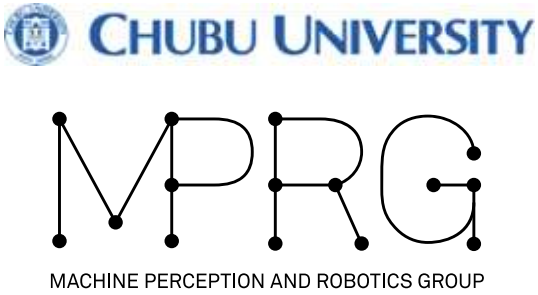
畳み込みニューラルネットワークの研究動向

株式会社ディー・エヌ・エー
内田 祐介

中部大学
山下 隆義



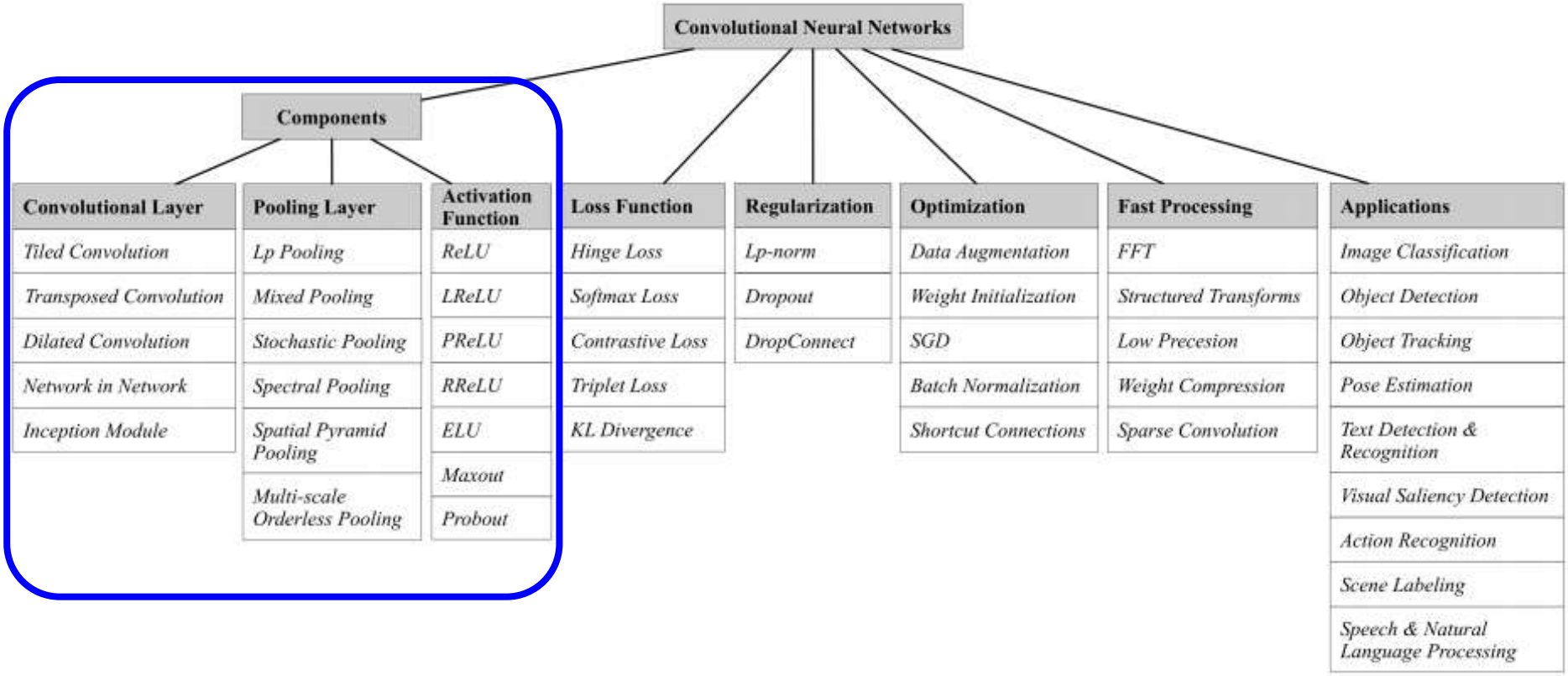
本日の発表の内容

- 主にアーキテクチャの観点から
  - ILSVRC歴代の優勝モデルから
    畳み込みニューラルネットワークの歴史を振り返る
  - 近年の畳み込みニューラルネットワークを
    アプローチをもとに分類、解説
  - 代表的なネットワークを実際に訓練・評価

- Outline
  - ILSVRCで振り返るCNNの進化
  - 最新のCNN改良手法
  - 各アーキテクチャの比較

本日の発表の内容

- 畳み込みニューラルネットワークにまつわる項目のうち**各構成要素**をどうモデルに組み上げるかにフォーカス
  - 細かい構成要素に関する説明は最小限



J. Gu et al., "Recent Advances in Convolutional Neural Networks," in arXiv:1512.07108, 2017.

ILSVRCで振り返るCNNの進化

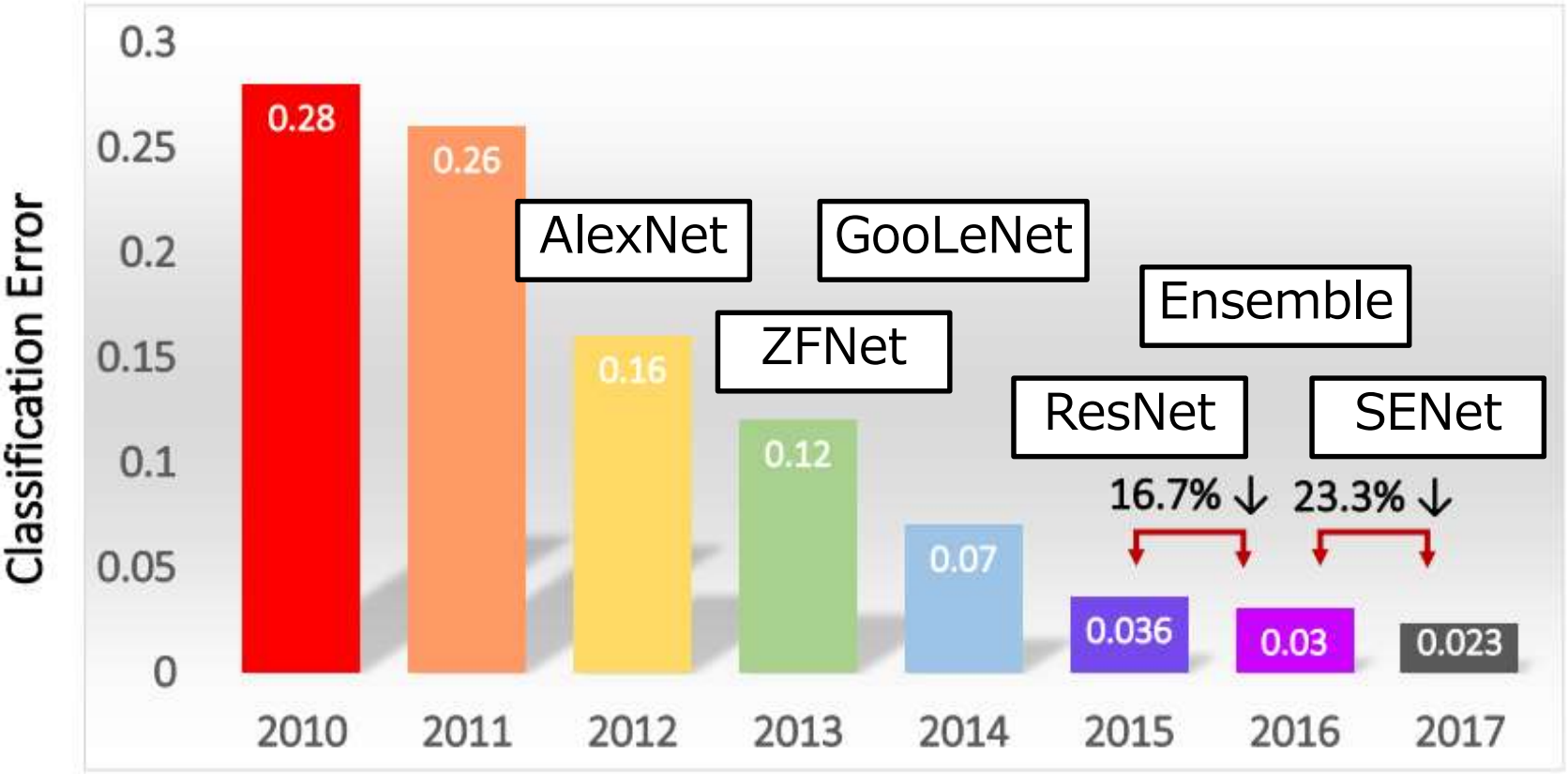
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- ImageNetデータセットを利用したコンペ
  - WordNetをもとにクラスを定義
  - 各クラスのデータを大量に収集＆アノテーション
  - 学習データ120万枚、テストデータ10万枚
- クラス分類、物体検出、位置特定等のタスクが存在
  - 特にクラス分類の精度の推移が画像認識の進化の指標として参考にされることが多い

- 2012年に深層学習ベースの手法が優勝して以降、（畳み込み）ニューラルネットワークの天下一武道会化
  - ここでの優勝モデルがデファクトスタンダードとして利用される
  - セグメンテーション等、他のタスクでも利用される

ILSVRC

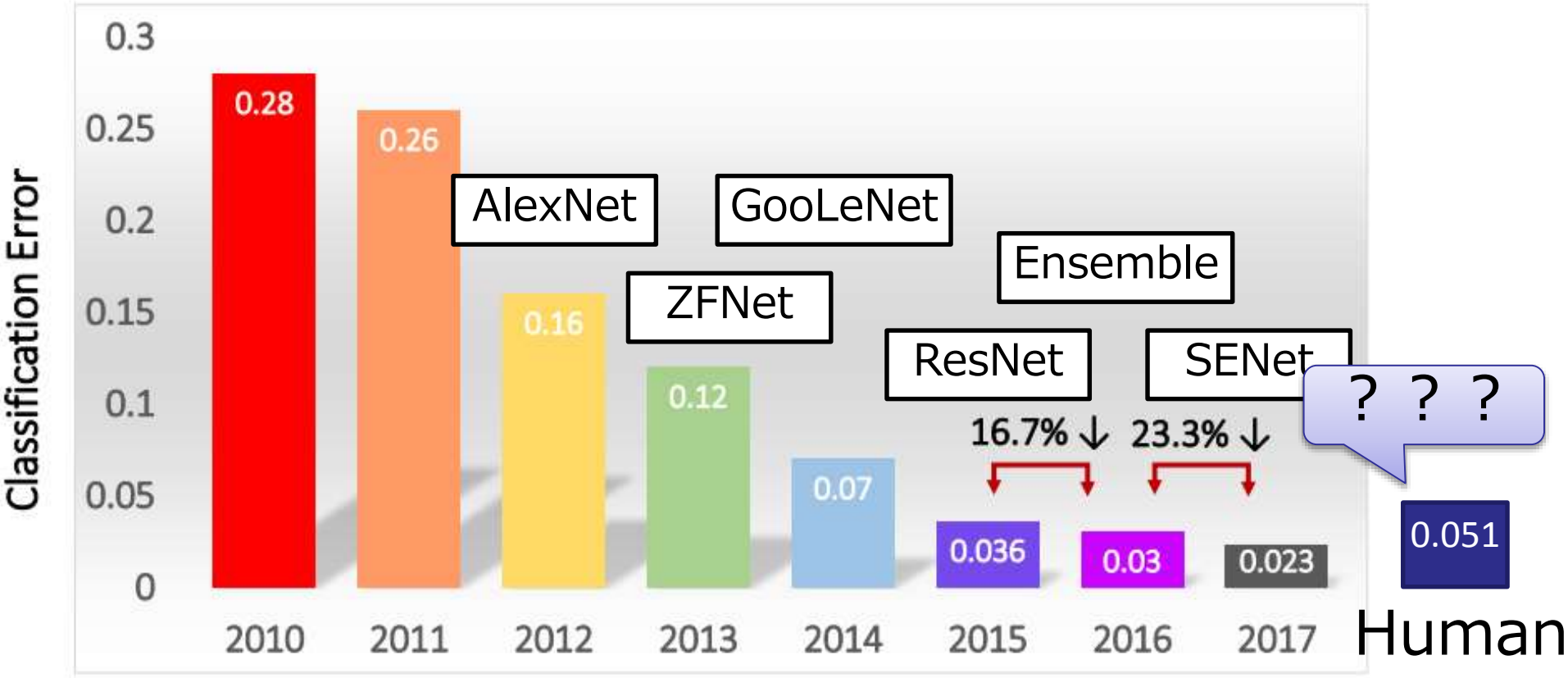
- クラス分類タスクのエラー率（top5 error）の推移



http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

ILSVRC

- クラス分類タスクのエラー率（top5 error）の推移



http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

人間のエラー率5.1%？

- そもそも人間のエラー率は0%なのでは？
  - No, ImageNetがどうやって作成されたかに関係

- ImageNetデータセットの作成手順
  - WordNetを元に1000クラスを定義
  - 検索エンジンで各クラスの画像を収集
  - クラウドワーカーが各画像とクラスが**対応しているか否かのバイナリ判定**

- ImageNetのタスク
  - 画像が与えられ、**1000クラスから1つのクラスを選択**

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

人間のエラー率5.1%？

- *“1 smart, trained and highly education human being performed at that error rate”*
  —— https://amundtveit.com/tag/imagenet/

- Andrej Karpathyさん (Stanford→OpenAI→Tesla)

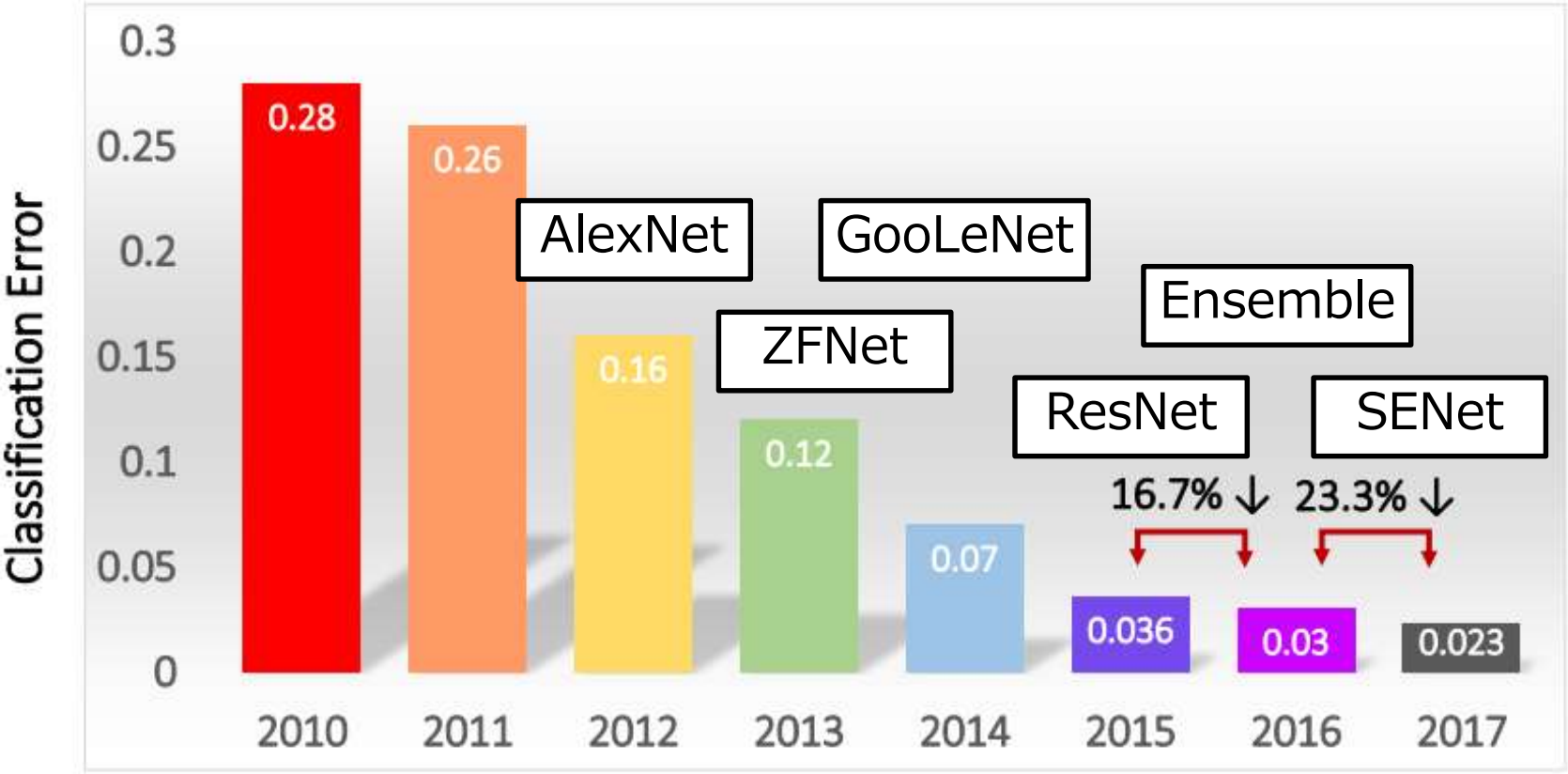
https://github.com/karpathy

Karpathyさんのエラー率5.1までの道のり

- 2014年のGoogLeNetの達成精度 (top5 error 6.7%) から人間との比較をすべきと考えた
- 専用のアノテーションツールを作成
  - ILSVRCに従事しているラボメンでも
    1000クラスから5クラス選択することは非常に難しい
- GoogLeNetの結果を利用して、候補を100クラスに限定
  - それでも難しくラボの熟練ラベラーでも13～15%のエラー率
- 自分が訓練データを用い、”苦痛を伴う長時間の学習”を経て、じっくりアノテーションするのが効率的と気づいた
  - train on 500 validation images -> 1500 test images
  - テストエラー5.1%を達成！
    - 犬の品種に詳しくなった！
- 上記のように、エラー率5.1%は天才がかなり頑張った結果
  - とはいえCNNの学習時間に比べるとまだ足りないという考え方も

ILSVRC

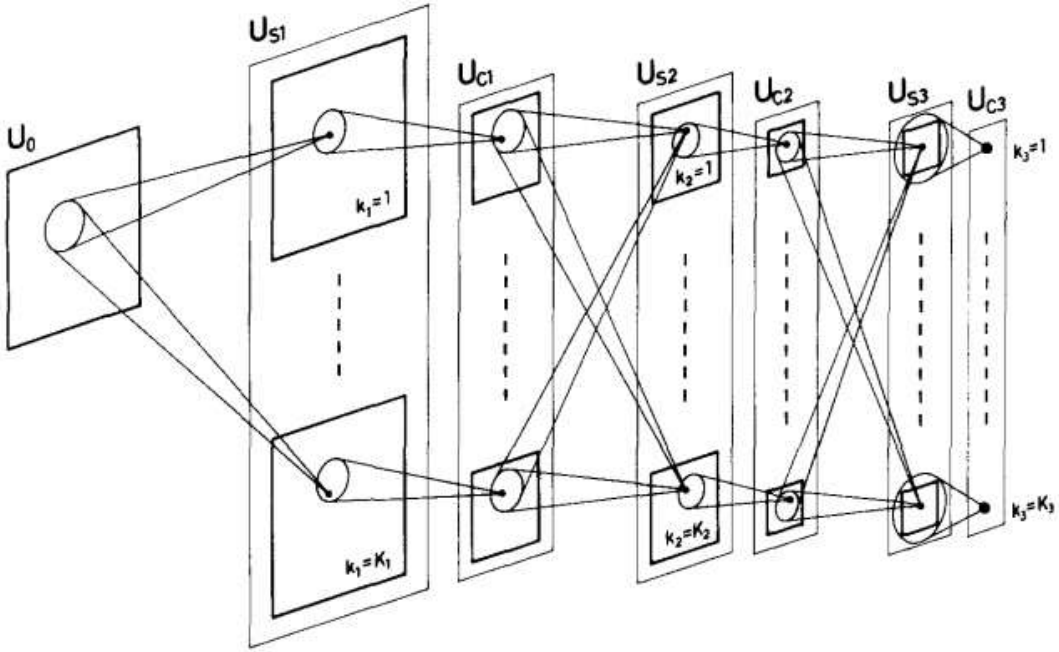
- クラス分類タスクのエラー率（top5 error）の推移



http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

Before AlexNet

- Neocognitron
  - 脳の視覚野に関する知見を元に考案
    - 単純型細胞＝特徴抽出を行う畳み込み層
    - 複雑型細胞：位置ずれを吸収するプーリング層
  - 自己組織化による特徴抽出



K. Fukushima and S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," in Pattern Recognition, 1982.

Before AlexNet

- Neocognitronの活性化関数は実は**ReLU**

$$u_{Sl}(k_l, \mathbf{n}) = r_l \cdot \varphi\left[\frac{1 + \sum_{k_{l-1}=1}^{K_{l-1}} \sum_{\mathbf{v} \in S_l} a_l(k_{l-1}, \mathbf{v}, k_l) \cdot u_{Cl-1}(k_{l-1}, \mathbf{n}+\mathbf{v})}{1 + \frac{2r_l}{1+r_l} \cdot b_l(k_l) \cdot v_{Cl-1}(\mathbf{n})} - 1\right],$$

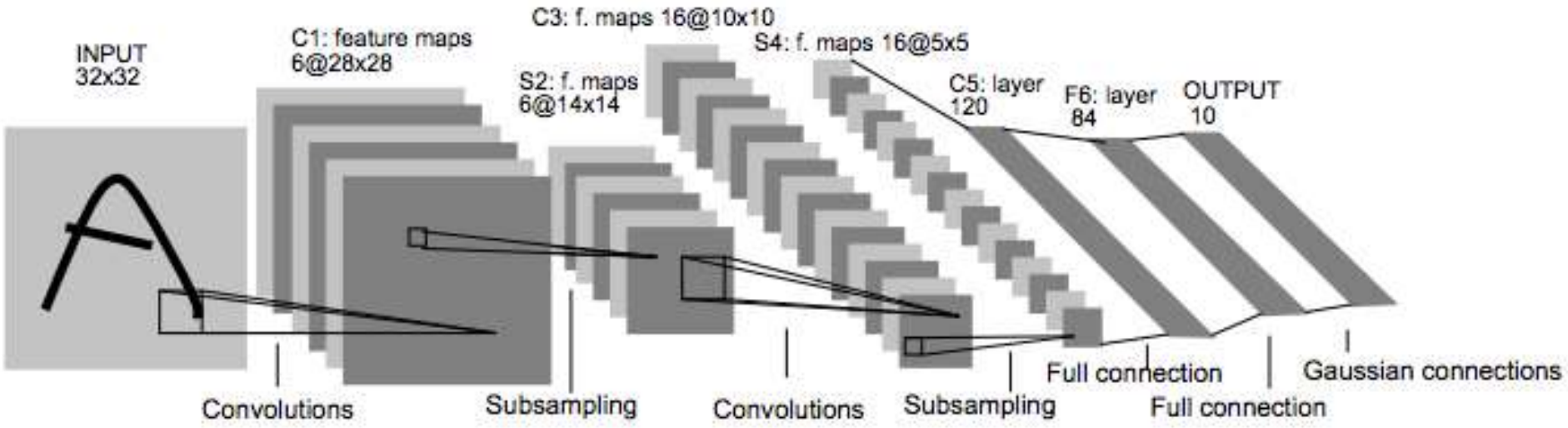
where

$$\varphi[x] = \begin{cases} x & x \geqq 0 \\ 0 & x < 0. \end{cases} \qquad (2)$$

K. Fukushima and S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," in Pattern Recognition, 1982.

Before AlexNet

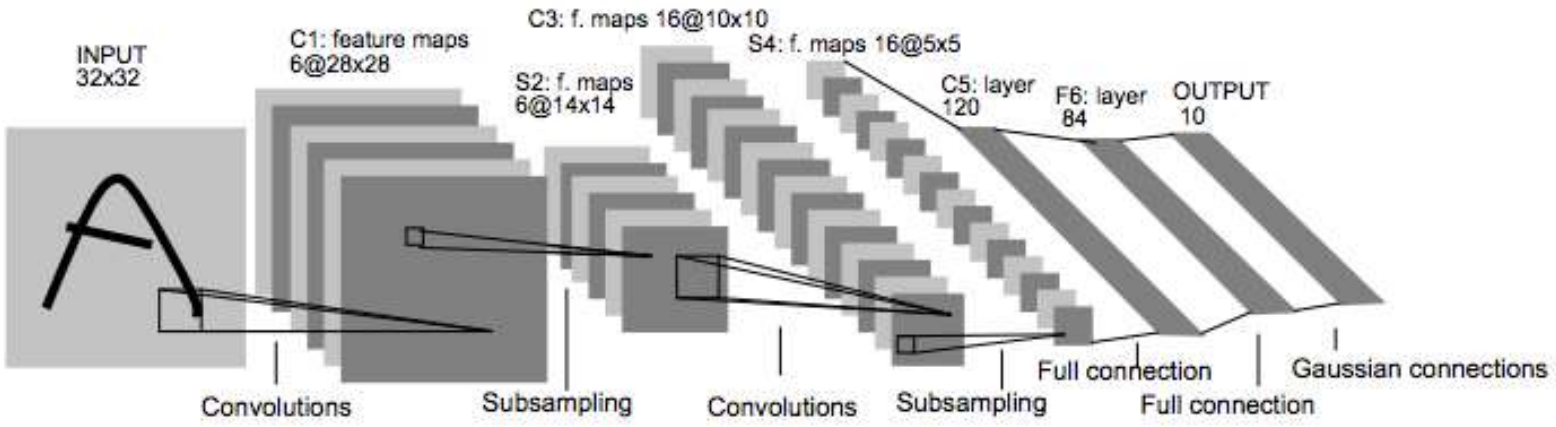
- LeNet
  - 逆誤差伝搬法による学習
  - 文字認識で成功をおさめる
  - 全体的なアーキテクチャは既に完成



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, 1998.

