知能工学特別講義 第３講

担当：和田山 正

名古屋工業大学

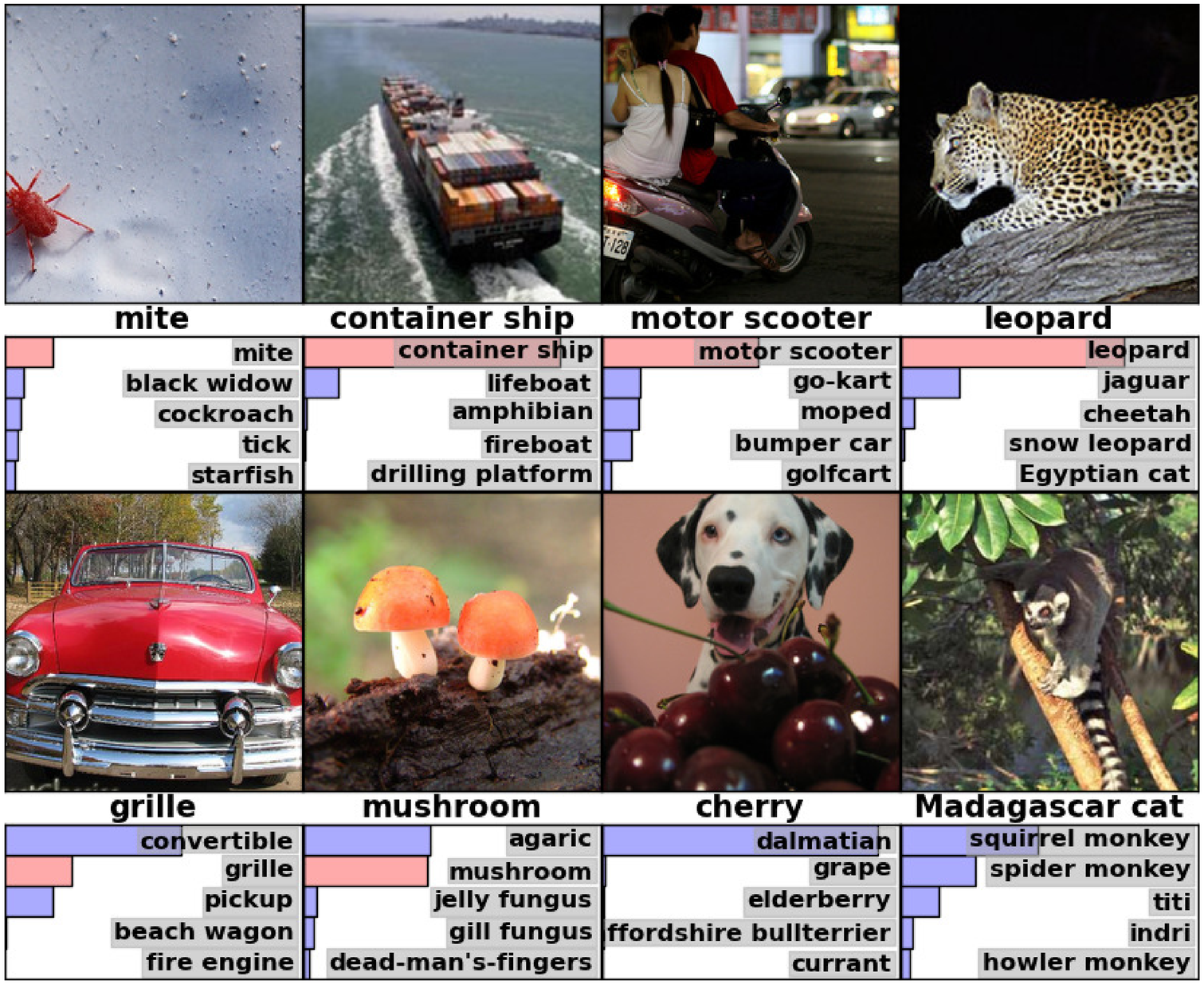
本講義の内容

- Imagenetについて
- MNIST数字認識コードを学ぶ
- 汎化誤差を小さくするための技術
- 畳み込みニューラルネットワーク

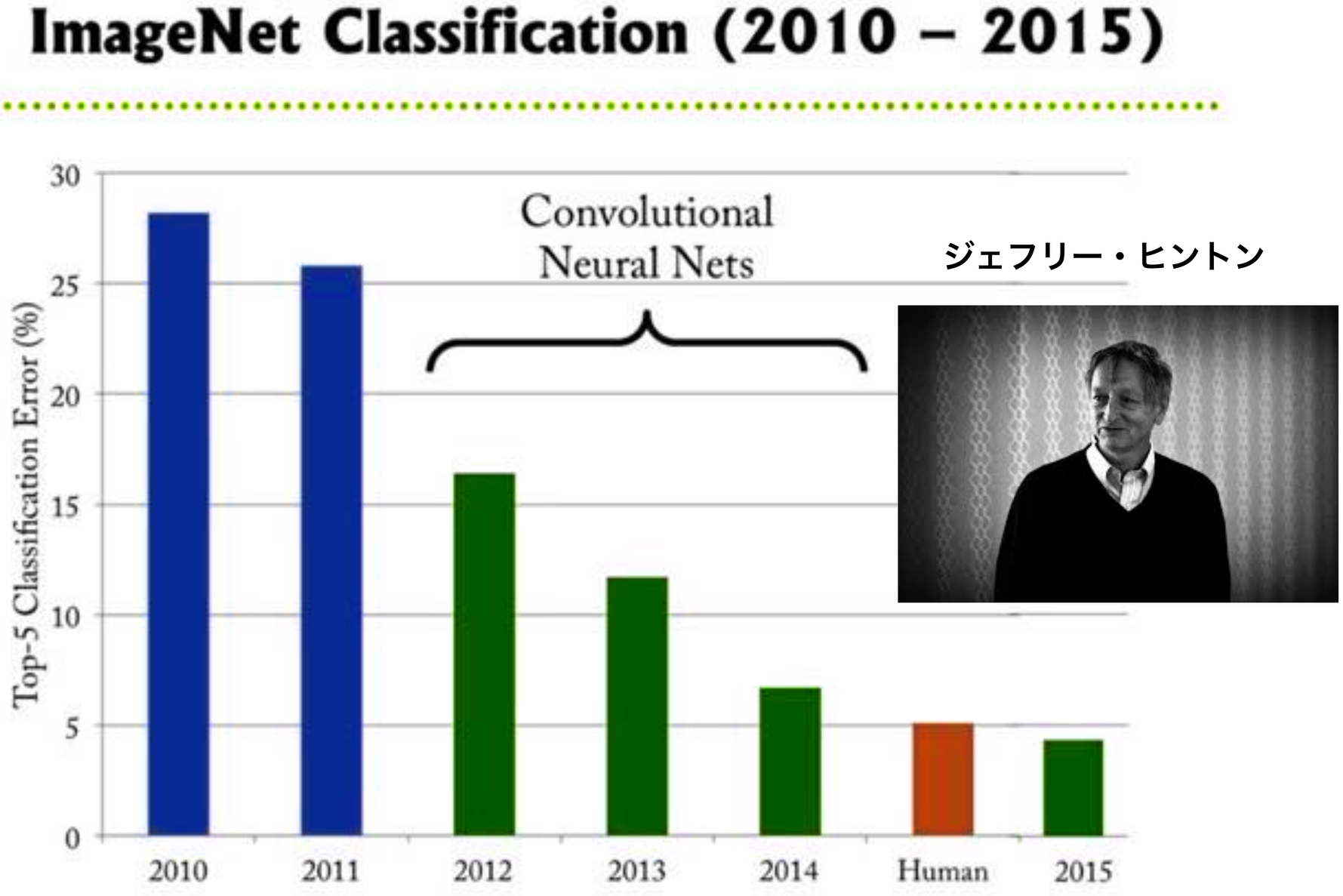
Imagenetにおける画像分類

- ImageNet: スタンフォード大学による画像データベース(2万カテゴリ,1400万枚の画像, ラベリングは人手)
- ILSVRC(ImageNet Large Scale Visual Recognition Challenge)を開催(公開コンペ)
- ILSVRCのデータ: 1000カテゴリ, 学習用画像120万枚, 確認用画像5万枚, テスト用画像15万枚
- 評価基準: トップ5 誤り率 (認識アルゴリズムが出力した上位の5つの答えに正解が含まれていれば認識成功)

