

академия  
больших  
данных

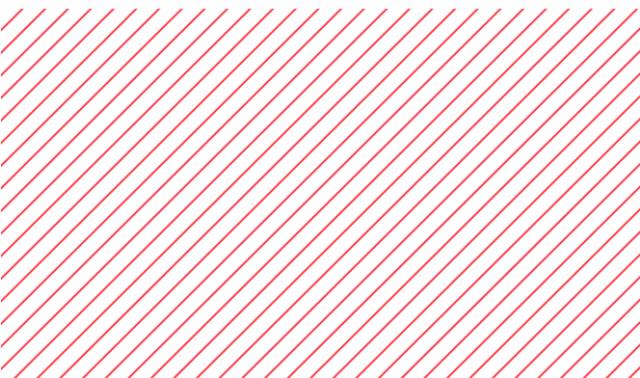


mail.ru  
group

# Прикладные задачи обработки изображений на мобильных устройствах

Андрей Савченко

Профессор НИУ ВШЭ-Нижний Новгород





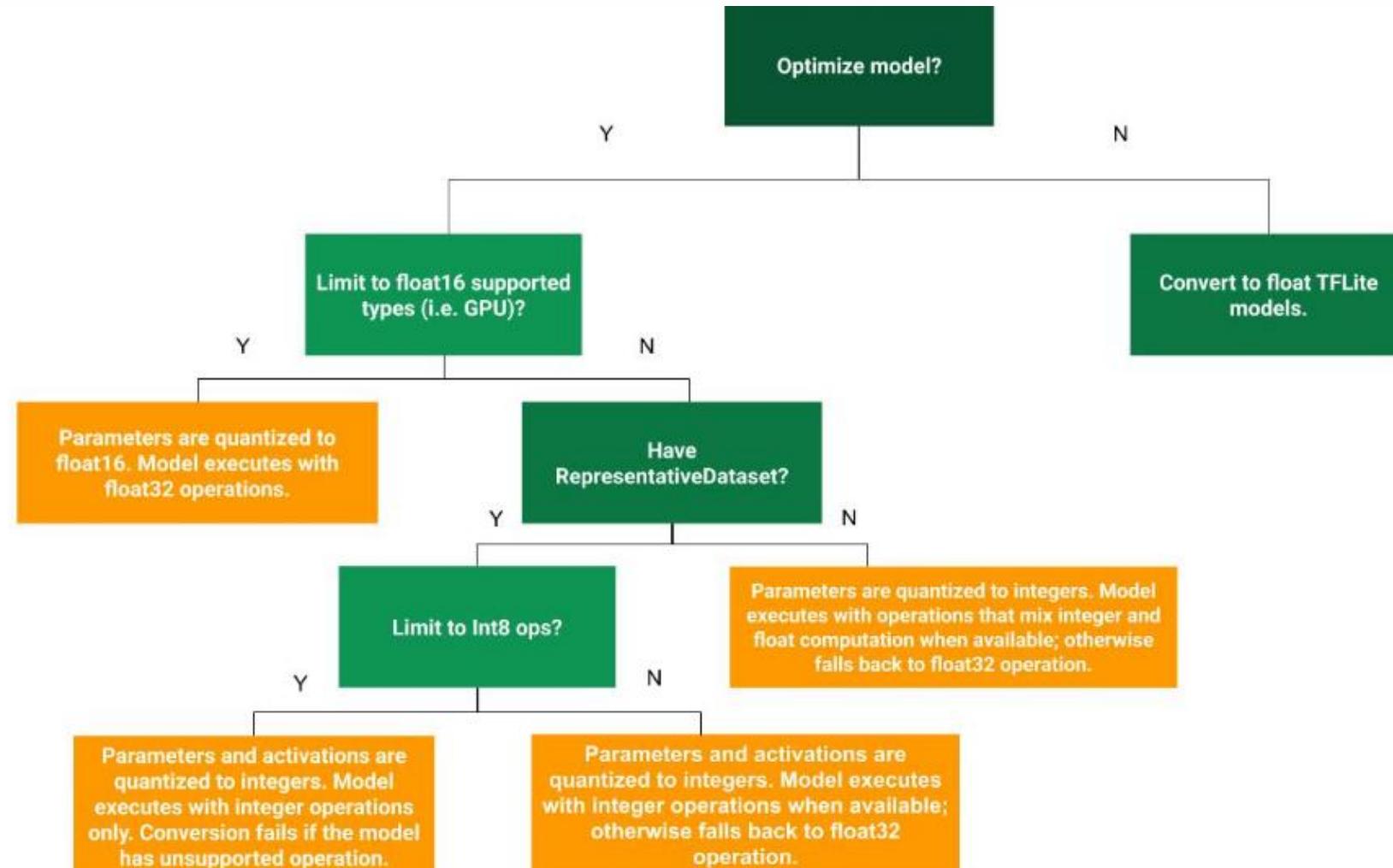
# Опрос

---

<https://forms.gle/dBdQw9HdJTMyXnAn6>

# Квантование нейронных сетей

# Варианты квантования



# Float32->Float16

- Самый простой вариант сжатия модели в 2 раза
- Практически нет потерь в точности
- Скорость inference может увеличиться только для GPU

## Пример Tensorflow

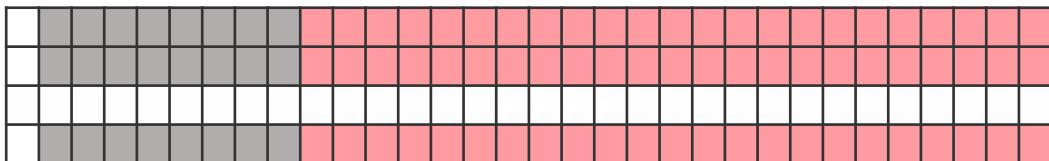
```
converter =  
tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)  
converter.optimizations = [tf.lite.Optimize.DEFAULT]  
converter.target_spec.supported_types = [tf.float16]  
tflite_quant_model = converter.convert()
```

## Пример PyTorch

```
quantized_model =  
torch.quantization.quantize_dynamic(model,  
dtype=torch.float16)
```

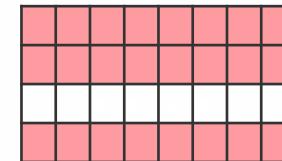
# Float32->int8

N × float32



$$\text{float\_val} = (\text{uint8\_val} - \text{zero\_point}) \times \text{scale}$$

N × uint8

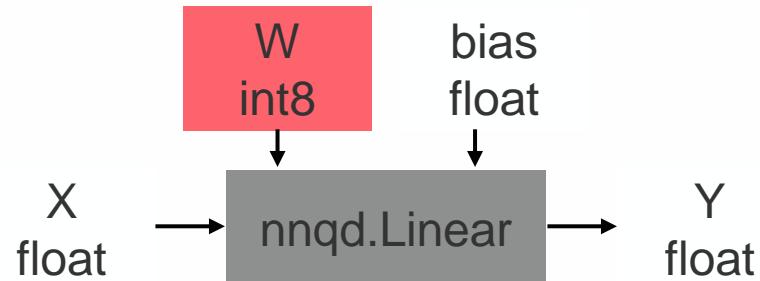


float32

scale

int32

zero\_point



- Сжатие модели в 4 раза
- inference в 2-4 раза быстрее
- Точность может существенно снизиться

## Пример Tensorflow

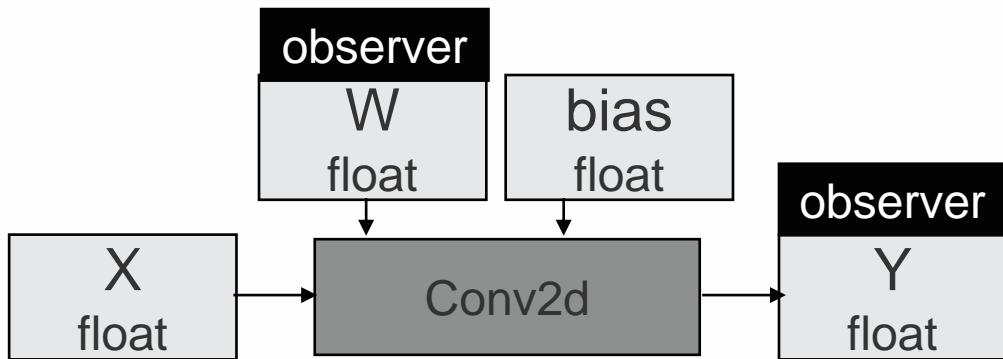
```
converter =  
tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)  
converter.optimizations = [tf.lite.Optimize.DEFAULT]  
tflite_quant_model = converter.convert()
```

## Пример PyTorch

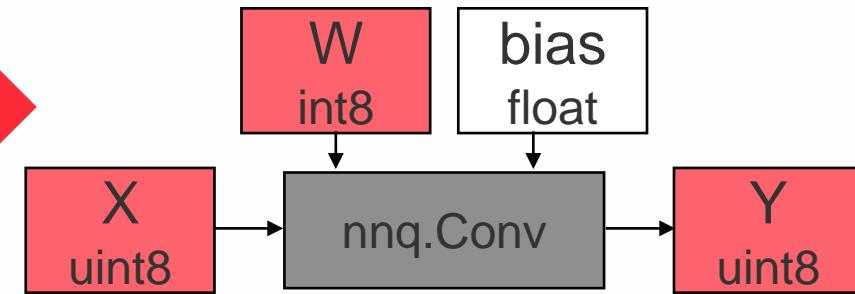
```
quantized_model =  
torch.quantization.quantize_dynamic(model,  
dtype=torch.qint8)
```

# Что делать с точностью (1)? Post Training

Калибровка на наборе данных



Квантование



# Post Training (2)

## Пример Tensorflow

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
def data_gen():
    for _ in range(num_calibration_steps):
        # Get sample input data as a numpy array
        yield [input]
converter.representative_dataset = data_gen
converter.target_spec.supported_ops =
[tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
tflite_quant_model = converter.convert()
```

## Пример PyTorch

```
model = quantization.fuse_modules(model,
        [["conv1", "bn1", "relu1"]])
model.qconfig =
quantization.get_default_qconfig('fbgemm')
# Use 'fbgemm' for server inference and # 'qnnpack'
for mobile inference.

qmodel = quantization.prepare(model,inplace=False)

qmodel.eval()
for batch, target in data_loader:
    model(batch)
    quantization.convert(qmodel)
```

# Что делать с точностью (2)? Quantization aware training

## Дообучение квантованной модели

### Пример Tensorflow

```
import tensorflow_model_optimization as tfmo
q_aware_model =
tfmo.quantization.keras.quantize_model(model)
q_aware_model.compile(...)
q_aware_model.fit(...)
converter =
tf.lite.TFLiteConverter.from_keras_model(q_aware_model)
converter.experimental_new_converter = True
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_q_aware_integer_quant_model = converter.convert()
```

### Пример PyTorch

