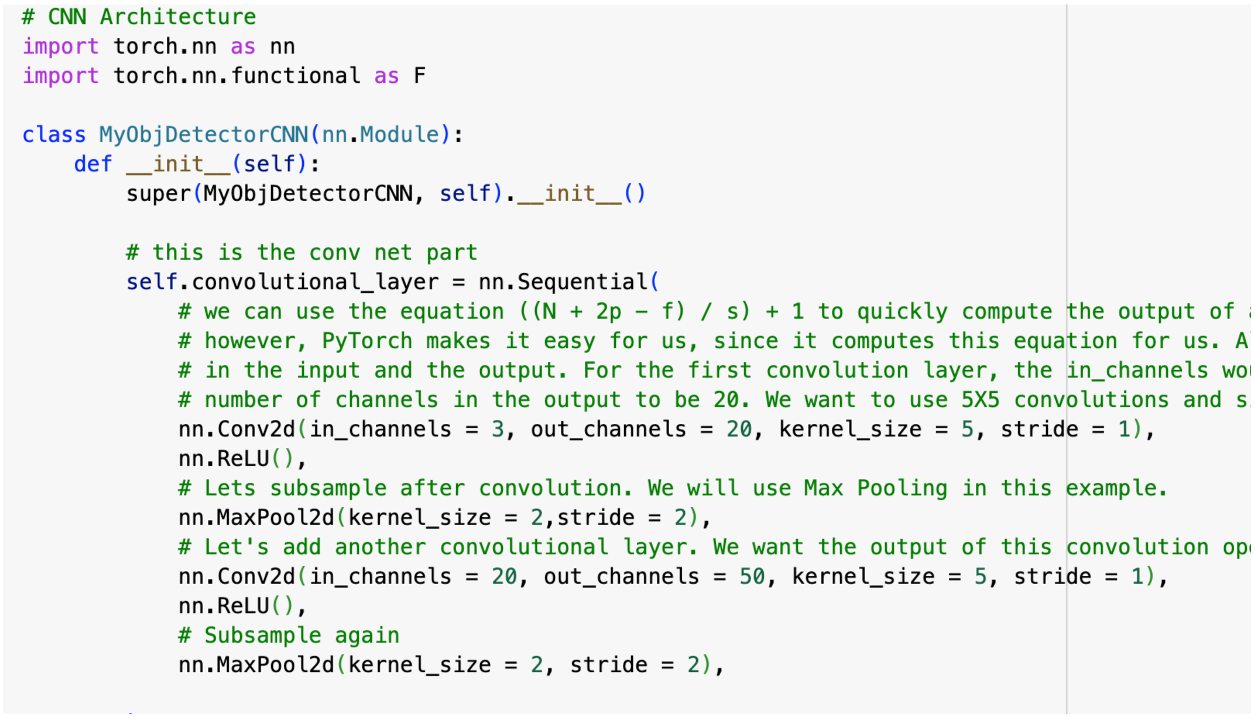
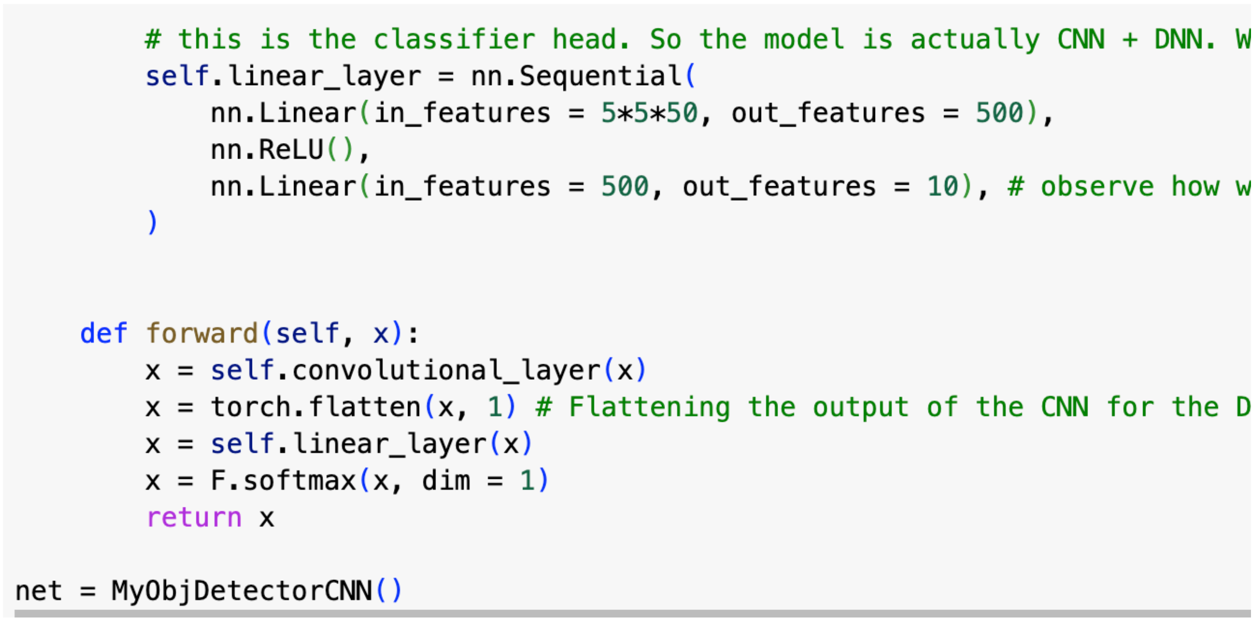
Lab 2: Convolutional Neural Networks for Computer Vision

