all_layer_test

January 7, 2022

0.1 # DSD-2021 Project (CIFAR-10)

0.2 All layer inference check

- This is a python script to help you check if your RTL impelementation for all the layers are correct

0.3 Usage

- Run all the cells to check if your HW works

[1]:
```
from utils.layers_cifar10 import *
from utils.bit_operation import *
from utils.setup_cifar10 import *
from utils.scale_uart import *
from utils.board import *
import time
import numpy as np
import time
```

0.3.1 Load dataset for image generate

[2]:
```
label_list = ["Airplane", "Automobile", "Bird", "Cat", "Deer", "Dog", "Frog", \
              "Horse", "Ship", "Truck"]
# TEST ORIGIN
X_test_origin, _ = load_CIFAR10_test()
# TEST SET
X_test, y_test = load_CIFAR10_test()

# Data Pre-processing
m = [0.4935, 0.4834, 0.4472]
std = [0.2476, 0.2626, 0.2626]
# Only pre-process the test dataset
X_test = np.reshape(X_test, (X_test.shape[0], 3, 32, 32))
for i in range(3):
    X_test[:,i,:,:] = (X_test[:,i,:,:]-m[i])/std[i]
```

0.3.2 Simulation dataset for our 8-bit MAC unit

```
[3]: X_test_ = np.load("./data/cifar10_dataset_quan/images_100.npy")
```

0.4 ## Load network parameter

```
[4]: # Load quantized network param
conv1_w_ = np.load("./data/cifar10_network_quan_param/cifar10_conv1_weight_quan.
 ↪npy")
conv1_b_ = np.load("./data/cifar10_network_quan_param/cifar10_conv1_bias_quan.
 ↪npy")
conv2_w_ = np.load("./data/cifar10_network_quan_param/cifar10_conv2_weight_quan.
 ↪npy")
conv2_b_ = np.load("./data/cifar10_network_quan_param/cifar10_conv2_bias_quan.
 ↪npy")
conv3_w_ = np.load("./data/cifar10_network_quan_param/cifar10_conv3_weight_quan.
 ↪npy")
conv3_b_ = np.load("./data/cifar10_network_quan_param/cifar10_conv3_bias_quan.
 ↪npy")
conv4_w_ = np.load("./data/cifar10_network_quan_param/cifar10_conv4_weight_quan.
 ↪npy")
conv4_b_ = np.load("./data/cifar10_network_quan_param/cifar10_conv4_bias_quan.
 ↪npy")
conv5_w_ = np.load("./data/cifar10_network_quan_param/cifar10_conv5_weight_quan.
 ↪npy")
conv5_b_ = np.load("./data/cifar10_network_quan_param/cifar10_conv5_bias_quan.
 ↪npy")
conv6_w_ = np.load("./data/cifar10_network_quan_param/cifar10_conv6_weight_quan.
 ↪npy")
conv6_b_ = np.load("./data/cifar10_network_quan_param/cifar10_conv6_bias_quan.
 ↪npy")
fc1_w_   = np.load("./data/cifar10_network_quan_param/cifar10_fc1_weight_quan.
 ↪npy")
fc1_b_   = np.load("./data/cifar10_network_quan_param/cifar10_fc1_bias_quan.
 ↪npy")
fc2_w_   = np.load("./data/cifar10_network_quan_param/cifar10_fc2_weight_quan.
 ↪npy")
fc2_b_   = np.load("./data/cifar10_network_quan_param/cifar10_fc2_bias_quan.
 ↪npy")
fc3_w_   = np.load("./data/cifar10_network_quan_param/cifar10_fc3_weight_quan.
 ↪npy")
fc3_b_   = np.load("./data/cifar10_network_quan_param/cifar10_fc3_bias_quan.
 ↪npy")
```

0.5 ## Test for accuracy

Do inference

0.5.1 Board connection

[5]:
```
port_list = get_port_list()
SU = get_scale_uart(port_list)
```

```
Current OS: Windows
['COM1', 'COM4']
COM1 port cannot be connected.
COM4 port connected!
```

0.5.2 Setting the VDMA

[6]:
```
## DO NOT CHANGE
## IT IS VDMA AND EACH MODULE'S BASE ADDRESS FOR CONTROL APB + AXI
##### PARAMETER INFORMATION
VDMA0_BASE_ADDR= 0x0c00_0000
VDMA1_BASE_ADDR= 0x0c10_0000
VDMA2_BASE_ADDR= 0x0c20_0000

FC_BASE_ADDR   = 0x0d00_0000
CONV_BASE_ADDR = 0x0d10_0000
POOL_BASE_ADDR = 0x0d20_0000

### FIXED FOR OUR NETWORK
OP_SIZE                        = 4
ADDR_SIZE                      = 28
DATA_SIZE                      = 32
```

0.6 Image address memory map

Addresss range: 0x0000_0000 ~ 0x01FF_FFFF
Size: 32768 KB

[7]:
```
start = time.time()
SU.su_write_data(0x0000_0000, 3)
data = SU.su_read_data(0x0000_0000)
SU.su_set_image({'BASE_ADDR': 0x0000_0000}, "./data/cifar10_dataset_quan/
 ↪images_100.npy")
print("image set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
image set done
        Total time: 33.02 sec
```

0.7 Conv1 memory map

Convolution 1
Weight
  Address range: 0x0200_0000 ~ 0x020F_FFFF

Size: 1024KB
bias
Address range: 0x0210_0000 ~ 0x021F_FFFF
Size: 1024KB
output
Addresss range: 0x0600_0000 ~ 0x060F_FFFF
Size: 1024KB

[8]:
```
print("conv1 parameter load")
start = time.time()
SU.su_set_conv_w({'BASE_ADDR': 0x0200_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv1_weight_quan.npy")
SU.su_set_conv_b({'BASE_ADDR': 0x0210_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv1_bias_quan.npy")
print("conv1 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
conv1 parameter load
conv1 set done
        Total time: 0.12 sec
```

0.8 Pool1 memory map

Max Pool 1
output
Addresss range: 0x0610_0000 ~ 0x061F_FFFF
Size: 1024KB

0.9 Conv2 memory map

Convolution 2
Weight
Address range: 0x0220_0000 ~ 0x026F_FFFF
Size: 5120KB
bias
Address range: 0x0270_0000 ~ 0x027F_FFFF
Size: 1024KB
output
Addresss range: 0x0620_0000 ~ 0x062F_FFFF
Size: 1024KB

[9]:
```
print("conv2 parameter load")
start = time.time()
SU.su_set_conv_w({'BASE_ADDR': 0x0220_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv2_weight_quan.npy")
SU.su_set_conv_b({'BASE_ADDR': 0x0270_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv2_bias_quan.npy")
print("conv2 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
conv2 parameter load
conv2 set done
        Total time: 2.15 sec
```

