AVILEN　E資格試験想定問題集 vol.2

Batch Normalization

【問題】
Batch　Normalizationはニューラルネットワークにおける強力な正則化手法であるが、(1)場合では学習が収束しないことが知られている。
(1)に入る文章としてふさわしいものを選べ。

【選択肢】
(a)バッチサイズが小さい
(b)学習パラメータの初期値における分散が大きい
(c)バッチサイズが大きい
(d)学習パラメータの初期値における分散が小さい

【解答】
(a)バッチサイズが小さい

【解説】
Batch　Normalizationにおける弱点は、バッチサイズが小さいと正規化の際に用いる統計量が母集団の統計量から乖離して不正確になってしまい、学習が収束しないケースが存在することである。その様なケースに対応するための手法としてLayer　NormalizatonやInstance Normalization,Group Normalizationなどが挙げられる。
なお学習パラメータの初期値における分散の大小は、当然ミニバッチ毎の統計量が母集団の統計量から乖離してしまう原因にはならず、Batch　Normalizationの処理の妨げになる他の原因にもならない。

Layer Normalizationの実装

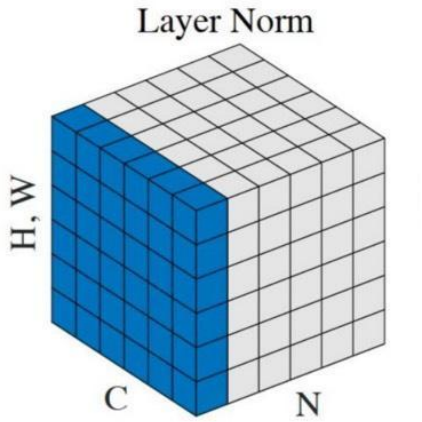
【問題】
以下はAffine LayerにLayer normalizationを挿入した際の順伝播計算を実装したコードである。
Aにあてはまるコードとしてふさわしいものを選べ。

```
def layer_normalize_with_affine_forward(x, w, b, g, f):
    """
    x: 入力 (batch_size, input_size)
    w: weight (input_size, output_size)
    b: bias (output_soze, )
    g: gain parameter (output_size, )
    f: 活性化関数
    """
    a = np.dot(x, w)
    layer_mean = np.mean(    A    )
    layer_var = np.var(    A    )
    a_normed = g * (a - layer_mean) / np.sqrt(layer_var + 1e-8)
    h = f(a_normed + b)
    return h
```

【選択肢】
(a)a,axis=1
(b)a,axis=0
(c)x,axis=0
(d)x,axis=1

【解答】
(a)a,axis=1

【解説】
Layer Normalizationは同じ層のニューロン間で正規化を行う手法である。
重みをかけた後の値でその層のニューロン間において平均と分散の計算を行うので、第一引数は「a」、第二引数は出力ニューロンの軸なので「axis=1」とするのが正しい。
Layer Normalizationは三次元データで考えると、以下の図の青色部分で正規化が行われる。(問題のコードにおけるoutput_sizeがchannelとheight,widthの二つの次元に分けられていると解釈できる)



計算式は以下の通り。(ある一つの層について考えるとする。)(x:入力、w:重みパラメータ、b:バイアスパラメータ、H:その層におけるノードの総数、g:ゲインパラメータ、f:活性化関数)

$$a_i = x_i w_i$$
$$\mu = \frac{1}{H}\sum_{i=1}^{H} a_i$$
$$\sigma^2 = \frac{1}{H}\sum_{i=1}^{H}(a_i - \mu)^2$$
$$\bar{a}_i = \frac{g_i}{\sigma}(a_i - \mu)$$
$$h_i = f(\bar{a}_i + b_i)$$

Instance Normalization

【問題】
正規化手法の一つであるInstance Normalizationについての説明として誤っているものを選べ。

【選択肢】
(a)各チャンネルで独立に画像の縦横方向についてのみ平均と分散をとる手法である。
(b)バッチサイズが十分に確保されているような場合には、Batch　Normalizationに比べて性能が落ちるとされている。

(c)Layer　Normalizationとは異なり、RNNなどの画像以外の分野にはあまり拡張できないことが知られている。
(d)Batch Normalizationと同様に学習時のみ適用される。

【解答】
(d)Batch Normalizationと同様に学習時のみ適用する手法である。

【解説】
Instace Normalizationは、学習時のみ適用されるBatch Normalizationとは異なり、学習時とテスト時の両方で適用される手法であるため誤り。
Instace　Normalizationは各チャンネルで独立に画像の縦横方向についてのみ平均と分散をとる手法である。画像以外の分野ではあまり有用では無いが、画像生成などの分野では注目されている。例えば、二つの入力画像の特徴を合わせ持つ様な合成画像を生成するスタイル転送タスクにおいては、Batch　Normalizationよりも高品質の画像の生成が可能であることが報告されている。
なお、バッチサイズが十分に確保されているような場合には、Batch Normalizationに比べて性能が落ちるとされている。

【参考】
https://qiita.com/kidach1/items/0e7af5981e39955f33d6

Instance Normalizationの実装

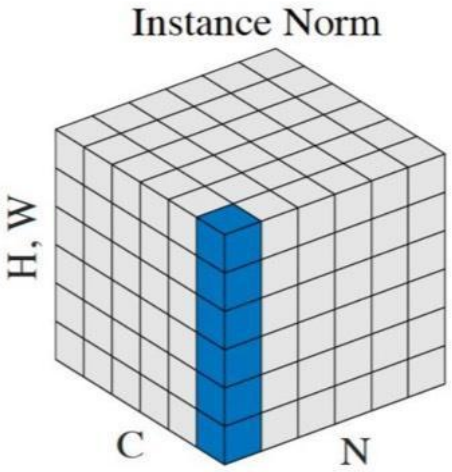
【問題】
以下はInstance Normalizationの順伝播計算を実装したコードである。
Aにあてはまるコードとしてふさわしいものを選べ。

```
def instance_normalize_forward(x):
    """
    x: 入力 (batch_size, channel, width, height)
    """
    mean = np.mean(x, axis=  A  )
    var = np.var(x, axis=  A  )
    x_normed = (x - mean) / np.sqrt(var + 1e-8)
    return x_normed
```

【選択肢】
(a)(0,1)
(b)(2,3)
(c)(0,2,3)
(d)(1,2,3)

【解答】
(b)(2,3)

【解説】
Instance　Normalizationは各チャンネルで独立に画像の縦横方向についてのみ平均と分散をとる手法である。よって平均と分散の計算の際は画像の縦横方向を軸として行うため、axis=(2,3)とするのが適切である。
Instance Normalizationは三次元データで考えると、以下の図の青色部分で正規化が行われる。(コードでは分けられていたHeightとWidthをまとめて、三次元としていると解釈できる)



計算は以下の様に行われる。(ある一つのチャンネルと一つのバッチについて考えるとする。)(x:入力、H:与えられたデータの縦方向のサイズ、W:与えられたデータの横方向のサイズ)

$$\mu = \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H} x_{wh}$$

$$\sigma^2 = \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H}(x_{wh}-\mu)^2$$

$$\bar{x}_{ij} = \frac{x_{ij}-\mu}{\sqrt{\sigma^2+\epsilon}}$$

SGDの理論式

【問題】
パラメータの最適化手法アルゴリズムの1つとして知られる確率勾配降下法(SGD)のパラメータの更新式として正しいものを選べ。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

ただし、$\boldsymbol{W}_t$ を最適化したいパラメータWの時刻tでの値、Lを損失関数、ηを学習率を表すハイパーパラメータとし、学習に用いるデータは更新ごとに1データずつであるとする。

【選択肢】
(a)

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

(b)

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

(c)

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

(d)

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

【解答】
(d)

【解説】
SGDの更新式は以下の通りである。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

時刻tのパラメータWの予測位置 において、

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

を動かした時にLが最も大きく降下する方向にパラメータを更新するのが最急降下法であった（その為、最急降下法の更新式もSGDと同じである）。
最急降下法とSGDが異なるのは、最急降下法は一回の更新に全データを用いているが、SGDでは一回の更新には一つのデータを用いるか、もしくはミニバッチとして全体から数個のデータを抽出して用いる、という点である。本問は「学習に用いるデータは各更新ごとに1データずつである」としていたので前者のSGDである。また、後者のSGDはミニバッチSGDとも呼ばれる。SGDが「確率」勾配降下法と呼ばれているのは、このデータの抽出がランダムに行われるためである。

話を更新式に戻そう。最適化アルゴリズムの共通の目的は「損失関数Lをなるべく小さくするパラメータW
を見つける」ことである。ここで

・LのWによる勾配 $W_{t+1} = W_t - \eta \left(\frac{\partial L}{\partial W} \right)_{W=W_t}$ が正であるなら、その近傍でLはWの値が小さくなるにつれ
減少し

・LのWによる勾配 $W_{t+1} = W_t - \eta \left(\frac{\partial L}{\partial W} \right)_{W=W_t}$ が負であるなら、その近傍でLはWの値が大きくなるにつれ
減少

