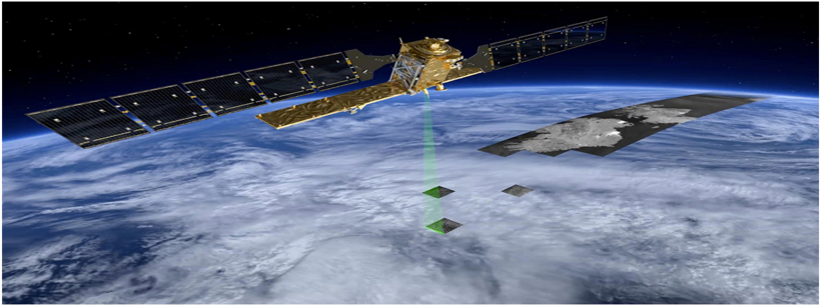
Calcul MATLAB sur GPU

Réunion des utilisateurs du mésocentre MUST - USMB

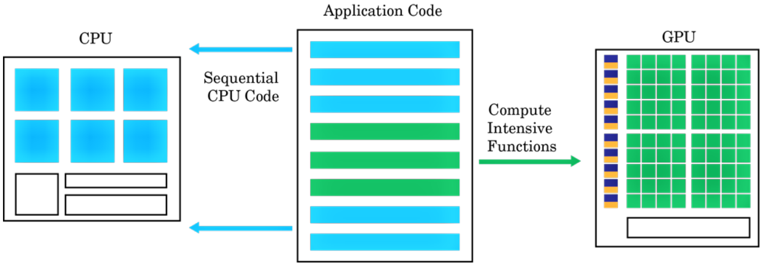
Abdourrahmane M. Atto
Université Savoie Mont Blanc, LISTIC, France

Le 26-06-2019
à l'Auditorium du Laboratoire d'Annecy de Physique des Particules



Calcul à Hautes Performances

Architectures duales (CPU, GPU)



Matlab@Programmeurs

Calculs CUDA@GPU "via" Matlab :

- Conversion (moindre effort) de code Matlab;
- Distribution d'une partie conséquente de ce code sur GPU;
- Accélération des calculs massifs (4x à 20x).

CUDA@Programmeurs

Exploiter l'ergonomie Matlab pour:

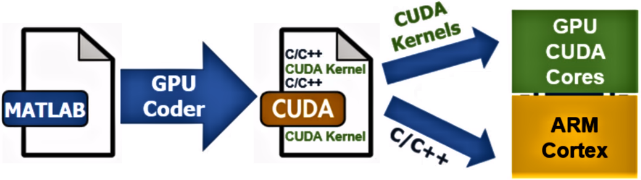
- Évaluer (tests) des noyaux CUDA;
- Explorer les paramètres d'une combinaison de noyaux CUDA.

GPU coder — Optimisation sur les architectures duales (CPU, GPU)

$\mathcal{T}$ = Partitionnement d'un programme séquentiel et génération d'un code CUDA

$$\mathcal{S} = \begin{cases} \text{Exécuter les opérations massivement parallèle sur GPU.} \\ \text{Exécuter les sections séquentielles de codes sur CPU.} \\ \text{Minimiser les coûts de communication/transfert entre CPU et GPU.} \end{cases}$$

GPU Coder $= \mathcal{T}_{\text{Optimal}}$ / Sous contraintes : $\mathcal{S}$



Exemple1@CPU

```
M3D = rand(NbSamples, NbSamples, NbIteration);
M2D = rand(NbSamples,NbSamples);
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

Exemple1@(CPU, GPU)

Adaptation (1)

```
M3D = rand(NbSamples, NbSamples, NbIteration, 'gpuArray');
M2D = rand(NbSamples,NbSamples, 'gpuArray');
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

Adaptation (2)

```
@() pagefun(@mtimes, rand(NbSamples, NbSamples, NbIteration, 'gpuArray'), rand(NbSamples, 'gpuArray'));
```

