Deep Learning Optimisé - Jean Zay

Compétition

cnrs

INSTITUT DU
DÉVELOPPEMENT ET DES
RESSOURCES EN
INFORMATIQUE
SCIENTIFIQUE

Hypers paramètres - Références - État de l’art

MLPerf – Resnet50

NVIDIA A100 (MXNET, DA with DALI)

- For training: Normalization, Scale to 256x256, Random resized crop to 224x224, Random horizontal flip
- For validation: Normalization, Scale to 256x256, Center crop to 224x224

	Num epochs	Warmup epochs	Lr scheduler	Global Batch size	Optimizer	LR	WD	mom	Label smoothing
REF (TF)	**41**	5	polynomial	**2048**	lars	**8.5**	2e-4	0.9	0.1
2 GPU	**37**	2	polynomial	**408**	sgdwfastlars	**3.0**	5e-5	0.9	0.1
8 GPU	**37**	2	polynomial	**3264**	sgdwfastlars	**10.5**	5e-5	0.9	0.1
64 GPU	**37**	2	polynomial	**3264**	sgdwfastlars	**10.5**	5e-5	0.9	0.1
1024 GPU	**60**	17	polynomial	**35840**	sgdwfastlars	**21.9**	2.5e-5	0.94	0.1
4216 GPU	**90**	31	polynomial	**67456**	sgdwfastlarsv2	**24.699**	1e-4	0.951807	0.1

Hypers paramètres et Ingrédients pour Resnet-50

Source : ResNet strikes back: An improved training procedure in timm

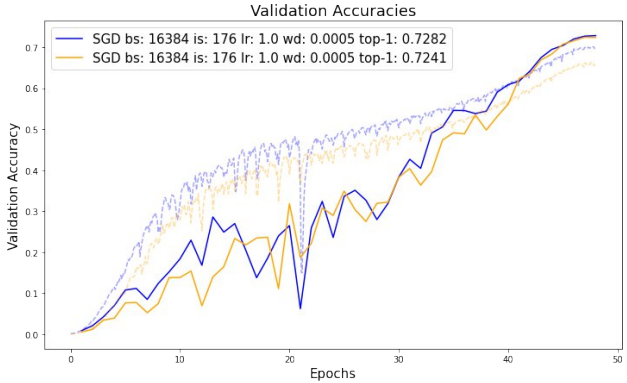
Table 2: Ingredients and hyper-parameters used for ResNet-50 training in different papers. We compare existing training procedures with ours.

	Previous approaches					Ours		
Procedure → Reference	ResNet [13]	PyTorch [1]	FixRes [48]	DeiT [45]	FAMS (×4) [10]	A1	A2	A3
Train Res	224	224	224	224	224	224	224	160
Test Res	224	224	224	224	224	224	224	224
Epochs	90	90	120	300	400	600	300	100
# of forward pass	450k	450k	300k	375k	500k	375k	188k	63k
Batch size	256	256	512	1024	1024	2048	2048	2048
Optimizer	SGD-M	SGD-M	SGD-M	AdamW	SGD-M	LAMB	LAMB	LAMB
LR	0.1	0.1	0.2	1×10^{-3}	2.0	5×10^{-3}	5×10^{-3}	8×10^{-3}
LR decay	step	step	step	cosine	step	cosine	cosine	cosine
decay rate	0.1	0.1	0.1	-	$0.02^{t/400}$	-	-	-
decay epochs	30	30	30	-	1	-	-	-
Weight decay	10^{-4}	10^{-4}	10^{-4}	0.05	10^{-4}	0.01	0.02	0.02
Warmup epochs	✗	✗	✗	5	5	5	5	5
Label smoothing ε	✗	✗	✗	0.1	0.1	0.1	✗	✗
Dropout	✗	✗	✗	✗	✗	✗	✗	✗
Stoch. Depth	✗	✗	✗	0.1	✗	0.05	0.05	✗
Repeated Aug	✗	✗	✓	✓	✗	✓	✓	✗
Gradient Clip.	✗	✗	✗	✗	✗	✗	✗	✗
H. flip	✓	✓	✓	✓	✓	✓	✓	✓
RRC	✗	✓	✓	✓	✓	✓	✓	✓
Rand Augment	✗	✗	✗	9/0.5	✗	7/0.5	7/0.5	6/0.5
Auto Augment	✗	✗	✗	✗	✓	✗	✗	✗
Mixup alpha	✗	✗	✗	0.8	0.2	0.2	0.1	0.1
Cutmix alpha	✗	✗	✗	1.0	✗	1.0	1.0	1.0
Erasing prob.	✗	✗	✗	0.25	✗	✗	✗	✗
ColorJitter	✗	✓	✓	✗	✗	✗	✗	✗
PCA lighting	✓	✗	✗	✗	✗	✗	✗	✗
SWA	✗	✗	✗	✗	✓	✗	✗	✗
EMA	✗	✗	✗	✗	✗	✗	✗	✗
Test crop ratio	0.875	0.875	0.875	0.875	0.875	0.95	0.95	0.95
CE loss	✓	✓	✓	✓	✓	✗	✗	✗
BCE loss	✗	✗	✗	✗	✗	✓	✓	✓
Mixed precision	✗	✗	✗	✓	✓	✓	✓	✓
Top-1 acc.	75.3%	76.1%	77.0%	78.4%	79.5%	80.4%	79.8%	78.1%

