

Lab 2: Convolutional Neural Networks for Computer Vision

In this lab, we will learn how to use CNNs for computer vision applications. We will look at two applications - 1) detecting objects in images, and 2) detecting cyberbullying in images.

Grading Breakdown:

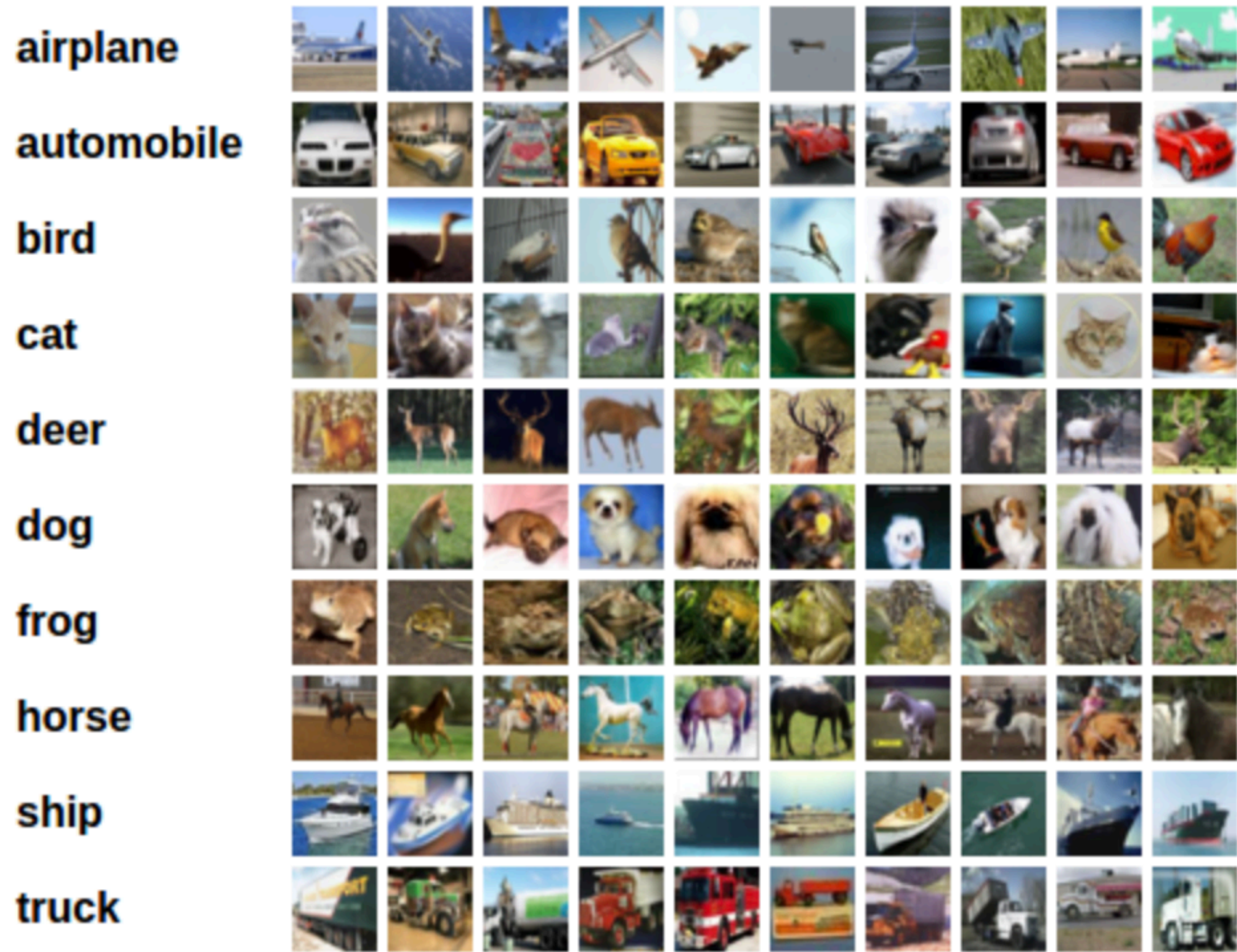
Part 1: Detecting objects 70%

Part 2: Detecting cyberbullying 30%

**You have the ability to fully view my code on Github at the following link kml614:<https://github.com/ecruz0369/DA6233ECruz-2023/blob/main/IS6733Lab2Eacruzkml614Actual.ipynb> **

Part 1: CNN to Detect Images of 10 Objects

In this section, we will design a CNN to classify images of 10 objects from the CIFAR10 dataset.



```
# Import packages first
import torch
import torchvision
import torchvision.transforms as transforms
```

```
# Just like in Lab 1, we will normalize our inputs. As we go along, notice the recurring themes in deep learning, such as data normalization
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

```
# set a batch size
batch_size = 4
```

```
# download the train set
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
# we'll just split the train into train and val
trainset, valset = torch.utils.data.random_split(trainset, [0.8, 0.2])
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True, num_workers=2)
valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, shuffle=True, num_workers=2)
```

