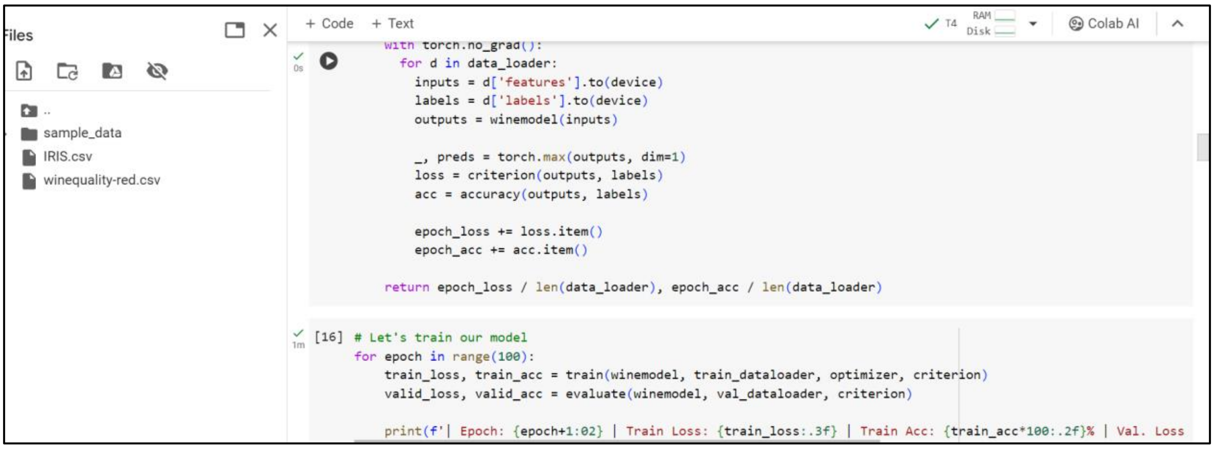
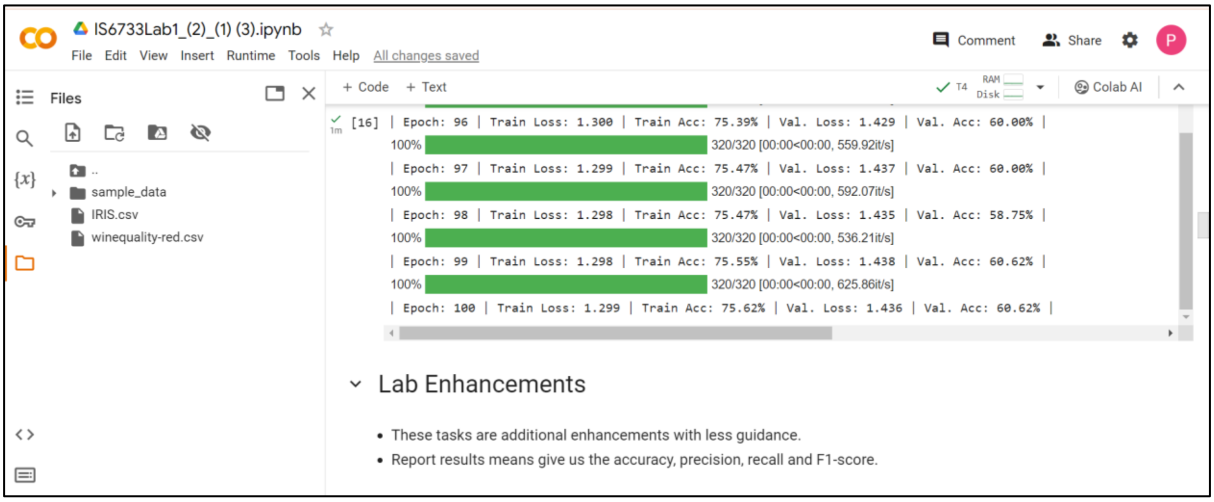
Partha Sai Madallapalli

