



Style Transferとは

- 画風を変換するアルゴリズムのこと(2015年8月に公開)
 - 2枚の画像を入力として, 片方をコンテンツ画像, 片方をスタイル画像とする
 - コンテンツ画像に書かれた物体の配置をそのままにして, 画風をスタイル画像に変換した画像を生成



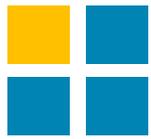
コンテンツ画像



スタイル画像



生成画像

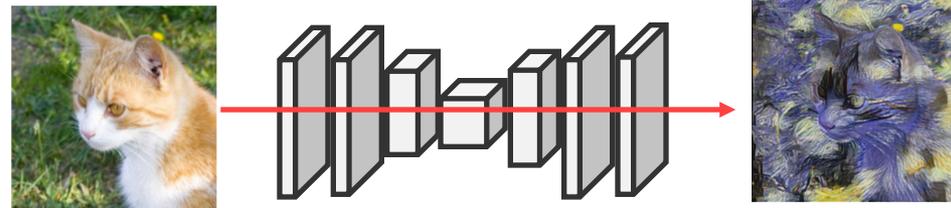


本研究の目的

「Neural Style Transferをモバイル上に実装」

- 既存手法

- 1つのモデルで1つのスタイル
- 学習に時間を要する
- 消費メモリの増大



- 提案手法

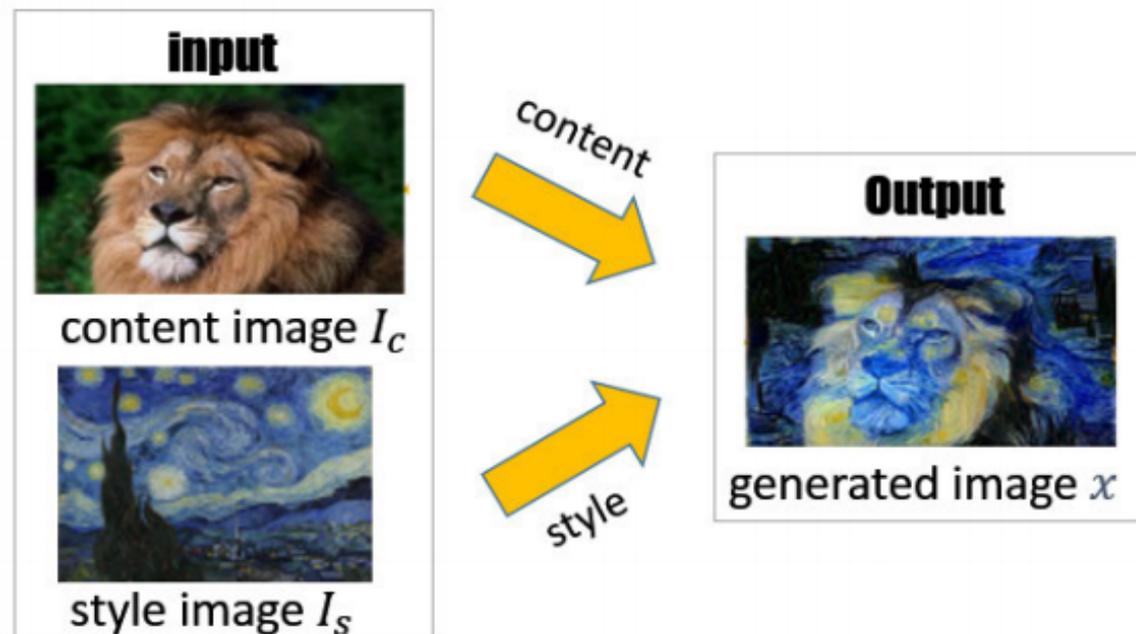
- 1つのモデルで複数のスタイルを変換可能
- モバイル上でのリアルタイム画風変換



関連研究

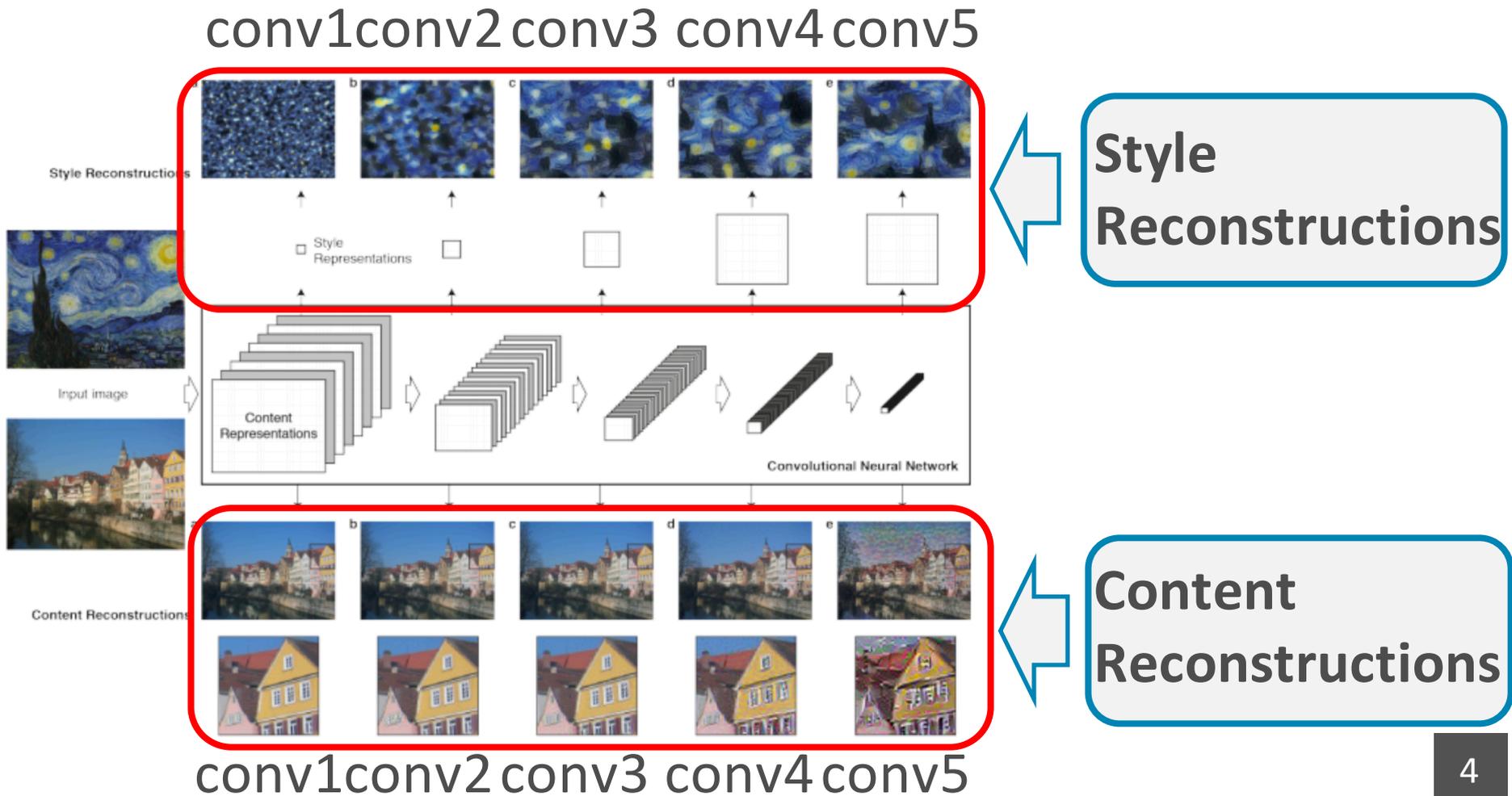


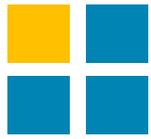
- Neural Style Transferを提案 [Gatys+ 2015]
 - 画像のコンテンツ(形状)を保持したまま,
スタイル(画風)を変化
 - AIによる芸術的画像の生成