```
# download the train set
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
```

testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False, num_workers=2)

labels. In this problem there are 10 labels.
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

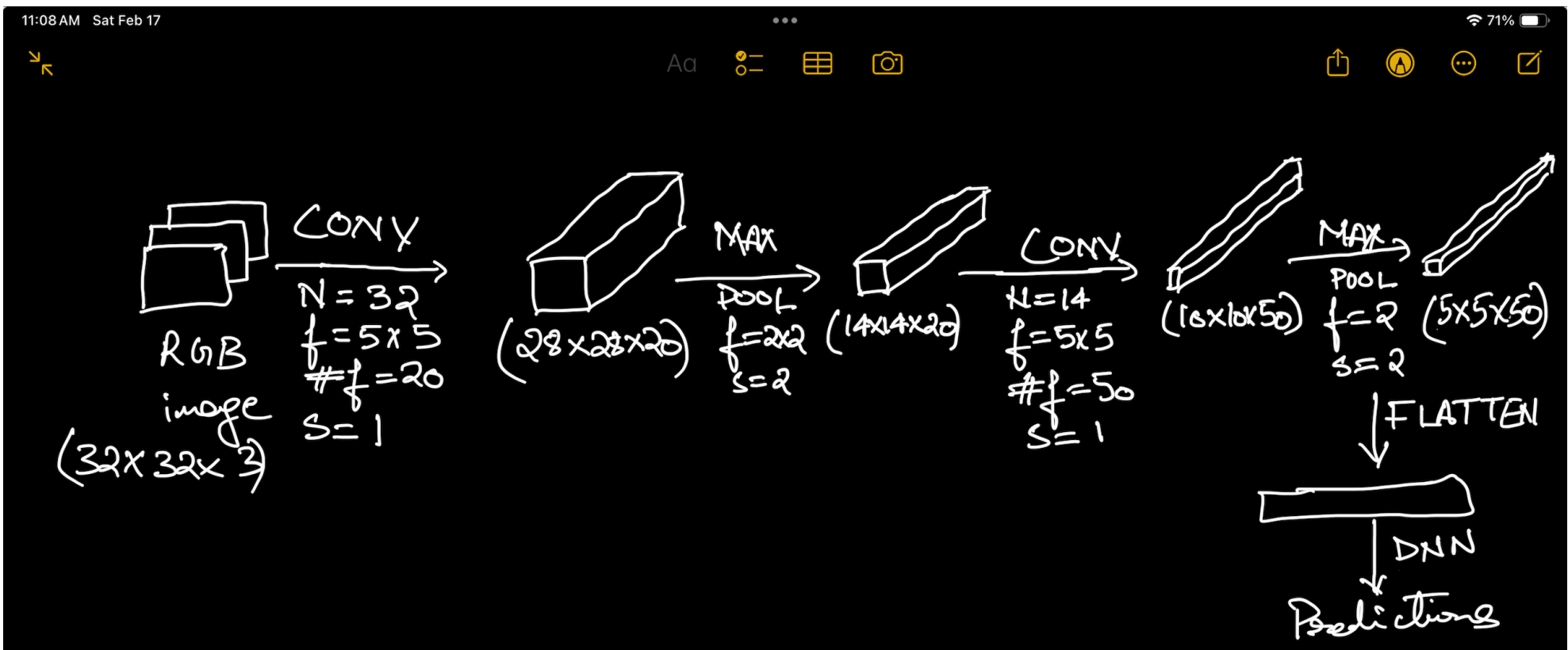
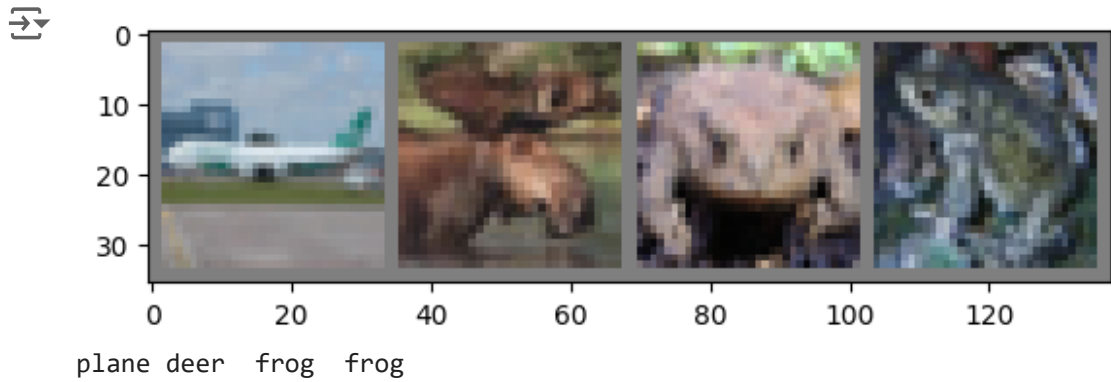
Files already downloaded and verified
Files already downloaded and verified

```
# Visualize a few images in the train set
import matplotlib.pyplot as plt
import numpy as np

def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
```



```
# Correct Code added to compute Convolutional Nueral Networks
import torch
import torch.nn as nn
import torch.nn.functional as F

class MyObjDetectorCNN(nn.Module):
    def __init__(self):
        super(MyObjDetectorCNN, self).__init__()

        # Define the convolutional and pooling layers
        self.convolutional_layer = nn.Sequential(
            # First convolution layer: Input is RGB (3 channels), output is 20 channels, kernel size is 5x5, stride is 1
            nn.Conv2d(in_channels=3, out_channels=20, kernel_size=5, stride=1), # RGB image has 3 input channels
            nn.ReLU(),
            # Max pooling layer with kernel size 2x2 and stride 2 for subsampling
            nn.MaxPool2d(kernel_size=2, stride=2),
            # Second convolution layer: Input is the 20 output channels from the previous layer, output is 50 channels
            nn.Conv2d(in_channels=20, out_channels=50, kernel_size=5, stride=1),
            nn.ReLU(),
            # Max pooling layer again with kernel size 2x2 and stride 2 for subsampling
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        # Calculate the flattened size after the convolution and pooling layers
        # Assume the input image is 32x32, and calculate how the size changes after each layer:
```

```
# After 1st Conv: (32 - 5 + 2*0)/1 + 1 = 28 -> MaxPool(28/2) = 14
# After 2nd Conv: (14 - 5 + 2*0)/1 + 1 = 10 -> MaxPool(10/2) = 5
# Final output from conv layers is 50 channels of 5x5 feature maps -> 50 * 5 * 5 = 1250

# Define the fully connected (linear) layers for classification
self.linear_layer = nn.Sequential(
    nn.Linear(in_features=50 * 5 * 5, out_features=500), # Flatten the CNN output for the DNN
    nn.ReLU(),
    nn.Linear(in_features=500, out_features=10), # 10 classes for classification
)

def forward(self, x):
    x = self.convolutional_layer(x) # Apply convolution and pooling layers
    x = torch.flatten(x, 1) # Flatten the output from conv layers into a 1D vector
    x = self.linear_layer(x) # Apply fully connected layers
    x = F.softmax(x, dim=1) # Apply softmax to get class probabilities
    return x

# Create the CNN object
net = MyObjDetectorCNN()
```

```
# define a loss function and optimizer
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr = 0.001, momentum = 0.9)
```

```
# train the network (better use a GPU for this, look at the first lab for moving objects to GPU)
net.train()
for epoch in range(2): # loop over the dataset multiple times, you could increase this.

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999: # print every 2000 mini-batches
            print(f'[epoch + 1], {i + 1:5d} loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

```
📄 [1, 2000] loss: 2.301
[1, 4000] loss: 2.258
[1, 6000] loss: 2.184
[1, 8000] loss: 2.147
[1, 10000] loss: 2.121
[2, 2000] loss: 2.098
[2, 4000] loss: 2.087
[2, 6000] loss: 2.063
[2, 8000] loss: 2.064
[2, 10000] loss: 2.047
Finished Training
```

```
# test your CNN on the test set
# Note: If your accuracy is low, you need to further train your CNN.
dataiter = iter(testloader)
images, labels = next(dataiter)

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

# What is the accuracy, precision, recall and F1-score on the test dataset?
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)

y_test = []
y_test_predictions = []

net.eval()
for i, data in enumerate(testloader, 0):
```