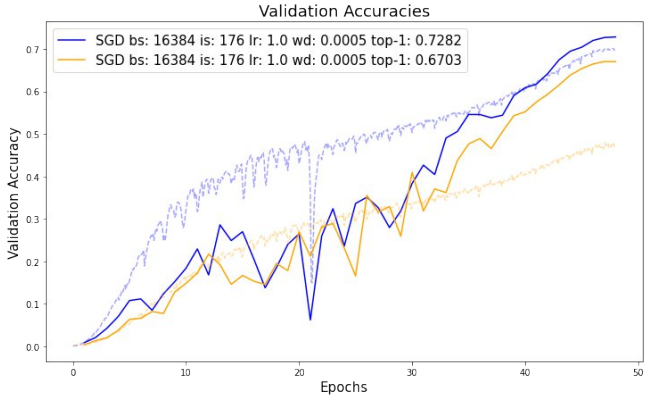
Data Augmentation

Attention à l'augmentation !!

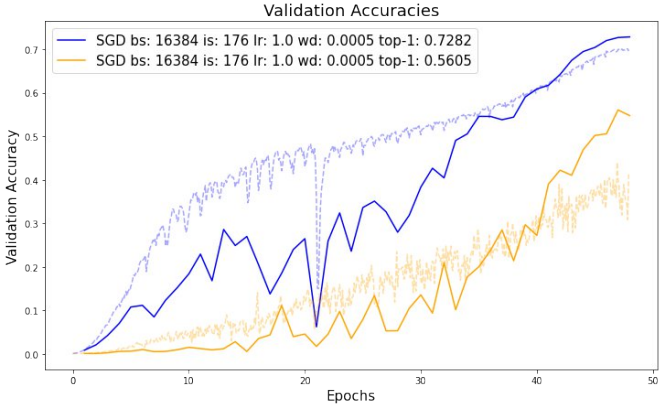
RandAugment



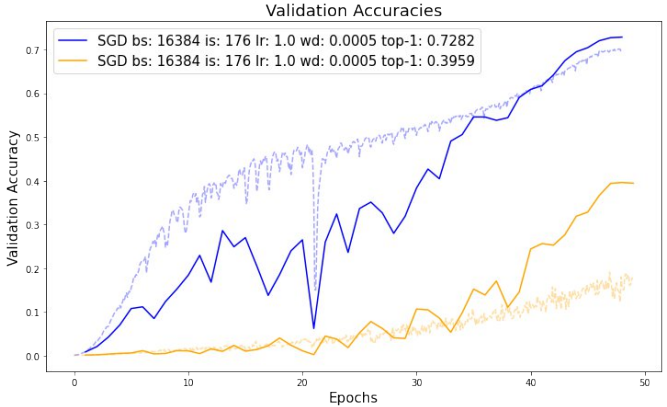
MixUp



CutMix



RandA. + MixUp + CutMix



Zoo de Modèles Imagenet

Modèles Torchvision

```
'alexnet',
'convnext_tiny',
'convnext_small',
'convnext_base',
'convnext_large',
'resnet18',
'resnet
'resnet50',
'resnet101',
'resnet152',
'resnext50_32x4d',
'resnext101_32x8d',
'wide_resnet50_2',
'wide_resnet101_2',
```

```
'vgg11',
'vgg11_bn',
'vgg13',
'vgg13_bn',
'vgg16',
'vgg16_bn',
'squeezenet1_0',
'squeezenet1_1',
'inception_v3',
'densenet121',
'densenet169',
'densenet201',
'densenet161',
```

```
'googlenet',
'mobilenet_v2',
'mobilenet_v3_large',
'mobilenet_v3_small',
'mnasnet0_5',
'mnasnet0_75',
'mnasnet1_0',
'mnasnet1_3',
'shufflenet_v2_x0_5',
'shufflenet_v2_x1_5',
'shufflenet_v2_x2_0',
'efficientnet_b0',
'efficientnet_b1',
'efficientnet_b2',
'efficientnet_b3',
'efficientnet_b4',
'efficientnet_b5',
'efficientnet_b6',
'efficientnet_b7',
```

https://pytorch.org/vision/main/models.html#classification

```
'regnet_y_400mf',
'regnet_y_800mf',
'regnet_y_1_6gf',
'regnet_y_3_2gf',
'regnet_y_8gf',
'regnet_y_16gf',
'regnet_y_32gf',
'regnet_y_128gf',
'regnet_x_400mf',
'regnet_x_800mf',
'regnet_x_1_6gf',
'regnet_x_3_2gf',
'regnet_x_8gf',
'regnet_x_16gf',
'regnet_x_32gf',
'vit_b_16',
'vit_b_32',
'vit_l_16',
'vit_l_32'
```

Modèles resnet TIMM

```
>>> for m in [m for m in timm.list_models() if 'resnet' in
m.lower()]: print(m)
...
cspresnet50
cspresnet50d
cspresnet50w
ecaresnet26t
ecaresnet50d
ecaresnet50d_pruned
ecaresnet50t
ecaresnet101d
ecaresnet101d_pruned
ecaresnet200d
ecaresnet269d
ecaresnetlight
ens_adv_inception_resnet_v2
gcresnet50t