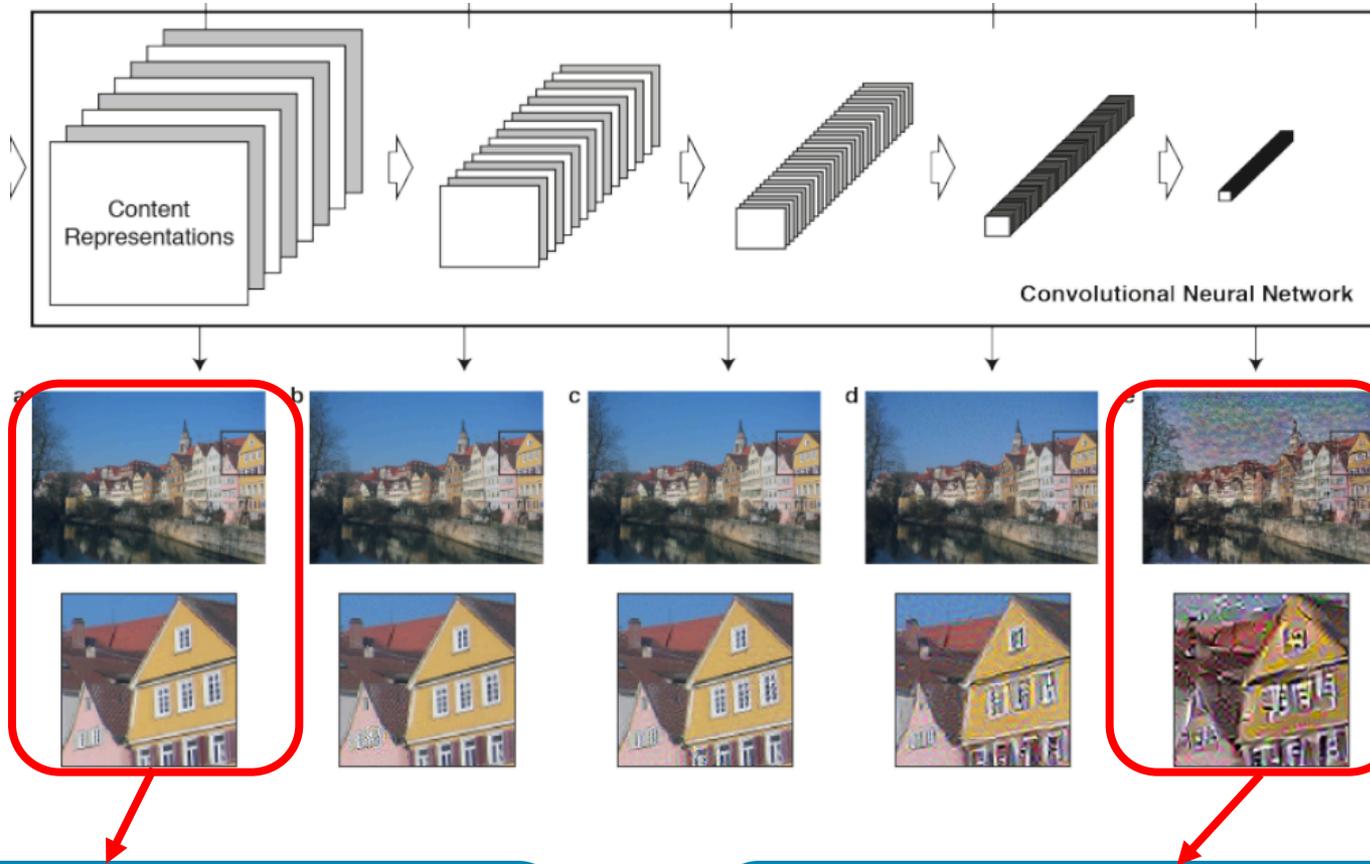


各レイヤの特徴マップから画像を再構築





コンテンツ画像の特徴マップ再構築

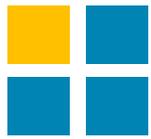


浅いレイヤ

- ・ 正確な元画像の情報を保持

深いレイヤ

- ・ **空間的な位置関係はそのまま**
→ 画像が持つ重要な情報を保持.
- ・ **正確な形などの情報は弱まる**



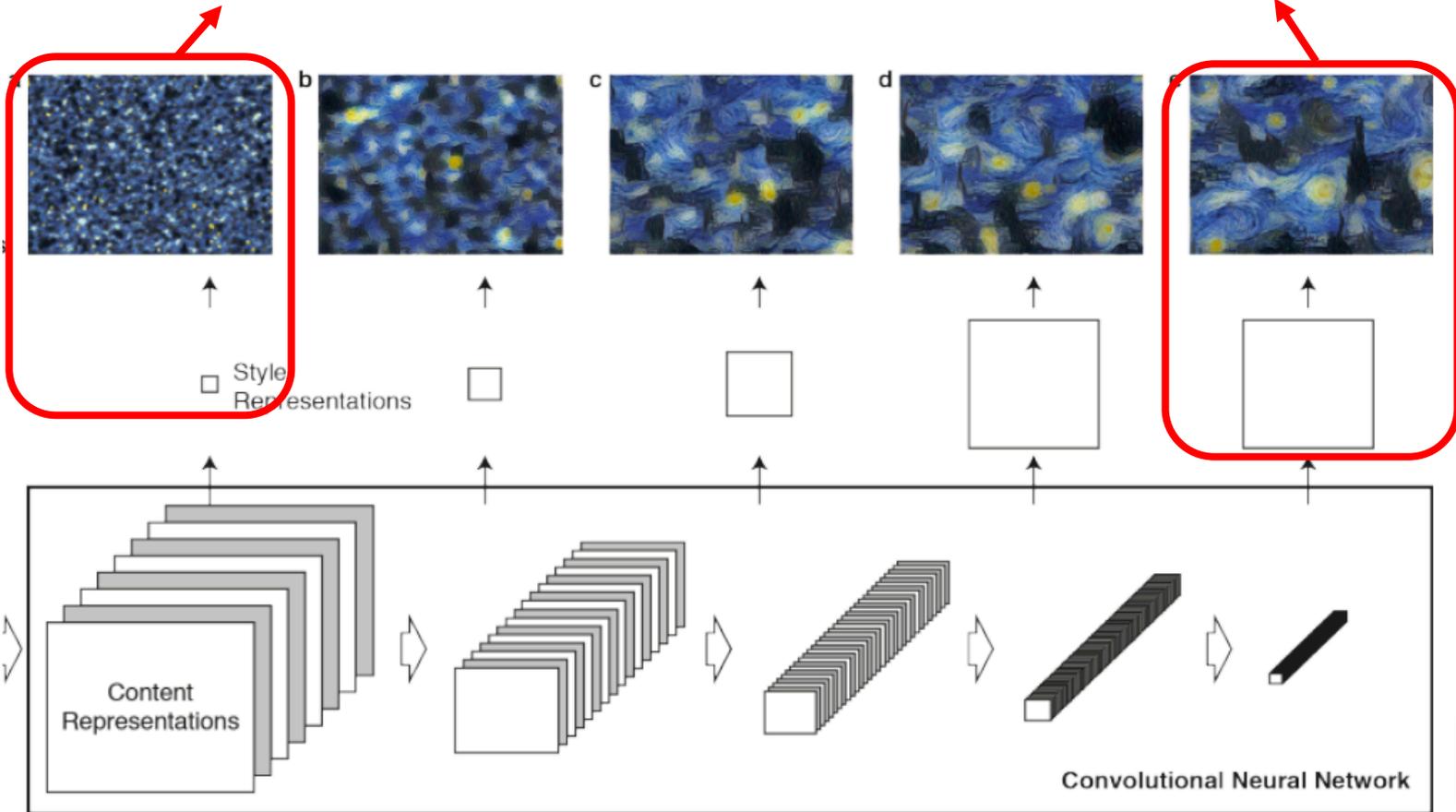
スタイル画像の特徴マップ再構築

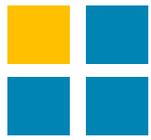
浅いレイヤ

- ・ 画像の**細かな**パターン情報を保持

深いレイヤ

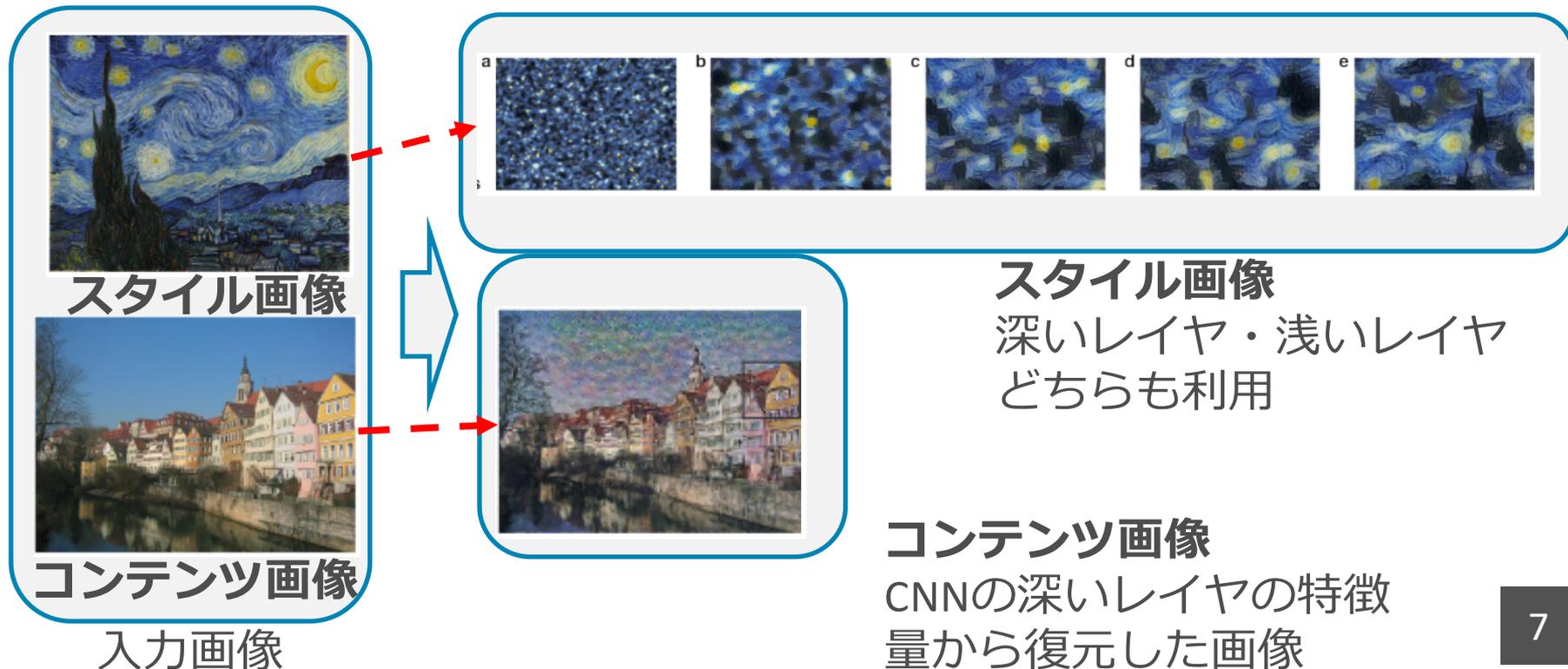
- ・ 画像の**大きめ**の空間パターンを保持





Gatysらの手法のポイント