```
model = quantization.fuse_modules(model,
[["conv1", "bn1", "relu1"]])
# specify which part to quantize and how
model.qconfig =
quantization.get_default_qat_qconfig('fbgemm')
qmodel = quantization.prepare_qat(model,
inplace=False)
# fine tune model
train(qmodel, train_data)
```

<https://research.fb.com/wp-content/uploads/2019/12/2.-Quantization.pptx>

[https://www.tensorflow.org/lite/performance/post\\_training\\_quantization](https://www.tensorflow.org/lite/performance/post_training_quantization)

<https://gist.github.com/NobuoTsukamoto/b42128104531a7612e5c85e246cb2dac>

# Результаты (Tensorflow). Flowers

---

**MobileNetV2 (fp32 accuracy 98.30%)**

Technique	accuracy change
TfLite (Fp32)	+0.07 98.37
Fp16	+0.07 98.37
int8	-12.74 85.56
Post training: Weight quantization (int8)	-0.48 97.82

<https://gist.github.com/NobuoTsukamoto/b42128104531a7612e5c85e246cb2dac>

<https://colab.research.google.com/gist/sayakpaul/468f70d075e5608bfebeb9ba639265b6/qat-bad-accuracy.ipynb#scrollTo=5HXdowikNYS7>

# Результаты (PyTorch). ImageNet

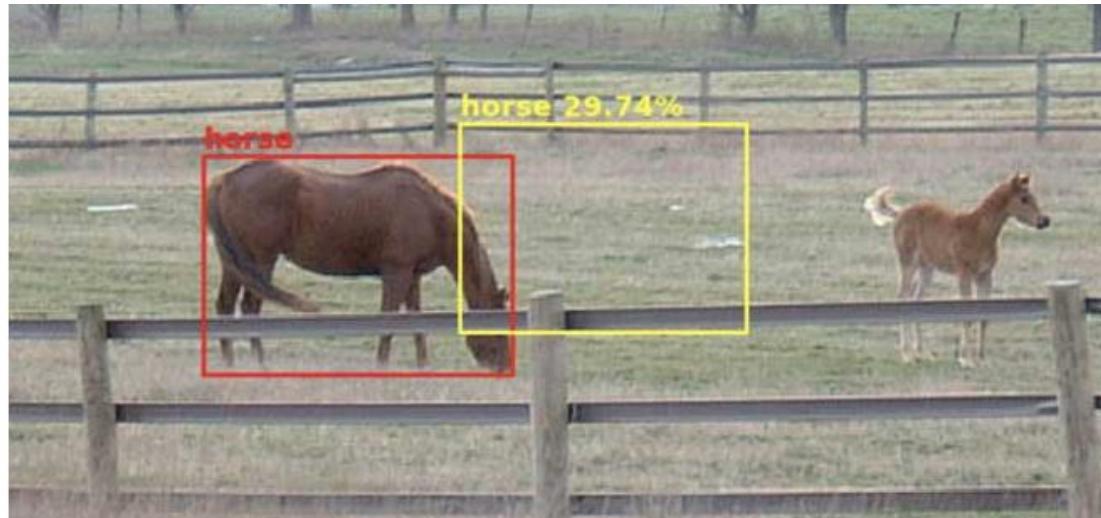
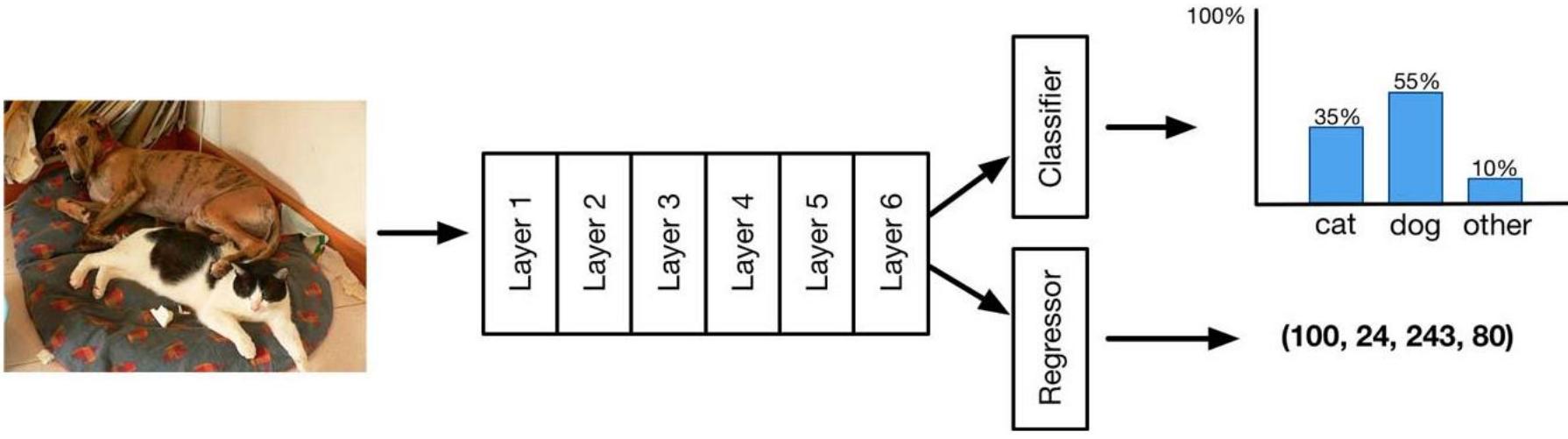
	<b>fp32 accuracy</b>	<b>int8 accuracy change</b>	<b>Technique</b>	<b>CPU inference speed up</b>
<b>ResNet50</b>	76.1	-0.2 75.9	Post Training	2x 214ms → 102ms
<b>MobileNetV2</b>	71.9	-0.3 71.6	Quantization-Aware Training	4x 75ms → 18ms

**MobileNetV2** (fp32 accuracy 71.9%)

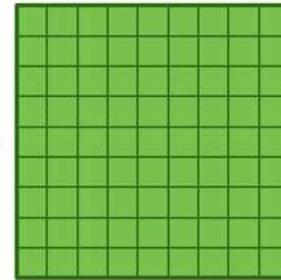
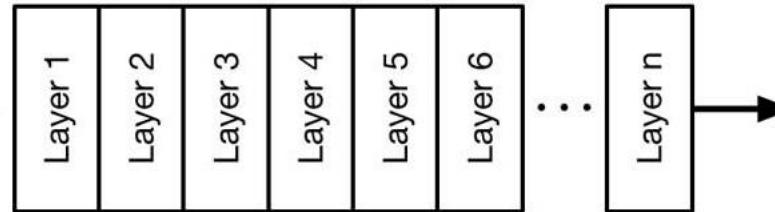
<b>Technique</b>	<b>int8 accuracy change</b>
Post Training: Per Tensor quantization	-6.3 65.6
Post-Training: Per-channel quantization	-4.8 67.1
Quantization-Aware Training	-0.3 71.6

# Детектирование объектов на мобильных устройствах

# One-shot detection. Наивный подход



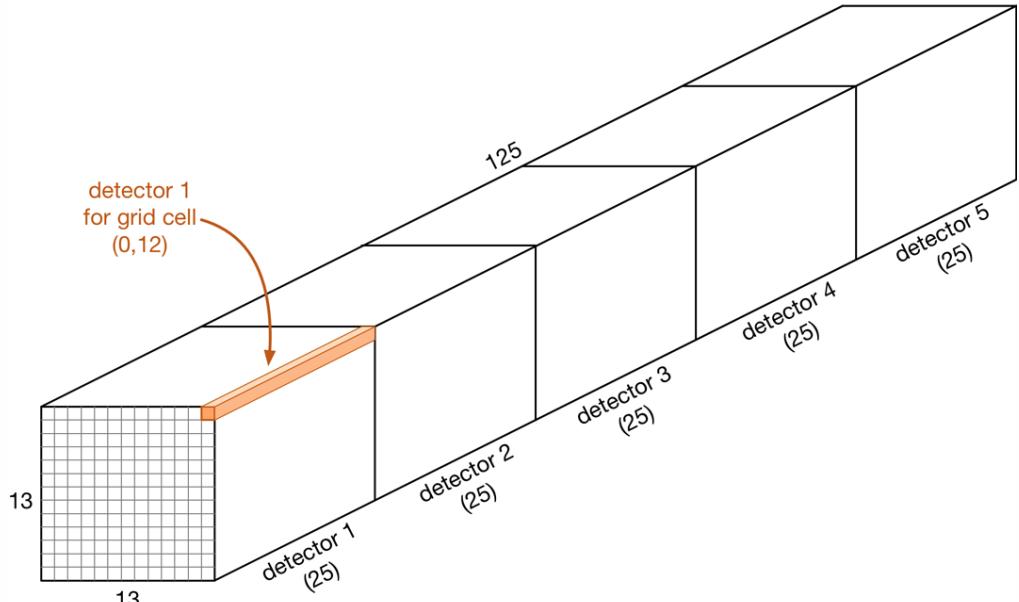