Exemple2@CPU

```
function u=WaveEquationCPU(u, u_n, h, b, NbIteration)

  for i=1:NbIteration
    v=conv2(u, h, 'same');
    u_temp = 2u - u_n + v - b(u - u_n) ;
    u_n = u;
    u = u_temp;
        ⋮
  end

end
```

Exemple2@(CPU, GPU)

```
function u=WaveEquationGPU(u, u_n, h, b, NbIteration)
  u=gpuArray(u); u_n=gpuArray(u_n); h=gpuArray(h);
  for i=1:NbIteration
    v=conv2(u, h, 'same');
    u_temp = 2u - u_n + v - b(u - u_n) ;
    u_n = u;
    u = u_temp;
        ⋮
  end
u=gather(u);
end
```

Informations sur les cartes GPU *via* Matlab

```
%% Nombre de cartes GPU
>> N = gpuDeviceCount()
      1

%% Informations sur la carte GPU n
>> ContenuCarteGPU = gpuDevice(n)
```

```
%% Selection/Imposer - carte GPU
if N > 1
      gpuDevice(2)
else
      gpuDevice(1)
end
```

MUST-1 @localGPU*

gpuDeviceCountM1	2
Name	'Tesla K80'
ComputeCapability	'3.7'
SupportsDouble	1
DriverVersion	9.1000
ToolkitVersion	8
MaxThreadsPerBlock	1024
MaxShmemPerBlock	49152
MaxThreadBlockSize	[1024 1024 64]
MaxGridSize	[2.1475e+09 65535 65535]
SIMDWidth	32
TotalMemory	1.1997e+10
AvailableMemory	1.1862e+10
MultiprocessorCount	13
ClockRateKHz	82 3500

MUST-2 @localGPU-Cécile

gpuDeviceCountM2	1
Name	'Tesla V100-PCIE-16GB'
ComputeCapability	'7.0'
SupportsDouble	1
DriverVersion	9.2000
ToolkitVersion	9.1000
MaxThreadsPerBlock	1024
MaxShmemPerBlock	49152
MaxThreadBlockSize	[1024 1024 64]
MaxGridSize	[2.1475e+09 65535 65535]
SIMDWidth	32
TotalMemory	1.6946e+10
AvailableMemory	9.2931e+09
MultiprocessorCount	80
ClockRateKHz	1 380 000

NbSamples = 10 (liliputiens) / 5000 (titans)
NbIteration = 10 (liliputiens) / 1000 (titans)

Exemple1@CPU

```
M3D = rand(NbSamples, NbSamples, NbIteration);
M2D = rand(NbSamples,NbSamples);
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

Exemple1@GPU

CPU2GPU Adaptation 1

```
M3D = rand(NbSamples, NbSamples, NbIteration, 'gpuArray');
M2D = rand(NbSamples,NbSamples, 'gpuArray');
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

CPU2GPU Adaptation 2

```
@() pagefun(@mtimes, rand(NbSamples, NbSamples, NbIteration, 'gpuArray'), rand(NbSamples, 'gpuArray'));
```

Exemple2@CPU

function u=WaveEquationCPU(u, u_n, h, NbIteration)
  % Size of wave field = 3000x3000 pixels
  for i=1:NbIteration
    v=conv2(u, h, 'same');
    $u_{\text{temp}} = 2u_n - u + v$;
    $u_n = u$;
    $u = u_{\text{temp}}$;
    $\vdots$
  end
  %%%%%%%%
end

Exemple2@GPU

function u=WaveEquationGPU(u, u_n, h, NbIteration)
  u=gpuArray(u); u_n=gpuArray(u_n); h=gpuArray(h);
  for i=1:NbIteration
    v=conv2(u, h, 'same');
    $u_{\text{temp}} = 2u_n - u + v$;
    $u_n = u$;
    $u = u_{\text{temp}}$;
    $\vdots$
  end
  u=gather(u);
end

NbSamples = 10 (liliputiens) / 5000 (titans)
NbIteration = 10 (liliputiens) / 1000 (titans)

