

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%

Part 1:

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.

airplane



automobile



bird



cat



deer



```

IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM

+ Code + Text
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)))

```

ship ship car plane

```

IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM

+ Code + Text

# CNN Architecture
import torch.nn as nn
import torch.nn.functional as F

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

        # this is the conv net part
        self.convolutional_layer = nn.Sequential(
            # we can use the equation  $((N + 2p - f) / s) + 1$  to quickly compute the output of a convolution op.
            # however, PyTorch makes it easy for us, since it computes this equation for us. All we have to do is tell it the number of channels we want
            # in the input and the output. For the first convolution layer, the in_channels would be the channels a standard RGB image has. Let's say we want
            # number of channels in the output to be 20. We want to use 5x5 convolutions and single strides. Write the convolution layers.
            nn.Conv2d(in_channels = 3, out_channels = 20, kernel_size = 5, stride = 1),
            nn.ReLU(),
            # Lets subsample after convolution. We will use Max Pooling in this example.
            nn.MaxPool2d(kernel_size = 2, stride = 2),
            # Let's add another convolutional layer. We want the output of this convolution operation to have 50 channels. Let's use 5x5 convolutions and single
            nn.Conv2d(in_channels = 20, out_channels = 50, kernel_size = 5, stride = 1),
            nn.ReLU(),
            # Subsample again
            nn.MaxPool2d(kernel_size = (2,2), stride = 2),
        )

        # this is the classifier head. So the model is actually CNN + DNN. We will flatten the output from the CNN above for the DNN.

```

```

IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM

+ Code + Text

nn.ReLU(),
# Lets subsample after convolution. We will use Max Pooling in this example.
nn.MaxPool2d(kernel_size = 2, stride = 2),
# Let's add another convolutional layer. We want the output of this convolution operation to have 50 channels. Let's use 5x5 convolutions and single
nn.Conv2d(in_channels = 20, out_channels = 50, kernel_size = 5, stride = 1),
nn.ReLU(),
# Subsample again
nn.MaxPool2d(kernel_size = (2,2), stride = 2),
)

# this is the classifier head. So the model is actually CNN + DNN. We will flatten the output from the CNN above for the DNN.
self.linear_layer = nn.Sequential(
    nn.Linear(in_features = 50*5*5, out_features = 500),
    nn.ReLU(),
    nn.Linear(in_features = 500, out_features = 10), # observe how we have output features = 10. There are 10 classes in this problem.
)

def forward(self, x):
    x = self.convolutional_layer(x)
    x = torch.flatten(x, 1) # Flattening the output of the CNN for the DNN
    x = self.linear_layer(x)
    x = F.softmax(x, dim = 1)
    return x

net = MyObjDetectorCNN()

```

```

Device Change to GPU if Available

[ ] device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

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

net.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr = 0.001, momentum = 0.9, weight_decay=0.0001)

```

```
# 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(35): # 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 # to use CPU
        inputs, labels = data[0].to(device), data[1].to(device) # using GPU

        # 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')
```

```
IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM


+ Code + Text
[30, 6000] loss: 1.509
[30, 8000] loss: 1.512
[30, 10000] loss: 1.513
[31, 2000] loss: 1.504
[31, 4000] loss: 1.508
[31, 6000] loss: 1.507
[31, 8000] loss: 1.511
[31, 10000] loss: 1.508
[32, 2000] loss: 1.505
[32, 4000] loss: 1.508
[32, 6000] loss: 1.505
[32, 8000] loss: 1.506
[32, 10000] loss: 1.505
[33, 2000] loss: 1.503
[33, 4000] loss: 1.504
[33, 6000] loss: 1.507
[33, 8000] loss: 1.505
[33, 10000] loss: 1.506
[34, 2000] loss: 1.500
[34, 4000] loss: 1.505
[34, 6000] loss: 1.504
[34, 8000] loss: 1.505
[34, 10000] loss: 1.506
[35, 2000] loss: 1.502
[35, 4000] loss: 1.505
[35, 6000] loss: 1.502
[35, 8000] loss: 1.501
[35, 10000] loss: 1.500
Finished Training
```

```
IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM

+ Code + Text
y_test_predictions = np.array(y_test_predictions)

accuracy = accuracy_score(y_test, y_test_predictions)
precision = precision_score(y_test, y_test_predictions, average='weighted', zero_division=1)
recall = recall_score(y_test, y_test_predictions, average='weighted')
f1 = f1_score(y_test, y_test_predictions, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```



