20
	VGG-13 like (表 3.4)	-	0.01	1,000	89.42 ±0.11
		0.1	0.01	1,000	**89.56** ±0.12
	ResNet18 (表 3.5)	-	0.1	1,000	89.67 ±0.10
		0.1	0.1	1,000	**89.75** ±0.20
CIFAR-100	Deep CNN (表 3.3)	-	0.01	1,000	64.07 ±0.34
		1.0	0.01	1,000	**69.22** ±0.19
	VGG-13 like (表 3.4)	-	0.01	1,000	66.61 ±0.07
		1.0	0.01	1,000	**66.83** ±0.30
	ResNet18 (表 3.5)	-	0.1	1,000	58.56 ±0.23
		0.1	0.1	1,000	**61.96** ±0.52

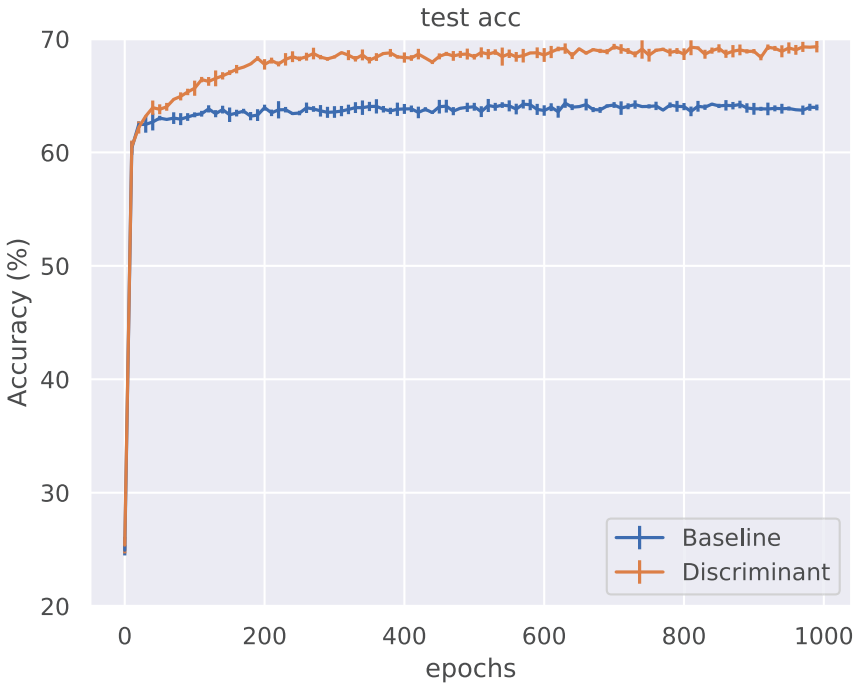


図 3.3: CIFAR-100 の学習曲線. 横軸は epoch 数, 縦軸はテストサンプルの分類精度. 青線は判別正則化なし, オレンジ線は判別正則化ありの場合の学習曲線.

得られたものであるのに対し, 下側のヒストグラムは, 判別基準を持つ Deep CNN から得られたものである. ヒストグラム中の赤い分布は, ターゲットクラスの出現数を示し, 青い分布は非ターゲットクラス (untarget class) の出現数を示している.

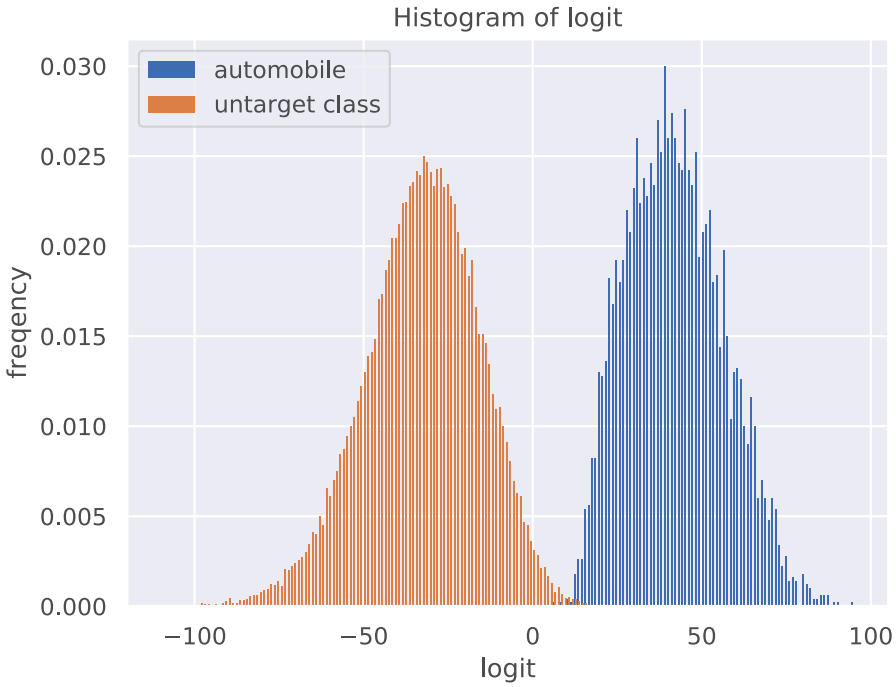
これらのヒストグラムから, CNN は, 入力画像を, ターゲットクラスと非ターゲットクラスの 2 つのガウス分布に似た分布に変換していることがわかる. また, 2 つの分布は, それぞれの平均は大きく異なるのに対し, 分散に関しては, あまり大きな違いはない. これらの結果は, 3.3 章で議論した, 分類器の各ニューロンが理想とする特徴に近いものが, 実際に生成されていることを示している. さらに, 損失関数に判別正則化を導入することで, 2 つの分布の重複部分が少なくなり, より分離されていることもわかる. そのことを, 分類器の各ニューロンの入力から得られる判別基準の値 (2 つの分布の分離度合い) を用いて, 定量的に示したのが表 3.7, 3.8 である. この表から, 判別正則化を導入することで, 各ニューロンの判別基準の値が大きくなった (2 つの分布がより大きくなった). 特に, CIFAR-10 をデータセットとした場合や, Deep CNN を用いた場合は, 顕著だった. これらのことからも, CNN に判別正則化を導入することで, 各ニューロンに入力される 2 つの分布がより大きく分離されていることがわかる.

表 3.7: 各ニューロンの判別基準の値 (テストサンプル). クラス 0 からクラス 4 までの結果を抜粋. 標準偏差は省略.

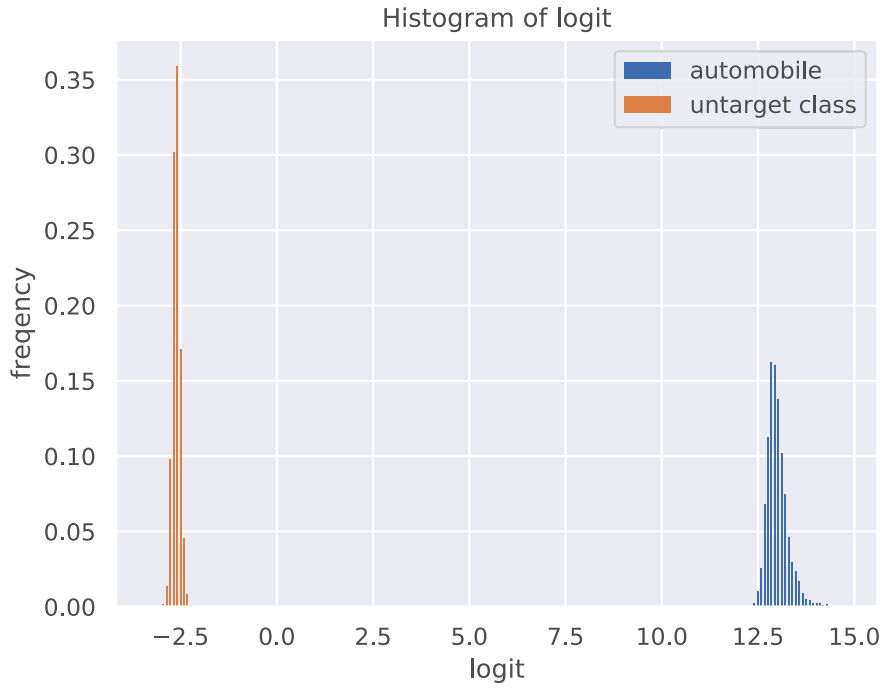
			クラス				
データセット	ネットワーク	λ	0	1	2	3	4
MNIST	LeNet-5 like 1 (表 3.1)	-	0.70	0.64	0.68	0.59	0.58
		1.0	**0.90**	**0.91**	**0.85**	**0.86**	**0.87**
	LeNet-5 like 2 (表 3.2)	-	0.72	0.71	0.70	0.70	0.64
		1.0	**0.96**	**0.97**	**0.94**	**0.95**	**0.95**
CIFAR-10	Deep CNN (表 3.3)	-	0..42	0.57	0.34	0.29	0.36
		1.0	**0.88**	**0.94**	**0.83**	**0.73**	**0.87**
	VGG-13 like (表 3.4)	-	0.42	0.62	0.37	0.35	0.35
		0.1	**0.73**	**0.82**	**0.66**	**0.57**	**0.71**
	ResNet-18 (表 3.5)	-	0.60	0.71	0.54	0.49	0.60
		0.1	**0.76**	**0.86**	**0.68**	**0.59**	**0.74**
CIFAR-100	Deep CNN (表 3.3)	-	0.10	0.08	0.06	0.04	0.04
		1.0	**0.50**	**0.46**	**0.31**	**0.26**	**0.31**
	VGG-13 like (表 3.4)	-	0.08	0.14	0.10	0.04	0.17
		1.0	**0.09**	**0.16**	0.10	0.04	**0.19**
	ResNet-18 (表 3.5)	-	0.07	0.11	0.08	0.04	0.13
		0.1	**0.1**	**0.12**	**0.09**	**0.05**	**0.15**

表 3.8: 各ニューロンの判別基準の値 (テストサンプル). クラス 5 からクラス 9 までの結果を抜粋. 標準偏差は省略.

			クラス				
データセット	ネットワーク	λ	5	6	7	8	9
MNIST	LeNet-5 like 1 (表 3.1)	-	0.61	0.58	0.53	0.66	0.55
		1.0	**0.85**	**0.88**	**0.84**	**0.83**	**0.83**
	LeNet-5 like 2 (表 3.2)	-	0.69	0.67	0.59	0.75	0.62
		1.0	**0.94**	**0.95**	**0.94**	**0.93**	**0.92**
CIFAR-10	Deep CNN (表 3.3)	-	0.33	0.42	0.47	0.50	0.51
		1.0	**0.79**	**0.91**	**0.92**	**0.92**	**0.92**
	VGG-13 like (表 3.4)	-	0.36	0.60	0.56	0.53	0.53
		0.1	**0.64**	**0.79**	**0.76**	**0.81**	**0.78**
	ResNet-18 (表 3.5)	-	0.56	0.68	0.70	0.69	0.70
		0.1	**0.68**	**0.80**	**0.80**	**0.83**	**0.83**
CIFAR-100	Deep CNN (表 3.3)	-	0.06	0.07	0.07	0.11	0.09
		1.0	**0.38**	**0.49**	**0.38**	**0.48**	**0.47**
	VGG-13 like (表 3.4)	-	0.11	0.09	0.08	0.06	0.08
		1.0	**0.12**	**0.10**	**0.09**	**0.07**	**0.09**
	ResNet-18 (表 3.5)	-	0.09	**0.08**	0.08	0.06	0.07
		0.1	**0.09**	**0.09**	**0.10**	**0.07**	**0.09**

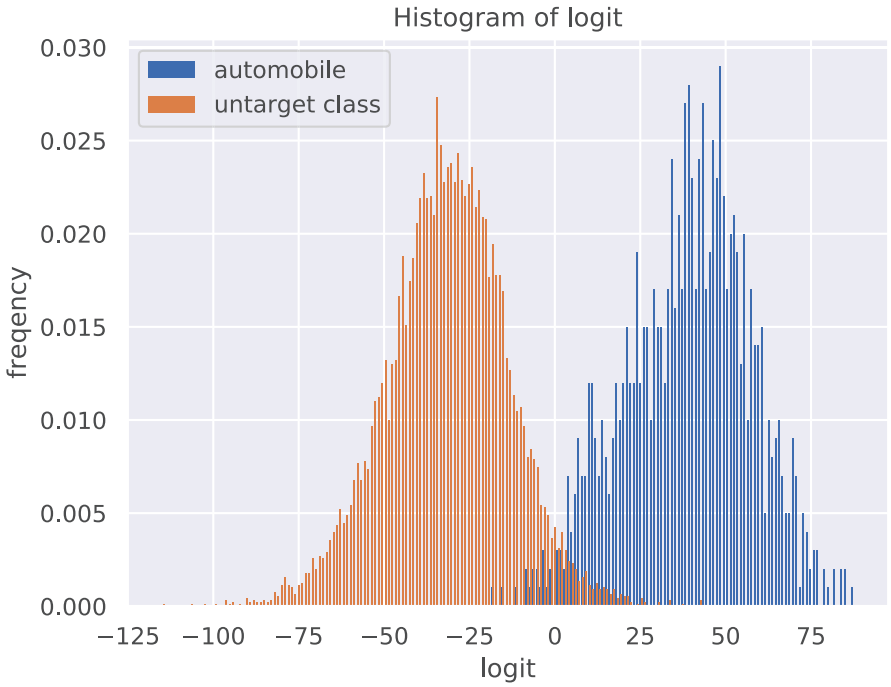


(a) 判別正則化なし

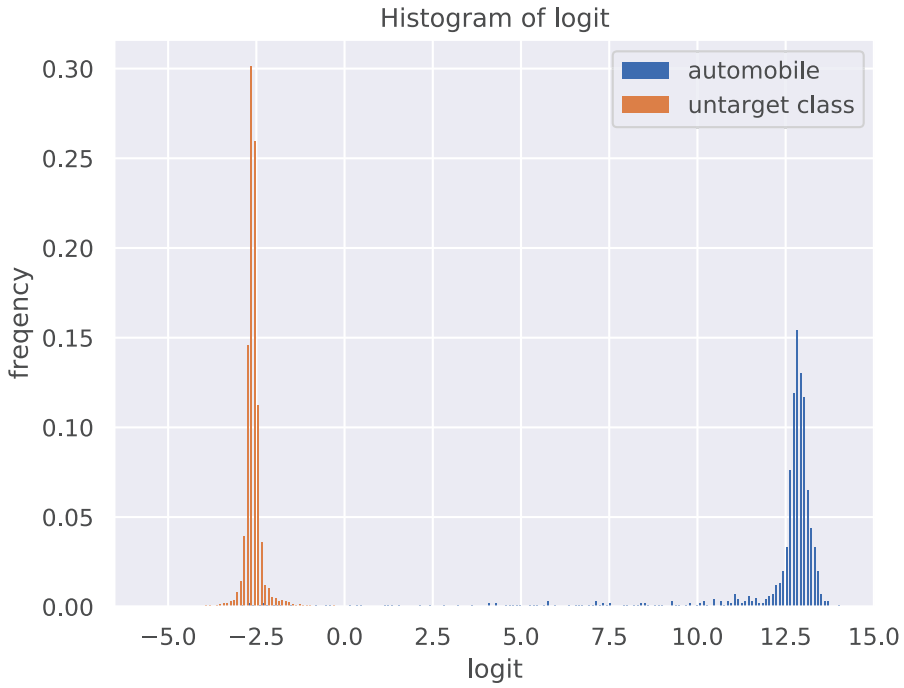


(b) 判別正則化あり

図 3.4: CIFAR-10 の学習サンプルを Deep CNN に入力し, 車クラスを分類するニューロンから得られたロジットのヒストグラム. 横軸がロジットの値, 縦軸がその値の出現数. 赤い分布は車クラスのサンプルから得られた分布を示し, 青い分布はそれ以外のクラスから得られた分布を示す.



(a) 判別正則化なし



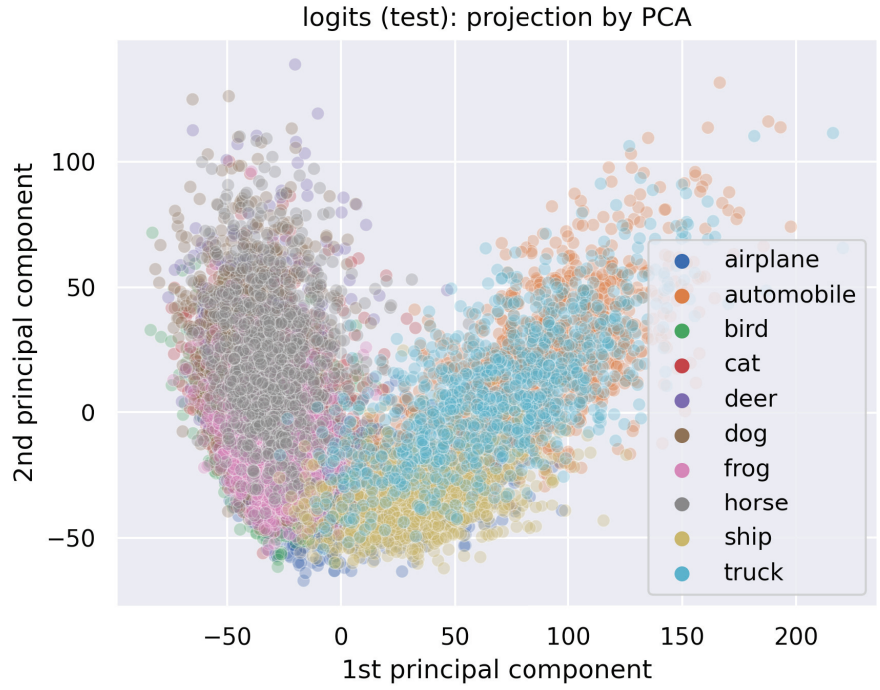
(b) 判別正則化あり

図 3.5: CIFAR-10 のテストサンプルを Deep CNN に入力し, 車クラスを分類するニューロンから得られたロジットのヒストグラム. 赤い分布は車クラスのサンプルから得られた分布を示し, 青い分布はそれ以外のクラスから得られた分布を示す.

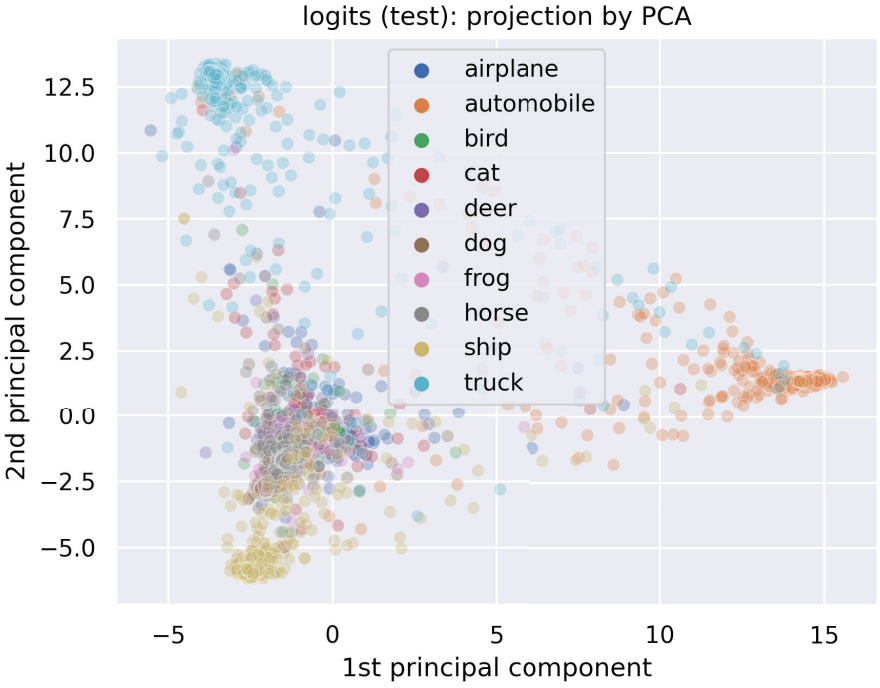
3.5.3 ロジットの可視化

つづいて, 分類器に入力される特徴ベクトル (ロジットのベクトル) に対して主成分分析 (PCA) を適用し, それらを可視化する. 可視化は, 学習サンプルから得られるロジットのベクトルを用いて 2 次元の主成分空間を作り, テストサンプルから得られるロジットのベクトルをその空間上に射影することで行なう.

図 3.6 は, CIFAR-10 を入力した Deep CNN から得られたロジットのベクトルを可視化したものである. 各図のうち, 上側の図は判別正則化を導入しなかった場合の特徴ベクトルであるのに対し, 下側の図は判別正則化を導入したものである. これらの図から, 判別正則化を用いることで, 特徴ベクトルを射影した低次元上でも, 各クラスがさらに分離していることがわかる.



(a) 判別正則化なし



(b) 判別正則化あり

図 3.6: Deep CNN に CIFAR-10 を入力したときのロジット．横軸が第一主成分, 縦軸が第二主成分である．色は, 各ベクトルが属するクラス．

3.6 ソフトマックス関数とシグモイド関数の比較

これまでは, 分類器の活性化関数としてソフトマックス関数を用いることを前提としてきたが, ここでは, ソフトマックス関数を用いた場合と, シグモイド関数を用いた場合を比較することで, 3.4 章での議論が正しいことを実験的に示す. 分類器の活性化関数を置き換えることと, 最適化手法を Adam にした以外は, 3.5 章と同様の条件で比較実験を行い, 分類精度, ロジットの分布, ロジットの低次元ベクトルを比較する.

3.6.1 分類精度を比較

ここでは, 3.5.1 章と同様の条件で各ネットワークの分類精度を求め, ソフトマックス関数とシグモイド関数を比較する. 表 3.9 は, 学習した各ネットワークにテスト画像を入力した結果である. 多くのケースにて, シグモイド関数を用いると分類精度が向上することがわかる. 特に, CIFAR100 をデータセットとした時はそれが顕著だった.

これらの結果から, ソフトマックスの代わりにシグモイドを分類器の活性化関数として用いることで, 分類精度が改善する場合があることがわかった.

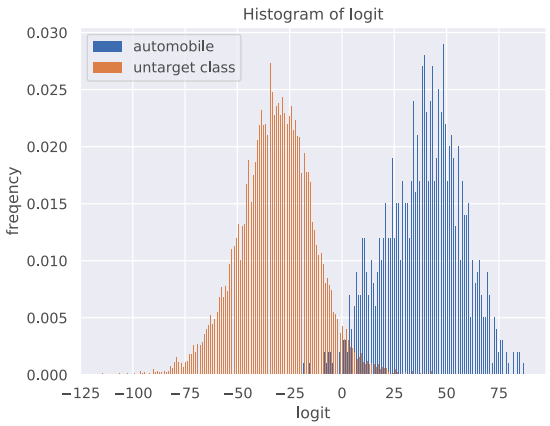
表 3.9: 分類器の活性化関数にソフトマックス関数, もしくはシグモイド関数を持つ CNN にテスト画像を入力した時の分類精度 (%). λ は判別正則化の係数. $\cdot/\cdot$ は, それぞれ, ソフトマックス関数, もしくはシグモイド関数を導入したときの係数. 最適化手法は全て Adam.

データセット	ネットワーク	λ	epochs	ソフトマックス	シグモイド
MNIST	LeNet-5 like 1 (表 3.1)	-	300	98.94 ±0.12	**98.96** ±0.08
		0.1 / 0.01	300	**99.00** ±0.04	98.99 ±0.04
	LeNet-5 like 2 (表 3.2)	-	300	99.34 ±0.08	**99.41** ±0.06
		1.0	300	**99.38** ±0.04	99.30 ±0.05
CIFAR-10	Deep CNN (表 3.3)	-	1,000	90.20 ±0.27	**91.38** ±0.03
		1.0	1,000	**92.63** ±0.16	92.28 ±0.04
	VGG-13 like (表 3.4)	-	1,000	**90.68** ±0.35	90.53 ±0.33
		0.01	1,000	90.51 ±0.22	**90.65** ±0.12
	ResNet-18 like (表 3.5)	-	1,000	**87.53** ±0.63	85.84 ±1.04
		0.1	1,000	**87.41** ±0.18	87.26 ±0.60
CIFAR-100	Deep CNN (表 3.3)	-	1,000	64.07 ±0.34	**66.47** ±0.28
		0.1	1,000	64.87 ±0.05	**70.59** ±0.35
	VGG-13 like (表 3.4)	-	1,000	66.26 ±0.58	**67.42** ±0.08
		1 / 0.01	1,000	67.19 ±0.23	**67.44** ±0.52
	ResNet-18 like (表 3.5)	-	1,000	66.26 ±0.58	**67.42** ±0.08
		1 / 0.01	1,000	67.19 ±0.23	**67.44** ±0.52

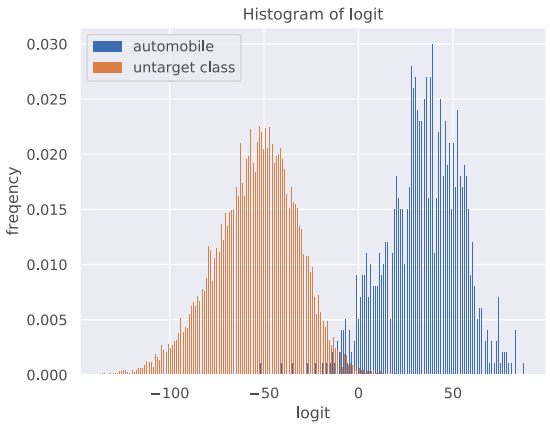
3.6.2　ロジットの分布

ここでは, 各ネットワークから得られるロジットの分布を比較する.

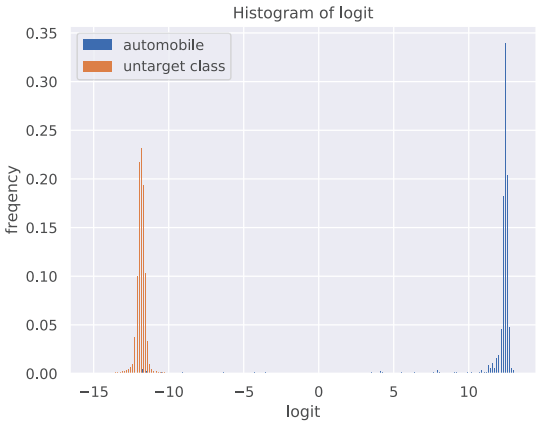
図 3.7 は CIFAR10 のテストサンプルを Deep CNN に入力して得られたヒストグラムである. 上から順に, ソフトマックス関数, シグモイド関数, シグモイド関数と判別正則化を組み合わせた結果である. より詳細に比較した図は, 本章末尾に掲載した (図 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15, 3.16, 3.17). それ以外は, 付録 A 章に掲載した. 3 つのヒストグラムを比較すると, 判別正則化なしの条件では, ソフトマックス関数よりもシグモイド関数の方が, 分布の重なりが小さくなっている. そのうえで判別正則化を適用すると, シグモイド関数であっても, 分布の重なりが小さくなった. これらの実験結果から, 判別正則化は活性化関数に関わらず分布の重なりを小さくすることに役立つことや, 分類器の活性化関数として, ソフトマックス関数よりもシグモイド関数の方が好ましい場合があることがわかった.



(a) ソフトマックス関数 (判別正則化なし)



(b) シグモイド関数 (判別正則化なし)



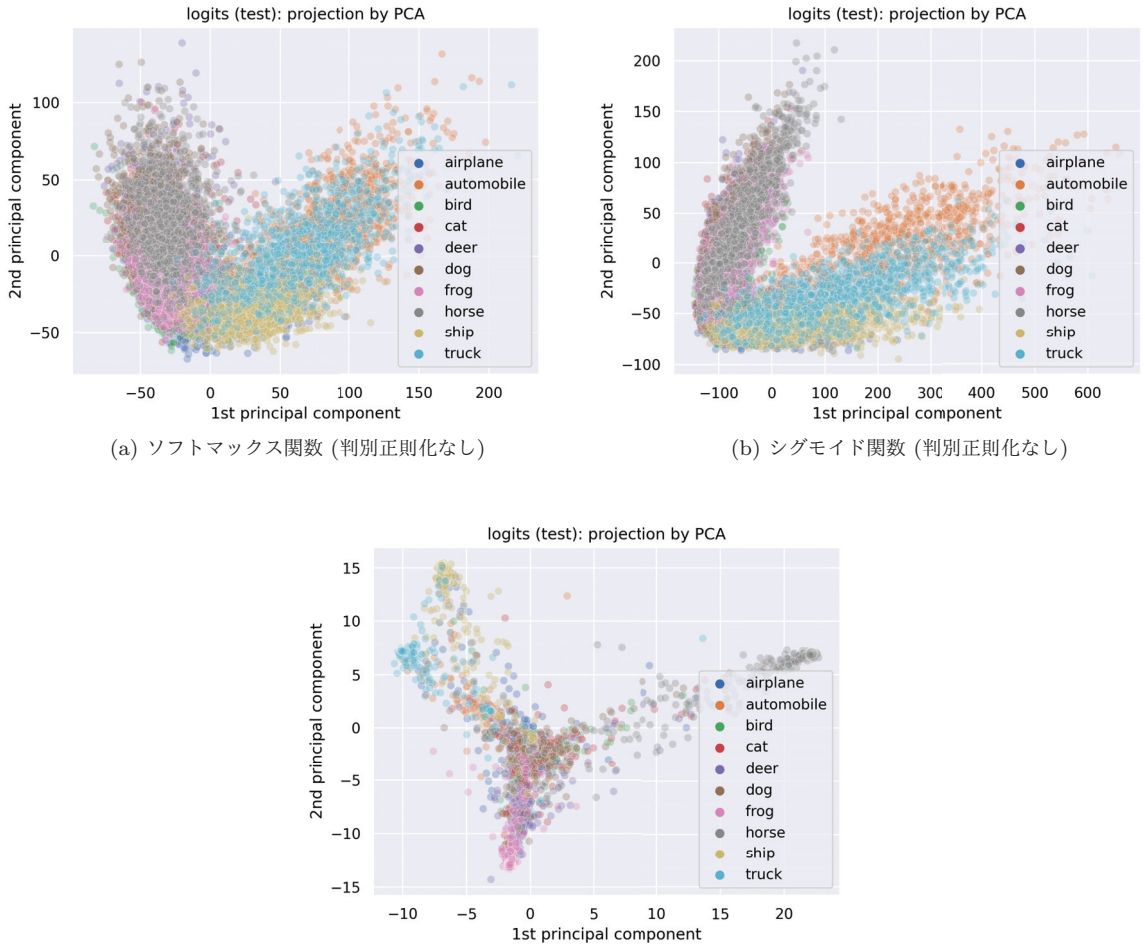
(c) シグモイド関数 (判別正則化あり)

図 3.7: Deep CNN に CIFAR-10 のテストサンプルを入力して得られたヒストグラム.

3.6.3　ロジットを可視化

最後に, ここでも 3.5.3 章と同様に, ロジットのベクトルに対して PCA を適用し, それらを可視化する. 図 3.8 は, CIFAR-10 のテスト画像を Deep CNN に入力して得られたロジットのベクトルを可視化したものである.

ここでは, ソフトマックス関数とシグモイド関数の間にあまり大きな違いは見られなかった. それでも, やはり判別正則化を導入することで, より判別的な特徴が得られた.