- In this lab, you will first train a neural network on a public dataset, then make several enhancements to the lab.
- Tasks breakdown:
    - Code running: 10%



```
    with torch.no_grad():
      for d in data_loader:
        inputs = d['features'].to(device)
        labels = d['labels'].to(device)
        outputs = winemodel(inputs)

        _, preds = torch.max(outputs, dim=1)
        loss = criterion(outputs, labels)
        acc = accuracy(outputs, labels)

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(data_loader), epoch_acc / len(data_loader)
```

```
[16] # Let's train our model
     for epoch in range(100):
         train_loss, train_acc = train(winemodel, train_dataloader, optimizer, criterion)
         valid_loss, valid_acc = evaluate(winemodel, val_dataloader, criterion)

         print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss
```



```
[16] | Epoch: 96 | Train Loss: 1.300 | Train Acc: 75.39% | Val. Loss: 1.429 | Val. Acc: 60.00% |
     100% 320/320 [00:00<00:00, 559.92it/s]
     | Epoch: 97 | Train Loss: 1.299 | Train Acc: 75.47% | Val. Loss: 1.437 | Val. Acc: 60.00% |
     100% 320/320 [00:00<00:00, 592.07it/s]
     | Epoch: 98 | Train Loss: 1.298 | Train Acc: 75.47% | Val. Loss: 1.435 | Val. Acc: 58.75% |
     100% 320/320 [00:00<00:00, 536.21it/s]
     | Epoch: 99 | Train Loss: 1.298 | Train Acc: 75.55% | Val. Loss: 1.438 | Val. Acc: 60.62% |
     100% 320/320 [00:00<00:00, 625.86it/s]
     | Epoch: 100 | Train Loss: 1.299 | Train Acc: 75.62% | Val. Loss: 1.436 | Val. Acc: 60.62% |
```

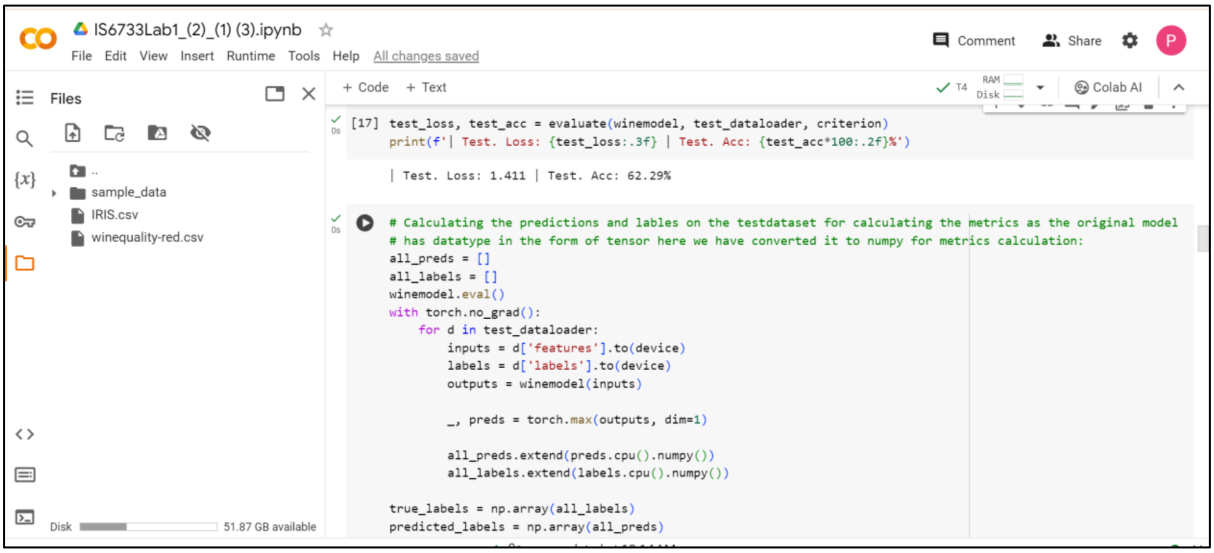
Lab Enhancements

- These tasks are additional enhancements with less guidance.
- Report results means give us the accuracy, precision, recall and F1-score.