0.10 Pool2 memory map

Max Pool 2
output
  Addresss range: 0x0630_0000 ~ 0x063F_FFFF
  Size: 1024KB

0.11 Conv3 memory map

Convolution 3
Weight
  Address range: 0x0280_0000 ~ 0x028F_FFFF
  Size: 5120KB
bias
  Address range: 0x02C0_0000 ~ 0x02CF_FFFF
  Size: 1024KB
output
  Addresss range: 0x0640_0000 ~ 0x064F_FFFF
  Size: 1024KB

```
[10]: print("conv3 parameter load")
start = time.time()
SU.su_set_conv_w({'BASE_ADDR': 0x0280_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv3_weight_quan.npy")
SU.su_set_conv_b({'BASE_ADDR': 0x02C0_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv3_bias_quan.npy")
print("conv3 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
conv3 parameter load
conv3 set done
        Total time: 8.54 sec
```

0.12 Conv4 memory map

Convolution 4
Weight
  Address range: 0x0300_0000 ~ 0x038F_FFFF
  Size: 9216KB
bias
  Address range: 0x0390_0000 ~ 0x039F_FFFF
  Size: 1024KB
output

Addresss range: 0x0650_0000 ~ 0x065F_FFFF
Size: 1024KB

```
[11]: print("conv4 parameter load")
start = time.time()
SU.su_set_conv_w({'BASE_ADDR': 0x0300_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv4_weight_quan.npy")
SU.su_set_conv_b({'BASE_ADDR': 0x0390_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv4_bias_quan.npy")
print("conv4 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
conv4 parameter load
conv4 set done
        Total time: 17.10 sec
```

0.13 Pool3 memory map

Max Pool 3
output
Addresss range: 0x0660_0000 ~ 0x066F_FFFF
Size: 1024KB

0.14 Conv5 memory map

Convolution 5
Weight
Address range: 0x03A0_0000 ~ 0x03EF_FFFF
Size: 5120KB
bias
Address range: 0x03F0_0000 ~ 0x03FF_FFFF
Size: 1024KB
output
Addresss range: 0x0670_0000 ~ 0x067F_FFFF
Size: 1024KB

```
[12]: print("conv5 parameter load")
start = time.time()
SU.su_set_conv_w({'BASE_ADDR': 0x03A0_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv5_weight_quan.npy")
SU.su_set_conv_b({'BASE_ADDR': 0x03F0_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv5_bias_quan.npy")
print("conv5 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
conv5 parameter load
conv5 set done
        Total time: 34.11 sec
```

0.15 Conv6 memory map

Convolution 6
Weight
  Address range: 0x0400_0000 ~ 0x048F_FFFF
  Size: 9216KB
bias
  Address range: 0x0490_0000 ~ 0x049F_FFFF
  Size: 1024KB
output
  Addresss range: 0x0680_0000 ~ 0x068F_FFFF
  Size: 1024KB

[13]:
```
print("conv6 parameter load")
start = time.time()
SU.su_set_conv_w({'BASE_ADDR': 0x0400_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv6_weight_quan.npy")
SU.su_set_conv_b({'BASE_ADDR': 0x0490_0000}, "./data/cifar10_network_quan_param/
 ↪cifar10_conv6_bias_quan.npy")
print("conv6 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
conv6 parameter load
conv6 set done
        Total time: 67.98 sec
```

0.16 Pool4 memory map

Max Pool 4
output
  Addresss range: 0x0690_0000 ~ 0x069F_FFFF
  Size: 1024KB

0.17 FC1 memory map

Fully-Connected 1
Weight
  Address range: 0x0500_0000 ~ 0x052F_FFFF
  Size: 3072KB
bias
  Address range: 0x0530_0000 ~ 0x053F_FFFF
  Size: 1024KB
output
  Addresss range: 0x06A0_0000 ~ 0x06AF_FFFF
  Size: 1024KB

[14]:
```
print("fc1 parameter load")
start = time.time()
```

```
SU.su_set_fc_w({'BASE_ADDR': 0x0500_0000}, "./data/cifar10_network_quan_param/
↪cifar10_fc1_weight_quan.npy")
SU.su_set_fc_b({'BASE_ADDR': 0x0530_0000}, "./data/cifar10_network_quan_param/
↪cifar10_fc1_bias_quan.npy")
print("fc1 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
fc1 parameter load
fc1 set done
        Total time: 29.99 sec
```

0.18 FC2 memory map

Fully-Connected 2
Weight
  Address range: 0x0540_0000 ~ 0x054F_FFFF
  Size: 1024KB
bias
  Address range: 0x0550_0000 ~ 0x055F_FFFF
  Size: 1024KB
output
  Addresss range: 0x06B0_0000 ~ 0x06BF_FFFF
  Size: 1024KB

[15]:
```
print("fc2 parameter load")
start = time.time()
SU.su_set_fc_w({'BASE_ADDR': 0x0540_0000}, "./data/cifar10_network_quan_param/
↪cifar10_fc2_weight_quan.npy")
SU.su_set_fc_b({'BASE_ADDR': 0x0550_0000}, "./data/cifar10_network_quan_param/
↪cifar10_fc2_bias_quan.npy")
print("fc2 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
fc2 parameter load
fc2 set done
        Total time: 1.90 sec
```

0.19 FC3 memory map

Fully-Connected 3
Weight
  Address range: 0x0560_0000 ~ 0x056F_FFFF
  Size: 1024KB
bias
  Address range: 0x0570_0000 ~ 0x057F_FFFF
  Size: 1024KB
output
  Addresss range: 0x06C0_0000 ~ 0x06CF_FFFF
  Size: 1024KB

[16]:
```
print("fc3 parameter load")
start = time.time()
SU.su_set_fc_w({'BASE_ADDR': 0x0560_0000}, "./data/cifar10_network_quan_param/
↪cifar10_fc3_weight_quan.npy")
SU.su_set_fc_b({'BASE_ADDR': 0x0570_0000}, "./data/cifar10_network_quan_param/
↪cifar10_fc3_bias_quan.npy")
print("fc3 set done")
print("\tTotal time: {:.2f} sec".format(time.time() - start))
```

```
fc3 parameter load
fc3 set done
        Total time: 0.08 sec
```

0.19.1 Parameter check (For debugging!)

In the below code, it verifies that **the data is stored correctly in DRAM**

[17]:
```
debug_data = np.load("./data/cifar10_dataset_quan/images_100.npy")
```

[18]:
```
print(debug_data.shape)
```

```
(100, 3, 32, 32)
```

0.19.2 First, just check first image (debug_data[0])

[ ]:
```
# Print in 4 Bytes
debug_flat = debug_data.flatten()
for i in range(int(1 * 3 * 32 * 32 / 4)):
    temp = debug_flat[i*4:i*4+4]
    print(i, "\t", temp)
```

[ ]:
```
debug_flat_bin = to_8bit_fixed_binary(debug_flat)
for i in range(int(1 * 3 * 32 * 32 / 4)):
    temp = debug_flat_bin[i*4:i*4+4]
    print(i, "\t", temp)
```

[ ]:
```
# Check for written data in DRAM
base_addr_debug = 0x0000_0000 # input image
for i in range(int(1 * 3 * 32 * 32 / 4)):
    data = SU.su_read_data(base_addr_debug + i*4)
    print(data)
```