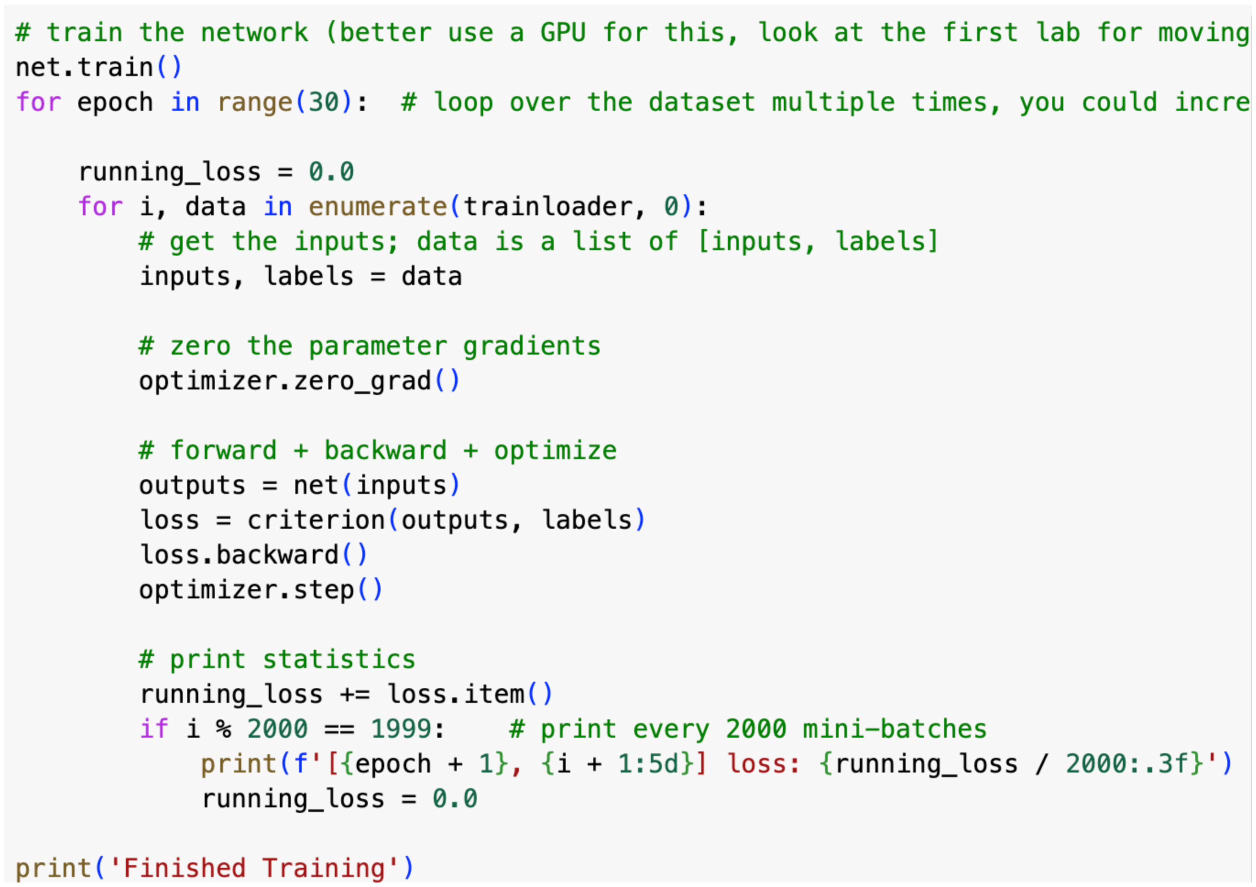
Part 1: CNN to Detect Images of 10 Objects