ImageNetデータ例とTOP５出力

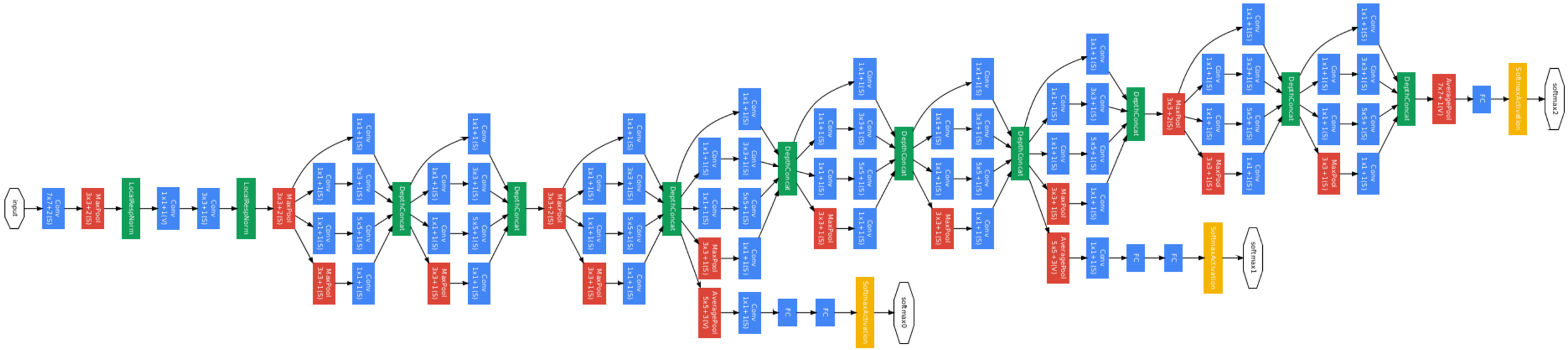


ImageNet認識の誤り率



ジェフリー・ヒントン

GoogleNet



- Googleが開発した深層ニューラルネットワークの一種
- 畳み込みニューラルネットワーク
- ILSVRC2014 トップ

MNIST数字認識

MNIST数字認識

MNISTデータセット

- 0-9の手書き数字のデータセット
- 28 x 28ピクセル (深さ8bit, モノクロ)
- 訓練データ6万枚/テストデータ１万枚
- パターン認識アルゴリズムのテストにしばしば利用される (パターン認識の”Hello World” と呼ばれている)

MNIST数字認識コード(1)

データローダの準備 (MNISTデータのダウンロードも含む)

In [ ]:

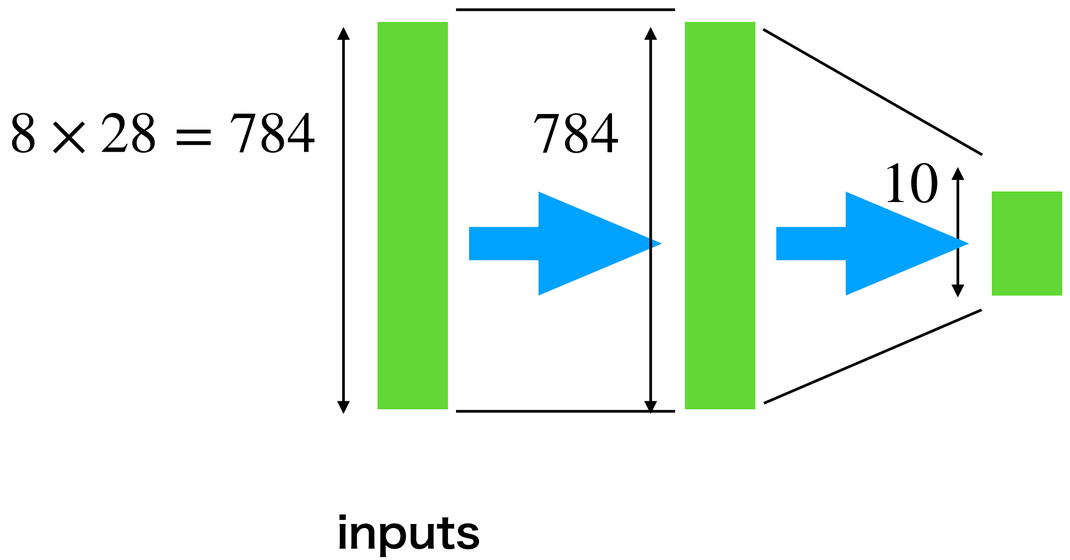
```
root = '.' # mnistデータの置き場所
download = True
trans = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,)
, (1.0,))])
train_set = datasets.MNIST(root=root, train=True, transform=trans, download=do
wnload)
test_set = datasets.MNIST(root=root, train=False, transform=trans)
# ローダの準備
train_loader = torch.utils.data.DataLoader(dataset=train_set, batch_size=batch
_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_set, batch_size=batch_s
ize, shuffle=False)
```

MNIST数字認識コード(2)

NNモデルの準備(全結合モデル)

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 784) # 28 x 28 = 784 次元の入力
        self.l2 = nn.Linear(784, 784)
        self.l3 = nn.Linear(784, 10)

    def forward(self, x):
        x = torch.sigmoid(self.l1(x))
        x = torch.sigmoid(self.l2(x))
        x = self.l3(x)
        return F.log_softmax(x, dim=1)
```



出力は１０次元で
0-9の各数字に対応

MNIST数字認識コード(3)

学習プロセス

訓練ループは
AND関数学習の
場合とほとんど同じ

```
model = Net() # モデルのインスタンス生成
optimizer = optim.SGD(model.parameters(), lr=sgd_lr)
running_loss = 0.0
i = 0
for loop in range(3): # 3エポックの訓練
    for (input, target) in train_loader:
        i = i + 1
        input = input.view(-1, 28*28) # テンソルのサイズを整える
        optimizer.zero_grad()    # optimizerの初期化
        output = model(input)     # 推論計算
        loss = F.nll_loss(output, target) # 損失関数の定義
        loss.backward()                # バックプロパゲーション(後ろ向き計算)
        optimizer.step()               # パラメータ更新
        running_loss += loss.item()
        if i % 100 == 99:    # print every 100 mini-batches
            print('[%5d] loss: %.3f' %
                  (i + 1, running_loss / 100))
            running_loss = 0.0
```

MNIST数字認識コード(4)

推定精度(正解率)の評価