geresnet50t
gluon_resnet18_v1b
gluon_resnet34_v1b
gluon_resnet50_v1b
gluon_resnet50_v1c
gluon_resnet50_v1d
gluon_resnet50_v1s
gluon_resnet101_v1b
gluon_resnet101_v1c
gluon_resnet101_v1d
gluon_resnet101_v1s
gluon_resnet152_v1b
gluon_resnet152_v1c
gluon_resnet152_v1d
gluon_resnet152_v1s
inception_resnet_v2
lambda_resnet26t
lambda_resnet50t
legacy_seresnet18
legacy_seresnet34
legacy_seresnet50
legacy_seresnet101
legacy_seresnet152
nf_ecaresnet26
nf_ecaresnet50
nf_ecaresnet101
nf_resnet26
nf_resnet50
nf_resnet101
nf_seresnet26
nf_seresnet50
resnet26
resnet26d
resnet26t
resnet34
resnet34d
resnet50
resnet50d
resnet50t
resnet51q
resnet61q
resnet101
resnet101d
resnet152
resnet152d
resnet200
resnet200d
iresnetblur18
resnetblur50
resnetrs50
resnetrs101
resnetrs152
resnetrs200
resnetrs270
resnetrs350
resnetrs420
resnetv2_50
resnetv2_50d
resnetv2_50t
resnetv2_50x1_bit_distilled
resnetv2_50x1_bitm
resnetv2_50x1_bitm_in21k
resnetv2_101d
resnetv2_101x1_bitm
resnetv2_101x1_bitm_in21k
resnetv2_101x3_bitm
resnetv2_101x3_bitm_in21k
resnetv2_152
resnetv2_152d
resnetv2_152x2_bit_teacher
resnetv2_152x2_bit_teacher_384
resnetv2_152x2_bitm
resnetv2_152x2_bitm_in21k
resnetv2_152x4_bitm
resnetv2_152x4_bitm_in21k
seresnet18
seresnet34
seresnet50
seresnet50t
seresnet101
seresnet152
seresnet152d
seresnet200d
seresnet269d
skresnet18
skresnet34
skresnet50
skresnet50d
ssl_resnet18
ssl_resnet50
swsl_resnet18
swsl_resnet50
tresnet_l
tresnet_l_448
tresnet_m
tresnet_m_448
tresnet_m_miil_in21k
tresnet_xl
tresnet_xl_448
tv_resnet34
tv_resnet50
tv_resnet101
tv_resnet152
vit_base_resnet26d_224
vit_base_resnet50_224_in21k
vit_base_resnet50_384
vit_base_resnet50d_224
vit_small_resnet26d_224
vit_small_resnet50d_s16_224
wide_resnet50_2
wide_resnet101_2
```

