修士論文

意味的な画像概念のDNN学習過程における汎化性能について

On generalization performance under DNN learning process of semantic image concepts

東京電機大学大学院 システムデザイン工学研究科

情報システム工学専攻 修士課程

23AMJ03 岩瀬 俊

研究指導教員　教授 前田 英作

要旨

深層ニューラルネットワーク（DNN）を用いたエンドツーエンド学習は，コンピュータビジョン（CV）および自然言語処理（NLP）の諸タスクにおいて高い性能を実証している．しかしながら，分散表現に依存する DNN の解釈可能性は依然として限定的であり，ブラックボックスとして扱われることが多い．この透明性の欠如により，DNN が何を，いつ，どのように学習するのかという深層学習のメカニズムに対する深い理解が妨げられている．複雑な画像認識タスクは一般的に，複数の意味的な画像概念を並行して学習することを伴い，異なる特徴が様々な時間スケールで学習される．従来研究では形状やテクスチャ特徴の学習が分析されてきたが，これらの概念は通常，与えられたデータセットに基づいて帰納的に定義されており，体系的な分析には制限があった．信頼性の高い分析を行うためには，適切かつ制御可能な難易度レベルと，それらを支持するための十分なデータを有する学習タスクの確立が不可欠である．これらの課題に対し，本研究では「数字」と「色」という 2 つの解釈可能な概念に着目し，サンプル間に固有のノイズを含む EMNIST Digits データセットに色情報を付加することで，100 クラスの分類タスクを構築した．その上で，標準的な条件下および追加的なラベルノイズ存在下における DNN 学習プロセスの分析を実施した．本研究の結果より，各概念の獲得難度に応じて学習のタイミングが異なること，また異なる概念の学習間に相互作用が存在することが明らかとなった．これらの知見は，深層学習における画像概念の学習プロセスに関する示唆を与えるものであり，画像ベースのタスクを超えた応用可能性を有するとともに，深層学習のダイナミクスの包括的な理解に寄与するものである．

Abstract

End-to-end learning using deep neural networks (DNNs) has demonstrated high performance across various computer vision (CV) and natural language processing (NLP) tasks. However, the interpretability of DNNs, which rely on distributed representations, remains limited, often rendering them as black boxes. This lack of transparency prevents a deeper understanding of the mechanics of deep learning, specifically regarding what, when, and how DNNs learn. In contrast, complex image recognition tasks generally involve learning multiple semantic image concepts in parallel, with different features learned at varying time scales. While previous studies have analyzed the learning of shape and texture features, these concepts are typically defined inductively based on the given dataset, limiting systematic analysis. For reliable analysis, it's crucial to establish learning tasks with appropriate, controllable difficulty levels and sufficient data to support them. To address these needs, we focused on two interpretable concepts—"numbers" and "colors"—and developed a classification task with 100 classes by adding color information to the EMNIST Digits dataset, which includes inherent noise across samples. We then analyzed the DNN learning process under standard conditions and with additional label noise. Our results reveal that the timing of learning differs depending on the difficulty of acquiring each concept and that there is an interaction between learning different concepts. These findings offer insights into the learning process of image concepts in deep learning, with potential applications beyond image-based tasks, contributing to a broader understanding of deep learning dynamics.

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究背景

深層学習は，現在多くの分野で利用されており，現代の科学技術において欠かせない技術の一つである．高性能な深層学習モデルを実現するためには，より大規模なデータセットの使用や，それに適したモデルのパラメータ設定が重要であるとされている．従来，機械学習モデルの性能は，underfitting と overfitting のトレードオフによって説明されてきた．訓練データが不足し，モデルのパラメータ数が少ない場合には underfitting が発生し，テストデータに対する性能が十分に向上しない．一方，訓練データに対してパラメータ数が過剰な場合には overfitting が生じ，テストデータに対する性能が低下する．このような現象は，一般にバイアス-バリアンスのトレードオフとして広く知られている [5]．しかし，近年の研究では，モデルのパラメータ数が非常に大きくなると，一度は overfitting によってテストデータに対する性能が低下するが，再び上昇する現象が観測された．この現象は，Belkin ら [1] によって発見され，「二重降下現象（double descent）」と名付けられた．さらに，Nakkiran らの研究では，DNN モデルにおいてパラメータ数の増加だけでなく，学習エポック数の増加によっても二重降下現象が生じることが確認されている [4]．二重降下現象は，特に訓練データに対して，ラベルノイズを付与した場合の過学習が起きやすい場合に顕著に現れる．こうした背景の中，髙橋ら [7] は ResNet18 と CIFAR-10 を用いて，二重降下現象と形状・テクスチャバイアスの関連性を示唆する研究を行った．この研究では，二重降下現象が発生する過程で，形状バイアスとテクスチャバイアスが逆転する現象が確認され，モデルの汎化性能が低下する段階において，これらのバイアスが影響していることが示された．

1.2　研究目的

この結果は，学習モデルが形状やテクスチャといった特徴を学習する過程に新たな洞察を与えたものの，色や数字といったよりシンプルな概念の学習過程については未だ明らかにされていない．そこで本研究では，異なる概念の獲得過程を明らかにするために，EMNIST Digits データセットを用い，色の概念を付与した数字データを対象として分類タスクを設定した．これにより，色クラスと数字クラスという 2 つの概念がどのように学習されるかを観測し，特にテスト誤り率に加えて，色のみの誤り率と数字のみの誤り率を同時に解析することで，複数の概念がどのように獲得されるのかを検証した．深層学習モデルにおける概念獲得メカニズムの解明は，より効率的かつ解釈可能な機械学習モデルの開発に寄与すると考え

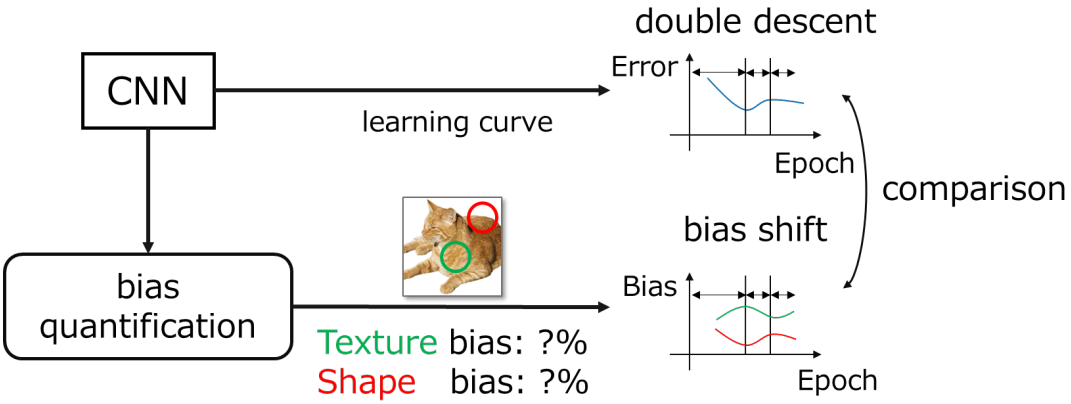


Fig. 1.1

られる．したがって，二重降下現象と概念獲得の関連を探ることで，CNN モデルにおける概念獲得過程の理解が深まり，より効果的なモデル構築への貢献が期待される．

1.3 論文構成

第 1 章「序論」では昨今における深層学習の隆盛，深層学習にける既存理論と経験的観測との乖離，深層学習における性質といった背景，及びそれらを理解するための着想と本研究の目的を述べた．

第 2 章「先行研究」では，深層学習，二重降下現象，形状・テクスチャバイアスと二重降下現象の関係性について画像認識における深層学習モデルの理解に関して述べる．

第 3 章「深層学習における二重降下現象と概念獲得」では，本研究において不可欠な深層学習における学習過程について，実験設定と結果を述べる．

第2章　先行研究

2.1　深層学習

一般に，機械学習で使用されるモデルは決定木，サポートベクターマシン（SVM），ニューラルネットワークなどが存在する．決定木は得られた予測に対して，どの説明変数が影響したのかの判断が容易であり，説明可能性が高いことで知られている．一方で，ニューラルネットワークは，パーセプトロンを筆頭に，層の増加やネットワークの複雑化が図られてきた．黎明期においては，非線形な問題をとけるように知見が盛り込まれた SVM や，生物が持つ視覚野の知見から提案されたネオコグニトロンなどの画期的な手法が提案されてきた．その中でも，ネオコグニトロンに端を発する，畳み込みニューラルネットワーク (CNN) は，LeNet[**?**] により，誤差逆伝播法が導入され，2010 年代以降には，AlexNet[**?**]，VGGNet[**?**]，ResNet[**?**]，と急速に進化を遂げてきた．このような深層化されたニューラルネットワークは興味深い性質や振る舞いを示す．しかし，そのような性質がどのような機序によって引き起こされるかについての完全な合意はとられていない．

2.2　二重降下現象

機械学習において，モデルの性能はモデルの複雑性（例えば，パラメータ数）と深い関係があり，モデルのパラメータ数が不足することによるアンダーフィッティング（Underfitting）[**?**] や，過剰なパラメータによるオーバーフィッティング（Overfitting）[**?**] などの現象が知られている．モデルの複雑性が増すにつれて，初めは性能が向上し（アンダーフィッティングを克服），その後過剰な複雑性により性能が低下するとされていた．これは U 字型のカーブ，いわゆるバイアス-バリアンス トレードオフ [**?**] として知られている．

ところが近年発見された Double Descent[**?**] と呼ばれている現象は，モデルの複雑性がさらに増すと，性能が再び向上する．つまり，最初の U 字型のカーブ（アンダーフィッティングからオーバーフィッティングへの移行）の後，さらに複雑性が増加すると，新たな性能向上のフェーズが現れるのである．過剰パラメータを持つディープニューラルネットワークが，理論的にはオーバーフィッティングを起こすべきなのに，実際には優れた汎化性能を示す場合がある [**?**, **?**].

この Double Descent は，Belkin ら [**?**] によって決定木や二層のニューラルネットワークで確認され，その後，Nakkiran ら [**?**] が，ディープニューラルネットワーク（DNN）においても観察されること，学習エポック数の増加に対しても Double Descent が起こることを示した．さらに，パラメータの枝刈りによるスパース性の増加に対しても Double Descent が起こ

ることが報告されている [**?**]．パラメータ数，学習エポック数，スパース性の増加に伴って観察される Double Descent は，それぞれ，Model-wise Double Descent，Epoch-wise Double Descent，Sparse Double Descent と呼ばれている [**?**, **?**].

2.2.1 Model-wise double descent

Yang らは，バイアス-分散のトレードオフに関する古典的な理論を，広範な実験を通して再検討した [**?**]．彼らは，分類理論が予測するようにバイアスが単調減少する一方で，分散は単峰性の挙動を示すことを発見した．このバイアスと分散の組み合わせは，3 つの典型的なリスクカーブパターンを示唆しており，すでに報告されている多くの実験結果と一致している．また，Somepalli らは，新たな決定境界可視化手法を提案し，二重降下におけるエラーの悪化する領域において，決定境界が断片化していることを報告している [**?**]．さらに，Curth らは決定木などの Belkin らが二重降下を観察した条件において，2 つの次元の軸による U 字のカーブの重ね合わせによって二重降下が起きると報告し，深層学習における二重降下においても，この観点は良い指針になることを示唆している [**?**].

2.2.2 Epoch-wise double descent

統計的シミュレーションの結果から, 学習過程における二重降下に関するいくつかの仮説が浮かび上がってきた．これらの仮説はデータの特徴に焦点を当てている．例えば, Stephenson らは，二重降下は遅いが有益な特徴によって起こると仮定し，理想的な線形モデルにおいてデータの主成分を除去することで二重降下の挙動を除去できることを示している [6]．一方，Pezeshki らは実験により，異なる時間スケールで学習された特徴が二重降下を引き起こすことを発見している [**?**]．さらに，Heckel らは，モデルの異なる部分が異なるエポックで学習することによる，複数のバイアスと分散の重複トレードオフが二重降下を引き起こすとしている [**?**]．そのうえで，層間で学習率を変えることで二重降下を緩和できることを実証している．

2.2.3 Sparse double descent

モデルのスパース性が高まるにつれて，つまり多くのパラメータがゼロまたは非常に小さくなるにつれ，まず性能の向上が見られる [**?**, **?**]．しかし，ある点を境に性能は低下する．さらにスパース性を高めると，性能は再び向上する．このことは，ネットワークの刈り込みのような方法で達成可能な適度なスパース性が，モデルのオーバーフィッティングを抑制し，汎化性能を向上させることを示唆している．また，枝刈り前のモデルの大きさに関わらず，枝刈り後の精度は一定である可能性が示唆されている [**?**].

2.3 画像認識における形状・テクスチャ

Geirhos らは，ImageNet で学習した CNN が，分類のために特に画像のテクスチャを重視することを示した [3]．彼らは，相反する形状とテクスチャ情報を持つ画像を CNN に入力し，出力が形状ベースのラベルとテクスチャベースのラベルのどちらに一致するかをチェックした．この結果に基づいて，CNN が認識において形状とテクスチャのどちらを優先するかを分析した．一方，Islam らは，ニューロンの潜在表現に基づくモデルにおいて，形状とテクスチャのどちらを重視するかを定量的に判断する方法を提案した [**?**]．この方法によって，CNN がどの特徴に偏重するのかを定量的に分析することができる．さらに，Ge らは人間の視覚系のモデル化を試み，Human Vision System (HVS) を開発した．HVS は，画像分類時にどの特徴（形状，テクスチャ，色など）が最も重要な役割を果たすかを定量的に評価可能である [2].