Exemple1@CPU 59.6" [titans]

```
M3D = rand(NbSamples, NbSamples, NbIteration);
M2D = rand(NbSamples,NbSamples);
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

Exemple1@GPU 0.8" [titans]

CPU2GPU Adaptation 1

```
M3D = rand(NbSamples, NbSamples, NbIteration, 'gpuArray');
M2D = rand(NbSamples,NbSamples, 'gpuArray');
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

CPU2GPU Adaptation 2

```
@() pagefun(@mtimes, rand(NbSamples, NbSamples, NbIteration, 'gpuArray'), rand(NbSamples, 'gpuArray'));
```

Exemple2@CPU 53.0" [titans]

```
function u=WaveEquationCPU(u, u_n, h, NbIteration)
  % Size of wave field = 3000x3000 pixels
  for i=1:NbIteration
    v=conv2(u, h, 'same');
    u_temp = 2u_n − u + v ;
    u_n = u;
    u = u_temp;
      ⋮
  end
  %%%%%%%
end
```

Exemple2@GPU 1.6" [titans]

```
function u=WaveEquationGPU(u, u_n, h, NbIteration)
  u=gpuArray(u); u_n=gpuArray(u_n); h=gpuArray(h);
  for i=1:NbIteration
    v=conv2(u, h, 'same');
    u_temp = 2u_n − u + v ;
    u_n = u;
    u = u_temp;
      ⋮
  end
  u=gather(u);
end
```

NbSamples = 10 (liluputiens) / 5000 (titans)
NbIteration = 10 (liluputiens) / 1000 (titans)

Exemple1@CPU 5.56" [liluput!s]

```
M3D = rand(NbSamples, NbSamples, NbIteration);
M2D = rand(NbSamples,NbSamples);
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

Exemple1@GPU 0.01" [liluput!s]

CPU2GPU Adaptation 1

```
M3D = rand(NbSamples, NbSamples, NbIteration, 'gpuArray');
M2D = rand(NbSamples,NbSamples, 'gpuArray');
parfor/for idx = 1:NbIteration
    mtimes(squeeze(M3D(:,:,idx)),M2D);
end
```

CPU2GPU Adaptation 2

