In [1]:
```
Name = 'Ati tesakulsiri'
ID = 'st123009'
```

Lab 01 Report

---

0) Objective

- Learning to train and evaluate the PyTorch AlexNet model on the CIFAR-10 dataset.
- Learn how the parameter work.

---

1) Introduction

The 2012 Imagenet large-scale visual recognition competition was won by Alexnet. The approach was put forth by Alex Krizhevsky and his colleagues in their 2012 research publication, Imagenet Classification with Deep Convolution Neural Network.

They discovered that the training process was nearly six times faster when the relu was used as an activation function. Additionally, they made use of dropout layers, which stopped their model from overfitting. Additionally, the Imagenet dataset is used to train the model. There are almost a thousand classes and nearly 14 million photos in the Imagenet collection.

- ### 1.1 RelU
  - The formula tanh is used to represent a neuron's output (f) as a function of its input. These saturating nonlinearities train with gradient descent far more slowly than the non-saturating nonlinearity. We term AI neurons with this nonlinearity as "rectified linear units," following Nair and Hinton (ReLUs). Deep convolutional neural networks that use ReLUs can learn much more quickly than those that use tanh units.

---

2) Lab method

parameter and hyper paramenter we use

- for img augmentation
  - we perform `Resize((70, 70))`,
  - `RandomHorizontalFlip()`
  - `RandomCrop(64)` for trainset of data
  -
  - we perform `Resize((70, 70))`,

    - `RandomHorizontalFlip()`
    - `CenterCrop(64)` for testset of data

- for Alexnet model we change the last output size to match the num output

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4),
padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

- Number of data to train,val,test
    - we train with 40000 image,
    - 10000 val,
    - 10000 test image
    - with 256 batch sizze
- Hyper parameter
    - Here is our hyper parameter set in pytorch

```
criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(alexnet.parameters(),lr =
0.005,momentum=.9)
```

```
alexnet.to(device)
num_epoch = 80
```

2.1 Setting

In [2]:

```
# Alexnet lab learning
# setting
import torch
import urllib
import torchvision
import os
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
os.environ['http_proxy'] = 'http://192.41.170.23:3128'
os.environ['https_proxy'] = 'http://192.41.170.23:3128'
```

In [3]:

```
# check the puffer nvidia cuda available
!nvidia-smi
```

```
Thu Jan 19 13:58:17 2023
+--------------------------------------------------------------------------
---+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6
|
|-------------------------------+----------------------+-------------------
---+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. E
CC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute 
M. |
|                               |                      |               MIG 
M. |
|===============================+======================+===================
===|
|   0  NVIDIA GeForce ...  On   | 00000000:84:00.0 Off |
N/A |
| 24%   39C    P2    61W / 250W |   3202MiB / 11264MiB |      6%      Defau
lt |
|                               |                      |
N/A |
+-------------------------------+----------------------+-------------------
---+
|   1  NVIDIA GeForce ...  On   | 00000000:85:00.0 Off |
N/A |
| 22%   28C    P8     1W / 250W |      3MiB / 11264MiB |      0%      Defau
lt |
|                               |                      |
N/A |
+-------------------------------+----------------------+-------------------
---+
|   2  NVIDIA GeForce ...  On   | 00000000:88:00.0 Off |
N/A |
| 22%   28C    P8     5W / 250W |      3MiB / 11264MiB |      0%      Defau
lt |
|                               |                      |
N/A |
+-------------------------------+----------------------+-------------------
---+
|   3  NVIDIA GeForce ...  On   | 00000000:89:00.0 Off |
N/A |
| 22%   27C    P8     4W / 250W |      3MiB / 11264MiB |      0%      Defau
lt |
|                               |                      |
N/A |
+-------------------------------+----------------------+-------------------
---+

+--------------------------------------------------------------------------
---+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                  GPU Memo
ry |
|        ID   ID                                                   Usage
|
|==========================================================================
===|
|    0   N/A  N/A      4238      C                                     161M
iB |
|    0   N/A  N/A      4491      C                                     161M
```

```
iB |
|    0   N/A  N/A      8554      C                                   161M
iB |
|    0   N/A  N/A      9540      C                                   161M
iB |
|    0   N/A  N/A     14619      C                                  2233M
iB |
|    0   N/A  N/A     30768      C                                   161M
iB |
|    0   N/A  N/A     32155      C                                   161M
iB |
+-----------------------------------------------------------------------------
---+
```

2.2 Modeling

After finish the setting now let load the non train alexnet template.

