第6回
y=ax+bから始める
初心者向けML講座

これまでのあらすじ

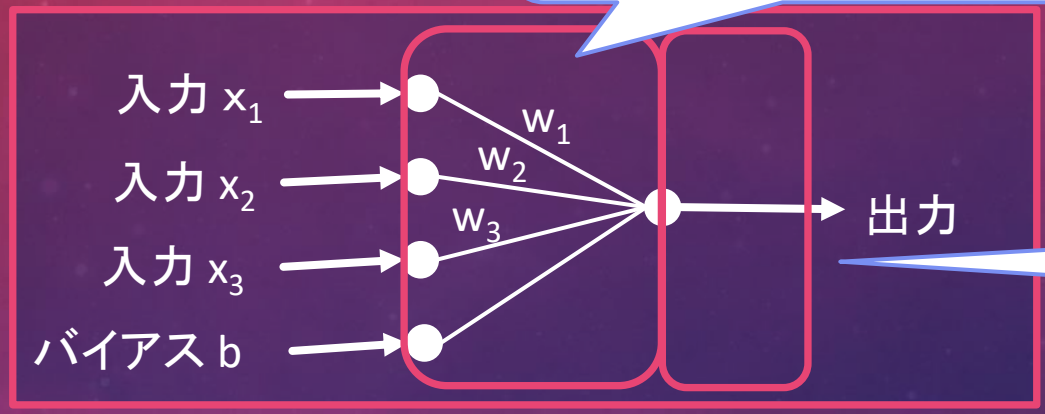
畳み込み処理の計算方法

ML Shukai

入力値×重み
を全て足す。
※重みwとbが学習するパラメータ

パーセプトロン

第4回の話だよ



活性化関数

畳み込み処理

入力X

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

×

フィルターF

1	1	1
2	0	1
−1	−1	0

＋ バイアスb 3 ＝ 出力Y

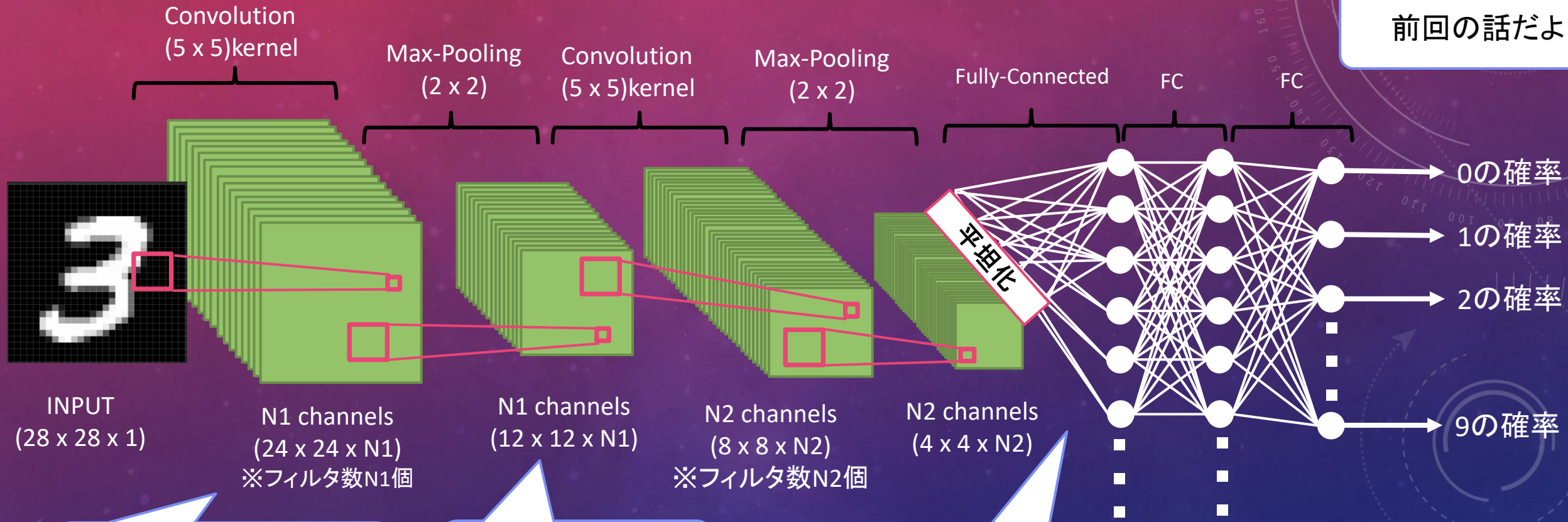
7	11
23	27

入力値×フィルター
を全て足す。
※フィルターとbが学習するパラメータ

出力Yを活性化関数に通す

最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。



前回の話だよ

畳み込みで
フィルタ数分の
チャンネル数

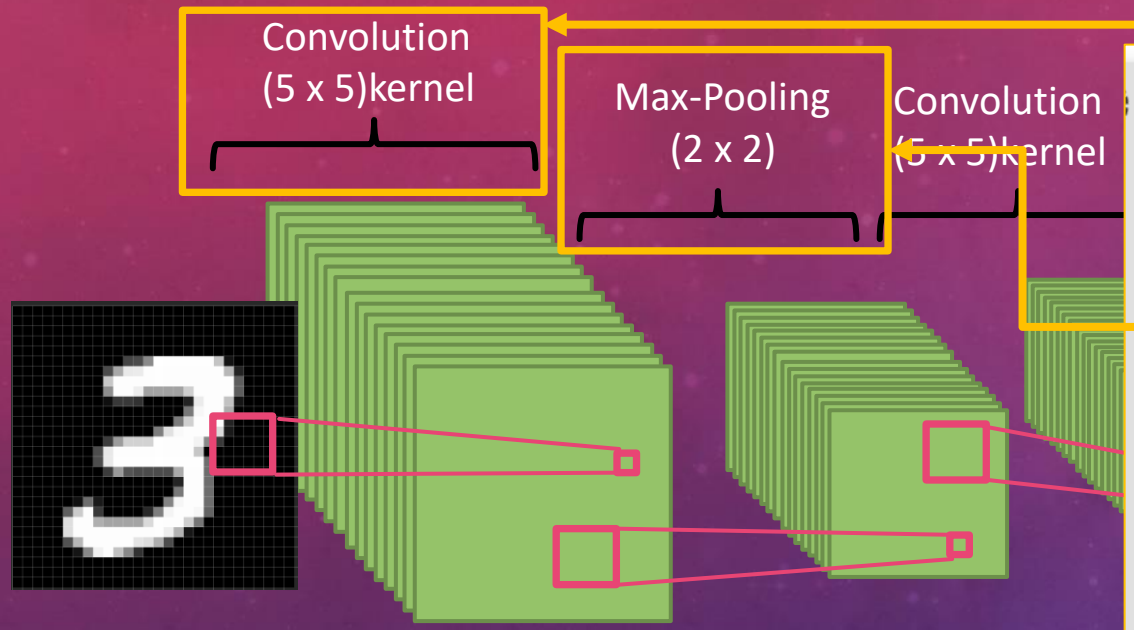
プーリングは
チャンネル数は
変わらない

平坦化 : (Flattend)
3次元の行列を1次元に
する事

最終的なCNN(Convolutional Neural Network)

ML Shukai

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。




```python
model = Sequential()
'''　畳み込み処理　フィルター数64個　カーネル5x5 入力データは28x28の1チャンネルの画像　'''
model.add(Conv2D(filters=64, kernel_size=(5,5), input_shape=(28,28,1), name='conv1'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool1'))

'''　畳み込み処理　フィルター数128　カーネル5x5 '''
model.add(Conv2D(filters=128, kernel_size=(5,5), name='conv2'))
model.add(Activation('relu')) #活性化関数はReLU関数

'''　2x2のフィルターでマックスプーリング　'''
model.add(MaxPooling2D(pool_size=(2,2), name='max_pool2'))

model.add(Flatten(name='flatten')) #平坦化処理
model.add(Dense(units=2048, activation='relu', name='fc1'))#2048個のノードの全結合
model.add(Dense(units=1024, activation='relu', name='fc2'))#1024個のノードの全結合
'''最終層は10個のノードで活性化関数はsoftmax '''
.add(Dense(units=num_classes, activation='softmax', name='predictions'))
.summary()
```

N1＝64

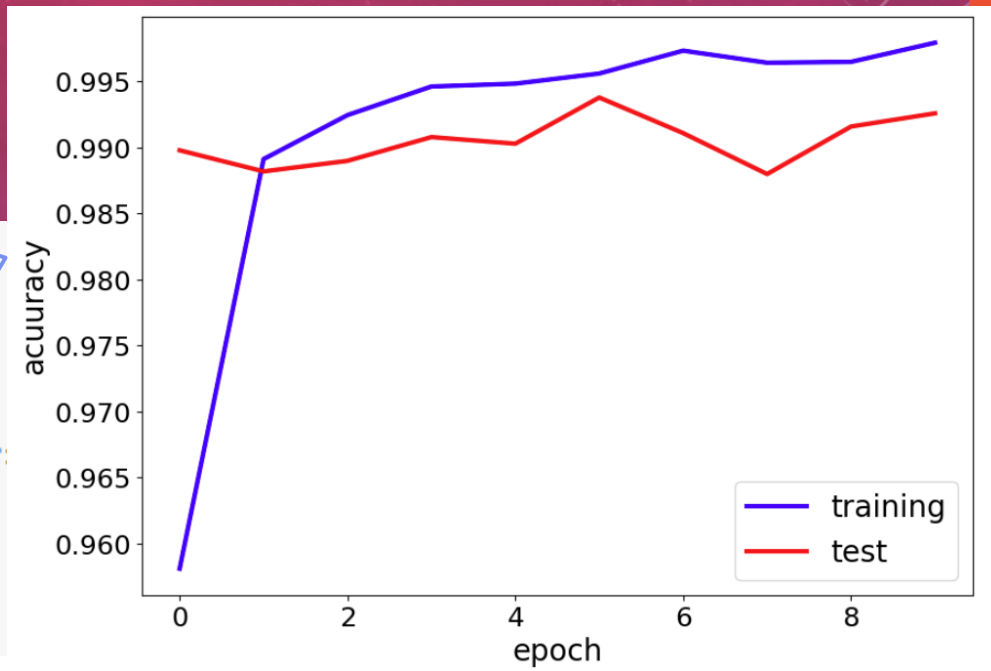
24x24x64
のデータになっているよ

12x12x64
のデータになっているよ

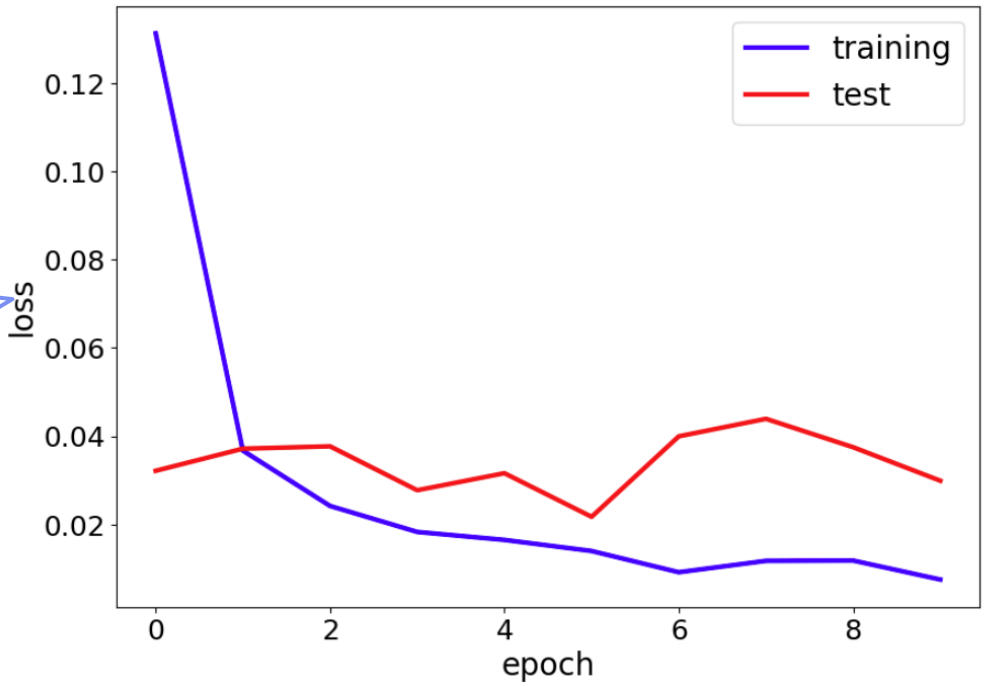
結果を可視化

```
'''結果の可視化'
plt.figure(figsi
plt.plot(history.
plt.plot(history.
plt.tick_params(labelsize=18)
plt.ylabel('acuuracy', fontsize=20)
plt.xlabel('epoch', fontsize=20)
plt.legend(['training', 'test'], loc='best', fo
plt.figure(figsize=(10,7))
plt.plot(history.history['loss'], color='b', l
plt.plot(history.history['val_loss'], color='r'
plt.tick_pa
plt.ylabel
plt.xlabel( epoch , fontsize=20)
plt.legend(['training', 'test'], loc='best', fo
plt.show()
```