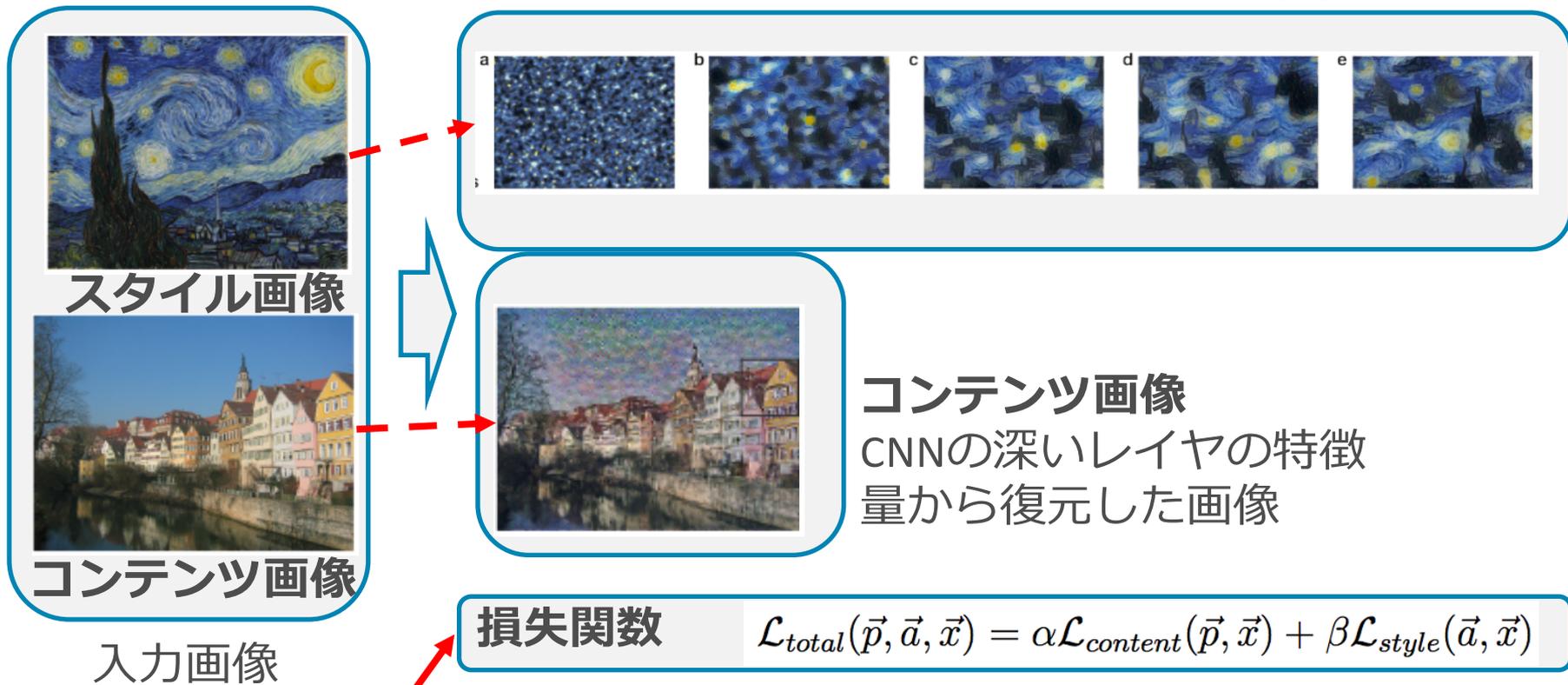
- 情報が弱まっている部分を，別の画像の画風(スタイル)に変換すれば，コンテンツ画像の形状を保持したまま，別のスタイルに画像を変換できるのでは？





損失関数

スタイル画像
深いレイヤ・浅いレイヤ
どちらも利用



コンテンツ画像の外形を保持しつつ、スタイルを変換したい...



最小化



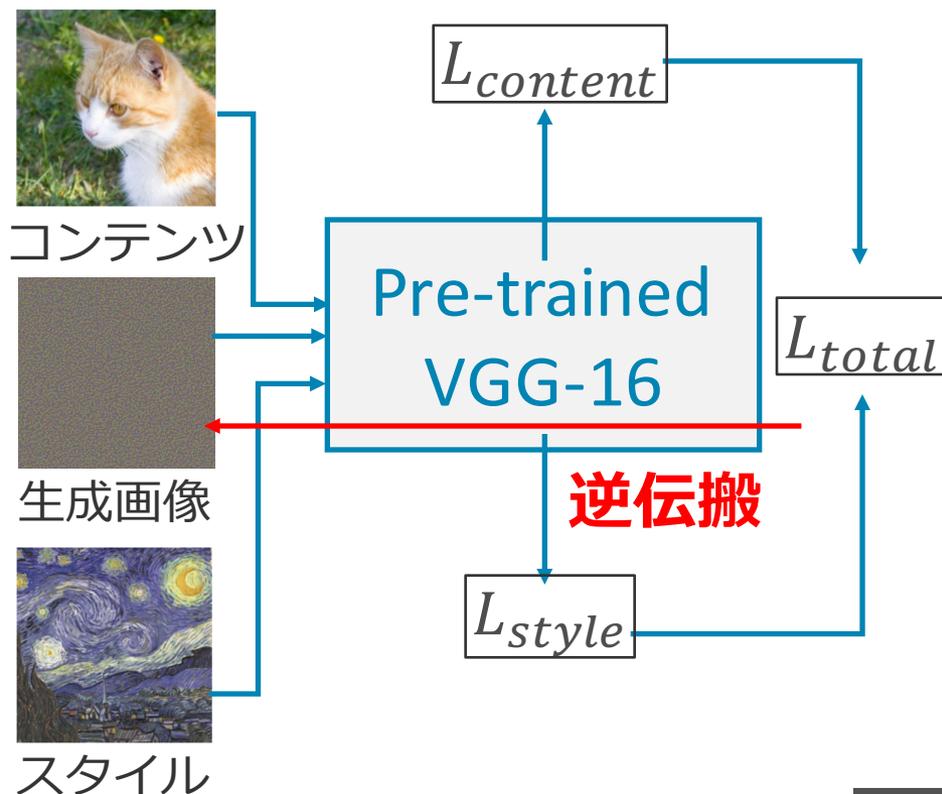
Gatysらの手法まとめ



- 損失関数に学習済みCNNを利用
- 生成画像は一様乱数で初期化
- 順伝搬 & 逆伝搬を反復

- **問題点**

- 生成に**時間**がかかる
- GPU利用で**数十秒**程度



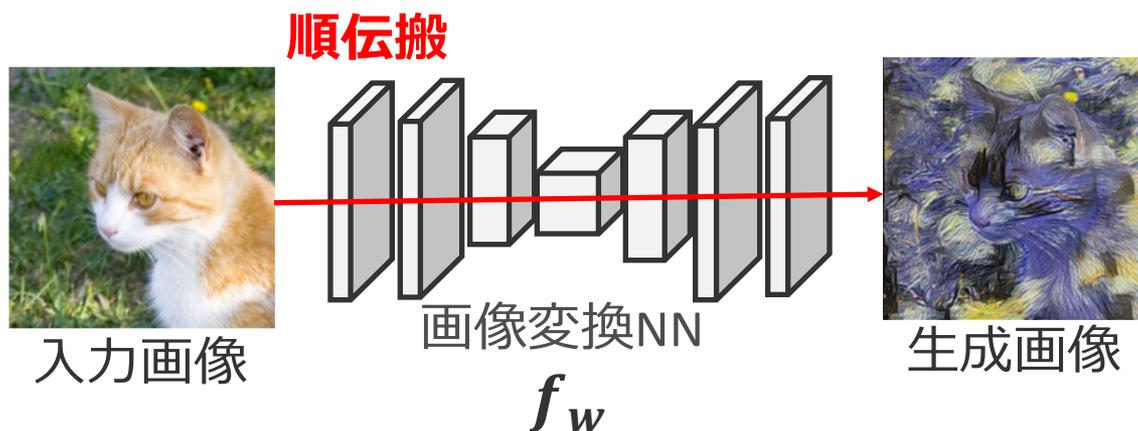
$$\operatorname{argmin} L_{total} = \alpha L_{content} + \beta L_{style}$$