```
correct =  0 # 正解数
count = 0 # 試行数
with torch.no_grad():
    for (input, target) in test_loader:
        input = input.view(-1, 28*28)
        output = model(input)      # 推論計算
        pred = output.argmax(dim=1)
        correct += pred.eq(target.data).sum()
        count += batch_size
print ('accuracy = ', float(correct)/float(count)) # 正解率の表示
```

テストデータを利用することに注意

2次元テンソルに変換

最大値を与える要素インデックス

汎化誤差を小さくするための技術

深層学習技術の分類

要素技術が大量にあるので、目的ごとに整理して理解しておくとよい

A. 関数近似のための最適化技術

確率的勾配法
誤差逆伝播法
勾配消失を抑制する活性化関数
スキップ構造(ResNet)
バッチ正規化

B. 汎化誤差を小さくするための技術

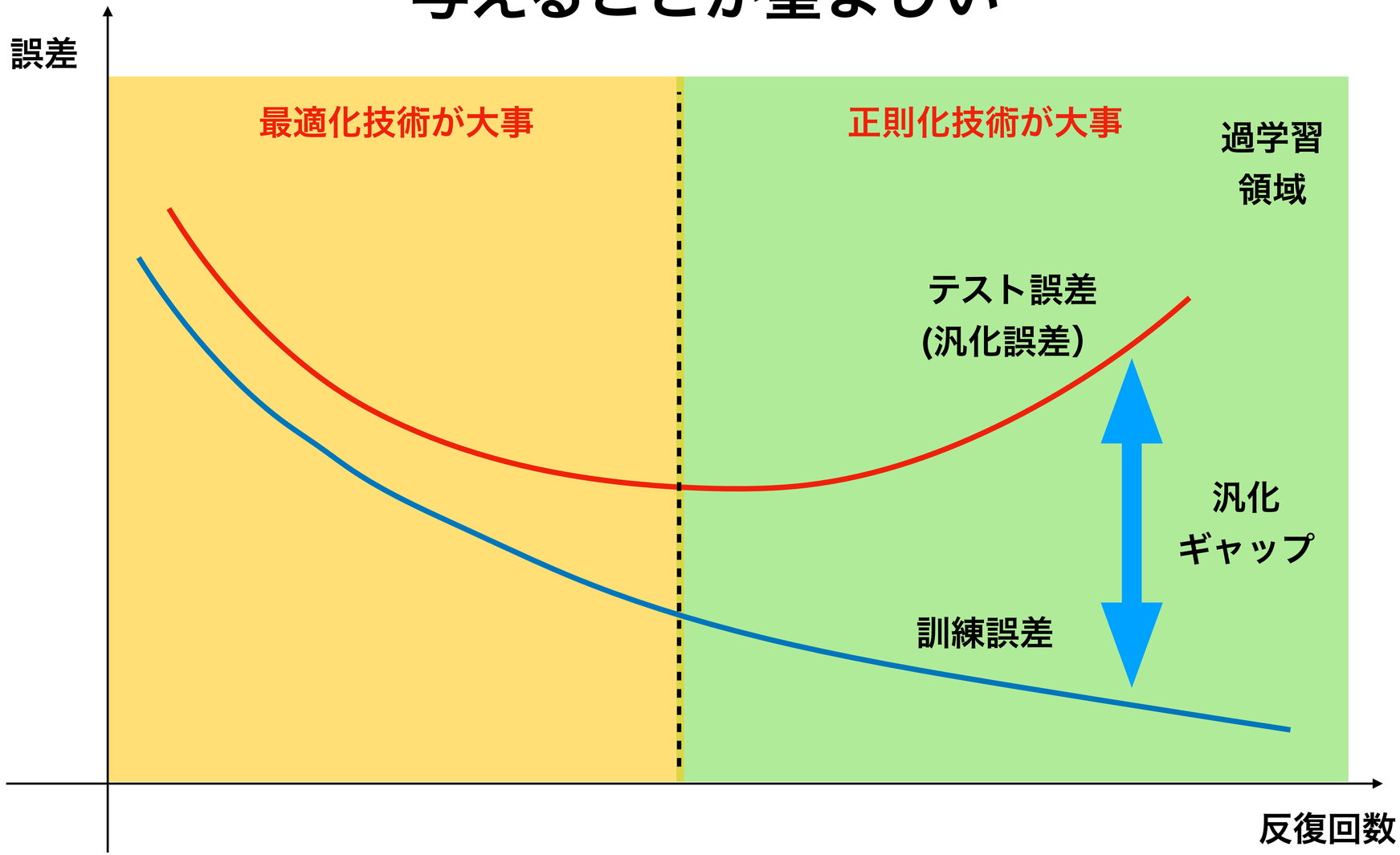
正則化
重み共有
バッチ正規化