0.19.3 INFERENCE

0.19.4 Just one image step by step

[19]:
```
###################################################################
#        Convolution 1 + ReLU
###################################################################
# Convolution
# - in:        (n, 3, 32, 32)
# - out:      (n, 32, 28, 28)
# - weight:     (32, 3, 3, 3)
# - bias:                 (32)
# ReLU
# - in:       (n. 32. 32. 32)
# - out:      (n. 32. 32. 32)
###################################################################
I = {'IN_CH': 3, 'OUT_CH': 32, 'FLEN': 32}
F = {'BASE_ADDR': 0x0000_0000, 'STRIDE_SIZE': 3*32*32, 'HSIZE': 3*32*32,␣
 ↪'VSIZE': 1}
W = {'BASE_ADDR': 0x0200_0000, 'STRIDE_SIZE': 3*32*9, 'HSIZE': 3*32*9, 'VSIZE':␣
 ↪1}
B = {'BASE_ADDR': 0x0210_0000, 'STRIDE_SIZE': 32, 'HSIZE': 32, 'VSIZE': 1}
R = {'BASE_ADDR': 0x0600_0000, 'STRIDE_SIZE': 32*32*32, 'HSIZE': 32*32*32,␣
 ↪'VSIZE': 1}
SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
```

[19]:
```
1
```

[ ]:
```
# You can check the result of first layer by below code
a = 0x0600_0000
for i in range(int(32*32*32/4)):
    temp = SU.su_read_data(a + 4*i)
    print(i, "\t", temp)
```

[20]:
```
###################################################################
#        Max Pool 1
###################################################################
# Max Pooling
# - in:       (n. 32. 32. 32)
# - out:      (n, 32, 16, 16)
###################################################################
I = {'IN_CH': 32, 'FLEN': 32}
F = {'BASE_ADDR': 0x0600_0000, 'STRIDE_SIZE': 32*32*32, 'HSIZE': 32*32*32,␣
 ↪'VSIZE': 1}
R = {'BASE_ADDR': 0x0610_0000, 'STRIDE_SIZE': 32*16*16, 'HSIZE': 32*16*16,␣
 ↪'VSIZE': 1}
SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
```

[20]: 1

```
[ ]: a = 0x0610_0000
     for i in range(int(32*16*16/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[21]: ##################################################################
      #        Convolution 2 + ReLU
      ##################################################################
      # Convolution
      # - in:       (n, 32, 16, 16)
      # - out:      (n, 64, 16, 16)
      # - weight:    (64, 32, 3, 3)
      # - bias:                (64)
      # ReLU
      # - in:       (n. 64. 16. 16)
      # - out:      (n. 64. 16. 16)
      ##################################################################
      I = {'IN_CH': 32, 'OUT_CH': 64, 'FLEN': 16}
      F = {'BASE_ADDR': 0x0610_0000, 'STRIDE_SIZE': 32*16*16, 'HSIZE': 32*16*16,␣
       ↪'VSIZE': 1}
      W = {'BASE_ADDR': 0x0220_0000, 'STRIDE_SIZE': 32*64*9, 'HSIZE': 32*64*9,␣
       ↪'VSIZE': 1}
      B = {'BASE_ADDR': 0x0270_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
      R = {'BASE_ADDR': 0x0620_0000, 'STRIDE_SIZE': 64*16*16, 'HSIZE': 64*16*16,␣
       ↪'VSIZE': 1}
      SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
```

[21]: 1

```
[ ]: a = 0x0620_0000
     for i in range(int(64*16*16/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[22]: ##################################################################
      #        Max Pool 2
      ##################################################################
      # Max Pooling
      # - in:      (n. 64. 16. 16)
      # - out:       (n, 64, 8, 8)
      ##################################################################
      I = {'IN_CH': 64, 'FLEN': 16}
      F = {'BASE_ADDR': 0x0620_0000, 'STRIDE_SIZE': 64*16*16, 'HSIZE': 64*16*16,␣
       ↪'VSIZE': 1}
```

```
R = {'BASE_ADDR': 0x0630_0000, 'STRIDE_SIZE': 64*8*8, 'HSIZE': 64*8*8, 'VSIZE':␣
 ↪1}
SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
```

[22]: 1

[ ]:
```
a = 0x0630_0000
for i in range(int(64*8*8/4)):
    temp = SU.su_read_data(a + 4*i)
    print(i, "\t", temp)
```

[23]:
```
###################################################################
#        Convolution 3 + ReLU
###################################################################
# Convolution
# - in:        (n, 64, 8, 8)
# - out:       (n, 128, 8, 8)
# - weight:  (128, 64, 3, 3)
# - bias:             (128)
# ReLU
# - in:        (n. 128. 8. 8)
# - out:       (n. 128. 8. 8)
###################################################################
I = {'IN_CH': 64, 'OUT_CH': 128, 'FLEN': 8}
F = {'BASE_ADDR': 0x0630_0000, 'STRIDE_SIZE': 64*8*8, 'HSIZE': 64*8*8, 'VSIZE':␣
 ↪1}
W = {'BASE_ADDR': 0x0280_0000, 'STRIDE_SIZE': int(64*128*9/2), 'HSIZE':␣
 ↪int(64*128*9/2), 'VSIZE': 2}
B = {'BASE_ADDR': 0x02C0_0000, 'STRIDE_SIZE': 128, 'HSIZE': 128, 'VSIZE': 1}
R = {'BASE_ADDR': 0x0640_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
 ↪'VSIZE': 1}
SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
```

[23]: 1

[ ]:
```
a = 0x0640_0000
for i in range(int(128*8*8/4)):
    temp = SU.su_read_data(a + 4*i)
    print(i, "\t", temp)
```

[24]:
```
###################################################################
#        Convolution 4 + ReLU
###################################################################
# Convolution
# - in:        (n, 128, 8, 8)
# - out:       (n, 128, 8, 8)
# - weight: (128, 128, 3, 3)
# - bias:              (128)
# ReLU
# - in:         (n. 128. 8. 8)
# - out:        (n. 128. 8. 8)
###################################################################
I = {'IN_CH': 128, 'OUT_CH': 128, 'FLEN': 8}
F = {'BASE_ADDR': 0x0640_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
 ↪'VSIZE': 1}
W = {'BASE_ADDR': 0x0300_0000, 'STRIDE_SIZE': int(128*128*9/4), 'HSIZE':␣
 ↪int(128*128*9/4), 'VSIZE': 4}
B = {'BASE_ADDR': 0x0390_0000, 'STRIDE_SIZE': 128, 'HSIZE': 128, 'VSIZE': 1}
R = {'BASE_ADDR': 0x0650_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
 ↪'VSIZE': 1}
SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
```

[24]: 1

[ ]:
```
a = 0x0650_0000
for i in range(int(128*8*8/4)):
    temp = SU.su_read_data(a + 4*i)
    print(i, "\t", temp)
```

[25]:
```
###################################################################
#        Max Pool 3
###################################################################
# Max Pooling
# - in:       (n. 128. 8. 8)
# - out:      (n, 128, 4, 4)
###################################################################
I = {'IN_CH': 128, 'FLEN': 8}
F = {'BASE_ADDR': 0x0650_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
 ↪'VSIZE': 1}
R = {'BASE_ADDR': 0x0660_0000, 'STRIDE_SIZE': 128*4*4, 'HSIZE': 128*4*4,␣
 ↪'VSIZE': 1}
SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
```