https://paperswithcode.com/lib/timm

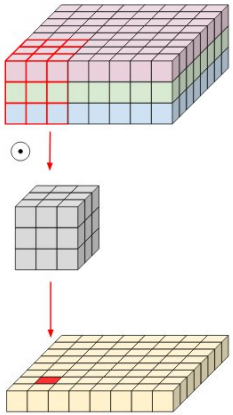
Modèles

Source : ResNet strikes back: An improved training procedure in timm

Architecture	# params $\times 10^6$	FLOPs $\times 10^9$	Throughput (im/s)	Peak mem (MB)	Top-1 Acc.	Real Acc.	V2 Acc.
ResNet-18 [13]	11.7	1.8	7960.5	588	71.5	79.4	59.4
ResNet-34 [13]	21.8	3.7	4862.6	642	76.4	83.4	65.1
ResNet-50 [13]	25.6	4.1	2536.6	1,155	80.4	85.7	68.7
ResNet-101 [13]	44.5	7.9	1547.9	1,264	81.5	86.3	70.3
ResNet-152 [13]	60.2	11.6	1094.0	1,355	82.0	86.4	70.6
RegNetY-4GF [32]	20.6	4.0	1690.6	1,585	81.5	86.7	70.7
RegNetY-8GF [32]	39.2	8.1	1122.3	2,139	82.2	86.7	71.1
RegNetY-16GF [32]	83.6	16.0	694.1	3,052	82.0	86.4	71.2
RegNetY-32GF [32]	145.0	32.4	431.5	3,366	82.5	86.6	71.7
SE-ResNet-50 [20]	28.1	4.1	2174.8	1,193	80.0	85.8	68.8
SENet-154 [20]	115.1	20.9	511.5	2,414	81.7	86.0	71.2
ResNet-50-D [14]	25.6	4.4	2418.8	1,205	80.7	85.9	68.9
ResNeXt-50-32x4d [51]	25.0	4.3	1727.5	1,247	80.5	85.5	68.4
EfficientNet-B0 [41]	5.3	0.4	3701.5	932	77.0	83.8	65.0
EfficientNet-B1 [41]	7.8	0.7	2365.2	1,077	79.2	85.3	67.7
EfficientNet-B2 [41]	9.2	1.0	1786.8	1,318	80.4	86.0	69.3
EfficientNet-B3 [41]	12.0	1.8	1082.4	2,447	81.4	86.7	70.4
EfficientNet-B4 [41]	19.0	4.2	561.3	5,058	81.6	85.9	70.8
ViT-Ti [45]	5.7	1.3	3497.7	346	74.7	82.1	62.4
ViT-S [45]	22.0	4.6	1762.3	682	80.6	85.6	69.4
ViT-B [11]	86.6	17.6	771.0	1,544	80.4	84.8	69.4
timm [50] specific architectures							
ECA-ResNet50-T	25.6	4.4	2139.7	1,155	81.3	86.1	69.9
EfficientNetV2-rw-S [42]	23.9	8.8	823.1	2,339	80.6	84.8	69.2
EfficientNetV2-rw-M [42]	53.2	18.5	456.8	2,916	82.3	87.1	71.7
ECA-Resnet269-D	102.1	70.6	168.1	4,134	83.3	86.9	71.9