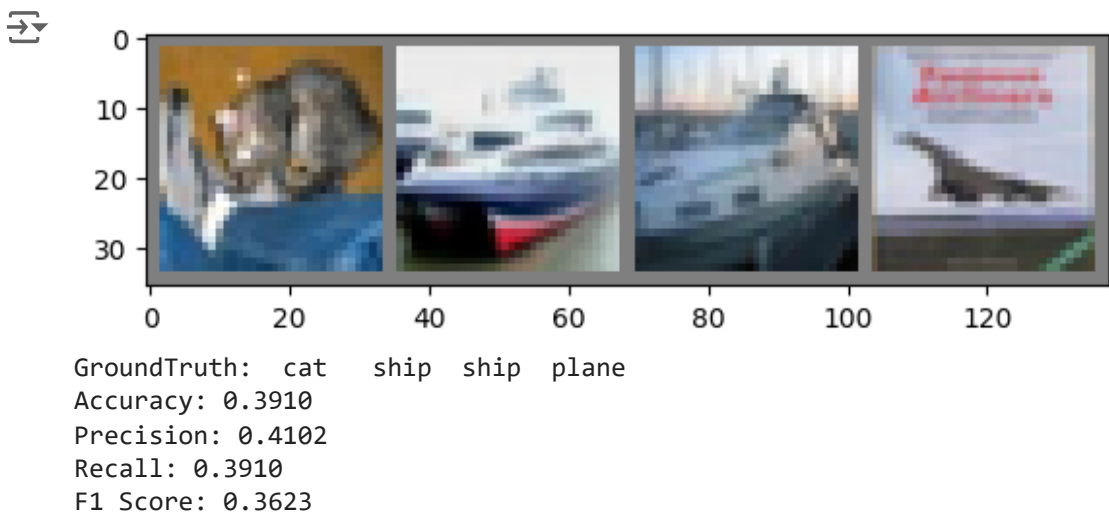


```
inputs, labels = data
y_test.extend([i.item() for i in labels])

outputs = net(inputs)
y_test_predictions.extend(torch.argmax(i).item() for i in outputs)

# print accuracy, prec, rec, f1-score here
# Calculate and print metrics
accuracy = accuracy_score(y_test, y_test_predictions)
precision = precision_score(y_test, y_test_predictions, average='macro')
recall = recall_score(y_test, y_test_predictions, average='macro')
f1 = f1_score(y_test, y_test_predictions, average='macro')

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```



Part 2: Using a Pre-trained CNN to Detect Cyberbullying in Images

With previous lab learning, you should have some knowledge about how to develop an AI model to detect cyberbullying lauguage. In this lab, we will keep learning how AI can be developed to detect cyberbullying. We will use a publicly available test dataset of cyberbullying images, and deploy an pre-trained AI model to automatically detect cyberbullying images. Approach towards analysing the cyber bullying in images in a dataset, there are three steps:

- 1. Understand and identify the factors related to cyberbullying in images.
- 2. Load the pre-trained model.
- 3. Fine-tune the model with a small dataset.
- 4. Evaluate the pre-trained model and your fine-tuned model with the same test dataset.
 - Get the results of accuracy, precision, recall and F1-score
 - plot out the confusion matrix figure

The models and datasets in this lab are taken from the paper "Towards Understanding and Detecting Cyberbullying in Real-world Images" (NDSS 2021). <https://www.ndss-symposium.org/ndss-paper/towards-understanding-and-detecting-cyberbullying-in-real-world-images/>

Download the pre-trained model, test dataset and the dependencies

First, we need to download the pre-trained model and the test dataset used in the lab. Just hit the 'play' button run the code below.

```
# download the model and dataset
!wget -O auxes_17.pt https://buffalo.box.com/shared/static/cjk39hq7prpwj2rkqz61c2jr6q2h5shy.pt # model checkpoints
!wget -O cyberbullying_data.zip https://github.com/cuadvancelab/materials/blob/main/lab2/cyberbullying_data.zip?raw=true # test dataset

--2024-10-20 14:44:05-- https://buffalo.box.com/shared/static/cjk39hq7prpwj2rkqz61c2jr6q2h5shy.pt
Resolving buffalo.box.com (buffalo.box.com)... 74.112.186.157
Connecting to buffalo.box.com (buffalo.box.com)|74.112.186.157|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /public/static/cjk39hq7prpwj2rkqz61c2jr6q2h5shy.pt [following]
--2024-10-20 14:44:05-- https://buffalo.box.com/public/static/cjk39hq7prpwj2rkqz61c2jr6q2h5shy.pt
Reusing existing connection to buffalo.box.com:443.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://buffalo.app.box.com/public/static/cjk39hq7prpwj2rkqz61c2jr6q2h5shy.pt [following]
--2024-10-20 14:44:06-- https://buffalo.app.box.com/public/static/cjk39hq7prpwj2rkqz61c2jr6q2h5shy.pt
Resolving buffalo.app.box.com (buffalo.app.box.com)... 74.112.186.157
Connecting to buffalo.app.box.com (buffalo.app.box.com)|74.112.186.157|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://public.boxcloud.com/d/1/b1!5hrXmSHDObZdatQSRp1ECcMoHU7DBkNy6i7WAYCNMuDzFDCbToWdJJFCXgiv1Tw3x5ouyiAX6dJJ89wZStdj8vev
--2024-10-20 14:44:07-- https://public.boxcloud.com/d/1/b1!5hrXmSHDObZdatQSRp1ECcMoHU7DBkNy6i7WAYCNMuDzFDCbToWdJJFCXgiv1Tw3x5ouyiAX6
Resolving public.boxcloud.com (public.boxcloud.com)... 74.112.186.164
Connecting to public.boxcloud.com (public.boxcloud.com)|74.112.186.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 742758003 (708M) [application/octet-stream]
Saving to: 'auxes_17.pt'
```

```
auxes_17.pt      100%[=====>] 708.35M   21.6MB/s    in 34s

2024-10-20 14:44:42 (21.0 MB/s) - 'auxes_17.pt' saved [742758003/742758003]

--2024-10-20 14:44:42--  https://github.com/cuadvancelab/materials/blob/main/lab2/cyberbullying_data.zip?raw=true
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github.com/cuadvancelab/materials/raw/refs/heads/main/lab2/cyberbullying_data.zip [following]
--2024-10-20 14:44:42--  https://github.com/cuadvancelab/materials/raw/refs/heads/main/lab2/cyberbullying_data.zip
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/cuadvancelab/materials/refs/heads/main/lab2/cyberbullying_data.zip [following]
--2024-10-20 14:44:42--  https://raw.githubusercontent.com/cuadvancelab/materials/refs/heads/main/lab2/cyberbullying_data.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 420336 (410K) [application/zip]
Saving to: 'cyberbullying_data.zip'

cyberbullying_data. 100%[=====>] 410.48K   --.-KB/s    in 0.02s

2024-10-20 14:44:43 (24.7 MB/s) - 'cyberbullying_data.zip' saved [420336/420336]
```