# Fixed grid of detectors



416x416x3

convolutional layers

13x13x125

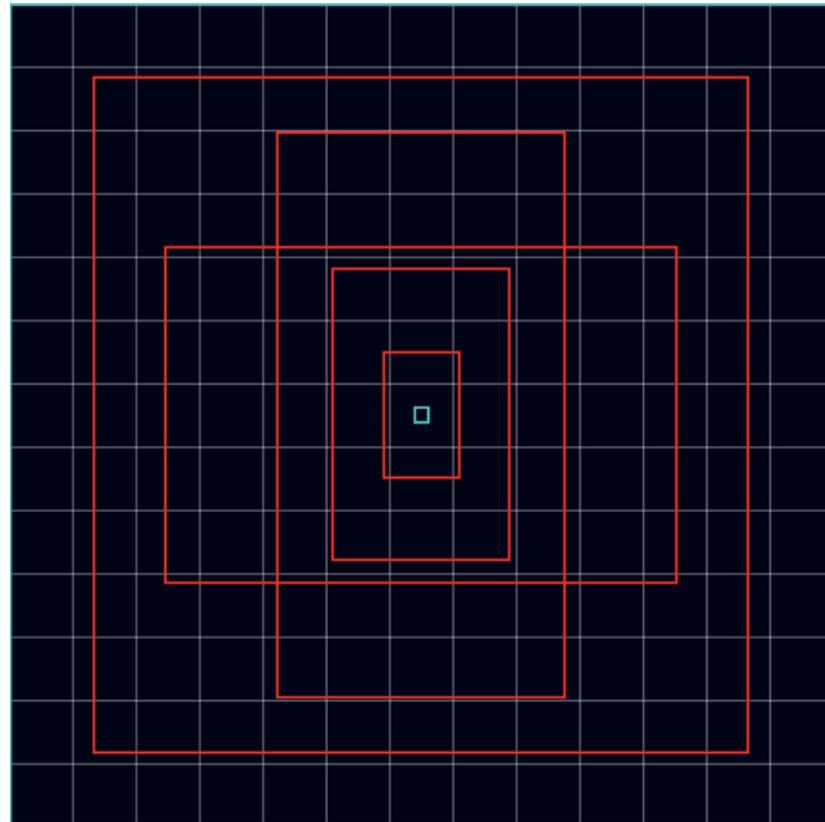


25 выходов для VOC (20 классов):

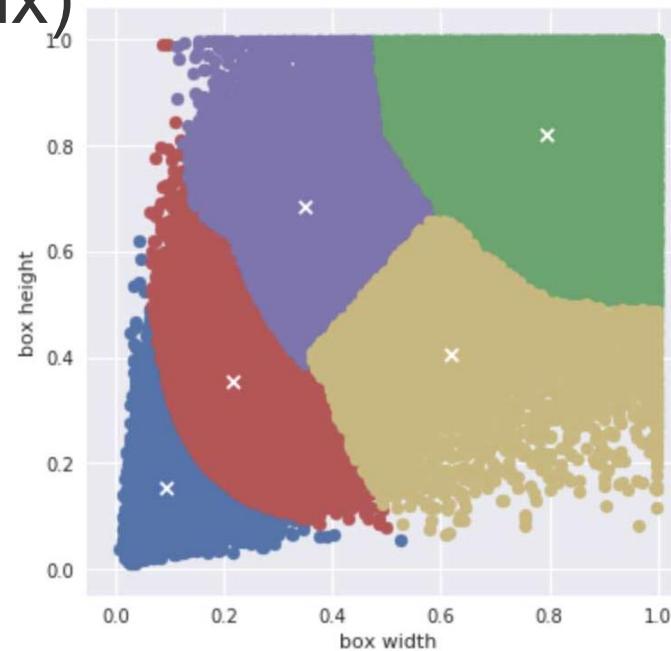
- 20 апостериорных вероятностей
- 4 bounding box
- 1 “object-ness” score

# Anchors (Priors)

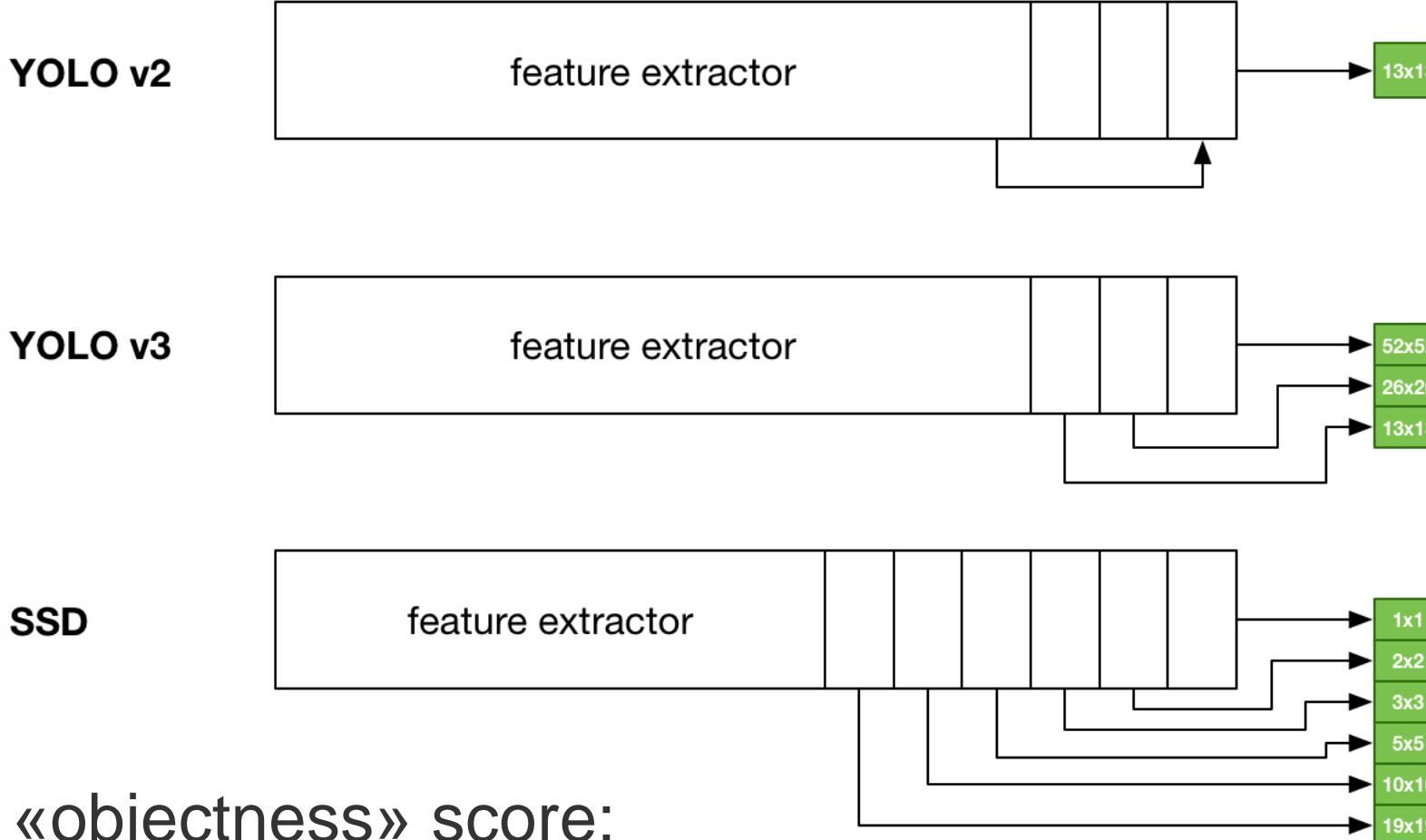
Несколько (5) детекторов  
для объектов разной формы!



- SSD: размеры выбираются заранее
- YOLO: k-means clustering  
(обучается на заданном наборе данных)



# One-shot detection. YOLO vs SSD



В SSD нет «objectness» score:  
добавляется класс фона

# Как выбрать детектор?

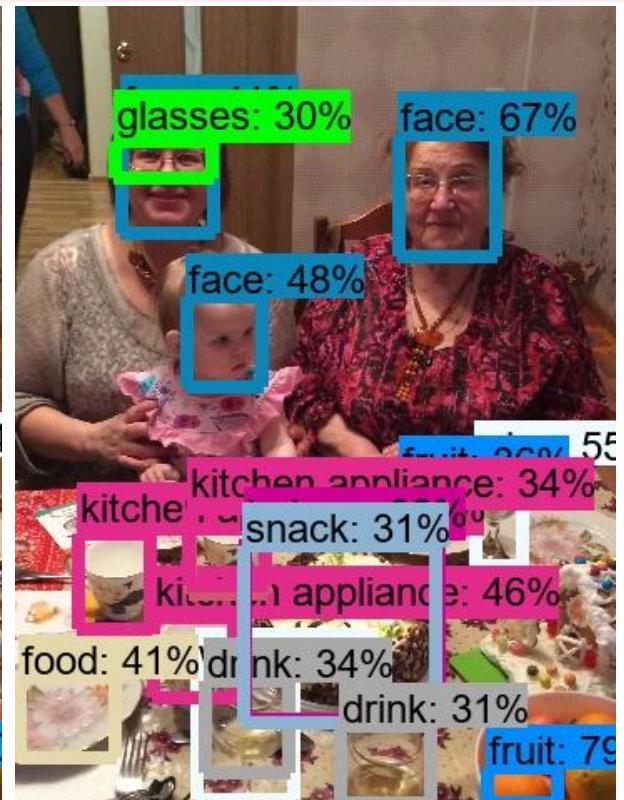
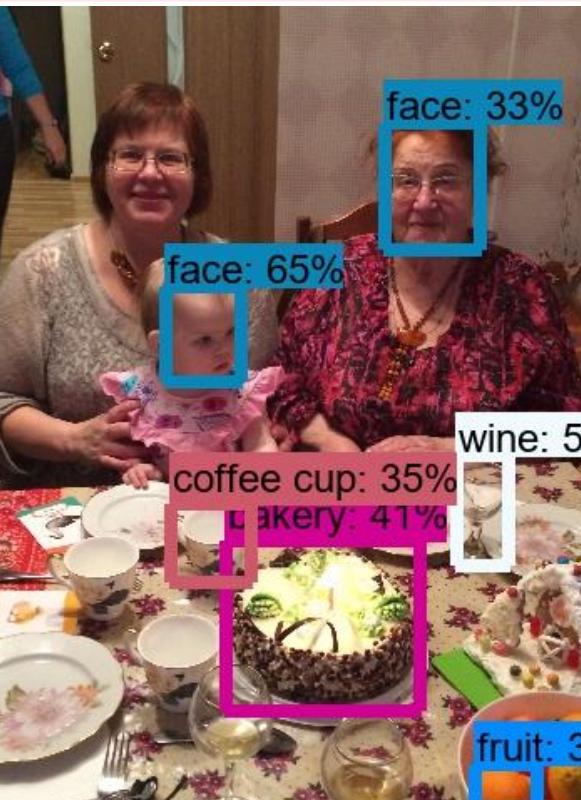
Model name	Speed (ms)	COCO mAP[^1]	Outputs
<a href="#">ssd_mobilenet_v1_coco</a>	30	21	Boxes
<a href="#">ssd_mobilenet_v2_coco</a>	31	22	Boxes
<a href="#">ssdlite_mobilenet_v2_coco</a>	27	22	Boxes
<a href="#">ssd_inception_v2_coco</a>	42	24	Boxes
<a href="#">faster_rcnn_inception_v2_coco</a>	58	28	Boxes
<a href="#">faster_rcnn_resnet50_coco</a>	89	30	Boxes
<a href="#">faster_rcnn_resnet101_coco</a>	106	32	Boxes
<a href="#">faster_rcnn_inception_resnet_v2_atrous_coco</a>	620	37	Boxes
<a href="#">CenterNet HourGlass104 512x512</a>	70	41.9	Boxes
<a href="#">EfficientDet D0 512x512</a>	39	33.6	Boxes
<a href="#">EfficientDet D1 640x640</a>	54	38.4	Boxes

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/tf1\\_detection\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md)

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/tf2\\_detection\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md)

# Результаты

	<b>SSDLite+ MobileNet v2</b>	<b>Faster RCNN+ Inception v2</b>	<b>Faster RCNN+ InceptionResNet</b>