するという数学的性質から、素直にLが減少する方向にWを更新しようとしたのがSGDや最急降下法の
更新式の大まかな意味である。

ただし、SGDは更新式の単純さ故に学習後半での学習速度の低下や収束に時間がかかるなどの問題も
指摘されており、代替のアルゴリズムが多数考案されている。

SGDの実装

【問題】
パラメータの最適化手法アルゴリズムの1つとして知られる確率勾配降下法(SGD)をpythonを用いて実装した。以下のコード中の空欄(あ)を埋めよ。ここで、lrは学習率、paramsは各パラメータに対する値、gradsは各パラメータの勾配を格納している辞書型である。また、学習に用いるデータは更新ごとに1データずつであるとする。

```
class SGD:

    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] （あ）
```

【選択肢】
(a)+= self.lr * grads[key]
(b)-= self.lr * grads[key]
(c)= self.lr / grads[key]
(d)*= self.lr / grads[key]

【解答】
(b)

【解説】
確率勾配降下法(SGD)の実装に関する問題である。1データごとサンプリングして更新するSGDでは、パラメータを以下の式で更新する。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

ただし、$\boldsymbol{W}_t$ を最適化したいパラメータWの時刻tでの値、Lを損失関数、ηを学習率とする。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

本問のコードでは、updateメソッドにおいてparams中のパラメータそれぞれの更新をfor文を通して実装している。理論式にならい(あ)を正しく埋めたコードは以下の通りである。

```
class SGD:

    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]
```

モメンタムの理論式

【問題】
パラメータの最適化手法アルゴリズムの1つとして知られるモメンタムのパラメータの更新式として正しいものを選べ。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

ただし、$\boldsymbol{W}_t$ は最適化したいパラメータWの時刻tにおける値、$\left(\frac{\partial L}{\partial \boldsymbol{W}} \right)$ はWに関する損失関数の勾配、α・ηはハイパーパラメータを表す。

【選択肢】

(a)
$$\begin{cases} v_{t+1} = \alpha v_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} = \boldsymbol{W} + v_{t+1} \end{cases}$$

(b)
$$\begin{cases} v_{t+1} = \alpha {v_t}^2 - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} = \boldsymbol{W} + v_{t+1} \end{cases}$$

(c)
$$\begin{cases} v_{t+1} = \alpha v_t + \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} = \boldsymbol{W} + v_{t+1} \end{cases}$$

(d)
$$\begin{cases} v_{t+1} = \alpha {v_t}^2 + \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} = \boldsymbol{W} + v_{t+1} \end{cases}$$

【解答】
(a)

【解説】
モメンタムの更新式は以下の通りである。

$$\begin{cases} \boldsymbol{v}_{t+1} & = \alpha \boldsymbol{v}_t - \eta \left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t + \boldsymbol{v}_{t+1} \end{cases}$$

モメンタムは「運動量」という意味であり、ボールが坂を転がるような物理的な挙動で最適化するのが特徴である。上の更新式において、vはこの例でいうボールの速度に対応する変数である。
2つ目の式は速度v方向にパラメータが移動している様子を表しており、わかりやすい。一方で、1つ目の式は $\begin{cases} \boldsymbol{v}_{t+1} & = \alpha \boldsymbol{v}_t - \eta \left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t + \boldsymbol{v}_{t+1} \end{cases}$ で1時刻前の速度情報を引き継ぎつつ、

$\begin{cases} \boldsymbol{v}_{t+1} & = \alpha \boldsymbol{v}_t - \eta \left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t + \boldsymbol{v}_{t+1} \end{cases}$ で損失関数LのWに対する勾配が正ならば減速、負ならば加速という、坂を転がるボールのような物理的振る舞いを表現している式である。

モメンタムの実装

【問題】
パラメータの最適化手法アルゴリズムの1つとして知られるモメンタムをpythonを用いて実装した。以下のコード中の空欄(あ)を埋めよ。ここで、lrは学習率、momentumはモメンタムのハイパーパラメータ、paramsは各パラメータに対する値、gradsは各パラメータの勾配を格納している辞書型である。

```
class Momentum:

    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = （あ）
            params[key] += self.v[key]
```

【選択肢】
(a)self.lr*self.v[key] + self.momentum*grads[key]
(b)self.lr*self.v[key] - self.momentum*grads[key]
(c)self.momentum*self.v[key] + self.lr*grads[key]
(d)self.momentum*self.v[key] - self.lr*grads[key]

【解答】
(d)

【解説】
モメンタムの実装に関する問題である。モメンタムでは、パラメータを以下の式で更新する。

$$\begin{cases} \boldsymbol{v}_{t+1} & = \alpha \boldsymbol{v}_t - \eta \left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t + \boldsymbol{v}_{t+1} \end{cases}$$

本問のコードにおけるself.vはモメンタムにおける速度を表現するvを表しており、updateメソッドが最初に呼び出された際に各キーに対してselfv.[key] = no.zeros_like(val)でparamsと同じ構造のデータを保持している。理論式にならい(あ)を正しく埋めたコードは以下の通りである。

```
class Momentum:

    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
            params[key] += self.v[key]
```

Adamの理論式

【問題】
以下はパラメータの最適化手法アルゴリズムの1つとして知られるAdamのパラメータの更新式である。空欄(あ)(い)を埋めよ。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

ただし $\left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$ は最適化したいパラメータWの時刻tにおける値、$\left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)$ はWに関する損失関数の勾配、β_1 β_2 はハイパーパラメータ、δは微小な正数とする。また選択肢中 $\beta_2 \boldsymbol{v}_t + (1-\beta_2) \left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\dfrac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$ は行列間の要

素積を表す。

$$
\begin{cases}
\boldsymbol{m}_{t+1} = & \boxed{\text{（あ）}} \\
\boldsymbol{v}_{t+1} = & \boxed{\text{（い）}} \\
\hat{\boldsymbol{m}}_{t+1} = \dfrac{\boldsymbol{m}_{t+1}}{1-\beta_1^{t+1}} \\
\hat{\boldsymbol{v}}_{t+1} = \dfrac{\boldsymbol{v}_{t+1}}{1-\beta_2^{t+1}} \\
\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \dfrac{\hat{\boldsymbol{m}}_{t+1}}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \delta}
\end{cases}
$$

【選択肢】
（あ）の選択肢:
(a)

$$\boldsymbol{m}_t + \beta_1 \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

(b)

$$\boldsymbol{m}_t + \beta_2 \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

(c)

$$\beta_1 \boldsymbol{m}_t + (1-\beta_1) \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

(d)

$$\beta_1 \boldsymbol{m}_t + \beta_2 \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

（い）の選択肢 :
(a)

$$\beta_2 \boldsymbol{v}_t + (1-\beta_2) \boldsymbol{v}_{t-1}$$

(b)

$$\beta_2 \boldsymbol{v}_t + (1-\beta_2) \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

(c)

$$\beta_2 \boldsymbol{v}_t + (1-\beta_2) \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

(d)

$$\beta_1 \boldsymbol{v}_t + \beta_2 \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

【解答】
（あ）(c)（い）(c)

【解説】
Adamの更新式は以下の通りである。

$$
\begin{cases}
\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\
\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\
\hat{\boldsymbol{m}}_{t+1} = \dfrac{\boldsymbol{m}_{t+1}}{1-\beta_1^{t+1}} \\
\hat{\boldsymbol{v}}_{t+1} = \dfrac{\boldsymbol{v}_{t+1}}{1-\beta_2^{t+1}} \\
\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \dfrac{\hat{\boldsymbol{m}}_{t+1}}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \delta}
\end{cases}
$$

Adamでは、主に2つの変数m、vを介して更新を行い、mに関してモメンタムの勾配に移動平均を用いるアルゴリズム、vに関してAdaGradの過去の勾配情報を指数平均を用いて調整するアルゴリズムを採用したアルゴリズムである。
また、上式の3・4行目でm・vそれぞれを調整するハイパーパラメータの「バイアス補正」を行なっている点はAdam特有のアルゴリズムである（詳しくは統計的な説明になるので割愛する）。