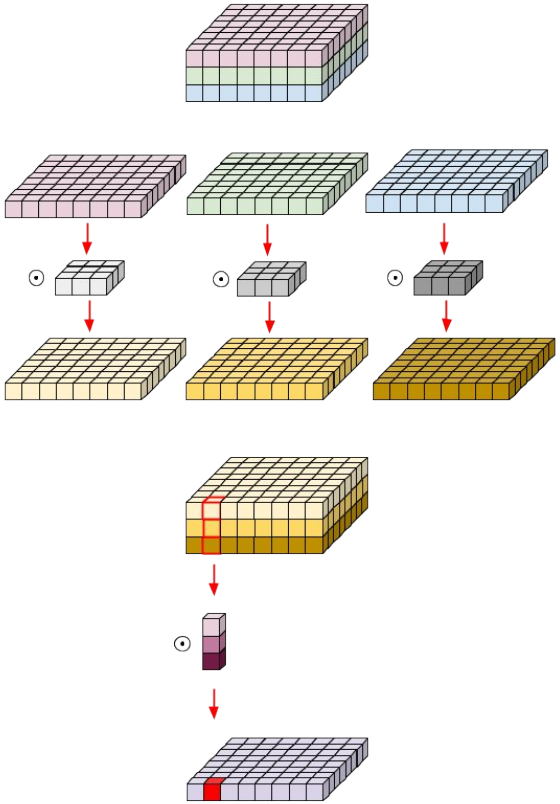
	A1-A2-org.		A3		Cost							ImageNet-1k-val			
	train	test	train	test	A1	A2	A1-A2		A3			A1	A2	A3	org.
↓ Architecture	res.	res.	res.	res.	time (hour)		# GPU	Pmem	time	# GPU	Pmem	Accuracy(%)			
ResNet-18 [13]†	224	224	160	224	186	93	2	12.5	28	2	6.5	71.5	70.6	68.2	69.8
ResNet-34 [13]†	224	224	160	224	186	93	2	17.5	27	2	9.0	76.4	75.5	73.0	73.3
ResNet-50 [13]†	224	224	160	224	110	55	4	22.0	15	4	11.4	80.4	79.8	78.1	76.1
ResNet-101 [13]†	224	224	160	224	74	37	8	16.3	8	8	8.5	81.5	81.3	79.8	77.4
ResNet-152 [13]†	224	224	160	224	92	46	8	22.5	9	8	11.8	82.0	81.8	80.6	78.3
RegNetY-4GF [32]	224	224	160	224	130	65	4	27.1	15	4	13.9	81.5	81.3	79.0	79.4
RegNetY-8GF [32]	224	224	160	224	106	53	8	19.8	10	8	10.3	82.2	82.1	81.1	79.9
RegNetY-16GF [32]	224	224	160	224	150	75	8	25.6	13	8	13.4	82.0	82.2	81.7	80.4
RegNetY-32GF [32]	224	224	160	224	120	60	16	17.6	12	16	9.4	82.5	82.4	82.6	81.0
SE-ResNet-50 [20]	224	224	160	224	102	51	4	27.6	16	4	14.2	80.0	80.1	77.0	76.7
SENet-154 [20]	224	224	160	224	110	55	16	23.3	12	16	12.2	81.7	81.8	81.9	81.3
ResNet-50-D [14]	224	224	160	224	100	50	4	23.9	14	4	12.3	80.7	80.2	78.7	79.3
ResNeXt-50-32x4d [51]†	224	224	160	224	80	40	8	14.3	15	4	14.6	80.5	80.4	79.2	77.6
EfficientNet-B0 [41]	224	224	160	224	110	55	4	22.1	15	4	11.4	77.0	76.8	73.0	77.1
EfficientNet-B1 [41]	240	240	160	224	62	31	8	17.9	8	8	7.9	79.2	79.4	74.9	79.1
EfficientNet-B2 [41]	260	260	192	256	76	38	8	22.8	9	8	11.9	80.4	80.1	77.5	80.1
EfficientNet-B3 [41]	300	300	224	288	62	31	16	19.5	6	16	10.1	81.4	81.4	79.2	81.6
EfficientNet-B4 [41]	380	380	320	380	64	32	32	20.4	8	32	14.3	81.6	82.4	81.2	82.9
ViT-Ti [45]*	224	224	160	224	98	49	4	16.3	14	4	7.0	74.7	74.1	66.7	72.2
ViT-S [45]*	224	224	160	224	68	34	8	16.1	8	8	7.0	80.6	79.6	73.8	79.8
ViT-B [11]*	224	224	160	224	66	33	16	16.4	5	16	7.3	80.4	79.8	76.0	81.8
timm [50] specific architectures															
ECA-ResNet-50-T	224	224	160	224	112	56	4	29.3	15	4	15.0	81.3	80.9	79.6	-
EfficientNetV2-rw-S [42]	288	384	224	288	52	26	16	16.6	7	16	10.1	82.3	82.9	80.9	83.8
EfficientNetV2-rw-M [42]	320	384	256	352	64	32	32	18.5	9	32	12.1	80.6	81.9	82.3	84.8
ECA-ResNet-269-D	320	416	256	320	108	54	32	27.4	11	32	17.8	83.3	83.9	83.3	85.0