本研究は，画像理解と二重降下における CNN に関する先行研究を基礎としている．Islam らの手法を使用して，CNN 学習中のテクスチャと形状情報に関する知識の獲得と二重降下現象との関係を明らかにすることを試みた．Islam らの手法を利用し，最終畳み込み層における形状・テクスチャそれぞれをエンコードするニューロン数を推定，推定した割合の値を形状・テクスチャ偏重度としている．

第3章　深層学習における二重降下現象

3.1　はじめに

本研究では，深層学習における二重降下現象を観察するために Nakkiran ら [**?**] の条件を使用している．本章では，学習時における具体的な実験設定を述べるとともに，その条件を使用した Model-wise Double Descent, Epoch-wise Double Descent の追試結果を示す．

3.2　Nakkiran's setting

ResNet18 を使用し，CIFAR-10 [**?**] を学習させる．学習データにはラベルノイズとデータ拡張を加える．ラベルノイズは，学習データの正しいラベルを p の確率でランダムに別のラベルに変更する．データ補強では，画像の上下左右に 4 ピクセルのマージンを加え，32x32 のサイズにトリミングし，画像をランダムに水平反転させる．バッチサイズを 128 に設定し，損失関数として CrossEntropyLoss を用いる．最適化には Adam [**?**] を用い，学習率は 0.0001 とする．

3.2.1　Model-wise Double Descent

一般的な ResNet18 の各ブロックの出力チャネル数が [64, 128, 256, 512] であるのに対し，[1×k, 2×k, 3×k, 4×k] と横幅を可変にすることでパラメータ数を制御し二重降下を観察している．以降横幅可変な ResNet18 を ResNet18k と呼称する．ラベルノイズは $p = 0.15$ として，$k = 1 \sim 64$ に設定した ResNet18k をそれぞれ 4,000 epoch 学習させ，最終的なテスト誤り率をグラフに図示する．実際の結果を Fig. 3.1 に示す．$k = 5$，$k = 10$ の付近でテスト誤り率が下降から上昇，上層から下降と切り替わり，二重降下の曲線が観察できる．

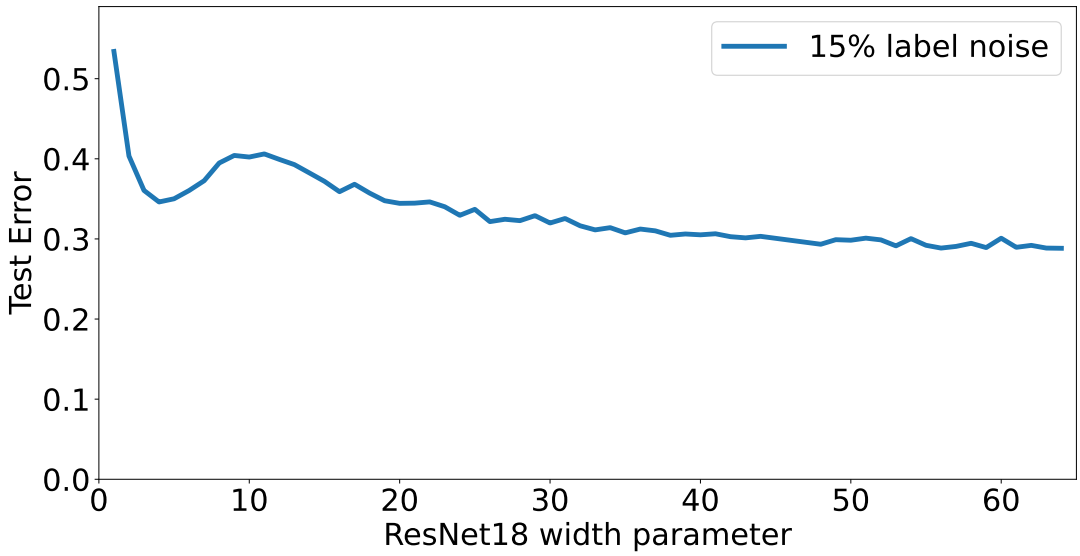


Fig. 3.1: Model-wise Double Descent by Nakkiran's setting

3.2.2 Epoch-wise Double Descent

先述した，ResNet18k を $k = 128$ で使用し，同様に 4,000epoch 学習させ，学習過程のテスト誤り率の推移を観察する．実際の結果を Fig. 3.2 に示す．20 から 30epoch 目，60 から 70epoch 目付近でテスト誤り率が下降から上昇，上層から下降と切り替わり，二重降下の曲線が観察できる．

以降の実験では，一般的な ResNet18($k = 64$ に設定した ResNet18k とほぼ同等) を先述した条件で学習，学習過程における二重降下を観察している．

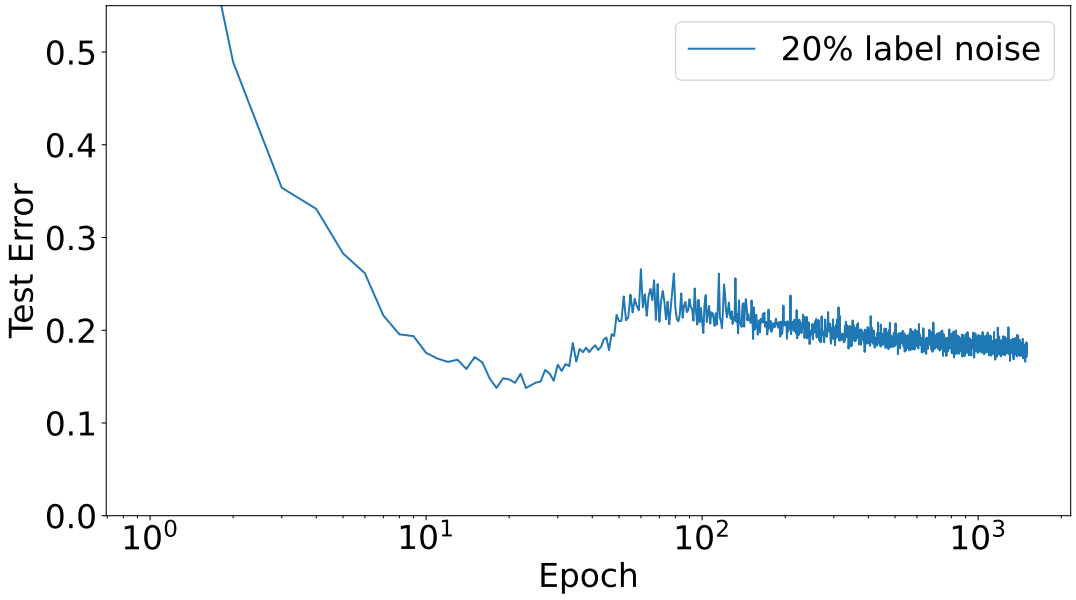


Fig. 3.2: Epoch-wise Double Descent by Nakkiran's setting

第4章　自然画像が持つ特徴に着目した分析

4.1　概要

本章では，深層学習における Epoch-wise Double Descent と自然画像が持つ形状・テクスチャ特徴との関係を調査する方法を説明する．Fig. 1.1 はこの方法の概要を示している．まず，二重降下が起きる条件で CNN を学習し，学習曲線の推移を観察する．さらに，各エポックの形状・テクスチャ偏重度を Islam らの方法により定量化し，学習過程での推移を同様に観察する．このようにして，二重降下の推移と偏重度の推移を比較する．さらに定量的な評価を行うために，二重降下を三つのフェーズに分け，各フェーズでのテスト誤り率と形状・テクスチャ偏重度との相関係数を評価する．以降では Epoch-wise Double Descent の観察方法，二重降下の各フェーズの分割方法，形状・テクスチャ偏重度の算出方法についてそれぞれ説明する．

4.2　二重降下現象のフェーズ分割

本研究では，二重降下とモデルの形状テクスチャの偏りとの関係を分析するために，二重降下におけるテスト誤差の推移に基づいて以下の 3 つのフェーズに分割する．**Phase1**: 学習開始からテスト誤差が最小になるまで．**Phase2**: Phase1 が終了してから，テスト誤差が再び減少するまで．**Phase3**: Phase2 が終了してから，以降の区間．

これらのフェーズを決定するために，勾配ベースの方法を利用する．具体的には，エポック間のテスト誤り率を監視し，連続するエポック間の差 Δe を $\Delta e = |e_i - e_{i+5}|$ として計算する．任意のエポック i において，差 Δe が指定された閾値 θ 以下であれば，テスト誤差は安定しているか，わずかに改善されている．このときの最小エポック番号までの区間を「Phase1」と定義する．‘Phase2’ については，‘Phase1’ の直後のエポック番号から，差が θ 以下となる最小のエポックまでの区間とする．‘Phase3’ は，‘Phase2’ 以降の区間を示す．実験では，閾値 θ は 0.1 に設定した．使用した実験セットアップでは，経験的に二重降下の二回目の降下は一回目の降下より低いテスト誤り率を持たないため，フェーズは上記のプロセスで分割可能である．

4.3　形状・テクスチャ偏重度の定量化

我々は，CNN の最終畳み込み層における形状・テクスチャ特徴を符号化するニューロン数 (=次元数) を Islam らの方法 [**?**] によって推定し，その割合をモデルが持つ形状・テクス

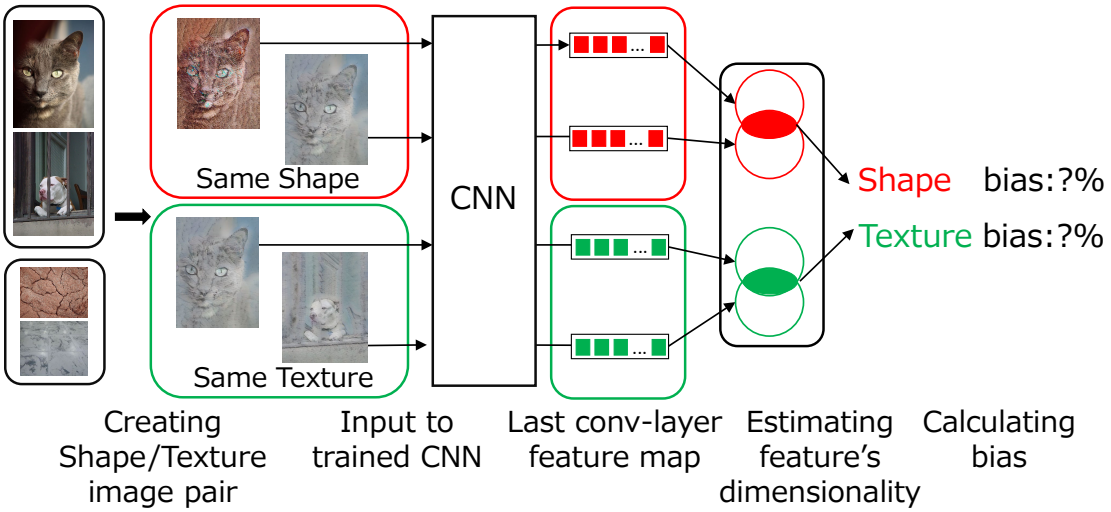


Fig. 4.1: Overview of the process of calculating the shape/texture bias using the method of [?].

