Transfer learning. Distillation. Tips and tricks

План лекции

- Transfer learning
- Distillation
- Quantization
- Pruning
- Interpretation

Finetuning

Предположим, у нас есть обученная модель под какую-то задачу.

Новая задача отличается, но в целом похожа.

Что делать?

Finetuning

Предположим, у нас есть обученная модель под какую-то задачу.

Новая задача отличается, но в целом похожа.

Что делать?

• Обучать с нуля? → можно, но долго, затратно

• Использовать уже обученную модель? → не можем, задача отличается

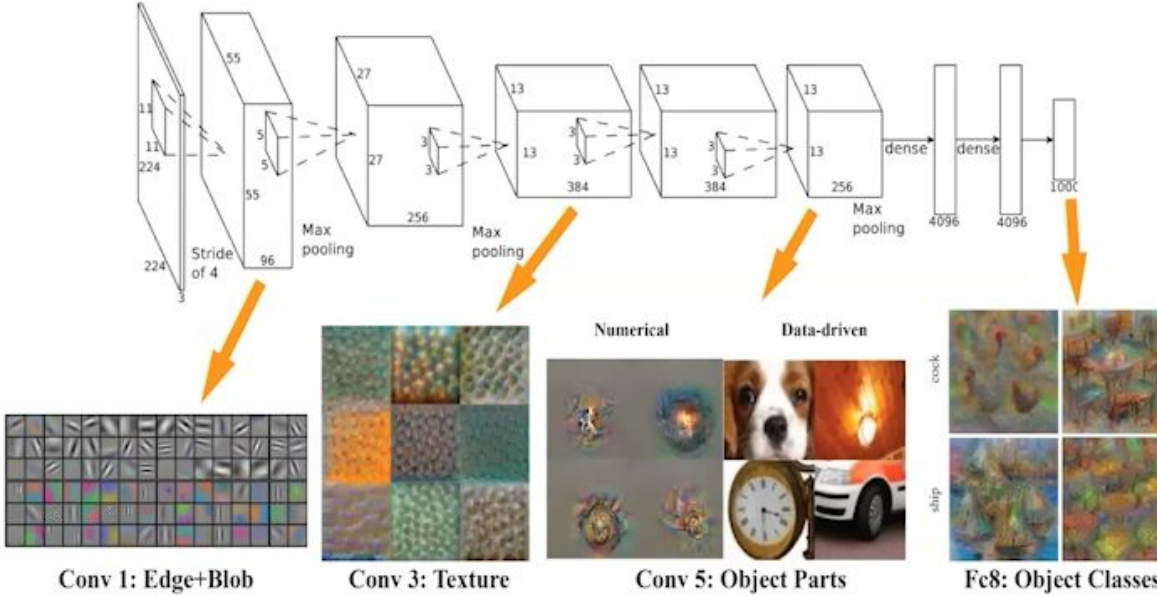
• Переиспользуем знания старой моделью для новой задачи!

Что из себя представляют значения промежуточных слоёв (для любых архитектур нейросетей)? Что выучивает нейросеть?

Finetuning

Внутри выучены полезные фичи, можно о них мыслить, как показано на картинке.

Значит, можно использовать основную часть сети, а сверху добавить новую необученную голову/классификатор



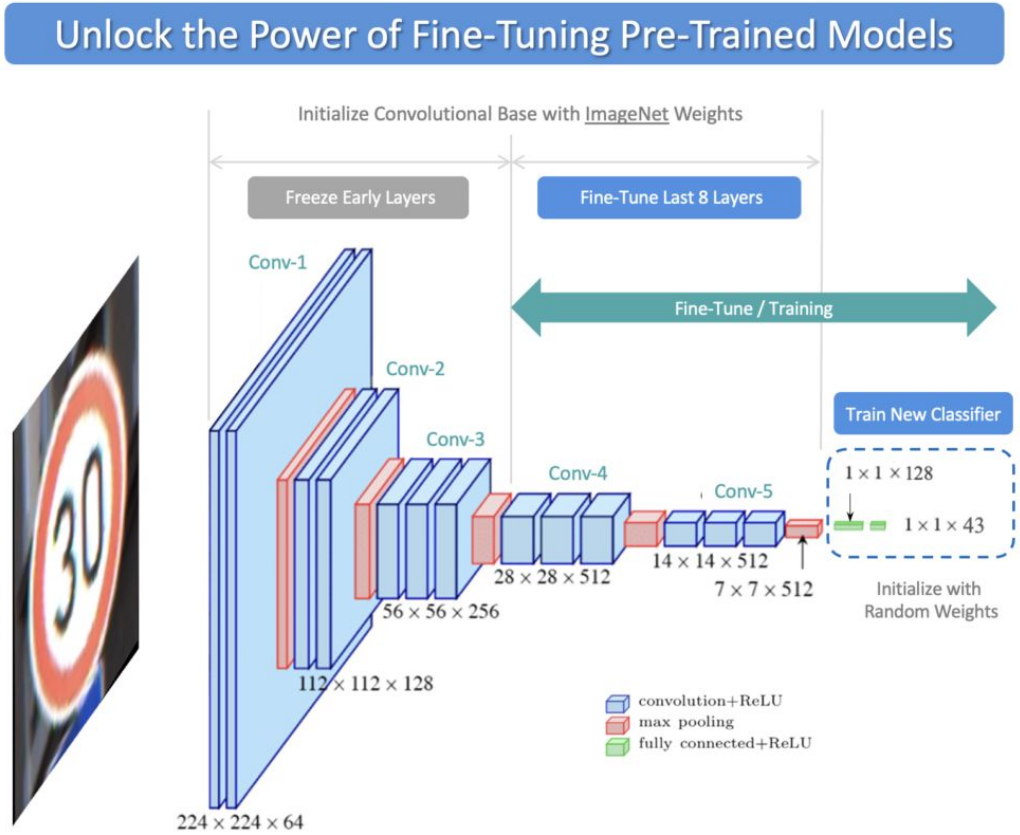
Finetuning

Можно “заморозить” первые слои основной сети, чтобы они не обновлялись

Можно не замораживать

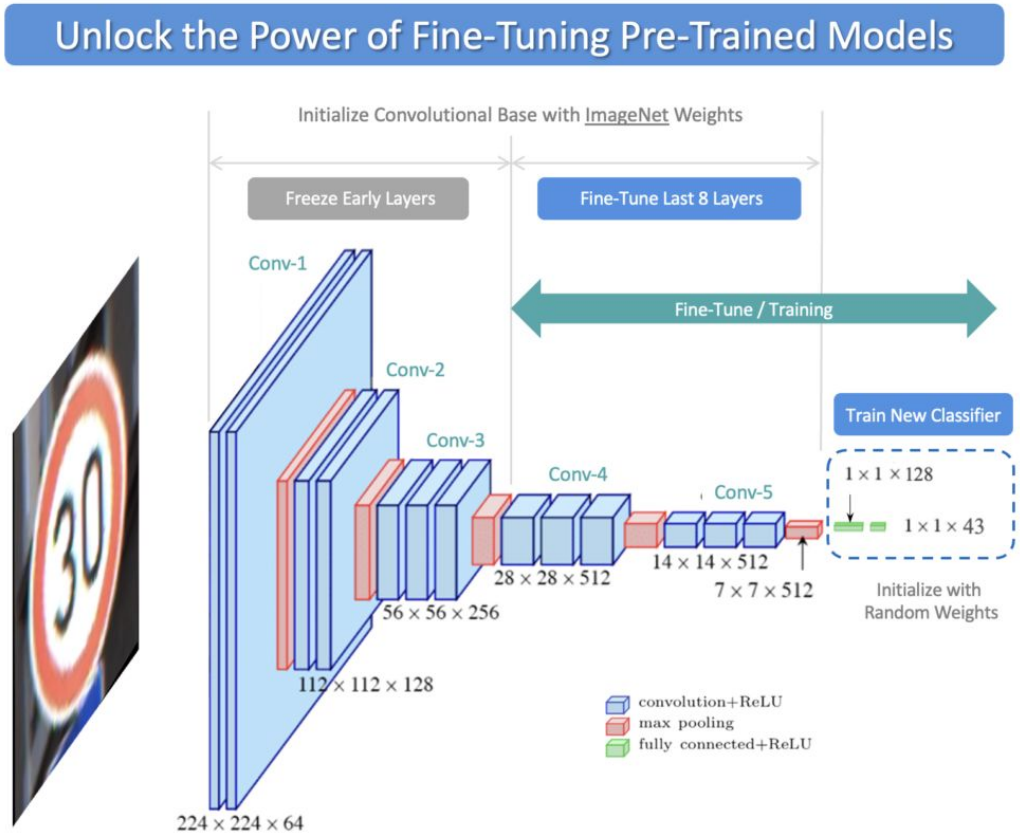
Важно не “развалить” сеть со слишком большим lr

Низкоуровневые выученные фичи сильно помогают учиться быстрее, так как выучили полезные фичи на больших наборах данных



