QS STARS RATED FOR EXCELLENCE 2012

Fpt University

TRƯỜNG ĐẠI HỌC FPT

Residual Networks with Application to Image Colorization

VUONG TUAN QUANG

DO TRUNG NGHIA

SUPERVISOR: DR. VU KHAC KY

BACHELOR OF COMPUTER SCIENCE

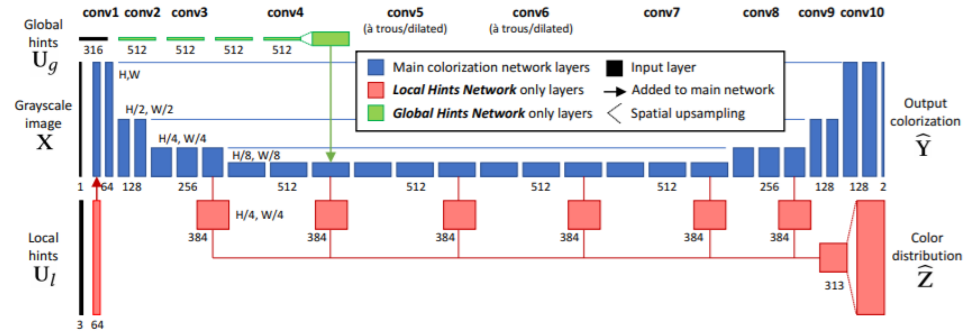
FPT UNIVERSITY - HOA LAC CAMPUS

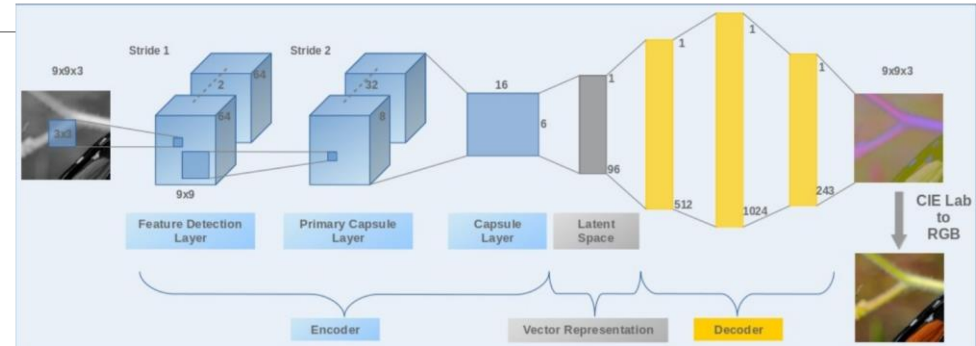
Contents

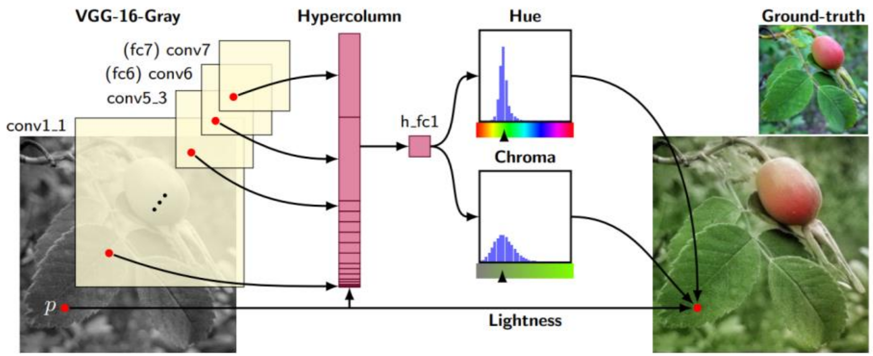
Introduction

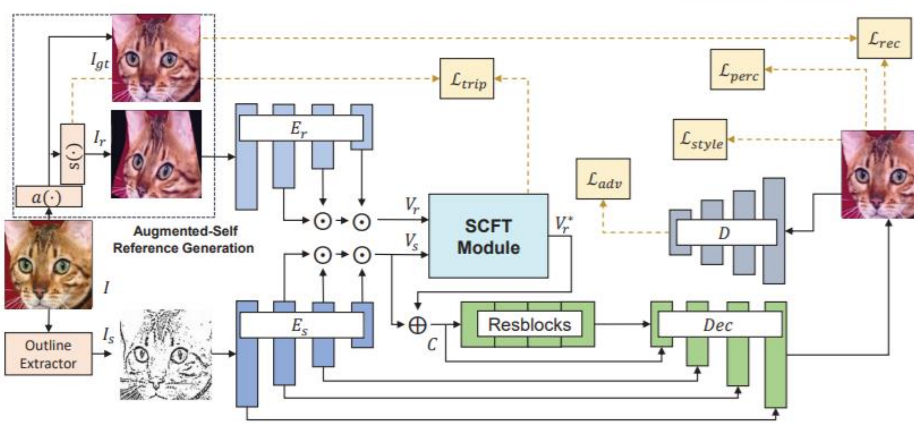
- RELATED WORKS

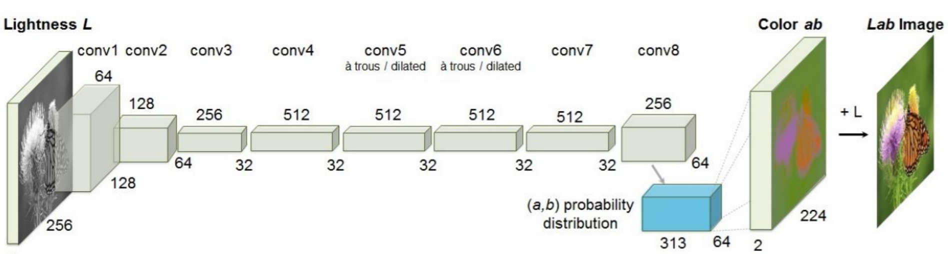
Introduction
Related Works

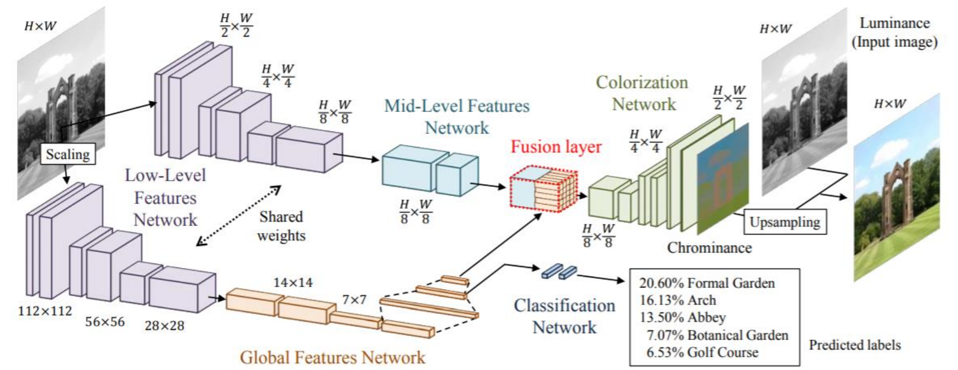


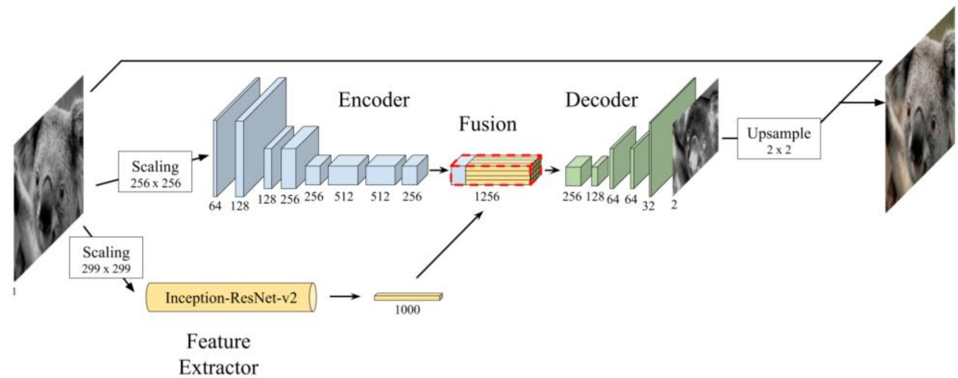


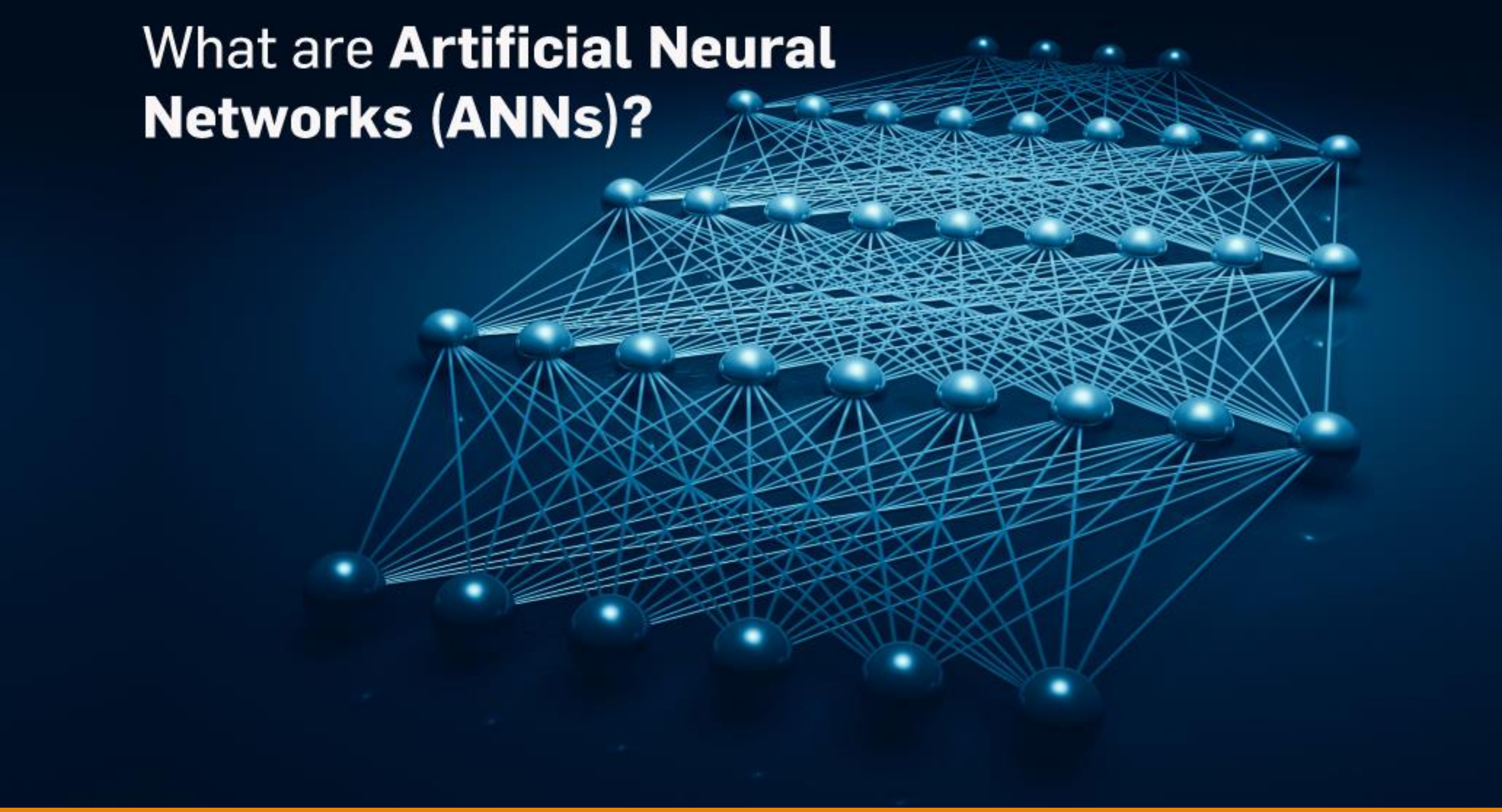








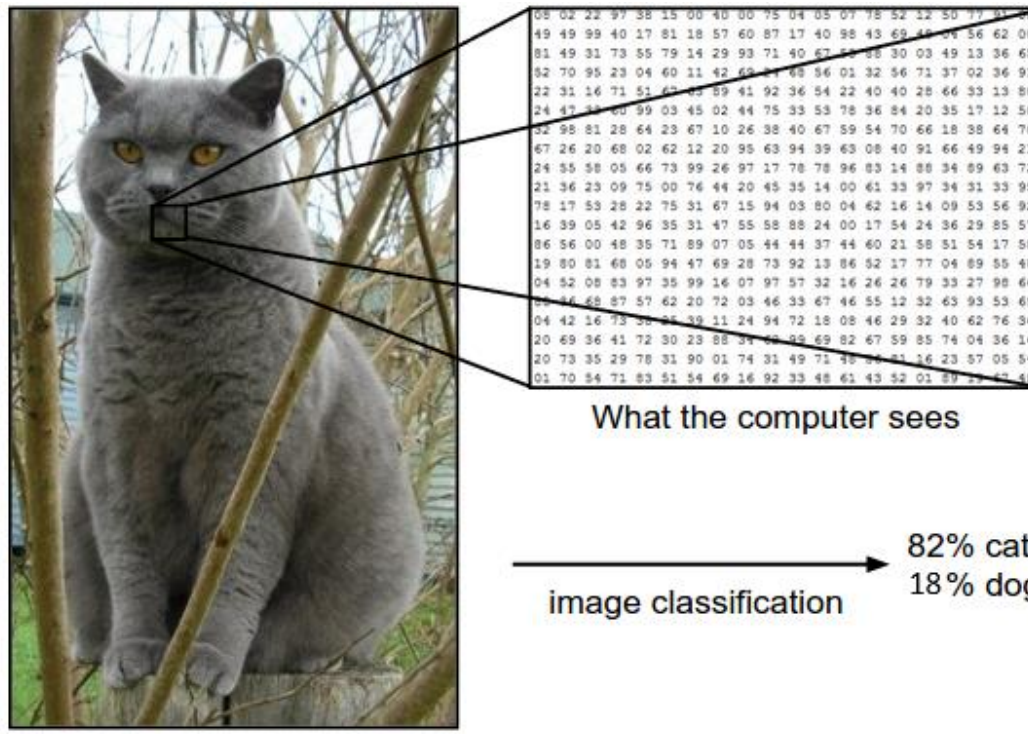




Neural Network

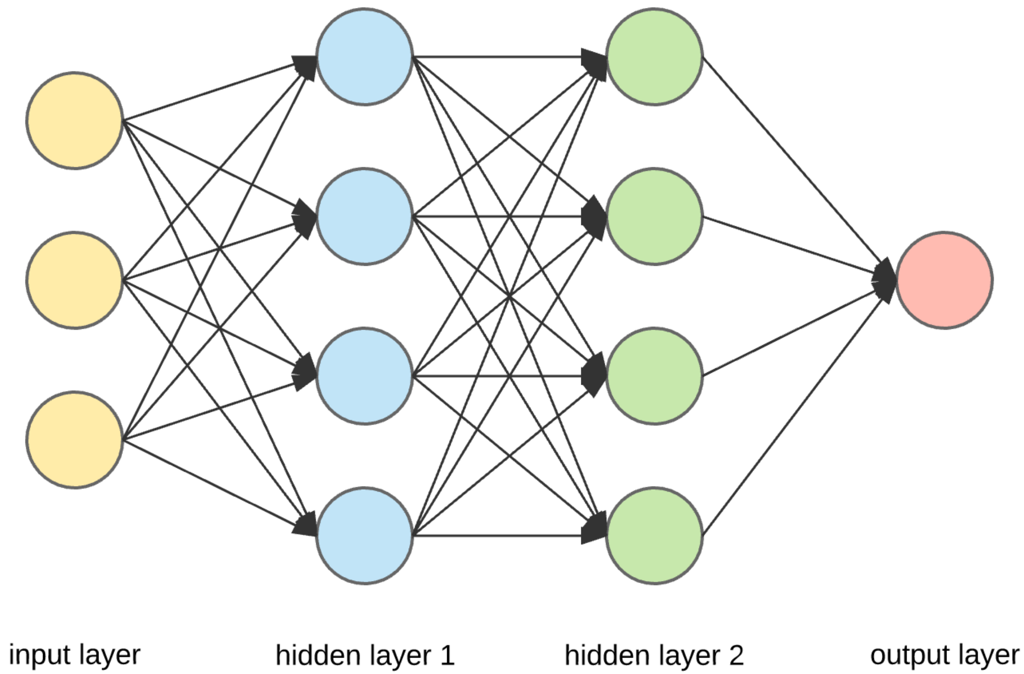
Study Case

Amongst thousands of dogs and
cats' images, how will the computer distinguish the cat photos from the dog photos?

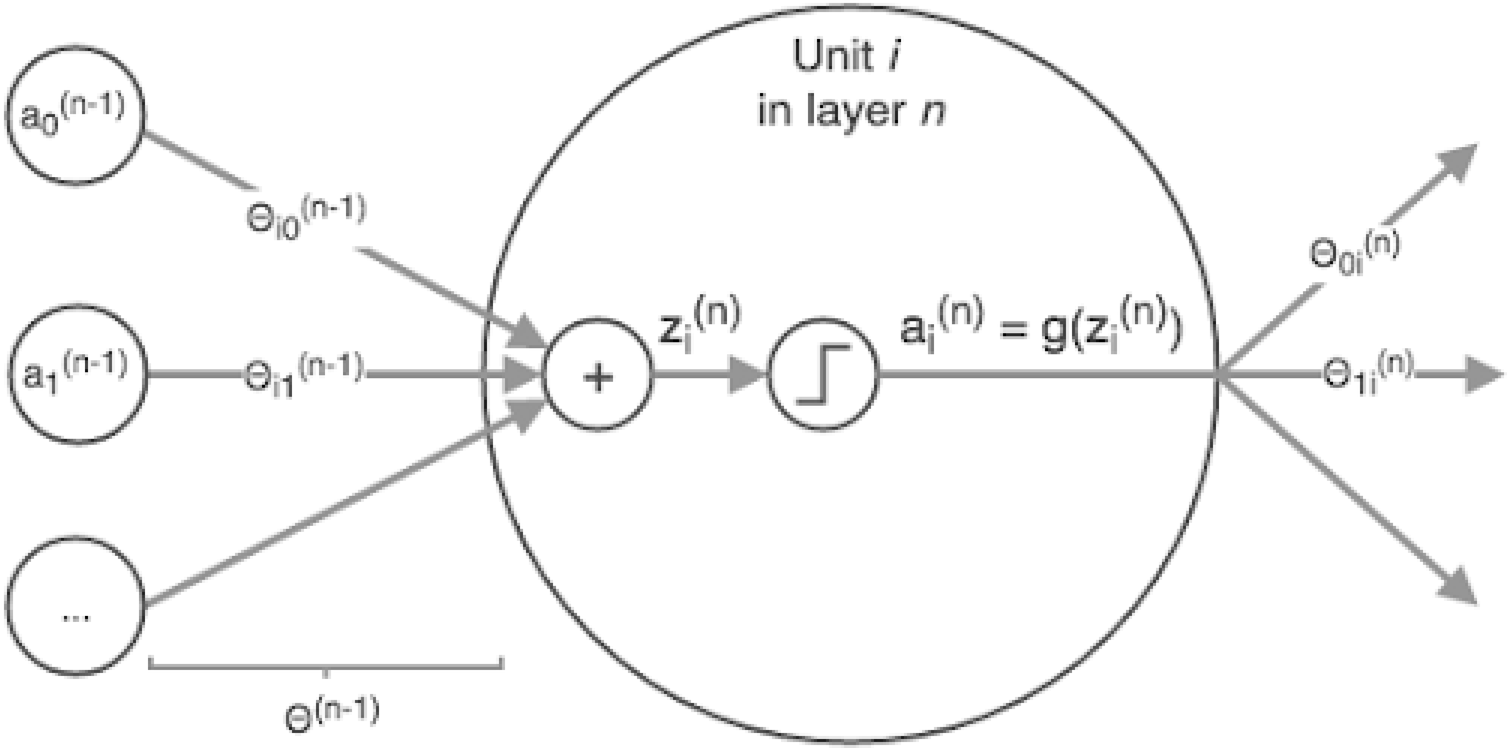


Feedforward Neural Network

Components

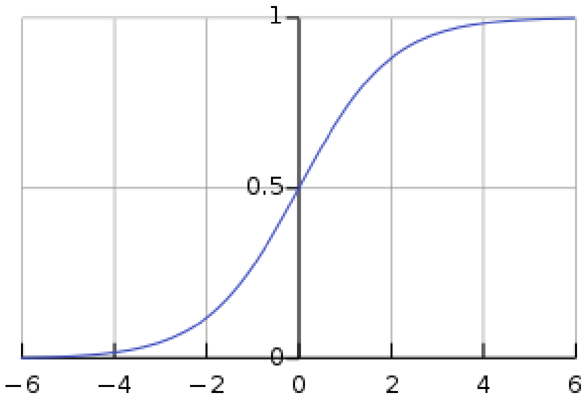