正解率だよ！
青が学習データ
赤がテストデータ



損失関数の数値の遷移だよ！



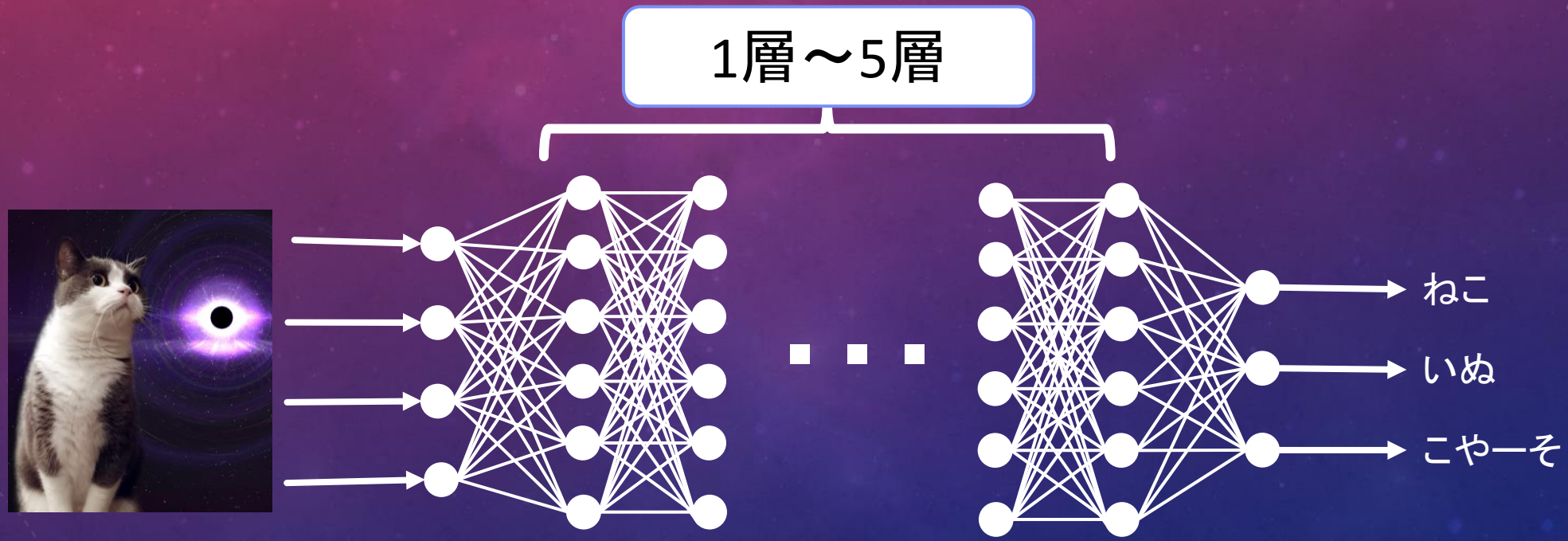
ニューラルネットワーク(NN)深層化その１

単純なCNNやDNNの限界　－　浅いNN

- 余り層が深くないCNNやDNNであれば学習データもそれほど必要としない。

　　→ただし、精度はあまり高くできないことが多い。

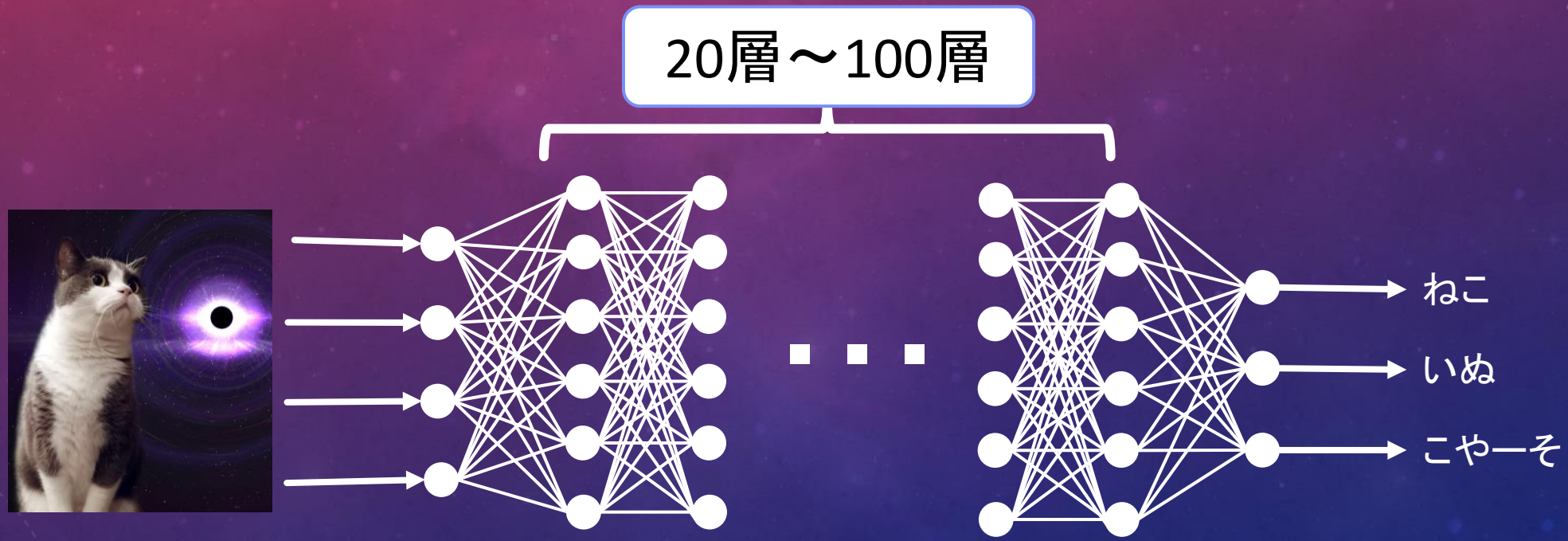
(精度が高くなるなら単純な機械学習でも精度は高くなりあまり意味がない)



単純なCNNやDNNの限界 － 深いNN

- 逆に層が深いCNNやDNNだと複雑なタスクに対応できるようになり、精度は高くなる可能性も高まるが、膨大な学習データや学習時間が必要となる。

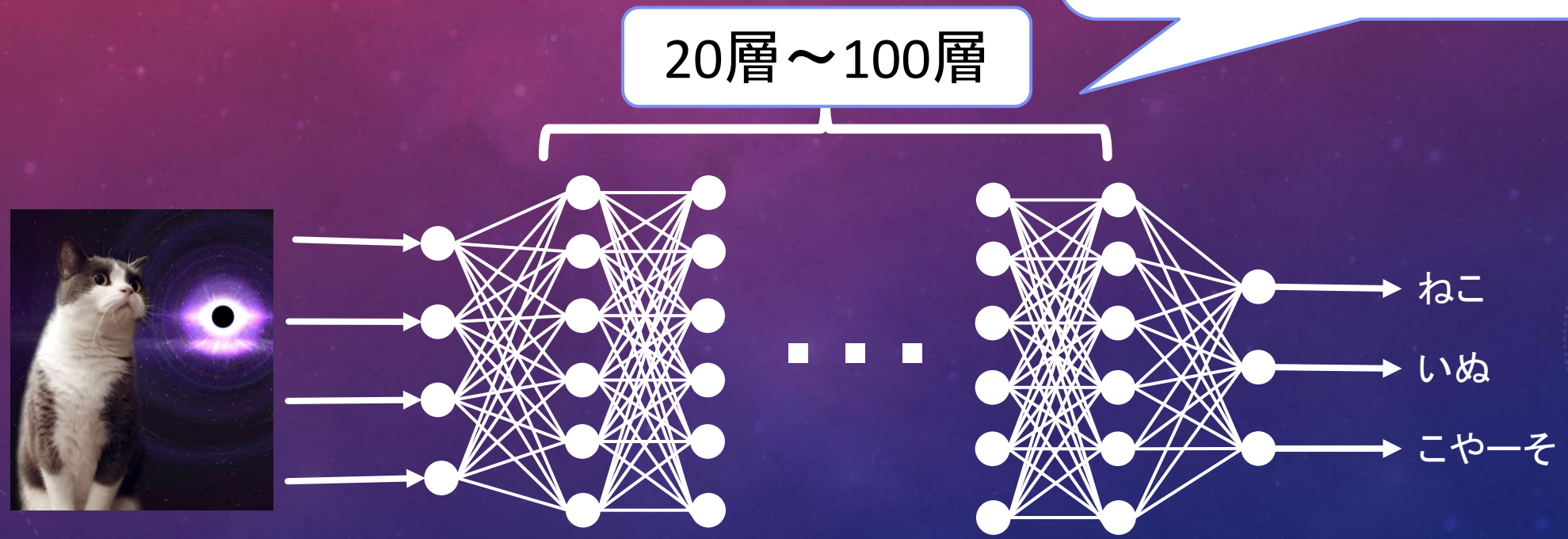
  →そもそも誤差が小さくならず学習が進まなくなってしまうケースも。



単純なCNNやDNNの限界 － 深いNN

- 逆に層が深いCNNやDNNだと複雑なタスクに対応できるようになり、精度は高くなる可能性も高まるが、膨大な学習データ
  →そもそも誤差が小さくならず学習が進まな

20層以上となると
おそらく高確率で
学習は進まなくなる

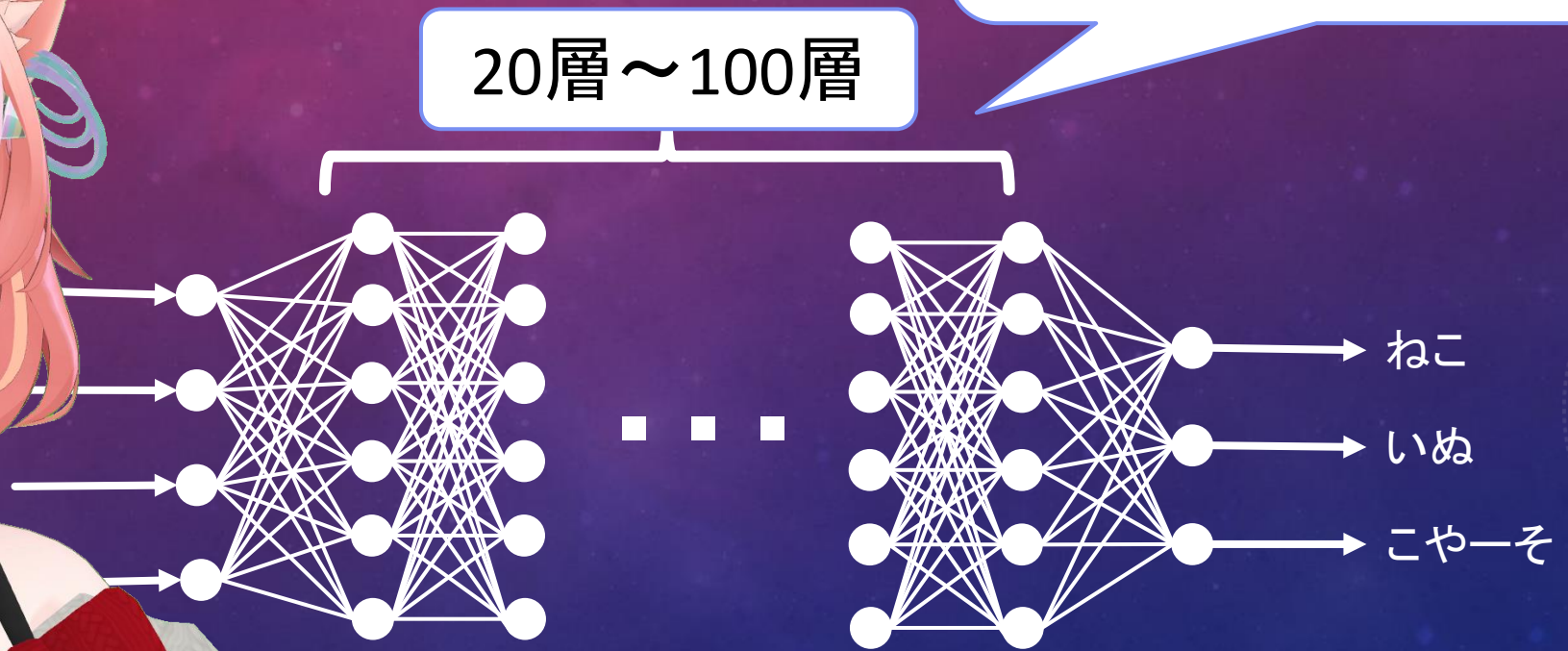


単純なCNNやDNNの限界 － 深いNN

- 逆に[illegible]ごと複雑なタスクに対応できるようになり、精度は高く[illegible]膨大な学習データ[illegible]
  →そもそも誤差が小さくならず学習が進まな[illegible]

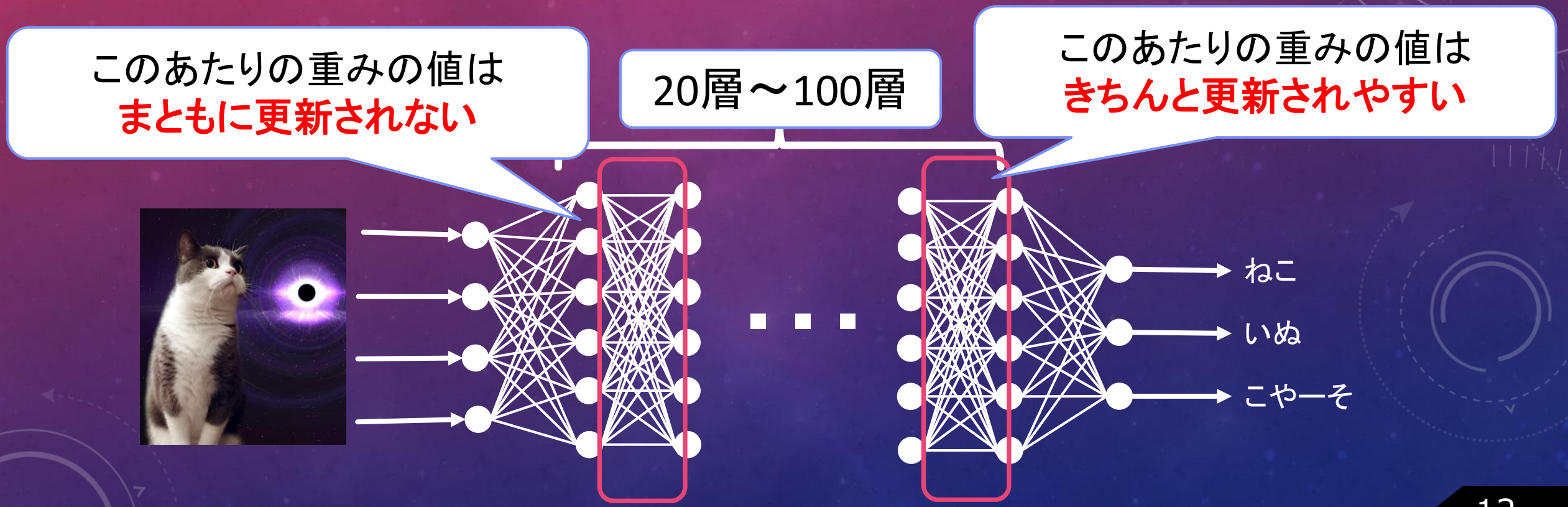
マジで！？

20層以上となると
おそらく高確率で
学習は進まなくなる



何故、層が深くなると学習が進まないのか？

- 重みのパラメーターの更新は**正解**と**予測値**の誤差を使った微分値(逆伝播)で更新するため出口付近の層の重みから更新されていくため、入り口付近はまともに更新されない。