Finetuning

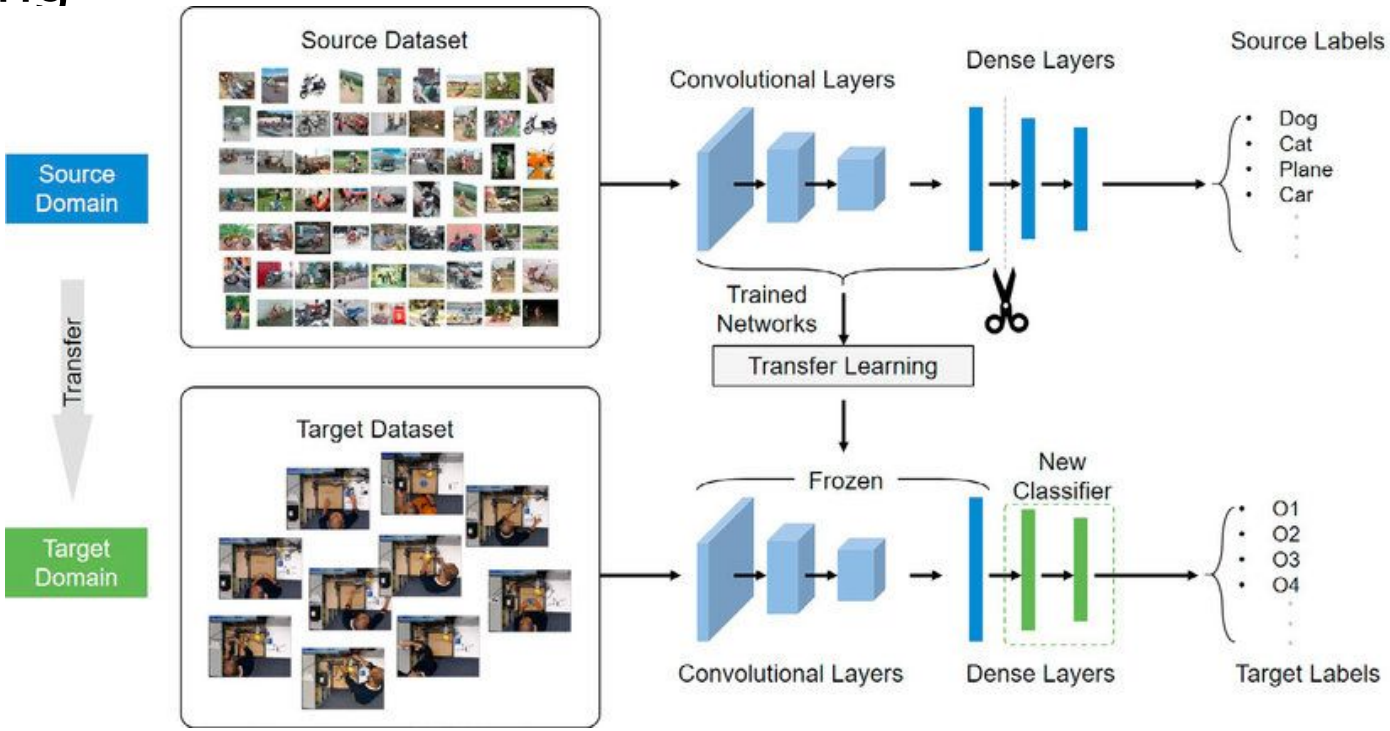
Интуитивно понятно, что чем сильнее отличаются домены, тем больше слоев нужно переобучать/обучать.



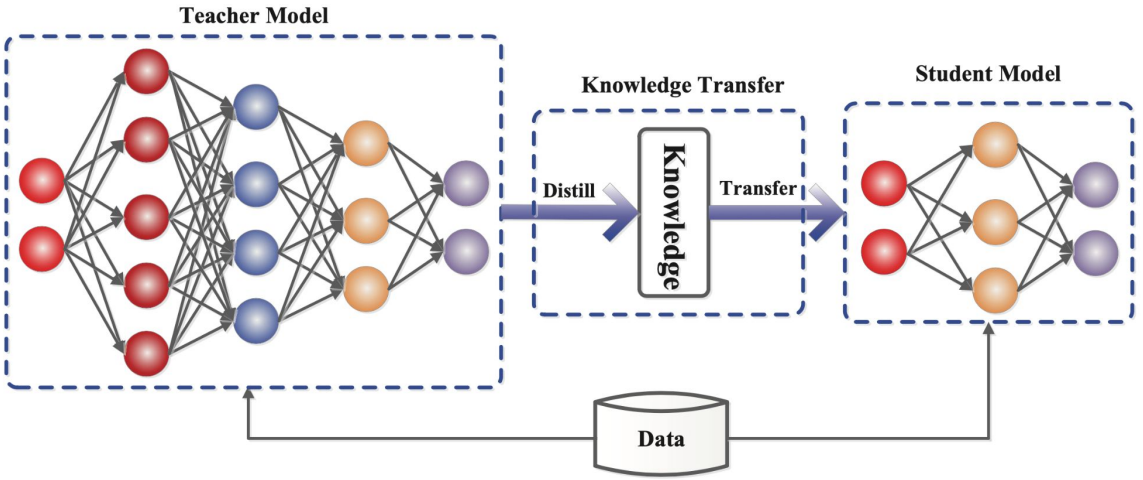
Transfer learning

А в этом примере используем обученную нейросеть просто как некоторую функцию, являющуюся фича-экстрактором.

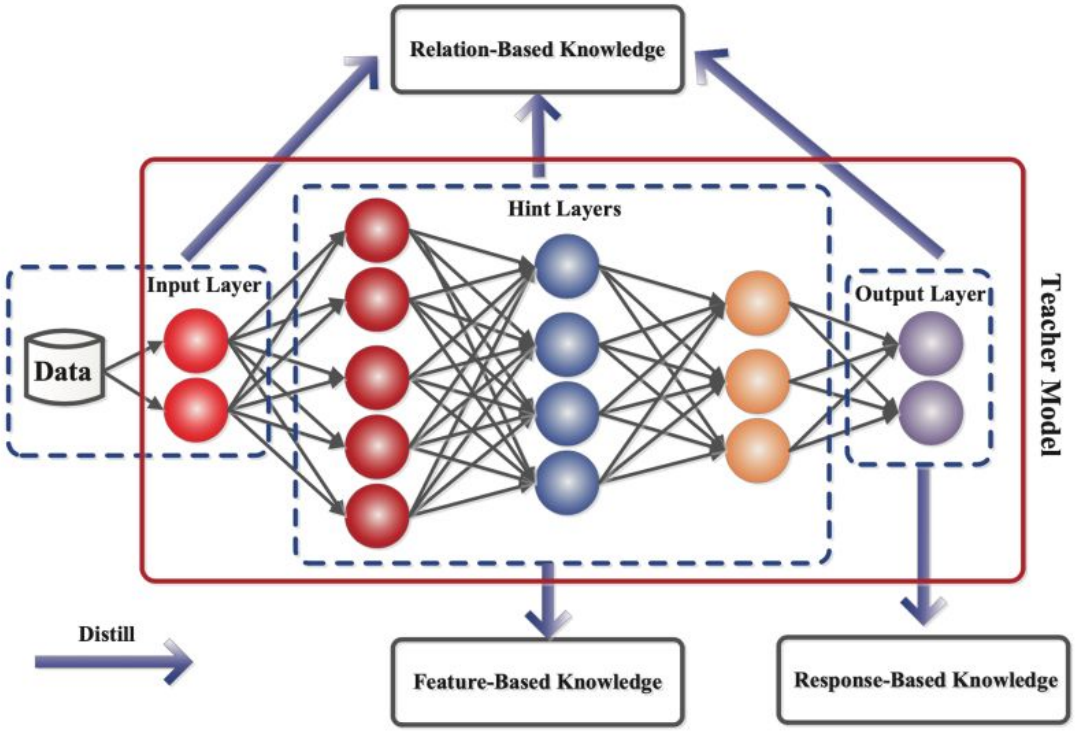
И дообучаем только голову/классификатор /новые настаканные слои



Knowledge distillation



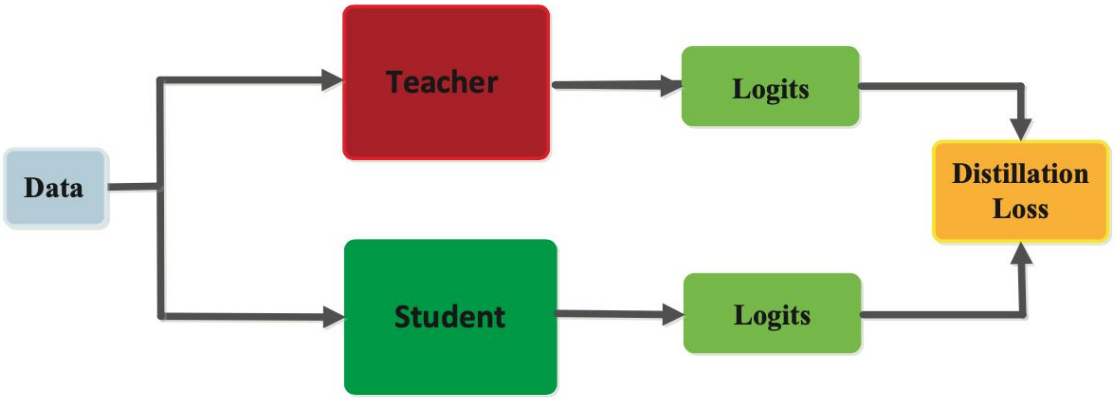
Knowledge distillation



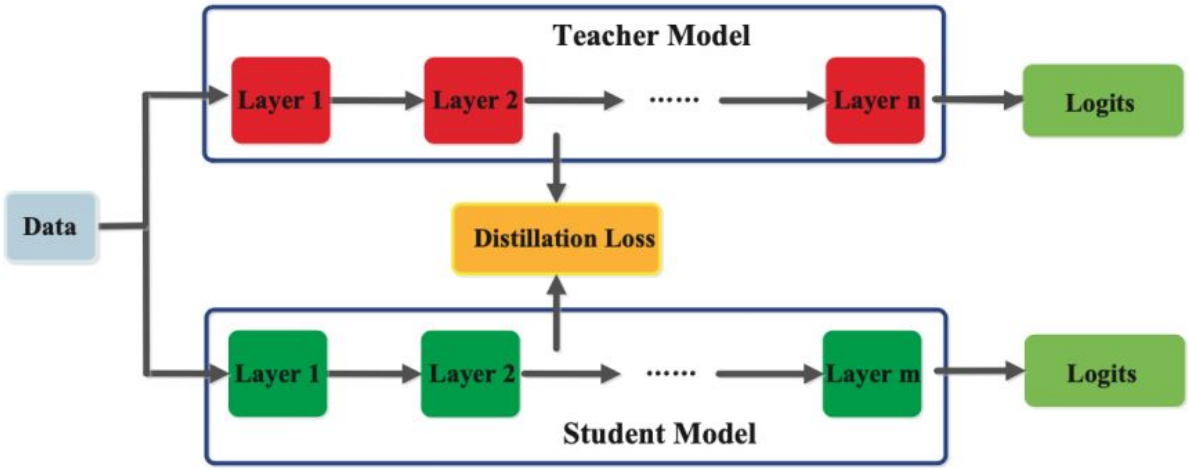
Knowledge distillation

Можно об этом думать так:

В своих моделях большая обученная модель научилась определять некоторую похожесть между объектами разных классов: пушистыми животными/ автомобилями, и это знание передается модели студента