Forward Propagation



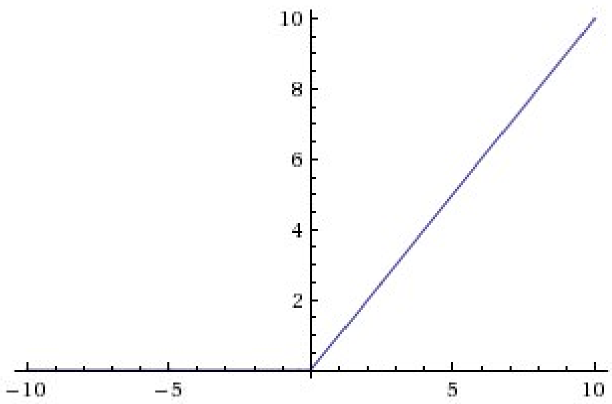
Activation Function

Sigmoid Function



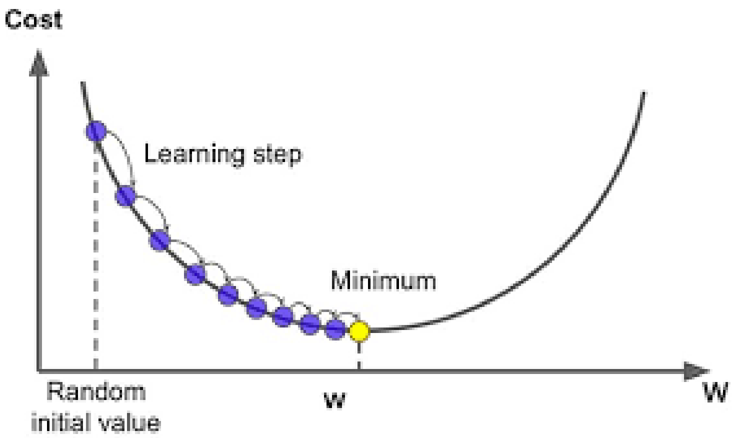
$$f(x) = \frac{1}{1 + e^{-x}}$$

ReLU Function



$$f(x) = \max(0, x)$$

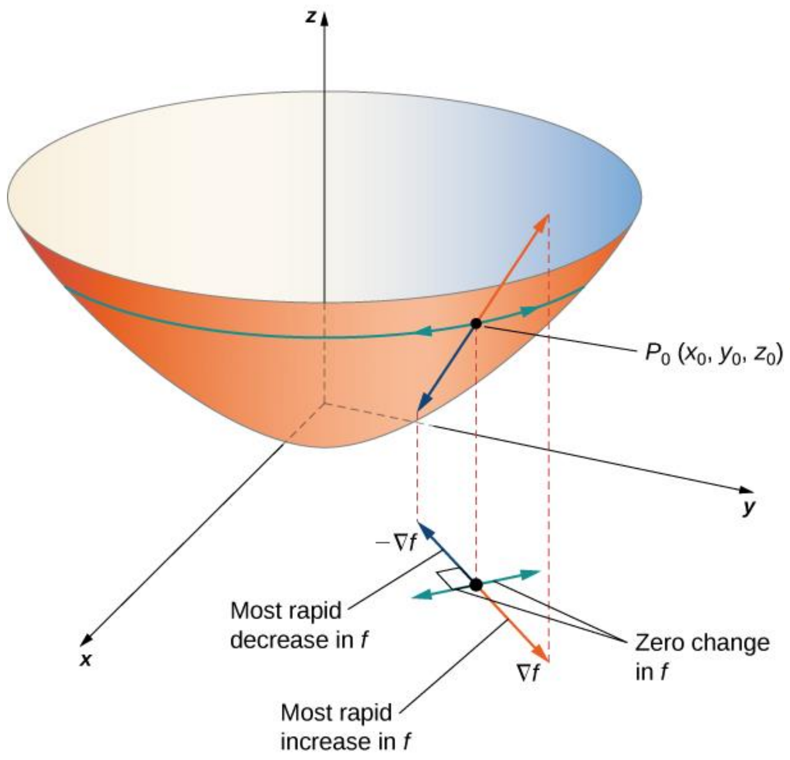
Gradient Descent



Algorithm 1 Gradient Descent

1: Initialize $\theta_0 = k \in \mathbb{R}^d$
2: Initialize $t \leftarrow 0$
3: **repeat**
4: $\quad \forall j \in \{1, ..., d\}, \theta_{j,t+1} \leftarrow \theta_{j,t} - \alpha \frac{\partial}{\partial \theta_j} L(\theta) \Big|_{\theta=\theta_t}$
5: $\quad t \leftarrow t + 1$
6: **until** converge

Why is gradient the direction of steepest ascent?



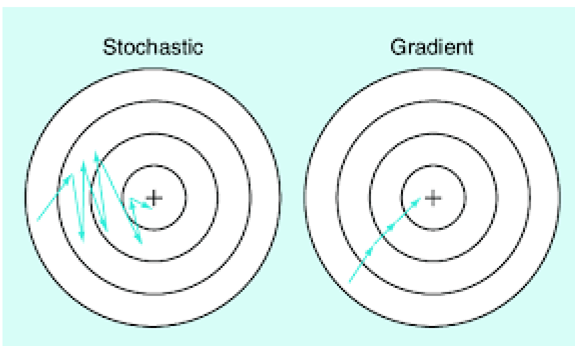
Assuming differentiability, $\nabla_{\hat{u}} f$ can be written as:

$$\nabla_{\hat{u}} f = \nabla f(\mathbf{x}) \cdot \hat{u} = |\nabla f(\mathbf{x})||\hat{u}| \cos\theta = |\nabla f(\mathbf{x})| \cos\theta$$

which is a maximum when $\theta = 0$: when $\nabla f(\mathbf{x})$ and $\hat{u}$ are parallel.