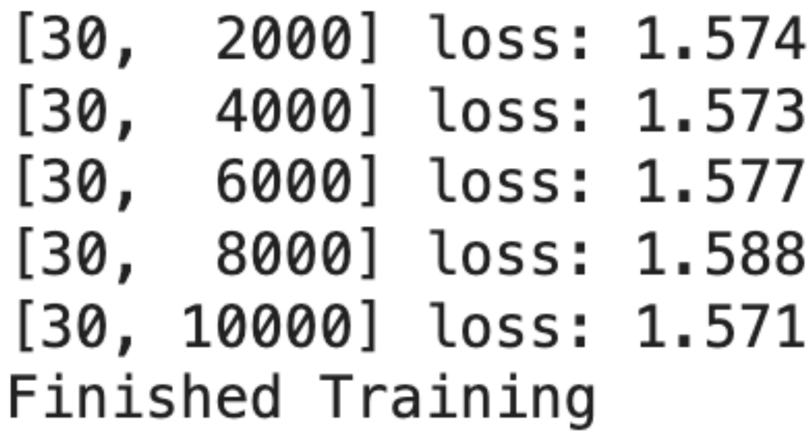
```
# CNN Architecture
import torch.nn as nn
import torch.nn.functional as F

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
            # however, PyTorch makes it easy for us, since it computes this equation for us. A
            # in the input and the output. For the first convolution layer, the in_channels wo
            # number of channels in the output to be 20. We want to use 5X5 convolutions and s
            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 op
            nn.Conv2d(in_channels = 20, out_channels = 50, kernel_size = 5, stride = 1),
            nn.ReLU(),
            # Subsample again
            nn.MaxPool2d(kernel_size = 2, stride = 2),
```