ドロップアウト
大規模データセットの利用
データ拡張

C. 表現学習のための技術

畳み込み
ネットワーク
ネットワーク構造

汎化誤差

訓練データに含まれない**初見のデータ**に対して適切な出力を与えることが望ましい

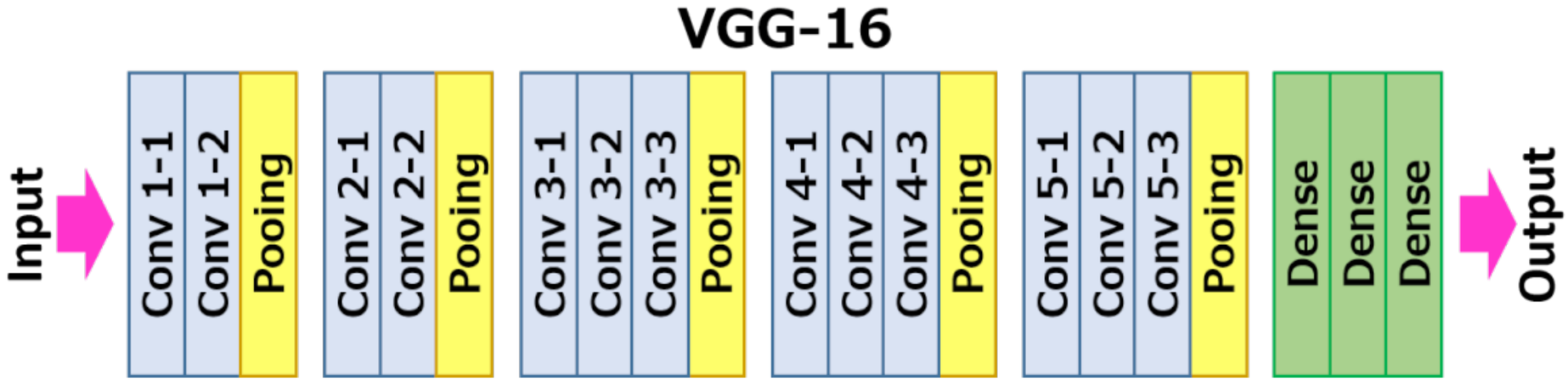


汎化誤差を小さくするための技術

- **早期学習停止**: 過学習が生じる手前で学習プロセスを停止（ただし何時、過学習がスタートするかを知ることは一般に難しい)
- **ドロップアウト**: 学習時に行列Wの要素をランダムに間引いて学習を進める
- **正則化**: 学習時に行列Wに何らかの制約条件を課す(フロベニウスノルムが小さくなるように、など)
- **バッチ正規化**: 各層を通るミニバッチについて平均・分散などを毎回正規化する

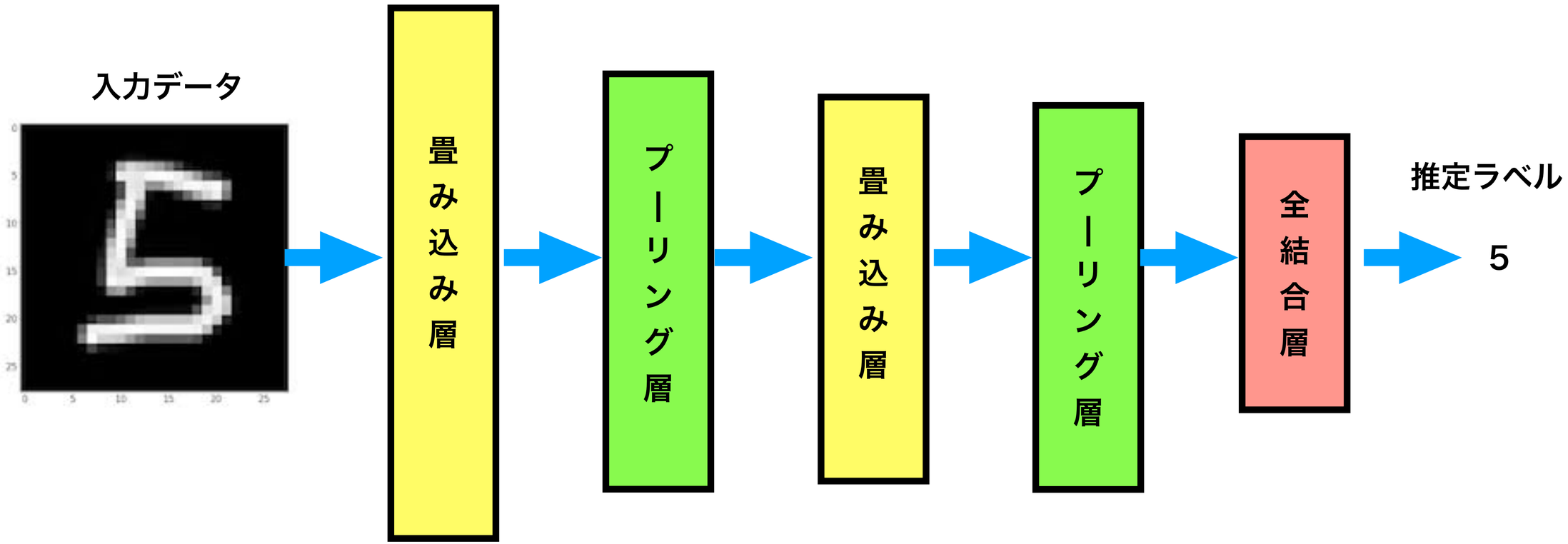
畳み込みニューラル

ネットワーク



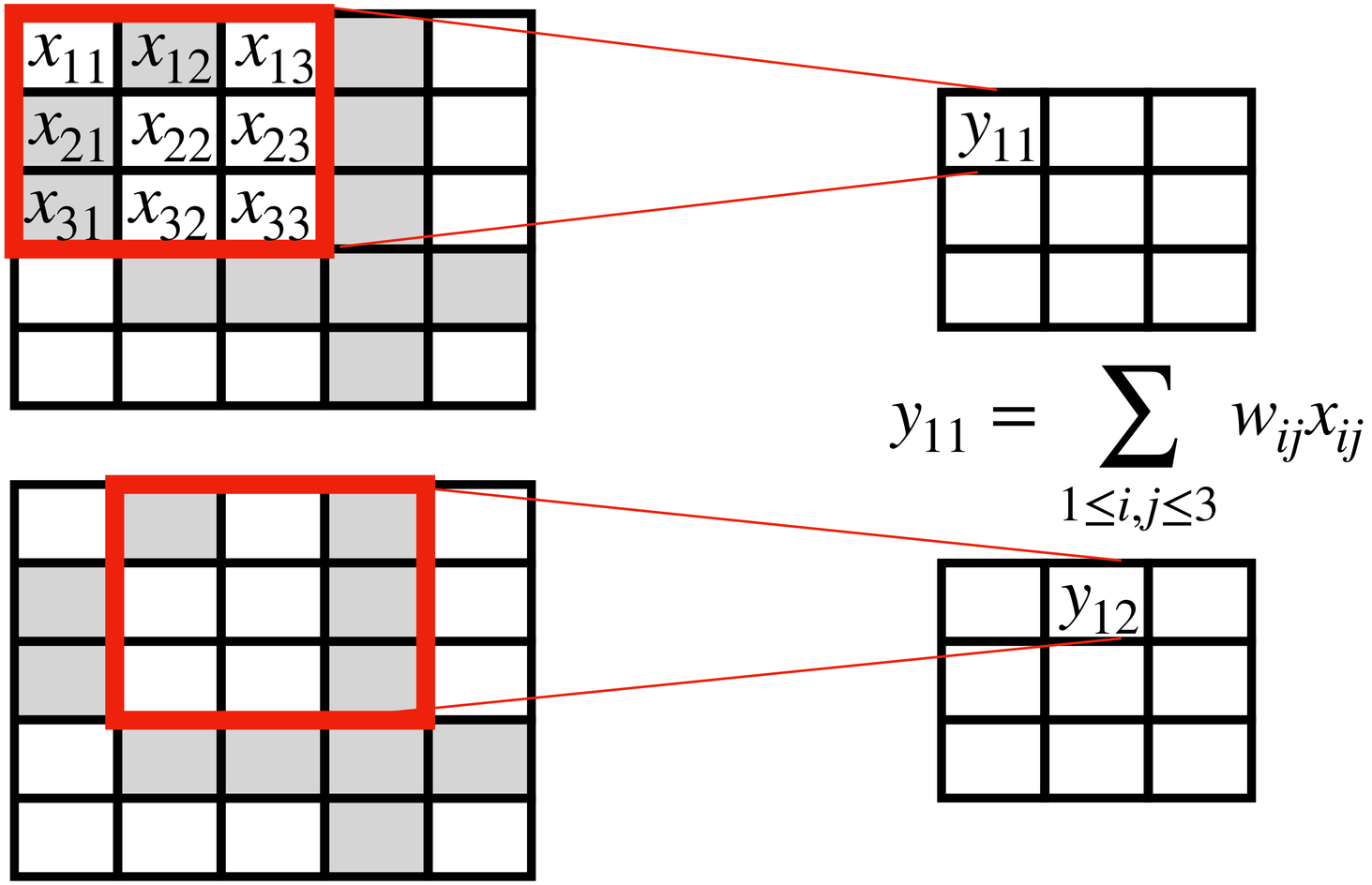
畳み込みニューラルネットワーク

- 2次元畳み込み演算に基づく層 = 畳み込み層(学習パラメータを含む)
- 2次元プーリング演算に基づく層 = プーリング層
- 全結合層



畳み込み層(1)

窓内のデータに学習可能パラメータを乗じて和を取る