Before AlexNet

- 巷のLeNetはLeNetではない



出力チャネル

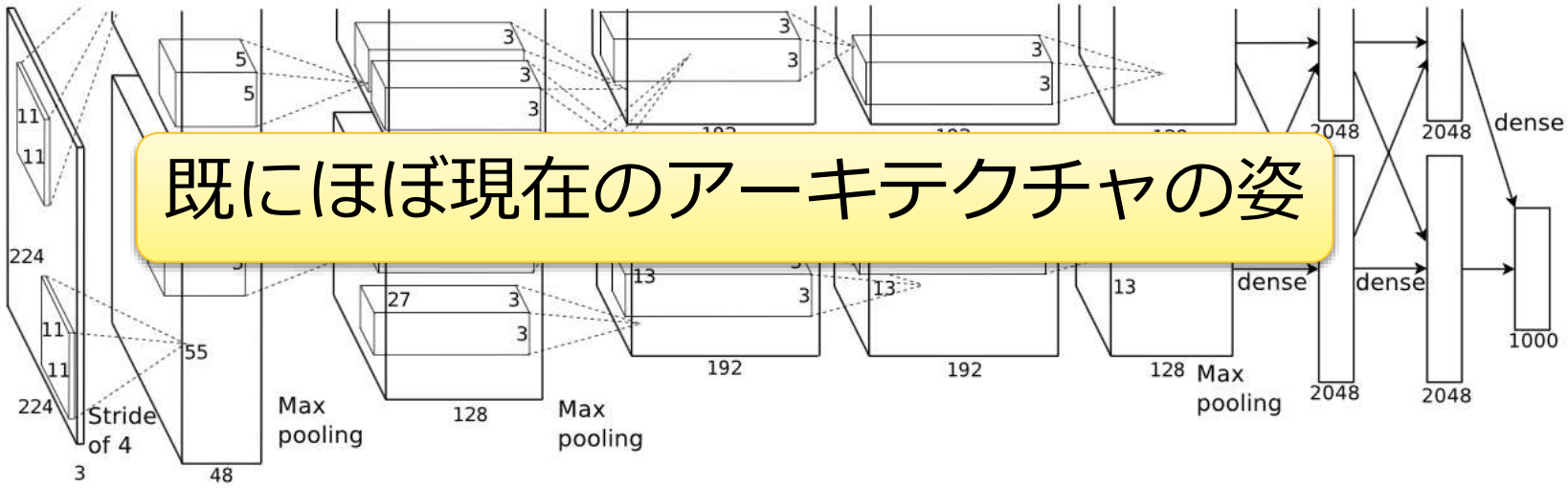
入力チャネル	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

Sparseな畳み込み！

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, 1998.

AlexNet

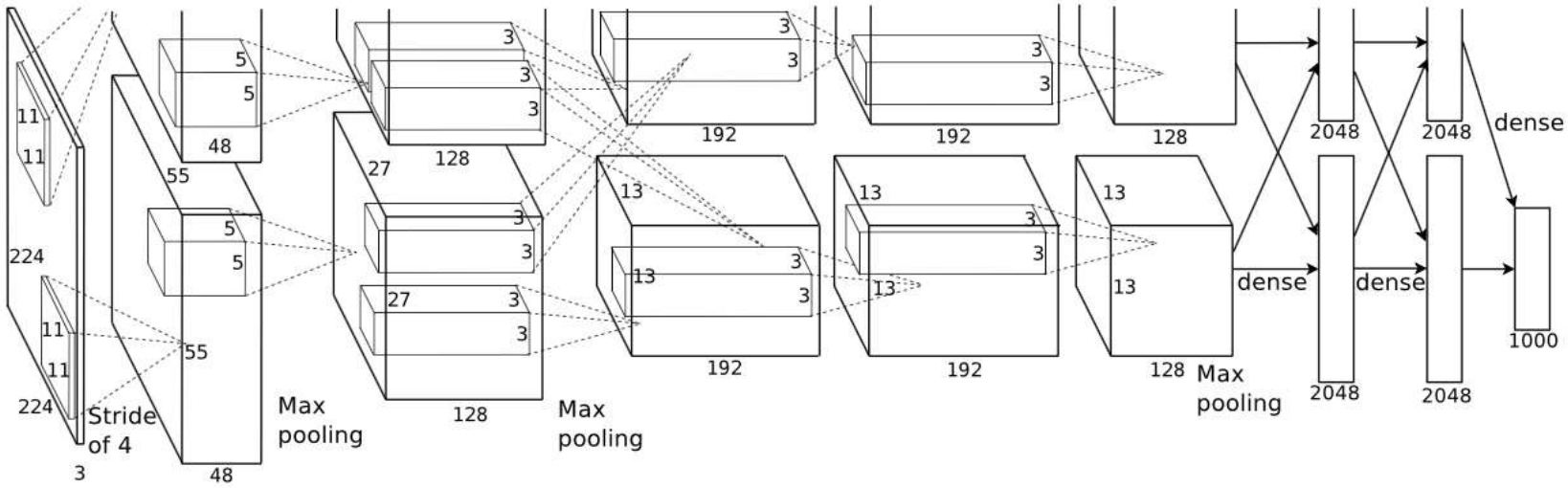
- 2012年のILSVRC優勝モデル
  - Rectified Linear Units (ReLU)
  - Local Response Normalization (LRN) （最近は使われない）
  - Overlapping pooling (3x3 max pool, stride =2)
  - Dropout（全結合層）
  - SGD + momentum, weight decay, learning rate decay



A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proc. of NIPS, 2012.

AlexNet

- 巷のAlexNetは（ｒｙ
  - いくつかの畳み込み層は、チャネル方向に分割され2つのGPUでそれぞれ独立に畳み込みが行われる「grouped convolution」となっている



A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proc. of NIPS, 2012.

ZFNet

- 2013年のILSVRC優勝モデル
  - CNNの可視化を行い、AlexNetの2つの問題を明らかにし、改良
    - 1つ目の畳み込みフィルタのカーネルサイズが大きく
      高周波と低周波の情報を取得するフィルタばかりになっている
    - Stride = 4により2層目の特徴マップにエイリアシングが発生

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

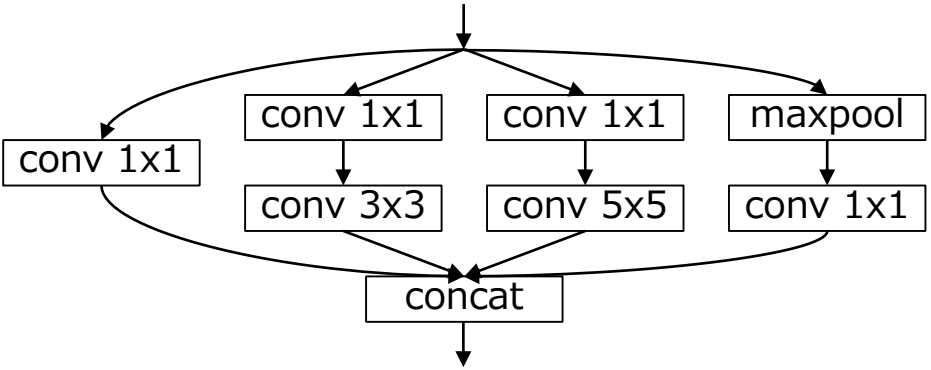
Layer	Filter size	Stride	Output size
input			(224, 224, 3)
conv	(7, 7)	(2, 2)	(110, 110, 96)
maxpool	(3, 3)	(2, 2)	(55, 55, 96)
conv	(5, 5)	(2, 2)	(26, 26, 256)
maxpool	(3, 3)	(2, 2)	(13, 13, 256)
conv	(3, 3)	(1, 1)	(13, 13, 384)
conv	(3, 3)	(1, 1)	(13, 13, 384)
conv	(3, 3)	(1, 1)	(13, 13, 256)
maxpool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

GoogLeNet

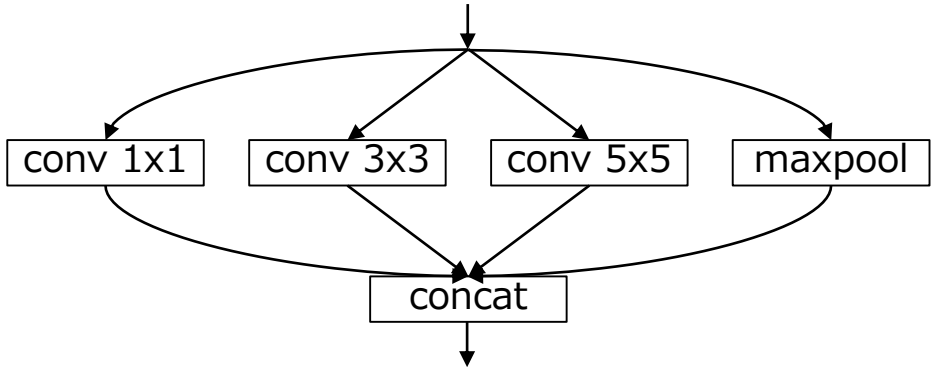
- 2014年のILSVRC優勝モデル
  - Inceptionモジュール
  - Global Average Pooling (GAP)
  - Auxiliary loss
    - ネットワークを途中で分岐させ、そこで分類を行うロスを追加
      →学習の効率化＋正則化

GoogLeNet

- Inceptionモジュール
  - **小さなネットワークを1つのモジュールとして定義**し
    モジュールの積み重ねでネットワークを構成
    - 3x3と複数の1x1畳み込みを1つのモジュールとする
      Network In Network (NIN) に起源
  - **Sparseな畳み込み**により、
    表現能力とパラメータ数のトレードオフを改善
  - 1x1畳み込みによる**次元削減**



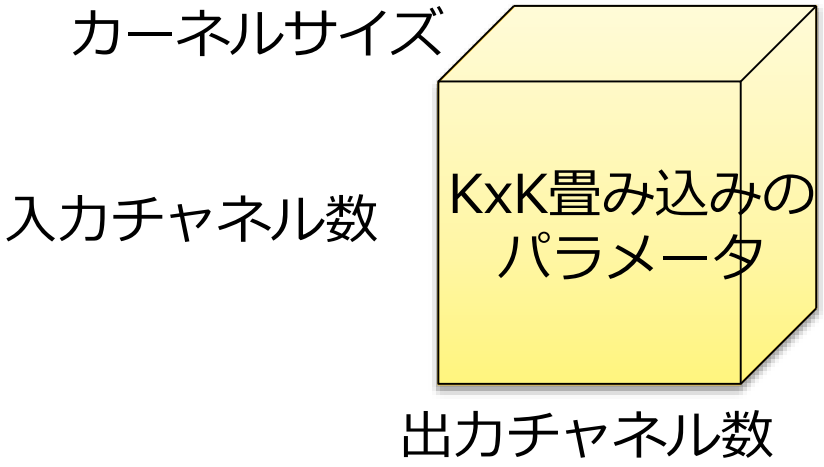
Inceptionモジュール（次元削減あり）



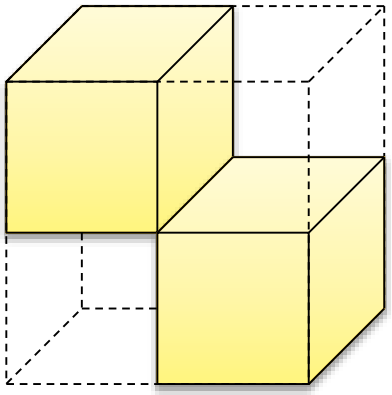
Inceptionモジュール（naive）

Sparseな畳み込み？

- 通常の畳み込みのパラメータ数＝$N \times M \times K^2$
  - N: 入力チャネル数、M: 出力チャネル数、K^2: カーネルサイズ



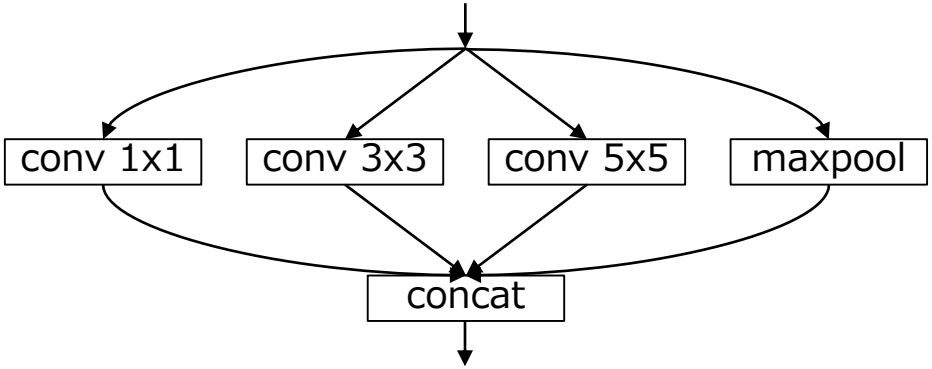
- （例）グループ化畳み込みのパラメータ数＝$N \times M \times K^2 / G$
  - G: グループ数



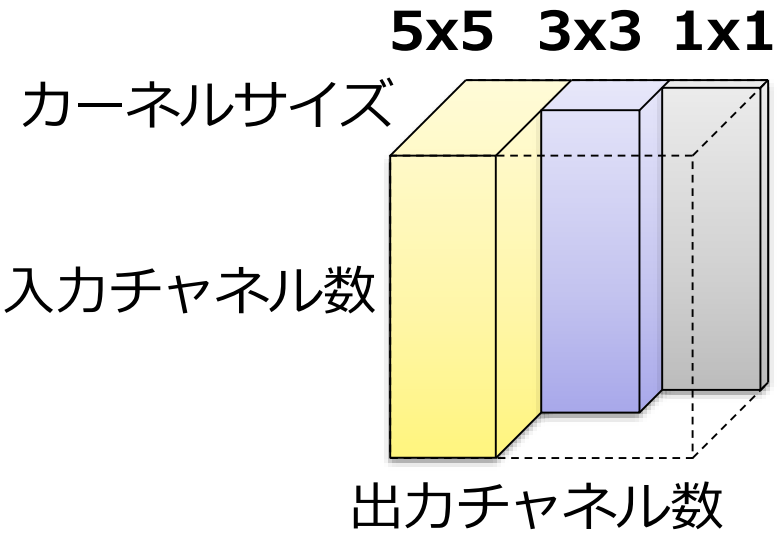
グループ数2の例

※簡単のためbiasは無視

Sparseな畳み込みとしてのInceptionモジュール



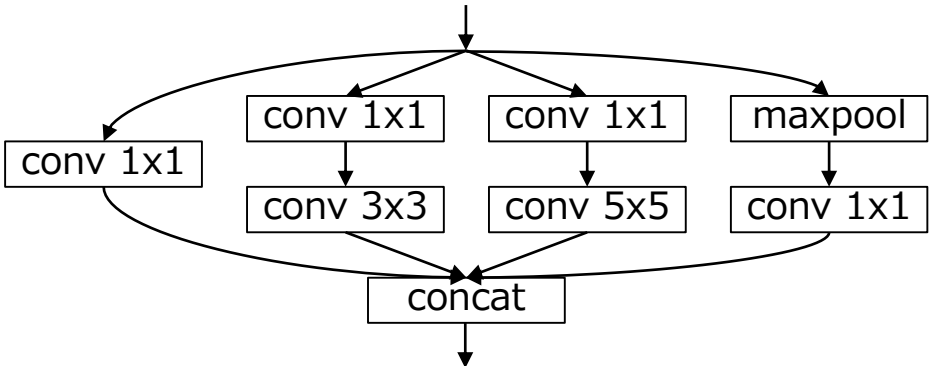
Inceptionモジュール（naive）



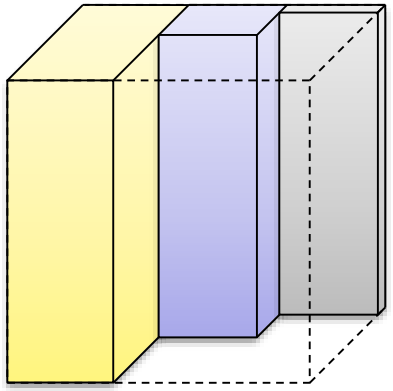
5x5の畳み込みを
少数パラメータで近似

※簡単のためmax poolを無視

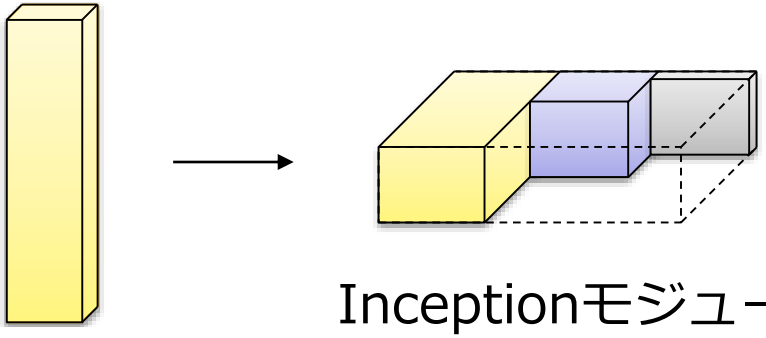
Sparseな畳み込みとしてのInceptionモジュール



Inceptionモジュール（次元削減あり）



naïveバージョン



1x1畳み込み

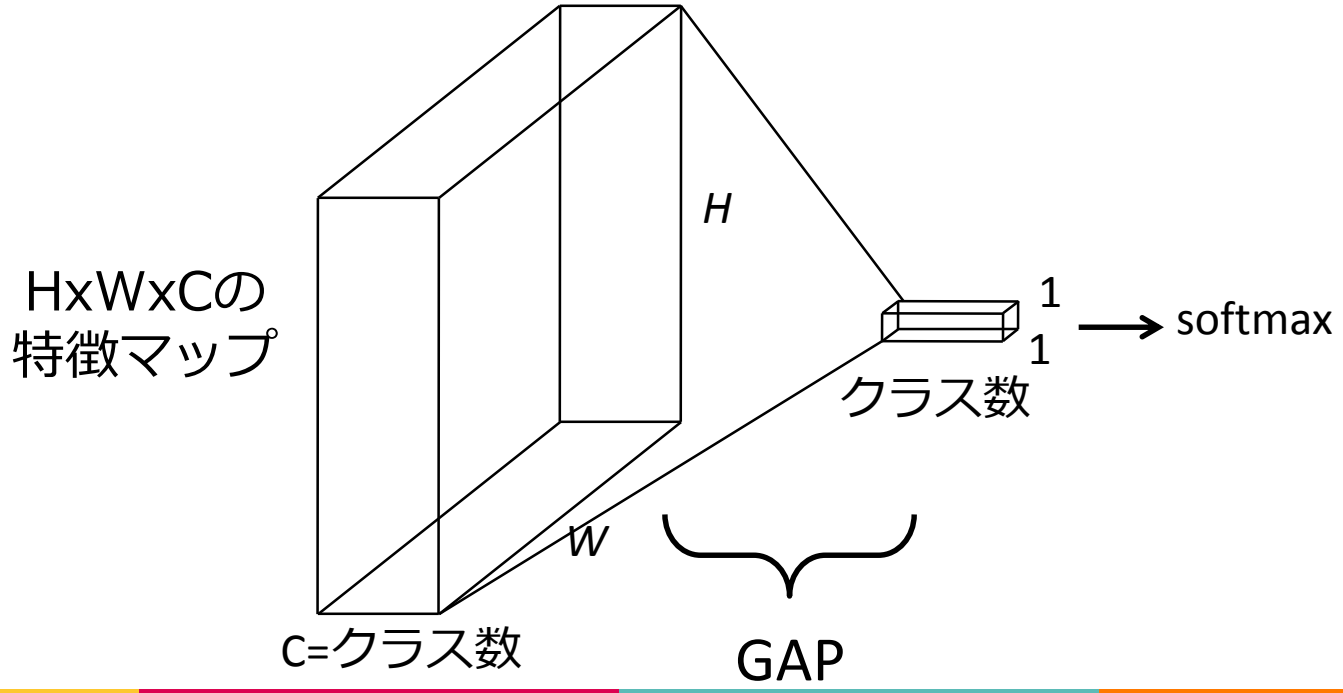
Inceptionモジュール

1x1畳み込みの次元削減で
更にパラメータを削減

※簡単のためmax poolを無視

GoogLeNet

- Global Average Pooling (GAP)
  - 特徴マップのサイズと同じaverage pooling
  - NINで最初に利用された
    - NINではチャネル数をクラス数に調整してからGAP
    - GoogLeNet以降では、GAPの後に全結合層を付ける
  - 全結合層を減らすことでパラメータ数を削減



Inception-v?

- Inception-v2,3
  - Batch normalization
  - 5x5 -> 3x3 (x2)
  - nxn -> nx1 + 1xn
- Inception-v4
  - 3種類のInceptionを使い分け
- Inception-ResNet-v1,2
  - 後述のショートカット機構の導入

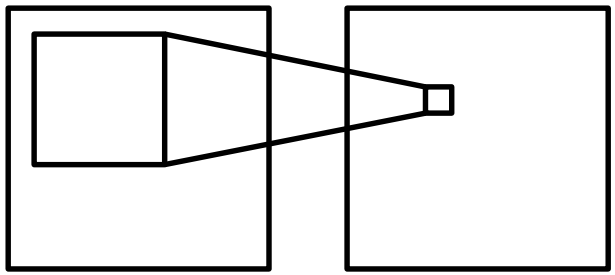
Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7 × 7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

= Inception-v3

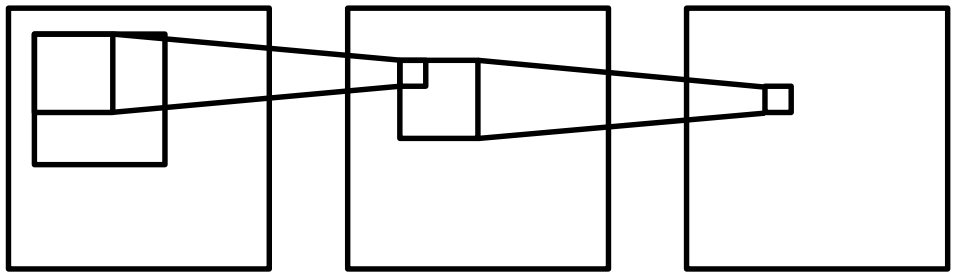
C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, "Rethinking the inception architecture for computer vision," in Proc. of CVPR, 2016.
C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in Proc. of AAAI, 2017..

VGGNet

- 2014年のILSVRCで2位となったモデル
- シンプルなモデルの設計方針
  - 3x3畳み込み層（＋全結合層）のみを利用
    - 5x5畳み込みと3x3畳み込み×2は同じreceptive fieldを持つ
    - 計算量は 25:18



conv 5x5



conv 3x3 - conv 3x3

  - 同一出力チャネルの畳み込み層を重ねた後にプーリングにより特徴マップを半分にしつつチャネル数を倍増
- まずは浅いネットワークを学習し、畳み込み層を追加してfine-tuningすることで深いネットワークを学習
  - Xavierの初期化を利用することで事前学習は不要に

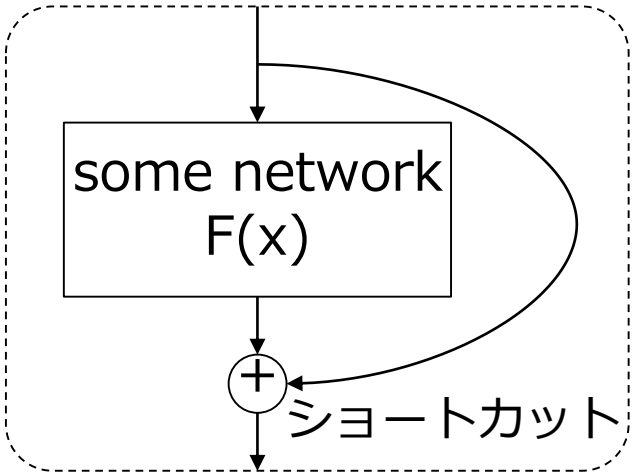
Residual Networks (ResNet)

- 2015年のILSVRC優勝モデル
- Residualモジュール（ショートカット機構）の導入→後述
- Batch normalizationの導入
  - ネットワーク内の共変量シフトを軽減
    - 学習バッチ毎に、自分より前の層のパラメータ更新により、入力される特徴マップの分布が大きく変わる
- Global Average Pooling (GAP) の利用
  - 全結合層の削減によるパラメータ数削減
- Heの初期化の利用
  - ReLUを考慮したパラメータの初期化
- ショートカットを展開していくと複数の深さの違うネットワークのアンサンブルとなっている [1]

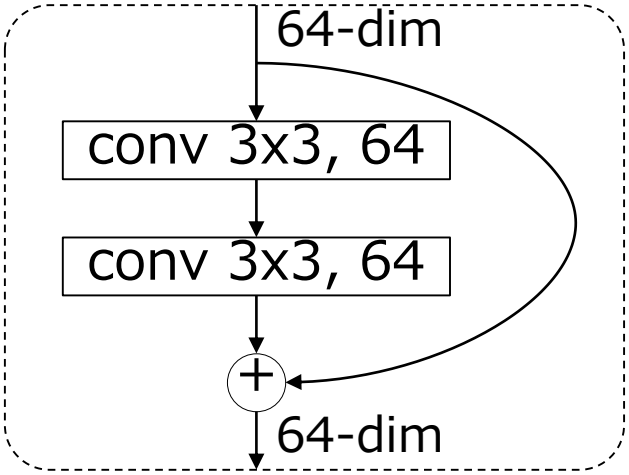
[1] A. Veit, M. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in Proc. of NIPS, 2016.