チャ偏重度と定義する．次元の推定には，Islam らのプログラム[1]を使用した．形状・テクスチャ偏重度を計算するフローを Fig. 4.1 に示す．

形状・テクスチャ偏重度の定量化に使用するデータセットの作成方法について述べる．この目的のために，PASCAL VOC 2012 データセット [?] と Describable Textures Dataset [?] を用いる．PASCAL VOC 2012 データセットからは，単一のオブジェクトを含む画像のみを選択し，Describable Textures Dataset からは，ランダムに 5 つのテクスチャ画像を選択する．Describable Textures Dataset から選択されたテクスチャ画像をスタイルとして使用し，PASCAL VOC 2012 データセットから選択されたすべての画像に対してスタイル変換を実行する．このスタイル転送には，AdaIn transfer アルゴリズム [?] を利用する．このようにして作成されたデータセットを Stylized PASCAL VOC 2012（SVOC）と呼ぶ．

SVOC を用いて偏重度を定量化する方法について詳しく説明する．SVOC を使うことで，共通の形状特徴を持つ画像ペアと共通のテクスチャ特徴を持つ画像ペアをサンプリングすることができる．これらのペアを定量化したい CNN モデルに入力すると，最終的な畳み込み層のニューロンから両方の共通特徴セットに対する特徴マップペアが得られる．これらの特徴マップを用いて，以下の式から各特徴の相関係数を計算し，偏重度を求める．

実際の計算過程を述べる．形状情報が共通する画像ペアを入力して i 番目のニューロンから得られる特徴マップペアを z_i^a, z_i^b とする．それら特徴マップペアから相関係数を計算，計算された相関係数を ρ_i^{shape} とする．同様に同様に，テクスチャ情報が共通する画像ペアを入力し，出力される特徴ベクトルのペアから $\rho_i^{texture}$ を求める．そして，ニューロンのごとの ρ_i^{shape}，$\rho_i^{texture}$ それぞれの総和，形状とテクスチャのとベースライン値（= ニューロン数 $|z|$）から，ソフトマックス関数を用いて形状とテクスチャの偏重度を決定する．

この方法は，i 番目のニューロンがある概念（例えば形状）をエンコードしている場合に，

[1] https://github.com/islamamirul/shape_texture_neuron

形状情報共通した画像ペアを入力すると，z_i^a, z_i^b の相互情報量は高くなり，相互情報量は (1) の式のように相関係数から下界が求まることから，相関係数を用いて計算される．

$$\mathrm{MI}\left(z_i^a, z_i^b\right) \geq -\frac{1}{2}\log\left(1-\rho_i^2\right), \quad \text{where } \rho_i = \frac{\mathrm{Cov}\left(z_i^a, z_i^b\right)}{\sqrt{\mathrm{Var}\left(z_i^a\right)\mathrm{Var}\left(z_i^b\right)}} \tag{4.1}$$

Islam らの実装では，論文内における説明とは異なり，高速化のため，ニューロンごとに ρ_i^{shape}，$\rho_i^{texture}$ を計算していないことに注意が必要である．

第5章　実験

5.1　はじめに

本節では，二重降下と形状・テクスチャ特徴との関係を検証するため，以下の検証を行う：(1) 先行研究において Epoch-wise Double Descent を確認した設定を参考に，テスト誤り率，形状・テクスチャ偏重度それぞれの推移を比較する．また，section 4.2 で定義したそれぞれの Phase において，どの程度相関があるかを定量的に調べる． (2)(1) に基づき，二重降下とテクスチャ・形状の偏りの関係の理解を深めるために，詳細なアブレーション実験を行う．本章の実験で使用している学習プログラムは，付録 A に記載する．

5.2　Nakkiran's setting result (Fig. 5.1)

Nakkiran [**?**] の実験設定 (section 3.2) を参考に，テスト誤り率，形状・テクスチャ偏重度それぞれの推移の比較を行った．二重降下とモデルの形状・テクスチャ偏重度を比較した結果を Fig. 5.1 に示す．青い線はテスト誤差の二重降下の推移を，赤い線はモデルの形状偏重度の推移を，緑の線はテクスチャ偏重度の推移を表している．形状・テクスチャ偏重度の推移においては，変動におけるノイズの軽減のために，5 項移動平均を取っていることにテスト誤り率，形状偏重度それぞれの推移を比較すると，Phase 1 では下降し，Phase 2 では上昇，Phase 3 では再度下降するといった相関がみられた．また，テスト誤り率とテクスチャ偏重度には逆の相関がみられている．

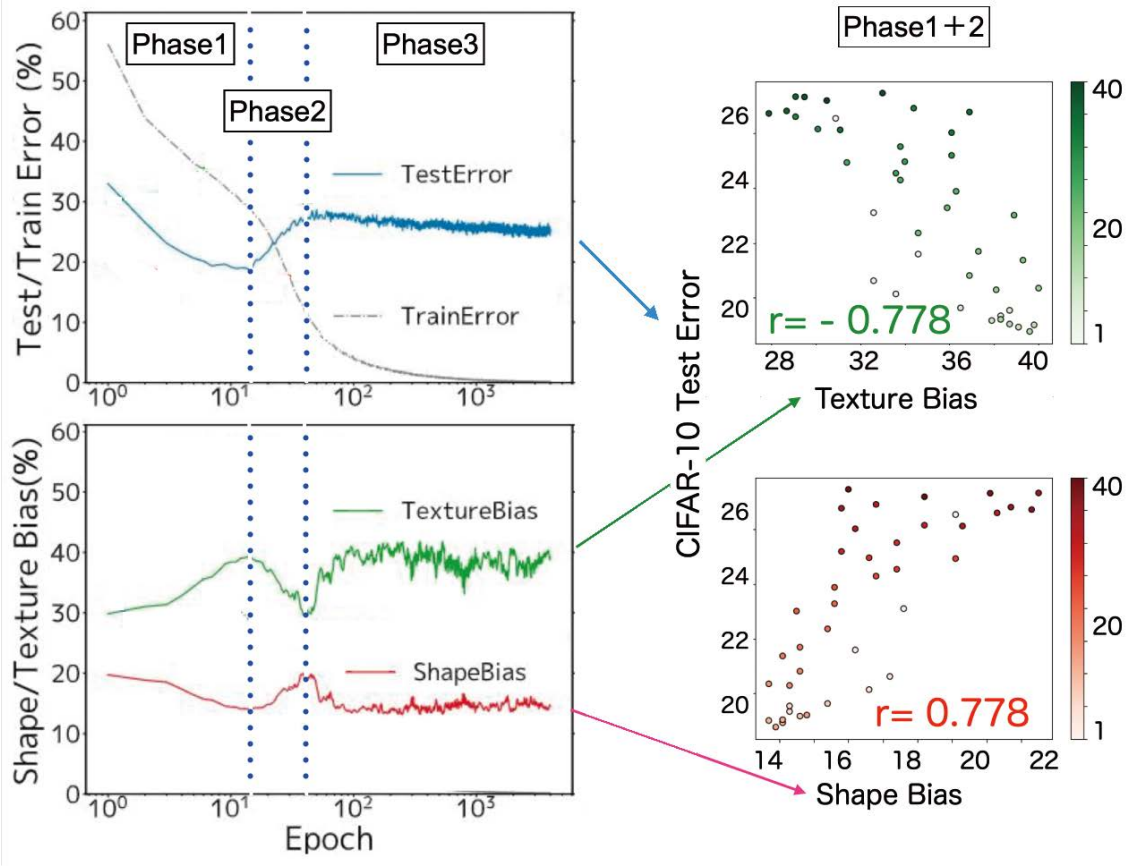


Fig. 5.1: Schematic overview of this study. Top left: Learning curve of the CIFAR-10 image recognition task where double descent was observed; test errors were divided into three phases based on their temporal differentiation. The experimental results are a reproduction of double descent under the same conditions as previously reported by Nakiran *et al.*[**?**]. Bottom left: This records the model's shape/texture bias during the aforementioned learning process. It shows the synchronous changes between test errors and shape/texture biases. Right: A scatter plot of test error and shape/texture bias. Especially in Phase 1 and Phase 2, there is a positive correlation between test error and shape bias, and a negative correlation between test error and texture bias.

5.3 Correlation analysis in each phase of the double descent (Tab. 5.1)

4.3 節で定義した Phase1，Phase2，Phase3 におけるテスト誤り率と形状・テクスチャ偏重度の相関係数を計算し，より詳細な評価を行った．これらの結果を Tab. 5.1 に示す．Phase1 と Phase2 におけるテスト誤り率と形状偏重度との相関係数はそれぞれ 0.898 と 0.771 であり，正の相関があることがわかる．逆に，Phase1 と Phase2 におけるテスト誤り率とテクスチャ偏重度との相関係数は−0.829 と−0.797 であり，負の相関を示している．Phase3 では，形状偏重度の相関値は−0.026，テクスチャ偏重度の相関値は 0.118 であり，相関は見られなかった．これらの結果から，Phase1 と 2 では相関が見られたが，Phase3 では相関が見られなかった．形状とテクスチャの関係を簡単に理解するために，形状とテクスチャの相関値を Score と呼ばれる指標で表す．Score は，これらの相関係数の絶対値の平均である．Score が高いほど，テスト誤り率と形状/テクスチャ偏重度それぞれとの相関が強いことを示す．以

降の実験では，今回の条件において高い相関を示した Phase1，2 と Phase3 それぞれの区間で相関係数と Score を算出し，定性的な分析に利用する．

Table 5.1: The correlation coefficients and scores for the three phases divided according to the method defined in (section 4.2). It shows the Epoch range used to calculate the correlation coefficients, the correlation coefficients calculated from Test Error and Shape bias, the correlation coefficients calculated from Test Error and Texture bias, and the correlation score computed from these two correlation coefficients.

Phase	Epoch range	Shape corr	Texture corr	Score
Phase1	2 - 12	0.898	−0.829	0.863
Phase2	12 - 41	0.771	−0.797	0.784
Phase3	41 - 1,000	−0.026	0.118	0.072

5.4 詳細なアブレーション実験

5.4.1 概要

本節では，先に述べた条件から一部を変更し，実験を構成する各条件が二重降下と偏重度の推移にどのように影響を与えるかを検証する．具体的には，以下の条件を変更した：”事前学習の有無”，”使用データセット”，”ResNet Family 内でのモデル変更”，”使用 CNN モデル”，”バッチサイズ”，”ラベルノイズの割合”，”シード値”．実際に変更した条件と該当する図の一覧を Tab. 5.2 に示す．以降の実験では，実験の高速化のために，学習回数を 1,000 epoch とした．

Table 5.2: List of changed conditions and corresponding fig. numbers in ablation studys.This table shows only the conditions changed from the base in bold.Blank spaces indicate the same as above.

pre-train	Model	dataset	batchsize	Label Noise	seed	Fig. No
ImageNet	ResNet18	CIFAT-10	128	0.2	42	5.1
None	ResNet18	CIFAT-10	128	0,2	42	5.2
ImageNet		**CIFAT-100**				5.3
	ResNet34	CIFAT-10				5.4
	ResNet50					5.4
	DenceNet					5.5
	MobileNet					5.5
	EfficientNet					5.5
	ResNet18		**8**			5.6
			16			5.6
			64			5.6
			128	**0.4**		5.7
				0.6		5.7
				0.2	**0**	5.8
					1	5.8

5.4.2 Init parameter (see Fig. 5.2)

ベースラインの条件では，先行研究において，ImageNet における学習の文脈で形状・テクスチャに関する研究が行われていたことを考慮し，ImageNet-1k の学習済みパラメータをモデルの初期値として用いた．本実験では，ImageNet-1k とランダム値（Scratch）を比較することで，初期値に基づく形状とテクスチャの偏りの関係を検証した．その結果を Fig. 5.2 に示す．この結果から，ImageNet-1k の初期パラメータを用いた場合，形状・テクスチャ偏重度が明瞭に変化することがわかる．このような結果になった理由の 1 つとして，ImageNet を事前学習することで得られる特徴表現と，CIFAR-10 を学習することで得られる特徴表現が大きく異なることが考えられる．それによって，ImageNet を事前学習したパラメータの状態から，CIFAR-10 に適応するために，大きくパラメータの状態が変化し，その結果偏重度の推移の変化として現れたと考えられる．また，Scratch を用いた場合においても，二重降下の Phase 1 から Phase 2 への移り変わりにおいて，特にテクスチャ偏重度はわずかながら上昇から下降に転じているように見受けられる．そのため，双方の条件において，学習傾向の移り変わりが起きている可能性が考えられる．しかし，CIFAR-10 の学習時に，ImageNet で事前学習した場合に，二重降下と同期したような挙動を強くすることは，非常に興味深い現象である．このような差異について，パラメータが如何に変化していっているかという点が重要であると考えられる．

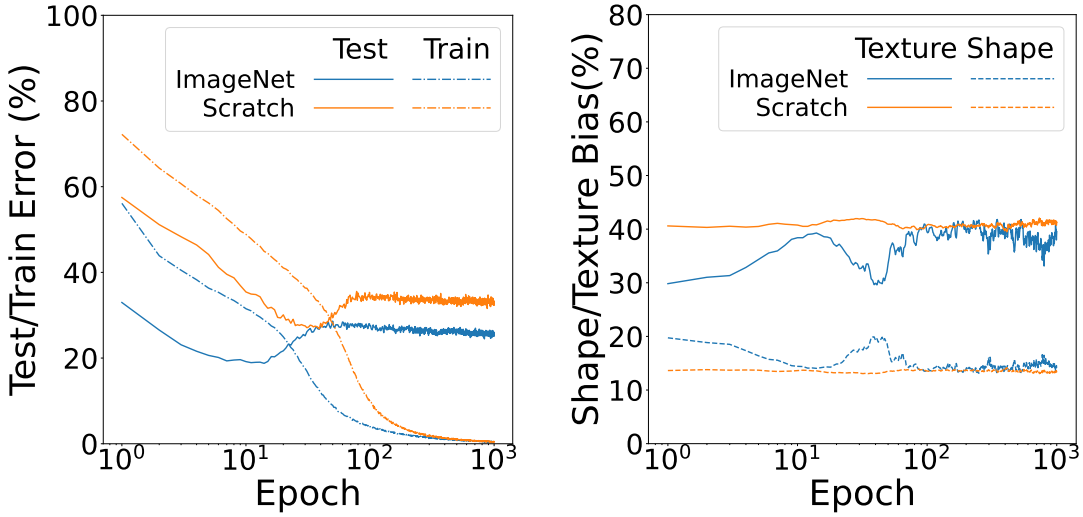


Fig. 5.2: Learning process with and without pretraining by ImageNet. Left: train/test errors Right: shape/texture bias values.