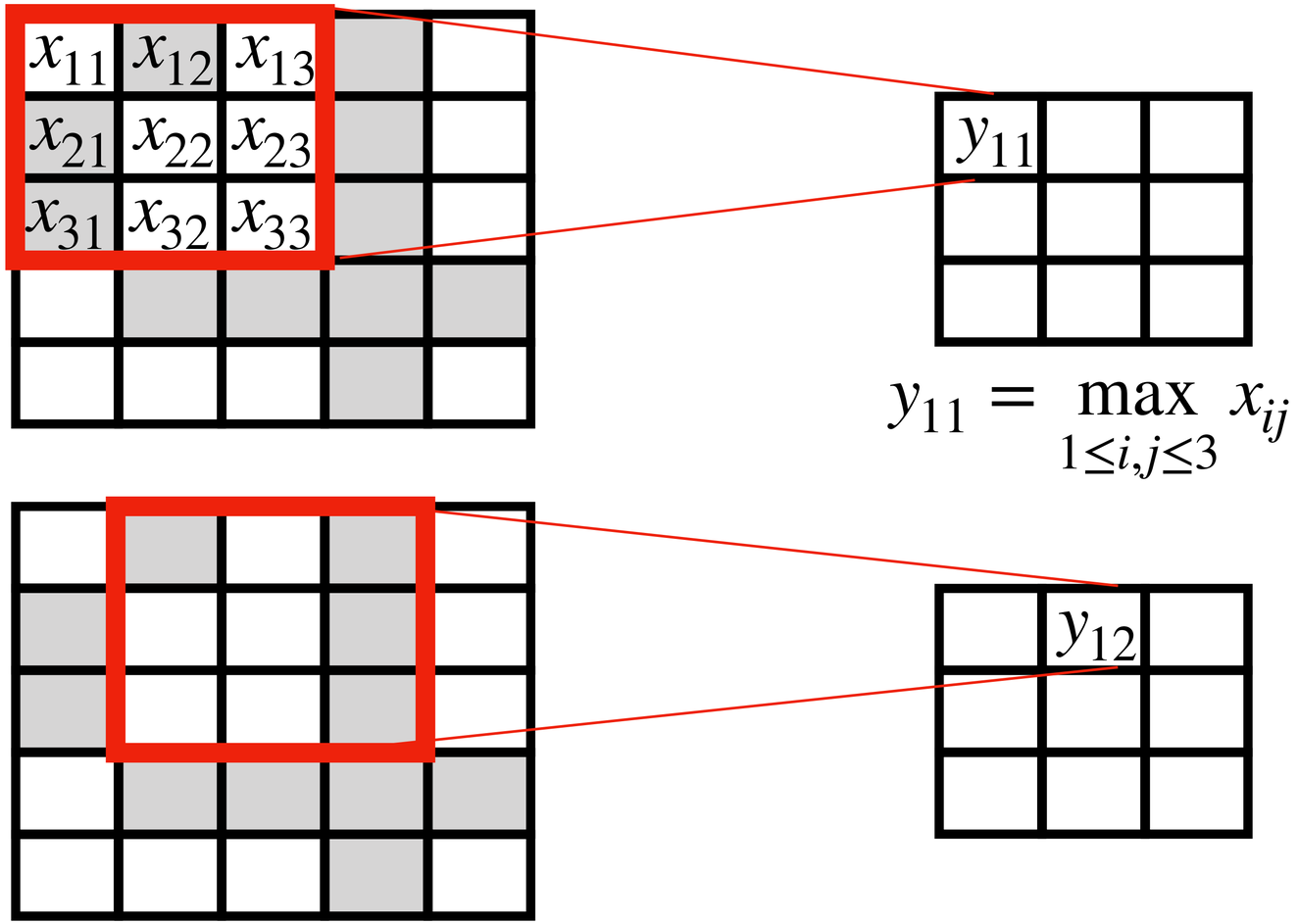


畳み込み層(2)

- w_{ij}は学習パラメータ(**畳み込みカーネル**と呼ばれることもある)であり、これを学習することで画像認識に適切な**2次元フィルタ**が学習される(**特徴学習・表現学習**)
- 画像の**局所的性質**を抽出している
- 上の説明は少し簡単化しており、本当は複数枚のチャネルから複数枚チャネルを生成する

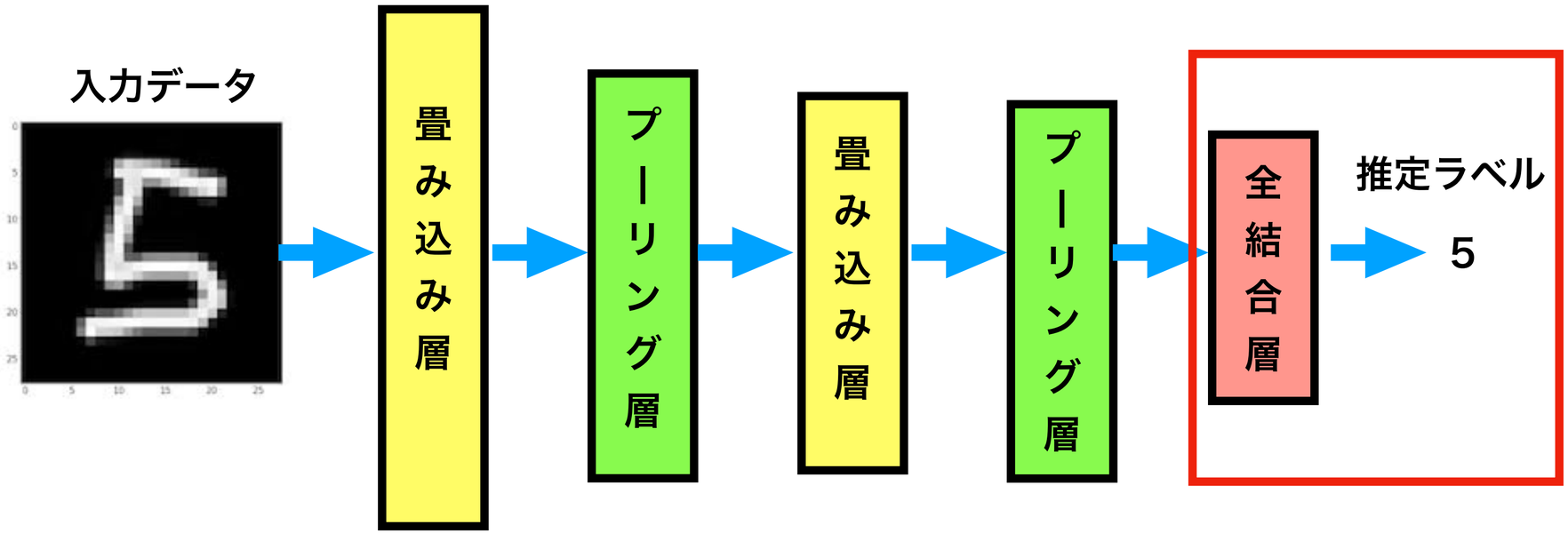
プーリング層

情報を縮約し、層のサイズを縮小させる



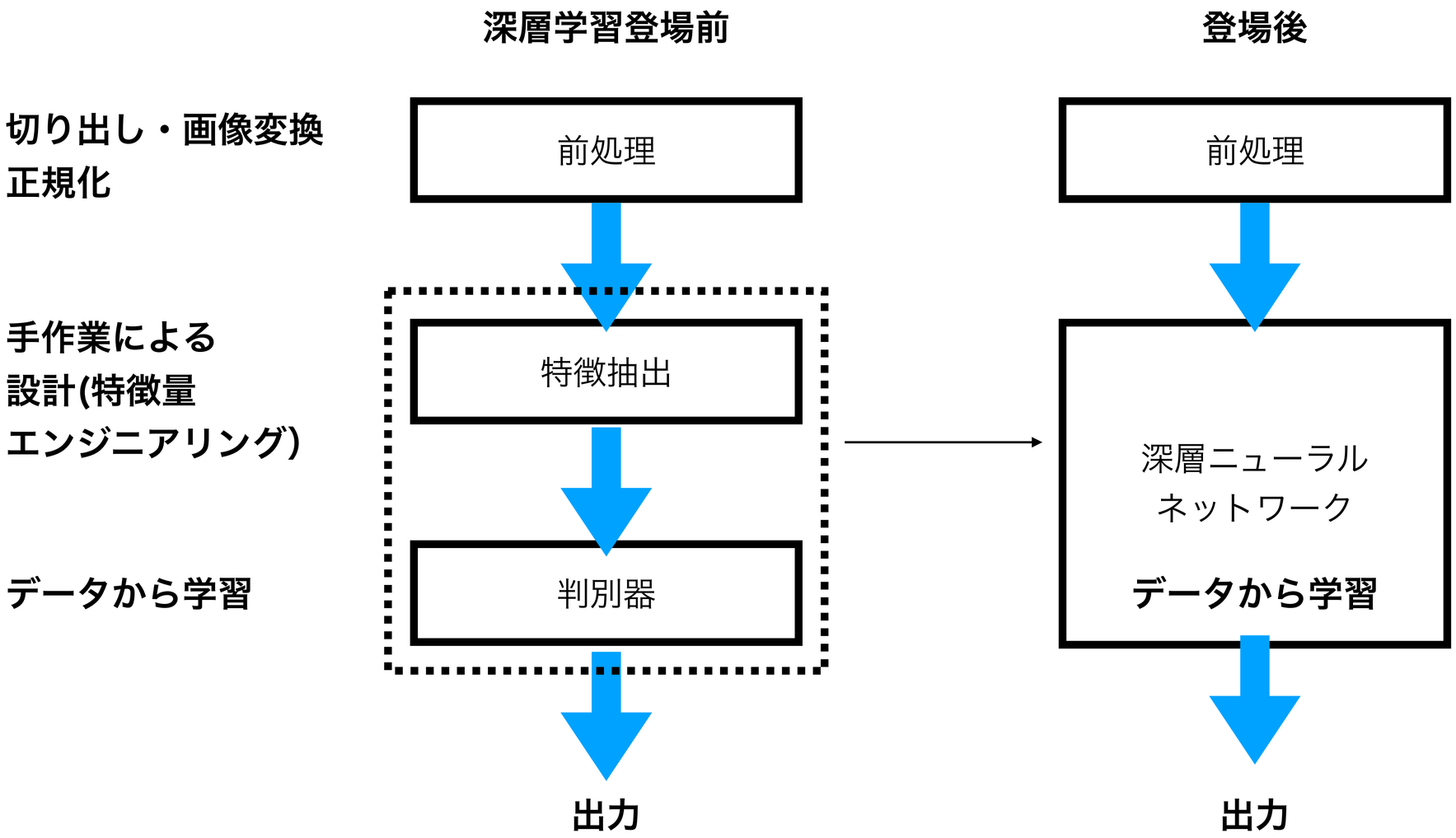
$$y_{11} = \max_{1 \le i,j \le 3} x_{ij}$$

畳み込みニューラルネットワークの構造



- 全結合層は$f(Wx + b)$型の通常のNN
- 最終層はソフトマックス層
- 現在においても優れた画像クラス判別のアルゴリズムは畳み込みNN構造

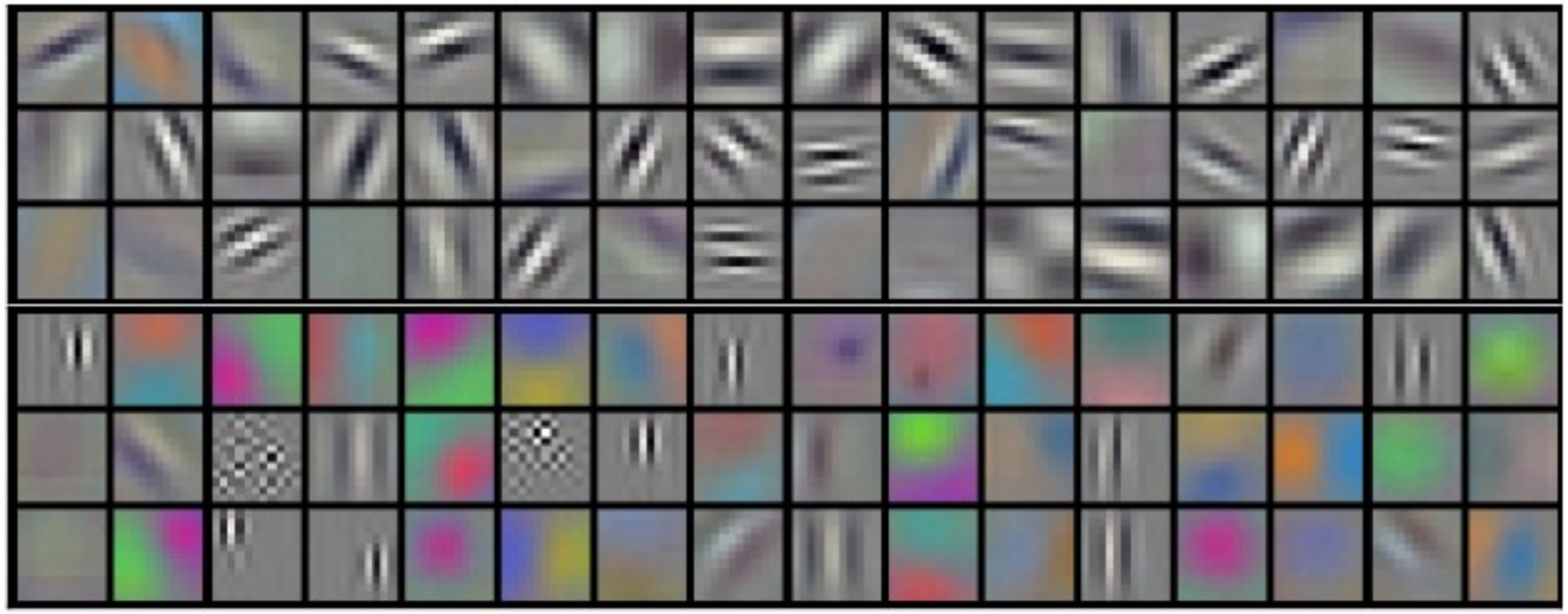
パターン認識構成のためのアプローチ



(過去)特徴量エンジニアリング→(現在）DNNによる表現学習

学習された畳み込みカーネルの可視化

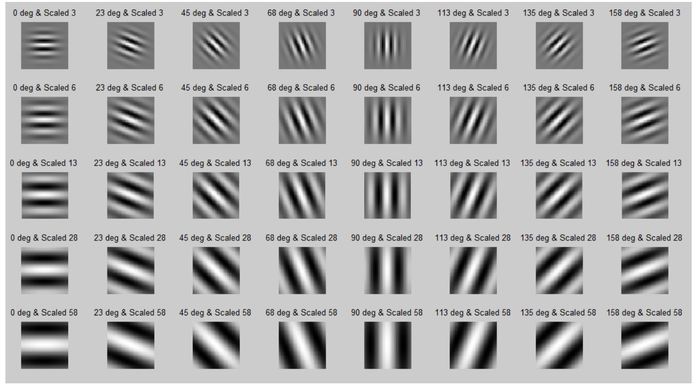