Residual Networks (ResNet)

- Residualモジュール
  - ある処理の出力 F(x) を次の層に渡すのではなく
    入力xをショートカットし、 F(x) + x を次の層に渡す処理単位



抽象的なresidualモジュール

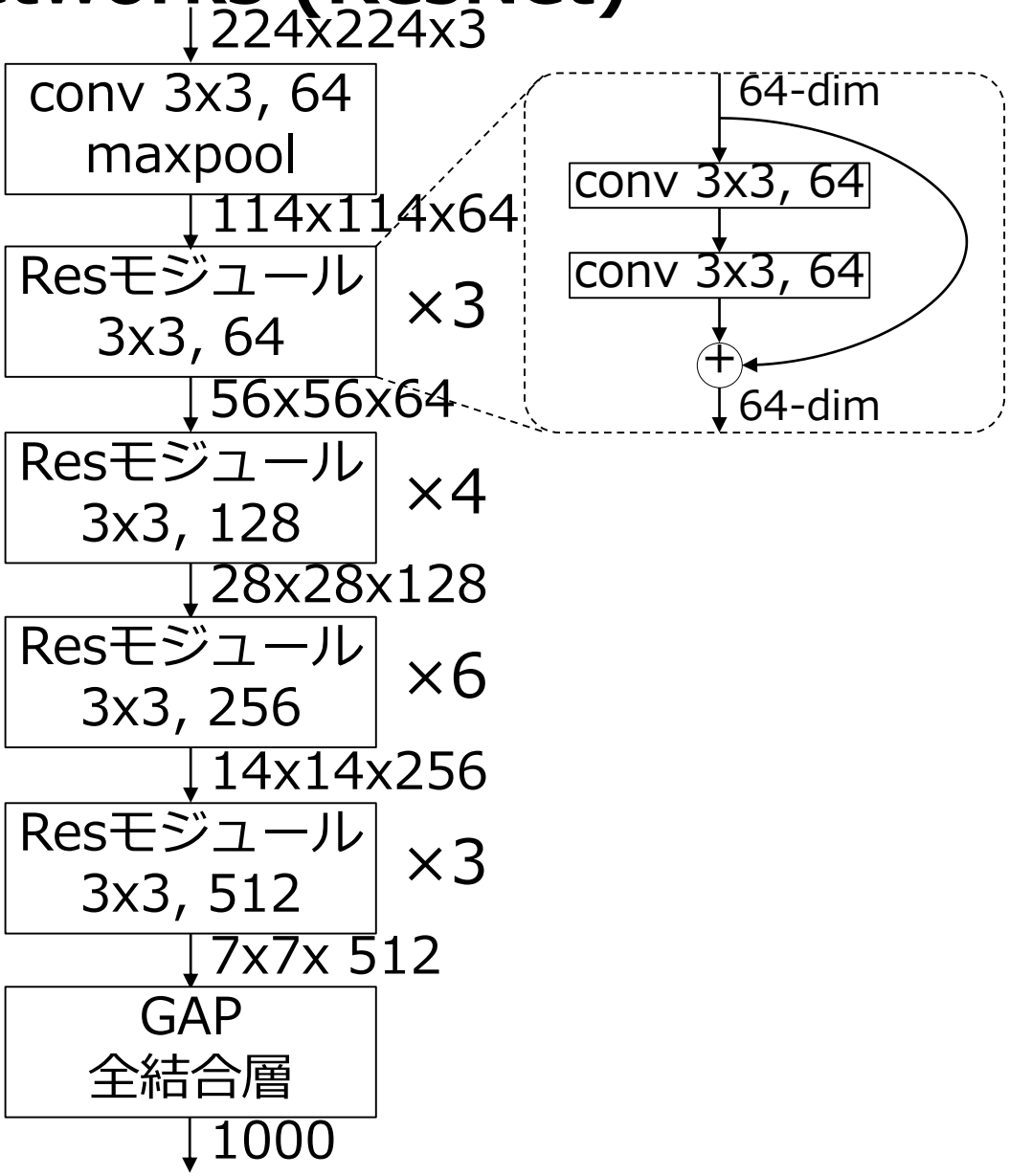


具体例

- Backpropagation時にショートカットを通じて
  直接勾配が下層に伝わることで深いネットワークでも
  効率的な学習が可能

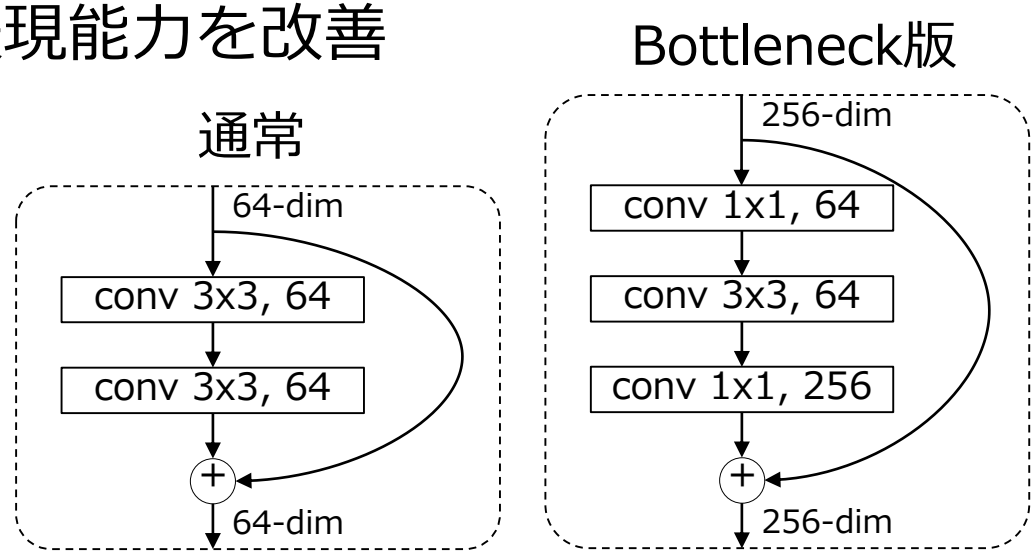
Residual Networks (ResNet)

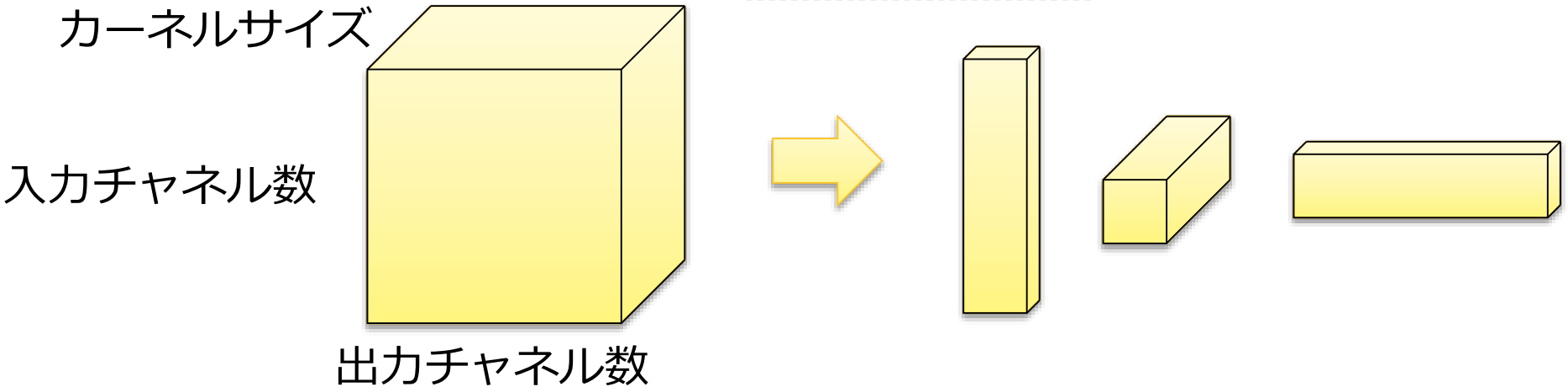
- 全体の構造
  - Residualモジュールのスタック
  - ダウンサンプルする際にチャネル数を2倍に（VGGと同じ）
  - チャネル数が違うショートカットはzero-padingかconv 1x1で調整



Bottleneckバージョン

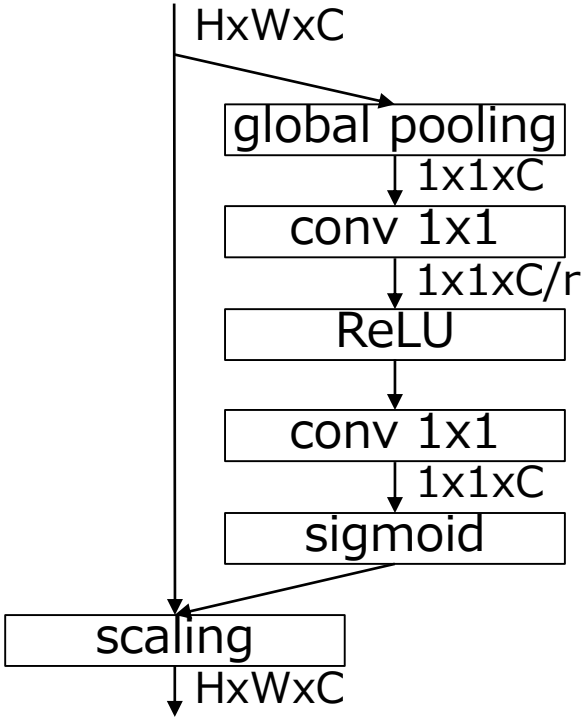
- 1x1畳み込みで次元削減、その後3x3畳み込みを行った後、更に1x1畳み込みで次元を復元
- 同等のパラメータ数で表現能力を改善
  - 深くもなる
- パラメータ数↓



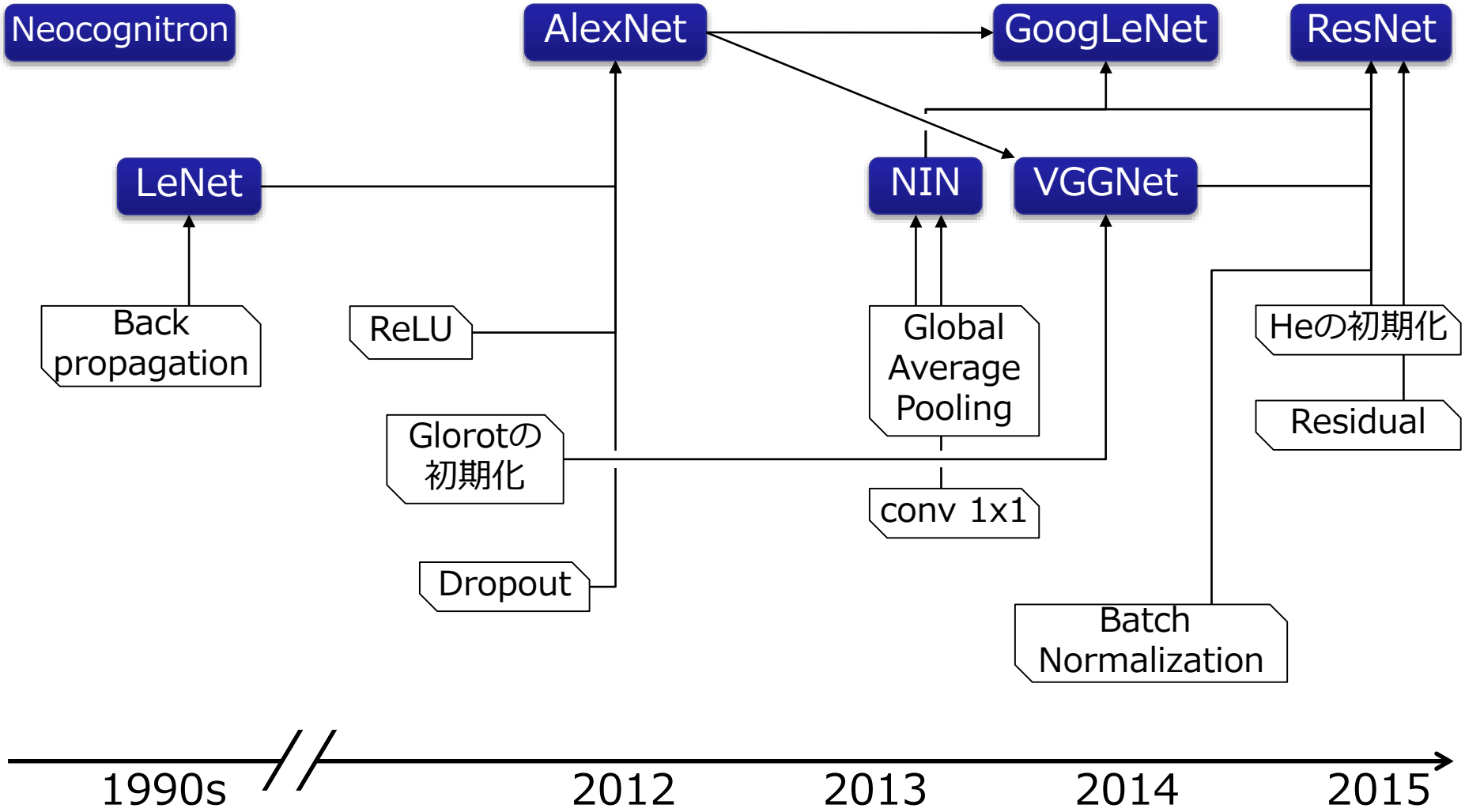


Squeeze-and-Excitation Networks (SENet)

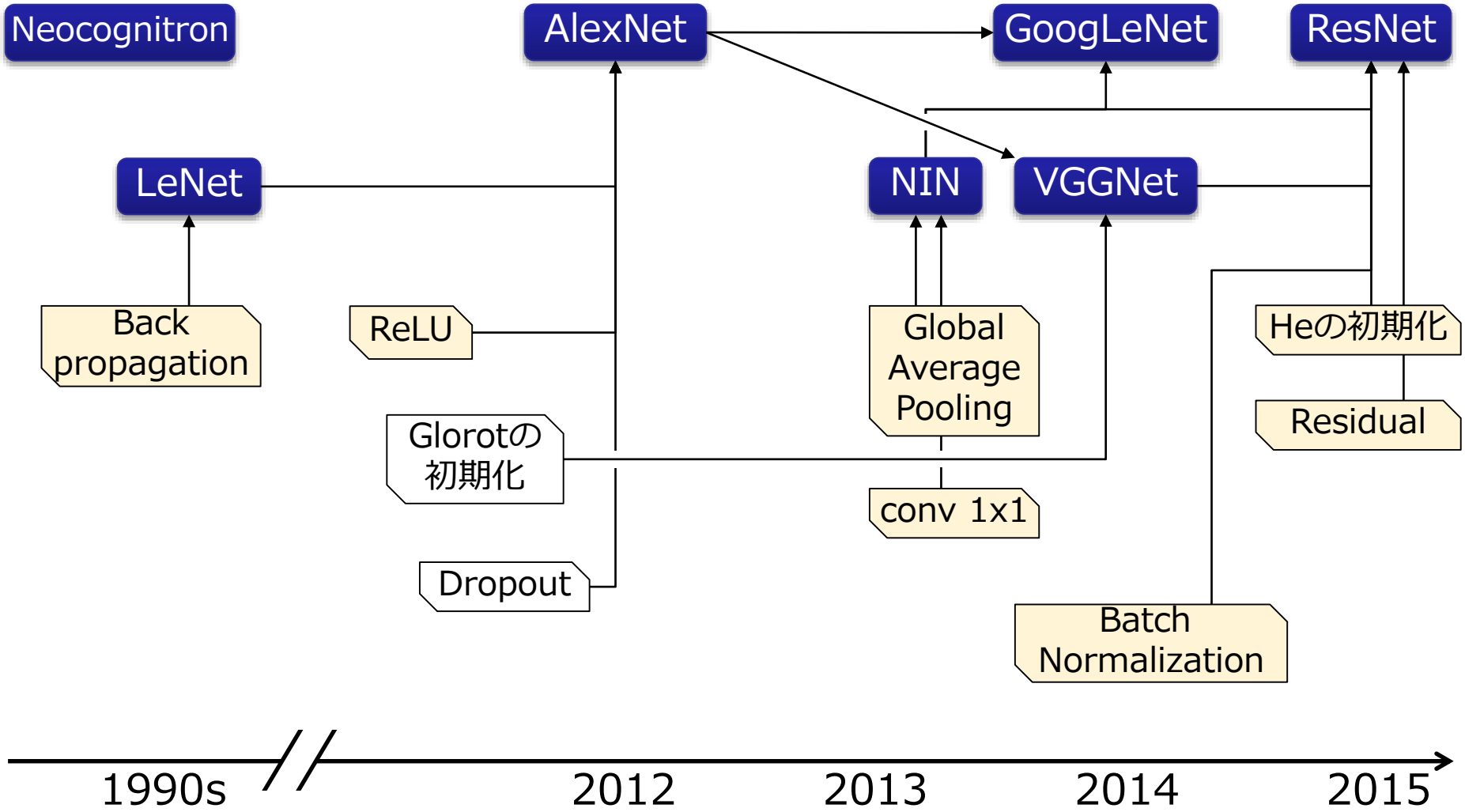
- 2017年のILSVRC優勝モデル
- チャネル毎の特徴マップを適応的に重み付けする SE Blockを導入
  - Globalなコンテキストを取得するsqueeze step
  - チャネル間の依存関係を抽出するexcitation step
- どんなネットワークにも導入可能
  - ResNet, ResNeXt, Inception, Inception-ResNet-v2等に導入し有効性を確認



まとめ


Neocognitron
AlexNet
GoogLeNet
ResNet
LeNet
NIN
VGGNet
Back propagation
ReLU
Global Average Pooling
Heの初期化
Residual
Glorotの初期化
conv 1x1
Dropout
Batch Normalization
1990s
2012
2013
2014
2015

まとめ


Neocognitron
AlexNet
GoogLeNet
ResNet
LeNet
NIN
VGGNet
Back propagation
ReLU
Global Average Pooling
Heの初期化
Residual
Glorotの初期化
conv 1x1
Dropout
Batch Normalization
1990s
2012
2013
2014
2015

最新のCNN改良手法

最新のCNN改良手法

- Residualモジュールの改良
- 独自モジュールの利用
- 独自マクロアーキテクチャの利用
- 正則化
- 高速化を意識したアーキテクチャ
- アーキテクチャの自動設計

最新のCNN改良手法

- **Residualモジュールの改良**
- 独自モジュールの利用
- 独自マクロアーキテクチャの利用
- 正則化
- 高速化を意識したアーキテクチャ
- アーキテクチャの自動設計

Residualモジュールの改良

- Residualモジュールをゴニョゴニョする

- WideResNet
  - 深くするより、幅（チャネル数）を大きくして（相対的に）浅いネットワークのほうが精度・学習速度が優秀
- PyramidNet
  - サウンサンプルするタイミングでチャネル数を2倍にするのではなく、全てのresidualモジュールで徐々にチャネル数を増加させる
    - チャネル数を2倍にするresidualモジュールにネットワークが過度に依存してしまうのを防ぐ

Residualモジュールのベストプラクティス

- 初代ResNet
  - conv - bn - relu - conv - bn - add - relu
- pre-act ResNet（add→ReLUだとショートカットの効果が半減）
  - bn - relu - conv - bn - relu - conv - add
- PyramidNet（ReLUを減らす、BN追加）
  - **bn - conv - bn - relu - conv - bn - add**
- WideResNet（dropoutがタスクによって効果がある）
  - bn - relu - conv - bn - relu - dropout - conv – add

- pre-actで、ReLUを1つ削除したPyramidNet系が良い（bnの追加は？個人の意見です）

最新のCNN改良手法

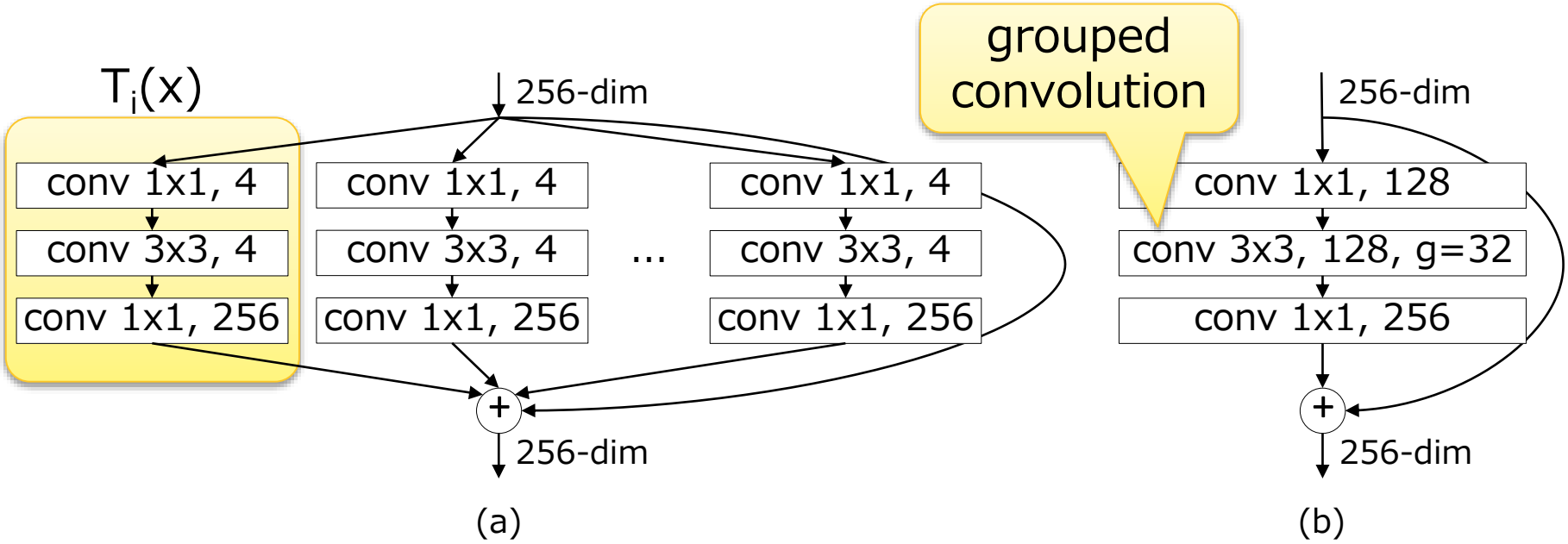
- Residualモジュールの改良
- **独自モジュールの利用**
- 独自マクロアーキテクチャの利用
- 正則化
- 高速化を意識したアーキテクチャ
- アーキテクチャの自動設計

ResNeXt

- ResNetの処理ブロックを”Network-in-Neuron”

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

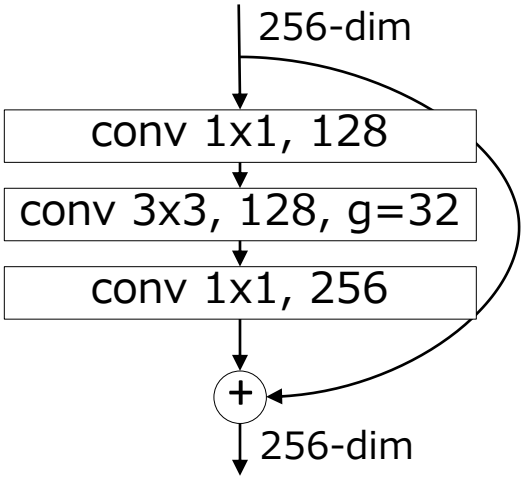
に置き換えたもの



- (a)と(b)は等価表現

ResNeXt

- ResNeXtとresidualモジュール（bottleneckバージョン）



次元削減

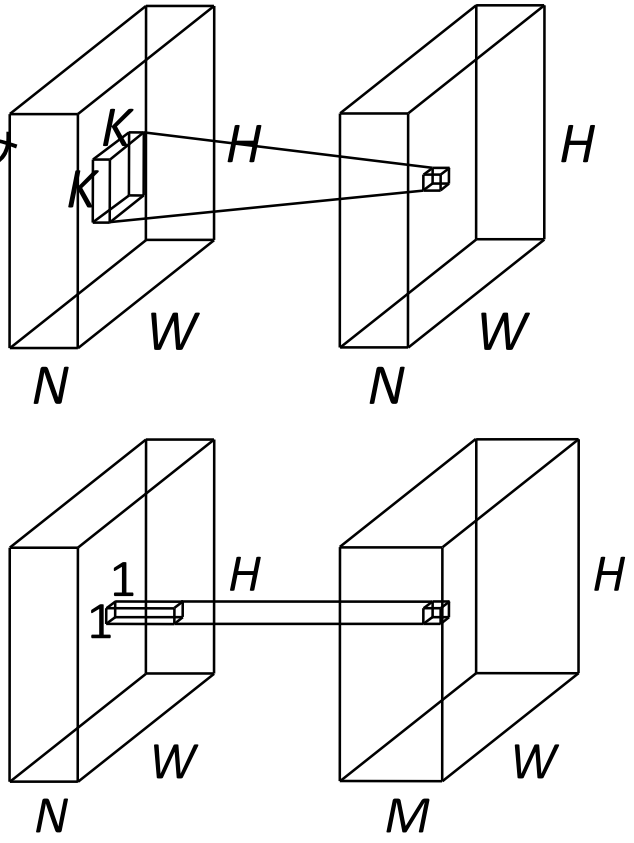
3x3畳み込み（grouped）

次元復元

- ResNeXtのモジュールは、bottleneckバージョンのresidualモジュールとみなすことができる
  - Grouped convによりパラメータ数と表現能力のトレードオフを改善