Enhancement 1: The current code does not actually evaluate the model on the test set, but it only evaluates it on the val set. When you write papers, you would ideally split the dataset into train, val and test. Train and val are both used in training, and the model trained on the training data, and evaluated on the val data. So why do we need test split? We report our results on the test split in papers. Also, we do cross-validation on the train/val split (covered in later labs).

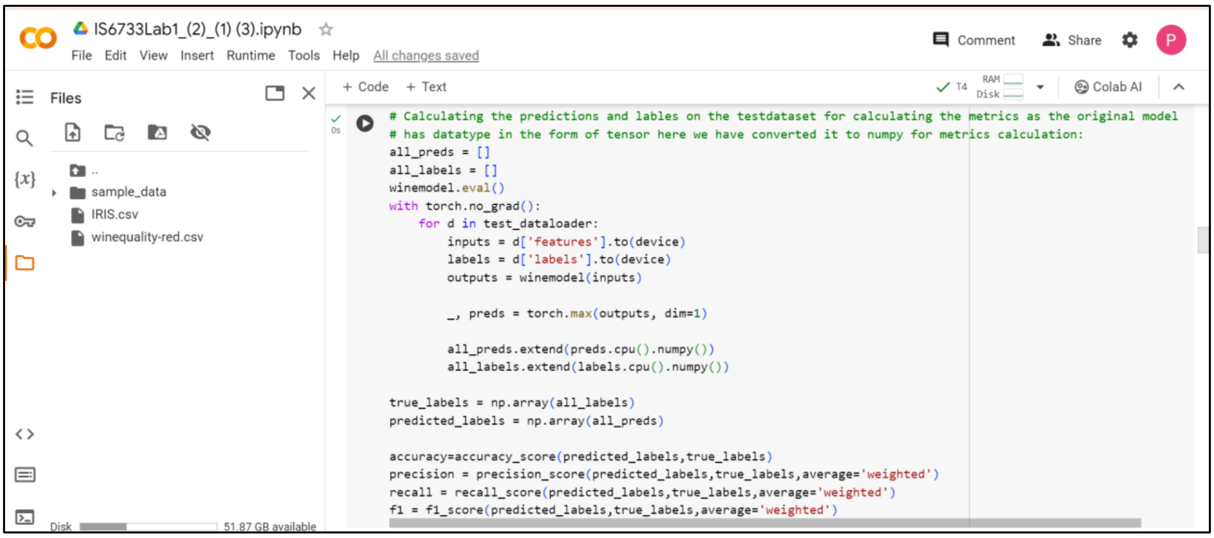
Partha Sai Madallapalli

Report the results of the model on the test split. (Hint: It would be exactly like the evaluation on the val dataset, except it would be done on the test dataset.)