Время	30-100 мс	400-500 мс	6000-7000 мс (6-7 с)
Размер	22 Мб	55 Мб	250 Мб (после квантования: 68 Мб)





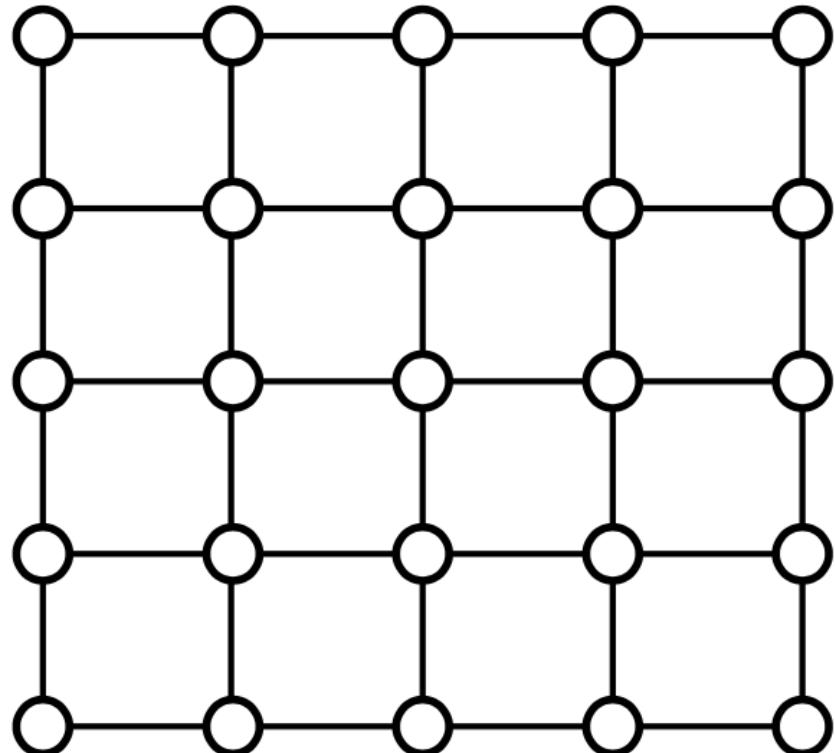
# Подготовка детектора

---

1. Загрузить репозиторий <https://github.com/tensorflow/models/> и запустить setup.py
2. Загрузить модель (pb)
3. Запустить скрипт export\_tflite\_ssd\_graph.py
4. Сохранить в tflite (обязательно с опцией allow\_custom\_ops)

# Сегментация изображений

# Вероятностная графическая модель Markov random field (MRF)



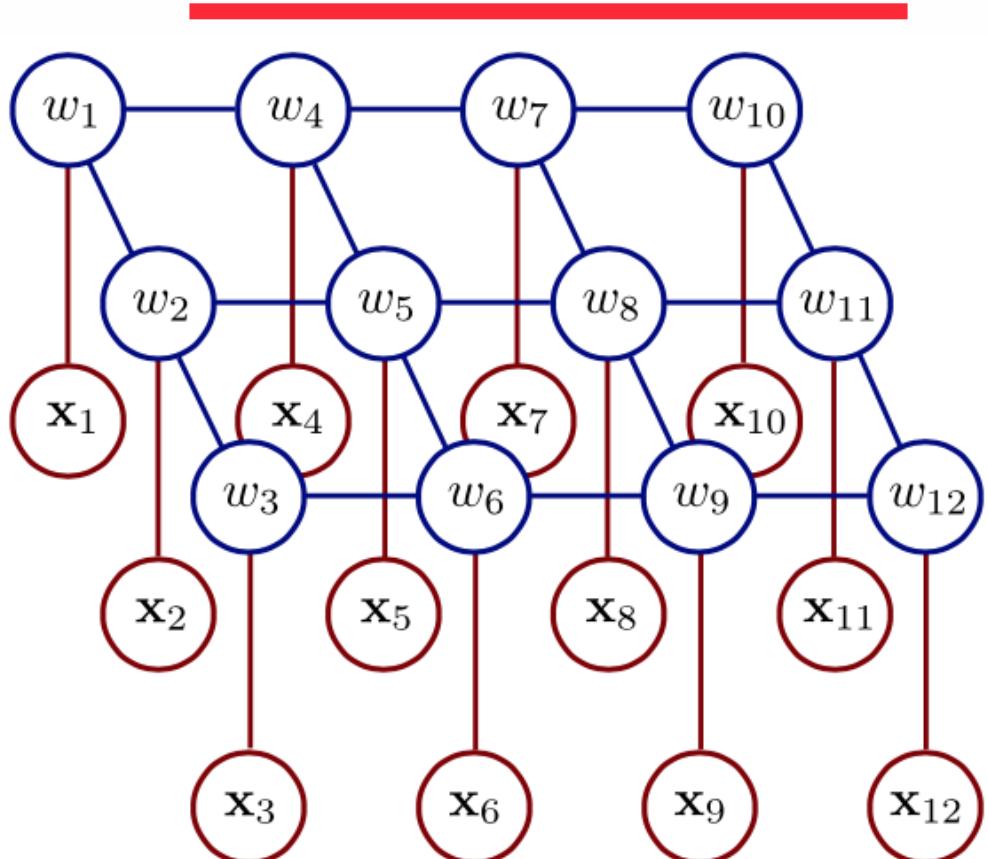
Свойство Маркова (условная вероятность случайной переменной, связанной с  $n$ -й точки, зависит только от случайных величин в заданной окрестности этой точки):

$$Pr(w_n | w_{\mathcal{S} \setminus n}) = Pr(w_n | w_{\mathcal{N}_n})$$

Совместное распределение всех случайных величин  $w$ :

$$Pr(\mathbf{w}) = \frac{1}{Z} \prod_{j=1}^J \phi_j[\mathbf{w}_{\mathcal{C}_j}] = \frac{1}{Z} \exp \left[ - \sum_{j=1}^J \psi_j[\mathbf{w}_{\mathcal{C}_j}] \right]$$

# Conditional random field (CRF)



- $x$  – значения пикселей
- $w$  – скрытые переменные, например, значение пикселей при очистке от шума

$$Pr(w_{1\dots N}|x_{1\dots N}) = \frac{\prod_{n=1}^N Pr(x_n|w_n)Pr(w_{1\dots N})}{Pr(x_{1\dots N})}$$

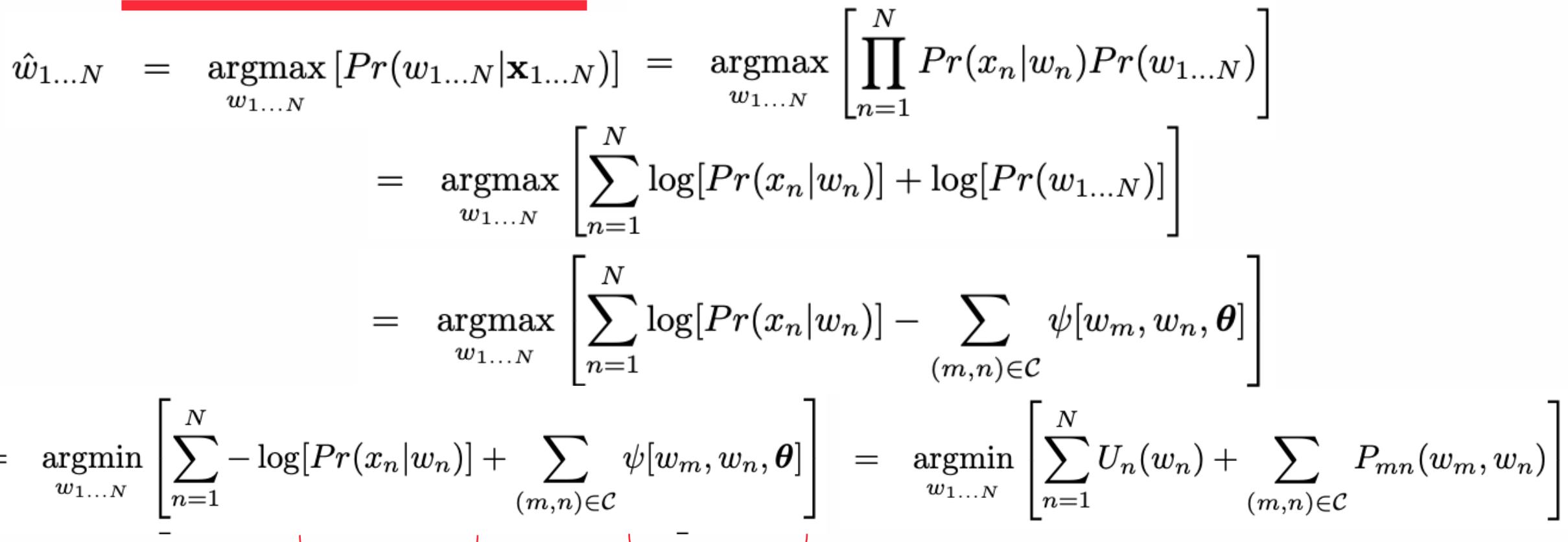
Априорное распределение – MRF:

$$Pr(w_{1\dots N}) = \frac{1}{Z} \exp \left[ - \sum_{(m,n) \in C} \psi[w_m, w_n, \theta] \right]$$

$$\psi[w_m = j, w_n = k, \theta] = \theta_{jk},$$

# MAP (Maximal A-posterior probability) inference

$$\begin{aligned}\hat{w}_{1\dots N} &= \underset{w_{1\dots N}}{\operatorname{argmax}} [Pr(w_{1\dots N} | \mathbf{x}_{1\dots N})] = \underset{w_{1\dots N}}{\operatorname{argmax}} \left[ \prod_{n=1}^N Pr(x_n | w_n) Pr(w_{1\dots