



# Caffe2

---

Jakub Powierza



Koło Naukowe „Gradient”



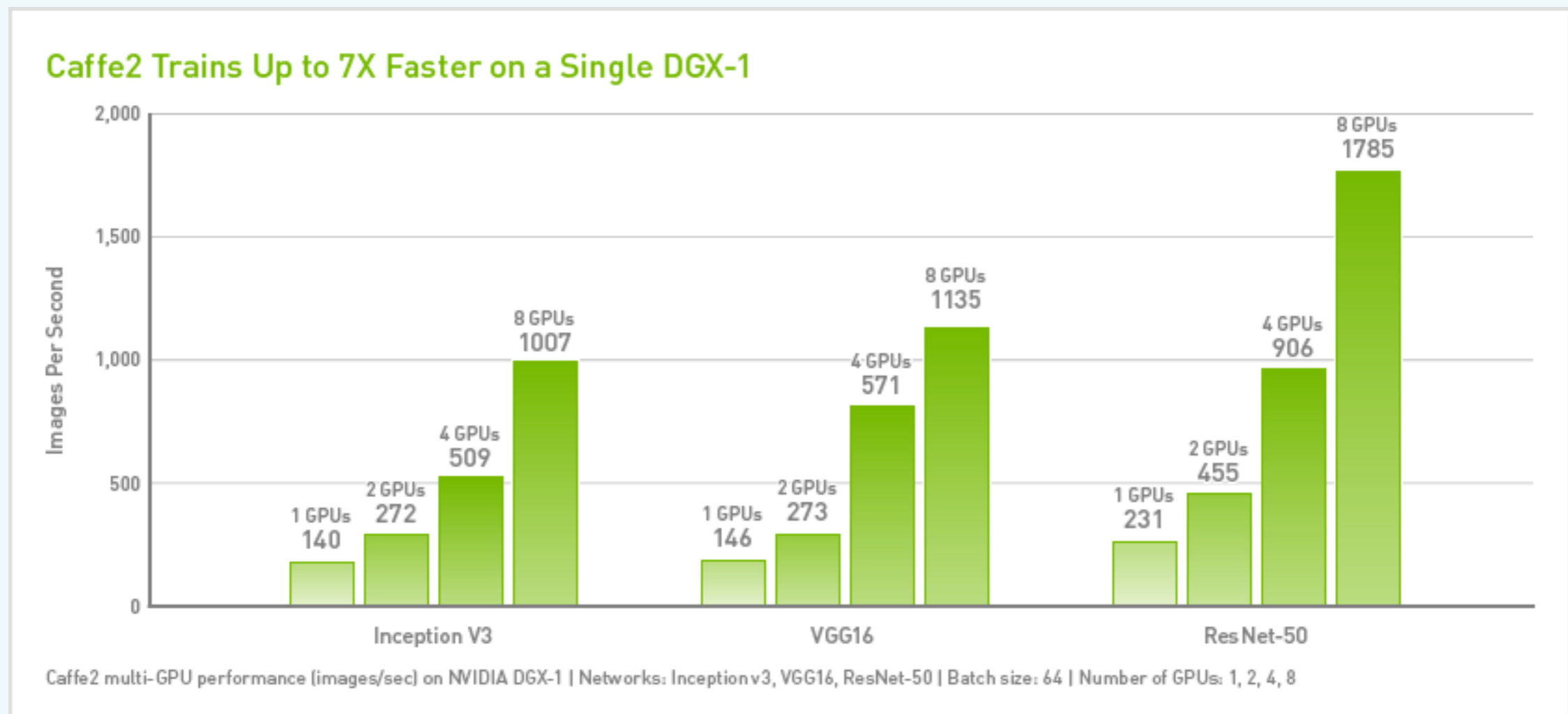
POLITECHNIKA  
GDAŃSKA

# Krótki opis

---

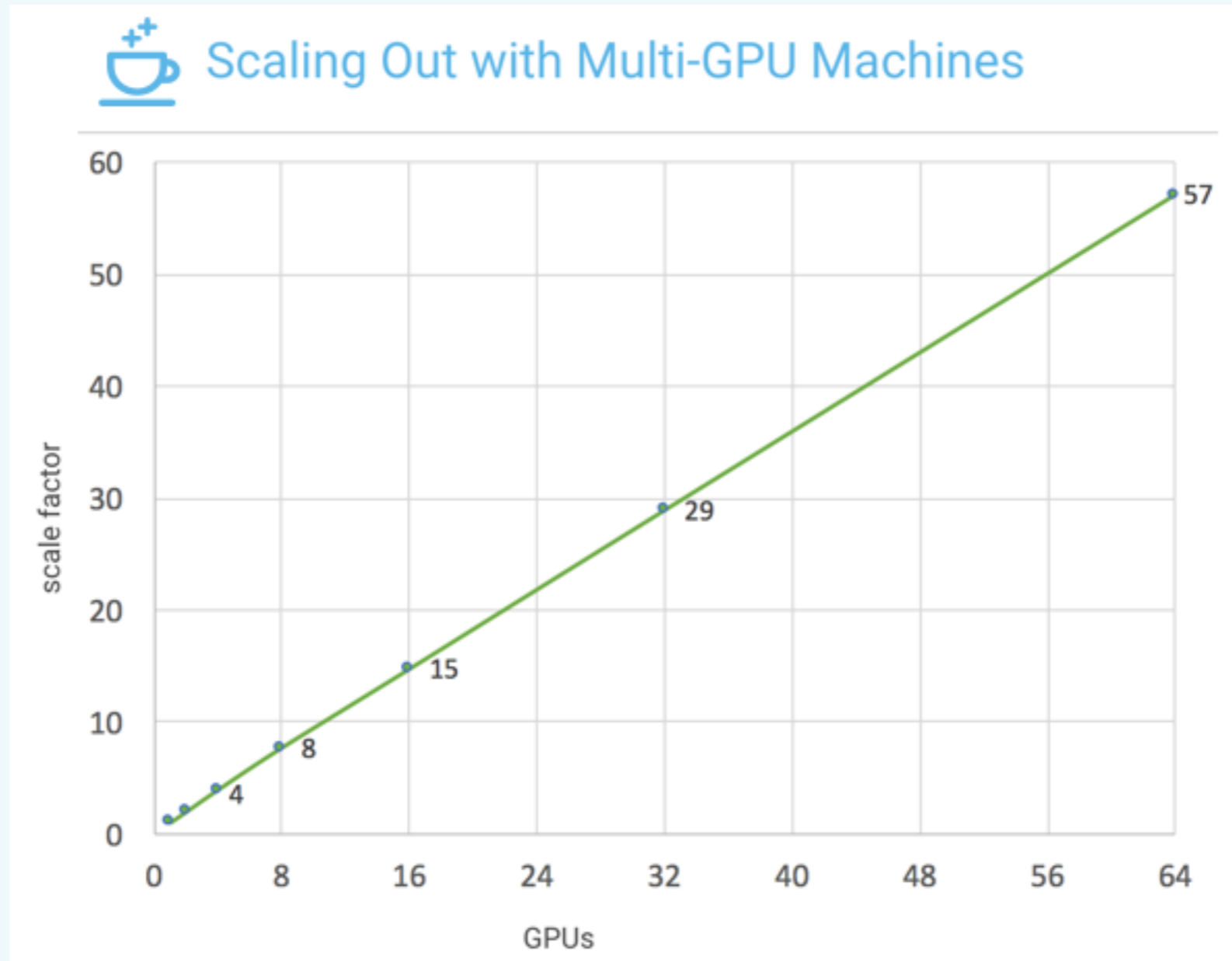
- Przygotowany przez Facebook'a (z Nvidią i Intelem),
- W pełni OpenSource,
- Backend w C++ (OpenMP dla CPU, CUDA dla GPU),
- Skalowalność - multi-GPU, multi-node (InfiniBand),
- Praca na CPU, GPU, iOS i Android (i nie tylko),
- Docs: <https://caffe2.ai/docs/>