Adamの実装

【問題】
パラメータの最適化手法アルゴリズムの1つとして知られるAdamをpythonを用いて実装した。以下のコード中の空欄（あ）（い）を埋めよ。
ここで、コード中の変数は以下のような意味を持っている。
lr…学習率における定数。lr_tの計算の際に用いられる。
lr_t…イテレータごとに更新される学習率
beta1、beta2…Adamのハイパーパラメータ

```
class Adam:

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)#イテレータiterごとに学習率を更新

        for key in params.keys():
            self.m[key] +=                    (あ)
            self.v[key] +=                    (い)

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

【選択肢】
(あ)の選択肢 :
(a)self.beta1*self.m[key] + (1-self.beta1)*grads[key]
(b)self.beta1*self.m[key] + (1-self.beta1)*(grads[key]**2)
(c)(1 - self.beta1) * (grads[key] - self.m[key])
(d)(1 - self.beta1) * (grads[key]**2 - self.m[key])
(い)の選択肢 :
(a)self.beta2*self.v[key] + (1-self.beta2)*grads[key]
(b)self.beta2*self.v[key] + (1-self.beta2)*(grads[key]**2)
(c)(1 - self.beta2) * (grads[key] - self.v[key])
(d)(1 - self.beta2) * (grads[key]**2 - self.v[key])

【解答】
(あ)(c)
(い)(d)

【解説】
Adamの実装に関する問題である。Adamでは、パラメータを以下の理論式で更新する。

$$
\begin{cases}
\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\
\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\
\hat{\boldsymbol{m}}_{t+1} = \dfrac{\boldsymbol{m}_{t+1}}{1-\beta_1^{t+1}} \\
\hat{\boldsymbol{v}}_{t+1} = \dfrac{\boldsymbol{v}_{t+1}}{1-\beta_2^{t+1}} \\
\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \dfrac{\hat{\boldsymbol{m}}_{t+1}}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \delta}
\end{cases}
$$

更新式をpythonでそのまま実装しても良いが、本問では空欄前の代入演算子が「+=」であるため、1・2行目をそれぞれ以下のように式変形する。

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$
$$\Leftrightarrow \boldsymbol{m}_{t+1} = \boldsymbol{m}_t + (\beta_1 - 1)\boldsymbol{m}_t + (1-\beta_1)\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$
$$\Leftrightarrow \boldsymbol{m}_{t+1} += (1-\beta_1)\left(\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} - \boldsymbol{m}_t\right)$$

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$
$$\Leftrightarrow \boldsymbol{v}_{t+1} = \boldsymbol{v}_t + (\beta_2 - 1)\boldsymbol{v}_t + (1-\beta_2)\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$
$$\Leftrightarrow \boldsymbol{v}_{t+1} += (1-\beta_2)\left(\left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\frac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} - \boldsymbol{v}_t\right)$$

この他に、本問のコードでは上の理論式での3・4行目部分を学習率を直接更新することで実現している、という違いがある。
これを踏まえて（あ）（い）を正しく埋めたコードは以下の通りである。

```
class Adam:

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)#イテレータiterごとに学習率を更新

        for key in params.keys():
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

その他の最適化手法

【問題】
さまざまな最適化手法について述べた以下の文の空欄を埋めよ。
「SGDは学習率が一定であったため、学習の後半で振動してしまうことがあり非効率であり、この問題に対し、学習率を勾配の二乗和を用いて徐々に減衰させる（あ）が考案された。一方で、SGDには学習が進むにつれてパラメータの更新速度が遅くなるという欠点もあったが、この問題に対し、古い勾配情報を一定の割合で忘れることで学習率を調整した（い）が考案された。

また、慣性項を用いボールが坂を転がるように収束する最適化アルゴリズムであるモメンタムと（い）の長所を併用したアルゴリズムとして（う）が知られている。」

【選択肢】
（あ）の選択肢：
(a)Adam(b)AdaGrad(c)RMSProp(d)ネステロフのモメンタム
（い）の選択肢：
(a)Adam(b)AdaGrad(c)RMSProp(d)ネステロフのモメンタム
（う）の選択肢：
(a)Adam(b)AdaGrad(c)RMSProp(d)ネステロフのモメンタム

【解答】
（あ）(b)
（い）(c)
（う）(a)

【解説】
さまざまな最適化手法のアルゴリズムとその特徴に関する問題である。各最適化手法間の関係は問題文で既に述べられているので、ここでは各アルゴリズムとその特徴についてざっとおさらいしておこう。

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$$

各アルゴリズムについて、$\left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t}$ は最適化したいパラメータWの時刻tにおける値、$\left(\frac{\partial L}{\partial \boldsymbol{W}} \right)$ はWに関する損失関数の勾配とする。
モメンタムは以下のような更新式で表されるアルゴリズムであり、収束するにあたりボールが転がるような物理的な振る舞いをするのが特徴である。（αはハイパーパラメータ、ηは学習率）

$$\begin{cases} \boldsymbol{v}_{t+1} & = \alpha \boldsymbol{v}_t - \eta \left(\frac{\partial L}{\partial \boldsymbol{W}} \right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t + \boldsymbol{v}_{t+1} \end{cases}$$

AdaGradは以下のような更新式で表されるアルゴリズムであり、パラメータの更新ごとに勾配の学習率

$$\begin{cases} \boldsymbol{v}_{t+1} & = \boldsymbol{v}_t + \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t - \eta \dfrac{1}{\sqrt{\boldsymbol{v}_{t+1}}} \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \end{cases}$$

にかかる　　が時間が進むにつれ小さくなり、更新速度が遅くなるという特徴がある。

$$\begin{cases} \boldsymbol{v}_{t+1} & = \boldsymbol{v}_t + \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{W}_{t+1} & = \boldsymbol{W}_t - \eta \dfrac{1}{\sqrt{\boldsymbol{v}_{t+1}}} \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \end{cases}$$

Adamはモメンタム、RMSpropをハイブリッドしたアルゴリズムであり、以下のような更新式で表される。（β_1 β_2 はハイパーパラメータ、δは微小な正数）

$$\begin{cases} \boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)\left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \odot \left(\dfrac{\partial L}{\partial \boldsymbol{W}}\right)_{\boldsymbol{W}=\boldsymbol{W}_t} \\ \hat{\boldsymbol{m}}_{t+1} = \dfrac{\boldsymbol{m}_{t+1}}{1-\beta_1^{t+1}} \\ \hat{\boldsymbol{v}}_{t+1} = \dfrac{\boldsymbol{v}_{t+1}}{1-\beta_2^{t+1}} \\ \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta \dfrac{\hat{\boldsymbol{m}}_{t+1}}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \delta} \end{cases}$$

畳み込み

【問題】
画像認識分野において幅広く用いられている畳み込みニューラルネットワーク(CNN)はフィルター（カーネル）と呼ばれる数値データと画像との畳み込み演算を行うことで画像の特徴量の抽出をしている。
以下のコードは畳み込み後の縦、横それぞれの特徴マップのサイズを求めるコードである。
空欄(あ)(い)に当てはまるものとしてふさわしいものを選べ。
また、変数H,Wはそれぞれ入力特徴マップの縦幅と横幅、f_h,f_wはそれぞれフィルターの縦幅と横幅である。パディング、ストライドの縦幅と横幅は同一でありその値をそれぞれp,sとする。

```
# カーネルを適用した際、出力の画像サイズがどうなるかを計算
out_h = (H + （あ） - filter_h)// （い） + 1
out_w = (W + （あ） - filter_w)// （い） + 1
```

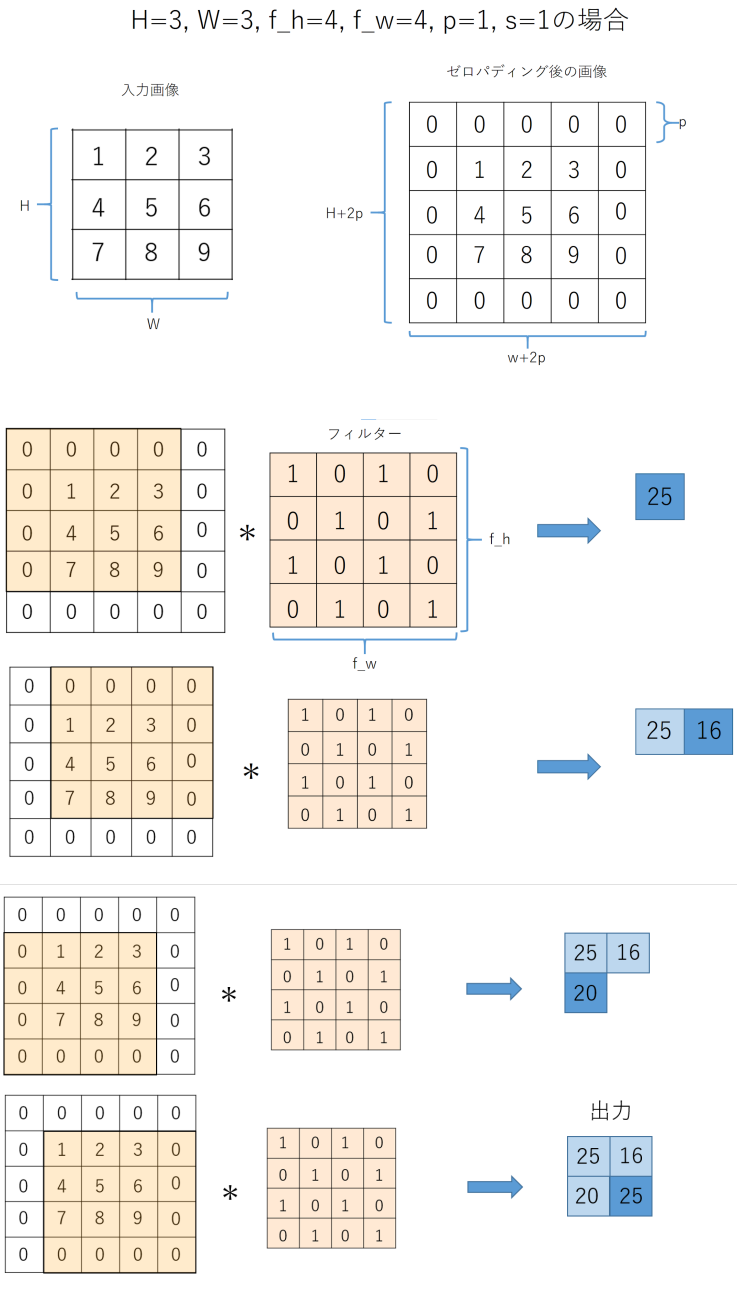
【選択肢】
(a)(あ) 2*p　(い) 2*s
(b)(あ) 2*p　(い) s
(c)(あ) p　(い) 2*s
(d)(あ) p　(い) s