(a) ソフトマックス関数 (判別正則化なし)

(b) シグモイド関数 (判別正則化なし)

(c) シグモイド関数 (判別正則化あり)

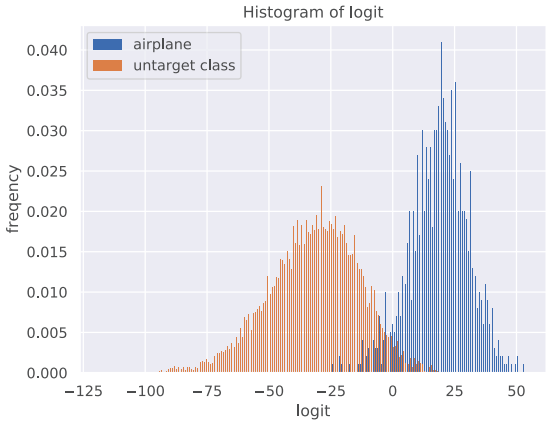
図 3.8: Deep CNN に CIFAR-10 のテストサンプルを入力して得られたロジットのベクトルを可視化.

3.7 本章のまとめ

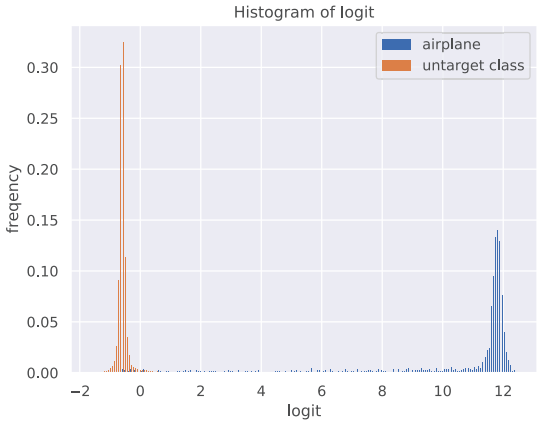
本章では，ニューラルネットワークの分類器を構成する各ニューロンにとって理想的な特徴について議論し, その特徴の生成を加速する判別正則化を提案した．分類器の活性化関数がソフトマックス関数であるとき, 分類器の各ニューロンはそれぞれ 2 クラス分類問題を解いているので, 各ニューロンにとって理想的な特徴は, クラスごとに平均が異なり, かつ全てのクラスで同じ分散を持つガウス分布であることを導出できた．また, ネットワークがそのような分布を実際に生成していること, それらの分布の一部が重なっているせいでネットワークの分類精度が低下していることを, 実験的に確認した．本章で提案した判別正則化は, 各分布のそのような重なりを小さくすることができたので, その結果として分類精度を向上させることができた．

また, 分類器の活性関数にシグモイド関数を用いることで, ソフトマックス関数よりも高い分類精度を示す場合があることも確認した．

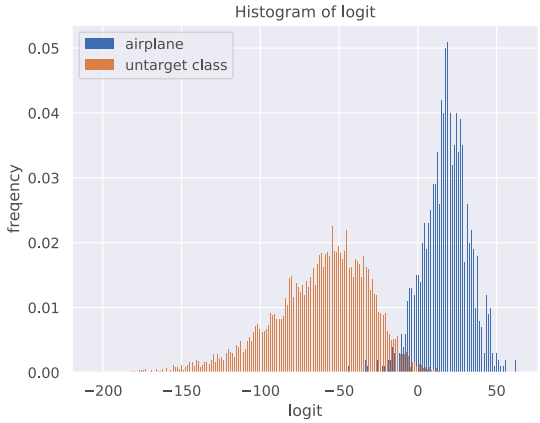
これらの結果は, ニューラルネットワークの出力層 (分類器) を改善する重要性を示しただけではなく, 未だ本質的な部分の多くが未知である CNN を理解するのにも役立つと考えられる．



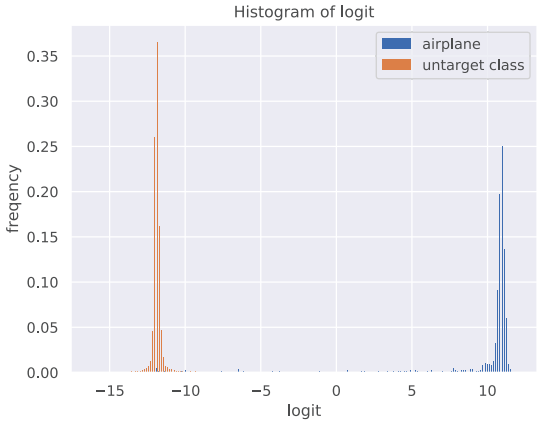
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

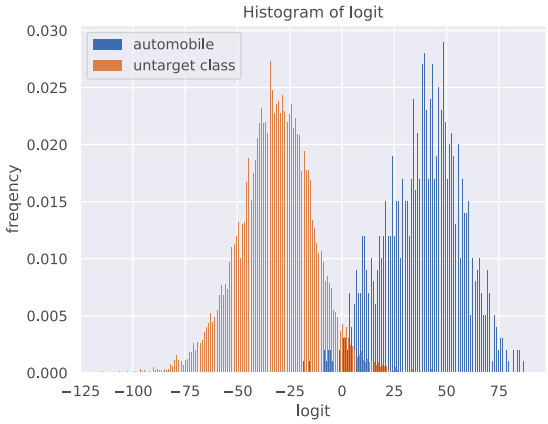


(c) シグモイド関数 (判別正則化なし)

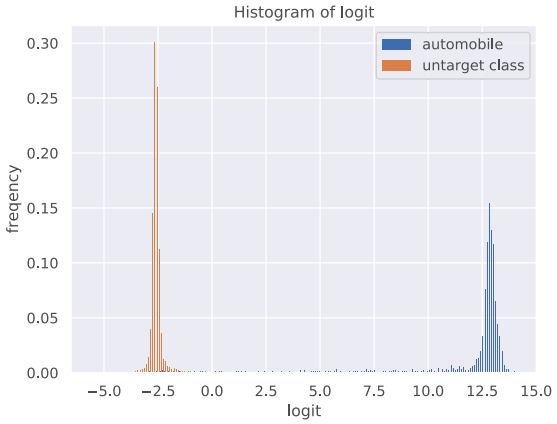


(d) シグモイド関数 (判別正則化あり)

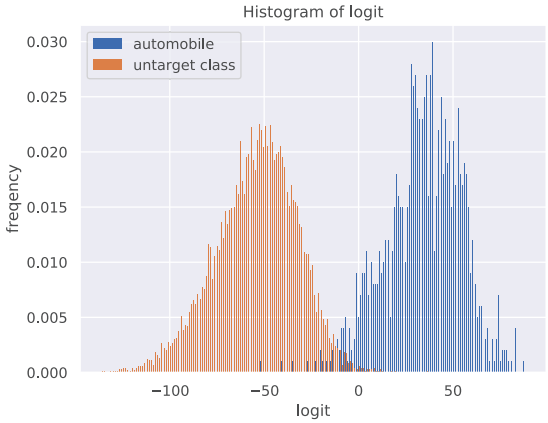
図 3.9: 飛行機



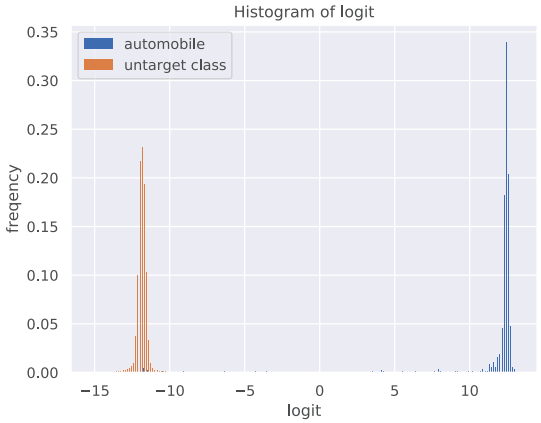
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

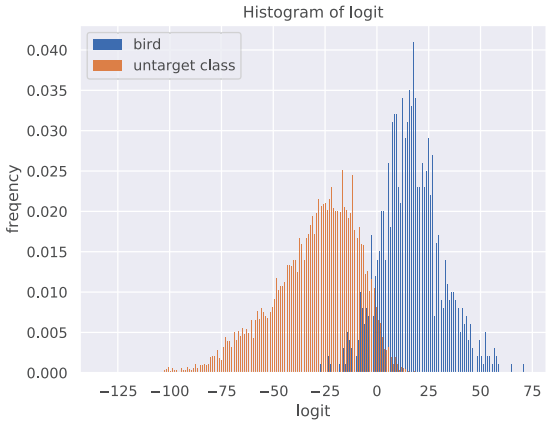


(c) シグモイド関数 (判別正則化なし)

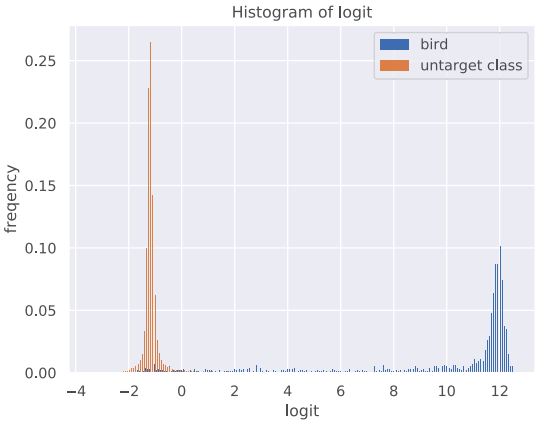


(d) シグモイド関数 (判別正則化あり)

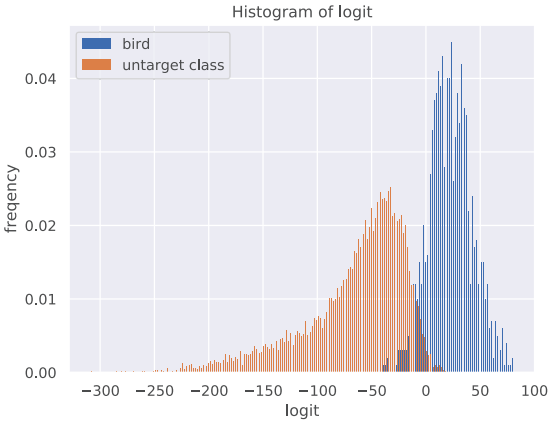
図 3.10: 自動車



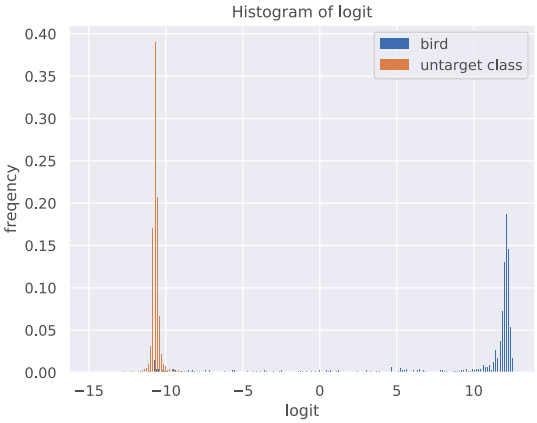
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

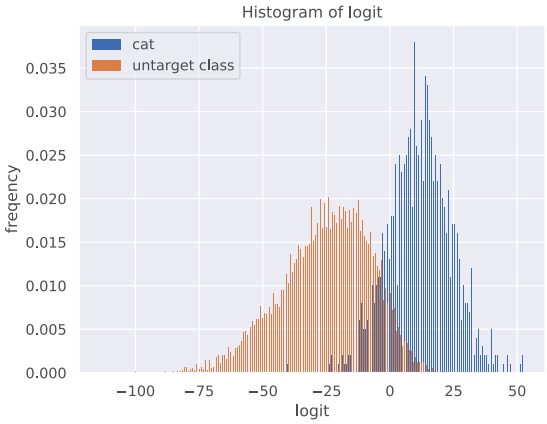


(c) シグモイド関数 (判別正則化なし)

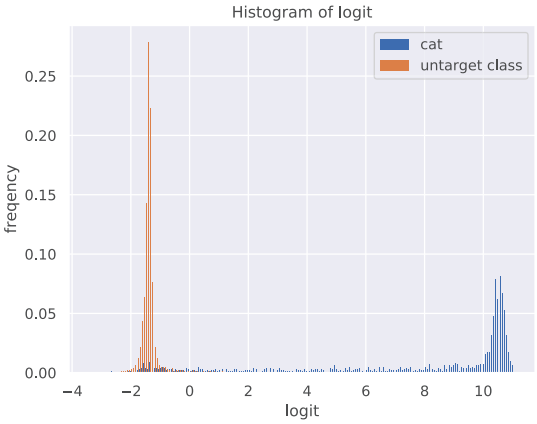


(d) シグモイド関数 (判別正則化あり)

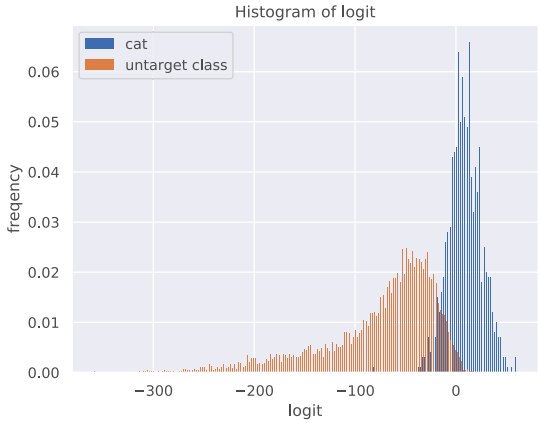
図 3.11: 鳥



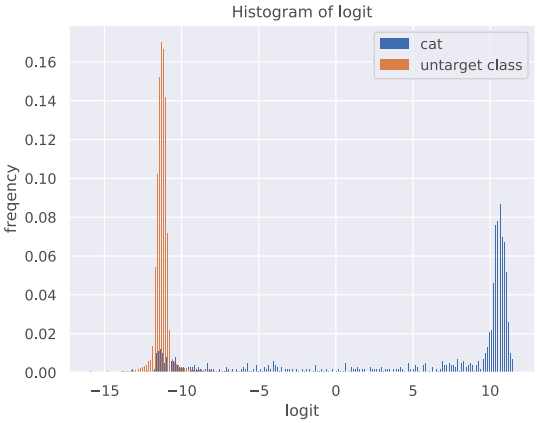
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

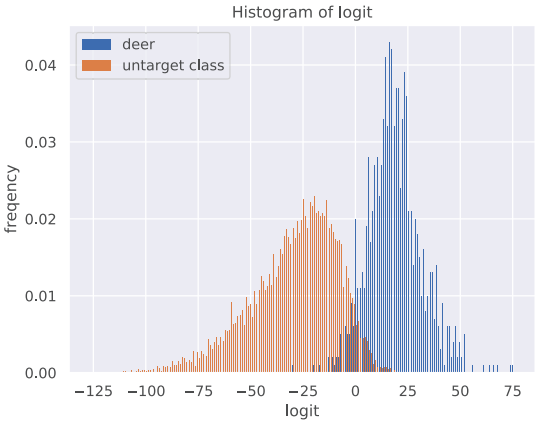


(c) シグモイド関数 (判別正則化なし)

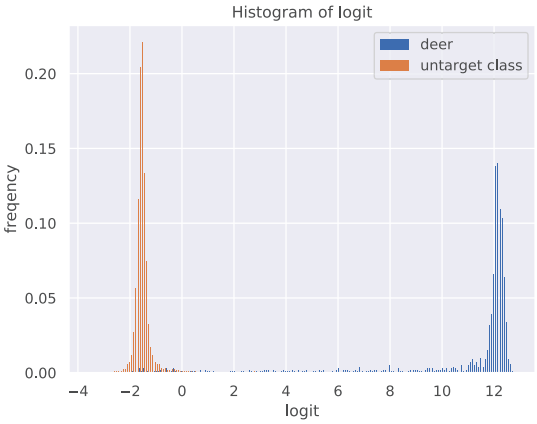


(d) シグモイド関数 (判別正則化あり)

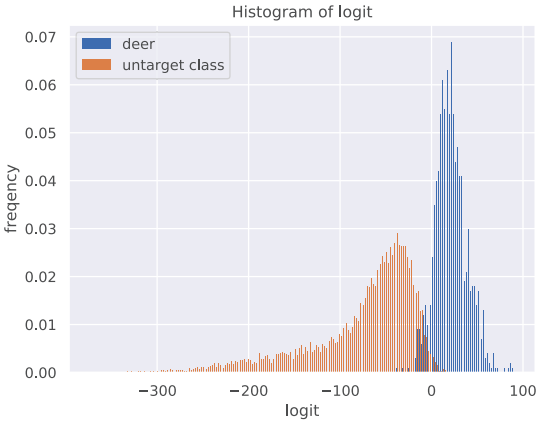
図 3.12: 猫



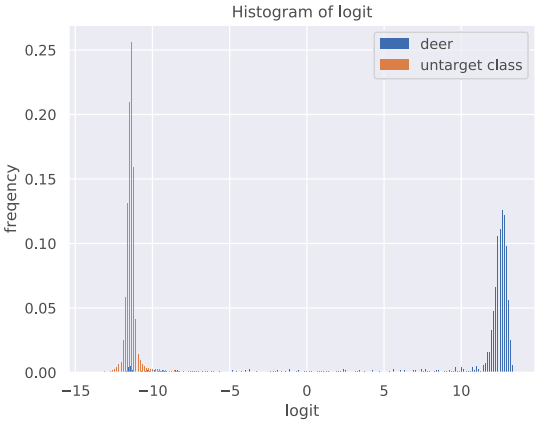
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

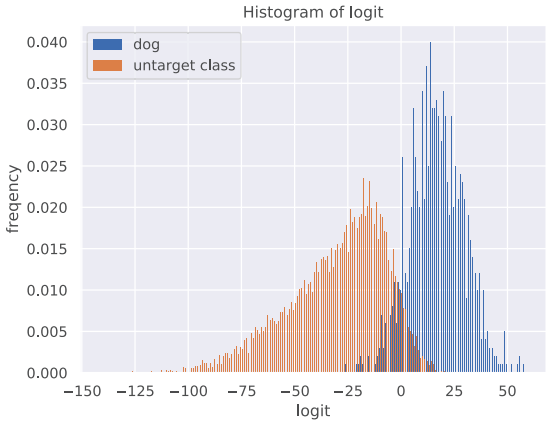


(c) シグモイド関数 (判別正則化なし)

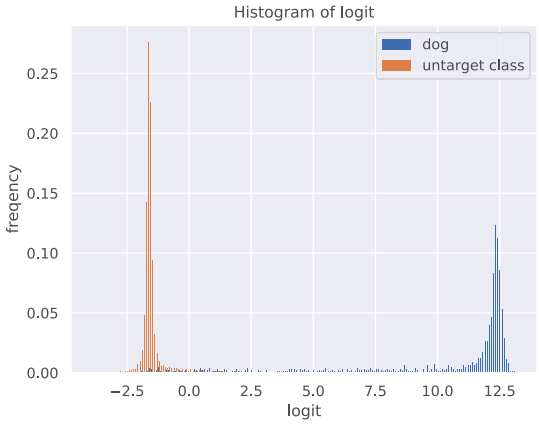


(d) シグモイド関数 (判別正則化あり)

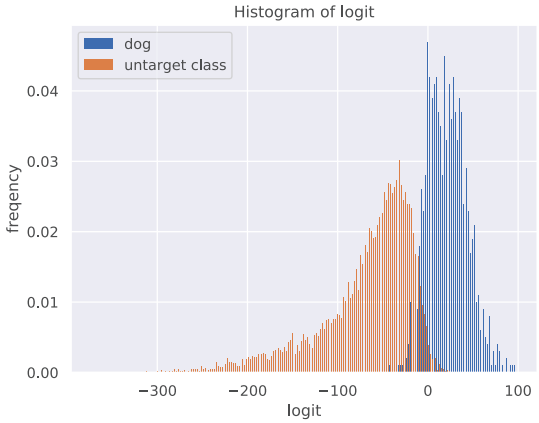
図 3.13: 鹿



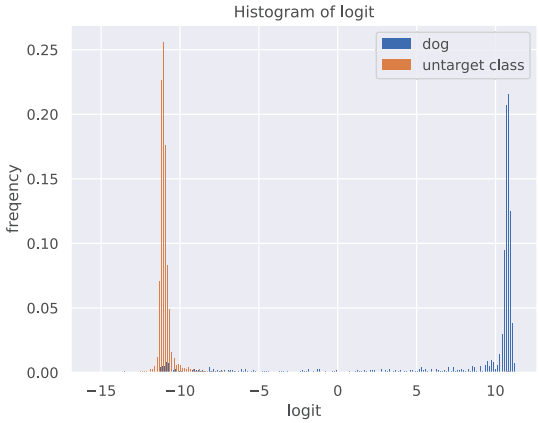
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

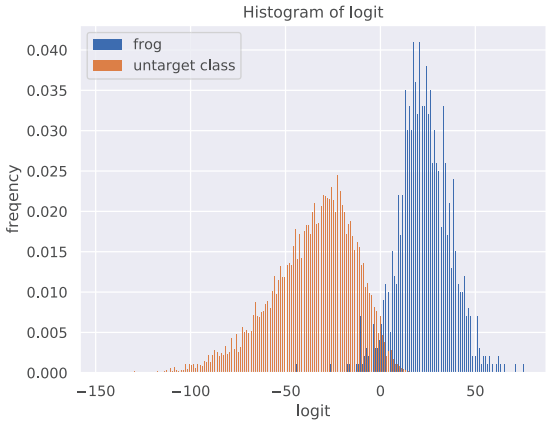


(c) シグモイド関数 (判別正則化なし)

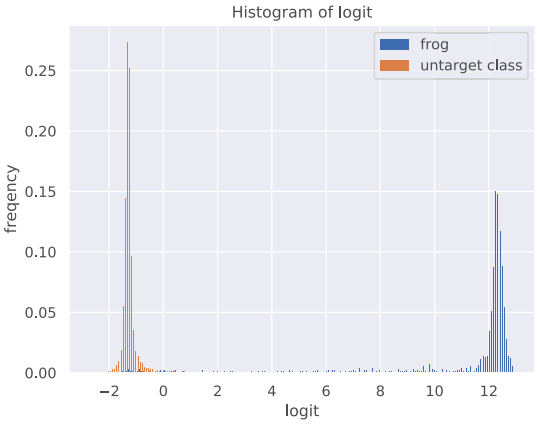


(d) シグモイド関数 (判別正則化あり)

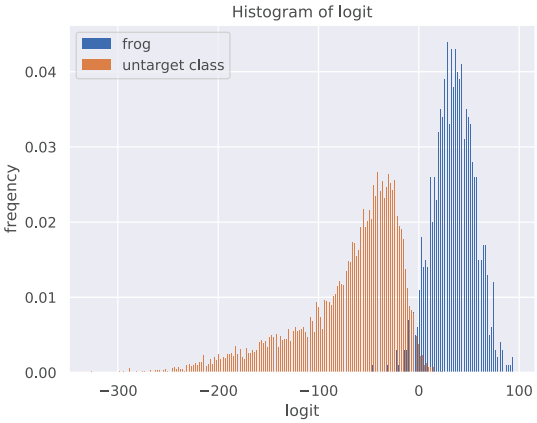
図 3.14: 犬



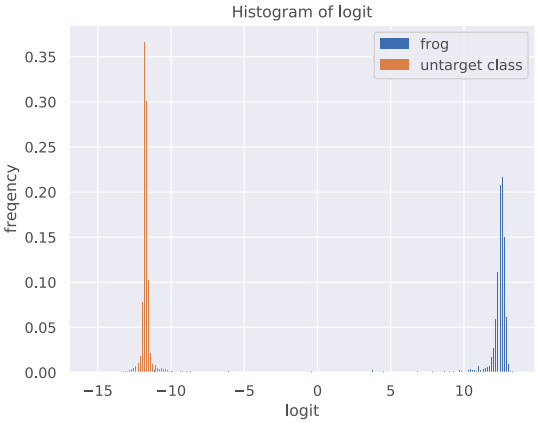
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

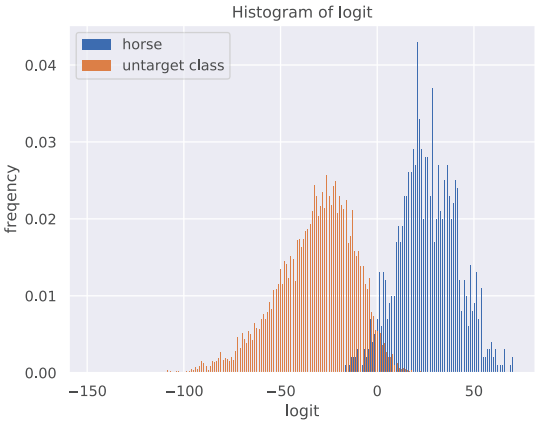


(c) シグモイド関数 (判別正則化なし)

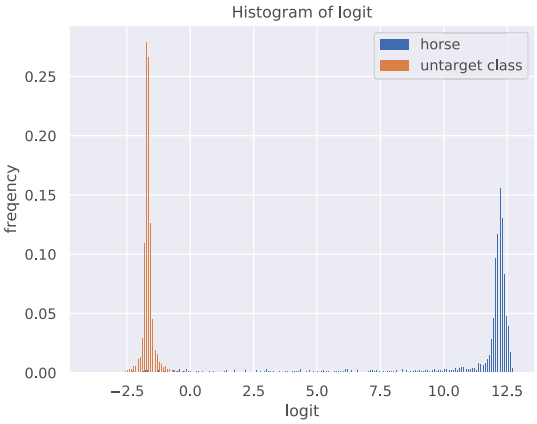


(d) シグモイド関数 (判別正則化あり)

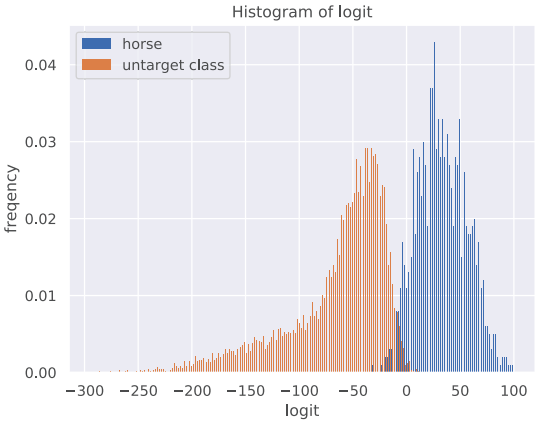
図 3.15: カエル



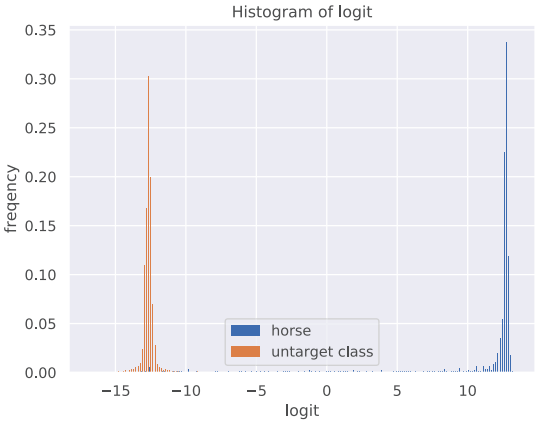
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

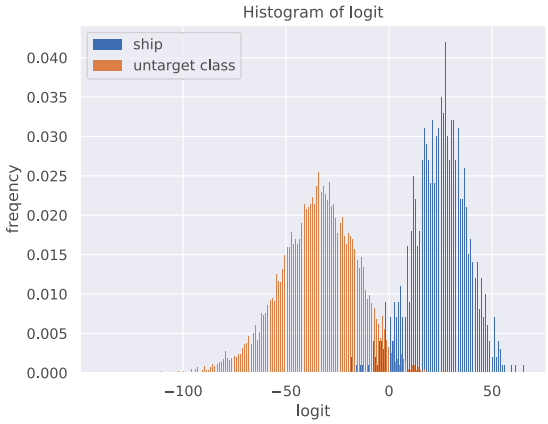


(c) シグモイド関数 (判別正則化なし)

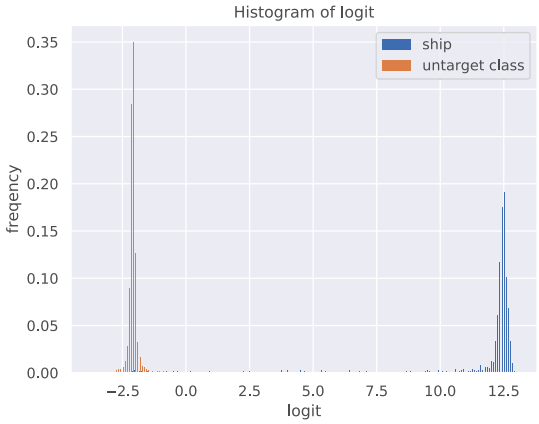


(d) シグモイド関数 (判別正則化あり)

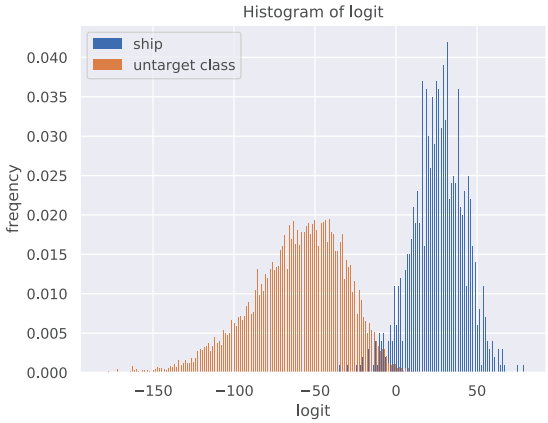
図 3.16: 馬



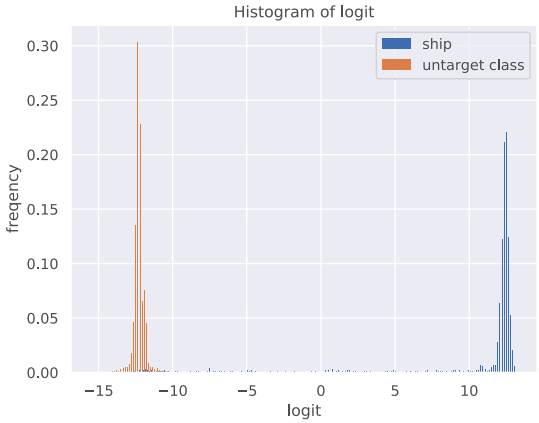
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

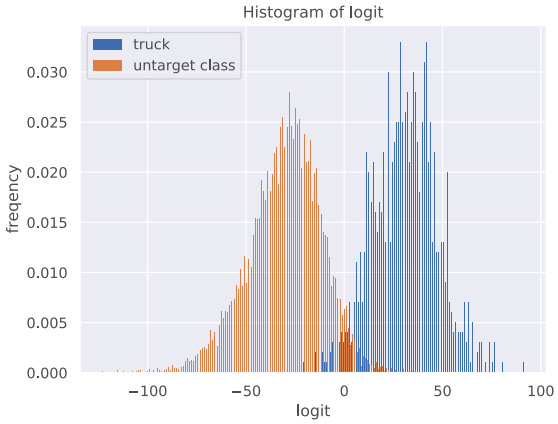


(c) シグモイド関数 (判別正則化なし)

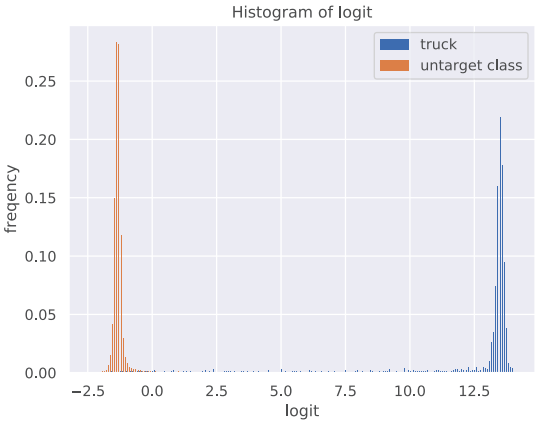


(d) シグモイド関数 (判別正則化あり)

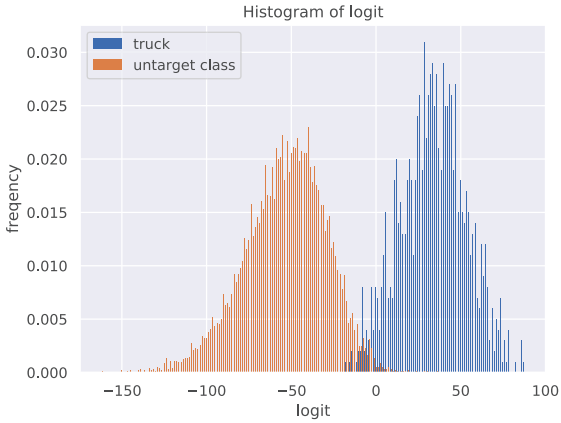
図 3.17: 船



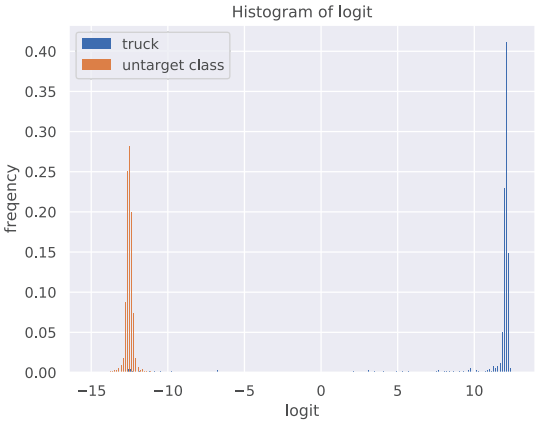
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)



(c) シグモイド関数 (判別正則化なし)



(d) シグモイド関数 (判別正則化あり)

図 3.18: トラック

4　効率的な CNN のための Pruning 手法

4.1　はじめに

近年, 畳み込みニューラルネットワーク (CNN) は, 画像認識や画像生成などの様々な分野で注目されている手法の一つである [3, 5, 41, 42]. CNN が優れた性能を発揮するためには，大量の変数を持つ層を何層も重ね, 階層構造を構築する必要がある [3, 5, 28, 26]．しかし，実用的な観点からは，そのような層が深い CNN を用いると, 大量の変数によって引き起こされる莫大な計算コストに苦しむことになる. 例えば，6,100 万の変数を持つ AlexNet [3] では，1 つの画像を分類するのに 7,200 万の浮動小数点演算 (FLOPs) が必要であるのに対し，画像分類の精度を向上させるために変数の数を 1 億 3,800 万に増やした VGG-16 では, 150 億 FLOPs が必要となる [12]. このような計算量の問題から，携帯電話のような, 計算資源が限られているエッジデバイスへの適用は, 一般的に困難である [13]．そのため，CNN の高い分類性能を維持しつつ，計算コストを削減することが注目されている．そのためのアプローチは, 主に 2 つに分類できる.