何故、層が深くなると学習が進まないのか？

予測値の計算

誤差を使った微分値の計算順序

何故、層が深くなると学習が進まないのか？

X^2を微分すると2Xとなるので

層を1つ進む毎に微分値が
2倍されるとすると単純計算で
20層となると2^{20}で**1,048,576倍**
となります。

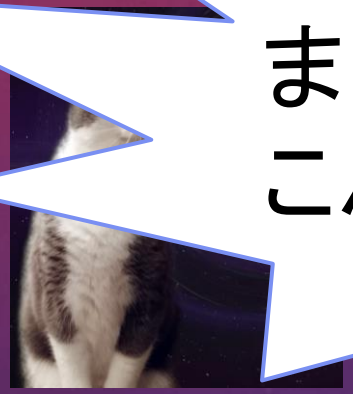
誤差を使った微分値の計算順序

何故、層が深くなると学習が進まないのか？

X^2を微分すると2Xとなるので

層を1つ進む毎に微分値が
るとすると単純計算で
で**1,048,576倍**

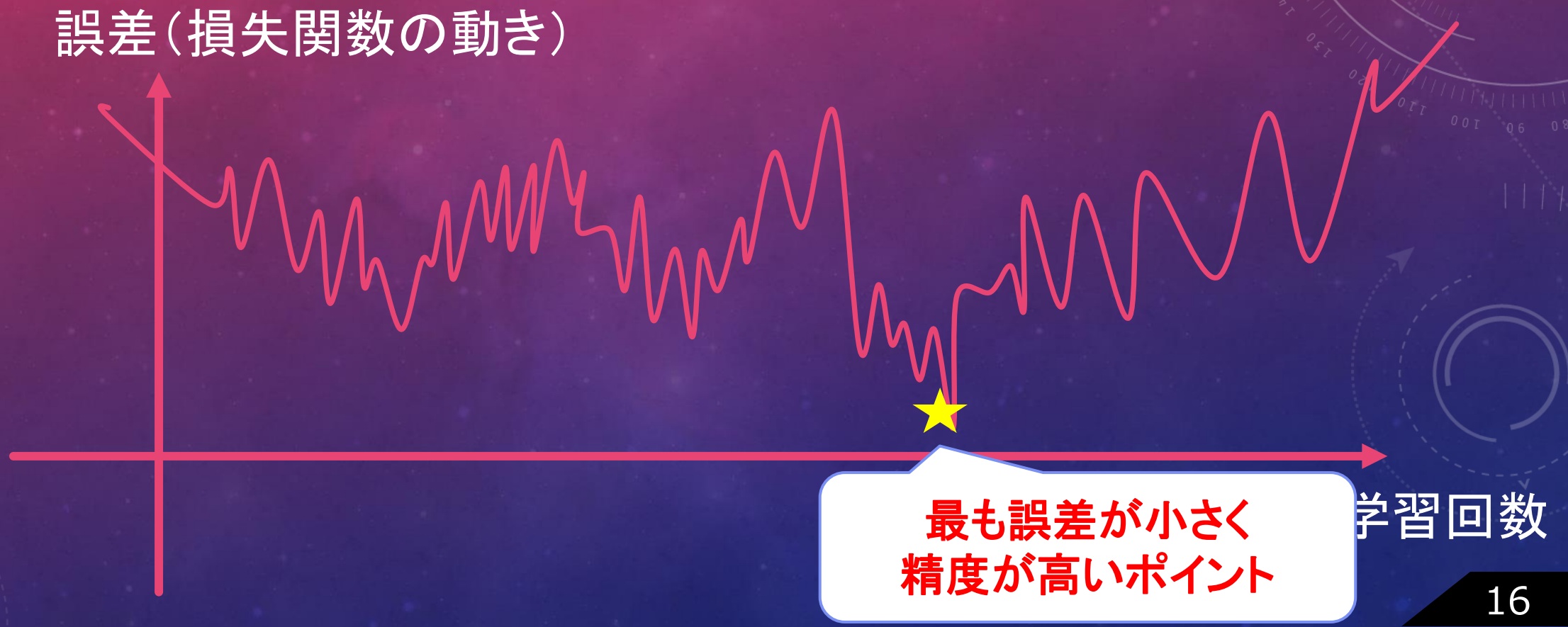
まともに学習できるかっ！
こんなもんっ・・・・！！



誤差を使った微分値の計算順

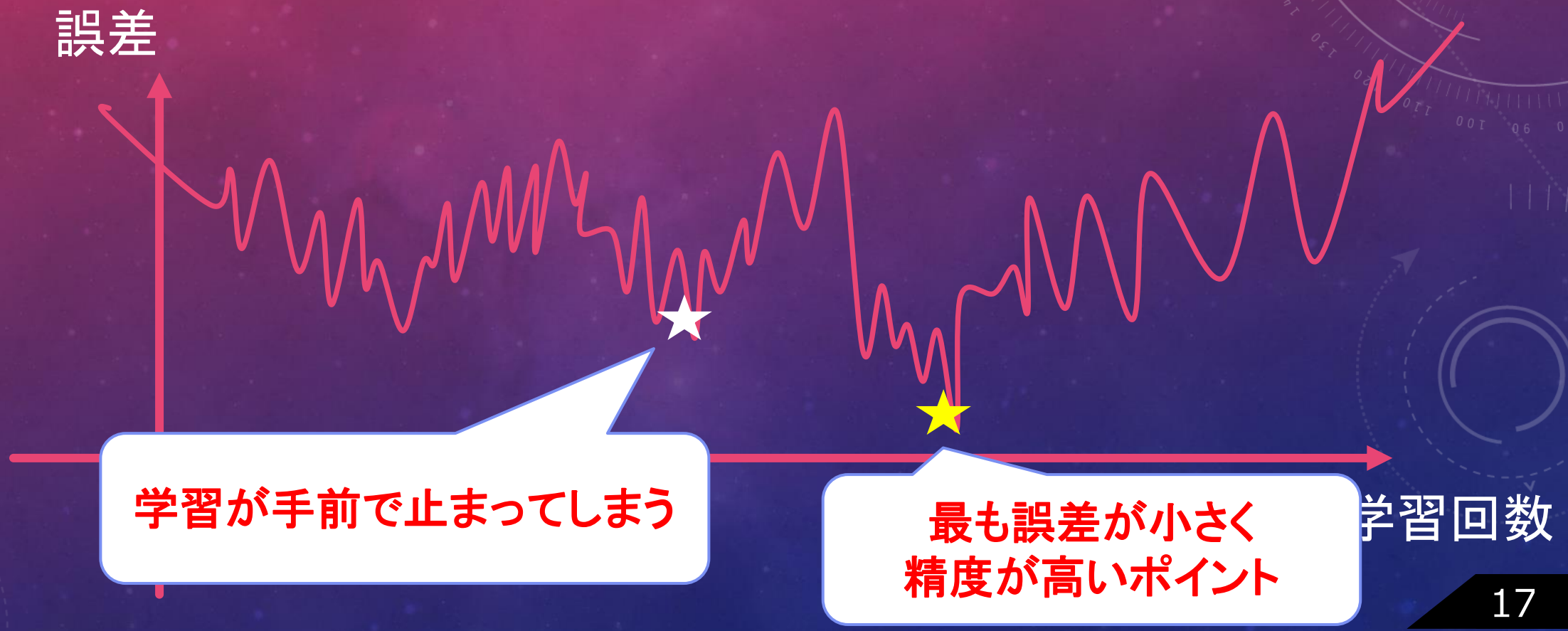
何故、層が深くなると学習が進まないのか？

- NNはあくまでも正解と予測値の誤差が一番小さい場所を探すアルゴリズム
- 誤差の曲線が複雑になり局所解に陥りやすくなる



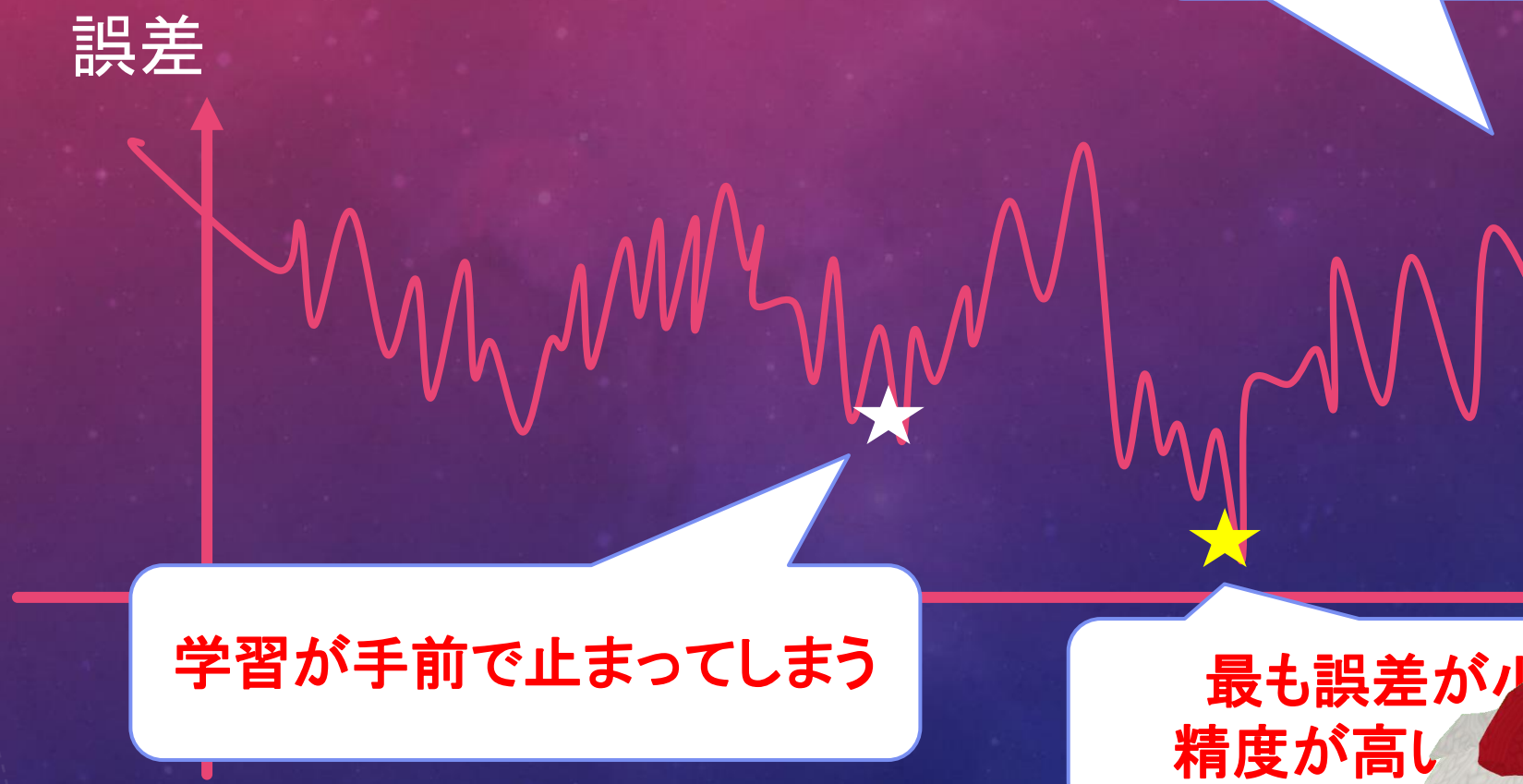
何故、層が深くなると学習が進まないのか？

- NNはあくまでも正解と予測値の誤差が一番小さい場所を探すアルゴリズム
- 誤差の曲線が複雑になり局所解に陥りやすくなる



何故、層が深くなると学習が進まないのか？

- NNはあくまでも正解と予測値の誤差が一番小
- 誤差の曲線が複雑になり局所解に陥りやすくな



CNNやDNNを深いNNにするには？

CNNやDNNを深いNNにするには？

- モデルそのものを大規模深層モデルにする

CNNやDNNを深いNNにするには？

- モデルそのものを大規模深層モデルにする

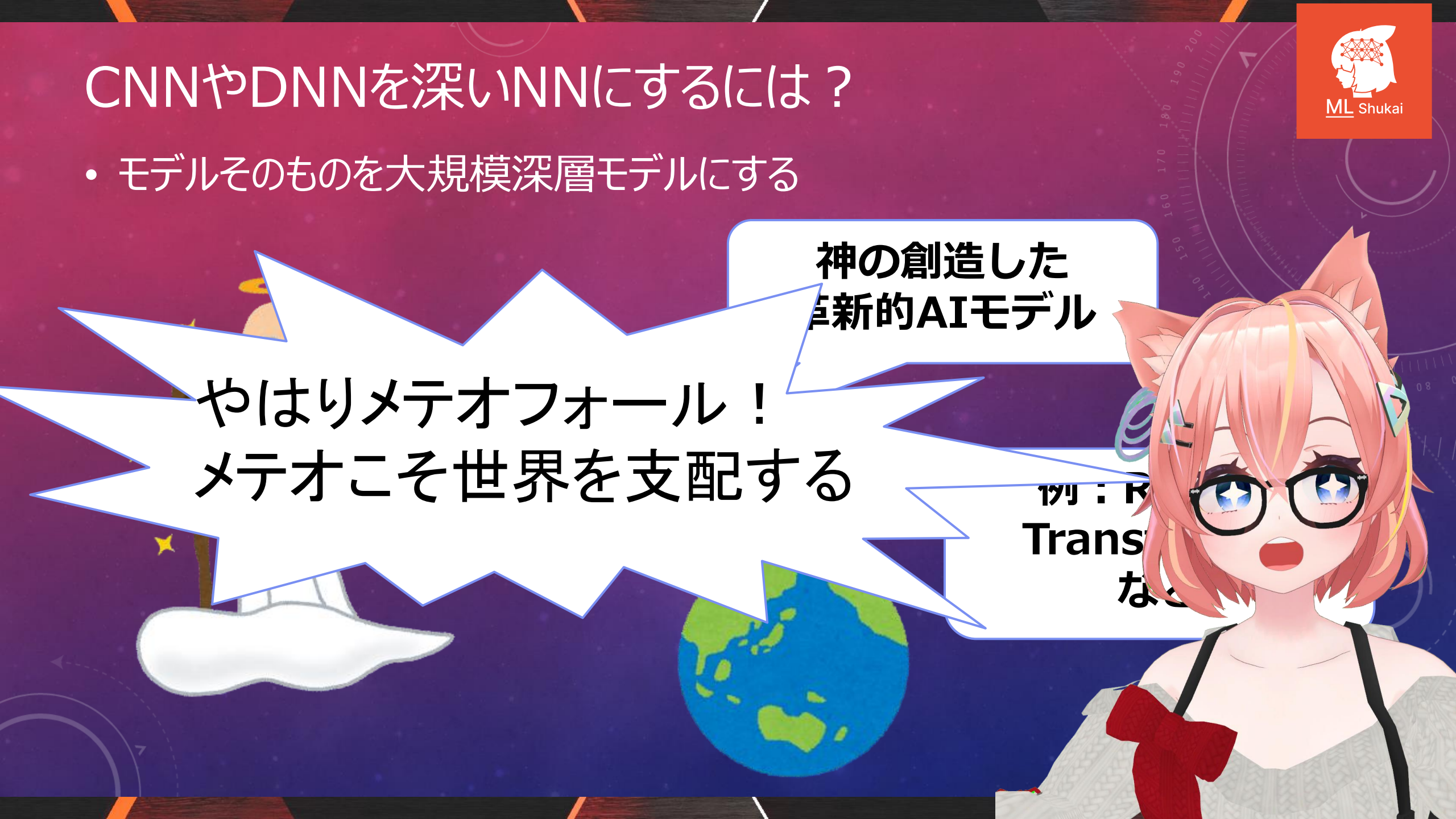