N}) \right] \\ &= \underset{w_{1\dots N}}{\operatorname{argmax}} \left[ \sum_{n=1}^N \log[Pr(x_n | w_n)] + \log[Pr(w_{1\dots N})] \right] \\ &= \underset{w_{1\dots N}}{\operatorname{argmax}} \left[ \sum_{n=1}^N \log[Pr(x_n | w_n)] - \sum_{(m,n) \in \mathcal{C}} \psi[w_m, w_n, \theta] \right] \\ &= \underset{w_{1\dots N}}{\operatorname{argmin}} \left[ \sum_{n=1}^N -\log[Pr(x_n | w_n)] + \sum_{(m,n) \in \mathcal{C}} \psi[w_m, w_n, \theta] \right] = \underset{w_{1\dots N}}{\operatorname{argmin}} \left[ \sum_{n=1}^N U_n(w_n) + \sum_{(m,n) \in \mathcal{C}} P_{mn}(w_m, w_n) \right]\end{aligned}$$



Классификатор  
пикселя      Априорное предположение  
о гладкости (штраф за  
сходство/различие  
соседних величин)

# Пример. Бинарные изображения

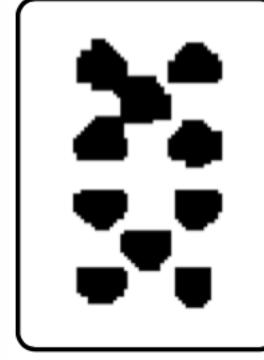
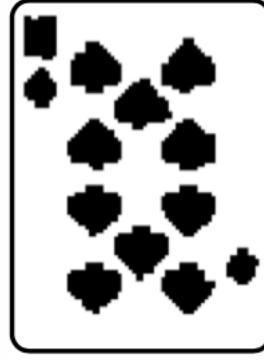
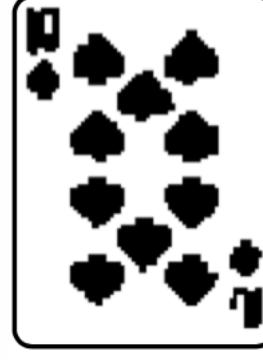
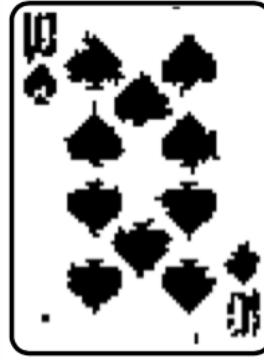
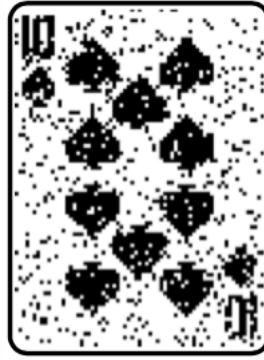
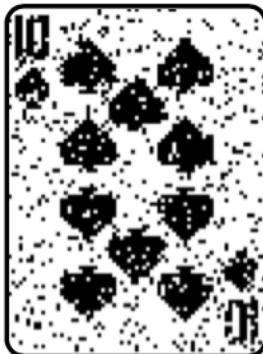
Условные значения яркости  
(распределение Бернулли с параметром  $\rho$ )

$$Pr(x_n|w_n = 0) = \text{Bern}_{x_n}[\rho]$$

$$Pr(x_n|w_n = 1) = \text{Bern}_{x_n}[1 - \rho]$$

Штрафы в MRF – симметричная матрица с нулевой диагональю и параметром  $\theta$ , отвечающим за штраф при изменении яркости с 0 на 1 и наоборот

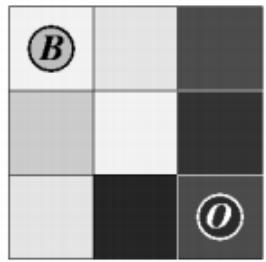
Исходное  
изображение



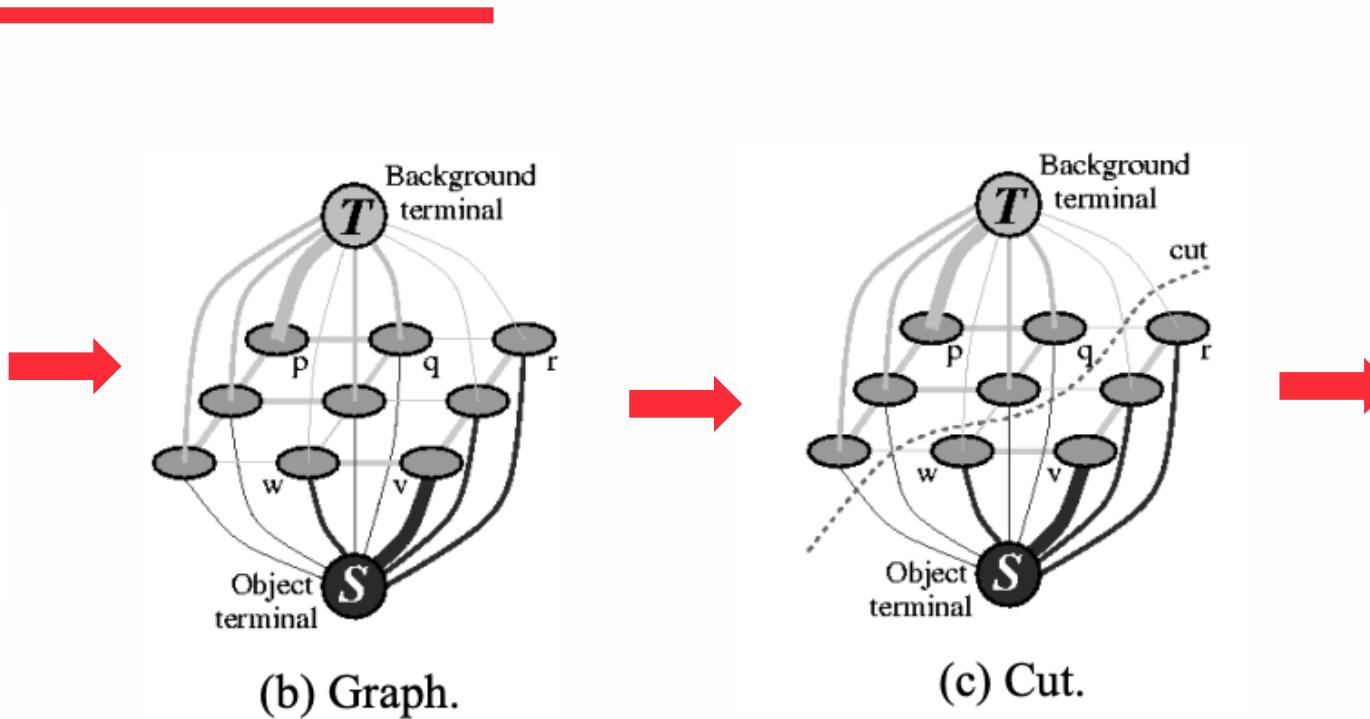
Результаты MAP inference в CRF

Увеличение  $\theta$

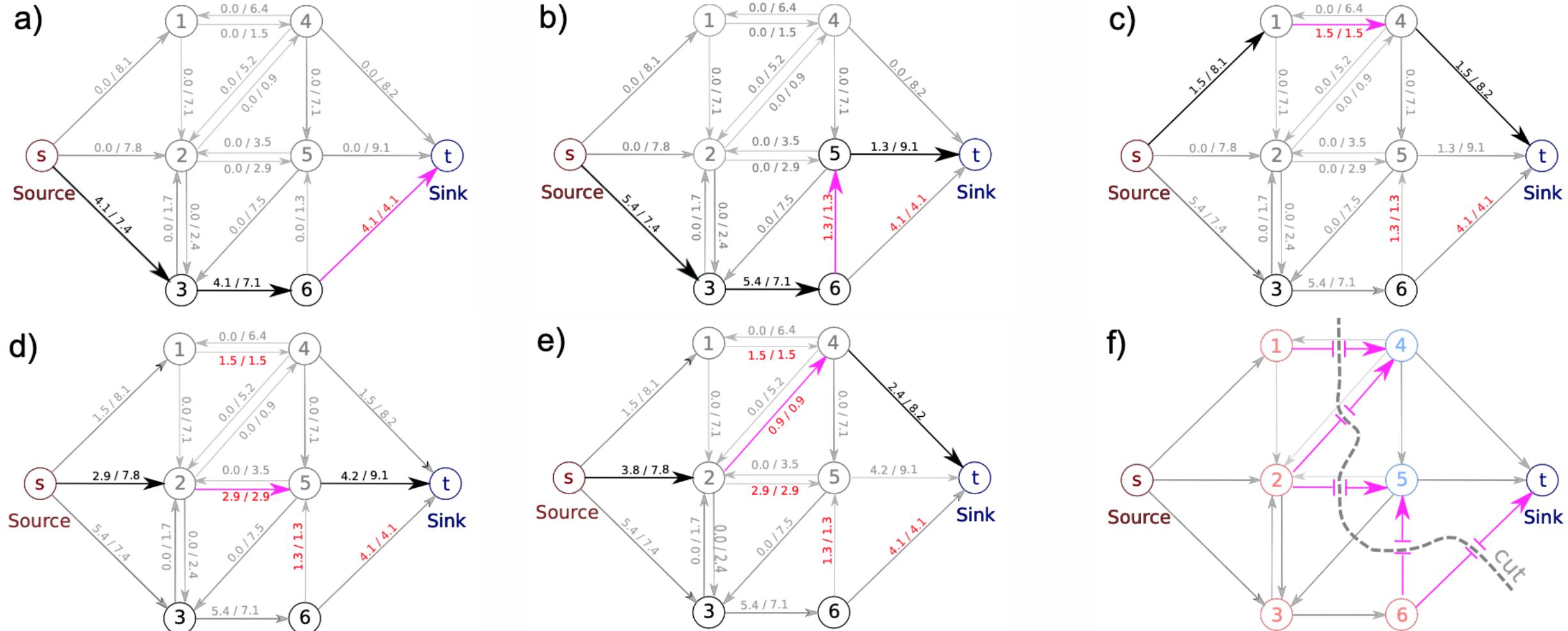
# Поиск МАР-решения. Graphcut



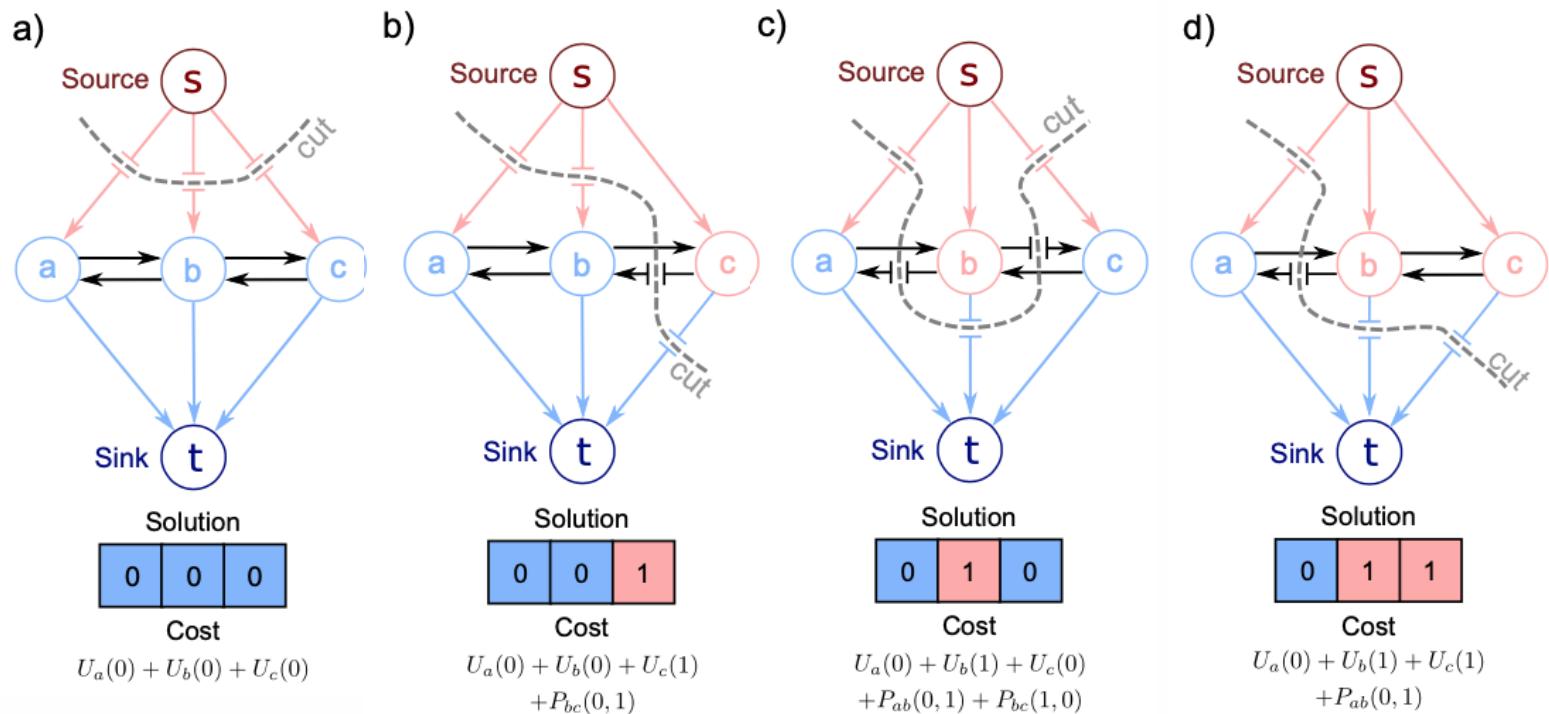
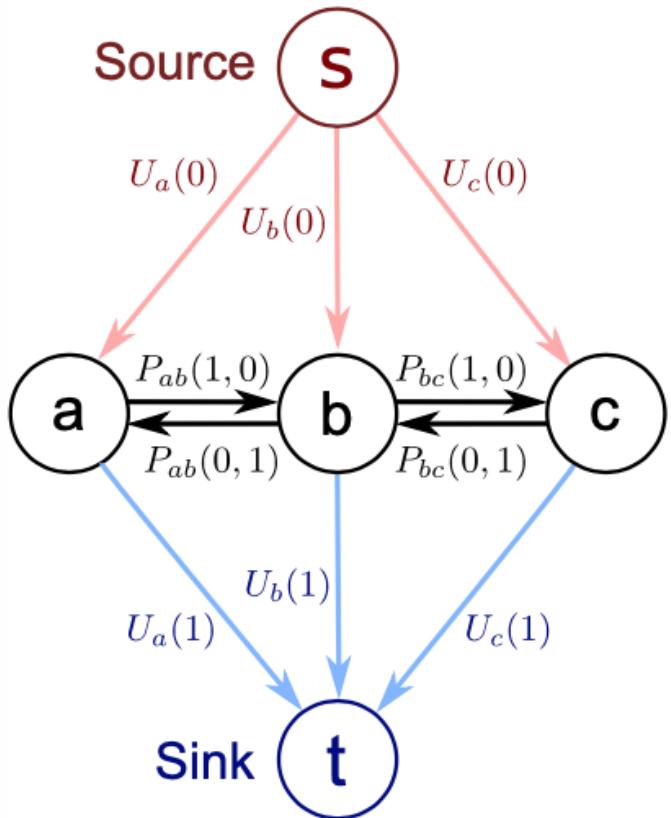
(a) Image with seeds.



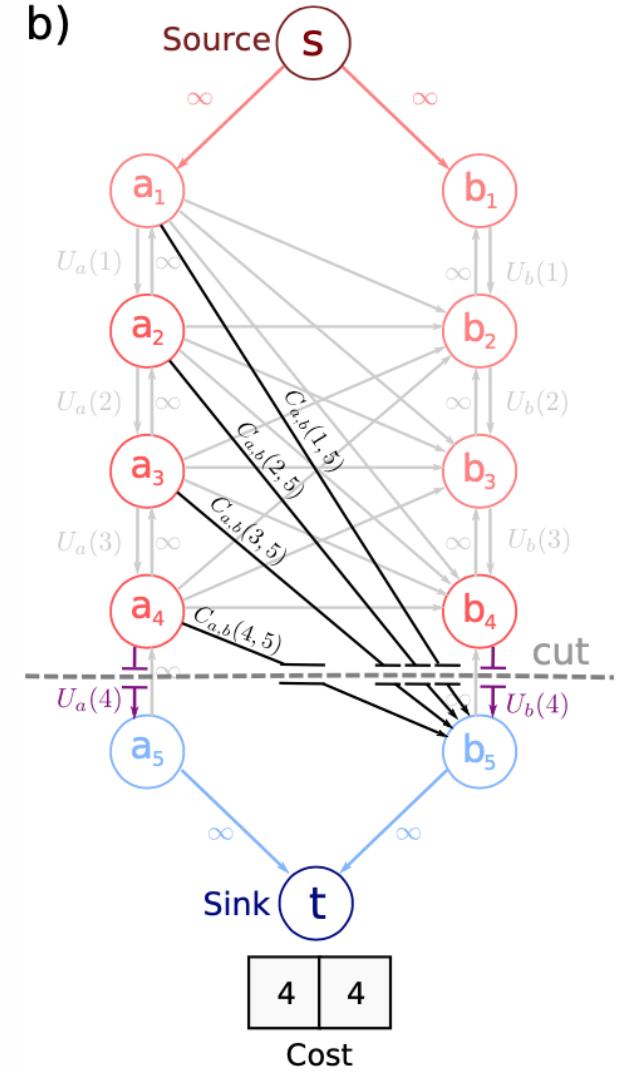
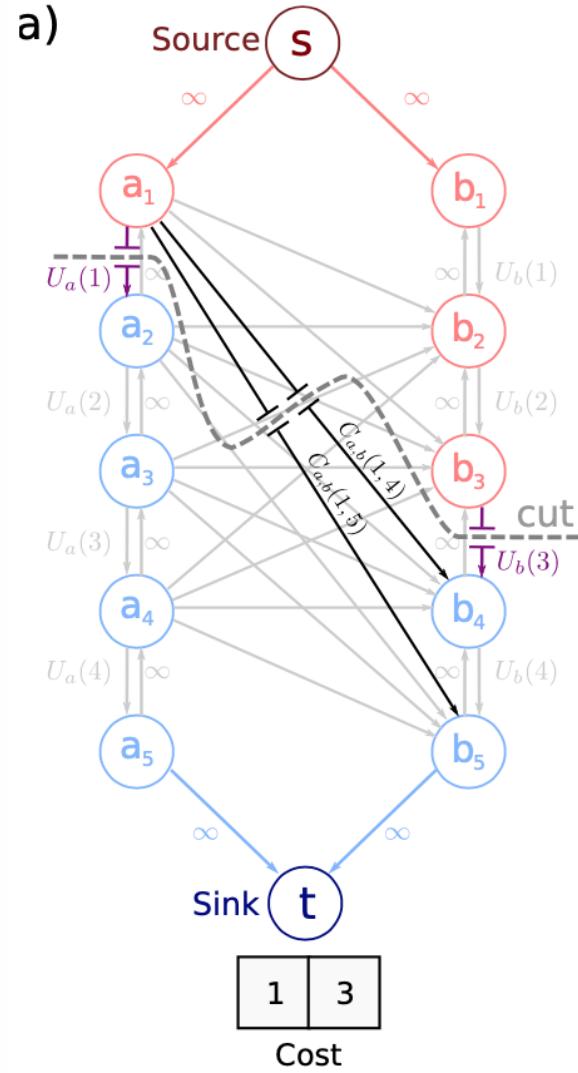