# Skalowalność



Przykład skalowalności z użyciem węzła DGX-1 (8x“Pascal” Tesla P100)

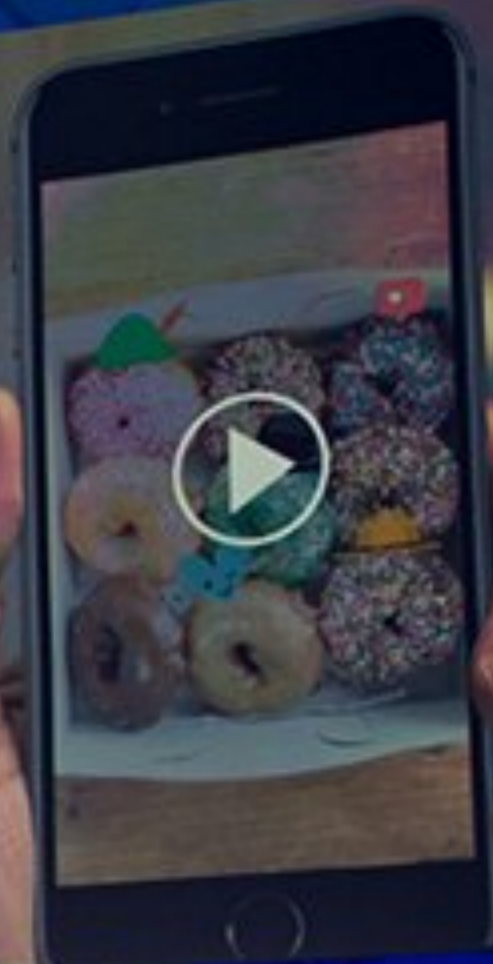
# Skalowalność



Wysoka skalowalność (bliska liniowej)