```
@() pagefun(@mtimes, rand(NbSamples, NbSamples, NbIteration, 'gpuArray'), rand(NbSamples, 'gpuArray'));
```

Exemple2@CPU 0.1" [liluput!s]

function u=WaveEquationCPU(u, u_n, h, NbIteration)
% Size of wave field = 3000x3000 pixels
for i=1:NbIteration
v=conv2(u, h, 'same');
$u_{\text{temp}} = 2u_n - u + v$;
$u_n = u$;
$u = u_{\text{temp}}$;
⋮
end
%%%%%%%
end

Exemple2@GPU 0.5" [liluput!s]

function u=WaveEquationGPU(u, u_n, h, NbIteration)
u=gpuArray(u); u_n=gpuArray(u_n); h=gpuArray(h);
for i=1:NbIteration
v=conv2(u, h, 'same');
$u_{\text{temp}} = 2u_n - u + v$;
$u_n = u$;
$u = u_{\text{temp}}$;
⋮
end
u=gather(u);
end

MUST-1 @localGPU*

gpuDeviceCountM1	2
Name	'Tesla K80'
ComputeCapability	'3.7'
SupportsDouble	1
DriverVersion	9.1000
ToolkitVersion	8
MaxThreadsPerBlock	1024
MaxShmemPerBlock	49152
MaxThreadBlockSize	[1024 1024 64]
MaxGridSize	[2.1475e+09 65535 65535]
SIMDWidth	32
TotalMemory	1.1997e+10
AvailableMemory	1.1862e+10
MultiprocessorCount	13
ClockRateKHz	82 3500

MUST-2 @localGPU-Cécile

gpuDeviceCountM2	1
Name	'Tesla V100-PCIE-16GB'
ComputeCapability	'7.0'
SupportsDouble	1
DriverVersion	9.2000
ToolkitVersion	9.1000
MaxThreadsPerBlock	1024
MaxShmemPerBlock	49152
MaxThreadBlockSize	[1024 1024 64]
MaxGridSize	[2.1475e+09 65535 65535]
SIMDWidth	32
TotalMemory	1.6946e+10
AvailableMemory	9.2931e+09
MultiprocessorCount	80
ClockRateKHz	1 380 000

MUST-1@K80 @localGPU*

MTimes-Rand-3Dx2D: 9.460067

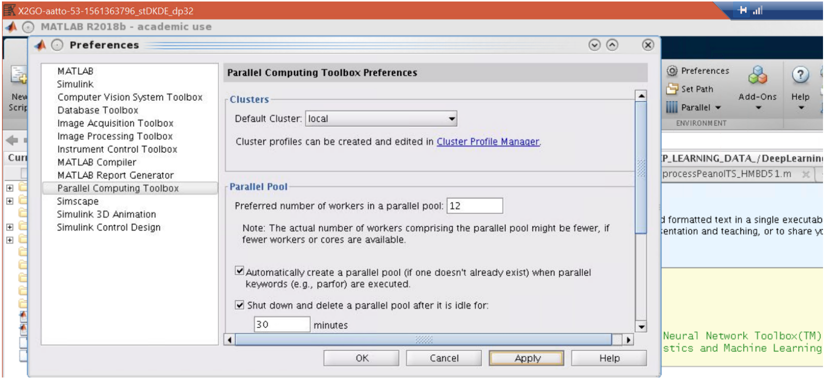
Wave-Propagation-Equation: 4.364735

MUST-2@V100 @localGPU-Cécile

MTimes-Rand-3Dx2D: 0.833975

Wave-Propagation-Equation: 1.632387

Si pool CPU plus conséquent Matlab@GPU ↗



Versions Matlab / CUDA Toolkit / $\exists_{\text{ance}}$ Matlab-Built-In@GPU

Error In DeepSegmentLearningGPU (line 176) → MultivariateKernelDensity

There is a problem with the graphics driver or with this GPU device.
Be sure that you have a supported GPU and that the latest driver is installed.

‹ Documentation Home

‹ Parallel Computing Toolbox
‹ GPU Computing
‹ GPU Computing in MATLAB