Xception

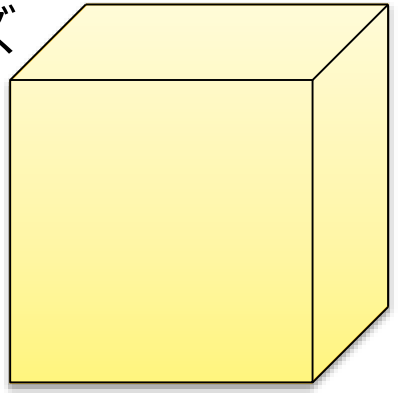
- 空間方向のとチャネル方向に独立に畳み込みを行う **separable畳み込み**を活用
  - Depthwise畳み込み（空間方向）
    - 特徴マップに対しチャネル毎に畳み込み
    - 計算量：$H \cdot W \cdot \cancel{N} \cdot K^2 \cdot M$ (M=N)
      $H \cdot W \cdot K^2 \cdot N$
  - Pointwise畳み込み（チャネル方向）
    - 1x1の畳み込み
    - 計算量：$H \cdot W \cdot N \cdot K^2 \cdot M$ (K=1)
      $H \cdot W \cdot N \cdot M$
  - Depthwise + pointwise
    - 計算量：$H \cdot W \cdot N \cdot (K^2 + M)$
      $\fallingdotseq H \cdot W \cdot N \cdot M$ ($M >> K^2$)



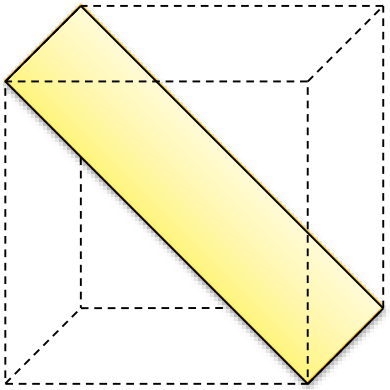
- 通常のKxK畳み込みから大幅に計算量を削減

Xception

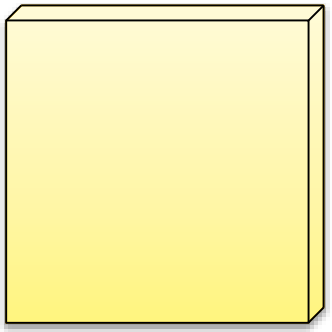
- Separable畳み込みのパラメータ
- 通常の畳み込み



- Separable畳み込み



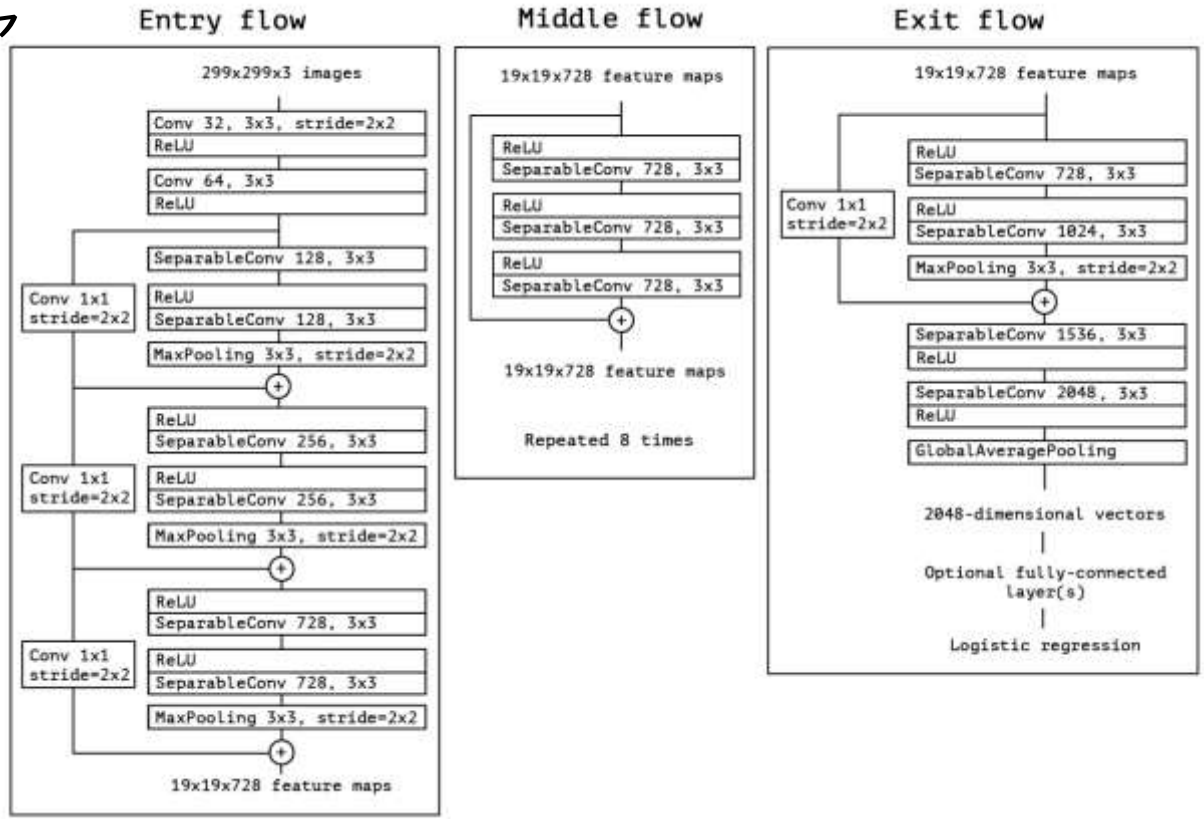
Depthwise畳み込み



Pointwise畳み込み

Xception

- 全体アーキテクチャ



F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proc. of CVPR, 2017.

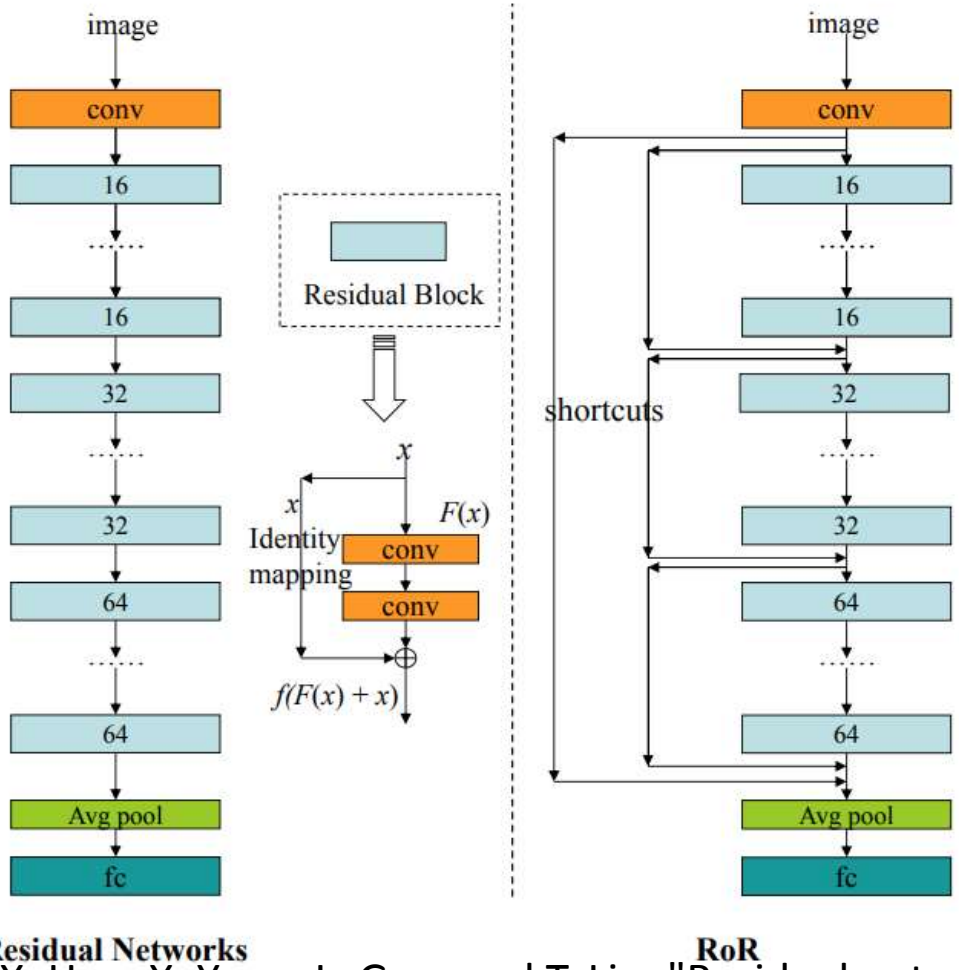
- モジュール構造 （sep: separable畳み込み）
  - relu - sep - bn - relu - sep - bn - relu - sep - bn - add

最新のCNN改良手法

- Residualモジュールの改良
- 独自モジュールの利用
- **独自マクロアーキテクチャの利用**
- 正則化
- 高速化を意識したアーキテクチャ
- アーキテクチャの自動設計

Residual Networks of Residual Networks (RoR)

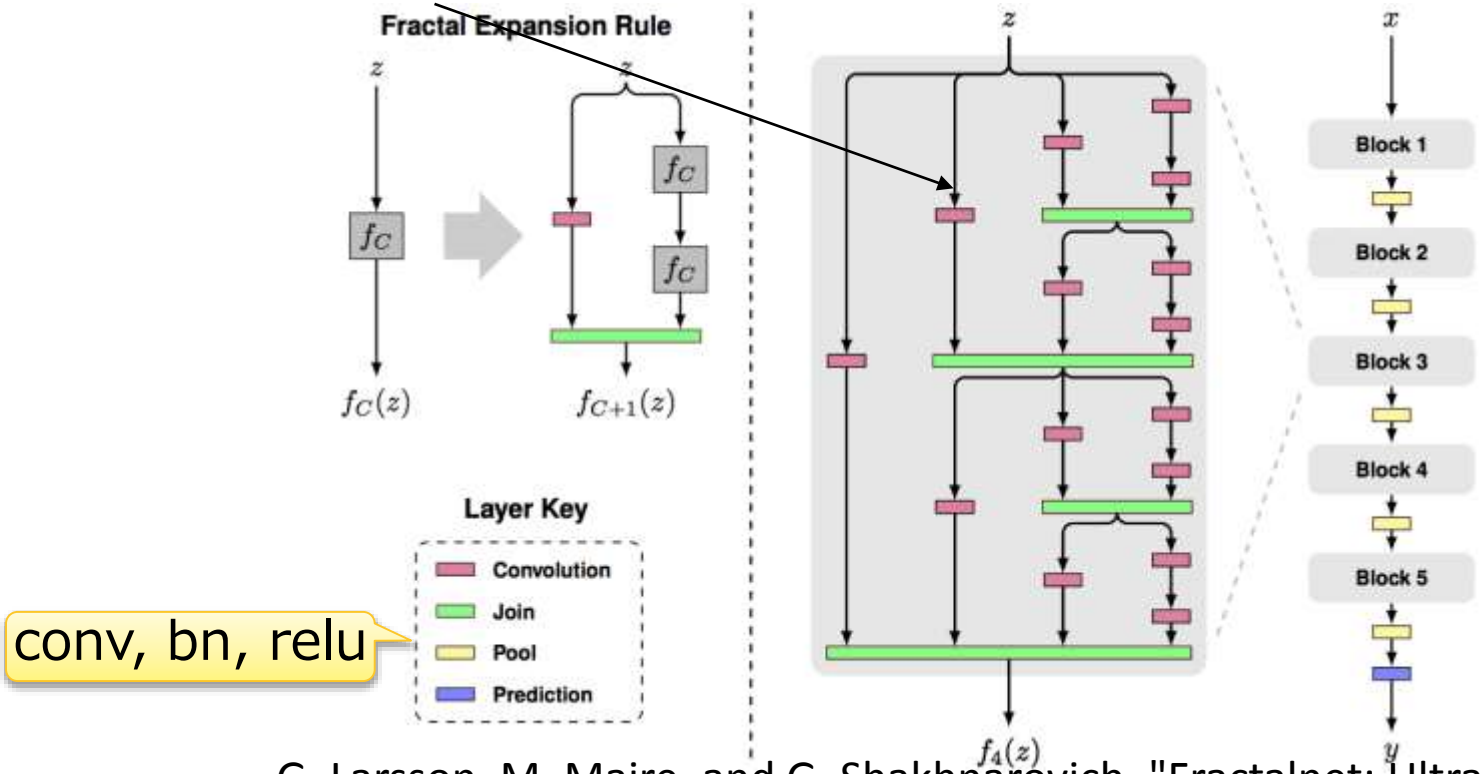
- 複数のResidualモジュールを1つのモジュールとみなし その間にショートカットを作成する



K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu, "Residual networks of residual networks: Multilevel residual networks," in TCSVT, 2017.

FractalNet

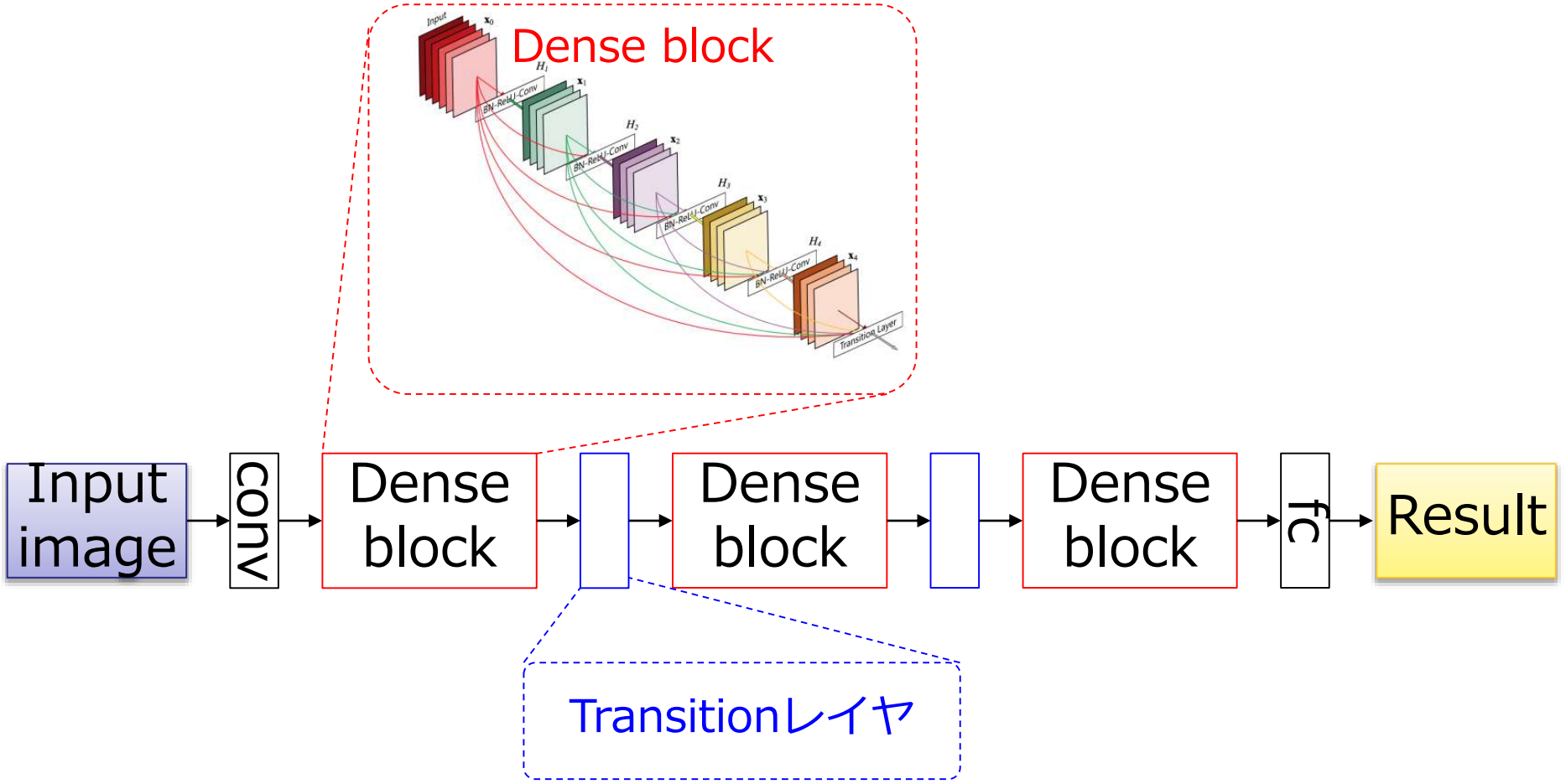
- 再帰的に定義されるネットワーク
  - パスをdropするdropout的な学習
    - 1カラムだけ残す or 各パスをランダムにdrop
  - 浅い列のパスがResNetのショートカットのような効果



G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," in Proc. ofICLR, 2017.

DenseNet (CVPR'17 best paper)

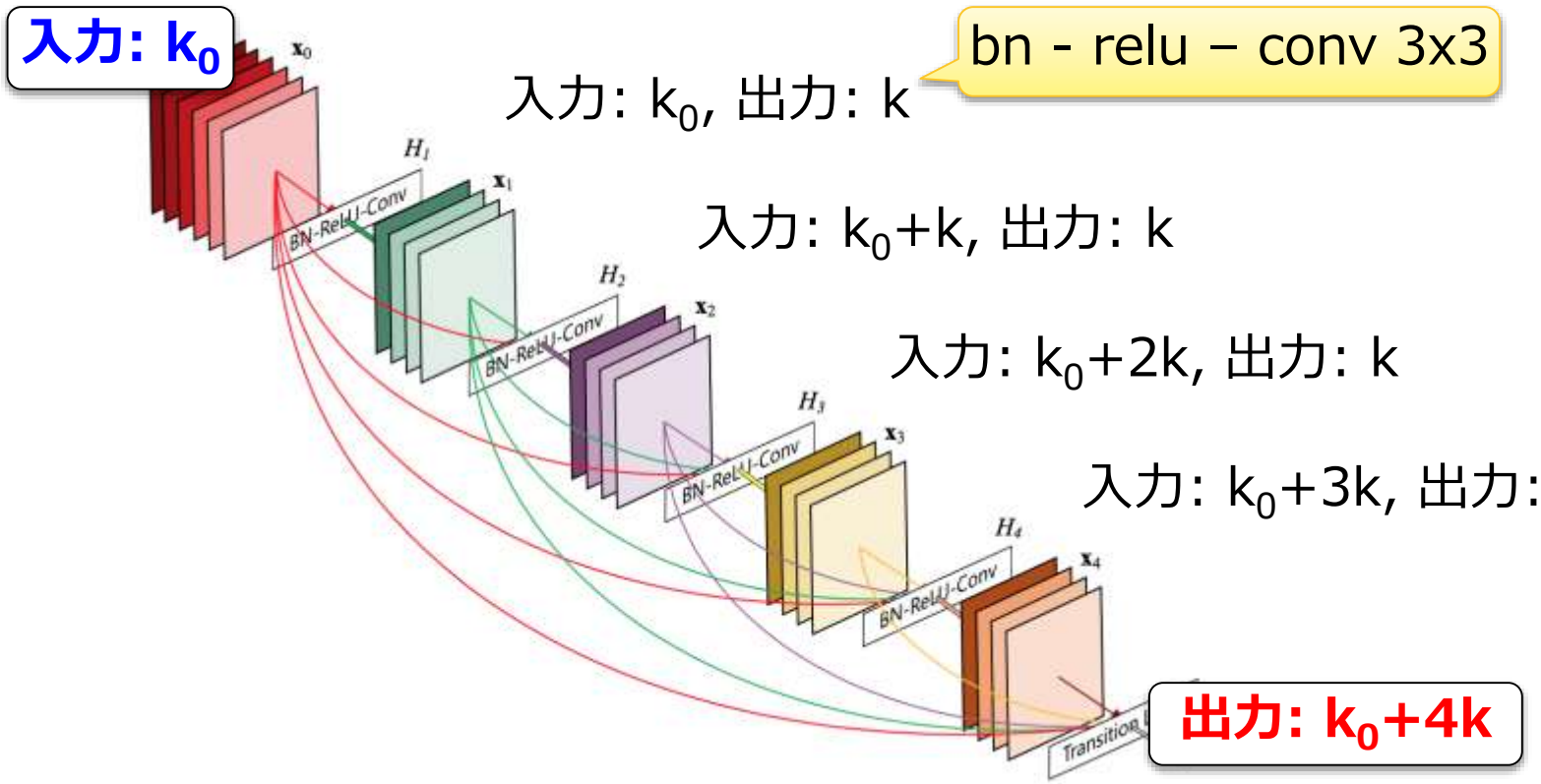
- **Dense block**を**transition layer**でつないだ構造



G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten, "Densely connected convolutional networks", in Proc. of CVPR, 2017.

Dense Block

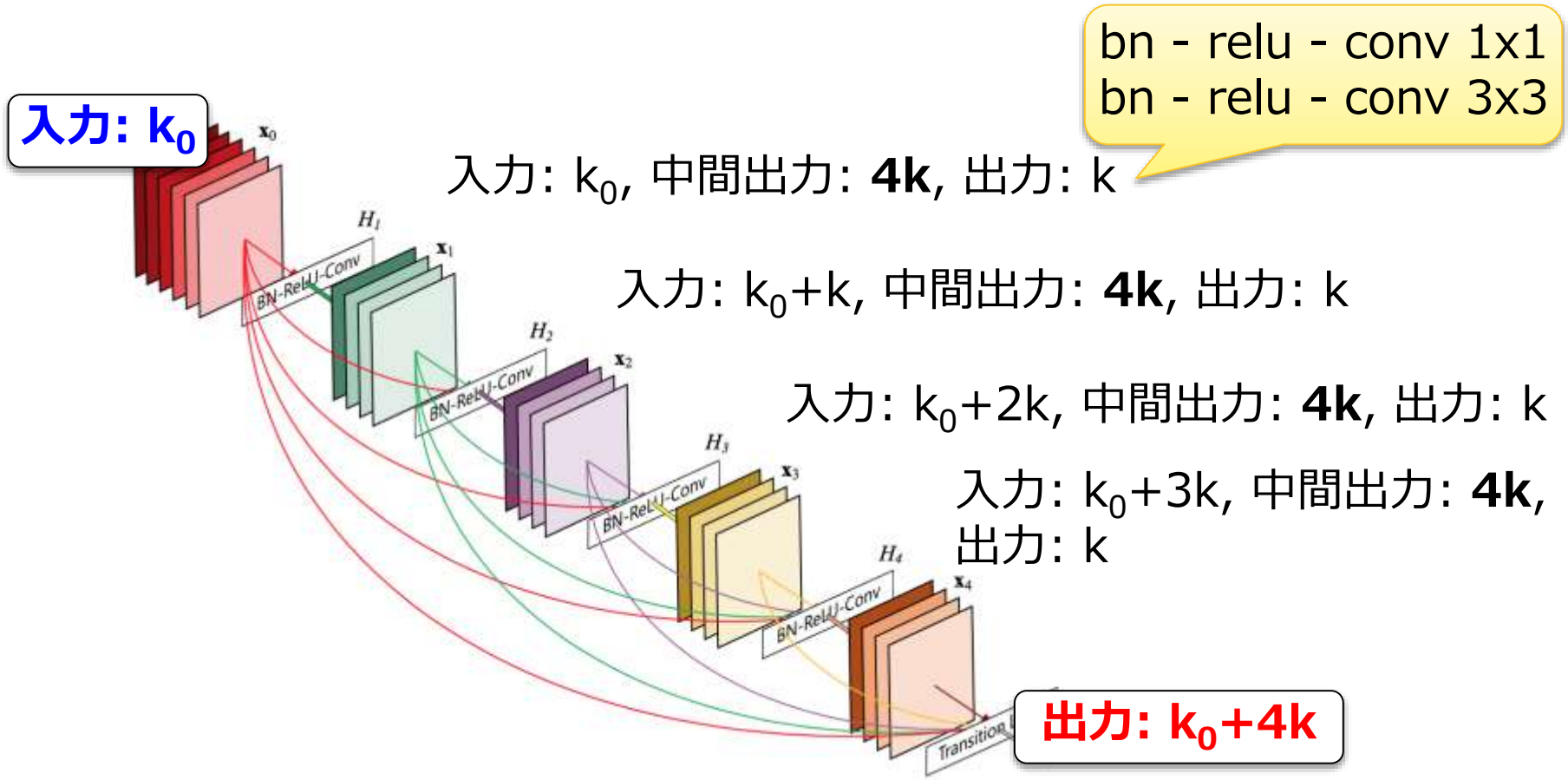
- 同一ブロック内の自分より前の層全ての出力を各層への入力とする処理ブロック
  - **入力k_0**、総数l、各層の出力k とすると**出力は$k_0+l\times k$**



G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten, “Densely connected convolutional networks", in Proc. of CVPR, 2017.