CNN の計算コストを削減するアプローチの一つは, ネットワーク構造の観点から CNN を改良するアプローチである．Howard らは，チャネルごとに 1×1 畳み込みフィルタを利用し, 畳み込み層の構造を大きく変更することで，FLOPs を大幅に削減しつつ AlexNet に匹敵する良好な性能を実現した (*MobileNet*) [14]. 類似したものとして, Zhang らは, *ShuffleNet* を提案した [15]. これらのアプローチは, ネットワークの構造が, 一般的に使用される CNN の構造と大きく異なるため, CNN の構造を再設計することが求められる.

それに対し, もう一つのアプローチは，ネットワークの変数を**剪定** *(Pruning)* することで，既存の（事前に学習された）CNN の変数の数を減らす Pruning 手法である．多くの Pruning 手法は，ニューラルネットワークの変数を削減する研究のパイオニア的な存在である LeCun ら [20] や Hassibi ら [21] の研究から派生したものである．近年では，Pruning 手法は CNN にも積極的に適用されている [22, 23, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84]．ＣＮＮでは, 主に畳み込みフィルタの変数が Pruning の対象となる. なぜなら，畳み込みフィルタによる計算は, CNN の計算コストの約 90 % を占めているからである [12]. CNN を 1 から再構築するアプローチに対して，Pruning によるアプローチは，優れた分類性能を達成した既存の CNN を再利用できるという点で有利である.

Pruning 手法は，分類などのタスクにあまり貢献しない, 重要度の低い変数をネットワークから剪定 (Pruning) することで, その計算コストを削減する手法である．そのため，Pruning 手法において最も重要なのは, 変数の**重要度**を測る Pruning 基準 (スコアリング関数) である．最も単純なスコアリング関数 [74] は，畳み込みフィルタの変数の値の大小を基に各フィルタの分類に対する貢献度を求めているが，分類損失を用いて, より直接的に貢献度を考慮すると, さらに洗練されたスコアリング関数にすることができる [22, 23]. 分類損失を考慮した手法は, *Optimal Brain Damage* [20] と *Optimal Brain Surgeon* [21] を基にしている場合が多い. 本当の意味での分類損失 (分類誤差) を扱うことは困難なので，それらの手法は, 学習サンプルと, そのサンプルが属する正解クラス (教師信号) を用いて計算された経験的分類損失を基に, 分類損失を推定する. しかし，経験的分類損失は，外れ値や分類が困難なサンプルに対して脆弱であるため，スコアリング関数の値 (Pruning Score) の信頼性を低下させる恐れがある.

本章では, まず, 4.2 章にて, 従来の Pruning 手法を 2 種類に大別し, その違いについて議論する. 4.3 章では, まず, 経験的分類損失の Taylor 展開を用いて, Pruning による分類損失の変動 (悪化) を近似 (予測) し, その変動が小さい変数を取り除く従来手法 [22, 23] のスコアリング関数を, 統一的に導出する. 次に, Taylor

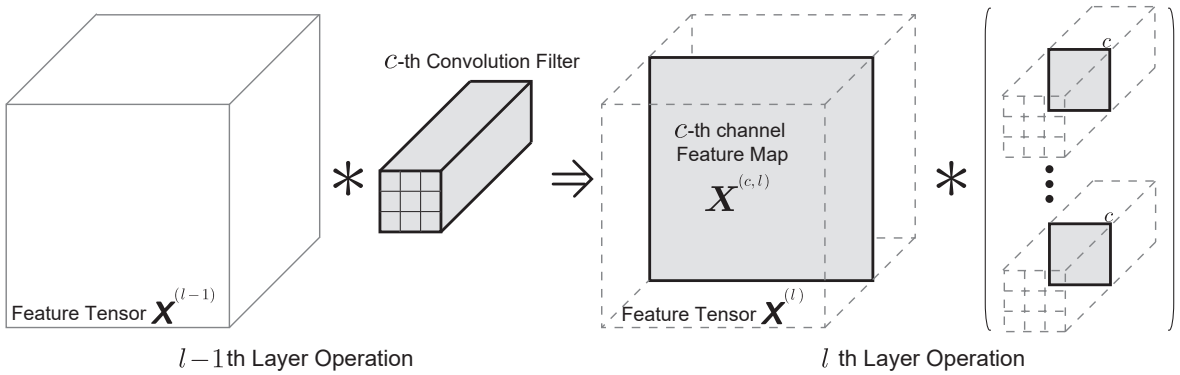


図 4.1: $l-1$ 層目と l-th 層での畳み込み処理．$l-1$-th 層で c 番目の畳み込みフィルターを除去すると，最終的に特徴量マップと濃い色で示されるフィルターの両方が除去される．

展開による近似 (予測) に悪影響を与える可能性のある極端なサンプルに対してロバスト性を持つ, 新しいスコアリング関数を提案する．また, Pruning Score のロバスト性を向上させる正規化手法も提案する．4.4 章では, Pruning 手法が本来解くべき問題を, 補助変数に関する最適化問題として改めて定式化し, 補助変数の最適化と CNN の再学習を交互に行うことで Pruning を行う手法も提案する．ここでは, 類似した従来手法の紹介や, 補助変数の離散最適化は不安定になりやすいこと, 連続最適化に拡張しても結果が芳しくないことを示したのち, それらの問題を改善した新たな手法を紹介する．4.5 章, 4.6 章では, 分類損失を考慮したスコアリング関数と, 従来手法の類似したものとの比較実験を行い, 提案したスコアリング関数の優位性を示したのち, 各手法の特性を解析する．4.7 章, 4.8 章では, 補助変数の最適化問題を解く Pruning 手法の比較実験や解析を行う．4.9 章では, CNN でよく用いられる Batch Normalization [11] と, 補助変数の連続最適化問題との相性の悪さを検証する．

本章は, 過去に発表した研究 [85] の論点を整理・追加したものである．

4.2 Pruning

Pruning 手法のパイオニア的な研究 [20, 21] は, フィードフォワードニューラルネットワークから重みを取り除くことで, ネットワークを剪定 (Pruning) するという課題に取り組んだ．近年では, 畳み込みニューラルネットワーク (CNN) に対しても Pruning を行うことで, CNN の計算コストを削減する研究が増えてきている．CNN に対する Pruning 手法は, **重みレベル** *(Weight-level)* のものと, **特徴レベル** *(Feature-level)* のものに分類することができるが [12], 本研究では後者の *Feature-level* の Pruning に焦点を当てている．

4.2.1 Weight-level Pruning

Weight-level Pruning は最もシンプルな Pruning 手法で, 各畳み込みフィルタの重要度の基準 (スコアリング関数) としてフィルタの重みの値を直接用いる手法である．*Weight-level Pruning* の一つに, 畳み込みフィルタの重みのノルムに基づいて, フィルタを Pruning するという単純な手法がある．Li らはスコアリング関数としての L_1 ノルムの効果を解析し [74], Molchanov らは L_2 ノルムの効果を調査した [23]．Wen らは, 畳み込みフィルタをスパースにする正則化を用いて CNN を学習した後, ノルムの小さいフィルタを除去した [75].

4.2.2 Feature-level Pruning

Feature-level Pruning は, スコアリング関数として特徴 (Feature) を用いる Pruning 手法である. 図 4.1 に示すように, $l-1$ 番目の層で c 番目の畳み込みフィルタを Pruning すると, 対応する l 番目の層のフィルタと c 番目の**特徴マップ** *(Feature Map)* も最終的に除去される. これらの異なる層の畳み込みフィルタは, c 番目の特徴マップを通して関係しているため, CNN に対する Pruning では, 畳み込みフィルタの代わりに**特徴マップ**の重要度を評価することも有効である.

Feature-level Pruning は, 一般的に次のように定式化される. ここで, 特徴マップの重要度を評価するためのスコアリング関数を $\mathtt{S}$, l 番目の層における c 番目のチャネルの特徴マップを $\boldsymbol{X}^{(c,l)} \in \mathbb{R}^{T\times T}$, そのベクトル化された表現を $\boldsymbol{x}^{(c,l)} = \mathtt{vec}(\boldsymbol{X}^{(c,l)}) \in \mathbb{R}^{T^2\times 1}$ とすると, 最も重要度の低い特徴マップは以下の式で求められる

$$(c^*, l^*) = \arg\min_{(c,l)\in\Omega} \mathtt{S}(\{\boldsymbol{x}_n^{(c,l)}\}_{n=1}^N). \tag{4.1}$$

ここで, Ω は現在のネットワークにおけるチャネル c と l 層のインデックスペアの集合を示し, $\boldsymbol{x}_n^{(c,l)}$ は N 個の学習サンプルのうち n 番目の画像から得られる特徴マップである. l^* 番目の層の c^* 番目のチャネルの特徴マップを除去すると, 図 4.1 に示すように, その 特徴マップに関連する畳み込みフィルタの Pruning も行われる. 重要度は, 各手法が用いるそれぞれのスコアリング関数 $\mathtt{S}$ を用いて計算されるが，このスコアリング関数 $\mathtt{S}$ はチャネル毎，層毎に適用される.

Ayinde らは, 特徴マップ間の cosine 類似度をスコアリング関数として採用し, 他の特徴マップとの類似度が高い特徴を Pruning した [76]. Peng らは，対象とする層で得られる特徴マップと, 前の層で得られる特徴マップの相関をスコアリング関数とした [79]. He ら [78] と Luo ら [77] は, ある特徴マップを Pruning した時のネットワーク全体の特徴マップの再構成誤差をスコアリング関数とした. Yu らは, ネットワークの出力層に入力される特徴の重要度をランク付けすることで, 関連する各ニューロンの重要度スコアを計算した [80]. そして, 出力層の入力で得られた重要度スコアをネットワーク全体に逆伝播させ, スコアの低い特徴マップを Pruning した.

上述した手法が 特徴マップの重要度を直接評価するのに対し, 特徴マップの分類への寄与度に着目した手法もある [81, 23, 22, 84]. Huang らは, 分類精度と Pruning 率の積として定義された報酬関数に基づいて, 重要度の低い特徴マップを Pruning するエージェントを訓練した [81]. また, いくつかの研究では, 学習サンプルを用いて経験的分類損失を計算することで, 各特徴マップが分類損失に与える影響を予測した. 特徴マップを Pruning することで発生する損失の変動は, テイラー展開を用いて数学的に近似することができ [20, 21] , それらを基にした手法 [23, 22, 84] は他の手法と比較しても良好な性能を示した.

4.3 スコアリング関数を用いた Pruning 手法の提案

本節では, 優れた性能を示した従来手法 [23, 22] を踏襲し, 経験的分類損失の**テイラー展開**に基づく新たなスコアリング関数 $\mathtt{S}$ を提案する. ここでは, 重要度の低い特徴マップは分類損失への影響が少ない, つまり, そのような特徴マップを Pruning しても分類損失はほとんど変わらないことを前提としている. 分類損失のテイラー展開を用いることで, 特徴マップの Pruning による分類損失の変動（劣化）を近似することができる.

4.3.1 損失のテイラー展開

教師あり学習で一般的な手法の一つである CNN の学習は, N 個の学習サンプルによる経験的分類損失を定式化するために, 主にソフトマックスクロスエントロピーを用いる.

$$\mathcal{L} = -\sum_{n=1}^{N} \log[p_{n,t_n}] \tag{4.2}$$

ここで, $p_{n,k}$ は k 番目のクラスに属する n 番目のサンプルを入力した時のソフトマックスの出力 ($\sum_{k=1}^{K} p_{n,k} = 1, \forall n$) を示し, $t_n \in \{1, \cdots, K\}$ は n 番目のサンプルに割り当てられたクラスラベルである.

次に, n 番目のサンプルから得られる特徴マップ $\boldsymbol{x}_n \in \mathbb{R}^{T^2 \times 1}$ の値を変動させることで発生する損失 $\mathcal{L}$ の摂動 $\Delta_n(\boldsymbol{\delta})$ は, 二次までのテイラー展開を用いて次のように近似できる [*2]

$$\Delta_n(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{x}_n + \boldsymbol{\delta}) - \mathcal{L}(\boldsymbol{x}_n) \approx \boldsymbol{\delta}^\top \boldsymbol{\nabla}_n + \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{H}_n \boldsymbol{\delta}. \tag{4.3}$$

ここで, 勾配ベクトルは $\boldsymbol{\nabla}_n = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_n} \in \mathbb{R}^{T^2 \times 1}$, Hessian 行列は $\boldsymbol{H}_n = \frac{\partial}{\partial x_n}\frac{\partial^\top}{\partial x_n}\mathcal{L} \in \mathbb{R}^{T^2 \times T^2}$ と表記される. Hessian 行列 $\boldsymbol{H}_n$ を求めるのは計算量の問題で困難なので, 次のように近似する

$$\boldsymbol{H}_n \approx \boldsymbol{\nabla}_n \boldsymbol{\nabla}_n^\top. \tag{4.4}$$

近似の詳細は B.1 章に記載してある.

特徴マップ $\boldsymbol{x}_n$ を取り除くことは, 特徴マップの値を 0 , つまり $\boldsymbol{x}_n + \boldsymbol{\delta} = \boldsymbol{0}$ にすることでもあるので, $\boldsymbol{\delta} = -\boldsymbol{x}_n$ と表すことができる. つまり, 式 (4.3) のテイラー展開を用いると, 特徴マップ $\{\boldsymbol{x}_n\}_{n=1}^N$ を取り除くことによる分類損失の変化を, 以下の式のように評価できる

$$\mathtt{S}(\{\boldsymbol{x}_n\}_{n=1}^N) = \mathrm{E}_n\left[\Delta_n(-\boldsymbol{x}_n)\right] = \mathrm{E}_n\left[-\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n + \frac{1}{2}\boldsymbol{x}_n^\top \boldsymbol{H}_n \boldsymbol{x}_n\right]. \tag{4.5}$$

これは, $\mathtt{S}(\{\boldsymbol{x}_n\}_{n=1}^N)$ の値が小さいほど分類に対するその特徴マップの重要度が低く, さらに, そのような特徴マップほど Pruning される可能性が高いことを示している.

4.3.2 よりロバストな pruning を目指して

前述したように, テイラー展開から理論的に導出されたスコアリング関数 S は, 分類問題における特徴マップの重要度を容易に評価できる. しかし, 本当の分類損失は未知で扱うことができないため, 代わりに学習サンプルを用いた**経験的な**分類損失を近似的に用いるが, それによりスコアリング関数のロバスト性に関して 2 つの問題が発生する.

一つ目の問題点は, テイラー展開の一次項 $-\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n$ は値の下界が無いため, 負の方向に大きな値を取りうるのに対し, 二次項はその下界が 0 である点である. 実用的な観点から考えると, このような下界の無い項は, 極端な値を出力するサンプルに対して脆弱である可能性がある. 予備実験では, 式 (4.5) の第一項のみ、第一項と第二項の両方、第二項のみの 3 種類のスコアリング関数を用いて, テイラー展開の各項の性能を実際に評価した. 比較結果は表 4.2 に示した. 実験の詳細な設定は 4.5.1 章に記載した. この比較結果から, 第一項を含むスコアリング関数は, 第二項のみのものよりもわずかに劣ることがわかる. ゆえに, 下界の無い第一項を除外したのち, 非負である第二項 $\mathtt{S}(\{\boldsymbol{x}_n\}_{n=1}^N) = \mathrm{E}_n[\boldsymbol{x}_n^\top \boldsymbol{H}_n \boldsymbol{x}_n]$ に基づいてスコアリング関数を設計する.

[*2] 特徴マップは実際には $\boldsymbol{x}_n^{(c,l)}$ で表されるが, 表記をシンプルにするためにチャネル c と層 l のインデックスを省略している.

二つ目の問題点は, 第二項が二次形式である点である. 式 (4.4) の Hessian $\boldsymbol{H}_n$ の近似形を用いると, テイラー展開の第二項を次のように書き換えることができる

$$\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{H}_n \boldsymbol{x}_n\right] = \mathrm{E}_n\left[(\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n)^2\right]. \tag{4.6}$$

上の式は, $\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n$ の二次形式である. このような二次形式は, 極端な値をとるサンプルに対して脆弱なため, そのようなサンプルがたとえ少数であっても, それらに影響されてスコアの値が極端に大きくなる. そのようなサンプルに対する Pruning スコアのロバスト性を向上させるために, 第二項の下界の形を考えると

$$\mathrm{E}_n\left[(\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n)^2\right] \geq \left(\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]\right)^2 \tag{4.7}$$

となる. つづいて, 重要度の高い特徴マップ (Important Feature Map) と重要度の低い特徴マップ (Redundant Feature Maps) の両方を含む toy データ (図 4.2) を用いて, 式 (4.7) の 2 つの項のロバスト性を議論する. 図 4.2 (a) の スコアリング関数 $\mathrm{E}_n\left[\left(\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right)^2\right]$ は, 3 番目の入力サンプルによってニューロンが偶然発火したため, Important Feature Maps より**高い**スコアを Redundant Feature Maps に与えている. 一方, 図 4.2 (b) の $\left(\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]\right)^2$ は, Redundant Feature Maps のスコアを**低く** , つまり重要ではないと判断することに成功している. このように, 式 (4.7) の下界である $\left(\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]\right)^2$ は, スコアリング関数として, よりロバストであることがわかる.

また, 式 (4.7) は, 過去の研究 [22, 23] との興味深い関連性を示している. 式の左側の項が Theis ら [22] のスコアリング関数であり, 右側の項が Molchanov ら [23] のスコアリング関数である[*3]. この 2 つの手法は異なるアプローチで提案されているが, テイラー展開を解析することで統一的に導出することができた.

つづいて, 以下の式のように分解を行い, $\mathrm{E}_n[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n]$ をさらに解析する

$$\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right] = \mathrm{E}_n[\boldsymbol{x}_n]^\top \mathrm{E}_n[\boldsymbol{\nabla}_n] + \mathrm{Corr}_n(\boldsymbol{x}_n, \boldsymbol{\nabla}_n). \tag{4.8}$$

Corr は

$$\mathrm{Corr}_n(\boldsymbol{x}_n, \boldsymbol{\nabla}_n) = \mathrm{E}_n\left[(\boldsymbol{x}_n - \mathrm{E}_n[\boldsymbol{x}_n])^\top (\boldsymbol{\nabla}_n - \mathrm{E}_n[\boldsymbol{\nabla}_n])\right] \tag{4.9}$$

と定義される相互相関である. ここで, $\boldsymbol{x}_n$ が**重要な**特徴マップであるならば, 特徴マップ $\boldsymbol{x}_n$ と勾配 $\boldsymbol{\nabla}_n$ の相互相関は**負**であることに注意する必要がある. 重要な特徴マップは有意なニューロンの発火 $\boldsymbol{x}_n \gg 0$ を示すと言われており, さらに式 (4.5) から, 対応する勾配は $\boldsymbol{\nabla}_n \ll 0$ となる. しかし, 式 (4.8) から, このような重要な特徴マップ $\boldsymbol{x}$ でさえ $\mathrm{E}_n[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n]$ のスコアを減少させ, 同様に $(\mathrm{E}_n[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n])^2$ の Pruning Score をも減少させる危険性があることがわかる. これは, $\left(\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]\right)^2$ を用いるスコアリング関数によって, 重要な特徴マップが取り除かれる可能性を示唆している. そのような可能性を減らすために, 特徴マップの重要度をより正しく評価するスコアリング関数を求める. 関数は, 式 (4.8) の相関項 Corr を無視した

$$\mathtt{S}(\{\boldsymbol{x}_n\}_{n=1}^N) = \left(\mathrm{E}_n[\boldsymbol{x}_n]^\top \mathrm{E}_n[\boldsymbol{\nabla}_n]\right)^2 = \bar{\boldsymbol{x}}^\top \left(\bar{\boldsymbol{\nabla}} \bar{\boldsymbol{\nabla}}^\top\right) \bar{\boldsymbol{x}} = \bar{\boldsymbol{x}}^\top \bar{\boldsymbol{H}} \bar{\boldsymbol{x}} \tag{4.10}$$

である. ここで

$$\bar{\boldsymbol{x}} = \mathrm{E}_n[\boldsymbol{x}_n], \quad \bar{\boldsymbol{\nabla}} = \mathrm{E}_n[\boldsymbol{\nabla}_n], \quad \bar{\boldsymbol{H}} = \bar{\boldsymbol{\nabla}} \bar{\boldsymbol{\nabla}}^\top \tag{4.11}$$

である. 式 (4.10) で提案したスコアリング関数は，平均化された特徴マップと勾配の内積という非常にシンプルな式である．さらに, 特徴マップと勾配の平均を取ることで, 勾配の値やニューロンの出力値が偶然大きく

[*3] 実際には, [23] は絶対値を用いている: $|\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]| = \sqrt{\left(\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]\right)^2}$

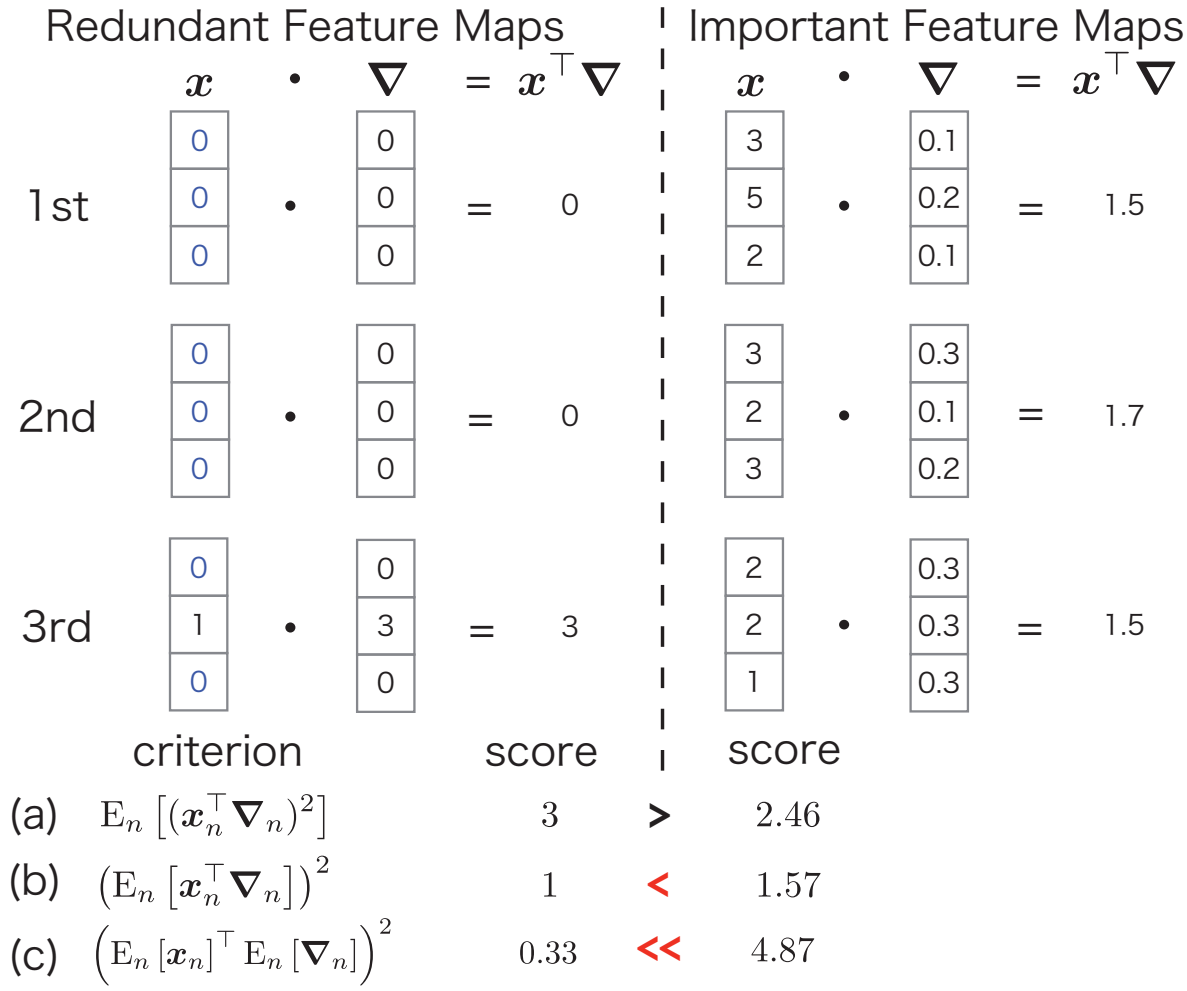


図 4.2: スコアリング関数のロバスト性.

なるサンプルに対する Pruning Score のロバスト性が向上する. Toy データを用いた 4.3.2 章の議論と同様に, 図 4.2 において, $\bar{\boldsymbol{x}}^\top \bar{\boldsymbol{H}} \bar{\boldsymbol{x}}$ のロバスト性を示す. $(E_n[\boldsymbol{x}_n^\top \nabla_n])^2$ に比べ, 提案するスコアリング関数 $\bar{\boldsymbol{x}}^\top \bar{\boldsymbol{H}} \bar{\boldsymbol{x}}$ の方が, Redundant Feature Maps に対してはより低いスコアを, Important Feature Maps に対してはより高いスコアを与えることができるため, スコアに悪影響を与えうるサンプルに対してロバスト性が高いことを示唆している.

4.3.3 提案したスコアリング関数の拡張

ここでは, 前節で提案したスコアリング関数 (式 (4.10)) の亜種と, 層の違いに対するロバスト性を高めるための正規化手法を示す.

■Variant 前節で提案したスコアリング関数 (式 (4.10)) は, $\bar{\boldsymbol{x}}$ の二次形式と半正定値 (Positive Semidifinite, PSD) 行列 $\hat{\boldsymbol{H}}$ で構成されているため, *metric* の観点からも解釈可能である. すなわち, *metric* 行列を $\mathtt{D}(\mathbf{0}, \boldsymbol{x}) = (\boldsymbol{x} - \mathbf{0})^\top \hat{\boldsymbol{H}} (\boldsymbol{x} - \mathbf{0})$ とすると, Pruning 前の特徴マップ $\boldsymbol{x}$ と Pruning 後の特徴マップ $\mathbf{0}$ との距離は, $\hat{\boldsymbol{H}}$ を用いて求められる. Pruning は, その後, 距離が $\mathbf{0}$ に最も近い特徴マップを取り除くことで行われる.

この metric の観点から, 情報幾何学の枠組みにおけるフィッシャー行列 [86] のように, 分類損失における $\mathbf{0}$ と $\boldsymbol{x}$ の距離を測る metric 行列として PSD 行列 $\hat{\boldsymbol{H}}$ が機能することがポイントである. 従来手法 [22, 23] も, 各サンプル $\boldsymbol{x}_n$ におけるサンプルごとの metric $\boldsymbol{H}_n$ を用いて, そのような距離を測定していると考えられる. しかし, これは画像のクラスや画質などのサンプル毎に変化する *ad-hoc* な特性の影響を受けるため, 距離の測定が**不安定**になる. これに対して, 式 (4.10) の我々の手法は, サンプル全体で変化しない**安定した** $\hat{\boldsymbol{H}}$ を用いているため, 各特徴マップには単一の metric 行列 $\hat{\boldsymbol{H}}$ が与えられている. このような観点から, 式 (4.10) の亜種を新たに導出する

$$\mathtt{S}(\{\boldsymbol{x}_n\}_{n=1}^N) = \mathrm{E}_n\left[\boldsymbol{x}_n^\top \hat{\boldsymbol{H}} \boldsymbol{x}_n\right] = \mathrm{trace}\left(\hat{\boldsymbol{H}}\, \mathrm{E}_n[\boldsymbol{x}_n \boldsymbol{x}_n^\top]\right). \tag{4.12}$$

このとき, 式 (4.10) から式 (4.12) へ拡張するために, 特徴マップの一次項 $\bar{\boldsymbol{x}} = \mathrm{E}_n[\boldsymbol{x}_n]$ を二次項 $\mathrm{E}_n[\boldsymbol{x}_n \boldsymbol{x}_n^\top]$ に置き換える. この 2 つのスコアリング関数 (式 (4.10), 式 (4.12)) も, 4.5 章にて比較する.

■Normalization　これまで, 個別の特徴マップ $\boldsymbol{x}^{(c,l)}$ に注目して定義された Pruning Score について議論してきた. しかし, feture map はチャネル c ごとに独立した表現ではないため, 各チャネル $c \in \{1, \cdots, C\}$ の間に何らかの関係がある. このような相関を考慮して Pruning を行うために, Pruning Score $S^{(c,l)} \triangleq \mathtt{S}(\{\boldsymbol{x}_n^{(c,l)}\}_{n=1}^N)$ をチャネル c 間で正規化する正規化手法

$$\tilde{S}_{L_p}^{(c,l)} = \frac{\sqrt{S^{(c,l)}}}{\left\{\sum_{c'=1}^{C} \sqrt{S^{(c',l)}}^{p}\right\}^{\frac{1}{p}}} \tag{4.13}$$

も提案する. これは, Molchanov ら [23] が行なっていた正規化を一般化したものである. 式 (4.10), 式 (4.12) はどちらも 2 次形式であるため, 統計学的な観点から, Pruning Score $S^{(c,l)}$ の平方根を用いて l 番目の層での Score の L_p ノルムを計算し, Score を正規化する. Pruning の際には, Score の高い特徴マップが多く含まれる特定の層を集中的に Pruning するのではなく, 各層から一様に Pruning するのが好ましいと考えられる. なぜならば, 特定の層から集中して特徴マップを取り除くと層全体に対して大きな影響が発生するが, 重要度の低い特徴マップを各層から**まばらに**取り除いても, 他の良い (重要な) 特徴マップで簡単に補うことができ, 層全体でみると良い表現を維持することができるからである. $p \leq 1$ の L_p ノルムは高い Score $\{S^{(c,l)}\}_{c \neq c^*}$ が密集している層では大きくなるので, 上記の議論を踏まえて, Score の正規化には $p \leq 1$ を考える.