Les GPU n’aiment pas la Depth-wise conv’



Normal convolution

≃



Depth-wise separable convolution

MobileNet
EfficientNet

Moins de FLOP pour un résultat identique : devrait être plus efficace computationnellement.

Mais provoque un mouvement de données plus important : **inefficace sur GPU**.

Tuning de Modèle

Fonction d'activation Mish

Source - Mish: A self regularized Non-Monotic Activation Function

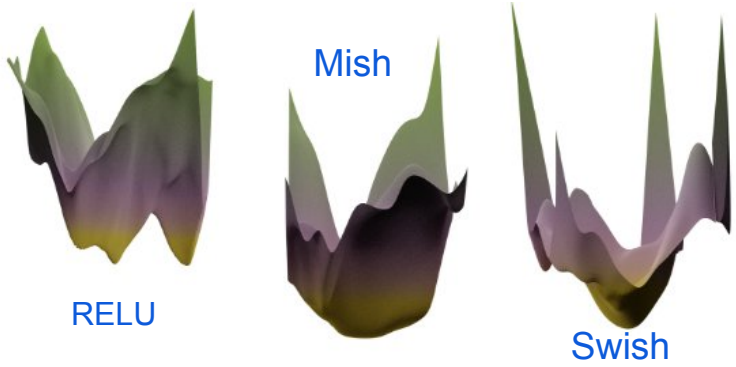
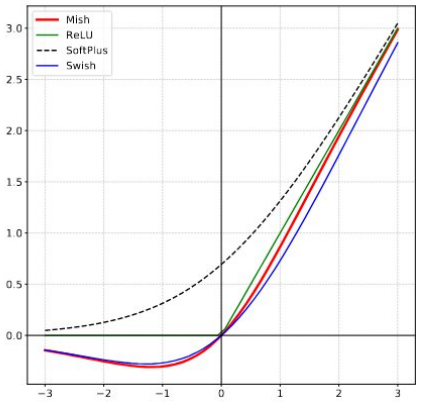
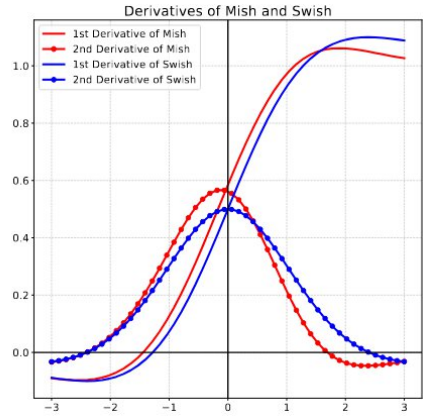


Figure 4: Comparison between the loss landscapes of (from left to right): (a) ReLU, (b) Mish and (c) Swish activation function for a ResNet-20 trained for 200 epochs on CIFAR-10.

$$f(x) = x\tanh(softplus(x)) = x\tanh(\ln(1+e^x))$$

```
model = ....
def convert_relu_to_mish(model):
    for child_name, child in model.named_children():
        if isinstance(child, torch.nn.ReLU):
            setattr(model, child_name, torch.nn.Mish())
        else:
            convert_relu_to_mish(child)
convert_relu_to_mish(model)
model = model.to(gpu)
```