5.4.3 Dataset (Fig. 5.3 and Tab. 5.3)

データセットを変更した場合の影響を調査するために，CIFAR-10 と性質が類似している CIFAR-100 で学習し，CIFAR-10 の場合と差異を検証した．結果を Fig. 5.3 に，定量評価を Tab. 5.3 に示す．その結果，CIFAR-10 では，Phase1 と Phase2 において，形状の偏りとの相関は 0.778 であり，テクスチャの偏りとの相関は-0.778 であった．一方，CIFAR-100 では，形状の偏りとの相関は-0.689，テクスチャの偏りとの相関は 0.745 となり，逆相関の関係を示した．CIFAR-10，100 ともに，Phase3 における形状とテクスチャの相関は，スコア結果からほぼ無視できる．CIFAR-100 で観測された逆の相関は，クラス数，クラスの種類，1 クラスあたりの枚数などの要因に影響された CIFAR-10，CIFAR-100 特有の特性の差異によるものであると推測される．一方で，逆相関をみせることから，二重降下と偏重度の推移に相関を引き起こす，共通の特性が存在する可能性を示唆している．

また，CIFAR-100 から 10 クラスを抜き出して同様の実験を行った場合にも，CIFAR-100 の場合と類似した形状・テクスチャ偏重度の推移を示した．これによってクラス数の差異が直接関係しているわけではないと考えられる．CIFAR-100 は 20 の上位クラスとそれぞれに含まれる 5 つの下位クラスで構成されている．そのため，たとえば，魚の上位クラスあり，その上位クラスには魚の質感をした 5 つのクラスが含まれていることになる．それによって CIFAR-10 とは重視される特徴に差異が出ている可能性が考えられる．

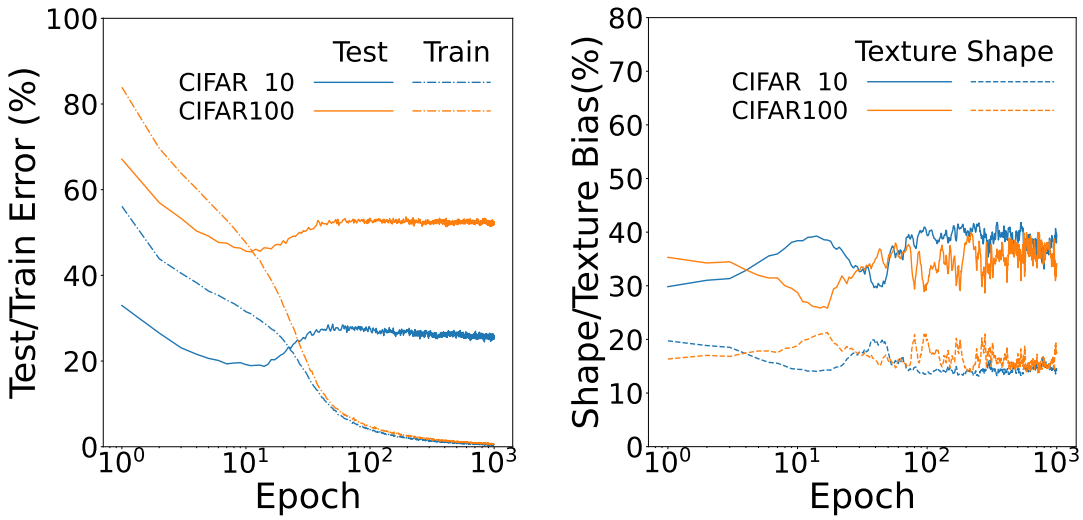


Fig. 5.3: Learning process by different tasks, CIFAR-10 and CIFAR-100. Left: train/test errors Right: shape/texture bias values.

Table 5.3: Correlation coefficients and scores in Phase 1, 2 and Phase 3 for different datasets.

CNN	Correlation of Phase1,2			Correlation of Phase3		
	SB	TB	Score	SB	TB	Score
C10	0.778	−0.778	0.778	−0.026	0.118	0.072
C100	−0.689	0.745	0.717	0.002	0.013	0.007

5.4.4 ResNet family (see Fig. 5.4 and Tab. 5.4)

モデルのパラメータ数の影響，特に ResNet family 内で一般性があるのかを検証するため，ResNet18 と同様の論文で提案されている ResNet34，ResNet50 でも実験を行った．使用するモデルを ResNet34，ResNet50 に変更した場合の結果を Fig. 5.4 に，定量的な評価のための表を Tab. 5.4 に示す．ResNet34 においては，偏重度の推移に相関があるように見られ，Phase1,2 の Score が 0.517 と ResNet18 の場合と同様の相関を定量的にも示す．しかし，ResNet50 においては，偏重度の推移には定性的には Phase2 での変化が見られず，定量的にも相関がみられない．この結果の理由として，ResNet は，ResNet18，ResNet34，ResNet50 とパラメータ数が増加するため，パラメータ数の上昇に従って，相関が消える可能性が考えられる．また，使用した ResNet の構造の差異を考えると，ResNet18，ResNet34 は basic block と呼ばれる構造から形成されているのに対して，ResNet50 は bottleneck と呼ばれる構造から形成されており，この点が影響を与えた可能性も考えられる．

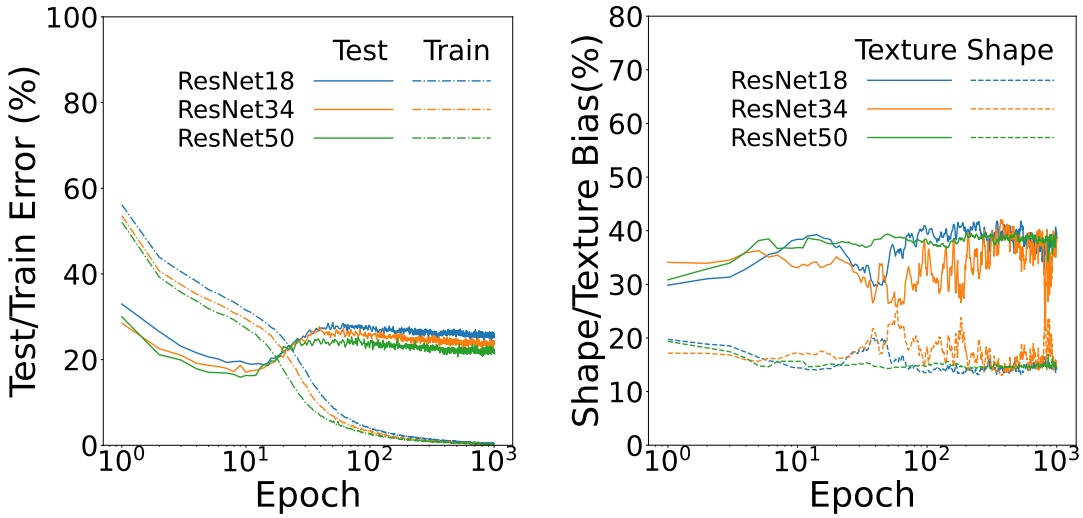


Fig. 5.4: Learning process under various size conditions of ResNet family. Left: train/test errors Right: shape/texture bias values.

Table 5.4: Correlation coefficients and scores in Phase 1, 2 and Phase 3 for different ResNet Family.

CNN	Correlation of Phase1,2			Correlation of Phase3		
	SB	TB	Score	SB	TB	Score
ResNet18	0.778	−0.778	0.778	−0.026	0.118	0.072
ResNet34	0.498	−0.536	0.517	0.165	−0.281	0.223
ResNet50	−0.153	0.136	0.144	−0.097	0.064	0.080

5.4.5 CNN models (see Fig. 5.5 and Tab. 5.5)

ResNet とは異なる CNN モデルを使用した場合における，二重降下と形状・テクスチャ偏重度の推移への影響を検証した．具体的には, DenseNet121[1] [**?**], MobileNetV2[2] [**?**], EffecientNet B0[3] [**?**] を使用した． ResNet が 4 つのブロックと, 1 ブロックごとに接続される skip connection から構成されるのに対して，DenseNet は 4 つの block と各 block から異なるすべての block に接続される skip connection から構成される CNN，MoblieNetV2 はモバイル向けに効率アが図られた CNN，EfficientNet は Neural Architecture Search と呼ばれる手法により，最適化が図られた CNN である．異なる CNN を使用した場合の結果を Fig. 5.5 に，定量的な評価のための表を Tab. 5.5 に示す．DenceNet121，MobileNetV2 を使用した場合において，緩やかであるが，推移が相関しているように観察でき，Score も 0.324，0.509 と，定量的にも相関がみられた．反対に，EfficientNet B0 においては特に相関は見られなかった．この点に関して，DenseNet121，MobileNetV2，および EfficientNet B0 の間に何らかの差異が存在すると考えられるが，これらのモデルの構造的な明確な差異は見受けられないため，原因は不明である．また，EfficientNet の場合においては今までのどの条件とも異なる偏重度の推移を見せており，特徴の学習過程という観点において，その他のモデルと大きく異なる可能性が示唆される．

Table 5.5: Correlation coefficients and scores in Phase 1, 2 and Phase 3 for different CNN models.

CNN	Correlation of Phase1,2			Correlation of Phase3		
	SB	TB	Score	SB	TB	Score
DenseNet	0.326	−0.322	0.324	0.293	−0.289	0.291
MobileNet	−0.506	0.511	0.509	−0.016	0.036	0.026
EfficientNet	−0.029	0.000	0.014	0.316	−0.343	0.330

[1] https://pytorch.org/vision/0.9/_modules/torchvision/models/densenet.html#densenet121

[2] https://pytorch.org/vision/0.9/models.html?highlight=mobilenet#torchvision.models.mobilenet_v2

[3] https://pytorch.org/vision/0.14/models/generated/torchvision.models.efficientnet_b0.html?highlight=efficientnet#torchvision.models.efficientnet_b0

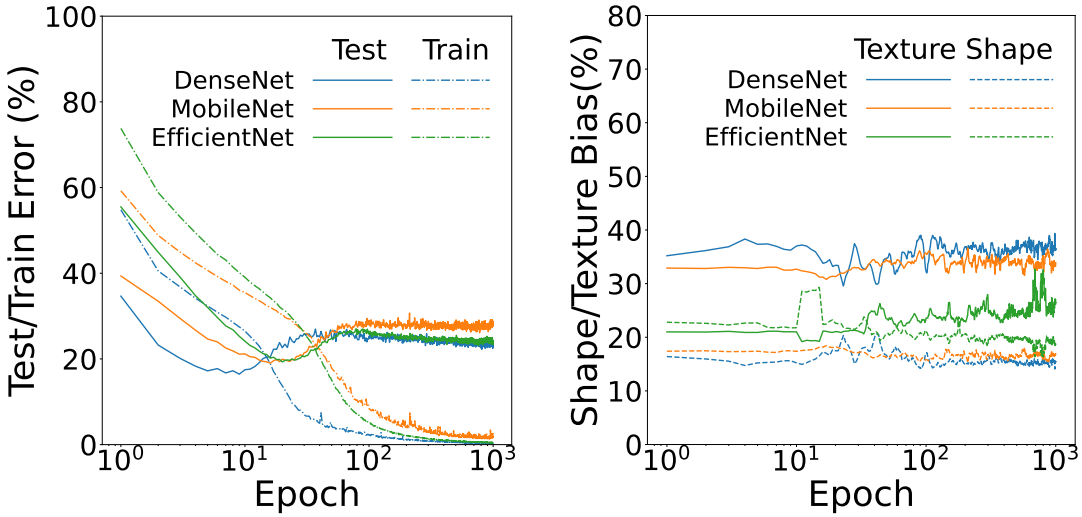


Fig. 5.5: Learning process under various size conditions of CNN models. Left: train/test errors Right: shape/texture bias values.

5.4.6 Batch size (see Fig. 5.6 and Tab. 5.6)

バッチサイズを変更した場合における，二重降下と形状・テクスチャ偏重度の推移への影響を検証した．異なるバッチサイズを使用した場合の結果を Fig. 5.5 に，定量的な評価のための表を Tab. 5.5 に示す．バッチサイズの減少に従って，テスト誤り率の曲線が右上にシフトしている．また，形状・テクスチャ偏重度の推移も，曲線の大きく落ち込んでいるところに注目すると，バッチサイズの減少に従って右にシフトしているよう見受けられる．定性的には，どの条件も相関を示していない．また，バッチサイズ 4 においては，定義した手法においては，Phase を分割不可能であったため，データなしとしている．

このような二重降下の挙動の差異について，バッチサイズの低下によって epoch あたりのイテレーション数は増加するため，バッチサイズが少ないほど，二重降下の挙動が右にシフトするのは予期しない結果となった．しかし，バッチサイズの増加によって学習の安定性が上昇するため，この影響であると考えられる．加えて，バッチサイズの増加につれて，右にシフトするのかについては，検証の余地がある．

Table 5.6: Correlation coefficients and scores in Phase 1, 2 and Phase 3 for different CNN models.

batch size	Correlation of Phase1,2			Correlation of Phase3		
	SB	TB	Score	SB	TB	Score
4	N/A	N/A	N/A	N/A	N/A	N/A
16	0.249	−0.206	0.227	−0.186	0.154	0.170
64	0.090	−0.065	0.077	0.108	−0.101	0.105

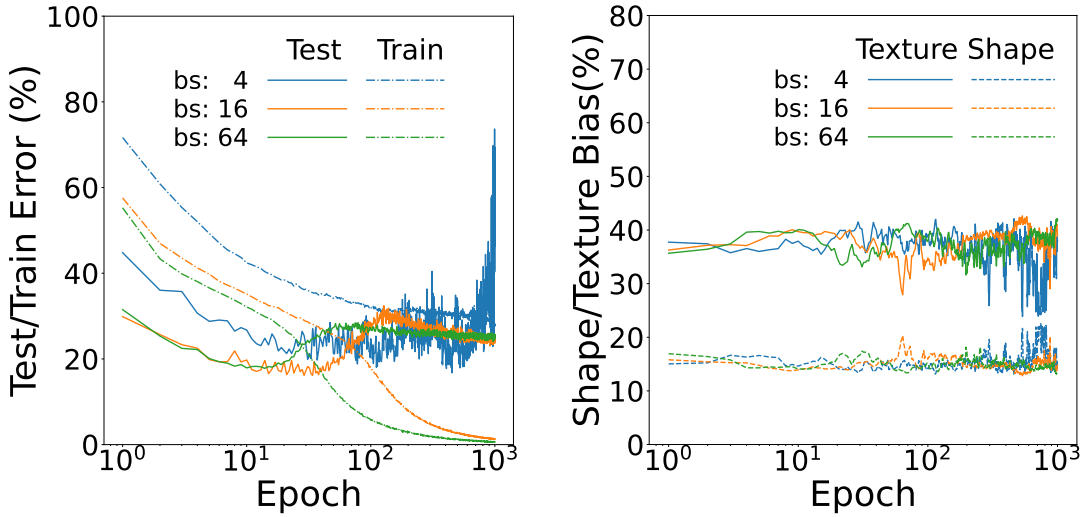


Fig. 5.6: Learning process under various batch size conditions. Left: train/test errors Right: shape/texture bias values.