CNNやDNNを深いNNにするには？

- モデルそのものを大規模深層モデルにする

神の創造した
革新的AIモデル

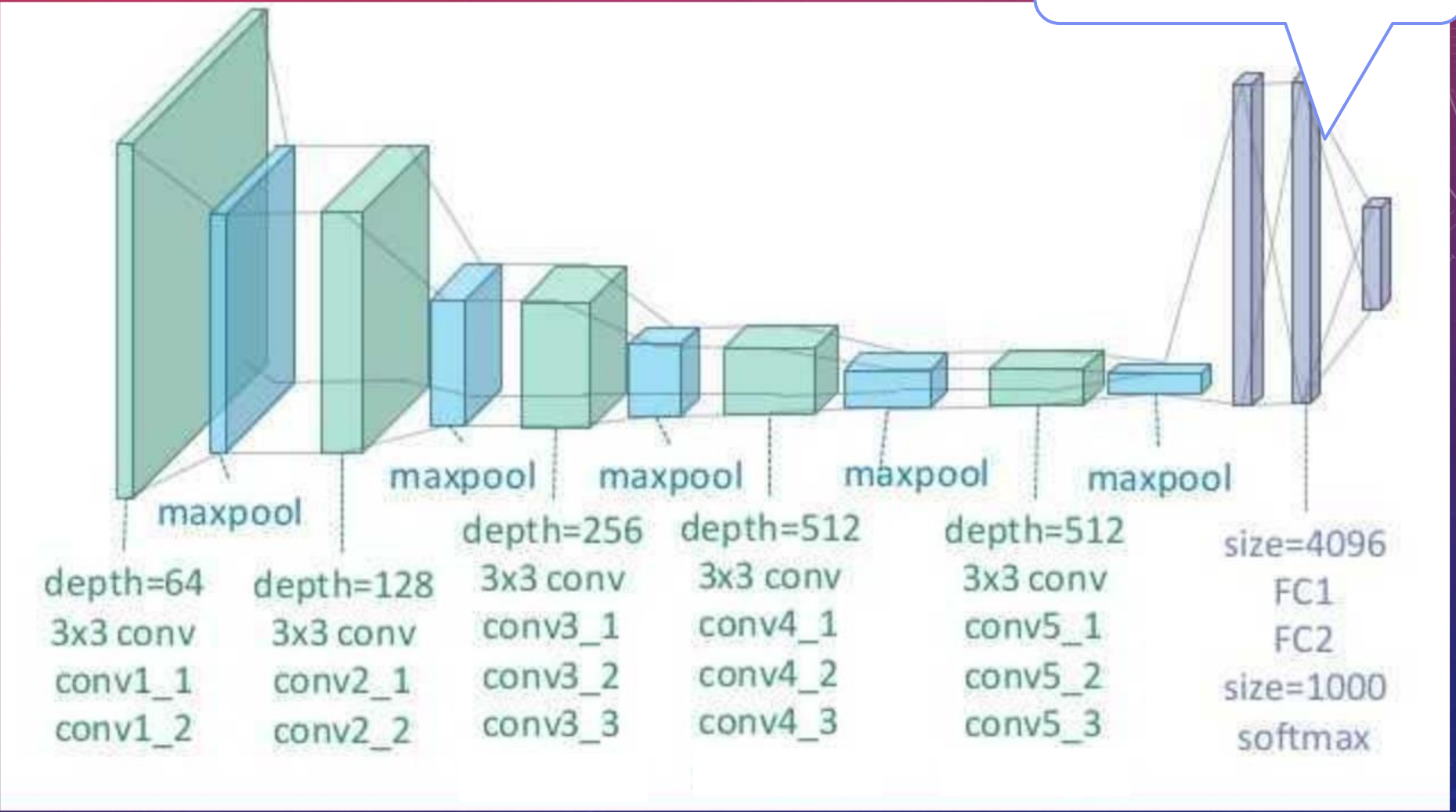
例：R
Trans
な

やはりメテオフォール！
メテオこそ世界を支配する



VGG16（最大でVGG19の19層）

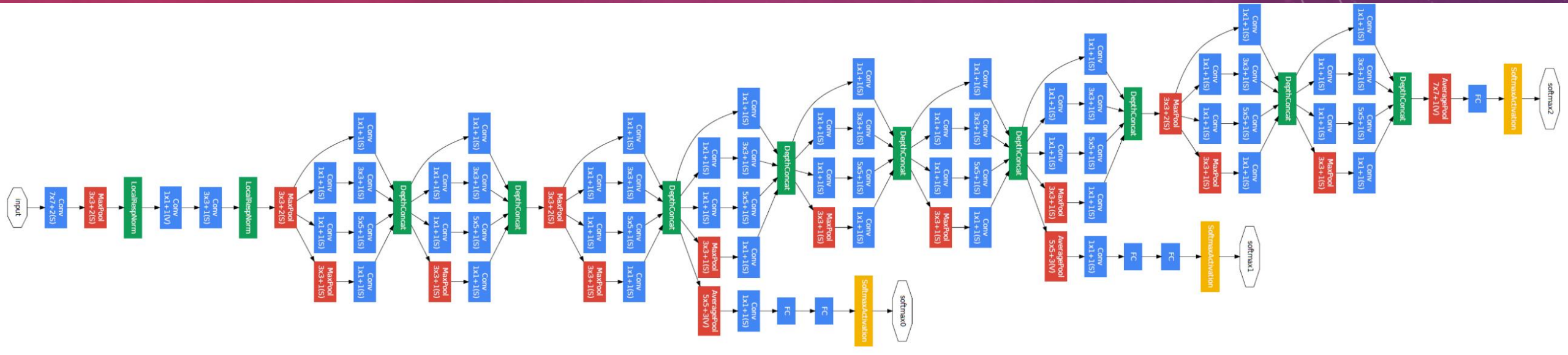
2014年に画像コンペで
2位となったモデル



※引用元：https://github.com/scofield7419/basic_NNs_in_frameworks/blob/master/assets/4.png

限界を超えられた最初のモデル(22層)

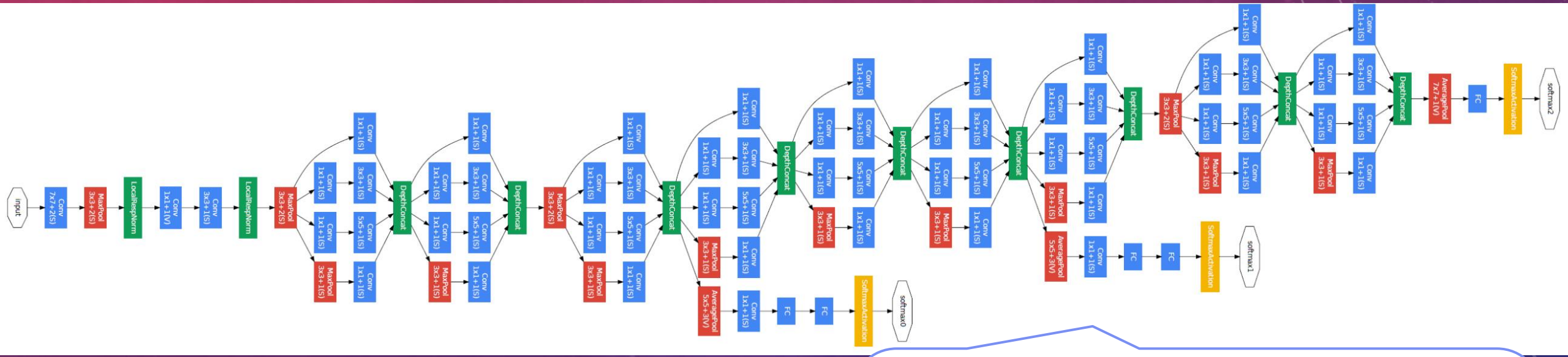
- GoogLeNet(2014年画像コンペ優勝モデル)



出典:[Szegedy,C.etal.,2014]

限界を超えられた最初のモデル(22層)

- GoogLeNet(2014年画像コンペ優勝モデル)



青：Convolution(畳み込み)
赤：Pooling
黄色：Softmax関数
緑:その他

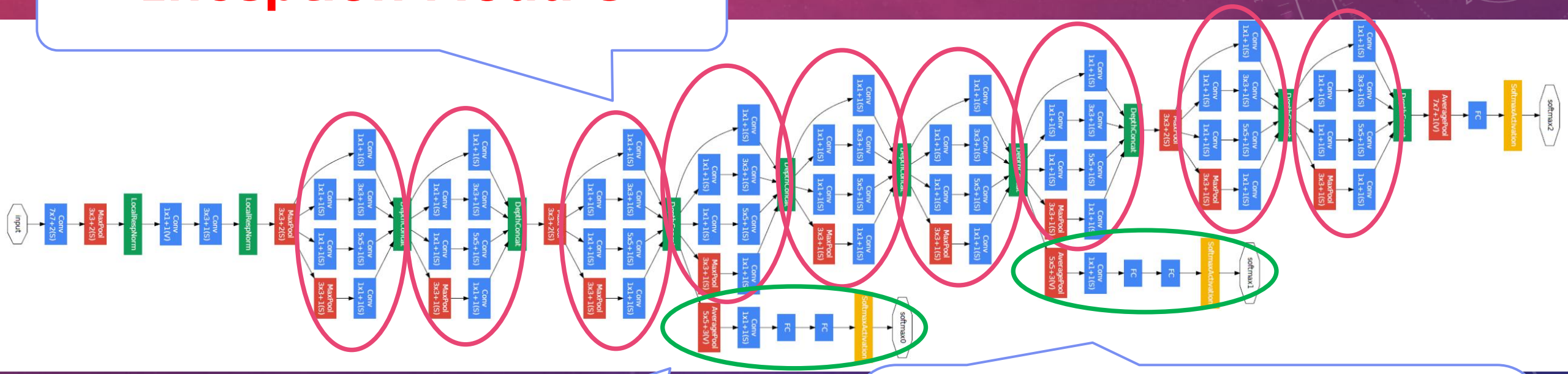
出典:[Szegedy,C.etal.,2014]

深くしても効率的に学習できるようになった最初のモデル(22層)

ML Shukai

Inception Module

コンペ優勝モデル）



Auxillary loss

青：Convolution(畳み込み)

赤：Pooling

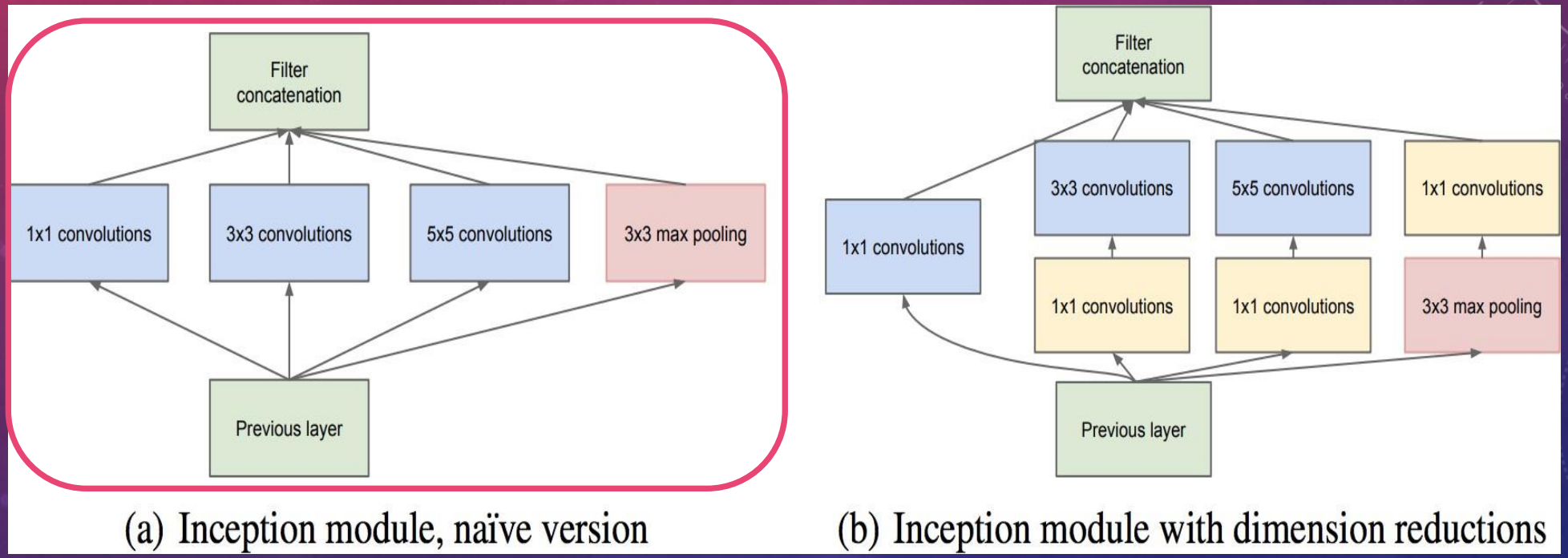
黄色：Softmax関数

緑:その他

出典：[Szegedy,C.etal.,2014]