Incase we build with pytorch the model should be create manually like this

```
features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4),
padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

In this lab I will load the template directly from pytorch.

In [4]:

```
alexnet = torch.hub.load('pytorch/vision:v0.11.2', 'alexnet', pretrained=Fal
alexnet.classifier = torch.nn.Sequential(
    torch.nn.Dropout(0.5),
    torch.nn.Linear(9216,4096 ), #why like this?  Because Chaky tried alreac
    torch.nn.ReLU(inplace=True),
    torch.nn.Dropout(0.5),
    torch.nn.Linear(4096, 4096),
    torch.nn.ReLU(inplace= True),
    torch.nn.Linear(4096,10))
```

```
Using cache found in /root/.cache/torch/hub/pytorch_vision_v0.11.2
```

In [5]:

```
alexnet
```

Out[5]:

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mod
e=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mod
e=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mo
de=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

2.3 Dataset loading

- Ref for upsize of the imgage (https://github.com/rasbt)

In [32]:

```
train_transform = transforms.Compose([
    transforms.Resize((70, 70)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(64),
    transforms.ToTensor(),

])


test_transform = transforms.Compose([
    transforms.Resize((70, 70)),
    transforms.CenterCrop(64),
    transforms.ToTensor(),

])
```

In [33]:
```
trainset = torchvision.datasets.CIFAR10(root='data_keep', train=True,
                                        download=True, transform=train_trans


testset = torchvision.datasets.CIFAR10(root='data_keep', train=False,
                                       download=True, transform=test_transfc
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [34]:
```
train_set, val_set = torch.utils.data.random_split(trainset, [40000, 10000])
```

In [35]:
```
batch_size = 256
train_loader = DataLoader(train_set,batch_size=batch_size,   shuffle=True)
val_loader   = DataLoader(val_set,  batch_size=batch_size,   shuffle=True)
test_loader  = DataLoader(testset, batch_size=batch_size,   shuffle=False)
```

In [36]:
```
device = torch.device('cuda:2' if torch.cuda.is_available() else 'cpu')
device
```