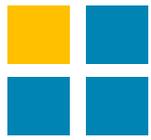


関連研究



- 順伝搬でスタイル変換できる手法を提案 [Johnson+ 2016]
 - 入力画像のスタイルを変換するNN (f_w) を学習
 - 生成時は順伝搬のみなので高速

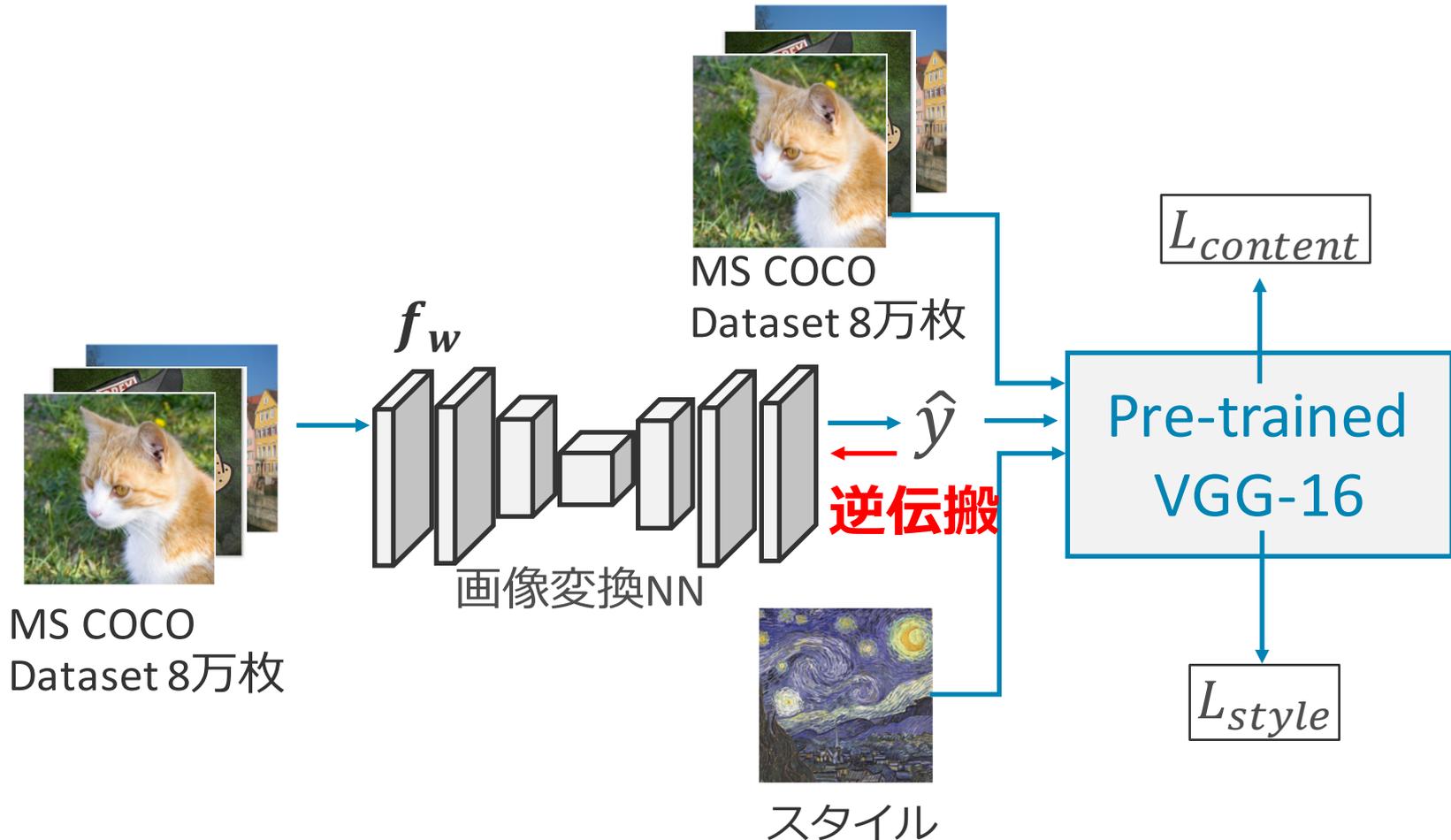




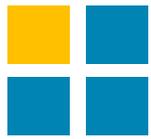
Johnsonらの手法



- MSCOCO 8万枚 + スタイル画像で f_w を学習



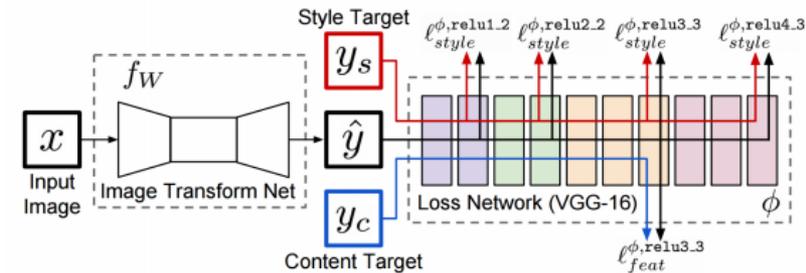
$$\operatorname{argmin} \hat{y} = \alpha L_{content} + \beta L_{style} + \text{正則化項}$$