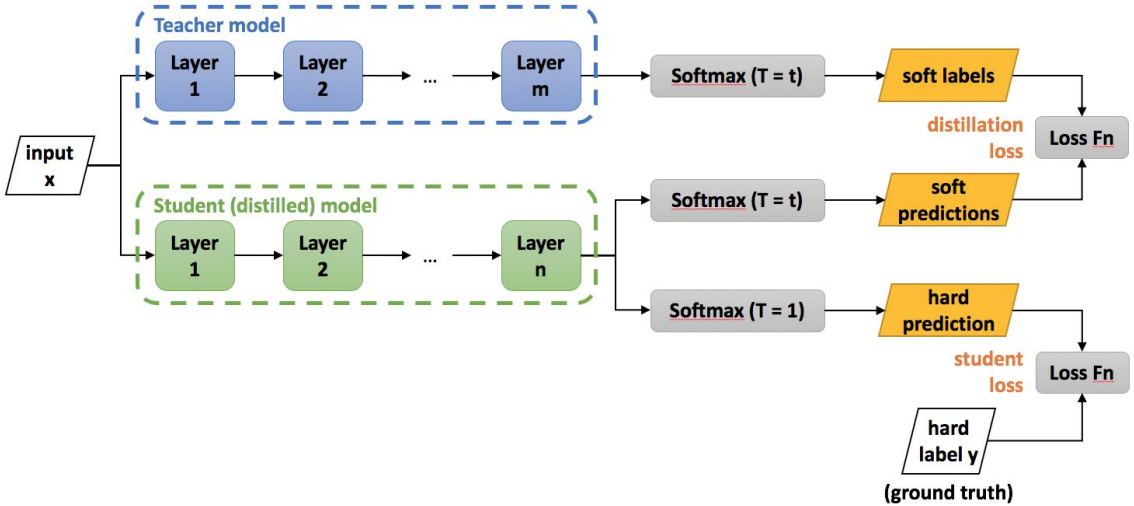


Knowledge distillation



Knowledge distillation

https://intellabs.github.io/distiller/knowledge_distillation.html



$$\mathcal{L} = \alpha \mathcal{L}_{CE} + (1-\alpha)\mathcal{L}_{distill} = \alpha \mathcal{L}_{CE} + (1-\alpha)\left\| z^{(T)} - z^{(S)} \right\|_2^2$$

$$\mathcal{L}(x;W) = \alpha * \mathcal{H}(y, \sigma(z_s; T=1)) + \beta * \mathcal{H}(\sigma(z_t; T=\tau), \sigma(z_s, T=\tau))$$

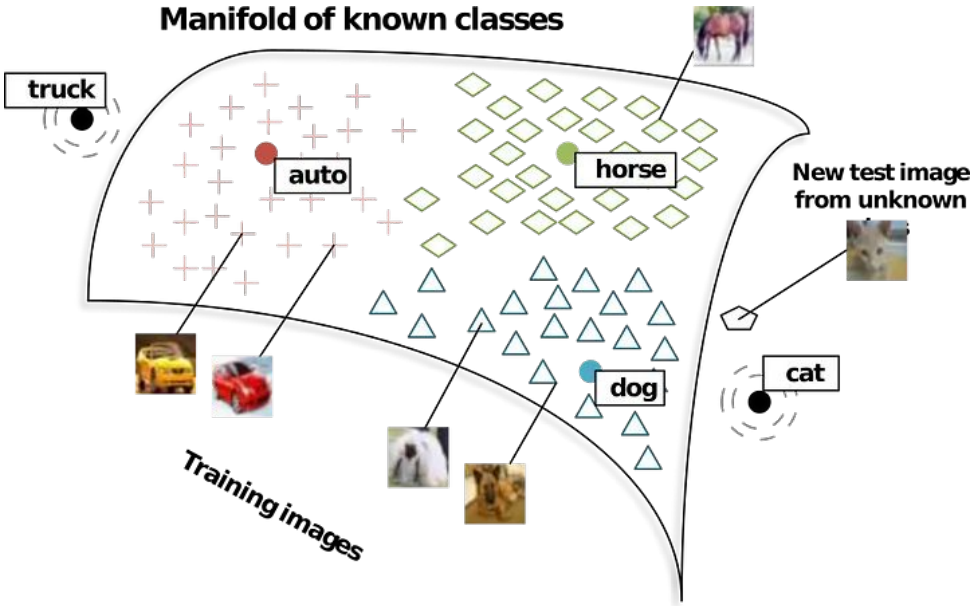
Few-shot learning

Что делать, если примеров для новой задачи прямо совсем мало(По 1-5 штук на класс)?

Few-shot learning

Решение – KNN на фичах из предобученной сети.

Так как есть основание полагать, что, обучаясь на большом объеме данных они научилась различать объекты и разносить из в пространстве признаков.



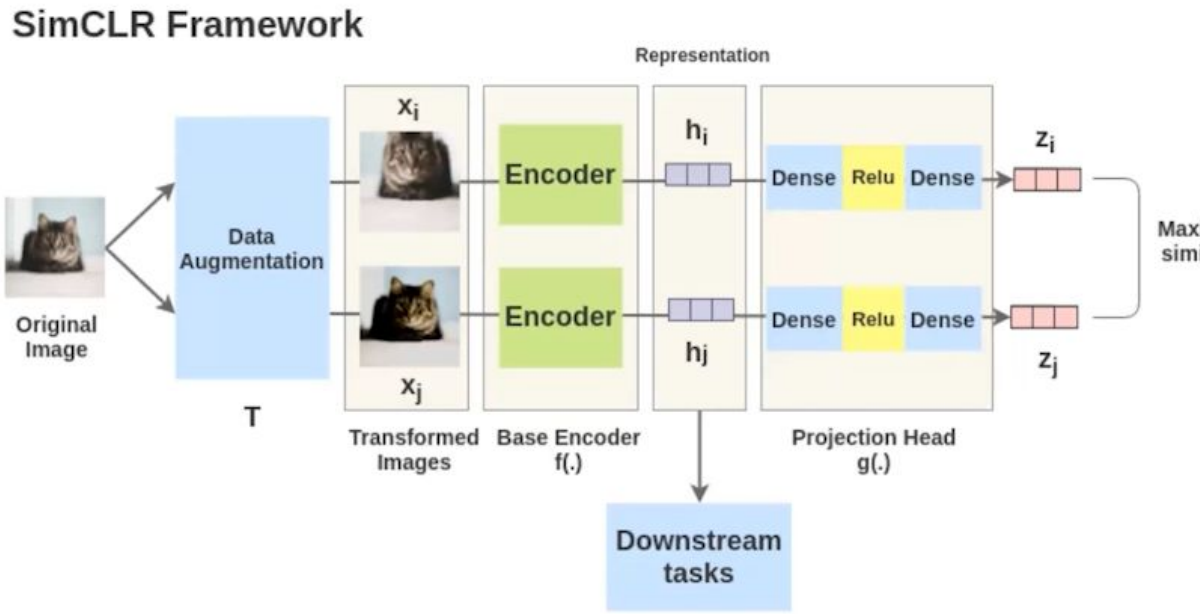
Zero shot learning

Что если размеченных новых данных вообще нет, есть только данные, без разметки.

Zero shot learning

Аугментации

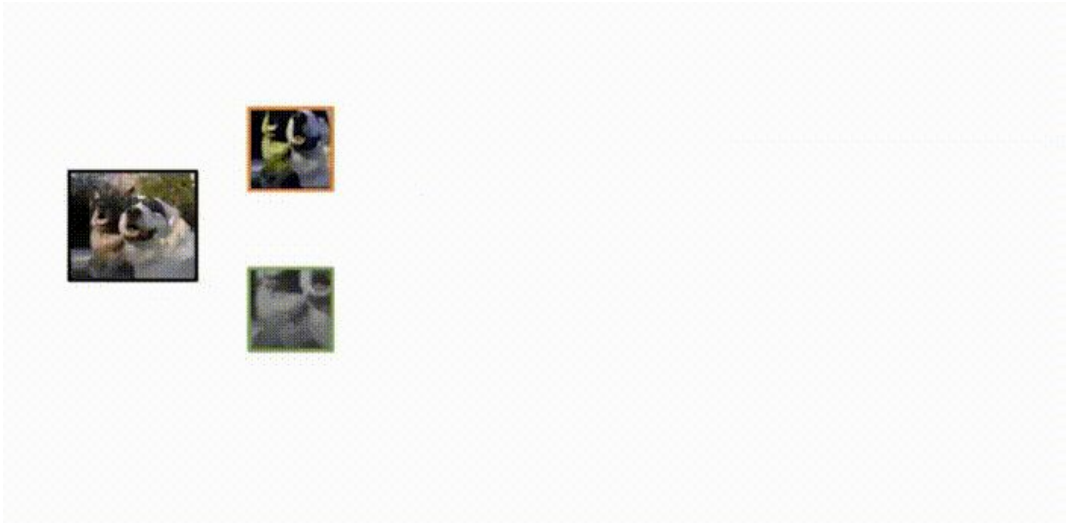
И предсказание, это один класс или нет



Zero shot learning

Аугментации

И предсказание, это один класс или нет



LR warmup

Если данные слишком разнородные, то неудачно перемешанный датасет может “сломать” обучением первыми несколькими неудачными батчами

Как это можно исправить?

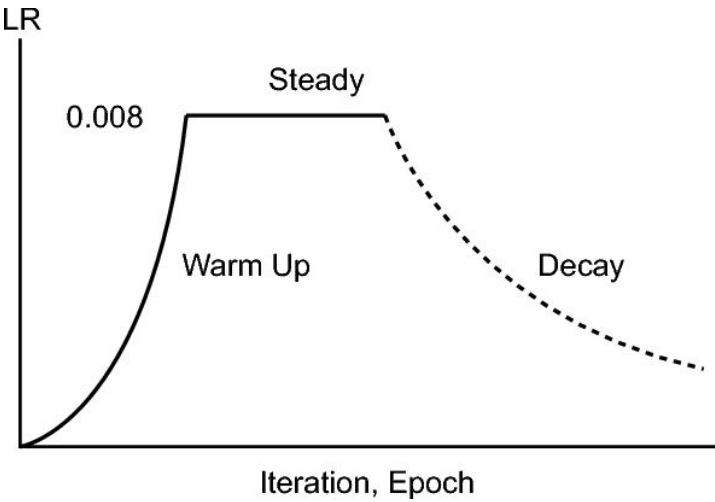
LR warmup

Если данные слишком разнородные, то неудачно перемешанный датасет может “сломать” обучением первыми несколькими неудачными батчами

Как это можно исправить?

Идея: Начнём с очень маленького LR и будем его постепенно увеличивать.

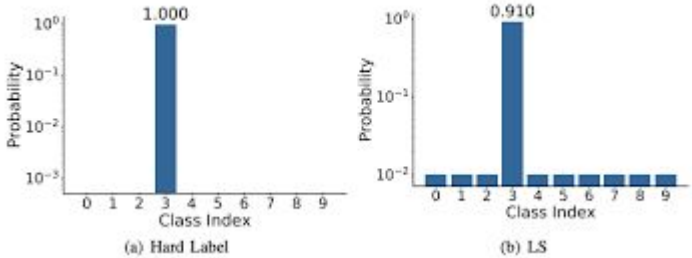
Период до стартового LR – разогрев.