【解答】
(b)(あ) 2*p　(い) s

【解説】
畳み込み演算後の出力サイズは以下の式で求めることができる。

$$(((H + 2p - F\_h)/s) + 1) \times (((W + 2p - F\_w)/s) + 1)$$

具体例で考えてみると以下のような図となる。



今回の例で本問題の式に代入を行うと以下のようになる。

定義式:$(((H + 2p - F\_h)/s) + 1) \times (((W + 2p - Fw)/s) + 1)$

H=3, W=3, f_h=4, f_w=4, p=1, s=1を代入すると

(((3+2*1-4)/1)+1) ×(((3+2*1-4)/1)+1)

=2×2

以上のようにスライドの例のサイズと一致することが分かる。
また問題中の//演算子は割り算の結果を整数型で出力することができる演算子である。

im2col

【問題】
im2colは入力データに対してフィルターを適用する領域を1列に展開を行う関数であり、これによって畳み込み演算をフィルターとの行列積のみで計算を行うことができる。
以下のコードはこのim2colの展開を行う部分の実装である。
空欄(あ)に入るコードとしてふさわしいものを選べ。
またコード中のfilter_h,filter_wはそれぞれフィルターの縦幅と横幅、out_h,out_wは出力サイズの縦幅と横幅、strideはストライド幅であり、縦幅と横幅は同一である。
入力画像であるimgの形状は(バッチサイズ,チャネル数, 画像の縦幅,画像の横幅)であるとし、colは(バッチサイズ, チャネル数, filter_h, filter_w , out_h,out_w)の6次元で初期化されているとする。

```
    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            col[:, :, y, x, :, :] =                    (あ)

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

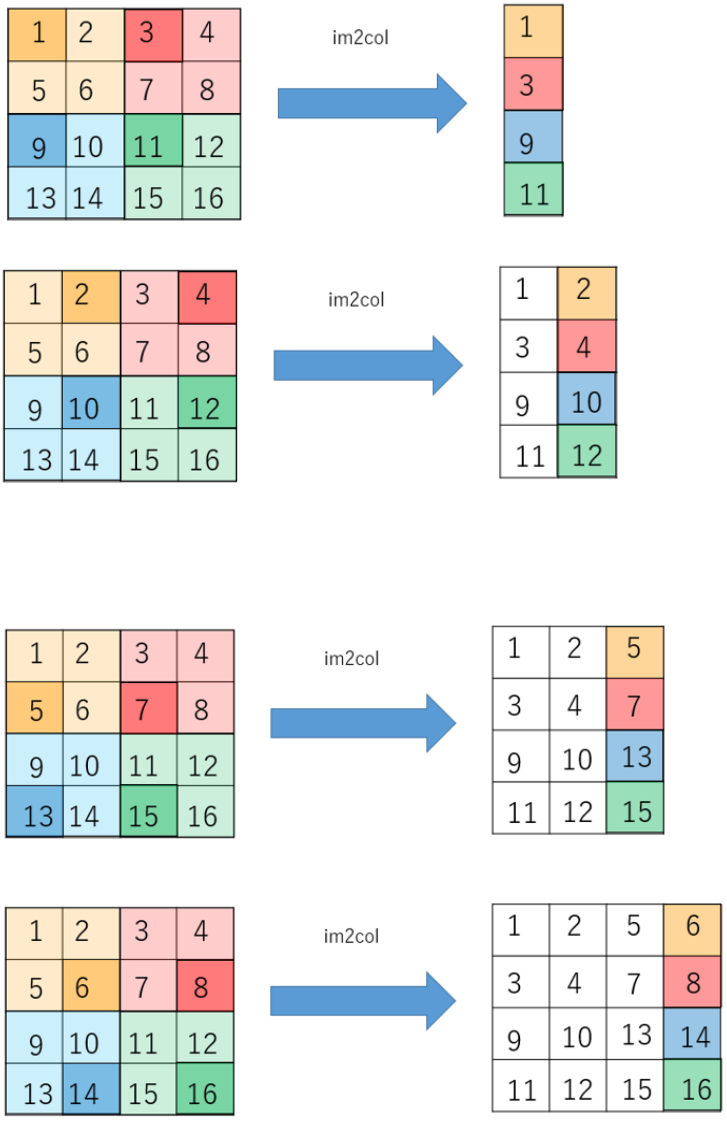
【選択肢】
(a)img[:, : , y_max : out_h, x_max : out_w]
(b)img[: ,: , y_max : stride : out_h, x_max : stride: out_w]
(c)img[:, :, y : y_max, x : x_max]
(d)img[:, :, y : y_max : stride, x : x_max : stride]

【解答】
(d)img[:, :, y : y_max : stride, x : x_max : stride]

【解説】
colの1,2次元目はバッチ数とチャネル数の為、展開の必要はない。
Pythonのスライスにおいて[a:b:c]のような書き方をする場合は対象の配列をaからbまでcずつ飛ばしながら得られた値が入るため、3次元目は画像の縦方向にyからy_maxまでstrideずつずらした時の値、4次元目も同様にxからx_maxまでstrideずつずらした時の値が入る。
つまり適用する各カーネルの左上をスタートとして、1次元に展開しながら値を代入していく処理を行う。

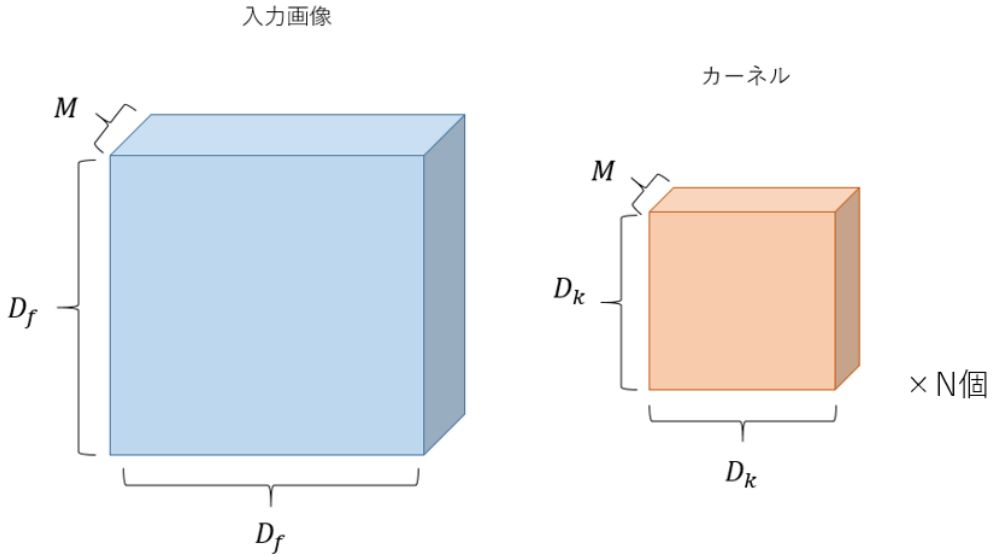
Stride=2,カーネルサイズ=2*2の時の具体的な処理の流れは以下の図の通りである。

ストライド＝２、フィルターサイズ=2*2の場合
Im2colの処理の流れ



畳み込みの計算量

【問題】
画像認識タスクに用いられるCNNの畳み込みの計算量を表した以下の式の中から正しいものを選べ。
また式中における$D_f \times D_f$は入力特徴マップサイズ、$D_k \times D_k$はカーネルサイズ、Mは入力チャンネル数、Nは出力チャンネル数であり
ストライド幅は縦横共に1、パディングを畳み込み前後の特徴マップのサイズが変わらないように適用したとする。

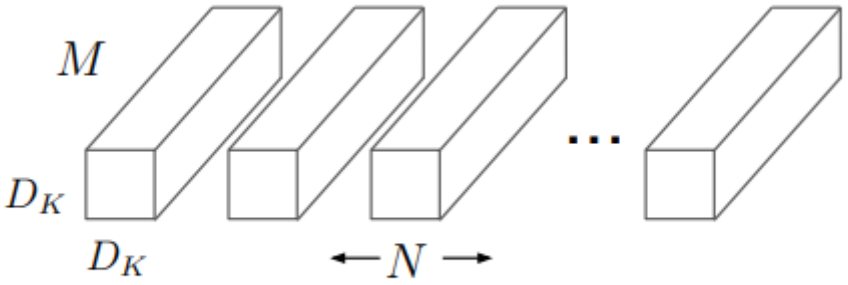


【選択肢】
(a)$D_f \times D_k \times N \times M$
(b)$D_f \times D_k \times M \times M$
(c)$D_f \times D_f \times D_k \times D_k \times M \times N$
(d)$D_f \times D_f \times D_k \times D_k \times M \times M \times N$

【解答】
(c)$D_f \times D_f \times D_k \times D_k \times M \times N$

【解説】
畳み込み計算は$D_f \times D_f$の特徴マップに下図のような$D_k \times D_k \times M \times N$のカーネルを適用する。そのため、畳み込み演算回数は$D_f \times D_f \times D_k \times D_k \times M \times N$である。

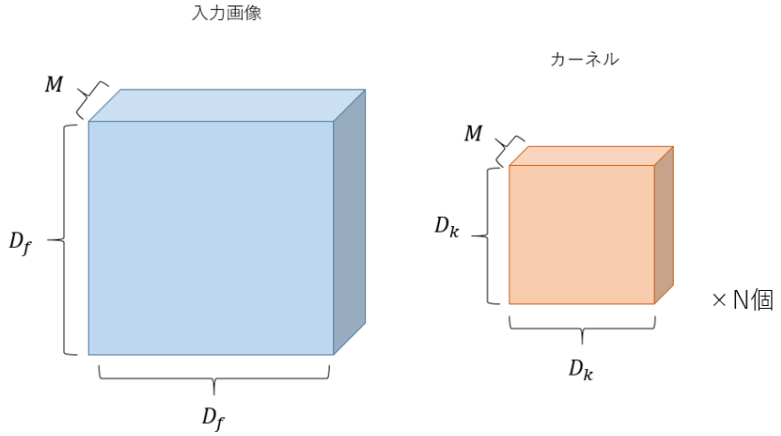