5.4.7 Label Noise (see Fig. 5.7 and Tab. 5.7)

ラベルノイズの増加は，二重降下の観測において重要なパラメータの一つである．そのため，特に偏重度の推移に対して，どのような影響を与えるか，ラベルノイズの割合を変更し検証を行った．ラベルノイズの増加による，二重降下と形状・テクスチャ偏重度の推移への影響を検証した結果を Fig. 5.7 に示す．ラベルノイズの割合は，20%に加えて，40%，60%と変化させた条件で行った．相関係数の定量評価を Tab. 5.7 に示す．テスト誤り率の推移からわかるように，ラベルノイズが大きくなるにつれて二重降下の Phase2 における変動幅が大きくなっている．しかし，形状とテクスチャ偏重度を見ると，ラベルノイズの割合に関係なく，形状・テクスチャ偏重度の明確な傾向が観察される．特に，テクスチャ偏重度では，ラベルノイズが大きくなるにつれて，上昇から下降へのシフトのタイミングが遅れているように見える．各条件の相関を比較すると，40%の場合には相関は見られていないが，60%の場合は Score が 0.579 と，相関がみられている．このことから，ラベルノイズが大きくなるにつれて，偏重度の推移が遅くなる可能性がある．

Table 5.7: Correlation coefficients and scores in Phase 1, 2 and Phase 3 on different label noise. It shows the correlation coefficients (SB, TB) between test error and shape/texture bias and the score calculated from these two correlation coefficients.

Label Noise	Correlation of Phase1,2			Correlation of Phase3		
	SB	TB	Score	SB	TB	Score
20%	0.778	−0.778	0.778	−0.026	0.118	0.072
40%	0.004	−0.091	0.048	0.451	−0.487	0.469
60%	−0.560	0.598	0.579	0.122	−0.142	0.132

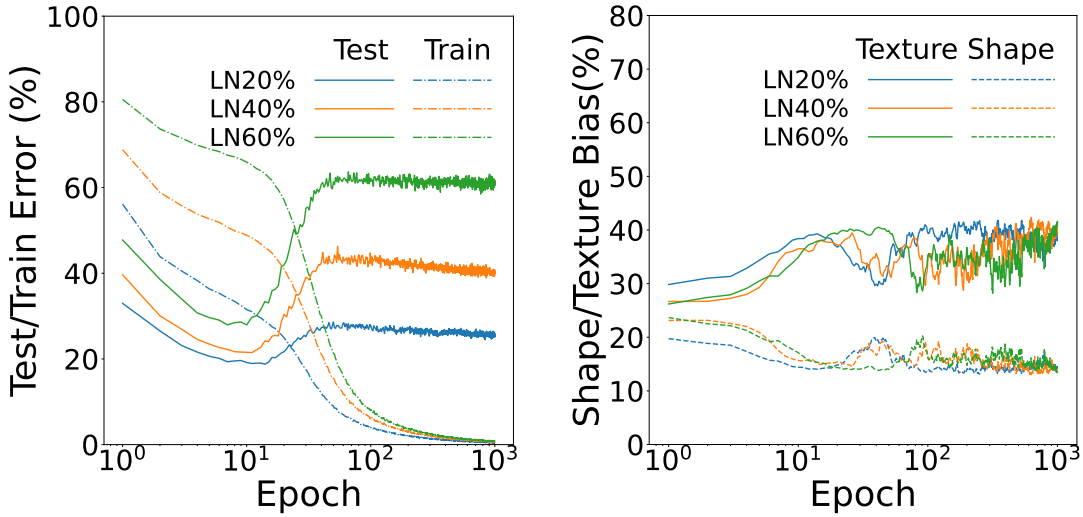


Fig. 5.7: Learning process under various label noise conditions. Left: train/test errors Right: shape/texture bias values.

5.4.8 Seed (see Fig. 5.8 and Tab. 5.8)

機械学習の実験においては，再現性を担保するために，シード値を固定して，実験を行う場合がある．そのため，シード値を変更した場合における二重降下と形状・テクスチャ偏重度の推移への影響を検証した．使用したシード値は，元の条件である 42 に加えて 0，1 を使用した．0，1，42 は機械学習の分野においてもっともよく使用されるシード値である．シード値を変更した場合の結果を Fig. 5.8 に，定量的な評価のための表を Tab. 5.8 に示す．シード値が 42 の場合において，ベースラインの結果と異なっているが，異なる計算機環境で実験で行った結果であることに留意されたい．すべての場合において，二重降下がほとんど一致している．一方で，例えばテクスチャ偏重度は，すべての条件において，上昇して下降する傾向が見られる．しかし，定量的には，すべての条件において二重降下と形状・テクスチャ偏重度との相関は捉えられなかった．

事前学習したパラメータを使用する場合，事前学習をしたパラメータを読み込んだのちに，モデルが持つ全結合層のみ再度の初期化を行うことが一般的である．この実験では，シード値を変更しているが，畳み込み層は事前学習したパラメータを使用しているため，モデルが持つ全結合層のパラメータのみに差異があると考えられる．そのため，全結合層におけるパラメータの初期値が，二重降下にはさほど影響を与えないが，偏重度の推移には大きく影響を与えていると推察される．

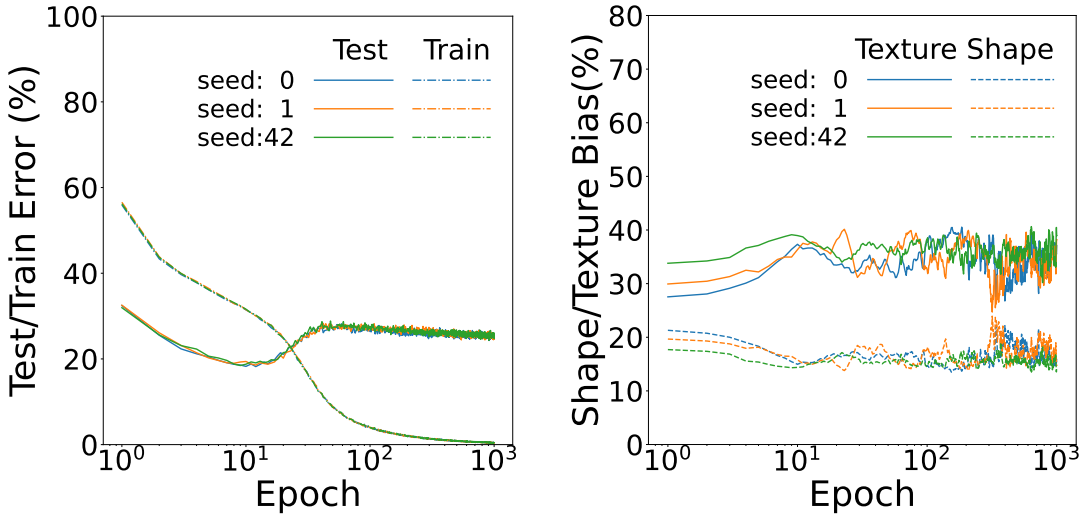


Fig. 5.8: Learning process under various seed conditions. Left: train/test errors Right: shape/texture bias values.

Table 5.8: Correlation coefficients and scores in Phase 1, 2 and Phase 3 on different label noise. It shows the correlation coefficients (SB, TB) between test error and shape/texture bias and the score calculated from these two correlation coefficients.

Seed	Correlation of Phase1,2			Correlation of Phase3		
	SB	TB	Score	SB	TB	Score
0	0.034	−0.178	0.105	−0.132	0.133	0.132
1	0.040	−0.013	0.026	−0.123	0.087	0.105
42	−0.015	−0.087	0.051	0.069	0.017	0.043

第6章　層ごとの学習過程に着目した検証

6.1　はじめに

Heckel らは，モデルの異なる部分が異なるエポックで学習とし，層ごとに学習率を変更することで二重降下が緩和することを示している [**?**]．本章では，Fig. 5.1 を観察した条件において，層ごとの特徴学習過程に着目した複数の検証を行う．今後の実験の理解を助けるために，ResNet18 の構造を説明する．He らの論文の Table 1 を Tab. 6.1 に示す．ResNet18 は，大まかに，17 層の畳み込み層と 1 層の全結合層から構成される．また最初の層を除いた畳み込み層は，4 層ごとにブロックを構成している．以降では，1 層目の畳み込み層を conv1，4 つのブロックを浅い位置から，block1，block2，block3，block4 と呼称する．

Table 6.1: ResNet architecture (citing Tab. 1 of [10]).

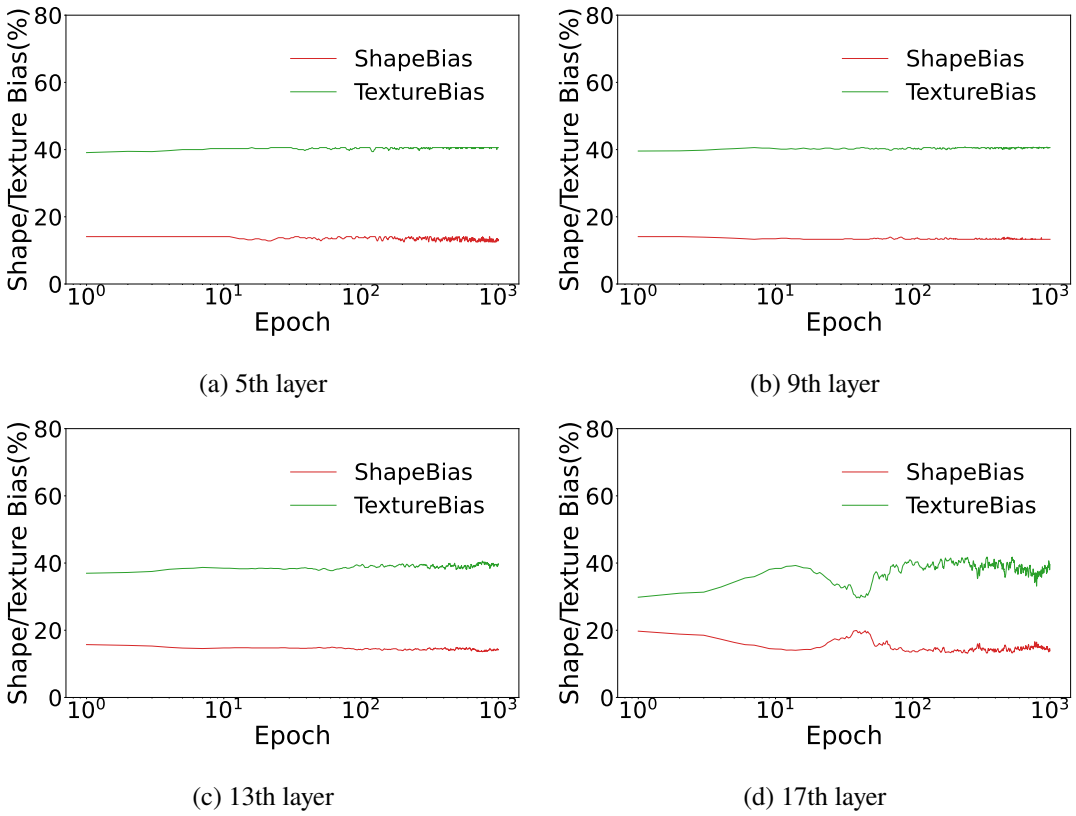
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

6.2　階層ごとの偏重度

本研究では，形状・テクスチャ偏重度をモデルが持つ最終畳み込み層が出力する特徴マップから算出している．しかし，section 4.3 の方法を使用して，異なる層からも形状・テクスチャへの偏重度が計算可能である．そのため，今まで使用していた 17 層（=block4 の最後の畳み込み層）に対して，各 block の最後の畳み込み層（5 層，9 層，13 層）で形状・テクスチャへの偏重度を算出し，最後の畳み込み層から算出した形状・テクスチャ偏重度の推移と比較した．

結果を Fig. 6.1 に示す．17 層においては形状・テクスチャ偏重度が特異な推移をしている

のに対して，5 層，9 層，13 層における形状・テクスチャへの偏重度の推移は基本的に一定で推移している．このことから，深い層のみで見られる特性があると考えられる．また，この特性を理解することで，二重降下に対してのより深い理解への糸口となる可能性がある．



(a) 5th layer

(b) 9th layer

(c) 13th layer

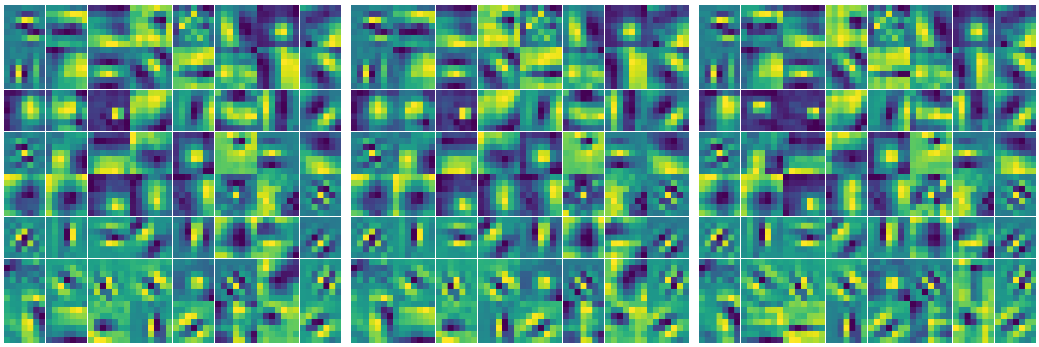
(d) 17th layer

Fig. 6.1: The shift of biases during the learning process in each layer consisting ResNet18.