[25]: 1

[ ]:
```
a = 0x0660_0000
for i in range(int(128*4*4/4)):
    temp = SU.su_read_data(a + 4*i)
    print(i, "\t", temp)
```

[26]:
```
###################################################################
#        Convolution 5+ ReLU
###################################################################
# Convolution
# - in:       (n, 128, 4, 4)
# - out:      (n, 256, 4, 4)
# - weight: (256, 128, 3, 3)
# - bias:              (256)
# ReLU
# - in:       (n. 256. 4. 4)
# - out:      (n. 256. 4. 4)
###################################################################
I = {'IN_CH': 128, 'OUT_CH': 256, 'FLEN': 4}
F = {'BASE_ADDR': 0x0660_0000, 'STRIDE_SIZE': 128*4*4, 'HSIZE': 128*4*4,␣
 ↪'VSIZE': 1}
W = {'BASE_ADDR': 0x03A0_0000, 'STRIDE_SIZE': int(128*256*9/8), 'HSIZE':␣
 ↪int(128*256*9/8), 'VSIZE': 8}
B = {'BASE_ADDR': 0x03F0_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
R = {'BASE_ADDR': 0x0670_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
 ↪'VSIZE': 1}
SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
```

```
[26]: 1
```

```
[ ]: a = 0x0670_0000
     for i in range(int(256*4*4/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[27]: ###################################################################
      #        Convolution 6 + ReLU
      ###################################################################
      # Convolution
      # - in:       (n, 256, 4, 4)
      # - out:      (n, 256, 4, 4)
      # - weight:  (256, 256, 3, 3)
      # - bias:               (256)
      # ReLU
      # - in:       (n. 256. 4. 4)
      # - out:      (n. 256. 4. 4)
      ###################################################################
      I = {'IN_CH': 256, 'OUT_CH': 256, 'FLEN': 4}
      F = {'BASE_ADDR': 0x0670_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
       ↪'VSIZE': 1}
      W = {'BASE_ADDR': 0x0400_0000, 'STRIDE_SIZE': int(256*256*9/16), 'HSIZE':␣
       ↪int(256*256*9/16), 'VSIZE': 16}
      B = {'BASE_ADDR': 0x0490_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
      R = {'BASE_ADDR': 0x0680_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
       ↪'VSIZE': 1}
      SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
```

```
[27]: 1
```

```
[ ]: a = 0x0680_0000
     for i in range(int(256*4*4/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[28]: ################################################################
      #        Max Pool 4
      ################################################################
      # Max Pooling
      # - in:      (n. 256. 4. 4)
      # - out:     (n, 256, 2, 2)
      ################################################################
      I = {'IN_CH': 256, 'FLEN': 4}
      F = {'BASE_ADDR': 0x0680_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
       ↪'VSIZE': 1}
      R = {'BASE_ADDR': 0x0690_0000, 'STRIDE_SIZE': 256*2*2, 'HSIZE': 256*2*2,␣
       ↪'VSIZE': 1}
      SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
```

```
[28]: 1
```

```
[ ]: a = 0x0690_0000
     for i in range(int(256*2*2/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[29]: ################################################################
      #        Fully-Connected 1 + ReLU
      ################################################################
      # Fully-Connected
      # - in:             (1024,)
      # - out:             (256,)
      # - weight:     (256, 1024)
      # - bias:            (256,)
      # ReLU
      # - in:              (256,)
      # - out:             (256,)
      ################################################################
      F = {'BASE_ADDR': 0x0690_0000, 'STRIDE_SIZE': 1024, 'HSIZE': 1024, 'VSIZE': 1}
      W = {'BASE_ADDR': 0x0500_0000, 'STRIDE_SIZE': int(1024*256/8), 'HSIZE':␣
       ↪int(1024*256/8), 'VSIZE': 8}
      B = {'BASE_ADDR': 0x0530_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
      R = {'BASE_ADDR': 0x06A0_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
      SU.su_fc_control(F, W, B, R, VDMA0_BASE_ADDR, FC_BASE_ADDR)

      [0, 0, 0, 201]
```

```
[29]: 1
```

```
[ ]: a = 0x06A0_0000
     for i in range(int(256/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[30]: ##################################################################
      #        Fully-Connected 2 + ReLU
      ##################################################################
      # Fully-Connected
      # - in:             (256,)
      # - out:             (64,)
      # - weight:      (64, 256)
      # - bias:            (64,)
      # ReLU
      # - in:              (64,)
      # - out:             (64,)
      ##################################################################
      F = {'BASE_ADDR': 0x06A0_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
      W = {'BASE_ADDR': 0x0540_0000, 'STRIDE_SIZE': 256*64, 'HSIZE': 256*64, 'VSIZE':␣
       ↪1}
      B = {'BASE_ADDR': 0x0550_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
      R = {'BASE_ADDR': 0x06B0_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
      SU.su_fc_control(F, W, B, R, VDMA0_BASE_ADDR, FC_BASE_ADDR)
```

```
[0, 0, 0, 7]
```

```
[30]: 1
```

```
[ ]: a = 0x06B0_0000
     for i in range(int(64/4)):
         temp = SU.su_read_data(a + 4*i)
         print(i, "\t", temp)
```

```
[31]: ##################################################################
      #        Fully-Connected 3 + ReLU
      ##################################################################
      # Fully-Connected
      # - in:               (64,)
      # - out:              (10,)
      # - weight:         (10, 64)
      # - bias:             (10,)
      # ReLU
      # - in:               (10,)
      # - out:              (10,)
      ##################################################################
      F = {'BASE_ADDR': 0x06B0_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
```

```
W = {'BASE_ADDR': 0x0560_0000, 'STRIDE_SIZE': 640, 'HSIZE': 640, 'VSIZE': 1}
B = {'BASE_ADDR': 0x0570_0000, 'STRIDE_SIZE': 10, 'HSIZE': 10, 'VSIZE': 1}
R = {'BASE_ADDR': 0x06C0_0000, 'STRIDE_SIZE': 10, 'HSIZE': 10, 'VSIZE': 1}
SU.su_fc_control(F, W, B, R, VDMA0_BASE_ADDR, FC_BASE_ADDR)
```

```
[0, 0, 0, 4]
```

```
[31]: 1
```

```
[ ]: a = 0x06C0_0000
for i in range(int(3)):
    temp = SU.su_read_data(a + 4*i)
    print(i, "\t", temp)
```

```
[32]: ##########################################################################################
# Below code can be revised according to your apb register setting
##########################################################################################
# Read label index from apb register (our design output the index to that␣
 ↪address)

# We assign FC_BASE_ADDR + 0x20 apb register to return max-value index
label = SU.su_read_data(FC_BASE_ADDR + 0x20)
label = int.from_bytes(label, 'big', signed=True)
# Predicted (computated) label
print(label-1)
```

```
3
```

```
[33]: # Real value
print(y_test[0])
```

```
3
```