補足:
今回はパディングを畳み込み前後の特徴マップサイズが変化しないように設定している為カーネルごとの適用回数は$D_f \times D_f$回で計算することができる。



(a) Standard Convolution Filters

MobileNet

【問題】
画像認識タスクに用いられるアーキテクチャの一つにMobileNetがある。
MobileNetではDepthwise Separable Convolutionと呼ばれる畳み込み手法を用いている。
これは畳み込みをDepthwise ConvolutionとPointwise Convolutionの2段階に分けることで、全体の計算量を削減する手法である。
Depthwise Convolutionでは特徴マップの平面方向のみを畳み込む為、計算量は(あ)となる。
またPointwise Convolutionはチャネル方向の畳み込みを行うため計算量は(い)となる。
また式中における$D_f \times D_f$は入力特徴マップサイズ、$D_k \times D_k$はカーネルサイズ、Mは入力チャンネル数、Nは出力チャンネル数であり、ストライド幅は縦横共に1、パディングを畳み込み前と後の特徴マップのサイズが変わらないように適用したとする。

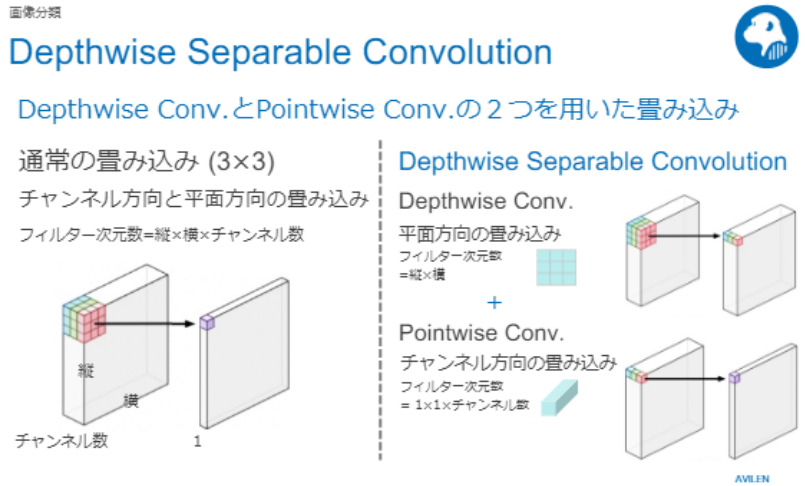


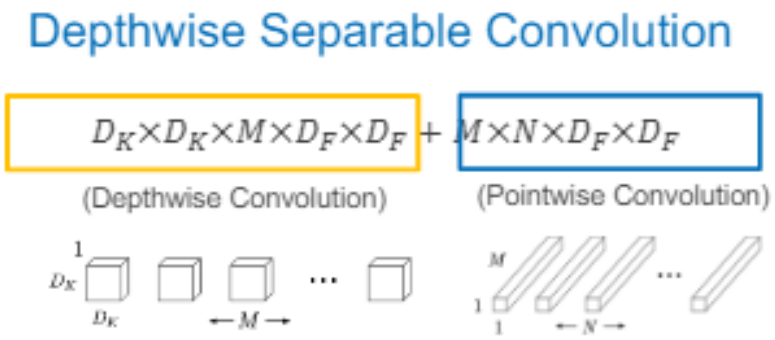
【選択肢】
（あ）の選択肢
(a)$D_f \times D_f \times N$
(b)$D_f \times D_k \times M \times N$
(c)$D_f \times D_f \times D_k \times D_k \times N$
(d)$D_f \times D_f \times D_k \times D_k \times M$

（い）の選択肢
(a)$D_f \times D_f \times M \times N$
(b)$D_f \times M \times N$
(c)$D_k \times D_k \times M$
(d)$D_k \times M \times N$
【解答】
(あ) (d)$D_f \times D_f \times D_k \times D_k \times M$
(い) (a)$D_f \times D_f \times M \times N$

【解説】
通常の畳み込みでは$D_k \times D_k \times M$のサイズのカーネルを$D_f \times D_f$の特徴マップにN回適用するため、計算量は$D_f \times D_f \times D_k \times D_k \times M \times N$となる。
MobileNetでは、Depthwise Separable Convolutionと呼ばれるDepthwise ConvolutionとPointwise Convolutionに分ける畳み込みによって計算量の削減を行った。
Depthwise Convolutionは平面方向のみの畳み込みで、M枚の$D_k \times D_k$のカーネルを$D_f \times D_f$の特徴マップに適用するため、計算量は$D_f \times D_f \times D_k \times D_k \times M$となる。
Pointwise Convolutionはチャネル方向のみの畳み込みで、1×1×Mサイズのカーネルを

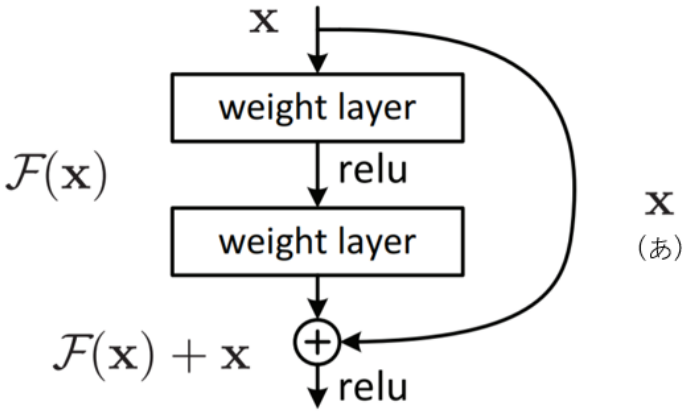
$D_f \times D_f$の特徴マップにN回適用するため計算量は$D_f \times D_f \times M \times N$となる。
他にもMobileNetにはモデルサイズを削減するため、チャネル数Mを削減するwidth multiplierや、計算コストを下げるために解像度を落とすresolution multiplierなどの工夫がある。





ResNetの特徴①

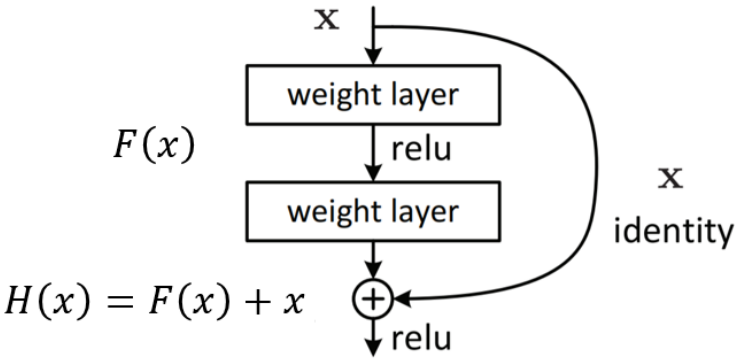
【問題】
2015年に発表されたResnetはそれ以前から問題視されていた、層を多層化することによる勾配消失問題の改善に取り組んだモデルである。
Resnetは層をまたがる結合として(あ) mappingを用いる。これによってResnetはブロック内の入出力の残差を学習することができる。
(あ)に入る単語としてふさわしいものを選べ。



【選択肢】
(a)Identity
(b)Pass
(c)Residual
(d)Response

【解答】
(a)Identity

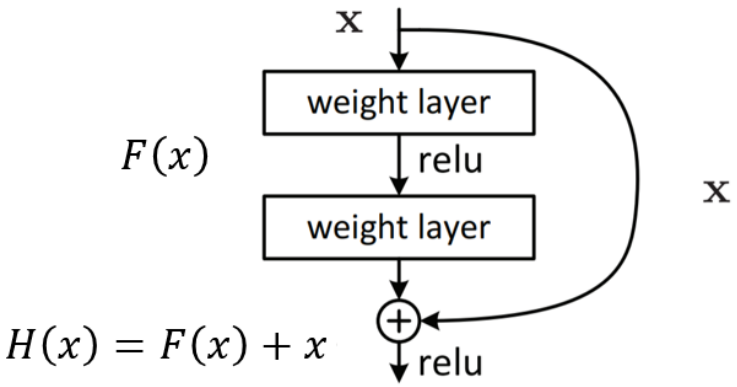
【解説】
Resnetはそれ以前から問題視されていた層を深くすることによる勾配消失問題を解決し、
従来では多くても20層前後であった層数を152層まで深くすることができたモデルである。
勾配消失問題を解決する方法としてIdentity mappingと呼ばれる層をまたぐ接続を行う。
下図のようにxを足し合わせた後の出力をH(x)とすると
H(x)=F(x)+xと表すことができる。