Other Gradient-based Learning Algorithm

SGD (Stochastic Gradient Descent)



Algorithm 1 Pseudo code of Stochastic Gradient Descent algorithm

Enter initial values (ϵ_k, θ)

while

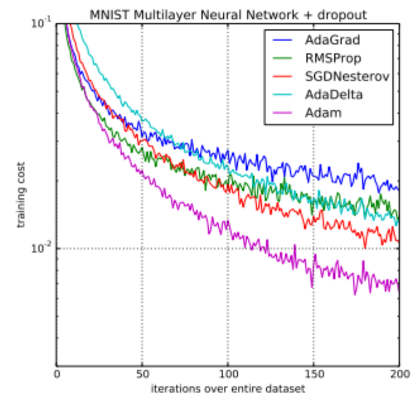
Sampling $\{x^{(1)}, \ldots, x^{(m)}\}$ *from Dataset by using* $\{y^{(1)}, \ldots, y^{(m)}\}$ *Labels*

$\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}, \theta), y^{(i)})$

$\theta \leftarrow \theta - \epsilon_k \, \hat{g}$

End while

Adam Optimizer (Adaptive Moment Estimation)



Step 1: while w_t do not converges

do{

Step 2: Calculate gradient $g_t = \frac{\partial f(x,w)}{\partial w}$

Step 3: Calculate $p_t = m_1 \cdot p_{t-1} + (1 - m_1) \cdot g_t$

Step 4: Calculate $q_t = m_2 \cdot q_{t-1} + (1 - m_2) \cdot g_t^2$

Step 5: Calculate $\widehat{p_t} = p_t / (1 - m_1^t)$

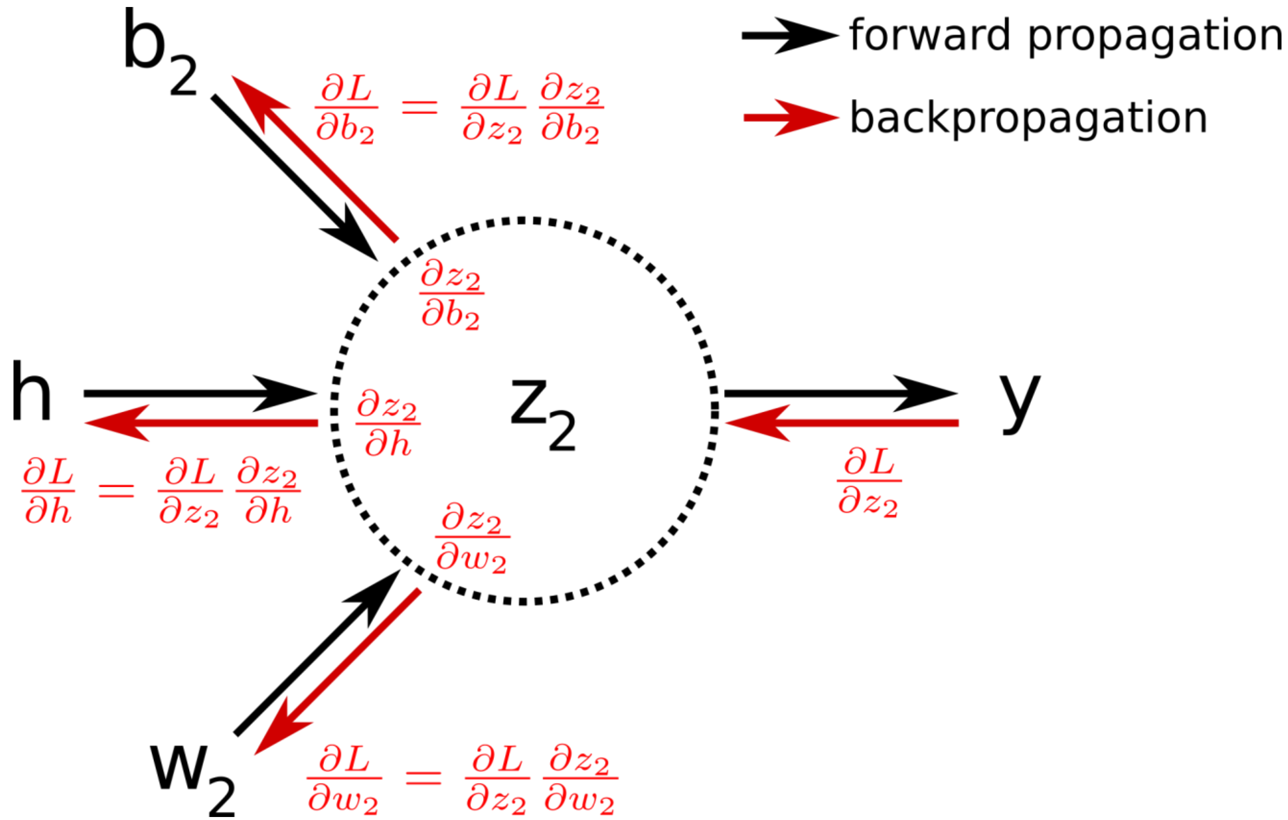
Step 6: Calculate $\widehat{q_t} = q_t / (1 - m_2^t)$

Step 7: Update the parameter $w_t = w_{t-1} - \alpha \cdot \widehat{p_t} / (\sqrt{\widehat{q_t}} + \epsilon)$

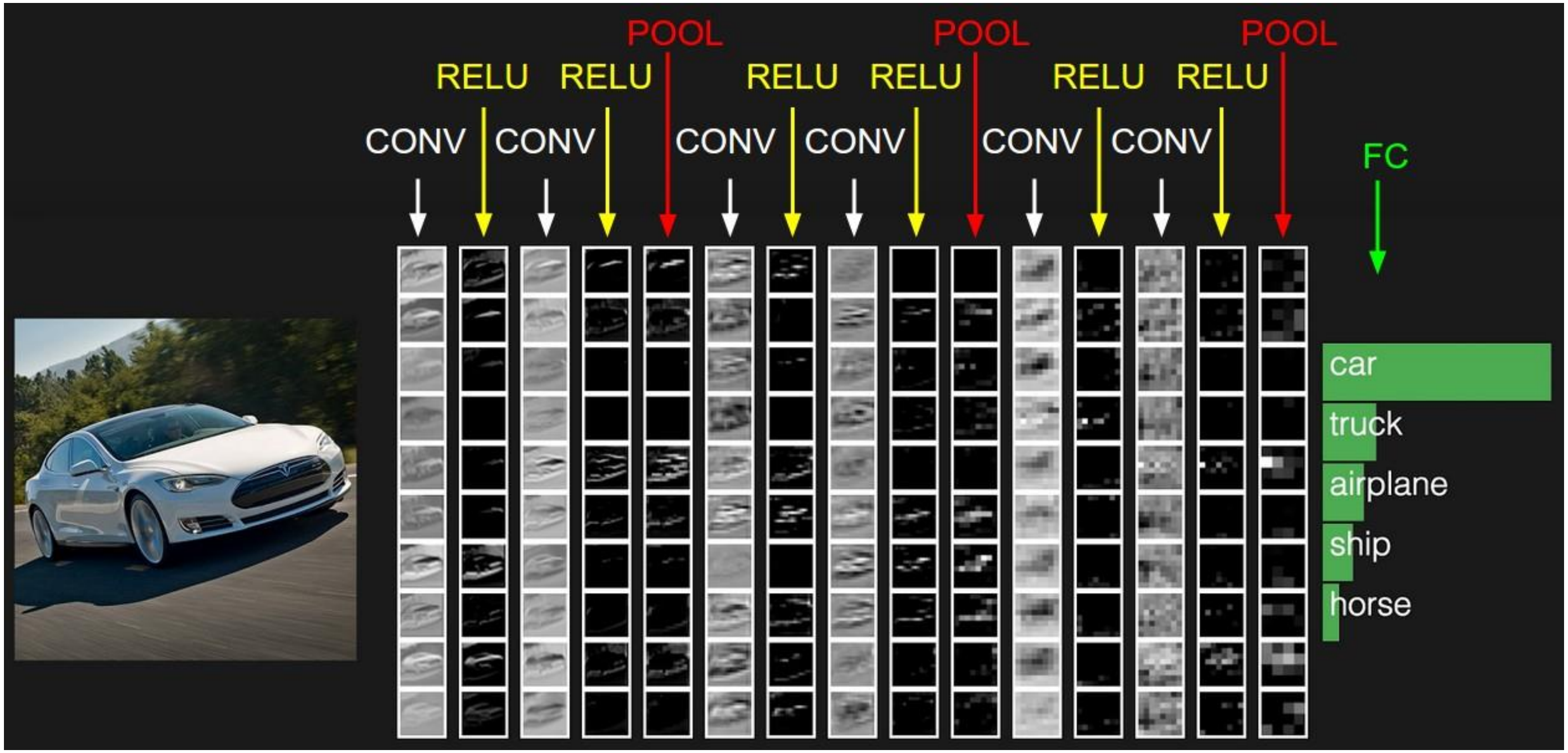
}

Step 8: return w_t

Backpropagation

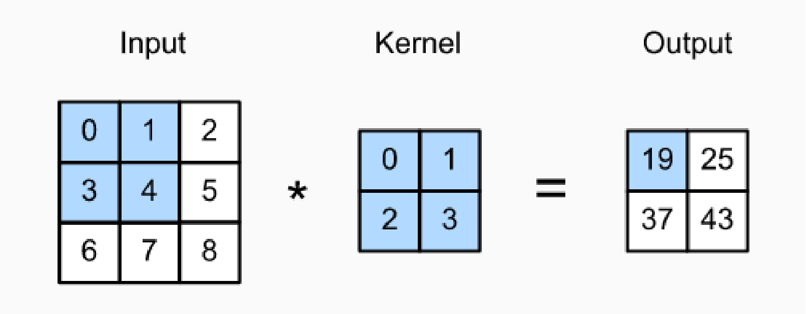


Convolutional Neural Network (CNN)



Convolution

- Inspired by animal's receptive field.
- The convolution operation can extract ubiquitous features of the picture that distinguish them from other classes.



Padding & Stride

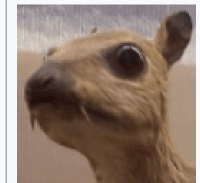
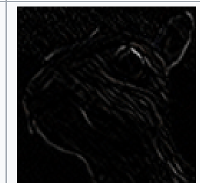
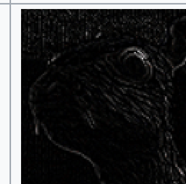
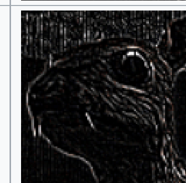
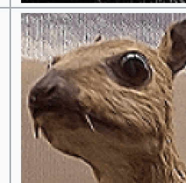
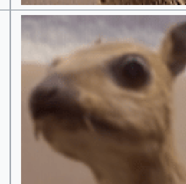
0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Why CNN works?

The purpose of performing convolution on images is to sharpen, blur, detecting edges,... Different kernels will give us different result.

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Sparse Connectivity

Sparse connections due to small convolution kernel

Viewed from below

Dense connections

Fully connected

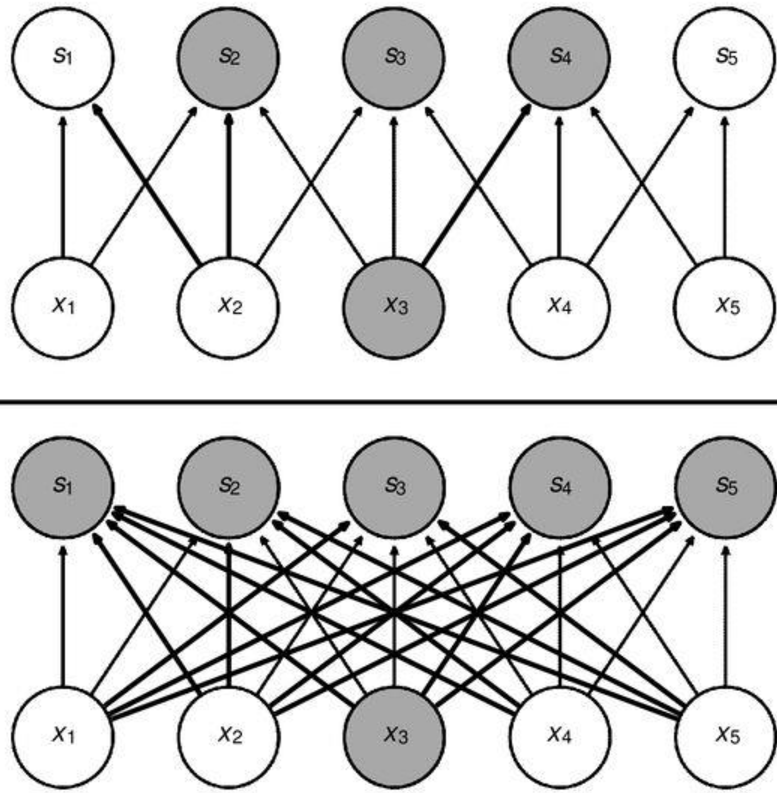


Figure 9.2

(Goodfellow 2016)