Dense Block (Bottleneckバージョン)

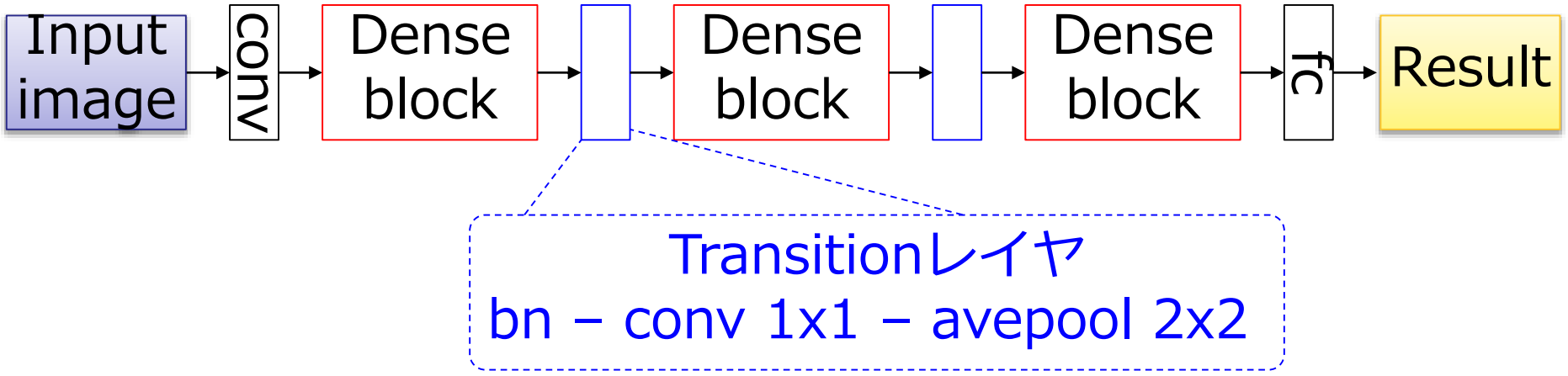
- k0 >> k なので通常の畳み込みよりも計算量が少ないが conv 1x1で次元削減を行うことで更に計算量を削減



G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten, “Densely connected convolutional networks", in Proc. of CVPR, 2017.

Transition Layer

- Transitionレイヤの基本的な役割はダウンダンプリング
  - conv 1x1の入出力チャネル数は k0+l×k
- Compressionバージョンのtransitionレイヤ
  - conv 1x1の出力チャネル数を $\boldsymbol{\theta}$(k0+l×k) とする
  - θ≦1、論文では0.5



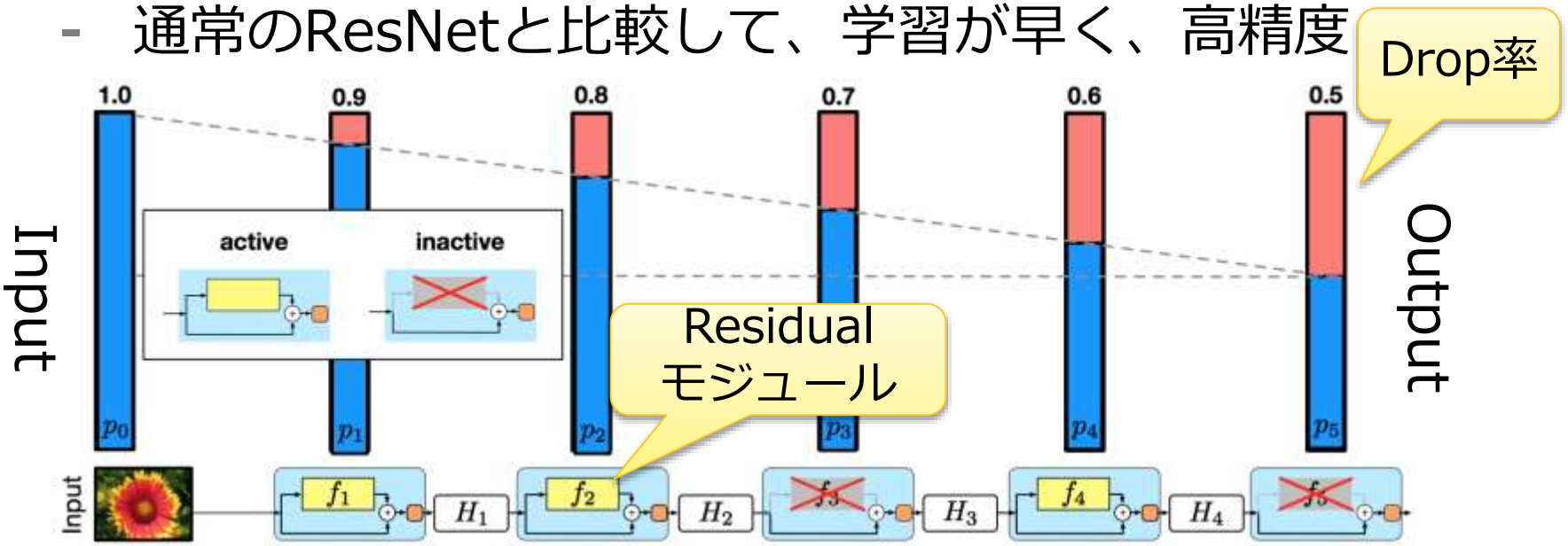
- 前述のbottleneck構造と圧縮するtransitionレイヤを利用する場合が精度とパラメータ数のトレードオフで優秀(DenseNet-BC)

最新のCNN改良手法

- Residualモジュールの改良
- 独自モジュールの利用
- 独自マクロアーキテクチャの利用
- **正則化**
- 高速化を意識したアーキテクチャ
- アーキテクチャの自動設計

Deep Networks with Stochastic Depth

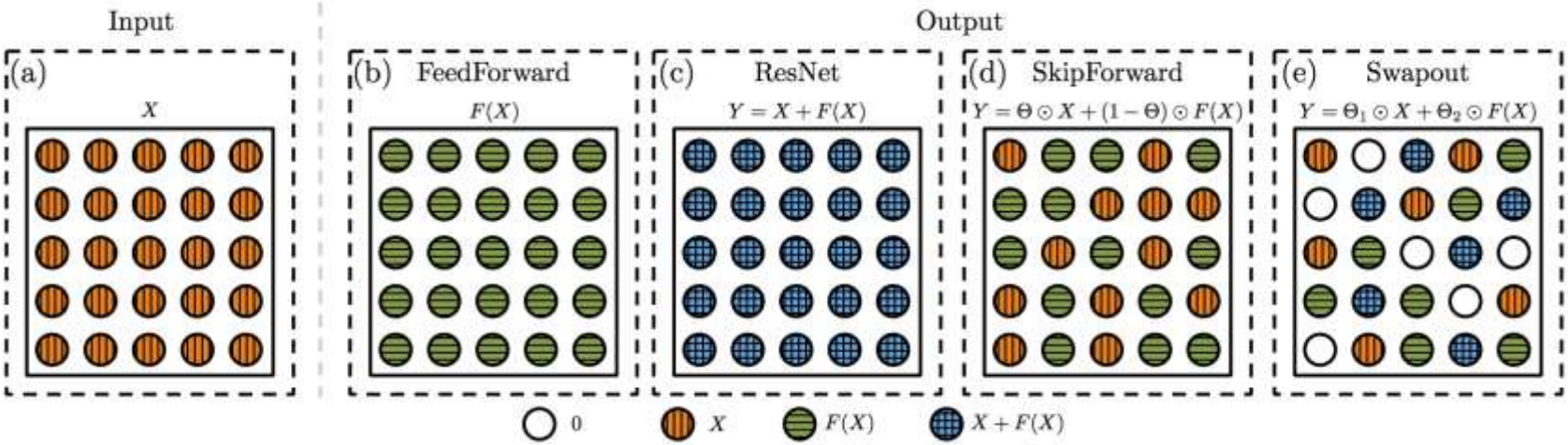
- ResNetは非常にdeepであることが学習を難しくしている
  - バッチ毎にresidual unitをランダムにdropすることで「期待値で見たときの深さ」を浅くする
  - Drop確率は出力に近いほど高くなるように線形に増加
- テスト時はdropせずに、drop率で出力をスケーリング
  - 通常のResNetと比較して、学習が早く、高精度



G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," in Proc. of ECCV, 2016.

Swapout

- Residualモジュールの出力　H(x) = F(x) + xについて
  - F(x) および x をそれぞれ個別にdropするdropout
  - 0, x, F(x), F(x) + x の4パターンの出力
    - Dropoutだと 0 か F(x) + x
    - モジュールの途中にいれると x か F(x) + x

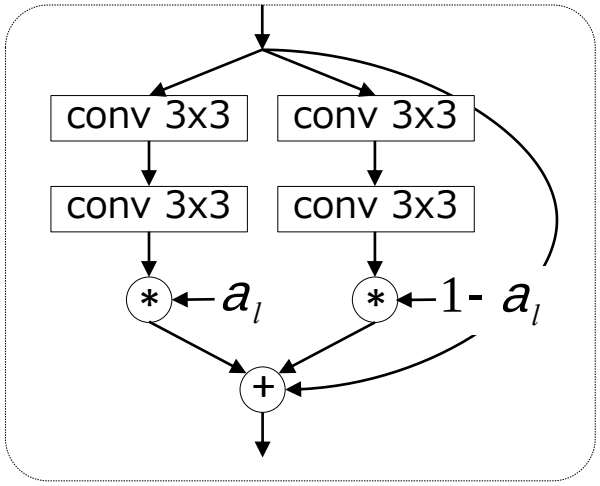


- 推論時もswapoutを有効にしたまま複数回forwardを行い、それらの平均を推論結果とする形でないと精度がでない

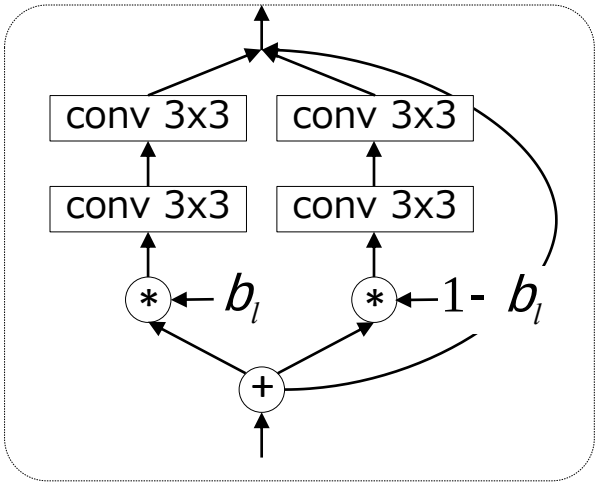
S. Singh, D. Hoiem, and D. Forsyth, "Swapout: Learning an ensemble of deep architectures," in Proc. of NIPS, 2016.

Shake-Shake Regularization

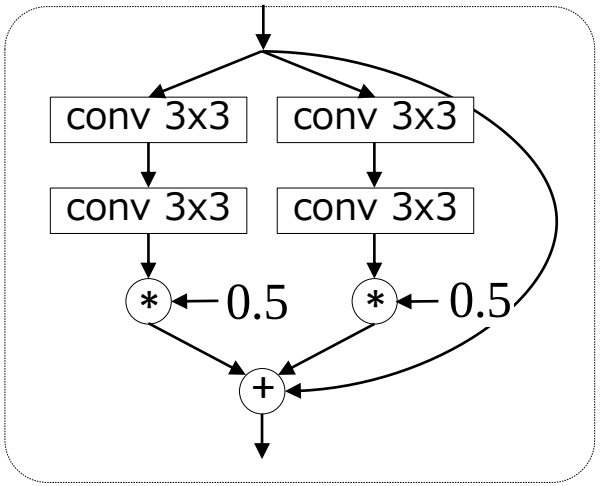
- 特徴マップレベルでのデータ拡張
  - 画像レベルではcroppingをすることで画像内の物体の比率を変化させることで、その変化にロバストに
  - 特徴マップレベルでも同じことができないか？
- モジュール内でネットワークを分岐させ画像レベルで各分岐の出力をランダムな比率（$\alpha \in [0, 1]$）で混合する
  - **Backward時には違う比率βを利用する**
- CIFAR-10/100でのSotA



Forward



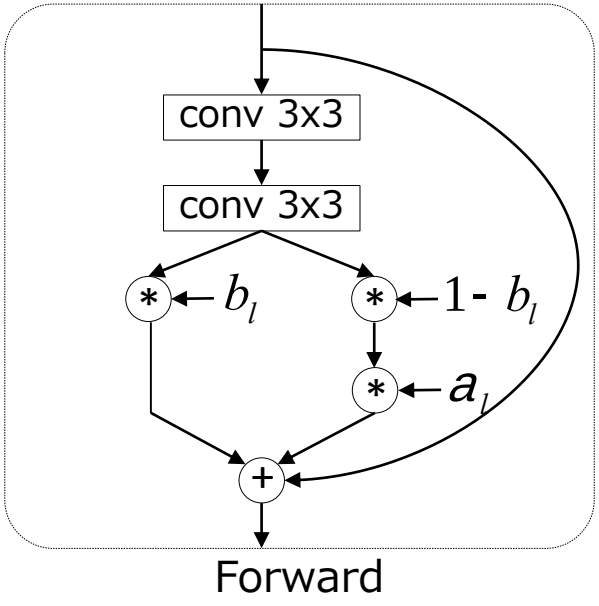
Backward

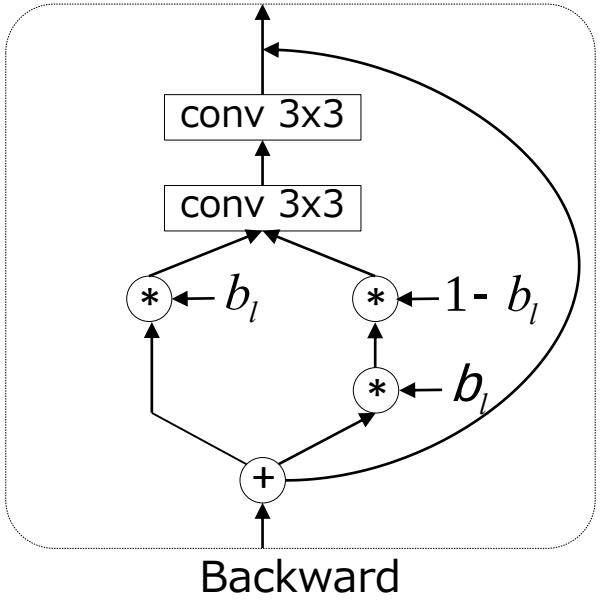


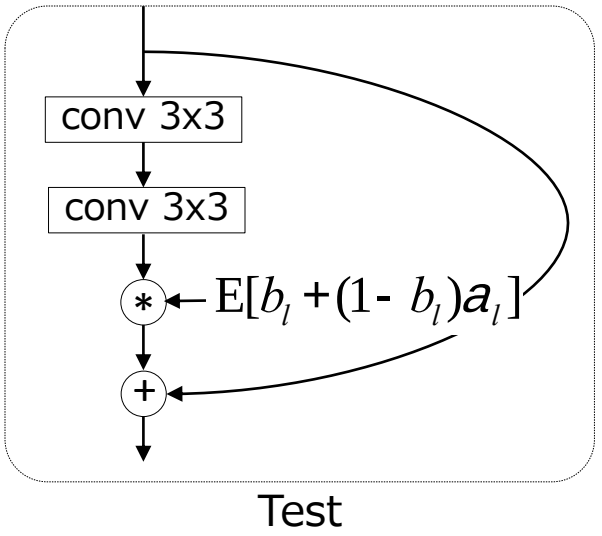
Test

ShakeDrop

- 分岐させずに特徴マップに対する外乱を加えられないか
  - PyramidNetとShakeShakeの統合
  - PyramidNetにおいてdropさせる代わりに
    ShakeShakeのような外乱を加える
- α∈[−1,1]、β∈[0,1]、チャネルレベルの外乱が最も良い
- CIFAR-10/100でのSotA



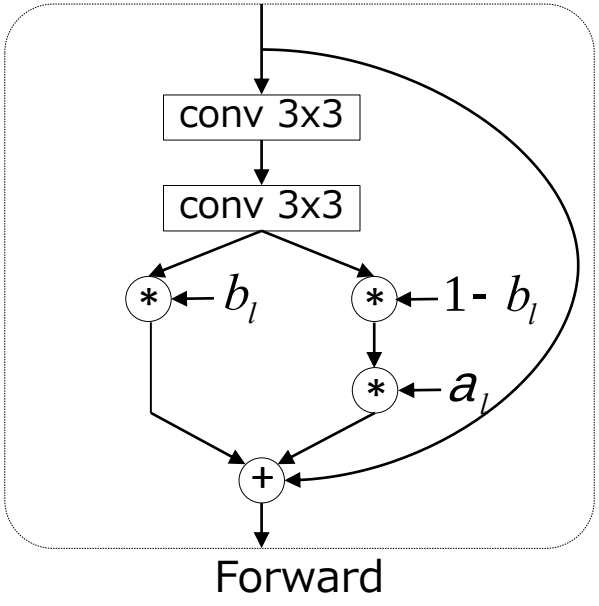


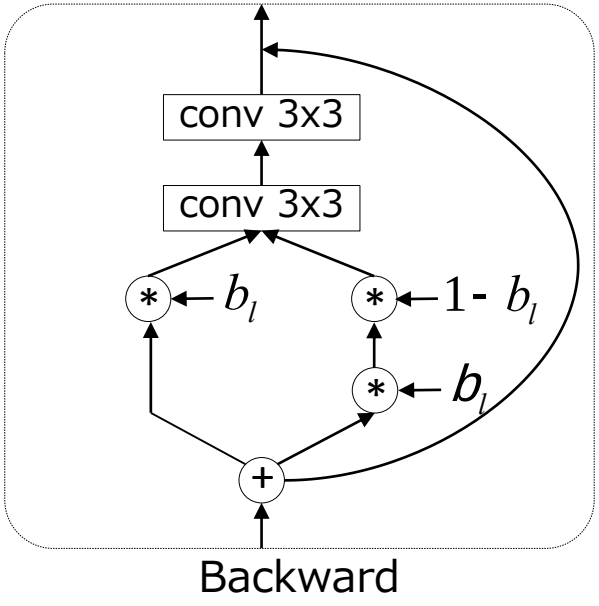


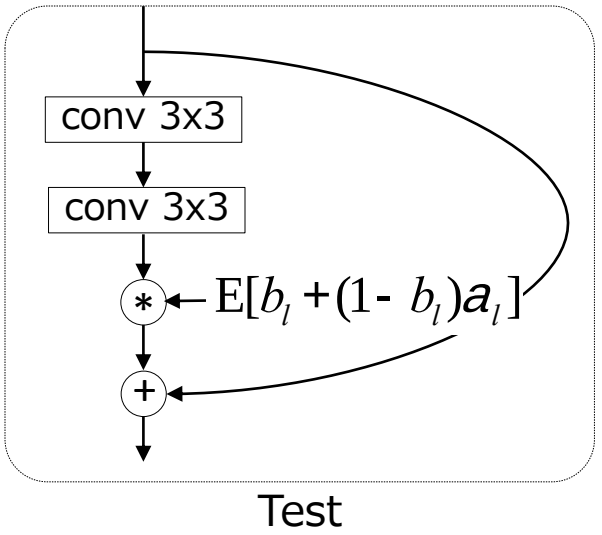
ShakeDrop

- 分岐させずに特徴マップに対する外乱を加えられないか
  - PyramidNetとShakeShakeの統合
  - PyramidNetにおいてdropさせる代わりに
    ShakeShakeのような外乱を
- α∈[−1,1]、β∈[0,1]、チャ が最も良い
- CIFAR-10/100でのSotA

少ないパラメータで
これを超えたやつが
また出るらしい

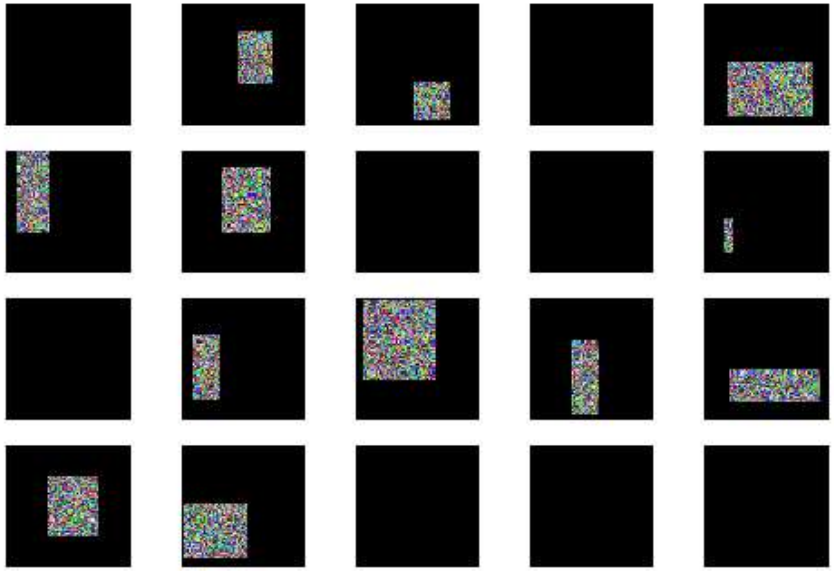






Random Erasing

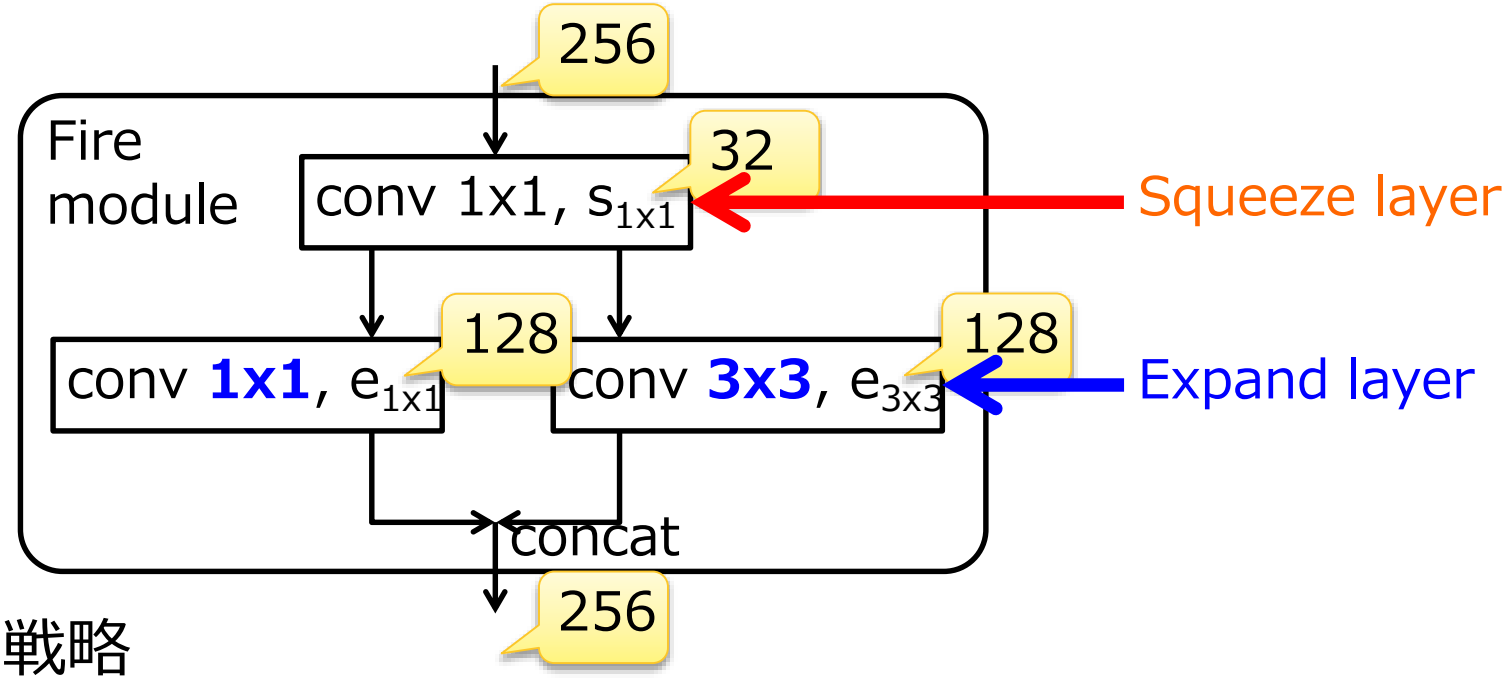
- データ拡張の一種
  - アーキテクチャではないが紹介
- 一定確率で、訓練画像のランダムな領域を
  ランダムなピクセルでマスク
- 汎化能力を高めテストエラーを軽減
  - 物体検出等でも効果あり