これを学習対象であるF(x)について表すとF(x)=H(x)-xと表すことができ、これは出力H(x)とxとの残差(Residual)を学習することからResnetと呼ばれている。



$$F(x) = H(x) - x$$

ResNetの特徴②

【問題】
ResNetに導入されたResidual　Blockによって層を深くしても勾配消失を改善することができた理由を述べた次の文のうち、正しいものを2つ選べ。

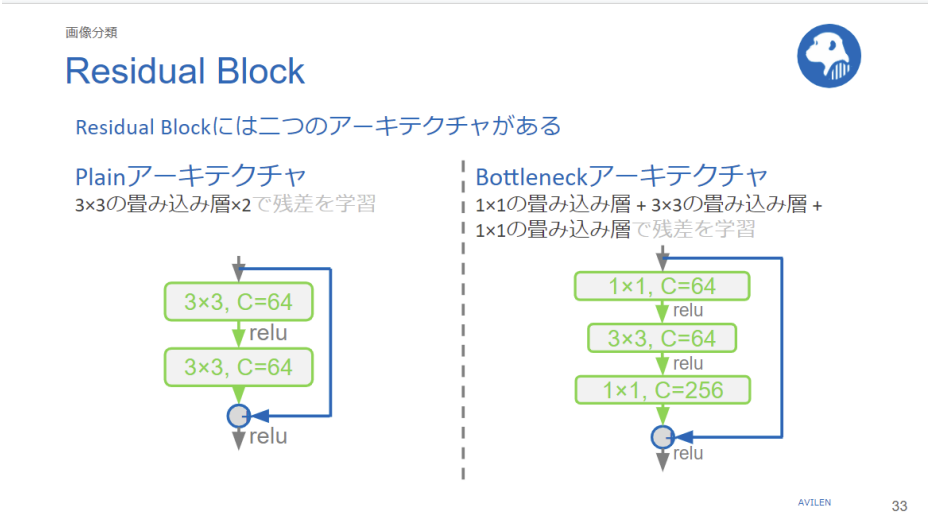


【選択肢】
(a)出力層からの微小な勾配もすべての層を通らずにskip connectionを通りながら逆伝播していくことにより、勾配の消失を防いでいる。
(b)出力H(x)はF(x)にxを足していることから逆伝播時に勾配が小さくならず、勾配消失を防いでいる。
(c)ブロックへの入力にこれ以上変換の必要がない場合の重みは1となり、勾配がそのまま伝播することから勾配消失を防いでいる。
(d)skip connectionにより学習対象のパラメータが減ることから勾配消失を防いでいる。

【解答】
(a)出力層からの微小な勾配もすべての層を通らずにskip connectionを通りながら逆伝播していくことにより、勾配の消失を防いでいる。

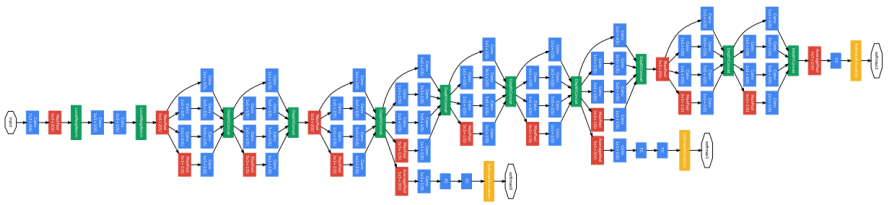
(b)出力H(x)はF(x)にxを足していることから逆伝播時に勾配が小さくならず、勾配消失を防いでいる。

【解説】
従来手法では誤差逆伝播時に各レイヤーを通るごとに1以下の勾配をかけ合わせることにより、勾配が徐々に小さくなっていってしまう勾配消失問題が問題視されていたが、ResNetでは、勾配がskip connection (identity mapping)を通ることにより、勾配が小さくなることなく伝播することができ、勾配消失を防いだため(a)と(b)は正しい。
また、ブロックへの入力にこれ以上変換の必要がない場合の重みは1ではなく、0となる。(つまりskip connectionを通ってきた入力xをそのまま伝播する)これは浅いCNNで十分学習が行えて深い中間層が不要な場合、その層の重みは0になることで実質的にその層は無かったことといえる。
その他、skip connectionは層をまたぐ結合であるためパラメータは減少しない。
このResnetにはPlainアーキテクチャとBottleneckアーキテクチャの2つのアーキテクチャが存在する。
Plainアーキテクチャは下図左のように3*3の畳み込み2つで構成されていることに対し、Bottleneckアーキテクチャは下図右のように1*1，3*3，1*1の畳み込みで構成されており、最初の1*1，　3*3の畳み込み層で次元圧縮を行い、最後の1*1畳み込み層で次元を入力と同じ次元に復元することからBottleneckアーキテクチャという名前がついている。



GoogLeNet

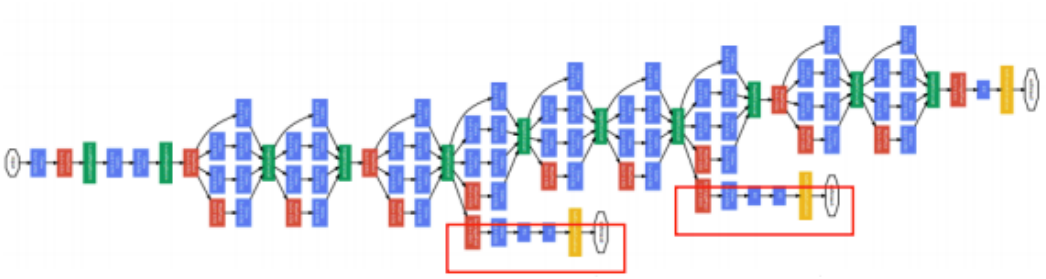
【問題】
画像認識に用いられるCNNのアーキテクチャの1つにGoogLeNetがある。GoogLeNetの特徴について述べた以下の文のうち誤っているものを選べ。



【選択肢】
(a)Inception moduleは複数のサイズの異なるフィルタにより構成されている。
(b)GoogLeNet はInception moduleを複数積み重ねてできたネットワークである。
(c)Auxiliary　lossによる中間層における損失計算を行うことによって勾配消失の防止や正則化が期待できる。
(d)Auxiliary lossを用いることによってネットワークの計算量削減の効果を得られる。

【解答】
(d)Auxiliary lossを用いることによってネットワークの計算量削減の効果を得られる。

【解説】



Inception moduleでは1*1や3*3,5*5などのフィルターサイズの異なる畳み込み処理を統合させている。また、1*1のフィルターを用いることで次元削減の効果が期待できるため0でないパラメータ数が減り、計算効率が向上する。
GoogLeNetはこのInception moduleを複数重ねてできたネットワークである。
他にも、GoogLeNetは上記の図の2つの赤枠のようなAuxiliary classifier(Auxiliary loss)を持つ。これらはメインのネットワークと分離された別の学習器とみなすことができるため、勾配消失の防止や正則化が期待できる。しかし、このような学習器を追加している為、計算量の削減は期待できない。

DenseNet

【問題】
CNNのアーキテクチャの一つにDenseNetがある。
DenseNetについて述べた次の文のうち誤っているものを選べ。

【選択肢】
(a)ResNetは少し前の層の出力を後方の層へ加算することに対し、DenseNetは前方すべての層の出力を後方の層へと入力を行う。
(b)各層の出力チャネル数(成長率)をkとして入力チャネル数をk_0、層数をlとするとDense Blockの出力チャネル数は$k_0 + kl$と表すことができる。
(c)各Dense Blockの間にはダウンサンプリングを目的としたTransition Layerが存在する。