0.19.5 All Inference function

```
[34]: def inference(image_idx):
    I = {'IN_CH': 3, 'OUT_CH': 32, 'FLEN': 32}
    F = {'BASE_ADDR': 0x0000_0000 + 3072*image_idx, 'STRIDE_SIZE': 3*32*32,␣
 ↪'HSIZE': 3*32*32, 'VSIZE': 1}
    W = {'BASE_ADDR': 0x0200_0000, 'STRIDE_SIZE': 3*32*9, 'HSIZE': 3*32*9,␣
 ↪'VSIZE': 1}
    B = {'BASE_ADDR': 0x0210_0000, 'STRIDE_SIZE': 32, 'HSIZE': 32, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x0600_0000, 'STRIDE_SIZE': 32*32*32, 'HSIZE': 32*32*32,␣
 ↪'VSIZE': 1}
    SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
    I = {'IN_CH': 32, 'FLEN': 32}
    F = {'BASE_ADDR': 0x0600_0000, 'STRIDE_SIZE': 32*32*32, 'HSIZE': 32*32*32,␣
 ↪'VSIZE': 1}
```

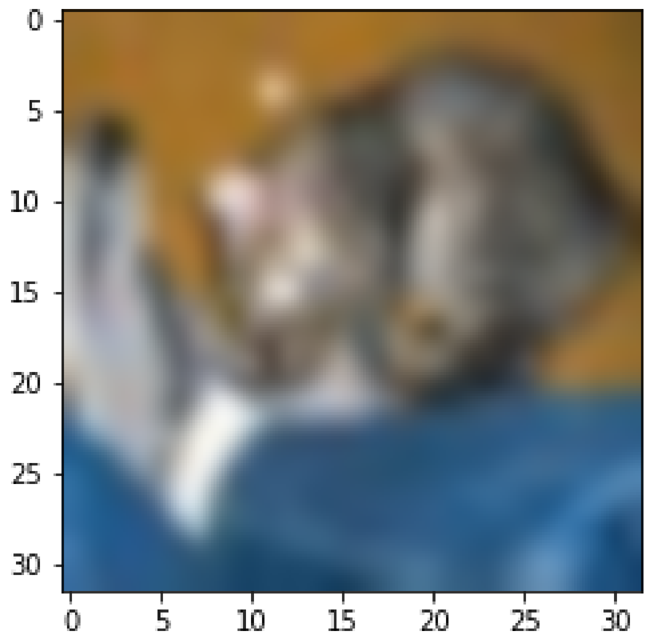
```
    R = {'BASE_ADDR': 0x0610_0000, 'STRIDE_SIZE': 32*16*16, 'HSIZE': 32*16*16,␣
↪'VSIZE': 1}
    SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
    I = {'IN_CH': 32, 'OUT_CH': 64, 'FLEN': 16}
    F = {'BASE_ADDR': 0x0610_0000, 'STRIDE_SIZE': 32*16*16, 'HSIZE': 32*16*16,␣
↪'VSIZE': 1}
    W = {'BASE_ADDR': 0x0220_0000, 'STRIDE_SIZE': 32*64*9, 'HSIZE': 32*64*9,␣
↪'VSIZE': 1}
    B = {'BASE_ADDR': 0x0270_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x0620_0000, 'STRIDE_SIZE': 64*16*16, 'HSIZE': 64*16*16,␣
↪'VSIZE': 1}
    SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
    I = {'IN_CH': 64, 'FLEN': 16}
    F = {'BASE_ADDR': 0x0620_0000, 'STRIDE_SIZE': 64*16*16, 'HSIZE': 64*16*16,␣
↪'VSIZE': 1}
    R = {'BASE_ADDR': 0x0630_0000, 'STRIDE_SIZE': 64*8*8, 'HSIZE': 64*8*8,␣
↪'VSIZE': 1}
    SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
    I = {'IN_CH': 64, 'OUT_CH': 128, 'FLEN': 8}
    F = {'BASE_ADDR': 0x0630_0000, 'STRIDE_SIZE': 64*8*8, 'HSIZE': 64*8*8,␣
↪'VSIZE': 1}
    W = {'BASE_ADDR': 0x0280_0000, 'STRIDE_SIZE': int(64*128*9/2), 'HSIZE':␣
↪int(64*128*9/2), 'VSIZE': 2}
    B = {'BASE_ADDR': 0x02C0_0000, 'STRIDE_SIZE': 128, 'HSIZE': 128, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x0640_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
↪'VSIZE': 1}
    SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
    I = {'IN_CH': 128, 'OUT_CH': 128, 'FLEN': 8}
    F = {'BASE_ADDR': 0x0640_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
↪'VSIZE': 1}
    W = {'BASE_ADDR': 0x0300_0000, 'STRIDE_SIZE': int(128*128*9/4), 'HSIZE':␣
↪int(128*128*9/4), 'VSIZE': 4}
    B = {'BASE_ADDR': 0x0390_0000, 'STRIDE_SIZE': 128, 'HSIZE': 128, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x0650_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
↪'VSIZE': 1}
    SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
    I = {'IN_CH': 128, 'FLEN': 8}
    F = {'BASE_ADDR': 0x0650_0000, 'STRIDE_SIZE': 128*8*8, 'HSIZE': 128*8*8,␣
↪'VSIZE': 1}
    R = {'BASE_ADDR': 0x0660_0000, 'STRIDE_SIZE': 128*4*4, 'HSIZE': 128*4*4,␣
↪'VSIZE': 1}
    SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
    I = {'IN_CH': 128, 'OUT_CH': 256, 'FLEN': 4}
    F = {'BASE_ADDR': 0x0660_0000, 'STRIDE_SIZE': 128*4*4, 'HSIZE': 128*4*4,␣
↪'VSIZE': 1}
```

```
    W = {'BASE_ADDR': 0x03A0_0000, 'STRIDE_SIZE': int(128*256*9/8), 'HSIZE':␣
↪int(128*256*9/8), 'VSIZE': 8}
    B = {'BASE_ADDR': 0x03F0_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x0670_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
↪'VSIZE': 1}
    SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
    I = {'IN_CH': 256, 'OUT_CH': 256, 'FLEN': 4}
    F = {'BASE_ADDR': 0x0670_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
↪'VSIZE': 1}
    W = {'BASE_ADDR': 0x0400_0000, 'STRIDE_SIZE': int(256*256*9/16), 'HSIZE':␣
↪int(256*256*9/16), 'VSIZE': 16}
    B = {'BASE_ADDR': 0x0490_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x0680_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
↪'VSIZE': 1}
    SU.su_conv_control(I, F, W, B, R, VDMA1_BASE_ADDR, CONV_BASE_ADDR)
    I = {'IN_CH': 256, 'FLEN': 4}
    F = {'BASE_ADDR': 0x0680_0000, 'STRIDE_SIZE': 256*4*4, 'HSIZE': 256*4*4,␣
↪'VSIZE': 1}
    R = {'BASE_ADDR': 0x0690_0000, 'STRIDE_SIZE': 256*2*2, 'HSIZE': 256*2*2,␣
↪'VSIZE': 1}
    SU.su_pool_control(I, F, R, VDMA2_BASE_ADDR, POOL_BASE_ADDR)
    F = {'BASE_ADDR': 0x0690_0000, 'STRIDE_SIZE': 1024, 'HSIZE': 1024, 'VSIZE':␣
↪1}
    W = {'BASE_ADDR': 0x0500_0000, 'STRIDE_SIZE': int(1024*256/8), 'HSIZE':␣
↪int(1024*256/8), 'VSIZE': 8}
    B = {'BASE_ADDR': 0x0530_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x06A0_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
    SU.su_fc_control(F, W, B, R, VDMA0_BASE_ADDR, FC_BASE_ADDR)
    F = {'BASE_ADDR': 0x06A0_0000, 'STRIDE_SIZE': 256, 'HSIZE': 256, 'VSIZE': 1}
    W = {'BASE_ADDR': 0x0540_0000, 'STRIDE_SIZE': 256*64, 'HSIZE': 256*64,␣
↪'VSIZE': 1}
    B = {'BASE_ADDR': 0x0550_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x06B0_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
    SU.su_fc_control(F, W, B, R, VDMA0_BASE_ADDR, FC_BASE_ADDR)
    F = {'BASE_ADDR': 0x06B0_0000, 'STRIDE_SIZE': 64, 'HSIZE': 64, 'VSIZE': 1}
    W = {'BASE_ADDR': 0x0560_0000, 'STRIDE_SIZE': 640, 'HSIZE': 640, 'VSIZE': 1}
    B = {'BASE_ADDR': 0x0570_0000, 'STRIDE_SIZE': 10, 'HSIZE': 10, 'VSIZE': 1}
    R = {'BASE_ADDR': 0x06C0_0000, 'STRIDE_SIZE': 10, 'HSIZE': 10, 'VSIZE': 1}
    SU.su_fc_control(F, W, B, R, VDMA0_BASE_ADDR, FC_BASE_ADDR)
   ␣
↪#######################################################################################
    # Below code can be revised according to your apb register setting
   ␣
↪#######################################################################################
    label = SU.su_read_data(FC_BASE_ADDR + 0x20)
    label = int.from_bytes(label, 'big', signed=True)
    # print(label-1)
    return (label-1)
```