```
# unzip the test data
%%capture
!unzip "/content/cyberbullying_data.zip" -d "/content"
```

Let's import all our softwares dependencies in our iPython notebook

```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torch import optim
import torch.utils.data as data_utils
import torchvision.transforms as transforms
from torchvision import models
from torch.optim.lr_scheduler import ReduceLROnPlateau
from torch.utils.data import Dataset, DataLoader

import pickle
import gzip
import sys
import time
import numpy as np
import math
import os
import random
from skimage import io, transform
```

> How to identify cyberbullying in images

```
[ ] ↳ 3 cells hidden
```

▼ Load datasets

Now, let's run the subsequent codes to load your data from a predefined class

```
class PosesDataset(Dataset):

    def __init__(self, root_dir, poses_dir, auxes_dir):

        self.samples = []
        self.root_dir = root_dir
        self.poses_dir = poses_dir
        self.auxes_dir = auxes_dir

        for _, _, cb_images in os.walk(self.root_dir + 'cyberbullying'): break
        for _, _, non_cb_images in os.walk(self.root_dir + 'non_cyberbullying'): break
        for _, _, cb_poses in os.walk(self.poses_dir + 'cyberbullying'): break
        for _, _, non_cb_poses in os.walk(self.poses_dir + 'non_cyberbullying'): break

        for i in cb_images:
            self.samples.append((self.root_dir + 'cyberbullying/' + i, self.poses_dir + 'cyberbullying/' + i, self.auxes_dir + 'cyberbullying/'

        for i in non_cb_images:
            self.samples.append((self.root_dir + 'non_cyberbullying/' + i, self.poses_dir + 'non_cyberbullying/' + i, self.auxes_dir + 'non_cyb
```

```
def __len__(self):
    return len(self.samples)

def __getitem__(self, idx):
    if torch.is_tensor(idx):
        idx = idx.tolist()

    img_name, pose_name, aux_name, label = self.samples[idx]
    image = io.imread(img_name)

    aux = pickle.load(open(aux_name + '.p', 'rb'))
    aux = torch.tensor(aux)

    # drop the alpha channel for some images
    if image.shape == (224, 224):
        # handle grayscale images
        image = np.stack([image, image, image], axis=2)

    if image.shape == (224, 224, 4):
        image = image[:, :, :3]

    image = image.transpose((2, 0, 1)) # C X H X W
    pose = io.imread(pose_name)
    if pose.shape != (224, 224):
        pose = pose[:, :, 0]
    pose = np.expand_dims(pose, axis = 0)
    image = np.concatenate((image, pose), axis = 0)
    sample = {'image': torch.from_numpy(image.copy()).float() / 255, 'aux': aux, 'label': label}
    return sample

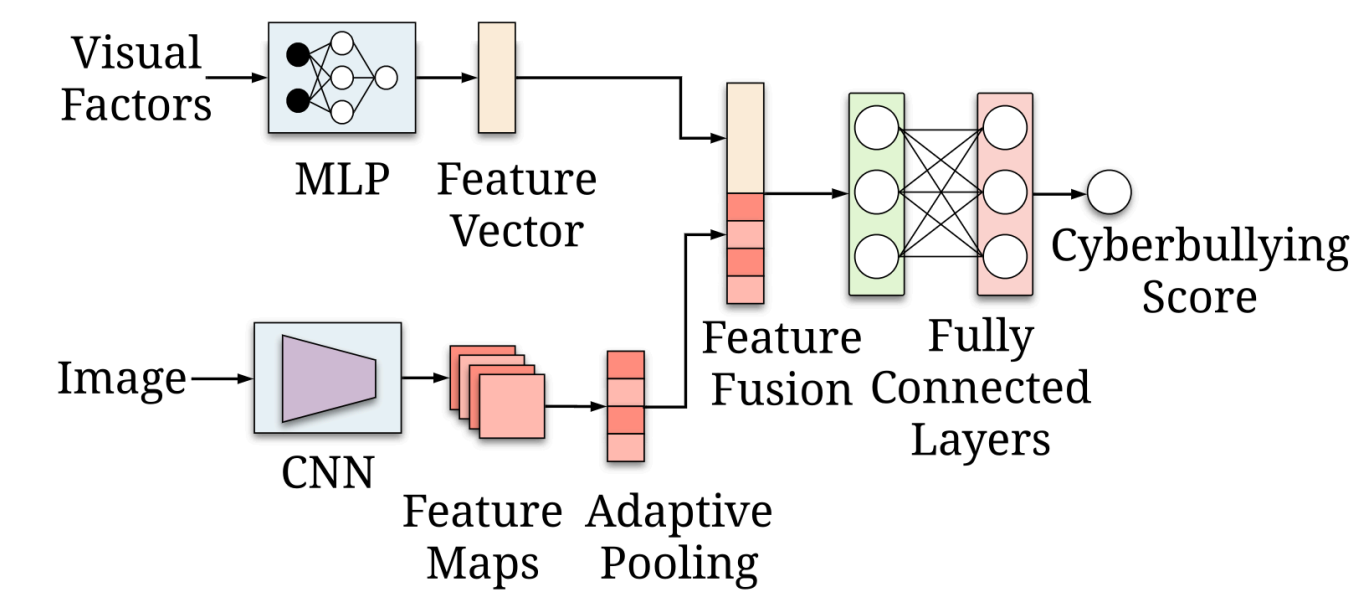
test_set = PosesDataset('cyberbullying_data/cyberbullying_data_splits_clean/test/', 'cyberbullying_data/cyberbullying_poses/test/', 'cybe
test_loader = torch.utils.data.DataLoader(test_set, batch_size = 1, shuffle = True)
```

Load pre-trained AI model

We will use GPU to test our AI if it is available.

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu') # edited to include CPU inlieau of CUDA
```

The AI model prediction process looks like the following figure.



In our AI model, we combine the low level image features with the cyberbulling factors identified before. We combine these features using feature fusion techniques.

We use the `VGG16` pre-trained model for image features `CNN` and use a multi-layer perceptron model `MLP` for the factors related features, and combine the feature vectors from both these models using late fusion.