Out[36]:
```
device(type='cuda', index=2)
```

2.4 Training

In [37]:
```
NEW_TRAIN = False
```

In [42]:
```
if NEW_TRAIN:
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(alexnet.parameters(),lr = 0.005,momentum=.9)
    alexnet.to(device)
    num_epoch = 80
    val_old_loss = float("Inf")
    train_loss_log = []
    train_acc_log = []
    val_acc_log = []
    val_loss_log = []
    filepath = "root/models/cifaralex2.pt"
    for e in range(num_epoch):
        total_train_corr = 0
        alexnet.train()
        for batch, (image, label) in enumerate(train_loader):
            image = image.to(device)
            label = label.to(device)
            optimizer.zero_grad()#3. clear gradients

            yhat  = alexnet(image)           #1. predict  yhat shape(100, 10
            loss  = criterion(yhat, label) #2. loss

            #add accuracy
            predicted  = torch.max(yhat, 1)[1]
            batch_train_corr = (predicted == label).sum()
            total_train_corr += batch_train_corr
            acc = (total_train_corr * 100) / ((batch_size) * (batch + 1))
            loss.backward() #4. backpropagate
            optimizer.step() #5. update

            if (batch + 1) % 60 == 0:
                # sys.stdout.write(f"\rBatch: {batch+1} - Loss: {loss}")
                print(f"Epoch: {e} - Batch: {batch+1:3.0f} - Loss: {loss:.2f
        train_loss_log.append(loss)
        train_acc_log.append(acc)
        #after each epoch, calculate the validation acc and loss
        with torch.no_grad():
            alexnet.eval()
            total_val_corr = 0
            for (val_image, val_label) in val_loader:
                val_image = val_image.to(device)
                val_label = val_label.to(device)
                val_yhat = alexnet(val_image)
                val_loss = criterion(val_yhat, val_label)
                #save the model with the lowest loss
                if val_loss < val_old_loss:
                    torch.save(alexnet.state_dict(), filepath) #state_dict i
                    val_old_loss = val_loss
                val_predicted = torch.max(val_yhat, 1)[1]
                total_val_corr += (val_predicted == val_label).sum()
            val_acc = (total_val_corr * 100) / len(val_set)
            print(f"++++++Validation++++++  Loss: {val_loss:.2f} - Acc: {val
            val_acc_log.append(val_acc)
            val_loss_log.append(val_loss)
else:
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(alexnet.parameters(),lr = 0.005,momentum=.9)
    alexnet.to(device)
    alexnet.load_state_dict(torch.load("root/models/cifaralex2.pt"))
```

In [43]:

```
import pickle
if NEW_TRAIN:
    to_save = (train_loss_log,train_acc_log,val_loss_log,val_acc_log)

    with open('root/models/lossacc_log2.atikeep', 'wb') as handle:
        pickle.dump(to_save, handle)
else:
    with open('root/models/lossacc_log2.atikeep', 'rb') as handle:
        train_loss_log,train_acc_log,val_loss_log,val_acc_log = pickle.load(
```

3) Result

- ## 3.1 Result From first 37 epoch

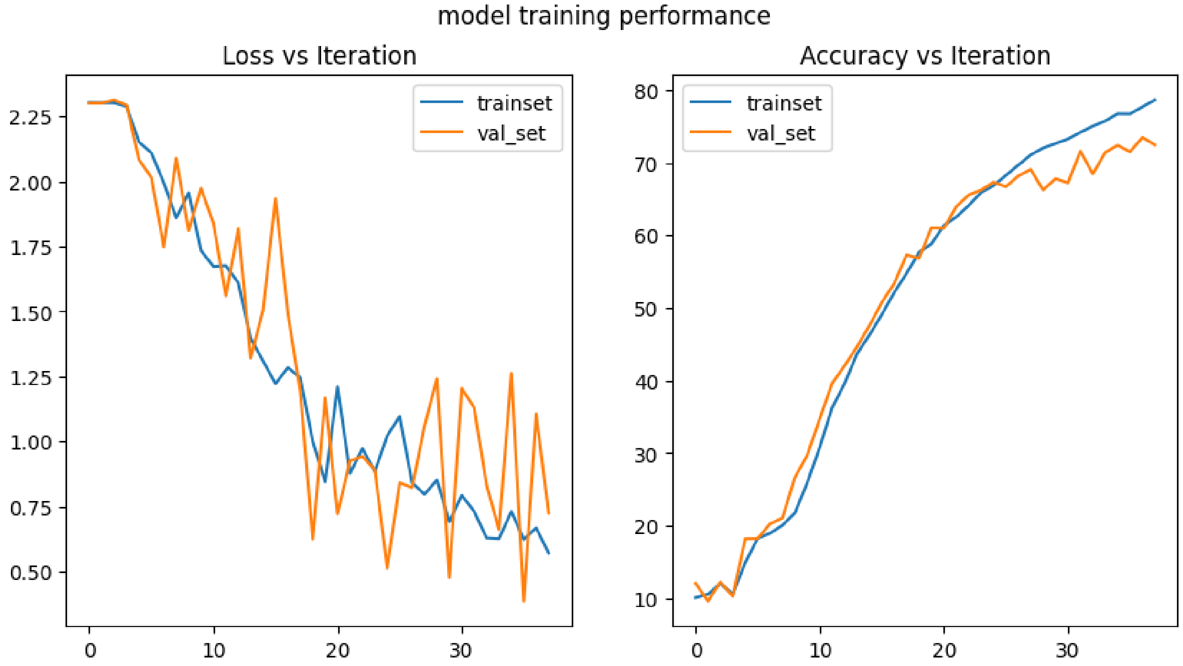
  - the loss is decreasing and the accuracy is around 75-76% with Lr = 0.005