4.4　補助変数の最適化による Pruning 手法

前節では, 特徴マップ (Feature Map) を CNN から削除すると分類損失がどの程度変化 (劣化) するかを近似 (予測) し, 変動が少ない特徴マップを Pruning する手法を提案した.

本節では, Pruning すべき特徴マップを学習によって求める新たな Pruning 手法を提案する. そのために, まず, 指標関数 $h(\cdot)$ と補助変数 $\boldsymbol{m}$ を導入し, Pruning 手法が本来解くべき問題を $\boldsymbol{m}$ に関する離散最適化問題として定式化する. 次に, $\boldsymbol{m}$ の離散最適化は不安定になりやすく, 連続最適化に拡張しても好ましい性能が得られにくいという問題について議論した後, それらの問題点を改善した新たな Pruning 手法を提案する.

4.4.1　Pruning 手法が解くべき最適化問題

ここでは, 重要度の低い畳み込みフィルタを CNN から取り除く Pruning の仕組み[*4]を改めて定式化した後, Pruning 手法が本来解くべき離散最適化問題を定式化する. そのために, まず, 畳み込み層の仕組みについて

[*4] 本質的には, Pruning で取り除くのは特徴マップであるが, 実際の作業で取り除くのは畳み込みフィルタである

改めておさらいする．Pruning の対象となる l 番目の層の畳み込みフィルタ $\boldsymbol{W}^{(l)}$ は

$$\boldsymbol{W}^{(l)} = \left(\boldsymbol{w}^{(1,l)}, \ldots, \boldsymbol{w}^{(C,l)}\right)^T \tag{4.14}$$

と表される．ここで, C はチャネル数であり, $\boldsymbol{w}^{(c,l)}$ は l 層目の c 番目のチャネルの畳み込みフィルタである．フィルタ $\boldsymbol{w}^{(c,l)}$ の出力 $\boldsymbol{u}^{(c,l)}$ は，C チャネルの特徴マップ $\boldsymbol{X}^{(l)} = \left(\boldsymbol{x}^{(1,l)}, \ldots, \boldsymbol{x}^{(C,l)}\right)^T$ が入力されると

$$\boldsymbol{u}^{(c,l)} = \boldsymbol{w}^{(c,l)} * \boldsymbol{X}^{(l)} \tag{4.15}$$

となる．$*$ は畳み込み演算である．そして, 畳み込み層の最終的な出力, すなわち次の層の入力 $\boldsymbol{x}^{(c,l+1)}$ は

$$\boldsymbol{x}^{(c,l+1)} = \sigma\left(\boldsymbol{u}^{(c,l)}\right) \tag{4.16}$$

と表すことができる．ここで, $\sigma(\cdot)$ は, 主に非線形変換を行う活性化関数である．このような畳み込み層から畳み込みフィルタ $\boldsymbol{w}^{(c,l)}$ を Pruning する作業は, フィルタの重み $\boldsymbol{w}^{(c,l)}$ の全要素の値を 0 にすることと等しいため, 指標関数 $h(\cdot)$ を用いて

$$\tilde{\boldsymbol{w}}^{(c,l)} = h(m^{(c,l)})\boldsymbol{w}^{(c,l)} \tag{4.17}$$

と表すことができる．$h(\cdot)$ は, 通常

$$h(m^{(c,l)}) = \begin{cases} 0, & \text{if } m^{(c,l)} \leq \theta \\ 1, & \text{otherwise} \end{cases} \tag{4.18}$$

で定義される指標関数である．$m^{(c,l)}$ は, 畳み込みフィルタ $\boldsymbol{w}^{(c,l)}$ の重要度を表す補助変数である．また, θ はフィルタの削減度合いをコントロールする閾値である．フィルタ $\boldsymbol{w}^{(c,l)}$ が Pruning されるならば, 指標関数 $h(m^{(c,l)})$ の値は 0 に, 逆に, Pruning されず CNN に残されるならば 1 となる．Pruning 後の畳み込みフィルタの出力 $\tilde{\boldsymbol{u}}^{(c,l)}$ は

$$\begin{aligned} \tilde{\boldsymbol{u}}^{(c,l)} &= \tilde{\boldsymbol{w}}^{(c,l)} * \boldsymbol{X}^{(l)} \\ &= h\left(m^{(c,l)}\right)\boldsymbol{w}^{(c,l)} * \boldsymbol{X}^{(l)} \\ &= h\left(m^{(c,l)}\right)\boldsymbol{u}^{(c,l)} \end{aligned} \tag{4.19}$$

となる．一般的な Pruning 手法は, $m^{(c,l)}$ や θ を間接的に最適化し, 閾値以下の $m^{(c,l)}$ に対応する畳み込みフィルタ $\boldsymbol{w}^{(c,l)}$ を CNN から取り除く．

式 (4.17), 式 (4.18) から, Pruning 手法が解くべき問題は, 可能な限り多くの畳み込みフィルタを取り除きつつ分類損失も最小化する最適な補助変数 $\mathcal{M}$ とフィルタ $\mathcal{W}$ を求める最適化問題

$$\min_{\mathcal{M},\mathcal{W}} \frac{1}{N}\left(\sum_{n}^{N} \mathcal{L}\left(X_n, \mathcal{M}, \mathcal{W}\right)\right) + \lambda \mathcal{R}\left(h\left(\mathcal{M}\right)\right) \tag{4.20}$$

であると考えることができる．ここで, $\mathcal{W}$ と $\mathcal{M}$ は, それぞれ, CNN の全てのフィルタの重みと, それに対応する補助変数である．$\mathcal{L}$ は分類損失を評価する損失関数であり, $\mathcal{R}(\cdot)$ は可能な限り多くのフィルタを取り除く (対応する指標関数の出力を 0 にする) ための正則化関数, λ はフィルタの削減量をコントロールするハイパーパラメータである．多くの Pruning 手法が式 (4.20) を間接的に解いているのに対し, Xiao らが提案した AutoPrune [24] は, ステップ関数を指標関数 $h(\cdot)$ とし, $\mathcal{M}$ と $\mathcal{W}$ に関して式 (4.20) の離散最適化問題を直接解くことで, CNN の計算コストをかなり削減した．

4.4.2 指標関数から連続関数への拡張とその限界

4.4.1 章での議論から, Pruning が解くべき問題は, 指標関数 $h(\cdot)$ を介して補助変数 $\mathcal{M}$ に関する離散最適化問題を解くことであるとわかる. しかし, 離散最適化問題を解くため, 最適化が不安定になる場合があることを実験的に確認した (4.7 章). それに対する最もシンプルな解決策は, 指標関数を次のような連続関数

$$h_\sigma(m^{(c,l)}) = \frac{1}{1+\exp(-\alpha m^{(c,l)})} \tag{4.21}$$

に置き換え, 連続最適化問題に拡張することである. ここで, $h_\sigma(\cdot)$ はシグモイド関数の一般式であり, α はその傾きをコントロールするハイパーパラメータである. しかし, 実際には最適化が安定せず, さらに, Pruning 後の CNN の分類性能が悪化することも確認した (4.7 章). これは, 多くの CNN に導入されている Batch Normalization [11] と, 補助変数の連続最適化の相性の悪さに起因している.

補助変数の連続最適化と Batch Normalization の相性の悪さについて, N 個の学習サンプルのうち, n 番目のサンプルを入力した時に得られる, ある畳み込みフィルタの出力 $\boldsymbol{u}_n^{(c,l)} \in \mathbb{R}^{D\times 1}$ を用いて説明する. D は特徴マップの要素数である. Batch Normalization はフィルタの出力の標準化 (正規化) をチャネルごとに行うため, 正規化後の出力は

$$BN(\boldsymbol{u}_n^{(c,l)}) = \frac{\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}}{\sqrt{\mathrm{E}_n\left[\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)^T\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)\right] + \epsilon}} \tag{4.22}$$

と表される. ここで, 正規化に用いられる平均は $\mu^{(c)} = \dfrac{1}{ND}\displaystyle\sum_{n,d}^{N,D} u_{n,d}^{(c,l)}$ である. また, ϵ は分母の値が 0 になるのを防ぐための定数である. 正規化後, 各チャネルの特徴マップの平均・分散を任意に変更したものが, Batch Normalization の最終的な出力である.

次に, 補助変数 $m^{(c,l)}$ と指標関数 $h(\cdot)$ をネットワークの l 層の畳み込みフィルタ $\boldsymbol{w}^{(c,l)}$ に導入した後の Batch Normalization の出力を考える. 導入後のフィルタの出力を $\tilde{\boldsymbol{u}}^{(c,l)}$, その平均を $\tilde{\mu}^{(c)}$ とすると, Batch

Normalization による正規化後の出力は

$$
\begin{aligned}
BN(\tilde{\boldsymbol{u}}_n^{(c,l)}) &= \frac{\tilde{\boldsymbol{u}}_n^{(c,l)} - \tilde{\mu}^{(c)}}{\sqrt{\mathrm{E}_n\left[\left(\tilde{\boldsymbol{u}}_n^{(c,l)} - \tilde{\mu}^{(c)}\right)^T \left(\tilde{\boldsymbol{u}}_n^{(c,l)} - \tilde{\mu}^{(c)}\right)\right] + \epsilon}} \\
&= \frac{h\left(m^{(c,l)}\right)\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)}{\sqrt{\left(h\left(m^{(c,l)}\right)\right)^2 \mathrm{E}_n\left[\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)^T \left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)\right] + \epsilon}} \\
&= \frac{h\left(m^{(c,l)}\right)\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)}{|h\left(m^{(c,l)}\right)|\sqrt{\mathrm{E}_n\left[\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)^T \left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)\right] + \epsilon}} \\
&= \frac{sgn\left(h\left(m^{(c,l)}\right)\right)\left(\boldsymbol{u}_n^{(c,l)} - \mathrm{E}_n\left[\boldsymbol{u}_n^{(c,l)}\right]\right)}{\sqrt{\mathrm{E}_n\left[\left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)^T \left(\boldsymbol{u}_n^{(c,l)} - \mu^{(c)}\right)\right] + \epsilon}} \\
&= sgn\left(h\left(m^{(c,l)}\right)\right) BN(\boldsymbol{u}_n^{(c,l)})
\end{aligned}
\tag{4.23}
$$

と表すことができる．$sgn\,(\cdot)$ は入力値の符号を取り出す符号関数である．式 (4.23) から, Batch Normalization をネットワークに導入すると, 指標関数 $h\,(\cdot)$ は符号関数に変化する．Xiao らが提案した AutoPrune [24] のように, 0 もしくは 1 を出力するステップ関数が指標関数ならば特に問題はないが, シグモイド関数のような正の連続値を出力する連続関数を指標関数の代わりに用いるならば, 0 以外は全て 1 となるため, 指標関数や補助変数の値の大小関係は意味のないものとなる．それによって閾値の調整が難しくなることが, 指標関数を連続関数へ拡張することで Pruning 後の CNN の分類性能が悪化する原因の一つであると考えられる．また, 連続関数を Batch Normalization の出力に対して導入しても, 次の層の Batch Normalization による正規化で, 同様に打ち消されてしまう可能性がある．

4.4.3 Pruning Block

ここまでは, Pruning 手法が本来解くべき最適化問題と, その過程で発生する最適化の不安定さについて議論してきた．ここでは, 特徴マップが持つ情報を圧縮し, 圧縮した空間上で補助変数 $\mathcal{M}$ の連続最適化問題を解くことで, 安定した最適化と分類性能の維持を試みる新たな Pruning 手法を提案する．

提案手法のコアとなる部分は, 図 4.3 に示す *Pruning Block* である．Pruning Block は, 学習済み CNN の各畳み込み層に導入される．一つの Pruning Block は, Encoder, 補助変数と連続関数, 活性化関数, Decoder で構成されており, それぞれ, 畳み込み層の特徴マップ $\boldsymbol{U}^{(l)}$ が持つ情報を低次元へ圧縮, 圧縮した空間上での Pruning, Pruning 後の特徴マップの非線型変換, Pruning 後の低次元特徴マップを元の次元への再構築, を行う．まず, Pruning Block の Encoder 部分は, フィルタサイズが 1×1 の畳み込みフィルタ $\boldsymbol{W}_{enc} \in \mathbb{R}^{C' \times C}$ を用いて, l 層目の畳み込み層から得られる C チャネルの特徴マップ $\boldsymbol{U}^{(l)} = \left(\boldsymbol{u}^{(1,l)}, \ldots, \boldsymbol{u}^{(C,l)}\right)^T$ を, チャネル数がより少ない C' チャネルの特徴マップ $\boldsymbol{Z}^{(l)} = \left(\boldsymbol{z}^{(1,l)}, \ldots, \boldsymbol{z}^{(C',l)}\right)^T$ に射影することで, $\boldsymbol{U}^{(l)}$ のチャネル方向の情報を圧縮する．それを式で表すと

$$
z_d^{(c',l)} = \sum_c^C w_{enc}^{(c',c)} u_d^{(c,l)} \tag{4.24}
$$

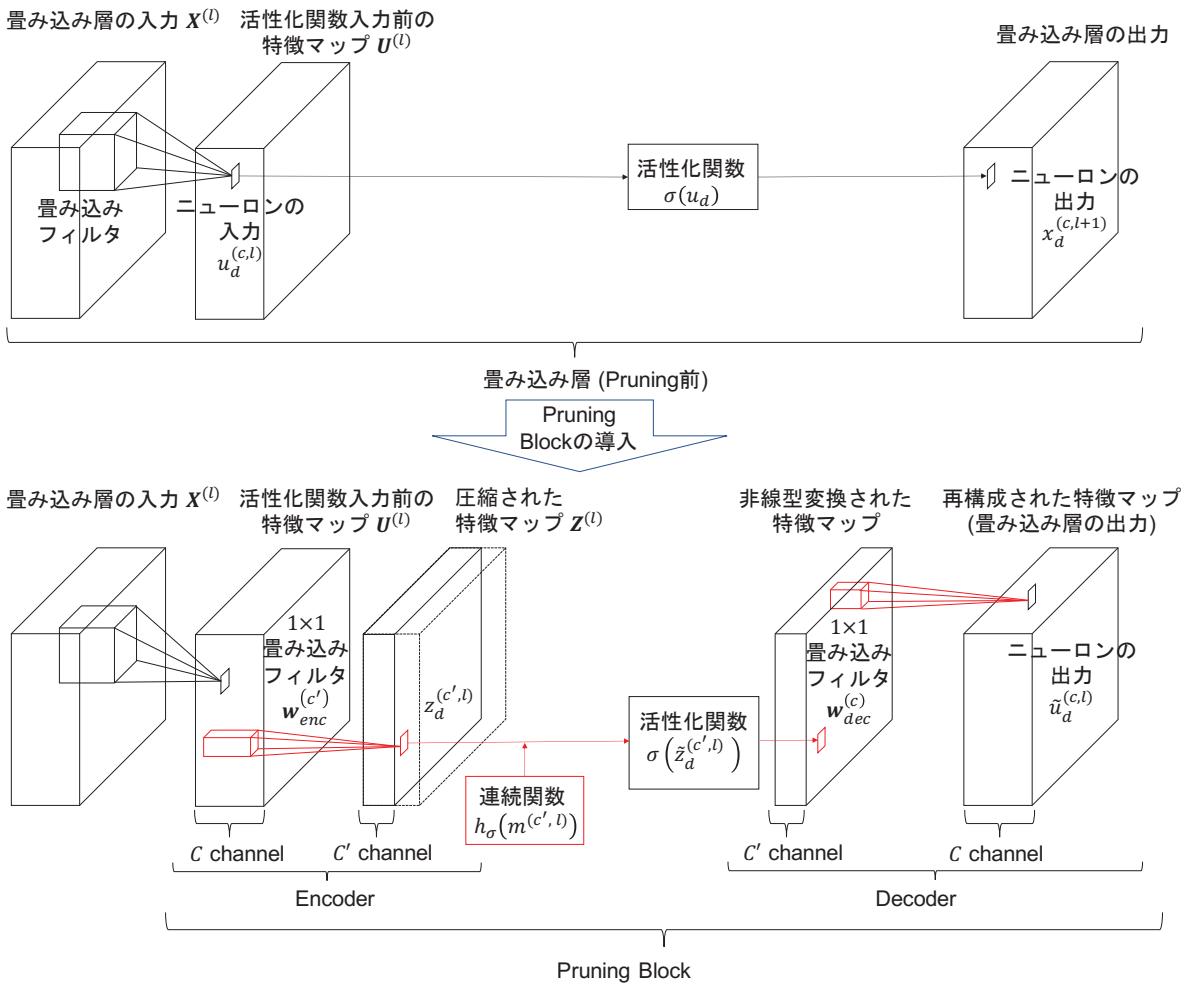


図 4.3: 畳み込み層に導入された Pruning Block.

となる. $u_d^{(c,l)}$ と $z_d^{(c',l)}$ は, それぞれ $\boldsymbol{u}^{(c,l)}$ と $\boldsymbol{z}^{(c',l)}$ の d 番目の要素である. $w_{enc}^{(c',c)}$ は, 入力チャネルが c, 出力チャネルが c', フィルタサイズが 1×1 である畳み込みフィルタ (1×1 畳み込みフィルタ) の重みである. 次に, 情報を圧縮した空間上で Pruning を行うために, 各チャネルの重要度を評価する補助変数 $m^{(c,l)}$ と連続関数 $h_\sigma(\cdot)$ を, 圧縮後の特徴マップ $\boldsymbol{Z}^{(l)}$ に対して適用する.

$$\tilde{\boldsymbol{z}}^{(c',l)} = h_\sigma\left(m^{(c',l)}\right)\boldsymbol{z}^{(c',l)} \tag{4.25}$$

その後, 活性化関数 $\sigma(\cdot)$ を用いて非線型変換を行ったのち, Decoder 部分の 1×1 畳み込みフィルタ $\boldsymbol{W}_{dec} \in \mathbb{R}^{C \times C'}$ が再構成した特徴マップ

$$\tilde{u}_d^{(c,l)} = \sum_{c'}^{C'} w_{dec}^{(c,c')} \sigma\left(\tilde{z}_d^{(c',l)}\right) \tag{4.26}$$

を畳み込み層の最終的な出力 (次の層の入力) とする. 以上のように, チャネル方向の情報を圧縮した空間上で補助変数の連続最適化問題を解くことで, Batch Normalization が行うチャネル単位の正規化による連続最適化問題への悪影響を緩和しつつ, 安定した Pruning を実現する.

また, Encoder と Decoder はどちらも 1 層の 1×1 畳み込みフィルタで構成され, ピクセル単位で線形変換を行うため, それぞれ l 層目の畳み込みフィルタと $l+1$ 層目のフィルタに線型結合できる. そのため, Pruning 後の CNN にとって, Pruning Block の計算コストは無視できる.

ネットワーク全体の補助変数 $\mathcal{M}$ と, Pruning Block の 1×1 畳み込みフィルタを含めた CNN 全体の重み $\mathcal{W}$ の更新は, 交互に行われる. $\mathcal{M}$ の更新式は

$$m^{(c,l)} \leftarrow m^{(c,l)} - \frac{\mu}{N} \sum_{n}^{N} \left(\frac{\partial \mathcal{L}\left(X_n, \mathcal{M}, \mathcal{W}\right)}{\partial m^{(c,l)}} \right) - \mu\lambda \frac{\partial \mathcal{R}\left(\mathcal{M}\right)}{\partial m^{(c,l)}} \tag{4.27}$$

である. ここで, $\mathcal{R}$ は

$$\mathcal{R}\left(\mathcal{M}\right) = \sum_{l}^{L} \sum_{c}^{C^{(l)}} \frac{h\left(m^{(c,l)}\right)}{C^{(l)}} \tag{4.28}$$

である. L はネットワークの畳み込み層の数, $C^{(l)}$ は l 層目の畳み込み層のチャネル数である. μ は変数の更新量をコントロールする学習係数である. 重み $\boldsymbol{w}^{(c,l)}$ の更新は, 一般的な更新式と同様に

$$\boldsymbol{w}^{(c,l)} \leftarrow \boldsymbol{w}^{(c,l)} - \frac{\mu}{N} \sum_{n}^{N} \left(\frac{\partial \mathcal{L}\left(X_n, \mathcal{M}, \boldsymbol{w}^{c,l}\right)}{\partial \boldsymbol{w}^{(c,l)}} \right) \tag{4.29}$$

である.

4.5 スコアリング関数を用いた Pruning 手法の有効性の検証

本節では，MNIST [60] と CIFAR-10/100 [61] のデータセット，及び画像分類を行う CNN を用いて, スコアリング関数を用いた Pruning 手法を評価する．MNIST は 6 万枚の学習画像と 10 桁のテスト画像を含み，CIFAR-10/100 は 5 万枚の学習画像と 10,000 枚のテスト画像を含む．それぞれのデータセットの詳細は 2.5 章に記載してある.

本実験では, CNN の畳み込みフィルタの剪定 (Pruning) を *1* つずつ 行う．すなわち，式 (4.1) で示される最も重要度の低いフィルタを CNN から一つ除去 (Pruning) し，学習サンプルを用いて, Pruning された CNN を再学習するという Pruning 手順を繰り返す.

その後，Pruning されたの CNN の分類精度と FLOPs の観点から, 各 Pruning 手法を評価する．以上のように各 Pruning 手法を適用することで，様々な FLOPs を示す Pruning 済み CNN が順次生成される．各 Pruning 手法の公平な比較のために, 指定した FLOPs を持つ CNN の分類精度を比較する．ここでは, 各 Pruning 手法による FLOPs 削減量が 0 %，50 %，70 %，90 % であるときの CNN の分類精度を比較する．FLOPs 削減量は, 例えば削減量が 90 % ならば, 元々の CNN に比べて, 計算コストが 10 分の 1 になったことを意味する.

4.5.1 Comparison of Taylor expansion terms

まず, 4.3.2 章で議論したように, 分類損失のテイラー展開の一次項と二次項を比較する予備実験を行う. Pruning 手法は，表 4.1 で詳しく説明している層の浅い LeNet5-like CNN に適用される．データセットは MNIST である．Pruning の対象となる CNN (初期モデル) は，学習係数 0.01 の SGD を用いて, 300 epochs 学習されることで生成される．そのときのミニバッチサイズは 100 である．各 Pruning 手法の性能評価は, 異なる初期値を持つ CNN を用いて 5 回行い, その結果の平均と標準偏差を比較することで行われる．表 4.2 に

表 4.1: LeNet5-like CNN.

Layer	Parameters
conv1	32 filters, 5×5, pad $= 2$, BatchNorm, ReLU
pool1	Max-pooling, 2×2, pad $= 0$
conv2	64 filters, 5×5, pad $= 2$, BatchNorm, ReLU
pool2	Max-Pooling, 2×2, pad $= 0$
FC1	1024 fully-connected filters, DropOut ($p = 0.5$)
FC2	10 fully-connected filters
output	Softmax

表 4.2: LeNet5-like CNN を Pruning したときの分類精度 (テスト精度 %). スコアリング関数は, それぞれ, テイラー展開の第一項, 第一項 + 第二項, 第二項である.

	FLOPs 削減量			
スコアリング関数	0 %	50%	70%	90%
$\mathrm{E}_n[-\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n]$	99.29±0.06	99.19±0.03	99.08±0.08	99.01±0.06
$\mathrm{E}_n[-\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n + \boldsymbol{x}_n^\top \boldsymbol{H}_n \boldsymbol{x}_n]$	99.29±0.06	**99.25**±0.02	**99.18**±0.03	98.97±0.05
$\mathrm{E}_n[\boldsymbol{x}_n^\top \boldsymbol{H}_n \boldsymbol{x}_n]$	99.29±0.06	99.23±0.05	99.18±0.05	**99.13**±0.06

示した評価結果は, Pruning におけるテイラー展開の第二項の有効性を示している. これは, 4.3.2 章で述べたように, スコアリング関数に Taylor 展開の二次項を用いる動機となる結果である.

4.5.2 Performance comparison

次に，提案手法 (式 (4.10)) とその亜種 (式 (4.12)) を，以下に列挙する他の Pruning 手法と比較する.

■Simple　最もシンプルな Feature-level Pruning 手法の一つは，特徴のノルムを考慮した

$$\mathtt{S}_{simple}(\{\boldsymbol{x}_n\}_{n=1}^N) = \mathrm{E}_n[\|\boldsymbol{x}_n\|_2^2] \tag{4.30}$$

をスコアリング関数として用いる手法である. これは, Li ら [74] のように, 特徴マップの L_2 ノルムに基づいている.

■Fisher　*Fisher Pruning* [22] は, 本章での提案手法 (式 (4.10), 式 (4.12)) と同様に, 分類損失のテイラー展開の枠組みでスコアリング関数を

$$\mathtt{S}_{fisher}(\{\boldsymbol{x}_n\}_{n=1}^N) = \mathrm{E}_n\left[\frac{1}{2}\boldsymbol{x}_n^\top \boldsymbol{H}_n \boldsymbol{x}_n\right] \tag{4.31}$$

と定式化する

■Oracle　*Oracle Pruning* [23] は, テイラー展開の一次項の絶対値から

$$\mathtt{S}_{oracle}(\{\boldsymbol{x}_n\}_{n=1}^N) = \left|\mathrm{E}_n\left[\boldsymbol{x}_n^\top \boldsymbol{\nabla}_n\right]\right|. \tag{4.32}$$

と導き出される.

表 4.3: VGG-13 like CNN.

Layer	Parameters
conv1	64 filters, 3×3, pad $= 1$, BatchNorm, ReLU
conv2	64 filters, 3×3, pad $= 1$, BatchNorm, ReLU
pool1	Max-pooling, 2×2, pad $= 0$
conv3	128 filters, 3×3, pad $= 1$, BatchNorm, ReLU
conv4	128 filters, 3×3, pad $= 1$, BatchNorm, ReLU
pool2	Max-pooling, 2×2, pad $= 0$
conv5	256 filters, 3×3, pad $= 1$, BatchNorm, ReLU
conv6	256 filters, 3×3, pad $= 1$, BatchNorm, ReLU
pool3	Max-pooling, 2×2, pad $= 0$
FC1	1024 fully-connected filters, DropOut ($p = 0.5$)
FC2	10 or 100 fully-connected filters
output	Softmax

■L1+stdv　Weight-level Pruning と Feature-level Pruning を比較するために, l 番目の層の c チャネル目の畳み込みフィルタの重みの L1 ノルムとそれらの標準偏差を用いてフィルタの重要度を評価するスコアリング関数 [87]

$$\mathrm{S}_{std+L1}(\{\boldsymbol{w}^{(c,l)}\}) = \lambda \frac{\sigma^{(c,l)}}{\sqrt{\sum_{c'}^{C} (\sigma^{(c',l)})^2}} + (1-\lambda) \left\| \boldsymbol{w}^{(c,l)} \right\|_1^1 \tag{4.33}$$

も比較する. ここで, $\boldsymbol{w}^{(c,l)}$ と $\sigma^{(c,l)}$ は, それぞれ畳み込みフィルタの重みとその標準偏差である. 最も良い性能が得られるように, ハイパーパラメータ λ を手動で調整する. このスコアリング関数は λ の調整が必要なのに対し, 本章で提案するスコアリング関数は, そのような調整が必要ではないパラメータフリーであることに注意してほしい.

各 Pruning 手法は, 層の浅い LeNet5-like CNN（表 4.1）に加えて, 表 4.3 で示した層の深い VGG-13 like CNN にも適用される. VGG-13 like の学習方法は 4.5.1 章で述べたものと同様であり, 3 回試行した結果の平均値と標準偏差を報告する.

4.5.3　MNIST dataset

データセットに MNIST, ネットワークに LeNet5-like (表 4.1) と VGG-13 like (表 4.3) を用いて, 各 Pruning 手法を比較する (表 4.4). MNIST の手書き数字画像は分類が容易であるため, VGG-13 like ではシンプルなスコアリング関数 (式 (4.30)) を除いてすべての手法が良好な性能を示している. シンプルなスコアリング関数は, シンプルな CNN では有効であるが (表 4.4 a), 複雑な CNN では性能が低下する (表 4.4 b). この結果は, 層が深くて複雑な CNN に対する Pruning には, 洗練されたスコアリング関数が必要であることを示唆している.

表 4.4: MNIST を用いた性能比較 (テストサンプルの分類精度 %).

(a) LeNet5-like CNN (表 4.1)

	FLOPs reduction			
Criterion	0 %	50%	70%	90%
L1+stdv (4.33)	99.29±0.06	99.28±0.07	99.22±0.06	98.91±0.07
Simple (4.30)	99.29±0.06	99.29±0.03	99.21±0.07	99.15±0.11
Fisher (4.31)	99.29±0.06	99.23±0.05	99.18±0.05	99.13±0.06
Oracle (4.32)	99.29±0.06	99.30±0.08	99.25±0.04	99.19±0.05
Ours (4.10)	99.29±0.06	99.25±0.06	99.19±0.07	99.13±0.04
Ours (4.12)	99.29±0.06	99.29±0.03	99.28±0.02	99.19±0.05

(b) VGG13-like CNN (表 4.3)

	FLOPs reduction			
Criterion	0 %	50%	70%	90%
L1+stdv (4.33)	99.55±0.01	99.61±0.02	99.62±0.07	99.59±0.04
Simple (4.30)	99.55±0.01	99.51±0.01	98.40±0.38	89.57±5.71
Fisher (4.31)	99.55±0.01	99.62±0.03	99.56±0.02	99.60±0.03
Oracle (4.32)	99.55±0.01	99.65±0.03	99.67±0.03	99.44±0.23
Ours (4.10)	99.55±0.01	99.63±0.04	99.64±0.02	99.61±0.04
Ours (4.12)	99.55±0.01	99.63±0.02	99.62±0.02	99.59±0.02

4.5.4 CIFAR datasets

さらに，前節で用いた MNIST よりも複雑な画像分類用データセットである CIFAR-10/100 を用いて, 各 Pruning 手法の性能を評価する．そのような複雑な分類問題に対応するために，ここでの実験では, 層が深い VGG-13 like (表 4.3) をネットワークに採用して, Pruning 手法の比較を行う (表 4.5).

MNIST の場合と同様に，式 (4.30) のシンプルな手法は，Pruning した CNN の分類性能の維持に失敗し，FLOPs を 90 % 削減すると分類性能が著しく低下したが，提案手法は良好な分類性能を維持した．Oracle Pruning [23] が Fisher Pruning [22] よりも優れた結果を示したのは，式 (4.7) で議論されているように，Oracle Pruning のスコアリング関数 (式 (4.32)) が Fisher Pruning のスコアリング関数 (式 (4.31)) の下界であることに由来する, スコアリング関数のロバスト性によるものと考えられる．これらの手法と比較しても，提案したスコアリング関数 (式 (4.10), 式 (4.12)) は良好な性能を示した．これらの手法がいずれも分類損失のテイラー展開に基づくものであることを考えると、この結果は, 4.3.2 章にてテイラー展開の二次項を解析して得られたロバスト性は Pruning において有効であることを示している．従来手法 (式 (4.31), 式 (4.32)) と L1+stdv を比較すると，従来手法は Weight-level Pruning の一つである L1+stdv よりも分類性能が低くなる場合がある．一方，提案するスコアリング関数 (式 (4.10), 式 (4.12)) は式 (4.33) の L1+stdv を上回る性能を示した．また，特徴マップの詳細な (二次の) 表現を用いた Pruning (式 (4.12)) は，CNN の分類性能をさらに向上させることに貢献しているかもしれない (表 4.5 では FLOPs を 90 % 削減できた) .