```
GroundTruth: cat ship ship plane
Accuracy: 0.7143
Precision: 0.7173
Recall: 0.7143
F1-score: 0.7153
```

Part 2:

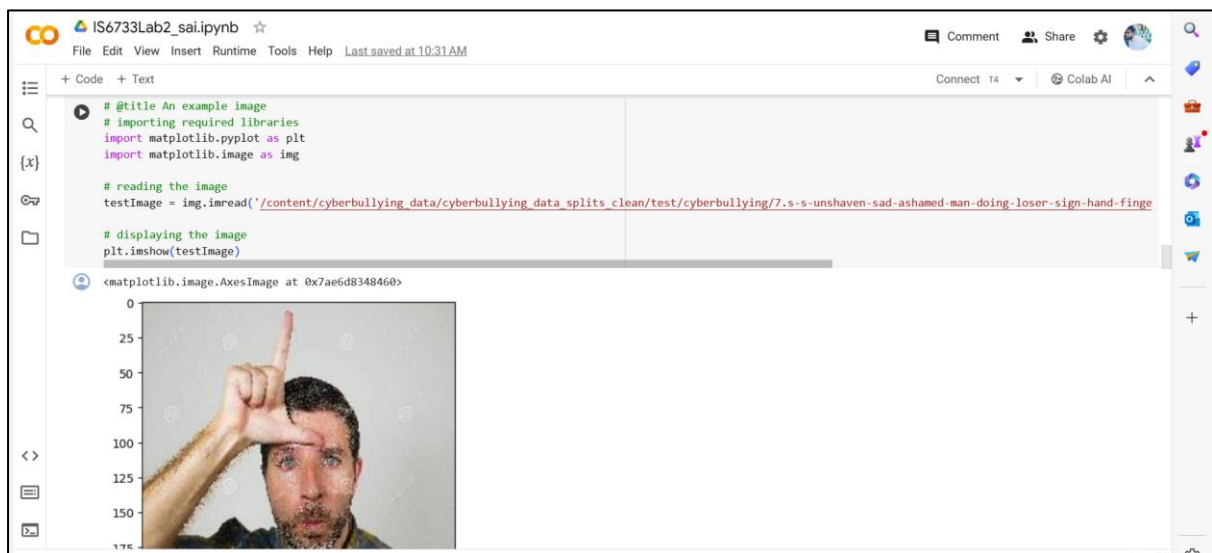
Part 2: Using a Pre-trained CNN to Detect Cyberbullying in Images

With previous lab learning, you should have some knowledge about how to develop an AI model to detect cyberbullying language. 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/>



Generate the detection results for test data

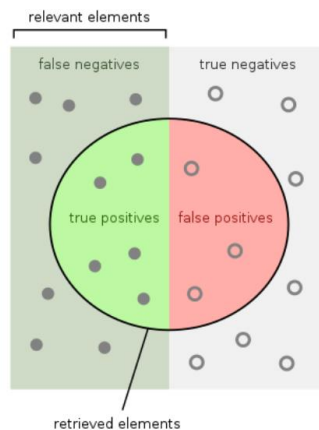
Now, it's time to evaluate the pre-trained model's capability with our test dataset

```
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: {:.4f}%'.format((correct / total) * 100))
```

Test loss is: 0.449
The accuracy for test dataset is: 85.0000%

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



▶ # TODO: Complete the following code to calculate the accuracy, precision, recall and F1 score.

```
accuracy = (tp + tn) / (tp + tn + fp + fn)

precision = tp / (tp + fp)

recall = tp / (tp + fn)