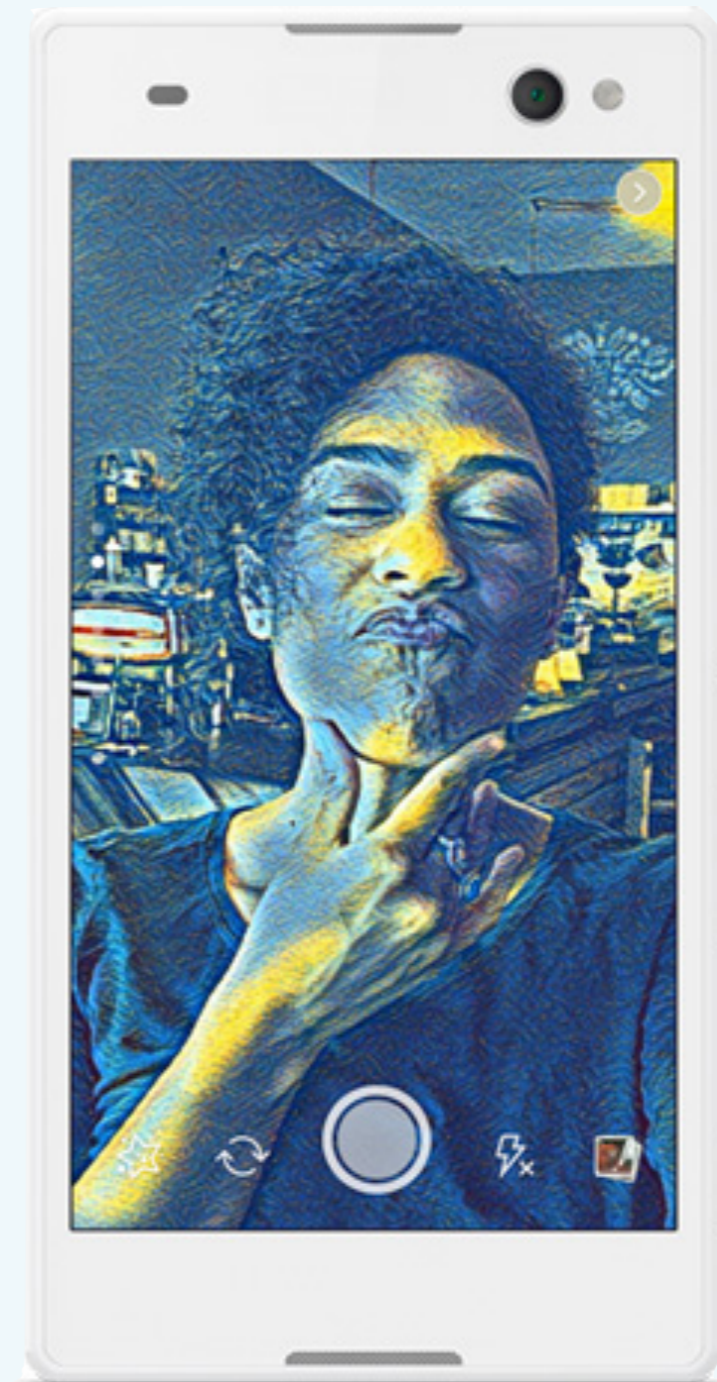
<http://caffe2.ai/>



# Caffe2go

---

- Przygotowany z myślą o urządzeniach mobilnych,
- Style transfer - 20FPS na iPhone 6S,
- Używany produkcyjnie w aplikacji mobilnej Facebooka.



Trochę konkretów...

# Jak pracować z Caffe2?

---

- Binarki - niedostępne :(
- Budowanie ze źródeł - same problemy! Tylko Ubuntu...
- Dostępny Docker Image (ufff...),
- Więcej: <https://caffe2.ai/docs/getting-started.html>



# Jak pracować z Caffe2?

---

Pobranie Docker Image'a:

```
docker pull caffe2ai/caffe2:c2.tutorial1.0.7.1
```











Uruchomienie Jupyter Notebook'a z tutorialami:

```
docker run -it -p 8888:8888 caffe2ai/caffe2:latest \  
  sh -c "jupyter notebook --no-browser \  
  -ip 0.0.0.0 /caffe2/caffe2/python/tutorials"
```

# Caffe2 Model Zoo

---

- Gotowe, zaimplementowane modele,
- Tylko niektóre zostały przeportowane do Caffe2...

NAME	TYPE	DATASET	CAFFE MODEL	CAFFE2 MODEL
<a href="#">Squeezenet</a>	image classification	ImageNet > AlexNet		
<a href="#">BVLC AlexNet</a>	image classification	ImageNet > AlexNet		
<a href="#">BVLC CaffeNet Model</a>	image classification	ImageNet > AlexNet		
<a href="#">BVLC GoogleNet Model</a>	image classification	ILSVRC 2014 > GoogleNet/Inception		
<a href="#">VGG Team ILSVRC14 16-layer</a>	image classification	ILSVRC 2014 > Very Deep CNN		
<a href="#">VGG Team ILSVRC14 19-layer</a>	image classification	ILSVRC 2014 > Very Deep CNN		
<a href="#">Network in Network</a>	small image	ImageNet		

Jak używać Caffe2?

# Workspace

---

```
>>> print("Blobs: {}".format(workspace.Blobs()))
```

```
Blobs: []
```

```
>>> print("Istnieje 'X'? {}".format(workspace.HasBlob("X")))
```

```
Istnieje 'X'? False
```

```
>>> workspace.FeedBlob("X", np.random.randn(2, 3).astype(np.float32))
```

```
>>> print("Blobs: {}".format(workspace.Blobs()))
```

```
Blobs: [u'X']
```

```
>>> print("Istnieje 'X'? {}".format(workspace.HasBlob("X")))
```

```
Istnieje 'X'? True
```

```
>>> print("Wartosc X:\n{}".format(workspace.FetchBlob("X")))
```

```
Wartosc X:
```

```
[[ 1.49933279  0.91606504  0.31837222]
 [ 0.3840501  -0.01287393 -0.09028989]]
```

# Przełączanie między Workspace'ami

---

```
>>> print("Aktualny workspace: {}".format(workspace.CurrentWorkspace()))
Aktualny workspace: default
```

```
>>> print("Blobs: {}".format(workspace.Blobs()))
Blobs: [u'X']
```

```
>>> workspace.SwitchWorkspace("gradient_demo", True)
>>> print("Aktualny workspace: {}".format(workspace.CurrentWorkspace()))
Aktualny workspace: gradient_demo
```

```
>>> print("Blobs: {}".format(workspace.Blobs()))
Blobs: []
```