また, ResNet18-like (表 4.6) でも各 Pruning 手法を比較する．この実験でも, SGD を用いてネットワークを 300 epochs 学習することで, Pruning の対象となる初期モデルを生成する．SGD の初期の学習係数は 0.1

表 4.5: VGG13-like (表 4.3) と CIFAR-10/100 データセットを用いた性能比較（分類精度 %).

(a) CIFAR-10

	FLOPs 削減量			
スコアリング関数	0 %	50%	70%	90%
L1+stdv (4.33)	87.73±0.27	87.45±0.27	85.97±0.69	81.37±0.34
Simple (4.30)	87.73±0.27	85.02±1.03	72.96±1.54	34.80±6.49
Fisher (4.31)	87.73±0.27	87.44±0.22	85.81±0.42	74.46±0.81
Oracle (4.32)	87.73±0.27	87.72±0.70	86.12±0.20	81.30±0.46
Ours (4.10)	87.73±0.27	89.38±0.22	88.94±0.46	83.80±1.87
Ours (4.12)	87.73±0.27	89.02±0.24	88.86±0.12	86.24±0.33

(b) CIFAR-100

	FLOPs 削減量			
スコアリング関数	0 %	50%	70%	90%
L1+stdv (4.33)	65.75±0.23	61.67±1.15	57.41±0.41	46.57±1.52
Simple (4.30)	65.75±0.23	60.68±0.36	37.72±1.16	6.12±0.15
Fisher (4.31)	65.75±0.23	63.52±0.47	59.59±0.49	40.84±1.54
Oracle (4.32)	65.75±0.23	59.34±0.66	52.06±1.96	41.67±2.06
Ours (4.10)	65.75±0.23	65.62±0.04	63.98±0.02	49.39±0.04
Ours (4.12)	65.75±0.23	62.98±0.02	61.15±0.02	53.75±0.02

に設定し, ネットワークを 80 epochs, 120 epochs 学習した時に, その都度, 学習係数を 10 で割る．モーメンタム, 重み減衰パラメータ, ミニバッチサイズは, それぞれ 0.9, 0.0001, 100 に設定した．表 4.7 に示すように, 提案したスコアリング関数 (式 (4.10)) は他の手法と比較しても遜色のない性能を示した．一方，式 (4.12) で提案したスコアリング関数は，ネットワークの分類性能を維持することができなかった．これは，特徴マップ $\boldsymbol{x}_n$ が偶然高くなってしまうサンプルが原因である可能性がある．式 (4.12) の $\mathrm{E}_n[\boldsymbol{x}_n\boldsymbol{x}_n^\top]$ の対角線は特徴マップの二次形式のサンプル平均を取るので，サンプルによっては，式 (4.12) の Pruning Score が極端に大きくなる可能性がある．そのため，式 (4.12) は式 (4.10) よりもそのような極端な値の影響を受けやすくなる可能性がある．実際，我々の実験では，$\boldsymbol{x}_n$ の一部の要素の値が非常に大きい, 疎な特徴マップが生成されることがあった．

最後に, 4.3.3 章で述べられている正規化手法 (式 (4.13)) をスコアリング関数に適用し, その性能を評価する．その結果を表 4.8 に示す．正規化により, 提案したスコアリング関数の性能が向上したことがわかる．まとめると, 提案したスコアリング関数 (式 (4.10)) と L_1 正規化 (式 (4.13)) は, 様々なデータセットで安定して良好な性能を発揮することがわかる．一方, L_1 で正規化した式 (4.12) は, 分類性能が低下するケースがあった．CIFAR-10 の場合, (表 4.5 a) では, L_1 正規化を用いない場合に比べて, FLOPs を 90 % 削減した時の分類性能が悪化した．これは上述したような, 極端な値を持つサンプルが原因である可能性がある．このようなサンプルから得られる Pruning Score が大きくなると，正規化により特定のチャネルの Score が大きくなり，同じ層に属する他のチャネルの Score 計算に悪影響を与えてしまう．これらのことから，勾配と特徴マップ, それぞれを平均したものを用いるスコアリング関数 (式 (4.10)) は, 式 (4.12) と比較しても, 安定して良好な性能を示すと結論づけることができる．

表 4.6: ResNet18-like. ( ) × 3 は, 積み重ねられた Residual block の数を表す. 各 Residual block は, ショートカット層として 1 × 1 畳み込み層を有する. ダウンサンプリングは, conv2_1 と conv3_1 の stride を 2 に設定することで行われる.

Layer	Parameters
conv0	16 filters, 3×3, pad=1, BatchNorm, ReLU
conv1_x	$\begin{pmatrix} \text{16 filters, } 3 \times 3, \text{ pad=1, BatchNorm, ReLU} \\ \text{16 filters, } 3 \times 3, \text{ pad=1, BatchNorm, ReLU} \end{pmatrix} \times 3$
conv2_x	$\begin{pmatrix} \text{32 filters, } 3 \times 3, \text{ pad=1, BatchNorm, ReLU} \\ \text{32 filters, } 3 \times 3, \text{ pad=1, BatchNorm, ReLU} \end{pmatrix} \times 3$
conv3_x	$\begin{pmatrix} \text{64 filters, } 3 \times 3, \text{ pad=1, BatchNorm, ReLU} \\ \text{64 filters, } 3 \times 3, \text{ pad=1, BatchNorm, ReLU} \end{pmatrix} \times 3$
pool	Average-pooling, 8×8, pad = 0
FC	10 fully-connected filters
output	Softmax

表 4.7: ResNet18 と CIFAR-10 データセットを用いた性能比較 (分類精度 %) .

	FLOPs 削減量			
スコアリング関数	0 %	50%	70%	90%
L1+stdv (4.33)	89.17±1.36	84.80±1.06	82.37±1.43	74.15±1.56
Simple (4.30)	89.17±1.36	82.24±0.26	76.36±1.56	52.88±10.93
Fisher (4.31)	89.17±1.36	86.80±0.31	84.93±0.37	79.17±1.0
Oracle (4.32)	89.17±1.36	85.73±0.05	85.03±0.44	78.29±0.24
Ours (4.10)	89.17±1.36	86.78±0.78	83.17±0.20	77.99±1.11
Ours (4.12)	89.17±1.36	82.24±2.11	78.80±2.71	62.69±9.04

表 4.8: 正規化 (4.13) を用いた Pruning 手法の性能結果 (分類精度 %). データセットは CIFAR-10/100, ネットワークは大規模な CNN(Table 4.3).

(a) CIFAR-10

	FLOPs 削減量			
スコアリング関数	0 %	50%	70%	90%
Oracle (4.32) + L_1	87.73±0.27	89.70±0.29	89.26±0.22	85.77±0.49
Ours (4.10) + L_1	87.73±0.27	89.62±0.13	89.15±0.11	86.52±0.42
Ours (4.10) + $L_{\frac{2}{3}}$	87.73±0.27	89.53±0.20	89.06±0.44	86.04±0.38
Ours (4.10) + $L_{\frac{1}{2}}$	87.73±0.27	89.25±0.13	89.00±0.45	86.09±0.42
Ours (4.12) + L_1	87.73±0.27	88.74±0.21	87.27±0.30	83.80±0.09

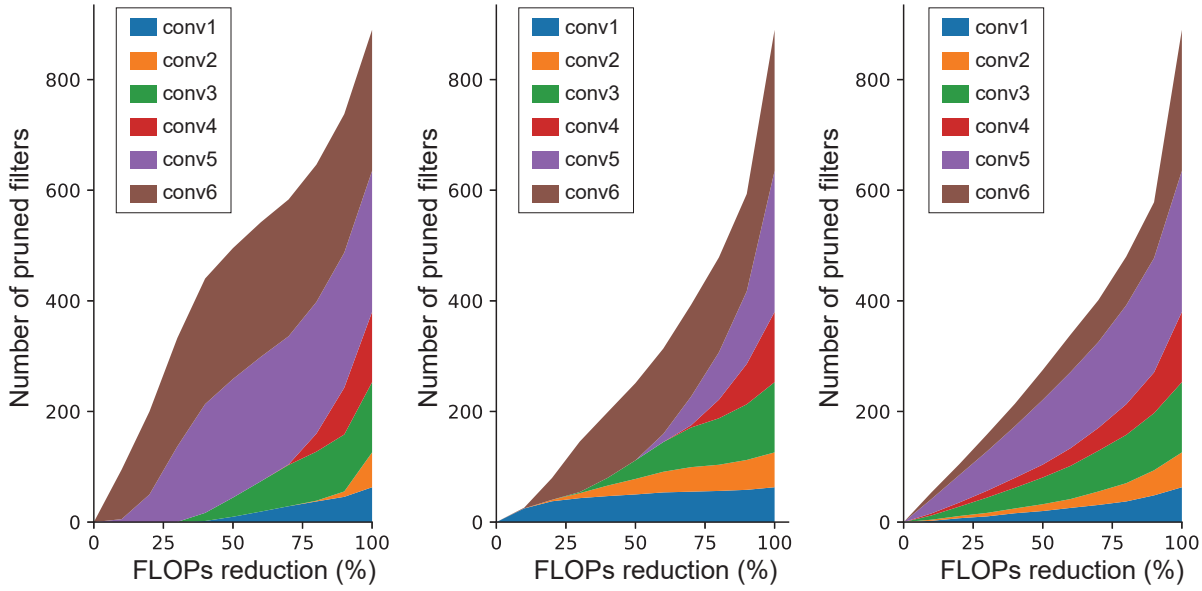
(b) CIFAR-100

	FLOPs 削減量			
スコアリング関数	0 %	50%	70%	90%
Oracle (4.32) + L_1	65.75±0.23	65.66±0.34	63.80±0.62	54.56±1.23
Ours (4.10) + L_1	65.75±0.23	66.03±0.25	64.26±0.27	55.79±1.23
Ours (4.10) + $L_{\frac{2}{3}}$	65.75±0.23	65.77±0.23	64.45±0.47	56.38±1.27
Ours (4.10) + $L_{\frac{1}{2}}$	65.75±0.23	66.18±0.13	64.25±0.15	56.38±0.92
Ours (4.12) + L_1	65.75±0.23	64.60±0.19	62.37±0.83	55.12±0.24

4.6 スコアリング関数の挙動の解析

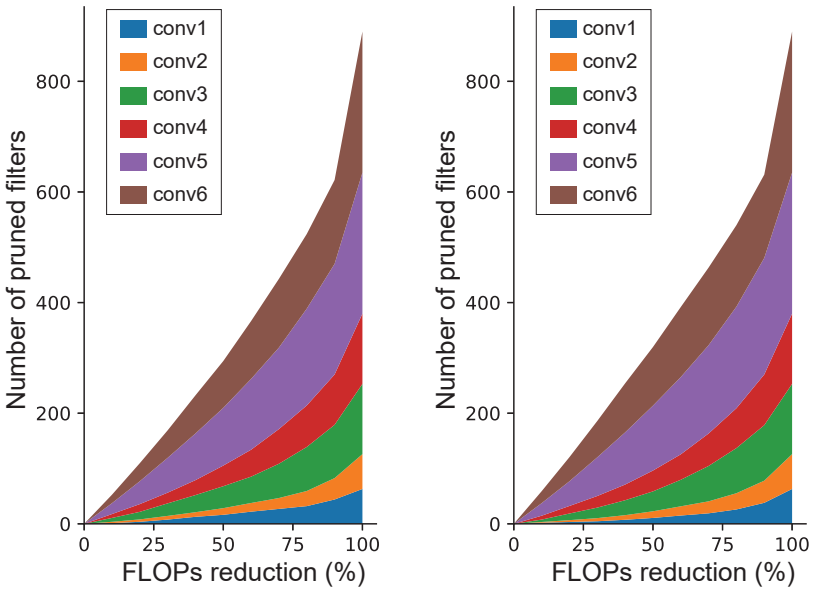
つづいて，スコアリング関数を用いた各 Pruning 手法のネットワーク上での挙動を解析する．そのために，各 Pruning 手法が, どの畳み込み層からどのようなフィルタを（1 つずつ）取り除いているのかを明らかにする．図 4.4 は, FLOPs を削減するために, 各 Pruning 手法が各畳み込み層から取り除いたフィルタの数を示している (表 4.5 のネットワーク). 図 4.4 から，従来手法は，フィルタを取り除く層に大きな偏りがあることがわかる．例えば, Fisher Pruning [22] は conv5 や conv6 のような深い層のフィルタを先に Pruning してから浅い層のフィルタを Pruning するのに対し，Oracle Pruning [23] は conv1 や conv6 のフィルタを先に Pruning してから他の層のフィルタを Pruning している．それに対して，提案手法 (式 (4.10)) は，層の偏りが無く，各層から畳み込みフィルタを均等に Pruning する．このような Pruning の挙動が, 他の手法に比べて, CNN の分類性能を比較的維持できた鍵であると考えられる．実際に，Pruning 手法の性能をさらに向上させた正規化手法 (式 (4.13)) を用いると，図 4.5 に示すように，Oracle Pruning [23] も層の偏りを緩和している．この解析結果は, 層の違いに左右されない Pruning 手法を研究することの重要性を示唆しているのかもしれない.

さらに, Pruning された畳み込みフィルタが出力していた特徴マップやその勾配を解析することで, Pruning される畳み込みフィルタの傾向も解析する. 図 4.6 は, FLOPs を削減するために, 各 Pruning 手法によって取り除かれたフィルタが出力していた特徴マップの要素の平均を示している. 図 4.6 から, 提案手法は, 従来手法に比べて, 特徴マップの平均の値が比較的小さい畳み込みフィルタを取り除いているものの, 必ずしも, 平均の値が小さい畳み込みフィルタを優先的に取り除いているわけではないことがわかる. この結果により, 提案手法は, 特徴マップの平均の値の大小関係以外も考慮していることが伺える. さらに, Pruning された特徴マップに関する勾配の平均を示した図 4.7, 図 4.8 を見ると, 従来手法では, FLOPs を削減するにつれて勾配の値が大きくなっていくのに対し, 提案手法では, FLOPs 削減量と勾配の値の大小関係の間に相関があまり見られない. このことも, 提案手法は, 単なる値の大小関係以上の情報を考慮していることを示唆している.



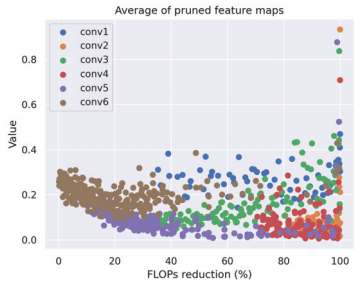
Fisher pruning (式 (4.31)) [22]　Oracle Pruning (式 (4.32)) [23]　提案手法 (式 (4.10))

図 4.4: 畳み込みフィルタを一枚ごとに取り除く Pruning 処理により取り除かれた畳み込みフィルタの数を層ごとにカウントしたもの．横軸は FLOPs 削減量, 縦軸は Pruning されたフィルターの量を示す.

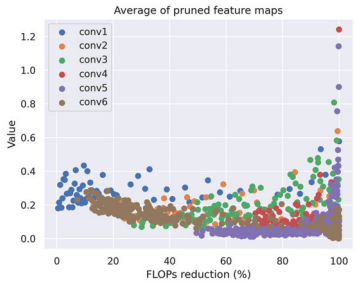


Oracle Pruning (式 (4.32)) + L_1-norm [22]　提案手法 (式 (4.10)) + L_1-norm

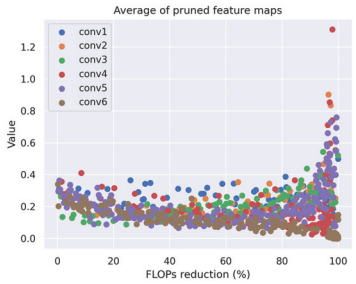
図 4.5: 正規化手法 (式 (4.13)) を用いた Pruning 手法によって Pruning された畳み込みフィルタの数.



Fisher pruning (式 (4.31))
[22]

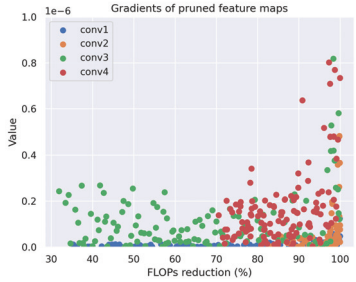


Oracle Pruning (式 (4.32))
[23]

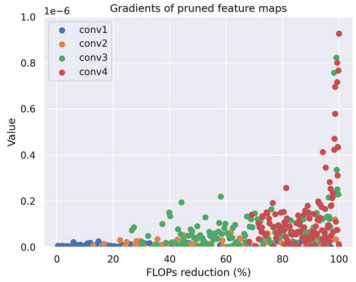


提案手法 (式 (4.10))

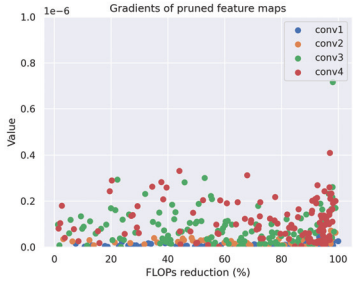
図 4.6: 各 Pruning 手法によって取り除かれた特徴マップの平均．横軸は FLOPs 削減量, 縦軸は Pruning された特徴マップの要素の平均を示す．



Fisher pruning (式 (4.31))
[22]

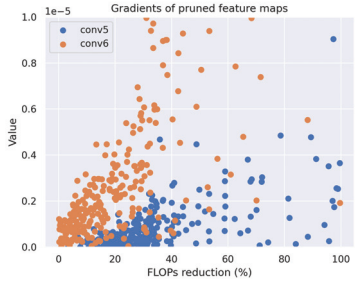


Oracle Pruning (式 (4.32))
[23]

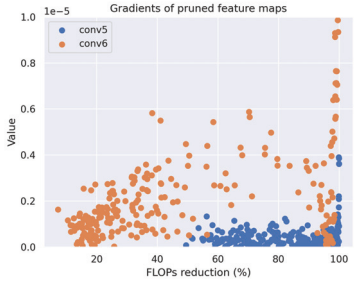


提案手法 (式 (4.10))

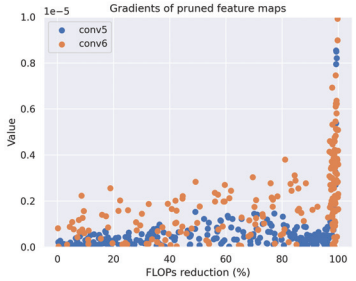
図 4.7: 各 Pruning 手法によって取り除かれた特徴マップに関する勾配の図 (conv1, conv2, conv3, conv4)．横軸は FLOPs 削減量, 縦軸は Pruning されたフィルタに関する勾配の値を示す．



Fisher pruning (式 (4.31))
[22]



Oracle Pruning (式 (4.32))
[23]



提案手法 (式 (4.10))

図 4.8: 各 Pruning 手法によって取り除かれた特徴マップに関する勾配の図 (conv5, conv6)．横軸は FLOPs 削減量, 縦軸は Pruning されたフィルタに関する勾配の値を示す．

4.7　補助変数を用いる Pruning 手法の有効性の検証

次に, 4.4 章で提案した, 補助変数を用いる Pruning 手法の有効性を検証する. ここでの実験では, MNIST, CIFAR-10/100 だけでなく, STL-10 [88] もデータセットとして用いる. 図 4.9 に示すように, STL-10 は, 10 クラスの物体画像からなるデータセットである. 画像の枚数は, 学習用画像が 5,000 枚, テスト画像が 8,000 枚である. 画像サイズは 96×96 である.

Pruning の手順は以下の通りである. AutoPrune [24] による Pruning 手順は, 既存の（事前に学習した）CNN に補助変数と指標関数を導入し, 補助変数と CNN の各種変数を交互に学習したのち, 学習終了後の CNN がそのまま Pruning 済み CNN となる. それに対し, Pruning Block を用いる場合, まず, 4.4.3 章で示した通り, 事前に学習した CNN に Pruning Block を導入し, 補助変数と CNN の各種変数を交互に学習する. 学習終了後, 閾値以下の補助変数に対応する畳み込みフィルタを取り除いた CNN を, 最終的な Pruning 済み CNN とする. 具体的な閾値の値や学習回数 (epochs) は, 各実験で, その都度表記する. なお, 連続関数としてシグモイド関数を用いる手法では, その傾きをコントロールするハイパーパラメータ α を全て 50 に設定する. Pruning Block を用いる提案手法の場合, Encoder の出力チャネル数 C' の値は, 導入された畳み込み層の出力チャネル数と同じ値に設定する.

4.5 章と同様に, 様々なデータセットや CNN を用いて, 分類性能と FLOPs の観点から各 Pruning 手法を評価する. このとき, MNIST, CIFAR をデータセットとする実験では, 4.5 章のネットワークを再び用いる. それに対し, STL-10 を入力とする実験では, 画像サイズが CIFAR より 3 倍大きいため, いくつか条件を変更する. STL-10 をデータセットとする VGG13-like 2 (表 4.9) は, VGG13-like（表 4.3）の各畳み込み層の次の層に Max-pooling を追加し, さらに FC1 を取り除いたネットワークである. ResNet18-like は, 4.5 章のネットワーク (表 4.6) と同じ構造である. また, これまでは入力画像のピクセル値を 0 から 1 に範囲に正規化していたが, STL-10 の画像では, -1 から 1 の範囲に正規化する. 各実験では, 3 回試行した結果の平均値と標準偏差を報告する.

図 4.9: STL-10 のサンプル画像. 10 クラスの物体画像で構成される.

4.7.1　MNIST における結果

ここでは, MNIST の画像を分類する LeNet5-like (表 4.1) に対して各 Pruning 手法を適用することで, それぞれの性能を評価する. AutoPrune [24] は, 補助変数の最適化とネットワークの学習をどちらも 400 epochs 行ない, それによって得られたネットワークを Pruning 済みネットワークとする. 補助変数やネットワークの

表 4.9: VGG-13-like CNN 2. STL-10 の分類に用いられる.

Layer	Parameters
conv1	64 filters, 3×3, pad = 1, BatchNorm, ReLU
pool1	Max-pooling, 2×2, pad = 0
conv2	64 filters, 3×3, pad = 1, BatchNorm, ReLU
pool2	Max-pooling, 2×2, pad = 0
conv3	128 filters, 3×3, pad = 1, BatchNorm, ReLU
pool3	Max-pooling, 2×2, pad = 0
conv4	128 filters, 3×3, pad = 1, BatchNorm, ReLU
pool4	Max-pooling, 2×2, pad = 0
conv5	256 filters, 3×3, pad = 1, BatchNorm, ReLU
pool5	Max-pooling, 2×2, pad = 0
conv6	256 filters, 3×3, pad = 1, BatchNorm, ReLU
pool6	Max-pooling, 2×2, pad = 0
FC	10 or 100 fully-connected filters
output	Softmax

表 4.10: MNIST の性能比較（テストサンプルの分類精度）.

				FLOPs reduction	
Dataset	Network	Method	閾値	分類精度 (%)	FLOPs 削減量 (%)
MNIST	LeNet-5 like (表 4.1)	Baseline	-	99.31 ±0.07	0
		AutoPrune	-	98.17 ±0.71	**97.01** ±0.15
		Sigmoid	0.001	99.09 ±0.045	81.63 ±2.70
		Ours	0.001	**99.24** ±0.062	91.46 ±0.72

最適化手法は, SGD である. そのときの補助変数の学習係数は 0.015, ネットワークの学習係数は 0.01 に設定する. バッチサイズは 100, フィルタの削減量をコントロールするハイパーパラメータ λ は 0.05 とする. それに対し, AutoPrune のステップ関数をシグモイド関数に置き換えた手法 (Sigmoid) や提案手法 (Ours) は, AutoPrune と同様の条件で学習を行なったのち, 値が閾値以下の補助変数, および, それらと対応するフィルタを Pruning したネットワークを, Pruning 済みネットワークとする.

それらの性能を評価した結果が, 表 4.10 である. 表中の Baseline は, 事前学習された, Pruning 前のネットワークの分類精度である. 表 4.10 に示すように, 提案手法は, ステップ関数を用いた従来手法 (AutoPrune) と比較すると, FLOPs 削減量は約 5 % 劣るものの, 分類精度ではやや上回ったことで, ほぼ同等の Pruning 性能を示している. さらに, 分類精度の標準偏差も, AutoPrune に比べ提案手法の方が小さいことから, より安定して Pruning を行えたことを示している. また, Sigmoid と提案手法を比較すると, 分類精度, FLOPs 削減量の両方の観点で, 提案手法の方が高い性能を示した. これは, Pruning Block の導入により, 補助変数の連続最適化と, Batch Normalization の正則化の相性の悪さを緩和したことを示唆している.

表 4.11: CIFAR-10/100 の性能比較（テストサンプルの分類精度）.

Dataset	Network	Method	閾値	分類精度 (%)	FLOPs reduction FLOPs 削減量 (%)
CIFAR-10	VGG-13 like (表 4.3)	Baseline	-	87.73 ±0.27	0
		AutoPrune	-	82.03 ±1.25	84.95 ±0.90
		Sigmoid	0.001	76.90 ±3.92	85.43 ±1.91
		Ours	0.001	**85.14** ±0.75	**90.66** ±0.39
	ResNet-18 like (表 4.6)	Baseline	-	89.17 ±1.36	0
		AutoPrune	-	82.80 ±2.26	**46.72** ±3.67
		Sigmoid	0.1	86.62 ±0.38	35.72 ±2.50
		Ours	0.1	**86.99** ±0.29	46.11 ±2.74
CIFAR-100	VGG-13 like (表 4.3)	Baseline	-	65.75 ±0.23	0
		AutoPrune	-	54.18 ±1.69	56.17 ±0.62
		Sigmoid	0.001	58.57 ±1.50	74.22 ±0.59
		Ours	0.001	**62.21** ±0.54	**82.62** ±1.04
	ResNet-18 like (表 4.6)	Baseline	-	62.53 ±0.20	0
		AutoPrune	-	**58.70** ±1.15	**30.28** ±2.30
		Sigmoid	0.1	54.92 ±0.65	29.47 ±2.00
		Ours	0.1	58.50 ±0.33	28.93 ±3.55

4.7.2 CIFAR における結果

つづいて, CIFAR-10/CIFAR100 の画像を分類する VGG-13 like (表 4.3) および ResNet-18 like (表 4.6) を用いて, 各 Pruning 手法を比較する. AutoPrune [24] は, 補助変数の最適化と VGG13 の学習をどちらも 1,000 epochs 行ない, それによって得られたネットワークを Pruning 済みネットワークとする. 補助変数やネットワークの最適化手法は SGD である. そのときの補助変数の学習係数は 0.001, ネットワークの学習係数は 0.01 に設定する. フィルタの削減量をコントロールするハイパーパラメータ λ は, 畳み込み層のものは 0.5, FC 層のものは 0.05 とする. また, 重み減衰の係数, バッチサイズと Dropout の確率を, それぞれ 0.0005, 100, 0.01 に設定する. それに対し, Sigmoid や Ours は, 補助変数の最適化とネットワークの学習を 200 epochs 行なう. Dropout を conv1, conv3, conv5, pool3, FC1 に導入し, それぞれの確率を 0.3, 0.4, 0.4, 0.5, 0.5 に設定する. そして, 補助変数の学習係数を 0.1 に設定した以外は, AutoPrune と同じ実験条件である. 補助変数の値が発散するのを防ぐために, これ以降の実験では, 補助変数の値が一定以上の大きさになる場合, それ以上大きな値にならないように値のクリッピングを行う.

表 4.11 から, VGG-13 like における比較では, FLOPs 削減量と分類精度の両方の観点で, 提案手法が最も優れた性能を達成した. CIFAR-10 での比較では, 提案手法は, AutoPrune に比べて FLOPs を約 5 % 削減したにも関わらず, 分類精度では約 3 % 上回った. CIFAR-100 での比較はより顕著な結果となり, 提案手法は AutoPrune より 20 % 以上 FLOPs を削減したうえで, 8 % 以上高い分類精度を達成した. また, AutoPrune は, CIFAR-10 では提案手法に次ぐ分類性能を示した一方で, CIFAR-100 では, 分類精度と FLOPs 削減量の両方の観点で最も低い性能を示した. これは, 複雑な画像分類では, 離散最適化問題を解くのがさらに難しくな

るからだと考えられる.

一方, ResNet-18 (表 4.6) における比較でも, 4.5 章のものを再び用いる. AutoPrune [24] は, 補助変数の最適化とネットワークの学習をどちらも 200 epochs 行ない, それによって得られたネットワークを Pruning 済みネットワークとする. 補助変数やネットワークの最適化手法は SGD である. そのときの補助変数の学習係数は 0.015, ネットワークの学習係数は 0.01 に設定する. 補助変数を最適化するときのモーメンタム項の係数は, 0.9 にする. フィルタの削減量をコントロールするハイパーパラメータ λ は, 畳み込み層のものは 0.5, FC 層ものは 0.05 とする. また, バッチサイズは 100 に設定する. それに対し, Sigmoid や Ours は, モーメンタムの係数を 0 に, 補助変数の学習係数を 0.1 に設定した以外は, AutoPrune と同様の実験条件で行う.

ここでも, 提案手法は AutoPrune と同等以上の性能を達成した. CIFAR-10 での比較では, 両手法の FLOPs 削減量は同等であるにも関わらず, 提案手法は 4 % 以上高い分類精度を達成した. また, CIFAR-100 では, 提案手法と AutoPrune の性能は同等であった. それに対し, Sigmoid は, CIFAR-10 および CIFAR-100 のどちらの条件であっても, 提案手法より低い分類精度を示した. これらは, 補助変数の離散最適化問題をそのまま連続最適化に拡張するだけでは Pruning 手法としての性能悪化を招くだけであり, 本章で提案する Pruning Block を用いることで, 初めてその性能が改善されることを意味している.

4.7.3 STL-10 における結果

さらに, STL-10 でも比較実験を行う. ただし, CIFAR-10/100 に比べ STL-10 の画像サイズは 3 倍大きいため, VGG-13 like (表 4.3) の代わりに VGG-13 like 2 (表 4.9) を用いる. ネットワークの事前学習は, 学習係数を 0.1, モーメンタム項の係数を 0.9 に設定した SGD を用いて行われる. 学習回数とバッチサイズは, それぞれ 400 epochs と 100 である. 事前学習後, AutoPrune による最適化と学習は, 250 epochs 行う. それ以外の条件は, Sigmoid や提案手法も含めて, 4.7.2 章と同様である. ResNet-18 は, ネットワーク構造と学習条件のどちらも, 4.7.2 章の ResNet-18 (表 4.6) と同様である.

表 4.12 から, どちらのネットワークでも提案手法が優れた性能を示した. VGG-13 like 2 では, AutoPrune よりも提案手法の方が FLOPs を 3 % 多く削減したにもかかわらず, 分類精度は約 0.4 % 高かった. ResNet-18 に至っては, 分類精度と FLOPs 削減量, 共に AutoPrune より大幅に優れた結果となった. これは, CIFAR-100 の場合と同様に, 補助変数の離散最適化問題を解くのが難しくなるからだと思われる (STL-10 は, 画像サイズが大きいにもかかわらず, 学習サンプルが少ない).

表 4.12: STL-10 の性能比較（テストサンプルの分類精度 %）.