学習されたフィルタ係数の可視化



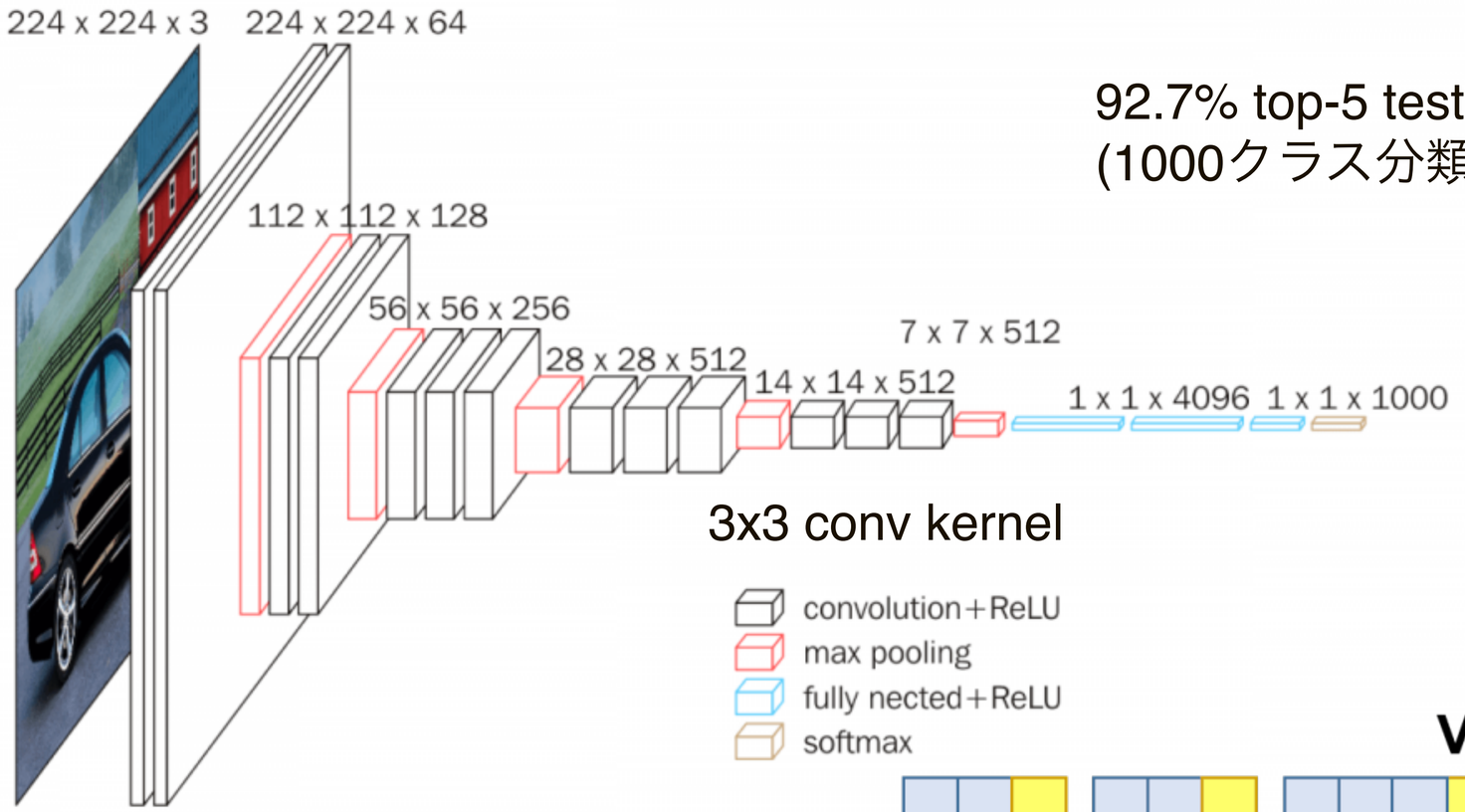
cited from https://cs231n.github.io/convolutional-networks/

参考: ガボールフィルタ

ガウス関数と三角関数の積として定義される2次元フィルタ。脳の一次視覚野にある単純型細胞の活動のモデルとしても利用されている



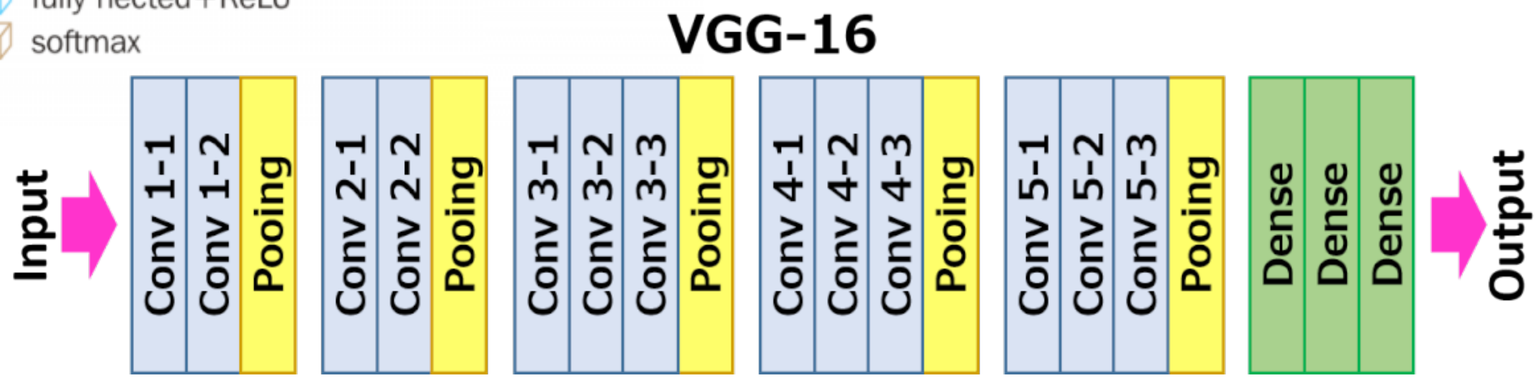
VGG16の構造



92.7% top-5 test accuracy in ImageNet
(1000クラス分類, ILSVRC-2014)

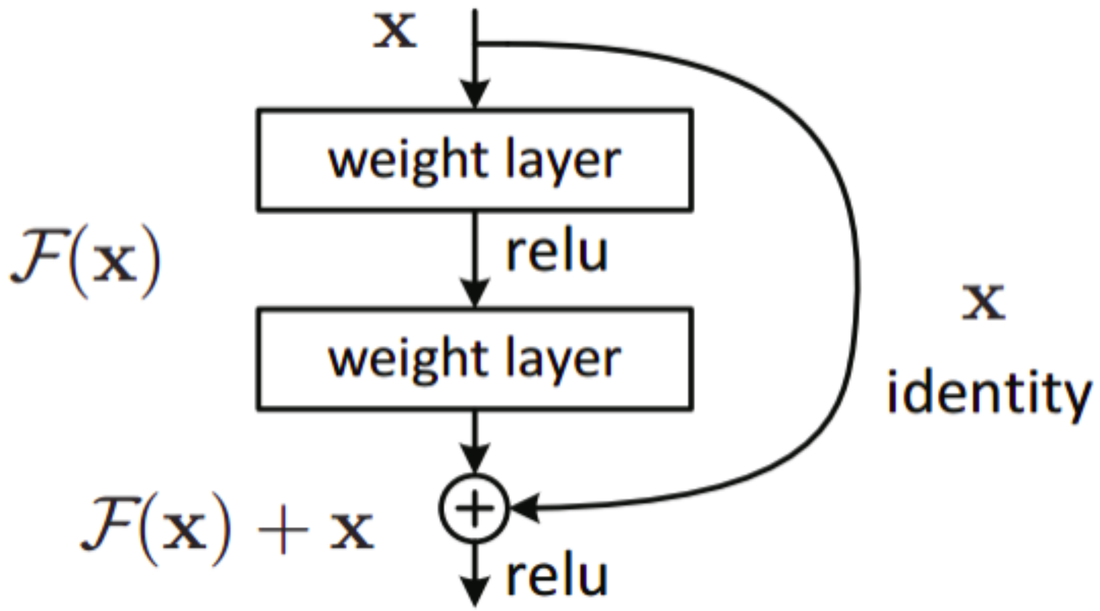
3x3 conv kernel

convolution+ReLU
max pooling
fully nected+ReLU
softmax

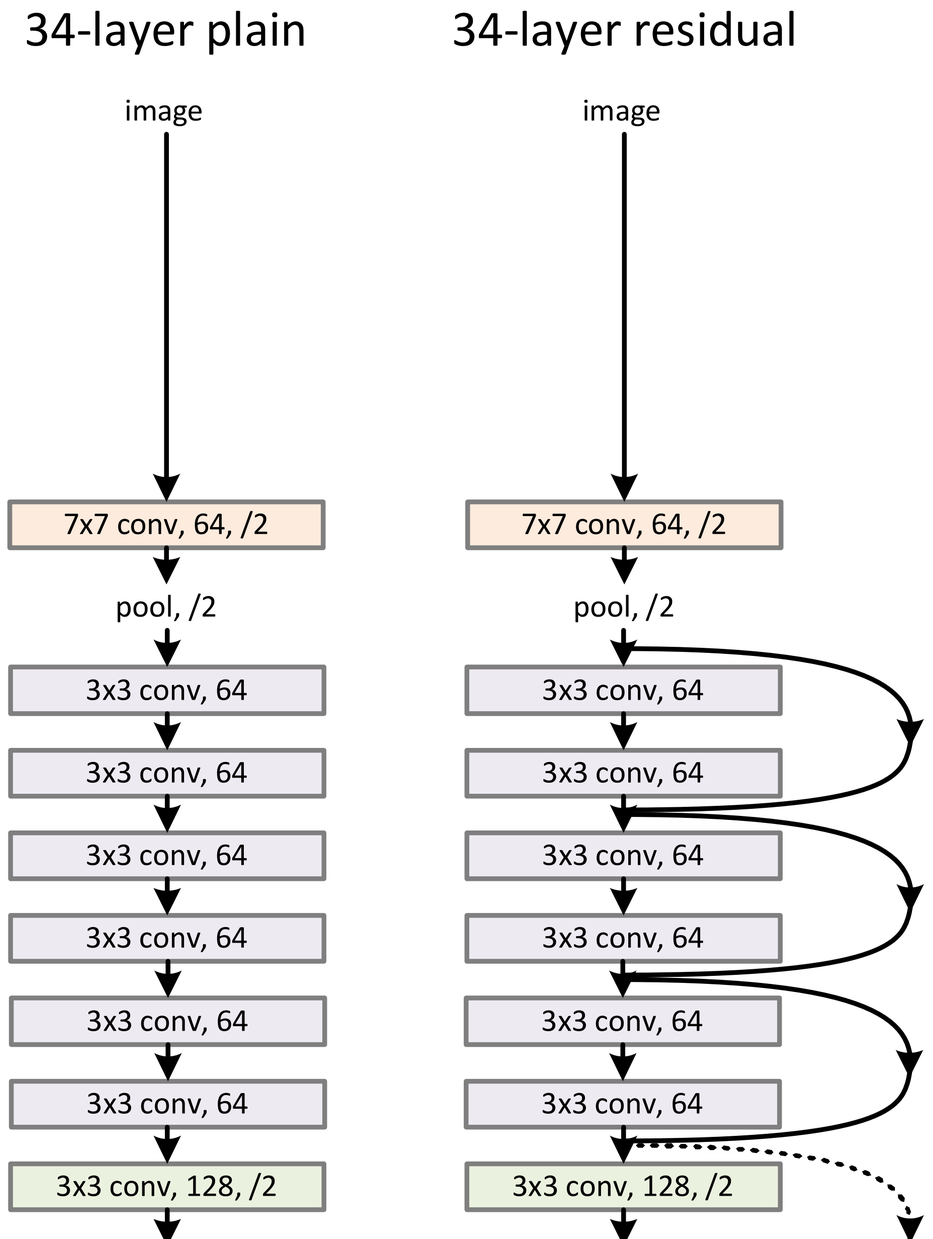


cited from https://neurohive.io/en/popular-networks/vgg16/

ResNet



スキップ構造が特徴的。勾配消失問題の軽減,
学習の安定化にスキップ構造は役立つ。



PyTorchで畳み込みNN

入力チャネル数