In [44]:

```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2,figsize = (10,5))
fig.suptitle('model training performance')
ax1.plot(torch.Tensor(train_loss_log).cpu(),label = 'trainset')
ax1.plot(torch.Tensor(val_loss_log).cpu(),label = 'val_set')
ax1.legend()
ax1.set_title('Loss vs Iteration')
```

```
ax2.plot(torch.Tensor(train_acc_log).cpu(),label = 'trainset')
ax2.plot(torch.Tensor(val_acc_log).cpu(),label = 'val_set')
ax2.legend()
ax2.set_title('Accuracy vs Iteration')
plt.show()
```



- ### 3.2 Testing the model with test set

In [90]:
```
alexnet.eval()
corr_log = []
wrong_log = []
with torch.no_grad():
    test_corr = 0
    for test_image, test_label in test_loader:
        test_image = test_image.to(device)
        test_label = test_label.to(device)
        test_yhat = alexnet(test_image)
        test_loss = criterion(test_yhat, test_label)
        test_predicted = torch.max(test_yhat, 1)[1]
        if len(corr_log) < 6:
            if test_predicted[0] == test_label[0]:
                corr_log.append((test_image[0],test_predicted[0]))
        if len(wrong_log) < 6:
            if test_predicted[0] != test_label[0]:
                wrong_log.append((test_image[0],test_predicted[0]))
        test_corr += (test_predicted == test_label).sum()
    test_acc = (test_corr * 100) / len(testset)

print(test_acc)
```

```
tensor(72.4500, device='cuda:2')
```

In [91]:
```
# print(test_predicted[0],test_label[0])
len(wrong_log),len(corr_log)
```

Out[91]:
```
(6, 6)
```

In [92]:
```
print(f'Test accuracy = {"%.2f" %test_acc}')
```

```
Test accuracy = 72.45
```

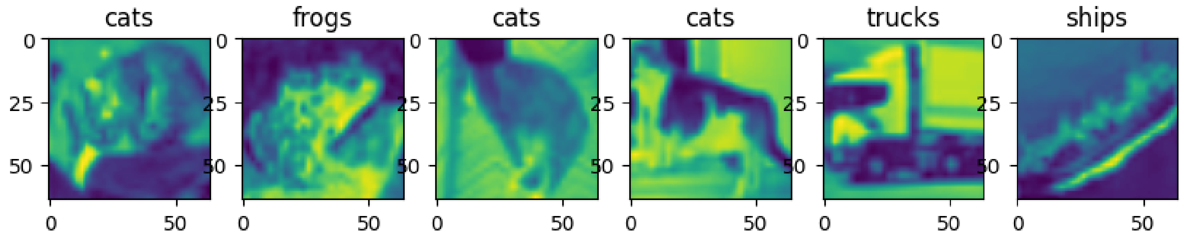
- ### 3.3 Example of some correct and wrong prediction

In [93]:

```
lab_dic = {0:'airplanes', 1:'cars', 2:'birds', 3:'cats', 4:'deer', 5:'dogs',
```