6.3 浅い層のカーネル可視化

前節では，深い層にのみ見られる何らかの特性が存在する可能性が示唆された，一方で，浅い層に変化はないのか，第 1 層の可視化を通して検証を行った．使用した条件は，前節と同様に，Fig. 5.1 と同様である．二重降下の推移が切り替わる点であり，Phase の切り替わりである 13，42，そして学習が終わる 1,000 epoch 目において可視化を行った．

結果を Fig. 6.2 に示す．各フィルターを確認すると，形状テクスチャ偏重度の特異な推移が確認される条件においても，変化は微小である．浅い層の変化は少ないことから，偏重度の特異な推移に，浅い層の学習が関係している可能性は引くと考えられる．また，事前学習したときに収束したパラメータが，浅い層においては，異なるデータセット（今回は CIFAR-10）に対しても有効であったためだと考える．



(a) 13th Epoch　(b) 42nd Epoch　(c) 1,000th Epoch

Fig. 6.2: Visualization of the 1st layer in the learning process. In the setting described in Sec. 5.2, we visualize the 1st layer in the Epoch (13th, 42nd Epoch) and the 1,000th Epoch, where the double descent is divided into 3 Phases. The 1st layer at the 1,000th Epoch is visualized.

6.4 ブロック凍結実験

深い層，浅い層における学習時の性質がそれぞれ二重降下と形状テクスチャ偏重度にどのように影響するかを観察するため，モデルが持つパラメータを凍結して，Fig. 5.1 と同様に実験を行った．パラメータの凍結は，指定した層のパラメータを更新しないというものである．深い層から順に凍結した場合の結果を Fig. 6.3 に示す．深い層から凍結するパラメータを増加させていくと，二重降下の曲線は著しく変化が見られる．それに対して，形状テクスチャ偏重度は，特に fc から block3 を凍結した場合に，二重降下の下降から上昇に移り変わる部分と同期しているように見られる．浅い層から順に凍結した場合の結果を Fig. 6.4 に示す．この場合は二重降下の曲線はほぼ一致しており，形状・テクスチャ偏重度の推移も類似点が確認できる．このような結果から，深い層のパラメータが学習される場合，浅い層よりも，二重降下などに対して強い影響を与えていると考えられる．

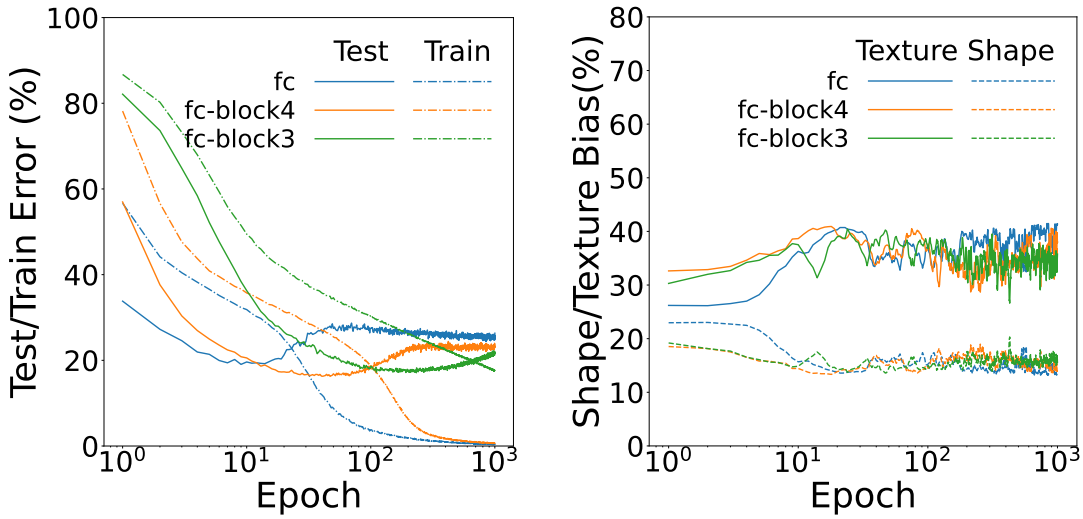


Fig. 6.3: Learning processes under deep layer parameter freezes. Left: train/test errors Right: shape/texture bias values.

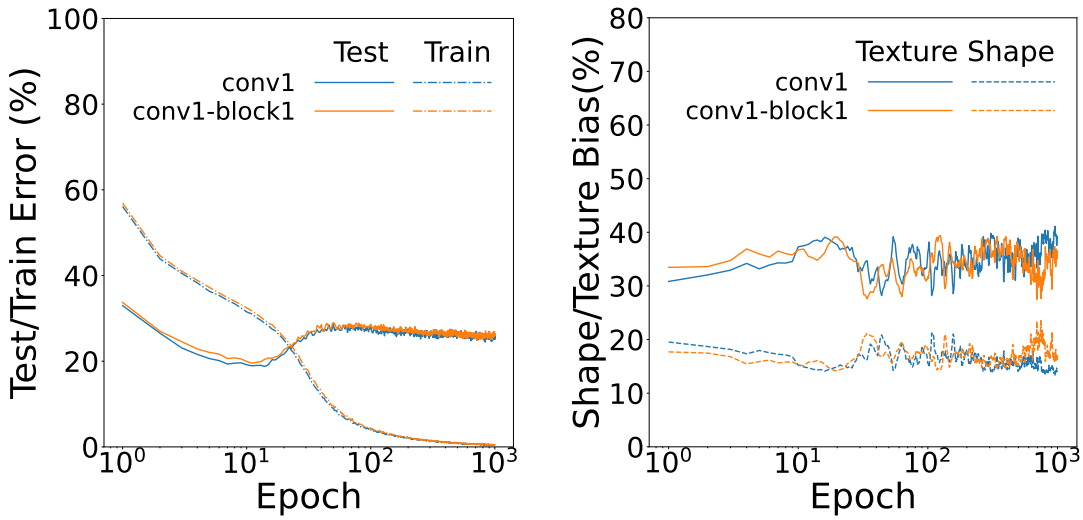


Fig. 6.4: Learning processes under shallow layer parameter freezes. Left: train/test errors Right: shape/texture bias values.

第7章　考察

本研究では，自然画像に存在する形状・テクスチャの特徴に着目し，二重下降現象との関係を分析した．その結果，ImageNet で事前学習を行った場合に，いくつかの条件において学習過程における二重降下とモデルが示す形状・テクスチャ偏重度の推移に相関がみられることがわかった．このような条件においては，二重降下における二度目の下降が始まるまでのテスト誤差と，形状・テクスチャ偏重度の推移との間に相関がみられる傾向にある．しかし，この相関は二度目の下降が始まると弱くなった．このような傾向は，二重降下を三つの段階に分割し，定性的，定量的に相関関係を評価することで判明した．

その後の実験では，CNN の最終畳み込み層で強く偏重度の独特なシフトが観察された．しかし，それ以前の中間層では，少なくとも検証を行った層においては，最終畳み込み層と同様の変化は見られなかった．この観察から，CNN の深い層は中間層とは異なる学習傾向を示す可能性が示唆される．

二重降下に関する先行研究では，データに存在する複数の特徴に影響されるのではないかという仮説が提唱されている．では，二重降下現象は形状やテクスチャーなどの特徴によって起こるのだろうか？ 我々は，その片鱗を観察したと考えている．今回の実験では，double descent を引き起こす何らかの学習傾向によって，CNN の特徴抽出傾向が影響され，ImageNet で事前学習した場合において，double descent と偏重度の推移に相関を見せたと考える．そのため，ImageNet で事前学習の有無で，パラメータの学習のされ方がどのように変わっているかを検証することは，二重降下を理解することにつながる可能性が考えられる．

実用的な観点からは，ImageNet を事前学習した条件下では，偏重度が最大，または最小となる epoch 付近でテスト誤差が最大になる可能性が示唆され，この偏重度を観察することで，早期に停止できる最適，または準最適な学習エポック数を決定できる可能性が示唆される．さらに，二重降下を引き起こす要因が，特に深い層における形状やテクスチャの特徴に対する CNN の偏重度合いにも影響を与える可能性があることを示した．

本研究では，CNN が形状やテクスチャといった画像特徴をどのように学習していくかに着目し，複数の条件において二重降下との関係を検証した．深層学習において，未解明なことは多数存在し，double descent もその一つである．そのような中で，特に深層に目を向けるべきであるとした本研究は，今後の研究の一つの方向性を指示したと考える．しかし，今回見られた現象の具体的なメカニズムを示せていないことは一つの限界点である．本研究では，CNN による形状やテクスチャといった画像特徴の学習過程に注目し，さまざまな条件下における二重降下現象との関連性を検証した．深層学習における未解明の領域は多数存

在し，二重降下現象はその一つである．本研究は，特に深層学習の深い層への理解を深めることの重要性を指摘し，将来の研究に対する一つの有望な方向性を提供するものと考えられる．しかしながら，観測された現象の具体的な機構に関しては明確な説明を提供できていない．この点は，今後の研究における主要な課題である．

第8章　結論

本稿では，epoch-wise double descent の先行研究に触発され，まだ理論的な解析がなされていない画像固有の特徴（形状・テクスチャ）と二重降下の関係に着目した．まず，画像特徴の獲得過程を追跡するため，既存の手法を用いて形状・テクスチャに対するモデルの偏重度を定量化し，この偏重度と学習中のテスト誤差の推移を比較した．結果として，ImageNet を事前学習した場合に，いくつかの条件下では，学習過程における形状・テクスチャ偏重度の推移とテスト誤り率が描く epoch-wise double descent の推移に相関がみられることがわかった．また，定量的な評価を通して，テスト誤り率の一度目の下降から上昇しきるまでの区間において特に相関が確認された．

さらに，定性的な可視化により，初期層におけるフィルタ，レイヤーの深さに基づく形状・テクスチャ偏重度の変化を示した．より深い層においては形状・テクスチャ偏重度に変化を示すが，初期層のフィルターはほとんど変化しないことが観察された．このような結果から，double descent の観点においては，深い層に着目するべきであると考えられる．我々の研究は，epoch-wise double descent と，深層学習と二重降下の一般的な分野の両方について，より広い理解に貢献すると考える．

謝辞

本修士論文は，東京電機大学システムデザイン工学研究科データ科学・機械学習研究室に所属し，前田英作教授の指導の下で執筆を行いました．3 年半の間，研究指導にとどまらず，人生の道標となるような教えを賜りました指導教員の前田英作教授に厚く感謝申し上げます．修士２年間，副指導教員の川勝真喜准教授には多くの助言を賜り，ときに先入観にとらわれていた自身が抜け出すための一助となりました．深く感謝申し上げます．研究テーマ決定の上で，初期に有益な助言をいただき，その後の自身の研究において多大な貢献をしていただいたアルムナイの鏡川 悠介氏に厚く感謝申し上げます．研究室の前任事務岩本陽子氏，現任事務佐々木理央氏には，学会参加等の事務手続きにおいて多くの助力を賜り，修士生活を支えていただきました．深く感謝申し上げます．

酒造正樹客員教授には，論文指導等，研究生活において多くの助力を賜りました．厚く感謝申しあげます．

研究活動における数回の論文提出に際して，多くの助力をいただいた同期の金田龍平氏には厚く御礼申し上げます．研究生活において，議論，相談に付き合っていただいたり，疑問に答えていただいたデータ科学機械学習研究室の先輩，同期，後輩の皆様にも感謝を申し上げます．福岡大学中村凌氏には，国際学会提出の際に，実験，執筆の双方に多くの助言と助力を賜りました．厚く御礼申し上げます．

産業技術総合研究所上級主任研究員片岡裕雄氏には，リサーチアシスタントの受け入れ先としてメンターをしていただいただけでなく，多くの議論の機会とディスカッションの場えを提供いただきました．厚く御礼申し上げます．東京工業大学横田理央教授，井上中順准教授には共著者として多くの助言と議論の機会をいただきました．深く感謝申し上げます．また，多くの助言や励ましをいただきました cvpaper.challenge の皆様にも深く御礼申し上げます．最後に，私の生活を支えてくださった家族に最大の感謝を申し上げます．

参考文献

[1] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off. Dec. 2018.