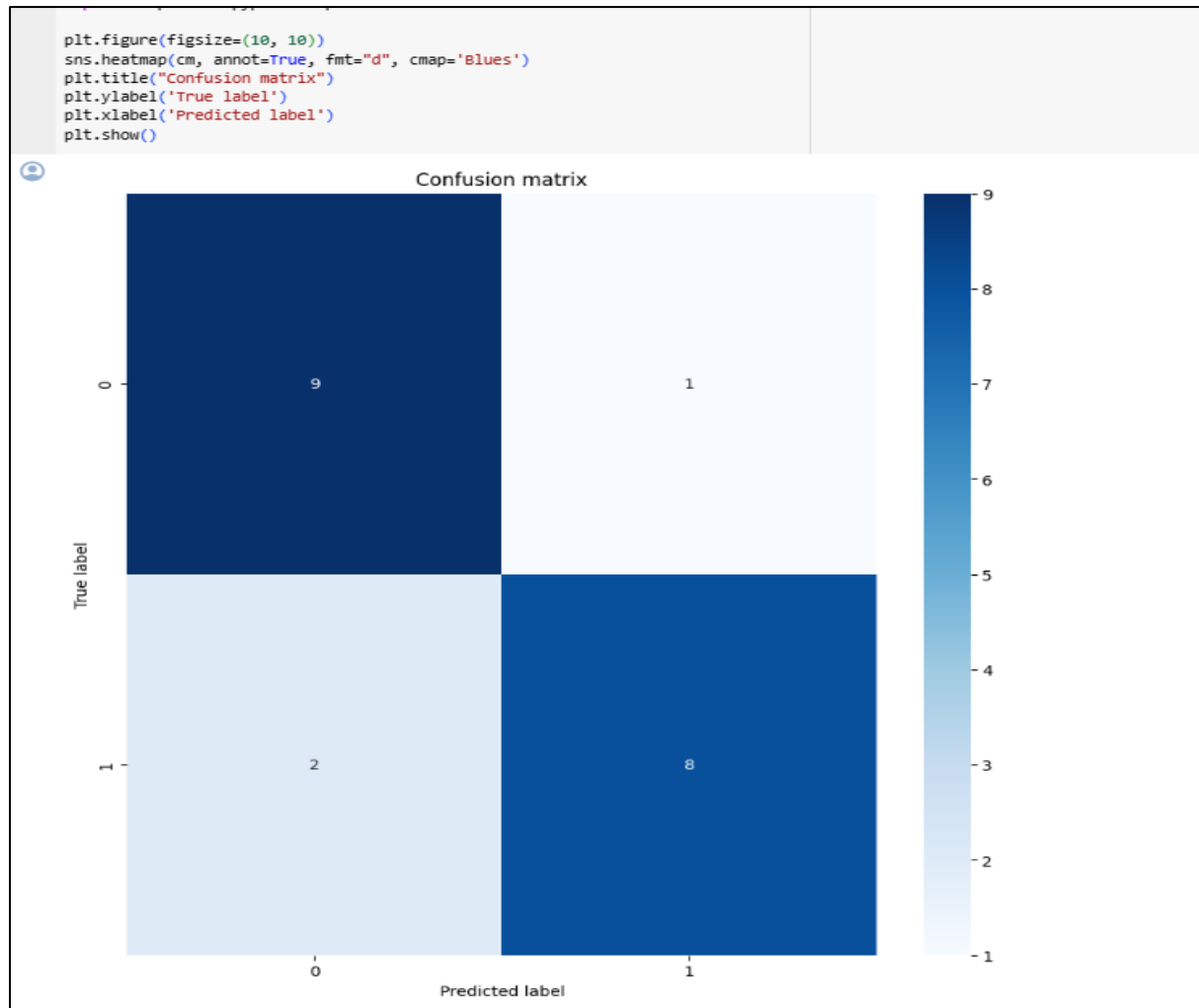
f1_score = 2 * (precision * recall) / (precision + recall)

acc =accuracy
precision =precision
recall =recall
f1 =f1_score

print('The accuracy for test dataset is: {:.4f}%'.format(acc * 100))
print('The precision for test dataset is: {:.4f}%'.format(precision * 100))
print('The recall for test dataset is: {:.4f}%'.format(recall * 100))
print('The f1 score for test dataset is: {:.4f}%'.format(f1 * 100))
```

👤 The accuracy for test dataset is: 85.0000%
 The precision for test dataset is: 88.8889%
 The recall for test dataset is: 80.0000%
 The f1 score for test dataset is: 84.2105%

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))



Let's check with one instance

```
[ ] # 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.