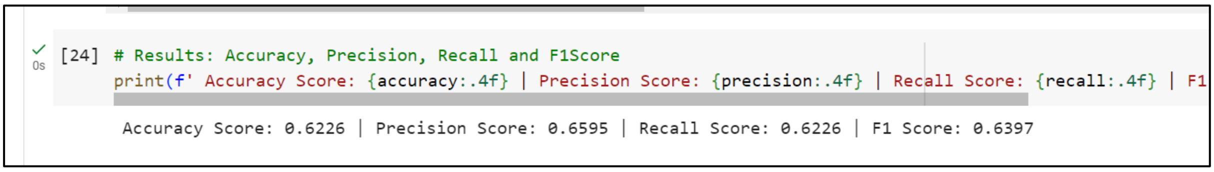


```
[17] test_loss, test_acc = evaluate(winemodel, test_dataloader, criterion)
     print(f'| Test. Loss: {test_loss:.3f} | Test. Acc: {test_acc*100:.2f}%')

     | Test. Loss: 1.411 | Test. Acc: 62.29%
```

```
# Calculating the predictions and lables on the testdataset for calculating the metrics as the original model
# has datatype in the form of tensor here we have converted it to numpy for metrics calculation:
all_preds = []
all_labels = []
winemodel.eval()
with torch.no_grad():
    for d in test_dataloader:
        inputs = d['features'].to(device)
        labels = d['labels'].to(device)
        outputs = winemodel(inputs)

        _, preds = torch.max(outputs, dim=1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

true_labels = np.array(all_labels)
predicted_labels = np.array(all_preds)
```



```
# Calculating the predictions and lables on the testdataset for calculating the metrics as the original model
# has datatype in the form of tensor here we have converted it to numpy for metrics calculation:
all_preds = []
all_labels = []
winemodel.eval()
with torch.no_grad():
    for d in test_dataloader:
        inputs = d['features'].to(device)
        labels = d['labels'].to(device)
        outputs = winemodel(inputs)

        _, preds = torch.max(outputs, dim=1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

true_labels = np.array(all_labels)
predicted_labels = np.array(all_preds)

accuracy=accuracy_score(predicted_labels,true_labels)
precision = precision_score(predicted_labels,true_labels,average='weighted')
recall = recall_score(predicted_labels,true_labels,average='weighted')
f1 = f1_score(predicted_labels,true_labels,average='weighted')
```

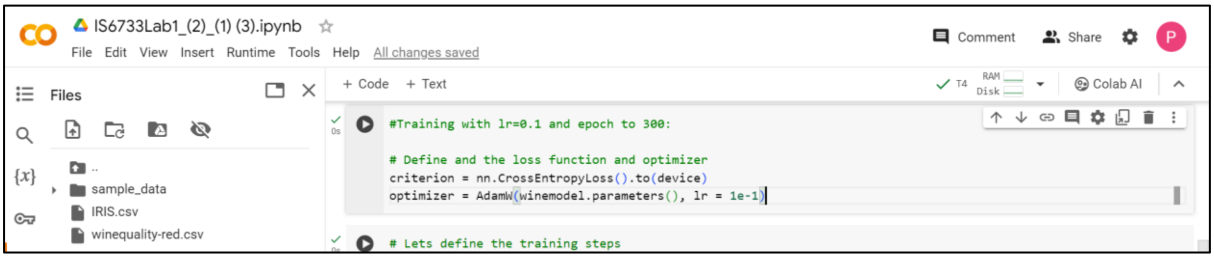


```
[24] # Results: Accuracy, Precision, Recall and F1Score
     print(f' Accuracy Score: {accuracy:.4f} | Precision Score: {precision:.4f} | Recall Score: {recall:.4f} | F1

      Accuracy Score: 0.6226 | Precision Score: 0.6595 | Recall Score: 0.6226 | F1 Score: 0.6397
```

```
## Enhancement 2: Increase the number of epochs (and maybe the learning
rate). Does the accuracy on the test set increase? Is there a
significant difference between the test accuracy and the train
accuracy? If yes, why?
```