Let's load the pre-trained model to test its capability

```
# load vgg16 pre-trained model
orig = models.vgg16(pretrained = True)
```

```
⚡ /usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` f
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:06<00:00, 79.7MB/s]
```

```
class CB(nn.Module):
    def __init__(self):
        super(CB, self).__init__()
        self.conv1 = nn.Conv2d(4, 3, 1)
```

```
self.f = nn.Sequential(*list(orig.features.children()))
self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
self.aux_classifier = nn.Sequential(
    nn.Linear(25097, 1024),
    nn.ReLU(),
    nn.Linear(1024, 25088),
    nn.ReLU()
)
self.classifier = nn.Sequential(*list(orig.classifier.children()))
self.classifier[-1] = nn.Linear(4096, 2)
self.sig = nn.Sigmoid()

def forward(self, x, aux):
    x = self.conv1(x)
    x = self.f(x)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = torch.cat((x, aux), dim = 1)
    x = self.aux_classifier(x)
    x = self.classifier(x)
    x = self.sig(x)

    return x
```

Pass the pre-trained checkpoints to the VGG model so that you can have our pre-trained model

```
# [22]
model = torch.load("auxes_17.pt", map_location=torch.device('cpu')) # Load the model to the CPU
model.to(device)

running_loss = []
criterion = nn.CrossEntropyLoss()
correct, incorrect, total = 0., 0., 0.

<ipython-input-26-f2e9451adf69>:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), w
    model = torch.load("auxes_17.pt", map_location=torch.device('cpu')) # Load the model to the CPU
```

Generate the detection results for test data

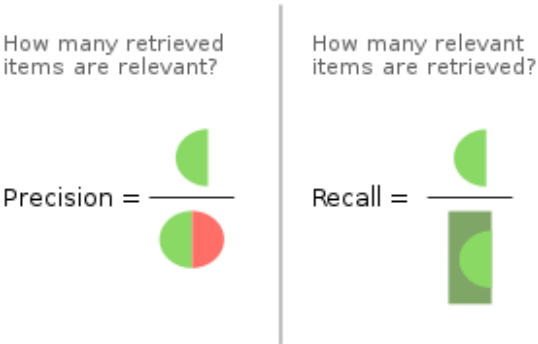
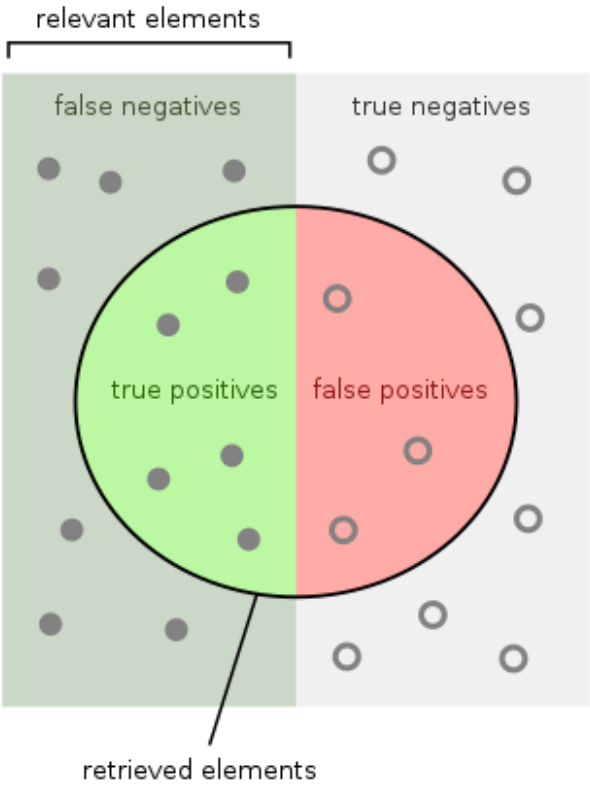
Now, it's time to evaluate the pre-trained model's capability with our test dataset

```
with torch.no_grad():
    for i_v, data_v in enumerate(test_loader):
        x_test, y_test, aux_test = data_v['image'], data_v['label'], data_v['aux']
        x_test, y_test, aux_test = x_test.to(device), y_test.to(device, dtype = torch.long), aux_test.to(device, dtype = torch.float)
        y_test_ = model(x_test, aux_test) # forward pass for the pre-trained model
        running_loss.append(criterion(y_test_, y_test))
        _, predicted = torch.max(y_test_.data, 1)
        total += y_test.size(0)
        correct += (predicted == y_test).sum().item()

print('Test loss is: {:.3f}'.format((sum(running_loss) / len(running_loss)).item()))
print('The accuracy for test dataset is: {}'.format((correct / total) * 100))

Test loss is: 0.454
The accuracy for test dataset is: 85.0%
```

Task 1: Write code to generate result report contains: Accuracy, Precision, Recall and F1-Score



reference link: <https://en.wikipedia.org/wiki/F-score>

```
# get the acc, precision, recall, f1 score for the test set

tp, tn, fp, fn = 0, 0, 0, 0

model.eval()
with torch.no_grad():
    for i_v, data_v in enumerate(test_loader):
        x_test, y_test, aux_test = data_v['image'], data_v['label'], data_v['aux']
        x_test, y_test, aux_test = x_test.to(device), y_test.to(device, dtype = torch.long), aux_test.to(device, dtype = torch.float)
        y_test_ = model(x_test, aux_test) # forward pass for the fine-tuned model
        _, predicted = torch.max(y_test_.data, 1)
        if y_test == 1 and predicted == 1:
            tp += 1
        elif y_test == 1 and predicted == 0:
            fn += 1
        elif y_test == 0 and predicted == 1:
            fp += 1
        elif y_test == 0 and predicted == 0:
            tn += 1
```

Double-click (or enter) to edit

```
# TODO: Complete the following code to calculate the accuracy, precision, recall and F1 score.
acc = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
precision = tp / (tp + fp) if (tp + fp) > 0 else 0
recall = tp / (tp + fn) if (tp + fn) > 0 else 0
f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

print('The accuracy for test dataset is: {}'.format(acc * 100))
print('The precision for test dataset is: {}'.format(precision * 100))
print('The recall for test dataset is: {}'.format(recall * 100))
print('The f1 score for test dataset is: {}'.format(f1 * 100))
```

➡ The accuracy for test dataset is: 85.0%
The precision for test dataset is: 88.88888888888889%
The recall for test dataset is: 80.0%
The f1 score for test dataset is: 84.21052631578948%

Task 2: Write code to plot the confusion matrix (you are allowed to borrow any python tools, such as scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html))