Label smoothing

Мы не хотим, чтобы наша модель была слишком уверена в своих прогнозах.

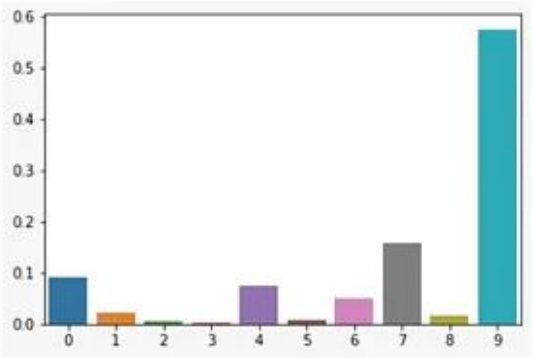
Применяя label smoothing, мы можем снизить достоверность модели и предотвратить ее скатывание к глубоким локальным минимумам в функции потерь, где чаще происходит переобучение.



Temperature

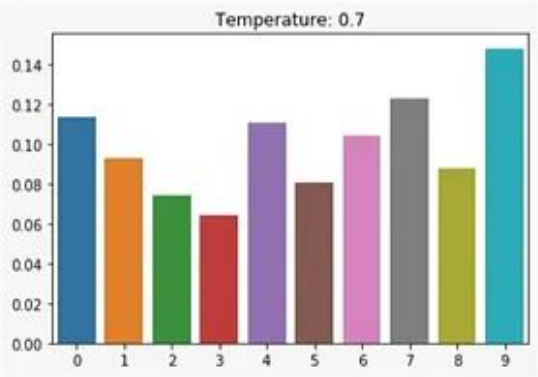
SOFTMAX WITHOUT TEMPERATURE (T=1)

$$\frac{e^{z_i}}{\sum_j e^{z_i}}$$



SOFTMAX WITH TEMPERATURE

$$\frac{e^{z_i/T}}{\sum_j e^{z_i/T}}$$



LESS ENTROPY → MORE ENTROPY

INCREASE IN ENTROPY WITH INCREASE IN T

Temperature

Температура – гиперпараметр, но будем её менять в ходе обучения (как делаем с LR).

На ранних итерациях хотим высокое значение t, т.е. уменьшаем уверенность модели в своих собственных ответах.

На поздних итерациях хотим низкое значение t, т.е. высокую уверенность модели в своих собственных ответах.

Начнём с довольно большого значения и будем постепенно снижать.

Noise

Добавление шума к данным:

• Усложняем нахождение простых локальных связей, т.к. шум портит зависимости

• Модели приходится (похоже на dropout) обучать несколько вариантов связей на одни

и те же задачи сквозь нейросеть

• Улучшаем обобщающую способность

• Увеличиваем защиту от атак или неудачных примеров

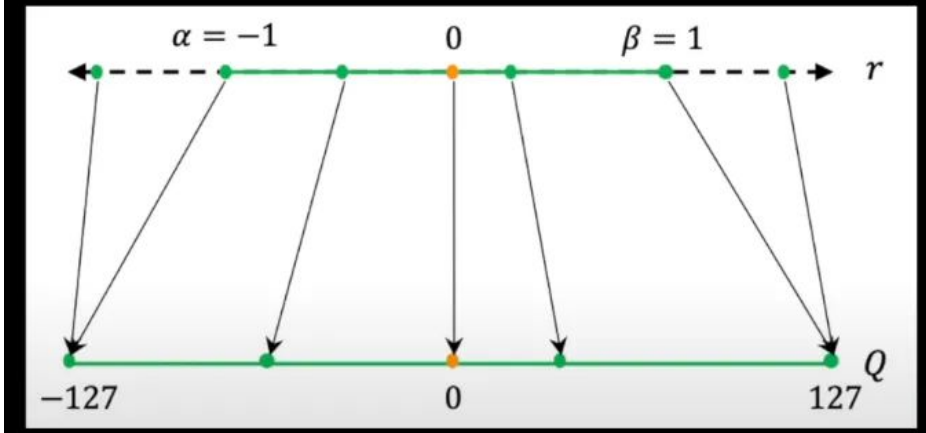
Quantization

Квантизация - это метод, позволяющий снизить вычислительные затраты и затраты памяти на выполнение за счет представления весов и активаций с использованием типов данных низкой точности, таких как 8-разрядное целое число (int8) вместо обычного 32-разрядного числа с плавающей запятой (float32).

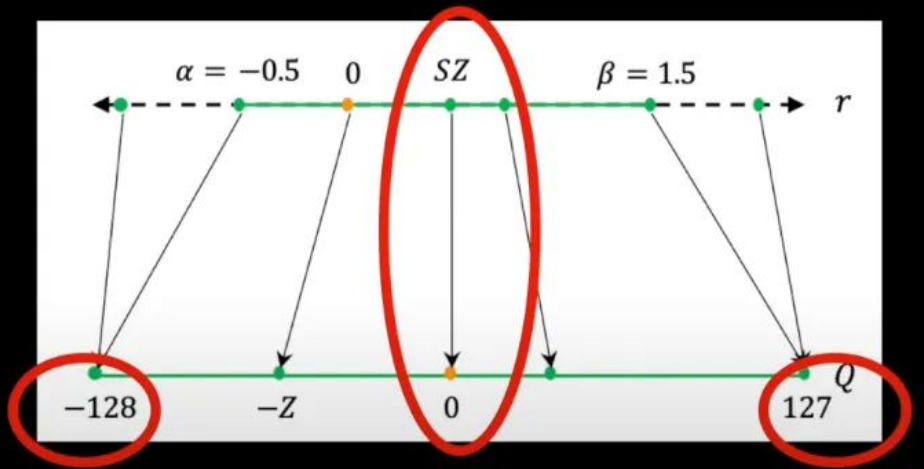
https://huggingface.co/docs/optimum/en/concept_guides/quantization

Quantization

$$Q = round(\frac{r}{S})$$



$$Q = round\frac{r}{S} + Z$$



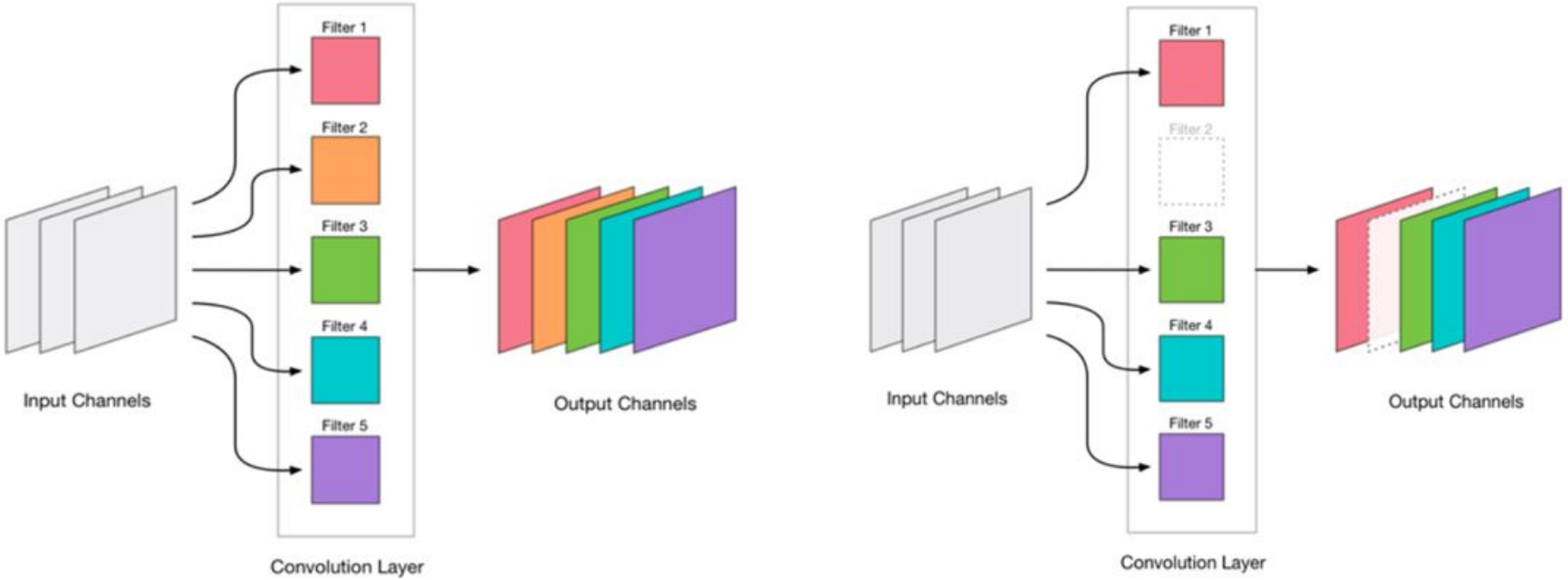
Quantization

Крайне актуально для LLM, так как очень много тензоров и даже forward pass для batch_size=1 не влезает на видеокарту.

Поэтому выпускают квантизованные версии моделей.
Q32, …, Q8…, Q2

Часто после квантизации приходится еще дообучать, иначе качество страдает.

