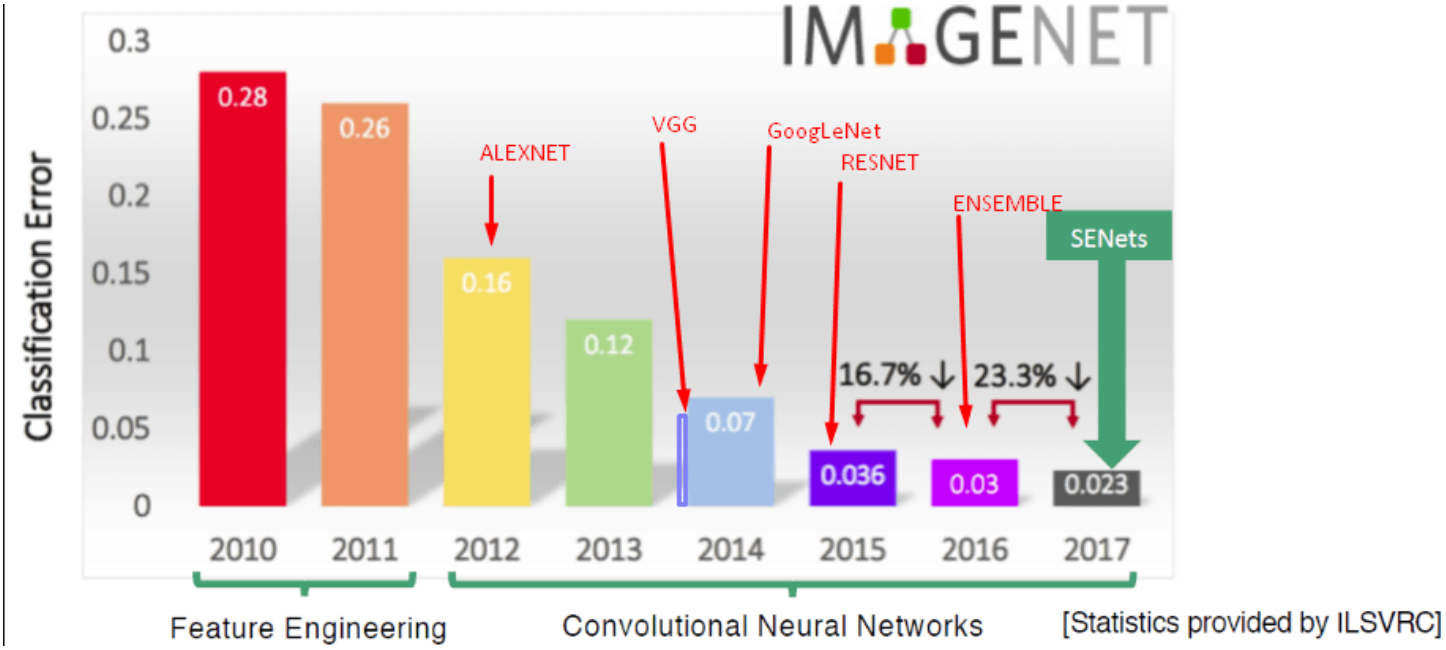
S8

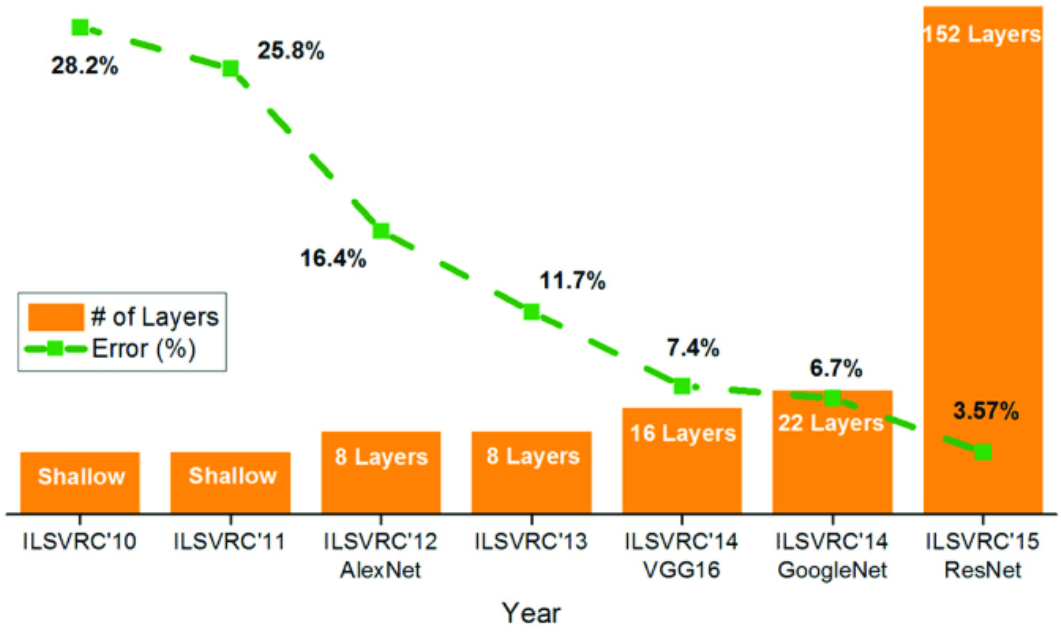
Due No Due Date **Points** None **Available** after Mar 4 at 6:30am

.

RECEPTIVE FIELDS AND NETWORK ARCHITECTURES



There is a general trend in going deeper as years are passing by.



But when we are going deeper, we are not using lesser MaxPoolings. This means our final Receptive Fields are sky-rocketing!