# Operatory

---

```
>>> op = core.CreateOperator(  
>>>     "Relu", # Nazwa operatora  
>>>     ["X"], # Lista blobow wejsciwych  
>>>     ["Y"], # Lista blobow wyjsciwych  
>>> )  
>>> print("Typ operatora: {}".format(type(op)))  
>>> print("Zawartosc:\n{}", str(op))  
Typ operatora: <class 'caffe2.proto.caffe2_pb2.OperatorDef'>  
Zawartosc:  
  
input: "X"  
output: "Y"  
name: ""  
type: "Relu"
```

# Uruchamianie operatora

---

```
>>> workspace.RunOperatorOnce(op)
>>> print("Blobs: {}".format(workspace.Blobs()))
Blobs: [u'X', u'Y']

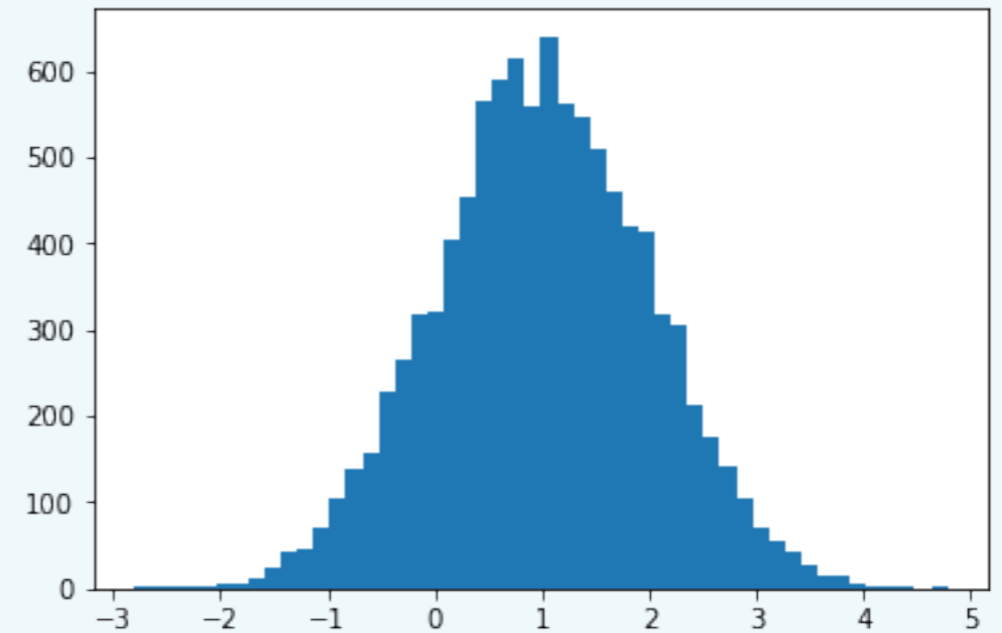
>>> print("Wartosc X: {}".format(workspace.FetchBlob("X")))
Wartosc X:
[[ -1.90759015  -0.22212909   0.8538388 ]
 [  0.49546063  -1.38485765  -0.05033699]]

>>> print("Wartosc Y: {}".format(workspace.FetchBlob("Y")))
Wartosc Y:
[[  0.          0.          0.8538388 ]
 [  0.49546063  0.          0.          ]]
```

# Bardziej skomplikowany operator

---

```
>>> op = core.CreateOperator(  
>>>     "GaussianFill",  
>>>     [], # Brak parametrow wejsciowych  
>>>     ["Z"],  
>>>     shape=[100, 100],  
>>>     mean=1.0,  
>>>     std=1.0,  
>>> )  
>>> workspace.RunOperatorOnce(op)  
>>> temp = workspace.FetchBlob("Z")  
>>> pyplot.hist(temp.flatten(), bins=50)  
>>> pyplot.show()
```