Parameter Sharing

Black arrows = particular parameter

Convolution shares the same parameters across all spatial locations

Traditional matrix multiplication does not share any parameters

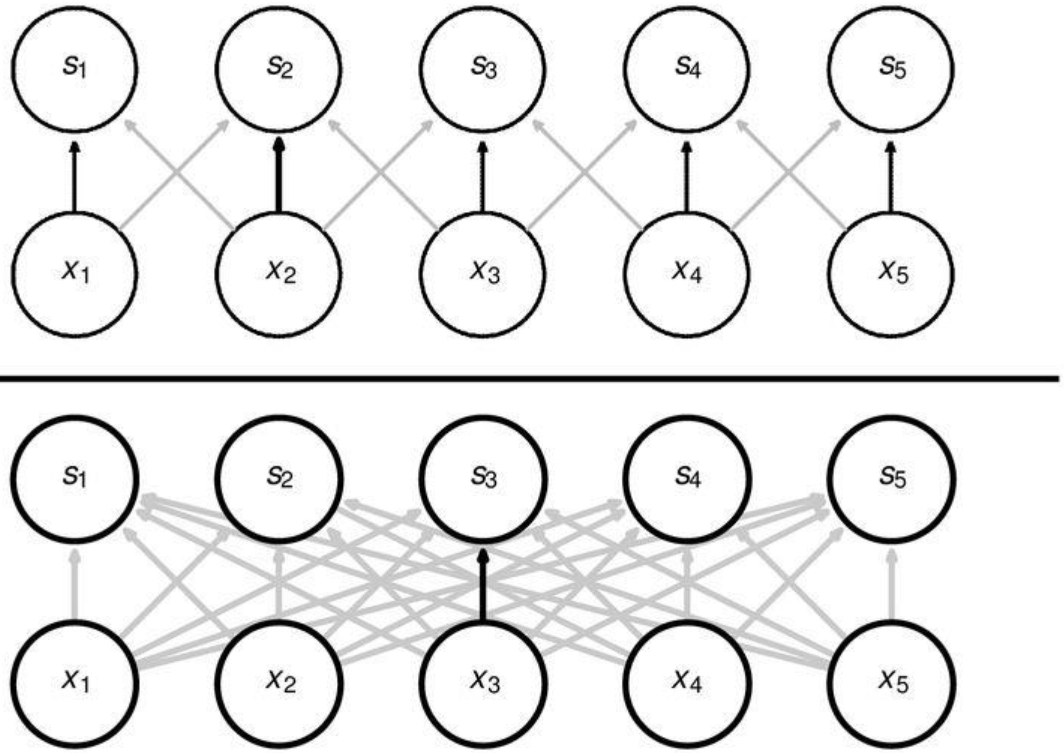
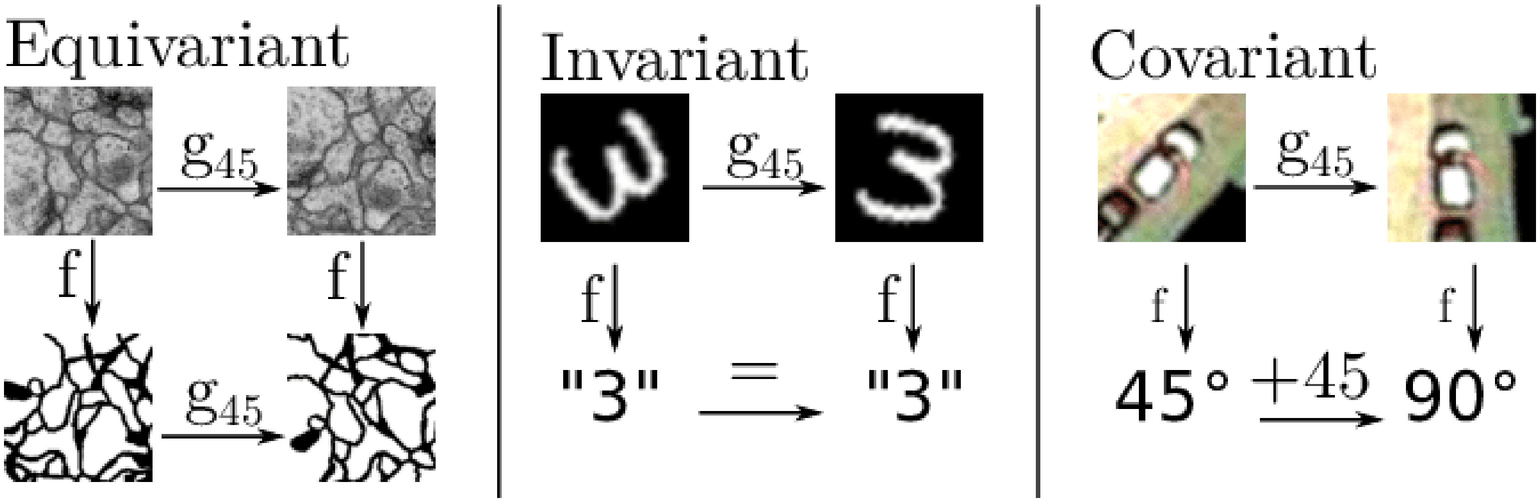
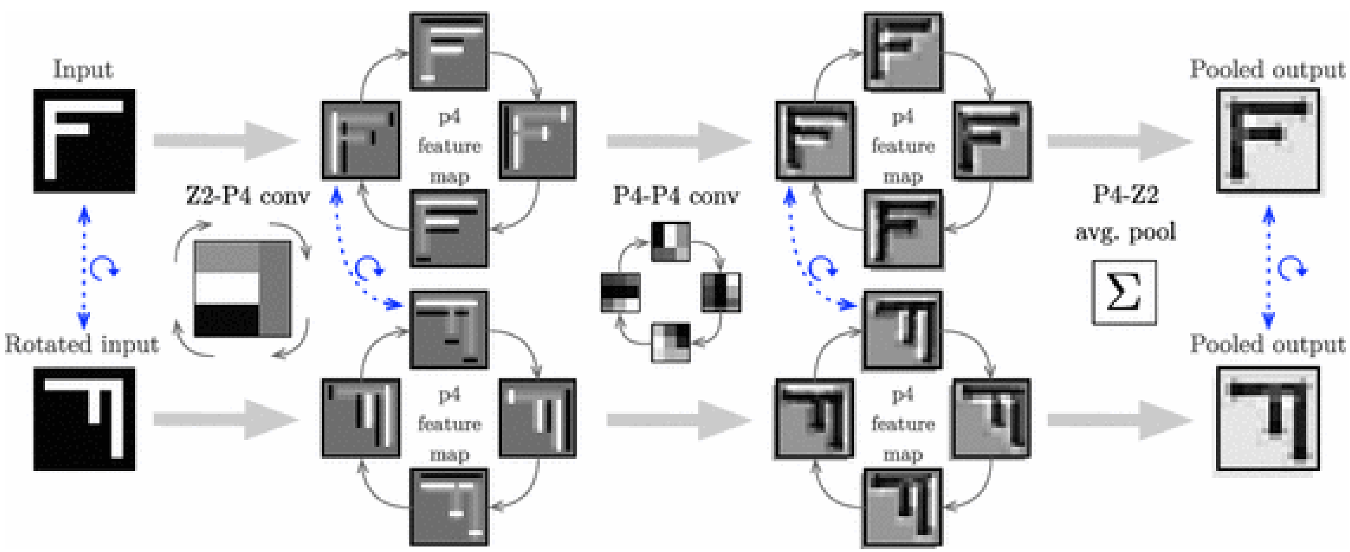


Figure 9.5

(Goodfellow 2016)



Equivariant
g45
f
f
g45
Invariant
g45
f
f
"3"
=
"3"
Covariant
g45
f
f
45°
+45
90°



Input
Rotated input
Z2-P4 conv
p4
feature
map
p4
feature
map
P4-P4 conv
p4
feature
map
p4
feature
map
P4-Z2
avg. pool
Σ
Pooled output
Pooled output

Residual Networks

- RESIDUAL LEARNING & IDENTITY MAPPING BY SHORTCUTS
- NETWORK ACHITECTURE

Motivation
Vanishing Gradient

$$\frac{\partial J(W,b;x,y)}{\partial W_{i\leftarrow j}^{(n_l-4)}} = \left(\frac{da_i^{(n_l-3)}}{dz_i^{(n_l-3)}} \cdot \sum_{k=1}^{s_{n_l-2}} W_{k\leftarrow i}^{(n_l-3)} \cdot \left(\frac{da_k^{(n_l-2)}}{dz_k^{(n_l-2)}} \cdot \sum_{q=1}^{s_{n_l-1}} W_{q\leftarrow k}^{(n_l-2)} \cdot \left(\frac{da_q^{(n_l-1)}}{dz_q^{(n_l-1)}} \cdot \left(\sum_{p=1}^{s_{n_l}} W_{p\leftarrow q}^{(n_l-1)} \left(-\frac{da_p^{(n_l)}}{dz_p^{(n_l)}} \left(y_p - h_p(W,b;x) \right) \right) \right) \right) \right) \right) \cdot a_j^{(n_l-4)}$$

Vanishing Gradient: multiply many “less than 1” numbers

→ Gradients converge to 0

→ Gradient update become meaningless

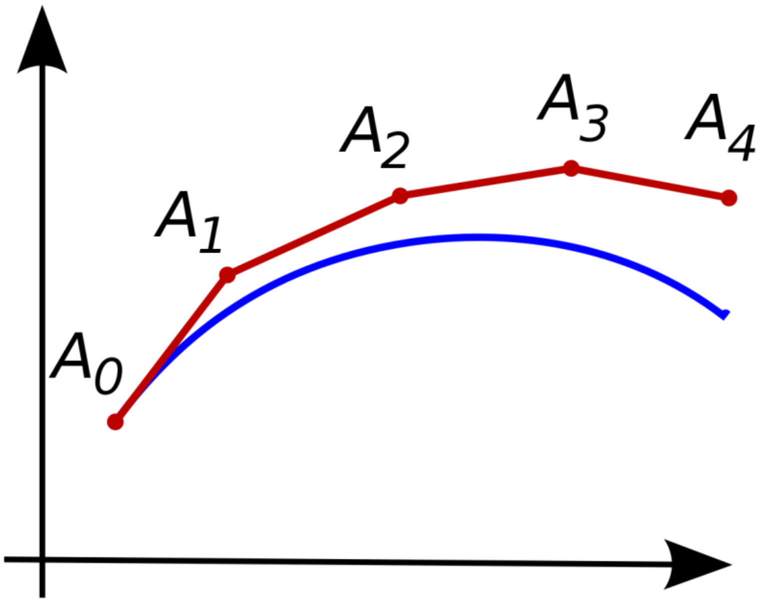
→ The model is unable to learn

Motivation
Learning to 0

Learning to 1: the model learns to approximate ŷ such that ŷ = Wx + b

→ In in the very last layers, the weights approximately equal to 1 (identity matrix)

Learning to 0: the model learns to minimize r such that r = y - ŷ



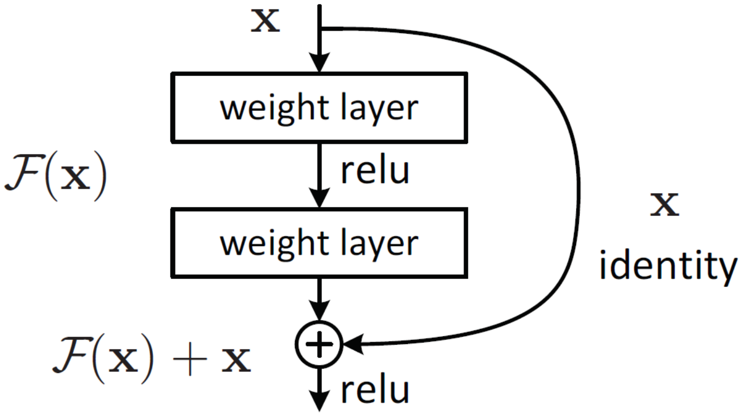
Residual Learning & Identity Mapping by Shortcuts

$\mathcal{H}(x)$ denotes the formal desired underlying mapping

$$\mathcal{F}(x) = \mathcal{H}(x) - x$$

the initial mapping is recast into

$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

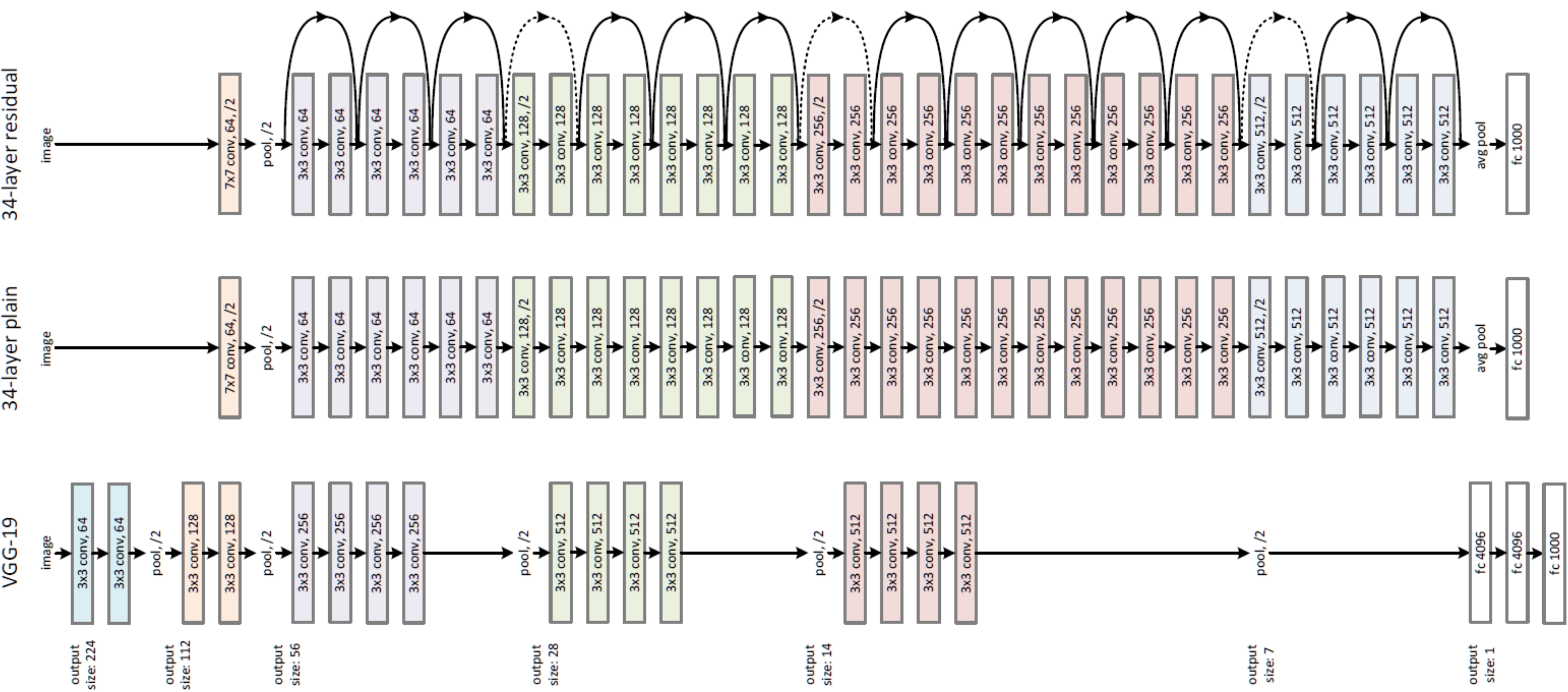


Identity shortcut connections do not require any extra parameters and they do not cost more computational complexity.

Can be easily implemented using common libraries without modifying the solvers, and trained end-to-end by SGD with backpropagation.