Dataset	Network	Method	閾値	分類精度 (%)	FLOPs reduction FLOPs 削減量 (%)
STL-10	VGG-13 like 2 (Table 4.9)	Baseline	-	70.03 ±0.44	0
		AutoPrune	-	60.86 ±0.95	85.60 ±0.42
		Sigmoid	0.1	66.49 ±1.70	18.74 ±3.86
		Ours	0.001	**61.29** ±1.83	**87.83** ±0.93
	ResNet-18 like (Table 4.6)	Baseline	-	75.00 ±0.50	0
		AutoPrune	-	61.21 ±3.53	32.01 ±5.04
		Sigmoid	0.1	69.62 ±0.47	30.79 ±7.47
		Ours	0.1	**71.40** ±1.12	**52.23** ±6.82

表 4.13: 事前学習をしなかった時の CIFAR-10/100 の性能比較（テストサンプルの分類精度 %）.

Dataset	Network	Method	閾値	分類精度 (%)	FLOPs reduction FLOPs 削減量 (%)
CIFAR-10	VGG-13 like	Baseline	-	87.73 ±0.27	0
	(Table 4.5)	AutoPrune	-	80.96 ±1.39	84.87 ±0.23
		Ours	0.001	**83.17** ±0.33	**95.22** ±0.48
	ResNet-18 like	Baseline	-	89.17 ±1.36	0
	(Table 4.6)	AutoPrune	-	**68.24** ±8.23	69.20 ±1.01
		Ours	0.001	68.12 ±0.16	**91.97** ±0.74
CIFAR-100	VGG-13 like	Baseline	-	65.75 ±0.23	0
	(Table 4.5)	AutoPrune	-	49.85 ±1.65	61.71 ±0.47
		Ours	0.001	**56.11** ±1.24	**93.15** ±0.85
	ResNet-18 like	Baseline	-	62.53 ±0.20	0
	(Table 4.6)	AutoPrune (step)	-	**40.01** ±2.73	64.24 ±4.04
		Ours	0.001	18.68 ±1.77	**97.55** ±0.28

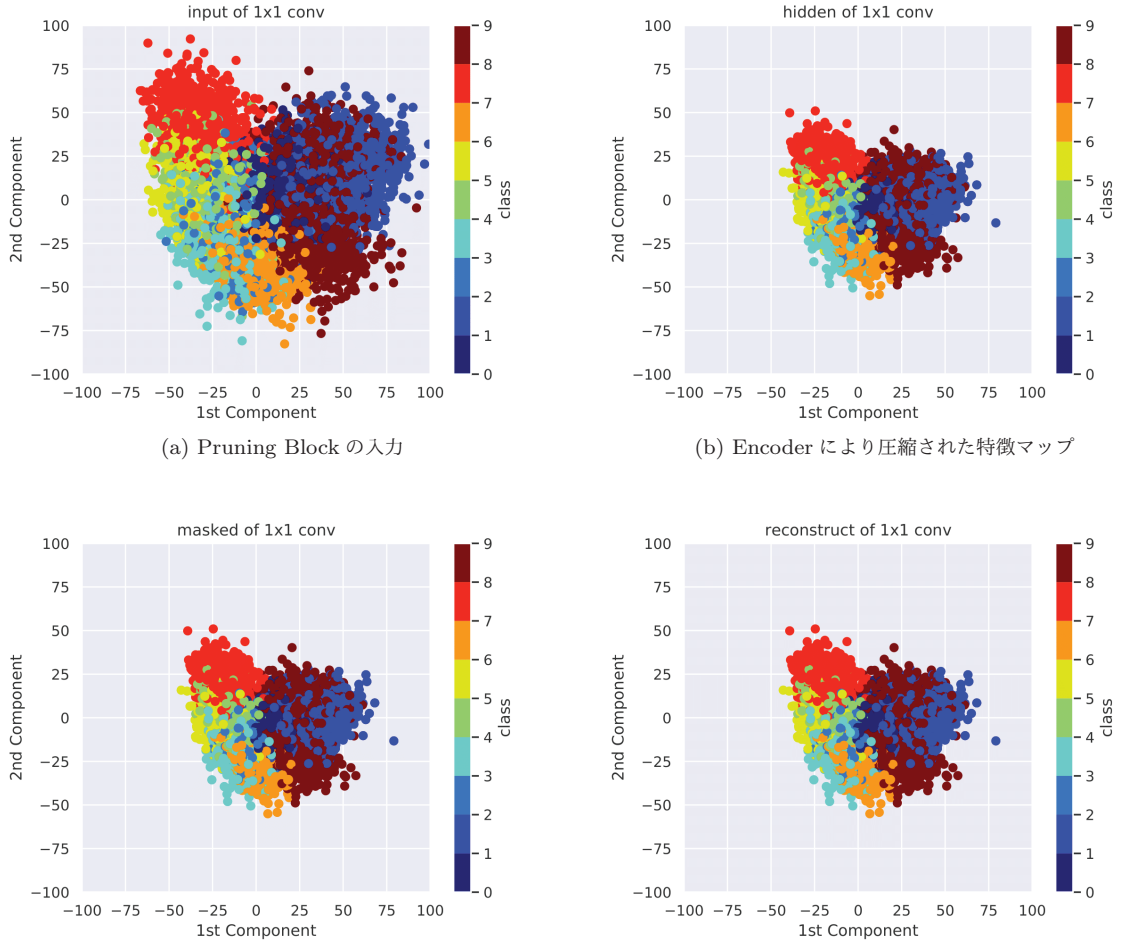
4.7.4 事前学習なしの場合

これまでは事前学習を行ったネットワークに対する Pruning 手法の有効性を検証してきたが, ここでは, 事前学習なしでネットワークの学習と補助変数の最適化を同時に行った場合の各 Pruning 手法の性能を検証する. 事前学習を行わないだけで, それ以外の実験条件はこれまでと同様である.

表 4.13 に示すように, 事前学習なしでは分類精度が大きく悪化するケースが多い. それでも, 多くのネットワークにて, 提案手法は AutoPrune と同等以上の分類精度と FLOPs 削減量を示している. しかし, これまでの実験結果 (表 4.11, 表 4.12) から, 全ての Pruning 手法において, CNN の事前学習を行った方が高い分類精度を達成できることがわかる. これは, CNN の学習が始まった頃の各畳み込みフィルタはほぼランダムであるため, その段階で補助変数の最適化問題を解いても正しい解が得られないにも関わらず, その解をもとに Pruning を行っているからだと思われる.

4.8 Pruning Block の挙動の解析

次に, Pruning Block が CNN 上でどのように機能するかを解析する. そのために, Pruning 後の CNN から得られる特徴マップに対して主成分分析 (PCA) を適用し, 2 次元に圧縮したものを可視化する. 可視化は, 学習サンプルから得られる特徴マップを用いて 2 次元の主成分空間を作り, テストサンプルから得られる特徴マップをその空間上に射影することで行う. 図 4.10 は, CIFAR-10 をデータセットとし, VGG-13 like の conv6 (最後の畳み込み層) に導入した Pruning Block から得られた特徴マップを可視化したものである. これらの図を見ると, Pruning Block は, 入力された特徴マップの情報をかなり保持した上で, 特徴マップの圧縮と再構成を行なっていることがわかる. 特に, Encoder が圧縮した特徴マップと, それ以後の特徴マップはほぼ同じものである. これは, 補助変数の最適化とネットワークの重みの学習を交互に行なったことで, Pruning に適した特徴マップを獲得できたことを示唆している.



(a) Pruning Block の入力

(b) Encoder により圧縮された特徴マップ

(c) 連続関数適用後の特徴マップ

(d) Decoder で再構成された特徴マップ

図 4.10: Pruning Block (conv6) の各特徴マップを可視化.

4.9 補助変数の連続最適化と Batch Normalization による正規化の相性を検証

補助変数の連続最適化と Batch Normalization による正規化の相性を検証するために, 正規化の効果を制限した場合の各手法の性能を比較する. 実験では, これまでの実験で用いられた, Batch Normalization を導入して事前学習した CNN を用いる. CNN の各種変数と補助変数の学習はこれまで通り行うが, 特徴マップの正規化は, 事前学習のときに獲得した平均・分散を定数として再利用することで行われる. 平均・分散を再計算しないので, Batch Normalization による正規化の効果は限定的である. 学習終了後, これまでと同様に Pruning を行う. その実験結果を表 4.14 に示す.

表 4.14 から, 正規化を制限することで, 連続最適化問題を解く Sigmoid は, 離散最適化問題を解く AutoPrune と同等以上の性能を示した. 特に, CIFAR-10 では, AutoPrune は最適化に失敗したにも関わらず, Sigmoid は提案手法とほぼ同等の性能を示した. AutoPrune の最適化が失敗したのは, 正規化の制限

により, 層の深い CNN の学習が不安定になったことで, もともと不安定だった離散最適化問題もさらに不安定になったからであると考えられる. それに対し, Sigmoid は, 正規化を制限したことで, 補助変数や連続関数の値の大小関係とフィルタの重要度が対応するようになったため, 閾値調整による Pruning が行いやすくなったからであると考えられる. また, CIFAR-100 では, 提案手法でさえも最適化が失敗する場合があった.

これらの結果から, Batch Normalization による正規化は, 補助変数の連続最適化と相性が悪く, その正規化を制限することで相性の悪さを緩和できることがわかった. しかし, 正規化の恩恵もあまり受けられないため, Pruning が不安定になる場合があることがわかる. これまでの実験結果も踏まえると, 正規化を制限せずに提案手法を用いる場合が, 最も安定して好ましい性能を達成できることがわかった.

表 4.14: Batch Normalization なしの性能比較（テストサンプル）.

					FLOPs reduction
Dataset	Network	Method	閾値	分類精度 (%)	FLOPs 削減量 (%)
MNIST	LeNet-5 like	Baseline	-	99.31 ±0.07	0
	(表 4.1)	AutoPrune (step)	-	99.03 ±0.08	**96.94** ±0.45
		AutoPrune (sigmoid)	0.1	99.14 ±0.02	90.17 ±0.25
		Ours	0.1	**99.16** ±0.09	94.93 ±0.65
CIFAR-10	VGG-13 like	Baseline	-	87.73 ±0.27	0
	(Table 4.5)	AutoPrune (step)	-	-	-
		AutoPrune (sigmoid)	0.001	86.34 ±0.68	79.41 ±2.47
		Ours	0.001	**87.31** ±0.19	**81.84** ±1.24
CIFAR-100	VGG-13 like	Baseline	-	65.75 ±0.23	0
	(Table 4.5)	AutoPrune (step)	-	44.29 ±2.08	52.55 ±19.35
		AutoPrune (sigmoid)	0.001	63.41 ±1.30	68.48 ±4.08
		Ours [a]	0.001	63.48 ±0.11	75.32 ±1.93

[a] 3 回試行のうち, 1 回は最適化に失敗したため, 2 回試行の平均と標準偏差を記載.

4.10 本章のまとめ

本章では, 経験的分類損失のテイラー展開をスコアリング関数として用いる新たな Pruning 手法, および, Pruning 手法が解くべき問題を補助変数に関する最適化問題に置き換え, それを安定して解く新たな Pruning 手法の 2 つを提案した.

学習サンプルを用いて各畳み込みフィルタの重要度を評価するスコアリング関数は, 従来のものだと, 極端な値を出力するサンプルに対して脆弱な場合があったが, 本章で提案したスコアリング関数は, そのようなサンプルに対するロバスト性を向上させることができた. また, そのスコアリング関数の導出過程で, 類似した関数を統一的に結びつけることもできた. metric の観点から, 本手法は, Pruning Score を計算するための安定した基準を提供するとともに, 畳み込みフィルタが取り除かれる層の偏りを緩和する正規化手法も提案した.

補助変数の最適化問題を安定して解く新たな Pruning 手法は, 従来手法が行なっていた補助変数の離散最適化問題を連続最適化問題に拡張し, さらに特徴マップの情報を圧縮した空間上でその最適化問題を解くことで, 類似した既存手法に比べ安定した Pruning を達成した. また, 補助変数の連続最適化問題と, Batch Normalization による特徴マップの正規化は相性がよくないことも確認した. 正規化との相性の問題は, その

効果を制限することである程度緩和できるが, 提案手法は, そのようなことをしなくても相性の問題を緩和できることも確認した.

CNN を用いた画像分類の実験では，どちらの提案手法も, 他の類似した Pruning 手法と比較して良好な性能を示した．これらのことから, CNN が獲得した特徴マップは, CNN の計算コスト削減に活用できることを示せた.

これらの結果は, 単に特徴マップの情報を活用することの重要性を示しただけでなく, ブラックボックスである CNN 内部の挙動を解析するのに役立つと考えられる.

5 結論

本章では, 本論文で得られた知見, および, 残された課題について述べる. 脳の視覚野を模倣して作られた畳み込みニューラルネットワーク (CNN) の本質は, 画像などの入力データに内在する重要な情報を捉えた情報表現 (特徴) をデータからの学習によって獲得することであるという認識から, CNN が獲得すべき特徴について議論・解析し, そのような特徴の獲得を促進する正則化手法や, 獲得した特徴を活用する Pruning 手法を提案した. その結果を要約すると, 以下の通りになる.

第 2 章では, CNN の中間層で獲得される特徴マップに対してスパース正則化を適用することで, CNN の分類精度が改善されることを確認した. これは, 特徴マップに対してスパース正則化を適用すると各ニューロンの出力が 0 に近づき, その値が必要以上に大きくならなくなったことで, ネットワークの学習が安定化したからである. 他にも, Batch Normalization の入力に対してスパース正則化を適用することで, Batch Normalization 以上の分類精度を達成できることも示した. また, 学習サンプルが極端に偏り, ネットワークの学習がうまくいかない条件下でも, スパース正則化ならばその悪影響を受けづらいことも示した.

第 3 章では, CNN の分類器を構成する各ニューロンにとって理想的な特徴について議論し, その特徴の獲得を加速する判別正則化を提案した. 分類器の活性化関数がソフトマックス関数であるとき, 分類器の各ニューロンはそれぞれ 2 クラス分類問題を解いているので, 各ニューロンにとって理想的な特徴は, クラスごとに平均が異なり, かつ全てのクラスで同じ分散を持つガウス分布であることを導出した. また, ネットワークがそのような分布を実際に生成していること, それらの分布の一部が重なっているせいでネットワークの分類精度が低下していることを, 実験的に確認した. 本章で提案した判別正則化は, 各分布のそのような重なりを小さくすることができたので, その結果として分類精度を向上させることもできた. また, 分類器の活性関数にシグモイド関数を用いることで, ソフトマックス関数よりも高い分類精度を示す場合があることも確認した.

第 4 章では, 経験的分類損失のテイラー展開をスコアリング関数として用いる新たな Pruning 手法, および, Pruning 手法が解くべき問題を補助変数に関する最適化問題に置き換え, それを安定して解く新たな Pruning 手法の 2 つを提案した. 学習サンプルを用いて各畳み込みフィルタの重要度を評価するスコアリング関数は, 従来のものだと, 極端な値を出力するサンプルに対して脆弱な場合があったが, 本章で提案したスコアリング関数は, そのようなサンプルに対するロバスト性を向上させることができた. また, そのスコアリング関数の導出過程で, 類似した関数を統一的に結びつけることもできた. それに対し, 補助変数の最適化問題を安定して解く新たな Pruning 手法は, 従来手法が行なっていた補助変数の離散最適化問題を連続最適化問題に拡張し, さらに特徴マップの情報を圧縮した空間上でその最適化問題を解くことで, 類似した既存手法に比べ安定した Pruning を達成した. また, 補助変数の連続最適化問題と, Batch Normalization による特徴マップの正規化は相性が悪いことも確認した. 正規化との相性の悪さは, その効果を制限することである程度緩和できるが, 提案手法を用いたほうが安定して高い性能が得られることも確認した. CNN を用いた画像分類の実験では, どちらの提案手法も, 他の類似した Pruning 手法と比較して良好な性能を示した. これらのことから, CNN が獲得した特徴マップは, CNN の計算コスト削減に活用できることを示せた.

本論文でこれまで行なってきた, CNN が獲得する特徴マップの解析や議論, 提案手法を用いた実験により, ネットワークが獲得すべき特徴や, 獲得した特徴の活用方法を示すという本研究の目的は, ある程度達成されたものと考える.

最後に, 今後の課題について述べる. 第 2 章では, CNN が獲得する特徴マップに対する正則化の重要性だけでなく, 脳の視覚野に関する知見を CNN に持ち込むことの重要性も示した. ここで持ち込んだ知見は, 視覚野

が行う情報処理の中でも低次元のものであるため, より高次元の情報処理に関する知見を持ち込むことも考えられる. 第 3 章では, CNN の分類器に注目し, その挙動や入力される特徴マップを解析することの重要性を示した. ここで行なった実験の一部では, 分類器の活性化関数としてソフトマックス関数よりもシグモイド関数を用いた方が優れた分類精度が得られる場合があることを確認したが, その理由に関しては未だ検証・考察が不十分なところがある. 今後は, その検証を進め, より好ましい分類器の設計に繋げていく必要がある. 第 4 章では, 獲得した特徴マップの活用方法の一つとして Pruning を示したが, 特徴マップは, CNN の情報処理を理解するのに重要な手がかりが数多く含まれているので, ネットワークの説明可能性や CNN のブラックボックス解明など, より多くの分野でも特徴マップの活用が期待される.

謝辞

本論文は, 広島大学ならびに産業技術総合研究において行ってきた, CNN の特徴マップに関する研究をまとめたものです.

本研究は, 広島大学大学院, 産業技術総合研究の多くの方のご支援のおかげで行うことができました. 特に, 栗田多喜夫教授 (広島大学), 小林匠氏 (産業技術総合研究), 渡辺顕司氏 (産業技術総合研究) の多大なご支援に, 心から感謝いたします.

指導教官である栗田多喜夫教授 (広島大学) には, 筆者が学部在学中の頃から, 研究に対する姿勢や心構えについて熱心に指導していただきました. 実際, 筆者の研究において, 栗田教授のご指導は大きな役割を果たしています. 特に, 本論文の第 2 章, 第 3 章は, そのご指導が無ければ, 日の目を見なかった可能性が大きいです.

産業技術総合研究所の小林匠氏, 渡辺顕司氏には, 筆者がリサーチアシスタント (産業技術総合研究) を兼任していた際に, 根気強い議論ならびに多大なご協力を頂きました. 本論文の第 4 章は, 栗田教授のご指導はもちろん, お二方のご協力も必要不可欠でした.

以上の方々に心から感謝いたします.

また, 研究室のメンバーにも, 議論や発表練習などに付き合っていただいたことに感謝いたします.

参考文献

[1] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, 2012.

[4] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cereb cortex.*, pp. 1–47, 1991.

[5] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[6] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Vol. 5, pp. 115–133, 1943.

[7] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[8] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, and S. Lacoste-Julien. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, 2017.

[9] D. A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2015.

[10] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, Vol. 15, pp. 1929–1958, 2014.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

[12] J. Cheng, P. S. Wang, G. Li, Q. H. Hu, and H. Q. Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, Vol. 19, pp. 64–77, 2018.

[13] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems 29*, pp. 1379–1387, 2016.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861*, 2017.

[15] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.

[16] 神嶌敏弘., 麻生英樹., 安田宗樹., 前田新一., 岡野原大輔., 岡谷貴之., 久保陽太郎., ボレガラダヌシカ. 深層学習 Deep Learning. 2015.

[17] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, Vol. 381, pp. 607–609, 1996.

[18] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1?. *Vision research*, Vol. 37, pp. 3311–3325, 1997.

[19] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, Vol. 7, pp. 179–188, 1936.

[20] Y. LeCun, J. S. Denker, and S.A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, pp. 598–605, 1990.

[21] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pp. 164–171, 1993.

[22] L. Theis, I. Korshunova, A. Tejani, and F. Huszár. Faster gaze prediction with dense networks and fisher pruning. In *arXiv preprint arXiv:1801.05787*, 2018.

[23] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017.

[24] X. Xiao, Z. Wang, and S. Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Advances in Neural Information Processing Systems 32*, pp. 13681–13691, 2019.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. In *arXiv preprint arXiv:1312.4400*, 2013.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition.*, pp. 1–9, 2015.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z Wojna. Rethinking the inception architecture for computer vision. In *Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[28] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, Vol. 1, pp. 541–551, 1989.

[30] H. Ide and T. Kurita. Improvement of learning for cnn with relu activation by sparse regularization. In *International Joint Conference on Neural Networks*, pp. 2684–2691. IEEE, 2017.

[31] 井手秀徳., 栗田多喜夫. Cnn における relu 活性化関数に対するスパース正則化の適用と分析. 電子情報通信学会論文誌 D, Vol. 101, pp. 1110–1119, 2018.

[32] H. T. Siegelmann and E. D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, Vol. 4, pp. 77–80, 1991.

[33] 林原香織. 山下雅史., 阿江忠. シグモイド関数の連続性/離散性とニューラルネットワークのマシン能力について. 電子情報通信学会論文誌 D, Vol. 73, pp. 1220–1226, 1990.

[34] D. O. Hebb. The organization of behavior: a neuropsychological theory. *Psychology Press*, 2005.

[35] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, Vol. 65, p. 386, 1958.

[36] M. Minsky and S. A. Papert. *Perceptrons: An introduction to computational geometry.* 1987.

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating

errors. *nature*, Vol. 323, pp. 533–536, 1986.

[38] S. Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, pp. 299–307, 1967.

[39] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, and L. Wiskott. Deep hierarchies in the primate visual cortex: What can we learn for computer vision?. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 35, pp. 1847–1871, 2012.

[40] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, Vol. 20, pp. 273–297, 1995.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pp. 2672–2680, 2014.

[42] M. Yuan and Y. Peng. Bridge-gan: Interpretable representation learning for text-to-image synthesis. *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2019.

[43] X. He, Y. Peng, and J. Zhao. Which and how many regions to gaze: Focus discriminative regions for fine-grained visual categorization. *International Journal of Computer Vision*, Vol. 127, pp. 1235–1255, 2019.

[44] D. Masters and C. Luschi. Revisiting small batch training for deep neural networks. In *arXiv preprint arXiv:1804.07612*.

[45] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Muller. Efficient backprop. *Neural networks: Tricks of the trade*, Vol. 1524, , 9–50.

[46] S. Wiesler and H. Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pp. 657–665, 2011.

[47] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, Vol. 9, pp. 147–169, 1985.

[48] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pp. 807–814, 2010.

[49] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[50] I. LeCun, Y. Kanter and S. A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, Vol. 66, p. 2396, 1991.

[51] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *Artificial intelligence and statistics*, pp. 924–932, 2012.

[52] K. Desjardins, G. Simonyan and R. Pascanu. Natural neural networks. In *Advances in neural information processing systems*, pp. 2071–2079, 2015.

[53] C. F. Cadieu, H. Hong, D. L. Yamins, N. Pinto, D. Ardila, E. A. Solomon, and J. J. DiCarlo. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS Comput Biol*, Vol. 10, , 2014.

[54] D. L. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, Vol. 111, pp. 8619–8624, 2014.

[55] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture

in the cat's visual cortex. *The Journal of physiology*, Vol. 160, pp. 106–154, 1962.

[56] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, Vol. 195, pp. 215–243, 1968.

[57] J. P. Jones and L. A. Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of neurophysiology*, Vol. 58, pp. 1233–1258, 1987.

[58] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, Vol. 313, pp. 504–507, 2006.

[59] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. Hrank: Filter pruning using high-rank feature map. In *Computer Vision and Pattern Recognition*, pp. 1529–1538, 2020.

[60] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, Vol. 86, pp. 2278–2324, 1998.

[61] A. Krizhevsky and G.E. Hinton. Learning multiple layers of features from tiny images. In *Technical report*, Vol. 1, p. 7. University of Toronto, 2009.

[62] R. Dorfer, M. Kelz and G. Widmer. Deep linear discriminant analysis. In *International Conference on Learning Representations*, pp. 165–175, 2016.

[63] H. Kurita, T. Asoh and N. Otsu. Nonlinear discriminant features constructed by using outputs of multilayer perceptron. In *International Conference on Speech, Image Processing and Neural Networks*, pp. 417–420. IEEE, 1994.

[64] H. Ide and T. Kurita. Convolutional neural network with discriminant criterion for input of each neuron in output layer. In *International Conference on Neural Information Processing*, pp. 332–339. Springer, 2018.

[65] N. Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems*, Vol. 9, pp. 62–66, 1979.

[66] K. Fukui and O. Yamaguchi. Facial feature point extraction method based on combination of shape extraction and pattern matching. *Systems and Computers in Japan*, Vol. 29, pp. 49–58, 1998.

[67] H. Osman and M. M. Fahmy. On the discriminatory power of adaptive feed-forward layered networks. *IEEE transactions on pattern analysis and machine intelligence*, Vol. 16, pp. 837–842, 1994.

[68] X. Chen, K. Yu and H. Chi. Combining linear discriminant functions with neural networks for supervised learning. *Neural Computing & Applications*, Vol. 6, pp. 19–41, 1997.

[69] C. Wu, L. Shen and A. Van Den Hengel. Deep linear discriminant analysis on fisher networks: A hybrid architecture for person re-identification. *Pattern Recognition*, Vol. 65, pp. 238–250, 2017.

[70] M. Abe, J. Miyao, and T. Kurita. Adaptive neuron-wise discriminant criterion and adaptive center loss at hidden layer for deep convolutional neural network. In *International Joint Conference on Neural Networks*, 2020.

[71] O. J. Kolesnikov A. Zhai X. Beyer, L. Henaff and A. V. D. Oord. Are we done with imagenet?. In *arXiv preprint arXiv:2006.07159.*, 2020.

[72] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[73] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.

[74] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H.P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, pp. 1–13, 2017.

[75] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29*, pp. 2074–2082, 2016.

[76] B.O. Ayinde and J.M. Zurada. Building efficient convnets using redundant feature pruning. In *arXiv preprint arXiv:1802.07653*, 2018.

[77] J. H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *International Conference on Computer Vision*, pp. 5068–5076, 2017.

[78] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision*, pp. 1398–1406, 2017.

[79] Y. Peng and J. Qi. Quintuple-media joint correlation learning with deep compression and regularization. *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2019.

[80] R. Yu, A. Li, C. F. Chen, J. H. Lai, V. I. Morariu, X. Han, C. Y. Gao, M. Lin, and L.S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

[81] Q. Huang, K. Zhou, S. You, and U. Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 709–718, 2018.

[82] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian. Channel pruning via automatic structure search. In *IJCAI*, 2020.

[83] H. Li, C. Ma, W. Xu, and X. Liu. Feature statistics guided efficient filter pruning. In *IJCAI*, 2020.

[84] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *Computer Vision and Pattern Recognition*, pp. 11264–11272.

[85] H. Ide, T. Kobayashi, K. Watanabe, and T. Kurita. Robust pruning for efficient cnns. *Pattern Recognition Letters*, Vol. 135, pp. 90–98, 2020.

[86] S Amari. *Information geometry and its applications*, Vol. 194. 2016.

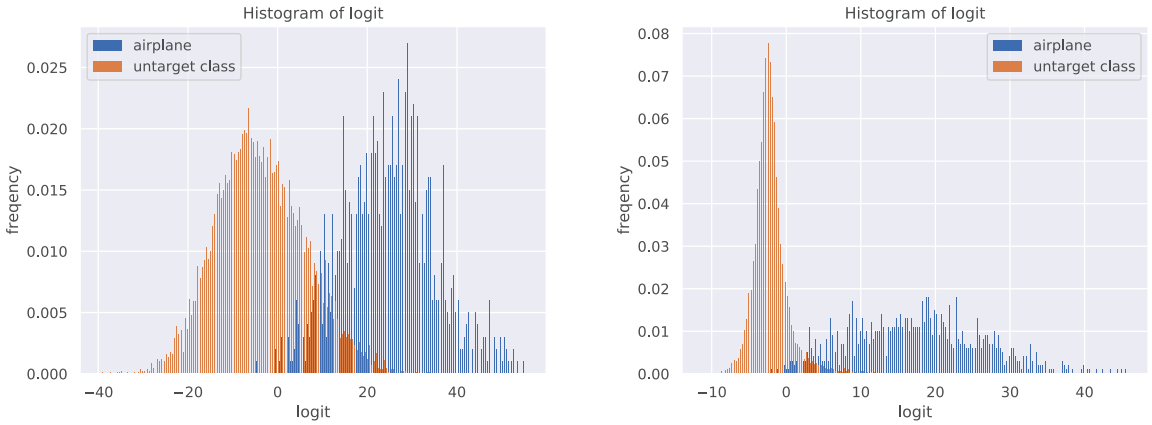
[87] X. Sun, D. Zhou, X. Pan, Z. Zhong, and F. Wang. Pruning filters with l1-norm and standard deviation for cnn compression. In *International Conference on Machine Vision*, Vol. 11041, p. 110412J, 2019.

[88] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *the fourteenth international conference on artificial intelligence and statistics*, 2011.

付録 A　第 3 章の付録

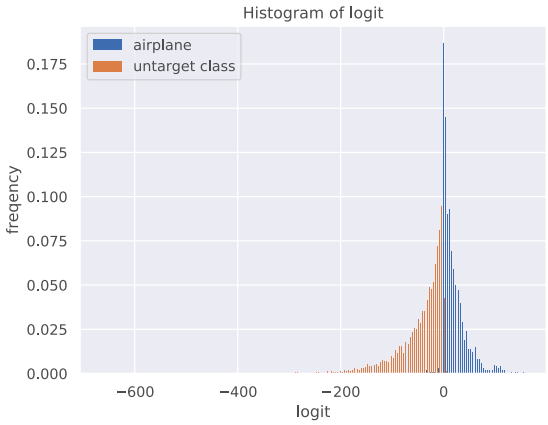
A.1　VGG13-like のロジットのヒストグラム一覧

3.5 章, 3.6 章のように, CIFAR10 のテストサンプルを VGG13-like (表 3.4) に入力したときの, 分類器のニューロンの入力のヒストグラム (図付録 A.1, 付録 A.2, 付録 A.3, 付録 A.4, 付録 A.5, 付録 A.6, 付録 A.7, 付録 A.8, 付録 A.9) を示す.

