社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# [サーベイ論文] 畳み込みニューラルネットワークの研究動向

内田　祐介†　　山下　隆義††

† 株式会社ディー・エヌ・エー
†† 中部大学

**あらまし**　2012 年の画像認識コンペティション ILSVRC における AlexNet の登場以降，画像認識においては畳み込みニューラルネットワーク (CNN) を用いることがデファクトスタンダードとなった．ILSVRC では毎年のように新たな CNN のモデルが提案され，一貫して認識精度の向上に寄与してきた．CNN は画像分類だけではなく，セグメンテーションや物体検出など様々なタスクを解くためのベースネットワークとしても広く利用されてきている．本稿では，AlexNet 以降の代表的な CNN の変遷を振り返るとともに，近年提案されている様々な CNN の改良手法についてサーベイを行い，それらを幾つかのアプローチに分類し，解説する．更に，代表的なモデルについて複数のデータセットを用いて学習および網羅的な精度評価を行い，各モデルの精度および学習時間の傾向について議論を行う．

**キーワード**　畳み込みニューラルネットワーク，deep learning，画像認識，サーベイ

## 1. はじめに

畳み込みニューラルネットワーク (Convolutional Neural Networks; CNN) （以降 CNN）は，主に画像認識に利用されるニューラルネットワークの一種である．CNN の原型は，生物の脳の視覚野に関する神経生理学的な知見 [24] を元に考案された Neocognitron [10] に見ることができる．Neocognitron は，特徴抽出を行なう単純型細胞に対応する畳み込み層と，位置ずれを許容する働きを持つ複雑型細胞に対応する pooling 層とを交互に階層的に配置したニューラルネットワークである．Neocognitron では自己組織化による学習が行われていたが，その後，LeCun らにより CNN の backpropagation を用いた学習法が確立され，例えば LeNet [31] は，文字認識において成功を収めた．

2000 年代においては，画像認識分野では，SIFT [36] 等の職人芸的に設計された特徴ベクトルと，SVM 等の識別器を組合せた手法が主流となっていた．その時代においてもニューラルネットワークの研究は進められており，ついに 2012 年の画像認識コンペティション ImageNet Large Scale Visual Recognition Competition (ILSVRC) において，AlexNet と呼ばれる CNN を用いた手法が，それまでの画像認識のデファクトスタンダードであった SIFT + Fisher Vector + SVM [43] というアプローチに大差をつけて優勝し，一躍深層学習が注目されることとなった．それ以降の ILSVRC では，CNN を用いた手法が主流となり，毎年新たな CNN のモデルが適用され，一貫して認識精度の向上に寄与してきた．そして ILSVRC で優秀な成績を収めたモデルが，画像認識やその他の様々なタスクを解くためのデファクトスタンダードなモデルとして利用されてきた．

CNN は画像認識だけではなく，セグメンテーション [3,5,35]，物体検出 [34, 41, 42]，姿勢推定 [4, 39, 58] など様々なタスクを解くためのベースネットワークとしても広く利用されてきている．また，画像ドメインだけではなく，自然言語処理 [2,28,64]，音響信号処理 [54,55]，ゲーム AI [45] 等の分野でも利用されるなど，ニューラルネットワークの中でも重要な位置を占めている．このような背景のもと，本稿では，AlexNet 以降の代表的な CNN のモデルの変遷を振り返るとともに，近年提案されている様々な改良手法についてサーベイを行い，それらを幾つかのアプローチに分類し，解説する．更に，代表的なモデルについて複数のデータセットを用いて学習および網羅的な精度評価を行い，各モデルの精度および学習時間の傾向について議論を行う．なお，本稿では，CNN のモデルをどのような構造にするかというモデルアーキテクチャに焦点をあてており，個々の構成要素や最適化手法については詳述しない．そのため，より広範な内容については文献 [14] を参照されたい．

本稿の構成は以下の通りである．まず 2 章において，ILSVRC にて優秀な成績を収めたモデルを解説しつつ CNN の進化を概観する．3 章では近年提案されている CNN の改良手法をそのアプローチから分類し，それぞれ解説する．4 章では，画像認識に関するベンチマークデータセットについて概説し，代表的なモデルについて複数のデータセットを用いて学習および精度評価を行い，精度および学習時間の傾向について議論を行い，5 章でまとめを述べる．

## 2. ILSVRC で振り返る CNN の進化

本章では，2012 年から 2017 年までの ILSVRC のクラス分類タスク（以降では単に ILSVRC）において優秀な成績を収めたモデルを順に振り返り，CNN がどのような進化を辿ってきたかを概観する．

### 2.1 AlexNet

AlexNet [29] は，2012 年の ILSVRC において，従来の画像

表 1　AlexNet の構造

| Layer | Filter size | Stride | Output size |
|---|---|---|---|
| input | | | (227, 227, 3) |
| conv | (11, 11) | (4, 4) | (55, 55, 96) |
| maxpool | (3, 3) | (2, 2) | (27, 27, 96) |
| conv | (5, 5) | (1, 1) | (13, 13, 256) |
| maxpool | (3, 3) | (2, 2) | (13, 13, 256) |
| conv | (3, 3) | (1, 1) | (13, 13, 384) |
| conv | (3, 3) | (1, 1) | (13, 13, 384) |
| conv | (3, 3) | (1, 1) | (13, 13, 256) |
| maxpool | (3, 3) | (2, 2) | (6, 6, 256) |
| fc | | | 4096 |
| fc | | | 4096 |
| fc | | | 1000 |

表 2　ZFNet の構造

| Layer | Filter size | Stride | Output size |
|---|---|---|---|
| input | | | (224, 224, 3) |
| conv | (7, 7) | (2, 2) | (110, 110, 96) |
| maxpool | (3, 3) | (2, 2) | (55, 55, 96) |
| conv | (5, 5) | (2, 2) | (26, 26, 256) |
| maxpool | (3, 3) | (2, 2) | (13, 13, 256) |
| conv | (3, 3) | (1, 1) | (13, 13, 384) |
| conv | (3, 3) | (1, 1) | (13, 13, 384) |
| conv | (3, 3) | (1, 1) | (13, 13, 256) |
| maxpool | (3, 3) | (2, 2) | (6, 6, 256) |
| fc | | | 4096 |
| fc | | | 4096 |
| fc | | | 1000 |

認識のデファクトスタンダードであった SIFT + Fisher Vector + SVM [43] というアプローチに大差をつけて優勝し，一躍深層学習の有効性を知らしめたモデルである．現在では比較的小規模なモデルということもあり，ベースラインとして利用されることもある．

表 1 に AlexNet の構造を示す．Filter size は各フィルタ処理のカーネルサイズを示し，Stride はそれらの適用間隔を示す．AlexNet は，畳み込み層を 5 層重ねつつ，pooling 層で特徴マップを縮小し，その後，3 層の全結合層により最終的な出力を得る．基本的なアーキテクチャの設計思想は Neocognitron や LeNet を踏襲している．学習時には，当時の GPU のメモリ制約から，各層の特徴マップをチャネル方向に分割し，2 台の GPU で独立して学習するというアプローチが取られた．幾つかの畳み込み層および全結合層では，より有効な特徴を学習するため，もう 1 台の GPU が担当しているチャネルも入力として利用している．

CNN の重みはガウス分布に従う乱数により初期化され，モーメンタム付きの確率的勾配降下法 (Stochastic Gradient Descent, SGD) により最適化が行われる．各パラメータは weigtht decay（$\ell_2$ 正則化）により正則化が行われている．ロスが低下しなくなったタイミングで学習率を 1/10 に減少させることも行われており，上記の最適化の手法は，現在おいてもベストプラクティスとして利用されている．以下では，AlexNet に導入された重要な要素技術について概説する．

**ReLU.** 従来，非線形な活性化関数としては，$f(x) = \tanh(x)$ や $f(x) = (1 + e^{-x})^{-1}$ が利用されていたが，$f(x) = \max(0, x)$ と定義される Rectified Linear Units (ReLU) [27,37] を利用することで学習を高速化している．これは，深いネットワークで従来の活性化関数を利用した場合に発生する勾配消失問題を解決できるためである．ReLU は，その後改良がなされた活性化関数も提案されている [7, 16, 40] が，最新のモデルでも標準的な活性化関数として広く利用されている．

**LRN.** Local Response Normalization (LRN) は，特徴マップの同一の位置にあり，隣接するチャネルの出力の値から，自身の出力の値を正規化する手法である．空間的に隣接する出力も考慮して正規化を行う Local Contrast Normalization (LCN) [27] と比較して，平均値を引く処理を行わず，より適切な正規化が行えるとしている．後述する VGGNet では効果が認められなかったことや，batch normalization [26] の登場により，近年のモデルでは利用されなくなっている．

**Overlapping Pooling.** Pooling 層は，$s$ ピクセルずつ離れたグリッドにおいて，周辺 $z$ ピクセルの値を max や average 関数によって集約する処理と一般化することができる．通常，pooling 層は $s = z$ とされ，集約されるピクセルが複数のグリッドにまたがって overlap しないことが一般的であった．AlexNet では，$s = 2$, $z = 3$ の max pooling を利用しており，この場合，集約されるピクセル領域がオーバラップすることになる．この overlapping pooling により，過学習を低減し，わずかに最終的な精度が向上すると主張されている．

