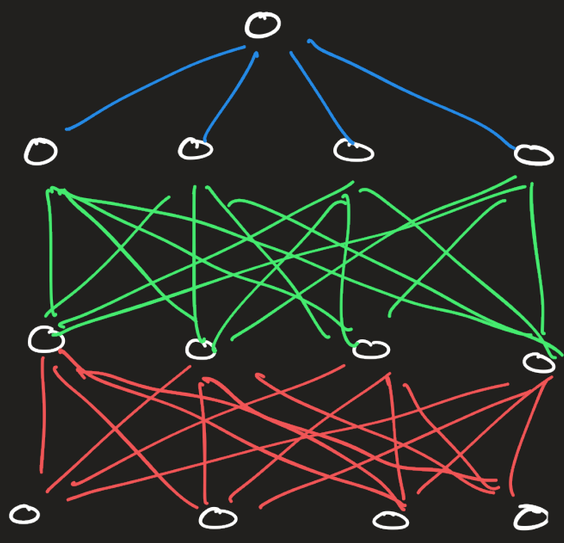
ML and Optimization Lecture 22

- Dropout regularization (model ensembling)
- Inception architecture
- Skip-connections & residual layers
  - briefly mention ResNet
- Introduce RNNs (recurrent neural networks)

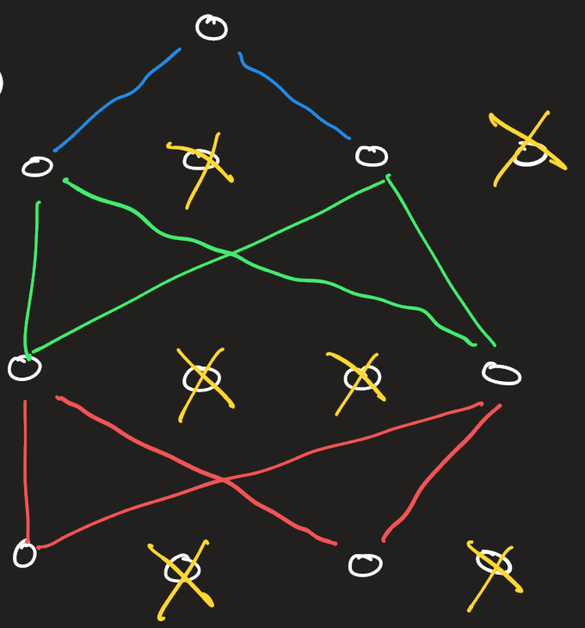
Dropout

- form of regularization introduced to prevent overfitting
- led to a series of followups: DropConnect, etc.

Intuition: to avoid "feature co-adaptation"
(where some neurons in one layer learn brittle, non-generalizing combinations of features in the previous layer)
randomly drop some activations from the previous layer to zero, so neurons have to learn to compute robust features during training. These random dropouts change at each minibatch during training.



Architecture of the NN



At each minibatch:

- randomly sample bernoulli mask for each neuron to be considered dropping
- do forward & backward pass on the resulting subnetwork to update the model parameters

If we add a dropout layer b/w layers $l-1$ and l of our NN, this is equivalent to changing our activation formulas to

$$a^l = w^l o^{l-1} + b^l$$
$$o^l = \sigma(a^l)$$

w/o dropout

$$\Rightarrow \quad a^l = w^l \left[\nu^{l-1} \odot o^{l-1} \right] + b^l$$
$$o^l = \sigma(a^l)$$

dropout, where

$$\nu^{l-1} \in \{0,1\}^{n_{l-1}}$$

and

$$(\nu^{l-1})_i \sim \text{Bern}(p)$$

where p is a hyperparameter and ν^{l-1} is resampled for each minibatch

How to use a network trained using dropout for inference?

Choices:

(1) could use dropout as before (random subnetwork) and get a random prediction

(2) (usually preferred). use the average activation in each layer:

$$\mathbb{E}a_l = \mathbb{E}\left[w^l(v^{l-1} \odot o^{l-1}) + b^l\right]$$

$$= w^l\left(\mathbb{E}[v^{l-1}] \odot o^{l-1}\right) + b^l$$

$$= p\left(w^l o^{l-1}\right) + b^l$$

An interpretation of Dropout as model ensembling

Very useful idea in machine learning: model averaging / ensembling

Fit models $f_{\theta_1}, \dots, f_{\theta_m}$ for the same task and take

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} f_{\theta_i}(x)$$

to be the model average. In practice f has lower error than the component models.

Problem is: training m models costs m times as much as training one

Dropout inexpensively trains the averaged model

$$f(x) = \mathbb{E}_{p(\theta)} f_\theta(x)$$

where $p(\theta)$ is the distribution over the masks of the architecture.