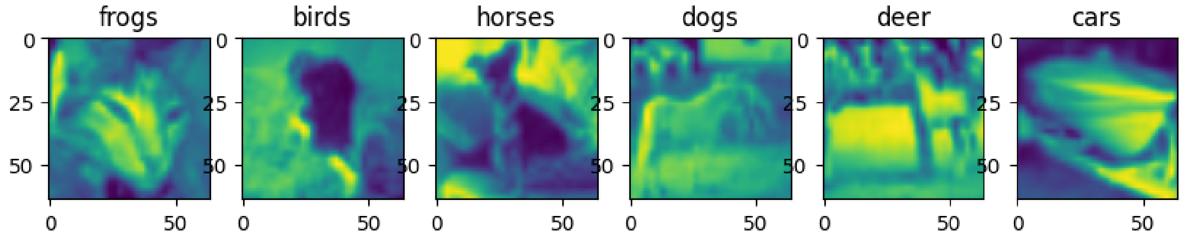
In [113…

```
fig,plat = plt.subplots(1,6,figsize=(10,3))
# print(len(plat))
for (i,l),pla in zip(corr_log,plat):
    pla.imshow(i.cpu()[0])
    pla.set_title(lab_dic[l.cpu().item()])
plt.show()
```



In [114…

```
fig,plat = plt.subplots(1,6,figsize=(10,3))
# print(len(plat))
for (i,l),pla in zip(wrong_log,plat):
    pla.imshow(i.cpu()[0])
    pla.set_title(lab_dic[l.cpu().item()])
plt.show()
```



4) Conclusion

- In this lab we use Alexnet argitechure to train CIFAR-10 Dataset, the result of loss function and accuracy can be observe in the Section 3.1.
- With 37 iteration of training, we manage to get 72.45 % accuracy with setting above.
- The wrong labeling is quite close to the answer ,figure above, if we train with lower Learning rate we might be able to get better performance.
    - Since we need to shared the resource, we quite happy with this result.

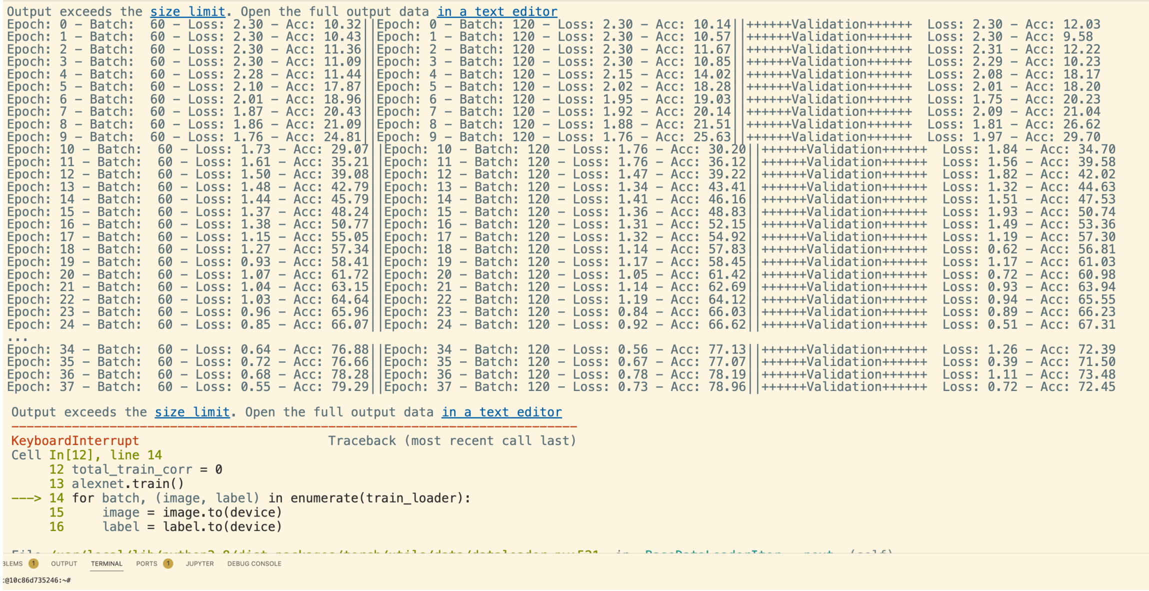
- ### Future work

    - Train more to get better result.