**Dropout.** Dropout [48] は，学習時のネットワークについて，隠れ層のニューロンを一定確率で無効化する手法である．これにより，擬似的に毎回異なるアーキテクチャで学習を行うこととなり，アンサンブル学習と同様の効果をもたらし，より汎化されたモデルを学習することができる．AlexNet では，最初の 2 つの全結合層にこの dropout が導入されている．Dropout を行わない場合にはかなりの過学習が発生したが，dropout によりこの過学習を抑えられる一方，収束までのステップ数が約 2 倍になったと報告されている．

### 2.2　ZFNet

ZFNet [61] は，2013 年の ILSVRC の優勝モデルである．表 2 に ZFNet のモデルを示す．文献 [61] では，CNN がどのように画像を認識しているかを理解し，またどうすれば CNN の改良ができるかを検討することを目的とし，CNN の可視化を行っている．その可視化の結果，AlexNet の 2 つの問題が明らかとなり，これらの問題を解決する改良を行い，高精度化につなげている．文献 [61] で明らかとなった AlexNet の第 1 の問題は，最初の畳み込み層のフィルタが，大きなカーネルサイズを利用していることから極端に高周波と低周波の情報を取得するフィルタとなっており，それらの間の周波数成分を取得するフィルタが殆ど無かったという点である．第 2 の問題は，2 層目の特徴マップにおいて，エイリアシングが発生していることである．これは，最初の畳み込み層において，stride に 4 という大きな値を使っているためである．これらの問題を解決するため，1) 最初の畳み込み層のフィルタサイズを 11 から 7 に縮小し，2)

表 3　GoogLeNet の構造

| Layer | Filter size | Stride | Output size |
|---|---|---|---|
| input | | | (224, 224, 3) |
| conv | (7, 7) | (2, 2) | (112, 112, 64) |
| maxpool | (3, 3) | (2, 2) | (56, 56, 64) |
| conv | (3, 3) | (1, 1) | (56, 56, 192) |
| maxpool | (3, 3) | (2, 2) | (28, 28, 192) |
| inception×2 | | | (28, 28, 480) |
| maxpool | (3, 3) | (2, 2) | (14, 14, 480) |
| inception×5 | | | (14, 14, 832) |
| maxpool | (3, 3) | (2, 2) | (7, 7, 832) |
| inception×2 | | | (7, 7, 1024) |
| averagepool | (7, 7) | (1, 1) | (1, 1, 1024) |
| fc | | | 1000 |

stride を 4 から 2 に縮小するという改良を行い，AlexNet を超える認識精度を達成した．

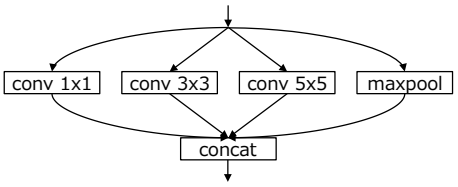
### 2.3　GoogLeNet

GoogLeNet [52] は，2014 年の ILSVRC の優勝モデルである．このモデルの特徴は，下記で詳述する Inception モジュールの利用である．

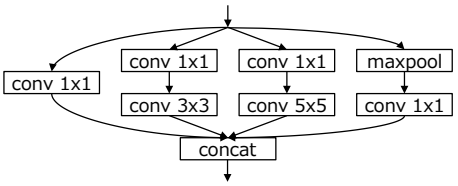
**Inception モジュール．** GoogLeNet の一番の特徴は，複数の畳み込み層や pooling 層から構成される Inception モジュールと呼ばれる小さなネットワーク (micro networks) を定義し，これを通常の畳み込み層のように重ねていくことで 1 つの大きな CNN を作り上げている点である．本稿では，このような小さなネットワークをモジュールと呼ぶこととする．この設計は，畳み込み層と多層パーセプトロン（実装上は $1 \times 1$ 畳み込み）により構成されるモジュールを初めて利用した Network In Network (NIN) [32] に大きな影響を受けている．

図 1 に Inception モジュールの構造を，表 3(注1)に Inception モジュールを利用した GoogLeNet の構造を示す．Inception モジュールでは，ネットワークを分岐させ，サイズの異なる畳み込みを行った後，それらの出力をつなぎ合わせるという処理を行っている．この目的は，畳み込み層の重みを sparse にし，パラメータ数のトレードオフを改善することである．

そもそも畳み込み層は，sparse かつ共有された重みを持つ全結合層とみなすことができるが，この畳み込み層自体も，入力チャネルとフィルタの重みとしては dense な結合をしていると言える．これに対し，Inception モジュールは，この入力チャネルとフィルタの重みが sparse になったものとみなすことができる．すなわち，図 1 (a) の *naive* な Inception モジュールは，maxpool を除けば，本来は重みが sparse な $5 \times 5$ の畳み込み 1 つで表現することができる．これを明示的に Inception モジュールを利用することで，遥かに少ないパラメータで同等の表現能力を持つ CNN を構築することができる．また，実際に利用されている図 1 (b) の Inception モジュールでは，各畳み込み層の前に $1 \times 1$ の畳み込み層を挿入し，次元削減を行うことで更にパラメータを削減している．




(a) Inception モジュール (*naive* version)




(b) Inception モジュール

図 1　Inception モジュールの構造

**Global Average Pooling.** GoogLeNet では Global Average Pooling (GAP) [32] が導入されている点にも注目したい．従来のモデルは，畳み込み層の後に複数の全結合層を重ねることで，最終的な 1000 クラス分類の出力を得る構造となっていたが，これらの全結合層はパラメータ数が多く，また過学習を起こすことが課題となっており，dropout を導入することで過学習を抑える必要があった．文献 [32] で提案されている GAP は，入力された特徴マップのサイズと同じサイズの average pooling を行う pooling 層である（すなわち出力は $1 \times 1 \times$ チャネル数のテンソルとなる）．文献 [32] では，CNN の最後の畳み込み層の出力チャネル数を最終的な出力の次元数（クラス数）と同一とし，その後 GAP（および softmax）を適用することで，全結合層を利用することなく最終的な出力を得ることを提案している．全結合層を利用しないことで，パラメータ数を大きく削減し，過学習を防ぐことができる．GoogLeNet では，最後の畳み込み層の出力チャネル数をクラス数と同一にすることはせず，GAP の後に全結合層を 1 層だけ適用し，最終出力を得る構成としている．この GAP の利用は，現在ではクラス分類を行う CNN のベストプラクティスとなっている．

GoogLeNet の学習では，ネットワークの途中から分岐させたサブネットワークにおいてもクラス分類を行い，auxiliary loss を追加することが行われている．これにより，ネットワークの中間層に直接誤差を伝搬させることで，勾配消失を防ぐとともにネットワークの正則化を実現している(注2)．

文献 [53] では，$5 \times 5$ の畳み込みを $3 \times 3$ の畳み込みを 2 つ重ねたものに置き換えることで更に Inception モジュールのパラメータを削減したり，後述する batch normalization [26] を導入したりする等の改良が行われた Inception-v3 が提案されている．更に，文献 [51] では，$n \times 1$ や $1 \times n$ の畳み込みを多数導入し，精度と計算量のトレードオフを改善した Inception-v4 や，後述する ResNet の構造を取り入れた Inception-ResNet が提案されている．

(注1)：複数の Inception モジュールが並んでいる箇所では，段階的に出力チャネル数が増加しているが，本表では最後の Inception モジュールの出力チャネル数を記載している．

(注2)：ただし，文献 [32] では，この改善効果は 0.5%程度で *minor* であると言及されている．

### 2.4 VGGNet

VGGNet [46] は，2014 年の ILSVRC において，2 位の認識精度を達成したモデルである．そのシンプルなモデルアーキテクチャや学習済みモデルが配布されたことから，現在においてもベースラインのモデルとして，またクラス分類以外のタスクのベースネットワークや特徴抽出器としても利用されている．文献 [46] の主な関心は，CNN の深さがどのように性能に影響するかを明らかにすることである．この目的のために，下記のようなモデルアーキテクチャの設計方針を明確にし，深さのみの影響が検証できるようにしている．

- $3 \times 3$（一部 $1 \times 1$）の畳み込みのみを利用する
- 同一出力チャネル数の畳み込み層を幾つか重ねた後に max pooling により特徴マップを半分に縮小する
- max pooling の後の畳み込み層の出力チャネル数を 2 倍に増加させる

この設計方針は，その後のモデルアーキテクチャにおいて広く取り入れられていくこととなる．文献 [46] は，上記の統一的な設計方針の元，ネットワークの深さを増加させていくとコンスタントに精度が改善することを示した．

上記の $3 \times 3$ 畳み込み層の利用は，モデルアーキテクチャをシンプルにするだけではなく，より大きなカーネルサイズの畳み込み層を利用する場合と比較して，表現能力とパラメータ数のトレードオフを改善する効果がある．例えば，$3 \times 3$ の畳み込み層を 2 つ重ねたネットワークは，$5 \times 5$ の畳み込み層と同一の receptive field(注3) を持ちつつ，パラメータ数を $5 \times 5 = 25$ から $3 \times 3 \times 2 = 18$ に削減できていると言える．更に，文献 [46] の主張である深さを増加させることができることから，その後のモデルアーキテクチャでは $3 \times 3$ の畳み込み層が標準的に利用されることとなる．

表 4 に，16 層の VGGNet である VGG16 の構造を示す．AlexNet や ZFNet で利用されていた LRN は，VGGNet のような深いネットワークではあまり効果がなかったことが確認されており，利用されていない．VGGNet は，従来と比較して深いネットワークであるため学習が難しく，まず浅いネットワークを学習し，その後畳み込み層を追加した深いネットワークを学習するという方針を取っている．一方，その後の検証で，Xavier の初期化 [13] を利用することで，事前学習なしでも深いネットワークの学習が可能であると報告されている．