Network Architecture

VGG19: 19.6 billion FLOPs	**Plain34:** 3.6 billion FLOPs	**ResNet34:** 3.6 billion FLOPs



Backprop in ResNet

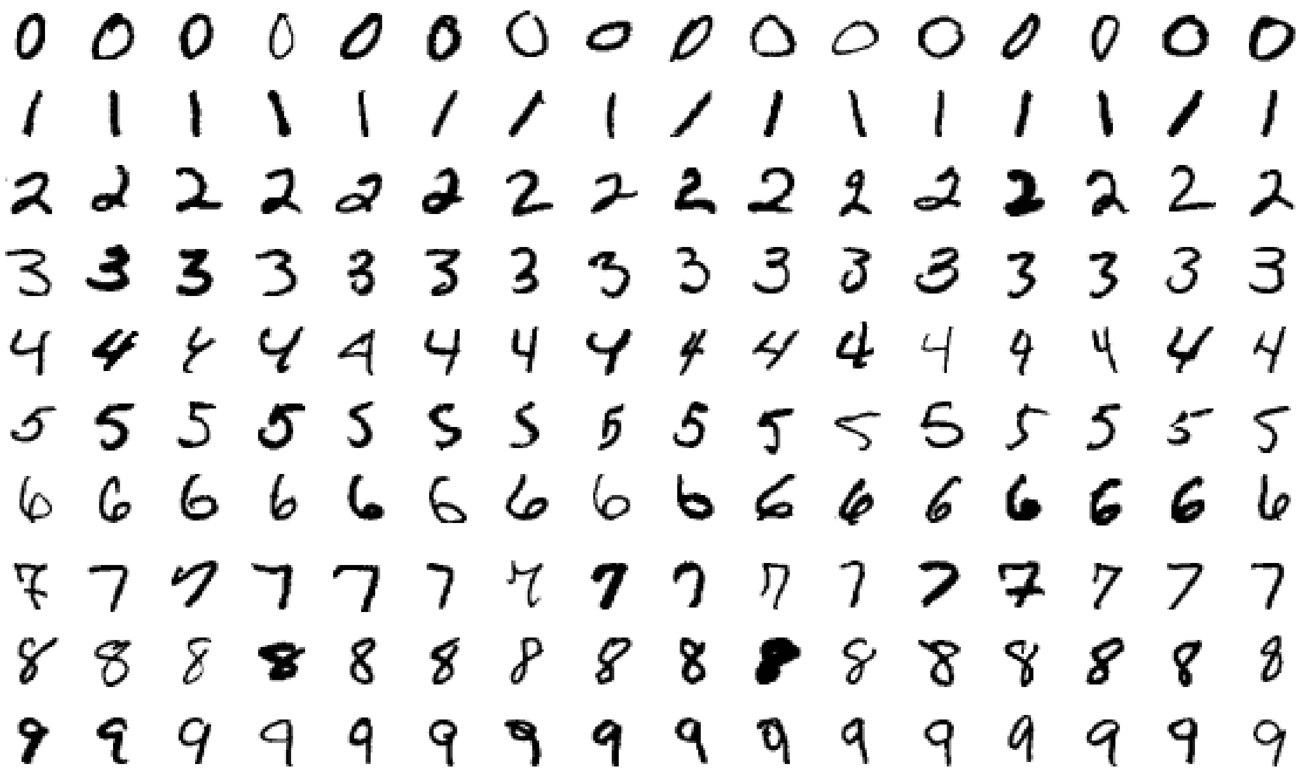
Backpropagation is a general algorithm that can be applied anywhere. ResNet is no exception.
Let $y = \mathcal{F}(x) + x$. Consider our main objective here is to calculate $\frac{\partial E}{\partial x}$, without the shortcut path, we would have

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} * \frac{\partial y}{\partial x} = \frac{\partial E}{\partial y} * F'(x)$$

Now with shortcut connection,

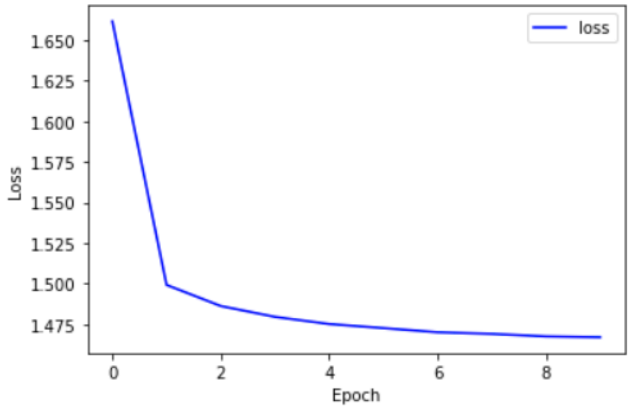
$$\begin{aligned} \frac{\partial E}{\partial x} &= \frac{\partial E}{\partial y} * \frac{\partial y}{\partial x} \\ &= \frac{\partial E}{\partial y} * (1 + F'(x)) \\ &= \frac{\partial E}{\partial y} + \frac{\partial E}{\partial y} * F'(x) \end{aligned} \tag{3.5}$$