0.19.6 Check accuracy

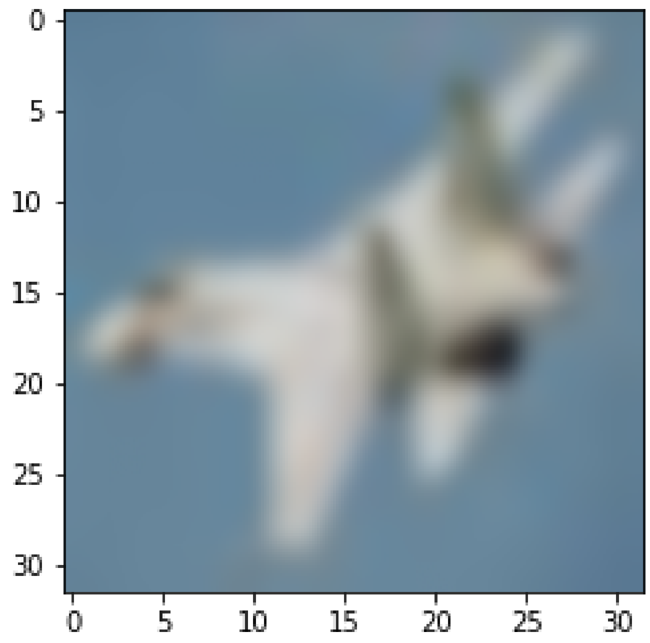
[35]:
```
acc = 0
for i in range(100):
    pred = inference(i)
    if pred == y_test[i]:
        acc += 1
    print("Progress: {:05.2f}%".format(100*i/100), end="\r", flush=True)
    # show sample image and predict result
    if i % 10 == 0:
        gen_image(X_test_origin[i]).show()
        print(pred)
        print("Label: %d (%s)" %(y_test[i], label_list[y_test[i]]))
        print("Predict: %d (%s)" %(pred, label_list[pred]))
print("\t100 images accuracy: {:.2f}%".format(acc/100 * 100))
```

```
[0, 0, 0, 201]
[0, 0, 0, 7]
[0, 0, 0, 4]
Progress: 00.00%
```



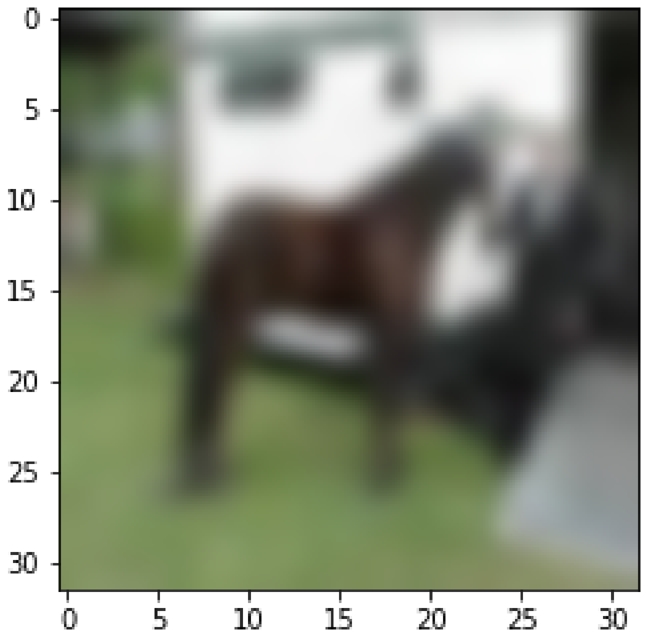
```
3
Label: 3 (Cat)
```

```
Predict: 3 (Cat)
[0, 0, 0, 210]
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 210]0%
[0, 0, 0, 46]
[0, 0, 0, 1]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 108]0%
[0, 0, 0, 31]
[0, 0, 0, 2]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 194]0%
[0, 0, 0, 2]
[0, 0, 0, 4]
[0, 0, 0, 108]0%
[0, 0, 0, 31]
[0, 0, 0, 10]
[0, 0, 0, 194]0%
[0, 0, 0, 18]
[0, 0, 0, 1]
Progress: 10.00%
```



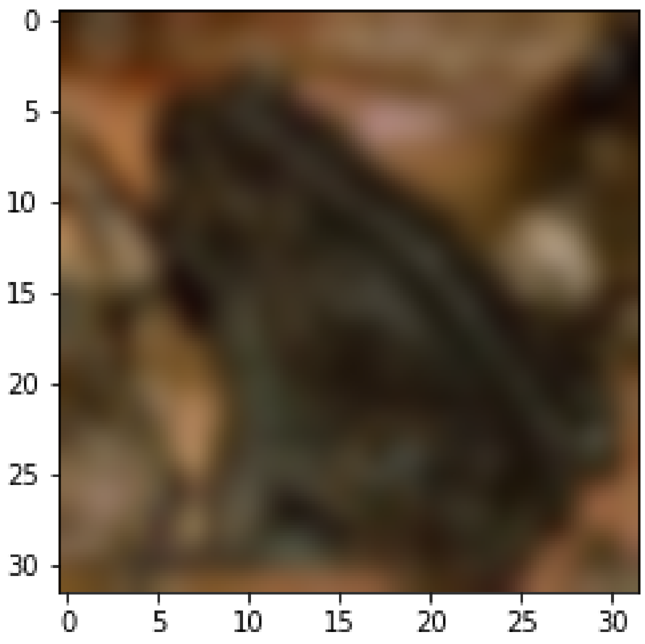
```
0
Label: 0 (Airplane)
Predict: 0 (Airplane)
[0, 0, 0, 100]
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 201]0%
[0, 0, 0, 52]
[0, 0, 0, 6]
[0, 0, 0, 25]00%
[0, 0, 0, 3]
[0, 0, 0, 8]
[0, 0, 0, 239]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 48]00%
[0, 0, 0, 52]
[0, 0, 0, 6]
[0, 0, 0, 104]0%
[0, 0, 0, 18]
[0, 0, 0, 8]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
```

```
[0, 0, 0, 9]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
Progress: 20.00%
```