Johnsonらの手法まとめ

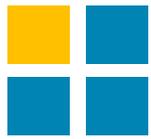


- 特定スタイルを変換可能なCNNを学習
- 順伝搬でスタイル変換可能
 - 非常に高速に画像生成可能に



- 問題点

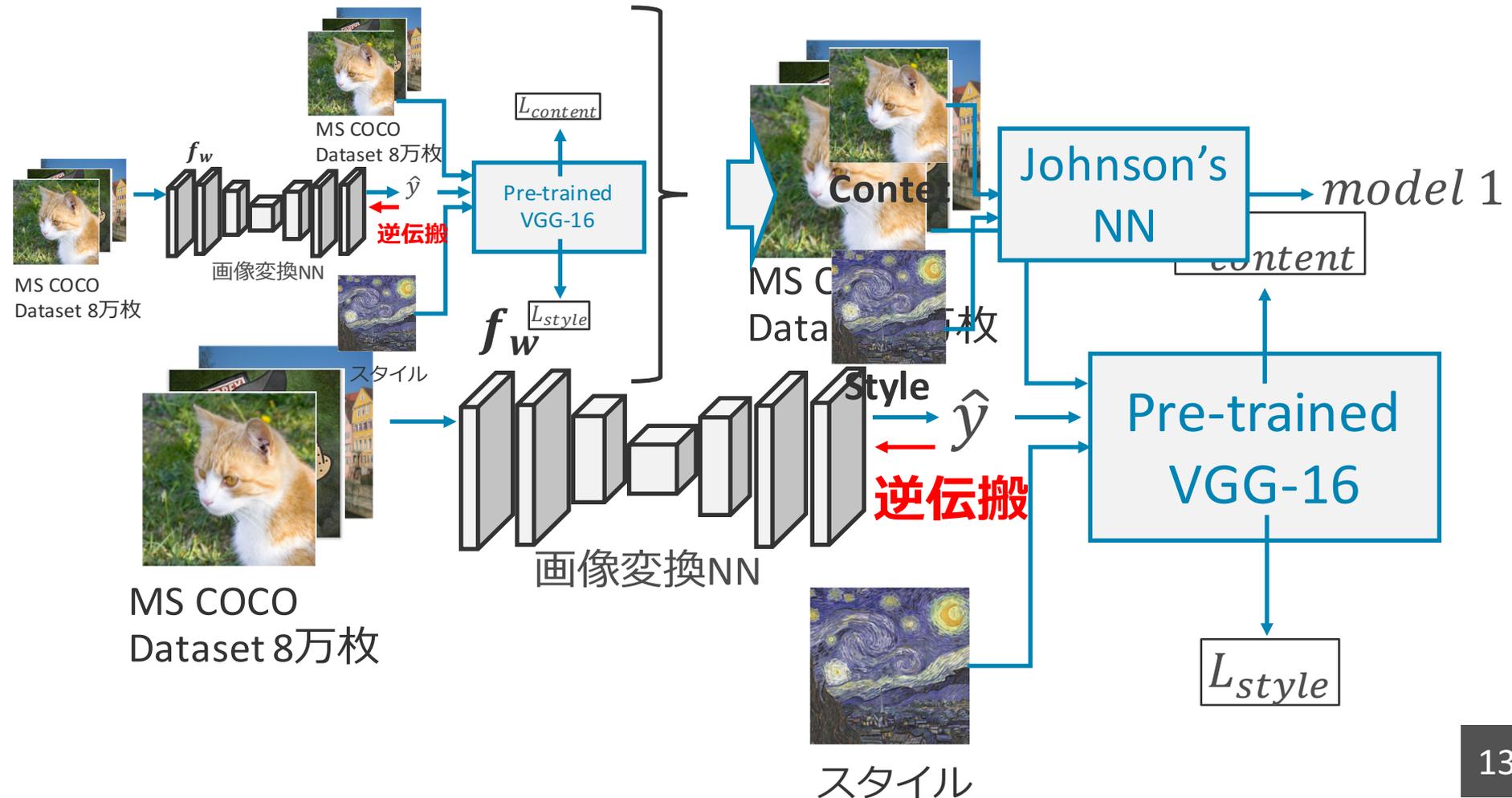
- 1つのモデルで1つのスタイル
- 学習に時間を要する
- 消費メモリの増大



Johnsonらの手法の問題点



● 1つのモデルで1つのスタイル

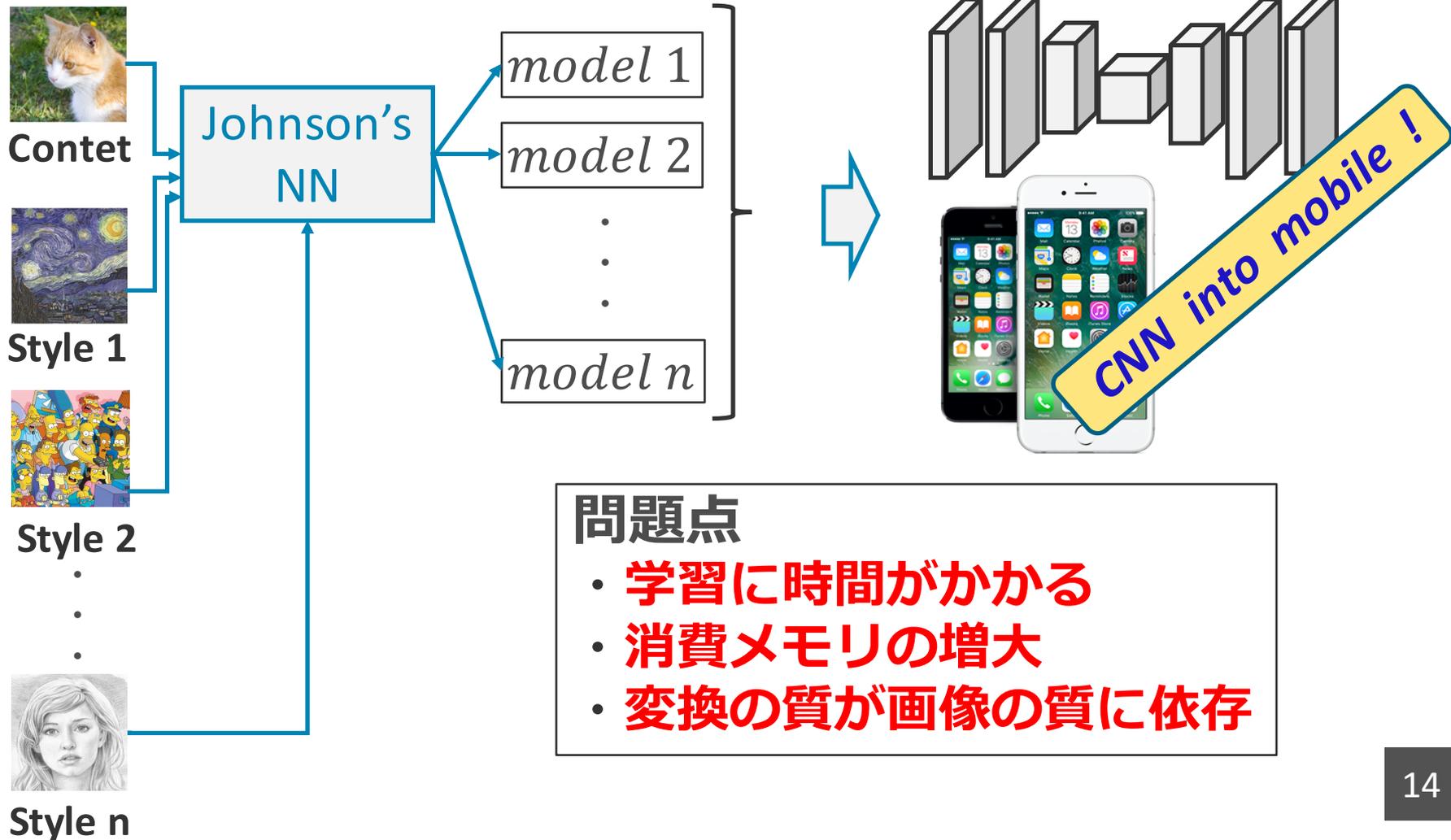




Johnsonらの手法の問題点

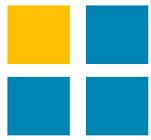


● 1つのモデルで1つのスタイル



問題点

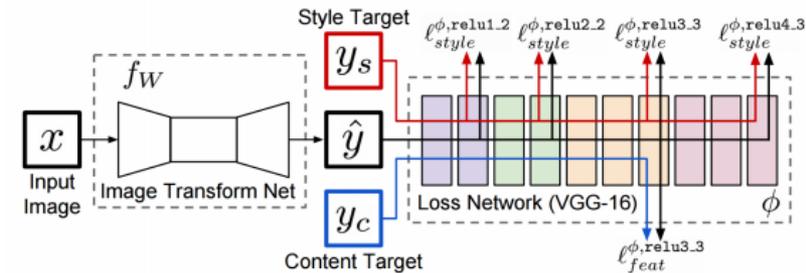
- 学習に時間がかかる
- 消費メモリの増大
- 変換の質が画像の質に依存



Johnsonらの手法まとめ(再掲)



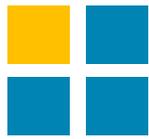
- 特定スタイルを変換可能なCNNを学習
- 順伝搬でスタイル変換可能
 - 非常に高速に画像生成可能に



● 問題点

- 1つのモデルで1つのスタイル
- 学習に時間を要する
- 消費メモリの増大

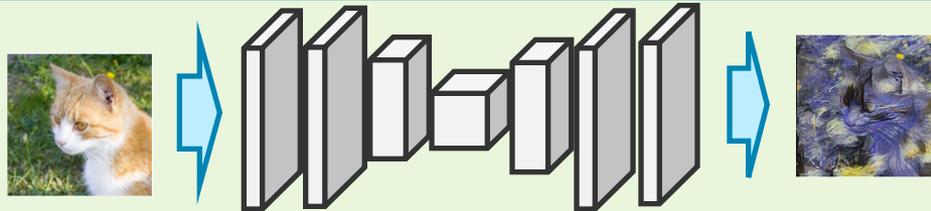




目的

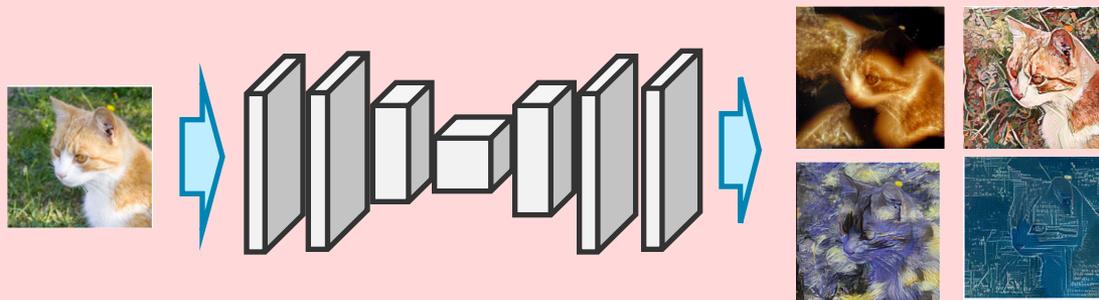
- Johnsonらの手法を拡張し, 1つのモデルで**複数**のスタイルの同時学習を可能に
 - 学習時間の削減
 - 消費メモリの削減