### 2.5 ResNet

Residual Networks (ResNet) [17] は，2015 年の ILSVRC の優勝モデルである．VGGNet で示されたように，ネットワークを深くすることは表現能力を向上させ，認識精度を改善するが，あまりにも深いネットワークは効率的な学習が困難であった．ResNet は，通常のネットワークのように，何かしらの処理ブロックによる変換 $F(x)$ を単純に次の層に渡していくのではなく，その処理ブロックへの入力 $x$ をショートカットし，$H(x) = F(x) + x$ を次の層に渡すことが行われる．このショートカットを含めた処理単位を residual モジュールと呼ぶ．ResNet では，ショートカットを通して，backpropagation 時に勾配が直接下層に伝わっていくことになり，非常に深いネットワークにおいても効率的に学習ができるようになった．ショートカットを利用するアイディアは，gate 関数によって $x$ と $F(x)$ の重みを適用的に制御する Highway Networks [49,50] においても利用されているが，非常に深いネットワークにおいて精度を改善するには至っていなかった．

表 4　VGG16 の構造

| Layer | Filter size | Stride | Output size |
|---|---|---|---|
| input | | | (224, 224, 3) |
| conv×2 | (3, 3) | (1, 1) | (224, 224, 64) |
| maxpool | (2, 2) | (2, 2) | (112, 112, 64) |
| conv×2 | (3, 3) | (1, 1) | (112, 112, 128) |
| maxpool | (2, 2) | (2, 2) | (56, 65, 256) |
| conv×3 | (3, 3) | (1, 1) | (56, 65, 512) |
| maxpool | (2, 2) | (2, 2) | (28, 28, 512) |
| conv×3 | (3, 3) | (1, 1) | (28, 28, 512) |
| maxpool | (2, 2) | (2, 2) | (14, 14, 512) |
| conv×3 | (3, 3) | (1, 1) | (14, 14, 512) |
| maxpool | (2, 2) | (2, 2) | (7, 7, 512) |
| fc | | | 4096 |
| fc | | | 4096 |
| fc | | | 1000 |

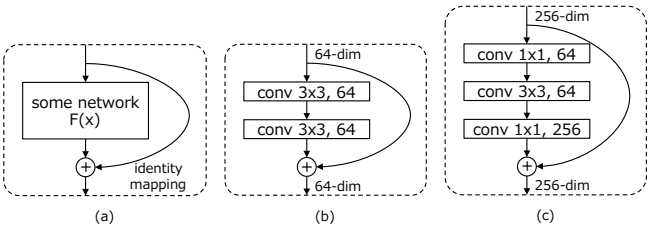



図 2　Residual モジュールの構造

**Residual モジュール．** 図 2 に residual モジュールの構造を示す．図 2 (a) は residual モジュールの抽象的な構造を示し，図 2 (b) は実際に使われる residual モジュールの例を示しており，出力チャネル数が 64 の $3 \times 3$ の畳み込み層(注4) が 2 つ配置されている．正確には，畳み込み層に加えて，後述する batch normalization と ReLU が配置されており，文献 [17] の ResNet では下記のような構造の residual モジュールが利用される：

conv - bn - relu - conv - bn - add - relu

ここで add は，$F(x)$ と $x$ の和を示している．この residual モジュールの構造に関しては複数の改良手法が提案されており，3.1 章で詳述する．

図 2 (c) は，residual モジュールの bottleneck バージョンと呼ばれるものであり，$1 \times 1$ の畳み込みにより，次元削減を行った後に $3 \times 3$ の畳み込みを行い，その後さらに $1 \times 1$ により次

(注3)：Receptive field（受容野）とは，ある特徴マップの 1 画素が集約している前の層の空間の広がりであり，受容野が広いほど認識に有効な大域的なコンテキスト情報を含んでいる．

(注4)：出力チャネル数は，ResNet 内の位置により変化するが，ここでは後述する bottleneck バージョンとの比較のために具体的なチャネル数を記載している．

表 5　ResNet-34 の構造

| Layer | Filter size | Stride | Output size |
|---|---|---|---|
| input | | | (224, 224, 3) |
| conv | (7, 7) | (2, 2) | (112, 112, 64) |
| maxpool | (3, 3) | (2, 2) | (56, 56, 64) |
| residual | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$ | (1, 1) | (56, 56, 64) |
| residual | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$ | (2, 2) | (28, 28, 128) |
| residual | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$ | (2, 2) | (14, 14, 256) |
| residual | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$ | (2, 2) | (7, 7, 512) |
| averagepool | (7, 7) | (1, 1) | (1, 1, 1024) |
| fc | | | 1000 |

元を復元するという形を取ることで，図 2 (b) と同等の計算量を保ちながら，より深いモデルを構築することができる．実際に，図 2 (b) の residual モジュールを利用した ResNet-34 と比較して，同等のパラメータ数を持つ図 2 (c) のモジュールを利用した ResNet-50 は大きく精度が改善していることが報告されている．

Residual モジュールのショートカットとして，基本的には identity function $f(x) = x$ が利用されるが，入力チャネル数と出力チャネル数が異なる場合には，不足しているチャネルを 0 で埋める zero-padding と，$1 \times 1$ の畳み込みによりチャネル数を調整する projection の 2 パターンのショートカットが選択肢となる．このうち，zero-padding のアプローチのほうが，パラメータを増加させないことから良いとされるが，実装が容易な projection が利用されることも多い．

**Batch normalization.** 深いネットワークでは，ある層のパラメータの更新によって，その次の層への入力の分布がバッチ毎に大きく変化してまう内部共変量シフト (internal covariate shift) が発生し，学習が効率的に進まない問題があった．Batch normalization [26] は，この内部共変量シフトを正規化し，なるべく各レイヤが独立して学習が行えるようにすることで，学習を安定化・高速化する手法である．ResNet ではこの batch normalization を residual モジュールに組み込むことで深いネットワークの効率的な学習を実現しており，ResNet 以降のモデルでは，batch normalization が標準的に用いられるようになった．

**ResNet の構造.** ResNet は，前述の residual モジュール複数積み重ねることにより構築される．表 5 に，例として 34 層の ResNet-34 の構造を示す．まず stride が (2, 2) の $7 \times 7$ 畳み込みを行った後，$s = 2, z = 3$ の max pooling を行うことで特徴マップを縮小する．その後は，VGGNet と同様に，同一の出力チャネル数を持つ residual モジュールを複数重ね，その後に特徴マップを半分に縮小しつつ出力チャネル数を 2 倍にすることを繰り返すことでネットワークが構成される．特徴マップを半分に縮小する処理は，max pooling ではなく，各 residual モジュールの最初に stride が (2, 2) の畳み込みを行うことで実現している．また GoogLeNet と同様に全結合層の前は GAP を利用するという方針を取っている．

**He の初期化.** VGGNet では，ランダムに初期化する重みのスケールを適切に設定する Xavier の初期化 [13] を利用することで，深いネットワークでも事前学習なしで学習が可能であると報告されていた．しかしながら，Xavier の初期化で行われるスケーリングは，線形の活性化関数を前提としており，ReLU を活性化関数として利用している場合には適切ではない．これに対し，文献 [16] では，ReLU を活性化関数として利用する場合の適切なスケーリングを理論的に導出しており，ResNet では，この He の初期化が用いられる．

ResNet の興味深い性質として，ランダムに 1 つだけ residual モジュールを削除したとしても，認識精度がほとんど低下しないことが挙げられる [56]．これは，residual モジュールのショートカットを再帰的に展開していくと，異なる深さのネットワークを統合しているネットワークと同値であることが示されているように，ResNet が暗黙的に複数のネットワークのアンサンブル学習を行っているためと考えられている．

### 2.6　SENet

Squeeze-and-Excitation Networks (SENet) [20] は，2017 年の ILSVRC の優勝モデルであり，特徴マップをチャネル毎に適応的に重み付けする Attention の機構を導入したネットワークである．この Attention の機構は，Squeeze-and-Excitation Block (SE Block) によって実現される．図 3 に SE Block の構造を示す．

SE Block は名前の通り，Squeeze ステップと Excitation ステップの 2 段階の処理が行われる．Squeeze ステップでは，$H \times W \times C$ の特徴マップに対し GAP が適応され，画像の全体的な特徴が抽出された $1 \times 1 \times C$ のテンソルが出力される．次に，Excitation ステップでは，$1 \times 1$ の畳み込みにより，特徴マップのチャネル間の依存関係が抽出される．具体的には，出力チャネル数が $C/r$ の $1 \times 1$ の畳み込みが適用され，ReLU を経て，更に出力チャネル数 $C$ の $1 \times 1$ の畳み込みが適用される．最後にシグモイド関数が適用され，特徴マップのチャネル毎の重みが出力される．このチャネル毎の重みを用いて特徴マップをスケーリングすることで，画像全体のコンテキストに基づいた特徴選択を実現している．

この SE Block の機構は極めて汎用的で，基本的にはどのようなモデルにも導入することができる．文献 [20] では，ResNet や，後述する ResNeXt，Inception-v3，Inception-ResNet-v2 といったモデルに SE Block を導入し，コンスタントに精度改善を実現している．

SENet では，チャネル毎の Attention を適用しているが，空間・チャネル両方に対して Attention を適用している手法も存在する [57].