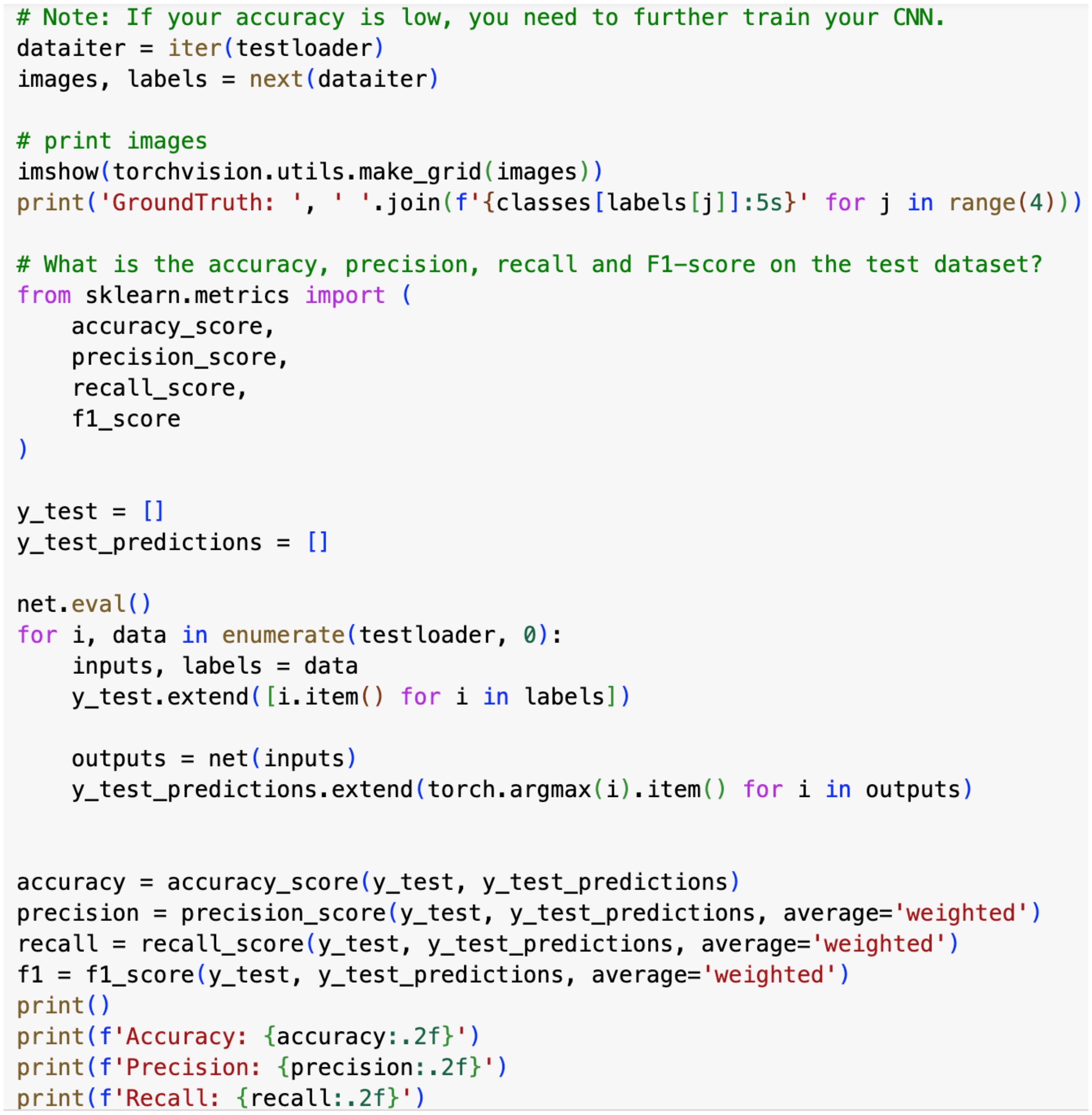
```
        # this is the classifier head. So the model is actually CNN + DNN. W
        self.linear_layer = nn.Sequential(
            nn.Linear(in_features = 5*5*50, out_features = 500),
            nn.ReLU(),
            nn.Linear(in_features = 500, out_features = 10), # observe how w
        )


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

net = MyObjDetectorCNN()
```