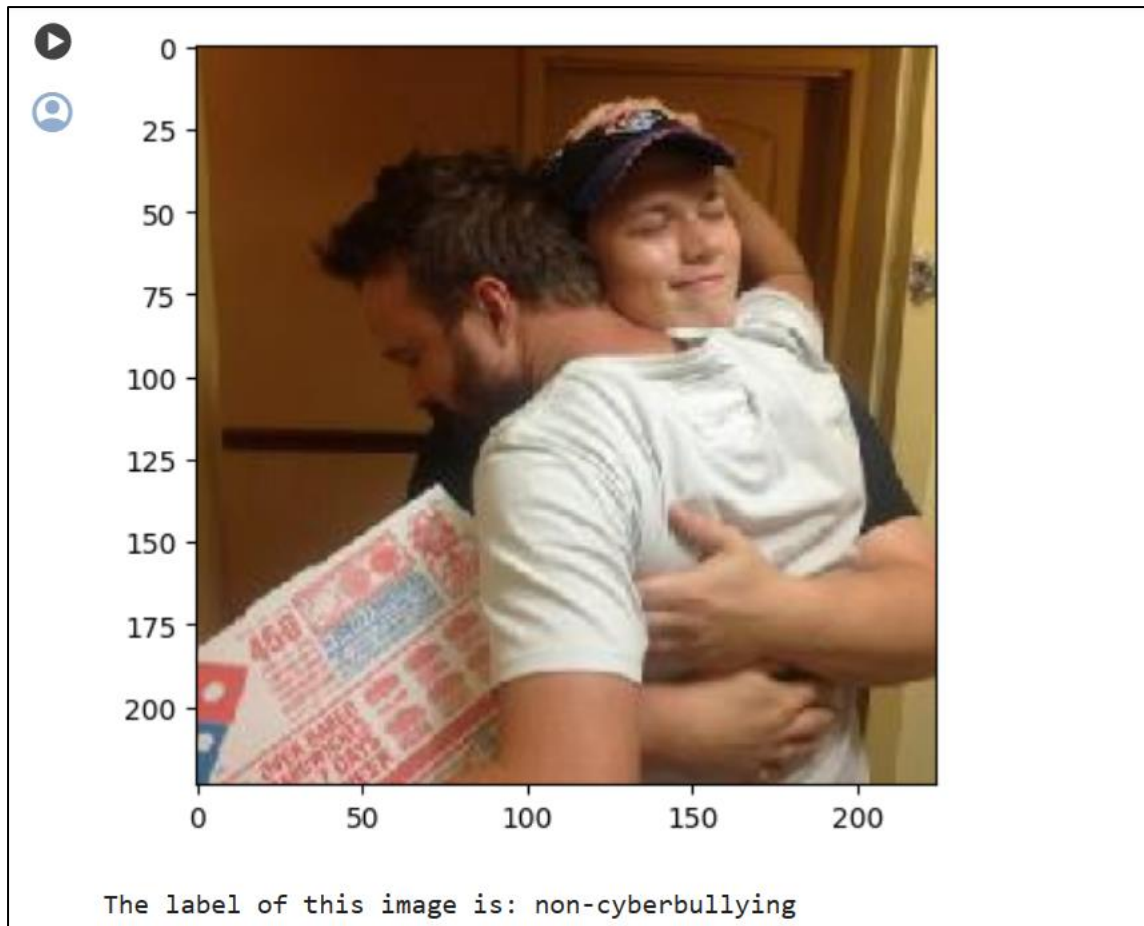
Select a number to view the image and its label.

picture_index: 18

```
picture_index = "18" #@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 "n
print('')
print("The label of this image is: {}".format(annot_label))
```

0



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), in

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

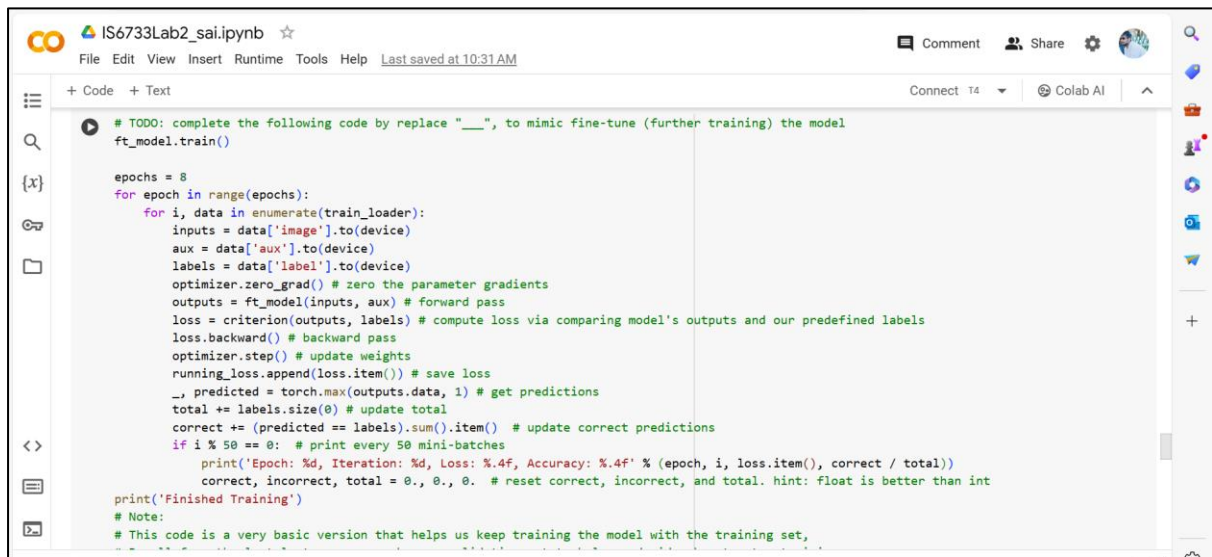
Task 3: Write code to fine-tune the model with the training dataset