Pruning

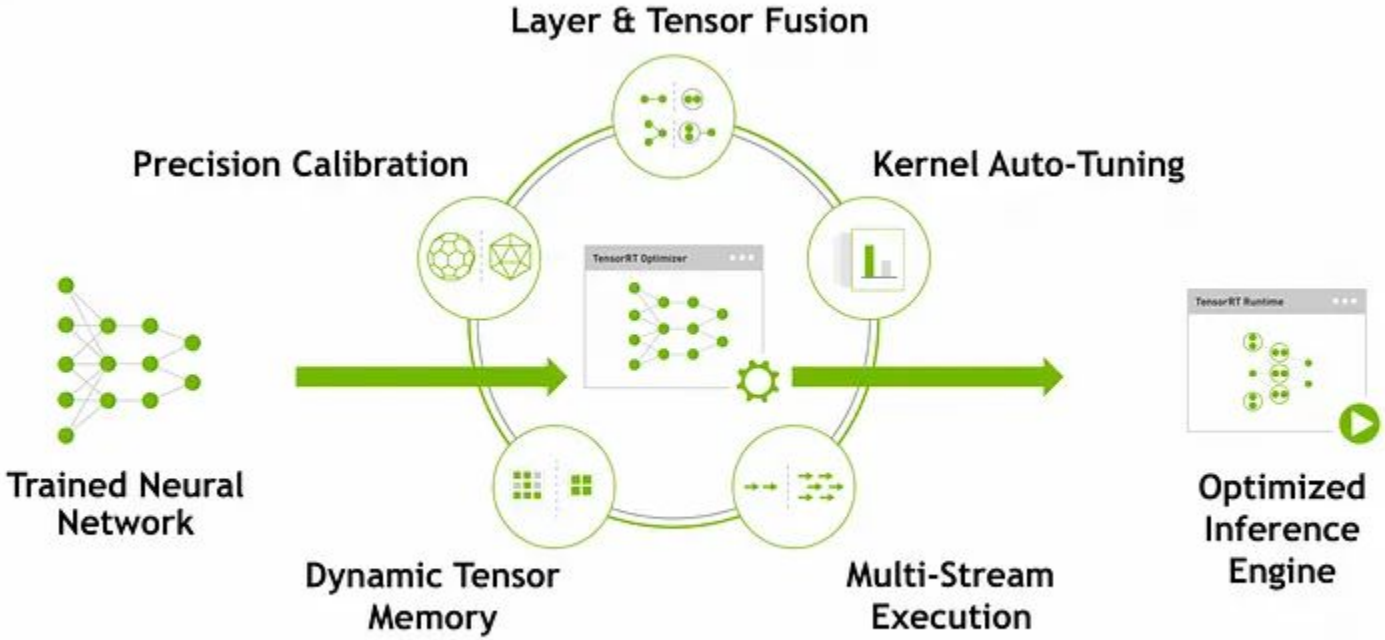

Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Input Channels
Output Channels
Convolution Layer
Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Input Channels
Output Channels
Convolution Layer

Pruning

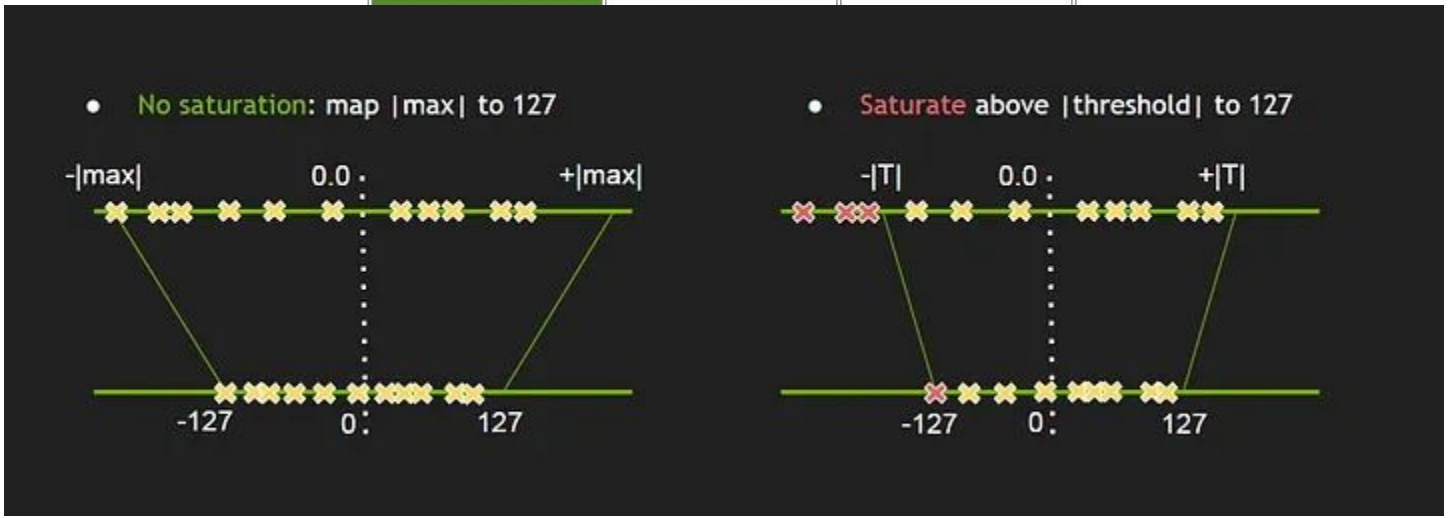
1. Наименьшая L1-мера или low_magnitude_pruning. Идея, говорящая о том, что свертки с малыми значениями весов, вносят малый вклад в итоговое принятие решения
2. Наименьшая L1-мера с учетом среднего и стандартного отклонения. Дополняем оценкой характера распределения.
3. Маскирование сверток и исключение наименее влияющих на итоговую точность. Более точное определение малозначимых свёрток, но весьма затратное по времени и ресурсам.
4. Другие

Pruning

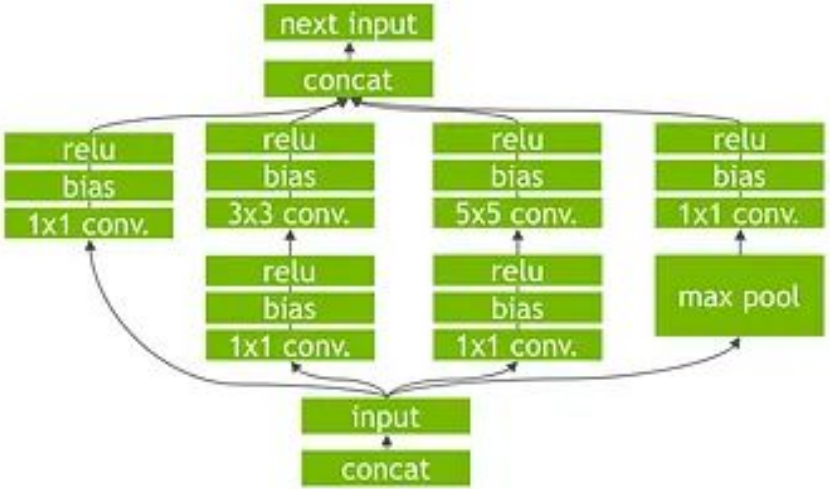
TensorRT


Layer & Tensor Fusion
Precision Calibration
Kernel Auto-Tuning
TensorRT Optimizer
TensorRT Runtime
Trained Neural Network
Dynamic Tensor Memory
Multi-Stream Execution
Optimized Inference Engine

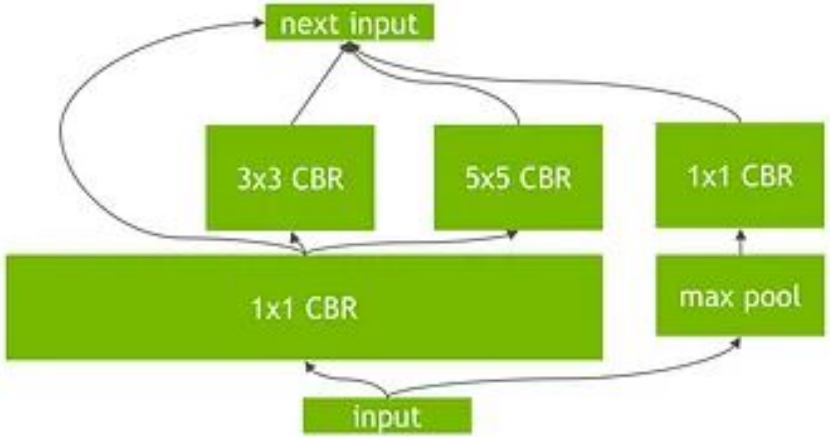
	Dynamic Range	Minimum Positive Value
FP32	-3.4×1038 ~ +3.4×1038	1.4 × 10−45
FP16	65504 ~ +65504	5.96 x 10-8
INT8	-128 ~ +127	1