Inceptionモジュール

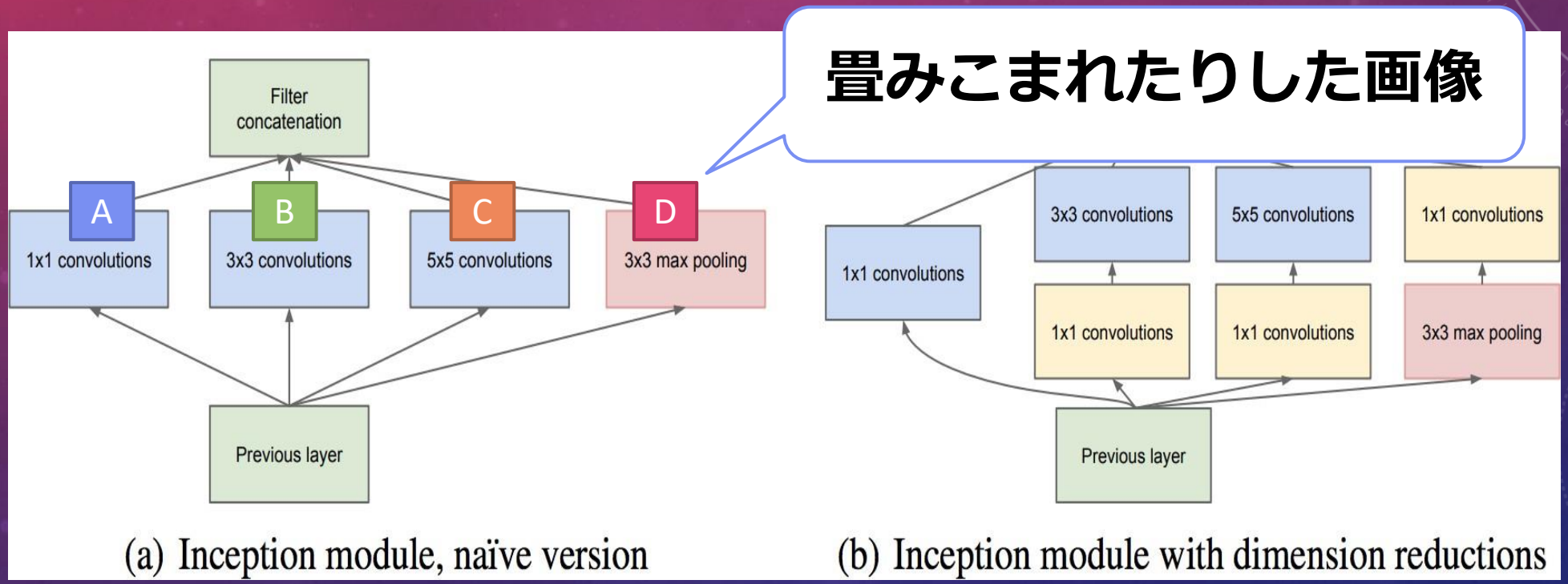
- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する



[Szegedy, C. et al., 2014]

Inceptionモジュール

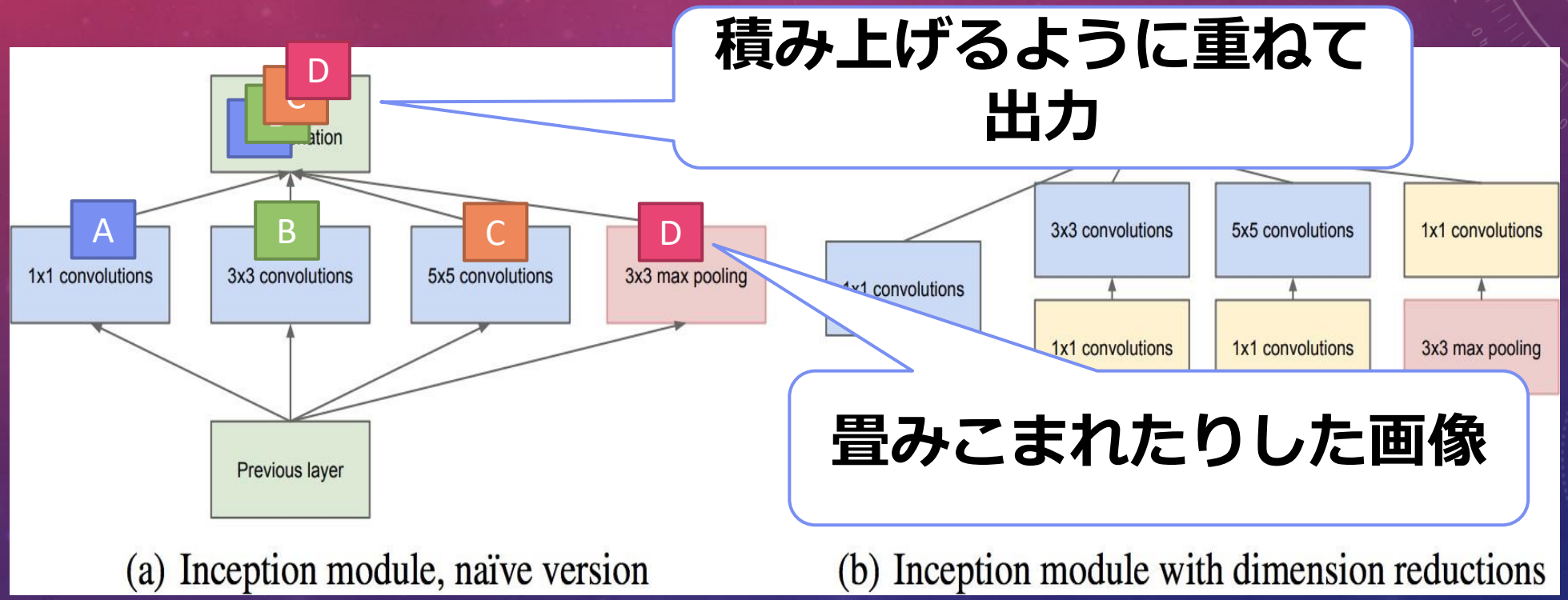
- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する



[Szegedy, C. et al., 2014]

Inceptionモジュール

- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する

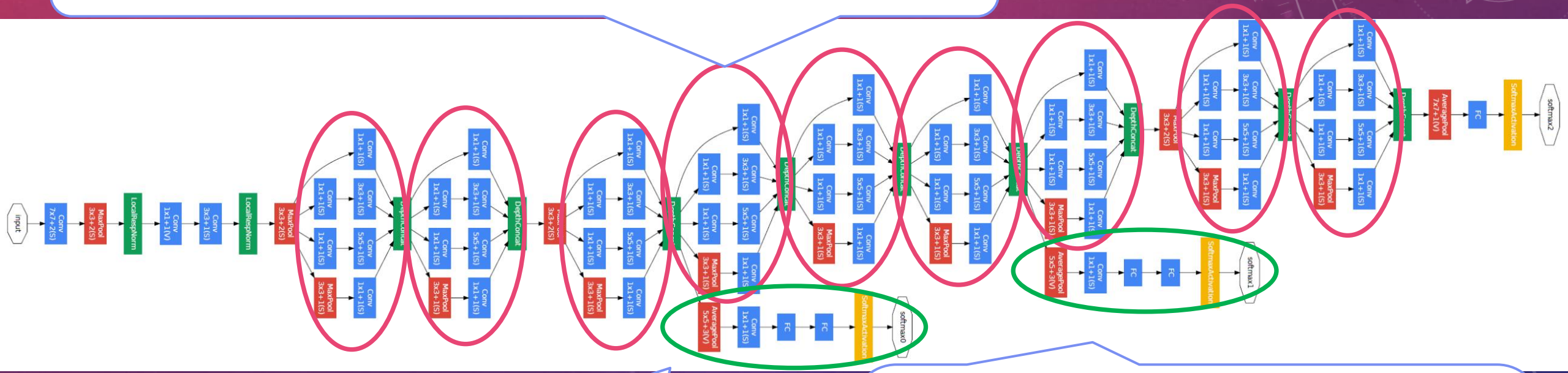


[Szegedy, C. et al., 2014]

深くしても効率的に学習できるようになった最初のモデル(22層)

ML Shukai

Inception Moduleいっぱいある！



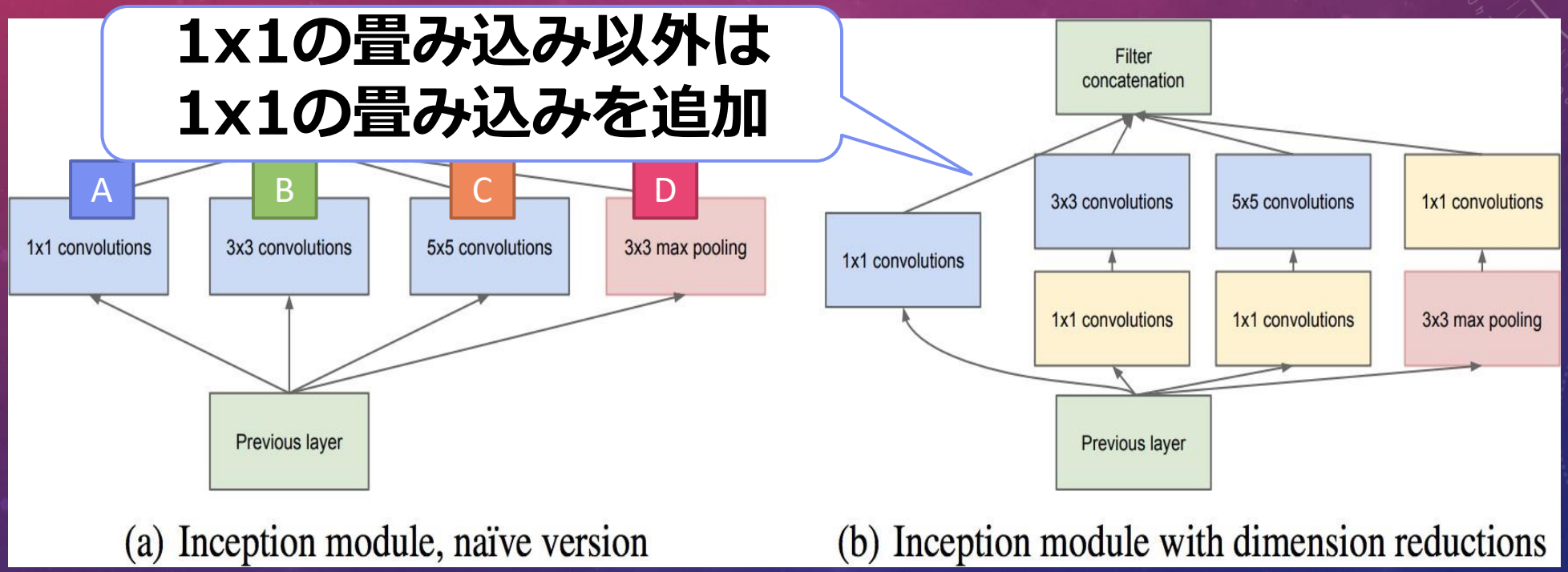
Auxillary loss

青：Convolution(畳み込み)
赤：Pooling
黄色：Softmax関数
緑:その他

出典:[Szegedy,C.etal.,2014]

Inceptionモジュール

- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する



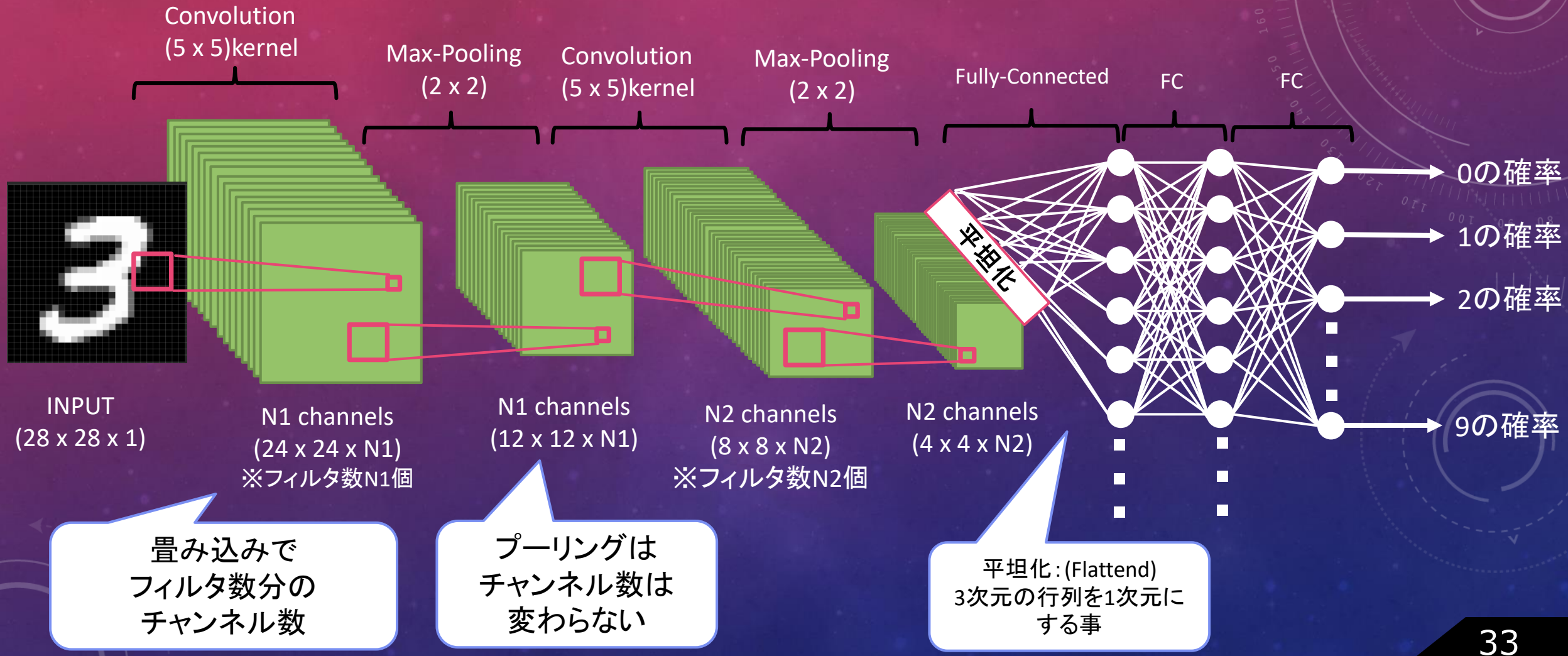
[Szegedy, C. et al., 2014]