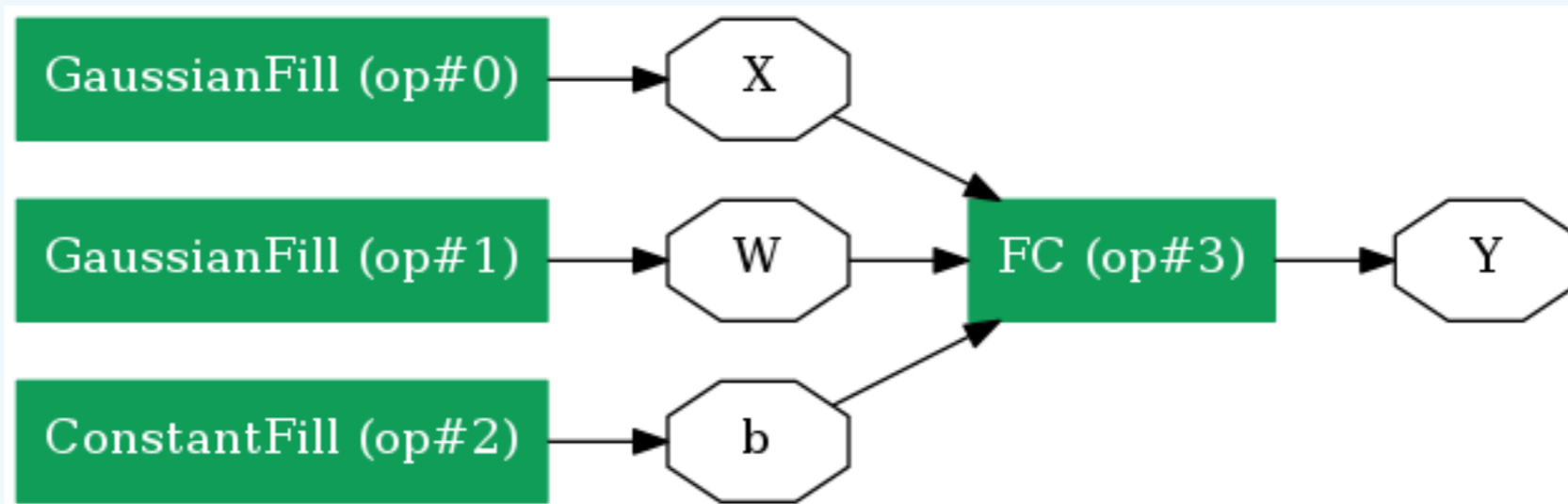


# Sieci

---

```
>>> net = core.Net("gradient_net")
>>> X = net.GaussianFill([], ["X"], mean=0.0, std=1.0, shape=[2, 3], run_once=0)
>>> W = net.GaussianFill([], ["W"], mean=0.0, std=1.0, shape=[5, 3], run_once=0)
>>> b = net.ConstantFill([], ["b"], shape=[5,], value=1.0, run_once=0)
>>> Y = X.FC([W, b], [„Y"])

>>> from caffe2.python import net_drawer
>>> from IPython import display
>>> graph = net_drawer.GetPydotGraph(net, rankdir="LR")
>>> display.Image(graph.create_png(), width=800)
```



# Uruchamianie sieci

---

```
>>> workspace.ResetWorkspace()
>>> print("Blobs: {}".format(workspace.Blobs()))
Blobs: []

>>> workspace.RunNetOnce(net)
>>> print("Blobs: {}".format(workspace.Blobs()))
Blobs: [u'W', u'X', u'Y', u'b']

>>> for name in workspace.Blobs():
>>>     print("{}:\n{}".format(name, workspace.FetchBlob(name)))
W:
[[ 1.15298831  2.02991176 -0.53879094]
 [ 0.63714212  0.16825198  0.95623571]
 [-0.13984574  1.80930626 -0.43459323]
 [-0.39667347  1.87240088 -0.72185475]
 [ 0.76232553  1.56994569  0.1997031  ]]
X:
[[ 0.84681964  3.25020695  1.22675776]
 [-2.34444427 -1.34224248 -1.98630714]]
Y:
[[ 7.91304064  3.2594676  6.2290554  5.86423874  6.9931879  ]
 [-3.35754633 -2.61895704 -0.23743153  0.85058808 -3.29114914]]
b:
[ 1.  1.  1.  1.  1.]
```