Dropout computes subgradient for f at each iteration of minibatch SGD:

if f_i is a sample from the dropout architecture distribution, then

$$\mathbb{E} f_i = f(x)$$

and

$$\mathbb{E} \nabla_w f_i = \nabla_w \mathbb{E} f_i = \nabla_w f(x)$$

so $\nabla_w f_i$ is a stochastic subgrad for $f(x)$

Inception architecture SOTA performance ↑ on ILSVRC 2014

22 layer deep initially

Ideas: max pooling loses spatial details

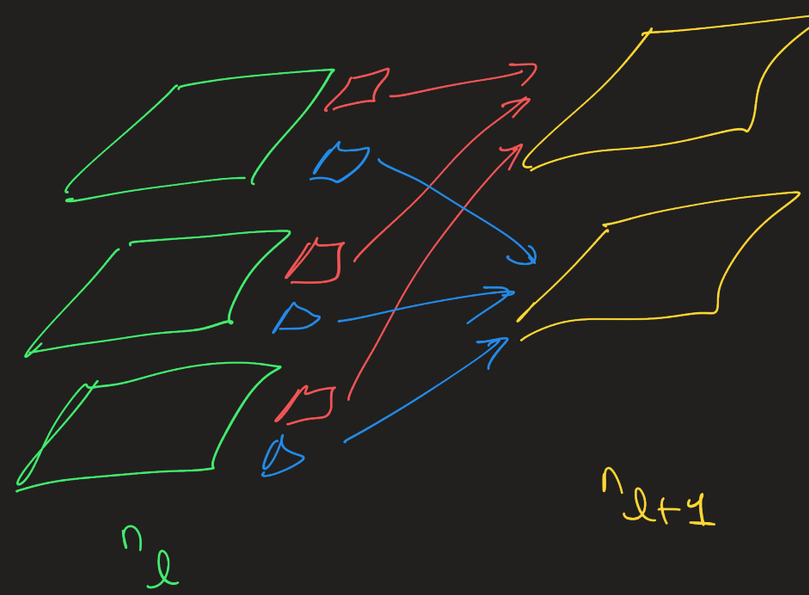
CNNs as we've seen have issues w/ detecting the same features at different scales

more channels you have, the more memory you use in training & slower it is.

Trend before inception: increase # layers & # filters per layer)

use dropout to prevent overfitting

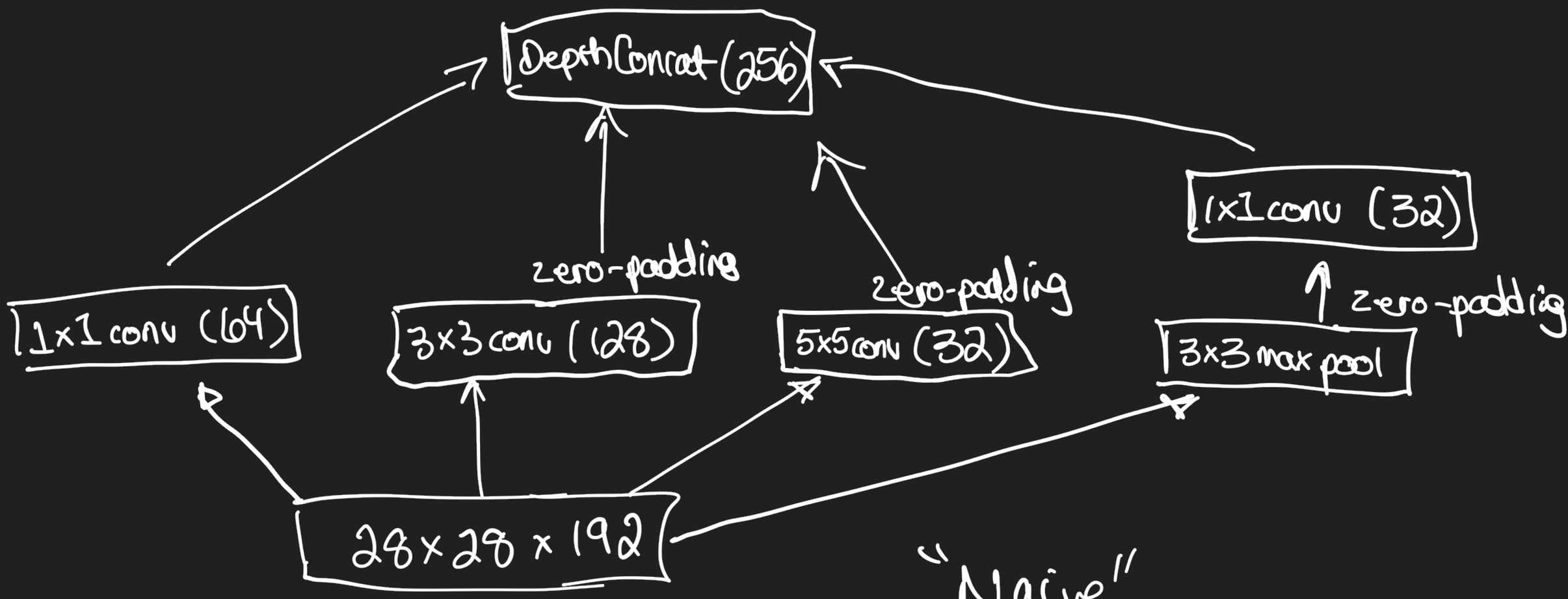
E.g. if have n_l channels on layer l then there are $n_l n_{l+1} k_1 k_2 + n_{l+1}$ parameters connecting layer l to layer $l+1$