ここでポイント

- 思い出してください畳み込みのフィルターの数が出力画像のチャンネル数です

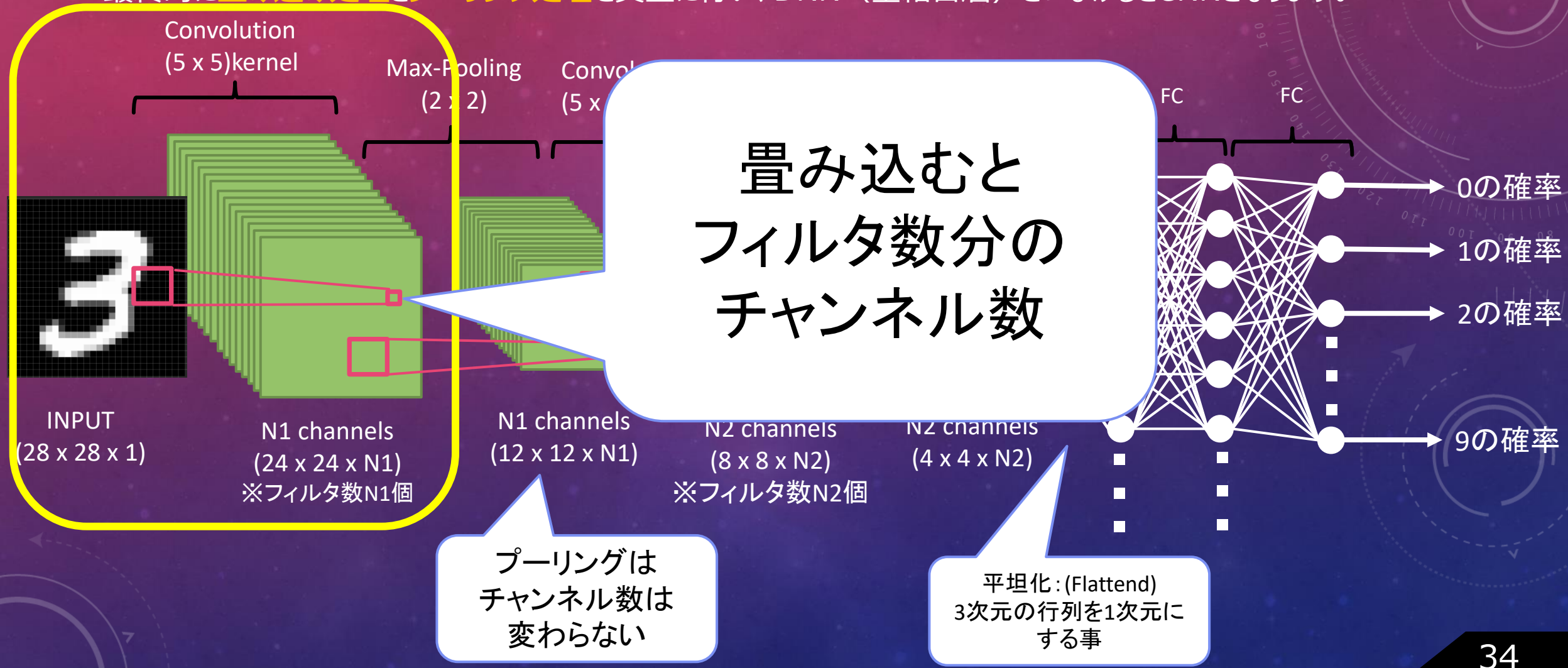
最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。



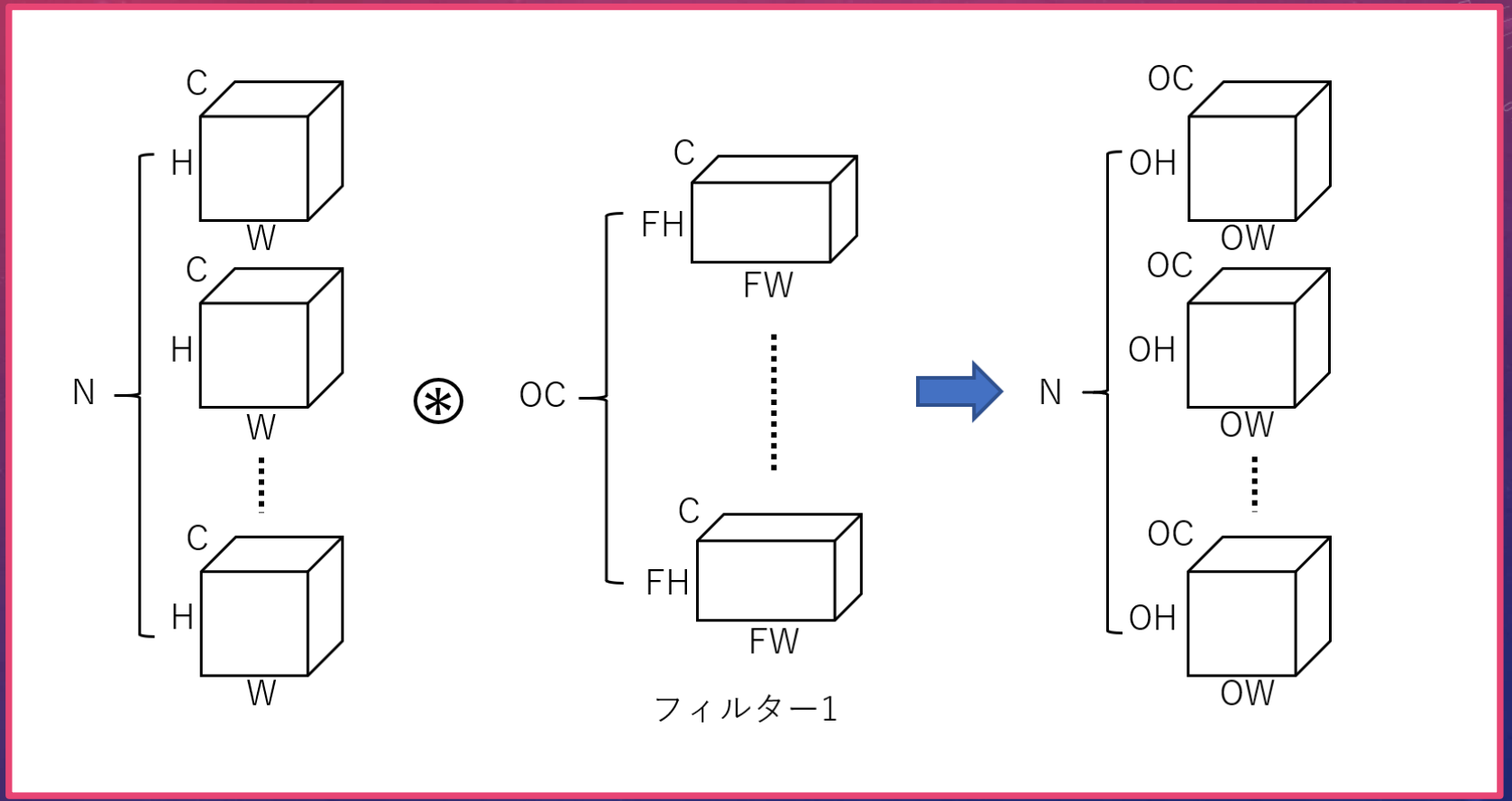
最終的なCNN(Convolutional Neural Network)

- 最終的に**畳み込み処理**と**プーリング処理**を交互に行い、DNN（全結合層）とつなげるとCNNとなります。



畳み込みの処理

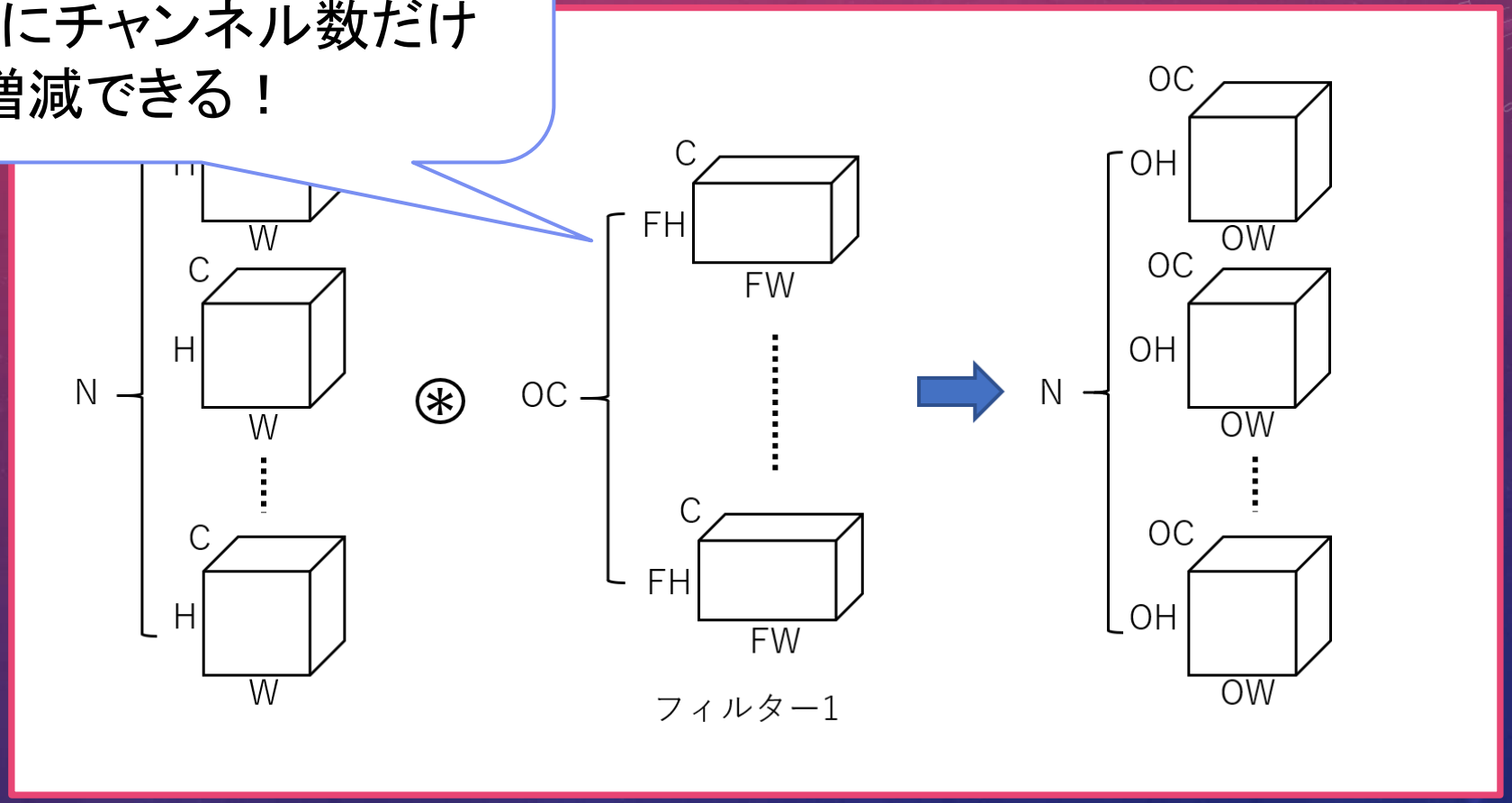
- N＝画像の数、H＝高さ、W＝幅、C＝チャンネル数、**OC＝フィルターの数＝出力画像のチャンネル数**



畳み込みの処理

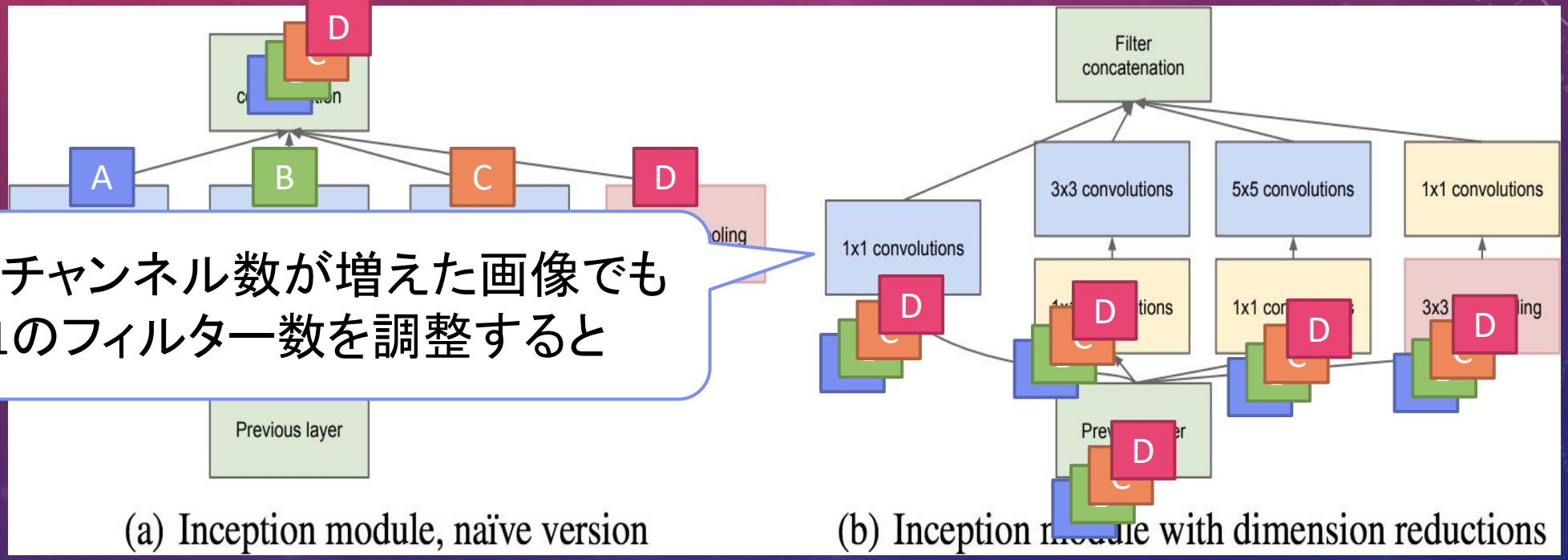
フィルターの高さと幅を
1x1とすると入力画像のサイズ
を変えずにチャンネル数だけ
増減できる！