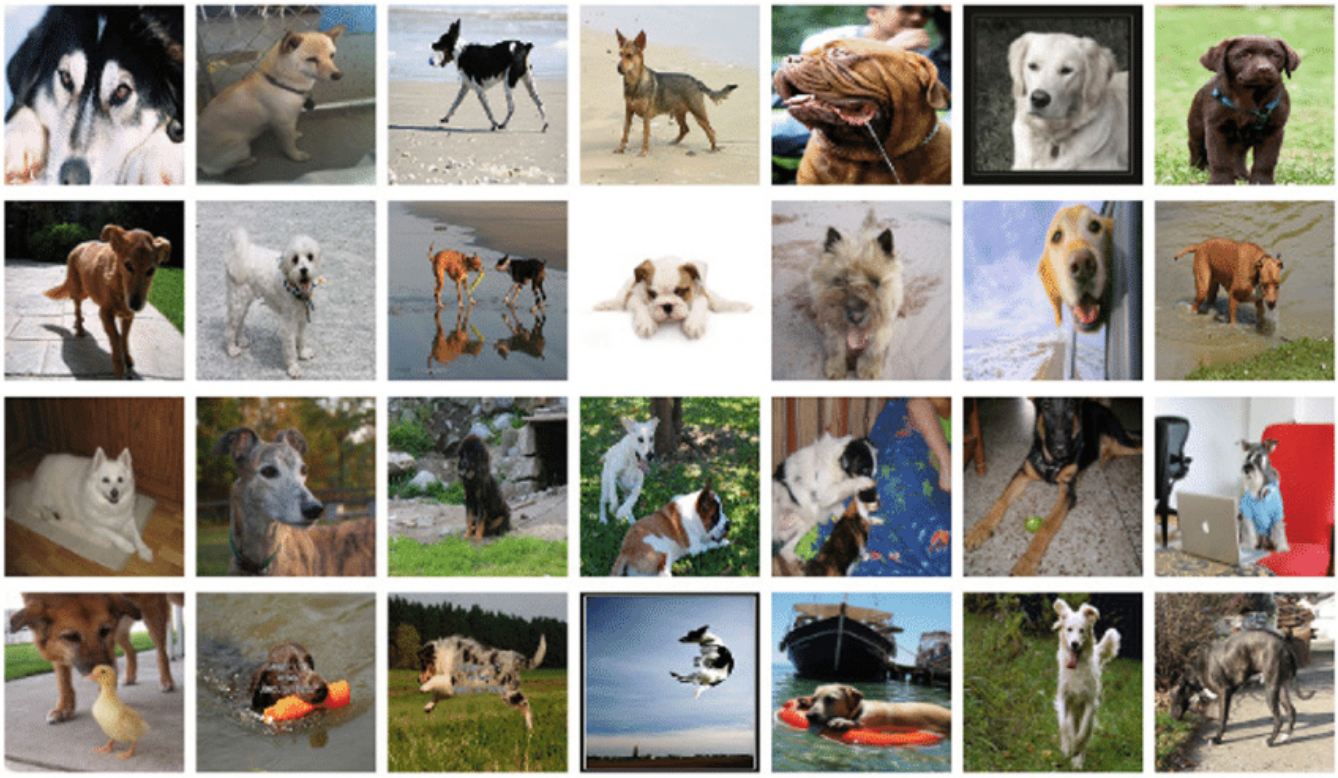
RECEPTIVE FIELDS MORE THAN IMAGE SIZE

Let's look at some questions:

What kind of receptive fields do we actually need?

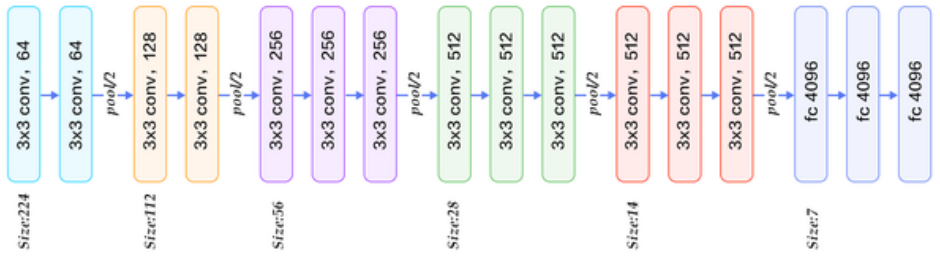
This is linked to the size of the objects:

And images might not even be able to cover the whole object



Result: We need multiple receptive fields!

VGG ARCHITECTURE



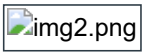
VGG was an audacious attempt in going deeper. BN wasn't invented, so they couldn't go deeper, but deep enough to be 2nd in the ISLRVC competition.

But, VGG has 1 receptive field, not many.

How do we get many receptive fields?

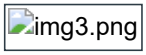
This is where the "group" concept we covered in the last session comes into play.

Let's look at **Inception Block**



Isn't it wonderful how this simple architecture allows us to target many receptive fields!