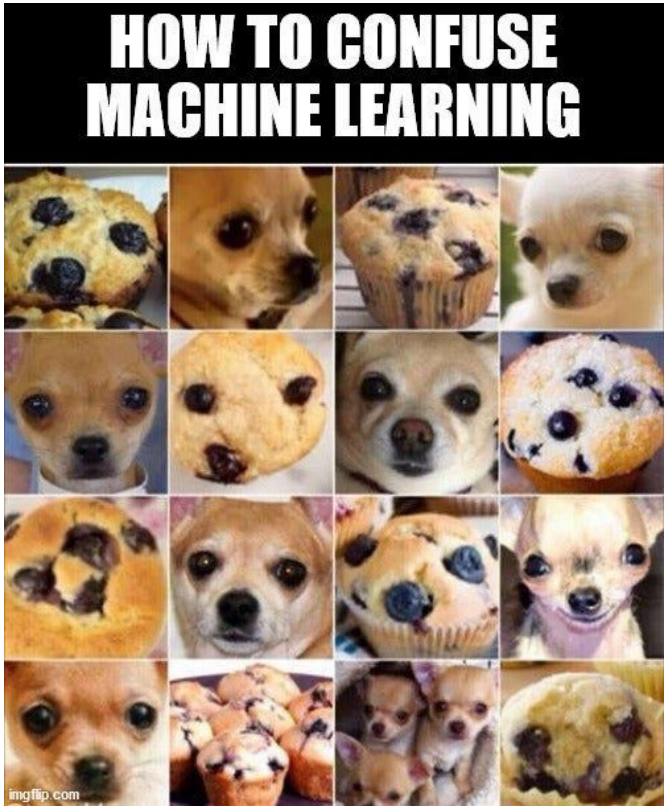
Un-Optimized Network

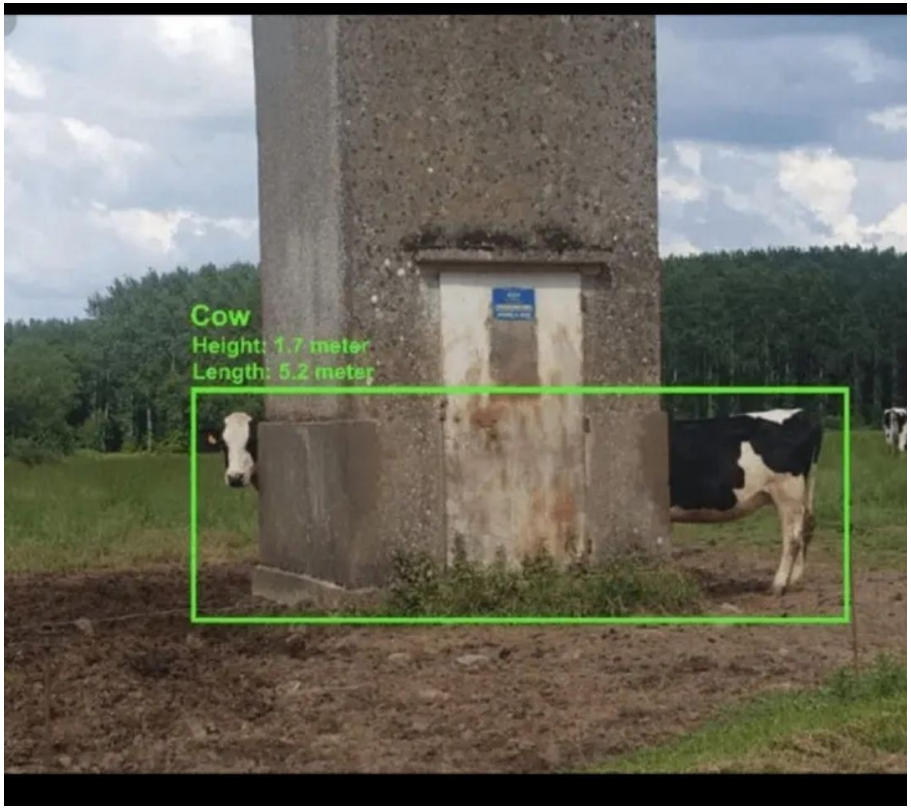


TensorRT Optimized Network



Интерпретируемость





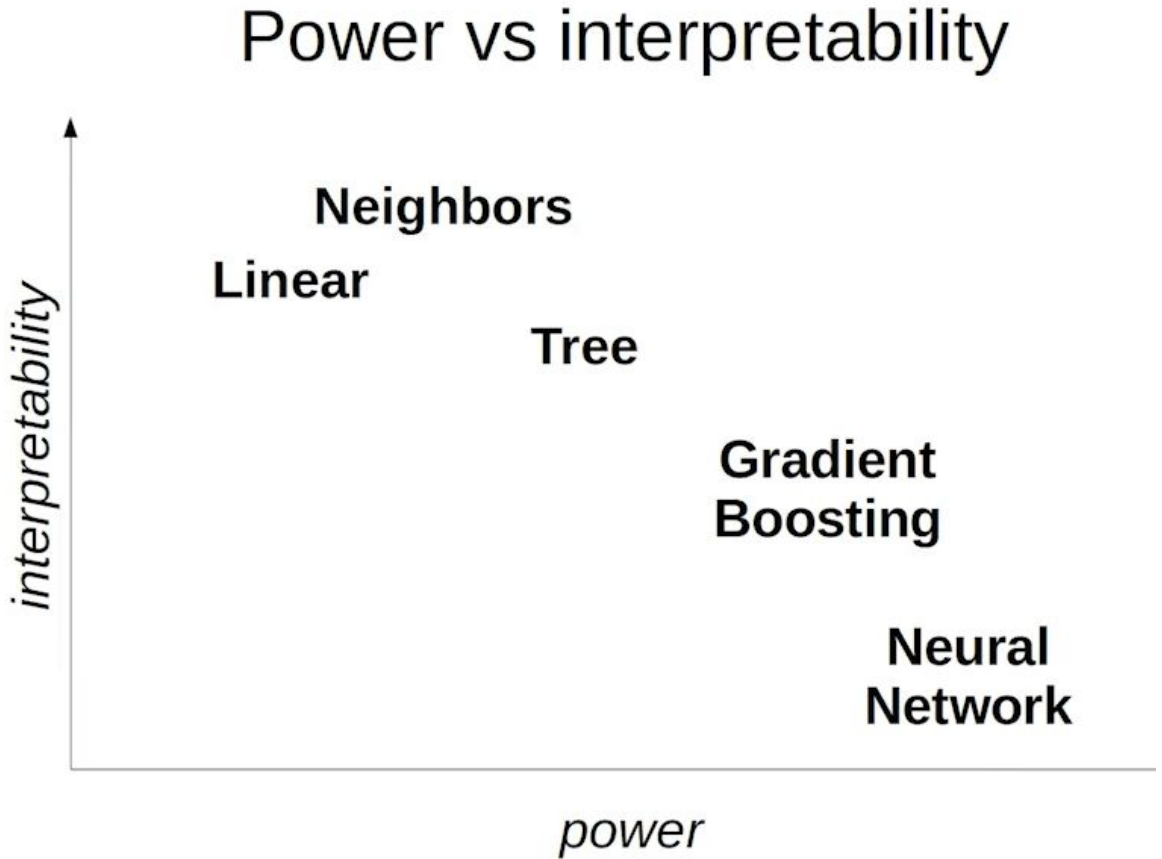
Интерпретируемость

Как объяснить предсказание модели?

Насколько модель уверена в своем предсказании?

Можно ли доверять данным, на которых она обучалась, могли ли они привести к неправильному поведению?

Интерпретируемость



Easy and hard cases

Плохо классифицируют объекты, которые не обладают какой-то характерной текстурой

Texture beats shape as a cue

(a) Texture image

81.4%	**Indian elephant**
10.3%	indri
8.2%	black swan

(b) Content image

71.1%	**tabby cat**
17.3%	grey fox
3.3%	Siamese cat

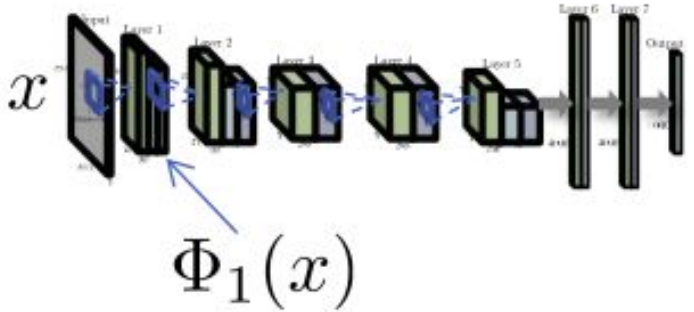
(c) Texture-shape cue conflict

63.9%	**Indian elephant**
26.4%	indri
9.6%	black swan

(this is indri)

Pattern sensitivity

Types of patterns in each layer:



$$\arg\max_{x \in S} \max_{p,q} \Phi_1(x)^t[p,q]$$

Pattern sensitivity

Types of patterns in each layer:

Layer 1

Layer 2

Types of patterns in each layer:

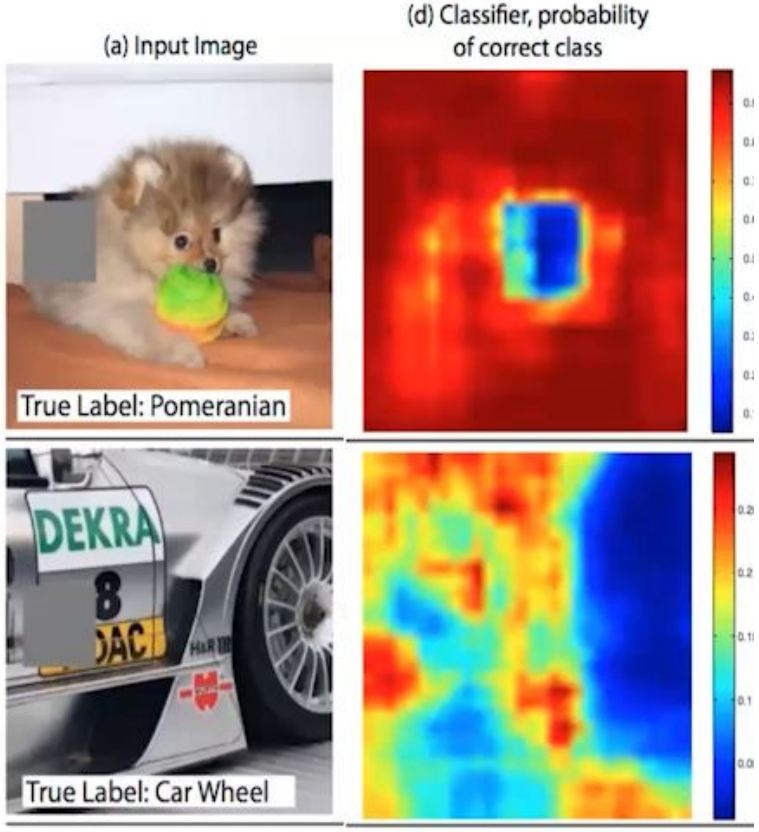
[Zeiler Fergus 14]

Layer 3

Explanation by occlusion

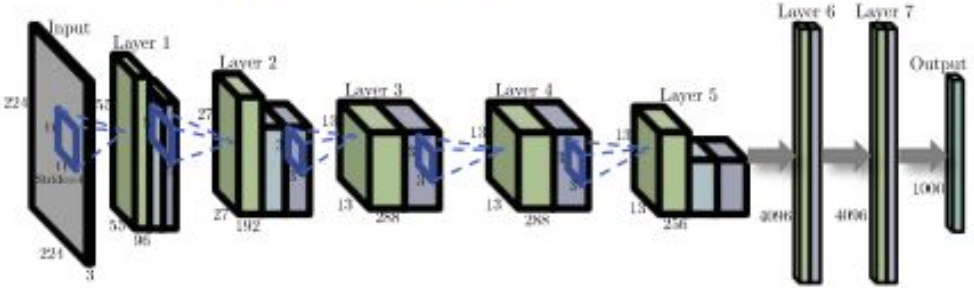
Idea:

- Let's add noise to inputs and see what happens!
- For images: slide a gray square over the image, measure how it affects predictions