ンネル数、**OC＝フィルターの数＝出力画像のチャンネル数**



Inceptionモジュール

- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する

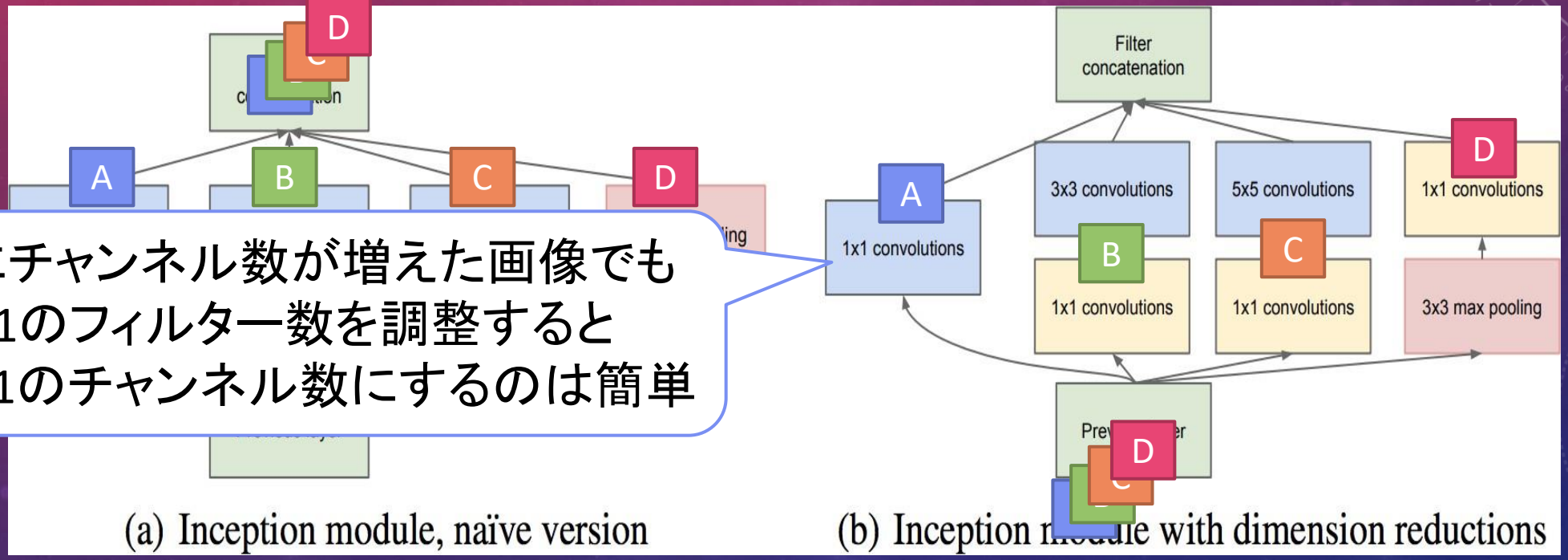


4倍にチャンネル数が増えた画像でも
1x1のフィルター数を調整すると

[Szegedy, C. et al., 2014]

Inceptionモジュール

- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する



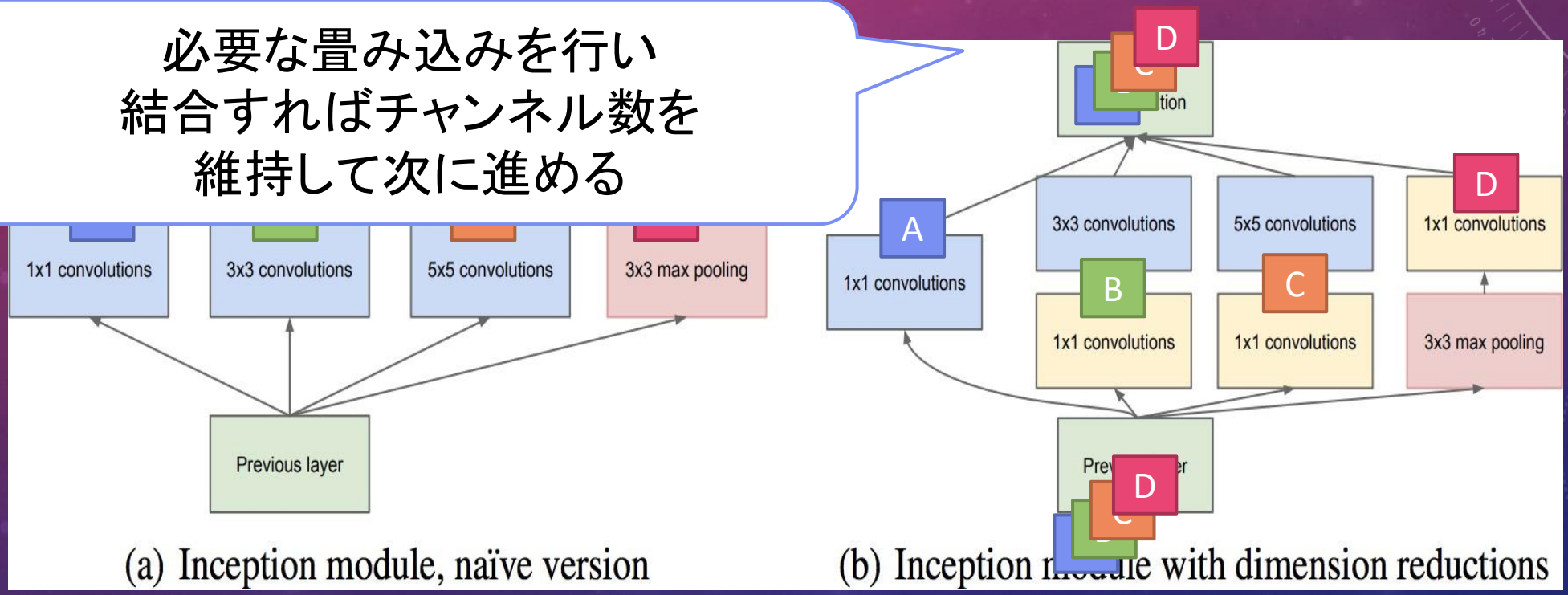
4倍にチャンネル数が増えた画像でも
1x1のフィルター数を調整すると
4分の1のチャンネル数にするのは簡単

[Szegedy, C. et al., 2014]

Inceptionモジュール

- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する

必要な畳み込みを行い
結合すればチャンネル数を
維持して次に進める



(a) Inception module, naïve version

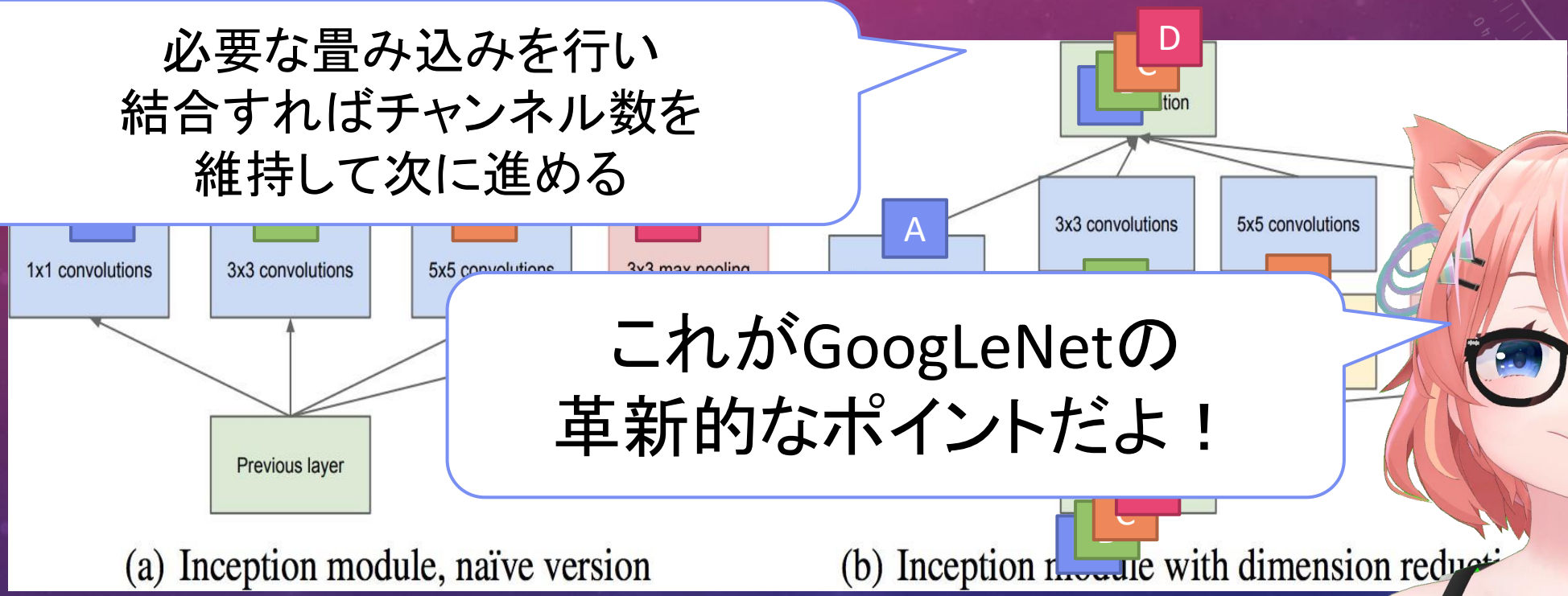
(b) Inception module with dimension reductions

[Szegedy, C. et al., 2014]

Inceptionモジュール

- 1x1と3x3と5x5のフィルターを使った畳み込み、3x3のフィルターを使ったマックスプーリングを別々に行いチャンネル数を積み上げるように結合する

必要な畳み込みを行い
結合すればチャンネル数を
維持して次に進める

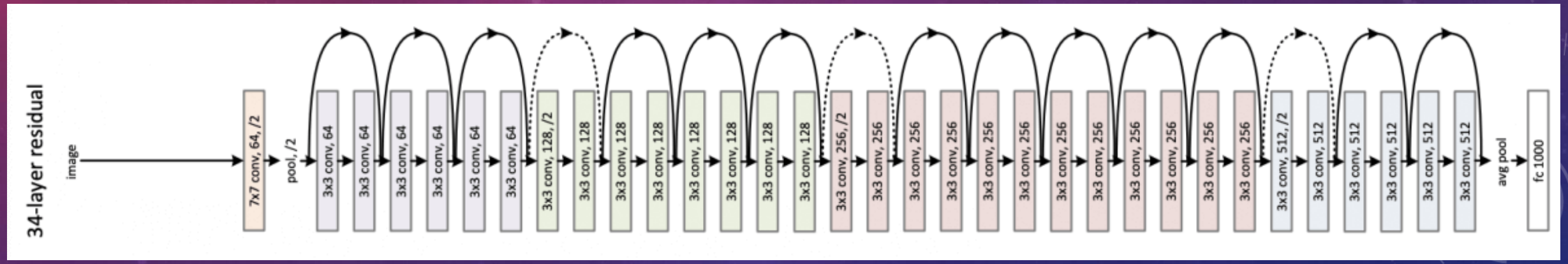


これがGoogLeNetの
革新的なポイントだよ！

[Szegedy, C. et al., 2014]

理論上無限？に繋げられるResNet(2015年登場)

- スキップ接続という処理を追加することによってどれだけ微分しても大丈夫にしている

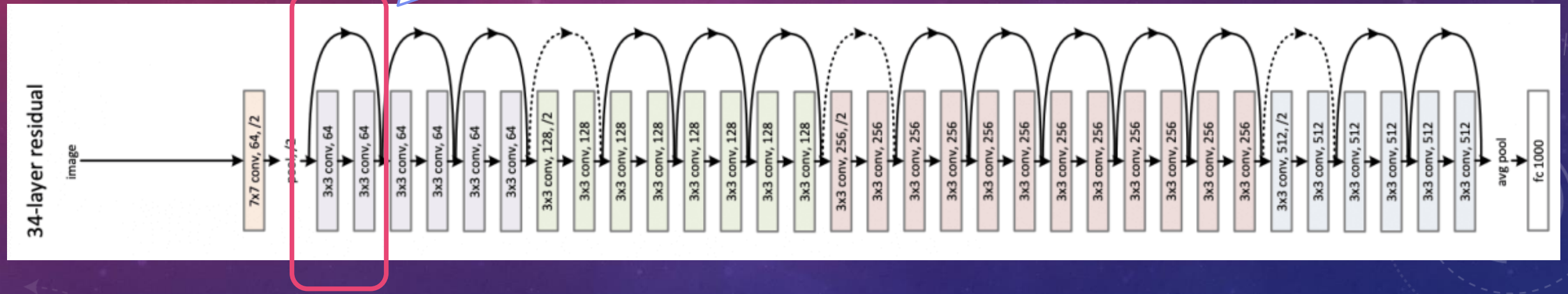


出典：**Deep Residual Learning for Image Recognition**
https://arxiv.org/abs/1512.03385

理論上無限？に繋げられるResNet(2015年登場)

- スキップ接続という処理を追加することによってどれだけ微分しても大丈夫にしている

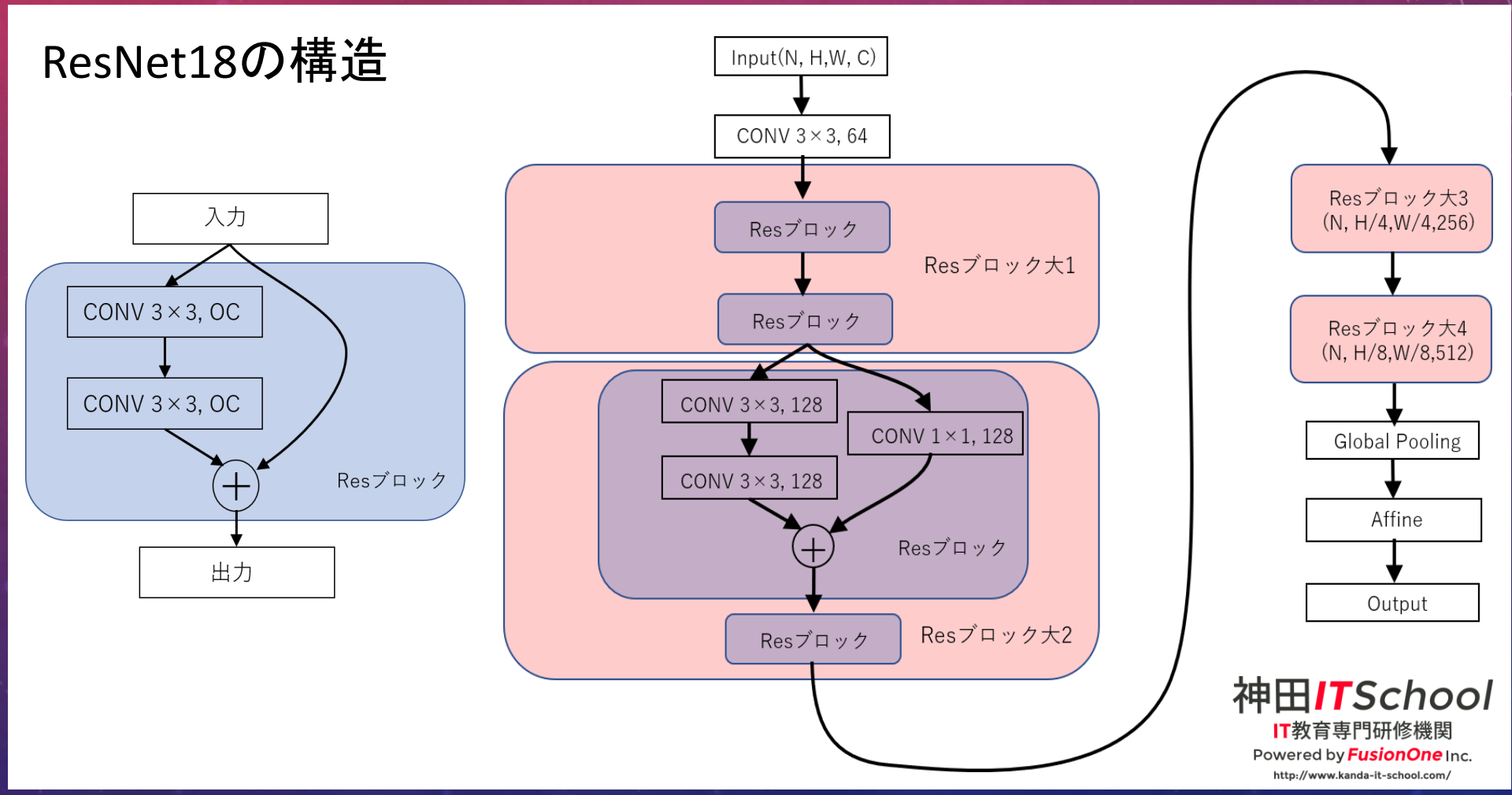
大量のResブロックで構成



出典：**Deep Residual Learning for Image Recognition**
https://arxiv.org/abs/1512.03385

