

U-net CNN in APL: Appendices

Exploring zero-framework, zero-library machine learning

ANONYMOUS AUTHOR(S)

ACM Reference Format:

Anonymous Author(s). 2022. U-net CNN in APL: Appendices: Exploring zero-framework, zero-library machine learning. In . ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICFP'22, Sep 11 - Sep 16, 2022, Ljubljana, Slovenia

© 2022 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

APPENDIX A: COMPLETE APL U-NET IMPLEMENTATION

```
:Namespace UNET
```

```
W←0 ⋄ V←0 ⋄ Z←0 ⋄ LR←1e-9 ⋄ MO←0.99
```

```
FWD←{Z←(≠W)ρ←0
```

```
CV←{0[Z←Z[α]←C[Z[α],←Z←(,[2+ι3]3 3⊞Z[α]←Cω)+.×,[ι3]α>W}
```

```
CC←{ω,~([p)+(−[p)+(α>Z)−p←2÷~(ρα>Z)−ρω}
```

```
MX←{[≠[2],[2 3](2 2ρ2)⊞Z[α]←Cω}
```

```
UP←{((2×-1+ρω),-1↑ρα>W)ρ0 2 1 3 4⊞ω+.×α>W←Z[α]←Cω}
```

```
C1←{1E-8+z÷[ι2]+/z←*z-[ι2][ /z←ω+.×α>W←Z[α]←Cω}
```

```
LA←{α≥≠Z:ω
```

```
down←(α+6)▽(α+2)MX(α+1)CV(α+0)CV ω
```

```
(α+2)CC(α+5)UP(α+4)CV(α+3)CV down}
```

```
2 C1 1 CV 0 CV 3 LA ωρ~3↑1,~ρω}
```

```
BCK←{Y←α ⋄ YΔ←ω
```

```
Δ←{0←W[α]←C(α>W)−LR×V[α]←Cω+MO×(ρω)ρα>V}
```

```
ΔCV←{w←,[ι3]⊞ϕ[1]0 1 3 2⊞α>W ⋄ x←α>Z ⋄ Δz←ω×0<1>α>Z
```

```
Δz←-2⊞ϕ[1](4+2↑ρΔz)↑Δz
```

```
_←α Δ 3 0 1 2⊞(⊞,[ι2]Δz)+.×,[ι2]3 3⊞x
```

```
w+.×~, [2+ι3]3 3⊞Δz}
```

```
ΔCC←{x←α>Z ⋄ Δz←ω ⋄ d←−[2÷~2↑(ρx)−ρΔz ⋄ (≡d)⊞(1≡d)⊞[1](ρx)↑Δz}
```

```
ΔMX←{x←α>Z ⋄ Δz←ω ⋄ y×x=y←(ρx)↑2÷2/[1]Δz}
```

```
ΔUP←{w←α>W ⋄ x←α>Z ⋄ Δz←ω ⋄ cz←(2 2ρ2)⊞Δz
```

```
_←α Δ(⊞,[ι2]x)+.×,[ι2]cz
```

```
(,[2+ι3]cz)+.×⊞;w}
```

```
ΔC1←{w←α>W ⋄ x←α>Z ⋄ Δz←ω ⋄ _←α Δ(⊞,[ι2]x)+.×,[ι2]Δz ⋄ Δz+.×⊞w}
```

```
ΔLA←{α≥≠Z:ω
```

```
down←(α+6)▽(α+3)ΔCV(α+4)ΔCV(α+5)ΔUP ω↑[2]~−2÷~⊞ϕρω
```

```
(α+0)ΔCV(α+1)ΔCV(ω ΔCC~α+2)+(α+2)ΔMX down}
```

```
3 ΔLA 0 ΔCV 1 ΔCV 2 ΔC1 YΔ−(~Y),[1.5]Y}
```

```
E←{−+/,⊞(α×ω[; ;1])+(~α)×ω[; ;0]}
```

```
RUN←{Y YΔ(Y E YΔ)←(Y←[0.5+nm↑ω↓~2÷~(ρω)−nm←2↑ρYΔ)BCK←YΔ←FWD α}
```

```
:EndNamespace
```

APPENDIX B: PYTORCH REFERENCE IMPLEMENTATION

```

99
100 import torch
101 import torch.nn as nn
102 import torchvision
103 import torchvision.transforms.functional
104
105 class TwoConv(nn.Module):
106
107     def __init__(self, in_channels, out_channels):
108         super().__init__()
109
110         self.path = nn.Sequential(
111             nn.Conv2d(in_channels, out_channels,
112                     kernel_size=(3, 3), bias=False),
113             nn.ReLU(inplace=True),
114             nn.Conv2d(out_channels, out_channels,
115                     kernel_size=(3, 3), bias=False),
116             nn.ReLU(inplace=True),
117         )
118
119     def forward(self, x):
120         return self.path(x)
121
122 class Down(nn.Module):
123
124     def __init__(self, in_channels):
125         super().__init__()
126
127         self.path = nn.Sequential(
128             nn.MaxPool2d(kernel_size=(2, 2), stride=2),
129             TwoConv(in_channels, 2 * in_channels),
130         )
131
132     def forward(self, x):
133         return self.path(x)
134
135 class Up(nn.Module):
136
137     def __init__(self, in_channels):
138         super().__init__()
139
140         self.upsampling = nn.ConvTranspose2d(
141             in_channels,
142             in_channels // 2,
143             kernel_size=(2, 2),
144             stride=2,
145             bias=False,

```

```

148         )
149         self.convolutions =
150             TwoConv(in_channels, in_channels // 2)
151
152     def forward(self, x_to_crop, x_in):
153
154         upped = self.upsampling(x_in)
155         cropped = torchvision.transforms.functional.center_crop(
156             x_to_crop, upped.shape[-2:])
157         )
158         x = torch.cat([cropped, upped], dim=1)
159         return self.convolutions(x)
160
161 class USegment(nn.Module):
162
163     def __init__(self, in_channels, bottom_u=None):
164         super().__init__()
165
166         # Default value for the bottom U.
167         if bottom_u is None:
168             bottom_u = lambda x: x
169
170         self.down = Down(in_channels)
171         self.bottom_u = bottom_u
172         self.up = Up(2 * in_channels)
173
174     def forward(self, x):
175         return self.up(x, self.bottom_u(self.down(x)))
176
177 class UNet(nn.Module):
178
179     def __init__(self):
180         super().__init__()
181
182         self.u = USegment(512)
183         self.u = USegment(256, self.u)
184         self.u = USegment(128, self.u)
185         self.u = USegment(64, self.u)
186         self.path = nn.Sequential(
187             TwoConv(1, 64),
188             self.u,
189             nn.Conv2d(64, 2, kernel_size=1, bias=False),
190         )
191
192     def forward(self, x):
193         return self.path(x)
194
195
196

```