MNIST Digits Classifier

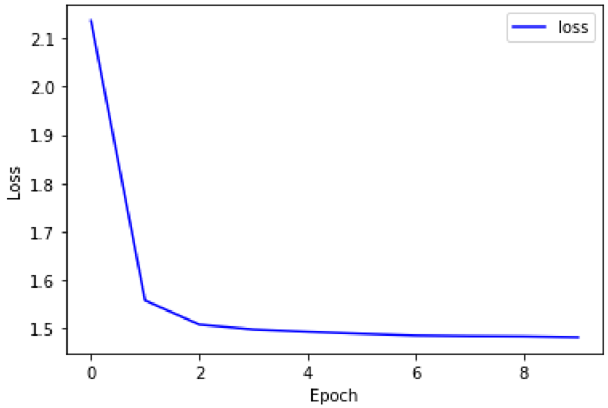


Comparison

ResNet-34

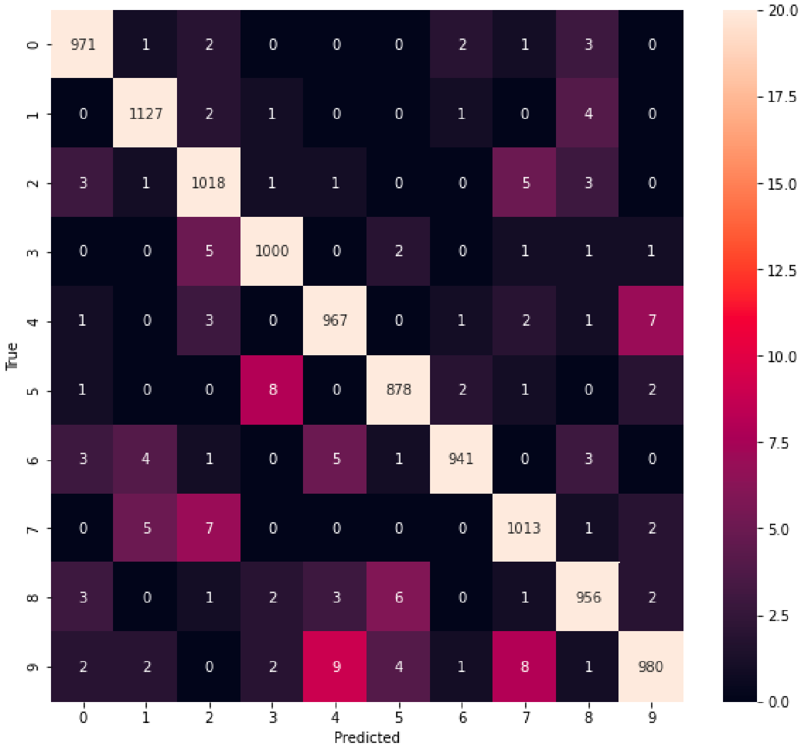


ConvNet-34

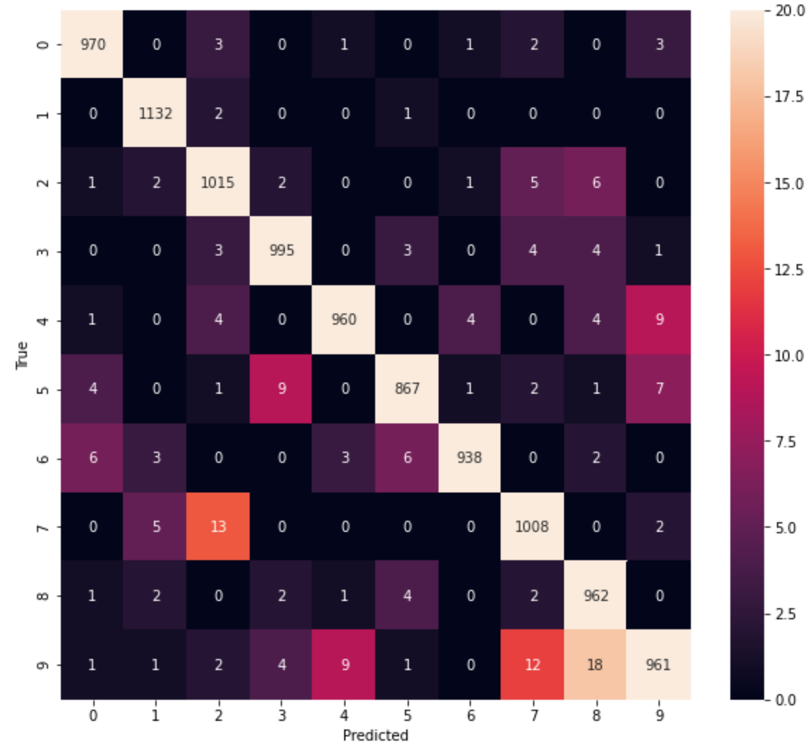


Comparison

ResNet-34

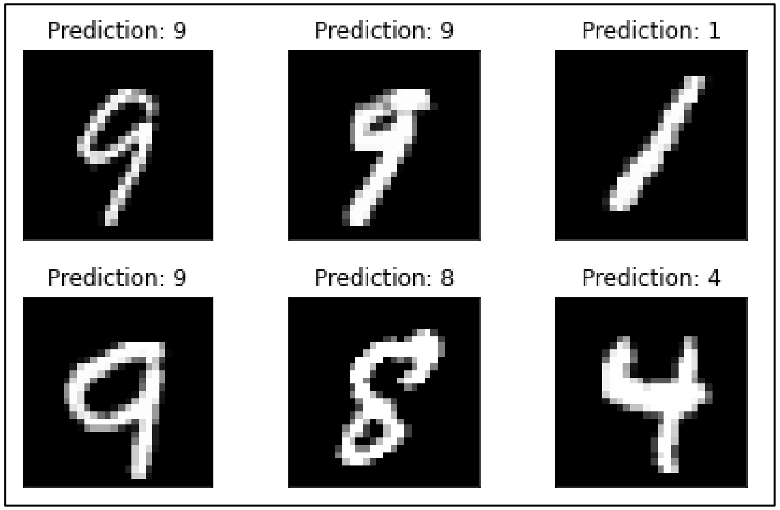


ConvNet-34

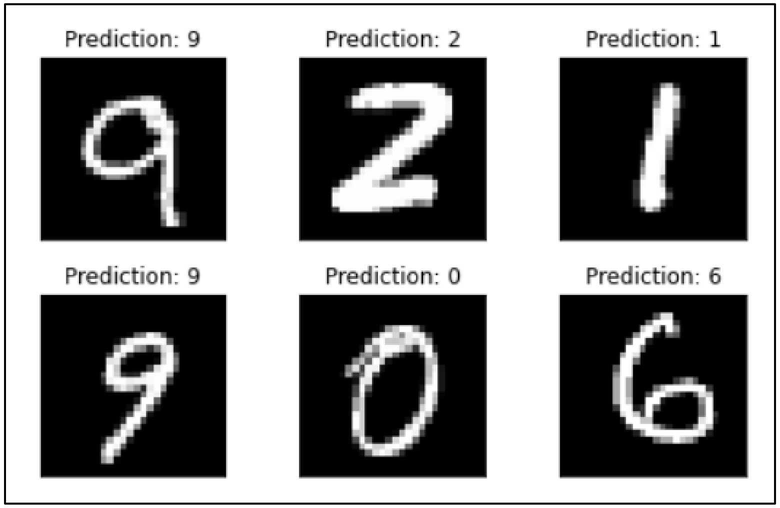


Comparison

ResNet-34



ConvNet-34

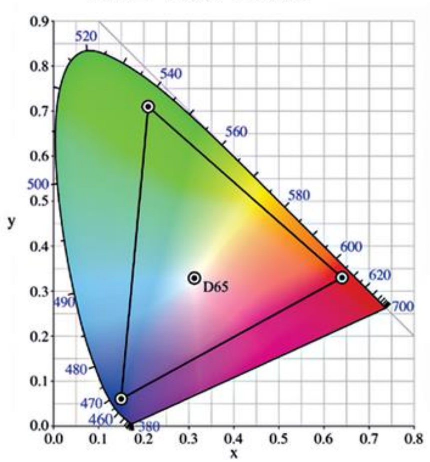


Application to Image Colorization

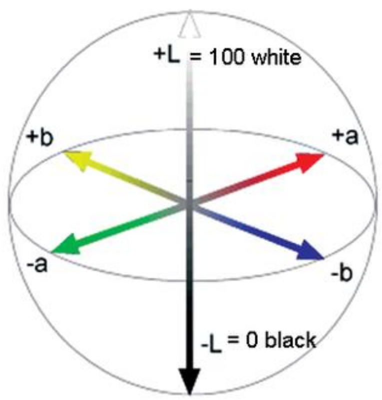
- METHODOLOGY & MODEL
- DATA PREPARATION
- RESULT & COMPARISON

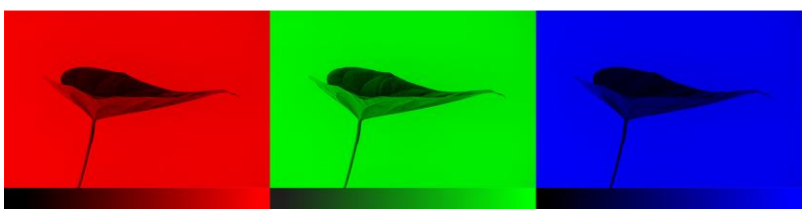
Methodology
Color Spaces

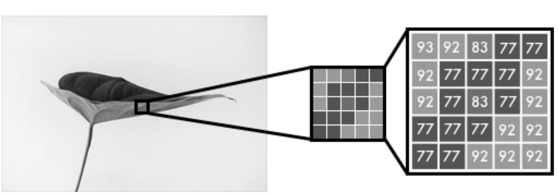
RGB Color Model

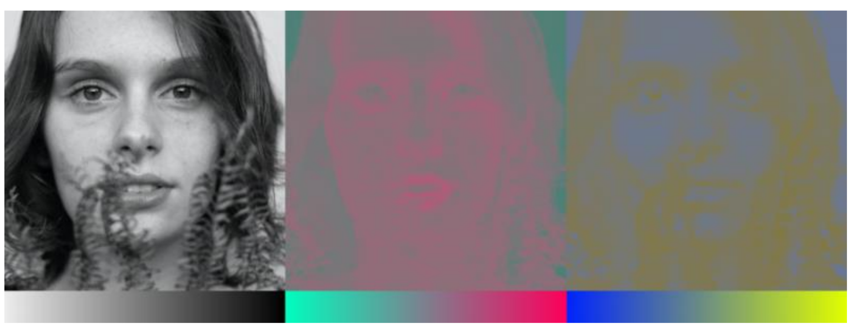


CIE L*a*b* Color Space

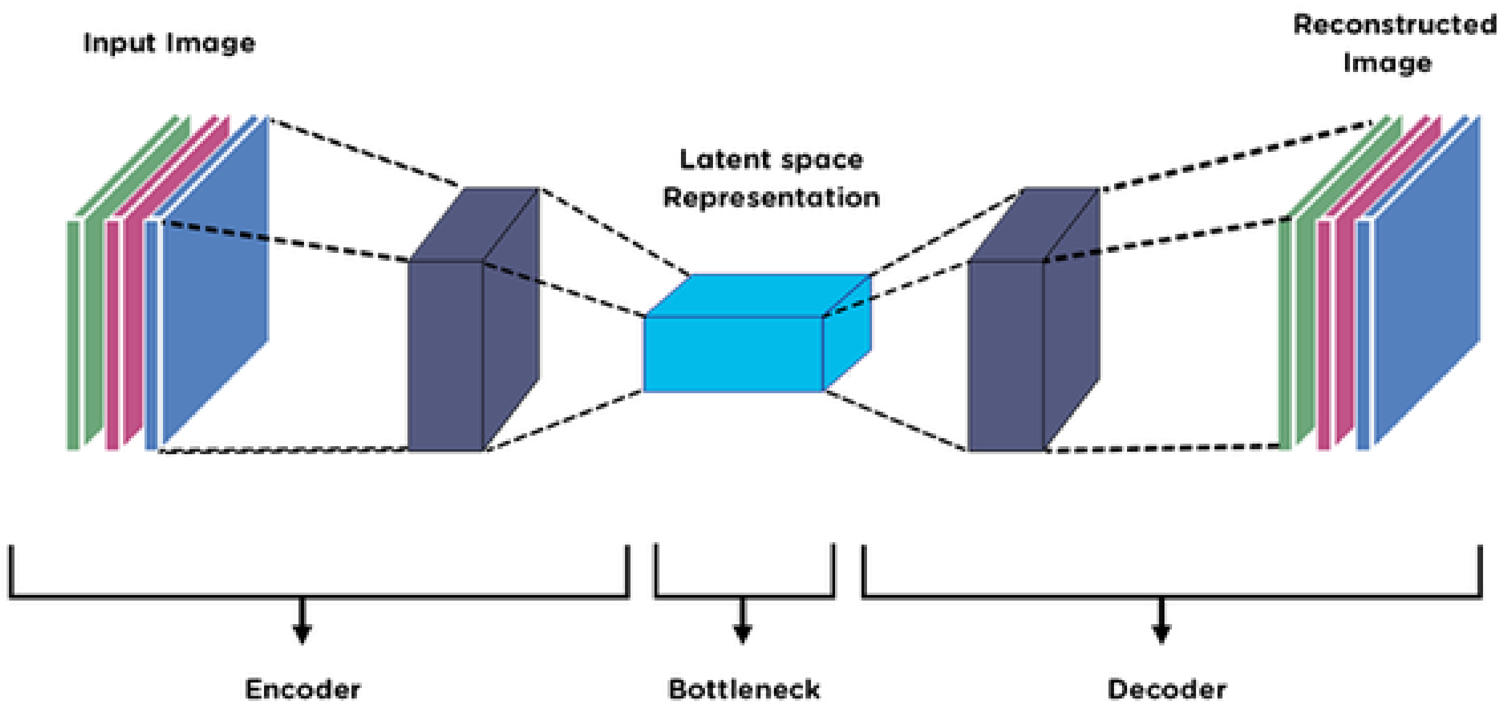








Methodology
Autoencoder



Methodology
Autoencoder

The image h (code, latent variables, or latent representation) and can simply be achieved by activating a linear transformation: $h = \sigma(Wx + b)$

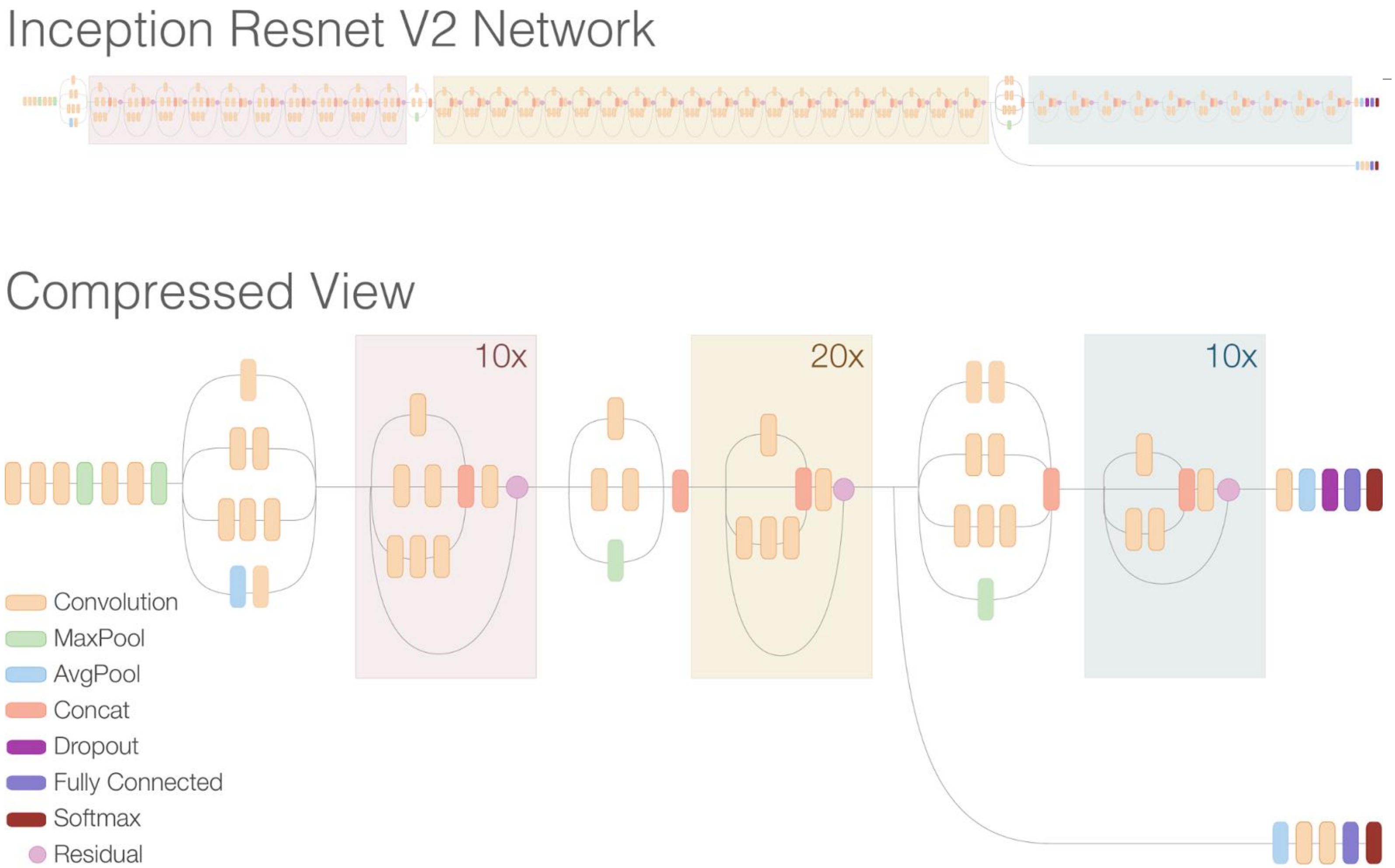
To retrieve the input x from the image h, the decoder also use a linear transformation followed by an activation function: $x' = \sigma'(W'h + b')$

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

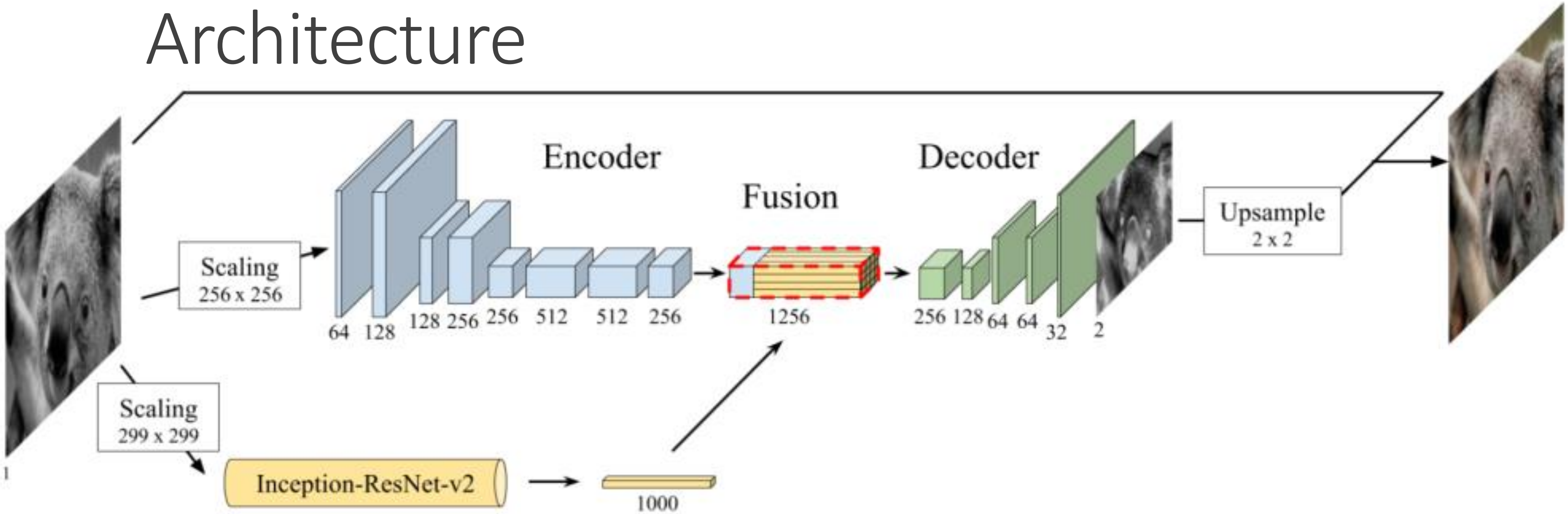
$$\phi, \psi = \underset{\phi, \psi}{\operatorname{argmin}} \|X - (\phi \circ \psi) X\|^2$$

Methodology



Model

Architecture



Feature Extractor

Encoder Network			Fusion Network			Decoder Network		
Layer	Kernel	Stride	Layer	Kernel	Stride	Layer	Kernel	Stride
conv	$64 \times (3 \times 3)$	2×2	fusion	—	—	conv	$128 \times (3 \times 3)$	1×1
conv	$128 \times (3 \times 3)$	1×1	conv	$256 \times (1 \times 1)$	1×1	upsamp	—	—
conv	$128 \times (3 \times 3)$	2×2				conv	$64 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	1×1				conv	$64 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	2×2				upsamp	—	—
conv	$512 \times (3 \times 3)$	1×1				conv	$32 \times (3 \times 3)$	1×1
conv	$512 \times (3 \times 3)$	1×1				conv	$2 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	1×1				upsamp	—	—

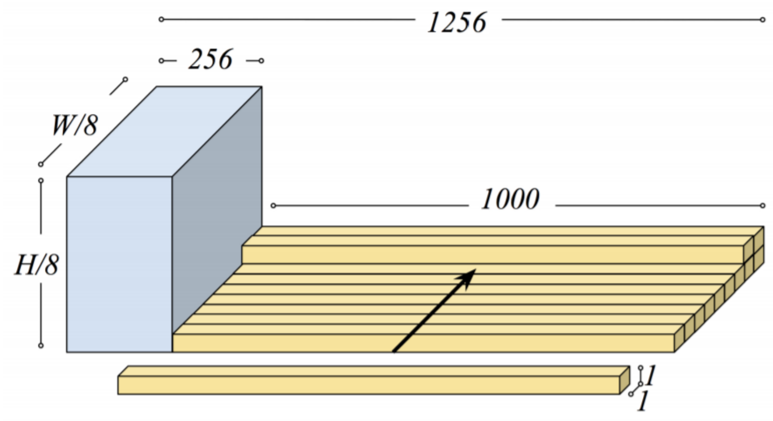
Model
Architecture

Encoder: 8 convolutional layers output a H/8 x W/8 x 256 latent space feature representation