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Initialize lists for true and predicted labels
y_true = []
y_pred = []
```



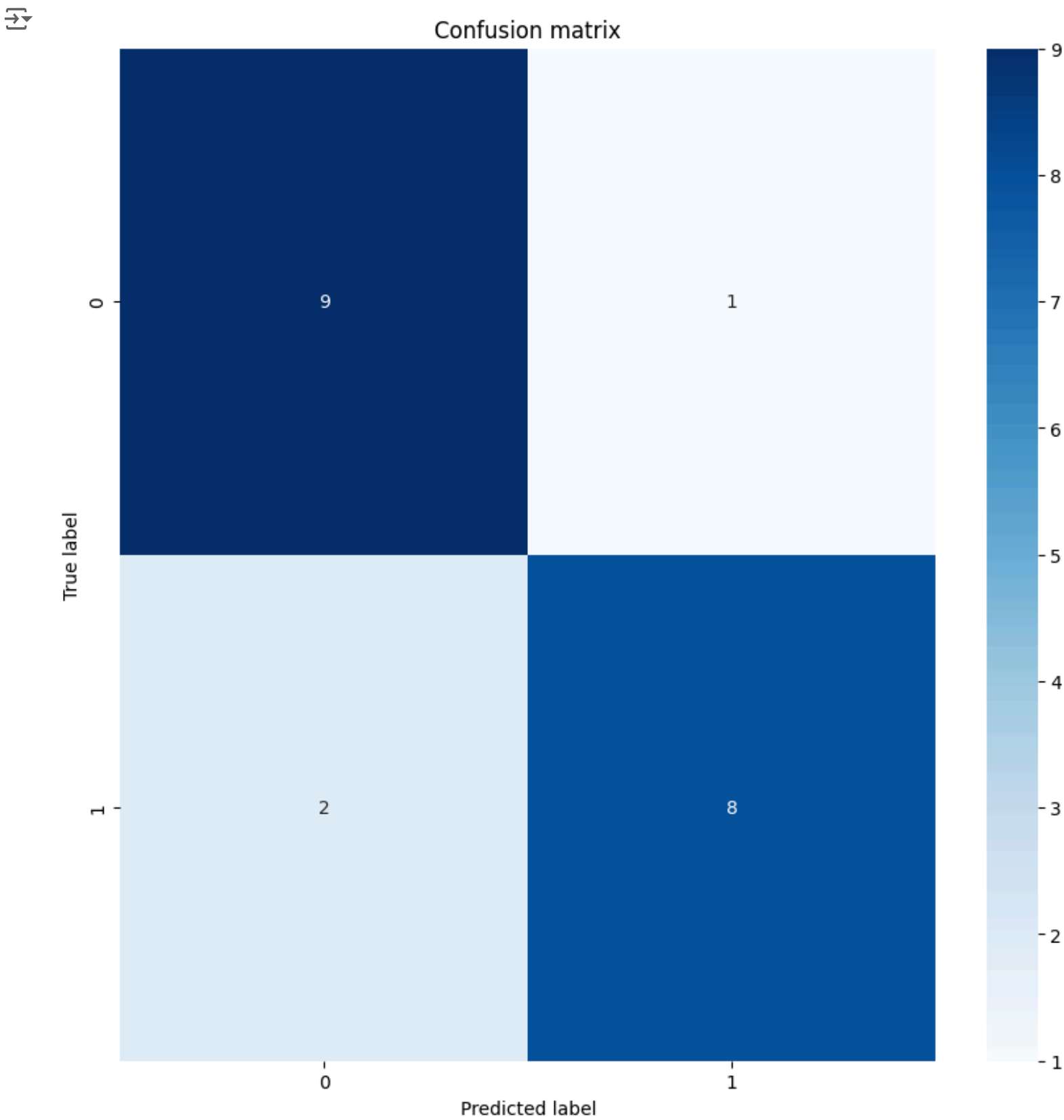
```
# Set the model to evaluation mode
model.eval()
with torch.no_grad():
    for i_v, data_v in enumerate(test_loader):
        x_test, y_test, aux_test = data_v['image'], data_v['label'], data_v['aux']
        x_test, y_test, aux_test = x_test.to(device), y_test.to(device, dtype=torch.long), aux_test.to(device, dtype=torch.float)

        # Get model predictions
        y_test_ = model(x_test, aux_test)
        _, predicted = torch.max(y_test_.data, 1)

        # Append the true and predicted labels
        y_true.extend(y_test.cpu().numpy()) # Move to CPU and convert to numpy
        y_pred.extend(predicted.cpu().numpy()) # Same for predictions

# Calculate confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 10))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.title("Confusion matrix")
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



Let's check with one instance

To better understand the performance, you can try to visualize one instance in the dataset

```
# check how many test data samples we have
print(f"we have {len(test_set)} samples in our test dataset, you can choose any of them to see the prediction.")
```

we have 20 samples in our test dataset, you can choose any of them to see the prediction.

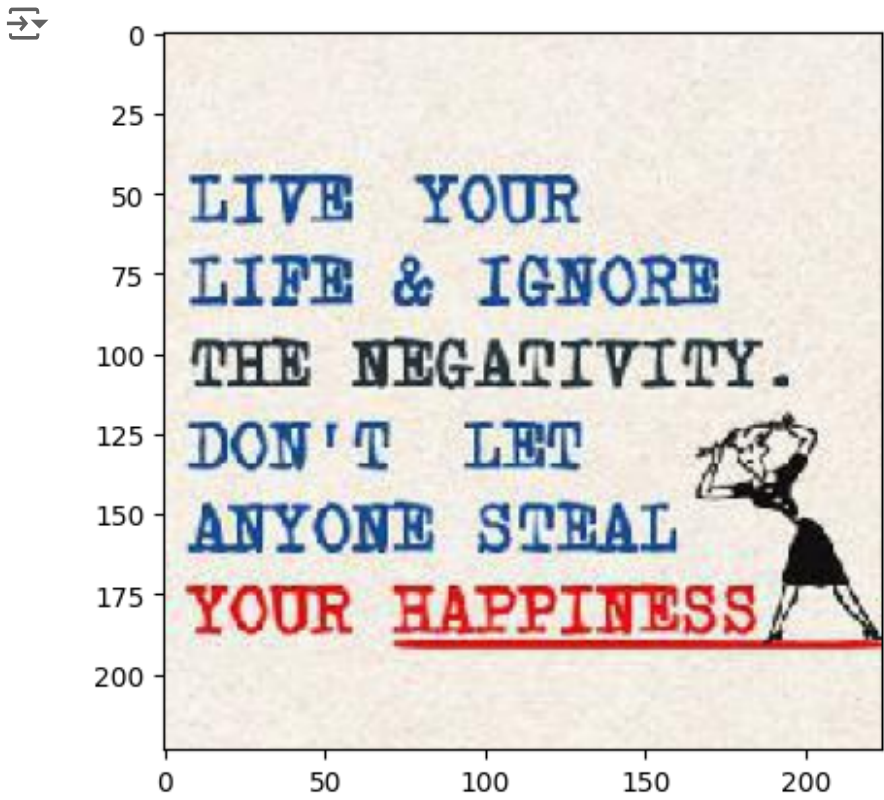
#@markdown Select a number to view the image and its label.