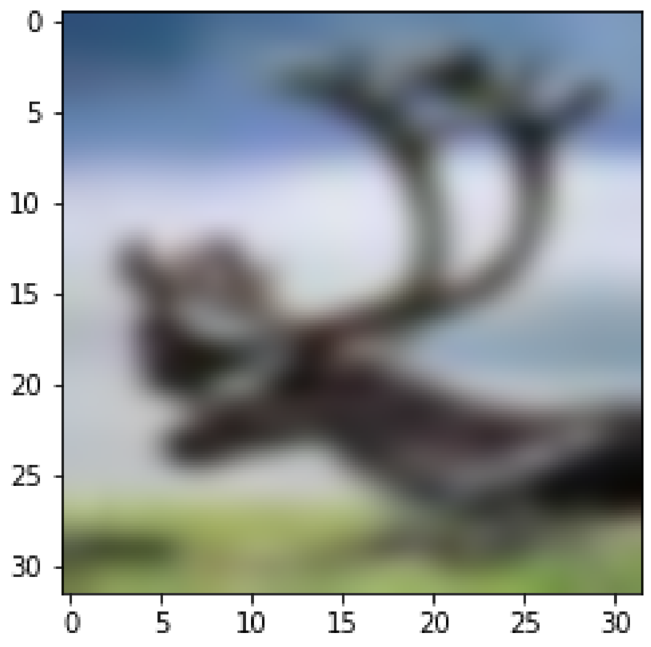
```
7
Label: 7 (Horse)
Predict: 7 (Horse)
[0, 0, 0, 201]
[0, 0, 0, 46]
[0, 0, 0, 3]
[0, 0, 0, 127]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
[0, 0, 0, 239]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 210]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
[0, 0, 0, 210]0%
[0, 0, 0, 18]
```

```
[0, 0, 0, 3]
[0, 0, 0, 201]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
[0, 0, 0, 29]00%
[0, 0, 0, 46]
[0, 0, 0, 1]
[0, 0, 0, 239]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
Progress: 30.00%
```

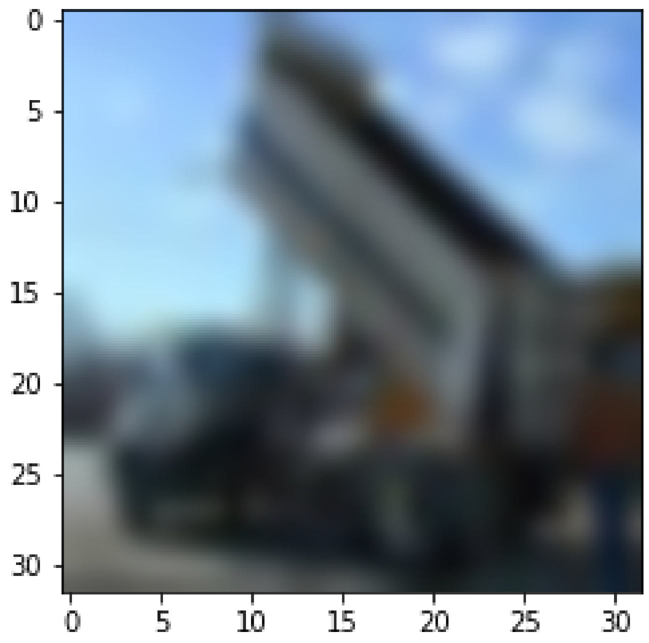


```
6
Label: 6 (Frog)
Predict: 6 (Frog)
[0, 0, 0, 201]
[0, 0, 0, 52]
[0, 0, 0, 6]
[0, 0, 0, 201]0%
[0, 0, 0, 18]
```

```
[0, 0, 0, 5]
[0, 0, 0, 201]0%
[0, 0, 0, 18]
[0, 0, 0, 6]
[0, 0, 0, 100]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 210]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 106]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
[0, 0, 0, 234]0%
[0, 0, 0, 31]
[0, 0, 0, 10]
[0, 0, 0, 230]0%
[0, 0, 0, 1]
[0, 0, 0, 10]
[0, 0, 0, 48]00%
[0, 0, 0, 52]
[0, 0, 0, 6]
[0, 0, 0, 210]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
Progress: 40.00%
```

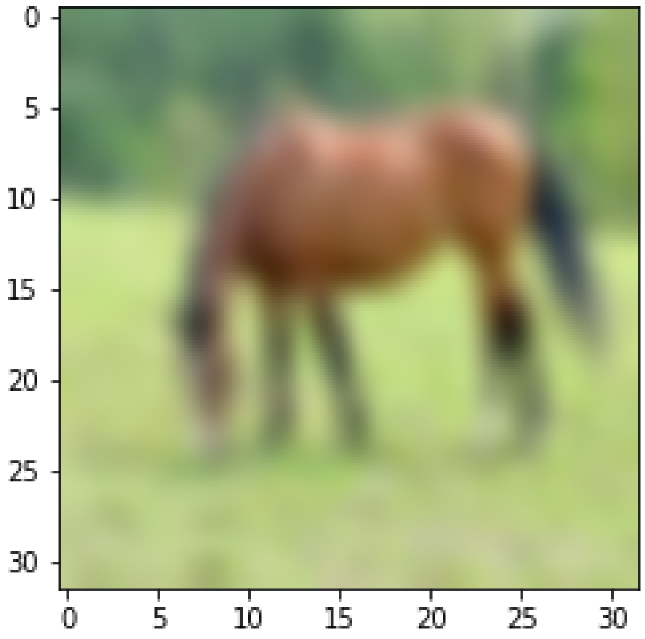


```
4
Label: 4 (Deer)
Predict: 4 (Deer)
[0, 0, 0, 201]
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 25]00%
[0, 0, 0, 52]
[0, 0, 0, 6]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 89]00%
[0, 0, 0, 46]
[0, 0, 0, 1]
[0, 0, 0, 100]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 201]0%
[0, 0, 0, 2]
[0, 0, 0, 4]
[0, 0, 0, 210]0%
[0, 0, 0, 31]
[0, 0, 0, 9]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 239]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
Progress: 50.00%
```



