Alexnet_Verification_with_Imgnet_drop_momentum_training

April 17, 2024

[1]:
```
import numpy as np
from functools import partial
from typing import Any, Optional

import os
import cv2
import time

import pandas as pd
import torch.nn.init as init

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from torchvision import transforms
# from torch.transforms._presets import ImageClassification
# from torch.utils import _log_api_usage_once
# from ._api import register_model, Weights, WeightsEnum
# from ._meta import _IMAGENET_CATEGORIES
# from ._utils import _ovewrite_named_param, handle_legacy_interface



model_alex_given = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet',␣
 ↪pretrained=True)
model_alex_given.eval()
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
Using cache found in C:\Users\Limit/.cache\torch\hub\pytorch_vision_v0.10.0
C:\Apps installed by Lim\anaconda3\Lib\site-
packages\torchvision\models\_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
C:\Apps installed by Lim\anaconda3\Lib\site-
```

```
packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=AlexNet_Weights.IMAGENET1K_V1`. You can also use
`weights=AlexNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[2]:
```python
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
#         _log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=2),
            nn.BatchNorm2d(96),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

    def initialize_weights(self):
```

```
        for m in self.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
                # Initialize weights for convolutional and linear layers
                init.xavier_uniform_(m.weight)
                if m.bias is not None:
                    # Initialize biases if they exist
                    init.constant_(m.bias, 0)
```

[3]:
```
class ConvertToRGB(object):
    def __call__(self, img):
        if img.shape[0] == 1:  # Check if the image has only one channel
            img = torch.stack([img[0]] * 3, dim=0)  # Convert single channel to␣
 ↪RGB
        return img


preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    ConvertToRGB(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
```

[4]:
```
image_path = "/Users/Limit/imagenet-object-localization-challenge_100"

filenames_image_path = []
label_train = []
root_image = []
counter = 0
current_label = 0
for root, _, filenames in os.walk(image_path):
    current_root = root
    for i in filenames:
#         print(i)
        counter += 1
        if ((counter) %1300 == 0):
            current_label += 1
            print(counter)
        label_train.append(current_label)


#         print(current_root)
#         print(i)
        temp = current_root + "\\" + i
#         print(temp)
        filenames_image_path.append(temp)
```

```
true_label = 0
correct_labels = 0
start_time = time.time()

counter_1=0
x_train = []
for i in range(26000):

#     print(i)
#     print(counter)
    image_name = filenames_image_path[i]
    # black and white
#     image_name = "/Users/Limit/imagenet-object-localization-challenge/
 ↪n01440764/n01440764_15560.JPEG"
    input_image = Image.open(image_name)



    input_tensor = preprocess(input_image)
#     input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected␣
 ↪by the model
    input_batch = input_tensor
    # move the input and model to GPU for speed if available
    if torch.cuda.is_available():
        input_batch = input_batch.to('cuda')
        model_alex_given.to('cuda')
    x_train.append(input_batch)
    counter_1 += 1
    if ((counter_1+1) %1300 == 0):
            counter_1 += 1
            print(counter_1)
```

```
1300
2600
3900
5200
6500
7800
9100
10400
11700
13000
14300
15600
16900
18200
```

19500
20800
22100
23400
24700
26000
27300
28600
29900
31200
32500
33800
35100
36400
37700
39000
40300
41600
42900
44200
45500
46800
48100
49400
50700
52000
53300
54600
55900
57200
58500
59800
61100
62400
63700
65000
66300
67600
68900
70200
71500
72800
74100
75400
76700
78000
79300
80600

81900
83200
84500
85800
87100
88400
89700
91000
92300
93600
94900
96200
97500
98800
100100
101400
102700
104000
105300
106600
107900
109200
110500
111800
113100
114400
115700
117000
118300
119600
120900
122200
123500
124800
126100
127400
128700
1300
2600
3900
5200
6500
7800
9100
10400
11700
13000
14300

```
15600
16900
18200
19500
20800
22100
23400
24700
26000
```

[ ]:

[5]:
```
y_train = label_train[:26000]
```

[6]:
```
len(x_train)
```

[6]: 26000

[7]:
```
# # Add channel to greyscale images so that it has 3 channels required by␣
 ↪Alexnet.
# # Add 1 more dimension to tensor to represent channel;
# labels = []

# # Function to load images from a folder directory with multiple sub-folders
# def load_images_from_folder(folder):
#     images = []
#     for root, _, filenames in os.walk(folder):
#         for filename in filenames:
#             # here filename has the label information
#             label_temp = filename.rsplit('_', 1)[0]
#             labels.append(label_temp)
#             img = cv2.imread(os.path.join(root, filename))
#             if img is not None:
#                 images.append(img)
#     return images

# image_path = "/Users/Limit/imagenet-object-localization-challenge/n01440764/"
# images = load_images_from_folder(image_path)


# # Ensure grayscale images have 3 channels
# for idx, image in enumerate(images):
#     if len(image.shape) == 2:  # If grayscale image
#         image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)  # Convert to␣
 ↪3-channel image
#         images[idx] = image
```