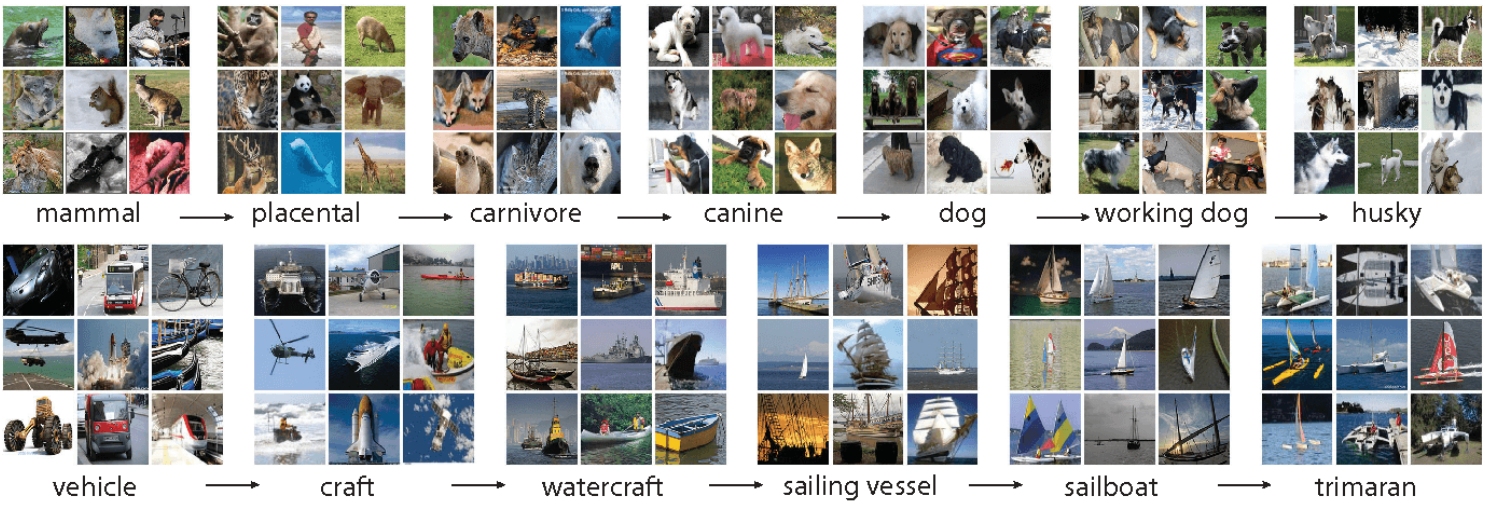
Now let's look at **ResNet Block**



ResNet is simpler and efficient!

But what happens when we increase our Receptive field, how do they "look" or what are they looking for?