最新のCNN改良手法

- Residualモジュールの改良
- 独自モジュールの利用
- 独自マクロアーキテクチャの利用
- 正則化
- **高速化を意識したアーキテクチャ**
- アーキテクチャの自動設計

Fire module (SqueezeNet)

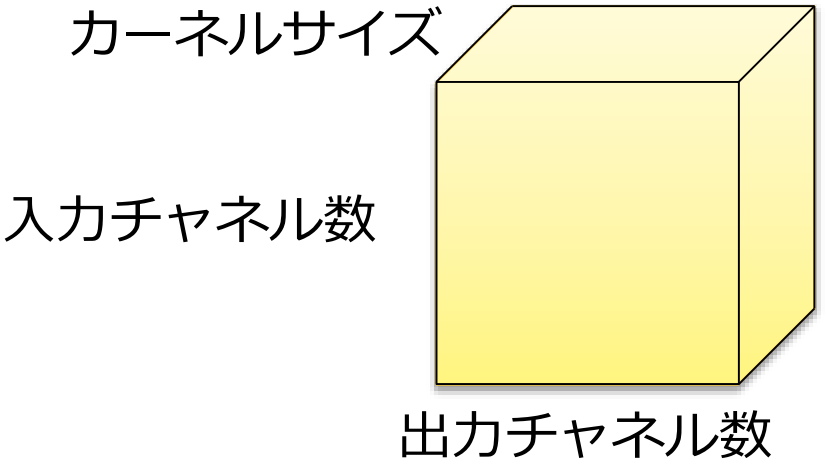


- 戦略
  - 3x3の代わりに1x1のフィルタを利用する
  - 3x3への入力となるチャネル数を少なくする
- Inceptionモジュールの特殊系とみなすこともできる

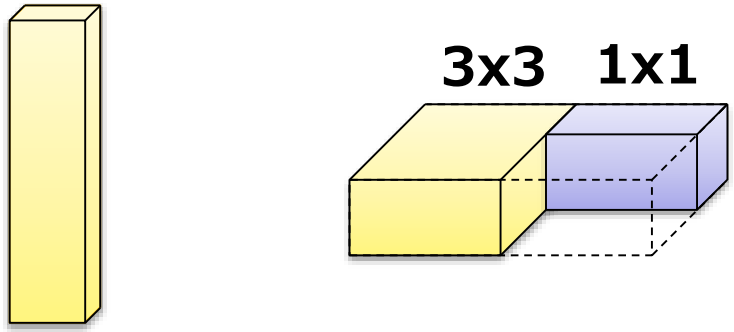
SqueezeNet

- パラメータ数

- 通常の畳み込み



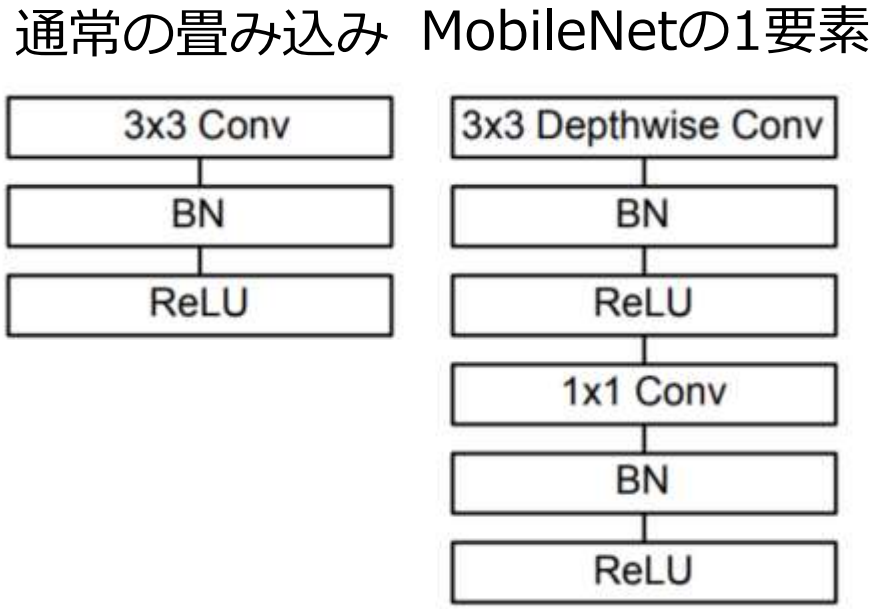
- Fire module



Squeeze layer　　Expand layer

MobileNet

- Xceptionと同様にseparable畳み込みを利用した軽量なネットワーク



- Xceptionとの違い
  - depthwiseとpointwiseの間にBNとReLUが入っている
    - Xceptionの論文では入れないほうが精度が良いとの主張
  - ショートカットは利用してない

A. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in arXiv:1704.04861, 2017.

最新のCNN改良手法

- Residualモジュールの改良
- 独自モジュールの利用
- 独自マクロアーキテクチャの利用
- 正則化
- 高速化を意識したアーキテクチャ
- **アーキテクチャの自動設計**

アーキテクチャの自動設計

- ネットワーク構造を進化的アルゴリズム最適化、250 GPUs [1]
- ネットワーク構造を出力するRNNをREINFORCEアルゴリズムで学習、800 GPUs 28days [2], 500 GPUs 4 days [4]
- 階層的なネットワーク構造を進化的アルゴリズムで最適化、~200 GPUs [3]
- [4] をSequential Model-Based Optimization (SMBO)で最適化、50 GPUs [5]

[1] E. Real, S. Moore, A. Selle, S. Saxena, Y. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," in Proc. of ICLR, 2017.
[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in arXiv:1611.01578, 2016.
[3] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in arXiv:1711.00436, 2017.
[4] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in arXiv:1707.07012, 2017.
[5] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive Neural Architecture Search," in arXiv:1712.00559, 2017.

アーキテクチャの自動設計

- ネットワーク構造を進化的アルゴリズム最適化、250 GPUs [1]
- ネットワーク構造を出力するRNNをREINFORCEアルゴリズムで学習、800 GPUs 28days [2], 500 GPUs 4 days [4]
- 階層的なネットワーク構造を進化的アルゴリズムで最適化、~200 GPUs [3]
- [4] をSequential Model-Based Optimization (SMBO)で最適化、50 GPUs [5]

[1] E. Real, S. Moore, A. Selle, S. Saxena, Y. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," in Proc. of ICLR, 2017.
[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in arXiv:1611.01578, 2016.
[3] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in arXiv:1711.00436, 2017.
[4] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in arXiv:1707.07012, 2017.
[5] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive Neural Architecture Search," in arXiv:1712.00559, 2017.

Google Brain

Google Brain

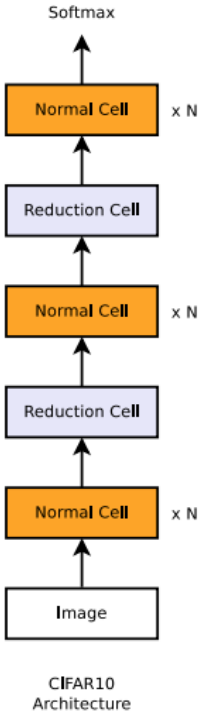
Deep Mind

Google Brain

Google **

Learning Transferable Architectures for Scalable Image Recognition

- ネットワーク構造を出力するRNNをREINFORCEアルゴリズムで学習
- 全体の構造とモジュール内で利用できる構成要素が決まっている（これの元となった論文ではそうではない）



CIFAR10 Architecture

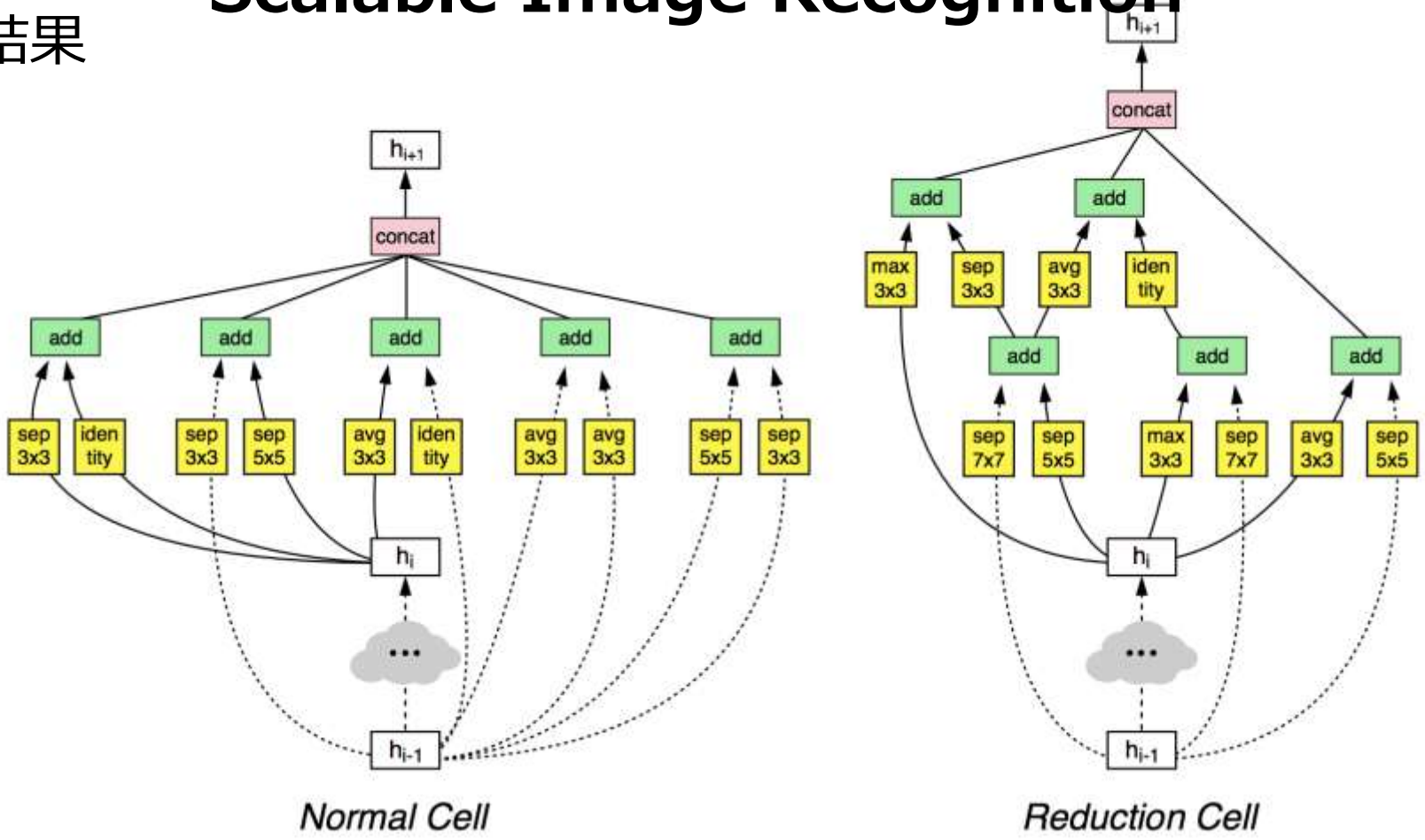
←全体の構造
↓利用できる構成要素
（計算量の小さいものが多い）

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

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

Learning Transferable Architectures for Scalable Image Recognition

- 結果



- Sepばっかり（見た目上の計算量が小さい→実測は？）

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

Learning Transferable Architectures for Scalable Image Recognition

- つよい

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [59]	299×299	23.8 M	5.72 B	78.0	93.9
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [57]	299×299	55.8 M	13.2 B	80.4	95.3
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [67]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [68]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
SENet [25]	**320×320**	**145.8 M**	**42.3 B**	**82.7**	**96.2**
NASNet-A (6 @ 4032)	**331×331**	**88.9 M**	**23.8 B**	**82.7**	**96.2**

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

各アーキテクチャの比較

アーキテクチャの比較

- 一般物体認識のデータセット
  - CIFAR10(10クラス，５万枚学習，１万枚テスト)
  - CIFAR100(100クラス，５万枚学習，１万枚テスト)
  - ImageNet(1000クラス，128枚学習，5万枚テスト)

CIFAR10/100

画像サイズ：32x32

https://www.cs.toronto.edu/~kriz/cifar.html

ImageNet

画像サイズ：バラバラ(224x224などにリサイズ)

http://karpathy.github.io/assets/cnntsne.jpeg

最新手法の実装

- Caffe

	学習環境	評価環境	学習済みモデル	公開コード
AlexNet	あり	あり	あり	https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet
VGG	あり	あり	あり	https://github.com/ruimashita/caffe-train
GoogLeNet	あり	あり	あり	https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet
ResNet	なし	あり	あり	https://github.com/KaimingHe/deep-residual-networks
MobleNet	なし	あり	あり	https://github.com/shicai/MobileNet-Caffe
SqueezeNet	あり	あり	あり	https://github.com/DeepScale/SqueezeNet
DenseNet	あり	あり	なし	https://github.com/liuzhuang13/DenseNetCaffe

最新手法の実装

- Tensorflow

	学習環境	評価環境	学習済みモデル	公開コード
AlexNet	なし	あり	なし	https://github.com/ethereon/caffe-tensorflow
VGG	なし	あり	なし	https://github.com/machrisaa/tensorflow-vgg https://www.cs.toronto.edu/~frossard/post/vgg16/
GoogLeNet	なし	あり	あり	https://www.tensorflow.org/versions/master/tutorials/image_recognition
ResNet	あり	あり	あり	https://github.com/ry/tensorflow-resnet
MobleNet	あり	あり	あり	https://github.com/Zehaos/MobileNet
SqueezeNet	あり	あり	あり	https://github.com/vonclites/squeezenet
DenseNet	あり	あり	なし	https://github.com/YixuanLi/densenet-tensorflow

最新手法の実装

- pytorch

	学習環境	評価環境	学習済みモデル	公開コード
AlexNet	あり	あり	あり	https://github.com/pytorch/examples/tree/master/imagenet
VGG	あり	あり	あり	https://github.com/pytorch/examples/tree/master/imagenet
GoogLeNet	あり	あり	あり	https://github.com/pytorch/examples/tree/master/imagenet
ResNet	あり	あり	あり	https://github.com/pytorch/examples/tree/master/imagenet
MobleNet	あり	あり	あり	https://github.com/marvis/pytorch-mobilenet
SqueezeNet			あり	https://github.com/pytorch/vision
DenseNet	あり	あり	あり	https://github.com/pytorch/vision https://github.com/bamos/densenet.pytorch

最新手法の実装

- chainer

	学習環境	評価環境	学習済みモデル	公開コード
AlexNet	あり	あり	あり	https://github.com/chainer/chainer/tree/master/examples/imagenet
VGG	なし	あり	あり	https://github.com/pytorch/examples/tree/master/imagenet
GoogLeNet	あり	あり	あり	https://github.com/chainer/chainer/tree/master/examples/imagenet
ResNet	あり	なし	あり	https://github.com/yasunorikudo/chainer-ResNet/tree/master/v2 https://github.com/chainer/chainer/tree/master/examples/imagenet
MobleNet				
SqueezeNet				https://github.com/ejlb/squeezenet-chainer
DenseNet	あり	あり	あり	https://github.com/yasunorikudo/chainer-DenseNet

最新手法の実装

- 各種学習済みモデルの比較（pytorch）

Model	Version	Acc@1	Acc@5
NASNet-A-Large	Tensorflow	82.693(82.566)	96.163(96.086)
InceptionResNetV2	Tensorflow	80.4(80.170)	95.3(95.234)
InceptionV4	Tensorflow	80.2(80.062)	95.3(94.926)
ResNeXt101_64x4d	Torch7	79.6(78.956)	94.7(94.252)
ResNeXt101_32x4d	Torch7	78.8(78.188)	94.4(93.886)
ResNet152	Pytorch	78.428	94.110
FBResNet152	Torch7	77.84(77.386)	93.84(93.594)
DenseNet161	Pytorch	77.560	93.798
InceptionV3	Pytorch	77.294	93.454
DenseNet201	Pytorch	77.152	93.548
ResNet101	Pytorch	77.438	93.672
DenseNet169	Pytorch	76.026	92.992
ResNet50	Pytorch	76.002	92.980
DenseNet121	Pytorch	74.646	92.136
VGG19_BN	Pytorch	74.266	92.066

https://github.com/Cadene/pretrained-models.pytorch

最新手法の実装

- 学習環境の実装（Chainer）
- 各種手法を同条件で学習・評価が可能
  - https://github.com/takedarts/resnetfamily
  - https://github.com/nutszebra/chainer_image_recognition

- 本比較実験では上記２つのgithubのコードを参考に行う

学習条件

	CIFAR10/100	Imagenet
学習データ数	１０万枚	１２８万枚
Data augmentation	位置ずれ，反転 （上下左右に４ピクセルのマージンを付与し，３２x３２をランダムにクロップ）	位置ずれ，反転 （２５６x２５６にリサイズし，２２４x２２４をランダムにクロップ）
エポック数	３００エポック	９０エポック
学習最適化法	モーメンタムSGD （モーメンタム:０．９）	モーメンタムSGD （モーメンタム:０．９）
学習係数スケジュール	初期値０．１，１５０エポック時，２２５エポック時に０．１倍	初期値０．１，３０エポック毎に０．１倍
バッチサイズ	１２８	２５６

学習環境

- CIFAR10/100 (single GPUで学習・評価)

GPU搭載ワークステーション：９７台

各端末のスペック
Quadoro P5000
Xeon E5-2620
Mem. 16GB
Infiniband

Ubuntu 16.04
chainer 3.0
chainer MN 1.0
opencv 3.2.0
nvidia-docker
singularity

学習環境

- Imagenet（８GPUで学習・評価）

DGX-1 (Tesla P100 x 8)

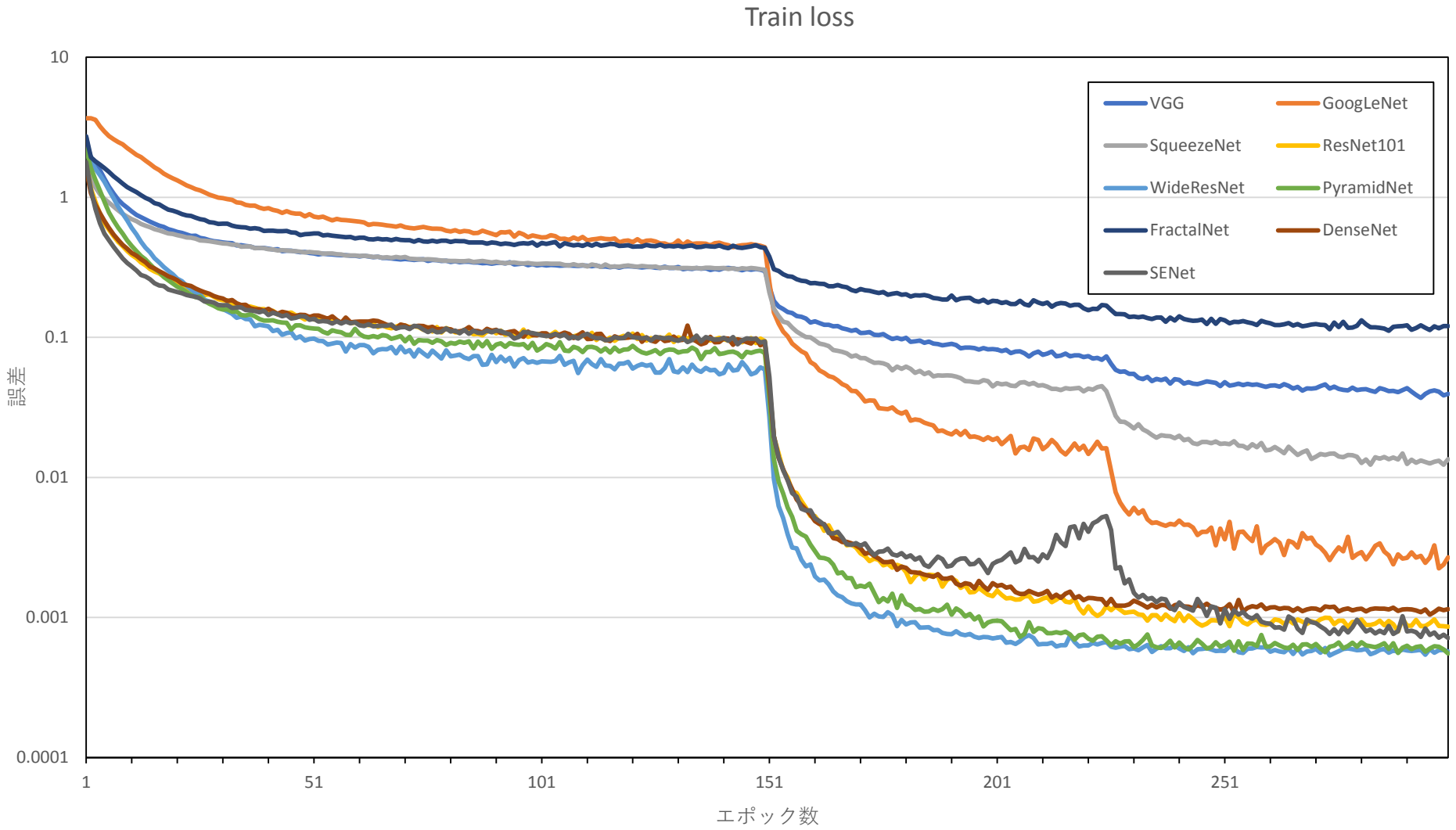
スペック
Tesla P100 x8
Xeon E5-2698
Mem. 512GB
NVLink

比較アーキテクチャ

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(α=84)	CIFAR10/100
FractalNet	20(Cols=3)	CIFAR10/100
DenseNet	72(k=12)	CIFAR10/100，ImageNet
SENet	50(r=4)	CIFAR10/100