## 3.　最新の CNN 改良手法

ILSVRC で優秀な成績を納めた手法以外にも様々な CNN の改良手法が提案されている．本章では，これらの手法を，1) residual モジュールの改良，2) 独自モジュールの利用，3) 独自マクロアーキテクチャの利用，4) 正則化，5) 高速化を意識

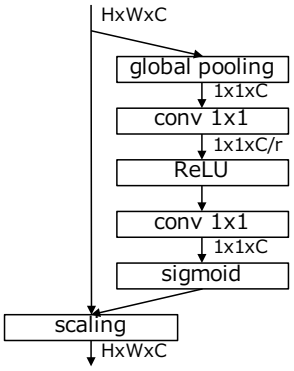



図 3　SE Block の構造

したアーキテクチャ，6) アーキテクチャの自動設計の 6 種類に分類し，それぞれ解説する．

### 3.1　Residual モジュールの改良

ResNet は，residual モジュールを重ねていくだけというシンプルな設計でありながら，高精度な認識を実現できることから，デファクトスタンダードなモデルとなった．これに対し，residual モジュール内の構成要素を最適化することで，性能改善を図る手法が複数提案されている．

初期の ResNet の文献 [17] では，下記のような residual モジュールの構造が提案されていた：

conv - bn - relu - conv - bn - add - relu

これに対し，文献 [18] では，下記のように，BN および ReLU を畳み込み層の前に配置することで精度が改善することが示されている：

bn - relu - conv - bn - relu - conv - add

これは，ショートカットの後に ReLU によるアクティベーションを行わないことで，勾配がそのまま入力に近い層に伝わっていき，効率的な学習ができるためと考えられる．単純に ResNet と参照した場合，こちらの構造を示していることもあるため注意が必要である．明示的に上記の構造の residual モジュールを利用した ResNet であることを示す場合には，pre-activation (pre-act) の ResNet と参照されることが多い．

文献 [15] では，pre-act の residual モジュール内の ReLU の数を 1 つにし，更に最後に BN を加えることが提案されている：

bn - conv - bn - relu - conv - bn - add

Residual モジュール内の ReLU の数を 1 つにすることで精度が改善するということは，文献 [9] でも主張されている．

文献 [60] では，pre-act の residual モジュールについて，最後の畳み込み層の直前に dropout を入れることで僅かに精度が向上することが示されている：

bn - relu - conv - bn - relu - dropout - conv - add

上記までの説明では，ResNet の bottleneck バージョンの構造は示していないが，bottleneck バージョンにおいても同様の傾向が確認されている．

**WideResNet.** 文献 [60] では，ResNet に対し，層を深くする代わりに，各 residual モジュール内の畳み込みの出力チャネル数を増加させた wide なモデルである，Wide Residual Networks (WideResNet) が提案されている．本論文の主張は，深く thin なモデルよりも，浅く wide なモデルのほうが，最終的な精度および学習速度の点で優れているというものである．例えば，16 層の WideResNet が，1000 層の ResNet と比較して，同等の精度およびパラメータ数で，数倍早く学習できることが示されている．また，WideResNet の中でも比較的深いモデルでは，residual モジュール内の 2 つの convolution 層の間に dropout を挿入することで精度が向上することも示されている．

**PyramidNet.** 文献 [56] では，ResNet が複数のネットワークのアンサンブル学習となっており，ランダムに residual モジュールを削除しても精度がほとんど低下しないことが示されていた．しかしながら，特徴マップのサイズを半分ダウンサンプルする residual モジュールを削除した場合に限っては，相対的に大きな精度低下が確認されていた．これは，ダウンサンプリングを行う residual モジュールでは，出力チャネル数を倍増させており，相対的にそのモジュールの重要度が大きくなってしまっているためと考えられる．アンサンブル学習の観点からは，特定のモジュールの重要度が大きくなってしまうことは望ましくなく，これを解決するネットワークとして Pyramidal Residual Networks (PyramidNet) が提案されている [15].

PyramidNet では，ダウンサンプルを行うモジュールのみで出力チャネル数を倍増させるのではなく，全ての residual モジュールで少しずつ出力チャネル数を増加させる．増加のさせ方として，単調増加させる場合と指数的に増加させる場合を比較し，単調増加させる場合のほうが精度が良いことが示されている．単調増加させる場合，$k$ 番目の residual モジュールの出力チャネル数 $D_k$ は次のように定義される：

$$D_k = \begin{cases} 16 & \text{if } k = 1 \\ \lfloor D_{k-1} + \alpha/N \rfloor & \text{otherwise.} \end{cases}$$

PyramidNet は，bottleneck バージョンの ResNet をベースとし，272 層という深いネットワークを学習させることで，非常に高精度な認識を実現している．

### 3.2　独自モジュールの利用

Residual モジュールや Inception モジュールの成功から，それらに代わる新たなモジュールが多数提案されている．多くの手法が residual モジュールをベースとしている．

#### 3.2.1　ResNeXt

ResNeXt [59] は，ResNet 内の処理ブロック $F(x)$ において，下式のように入力 $x$ を多数分岐させ，同一の構造を持つニューラルネットワーク $\mathcal{T}_i(x)$ で処理を行った後，それらの和を取る ResNeXt モジュールを利用する手法である：

$$F(x) = \sum_{i=1}^{C} \mathcal{T}_i(x).$$

ここで，分岐数 $C$ は cardinality と呼ばれている．このアイ

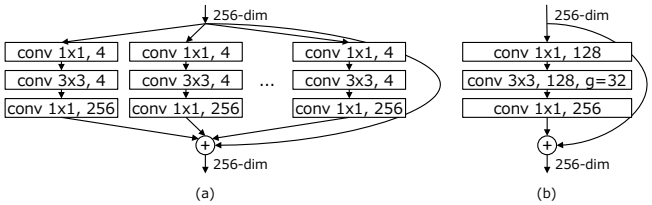
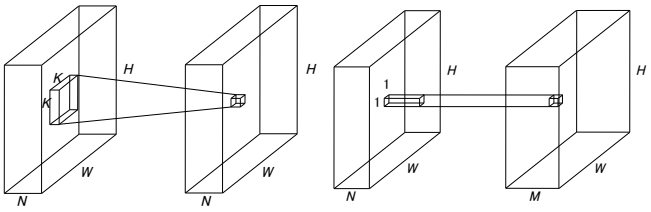



図 4　ResNeXt モジュールの構造




(a) Depthwise 畳み込み　　(b) Pointwise 畳み込み

図 5　Separable 畳み込みの各処理

ディアは，通常のニューラルネットワークの処理 $\sum_{i=1}^{D} w_i x_i$ を，より汎用的な $\mathcal{T}_i(x)$ に置き換えたものであることから，Network-in-Neuron と呼ばれる．

図 4 (a) に，入出力チャネル数が 256，$C = 32$ とした場合の ResNeXt モジュールの構造を示す．ここで $\mathcal{T}_i(x)$ は，conv 1x1, 4 - conv 3x3, 4 - conv 1x1, 256 を順に適用する処理として定義されている．図 4 (a) の構造は，実は図 4 (b) のように書き換えることができる．図 4 (b) において 2 つ目の畳み込みは grouped convolution（本稿では group 化畳み込みと呼ぶ）と呼ばれ，入力特徴マップを $g$ 分割し，それぞれ独立に畳み込みを行う処理である．図 4 (b) の構造は，conv 1x1 により次元削減を行い，conv 3x3 を行った後，conv 1x1 により次元の復元を行っており，実は bottleneck バージョンの residual モジュールにおいて，$3 \times 3$ の畳み込みを group 化畳み込みに変更したものとみなすことができる．Group 化畳み込みは，Inception モジュールと同様に，チャネル方向の結合が疎なパラメータの少ない畳み込みである．結果的に ResNeXt は，ResNet と比較して表現力とパラメータ数のトレードオフが改善され，同等のパラメータ数では精度向上を実現することができている．

#### 3.2.2　Xception

Xception [6] は，端的には，通常の畳み込みの代わりに，depthwise separable convolution [44]（以降 separable 畳み込み）を用いた ResNet である．

**Separable 畳み込み．**通常の畳み込みが，入力特徴マップの空間方向とチャネル方向に対し同時に畳み込みを行うのに対し，separable 畳み込みは，空間方向とチャネル方向に独立に畳み込みを行う．これは，畳み込みがこれらの方向にある程度分離することができるという仮説に基づいている．空間方向の畳込みは depthwise 畳み込み，チャネル方向の畳み込みは pointwise 畳み込みと呼ばれる．図 5 に，separable 畳み込みの各処理を示す．

Depthwise 畳み込みは，特徴マップのチャネル毎にそれぞれ独立して空間方向の畳込みを行う処理である．Pointwise 畳み込みは，本稿でも何度も登場した，$1 \times 1$ の畳み込みのことを指す．入力特徴マップのサイズが $H \times W \times N$，出力チャネル数が $M$ の場合，通常の $K \times K$ 畳み込みの計算量は $\mathcal{O}(HWNK^2M)$ となる．他方，depthwise 畳み込みの計算量は $\mathcal{O}(HWNK^2)$，pointwise 畳み込みの計算量は $\mathcal{O}(HWNM)$ となる．すなわち，通常の畳み込みを separable 畳み込み（depthwise 畳み込み + pointwise 畳み込み）に置き換えることで，計算量が $\mathcal{O}(HWNK^2M)$ から $\mathcal{O}(HWNK^2 + HWNM)$ に削減される．比率では，$1/K^2 + 1/M$ になっており，通常 $M >> K^2$ である (e.g. $K = 3$, $M = 64$) ことから，計算量が $1/K^2$ 程度に削減される(注5)．