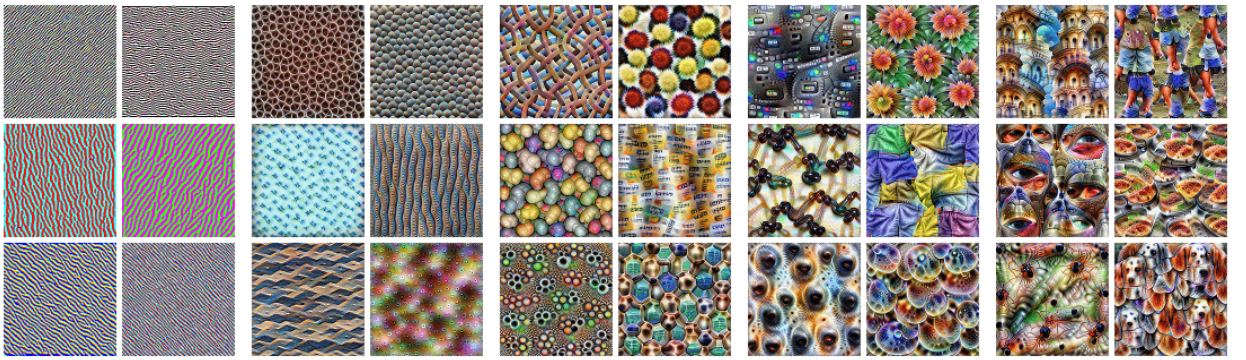
Well, this is how data looks like:



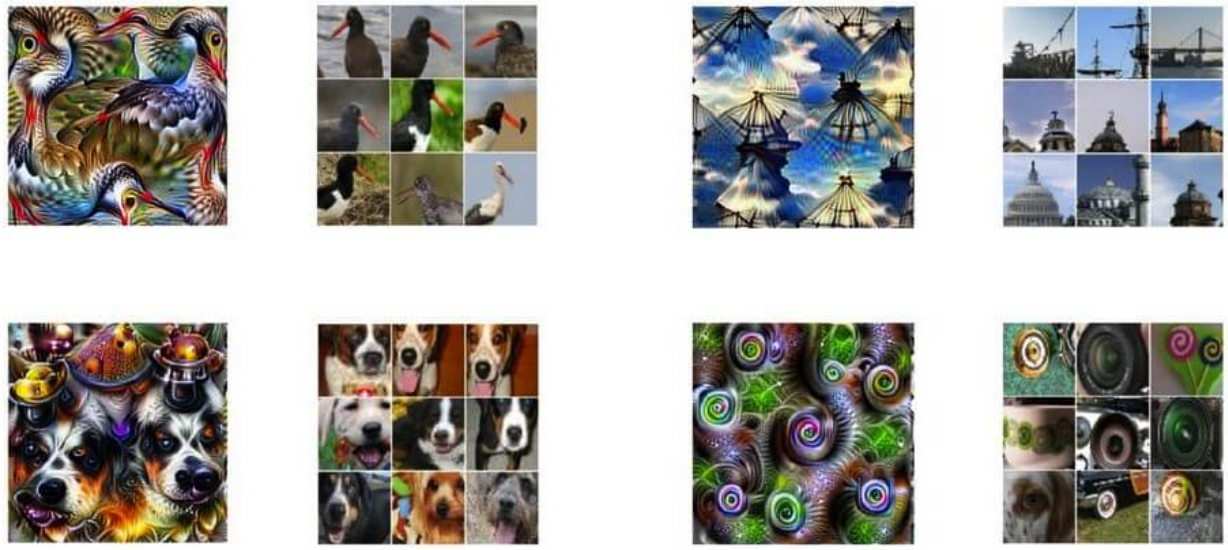
Varied in size, different but sometimes semantic backgrounds.