(d)成長率kはハイパーパラメータであり、小さくすることによってパラメータの更新量は小さくなる。

【解答】
(d)成長率kはハイパーパラメータであり、小さくすることによってパラメータの更新量は小さくなる。

【解説】
DenseNetはResNetを改良し、よりレイヤー間の情報伝達を強化したモデルである。
下図のようにRes　Blockでは自身の層より少し前方の出力を加算することに対し、Dense　Blockは前方すべての層の出力を後方の層へと入力を行う。この時のDense　Block内のレイヤーの出力チャネル数は成長率と呼ばれ、ハイパーパラメータとして設定する。成長率の調整を行うことにより、ネットワークが大きくなりすぎることを防ぐ目的で設定を行う。その為、パラメータ更新量を調節するものではないことから誤り。
DenseNetではDense Block内にskip connectionがある影響からダウンサンプリング(プーリング)ができない問題を、Transition　Layerと呼ばれる畳み込み層とpooling層を組み合わせたレイヤーを各Dense Blockの間に組み込むことで解決した。

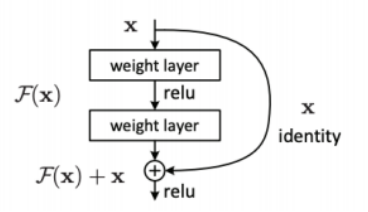


図1:ResBlock

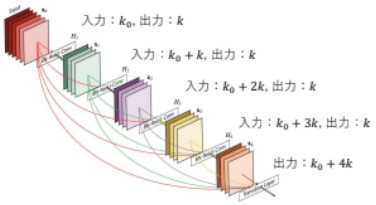


図2:DenseBlock

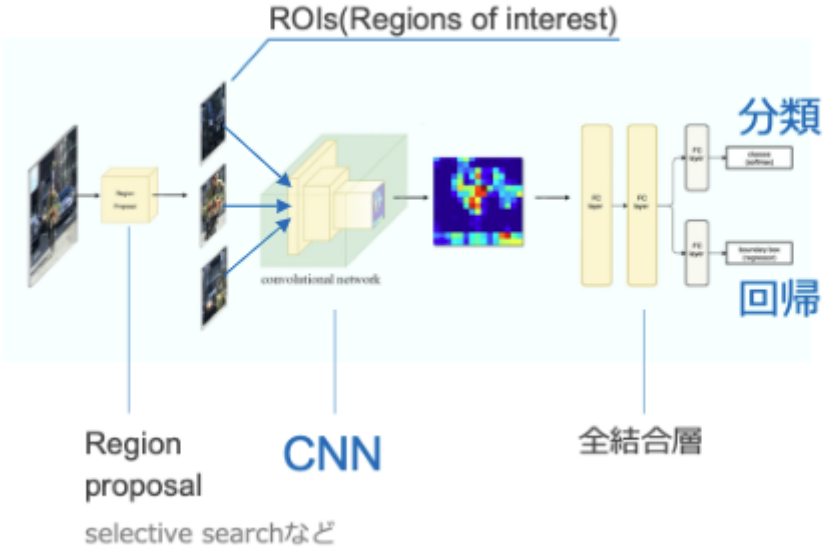
R-CNN

【問題】
深層学習を用いた物体検出アルゴリズムの先駆けとしてR-CNNがある。
これについて述べた以下の文のうち誤っているものを選べ。

【選択肢】
(a)候補領域提案を行ったのち固定サイズにリサイズしてCNNへ入力する。
(b)候補領域提案にはRPN(Region Proposal Network)を用いている。
(c)CNNの出力に対しSVMでクラス分類，回帰モデルで正確な領域の推定を行う。
(d)特徴抽出やクラス分類など各学習の目的ごとに個別に学習させる必要がある。

【解答】
(b)候補領域提案にはRPN(Region Proposal Network)を用いている。

【解説】
R-CNNでは候補領域提案にSelective Searchを用いている為、誤り。
R-CNNの処理の流れは入力画像に対してSelective　Searchを用いて候補領域提案を行い、提案された各領域を固定サイズ(論文では227*227)にリサイズをしてCNNによる特徴抽出を行う。
このCNNの出力をSVMと回帰モデルの2つに入力し、SVMで各領域のクラス分類を行い、回帰モデルで正確な領域の推定を行う。
しかしR-CNNはCNNやSVMを同時に学習することができない他、候補領域ごとにCNNへ入力を行う為、学習速度、認識速度ともに遅い。



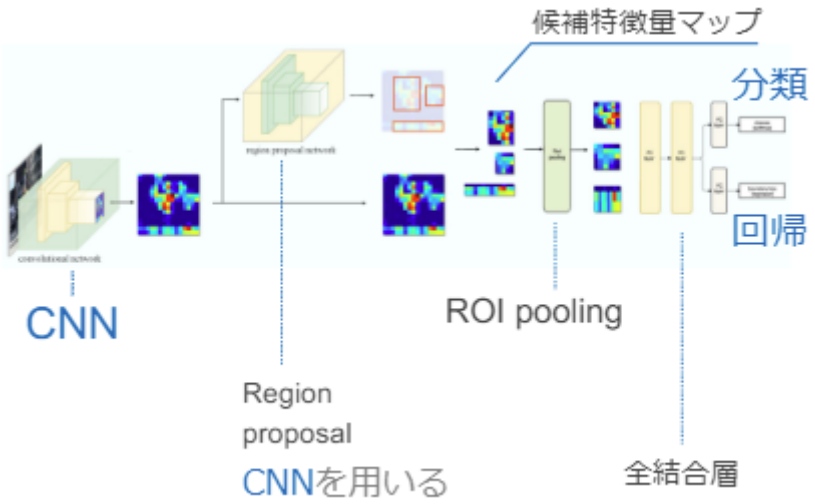
Faster R-CNN

【問題】
深層学習を用いた物体検出アルゴリズムの先駆けとなったR-CNNを改良したモデルとしてFaster R-CNNがある。
Faster R-CNNについて述べた次の文のうち誤っているものを選べ。

【選択肢】
(a)候補領域提案を行ったのちにCNNに入力する。
(b)Faster R-CNNではEnd-to-Endの学習が可能となった。
(c)RoI Poolingは特徴マップを固定サイズにリサイズすることができる。
(d)候補領域提案にはRPN(Region Proposal Network)を用いている。

【解答】
(a)候補領域提案を行ったのちにCNNに入力する。

【解説】
R-CNNの欠点として候補領域ごとにCNNへ入力を行う為、処理速度が遅いという欠点があった。しかしFaster R-CNNでは入力画像をCNNへと入力してから、その出力を用いてRegion Proposal Networkで候補領域提案を行う。そのため、回答はR-CNNと同様の手順の為誤り。
Faster R-CNNではRegion Proposal Networkで候補領域提案を行った後、それらをRoI Poolingを用いて固定サイズに変換し、座標回帰とクラス分類を行う。
Faster R-CNNでは候補領域提案にCNNを利用したRegion Proposal Networkを用いていることや、従来のR-CNNのようにクラス分類にSVMなどの別のモデルを使用していないことから、誤差逆伝播によって一度にすべてのパラメータが更新できるEnd-to-Endな学習が可能となった。



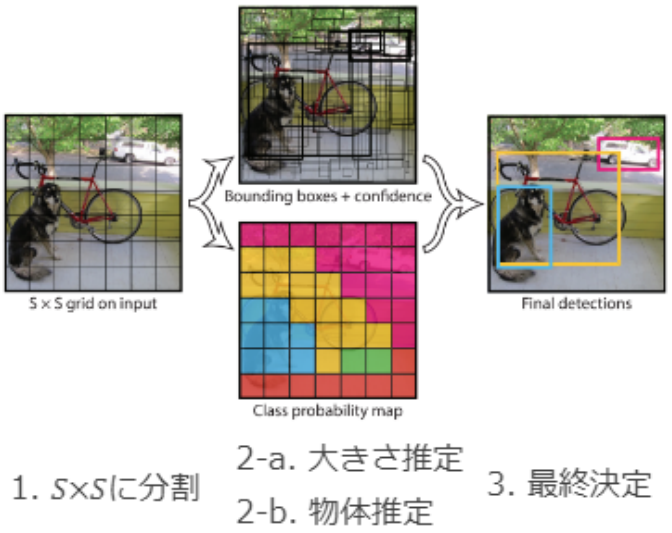
YOLO

【問題】
物体検出アルゴリズムの1つであるYOLOについて述べた次の文のうち、誤っているものを選べ。

【選択肢】
(a)入力画像を複数のセルに分割し、セルごとに物体の分類確率を推定する。
(b)non-maximum suppressionによって同じ物体クラスでIoU値が閾値以上のバウンディグボックスを結合する。
(c)セルごとに一定数のバウンディングボックスを推定することにより密集した物体の検出を行うことができる。
(d)識別と検出を同時に行うことができるためFaster R-CNNと比較して高速である。

【解答】