- ### Reference

- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. https://doi.org/10.1145/3065386

appendix



```
Output exceeds the size limit. Open the full output data in a text editor
Epoch: 0 - Batch:  60 - Loss: 2.30 - Acc: 10.32||Epoch: 0 - Batch: 120 - Loss: 2.30 - Acc: 10.14||++++++Validation++++++  Loss: 2.30 - Acc: 12.03
Epoch: 1 - Batch:  60 - Loss: 2.30 - Acc: 10.43||Epoch: 1 - Batch: 120 - Loss: 2.30 - Acc: 10.57||++++++Validation++++++  Loss: 2.30 - Acc: 9.58
Epoch: 2 - Batch:  60 - Loss: 2.30 - Acc: 11.36||Epoch: 2 - Batch: 120 - Loss: 2.30 - Acc: 11.67||++++++Validation++++++  Loss: 2.31 - Acc: 12.22
Epoch: 3 - Batch:  60 - Loss: 2.30 - Acc: 11.09||Epoch: 3 - Batch: 120 - Loss: 2.30 - Acc: 10.85||++++++Validation++++++  Loss: 2.29 - Acc: 10.23
Epoch: 4 - Batch:  60 - Loss: 2.28 - Acc: 11.44||Epoch: 4 - Batch: 120 - Loss: 2.15 - Acc: 14.02||++++++Validation++++++  Loss: 2.08 - Acc: 18.17
Epoch: 5 - Batch:  60 - Loss: 2.10 - Acc: 17.87||Epoch: 5 - Batch: 120 - Loss: 2.02 - Acc: 18.28||++++++Validation++++++  Loss: 2.01 - Acc: 18.20
Epoch: 6 - Batch:  60 - Loss: 2.01 - Acc: 18.96||Epoch: 6 - Batch: 120 - Loss: 1.95 - Acc: 19.03||++++++Validation++++++  Loss: 1.75 - Acc: 20.23
Epoch: 7 - Batch:  60 - Loss: 1.87 - Acc: 20.43||Epoch: 7 - Batch: 120 - Loss: 1.92 - Acc: 20.14||++++++Validation++++++  Loss: 2.09 - Acc: 21.04
Epoch: 8 - Batch:  60 - Loss: 1.86 - Acc: 21.09||Epoch: 8 - Batch: 120 - Loss: 1.88 - Acc: 21.51||++++++Validation++++++  Loss: 1.81 - Acc: 26.62
Epoch: 9 - Batch:  60 - Loss: 1.76 - Acc: 24.81||Epoch: 9 - Batch: 120 - Loss: 1.76 - Acc: 25.63||++++++Validation++++++  Loss: 1.97 - Acc: 29.70
Epoch: 10 - Batch:  60 - Loss: 1.73 - Acc: 29.07||Epoch: 10 - Batch: 120 - Loss: 1.76 - Acc: 30.20||++++++Validation++++++  Loss: 1.84 - Acc: 34.70
Epoch: 11 - Batch:  60 - Loss: 1.61 - Acc: 35.21||Epoch: 11 - Batch: 120 - Loss: 1.76 - Acc: 36.12||++++++Validation++++++  Loss: 1.56 - Acc: 39.58
Epoch: 12 - Batch:  60 - Loss: 1.50 - Acc: 39.08||Epoch: 12 - Batch: 120 - Loss: 1.47 - Acc: 39.22||++++++Validation++++++  Loss: 1.82 - Acc: 42.02
Epoch: 13 - Batch:  60 - Loss: 1.48 - Acc: 42.79||Epoch: 13 - Batch: 120 - Loss: 1.34 - Acc: 43.41||++++++Validation++++++  Loss: 1.32 - Acc: 44.63
Epoch: 14 - Batch:  60 - Loss: 1.44 - Acc: 45.79||Epoch: 14 - Batch: 120 - Loss: 1.41 - Acc: 46.16||++++++Validation++++++  Loss: 1.51 - Acc: 47.53