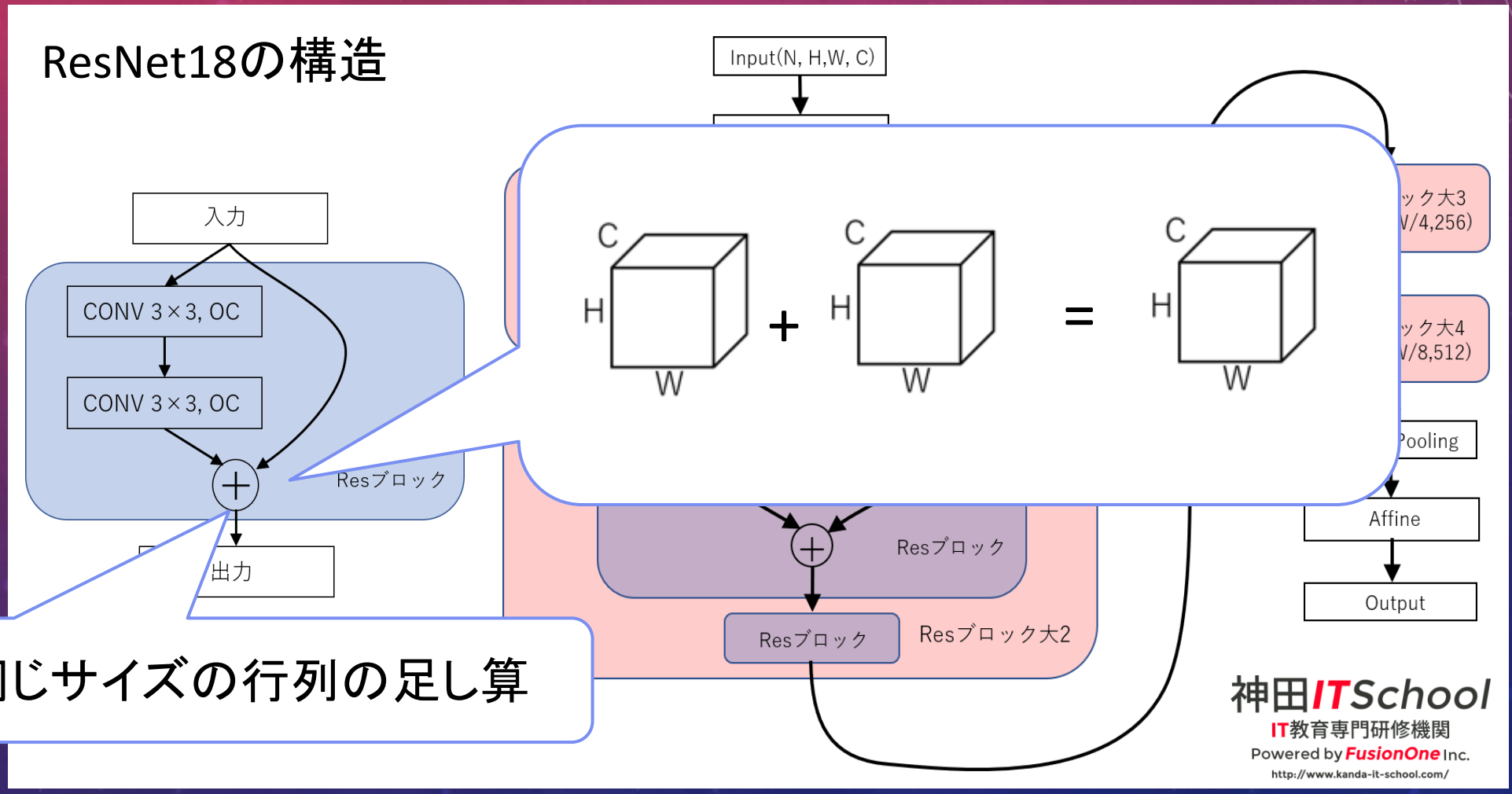
スキップ接続が最大の特徴

- 畳み込みをしたものしないものを足し合わせている（スキップ接続）



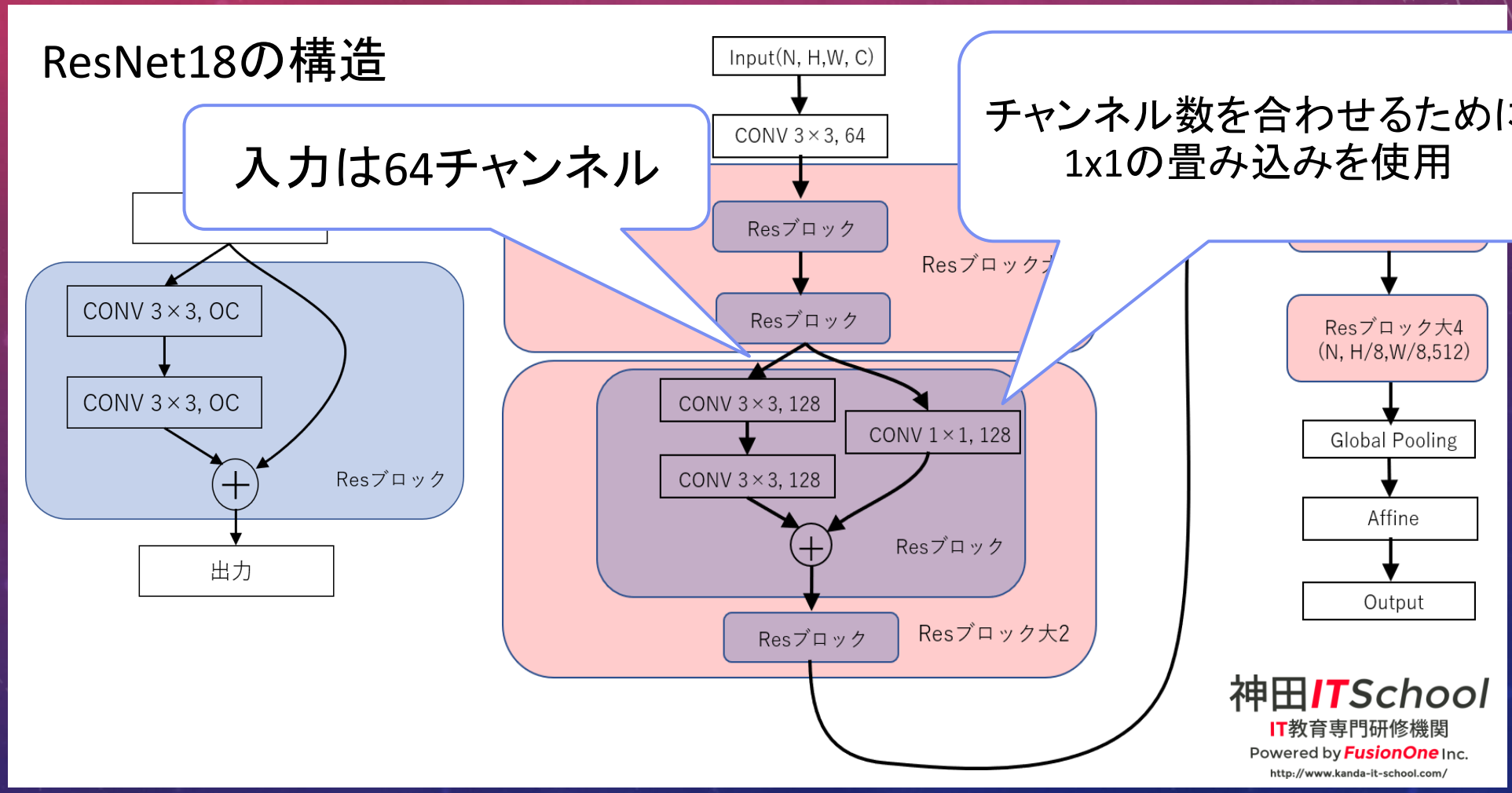
スキップ接続が最大の特徴

- 畳み込みをしたものしないものを足し合わせている（スキップ接続）



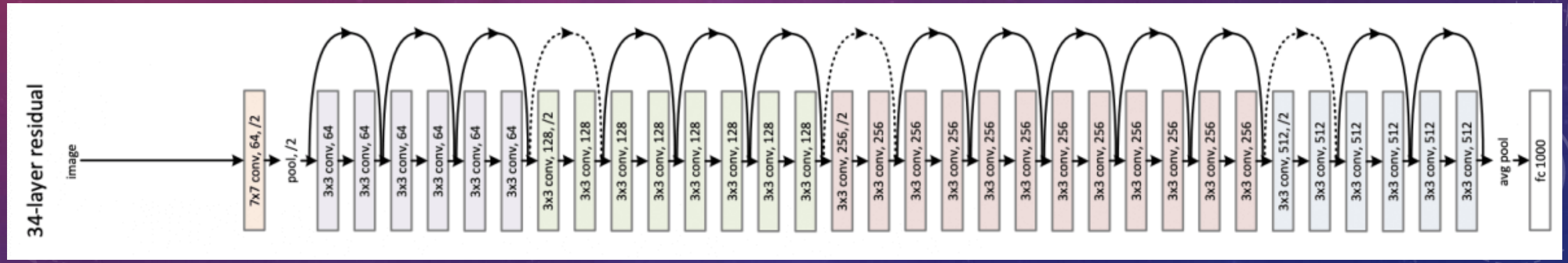
スキップ接続が最大の特徴

- 畳み込みをしたものしないものを足し合わせている（スキップ接続）



理論上無限？に繋げられるResNet(2015年登場)

- 論文上では最大152層繋げて学習している。



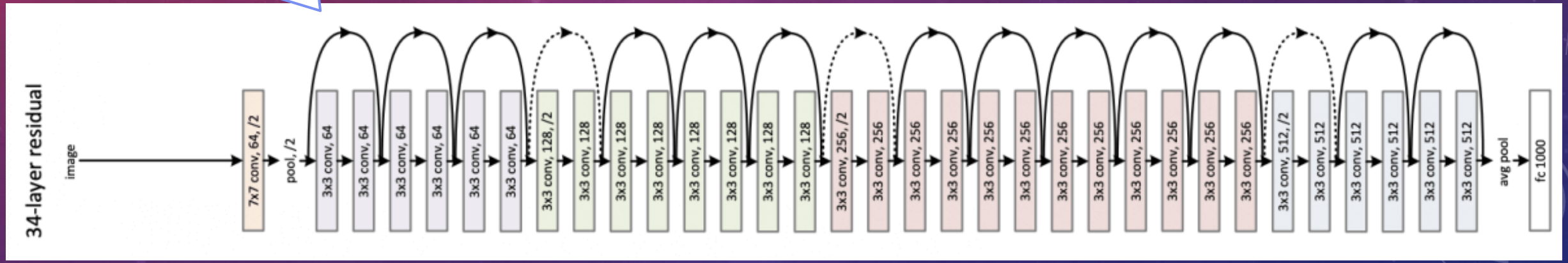
出典：**Deep Residual Learning for Image Recognition**
https://arxiv.org/abs/1512.03385

理論上無限？に繋げられるResNet(2015年登場)

- 論文上では最大152層繋げて学習している。

Transformer等に
つながる革新的モデル

ChatGPT
Stable Diffusion



出典：**Deep Residual Learning for Image Recognition**
https://arxiv.org/abs/1512.03385

理論上無限？に繋げられるResNet(2015年登場)

- 論文上では最大[illegible]層繋げて学[illegible]いる。

Tran[illegible]

[illegible]iffusion

俺のAIか？
欲しけりゃくれてやる！

出典：**Deep Residual Learning for Image Recognition**
https://arxiv.org/abs/1512.03385

理論上無限？に繋げられるResNet(2015年登場)

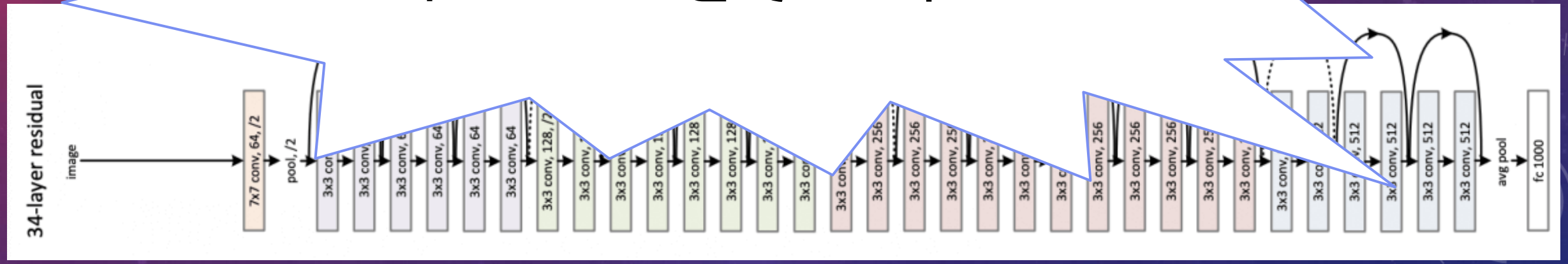
- 論文上では最大[illegible]層繋げて学[illegible]いる。

Tran[illegible]

[illegible]iffusion

探せ！

この世の全てをそこに置いてきた！



出典：**Deep Residual Learning for Image Recognition**
https://arxiv.org/abs/1512.03385

理論上無限？に繋げられるResNet(2015年登場)

- 論文上では最大152層繋げて学習している。

Transformer等に
つながる革新的モデル

ChatGPT
Stable

世はまさに大階層時代！！

出典：**Deep Residual Learning for**
https://arxiv.org/abs/1512.03385

おわり

ありがとうございました

次回はニューラルネットワーク深層化する時に使われる工夫
（予定）