‹ Parallel Computing Toolbox
‹ GPU Computing

Run Built-In Functions on a GPU

ON THIS PAGE

MATLAB Functions with gpuArray Arguments

Example: Functions with gpuArray Input and Output

Sparse Arrays on a GPU

Considerations for Complex Numbers

Acknowledgments

See Also

MATLAB Functions with gpuArray Arguments

Many MATLAB® built-in functions support `gpuArray` input arguments. Whenever any of these functions is called with at least one gpuArray as an input argument, the function executes on the GPU and generates
same function call; the MATLAB arrays are transferred to the GPU for the function execution. Supporting functions include the discrete Fourier transform (`fft`), matrix multiplication (`mtimes`), and left matrix divisio

The following functions and their symbol operators are enhanced to accept gpuArray input arguments so that they execute on the GPU:

abs	compan	flip	isnan
acos	complex	fliplr	isnumeric
acosd	cond	flipud	isreal
acosh	conj	floor	isrow
acot	conv	fprintf	issorted
acotd	conv2	full	issparse
acoth	convn	gamma	issymmetric
acsc	corrcoef	gammainc	istril
acscd	cos	gammaincinv	istriu
acsch	cosd	gammaln	isvector
accumarray	cosh	gather	kron
all	cot	ge	ldivide
and	cotd	gmres	le
angle	coth	gradient	legendre
any	cov	gt	length
arrayfun	cross	hankel	log
asec	csc	head	log10
asecd	cscd	histcounts	log1p
asech	csch	horzcat	log2
asin	ctranspose	hsv2rgb	logical
asind	cummax	hypot	lsqr
asinh	cummin	idivide	lt
assert	cumprod	ifft	lu
atan	cumsum	ifft2	mat2str
atan2	deg2rad	ifftn	max
atan2d	del2	ifftshift	median
atand	det	imag	mean
atanh	detrend	ind2sub	meshgrid
bandwidth	diag	Inf	min
besselj	diff	inpolygon	minus
bessely	discretize	int16	mldivide
beta	disp	int2str	mod
betainc	display	int32	mode
betaincinv	dot	int64	movmean
betaln	double	int8	movstd
bicg	eig	interp1	movsum
bicgstab	eps	interp2	movvar
bitand	eq	interp3	mpower
bitcmp	erf	interpn	mrdivide
bitget	erfc	intersect	mtimes
bitor	erfcinv	inv	NaN

Set options for training

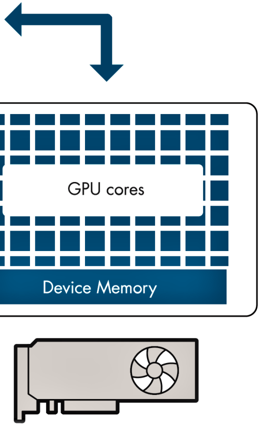
```
opts = trainingOptions('sgdm');
```

Train the network