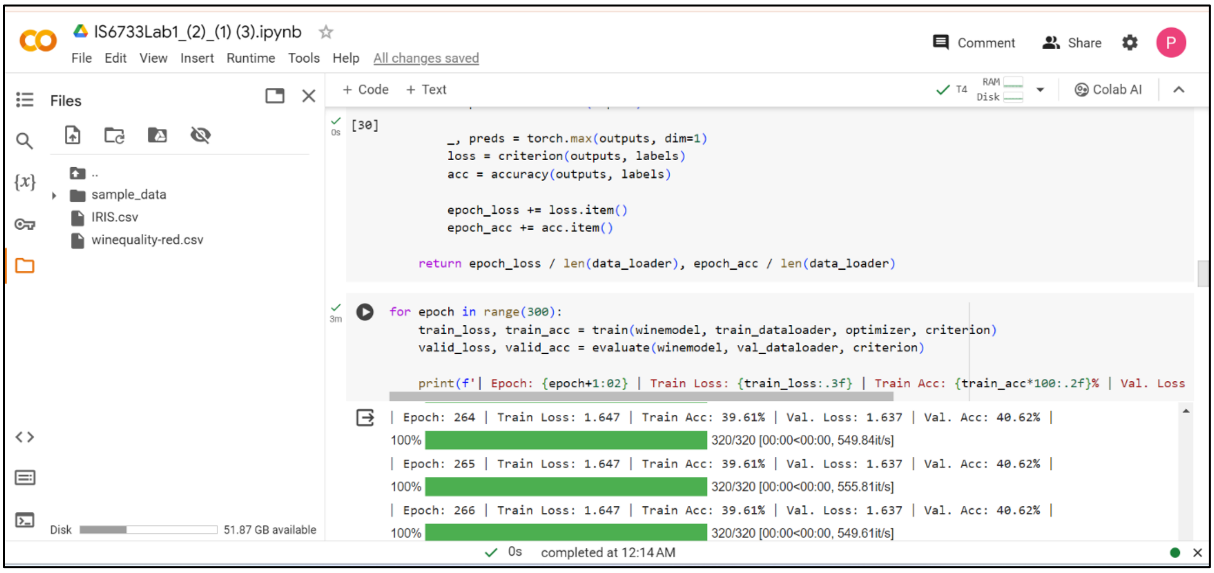
Partha Sai Madallapalli

```
Answer: Post increasing the number of epochs to 300 and learning rate
at 0.1 . No, the accuracy on the test set has decreased at 39.79% from
62.29%. Post changing the number of epochs and learning rate, their
isn't greater significance difference between the test and train
accuracy( train accuracy: 43.67% ,test accuracy: 39.79%)

The reason is as there isn’t a greater significance difference between
the test and train accuracy because there would be increase in epochs
to 300.
```



```
#Training with lr=0.1 and epoch to 300:

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss().to(device)
optimizer = AdamW(winemodel.parameters(), lr = 1e-1)
```

```
# Lets define the training steps
```

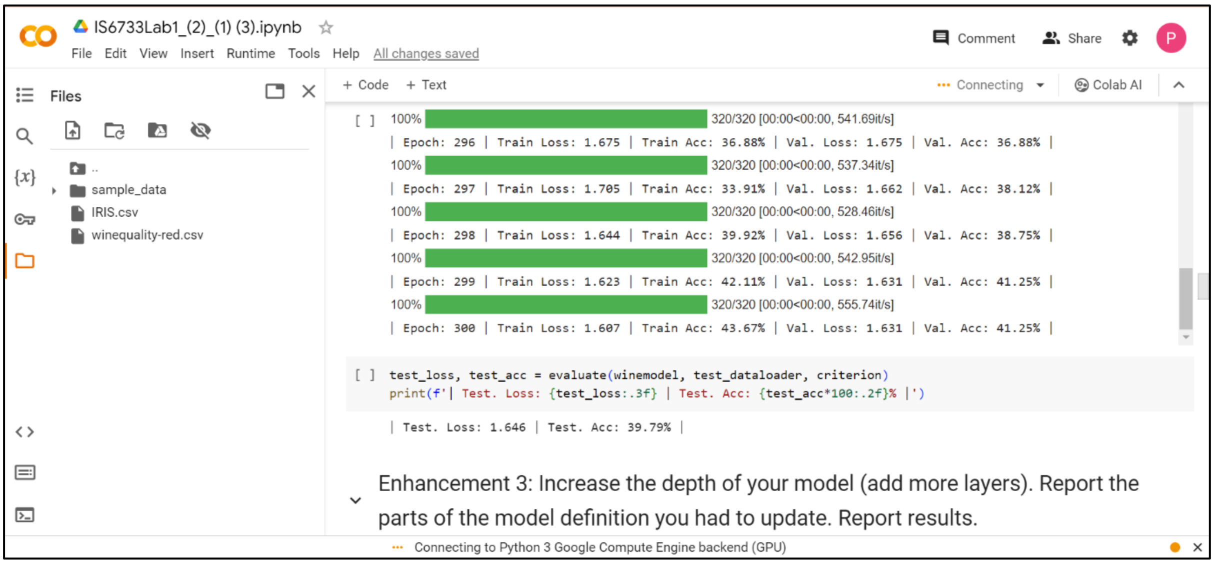


```
[30]
            _, preds = torch.max(outputs, dim=1)
            loss = criterion(outputs, labels)
            acc = accuracy(outputs, labels)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

        return epoch_loss / len(data_loader), epoch_acc / len(data_loader)
```

```
for epoch in range(300):
    train_loss, train_acc = train(winemodel, train_dataloader, optimizer, criterion)
    valid_loss, valid_acc = evaluate(winemodel, val_dataloader, criterion)

    print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss
```

```
| Epoch: 264 | Train Loss: 1.647 | Train Acc: 39.61% | Val. Loss: 1.637 | Val. Acc: 40.62% |
100% 320/320 [00:00<00:00, 549.84it/s]
| Epoch: 265 | Train Loss: 1.647 | Train Acc: 39.61% | Val. Loss: 1.637 | Val. Acc: 40.62% |
100% 320/320 [00:00<00:00, 555.81it/s]
| Epoch: 266 | Train Loss: 1.647 | Train Acc: 39.61% | Val. Loss: 1.637 | Val. Acc: 40.62% |
100% 320/320 [00:00<00:00, 549.61it/s]
```