**Xception モジュール．**Xception で用いられるモジュールは下記のようなものである：

relu - sep - bn - relu - sep - bn - relu - sep - bn - add

ここで sep は separable 畳み込みを表す．全体の設計としては，ネットワークの入出力に近い箇所以外は上記の Xception モジュールを用い，$s = 2$, $z = 3$ の max pooling により特徴マップを縮小しつつ，そのタイミングでチャネル数を増加させる方針を取っている．

Xception は，上記のように通常の畳み込みよりも計算量およびパラメータ数の小さい separable 畳み込みを用いることで，その分モデルの深さや幅を大きくすることができ，結果的に ResNet や Inception-v3 よりも高精度な認識を実現している．

### 3.3　独自マクロアーキテクチャの利用

Residual モジュールの改良や，独自モジュールの利用では，特定のモジュールを順番に積み重ねるというマクロなアーキテクチャは同じであった．一方，そのマクロなアーキテクチャについても独自の提案している文献も存在する．

#### 3.3.1　RoR

Residual Networks of Residual Networks (RoR) [63] は，複数の residual モジュール間に更にショートカットを追加することで，ResNet を更に最適化しやすくするモデルである．ショートカットは，階層的に構築することが提案されており，実験的に 3 階層までのショートカットが効果的であったことが示されている（1 階層は通常の ResNet）．RoR は，ベースネットワークとして，ResNet，pre-act の ResNet，WideResNet を比較しており，WideResNet をベースとし，3.4.1 章で説明する Stochastic Depth と RoR を組み合わせた場合に最も良い認識精度が得られている．なお，Stochastic Depth を導入しない場合は逆に精度が低下することが確認されている．

#### 3.3.2　FractalNet

FractalNet [30] は，下記のように再帰的に定義される fractal ブロックを利用することで，ResNet のようなショートカットを利用することなく深いネットワークを学習することができるモデルである：

(注5)：Depthwise 畳み込みの処理時間は実装に大きく依存し，計算量通りの処理時間比にはならないことが多い．

$$f_1(x) = \mathrm{conv}(x),$$
$$f_{C+1}(x) = (f_C \circ f_C)(x) \oplus \mathrm{conv}(x).$$

ここで $\oplus$ は複数のパスを統合する処理であり，文献 [30] では，要素ごとの平均値を取るオペレーションとして定義される．上記の処理ブロックを，間に $s = 2, z = 2$ の max pooling をはさみながら重ねていくことで，FractalNet が構成される．ブロック数が $B$ の FractalNet の層数は $B \cdot 2^{C-1}$ となる．FractalNet の学習で特徴的であるのは，出力層まで存在する多数のパスを dropout のように確率的に drop することを行う点である．Drop の種類として，local と global の 2 種類の drop 方法を提案している．

**Local.** パスを統合する層で入力パスをランダムに drop させる．但し，最低 1 つのパスを残す．

**Global.** 出力層へ至る同一の列により定義されるパスを 1 つだけ利用する．

上記のパスを drop する処理により，ネットワークの正則化が行われ，ResNet よりも高精度な認識を実現している．但し，WideResNet や DenseNet に対しては，精度面で劣っている．

#### 3.3.3 DenseNet

Dense Convolutional Network (DenseNet) [22] は，ネットワークの各レイヤが密に結合している構造を持つことが特徴のモデルである．

**Dense ブロック.** ResNet では，$l$ 番目の residual モジュールの出力は，内部の処理ブロックの出力 $F_l(x_{l-1})$ とショートカット $x_{l-1}$ の和としていた：

$$x_l = F_l(x_{l-1}) + x_{l-1}.$$

これに対し，DenseNet では，その内部では，あるレイヤより前のレイヤの出力全てを連結した特徴マップをそのレイヤの入力にする，Dense ブロックを利用する．Dense ブロック内における $l$ 番目のレイヤの出力は下式で定義される：

$$x_l = F_l([x_0, x_1, \cdots, x_{l-1}]).$$

ここで，Dense ブロックへの入力のチャネル数を $k_0$，各レイヤの出力 $F_l(\cdot)$ のチャネル数を $k$ とすると，$l$ 番目のレイヤの入力チャネル数は $k_0 + k(l-1)$ となる．このように，入力チャネル数が $k$ ずつ増加するため，$k$ はネットワークの成長率パラメータと呼ばれる．なお，各レイヤの処理 $F_l$ は，bn - relu - conv3x3 により構成される．

**Bottleneck バージョン.** DenseNet では，各レイヤの出力チャネル数 $k$ は小さい値となっているが，入力チャネル数が非常に大きくなるため，計算量を削減するために ResNet で利用されている入力チャネル数を圧縮する bottleneck 構造を利用する．具体的には，各レイヤの処理 $F_l$ を下記により定義する：

bn - relu - conv1x1 - bn - relu - conv3x3

ここで，conv 1x1 の出力チャネル数は $4k$ と設定される．

**Transition レイヤ.** DenseNet は，上記の Dense ブロックを複数積み重ねることで構築され，各 Dense ブロックは transition レイヤにより接続される．Transition レイヤは，

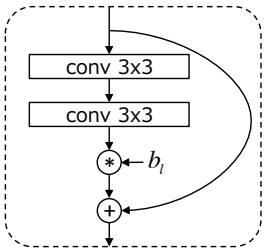



図 6　Stochastic depth における residual モジュールの構造

bn - conv1x1 - avepool2x2

により構成される．この transition レイヤは，通常入力チャネル数と出力チャネル数は同一とされるが，$\theta \in (0, 1]$ により定義される圧縮率だけ出力チャネル数を削減することも提案されており，$\theta = 0.5$ が用いられる．評価実験では，bottleneck を利用し，かつ transition レイヤでの圧縮を行い，$k$ および層数を大きくしたバージョンが高精度な認識を実現している．

文献 [21] では，DenseNet を複数スケールの特徴マップを持つように拡張し，更にネットワークの途中で結果を出力することで，サンプルの難易度によって処理時間を可変とする Multi-Scale DenseNet (MSDNet) が提案されている．

### 3.4 正　則　化

DNN においては，いかに過学習を回避し，汎化されたモデルを学習するかが重要であり，モデルに対しては dropout や weight decay 等が，学習データおよびテストデータに対しては，ランダムクロッピングや左右反転等のデータ拡張が正則化のために用いられてきた．近年，このような正則化に関してもシンプルでありながら有効な手法が提案されている．

#### 3.4.1 Stochastic Depth

Stochastic Depth [23] は，ResNet において，訓練時に residual モジュールをランダムに drop するという機構を持つモデルである．図 6 に，Stochastic Depth における $l$ 番目の residual モジュールの構造を示す．ここで，$b_l$ は確率 $p_l$ で 1 を，確率 $1 - p_l$ で 0 を取るベルヌーイ変数である．$p_l$ は，$l$ 番目の residual モジュールが drop されずに生き残る確率（生存確率）であり，ネットワークの出力層に近いほど小さな値を取るように設計され，$p_l = 1 - \frac{l}{2L}$ と定義される（$L$ は residual モジュール数）．これにより，訓練時の「期待値で見たときの深さ」が浅くなり，学習に必要な時間が短縮されるとともに，dropout のような正則化の効果が実現される．なお，テスト時には，それぞれの residual モジュールについて生存確率の期待値 $p_l$ を出力にかけることでスケールのキャリブレーションを行う．

#### 3.4.2 Swapout

Swapout [47] は，ResNet に対して dropout の拡張を行う正則化手法である．ResNet の residual モジュールでは，入力を $x$，residual モジュール内での処理の出力を $F(x)$ とすると，$H(x) = F(x) + x$ を次の層に出力する．これに対し，Swapout では，入力のショートカット $x$ および $F(x)$ に対し，個別に dropout を適用する．正確には，$H(x)$ が下記のように定義さ

れる：

$$H(x) = \Theta_1 \odot x + \Theta_2 \odot F(x).$$

ここで，$\Theta_1$ および $\Theta_2$ は，各要素が独立に生成されるベルヌーイ変数により構成される，出力テンソルと同サイズのテンソルであり，$\odot$ はアダマール積である．文献 [47] では，stochastic depth と同様に，drop 率を入力層から出力層まで，0 から 0.5 まで線形に増加させる場合に精度が高くなることが示されている．推論時は，dropout と異なり，明示的に各層の出力を期待値によりキャリブレーションできないため，テストデータに対し swapout を有効にしたまま複数回 forward を行い，それらの平均値を推論結果とする形でないと精度がでないことが特徴である．

**3.4.3** Shake-Shake Regularization

Shake-Shake [11, 12] は ResNet をベースとし，ネットワークの中間の特徴マップに対する data augmentation を行うことで，強力な正則化を実現する手法である．

図 7 に，Shake-Shake で利用される $l$ 番目の residual モジュールの構造を示す．Shake-Shake では，residual モジュール内の畳み込みを 2 つに分岐させ，forward 時にはそれらの出力を一様乱数 $\alpha_l \in [0, 1]$ によって混合することを行う．これにより，画像を対象とした data augmentation においてランダムクロッピングを行うことで，その画像内に含まれている物体の割合が変動してもロバストな認識ができるように学習ができるように，Shake-Shake では特徴レベルにおいても各特徴の割合が変動してもロバストな認識ができるようにしていると解釈することができる．

特徴的なのは，backward 時には，forward 時の乱数 $\alpha_l$ とは異なる一様乱数 $\beta_l \in [0, 1]$ を利用する点である．これは，勾配にノイズを加えると精度が向上する [38] ように，forward 時とは異なる乱数を利用することで，更に強い正則化の効果をもたらしていると考えられる．推論時には，乱数の期待値である 0.5 を固定で利用する．