Epoch: 15 - Batch:  60 - Loss: 1.37 - Acc: 48.24||Epoch: 15 - Batch: 120 - Loss: 1.36 - Acc: 48.83||++++++Validation++++++  Loss: 1.93 - Acc: 50.74
Epoch: 16 - Batch:  60 - Loss: 1.38 - Acc: 50.77||Epoch: 16 - Batch: 120 - Loss: 1.31 - Acc: 52.15||++++++Validation++++++  Loss: 1.49 - Acc: 53.36
Epoch: 17 - Batch:  60 - Loss: 1.15 - Acc: 55.05||Epoch: 17 - Batch: 120 - Loss: 1.32 - Acc: 54.92||++++++Validation++++++  Loss: 1.19 - Acc: 57.30
Epoch: 18 - Batch:  60 - Loss: 1.27 - Acc: 57.34||Epoch: 18 - Batch: 120 - Loss: 1.14 - Acc: 57.83||++++++Validation++++++  Loss: 0.62 - Acc: 56.81
Epoch: 19 - Batch:  60 - Loss: 0.93 - Acc: 58.41||Epoch: 19 - Batch: 120 - Loss: 1.17 - Acc: 58.45||++++++Validation++++++  Loss: 1.17 - Acc: 61.03
Epoch: 20 - Batch:  60 - Loss: 1.07 - Acc: 61.72||Epoch: 20 - Batch: 120 - Loss: 1.05 - Acc: 61.42||++++++Validation++++++  Loss: 0.72 - Acc: 60.98
Epoch: 21 - Batch:  60 - Loss: 1.04 - Acc: 63.15||Epoch: 21 - Batch: 120 - Loss: 1.14 - Acc: 62.69||++++++Validation++++++  Loss: 0.93 - Acc: 63.94
Epoch: 22 - Batch:  60 - Loss: 1.03 - Acc: 64.64||Epoch: 22 - Batch: 120 - Loss: 1.19 - Acc: 64.12||++++++Validation++++++  Loss: 0.94 - Acc: 65.55
Epoch: 23 - Batch:  60 - Loss: 0.96 - Acc: 65.96||Epoch: 23 - Batch: 120 - Loss: 0.84 - Acc: 66.03||++++++Validation++++++  Loss: 0.89 - Acc: 66.23
Epoch: 24 - Batch:  60 - Loss: 0.85 - Acc: 66.07||Epoch: 24 - Batch: 120 - Loss: 0.92 - Acc: 66.62||++++++Validation++++++  Loss: 0.51 - Acc: 67.31
...
Epoch: 34 - Batch:  60 - Loss: 0.64 - Acc: 76.88||Epoch: 34 - Batch: 120 - Loss: 0.56 - Acc: 77.13||++++++Validation++++++  Loss: 1.26 - Acc: 72.39
Epoch: 35 - Batch:  60 - Loss: 0.72 - Acc: 76.66||Epoch: 35 - Batch: 120 - Loss: 0.67 - Acc: 77.07||++++++Validation++++++  Loss: 0.39 - Acc: 71.50
Epoch: 36 - Batch:  60 - Loss: 0.68 - Acc: 78.28||Epoch: 36 - Batch: 120 - Loss: 0.78 - Acc: 78.19||++++++Validation++++++  Loss: 1.11 - Acc: 73.48
Epoch: 37 - Batch:  60 - Loss: 0.55 - Acc: 79.29||Epoch: 37 - Batch: 120 - Loss: 0.73 - Acc: 78.96||++++++Validation++++++  Loss: 0.72 - Acc: 72.45

Output exceeds the size limit. Open the full output data in a text editor
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[12], line 14
     12 total_train_corr = 0
     13 alexnet.train()
---> 14 for batch, (image, label) in enumerate(train_loader):
     15     image = image.to(device)
     16     label = label.to(device)
```