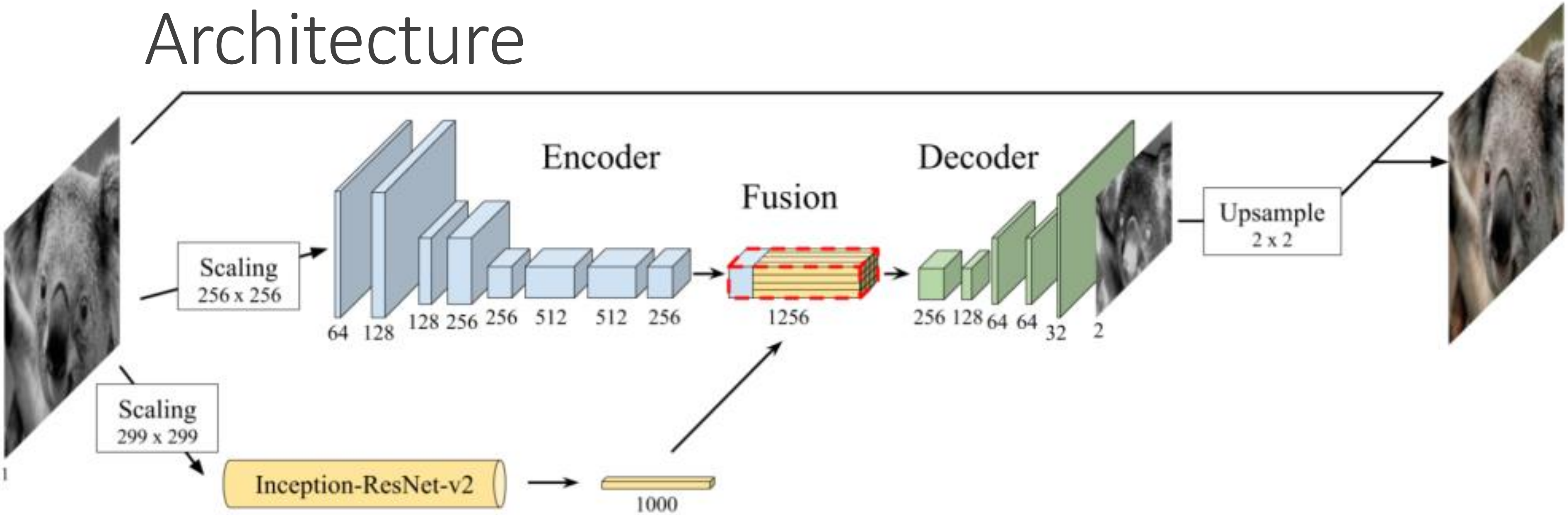
Feature Extractor: Inception-Resnet-v2 extracts a 299 x 299 x 3 image to a 1 x 1 x 1000 vector

Fusion: concatenates the encoder with the feature extractor, obtains the space of H/8 x W/8 x 1256

Decoder: convolutes the fusion output to a H/8 x W/8 x 256 volume and start to decode after that



Model
Architecture



Feature Extractor

Encoder Network			Fusion Network			Decoder Network		
Layer	Kernel	Stride	Layer	Kernel	Stride	Layer	Kernel	Stride
conv	$64 \times (3 \times 3)$	2×2	fusion	—	—	conv	$128 \times (3 \times 3)$	1×1
conv	$128 \times (3 \times 3)$	1×1	conv	$256 \times (1 \times 1)$	1×1	upsamp	—	—
conv	$128 \times (3 \times 3)$	2×2				conv	$64 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	1×1				conv	$64 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	2×2				upsamp	—	—
conv	$512 \times (3 \times 3)$	1×1				conv	$32 \times (3 \times 3)$	1×1
conv	$512 \times (3 \times 3)$	1×1				conv	$2 \times (3 \times 3)$	1×1
conv	$256 \times (3 \times 3)$	1×1				upsamp	—	—

Model
Training & Loss Function

$$\mathcal{L} = MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Adam Optimizer: initial learning rate of 1×10^{-3}

ReduceLROnPlateau : halves the learning rate but no less than 1×10^{-5}

Checkpoint: monitors the training loss, saves only best model

Batch size: 20 images per iteration

Data Preparation
Dataset

Data Preparation
Pre-processing

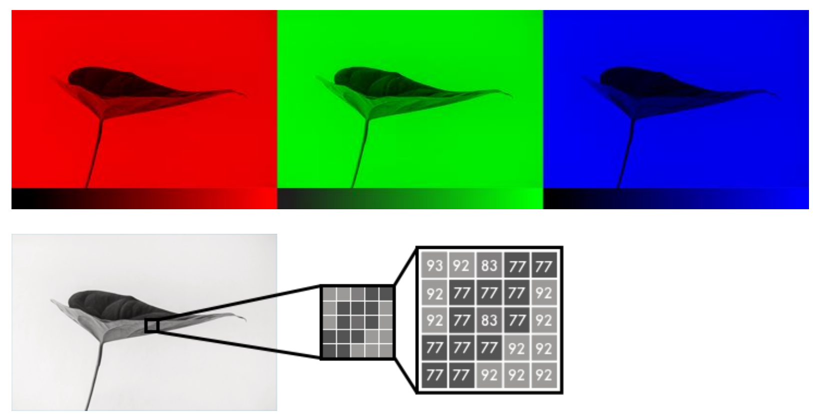
ImageDataGenerator: shearing, zooming, rotating, flipping

Data Preparation

Pre-processing

Encoder input: covert an RGB image to Lab color space and extract the L* component (256 x 256 x 1)

Inception input: convert to a grayscale RGB image (299 x 299 x 3)



Data Preparation
Pre-processing

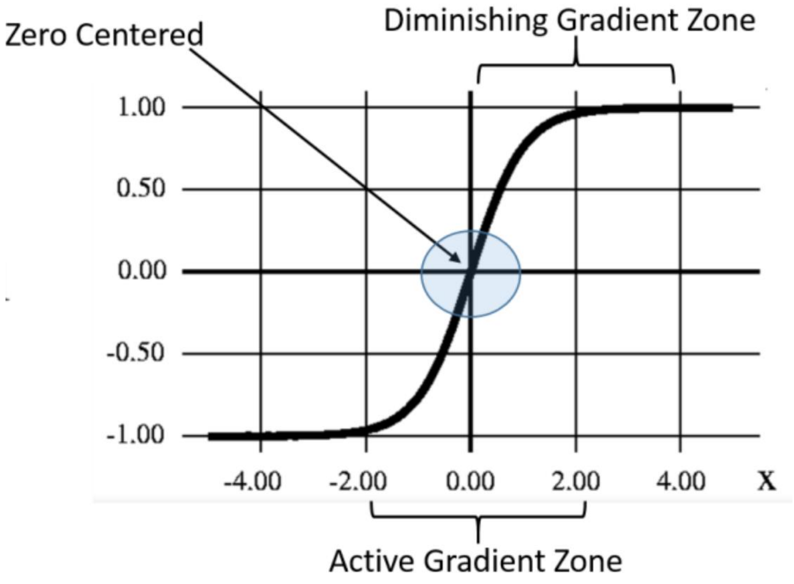
Rescale the original RGB image by multiplication with 1/255 to obtain the value in range [0, 1]

Convert to Lab color space → value in range [-1, 1]

→ During backward propagation, Gradient is actively updated

Post-processing: multiply the a*b* prediction with 256 instead of 128 to reduce the training effort.

Add the L* component and convert back to RGB



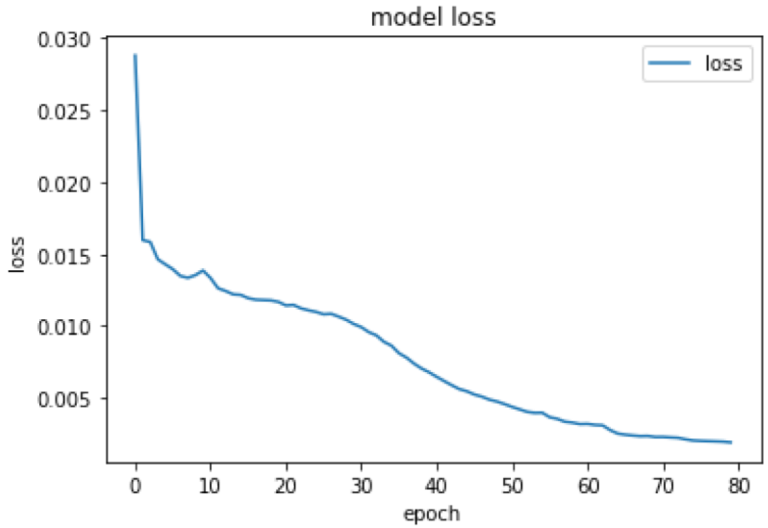
Result
Landscape Pictures

Kaggle Kernel with the configuration:

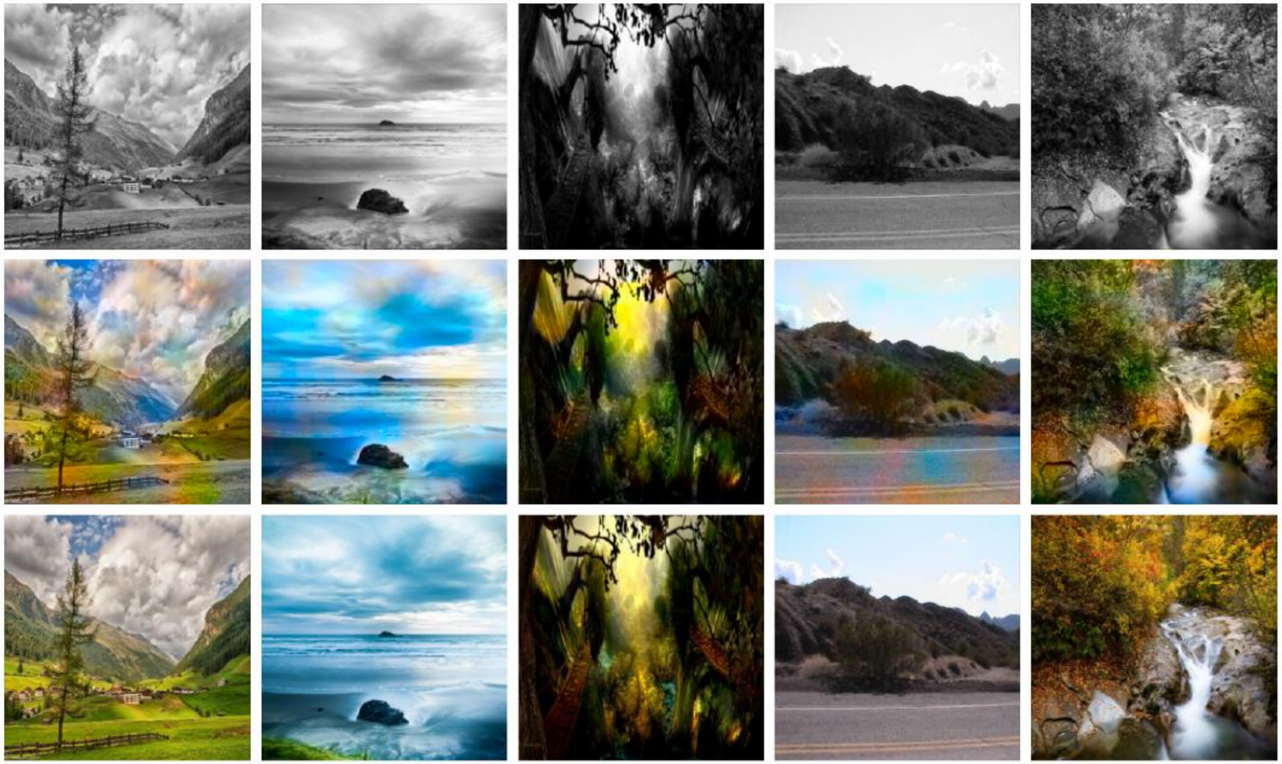
- CPU: 1x single core hyper threaded (1 core, 2 threads) Intel(R) Xeon(R) Processors @ 2.2Ghz, 55MB Cache
- RAM: 13GB
- GPU: NVIDIA Tesla P100 PCIe 16 GB
- Disk: 20GB

8 hours of training:

- 80 epochs
- 365s per epoch
- 1.5s per step

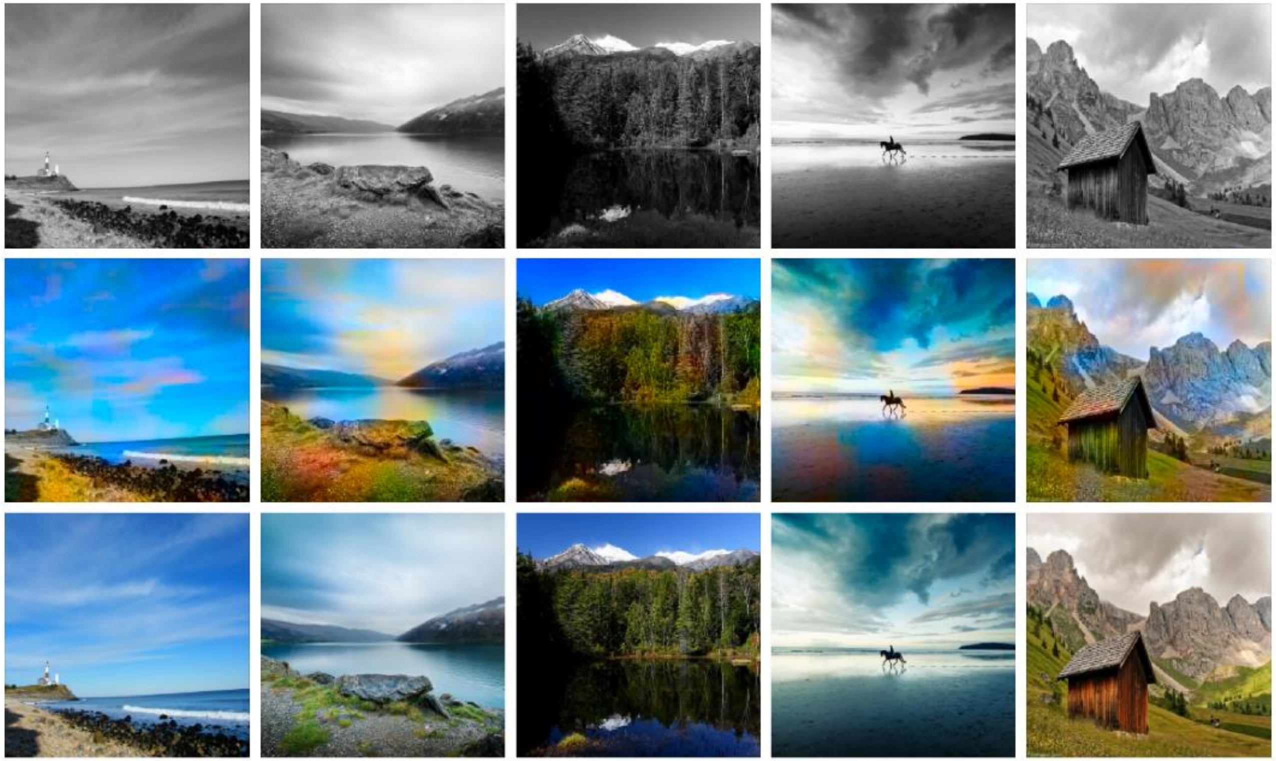


Result
Landscape Pictures



(a) Image 1 – 5
First row: Gray-scale image
Second row: Colorized image
Third row: Expected color image