上記の $\alpha_l$ と $\beta_l$ については，複数のパターンの組み合わせを網羅的に検証した結果，どちらも独立した乱数とする形が良いと結論付けている．また，バッチ単位で上記の乱数を同一のものを利用するか，画像単位で独立に決定するかについても，画像単位で独立に決定するほうが良いと実験的に示されている．このように，forward/backward 時の外乱により，強い正則化の効果がもたらされ，4.4 章でも実験的に示すように，非常に高精度な認識を実現している．

Shake-Shake の学習で特徴的な点として，学習率の減衰を cosine 関数で制御し，通常 300 エポックかけて学習を行うところを，1800 エポックかけて学習することが挙げられる(注6)．これは Shake-Shake の効果により，擬似的に学習データが非常に大量にあるような状態となっているため，長時間の学習が有効であるためと考えられる．

**3.4.4** ShakeDrop

ShakeDrop [1] は，Stochastic Depth において，層を drop する代わりに，forward/backward 時に Shake-Shake のような外乱を加える手法である．図 8 に ShakeDrop で利用される $l$ 番目の residual モジュールの構造を示す．ここで，$b_l$ は，確率 $p_l$ で 1 を，$1 - p_l$ で 0 を取るベルヌーイ変数である．Stochastic Depth と同様に $p_l$ はネットワークの出力層に近いほど大きな値を取り，$p_l = 1 - \frac{l}{2L}$ と定義される（$L$ は residual モジュールの数）．

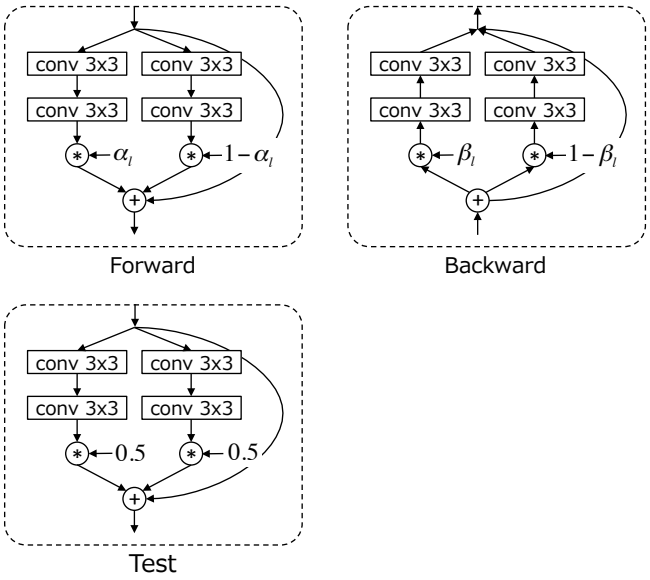



図 7　Shake-Shake における $l$ 番目の residual モジュールにおける Forward, Backward, Test 時の構造

$\alpha_l$ と $\beta_l$ は Shake-Shake と同様に，forward/backward 時に出力をスケーリングする乱数である．テスト時には，forward 時のスケーリング $b_l + (1 - b_l)\alpha_l$ の期待値を用いて，出力のキャリブレーションを行う．ShakeDrop において，$p_l = 1$（常に drop しない）とすれば通常の residual モジュールと同一になり，$p_l = 0$（常に drop する）とすれば，Shake-Shake と同じように全ての入力に対し外乱を加える reisual モジュールとなる．また，$\alpha_l = 0$ と $\beta_l = 0$ とすれば，Stochastic Depth と同一のモデルとなる．

$\alpha_l$ と $\beta_l$ が取りうる範囲は，幾つかの候補から最も精度が良い $\alpha_l \in [-1, 1]$ と $\beta_l \in [0, 1]$ が採用されている．$\alpha_l$ と $\beta_l$ をどの単位で変化させるかについて，Shake-Shake ではバッチ単位と画像単位の比較が行われていたが，本論文では更にチャネル単位と画素単位でも比較が行われており，チャネル単位での精度が良いことが示されている．

ShakeDrop は，Shake-Shake と比較して，2 つに分岐させていた畳み込みを 1 つにした構造でも同様の正則化を実現している点で優れている．これにより，パラメータ数を削減できるため，相対的にモデルをより深くすることが可能となり，ベースネットワークとして PyramidNet を利用し，更に後述する random erasing と組み合わせることで，本稿執筆時点（2017 年 11 月）では，CIFAR10/100 において最も低いエラー率を達成している．

(注6)：4.4 章で示すように，300 エポックの学習でも十分な精度が得られる．

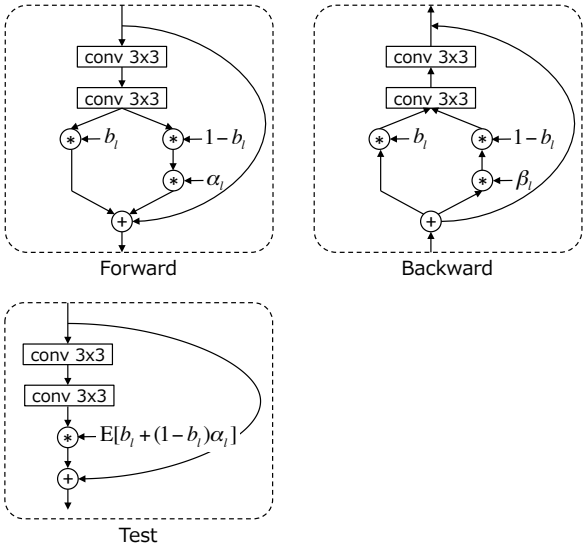



図 8　ShakeDrop における $l$ 番目の residual モジュールにおける Forward, Backward, Test 時の構造

#### 3.4.5　Cutout / Random Erasing

Cutout [8] および Random Erasing [65] は，モデルの正則化を目的とした data augmentation の手法である．同じく正則化を目的とした dropout は全結合層では効果的である一方，畳み込み層に対しては元々パラメータが少ないため効果が限定的であった．また，CNN の入力である画像が対象の場合，隣接画素に相関があり，ランダムに drop したとしてもその周りのピクセルで補間できてしまうため，正則化の効果が限定的であった．これに対し，Cutout/Random Erasing では，入力画像のランダムな領域をマスクしてしまうことで，より強い正則化の効果を実現している．

Cutout では，マスクの形よりもサイズが重要であるとの主張から，マスクの形状は単純なサイズ固定の正方形を採用し，そのマスク領域を平均画素に置き換える処理を行っている．

Random Erasing では，各画像に対しマスクを行うかどうか，マスク領域のサイズ，アスペクト比，場所をランダムに決定し，マスク領域の画素値をピクセルレベルでランダムな値に置き換える処理を行っている．単純な手法ながら，画像分類タスクだけではなく，物体検出や人物照合タスクについても有効性が確認されている．図 9 に，Random Erasing を行った画像例を示す．

#### 3.4.6　mixup

mixup [62] は，2 つの訓練サンプルのペアを混合して新たな訓練サンプルを作成する data augmentation 手法である．具体的には，データとラベルのペア $(X_1, y_1)$, $(X_2, y_2)$ から，下記の式により新たな訓練サンプル $(X, y)$ を作成する：

$$X = \lambda X_1 + (1-\lambda)X_2,$$
$$y = \lambda y_1 + (1-\lambda)y_2.$$

ここで，ラベル $y_1, y_2$ は one-hot 表現のベクトルであり，$X_1, X_2$ は任意のベクトルやテンソルで表現される学習データである．また，$\lambda \in [0,1]$ は，ベータ分布 $Be(\alpha, \alpha)$ からのサンプリング

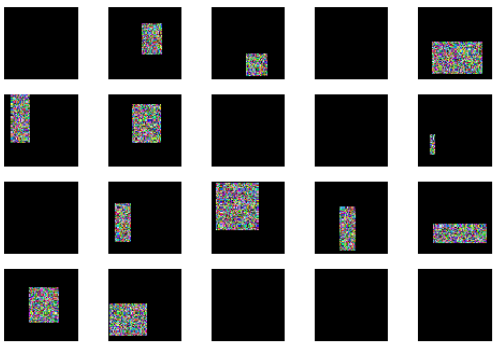


図 9　Random erasing を適用した画像例

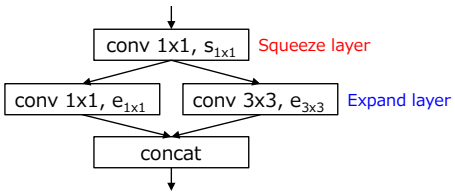



図 10　Fire モジュールの構造

により取得し，$\alpha$ はハイパーパラメータである．データ $X_1, X_2$ だけではなく，ラベル $y_1, y_2$ も混合する点が特徴的であり，画像認識においてもその有効性が主張されている．

### 3.5　高速化を意識したアーキテクチャ

組み込みデバイスや，スマートフォン等の，計算資源が潤沢ではない環境においては，高速に動作するモデルが重要となる．そのような高速化を意識したアーキテクチャも提案されている．

#### 3.5.1　SqueezeNet

SqueezeNet [25] は，$1 \times 1$ 畳み込みを活用してパラメータと計算量を削減した fire モジュールを積み重ねることで構築される軽量なモデルである．図 10 に fire モジュールの構造を示す．