Select a number to view the image and its label.

```
picture_index = "10" #@param [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
index = int(picture_index)
instance = test_set[index]

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread(test_set.samples[index][0])
imgplot = plt.imshow(img)
plt.show()
annot_label = "cyberbullying" if test_set[index]['label']==1 else "non-cyberbullying"
print('')
print("The label of this image is: {}".format(annot_label))
```

picture_index: 10



The label of this image is: non-cyberbullying

Run the following code cell to check the AI's prediction

```
# check if the prediction is correct
instance_image, instance_label, instance_aux = instance['image'].to(device), torch.tensor(instance['label']).to(device, dtype = torch.long)

output = model(instance_image.unsqueeze(0), instance_aux.unsqueeze(0)).data
_, prediction = torch.max(output.data, 1)
predict_label = "cyberbullying" if prediction.item()==1 else "non-cyberbullying"
comparision = "correct" if prediction==instance_label else "not correct"

print("The AI prediction for this image is: {}, which is {}".format(predict_label, comparision))
```

The AI prediction for this image is: non-cyberbullying, which is correct!

Model Fine-Tuning

Task 3: Write code to fine-tune the model with the training dataset

The training dataset will be prepared via the following code cells.

```
# download the training data
!wget -O cyberbullying_train_data.zip https://buffalo.box.com/shared/static/4tq3wxly5pk2k8hpx7brvtr89icx7e7f.zip

--2024-10-20 14:47:08-- https://buffalo.box.com/shared/static/4tq3wxly5pk2k8hpx7brvtr89icx7e7f.zip
Resolving buffalo.box.com (buffalo.box.com)... 74.112.186.157
Connecting to buffalo.box.com (buffalo.box.com)|74.112.186.157|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /public/static/4tq3wxly5pk2k8hpx7brvtr89icx7e7f.zip [following]
--2024-10-20 14:47:09-- https://buffalo.box.com/public/static/4tq3wxly5pk2k8hpx7brvtr89icx7e7f.zip
Reusing existing connection to buffalo.box.com:443.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://buffalo.app.box.com/public/static/4tq3wxly5pk2k8hpx7brvtr89icx7e7f.zip [following]
--2024-10-20 14:47:09-- https://buffalo.app.box.com/public/static/4tq3wxly5pk2k8hpx7brvtr89icx7e7f.zip
Resolving buffalo.app.box.com (buffalo.app.box.com)... 74.112.186.157
Connecting to buffalo.app.box.com (buffalo.app.box.com)|74.112.186.157|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://public.boxcloud.com/d/1/b1!dSxJbRMwIybZ6hNytsI-g-5uI_f0xI2v-IBVcY0lnDaPOPAztdHfTnEH1P-Rn2b4fC17TX5EjNI104FJpeeaBThI
--2024-10-20 14:47:10-- https://public.boxcloud.com/d/1/b1!dSxJbRMwIybZ6hNytsI-g-5uI_f0xI2v-IBVcY0lnDaPOPAztdHfTnEH1P-Rn2b4fC17TX5Ej
Resolving public.boxcloud.com (public.boxcloud.com)... 74.112.186.164
Connecting to public.boxcloud.com (public.boxcloud.com)|74.112.186.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 47450889 (45M) [application/zip]
Saving to: 'cyberbullying_train_data.zip'
```


cyberbullying_train 100%[=====>] 45.25M 16.0MB/s in 2.8s

2024-10-20 14:47:14 (16.0 MB/s) - 'cyberbullying_train_data.zip'saved [47450889/47450889]

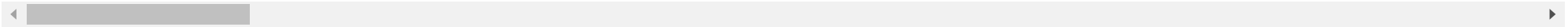
```
# unzip the training data
%%capture
!unzip "/content/cyberbullying_train_data.zip" -d "/content"
```

```
# prepare the training data
train_set = PosesDataset('cyberbullying_train_data/cyberbullying_data_splits_clean/train/', 'cyberbullying_train_data/cyberbullying_poses
# remove .DS_Store files if they exist
train_set.samples = [x for x in train_set.samples if '.DS_Store' not in x[0]]
# prepare the dataloader
train_loader = torch.utils.data.DataLoader(train_set, batch_size = 12, shuffle = True)
```

```
# copy the pre-trained model for fine-tuning
ft_model = torch.load("auxes_17.pt")
ft_model.to(device)
```

 <ipython-input-38-29b475d46782>:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), w

```
ft_model = torch.load("auxes_17.pt")
CB(
  (conv1): Conv2d(4, 3, kernel_size=(1, 1), stride=(1, 1))
  (f): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (aux_classifier): Sequential(
    (0): Linear(in_features=25097, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=25088, bias=True)
    (3): ReLU()
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
  (sig): Sigmoid()
)
```




```
# prepare the optimizer, loss function, learning rate
optimizer = optim.Adam(ft_model.parameters(), lr = 1e-4)
```

```
#Actual working code
# TODO: complete the following code by replace "___", to mimic fine-tune (further training) the model
ft_model.train()

# Assign an integer value to epochs, e.g., 10
epochs = 5
for epoch in range(epochs):
    for i, data in enumerate(train_loader):
```

```
inputs = data['image'].to(device)
aux = data['aux'].to(device)
labels = data['label'].to(device)
optimizer.zero_grad() # zero the parameter gradients
outputs = ft_model(inputs, aux) # forward pass
# Pass the outputs and labels to the criterion function
loss = criterion(outputs, labels)
loss.backward() # backward pass
optimizer.step() # update weights
running_loss.append(loss.item()) # save loss
_, predicted = torch.max(outputs.data, 1) # get predictions
total += labels.size(0) # update total
# Compare predicted labels with actual labels
correct += (predicted == labels).sum().item()
if i % 50 == 0: # print every 50 mini-batches
    print('Epoch: %d, Iteration: %d, Loss: %.4f, Accuracy: %.4f' % (epoch, i, loss.item(), correct / total))
    # Reset correct, incorrect, and total for the next batch
    correct, total = 0, 0

# Note:
# This code is a very basic version that helps us keep training the model with the training set,
# Recall from the last lecture, we can have a validation set to help us decide when to stop training
# the model.
# If you are interested, you can try to split the training set into training set and validation set,
# and use the validation set to help you decide when to stop training the model
```



```
Epoch: 0, Iteration: 0, Loss: 0.3136, Accuracy: 0.9062
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1056: UserWarning: Palette images with Transparency expressed in bytes should be
warnings.warn(
Epoch: 0, Iteration: 50, Loss: 0.5633, Accuracy: 0.8617
Epoch: 0, Iteration: 100, Loss: 0.3133, Accuracy: 0.8667
Epoch: 0, Iteration: 150, Loss: 0.4799, Accuracy: 0.8750
Epoch: 0, Iteration: 200, Loss: 0.3133, Accuracy: 0.8833
Epoch: 1, Iteration: 0, Loss: 0.4799, Accuracy: 0.8333
Epoch: 1, Iteration: 50, Loss: 0.3966, Accuracy: 0.8483
Epoch: 1, Iteration: 100, Loss: 0.4799, Accuracy: 0.8517
Epoch: 1, Iteration: 150, Loss: 0.5633, Accuracy: 0.8917
Epoch: 1, Iteration: 200, Loss: 0.3966, Accuracy: 0.8883
Epoch: 2, Iteration: 0, Loss: 0.3133, Accuracy: 1.0000
Epoch: 2, Iteration: 50, Loss: 0.3966, Accuracy: 0.8783
Epoch: 2, Iteration: 100, Loss: 0.4799, Accuracy: 0.8683
Epoch: 2, Iteration: 150, Loss: 0.5633, Accuracy: 0.8733
Epoch: 2, Iteration: 200, Loss: 0.3133, Accuracy: 0.8583
Epoch: 3, Iteration: 0, Loss: 0.3966, Accuracy: 0.8889
Epoch: 3, Iteration: 50, Loss: 0.4799, Accuracy: 0.8750
Epoch: 3, Iteration: 100, Loss: 0.5633, Accuracy: 0.8650
Epoch: 3, Iteration: 150, Loss: 0.3133, Accuracy: 0.8650
Epoch: 3, Iteration: 200, Loss: 0.3966, Accuracy: 0.8733
Epoch: 4, Iteration: 0, Loss: 0.3966, Accuracy: 0.9444
Epoch: 4, Iteration: 50, Loss: 0.4799, Accuracy: 0.8750
Epoch: 4, Iteration: 100, Loss: 0.4799, Accuracy: 0.8583
Epoch: 4, Iteration: 150, Loss: 0.3133, Accuracy: 0.8817
Epoch: 4, Iteration: 200, Loss: 0.4799, Accuracy: 0.8633
```