[2] Yunhao Ge, Yao Xiao, Zhi Xu, Xingrui Wang, and Laurent Itti. Contributions of shape, texture, and color in visual recognition. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *ECCV*, pages 369–386, 2022.

[3] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *ICLR*, 2019.

[4] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: where bigger models and more data hurt*. *J. Stat. Mech.*, 2021(12):124003, Dec. 2021.

[5] Dev Rajnarayan and David Wolpert. Bias-Variance trade-offs: Novel applications. In Claude Sammut and Geoffrey I Webb, editors, *Encyclopedia of Machine Learning*, pages 101–110. Springer US, Boston, MA, 2010.

[6] Cory Stephenson and Tyler Lee. When and how epochwise double descent happens. *CoRR*, abs/2108.12006, 2021.

[7] 髙橋 秀弥, 井上 中順, 横田 理央, 片岡 裕雄, and 前田 英作. 画像識別における形状・テクスチャ偏重度と二重降下現象の関係について. *IEICE Conferences Archives*, IEICE-122(IEICE-PRMU-404,IEICE-IBISML-405):IEICE–PRMU–13,IEICE–IBISML–13–IEICE–PRMU–16,IEICE–IBISML–16, Feb. 2023.

付 録A　学習プログラム

学習に利用したプログラムを示す．掲載したプログラムは，複数あるバージョンの一つである．

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as f
import torch.backends.cudnn as cudnn
import torchvision
import torchvision.transforms as transforms;
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import math
from model import resnet18k
import torchvision.models as models
# from transformers import ViTForImageClassification

import argparse
import random
import matplotlib.pyplot as plt
import csv
import warnings
import os
import sys
import wandb
import numpy as np
import cv2
from PIL import Image

def get_model(args):
    """モデルの読み込み

    Arguments:
        args:
    Returns:
        model
    """
    if args.model == "resnet18k":
        assert args.model_width is not None, "please check k value"
        model = resnet18k.make_resnet18k(k=args.model_width, num_classes=args.num_classes)
        args.model_fullname = "SR_resnet18k-{}".format(args.model_width)
    elif args.model == "resnet18":
        if args.pretrained == "SR":
            # model
            model = models.resnet18(pretrained=False, num_classes=args.num_classes)
            args.model_fullname = "SR_resnet18" # SR means scratch
        elif args.pretrained == "IN": ## 事前学習モデルを利用する場合，一度重みを読み込んで，全結合層だけ初期化する
            model = models.resnet18(pretrained=True)
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "IN_resnet18"
        elif args.pretrained == "INv1": ## 独自に学習したバージョン、バッチサイズ 256
            model = models.resnet18()
            model.load_state_dict(
            torch.load('/workspace/Epoch_dd/checkpoint/SR_resnet18_90epochs_bs256_ImageNet_ln0pc.tar')["model_state_dict"]
            )
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "INv1_resnet18"
        elif args.pretrained == "INv2": ## 独自に学習したバージョン、バッチサイズ 256
            model = models.resnet18()
            model.load_state_dict(
            torch.load('/workspace/Epoch_dd/checkpoint/SR_resnet18_90epochs_bs32_ImageNet_ln0pc.tar')["model_state_dict"]
            )
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "INv2_resnet18"
```

```
        elif args.pretrained == "INv3": ## pytorch のプログラムで事前学習
            model = models.resnet18()
            model.load_state_dict(
            torch.load('/workspace/Epoch_dd/checkpoint/INv3_made_by_pytorch_code.pth')["model"]
            )
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "INv3_resnet18"
        elif args.pretrained == "INv4": ## pytorch のプログラムで事前学習
            model = models.resnet18()
            model.load_state_dict(
            torch.load('/workspace/Epoch_dd/checkpoint/SR_resnet18_90epochs_bs32_ImageNet_ln0pc_v2.tar')["model_state_dict"]
            )
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "INv4_resnet18"
        elif args.pretrained == "FDB1kv1": ## 独自に学習したバージョン、バッチサイズ 256
            model = models.resnet18()
            model.load_state_dict(
            torch.load('/workspace/Epoch_dd/checkpoint/SR_resnet18_90epochs_bs32_FDB_1k_ln0pc.tar')["model_state_dict"]
            )
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "FDB1kv1_resnet18"
        elif args.pretrained == "RCDB1kv1": ## 独自に学習したバージョン、バッチサイズ 256
            model = models.resnet18()
            model.load_state_dict(
            torch.load('/workspace/Epoch_dd/checkpoint/SR_resnet18_90epochs_bs32_RCDB1k_ln0pc.tar')["model_state_dict"]
            )
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "RCDB1kv1_resnet18"
        elif args.pretrained == "FR":
            model = models.resnet18()
            model.load_state_dict(torch.load('/workspace/Epoch_dd/model_weight/Fractal-1k/FractalDB-1000_res18.pth'))
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "FR_resnet18"
    elif args.model == "resnet34":
        if args.pretrained == "SR":
            # model
            model = models.resnet34(pretrained=False, num_classes=args.num_classes)
            args.model_fullname = "SR_resnet34" # SR means scratch
        else: ## 事前学習モデルを利用する場合，一度重みを読み込んで，全結合層だけ初期化する
            model = models.resnet34(pretrained=True)
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "IN_resnet34"
    elif args.model == "resnet50":
        if args.pretrained == "SR":
            # model
            model = models.resnet50(pretrained=False, num_classes=args.num_classes)
            args.model_fullname = "SR_resnet50" # SR means scratch
        else: ## 事前学習モデルを利用する場合，一度重みを読み込んで，全結合層だけ初期化する
            model = models.resnet50(pretrained=True)
            in_features = model.fc.in_features
            model.fc = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "IN_resnet50"
    elif args.model == "densenet121":
        if args.pretrained == "SR":
            # model
            model = models.densenet121(pretrained=False, num_classes=args.num_classes)
            args.model_fullname = "SR_densenet121" # SR means scratch
        else: ## 事前学習モデルを利用する場合，一度重みを読み込んで，全結合層だけ初期化する
            model = models.densenet121(pretrained=True)
            in_features = model.classifier.in_features
            model.classifier = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "IN_densenet121"
    elif args.model == "mobilenetV2":
        if args.pretrained == "SR":
            # model
            model = models.mobilenet_v2(pretrained=False, num_classes=args.num_classes)
            args.model_fullname = "SR_mobilenetV2" # SR means scratch
        else: ## 事前学習モデルを利用する場合，一度重みを読み込んで，全結合層だけ初期化する
            model = models.mobilenet_v2(pretrained=True)
            in_features = model.classifier[1].in_features
            model.classifier[1] = nn.Linear(in_features=in_features, out_features=args.num_classes, bias=True)
            args.model_fullname = "IN_mobilenetV2"
    elif args.model == "vitB16":
        if args.pretrained == "SR":
```

```
            # model
            model = models.vit_b_16(pretrained=False, num_classes=args.num_classes)
            args.model_fullname = "SR_vitB16" # SR means scratch
        else:
            model = models.vit_b_16(pretrained=True)
            in_features = model.heads[0].in_features
            model.heads[0] = nn.Linear(in_features=in_features, out_features=args.num_classes)
            args.model_fullname = "IN_vitB16"
    return model

def get_dataset_train(args, transform):
    if args.dataset == "cifar10":
        return torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
    elif args.dataset == "cannyShapeCifar10ColorV1":
        train_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
        # RGB で形状を取り出して代入
        for i in range(len(train_set)):
            new_image = np.array(train_set.data[i], dtype=np.uint8)
            new_image = cv2.cvtColor(new_image, cv2.COLOR_RGB2BGR)
            edges = cv2.Canny(image=new_image,threshold1=255 / 2, threshold2=255)
            edges = cv2.bitwise_and(new_image, new_image, mask=edges)
            train_set.data[i] = Image.fromarray(edges)
        return train_set
    elif args.dataset == "cifar100":
        return torchvision.datasets.CIFAR100(root='./data', train=True, download=True, transform=transform)
    elif args.dataset == "tinyImageNet":
        return torchvision.datasets.ImageFolder(root='./data/tiny-imagenet-200/train', transform=transform)
    elif args.dataset == "DTD":
        return torchvision.datasets.DTD(root='./data', split="train", download=True, transform=transform)
    elif args.dataset == "STL10":
        return torchvision.datasets.STL10(root='./data', split="train", download=True, transform=transform)
    elif args.dataset == "flower102":
        return torchvision.datasets.Flowers102(root='./data', split="train", download=True, transform=transform)
    elif args.dataset == "cifar100_to_10class":
        selected_classes = list(range(10))
        train_set = torchvision.datasets.CIFAR100(root='./data', train=True, download=True, transform=transform)
        targets = torch.tensor(train_set.targets)

        # 選択したクラスに対応するブーリアンマスクを作成
        mask = torch.zeros_like(targets, dtype=torch.bool)
        for c in selected_classes:
            mask |= (targets == c)

        # 選択したクラスに対応するデータとターゲットを抽出
        train_set.data = train_set.data[mask]
        train_set.targets = targets[mask].tolist()
        return train_set

def get_dataset_test(args, transform):
    if args.dataset == "cifar10":
        return torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
    elif args.dataset == "cannyShapeCifar10ColorV1":
        test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
        # # RGB で形状を取り出して代入
        # for i in range(len(test_set)):
        #     new_image = np.array(test_set.data[i], dtype=np.uint8)
        #     new_image = cv2.cvtColor(new_image, cv2.COLOR_RGB2BGR)
        #     edges = cv2.Canny(image=new_image,threshold1=255 / 2, threshold2=255)
        #     edges = cv2.bitwise_and(new_image, new_image, mask=edges)
        #     test_set.data[i] = Image.fromarray(edges)
        return test_set
    elif args.dataset == "cifar100":
        return torchvision.datasets.CIFAR100(root='./data', train=False, download=True, transform=transform)
    elif args.dataset == "tinyImageNet":
        return torchvision.datasets.ImageFolder(root='./data/tiny-imagenet-200/val', transform=transform)
    elif args.dataset == "DTD":
        return torchvision.datasets.DTD(root='./data', split="test", download=True, transform=transform)
    elif args.dataset == "STL10":
        return torchvision.datasets.STL10(root='./data', split="test", download=True, transform=transform)
    elif args.dataset == "flower102":
        return torchvision.datasets.Flowers102(root='./data', split="test", download=True, transform=transform)
    elif args.dataset == "cifar100_to_10class":
        selected_classes = list(range(10))
        test_set = torchvision.datasets.CIFAR100(root='./data', train=False, download=True, transform=transform)
        targets = torch.tensor(test_set.targets)

        # 選択したクラスに対応するブーリアンマスクを作成
        mask = torch.zeros_like(targets, dtype=torch.bool)
        for c in selected_classes:
            mask |= (targets == c)
```

```
225
226         # 選択したクラスに対応するデータとターゲットを抽出
227         test_set.data = test_set.data[mask]
228         test_set.targets = targets[mask].tolist()
229         return test_set
230
231 def get_imagesize(args):
232     if args.dataset == "cifar10":
233         return 32
234     elif args.dataset == "cannyShapeCifar10ColorV1":
235         return 32
236     elif args.dataset == "cifar100":
237         return 32
238     elif args.dataset == "tinyImageNet":
239         return 64
240     elif args.dataset == "DTD":
241         return 224
242     elif args.dataset == "STL10":
243         return 96
244     elif args.dataset == "flower102":
245         return 224
246     elif args.dataset == "cifar100_to_10class":
247         return 32
248
249 def get_num_classes(args):
250     if args.dataset == "cifar10":
251         return 10
252     elif args.dataset == "cannyShapeCifar10ColorV1":
253         return 10
254     elif args.dataset == "cifar100":
255         return 100
256     elif args.dataset == "tinyImageNet":
257         return 200
258     elif args.dataset == "DTD":
259         return 47
260     elif args.dataset == "STL10":
261         return 10
262     elif args.dataset == "flower102":
263         return 102
264     elif args.dataset == "cifar100_to_10class":
265         return 10
266
267
268 def fix_seed(seed=42):
269     # random
270     random.seed(seed)
271     # Numpy
272     # np.random.seed(seed)
273     # Pytorch
274     torch.manual_seed(seed)
275     torch.cuda.manual_seed_all(seed)
276     torch.backends.cudnn.benchmark = False
277     torch.backends.cudnn.deterministic = True
278     # Tensorflow
279     # tf.random.set_seed(seed)
280
281 def main():
282     args = parse_args()
283     fix_seed(args.fix_seed)
284     #epoch 数指定
285     epoch = args.epoch
286     label_noise_rate = args.label_noise_rate
287     args.num_classes = get_num_classes(args)
288     device = 'cuda' if torch.cuda.is_available() else 'cpu'
289     imageSize = get_imagesize(args)
290
291     if args.model == "vitB16" or args.dataset == "DTD":
292         args.resize = True
293
294     if args.resize:
295         transform_train = transforms.Compose([
296             transforms.RandomCrop(imageSize, padding=imageSize // 8),
297             transforms.RandomHorizontalFlip(),
298             transforms.ToTensor(),
299             transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
300             transforms.Resize(size=(224, 224)),
301         ])
302         transform_test = transforms.Compose([
303             transforms.ToTensor(),
304             transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
```