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

```
[ ] # prepare the optimizer, loss function, learning rate
import torch.optim as optim

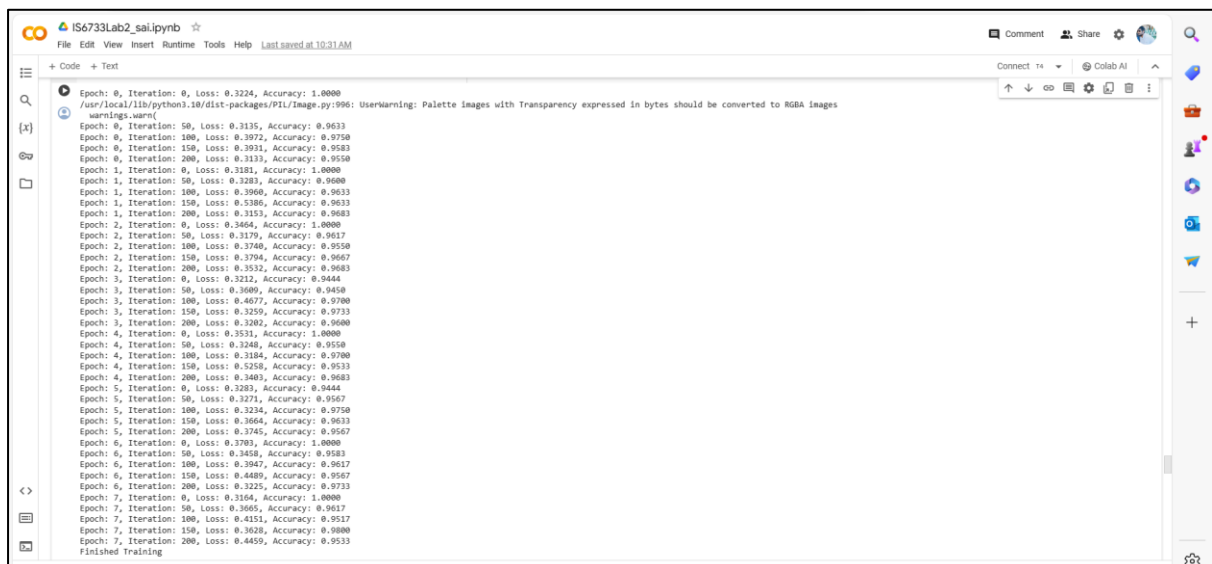
criterion = nn.CrossEntropyLoss().to(device)
# optimizer = optim.Adam(ft_model.parameters(), lr = 1.0000e-06, weight_decay=0.00001)
optimizer = optim.Adam(ft_model.parameters(), lr = 1.0e-06, weight_decay=0.1)
#optimizer = optim.Adam(ft_model.parameters(), lr = 0.001, weight_decay=0.0005)
# Fine-tune only the top layers while keeping the early layers frozen
for param in ft_model.parameters():
    param.requires_grad = False

# Unfreeze the final layers for fine-tuning
for param in ft_model.classifier.parameters():
    param.requires_grad = True
```



```
IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM
+ Code + Text
# TODO: complete the following code by replace "___", to mimic fine-tune (further training) the model
ft_model.train()

epochs = 8
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
        loss = criterion(outputs, labels) # compute loss via comparing model's outputs and our predefined 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
        correct += (predicted == labels).sum().item() # update correct predictions
        if i % 50 == 0: # print every 50 mini-batches
            print('Epoch: %d, Iteration: %d, Loss: %.4f, Accuracy: %.4f' % (epoch, i, loss.item(), correct / total))
            correct, incorrect, total = 0, 0, 0. # reset correct, incorrect, and total. hint: float is better than int
    print('Finished Training')
# Note:
# This code is a very basic version that helps us keep training the model with the training set,
```



```
IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM
+ Code + Text
Epoch: 0, Iteration: 0, Loss: 0.3224, Accuracy: 1.0000
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:996: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
Epoch: 0, Iteration: 50, Loss: 0.3135, Accuracy: 0.9633
Epoch: 0, Iteration: 100, Loss: 0.3972, Accuracy: 0.9750
Epoch: 0, Iteration: 150, Loss: 0.3931, Accuracy: 0.9583