# Implementacja LeNet (dla MNISTa)

---

```
>>> def AddLeNetModel(model, data):
>>>     # Image size: 28 x 28 -> 24 x 24
>>>     conv1 = model.Conv(data, 'conv1', dim_in=1, dim_out=20, kernel=5)
>>>     # Image size: 24 x 24 -> 12 x 12
>>>     pool1 = model.MaxPool(conv1, 'pool1', kernel=2, stride=2)
>>>     # Image size: 12 x 12 -> 8 x 8
>>>     conv2 = model.Conv(pool1, 'conv2', dim_in=20, dim_out=50, kernel=5)
>>>     # Image size: 8 x 8 -> 4 x 4
>>>     pool2 = model.MaxPool(conv2, 'pool2', kernel=2, stride=2)
>>>     # 50 * 4 * 4 = dim_out from previous layer * image size
>>>     fc3 = model.FC(pool2, 'fc3', dim_in=50 * 4 * 4, dim_out=500)
>>>     fc3 = model.ReLU(fc3, fc3)
>>>     pred = model.FC(fc3, 'pred', 500, 10)
>>>     softmax = model.Softmax(pred, 'softmax')
>>>     return softmax
```

# Przydatne operatory

---

```
# Zatrzymanie gradientu
Y = Y.StopGradient([], "Y")

# Iteracja
net.Iter(ITER, ITER)

# Learning Rate
LR = net.LearningRate(ITER, "LR", base_lr=-0.1,
                      policy="step", stepsize=20, gamma=0.9)

# Suma = W + grad(W) * LR
net.WeightedSum([W, ONE, gradient_map[W], LR], W)

# Cross entropy
xent = model.LabelCrossEntropy([prediction, label], 'xent')

# Uczenie epokami
for i in range(50):
    workspace.RunNet(net.Proto().name)
```

DEMO

# Protocol buffer

---

- Struktura danych (tak samo jak XML, JSON),
- Lekka, niezależna od platformy,
- Zapis „w miarę czytelny” dla człowieka,
- Gotowe biblioteki w C++, C#, Go, Java, Python.

# Przykład

---

```
name: "gradient_net"
op {
  output: "X"
  name: ""
  type: "GaussianFill"
  arg {
    name: "std"
    f: 1.0
  }
  arg {
    name: "run_once"
    i: 0
  }
  arg {
    name: "shape"
    ints: 2
    ints: 3
  }
  arg {
    name: "mean"
    f: 0.0
  }
}
```

```
op {
  output: "W"
  name: ""
  type: "GaussianFill"
  arg {
    name: "std"
    f: 1.0
  }
  arg {
    name: "run_once"
    i: 0
  }
  arg {
    name: "shape"
    ints: 5
    ints: 3
  }
  arg {
    name: "mean"
    f: 0.0
  }
}
```

```
op {
  output: "b"
  name: ""
  type: "ConstantFill"
  arg {
    name: "run_once"
    i: 0
  }
  arg {
    name: "shape"
    ints: 5
  }
  arg {
    name: "value"
    f: 1.0
  }
}
op {
  input: "X"
  input: "W"
  input: "b"
  output: "Y"
  name: ""
  type: "FC"
}
```

# Ciekawe/przydatne linki

---

- Dokumentacja: <https://caffe2.ai>
- Docker: <https://www.docker.com>
- Instalacja: <https://caffe2.ai/docs/getting-started.html>
- Tutoriale: <https://caffe2.ai/docs/tutorials>
- GitHub: <https://github.com/caffe2/caffe2>
- Przykład aplikacji: <https://github.com/bwasti/AICamera>
- Ciekawy artykuł: <https://www.nextplatform.com/2017/04/19/machine-learning-gets-infiniband-boost-caffe2/>