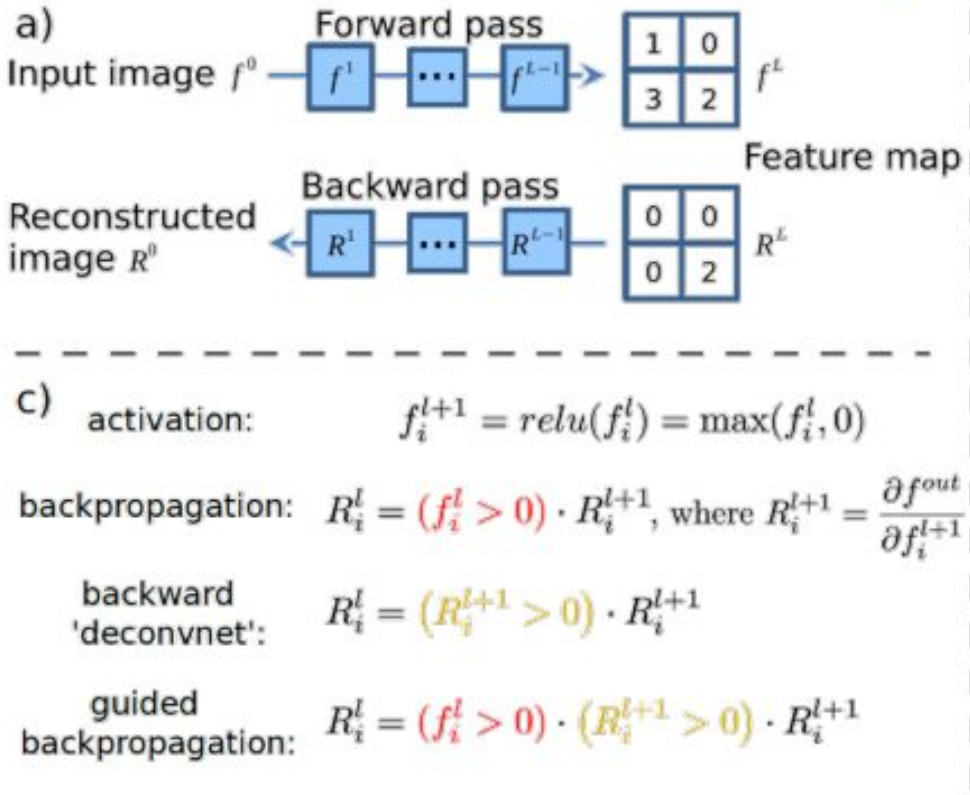


Grounding using gradient

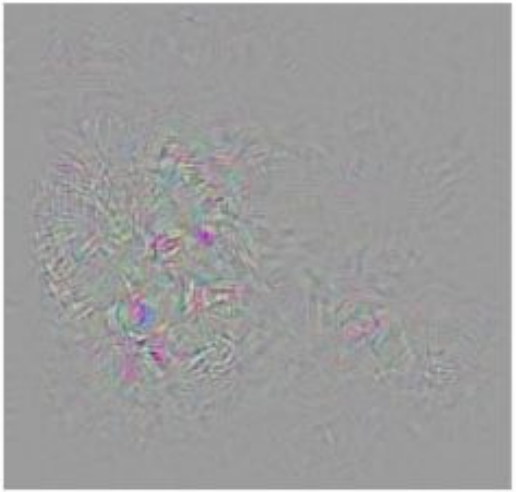
$$\left\| \frac{\partial \Phi_{\text{Last}}[y_0]}{\partial x} \right\|$$



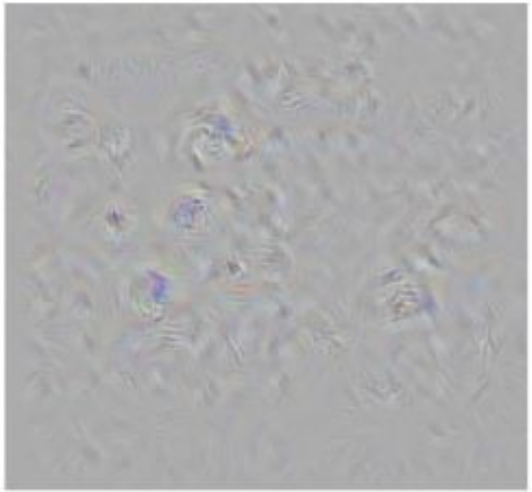
Better grounding using gradient



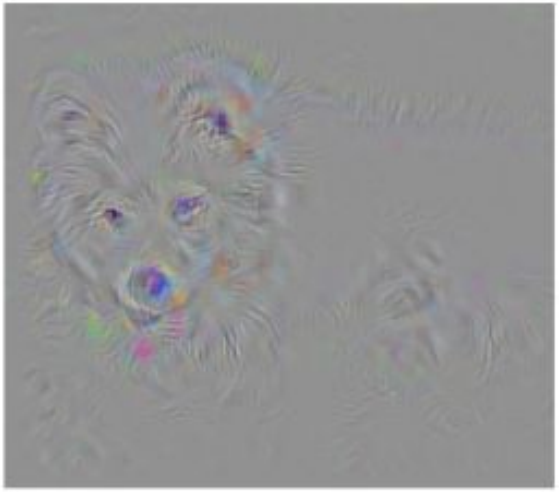
Better grounding using gradient



gradient



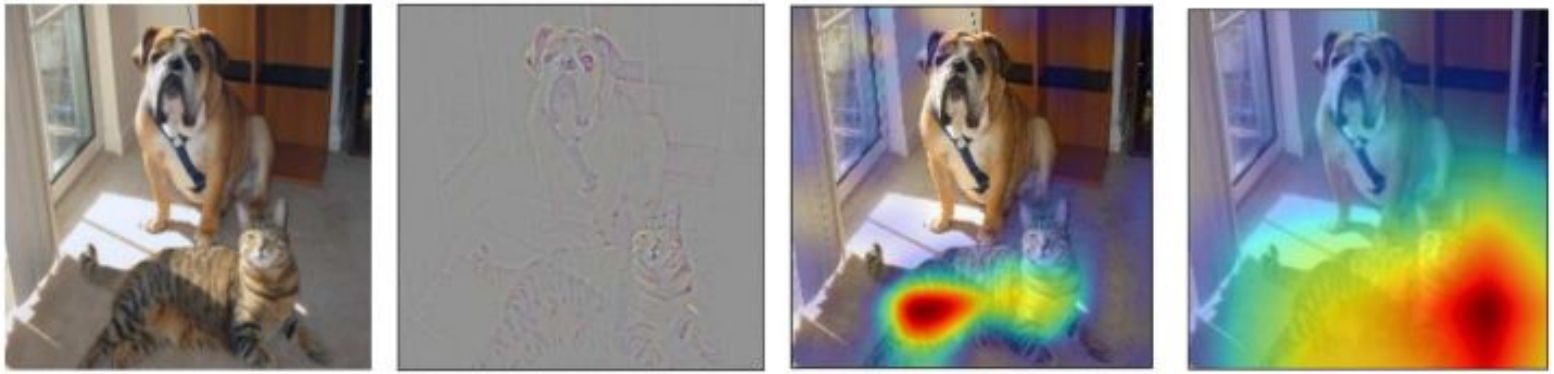
DeConvNet



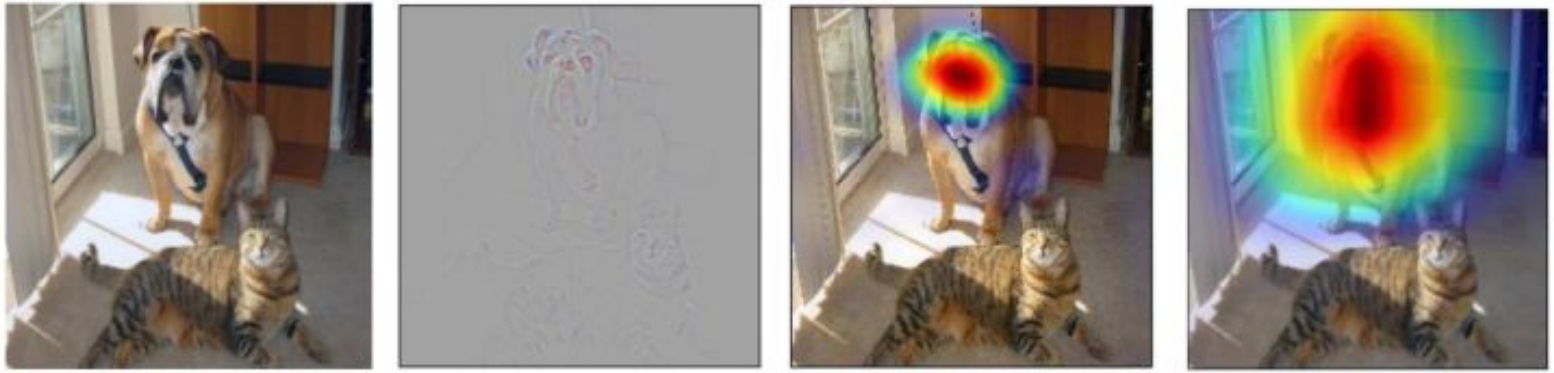
Guided backprop

[Springenberg et al. 2015]

Grad-CAM visualizer [Selvaraju et al. ICCV 17]

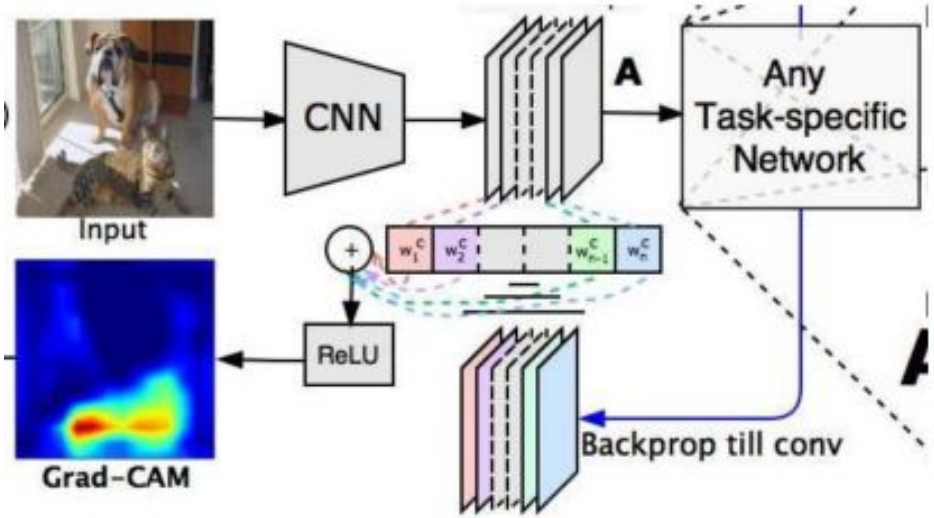


(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (f) ResNet Grad-CAM 'Cat'



(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (l) ResNet Grad-CAM 'Dog'

Grad-CAM visualizer

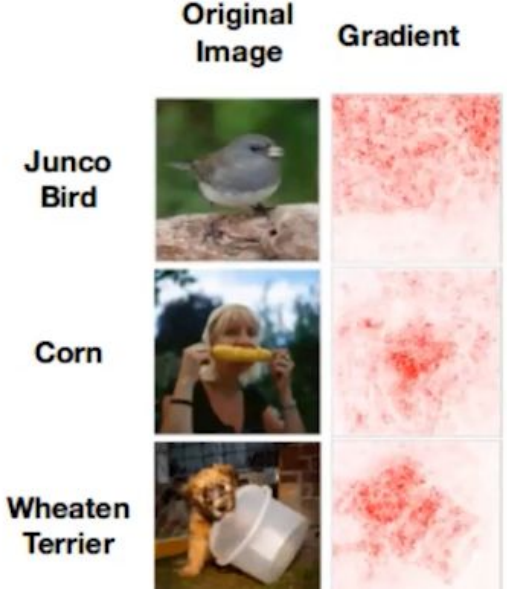


$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$

$$L^c_{\text{Grad-CAM}} = ReLU\underbrace{\left(\sum_k \alpha_k^c A^k\right)}_{\text{linear combination}}$$