Fire モジュールではまず，residual モジュールの bottleneck バージョンや，Inception モジュールのように，squeeze レイヤの $1 \times 1$ 畳み込みによって入力特徴マップの次元が削減される．その後，expand レイヤの $3 \times 3$ の畳み込みにより特徴抽出を行いつつ次元の復元を行うが，その一部を $1 \times 1$ の畳み込みに置き換える．これは Inception モジュールと同様に，明示的に sparse な畳み込みを行うことでパラメータ数を削減する効果がある．文献 [25] では，入力チャネル数と squeeze レイヤの出力チャネル数 $s_{1\times1}$ の比，および expand レイヤの $1 \times 1$ 畳み込みの出力チャネル数 $e_{1\times1}$ と $3 \times 3$ 畳み込みの出力チャネル数 $e_{3\times3}$ の比を調整することで，精度低下を抑えながらパラメータ数を大幅に削減できることが示されている．

#### 3.5.2　MobileNet

MobileNet [19] は，Xception と同様に separable 畳み込みを多用することにより，計算量を削減した軽量なモデルである．具体的には，下記の処理ブロックを，stride が 2 の畳み込みにより特徴マップを縮小しつつ，チャネル数を 2 倍に増加させる

ことを行いながら積み重ねることでモデルが構築される：

dw - bn - relu - pw - bn - relu

ここで dw は depthwise 畳み込み，pw は pointwise 畳み込みを表している．Xception との大きな違いは，ResNet のようなショートカットを利用しないことと，depthwise 畳み込みと pointwise 畳み込みの間に batch normalization と ReLU を利用している点である．特に後者については，文献 [6] では batch normalization と ReLU を利用しないほうが精度が高いことが示されており，全体的なモデルアーキテクチャによって局所的なモジュールの最適な構成が異なることが示唆されている．MobileNet では，チャネル数を制御するパラメータ $\alpha \in (0, 1]$ と，入力画像サイズを制御するパラメータ $\rho \in (0, 1]$ を導入し，同一のモデルアーキテクチャで，これらのハイパーパラメータにより精度と処理速度のトレードオフを簡単に調整することができる．

### 3.6 モデルアーキテクチャの自動設計

これまで説明してきたモデルは全て人手によりモデルアーキテクチャのデザインが行われてきた．これに対し，自動的にモデルアーキテクチャを設計する試みも存在する [33, 66, 67]．例えば，文献 [66] では，ネットワークを構成するレイヤのパラメータを出力する Recurrent Neural Networks (RNN) を構築し，この RNN を REINFORCE アルゴリズムで学習させることを提案している．この RNN を 800 個の GPU により学習し，12,800 個のモデルを実際に生成および学習した結果，DenseNet に迫る精度のモデルの生成に成功している．

## 4. 各モデルの精度および処理速度の検証

本章では，これまで説明してきた代表的なモデルについて，複数のデータセットを用いて学習および網羅的な精度評価を行い，各モデルの精度および学習時間の傾向について議論を行う．

### 4.1 データベース

各種アーキテクチャを精度比較するデータベースとして，小規模な CIFR10 および CIFAR100, 大規模な ImageNet, Places および Places2，COCO などある．CIFAR10 および CIFAR100 は 10 クラスまたは 100 クラスの一般物体認識データセットである．画像サイズが $32 \times 32$ ピクセル，学習データ数が 5 万枚，評価データが 1 万枚となっており，比較的短時間で学習することができる．

一方，ImageNet は ILSVRC で利用されているデータセットである．ILSVRC の位置特定 (Object localization:LOC) タスクでは，1000 クラスの物体の認識と位置特定の精度を競っている．学習データは 120 万枚，テストデータは 10 万枚あり，画像サイズは一定ではない．多くのアーキテクチャでは，画像サイズを $224 \times 224$ または $229 \times 229$ にリサイズして入力している．

ImageNet と同様の大規模な Places データセットは，1000 万枚規模となっており，400 クラス以上の屋内または屋外シーンを対象とした画像から構成されている．また，COCO データセットは，物体検出やセマンティックセグメンテーション，人体の骨格検出だけでなく，画像に対するキャプションデータも付与されている大規模なデータセットである．ILSVRC と同様に Places と COCO データセットを利用したコンペティションも開催されている．

表 6　精度比較を行うアーキテクチャ

| アーキテクチャ名 | 層数 | データベース |
|---|---|---|
| AlexNet | 8 | ImageNet |
| VGG | 16 | CIFAR10/100，ImageNet |
| GoogLeNet | 22 | CIFAR10/100，ImageNet |
| SqueezeNet | 10 | CIFAR10/100，ImageNet |
| ResNet | 18/50/101 | CIFAR10/100，ImageNet |
| WideResNet | 101 ($k$=4) | CIFAR10/100 |
| PyramidNet | 101 ($\alpha = 84$) | CIFAR10/100 |
| FractalNet | 20 ($Cols = 3$) | CIFAR10/100 |
| DenseNet | 72 ($k = 12$) | CIFAR10/100，ImageNet |
| SENet | 50 ($r = 4$) | CIFAR10/100 |

本稿では，各種アーキテクチャの精度比較に CIFAR10 および CIFAR100 を用いる．また，ImageNet データセットについても代表的なアーキテクチャの比較に用いる．

### 4.2 精度比較を行うアーキテクチャ

精度比較するアーキテクチャを表 6 に示す．AlexNet，VGG，GoogLeNet および SqueezeNet は原著と同じ層数，フィルタサイズ，フィルタ数とするが，学習の収束性を高めるために，Batch Normalization を畳み込み層の後に配置する．ResNet の Batch Normalization および ReLU は畳込み後に配置し，各 Residual モジュールの数も原著と同じである．Wide Residual Network (WideResNet) は，層数を 101 とし，幅パラメータ $k$ を 4 としている．また，各 Residual モジュールの数は 101 層の ResNet と同じである．Deep Pyramid Residual Network (PyramidNet) は，層数を 101，$\alpha$ を 84 としている．Fractal Network (FractalNet) は，ブロック数 $B = 5$，フラクタル分割数 $C = 3$ の 20 層のネットワークとする．Densely Network (DenseNet) は，層数を 72，成長率 $k$ を 12 としている．SENet は， 層数を 50，圧縮率 $r$ を 4 とする．

### 4.3 学習および評価の設定

学習において，data augmentation が有効なことは広く知られている．data augmentation の方法としては，位置ズレや回転，反転，スケール変化を加えることが多い．これらの変化を組みわせることで学習データのバリエーションを増やすことができる．CIFAR10 および CIFAR100 における精度比較では，画像の上下左右にそれぞれ 4 ピクセルのマージンを付与し，ランダムに $32 \times 32$ の領域を切り出すことで，位置ズレの変化を加える．また，左右反転をさらにランダムに加える．学習エポック数は 300 エポックとし，最適化にはモーメンタム付きの SGD を用い，学習率は 0.1，モーメンタムは 0.9 とする．学習率は，150 エポックで 0.01，225 エポックで 0.001 に小さくする．バッチサイズは 128 としている．

ImageNet における精度比較では，画像サイズを $256 \times 256$ にリサイズし，$224 \times 224$ の領域をランダムに切り出す．左右反転についてもランダムに加える．バッチサイズは 256，最適

表 7　CIFAR10/100 におけるエラー率および学習時間

| アーキテクチャ名 | エラー率 [% ] | | 学習時間 [h] |
|---|---|---|---|
| | CIFAR10 | CIFAR100 | |
| VGG | 6.0 | 27.4 | 2.9 |
| GoogLeNet | 9.3 | 31.7 | 4.6 |
| SqueezeNet | 10.8 (18.0) | 36.0 | 1.8 |
| ResNet 101 | 5.3 (6.5) | 25.9 | 9.2 |
| WideResNet | 4.8 (4.0) | 20.7 | 19.6 |
| PyramidNet | 4.3 (3.8) | 22.0 | 15.6 |
| ResNeXt | 4.5 (3.6) | 20.7 | 27.0 |
| FractalNet | 5.0 (4.6) | 26.0 | 5.1 |
| DenseNet | 5.1 (3.8) | 24.6 | 19.6 |
| SENet | 4.8 | 24.2 | 36.7 |

表 8　正則化を導入した場合のエラー率および学習時間

| 正則化手法 | エラー率 [% ] | | 学習時間 [h] |
|---|---|---|---|
| | CIFAR10 | CIFAR100 | |
| ResNet 101 | 5.3 (6.5) | 25.9 | 9.2 |
| Stochastic Depth | 5.2 | 23.5 | 12.0 |
| Shake-Shake | 3.0 | 18.4 | 77.8 |
| Swapout | 6.0 | 26.8 | 42.0 |
| ResNet101 + Cutout | 3.5 | 25.0 | 13.1 |

化にはモーメンタム付きの SGD を用い，学習率は 0.1，モーメンタムは 0.9 とする．学習エポック数は 90 エポックとし，学習率を 30 エポックごとに 0.1 倍する．

### 4.4　CIFA10/100 における精度比較

表 7 に CIFAR10 および CIFAR100 におけるエラー率と学習時間を示す．実装時の学習率の設定や初期パラメータにより精度に差があるため，各要素の括弧内に原著での精度を示している．CIFAR10 において，ほとんどの手法がエラー率 10%以下となっている．SqueezeNet は原著では約 18%であったが，本実装では 10.8%と精度が向上しており，学習率などの設定により精度が大きく変わることがわかる．VGG や GoogLeNet，ResNet はそれぞれ 6.0%，9.3%，5.3%となっており，ILSVRC と同様に ResNet のエラー率が低くなっている．ResNet を応用した WideResNet や PyramidNet は，それぞれ 4.8%，4.3%とさらに精度を向上させることができている．一方で，FractalNet や DenseNet は，ResNet とほぼ同等の 5.0%程度となっている．