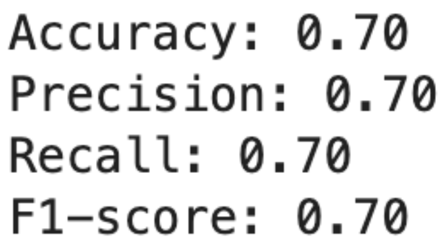
```python
# train the network (better use a GPU for this, look at the first lab for moving
net.train()
for epoch in range(30):  # loop over the dataset multiple times, you could incre

    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')
```



```
[30,  2000] loss: 1.574
[30,  4000] loss: 1.573
[30,  6000] loss: 1.577
[30,  8000] loss: 1.588
[30, 10000] loss: 1.571
Finished Training
```



```python
# 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(f'{classes[labels[j]]:5s}' 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)


accuracy = accuracy_score(y_test, y_test_predictions)
precision = precision_score(y_test, y_test_predictions, average='weighted')
recall = recall_score(y_test, y_test_predictions, average='weighted')
f1 = f1_score(y_test, y_test_predictions, average='weighted')
print()
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

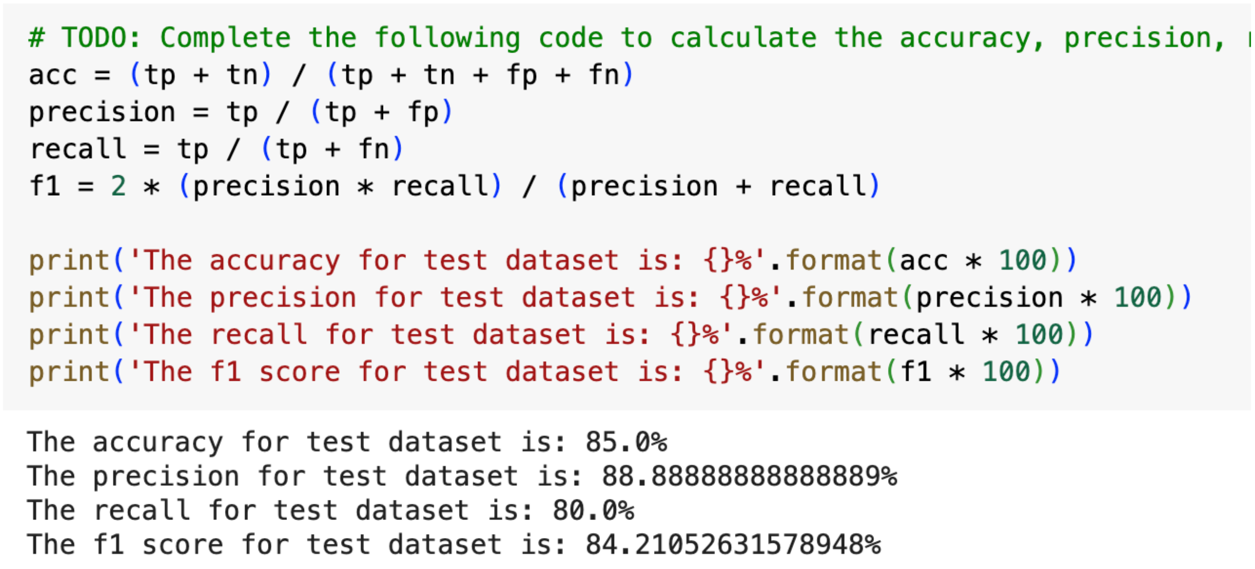