```
# # print the number of unique labels:
# # unique_elements = list(set(labels))
# # print(len(unique_elements))
```

[8]:
```
len(x_train)
```

[8]: 26000

[9]:
```
# Define your dataset class
class CustomDataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx], torch.tensor(self.labels[idx])
```

[10]:
```
# unique_labels = labels.copy()
# unique_labels = list(set(unique_labels))
# unique_labels.sort()
# dict_labels = {}
# for i in range(len(unique_labels)):
#     dict_labels[unique_labels[i]] = i
```

[11]:
```
# labels_numerical = []
# for i in labels:
#     labels_numerical.append(dict_labels[i])
```

[ ]:
```
```

[12]:
```
# Define your loss function
criterion = nn.CrossEntropyLoss()

# Define your optimizer
model = AlexNet().to(device)
model.initialize_weights()

optimizer = optim.Adam(model.parameters(), lr=0.005, weight_decay=5e-4)

# Prepare your data
train_data = x_train  # List of input tensors
train_labels = y_train  # List of corresponding labels
dataset = CustomDataset(train_data, train_labels)
```

```
dataloader = DataLoader(dataset, batch_size=128, shuffle=True)

# Train your models
num_epochs = 45
for epoch in range(num_epochs):
    current_loss = 0.0
    for inputs, labels in dataloader:

        optimizer.zero_grad()
        outputs = model(inputs)
        Before = list(model.parameters())[0].clone()

        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        current_loss += loss.item()
        After = list(model.parameters())[0].clone()

    # Verify whether gradient is computed successfully
    print(torch.equal(Before.data, After.data))
    print(f'Epoch {epoch+1} finished')
    epoch_loss = current_loss / len(dataset)
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
    print()
```

```
False
Epoch 1 finished
Epoch [1/45], Loss: 3.0620

False
Epoch 2 finished
Epoch [2/45], Loss: 3.0935

False
Epoch 3 finished
Epoch [3/45], Loss: 2.9840

False
Epoch 4 finished
Epoch [4/45], Loss: 3.0261

False
Epoch 5 finished
Epoch [5/45], Loss: 3.0419

False
Epoch 6 finished
```

```
Epoch [6/45], Loss: 3.0003

False
Epoch 7 finished
Epoch [7/45], Loss: 2.9992

False
Epoch 8 finished
Epoch [8/45], Loss: 246.9244

False
Epoch 9 finished
Epoch [9/45], Loss: 2.9323

False
Epoch 10 finished
Epoch [10/45], Loss: 3.0294

False
Epoch 11 finished
Epoch [11/45], Loss: 3.0235

False
Epoch 12 finished
Epoch [12/45], Loss: 3.0233

False
Epoch 13 finished
Epoch [13/45], Loss: 2.9803

False
Epoch 14 finished
Epoch [14/45], Loss: 2.9980

False
Epoch 15 finished
Epoch [15/45], Loss: 3.0678

False
Epoch 16 finished
Epoch [16/45], Loss: 2.9957

False
Epoch 17 finished
Epoch [17/45], Loss: 3.0491

False
Epoch 18 finished
```

```
Epoch [18/45], Loss: 2.9718

False
Epoch 19 finished
Epoch [19/45], Loss: 3.0033

False
Epoch 20 finished
Epoch [20/45], Loss: 2.9877

False
Epoch 21 finished
Epoch [21/45], Loss: 2.9983

False
Epoch 22 finished
Epoch [22/45], Loss: 3.1280

False
Epoch 23 finished
Epoch [23/45], Loss: 3.0390

False
Epoch 24 finished
Epoch [24/45], Loss: 3.0053

False
Epoch 25 finished
Epoch [25/45], Loss: 3.0079

False
Epoch 26 finished
Epoch [26/45], Loss: 3.0115

False
Epoch 27 finished
Epoch [27/45], Loss: 2.9949

False
Epoch 28 finished
Epoch [28/45], Loss: 2.9959

False
Epoch 29 finished
Epoch [29/45], Loss: 3.0210

False
Epoch 30 finished
```

```
Epoch [30/45], Loss: 3.0231

False
Epoch 31 finished
Epoch [31/45], Loss: 3.0125

False
Epoch 32 finished
Epoch [32/45], Loss: 3.0071
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[12], line 23
     20 for inputs, labels in dataloader:
     22     optimizer.zero_grad()
---> 23     outputs = model(inputs)
     24     Before = list(model.parameters())[0].clone()
     26     loss = criterion(outputs, labels)

File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1511, in Module.
 ↪_wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1520, in Module.
 ↪_call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

Cell In[2], line 34, in AlexNet.forward(self, x)
     33 def forward(self, x: torch.Tensor) -> torch.Tensor:
---> 34     x = self.features(x)
     35     x = self.avgpool(x)
     36     x = torch.flatten(x, 1)
```

```
File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1511, in Module.
 ↪_wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1520, in Module.
 ↪_call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\container.py:217, in␣
 ↪Sequential.forward(self, input)
    215 def forward(self, input):
    216     for module in self:
--> 217         input = module(input)
    218     return input

File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1511, in Module.
 ↪_wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File C:\Apps installed by␣
 ↪Lim\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1520, in Module.
 ↪_call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None
```