When we target higher receptive fields our target becomes:

Edges >> Textures >> Patterns >> Parts >> Objects >> Object Templates

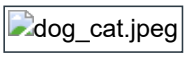


Object templates capture same object in different sizes and with different backgrounds!



But it is even more interesting to see what happens on the micro level at each layer!

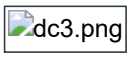
Let's start with this cute image:



After a few Conv layers

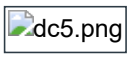
After a few more Conv layers

After even move Conv layers



After a few more Conv layers, but now we are getting closer to SoftMax

And closer to SoftMax



INCEPTION

Inception V1

- **Salient parts** in the image can have an extremely **large variation** in size. For instance, an image with a dog can be either of the following, as shown below. The area occupied by the dog is different in each image.
- Because of this huge variation in the location of the information, choosing the **right kernel size** for the convolution operation becomes tough. A **larger kernel** is preferred for information that is distributed more **globally**, and a **smaller kernel** is preferred for information that is distributed more **locally.**
- Very deep networks are prone to overfitting. It is also hard to pass gradient updates through the entire network
- Naively stacking large convolution operations is **computationally expensive**.

Inception V1 solves this through:

- with multiple sized filters operating on the same level
- going wider to capture different GRF contexts
- not using addition, but concatenation to make sure GRF links are maintained till the SoftMax layer

But this was expensive (think of the next 3x3s, 5x5s which would now need a large number of channels to properly convolve).

To make it cheaper, the authors **limit** the number of **input channels** by adding an **extra 1x1 convolution** before the 3x3 and 5x5 convolutions.

- GoogLeNet has 9 such inception modules stacked linearly.
- It is 22 layers deep (27, including the pooling layers).
- It uses global average pooling at the end of the last inception module.

- it is a pretty **deep classifier**. As with any very deep network, it is subject to the **vanishing gradient problem**.
- To prevent the **middle part** of the network from "**dying out**", the authors introduced **two auxiliary classifiers** (The purple boxes in the image). They essentially applied softmax to the outputs of two of the inception modules and computed an **auxiliary loss** over the same labels. The **total loss function** is a **weighted sum** of the **auxiliary loss** and the **real loss**. The weight value used in the paper was 0.3 for each auxiliary loss.
- auxiliary loss is purely used for training purposes and is ignored during inference.