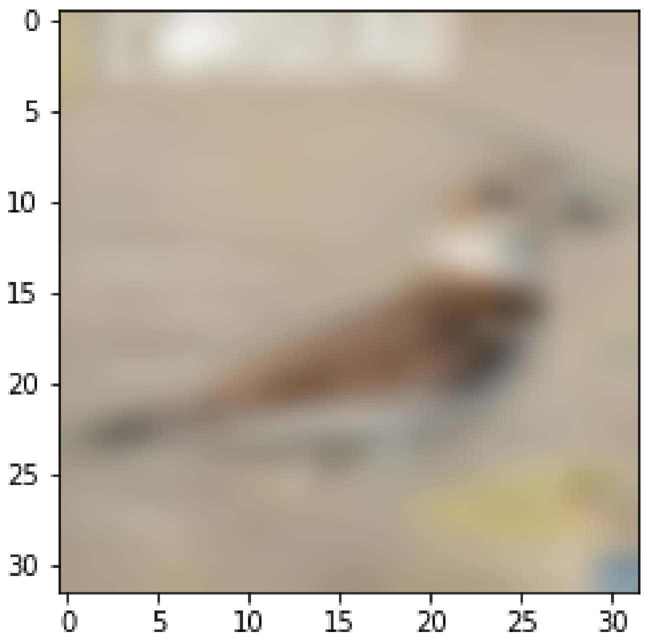
```
9
Label: 9 (Truck)
Predict: 9 (Truck)
[0, 0, 0, 227]
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 210]0%
[0, 0, 0, 2]
[0, 0, 0, 6]
[0, 0, 0, 201]0%
[0, 0, 0, 38]
[0, 0, 0, 4]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 5]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 194]0%
[0, 0, 0, 52]
```

```
[0, 0, 0, 6]
[0, 0, 0, 201]0%
[0, 0, 0, 38]
[0, 0, 0, 7]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
Progress: 60.00%
```



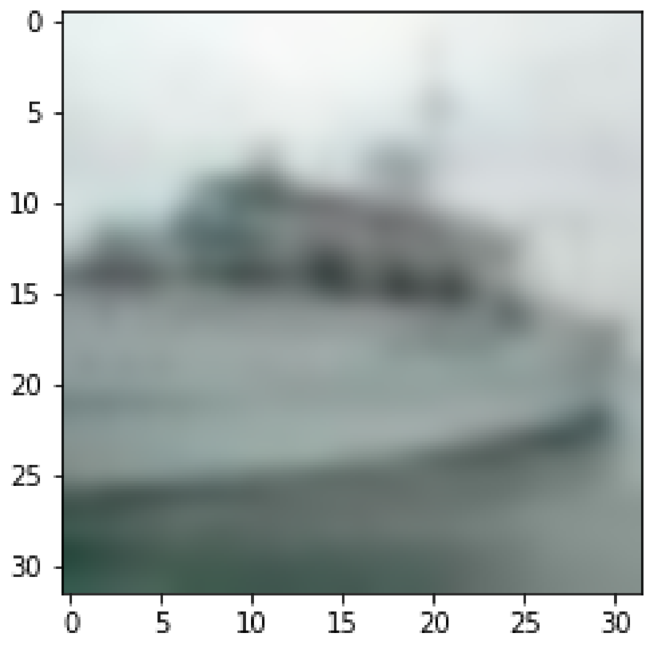
```
7
Label: 7 (Horse)
Predict: 7 (Horse)
[0, 0, 0, 201]
[0, 0, 0, 2]
[0, 0, 0, 6]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 210]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 201]0%
[0, 0, 0, 18]
```

```
[0, 0, 0, 3]
[0, 0, 0, 108]0%
[0, 0, 0, 31]
[0, 0, 0, 9]
[0, 0, 0, 201]0%
[0, 0, 0, 46]
[0, 0, 0, 3]
[0, 0, 0, 210]0%
[0, 0, 0, 49]
[0, 0, 0, 6]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
[0, 0, 0, 210]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
Progress: 70.00%
```

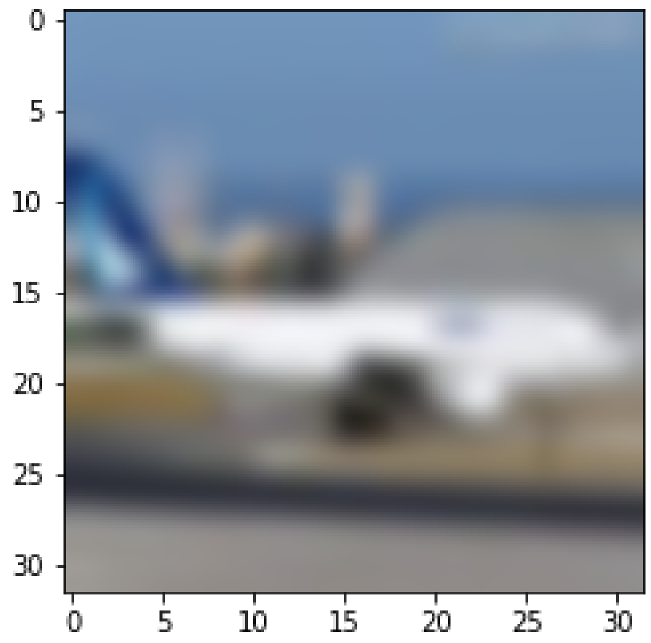


```
4
Label: 2 (Bird)
Predict: 4 (Deer)
[0, 0, 0, 201]
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
```

```
[0, 0, 0, 9]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 100]0%
[0, 0, 0, 46]
[0, 0, 0, 10]
[0, 0, 0, 1].00%
[0, 0, 0, 32]
[0, 0, 0, 3]
[0, 0, 0, 100]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 4]
[0, 0, 0, 201]0%
[0, 0, 0, 52]
[0, 0, 0, 6]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
Progress: 80.00%
```



```
8
Label: 8 (Ship)
Predict: 8 (Ship)
[0, 0, 0, 210]
[0, 0, 0, 31]
[0, 0, 0, 9]
[0, 0, 0, 108]0%
[0, 0, 0, 31]
[0, 0, 0, 2]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 210]0%
[0, 0, 0, 18]
[0, 0, 0, 6]
[0, 0, 0, 210]0%
[0, 0, 0, 18]
[0, 0, 0, 7]
[0, 0, 0, 89]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
[0, 0, 0, 227]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 100]0%
[0, 0, 0, 6]
[0, 0, 0, 10]
[0, 0, 0, 110]0%
[0, 0, 0, 46]
[0, 0, 0, 1]
Progress: 90.00%
```



```
0
Label: 0 (Airplane)
Predict: 0 (Airplane)
[0, 0, 0, 201]
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 210]0%
[0, 0, 0, 24]
[0, 0, 0, 9]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 194]0%
[0, 0, 0, 18]
[0, 0, 0, 5]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 201]0%
[0, 0, 0, 7]
[0, 0, 0, 7]
[0, 0, 0, 210]0%
[0, 0, 0, 46]
[0, 0, 0, 5]
[0, 0, 0, 29]00%
[0, 0, 0, 46]
```

```
[0, 0, 0, 1]
[0, 0, 0, 25]00%
[0, 0, 0, 18]
[0, 0, 0, 8]
        100 images accuracy: 77.00%
```

```
[ ]:
```