従来手法



1対**1**

提案手法



1対
複数



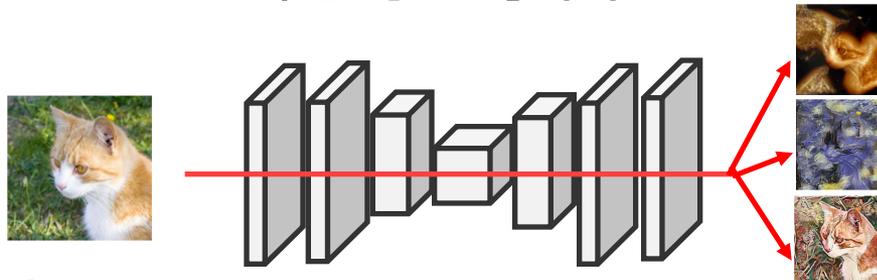
本研究の貢献

- ① 単一のモデルで複数のスタイルを任意の重みで合成
- ② リアルタイムで合成したスタイルを画像に転送
- ③ モバイルアプリとして実装

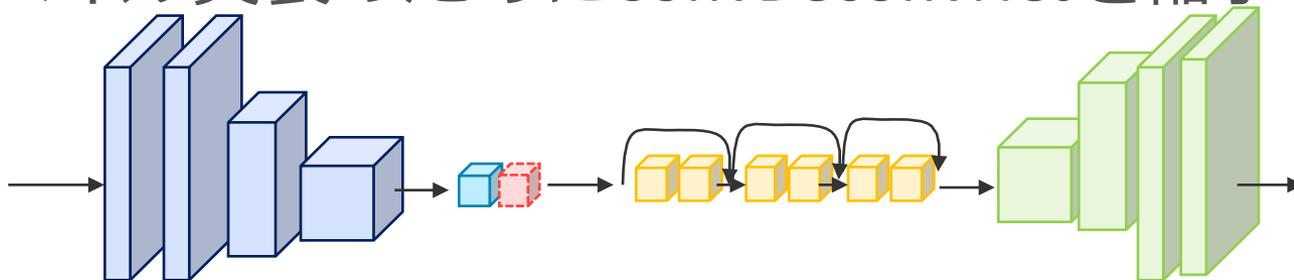


提案手法

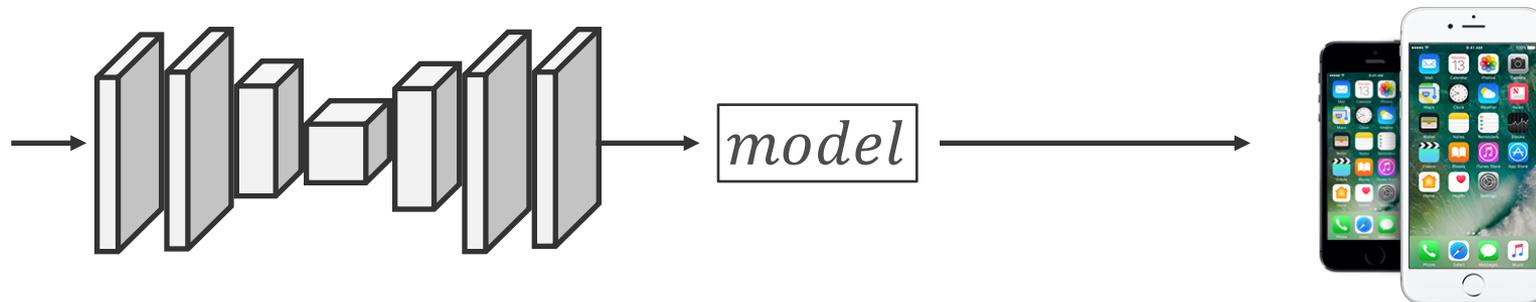
- ① 複数のスタイルを同時に学習できるように拡張

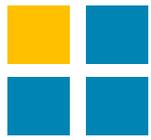


- ② モバイル実装のためにConvDeconvNetを縮小



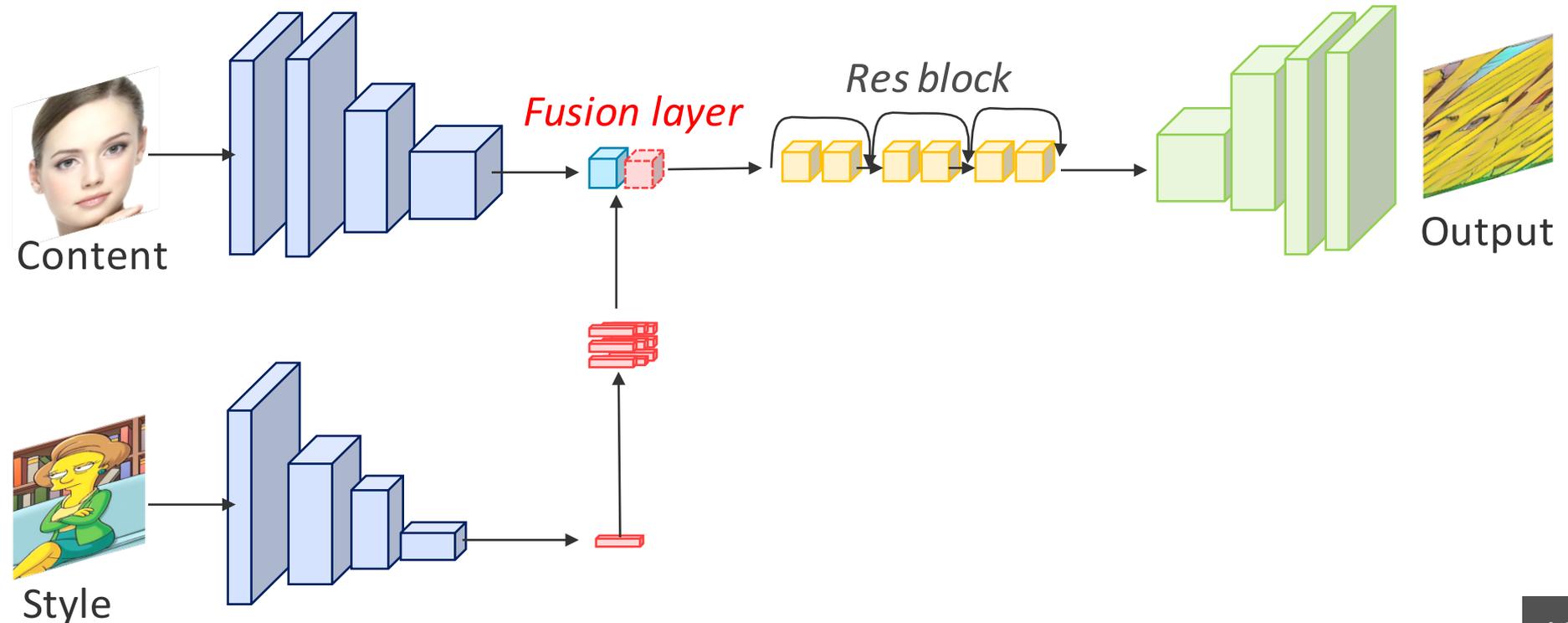
- ③ モバイル上での効率的なCNN実装及び工夫





提案手法①

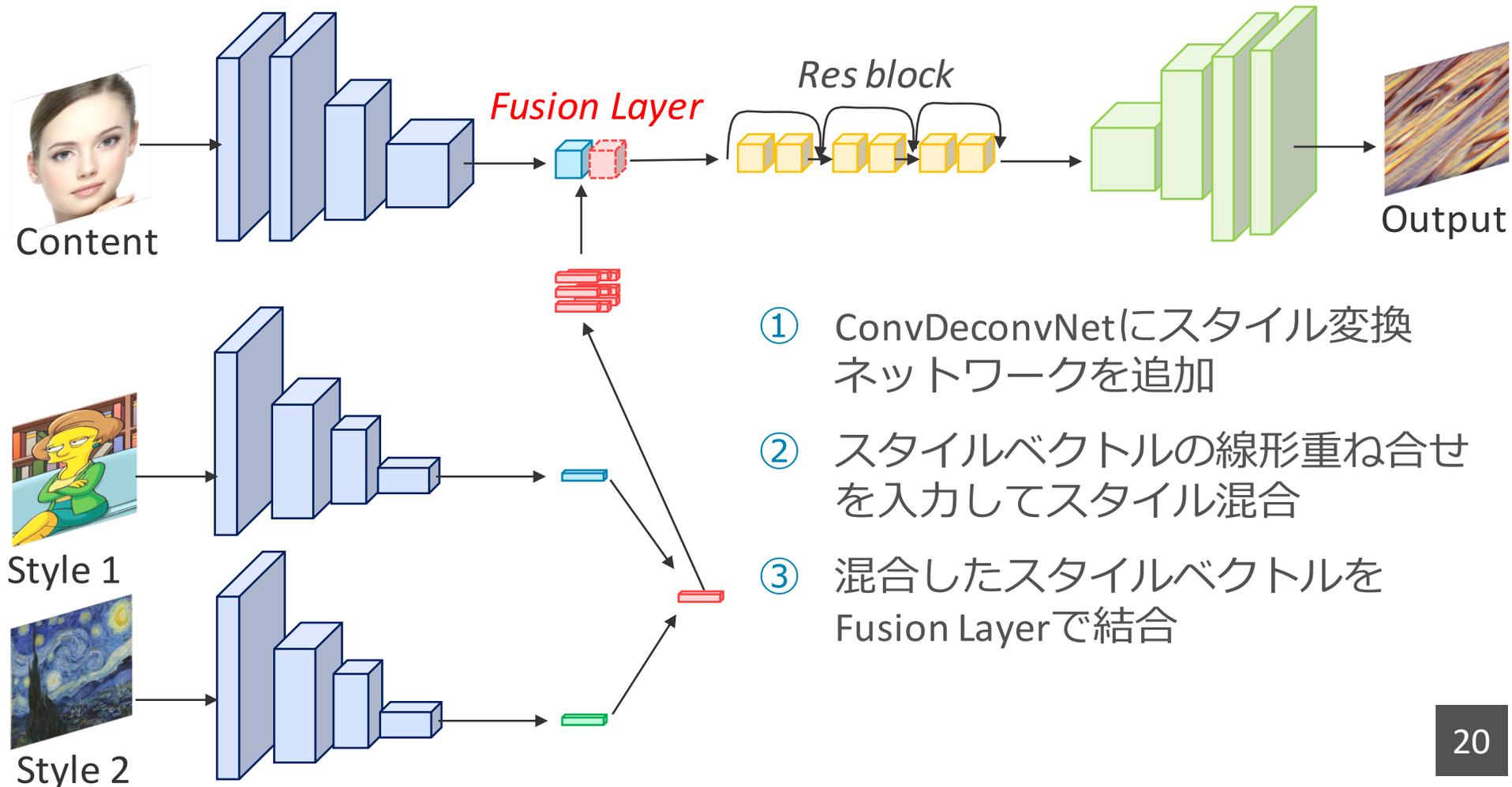
- 複数のスタイルを同時に学習できるように拡張
 - Fusion Layerの追加
 - 複数の任意重みを合成可能