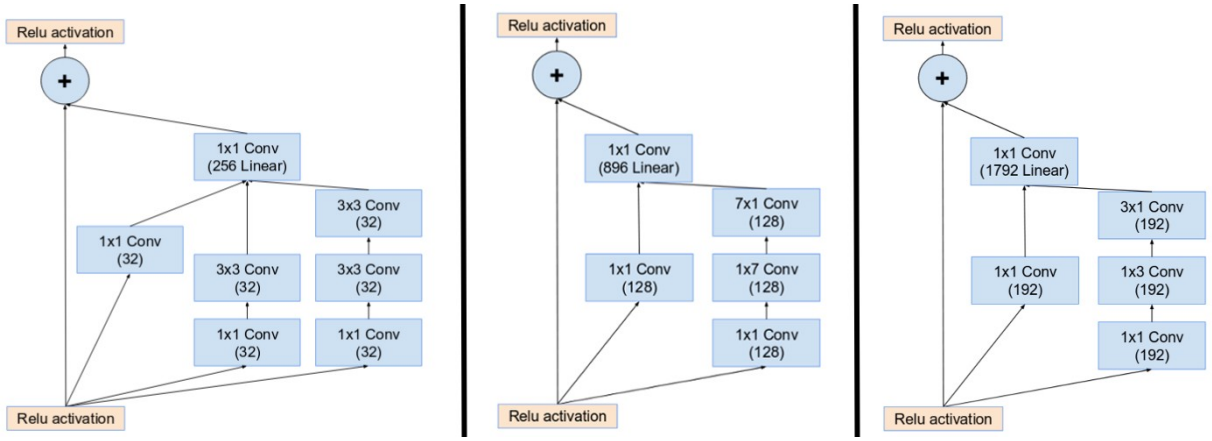

```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

INCEPTION V2 & V3

Inception v2 and **Inception v3** were presented in the **same** **paper** (https://arxiv.org/pdf/1512.00567v3.pdf) . The authors proposed a number of upgrades which increased the accuracy and reduced the computational complexity. Inception v2 explores the following:

- Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
- **Factorize 5x5** convolution **to two 3x3** convolution operations to improve computational speed. 5x5 convolution is **2.78 times more expensive** than a 3x3 convolution. So stacking two 3x3 convolutions in fact leads to a boost in performance. This is illustrated in the below image

INSANITY



Inception is not designed by Humans any more and Google has been using RL agents to design even weirder but powerful Inception Models. Inception has since fallen back in its acceptance and usability

At TSAI too, we will not be working with Inception, however, you're allowed to do any assignment (unless asked specifically to work with ResNet) with Inception.

RESNET

Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper. While AlexNet had only 5 convolutional layers, the VGG network and GoogleNet (also codenamed Inception_v1) had 19 and 22 layers respectively.

However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

BEFORE RESNET

AFTER RESNET

The authors of the ResNet paper argue, that the stacking layers shouldn't degrade the network performance, because we could simply stack identity mappings (layers that don't do anything) upon the current network, and the resulting architecture would perform the same.

This indicates that the deeper model should not produce a training error higher than its shallower counterparts.

If you think about it, ResNet can be considered as an ensemble of Smaller networks!

WHERE IS THE RESIDUE?

But why call it residual? Where is the residue? It's time we let the mathematicians within us to come to the surface. Let us consider a neural network block, whose input is *x* and we would like to learn the true distribution *H(x)*. Let us denote the difference (or the residual) between this as

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

Rearranging it, we get,

$$H(x) = F(x) + x$$

THE CORE IDEA

The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers.

A residual block is displayed as the following:

The residual unit obtains ***F(x)*** by processing ***x*** with two weight layers. Then it adds ***x*** to ***F(x)*** to obtain ***H(x)***.

BACKPROPAGATION IN RESNET

BUT

If **λ ~ 0** (because of multiple positive ReLUs or other activation functions) the left term will be exponentially large, and **gradient exploding** problem occurs. As we should remember, when the gradient exploded, the **loss cannot be converged**.

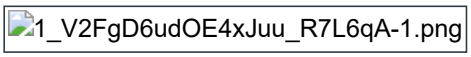
If **λ~0** (because of multiple ReLUs or other activation functions), the left term will be exponentially small, and the **gradient vanishing** problem occurs. We cannot update the gradient with a large value, **the loss stays at the plateau and ends up converged with a large loss**.

Thus, that’s why we need to keep clean for the shortcut connection path from input to output without any Conv layers, BN and ReLU.

RESNET v2

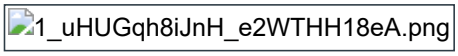
We ***add*** F(x) and x, and then perform ReLU.

This was the original architecture. In an update, it was proposed to move ReLU out.



Here's why!

the input signal x_l is still kept alive! And



the gradients can *(almost)* never be zero!