Model	Data Augmentation	LReLU/ ReLU †		Swish		Mish	
		Top-1	Top- 5	Top-1	Top- 5	Top-1	Top- 5
ResNet-18 [15]	No	69.8%†	89.1%†	**71.2%**	**90.1%**	**71.2%**	89.9%
ResNet-50 [15]	No	75.2%†	92.6%†	75.9%	92.8%	**76.1%**	**92.8%**
SpineNet-49 [8]	Yes	77.0%†	93.3%†	78.1%	94%	**78.3%**	**94.6%**
PeleeNet [46]	No	70.7%	90.0%	**71.5%**	**90.7%**	71.4%	90.4%
CSP-ResNet-50 [45]	Yes	77.1%	94.1%	-	-	**78.1%**	**94.2%**
CSP-DarkNet-53 [2]	Yes	77.8%	94.4%	-	-	**78.7%**	**94.8%**
CSP-ResNext-50 [45]	No	77.9%	94.0%	64.5%	86%	**78.9%**	**94.5%**
CSP-ResNext-50 [45]	Yes	78.5%	94.8%	-	-	**79.8%**	**95.2%**

Table 3: Comparison between Mish, Swish, ReLU and Leaky ReLU activation functions on image classification of ImageNet-1k dataset across various standard architectures. Data Augmentation indicates the use of CutMix, Mosaic, and Label Smoothing. † indicate scores for ReLU.

Blur Pooling

Source - MosaicML

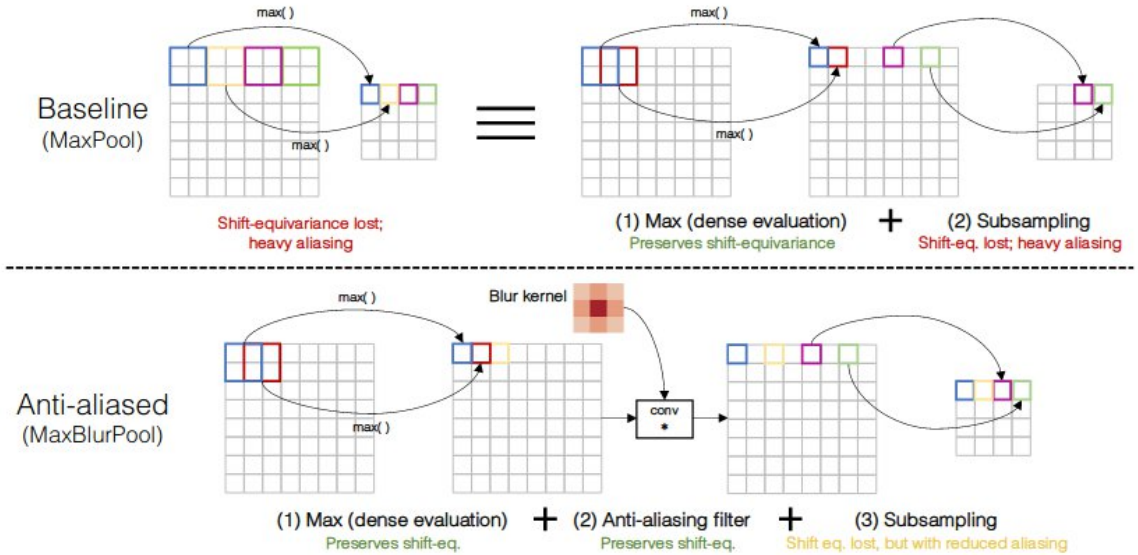


Figure 3. **Anti-aliased max-pooling. (Top)** Pooling does not preserve shift-equivariance. It is functionally equivalent to densely-evaluated pooling, followed by subsampling. The latter ignores the Nyquist sampling theorem and loses shift-equivariance. **(Bottom)** We low-pass filter between the operations. This keeps the first operation, while anti-aliasing the appropriate signal. Anti-aliasing and subsampling can be combined into one operation, which we refer to as **BlurPool**.