Consequence

If increase # of output channels on each layer by c, # parameters increase by factor of $O(c^2)$

GoogLeNet (22 layer) Inception V1



"Naive" Inception Block

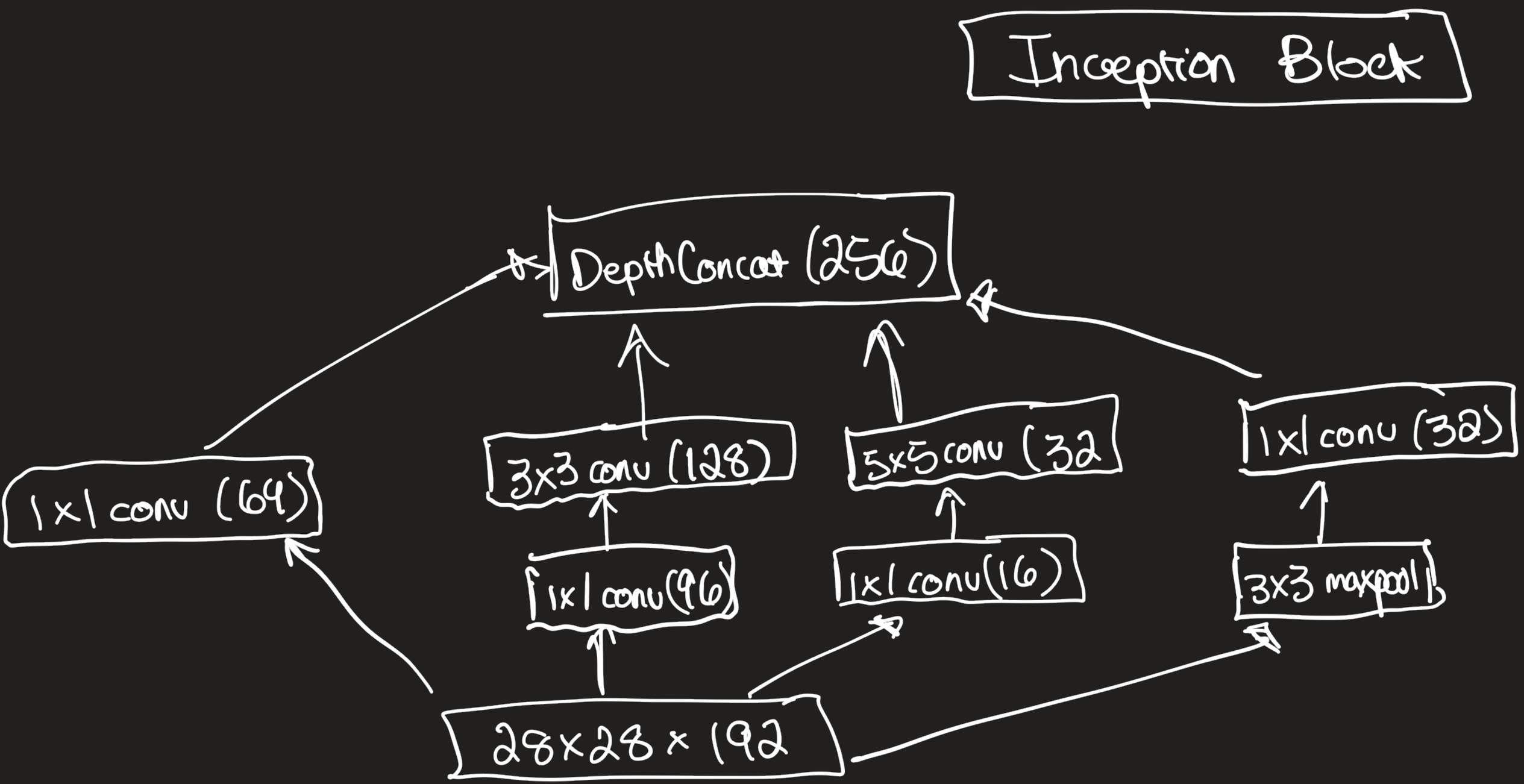
Downside: lots of parameters

E.g. for 3x3 conv block, we have

$$192 \times 128 \times 3 \times 3 + 128 = 221,312 \text{ parameters}$$

Issue: increased # params

Soln: use dimensionality reduction via 1x1 convolutions



With the 1×1 convs for dim reduction, the # of parameters for the 3×3 conv features now becomes

$$\underbrace{(192 \times 96 \times 1 \times 1 + 96)}_{\text{for the } 1\times1 \text{ conv}} + \underbrace{(96 \times 128 \times 3 \times 3 + 128)}_{\text{for the } 3\times3 \text{ conv}}$$

$$= 129,248 \text{ parameters}$$

significantly less

Original Inception (v1) architecture

CNN → Inception Block ×2 → 3×3 max pool
                         ↳ → aux classification loss

→ Inception Block ×5 → 3×3 max pool
                     ↳ → aux classification loss

→ Inception Block ×2 → AvgPool → Dropout → Linear

→ softmax → classification loss

Train this architecture to minimize the weighted sum of the three losses