Epoch: 0, Iteration: 200, Loss: 0.3133, Accuracy: 0.9550
Epoch: 1, Iteration: 0, Loss: 0.3181, Accuracy: 1.0000
Epoch: 1, Iteration: 50, Loss: 0.3283, Accuracy: 0.9600
Epoch: 1, Iteration: 100, Loss: 0.3960, Accuracy: 0.9633
Epoch: 1, Iteration: 150, Loss: 0.5386, Accuracy: 0.9633
Epoch: 1, Iteration: 200, Loss: 0.3153, Accuracy: 0.9683
Epoch: 2, Iteration: 0, Loss: 0.3464, Accuracy: 1.0000
Epoch: 2, Iteration: 50, Loss: 0.3179, Accuracy: 0.9617
Epoch: 2, Iteration: 100, Loss: 0.3740, Accuracy: 0.9550
Epoch: 2, Iteration: 150, Loss: 0.3794, Accuracy: 0.9667
Epoch: 2, Iteration: 200, Loss: 0.3532, Accuracy: 0.9683
Epoch: 3, Iteration: 0, Loss: 0.3212, Accuracy: 0.9444
Epoch: 3, Iteration: 50, Loss: 0.3699, Accuracy: 0.9450
Epoch: 3, Iteration: 100, Loss: 0.4677, Accuracy: 0.9700
Epoch: 3, Iteration: 150, Loss: 0.3259, Accuracy: 0.9733
Epoch: 3, Iteration: 200, Loss: 0.3202, Accuracy: 0.9600
Epoch: 4, Iteration: 0, Loss: 0.3531, Accuracy: 1.0000
Epoch: 4, Iteration: 50, Loss: 0.3248, Accuracy: 0.9550
Epoch: 4, Iteration: 100, Loss: 0.3184, Accuracy: 0.9700
Epoch: 4, Iteration: 150, Loss: 0.5258, Accuracy: 0.9533
Epoch: 4, Iteration: 200, Loss: 0.3403, Accuracy: 0.9683
Epoch: 5, Iteration: 0, Loss: 0.3283, Accuracy: 0.9444
Epoch: 5, Iteration: 50, Loss: 0.3271, Accuracy: 0.9567
Epoch: 5, Iteration: 100, Loss: 0.3234, Accuracy: 0.9750
Epoch: 5, Iteration: 150, Loss: 0.3664, Accuracy: 0.9633
Epoch: 5, Iteration: 200, Loss: 0.3745, Accuracy: 0.9567
Epoch: 6, Iteration: 0, Loss: 0.3703, Accuracy: 1.0000
Epoch: 6, Iteration: 50, Loss: 0.3458, Accuracy: 0.9583
Epoch: 6, Iteration: 100, Loss: 0.3947, Accuracy: 0.9617
Epoch: 6, Iteration: 150, Loss: 0.4489, Accuracy: 0.9567
Epoch: 6, Iteration: 200, Loss: 0.3225, Accuracy: 0.9733
Epoch: 7, Iteration: 0, Loss: 0.3164, Accuracy: 1.0000
Epoch: 7, Iteration: 50, Loss: 0.3665, Accuracy: 0.9617
Epoch: 7, Iteration: 100, Loss: 0.4151, Accuracy: 0.9517
Epoch: 7, Iteration: 150, Loss: 0.3628, Accuracy: 0.9800
Epoch: 7, Iteration: 200, Loss: 0.4459, Accuracy: 0.9533
Finished Training
```