出力チャネル数
カーネルサイズ
全結合層

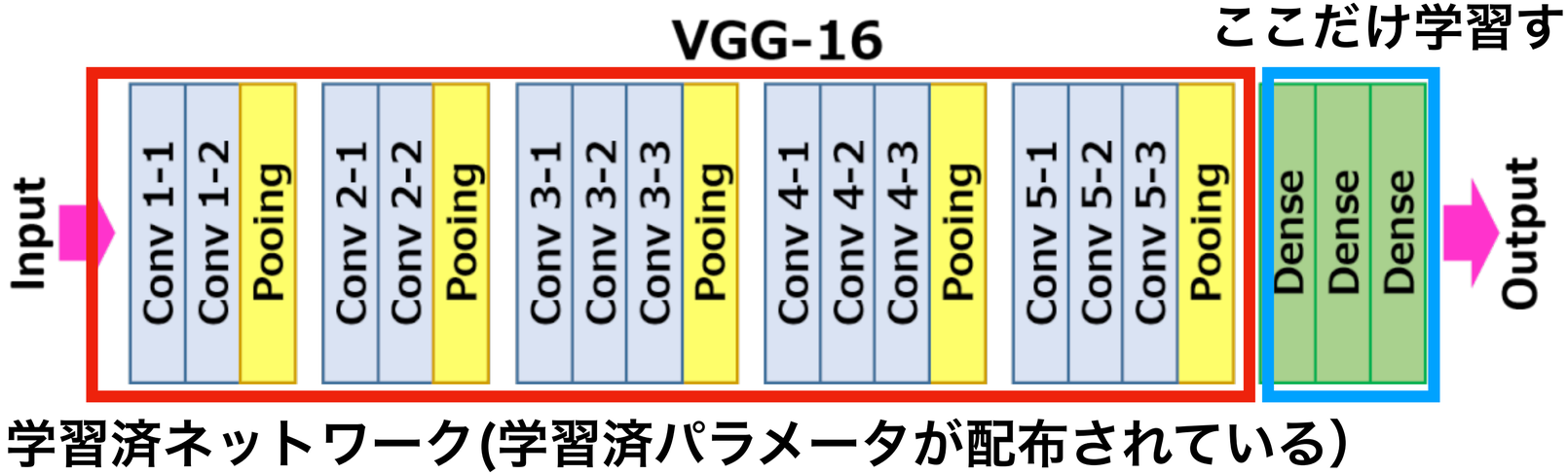
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3) # 28x28x1 -> 26x26x32
        self.conv2 = nn.Conv2d(32, 64, 3) # 26x26x32 -> 24x24x64
        self.pool = nn.MaxPool2d(2, 2) # 24x24x64 -> 12x12x64
        self.fc1 = nn.Linear(12 * 12 * 64, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x)) #  第１畳み込み層
        x = F.relu(self.conv2(x)) # 第２畳み込み層
        x = self.pool(x) # プーリング層
        x = x.view(-1, 12 * 12 * 64)
        x = F.relu(self.fc1(x)) #全結合層 1
        x = self.fc2(x) # 全結合層 2
        return F.log_softmax(x, dim=1)
```