```
            transforms.Resize(size=(224, 224)),
        ])
    else:
        transform_train = transforms.Compose([
            transforms.RandomCrop(imageSize, padding=imageSize // 8),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
        ])
        transform_test = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
        ])
    #resize 分岐
    train_set = get_dataset_train(args, transform_train)

    #インスタンス変数にアクセスしてラベルの張替え
    if hasattr(train_set, "targets"):
        for i in range(len(train_set.targets)):
            if(random.randint(0, 9999) < int(label_noise_rate * 10000)):
                train_set.targets[i] += random.randint(1, args.num_classes - 1)
                train_set.targets[i] %= args.num_classes
    elif hasattr(train_set, "_labels"):
        #_labels でラベル情報を持っている場合
        for i in range(len(train_set._labels)):
            if(random.randint(0, 9999) < int(label_noise_rate * 10000)):
                train_set._labels[i] += random.randint(1, args.num_classes - 1)
                train_set._labels[i] %= args.num_classes
    else:
        #labels でラベル情報を持っている場合
        for i in range(len(train_set.labels)):
            if(random.randint(0, 9999) < int(label_noise_rate * 10000)):
                train_set.labels[i] += random.randint(1, args.num_classes - 1)
                train_set.labels[i] %= args.num_classes

    train_loader = torch.utils.data.DataLoader(train_set, batch_size=128, shuffle=True, num_workers=2)
    test_set = get_dataset_test(args, transform_test)
    test_loader = torch.utils.data.DataLoader(test_set, batch_size=128, shuffle=True, num_workers=2)
    # CIFAR10 のクラス
    class_names = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    model = get_model(args)
    model = model.to(device)
    print(args.model_fullname)

    if args.resize:
        args.dataset = "resize_" + args.dataset
        #resize の有無を path 名で明示

    # if device == 'cuda':
    #     model = torch.nn.DataParallel(model)

    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = optim.Adam(model.parameters(), lr=0.0001)
    x1 = []
    x2 = range(epoch + 1)
    x1.append(0)

    #wandb

    wandb.init(
    # set the wandb project where this run will be logged
        project="DDvsShapeTexture",

        # track hyperparameters and run metadata
        config={
            "architecture": args.model_fullname,
            "epochs": args.epoch,
            "dataset": args.dataset,
            "label_noise_rate": args.label_noise_rate,
            "batch_size": args.batch_size,
            "learning_rate": args.learning_rate,
            "seed": args.fix_seed,
        }
    )


    #----------#
    ### csv 作成
    #----------#
```

```
    # 初期値取得

    train_acc, train_loss = test(model, device, test_loader, criterion) # 初期値取得
    test_acc, test_loss = test(model, device, test_loader, criterion)
    wandb.log({"Epoch": 0,"train acc": train_acc, "train loss": train_loss, "test acc": test_acc, "test loss": test_loss})

    # train
    with open('./csv/{}_{}_{}epochs_ln{}pc_seed{}_train.csv'.format(args.model_fullname, args.dataset, args.epoch,
    ↪  int(label_noise_rate * 100), args.fix_seed),'w') as file:
        file.write("epoch,error,loss\n")
        file.write(f"0,{1.0 - train_acc},{train_loss}" + "\n")
    # test
    with open('./csv/{}_{}_{}epochs_ln{}pc_seed{}_test.csv'.format(args.model_fullname, args.dataset, args.epoch,
    ↪  int(label_noise_rate * 100), args.fix_seed),'w') as file:
        file.write("epoch,error,loss\n")
        file.write(f"0,{1.0 - test_acc},{test_loss}" + "\n")

    if not os.path.isdir(f'./model_weight1/{args.model_fullname}_{args.dataset}_ln{int(label_noise_rate *
    ↪  100)}pc_seed{args.fix_seed}'):
        os.makedirs(f'./model_weight1/{args.model_fullname}_{args.dataset}_ln{int(label_noise_rate * 100)}pc_seed{args.fix_seed}')
    ## ----------------#
    ## training #
    # ----------------#
    for epoch in range(epoch):
        # Train and test a model.
        model.train()
        #train の各数値は batch ごとに出す
        #calc_score の返却値の loss は batch 数で割っているので無視,loss.item() を用いる
        for batich_idx, (inputs, targets) in enumerate(train_loader):
            output_list = []
            target_list = []
            running_loss = 0.0
            xpoint = 0.0 + epoch + (float(batich_idx + 1) / len(train_loader))
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            output_list += [int(o.argmax()) for o in outputs]
            target_list += [int(t) for t in targets]
            running_loss += loss.item()

            train_acc, train_loss = calc_score(target_list, output_list, running_loss, train_loader)
            x1.append(xpoint)
            ## csv writer
            with open('./csv/{}_{}_{}epochs_ln{}pc_seed{}_train.csv'.format(args.model_fullname, args.dataset, args.epoch,
            ↪  int(label_noise_rate * 100), args.fix_seed),'a') as file:
                file.write(f"{xpoint},{1.0 - train_acc},{loss.item()}" + "\n")

            # if batich_idx % 100 == 0 and batich_idx != 0:
            #     stdout_temp = 'batch: {:>3}/{:<3}, train acc:{:<8}, train loss: {:<8}'
            #     print(stdout_temp.format(batich_idx, len(train_loader), train_acc, loss.item()))

        ## これないとダメ model.eval
        model.eval()
        test_acc, test_loss = test(model, device, test_loader, criterion)
        # csv writer
        with open('./csv/{}_{}_{}epochs_ln{}pc_seed{}_test.csv'.format(args.model_fullname, args.dataset, args.epoch,
        ↪  int(label_noise_rate * 100), args.fix_seed),'a') as file:
                file.write(f"{epoch + 1},{1.0 - test_acc},{test_loss}" + "\n")

        # Output score.
        stdout_temp = 'epoch: {:>3}, train acc: {:<8}, train loss: {:<8}, test acc: {:<8}, test loss: {:<8}'
        print(stdout_temp.format(epoch+1, train_acc, loss.item(), test_acc, test_loss))
        wandb.log({"Epoch": epoch + 1,"train acc": train_acc, "train loss": loss.item(), "test acc": test_acc, "test loss":
        ↪  test_loss, "leraning rate": optimizer.param_groups[0]['lr']})

        ## model save###
        # torch.save(model.state_dict(), './model_weight/resnet18*' + str(args.model_width) + '-cifar10-train.csv')
        # pytorch の慣例で pth ファイルで保存する
        # 実験のために epoch ごとに保存
        model_path = f'./model_weight1/{args.model_fullname}_{args.dataset}_ln{int(label_noise_rate *
        ↪  100)}pc_seed{args.fix_seed}/{args.model_fullname}_epoch{(epoch + 1):04}_{args.dataset}_ln{int(label_noise_rate *
        ↪  100)}pc_seed{args.fix_seed}.pth'
        torch.save(model.state_dict(), model_path)

    # checkpoint
```

```
    torch.save({
    "epoch": epoch,
    "model_state_dict": model.state_dict(),
    "optimizer_state_dict": optimizer.state_dict()
    },
    f'./checkpoint/{args.model_fullname}_{(epoch + 1)}epochs_{args.dataset}_ln{int(label_noise_rate * 100)}pc_seed{args.fix_seed}.tar')



def train (model, device, train_loader, criterion, optimizer):
    model.train()
    output_list = []
    target_list = []
    running_loss = 0.0
    for batich_idx, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        output_list += [int(o.argmax()) for o in outputs]
        target_list += [int(t) for t in targets]
        running_loss += loss.item()

        train_acc, train_loss = calc_score(target_list, output_list, running_loss, train_loader)
        if batich_idx % 100 == 0 and batich_idx != 0:
            stdout_temp = 'batch: {:>3}/{:<3}, train acc:{:<8}, train loss: {:<8}'
            print(stdout_temp.format(batich_idx, len(train_loader), train_acc, train_loss))
    train_acc, train_loss = calc_score(target_list, output_list, running_loss, train_loader)


    return train_acc, train_loss

def test(model, device, test_loader, criterion):
    model.eval()

    output_list = []
    target_list = []
    running_loss = 0.0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(test_loader):
            # Forward processing.
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            # Set data to calculate score.
            output_list += [int(o.argmax()) for o in outputs]
            target_list += [int(t) for t in targets]
            running_loss += loss.item()

    test_acc, test_loss = calc_score(target_list, output_list, running_loss, test_loader)

    return test_acc, test_loss
def calc_score(true_list, predict_list, running_loss, data_loader):
    # import pdb;pdb.set_trace()
    # result = classification_report(true_list, predict_list, output_dict=True)
    # acc = round(result['accuracy'], 6)
    acc = accuracy_score(true_list, predict_list)
    loss = round(running_loss / len(data_loader), 6)

    return acc, loss

def parse_args():
    arg_parser = argparse.ArgumentParser(description="ResNet trained by CIFAR-10")

    arg_parser.add_argument("-k", "--model_width", type=int, default=64)
    arg_parser.add_argument("-e", "--epoch", type=int, default=4000)
    arg_parser.add_argument("-l", "--label_noise_rate", type=float, default=0.0)

    # 追加
    arg_parser.add_argument("--model", type=str, choices=["resnet18k", "resnet18", "resnet34", "resnet50", "densenet121", "mobilenetV2", "vitB16"], help="モデルアーキテクチャの選択")
    arg_parser.add_argument("-pt", "--pretrained", type=str, choices=["SR", "IN", "INv1", "INv2", "INv3", "INv4", "FR", "FDB1kv1", "RCDB1kv1"], default="IN", help="事前学習モデルの利用. 指定しないと事前学習なし")
```

```
    arg_parser.add_argument("-ds", "--dataset", type=str, choices=["cifar10", "cifar100", "tinyImageNet", "DTD", "STL10",
    ↪  "cannyShapeCifar10ColorV1", "cifar100_to_10class"], default="cifar10")
    arg_parser.add_argument("--num_classes", type=int, default=10, help="分類クラス数")
    arg_parser.add_argument("--resize", action='store_true', help="224*224 に resize,ViT はデフォルトで true")
    arg_parser.add_argument("--imagesize", type=int, help="resize=true の場合指定サイズに変更")
    arg_parser.add_argument("-bs", "--batch_size", type=int, default=128)
    arg_parser.add_argument("-lr", "--learning_rate", type=float, default=0.0001)
    arg_parser.add_argument("-seed", "--fix_seed", type=int, default=42)

    return arg_parser.parse_args()

if __name__ =='__main__':
    warnings.filterwarnings('ignore')
    import time
    start = time.perf_counter()
    main()

    print(time.perf_counter() - start)

```

付 録B　対外成果発表リスト

2022-03-15 髙橋 秀弥, 鏡川 悠介，前田 英作, “深層ニューラルネットワークにおける二重降下現象,” 電子情報通信学会 2022 年総合大会 情報・システムソサイエティ特別企画 ジュニア＆学生ポスターセッション予稿集, ISS-SP-028, ポスター発表, 開催地 zoom.

2022-04-16 杉田 拓磨, 岡澤 律来, 金田 龍平, 髙橋 秀弥, 鏡川 悠介, 出場チーム名 AID, AI イノベーションアワード 2022, 開催地 立教大学,
最優秀賞 (https://www.nttpc.co.jp/press/2022/04/202204211500.html).

2022-07-26 髙橋 秀弥, 鏡川 悠介，前田 英作, “深層学習における二重降下現象と画像のテクスチャ・形状性について,” 第 25 回 画像の認識・理解シンポジウム (MIRU2022), OS1A-5, 口頭発表（査読あり），ポスター発表, 開催地 姫路市文化コンベンションセンター アクリエひめじ (兵庫県).

2022-07-28 杉田 拓磨, 岡澤 律来, 金田 龍平, 髙橋 秀弥, 鏡川 悠介, 前田 英作, “物語文を入力とする自動挿絵生成システム,” 第 25 回 画像の認識・理解シンポジウム (MIRU2022), IS3-75, ポスター発表, 開催地 姫路市文化コンベンションセンター アクリエひめじ (兵庫県).

2023-03-02 髙橋 秀弥，井上中順，横田理央，片岡裕雄，前田英作, “画像識別における形状・テクスチャ偏重度と二重降下現象の関係について,” パターン認識・メディア理解（PRMU）2023 年 3 月研究会, PRMU-3, 口頭発表（査読なし）, 公立はこだて未来大 (北海道).

2023-03-02 遠藤隆斗，髙橋 秀弥，前田英作, “医療画像タスクにおける数式駆動型教師あり学習の有効性について,” パターン認識・メディア理解（PRMU）2023 年 3 月研究会, PRMU-22, 口頭発表（査読なし）, 公立はこだて未来大 (北海道).

2023-07-28 髙橋 秀弥，井上中順，横田理央，片岡裕雄，前田英作, “学習過程における形状・テクスチャ偏重度の推移と事前学習データセットとの関係について,” 第 26 回 画像の認識・理解シンポジウム (MIRU2023), IS3-26，ポスター発表, 開催地 アクトシティ浜松 (静岡県).