The identity matrix summation speeds up the training process and improves gradient flow since the skip connections are taken from previous conv operations. Thus the backpropagation can effectively transfer error corrections to earlier layers much easier. This addresses the vanishing gradient problem.

RESNET ARCHITECTURE

For ResNet, there are **two kinds of residual connections:**

1. the identity shortcuts (x) can be directly added with the input and output are of the same dimensions
2. when the dimensions change (input is larger than residual output, but we need to add them). The default way of solving this is to use a 1x1 Conv with a stride of 2. **Yes, half of the pixels will be ignored.**

François Chollet, Author of Keras .

He says: They do this because it's what you should be doing. Residual connections with different shapes should be handled via a learned linear transformation between the two tensors, e.g. a 1x1 convolution with appropriate strides and border_mode, or for Dense layers, just matrix multiplication.

RESNET MODEL WALKTHROUGH

ResNet34

R18 - Regular - 2, 2, 2, 2
R34 - Regular - 3, 4, 6, 3

We'll go through ResNet34.

- It has 1 convolution layer of 7x7 sized kernel (64), with a stride of 2
- It is followed by MaxPooling. In fact, ResNet has only 1 MaxPooling operation!
- It is followed by 4 ResNet blocks (config: 3, 4, 6, 3)
- The channels are constant in each block (64, 128, 256, 512 respectively). Each block has only 3x3 kernels.
- The channel size is constant in each block
- Except for the first block, each block starts with a 3x3 kernel of stride 2 (this handles MaxPooling)
- The dotted lines are 1x1 convs with stride 2

SideNote:

The accuracy of convolutional networks evaluated on ImageNet is **vastly underestimated**. We find that when the mistakes of the model as assessed by human subjects and considered correct when four out of five humans agree with the model's prediction, the **top-1 error of a ResNet-101**trained on ImageNet... decreases from **22.69% to 9.47%**. Similarly, the top-5 error decreases from **6.44% to 1.94%**. (This is true for other models as well). **Ref (https://eccv2018.org/openaccess/content_ECCV_2018/papers/Pierre_Stock_ConvNets_and_ImageNet_ECCV_2018_paper.pdf)**

ResNet Bottleneck Blocks

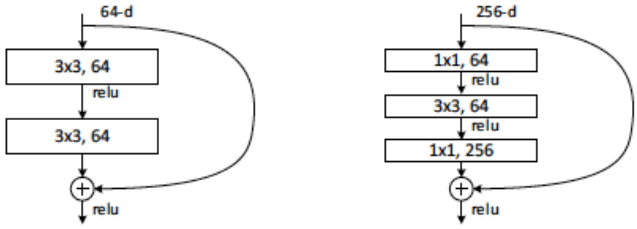


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

- deeper non-bottleneck ResNets also gain accuracy from increasing depth, but are not as economical as the bottleneck ResNets. So the usage of bottleneck designs is mainly due to practical considerations.

resnet.png

R18 - Regular - 2, 2, 2, 2
R34 - Regular - 3, 4, 6, 3
R50 - Bottleneck - 3, 4, 6, 3
R101 - Bottleneck - 3, 4, 23, 3
R152 - Bottleneck - 3, 8, 36, 3

ResNet V3 or ResNeXt

Aggregated Transformations

Screenshot_1-3.png

We'll limit ourselves to ResNet V1 or V2 .

Assignment:

1. Go through this repository: **https://github.com/kuangliu/pytorch-cifar** (https://github.com/kuangliu/pytorch-cifar)
2. Extract the ResNet18 model from this repository and add it to your API/repo.

3. Use your data loader, model loading, train, and test code to train ResNet18 on Cifar10
4. Your Target is 85% accuracy. No limit on the number of epochs. Use default ResNet18 code (so params are fixed).
5. Once done finish S8-Assignment-Solution.

These are the questions in S8-A-S

1. Share the link to your GitHub S8 code. Please make sure it is public. If your code is not modular or structured into different functional files, you will get 0 for the whole submission.
2. What is the final accuracy of your model
3. Paste your training or epoch logs here

Session Video - Wednesday