CIFAR100 においても ResNet が VGG や GoogLeNet よりもエラー率が低く，25.9%となっている．また，WideResNet や PyramidNet，ResNeXt が ResNet よりも精度が高く，20%近くのエラー率となっている．これより，ResNet をベースとして，より幅を持たせた手法が高い性能を達成していることがわかる．

学習時間を比較すると，VGG や GoogLeNet，SqueezeNet は非常に短時間で学習できている．一方で，WideResNet や ResNeXt はそれぞれ 19 時間, 27 時間と 1 日前後かかる．SENet が最も学習に時間がかかり 36 時間程度となっている．

### 4.5　正則化の効果

ResNet をベースとした正則化の手法について，その効果を評価する．ベースとするアーキテクチャは 101 層の ResNet とする．表 8 に各正則化手法によるエラー率を示す．Stochastic Depth は，CIFAR10 では効果が見られないが，CIFAR100 においてはエラー率が低下している．対象とするデータセット内におけるクラスのバリエーションが多い場合に効果があることがわかる．Shake-Shake は，他の正則化手法と比較して大幅にエラー率を低下させており，効果の高い手法であることがわかる．本比較実験では，学習のエポック数を 600 とし，学習率のスケジューリングは 300 エポック時と 450 エポック時に学習率を 0.1 倍している．Shake-Shake の原著では，学習エポック数を 1200，学習率のスケジューリングには cosine を利用している．本比較実験では学習エポック数が少ないのにも関わらず非常に高い精度となっている．Cutout は，ResNet101 に導入することで，エラー率を低下させることができている．CIFAR100 では効果が少ないが，学習エポック数が不十分であることが考えられる．学習時間を比較すると，Shake-Shake が 77.8 時間と他の正則化手法に比べると大幅に時間がかかっている．一方で，Cutout は，画像にマスクをかける処理時間分処理時間が増えているが，13.1 時間で学習できている．学習時間とエラー率から，Cutout は効率的な正則化手法であると言える．

表 9　ImageNet におけるエラー率および学習時間

| アーキテクチャ名 | エラー率 [% ] | | パラメータ数 [M] |
|---|---|---|---|
| | Top1 | Top5 | |
| AlexNet | 49.9 | 26.2 (16.4) | 62 |
| VGG16 | 26.5 | 8.4 (7.3) | 138 |
| SqueezeNet | 44.2 | 21.3 | 0.7 |
| Inception-v3 | 22.8 | 6.6 | 22.5 |
| ResNet-18 | 30.4 | 10.7 | 0.2 |
| ResNet-34 | 26.7 | 8.6 | 0.5 |
| ResNet-152 | 21.9 | 6.1 (3.5) | 1.7 |
| DenseNet-201 | 23.5 | 6.6 | 15.0 |

### 4.6　ImageNet における精度比較

ILSVRC を通じて優れたアーキテクチャや手法が提案されている．一般に公開されているテスト結果は，複数のネットワークモデルの利用や評価画像の切り出し，反転などの data augmentation を行い，それらによる結果を統合する精度向上のテクニックを用いている．一方で，ベンチマーク結果からは，アーキテクチャによる純粋な精度を比較するのが難しい．本節では，単一アーキテクチャによる精度を比較する．表 9 に Top1 エラー率と Top5 エラー率を示す．ここで，各エラー率は Val データに対する精度であり，TOP5 の括弧内の数値は ILSVRC における test データに対する精度である．test データと val データは，データ自体が異なるが，ほとんどの文献において，ほぼ同等の精度になっている．AlexNet は，ILSVRC2012 のテストデータに対して，16.4%の Top5 エラー率であるが，単一アーキテクチャの場合は 26.2%となっており，精度向上のテクニックが有効性であることがわかる．VGG は，単一アーキテクチャの場合でも 8.4%の Top5 エラー率となっており，テストデータに対する精度 (7.3%) と大きな差はない．VGG は，単一アーキテクチャでも十分な精度を達成できている．

GoogLeNet の Inception モジュールを改良した Inceptiion-v3 は 6.6%，ISLVRC の 2015 年に優勝した ResNet（152 層）は 6.1%，DenseNet（201 層）は 6.6%と単一アーキテクチャでほぼ同等の精度となっている．これより，ネットワークを深くすることと幅を持たせることが単一アーキテクチャの精度向上に必要であることがわかる．

## 5. ま と め

本稿では，ニューラルネットワークの中でも特に発展の著しい畳み込みニューラルネットワーク (CNN) について，画像認識コンペティション ILSVRC にて優秀な成績を収めたモデルを概観することで，CNN の変遷を振り返った．また，近年提案されている様々な CNN の改良手法についてサーベイを行い，各手法のアプローチから分類を行い，それぞれ解説を行った．代表的なモデルについて複数のデータセットを用いて学習および網羅的な精度評価も行い，各モデルの精度および学習時間の傾向について議論を行った．

ここでの重要な気づきは，モデルアーキテクチャを提案している手法の殆どが，$1 \times 1$ 畳み込みによる次元削減と，sparse な畳み込みにによりパラメータ削減を行い，その分ネットワークの深さや幅を増加させることで精度向上を実現している点である．また，ネットワークおよびデータに対する強力な正則化手法が大きな精度改善を実現していた点にも注目したい．本稿が，様々なタスクを解く上でのモデルの選定や，新たなモデルアーキテクチャを検討する上での参考になれば幸いである．

## 文 献


[1] Anonymous. Shakedrop regularization. *ICLR Under Review*, 2018.

[2] M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *Proc. of ICML*, 2017.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI*, 32(12), 2017.

[4] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proc. of CVPR*, 2017.

[5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *Proc. of ICLR*, 2015.

[6] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proc. of CVPR*, 2017.

[7] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *Proc. of ICLR*, 2016.

[8] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv:1708.04552*, 2017.

[9] X. Dong, G. Kang, K. Zhan, and Y. Yang. Eraserelu: A simple way to ease the training of deep convolution neural networks. *arXiv:1709.07634*, 2017.

[10] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469, 1982.

[11] Gastaldi. Shake-shake regularization of 3-branch residual networks. In *Proc. of ICLR Workshop*, 2017.

[12] X. Gastaldi. Shake-shake regularization. *arXiv:1705.07485v2*, 2017.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of AISTATS*, 2010.

[14] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang, J. Cai, and T. Chen. Recent advances in convolutional neural networks. *arXiv:1512.07108v6*, 2017.

[15] D. Han, J. Kim, and J. Kim. Deep pyramidal residual networks. In *Proc. of CVPR*, 2017.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. of ICCV*, 2015.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of CVPR*, 2016.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Proc. of ECCV*, 2016.

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

[20] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv:1709.01507*, 2017.

[21] G. Huang, D. Chen, T. Li, F. Wu, L. Maaten, and K. Q. Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv:1703.09844*, 2017.

[22] G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten. Densely connected convolutional networks. In *Proc. of CVPR*, 2017.

[23] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. In *Proc. of ECCV*, 2016.

[24] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

[25] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016.

[26] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. of ICML*, 2015.

[27] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proc. of ICCV*, 2009.

[28] Y. Kim. Convolutional neural networks for sentence classification. In *Proc. of EMNLP*, 2014.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of NIPS*, 2012.

[30] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *Proc. of ICLR*, 2017.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] M. Lin, Q. Chen, and S. Yan. Network in network. In *Proc. of ICLR*, 2014.

[33] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv:1711.00436*, 2017.

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Proc. of ECCV*, 2016.

[35] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. of CVPR*, 2015.

[36] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[37] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. of ICML*, 2010.

[38] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens. Adding gradient noise improves learning for very deep networks. In *Proc. of ICLR Workshop*, 2016.

[39] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *Proc. of ECCV*, 2016.

[40] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv:1710.05941*, 2017.

[41] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *Proc. of CVPR*, 2017.

[42] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proc. of NIPS*, 2015.

[43] J. Sanchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proc. of CVPR*, 2011.

[44] L. Sifre and S. Mallat. Rigid-motion scattering for texture classification. *arXiv:1403.1687*, 2014.

[45] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of ICLR*, 2015.

[47] S. Singh, D. Hoiem, and D. Forsyth. Swapout: Learning an ensemble of deep architectures. In *Proc. of NIPS*, 2016.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15:1929–1958, 2014.

[49] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. In *Proc. of ICML Workshop*, 2015.

[50] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Proc. of NIPS*, 2015.

[51] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proc. of AAAI*, 2017.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. of CVPR*, 2015.

[53] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens. Rethinking the inception architecture for computer vision. In *Proc. of CVPR*, 2016.

[54] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv:1609.03499*, 2016.

[55] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, , G. van den Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *https://deepmind.com/documents/131/Distilling_WaveNet.pdf*, 2017.

[56] A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Proc. of NIPS*, 2016.

[57] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang. Residual attention network for image classification. In *Proc. of CVPR*, 2017.

[58] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *Proc. of CVPR*, 2016.

[59] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proc. of CVPR*, 2017.

[60] S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proc. of BMVC*, 2016.

[61] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proc. of ECCV*, 2014.

[62] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv:1710.09412*, 2017.

[63] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu. Residual networks of residual networks: Multilevel residual networks. *TCSVT*, 2017.

[64] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Proc. of NIPS*, 2015.

[65] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. *arXiv:1708.04896*, 2017.

[66] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.

[67] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv:1707.07012*, 2017.