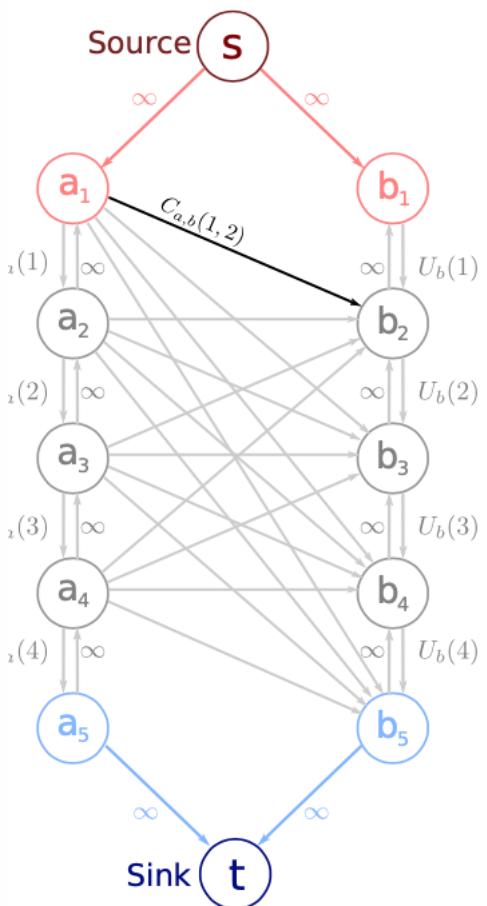
# Max-flow/Min-cut. Алгоритм Форда-Фалкерсона



# Бинарные штрафы, одномерное изображение

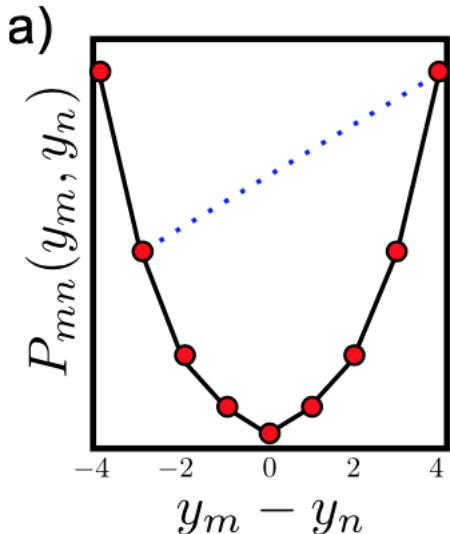


# Полутоновые изображения

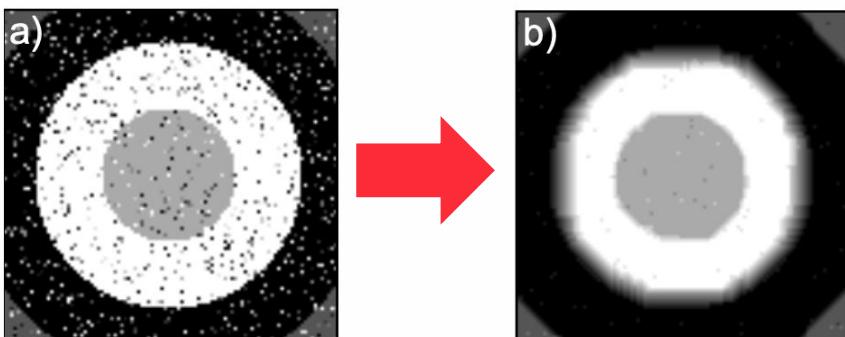
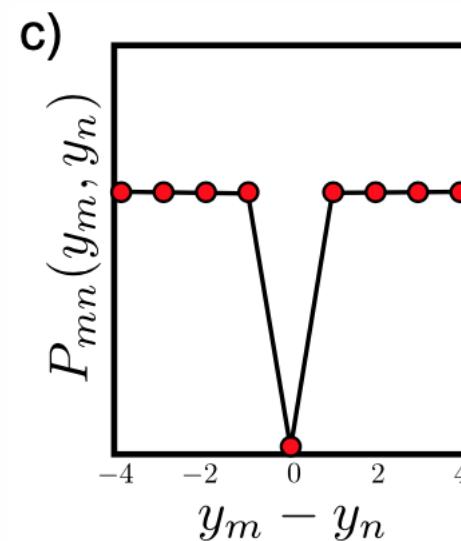
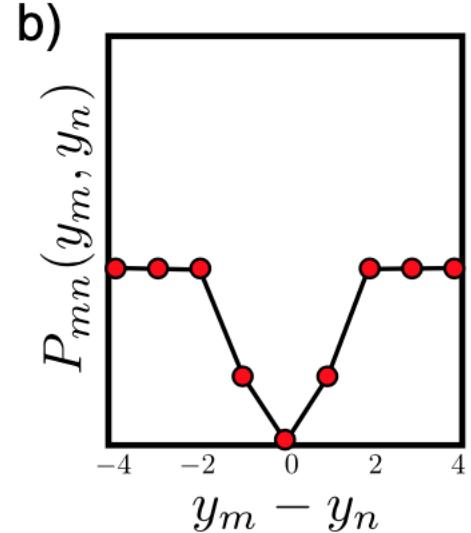


# Проблемы min-cut

Чтобы min-cut работал, нужны выпуклые потенциалы



На практике интереснее другие потери



Модель Поттса

# Alpha expansion

На каждом шаге выбирается одна метка (яркость)  $\alpha$  и задача рассматривается как бинарная ( $\alpha$  и не- $\alpha$ ).

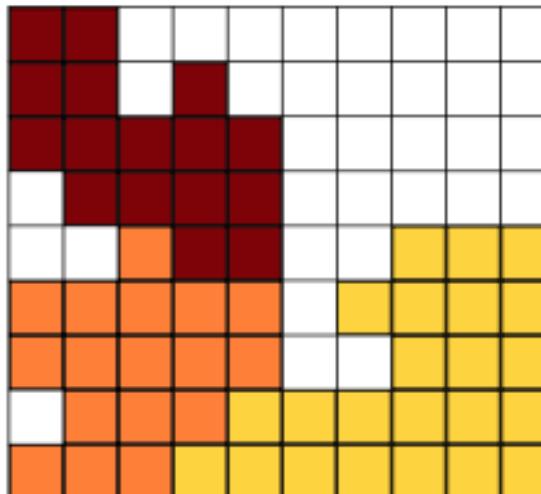
Штрафы должны быть метрикой

$$P(\alpha, \beta) = 0 \Leftrightarrow \alpha = \beta$$

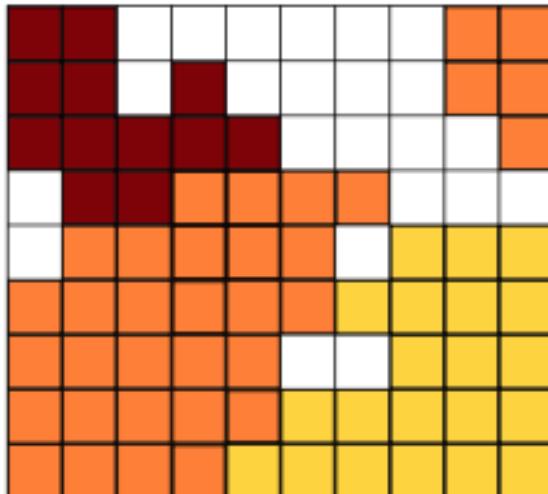
$$P(\alpha, \beta) = P(\beta, \alpha) \geq 0$$

$$P(\alpha, \beta) \leq P(\alpha, \gamma) + P(\gamma, \beta).$$

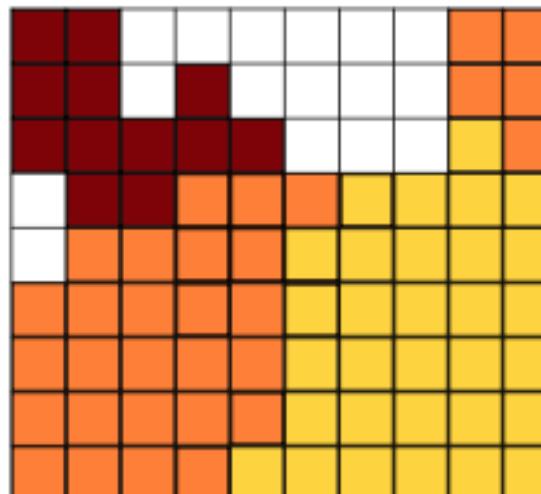
a)



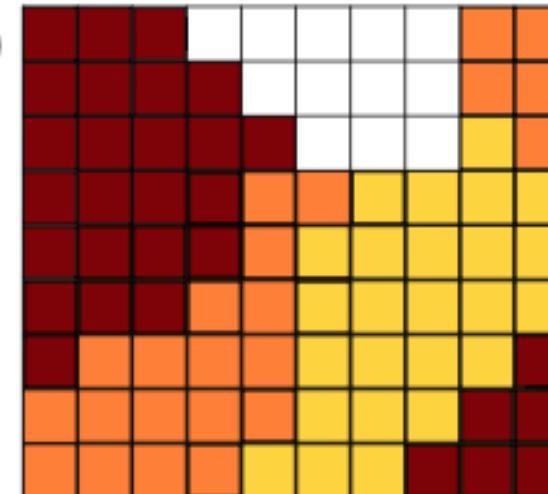
b)



c)



d)



# Alpha expansion. Пример



Исходное изображение

Расширение черного цвета волос

Расширение цвета ботинок

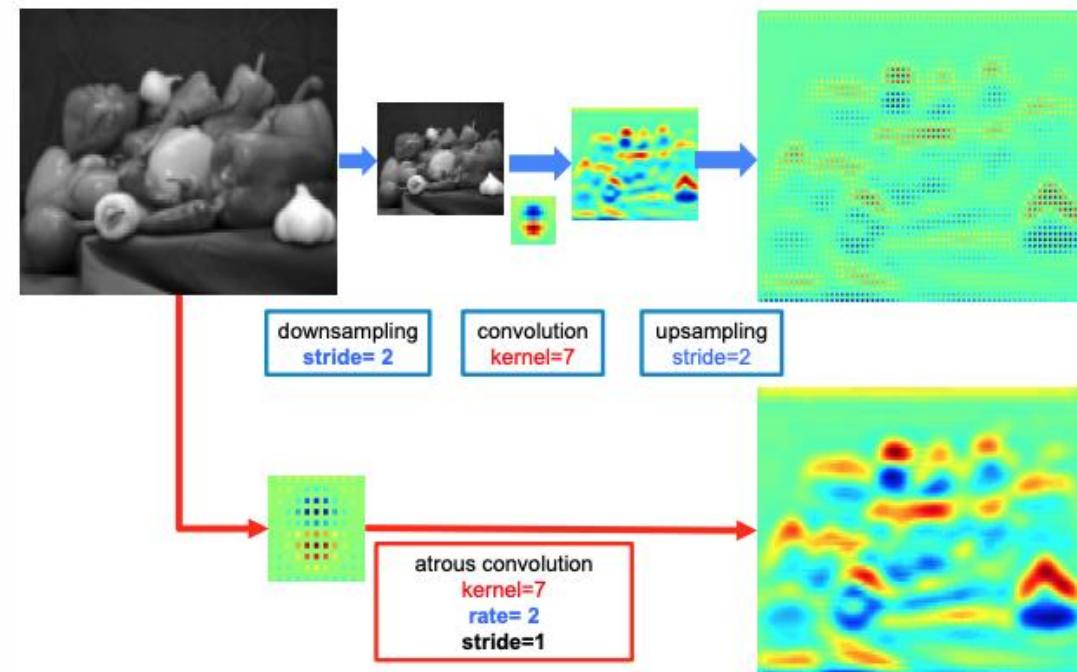
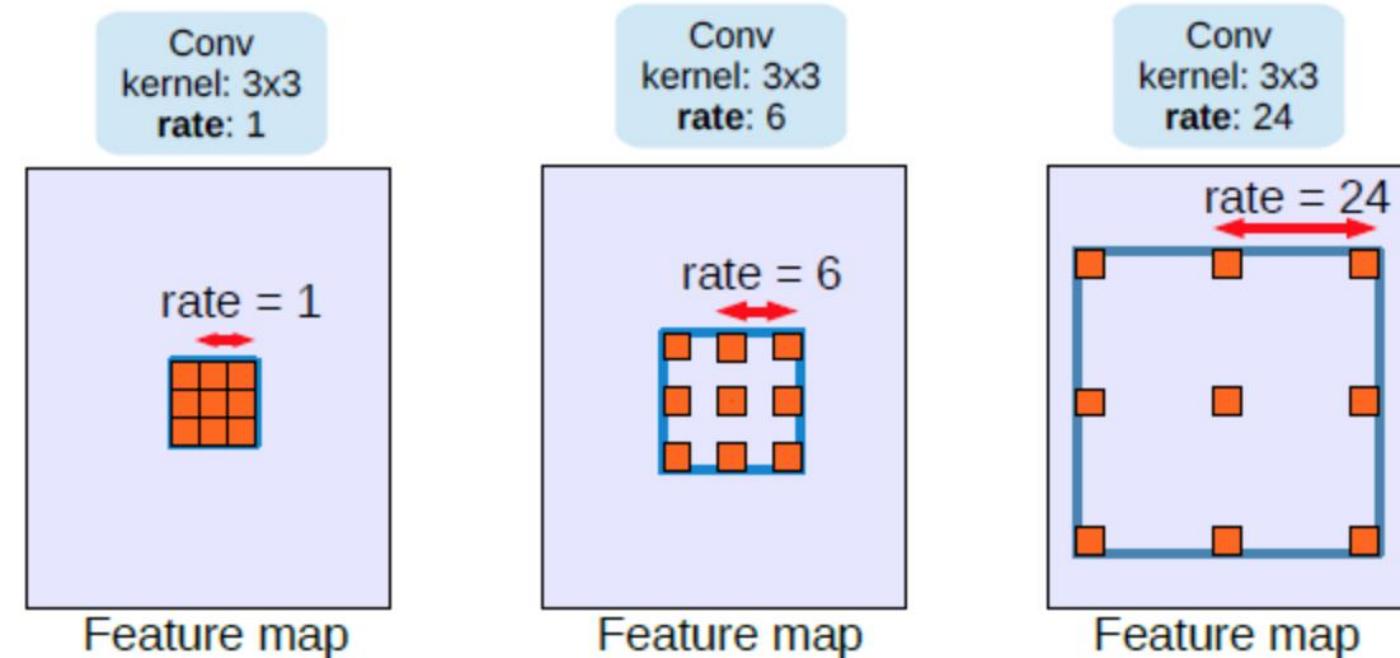
Расширение цвета брюк

Расширение цвета кожи

Расширение фона

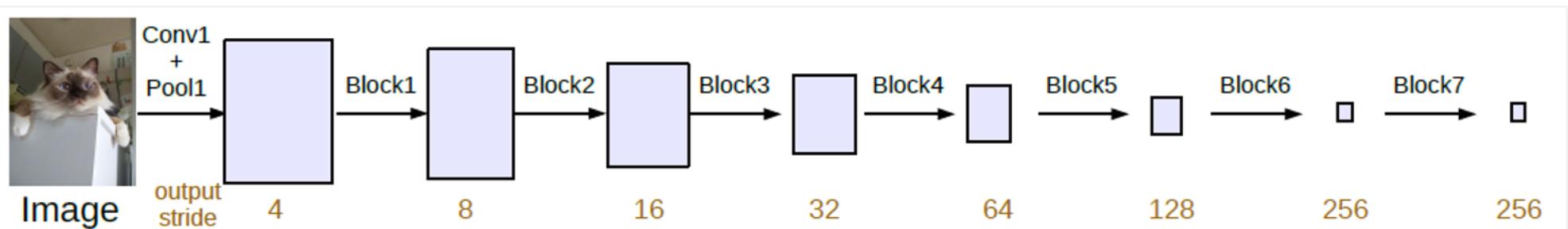
# Atrous convolution

$$y[i] = \sum_k x[i + r \cdot k] w[k]$$

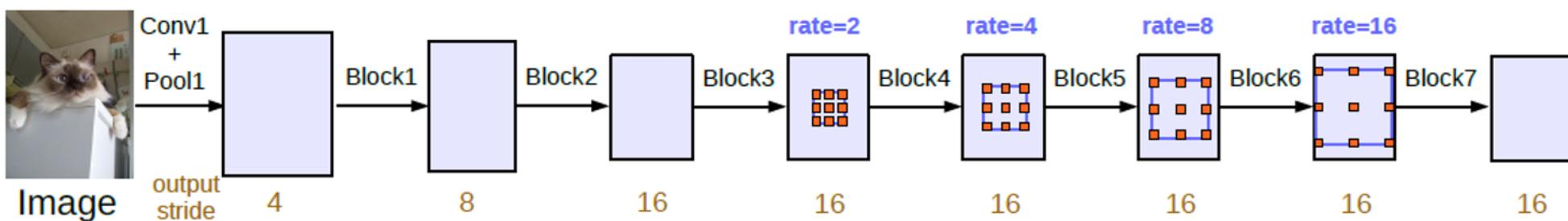


# pooling vs atrous convolution

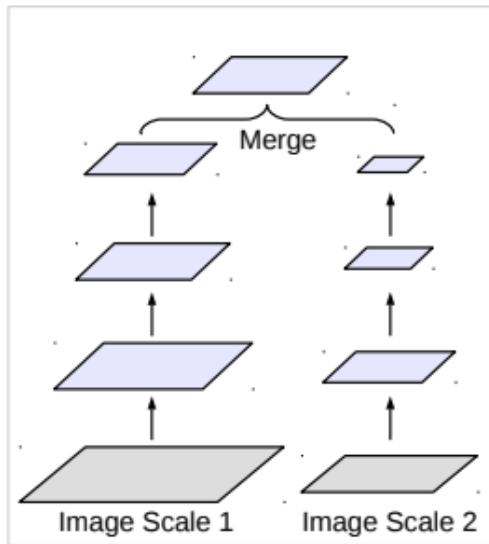
Стандартная сверточная сеть



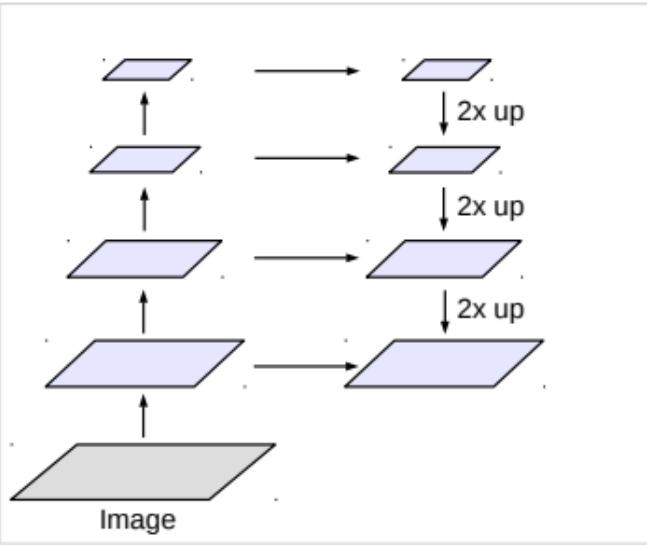
Atrous свертки



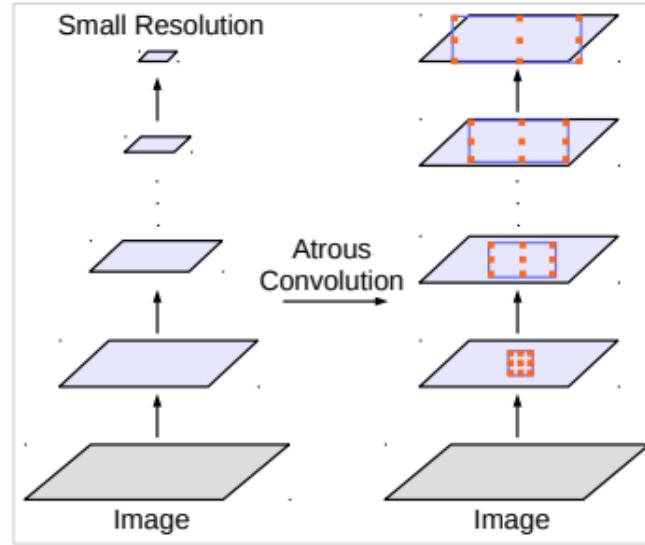
# Multi-scale: Atrous Spatial Pyramid Pooling (ASPP)



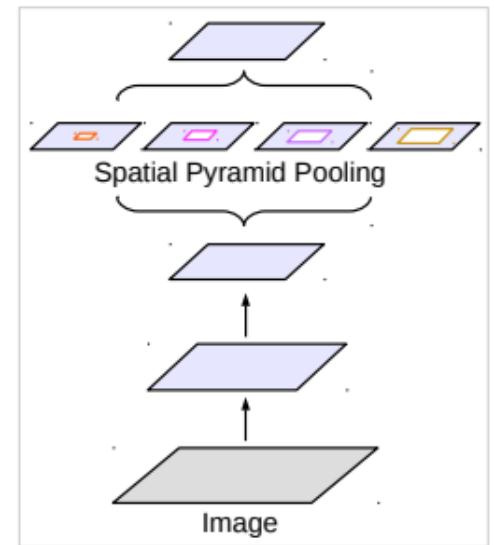
(a) Image Pyramid



(b) Encoder-Decoder



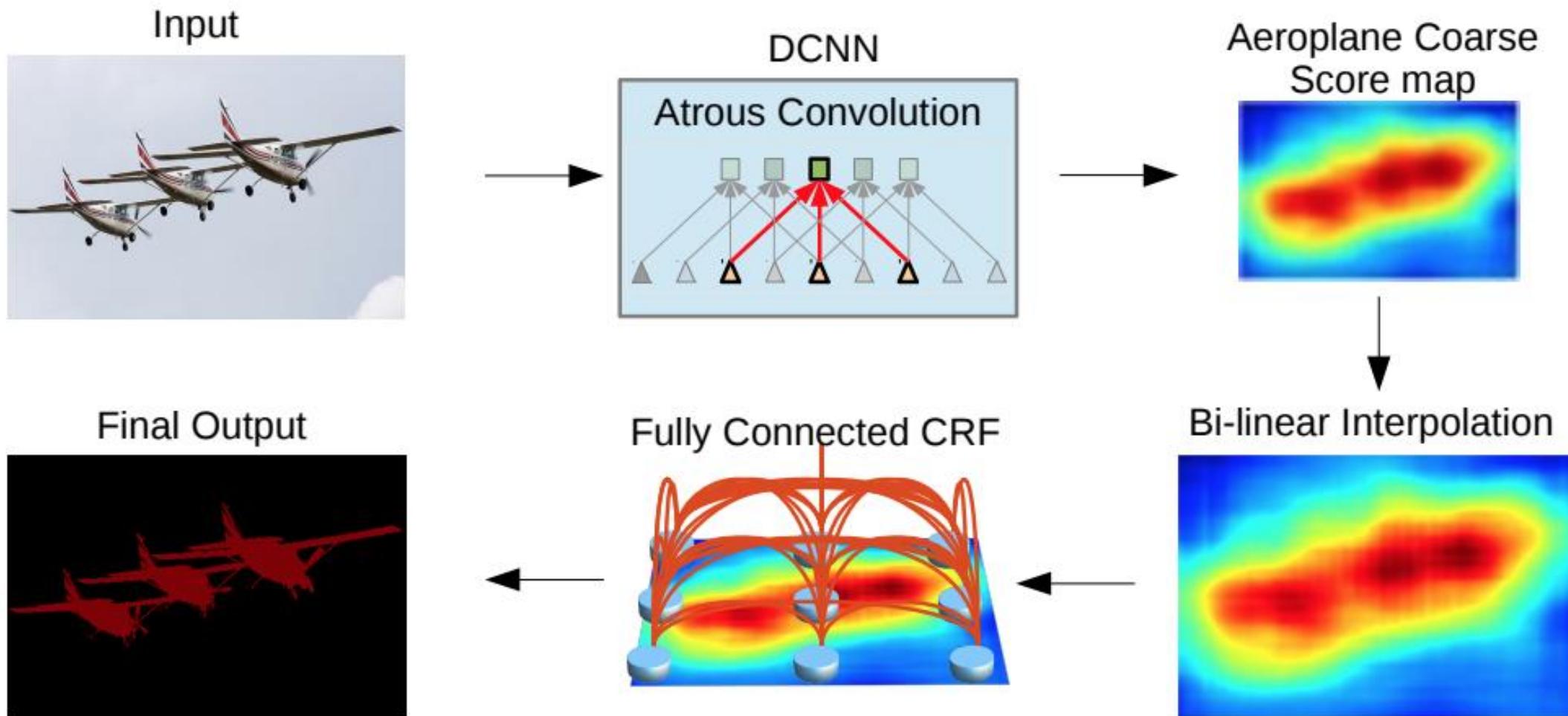