```
Accuracy: 0.70
Precision: 0.70
Recall: 0.70
F1-score: 0.70
```

Changing the number of epochs to 30 improved the metrics to 0.7.

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

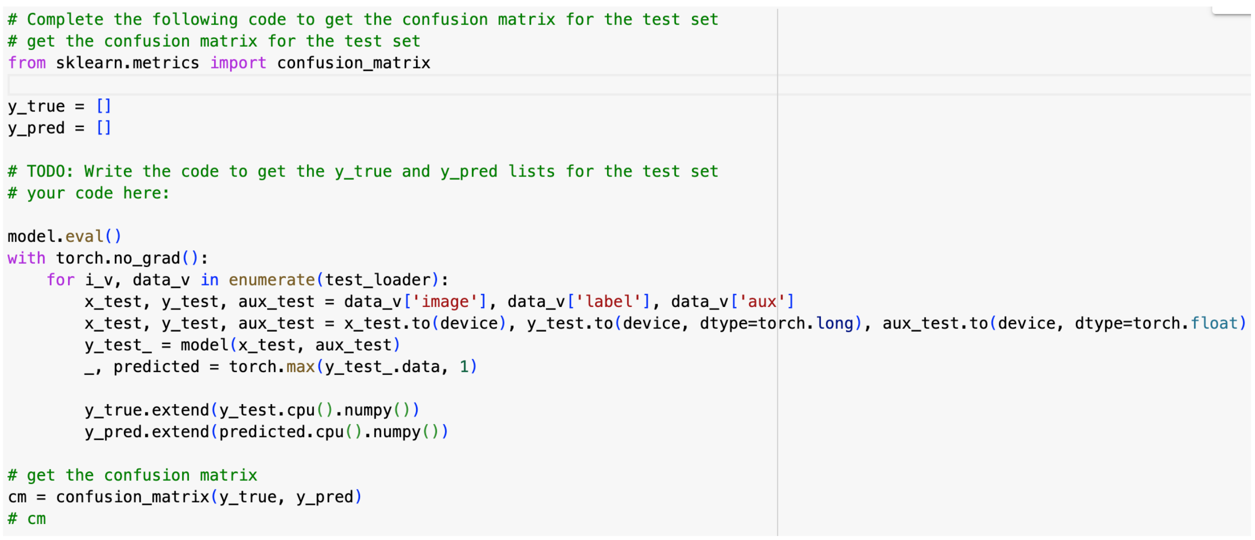
Task 1: Write code to generate result report contains: Accuracy, Precision, Recall and F1-Score



```
# TODO: Complete the following code to calculate the accuracy, precision,
acc = (tp + tn) / (tp + tn + fp + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1 = 2 * (precision * recall) / (precision + recall)

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))



```
# Complete the following code to get the confusion matrix for the test set
# get the confusion matrix for the test set
from sklearn.metrics import confusion_matrix

y_true = []
y_pred = []

# TODO: Write the code to get the y_true and y_pred lists for the test set
# your code here:

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)
        _, predicted = torch.max(y_test_.data, 1)

        y_true.extend(y_test.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

# get the confusion matrix
cm = confusion_matrix(y_true, y_pred)
# cm
```

```python
# plot the confusion matrix
import seaborn as sns
import matplotlib.pyplot as plt

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()
```

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

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

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

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

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
        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, 0.0, 0.0 # reset correct, incorrect, and total. hint: float is better t
```

```
Epoch: 4, Iteration: 0, Loss: 0.5633, Accuracy: 0.7778
Epoch: 4, Iteration: 50, Loss: 0.3133, Accuracy: 0.8633
Epoch: 4, Iteration: 100, Loss: 0.3133, Accuracy: 0.8700
Epoch: 4, Iteration: 150, Loss: 0.3133, Accuracy: 0.8750
Epoch: 4, Iteration: 200, Loss: 0.4799, Accuracy: 0.8750
```

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.

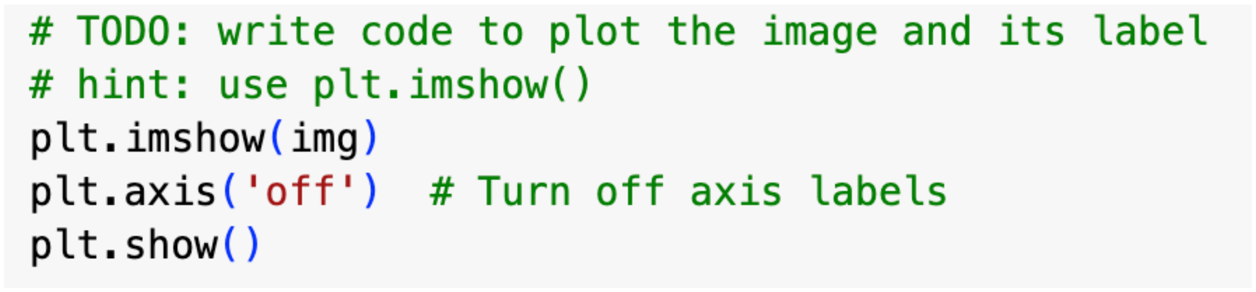
Compare the two results you get, is the prediction accuracy better than the previous model? If not, think about the reasons for it perhaps.