Task 4: Write code to print out your fine-tuned model's results, you can refer the code how we generate results for test dataset previously.

Compare the the two results you get, is the prediction accuracy better than the previous model? If not, think about the reasons for it perhaps.

```
#ft_model.eval()
# TODO: write code to evaluate the model on the test set
ft_model.eval()

# Initialize variables to track total and correct predictions
total = 0.0
correct = 0.0

# Disable gradient calculation for evaluation
with torch.no_grad():
    for data in test_loader:
        inputs = data['image'].to(device)
        aux = data['aux'].to(device)
        labels = data['label'].to(device)

        # Forward pass to get outputs
        outputs = ft_model(inputs, aux)

        # Get predicted class by taking the index with the highest score
        _, predicted = torch.max(outputs.data, 1)

        # Update the total and correct counts
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

# Calculate accuracy
accuracy = correct / total
print(f'Accuracy of the fine-tuned model on the test dataset: {accuracy:.4f}')
```


➡ Accuracy of the fine-tuned model on the test dataset: 0.5000

Task 5: Write code to visualize the image

["/content/cyberbullying_data/cyberbullying_data_splits_clean/test/cyberbullying/fingerGunAnnotated_239.JPEG"](/content/cyberbullying_data/cyberbullying_data_splits_clean/test/cyberbullying/fingerGunAnnotated_239.JPEG).

Then test this image with the fine-tuned model and print the prediction results.

```
from PIL import Image
import matplotlib.pyplot as plt

# Open the image file
image = "/content/cyberbullying_data/cyberbullying_data_splits_clean/test/cyberbullying/fingerGunAnnotated_239.JPEG"
img = Image.open(image)

# Plot the image
plt.imshow(img)
plt.axis('off') # Optional: to hide axes for a cleaner display
plt.title("Cyberbullying Label") # Add the label here, adjust as needed
plt.show()

# Define the transformations (ensure these match the ones used during training)
transform = transforms.Compose([
    transforms.Resize((224, 224)), # Resize to the input size expected by the model
    transforms.ToTensor(),         # Convert the image to a tensor
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Normalize using ImageNet stats
])

# Apply transformations
img_transformed = transform(img).unsqueeze(0) # Add a batch dimension

# Add an additional channel (e.g., a zero channel)
additional_channel = torch.zeros((1, 1, 224, 224)) # Shape: [batch_size, channels, height, width]
img_transformed = torch.cat((img_transformed, additional_channel), dim=1)

# Move the image to the appropriate device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
img_transformed = img_transformed.to(device)


# Set the model to evaluation mode
ft_model.eval()

# Assuming you have a list of class names
class_names = ['non-cyberbullying', 'cyberbullying'] # Update with your actual class names

# Print the prediction result
print(f'Predicted class: {class_names[predicted.item()]})')
```

➡

Cyberbullying Label



Predicted class: non-cyberbullying

```
from PIL import Image
from torchvision import transforms

# TODO: find the picture_index of the chosen image by comparing with the previous visualization cell
#picture_index =
#instance = test_set[picture_index]

# Load the reference image
image = "/content/cyberbullying_data/cyberbullying_data_splits_clean/test/cyberbullying/fingerGunAnnotated_239.JPEG"
loaded_img = Image.open(image)

# Define the transformations (ensure these match your dataset transformations)
transform = transforms.Compose([
```

```
transforms.Resize((224, 224))), # Resize to the input size expected by the model
transforms.ToTensor(),          # Convert the image to a tensor
transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])) # Normalize using ImageNet stats
])

# Transform the loaded image
loaded_img_transformed = transform(loaded_img)

# Find the index of the matching image
picture_index = None
for idx, instance in enumerate(test_set):
    img = instance['image'] # Assuming each instance is a dictionary with an 'image' key
    if torch.equal(img, loaded_img_transformed):
        picture_index = idx
        break

# Check if a match was found
if picture_index is not None:
    instance = test_set[picture_index]
    print(f"Picture index: {picture_index}")

    # check if the prediction is correct
    instance_image = instance['image'].to(device)
    instance_label = torch.tensor(instance['label']).to(device, dtype=torch.long)
    instance_aux = instance['aux'].to(device, dtype=torch.float)


    # TODO: get the prediction for the image
    # Set the model to evaluation mode
    ft_model.eval()

    # Disable gradient calculation
    with torch.no_grad():
        # Get model predictions, providing the auxiliary input
        outputs = ft_model(instance_image.unsqueeze(0), aux=instance_aux.unsqueeze(0))
        _, predicted = torch.max(outputs.data, 1)

        # Assuming you have a list of class names
        class_names = ['non-cyberbullying', 'cyberbullying'] # Update with your actual class names

        # Print the prediction result
        print(f'Predicted class: {class_names[predicted.item()]})')
        print(f'Actual class: {class_names[instance_label.item()]})')

        # Check if the prediction is correct
        if predicted.item() == instance_label.item():
            print("The prediction is correct.")
        else:
            print("The prediction is incorrect.")
else:
    print("Image not found in the dataset.")
```

 Image not found in the dataset.