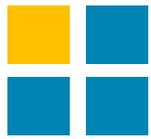




提案手法①

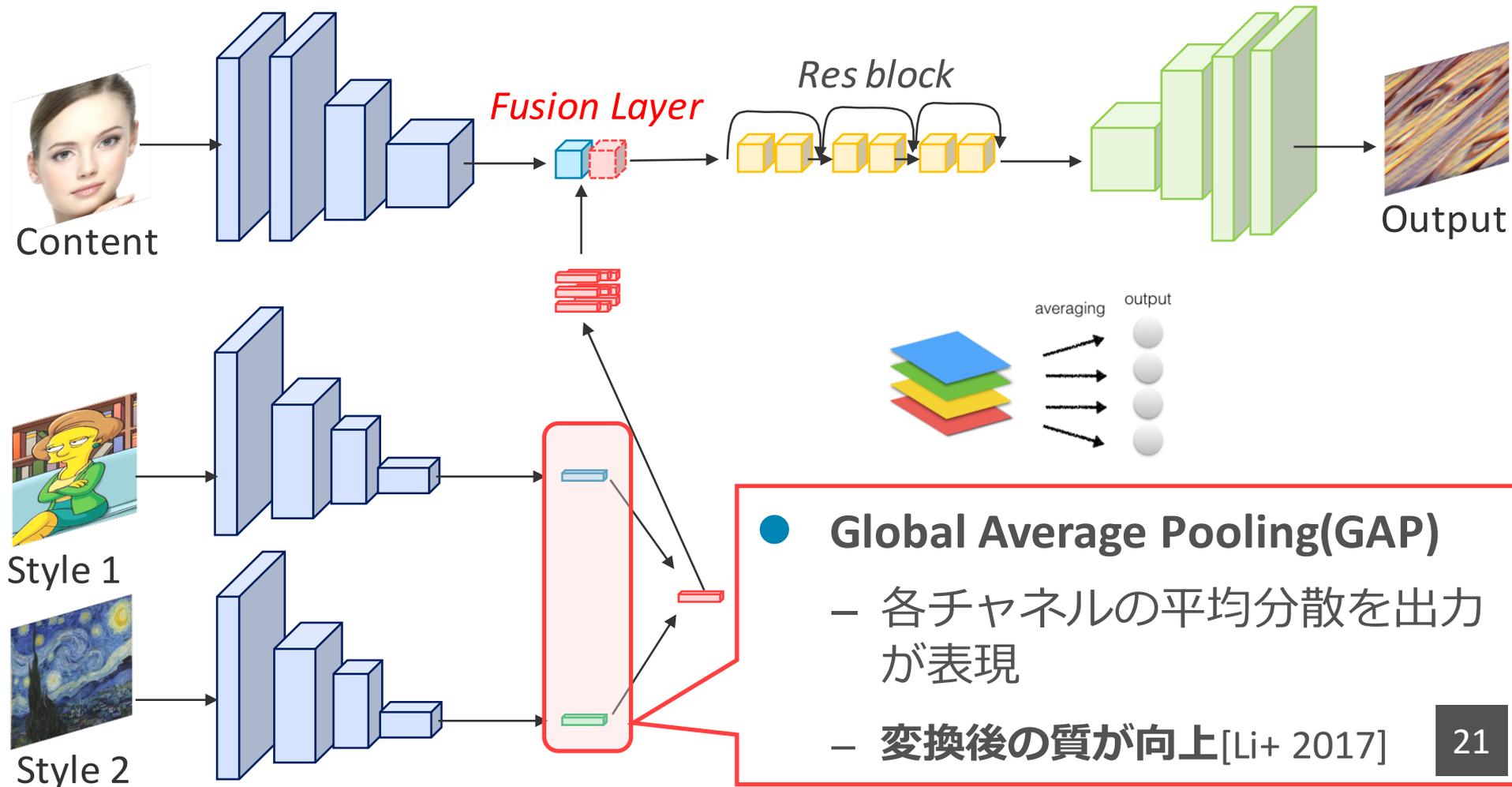
- 複数のスタイルを同時に学習できるように拡張

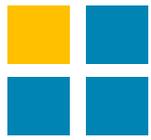




提案手法①

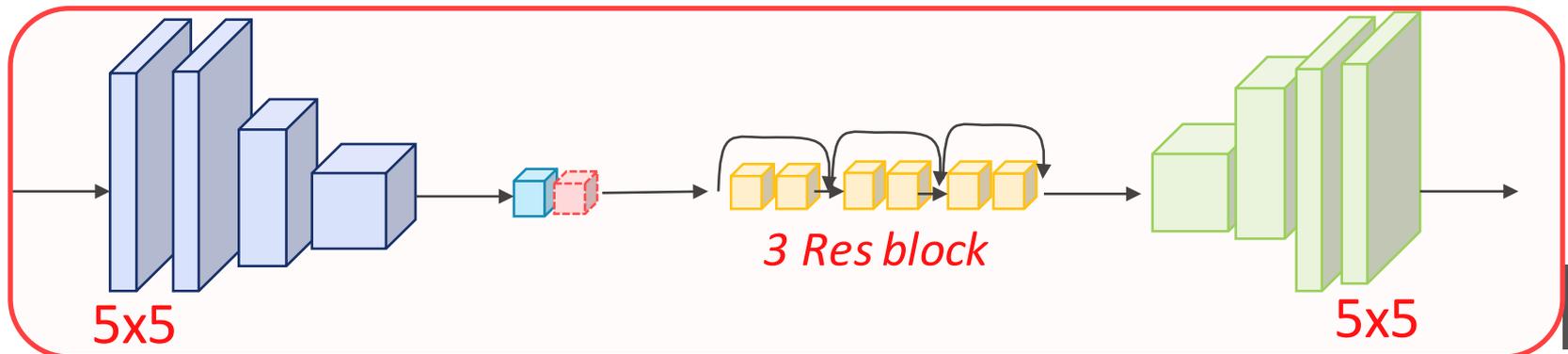
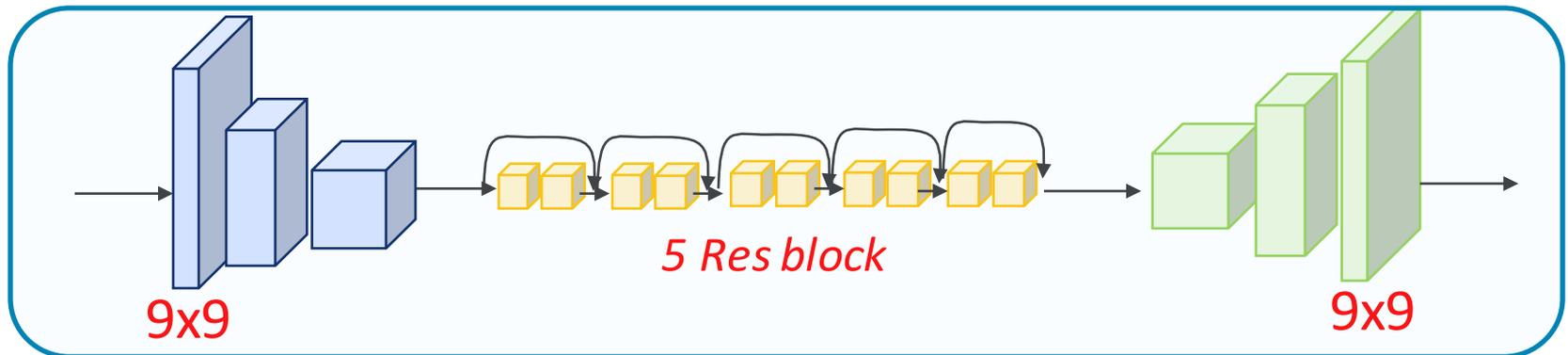
- 複数のスタイルを同時に学習できるように拡張