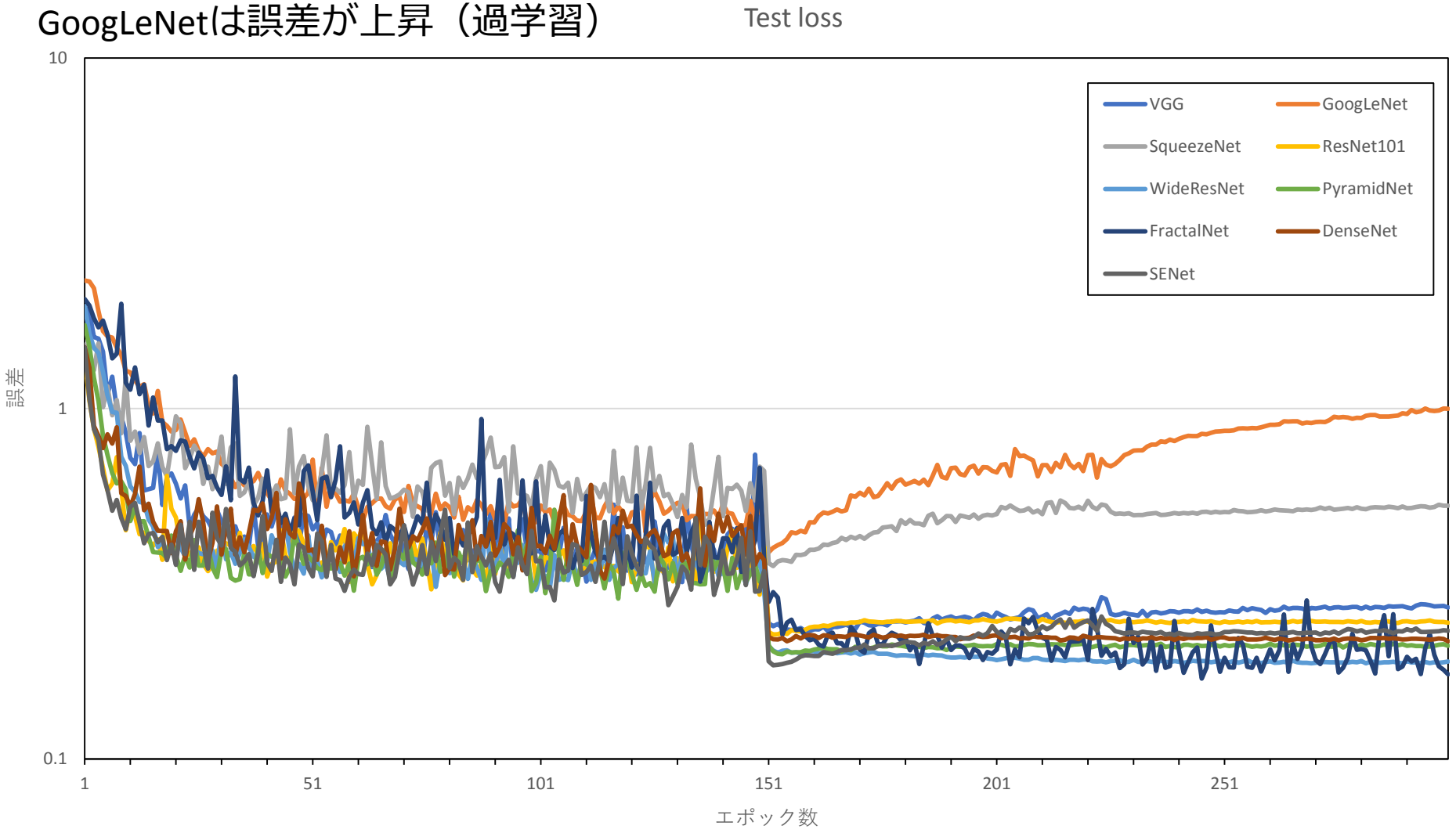
CIFAR10での誤差（学習データ）

WideRexNet・PyramidNetが最も誤差が低い
SENetは誤差が上昇しており，過学習気味



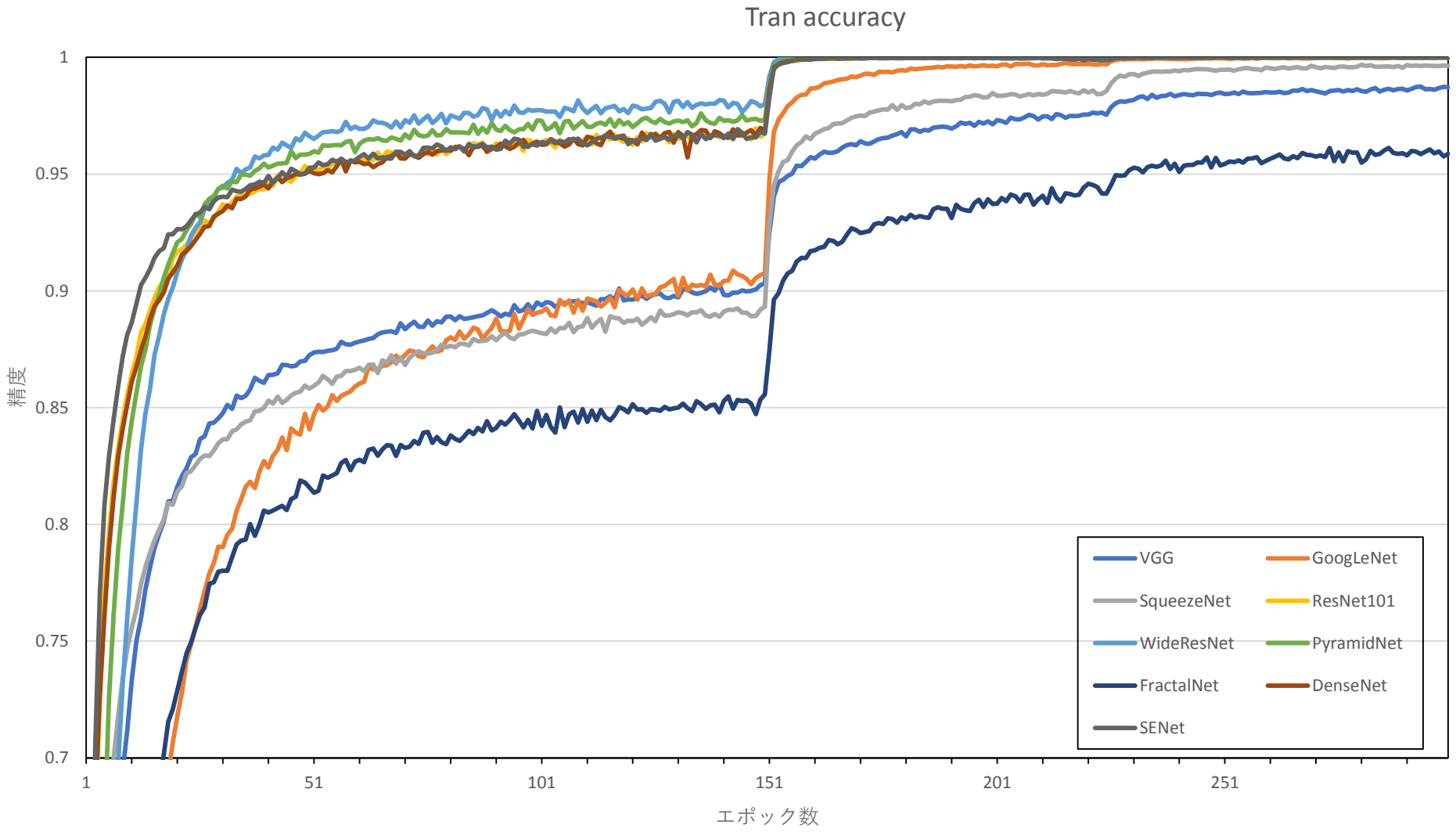
CIFAR10での誤差（テストデータ）

WideRexNetが最も誤差が低い
FractalNetは学習後期でもばらつきが大きい
GoogLeNetは誤差が上昇（過学習）



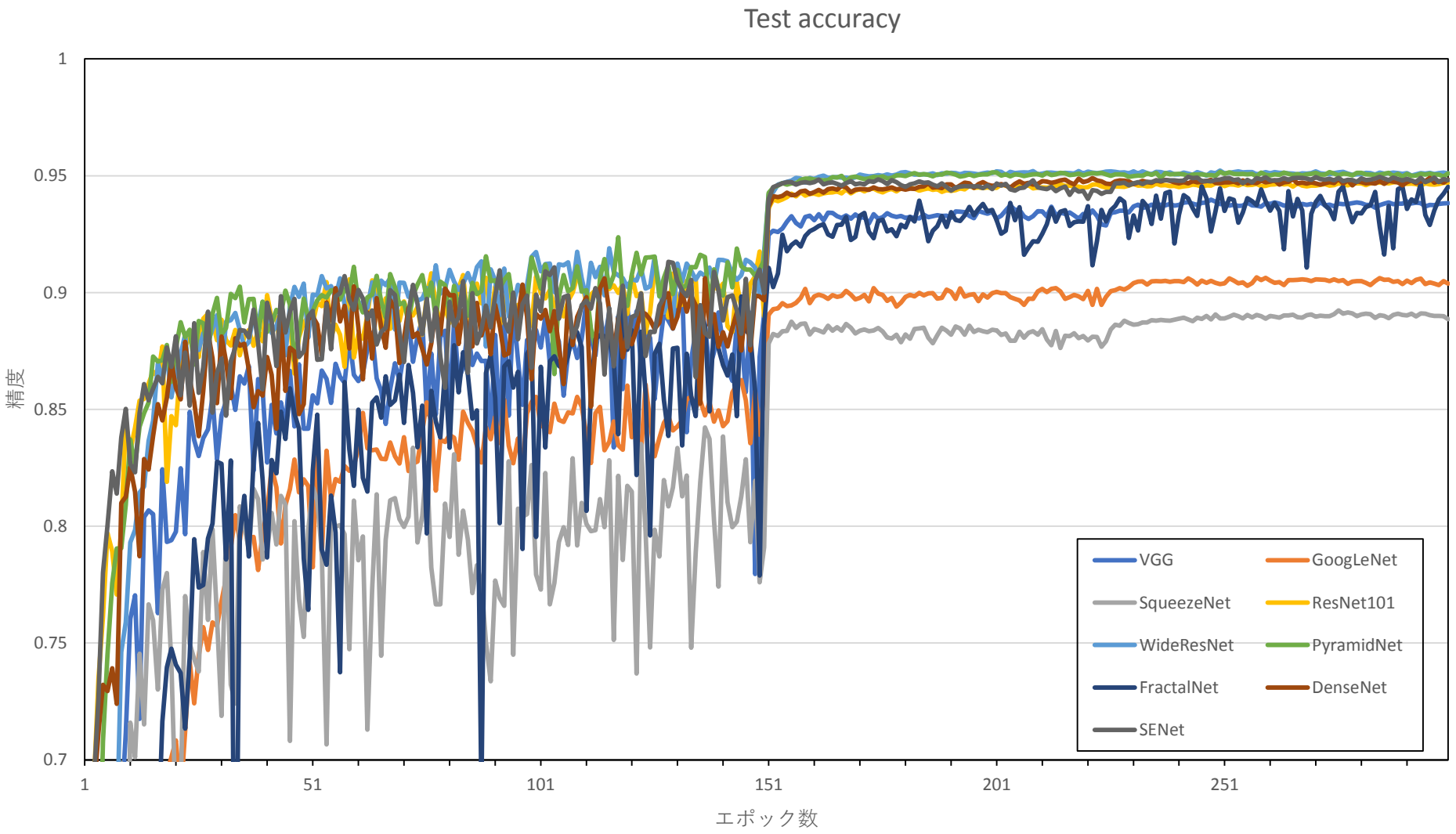
CIFAR10での精度（学習データ）

WideResNet・PyramidNet・DenseNetなど幅を持つアーキテクチャはほとんど同精度



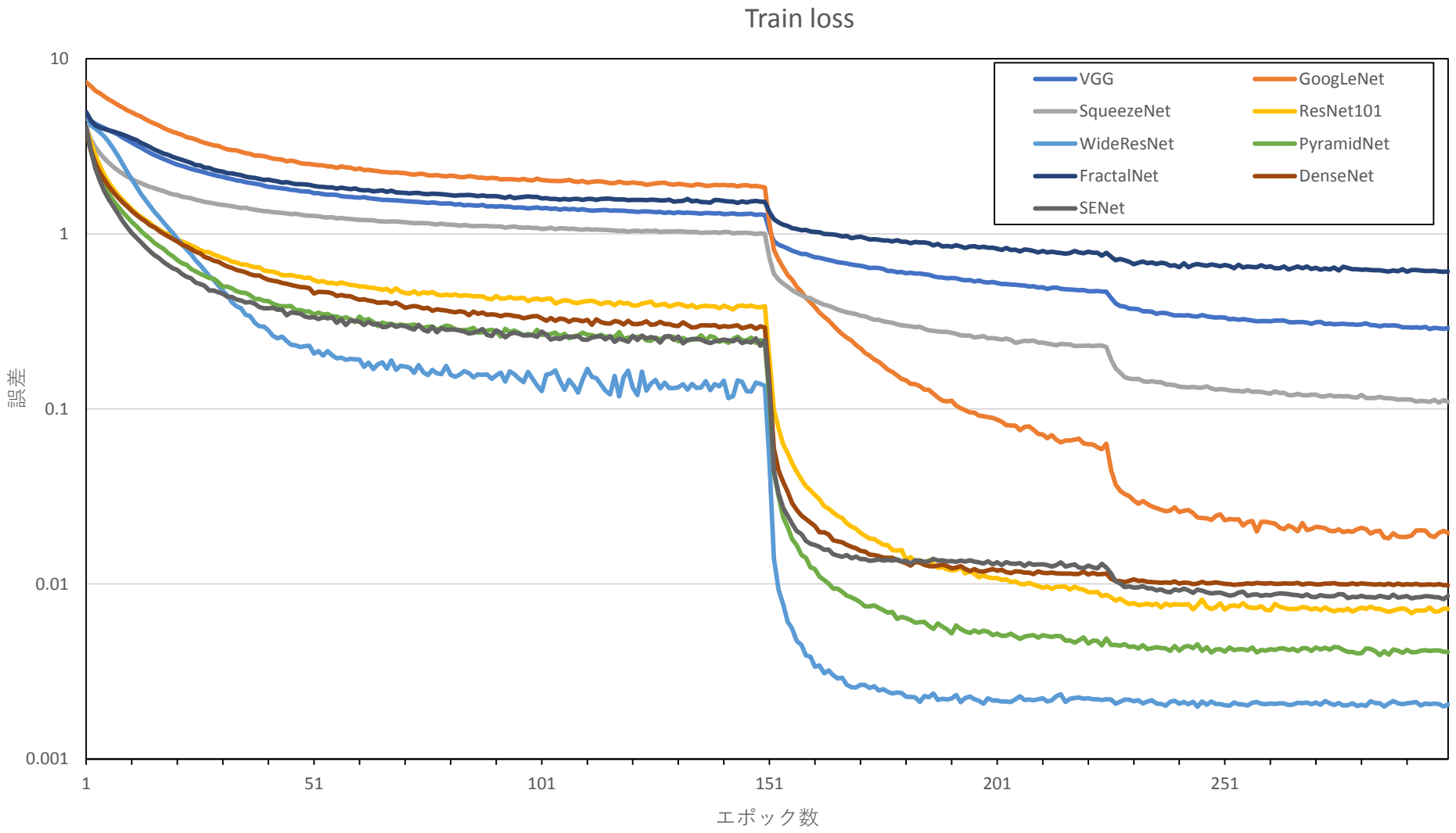
CIFAR10での精度（テストデータ）

WideResNet・PyramidNet・DenseNetなど幅を持つアーキテクチャはほとんど同精度



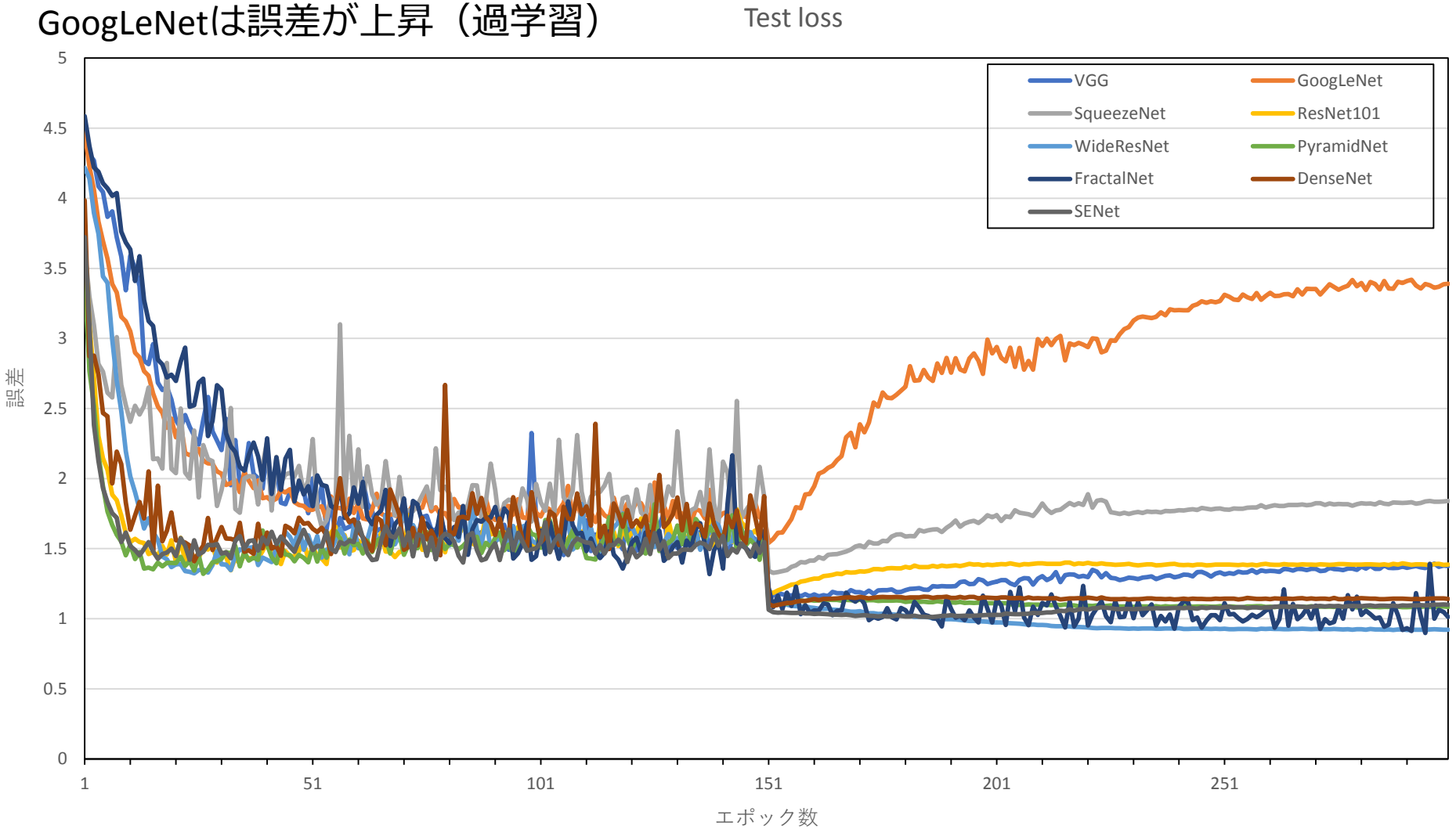
CIFAR100での誤差（学習データ）

CIFAR10と同様にWideResNet・PyramidNetの誤差が低い



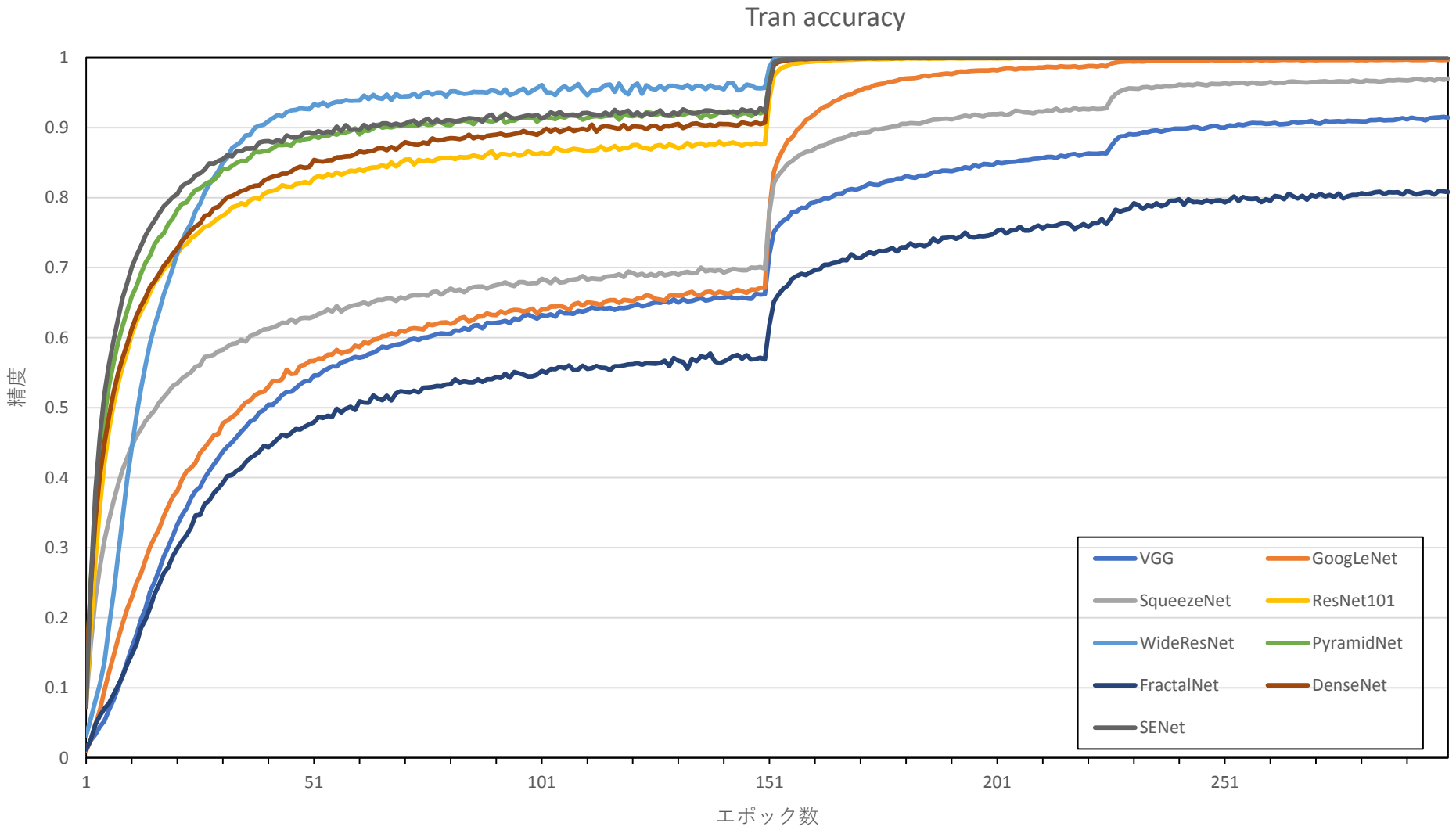
CIFAR100での誤差（テストデータ）

WideResNetが最も誤差が低い
FractalNetは学習後期でもばらつきが大きい
GoogLeNetは誤差が上昇（過学習）



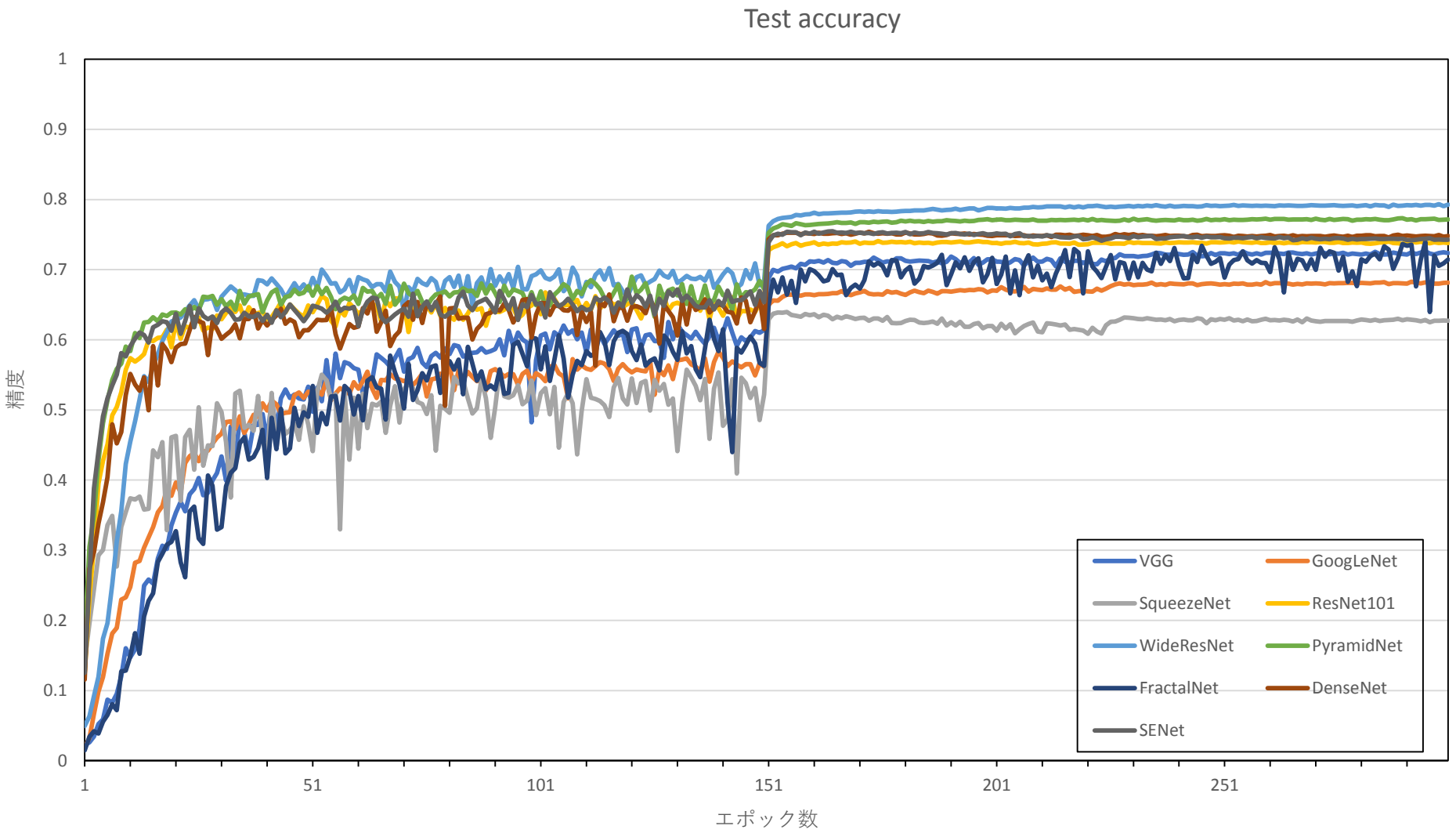
CIFAR100での精度（学習データ）

WideResNet・PyramidNet・DenseNetなど幅を持つアーキテクチャはほとんど同精度



CIFAR100での精度（テストデータ）

WideResNetが最も精度が高い



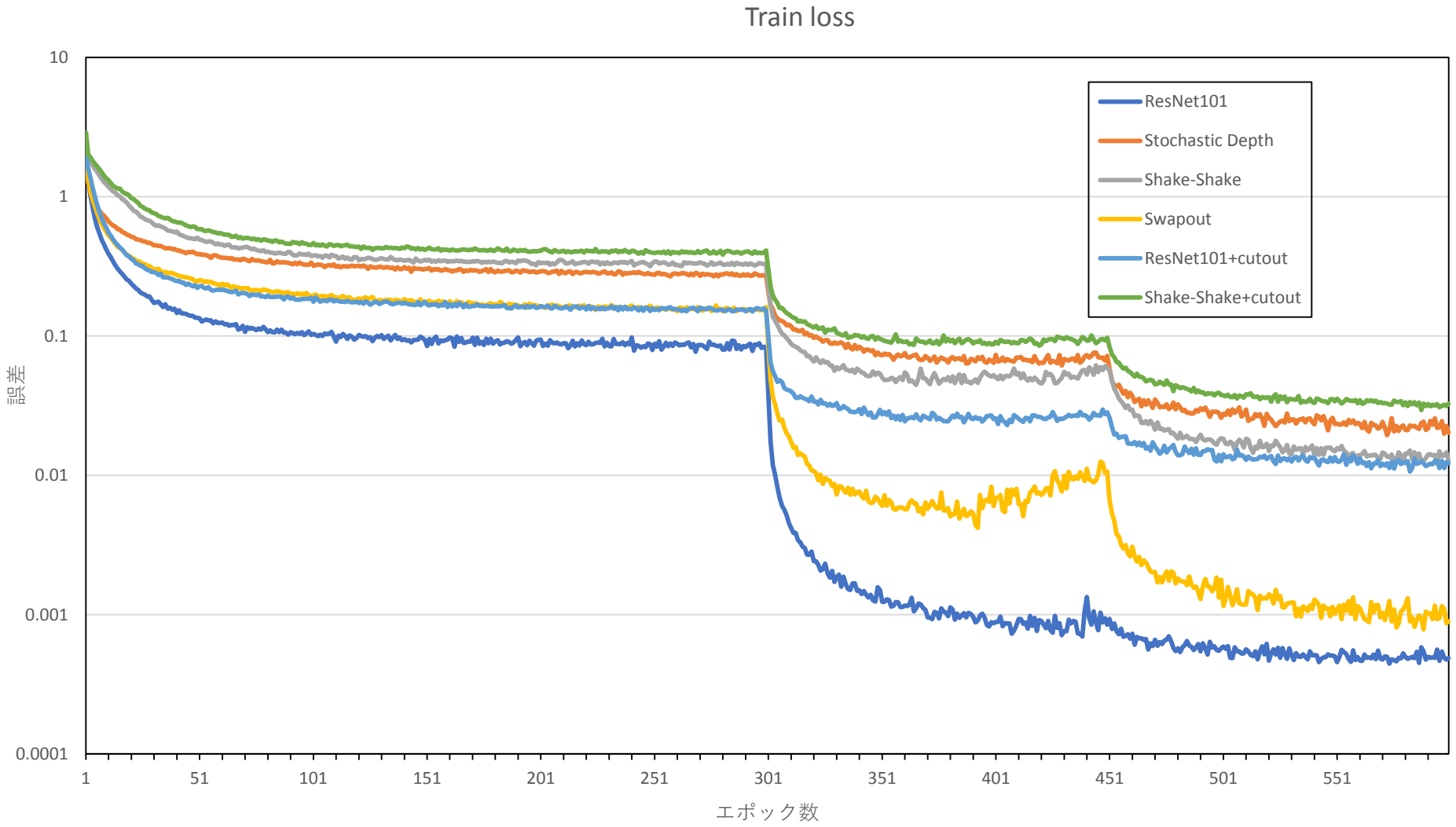
CIFAR10/100での精度

WideResNet・PyramidNetは精度が良いが，学習時間はResNetの1.5-2倍程度

アーキテクチャ名	エラー率[%] CIFAR10	CIFAR100	学習時間[h]
VGG	6.0	27.4	2.9
GoogLeNet	9.3	31.7	4.6
SqueezeNet	10.8(18.0)	36.0	1.8
ResNet101	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
FractalNet	5(4.6)	26.0	5.1
DenseNet	5.1(3.8)	24.6	19.6
SENet	4.8	24.2	36.7