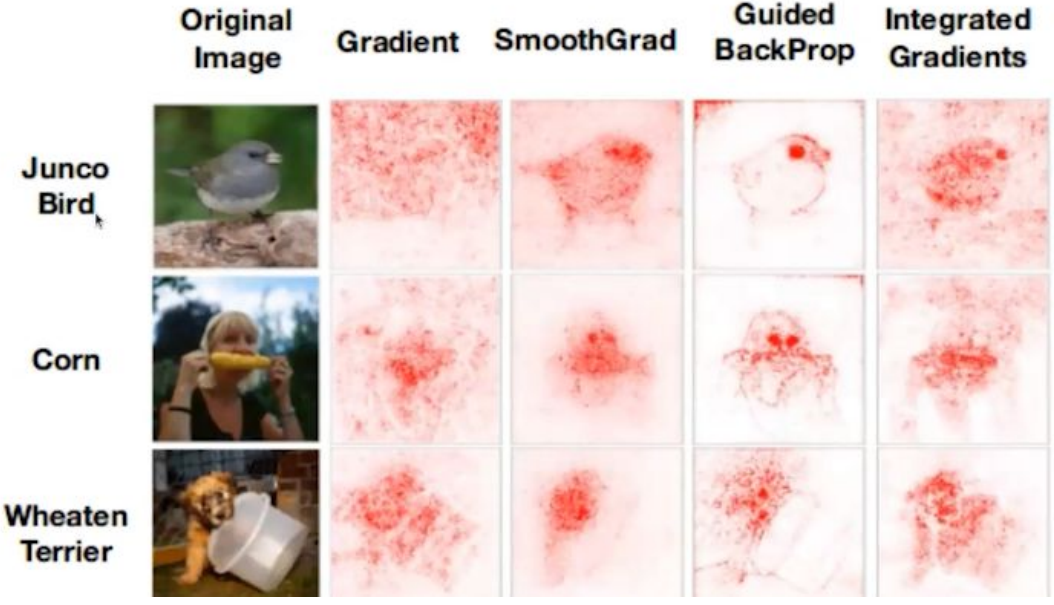
Explanation by gradients

Idea: use gradients! $\nabla_{x_i} model(x) = \frac{\partial model(x)}{\partial x_i}$



Explanation by gradients

Idea: use gradients! $\nabla_{x_i} model(x) = \frac{\partial model(x)}{\partial x_i}$



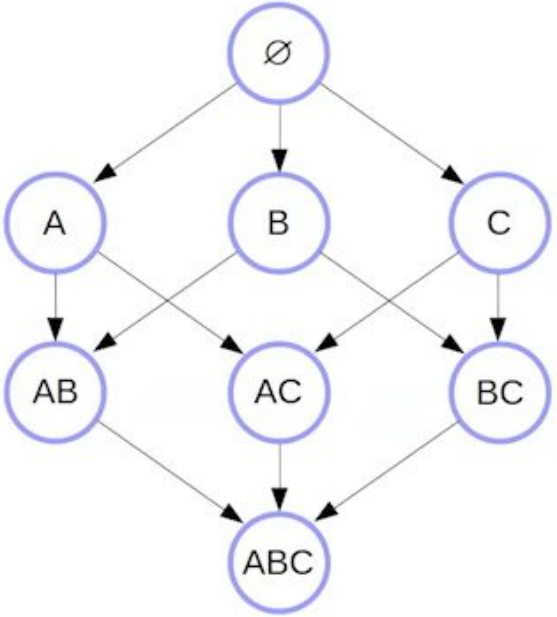
Shapley values explained

Same old table

Who goes	**A**lice	**B**ob	**C**arol	**A** & **B**	**A** & **C**	**B** & **C**	**A**, **B** & **C**
Total price	400	560	720	740	780	980	1000

Shapley(X) = average increase in cost from adding X to a group

Note: average over all ***paths***.



Shapley values explained

Same old table

Who goes	**A**lice	**B**ob	**C**arol	**A** & **B**	**A** & **C**	**B** & **C**	**A**, **B** & **C**
Total price	400	560	720	740	780	980	1000

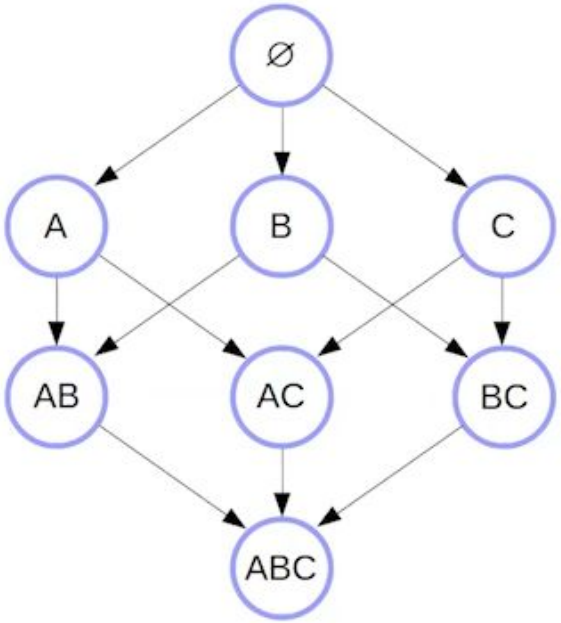
Shapley(X) = average increase in cost from adding X to a group

Note: average over all ***paths***.

$$\text{Shapley(A)} = \frac{2}{6}\cdot 400 \; + \; \ldots$$

$$+ \; \frac{1}{6}\cdot(740-560) \; + \; \frac{1}{6}\cdot(780-720) \; +$$

$$+ \; \frac{2}{6}\cdot(1000-990) \; = \; 180$$



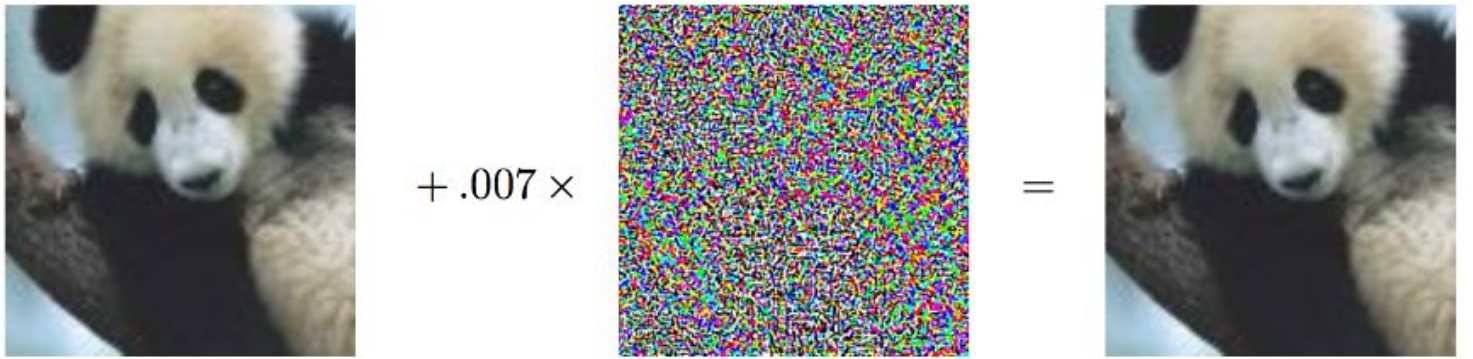
Explanation by game theory

SHAP *= Shapley values for features + clever approximation*
State of the art in after-the-fact model explanation

MOAR:

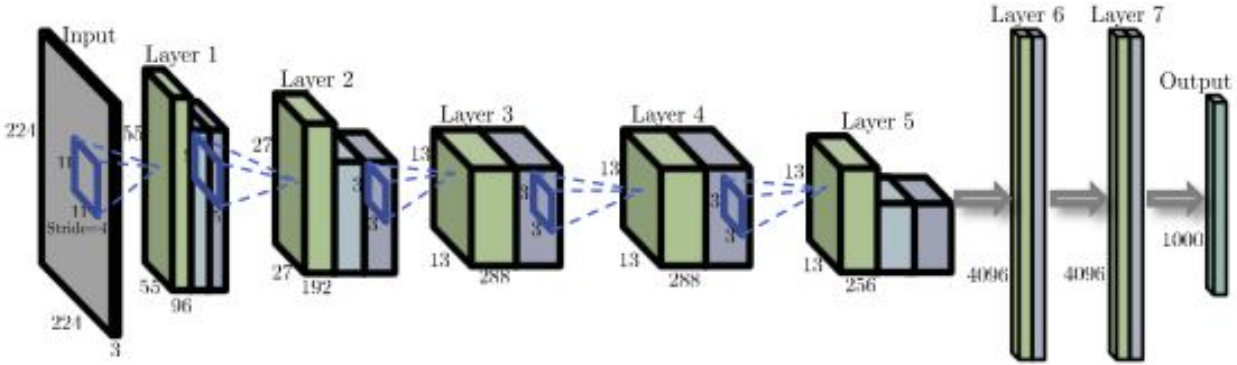
- SHAP original paper: tinyurl.com/shap-paper (NeurIPS'17)
- SHAP explained by paper author: youtu.be/ngOBhhINWb8
- Shapley values in game theory: youtu.be/w9O0fkfMkx0

Adversarial attacks. FGSM, 2015



From the figure, $\mathbf{x}$ is the original input image correctly classified as a “panda”, y is the ground truth label for $\mathbf{x}$, θ represents the model parameters, and $J(\theta, \mathbf{x}, y)$ is the loss that is used to train the network. The attack backpropagates the gradient back to the input data to calculate $\nabla_x J(\theta, \mathbf{x}, y)$. Then, it adjusts the input data by a small step (ϵ or 0.007 in the picture) in the direction (i.e. $sign(\nabla_x J(\theta, \mathbf{x}, y))$) that will maximize the loss. The resulting perturbed image, x', is then *misclassified* by the target network as a “gibbon” when it is still clearly a “panda”.

Generating adversarial perturbations



$$\max_r -\lambda\|r\| + \Phi_{\text{Last}}(x+r)[y']$$

Generating adversarial perturbations