(c)セルごとに一定数のバウンディングボックスを推定することにより密集した物体の検出を行うことができる

【解説】
YOLOは入力画像を複数のセルに分割し、セルごとに物体の分類確率の推定を行っている為、1つのセルに対して1つのクラスしか割り当てることができない。そのため、1つのセルの中に複数の物体が密集している場合などはうまく検出を行えないため誤り。
YOLOの処理の流れは、画像を複数のセルに分割したのち、セルごとに物体を分類する。
そのセルごとに最大2つのバウンディングボックスを出力し、同じ物体クラスで閾値以上のIoU値であるバウンディングボックスに対し、non-maximum suppressionを用いて結合を行う。



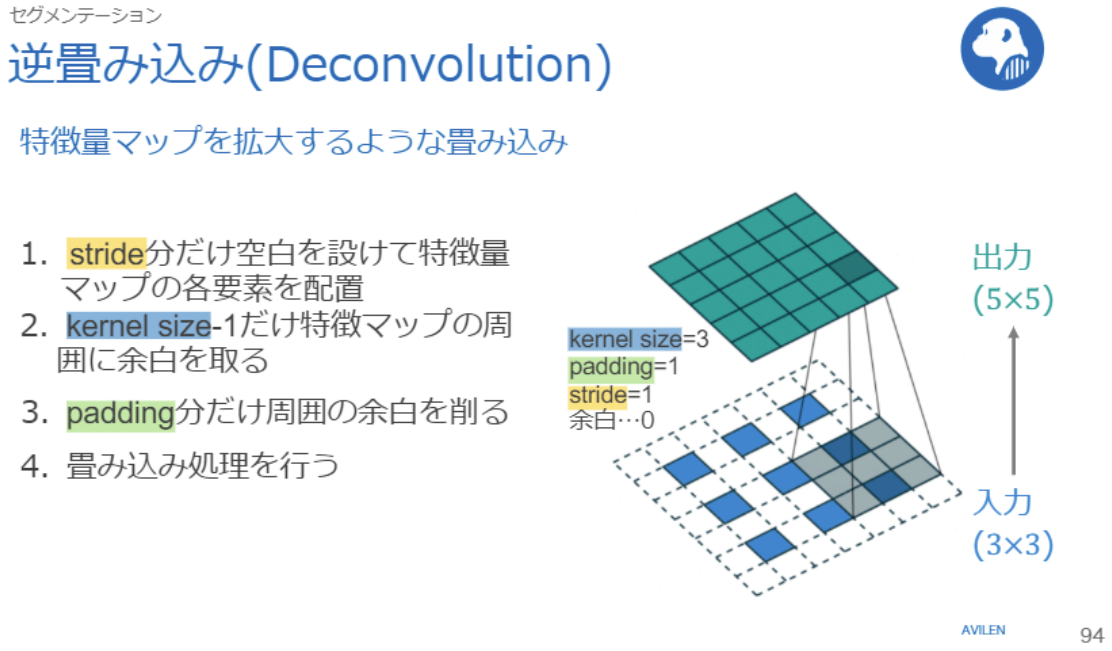
FCN

【問題】
CNNを用いた画像のセグメンテーションを行う手法の1つにFCNがある。
このFCNについて述べた以下の文のうち誤っているものを選べ。

【選択肢】
(a)FCNでは任意のサイズの画像入力が可能となった。
(b)FCNは1*1畳み込みを全結合層へと置き換えたものである。
(c)逆畳み込みを用いることで圧縮された特徴のアップサンプリングを行う。
(d)畳み込み時のpooling出力はスキップ接続によってアップサンプリング時に足し合わせる。

【解答】
(b)FCNは1*1畳み込みを全結合層へと置き換えたものである。

【解説】
FCNは全結合層を1*1畳み込みと置き換えることで任意サイズの画像入力を可能としたモデルであるため誤り。
また圧縮された特徴マップをアップサンプリングするために逆畳み込み(Deconvolution)を用いている。逆畳み込みの処理手順は下図の通りである。
他にも圧縮された特徴をそのまま逆畳み込みを適用するだけでは出力が粗くなってしまうため、スキップ接続によって畳み込み時のpooling出力を逆畳み込み時に足し合わせている。



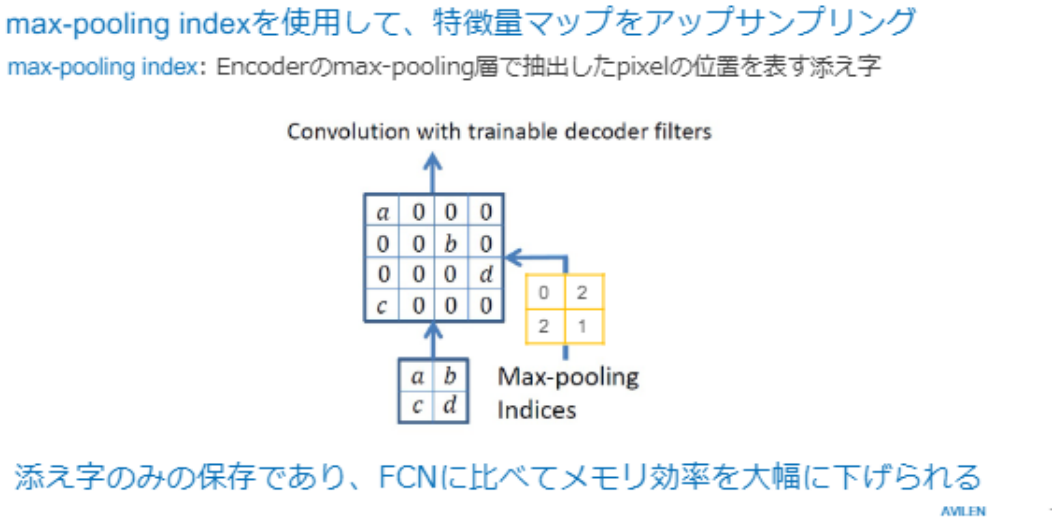
SegNet

【問題】
FCNはpoolingの出力をメモリに保存する必要があることから、メモリ効率が悪いという問題点を
SegNetは(あ)を用いることでこの問題を改善した。

【選択肢】
(あ)に当てはまる単語としてふさわしいものを選べ。
(a)Average-Pooling index
(b)Max-Pooling index
(c)Dilated Convolution
(d)Depthwise Separable Convolution

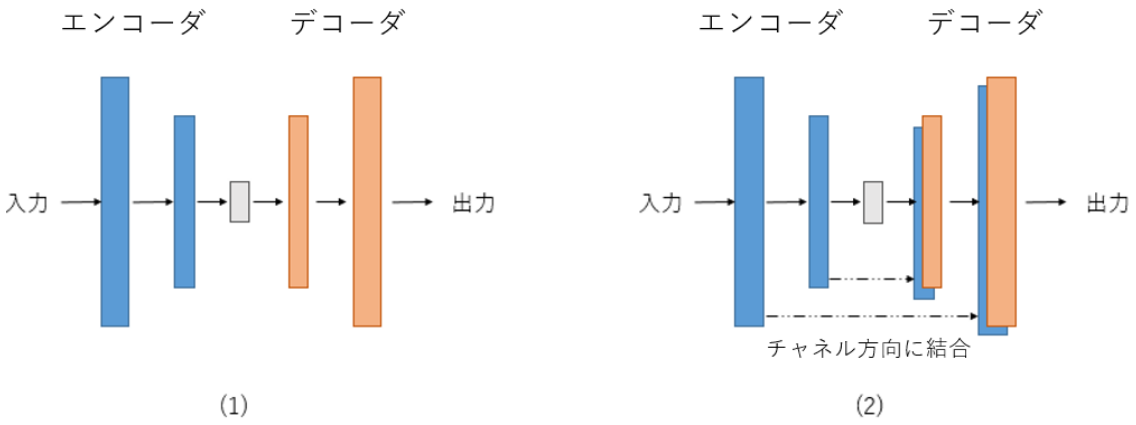
【解答】
(b)Max-Pooling index

【解説】
FCNでは各poolingの出力を保持することでメモリ効率が悪くなってしまう問題を、
SegNetはMax-Pooling index と呼ばれるMaxpooling層で抽出したpixelのインデックスのみ保存することによって解決した。
他選択肢のDilated　Convolutionはフィルターと画像との積を計算する間隔を開ける畳み込み手法で小さなカーネルサイズでも1回の畳み込みで広い部分の特徴を捉えることができる。Wave NetなどにはDilated　Convolutionの派生形であるDilated　Causal　Convolutionが用いられている。
Depthwise　Separable　ConvolutionはMobile　Netに用いられている畳み込み手法で、畳み込みを空間方向とチャネル方向に分けて行う手法である。
Average-Pooling indexは造語であり、SegNetには用いられていない。



U-Net

【問題】
画像のセグメンテーションを行うネットワークの一つにU-Netがある。
以下の図についてU-Netを表した画像は(あ)でありU-Netは (い)によって
(う)の効果がある。
(あ)、(い)、(う)に入るものとして適切なものを選べ。ただし(2)の点線はエンコーダの情報をデコーダ側にチャネル方向で結合させるものである。



【選択肢】
(a)(あ)1 (い)ショートカット接続 (う) メモリを節約
(b)(あ)2 (い)ショートカット接続 (う)エンコーダで失われる局所的な空間情報を補完する
(c)(あ)1 (い) Max-Pooling index (う) エンコーダで失われる局所的な空間情報を補完する
(d)(あ)2 (い) Max-Pooling index (う)メモリを節約

【解答】
(b)（あ）2　(い)ショートカット接続　(う)エンコーダで失われる局所的な空間情報を補完する

【解説】
U-Netは以下の図のようなショートカット接続によって、エンコーダの各層の出力デコーダ側の各層のチャンネル方向側に結合することにより、エンコーダ部分で失われてしまう局所的な空間情報(エッジなど)を補完することができるネットワークである。
(2)の点線はエンコーダ側からデコーダ側に結合を行うショートカット接続を表していることから(2)がU-Net である。
Max-Pooling　　indexはSegNetに用いられた工夫であり、Maxpooling層で抽出したpixelのインデックスのみ保存することでメモリ効率を向上させるものである。