0s completed at 12:14 AM


IS6733Lab1_(2)_(1) (3).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connecting
Colab AI
Files
sample_data
IRIS.csv
winequality-red.csv
Connecting to Python 3 Google Compute Engine backend (GPU)

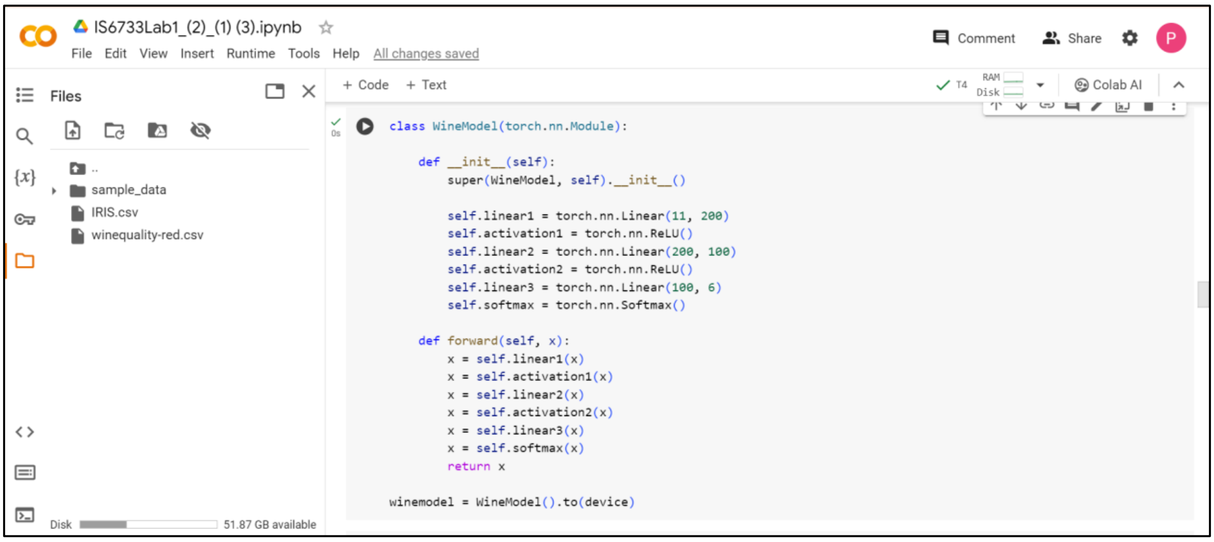
```
[ ] 100% 320/320 [00:00<00:00, 541.69it/s]
    | Epoch: 296 | Train Loss: 1.675 | Train Acc: 36.88% | Val. Loss: 1.675 | Val. Acc: 36.88% |
    100% 320/320 [00:00<00:00, 537.34it/s]
    | Epoch: 297 | Train Loss: 1.705 | Train Acc: 33.91% | Val. Loss: 1.662 | Val. Acc: 38.12% |
    100% 320/320 [00:00<00:00, 528.46it/s]
    | Epoch: 298 | Train Loss: 1.644 | Train Acc: 39.92% | Val. Loss: 1.656 | Val. Acc: 38.75% |
    100% 320/320 [00:00<00:00, 542.95it/s]
    | Epoch: 299 | Train Loss: 1.623 | Train Acc: 42.11% | Val. Loss: 1.631 | Val. Acc: 41.25% |
    100% 320/320 [00:00<00:00, 555.74it/s]
    | Epoch: 300 | Train Loss: 1.607 | Train Acc: 43.67% | Val. Loss: 1.631 | Val. Acc: 41.25% |
```

```
[ ] test_loss, test_acc = evaluate(winemodel, test_dataloader, criterion)
    print(f'| Test. Loss: {test_loss:.3f} | Test. Acc: {test_acc*100:.2f}% |')
```

```
| Test. Loss: 1.646 | Test. Acc: 39.79% |
```