```python
# TODO: write code to evaluate the model on the test set
correct = 0
total = 0

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)
        outputs = ft_model(images, aux)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

cm = confusion_matrix(y_true, y_pred)
print(cm)
accuracy = correct / total

print('Accuracy on the test set: {:.2f}%'.format(accuracy * 100))
```

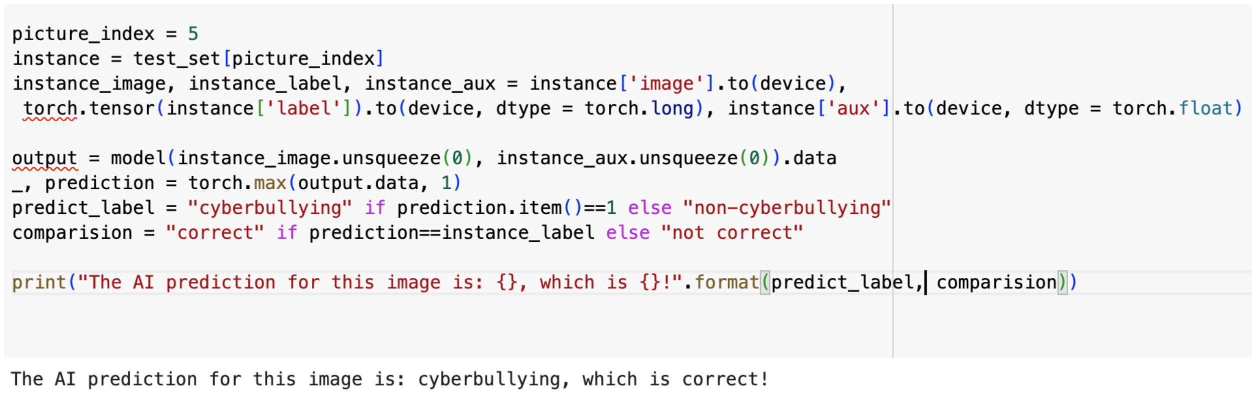
```
[[10  0]
 [10  0]]
Accuracy on the test set: 50.00%
```

As you can see above, the accuracy on the test set is 50%, which is worse than the previous model. Based on the confusion matrix, we can see that there were 10 true positives and 10 false negatives. This means that the model correctly identified 10 instances of one class, but it incorrectly classified all instances of the other class as belonging to the first class. This suggests that the model is biased or not effectively capturing the patterns in the data, leading to poor performance. The poor performance could be due to incorrect hyperparameters, meaning accuracy could improve if maybe the learning rate or number of epochs was adjusted. Another reason for the poor performance could be that the distribution of the test set might differ significantly from the distribution of the training set. If the fine-tuned model is specialized to the training data but fails to generalize to the unseen test data with different characteristics, it could result in a drop in accuracy on the test set.

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.



```python
# TODO: write code to plot the image and its label
# hint: use plt.imshow()
plt.imshow(img)
plt.axis('off')  # Turn off axis labels
plt.show()
```



```python
picture_index = 5
instance = test_set[picture_index]
instance_image, instance_label, instance_aux = instance['image'].to(device),
 torch.tensor(instance['label']).to(device, dtype = torch.long), instance['aux'].to(device, dtype = torch.float)

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: cyberbullying, which is correct!
```