```
net = trainNetwork ...
(XTrain, TTrain, layers, opts);
```

Make predictions

```
trainFeatures = ...
activations(net, XTrain, 6);
```




```
layers = [ imageInputLayer([512 512 3])
       convolution2dLayer([3 3],96);
           ...
       softmaxLayer()
       pixelClassificationLayer() ]
```

```
opts = trainingOptions('sgdm',...
       'ExecutionEnvironment', 'multi-gpu');
```

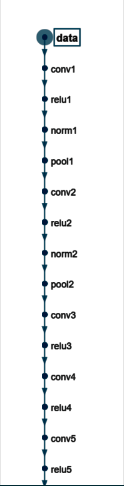
```
>> net = alexnet;                    >> analyzeNetwork(net)
```

Deep Learning Network Analyzer

net

Analysis date: 25-Jun-2019 21:59:33

25 layers · 0 warnings · 0 errors



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	data 227x227x3 images with 'zerocenter' normalization	Image Input	227×227×3	-
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55×55×96	Weights 11×11×3×96 Bias 1×1×96
3	relu1 ReLU	ReLU	55×55×96	-
4	norm1 cross channel normalization with 5 channels per element	Cross Channel Normalization	55×55×96	-
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27×27×96	-
6	conv2 256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	27×27×256	Weights 5×5×48×256 Bias 1×1×256
7	relu2 ReLU	ReLU	27×27×256	-
8	norm2 cross channel normalization with 5 channels per element	Cross Channel Normalization	27×27×256	-
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13×13×256	-
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×384	Weights 3×3×256×384 Bias 1×1×384
11	relu3 ReLU	ReLU	13×13×384	-
12	conv4 384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×384	Weights 3×3×192×384 Bias 1×1×384
13	relu4 ReLU	ReLU	13×13×384	-
14	conv5 256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×256	Weights 3×3×192×256 Bias 1×1×256
15	relu5	ReLU	13×13×256	-

LISTIC

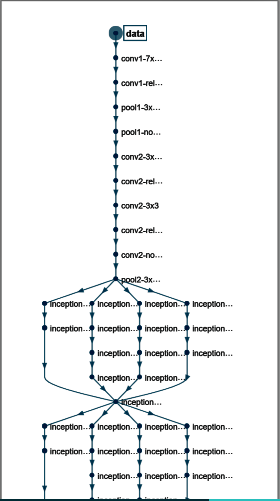
```
>> net = googlenet;                    >> analyzeNetwork(net)
```

Deep Learning Network Analyzer

net

Analysis date: 25-Jun-2019 21:59:33

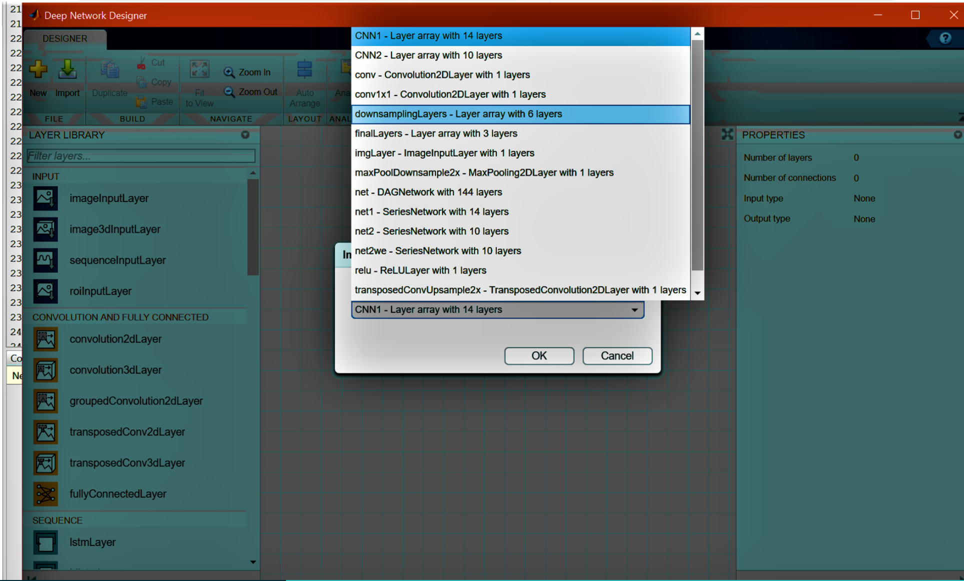
144 layers



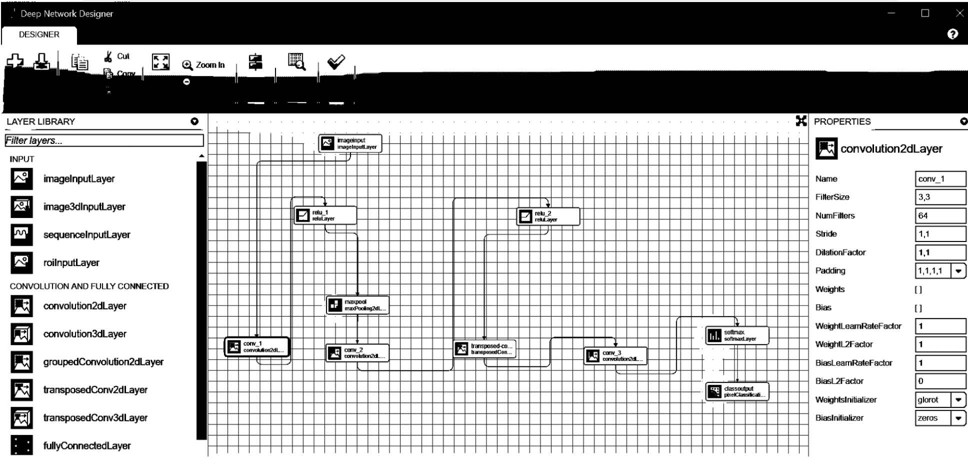
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABL
1	data 224x224x3 images with 'zerocenter' normalization	Image Input	224×224×3	-
2	conv1-7x7_s2 64 7x7x3 convolutions with stride [2 2] and padding [3 3 3 3]	Convolution	112×112×64	Weights Bias
3	conv1-relu_7x7 ReLU	ReLU	112×112×64	-
4	pool1-3x3_s2 3x3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	56×56×64	-
5	pool1-norm1 cross channel normalization with 5 channels per element	Cross Channel Normalization	56×56×64	-
6	conv2-3x3_reduce 64 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	56×56×64	Weights Bias
7	conv2-relu_3x3_reduce ReLU	ReLU	56×56×64	-
8	conv2-3x3 192 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	56×56×192	Weights Bias
9	conv2-relu_3x3 ReLU	ReLU	56×56×192	-
10	conv2-norm2 cross channel normalization with 5 channels per element	Cross Channel Normalization	56×56×192	-