CIFAR10での正則化手法の比較（学習誤差）

学習誤差はResNet101（正則化なし）が最も低い

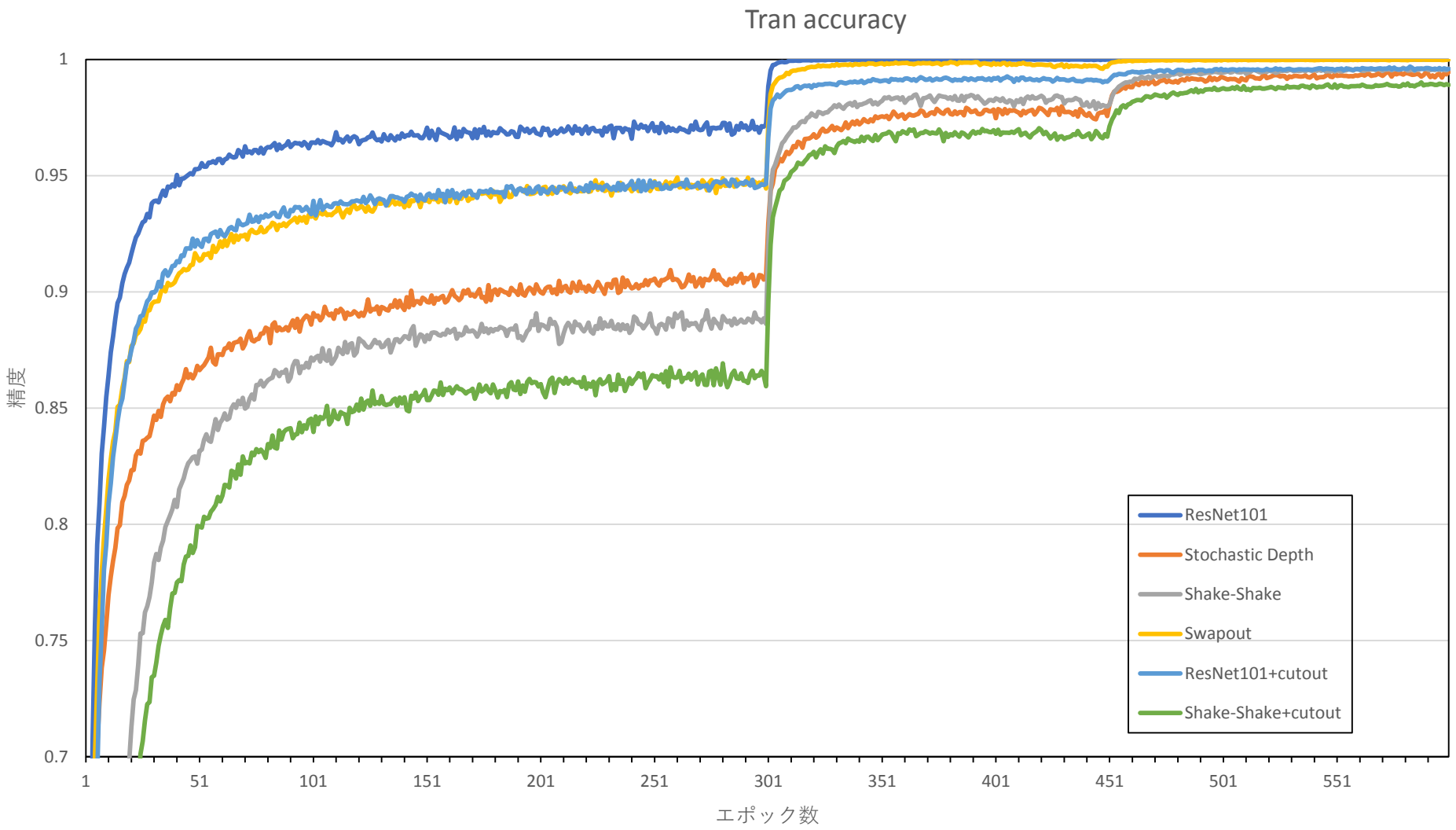


CIFAR10での正則化手法の比較（テスト誤差）

テスト誤差はShake-Shake，Shake-Shake+cutout（正則化あり）が最も低い
ResNet101は過学習している

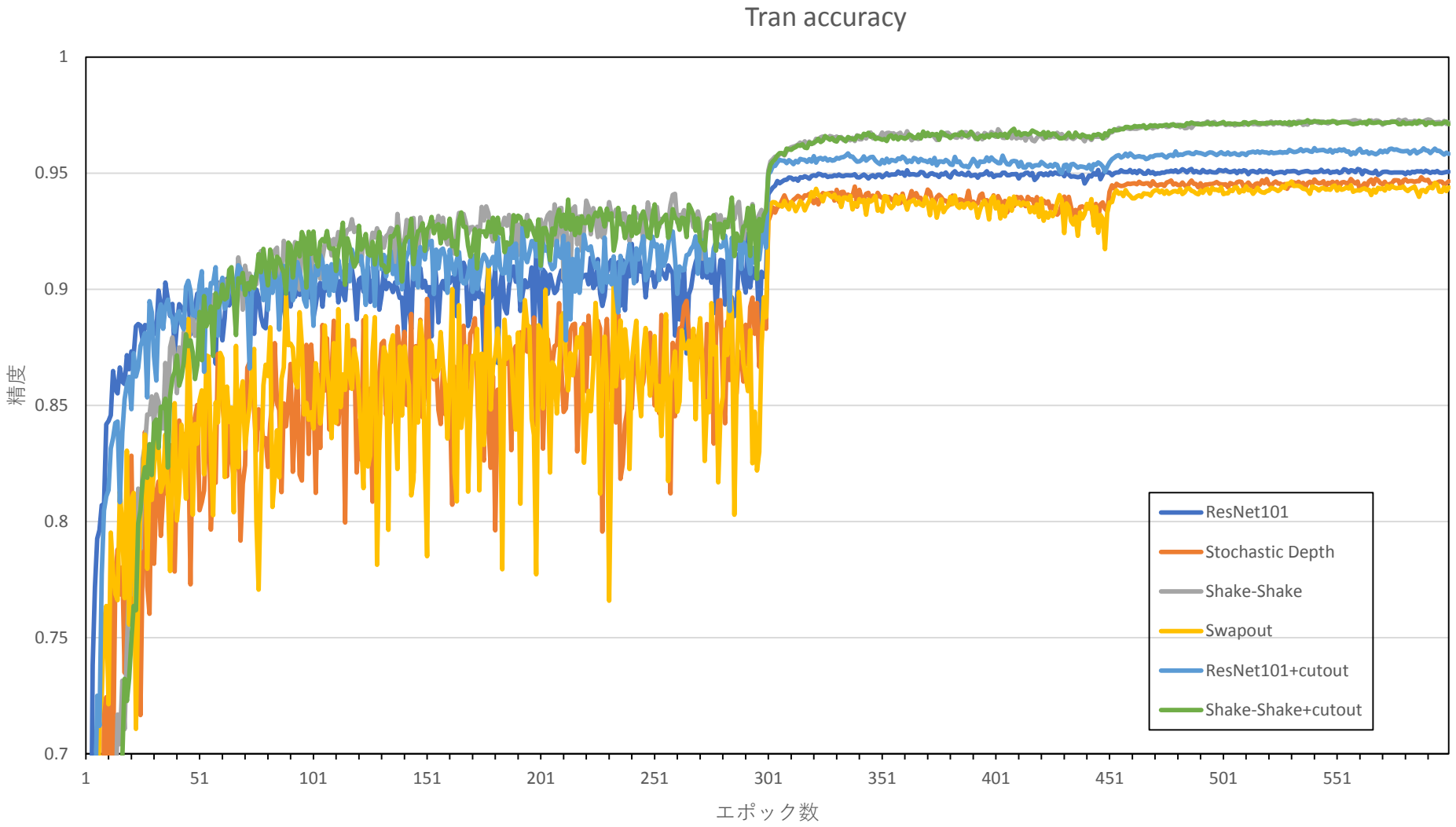
CIFAR10での正則化手法の比較（精度：学習データ）

ResNet・Swapoutが最も高い（過学習している）



CIFAR10での正則化手法の比較（精度：テストデータ）

Shake-Shake+cutout（正則化あり）の最も精度が高い



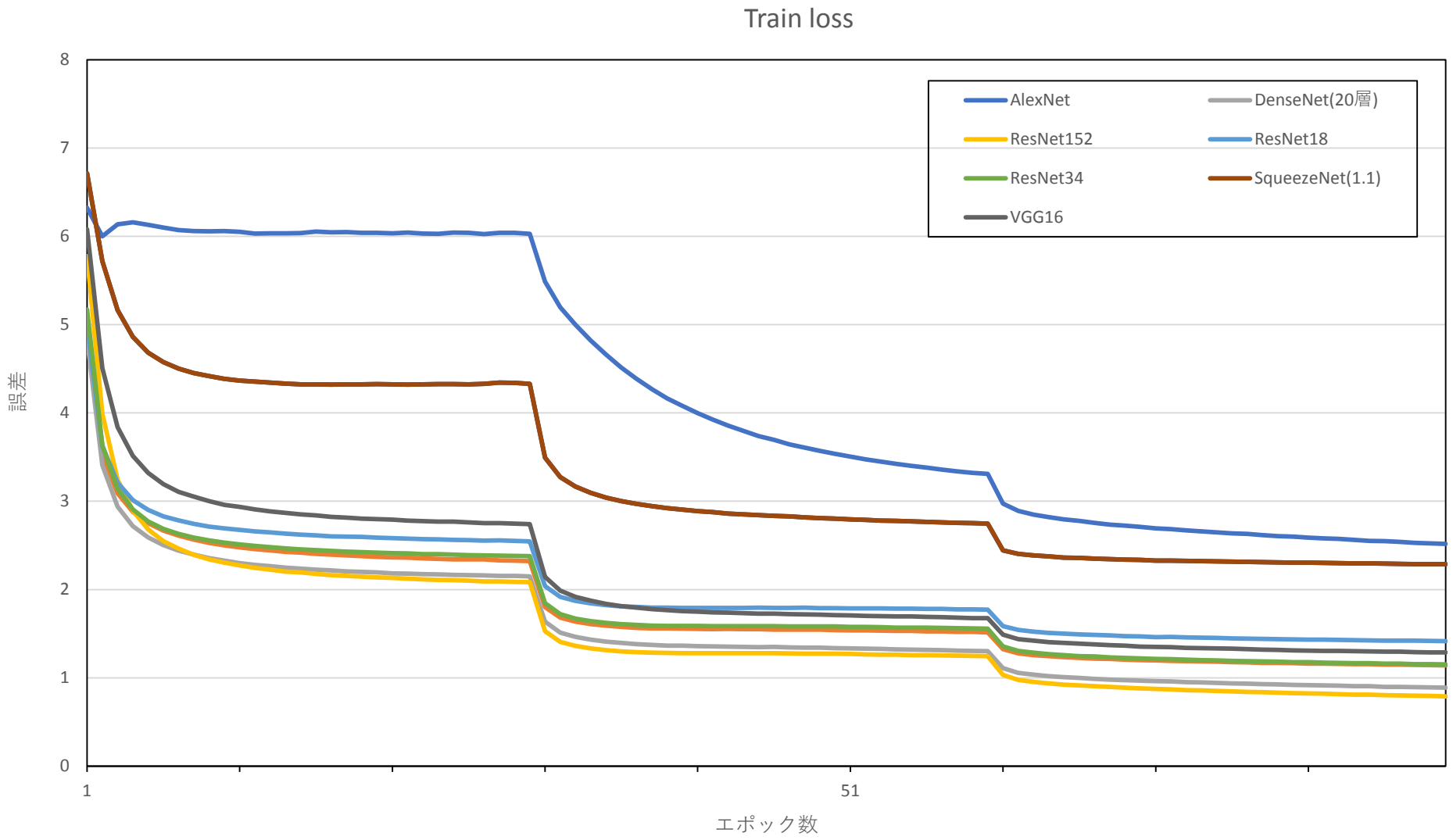
CIFAR10/100での正則化手法の比較

Shake-Shakeは正則化として効果が高い　（ただし，学習時間が大幅に増加）
Cutoutは汎化性向上に効く

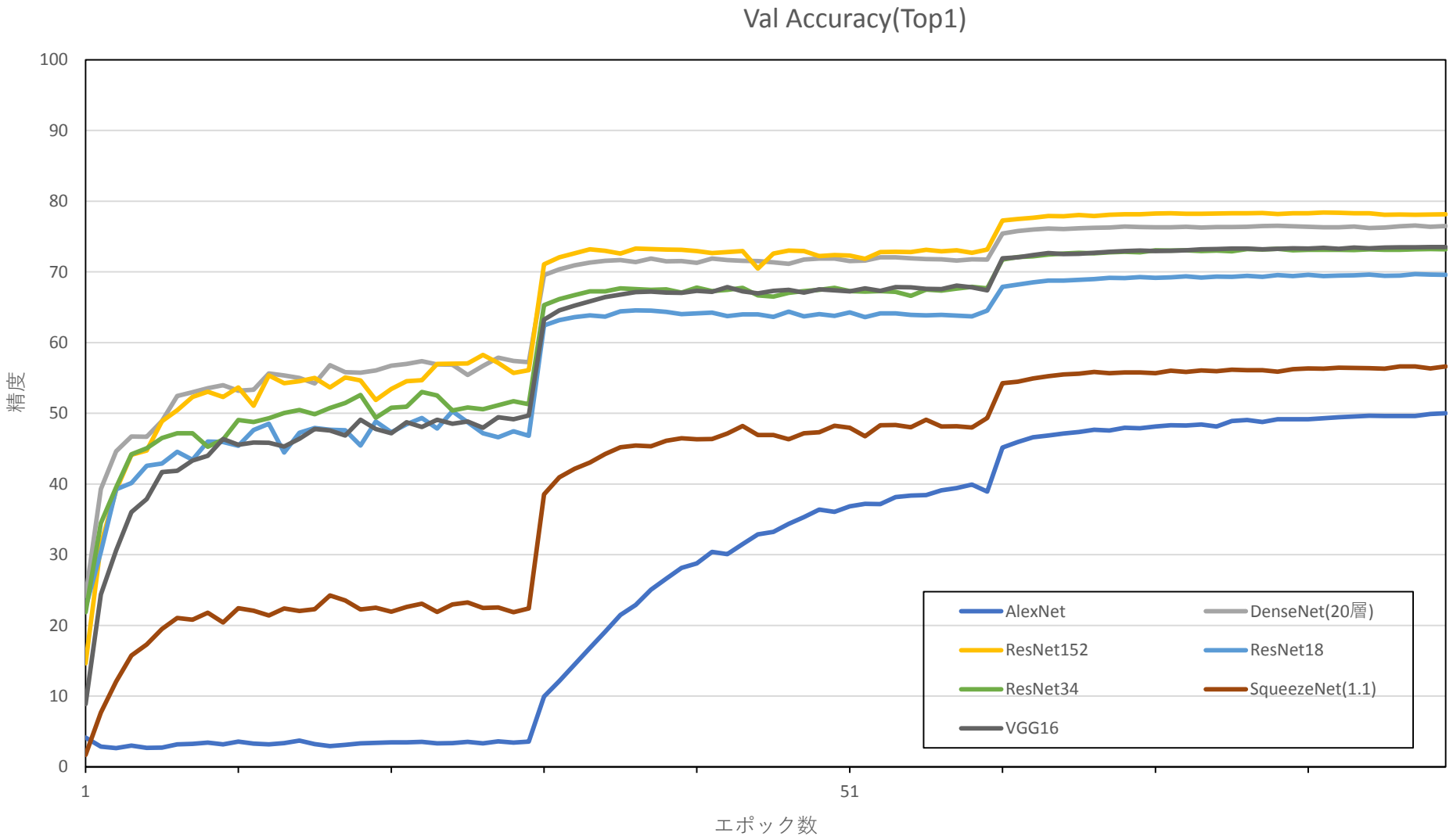
正則化手法	エラー率[%]		学習時間[h]
	CIFAR10	CIFAR100	
ResNet101	5.3(6.5)	25.9	13.2
Stochastic Depth	5.2	23.5	12
Shake-Shake	3.0	18.4	77.8
Swapout	6.0	26.8	42.0
ResNet101+cutout	3.5	25.0	13.1
Shake-Shake+cutout	2.7	17.5	76.5

Imagenetでの比較（学習誤差）

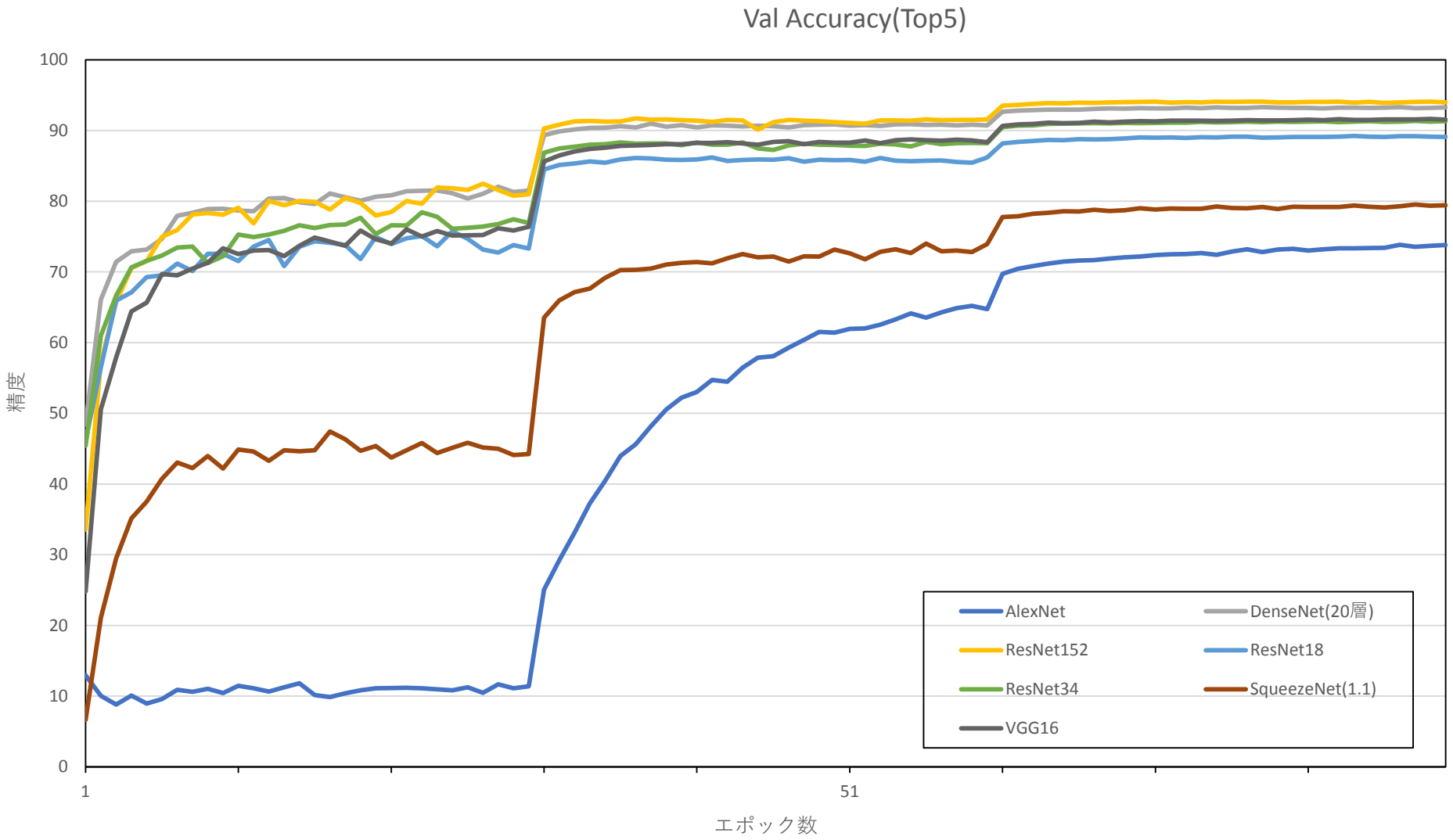
ResNet系のアーキテクチャは誤差が低い



Imagenetでの比較(TOP1)



Imagenetでの比較(TOP5)



Imagenetでの評価

アーキテクチャ名	Top1エラー率[%]	Top5エラー率[%]	パラメータ数[M]
AlexNet	49.9	26.2(16.4)	62.0
VGG16	26.5	8.4(7.3)	138.0
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

まとめ（アーキテクチャサーベイより）

- 主にアーキテクチャの観点から
  - ILSVRC歴代の優勝モデルからCNNの歴史を振り返る
  - 近年のCNNをアプローチをもとに分類し、解説
- 全体的なアーキテクチャとしてはほとんどResNet
  - DenseNet以外に独自の全体的なアーキテクチャの成功例がない
- Residualモジュールの改良は、次元削減とsparseな畳み込みを組合せているものがほとんどと言える
  - 畳み込みのパラメータを削減し、その分深さや幅を大きくすることで精度を向上する
- データおよびモデル内の正則化は他の手法と組合せられる上に効果が大きく期待できる
  - 学習時は外乱を加えるために複雑なネットワークだが、推論時はそれらを統合したシンプルなネットワークにするとかができないか？
- 富豪じゃなくてもできるアーキテクチャの自動設計に期待

まとめ（比較実験より）

- 幅をもたせたアーキテクチャの精度は高い
- 正則化として，Shake-Shakeは効果高い
- cutoutはシンプルでさらに精度向上に効く

今日話せなかったこと

- セグメンテーションや物体検出ならではのアーキテクチャ
- 高速化手法