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

y_true = []
y_pred = []

with torch.no_grad():
    for data in test_loader:
        images, labels, aux = data['image'].to(device), data['label'].to(device), data['aux'].to(device)

        # Forward pass
        outputs = ft_model(images, aux)

        # Get predicted labels
        _, predicted = torch.max(outputs, 1)

        # Store true and predicted labels
        y_true.extend(labels.cpu().numpy())
```

```
IS6733Lab2_sai.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:31 AM
+ Code + Text
# Store true and predicted labels
y_true.extend(labels.cpu().numpy())
y_pred.extend(predicted.cpu().numpy())

# Calculate metrics using sklearn
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_last = accuracy_score(y_true, y_pred)
precision_last = precision_score(y_true, y_pred, average='weighted', zero_division=1)
recall_last = recall_score(y_true, y_pred, average='weighted')
f1_last = f1_score(y_true, y_pred, average='weighted')

print('The accuracy for test dataset is: {:.4f}%'.format(accuracy_last * 100))
print('The precision for test dataset is: {:.4f}%'.format(precision_last * 100))
print('The recall for test dataset is: {:.4f}%'.format(recall_last * 100))
print('The f1 score for test dataset is: {:.4f}%'.format(f1_last * 100))

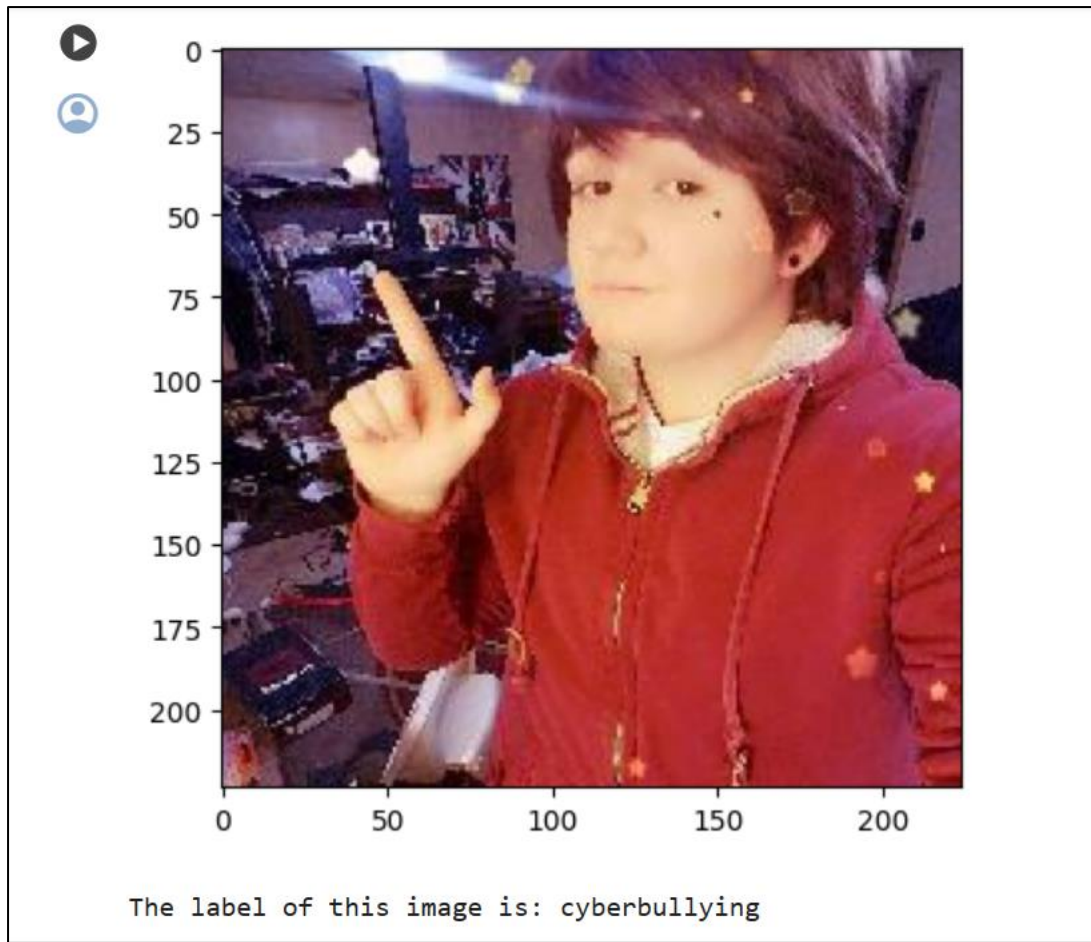
The accuracy for test dataset is: 80.0000%
The precision for test dataset is: 81.2500%
The recall for test dataset is: 80.0000%
The f1 score for test dataset is: 79.7980%
```

Here the fine tune(ft_model) model test dataset accuracy is of 80% and the previous model test dataset accuracy is of 85%, there is slight decrease in test dataset accuracy about 5%, perhaps the reasons could be of change in parameter tuning and may be the fine tune(ft_model) model would have converged as it reached the maximum performance for this dataset. Furthermore, when compared to fine tune(ft_model) test dataset accuracy(80%) to the original model test dataset accuracy(85%) their isn't large number change in the test accuracy metric and more over when predicting the images for this data with fine tune(ft_model) model it is predicting correct results and the fine tune(ft_model) model is neither over-fitting or under-fitting and model reached its convergence.

Task 5: Write code to visualize the image

["/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.