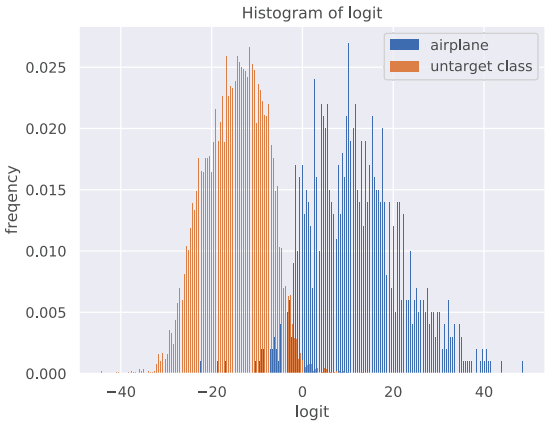


(a) ソフトマックス関数 (判別正則化なし)

(b) ソフトマックス関数 (判別正則化あり)

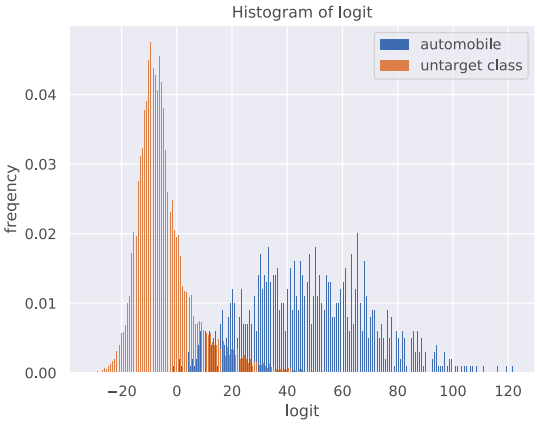


(c) シグモイド関数 (判別正則化なし)

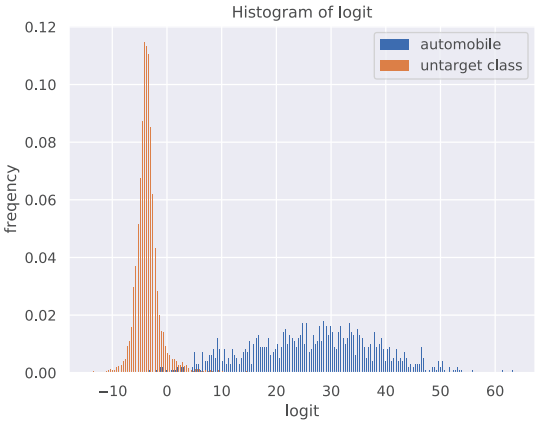


(d) シグモイド関数 (判別正則化あり)

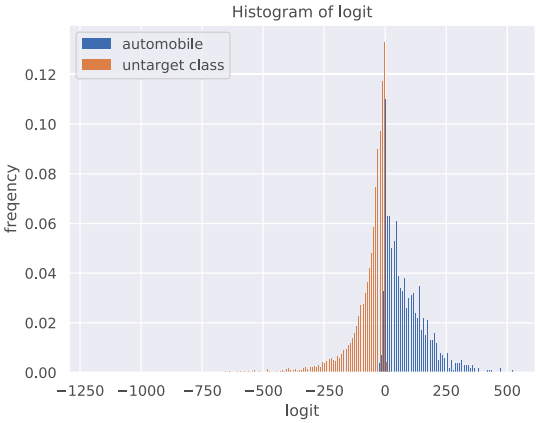
図付録 A.1: 飛行機



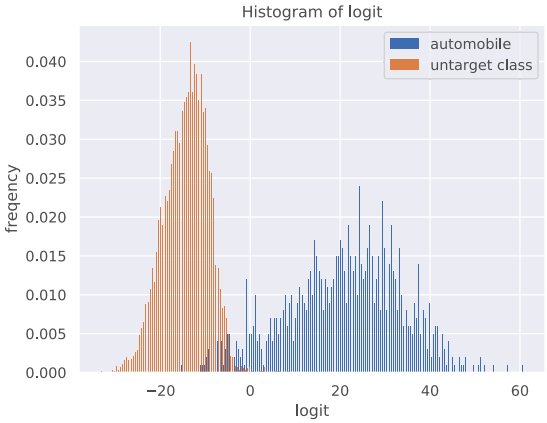
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

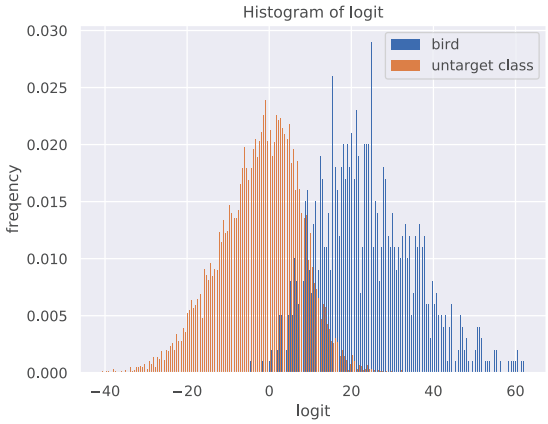


(c) シグモイド関数 (判別正則化なし)

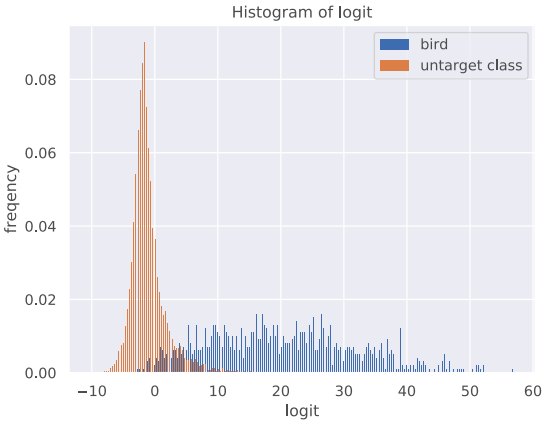


(d) シグモイド関数 (判別正則化あり)

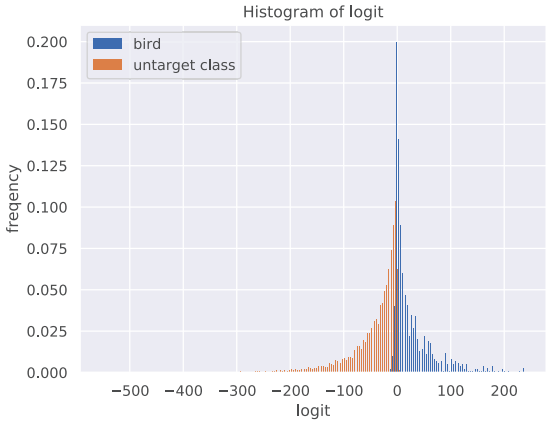
図付録 A.2: 自動車



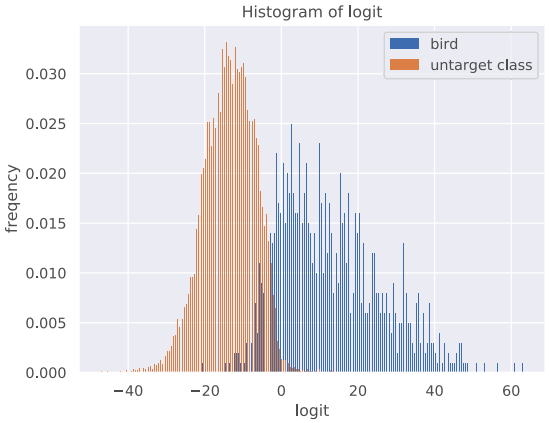
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

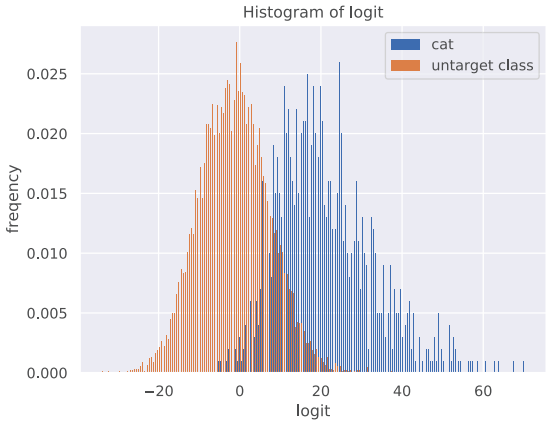


(c) シグモイド関数 (判別正則化なし)

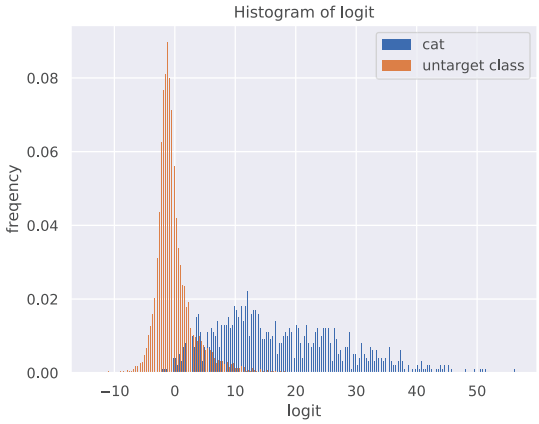


(d) シグモイド関数 (判別正則化あり)

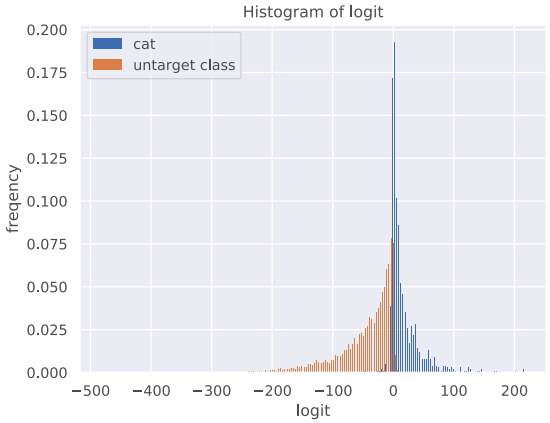
図付録 A.3: 鳥



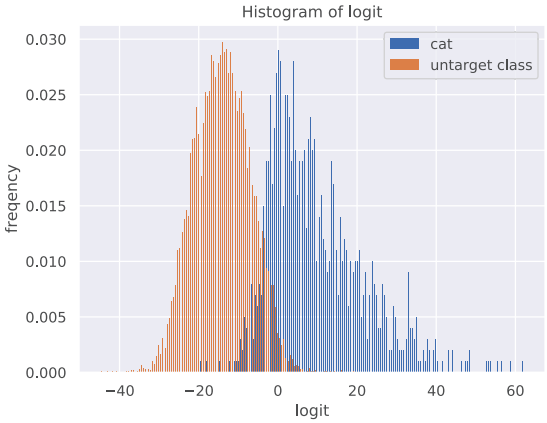
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

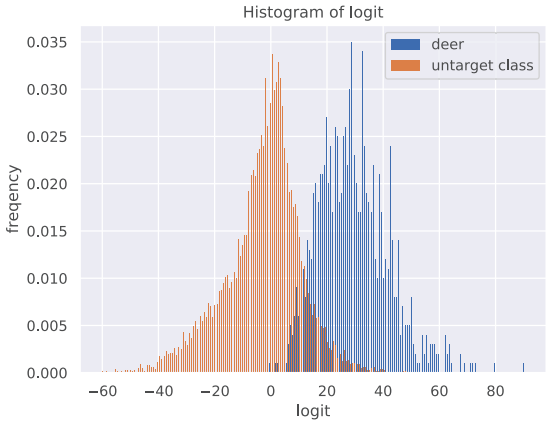


(c) シグモイド関数 (判別正則化なし)

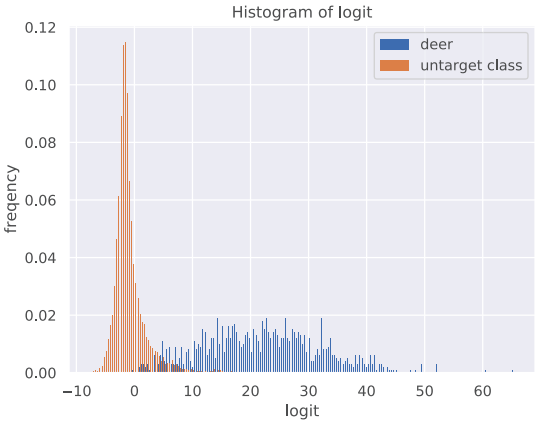


(d) シグモイド関数 (判別正則化あり)

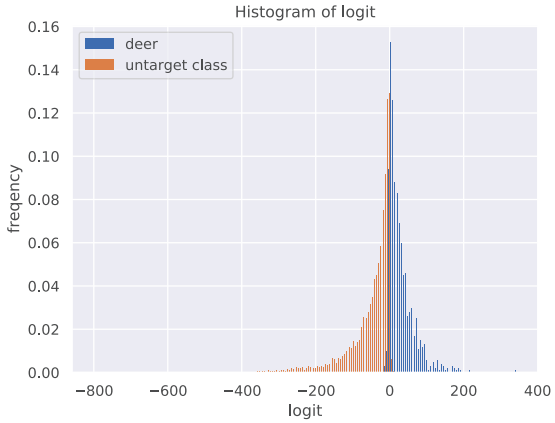
図付録 A.4: 猫



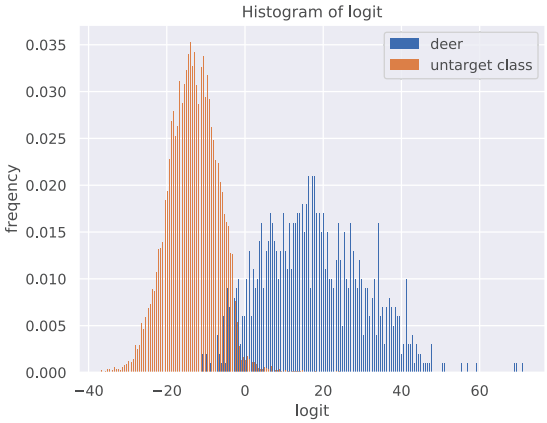
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

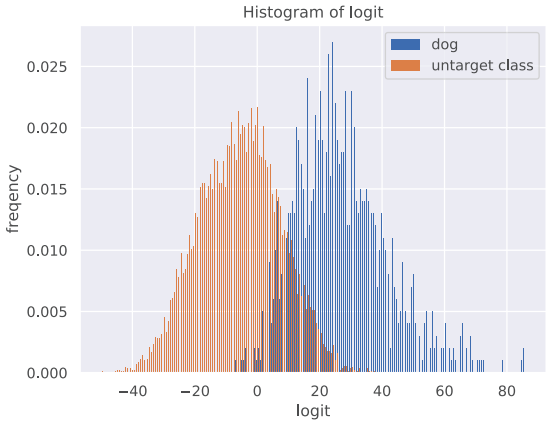


(c) シグモイド関数 (判別正則化なし)

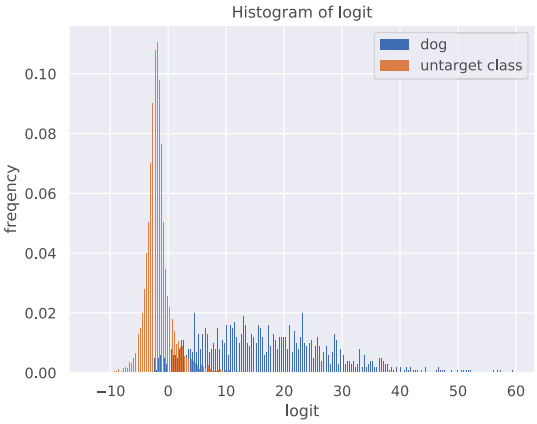


(d) シグモイド関数 (判別正則化あり)

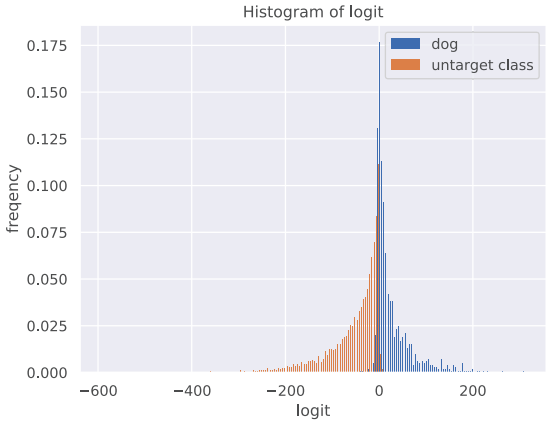
図付録 A.5: 鹿



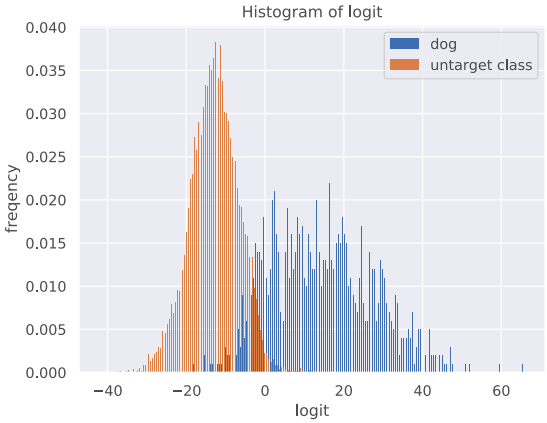
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

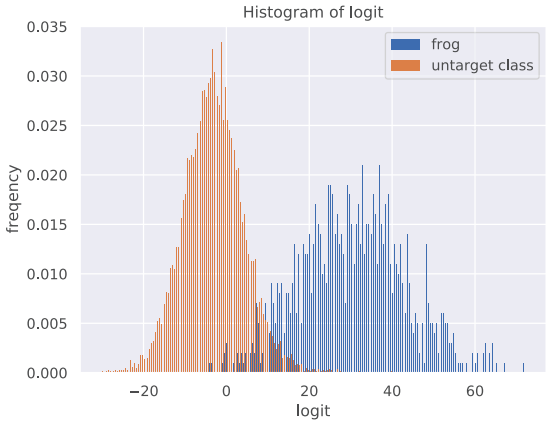


(c) シグモイド関数 (判別正則化なし)

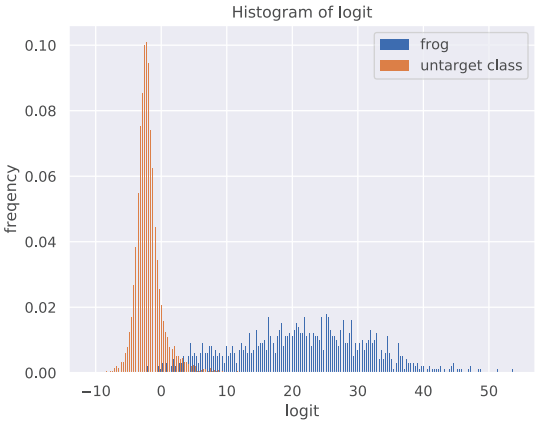


(d) シグモイド関数 (判別正則化あり)

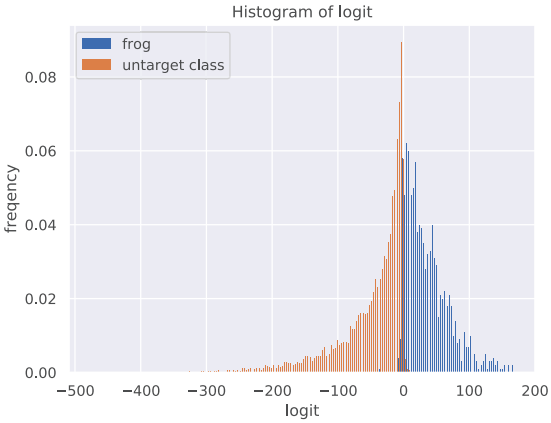
図付録 A.6: 犬



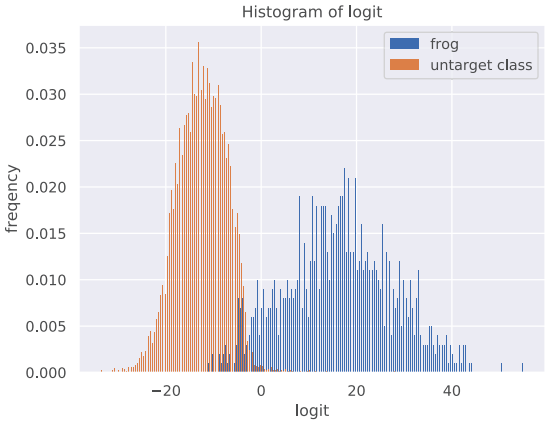
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

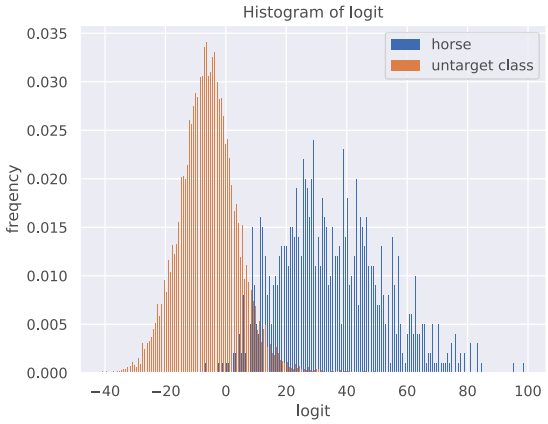


(c) シグモイド関数 (判別正則化なし)

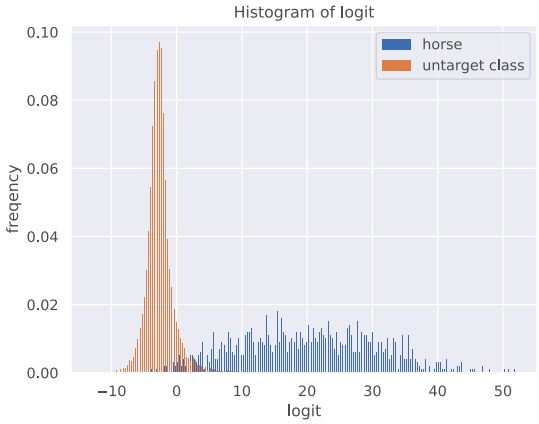


(d) シグモイド関数 (判別正則化あり)

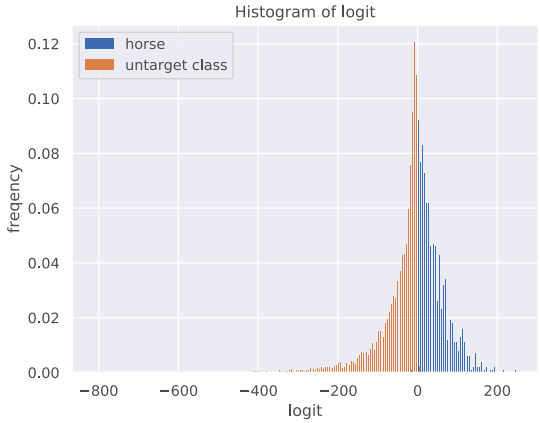
図付録 A.7: カエル



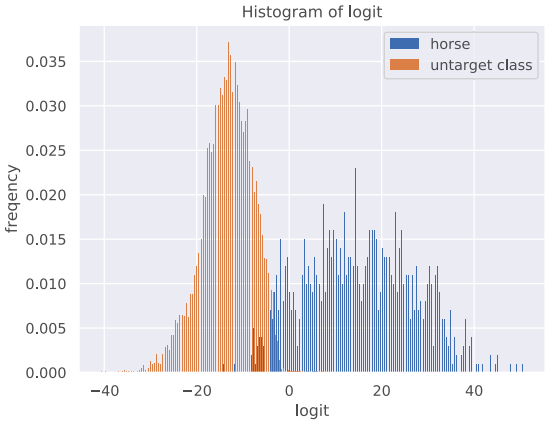
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

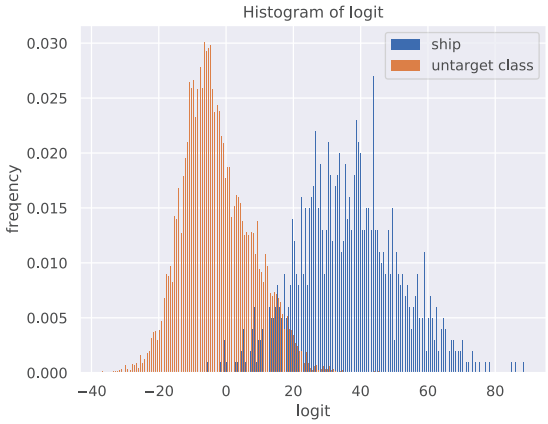


(c) シグモイド関数 (判別正則化なし)

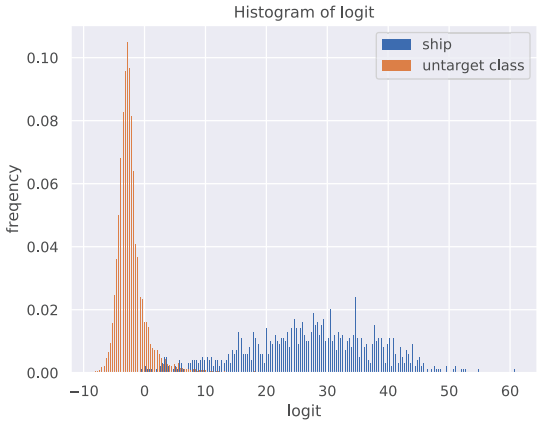


(d) シグモイド関数 (判別正則化あり)

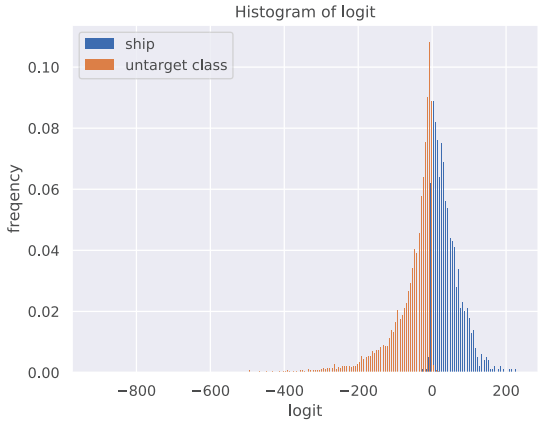
図付録 A.8: 馬



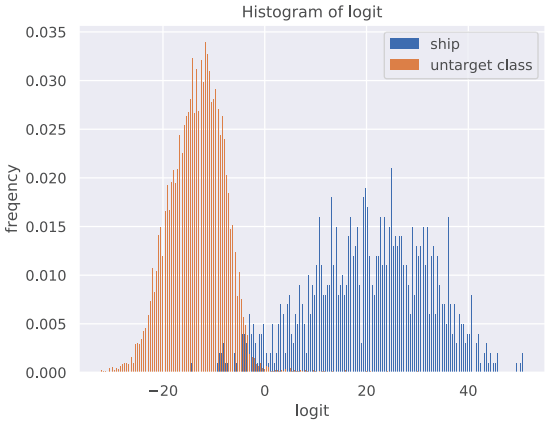
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

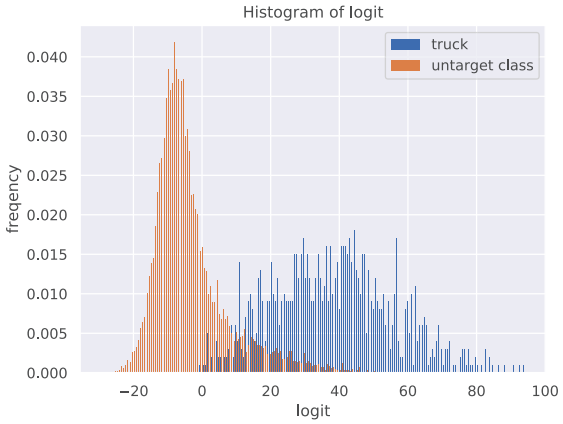


(c) シグモイド関数 (判別正則化なし)

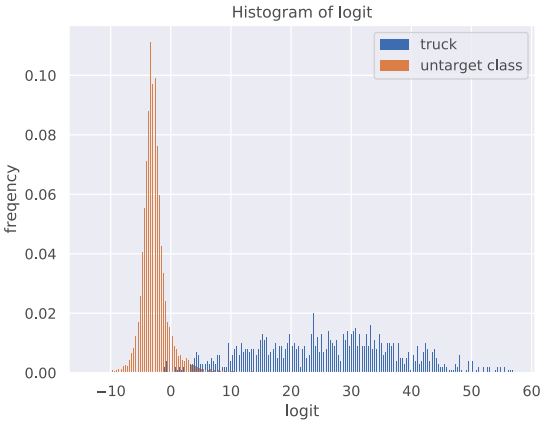


(d) シグモイド関数 (判別正則化あり)

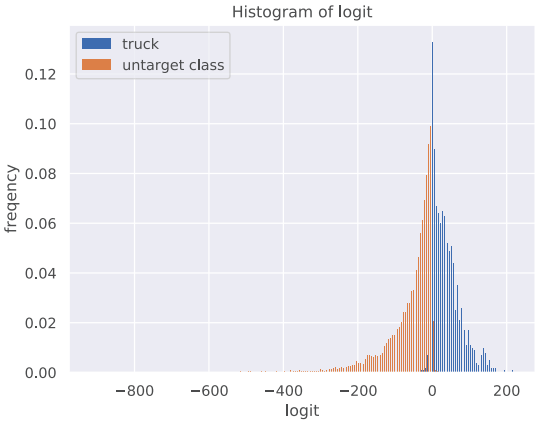
図付録 A.9: 船



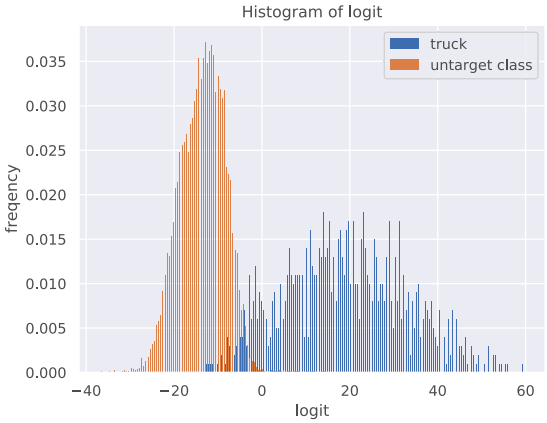
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)



(c) シグモイド関数 (判別正則化なし)

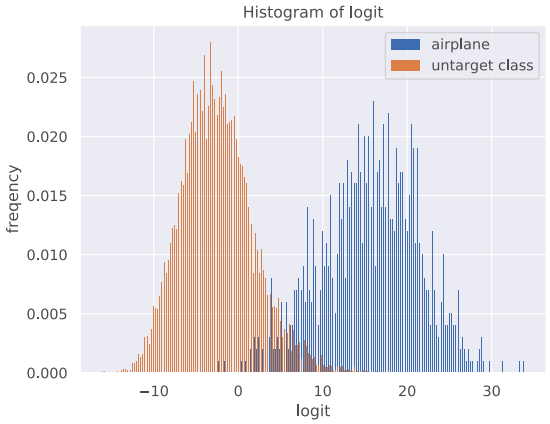


(d) シグモイド関数 (判別正則化あり)

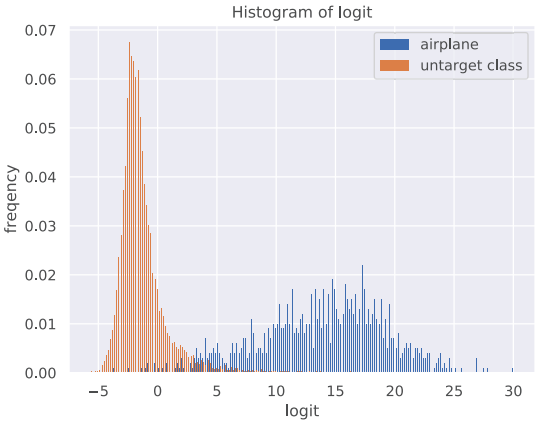
図付録 A.10: トラック

A.2　ResNet18-like のロジットのヒストグラム一覧

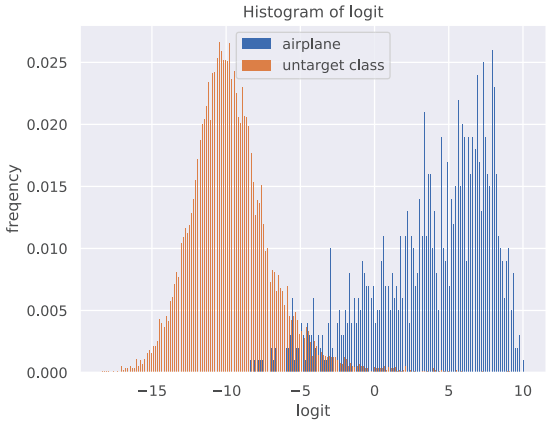
ここでは, CIFAR10 のテストサンプルを ResNet18-like (表 3.5) に入力したときの, 分類器のニューロンの入力のヒストグラム (図付録 A.11, 付録 A.12, 付録 A.13, 付録 A.14, 付録 A.15, 付録 A.16, 付録 A.17, 付録 A.18, 付録 A.19) を示す.