(c) Deeper w. Atrous Convolution



(d) Spatial Pyramid Pooling

“multi-scale features” in atrous SPP: multiple parallel filters with different rates

# DeepLab: нейронные сети+CRF



# CRF в DeepLab

$$E(\mathbf{x}) = \sum_i \theta_i(x_i) + \sum_{ij} \theta_{ij}(x_i, x_j)$$

Предсказания сверточной сети:

$$\theta_i(x_i) = -\log P(x_i),$$

Полносвязный граф (связаны все точки i и j)

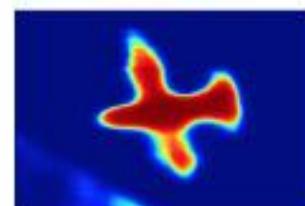
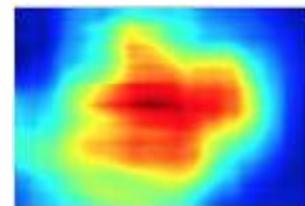
$$\theta_{ij}(x_i, x_j) = \mu(x_i, x_j) \left[ w_1 \exp \left( -\frac{\|p_i - p_j\|^2}{2\sigma_\alpha^2} \right) - \frac{\|I_i - I_j\|^2}{2\sigma_\beta^2} \right] + w_2 \exp \left( -\frac{\|p_i - p_j\|^2}{2\sigma_\gamma^2} \right)$$

Модель Поттса

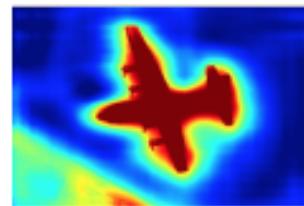
Билатеральный  
фильтр



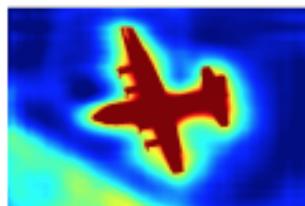
Image/G.T.



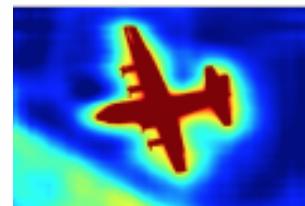
DCNN output



CRF Iteration 1

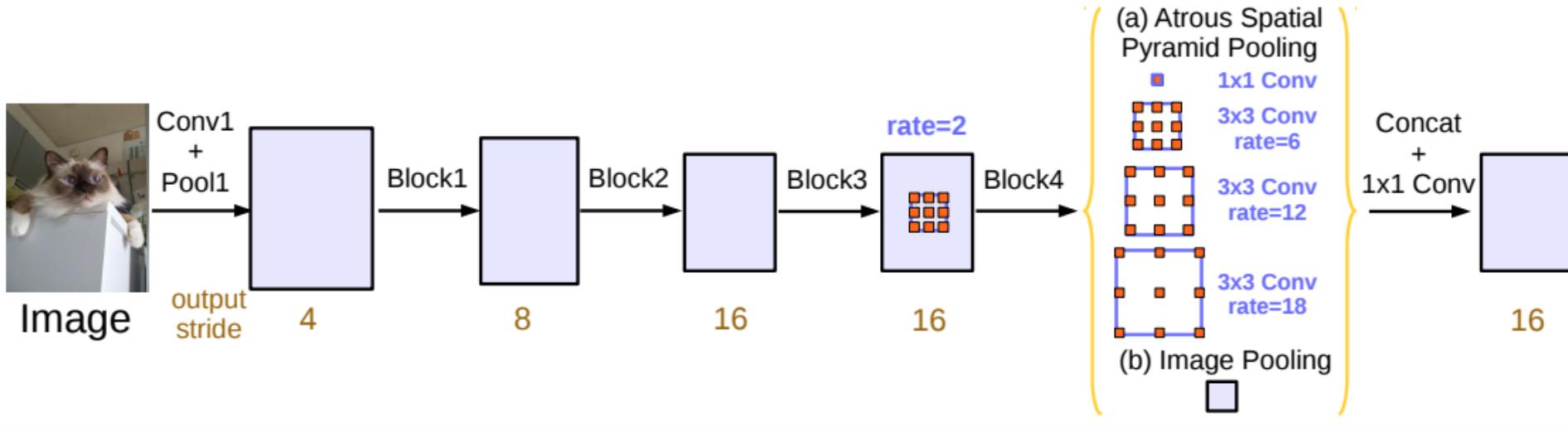


CRF Iteration 2

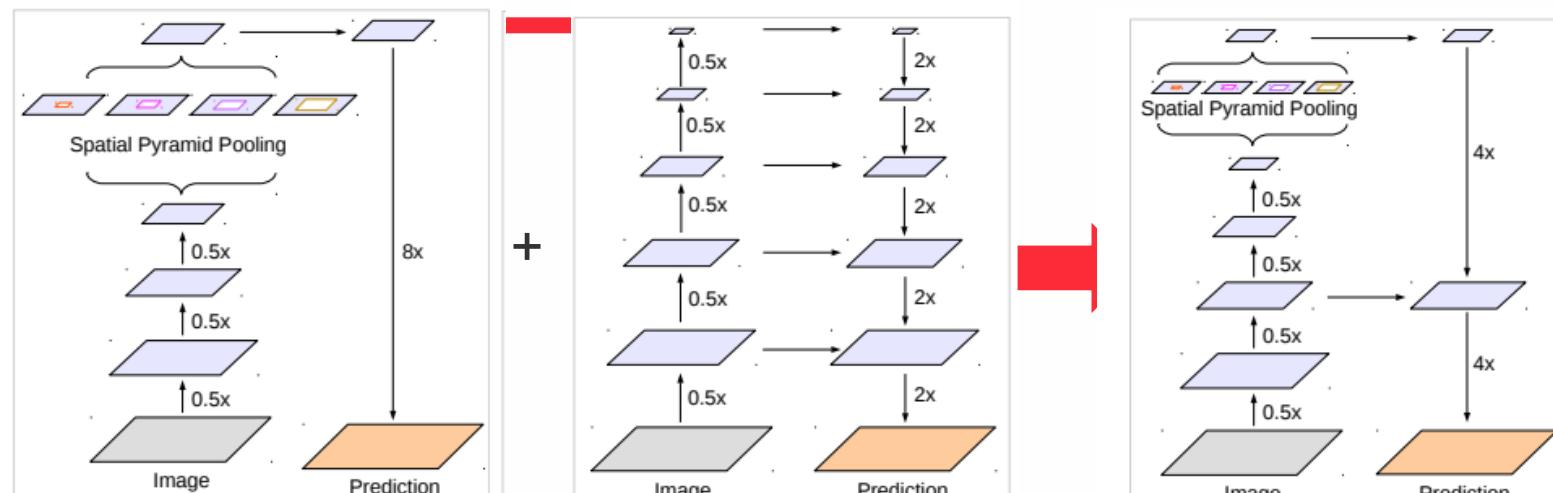


CRF Iteration 10

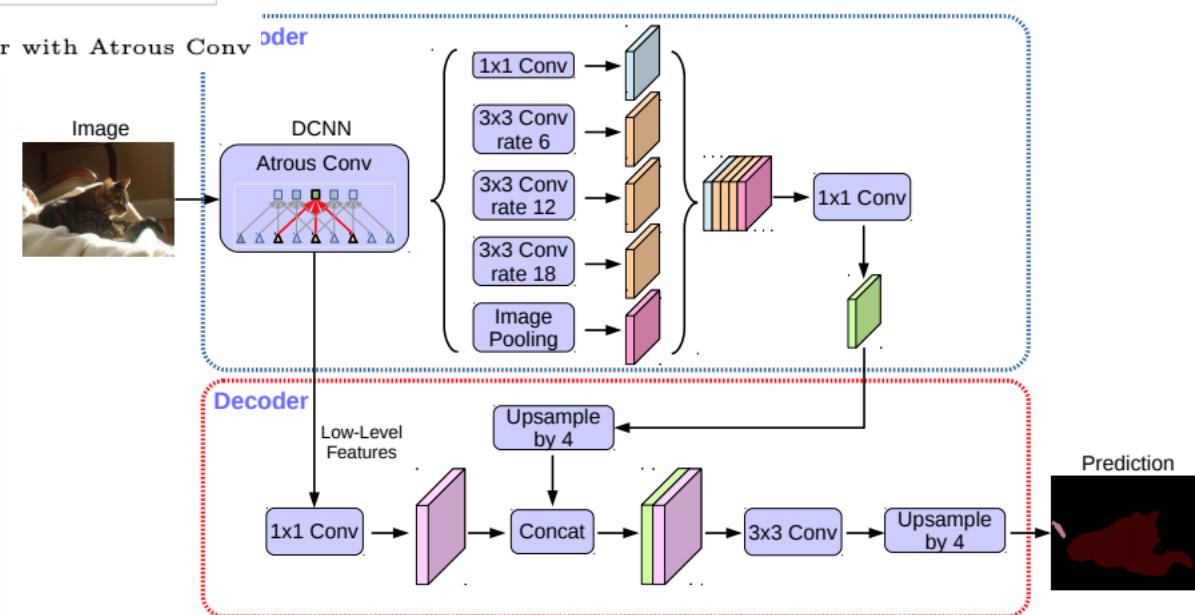
# DeepLab v3: DeepLab - CRF



# DeepLab v3+



К DeepLab v3 добавлен декодер для повышения качества сегментации на границах объектов



# Перенос стиля

# Постановка задачи

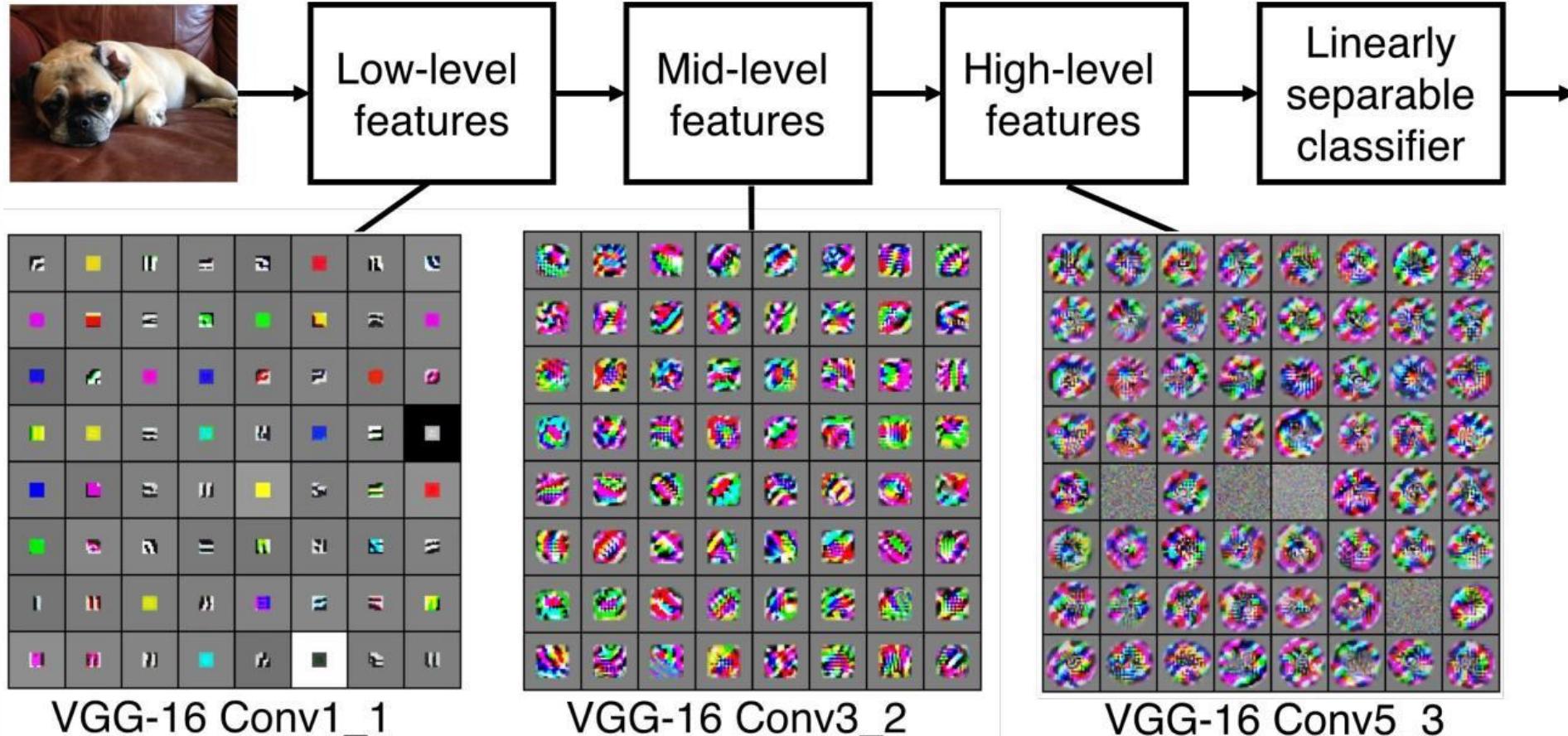
Модифицировать заданное входное изображение (content) с использованием некоторого стилевого (style) изображения так, чтобы по части признаков оно было похоже на исходное (содержание), а часть – на целевое (стиль)



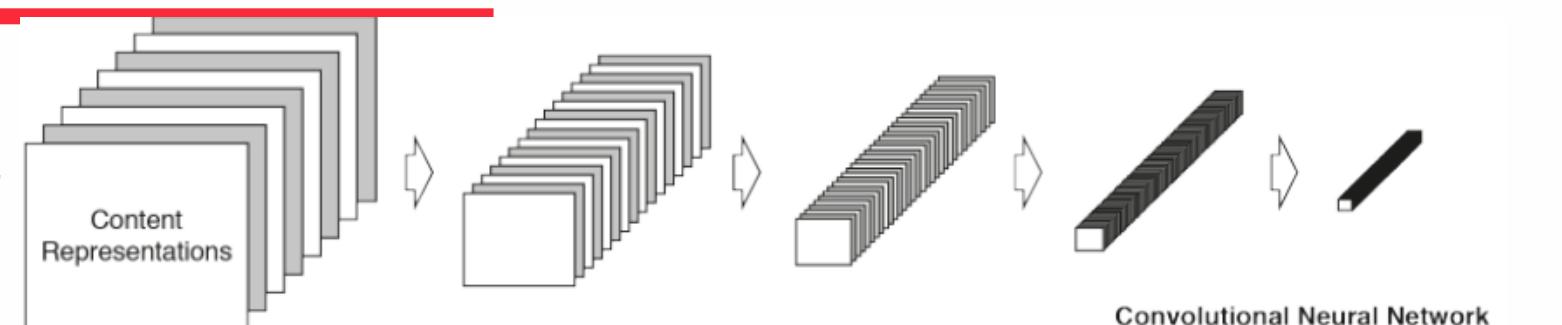
Neural Style Transfer: A Review, <https://arxiv.org/pdf/1705.04058.pdf>

Gatys et al. 2015, <https://arxiv.org/pdf/1508.06576.pdf>

# Фильтры VGG16



# Content representation



Реконструкция  