11	pool2-3x3_s2 3x3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	28×28×192	-
12	inception_3a-1x1 64 1x1x192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×64	Weights Bias
13	inception_3a-relu_1x1 ReLU	ReLU	28×28×64	-
14	inception_3a-3x3_reduce 96 1x1x192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×96	Weights Bias
15	inception_3a-relu_3x3_reduce ReLU	ReLU	28×28×96	-
16	inception_3a-3x3 128 3x3x96 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	28×28×128	Weights Bias
17	inception_3a-relu_3x3 ReLU	ReLU	28×28×128	-

Outil pour DL Designer >> deepNetworkDesigner



>> deepNetworkDesigner — Une histoire de *chemins*



Explorer les chemins le jour chasse la peur de se promener la nuit.

Aller là où il n'y a pas de chemin et laissez une trace (RWE).

Le chemin se construit en marchant (AM). La bonne volonté raccourci le chemin (BR).

Si vous ne savez pas où vous allez, n'importe quel chemin vous y mènera (LC).

La route fût longue mais le chemin est beau (DPY).

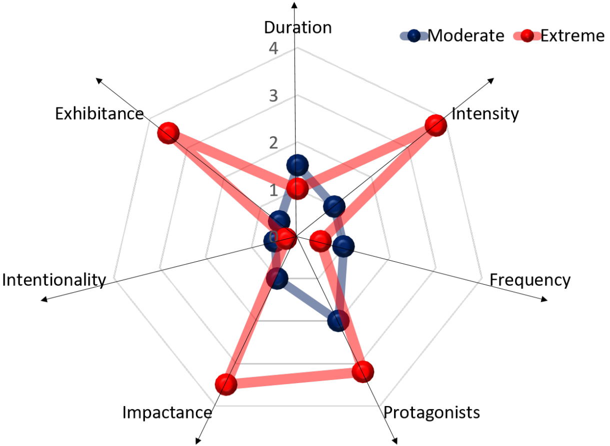
Si tous les chemins mènent à l'IA, les Krees sont une facette de l'homo-3.0 au sens du Suprémor.

Application DL pour la détection de violence visuelle par analyse de vidéos

[1] A. M. Atto, A. Benoit, P. Lambert,
Hilbert Based Video To Timed-Image Filling and Learning To Recognize Visual Violence
`https://dx.doi.org/10.13140/RG.2.2.12648.11526/1`

	Observables	Examples
Violence from interactions (external induction)		
1	Human/Human	Battle, slap, punch.
2	Human/Object	Surgery, mutilation, projectile throw.
3	Human/Fluid	Drowning, gazing.
4	Human/Animal	Human attack by animals, animal hunting and shoot.
5	Object/Object	Car *versus* car accidents, crash of an airplane (ground or sea).
6	Animal/Animal	Animals fight clubs, predator *versus* prey showdowns.
Suspicious motions (self-induction)		
7	Living body abnormal motion	Terror or aggressive faces, person falling down
8	Inert structure abnormal motion	Conflagrations, explosion, smoke, flames and ashes, flowing blood.
Sensitive objects and symbols		
9	Sensitive objects	Guns, weapons, bombs, broken glasses, stripped electrical socket, blood stain frightful masks and veils.
10	Sensitive symbols	PEGI "-18" / "-16" / "-12" / "-10", flammable material signs, swastika injury and hatred posters

Name	Categories and number of video samples		
VSD-L2	Violence 1 137		Non-Violence 10398
VSD-L3	Moderate Violence 406	Extreme Violence 418	Non-Violence 10398



Deep-learning / MUST VSD-L2

3D CNN

Category	*Non – Violence*	*Violence*
Non – Violence	94 %	06 %
Violence	78 %	22 %
Mean accuracy	58 %	

2D-Timed-Image CNN

Category	*Non – Violence*	*Violence*
Non – Violence	94 %	6 %
Violence	36 %	64 %
Mean accuracy	79 %	

Information détaillées disponibles dans [1] :
Hilbert Based Video To Timed-Image Filling and Learning To Recognize Visual Violence