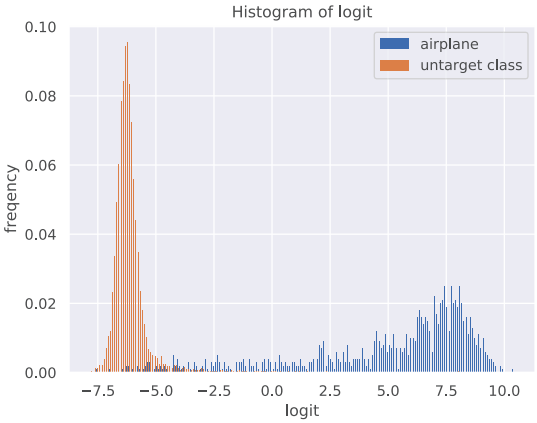
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

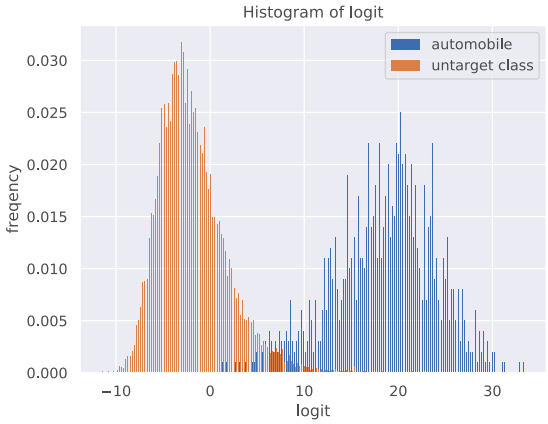


(c) シグモイド関数 (判別正則化なし)

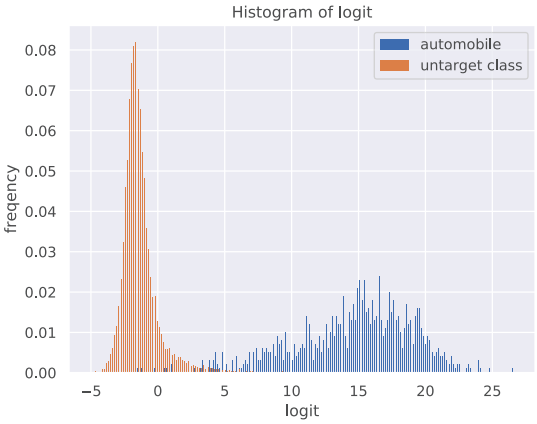


(d) シグモイド関数 (判別正則化あり)

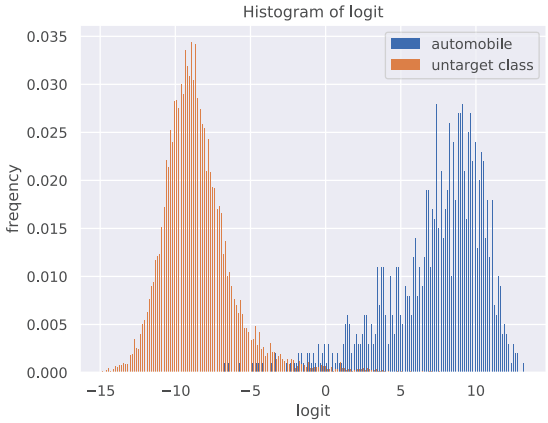
図付録 A.11: 飛行機



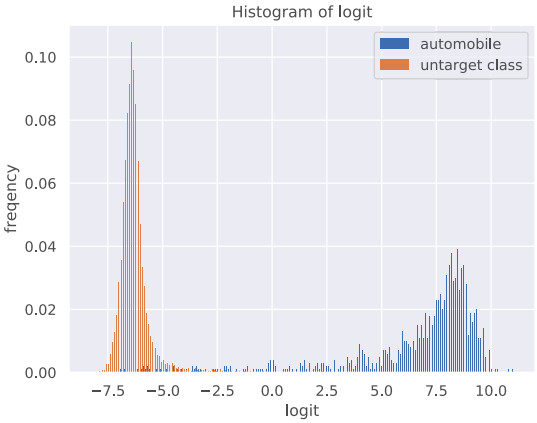
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

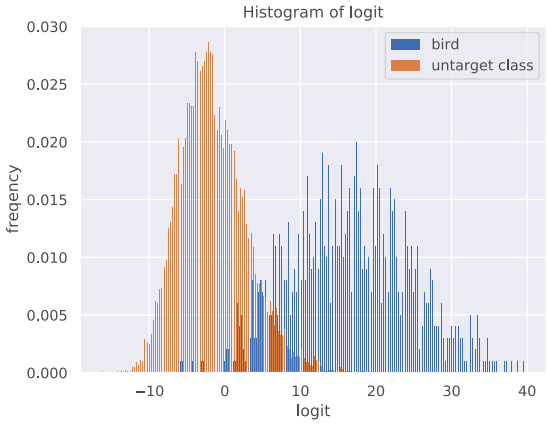


(c) シグモイド関数 (判別正則化なし)

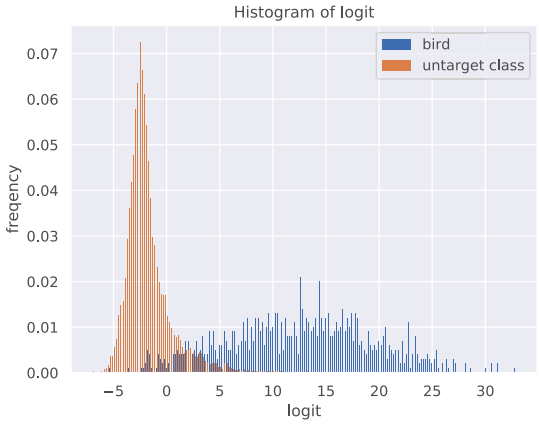


(d) シグモイド関数 (判別正則化あり)

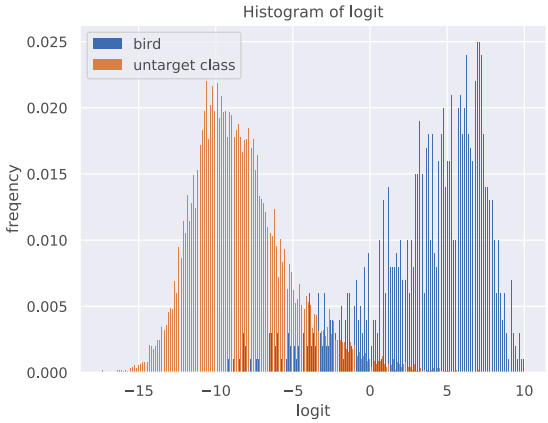
図付録 A.12: 自動車



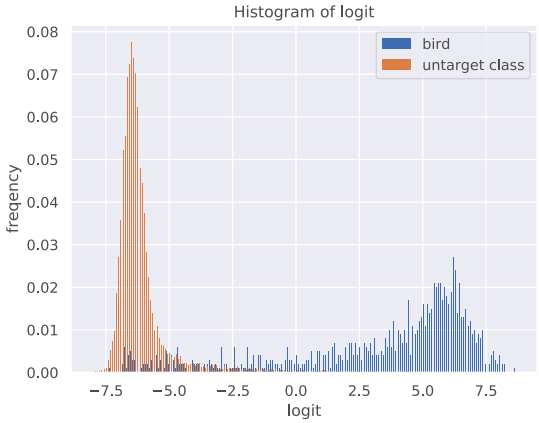
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

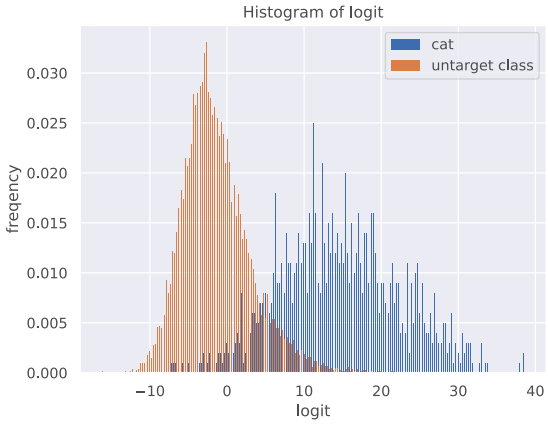


(c) シグモイド関数 (判別正則化なし)

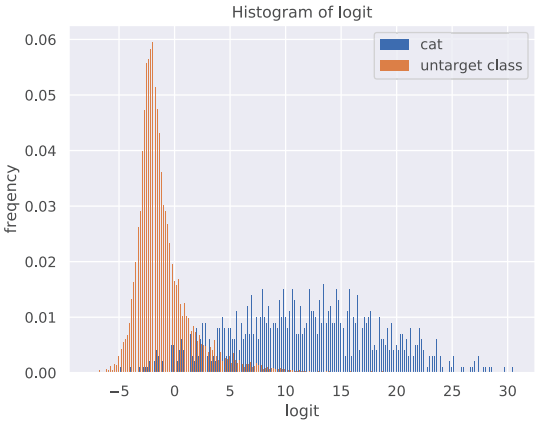


(d) シグモイド関数 (判別正則化あり)

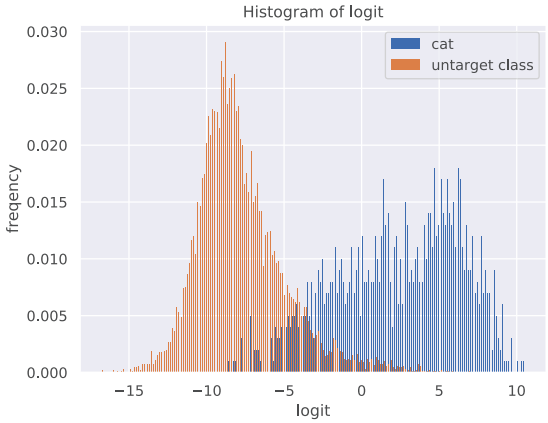
図付録 A.13: 鳥



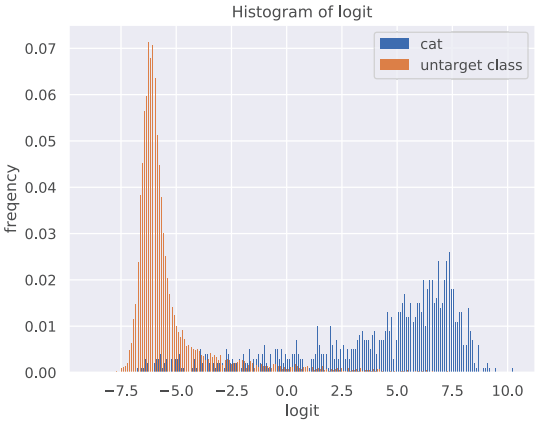
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

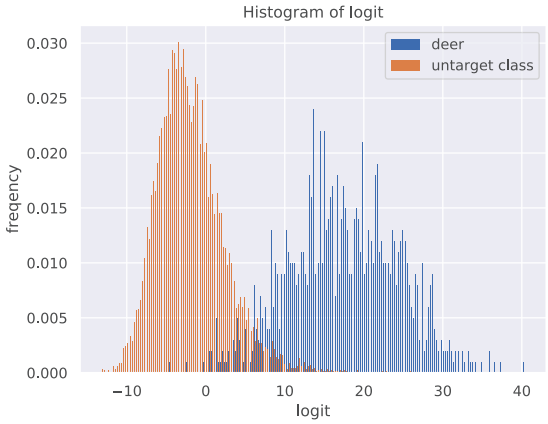


(c) シグモイド関数 (判別正則化なし)

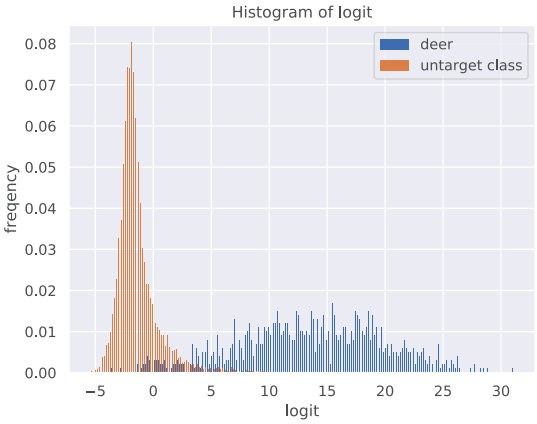


(d) シグモイド関数 (判別正則化あり)

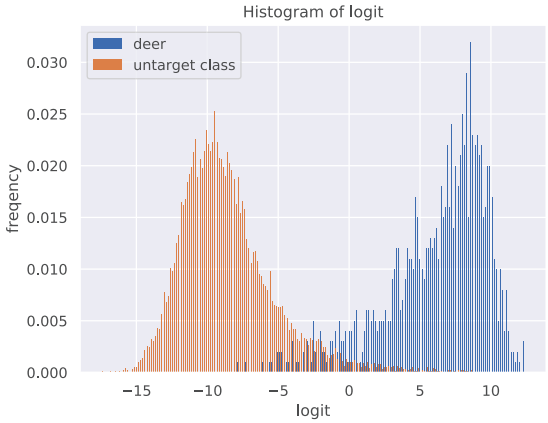
図付録 A.14: 猫



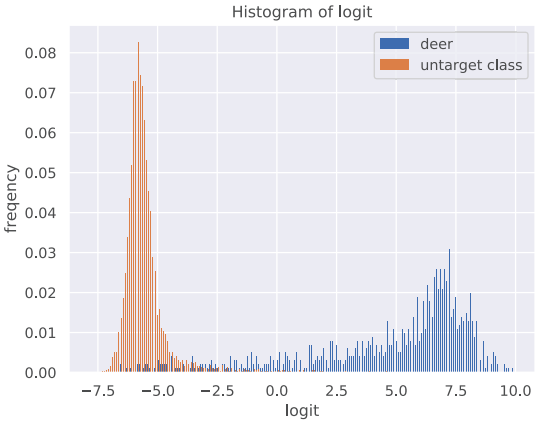
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

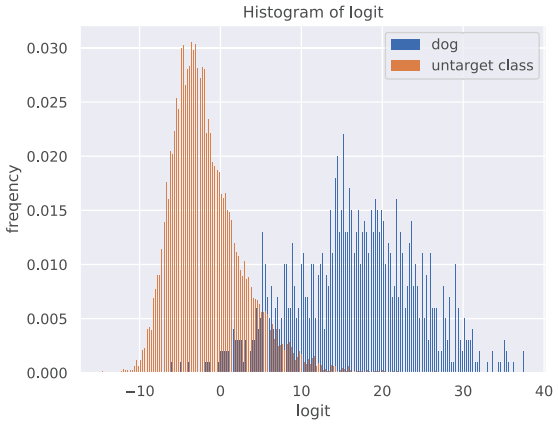


(c) シグモイド関数 (判別正則化なし)

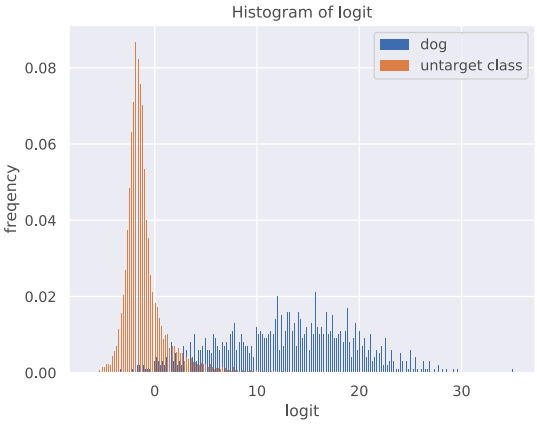


(d) シグモイド関数 (判別正則化あり)

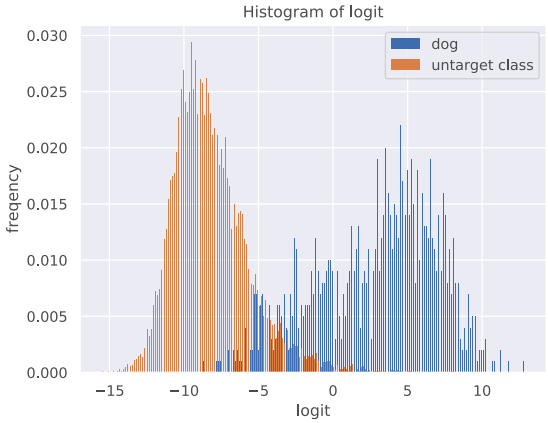
図付録 A.15: 鹿



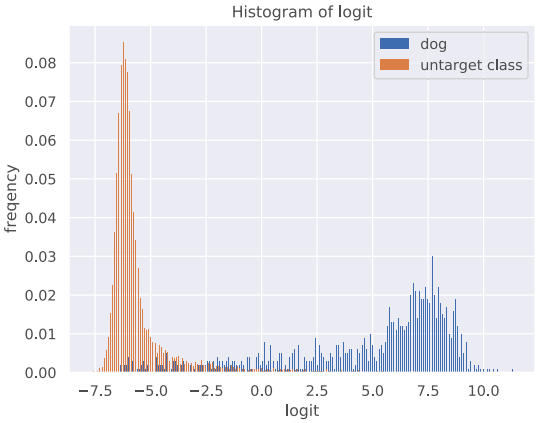
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

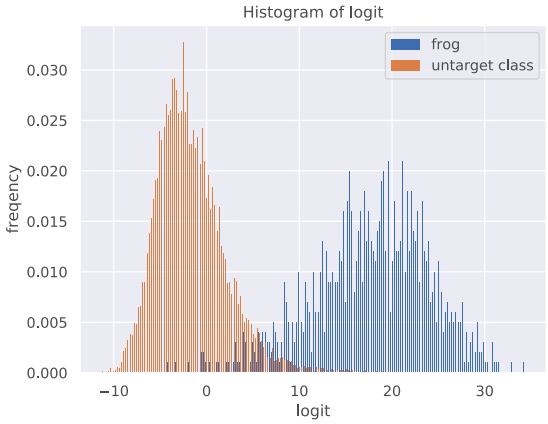


(c) シグモイド関数 (判別正則化なし)

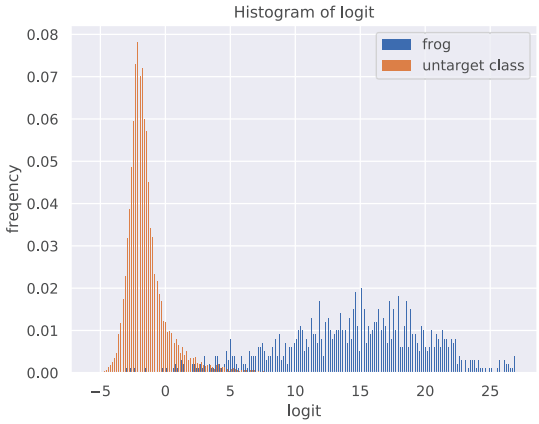


(d) シグモイド関数 (判別正則化あり)

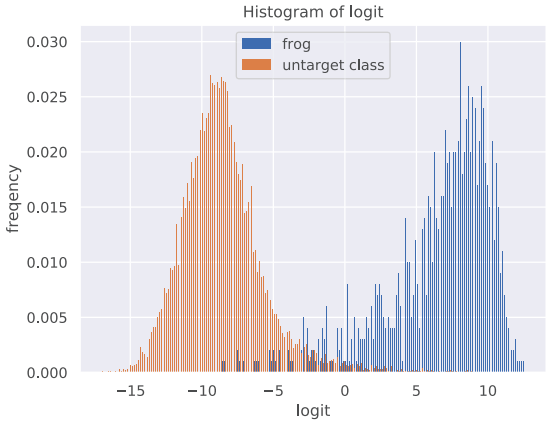
図付録 A.16: 犬



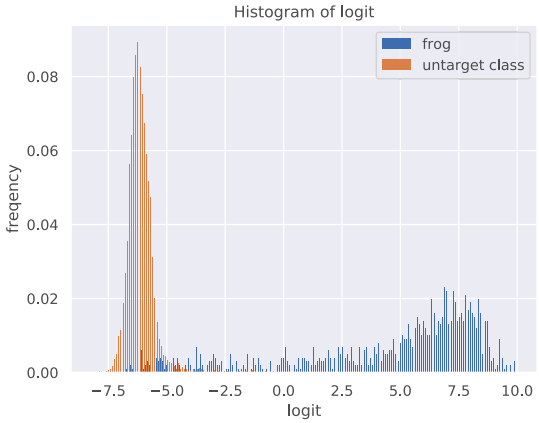
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

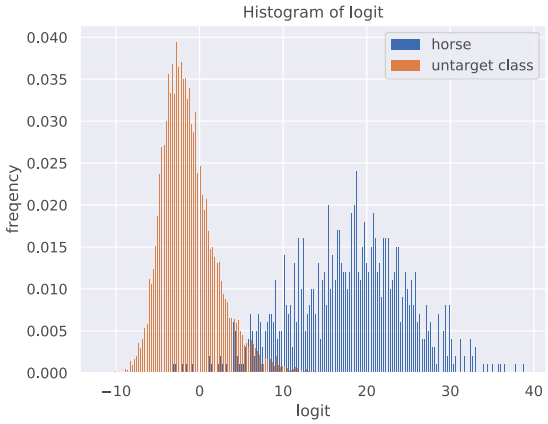


(c) シグモイド関数 (判別正則化なし)

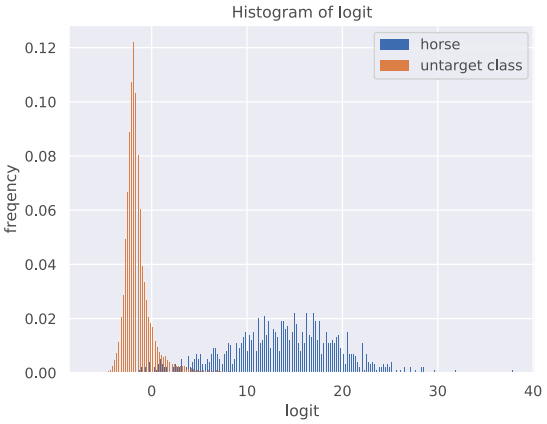


(d) シグモイド関数 (判別正則化あり)

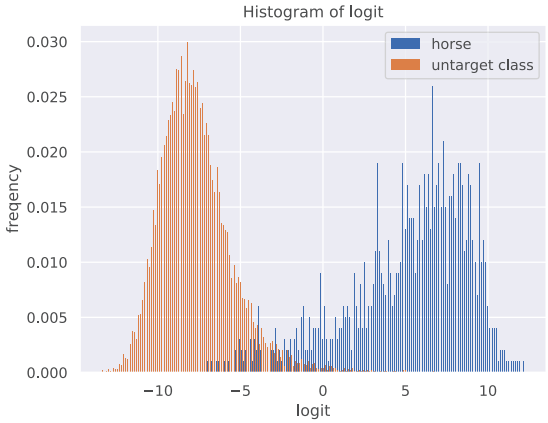
図付録 A.17: カエル



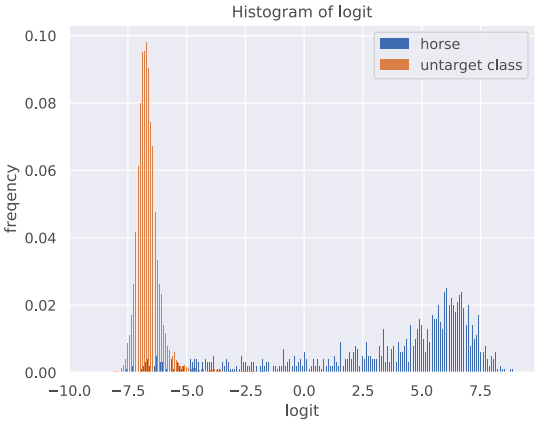
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

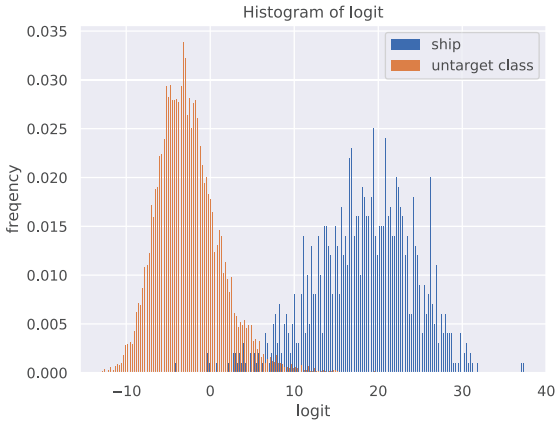


(c) シグモイド関数 (判別正則化なし)

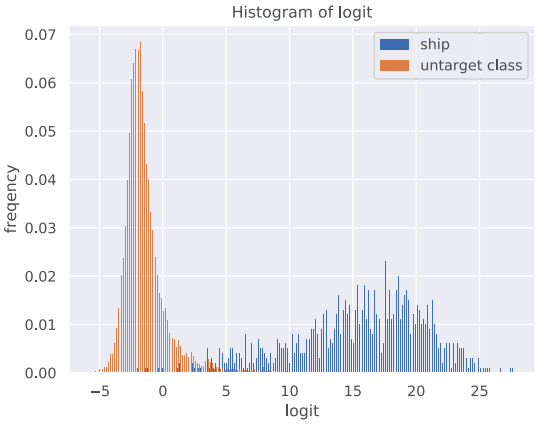


(d) シグモイド関数 (判別正則化あり)

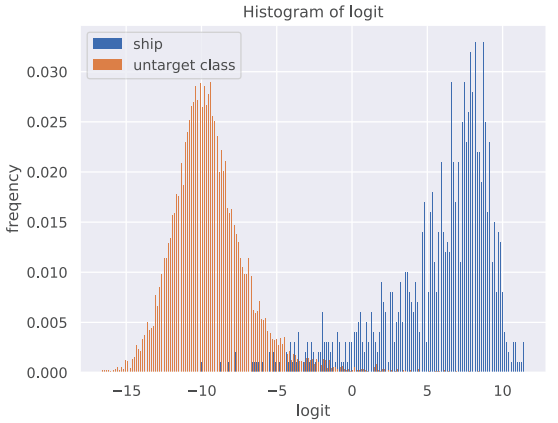
図付録 A.18: 馬



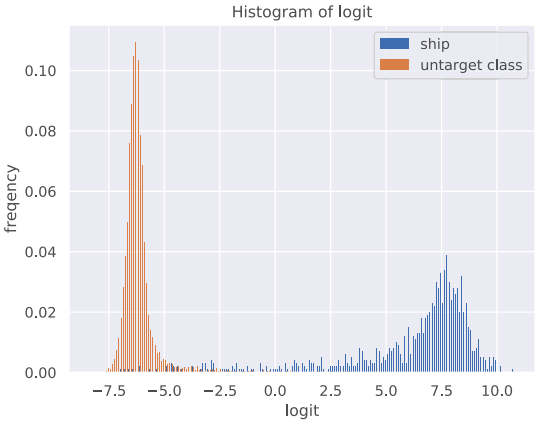
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)

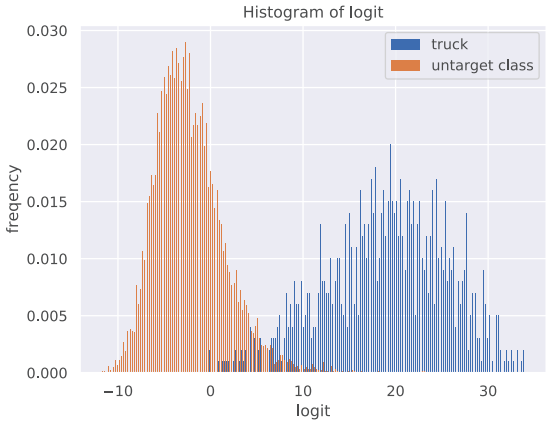


(c) シグモイド関数 (判別正則化なし)

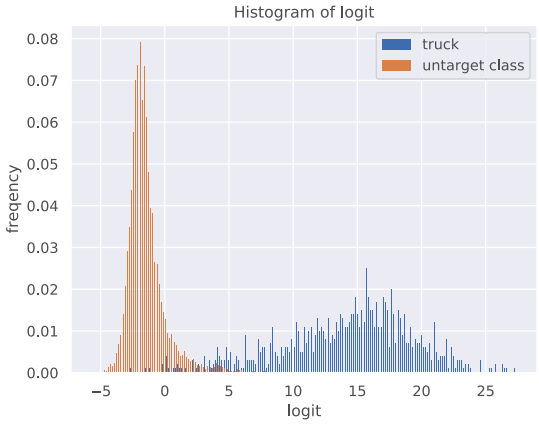


(d) シグモイド関数 (判別正則化あり)

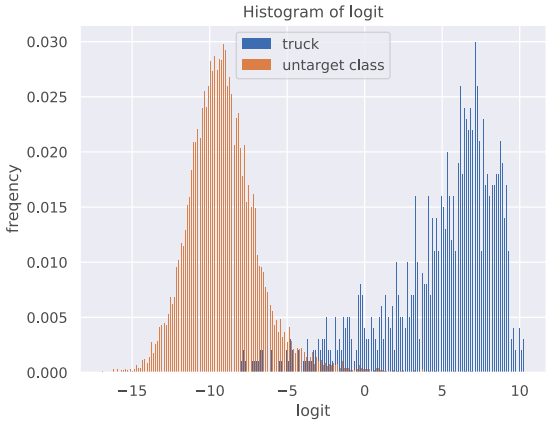
図付録 A.19: 船



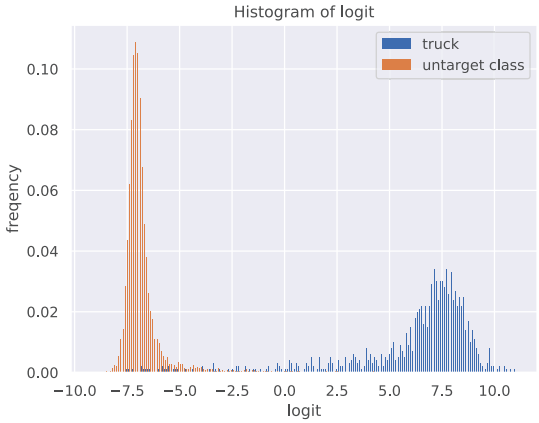
(a) ソフトマックス関数 (判別正則化なし)



(b) ソフトマックス関数 (判別正則化あり)



(c) シグモイド関数 (判別正則化なし)



(d) シグモイド関数 (判別正則化あり)

図付録 A.20: トラック

付録 B　第 4 章の付録

B.1　Hessian 行列の近似

4.3.1 章で述べたように，特徴マップ $\boldsymbol{x}$ に関する損失関数 $\mathcal{L}$ のテイラー展開 (式 (4.3)) の計算に必要な Hessian 行列 $\boldsymbol{H}$ を近似する．式 (4.4) で用いた近似は

$$\begin{aligned}\boldsymbol{H} &= \frac{\partial^2 \mathcal{L}}{(\partial \boldsymbol{x})\,(\partial \boldsymbol{x})^T} = \frac{\partial}{\partial \boldsymbol{x}} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} \right)^T \\ &= \frac{\partial}{\partial \boldsymbol{x}} \left(\frac{\partial \mathcal{L}}{\partial p_t} \frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T \\ &= \frac{\partial}{\partial \boldsymbol{x}} \left(-\frac{1}{p_t} \frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T \end{aligned} \tag{付録 B.1}$$

と導出される．ここで, 損失 $\mathcal{L}$ は式 (4.2) で定義され, p_t は, $\boldsymbol{x}$ の正解クラス t の事後確率 (ソフトマックス) であることに注意してほしい．そして, Hessian $\boldsymbol{H}$ を書き換えると

$$\begin{aligned}\boldsymbol{H} &= \frac{\partial}{\partial \boldsymbol{x}} \left(-\frac{1}{p_t} \frac{\partial y_t}{\partial \boldsymbol{x}} \right)^T \\ &= \left(-\frac{\partial}{\partial \boldsymbol{x}} \frac{1}{p_t} \right) \left(\frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T - \frac{1}{p_t} \frac{\partial}{\partial \boldsymbol{x}} \left(\frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T \\ &\approx \left(-\frac{\partial}{\partial \boldsymbol{x}} \frac{1}{p_t} \right) \left(\frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T \\ &= \frac{1}{p_t^2} \left(\frac{\partial p_t}{\partial \boldsymbol{x}} \right) \left(\frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T \\ &= \left(-\frac{1}{p_t} \frac{\partial p_t}{\partial \boldsymbol{x}} \right) \left(-\frac{1}{p_t} \frac{\partial p_t}{\partial \boldsymbol{x}} \right)^T \\ &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} \right)^T \end{aligned} \tag{付録 B.2}$$

となる．