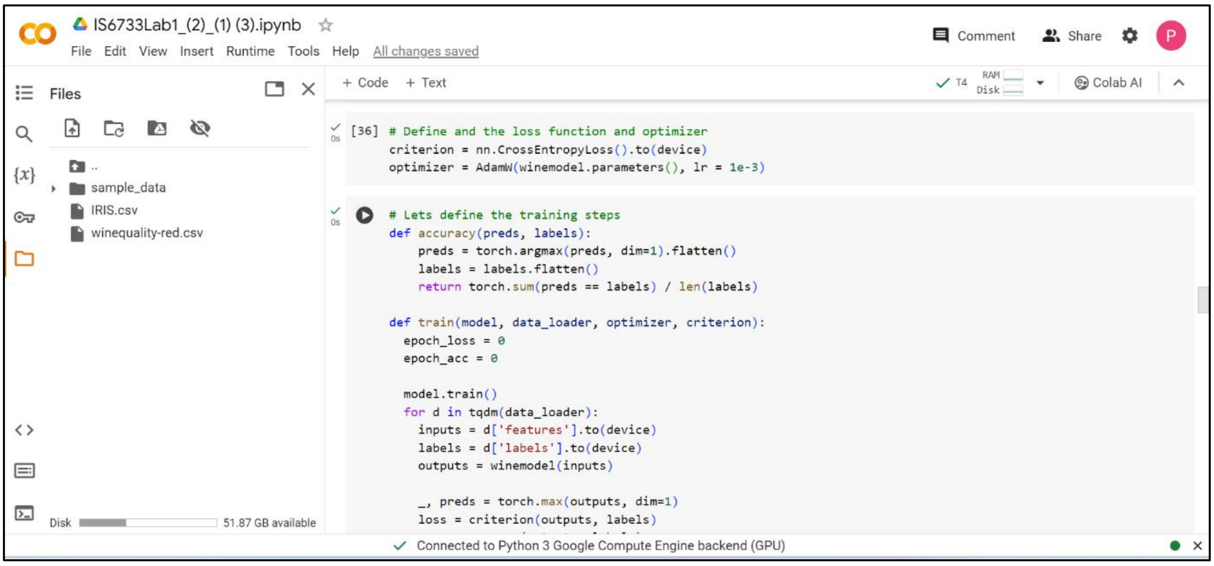
Enhancement 3: Increase the depth of your model (add more layers). Report the parts of the model definition you had to update. Report results.