Result
Landscape Pictures



(b) Image 6 – 10
First row: Gray-scale image
Second row: Colorized image
Third row: Expected color image

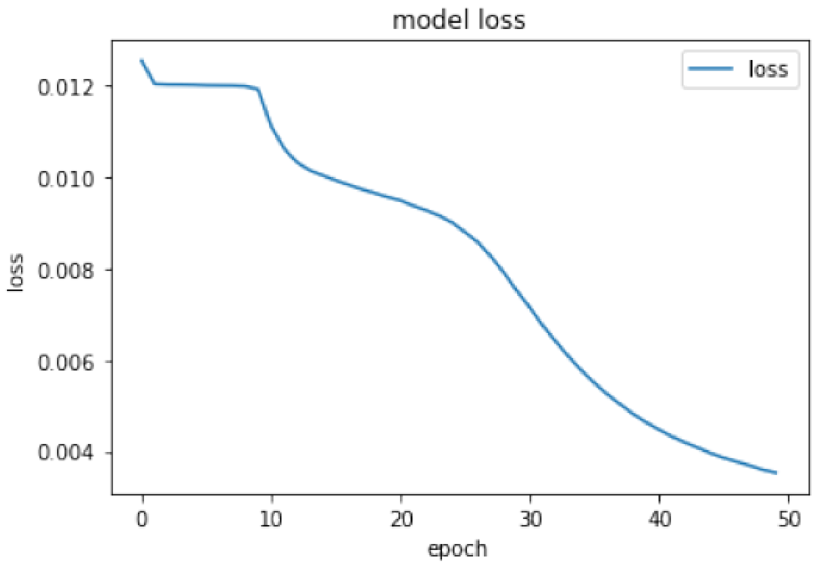
Result
Places365-Standard

Microsoft Azure's Virtual Machine:

- CPU: 4x vCPU (4 core, 8 threads) Intel(R) Xeon(R) E5-2673 v3 2.4 GHz (Haswell) processors
- RAM: 16GB
- GPU: None
- Disk: 30GB
- Operating system: Ubuntu 18.04

5 days of training:

- 25 epochs
- 5h per epoch
- 10s per step



Result
Places365-Standard

(a) Image 1 – 5
First row: Gray-scale image
Second row: Colorized image
Third row: Expected color image

Result
Places365-Standard

(b) Image 6 – 10
First row: Gray-scale image
Second row: Colorized image
Third row: Expected color image

Result

Image of an object captures the optical representation of that object

→ May be varied by many factors: lighting, weather condition, and even the precision of the camera's sensor itself.

A colorized image is just an item in the set of many other possible color mixtures. What we are trying to do is to colorize an image that it seems “natural” and everybody can “feel” that it is natural.

We believe that there is no definition of a “natural” image but there is “look and feel” of the human brain that decides whether an image is natural or not.

Result
Historical images

Result
Historical images

Comparison
Result of Basic CNN model

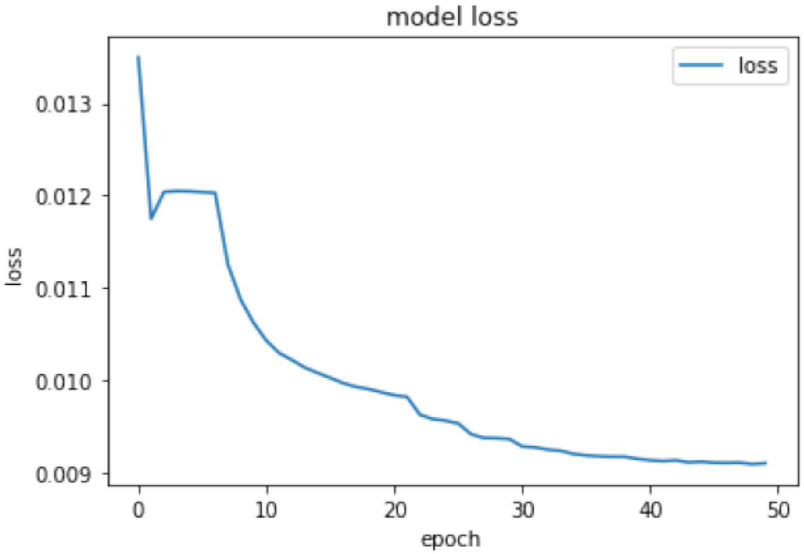
Remove the Feature Extractor and Fusion layers to create a simple Autoencoder

7 days of training:

- 50 epochs
- 4h per epoch
- 5s per step

There is no significant difference in result between the two proposed approaches

Original model has achieved the same loss value in 70% the time, and half the number of epochs



Conclusion & Future Work

Conclusion

Covered the background knowledge related to our topic

Elucidated the architecture of a residual network

Re-implemented a residual network for a simple image classifier

Application to Image Colorization will help reduce the amount of work in recovering and colorizing black-and-white images down to few seconds

Future work with the video colorization problem

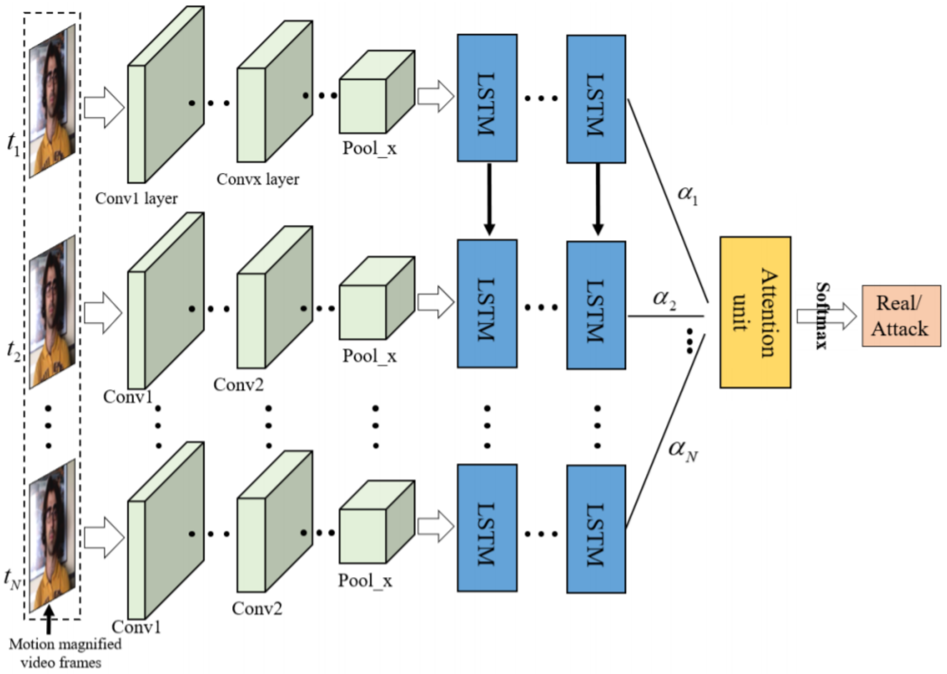
We hope our work in the future will make a considerable contribution to the community.

Future work
Motivation

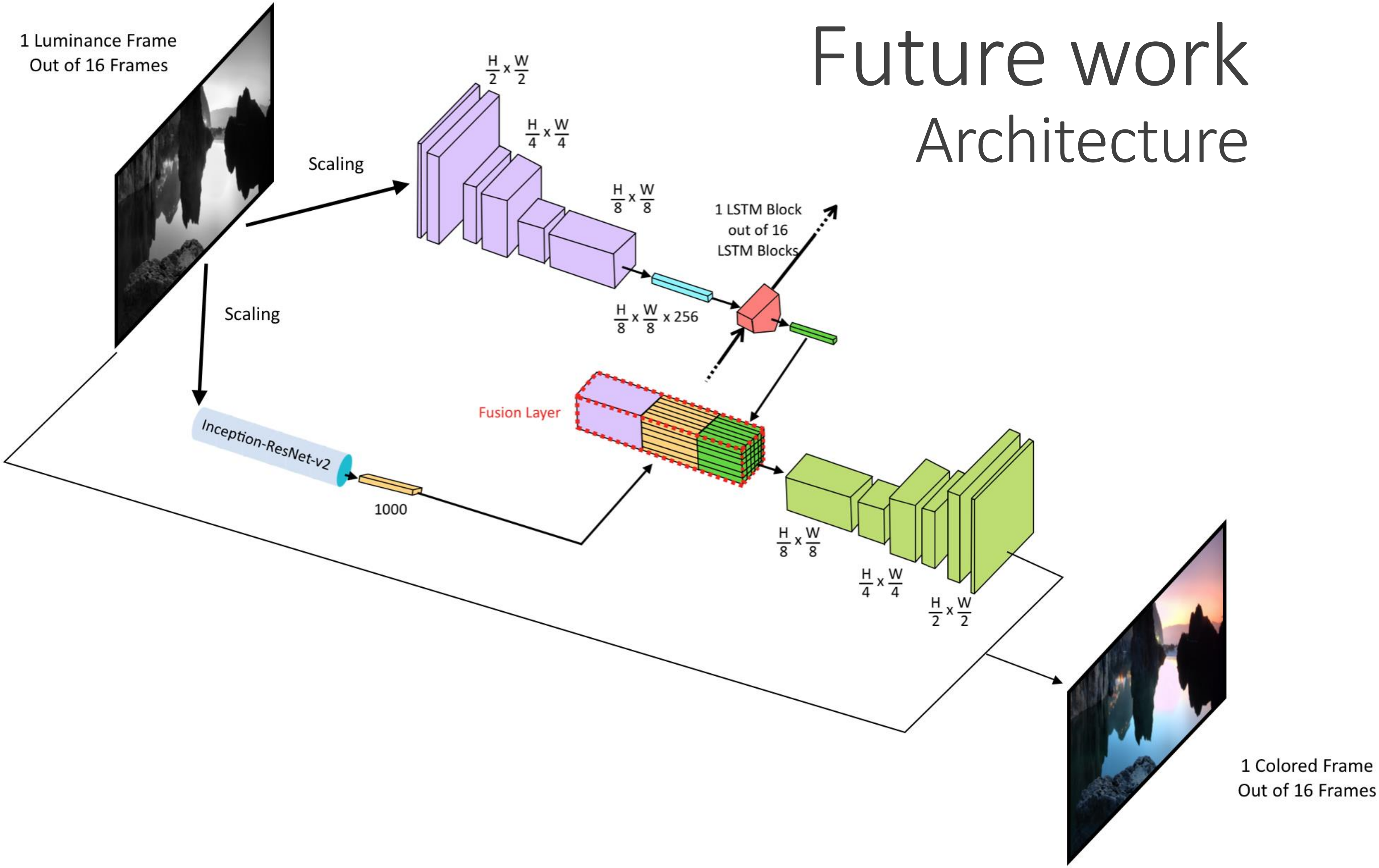
Video colorizing has largely been left behind

Video colorization could be taken as a direct extension of image colorization where we capture a frame as an image and treat it as an image colorization task

"Temporal coherence" is not guaranteed → flickering colors, unusable results



Future work
Architecture



Future work
Architecture

Time distributed CNN encoder

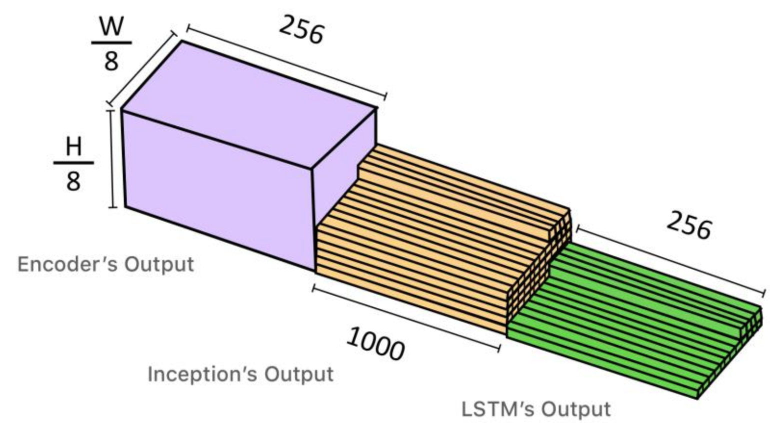
Feature Extractor

LSTM: obtains a global average of the encoder output as input

Fusion: concatenate the repeated feature extractor output and LSTM output

Time distributed CNN decoder

→ Color of a frame depends on the color distribution of the last frames → consistent output



Thank you for your attention!

Q&A