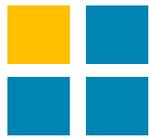




提案手法②

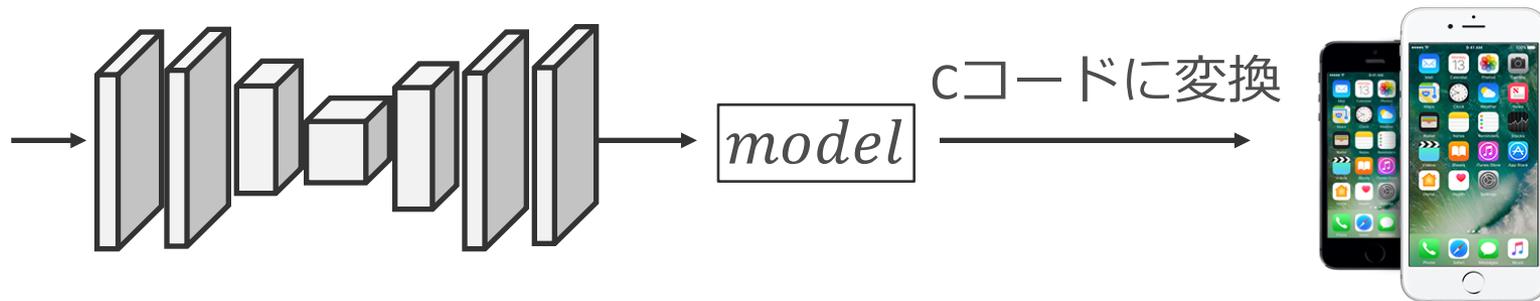
- モバイル実装のためにConvDeconvNetを縮小
 - Down Sampling Layer と Up Sampling Layer を追加
 - 最初と最後のConv Layerのカーネルを $9 \times 9 \rightarrow 5 \times 5$ に変更
 - Residual Elementsを $5 \rightarrow 3$ に変更



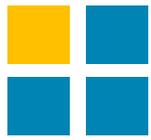


提案手法③

- モバイル上での効率的なCNN実装及び工夫

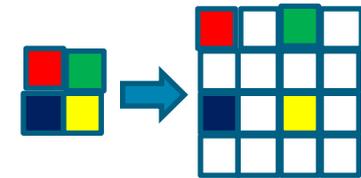


- ① DNNを直接Cコードに変換(コンパイラ的に実行)
- ② Multithread化によるNEON / BLAS の効率的な利用
- ③ CNNに掛かる演算の可能な限りの事前計算の実行
- ④ **Unpooling+Conv -> 1/2 strideを実現**
 - ✓ Deconvolutional layerの代替



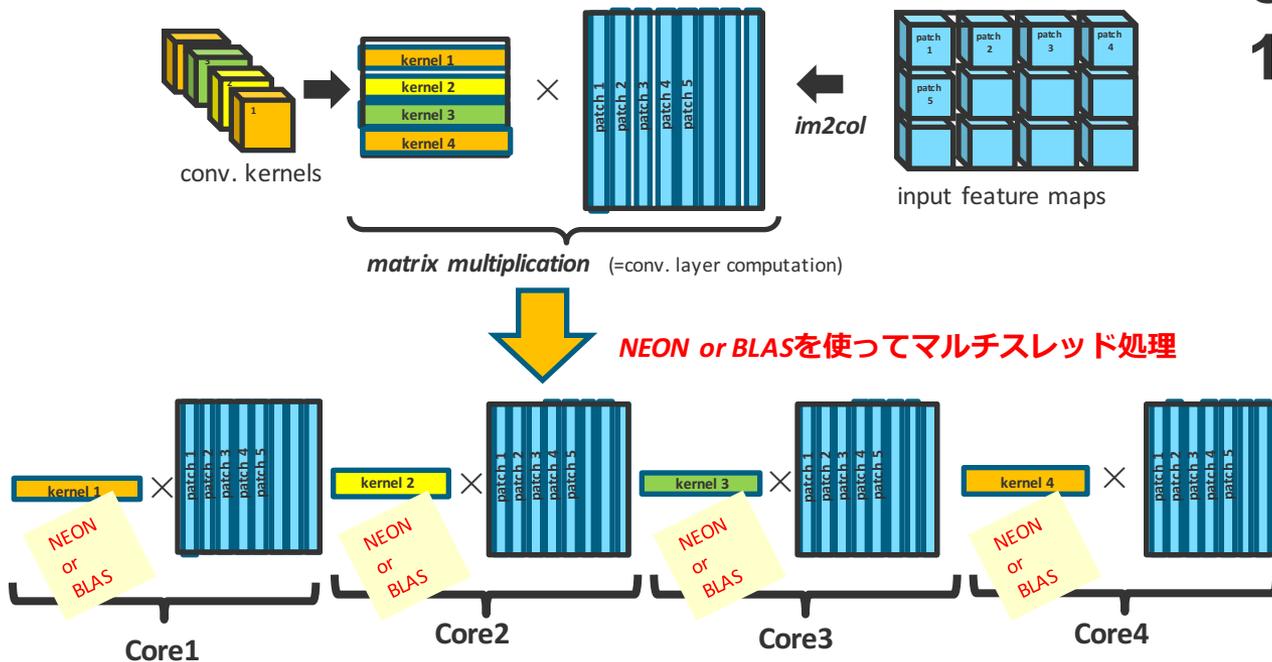
Unpooling + Conv -> 1/2 strideを実現

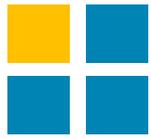
- Convの逆演算を行うDeconvをConvで表現したい...
 - 高速GEMM演算(BLAS)を利用したい



Unpooling =
1/2 stride Conv

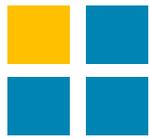
- **Unpoolingで縦横2倍に拡大 -> Conv**



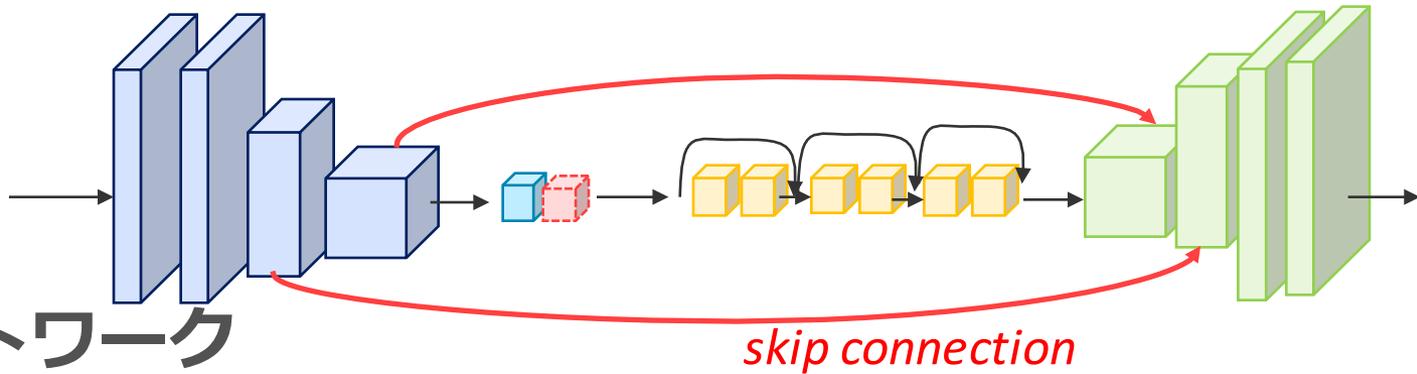


デモ動画





考察



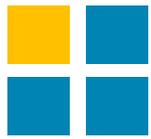
● 変換ネットワーク

- Res block 3個は過剰？
- skip connectionの追加
- artifactの対策

● モバイル実装

- 画像の解像度を上げたい -> Conv処理を高速化必須
- 明示的にGPUを利用？





まとめ

- 目的
 - 「Neural Style Transferのモバイル上への実装」
- 既存手法
 - **1つのモデルで1つのスタイル**
 - **学習に時間を要する**
 - **消費メモリの増大**
- 提案手法
 - **1つのモデルで複数のスタイルを変換可能**
 - **モバイル上でのリアルタイム画風変換**
- 今後の課題
 - **線画のモバイル上でのリアルタイム着色**