содержания



conv1\_1



conv2\_1



conv3\_1



conv4\_1



conv5\_1

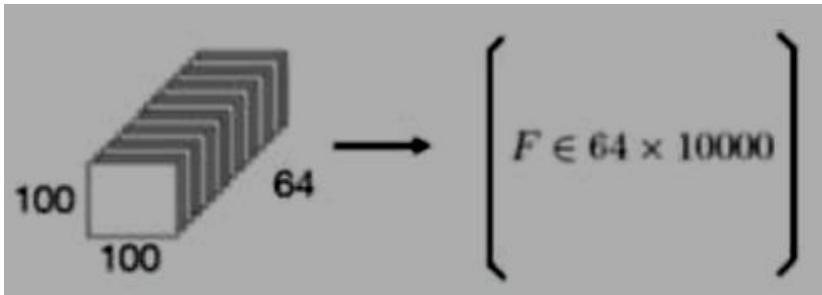
Функция  
потерь

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

# Как описать стиль?

«Стиль» как текстура: корреляция откликов фильтров

**Матрица Грама на  $l$ -м слое (корреляция между  $i$ -м и  $j$ -м фильтрами)**



$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

ФУНКЦИЯ ПОТЕРЬ

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Gatys et al. Texture Synthesis Using Convolutional Neural Networks, NIPS 2015

Gatys et al. 2015, <https://arxiv.org/pdf/1508.06576.pdf>

# Neural style transfer

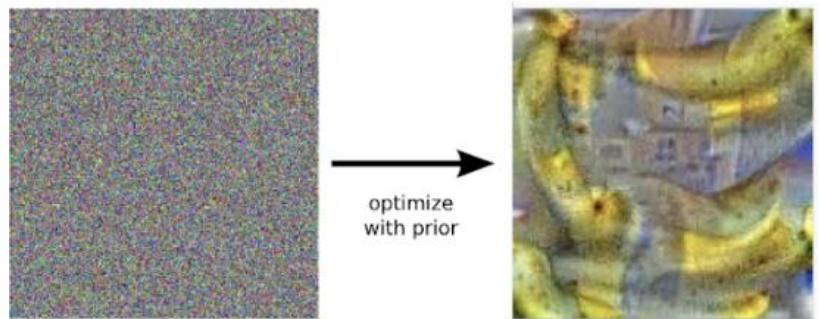
1. Извлекаются признаки входного и стилевого изображений

2. На вход сети подается изображение типа «белый шум» (случайный набор пикселей с нормальным распределением).

3. Выполняется оптимизация (Limited-memory BFGS) пикселей изображения из п.2 для функции потерь вида

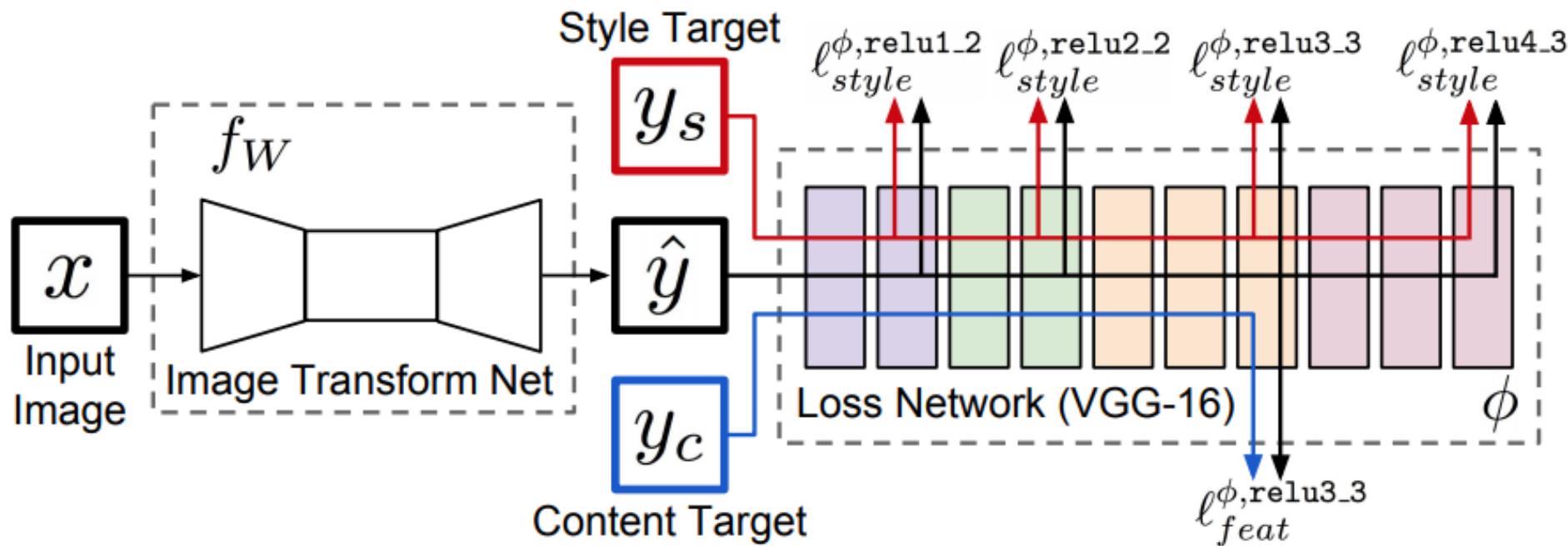
$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Веса сети не меняются!



- хорошее качество
- очень медленно

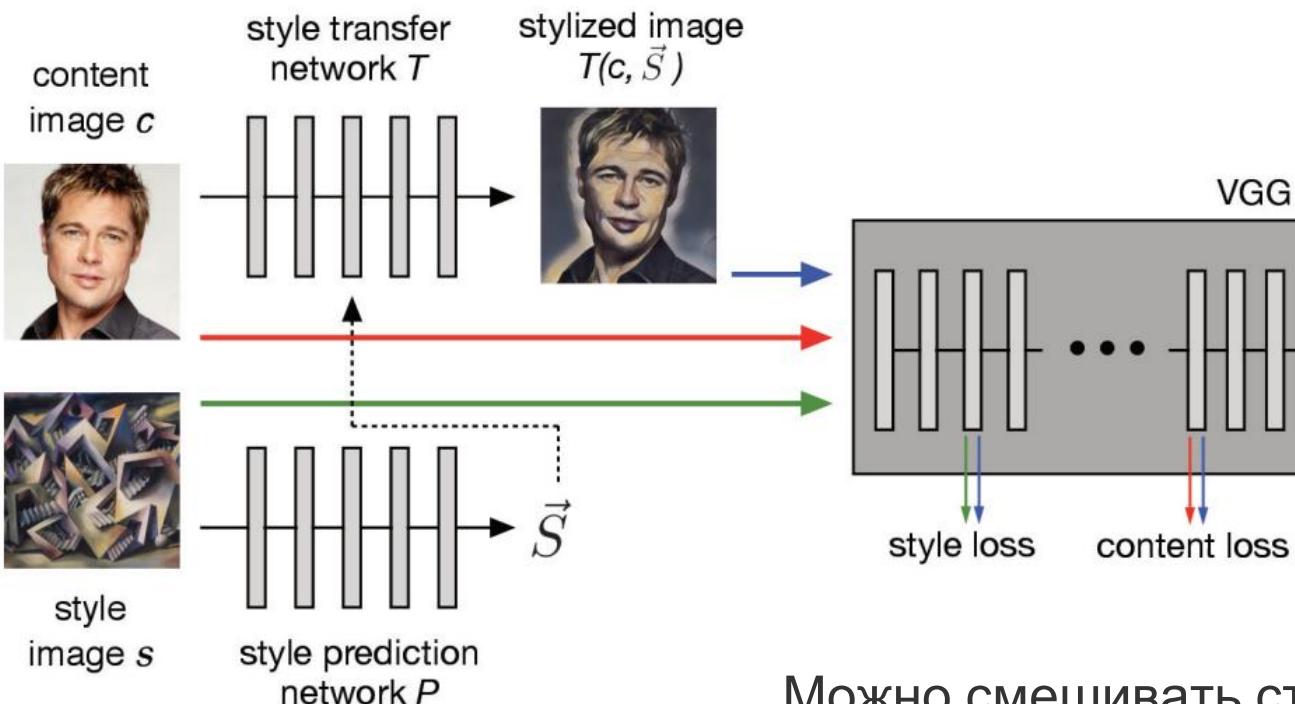
# Perceptual loss



$$W^* = \arg \min_W \mathbf{E}_{x, \{y_i\}} \left[ \sum_{i=1} \lambda_i \ell_i(f_W(x), y_i) \right]$$

- Одна сеть для каждого стилевого изображения
- Total Variance loss для сглаживания выходного изображения
- VGG-16 вместо VGG-19

# Fast style transfer (Magenta)

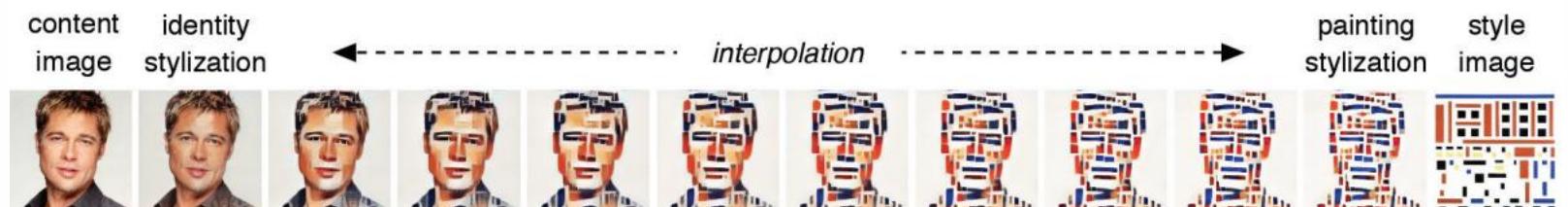


Обучение на большом числе примеров стилей



Можно использовать любое стилевое изображение

Можно смешивать стили исходного и целевого изображений





# Перейдем к примерам

---

<https://github.com/HSE-asavchenko/MADE-mobile-image-processing/tree/master/lesson7/src>