コードはmnist_conv.ipynb です。演習課題も含まれています。

転移学習

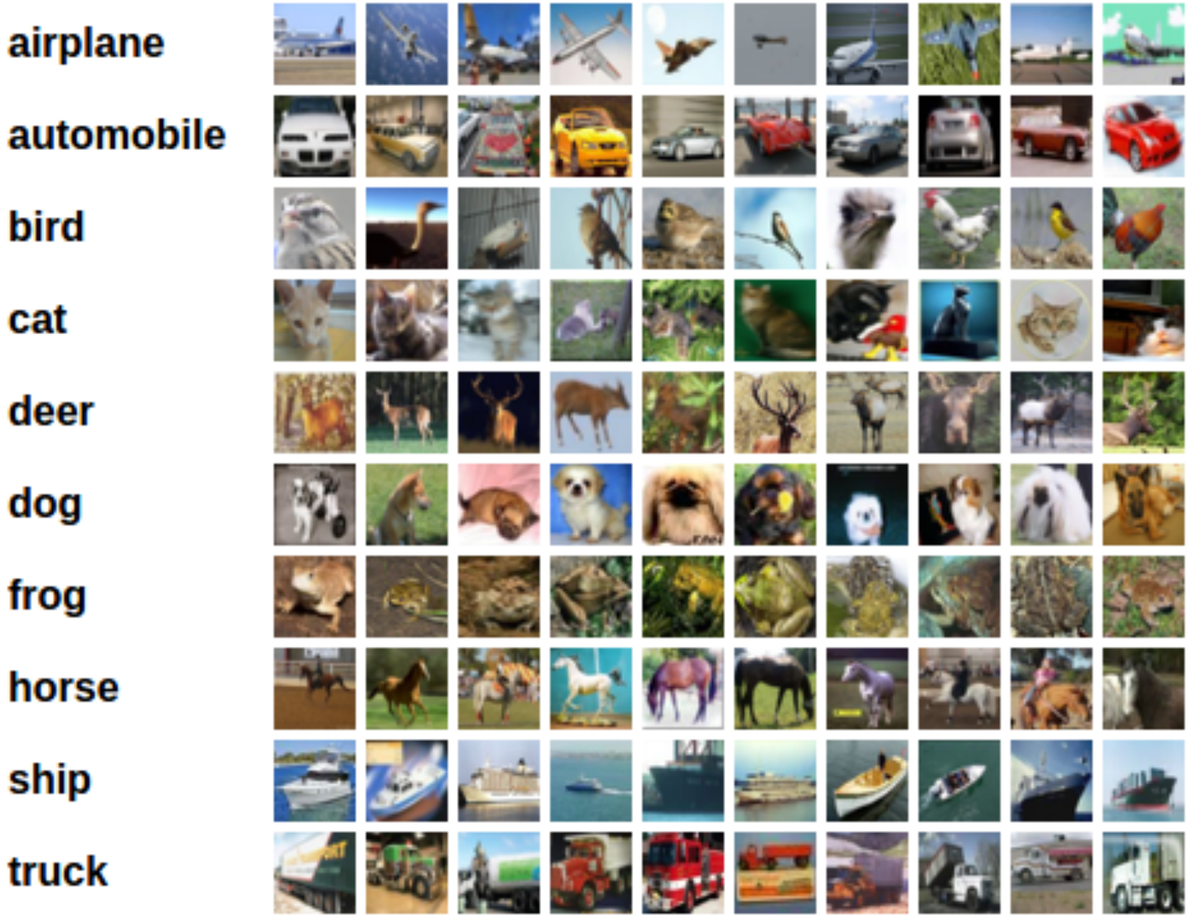
- VGG16などの大型のネットワークを訓練するためには、大量の画像データと計算パワー(多数のGPU)が必要
- 新しい画像認識タスクですべての層を学習し直すのは大変→ImageNetなどで学習済みのネットワークを使い回す(転移学習)
- 畳み込み層をフリーズして、全結合層のみを自分のタスク用のデータセットで再学習する



学習済ネットワーク(学習済パラメータが配布されている)

参考コード (1)

CIFAR10用のクラス分類コードが本家チュートリアルにあります。

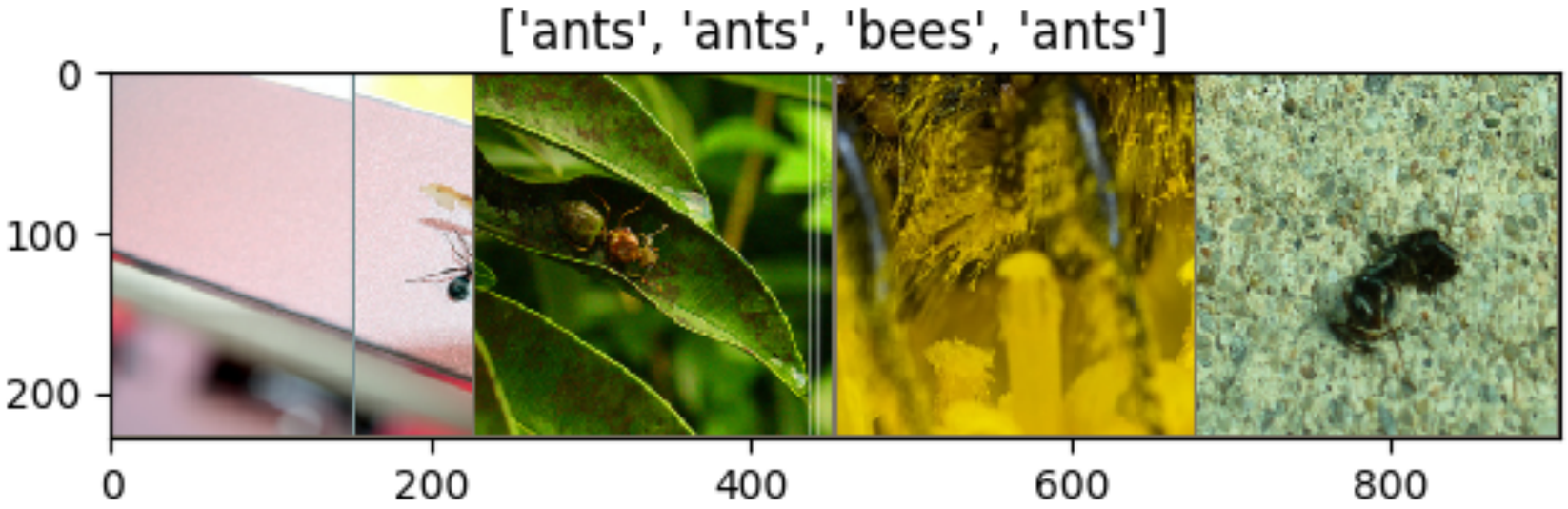


https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py

参考コード (2)

転移学習のコードが本家チュートリアルにあります。

アリとハチ 120枚の画像, ResNet-18 を利用



https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

本講義のまとめ

- Imagenetについて
- MNIST数字認識コードを学ぶ
- 汎化誤差を小さくするための技術
- 畳み込みニューラルネットワーク
- 転移学習