```
File C:\Apps installed by Lim\anaconda3\Lib\site-packages\torch\nn\modules\conv
 ↪py:460, in Conv2d.forward(self, input)
    459 def forward(self, input: Tensor) -> Tensor:
--> 460     return self._conv_forward(input, self.weight, self.bias)

File C:\Apps installed by Lim\anaconda3\Lib\site-packages\torch\nn\modules\conv
 ↪py:456, in Conv2d._conv_forward(self, input, weight, bias)
    452 if self.padding_mode != 'zeros':
    453     return F.conv2d(F.pad(input, self._reversed_padding_repeated_twice,
 ↪mode=self.padding_mode),
    454                     weight, bias, self.stride,
    455                     _pair(0), self.dilation, self.groups)
--> 456 return F.conv2d(input, weight, bias, self.stride,
    457                 self.padding, self.dilation, self.groups)

KeyboardInterrupt:
```

[ ]:

[13]:
```
labels_path = '/Users/Limit/imagenet_annot/validation_set_labels.csv'

labels_df = pd.read_csv(labels_path)

labels_df_leq_20 = labels_df[labels_df['label'] <= 20]

labels_validation_images = labels_df_leq_20['label'].tolist()
```

[14]:
```
len(labels_df_leq_20['ImageId'].tolist())
model.eval()
```

[14]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 96, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (8): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
```

```
    (10): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): 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)
  )
)
```

[15]:
```
image_path = "/Users/Limit/imagenet-object-localization-challenge_validation/
 ↪val"
filenames_image_path = []
for root, _, filenames in os.walk(image_path):
    for i in filenames:
#         print(i)
        if (i.split('.')[0] in labels_df_leq_20['ImageId'].tolist()):
            filenames_image_path.append(i)
true_label = 0
counter = 0
correct_labels = 0
start_time = time.time()
for i in range(len(filenames_image_path)):
    counter +=1
#     print(i)
#     print(counter)
    image_name = image_path + '/' + filenames_image_path[i]
    # black and white
#     image_name = "/Users/Limit/imagenet-object-localization-challenge/
 ↪n01440764/n01440764_15560.JPEG"
    input_image = Image.open(image_name)



    input_tensor = preprocess(input_image)
    input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected␣
 ↪by the model
```

```
    # move the input and model to GPU for speed if available
    if torch.cuda.is_available():
        input_batch = input_batch.to('cuda')
        model.to('cuda')

    if (counter%100 == 0):
        print("currently at", counter, 'current time is', time.time() -␣
 ↪start_time)
    with torch.no_grad():
        output = model(input_batch)
    # Tensor of shape 1000, with confidence scores over ImageNet's 1000 classes
#     if (torch.argmax(output[0]).item() == true_label):
#         correct_labels += 1
    if (torch.argmax(output[0]).item() == labels_validation_images[i]):
        print('yes')
        correct_labels += 1
    else:
        print('no')
    # The output has unnormalized scores. To get probabilities, you can run a␣
 ↪softmax on it.
    probabilities = torch.nn.functional.softmax(output[0], dim=0)
    # print(probabilities)

print('the overall testing error is')
print(correct_labels/counter)
```

```
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
```

no
no
no
no
no
yes
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
yes
no
no
no

```
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
yes
currently at 100 current time is 4.329607009887695
yes
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
yes
no
no
no
no
no
no
no
no
yes
no
no
no
no
no

```
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
currently at 200 current time is 8.635331153869629
no
no
no
no
no
no
no
no
no
no
no
no
no
```

yes
no
no
no
no
no
no
no
no
no
no
no
no
yes
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no

```
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
currently at 300 current time is 13.069218873977661
no
no
no
no
no
no
no
```

```
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
no
yes
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
currently at 400 current time is 17.530898809432983
no
no
```

no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no

```
no
no
no
no
yes
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
```

```
no
no
currently at 500 current time is 21.898053884506226
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
yes
no
no
no
no
no
yes
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
no
no
no
no
no
currently at 600 current time is 26.195046424865723
yes
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
no
no
no
no
no
no
yes
no
no
yes
currently at 700 current time is 30.60791301727295
no
no
yes
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
yes
no
no
```

```
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
currently at 800 current time is 34.989232540130615
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
```

yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no

```
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
currently at 900 current time is 39.21831011772156
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
```

```
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
no
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
currently at 1000 current time is 43.682902812957764
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
```

```
no
yes
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
no
the overall testing error is
0.047619047619047616
```

[ ]:
```
# OLD code



# model_alex_given.eval()

# for i in range(40):
#     input_batch = input_tensor[i].unsqueeze(0) # create a mini-batch as␣
 ↪expected by the model

#     prediction = model_alex_given(input_batch)
#     print('prediction:')
#     print(torch.argmax(prediction))
```

```
#     print("answer:")
# #     print(labels_numerical[i])
#     print()
```

[16]:
```
torch.save(model.state_dict(), 'model_weights_alexnet_adam_dont_work_45_epoch.
↪pth')
```

[ ]:
```
# filenames_image_path
```