```
# TODO: find the picture_index of the chosen image by comparing with the previous visualization cell
picture_index = 6
instance = test_set[picture_index]

# 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), in

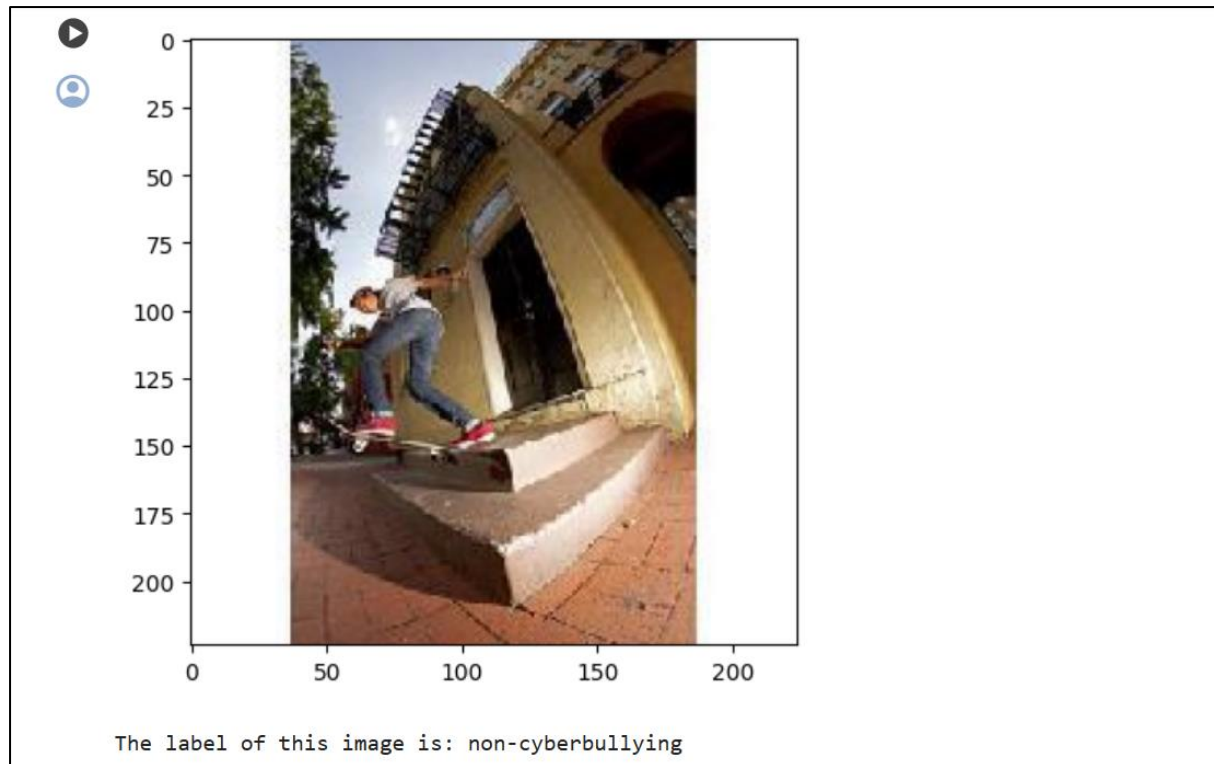
# TODO: get the prediction for the image
output = ft_model(instance_image.unsqueeze(0), instance_aux.unsqueeze(0)).data
_, predicted = torch.max(output.data, 1)

predict_label = "cyberbullying" if predicted.item() == 1 else "non-cyberbullying"
comparison = "correct" if predicted == instance_label else "not correct"

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

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

Testing for another image with fine tuned model ft_model:



```
# TODO: find the picture_index of the chosen image by comparing with the previous visualization cell
picture_index = 14
instance = test_set[picture_index]

# 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), instance['aux']

# TODO: get the prediction for the image
output = ft_model(instance_image.unsqueeze(0), instance_aux.unsqueeze(0)).data
_, predicted = torch.max(output, 1)

predict_label = "cyberbullying" if predicted.item() == 1 else "non-cyberbullying"
comparison = "correct" if predicted == instance_label else "not correct"

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

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

Another Image: Predicting on the whole Test_set using the fine tuned model called "ft_model":

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

picture_index = "0" #@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))
```

Select a number to view the image and its label.

picture_index: 0



```
[ ] # 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), in

output = ft_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: cyberbullying, which is correct!
```