Enhancement 3: Increase the depth of your model (add more layers). Report the parts of the model definition you had to update. Report results.



```
class WineModel(torch.nn.Module):

    def __init__(self):
        super(WineModel, self).__init__()

        self.linear1 = torch.nn.Linear(11, 200)
        self.activation1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 100)
        self.activation2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(100, 6)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation1(x)
        x = self.linear2(x)
        x = self.activation2(x)
        x = self.linear3(x)
        x = self.softmax(x)
        return x

winemodel = WineModel().to(device)
```

Partha Sai Madallapalli

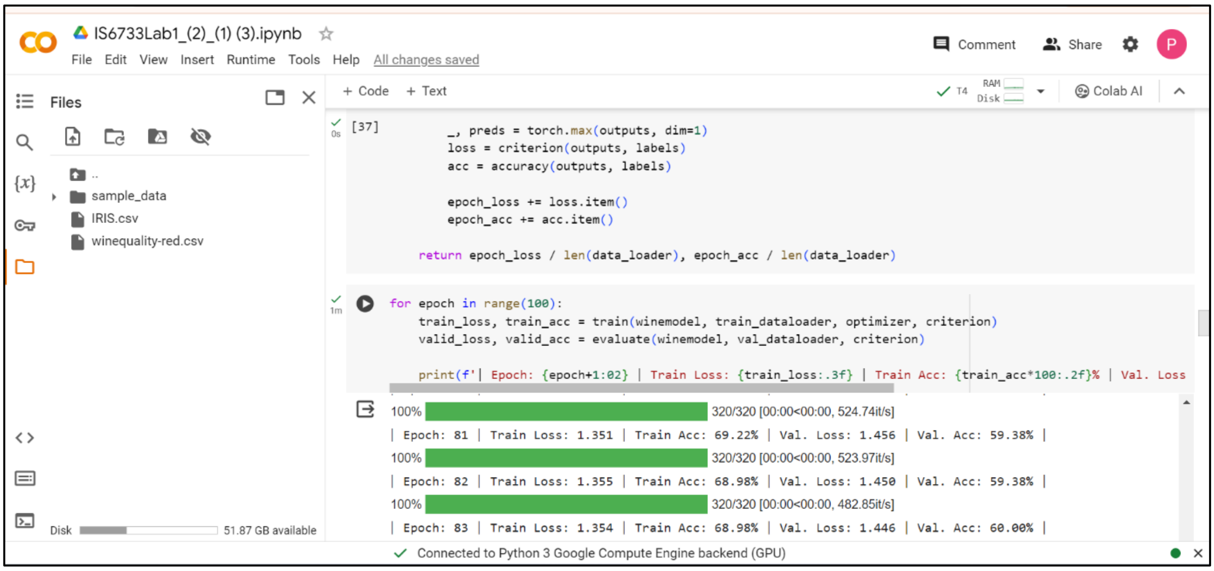


```
[36] # Define and the loss function and optimizer
     criterion = nn.CrossEntropyLoss().to(device)
     optimizer = AdamW(winemodel.parameters(), lr = 1e-3)
```

```
# Lets define the training steps
def accuracy(preds, labels):
    preds = torch.argmax(preds, dim=1).flatten()
    labels = labels.flatten()
    return torch.sum(preds == labels) / len(labels)

def train(model, data_loader, optimizer, criterion):
  epoch_loss = 0
  epoch_acc = 0

  model.train()
  for d in tqdm(data_loader):
    inputs = d['features'].to(device)
    labels = d['labels'].to(device)
    outputs = winemodel(inputs)

    _, preds = torch.max(outputs, dim=1)
    loss = criterion(outputs, labels)
```

Connected to Python 3 Google Compute Engine backend (GPU)

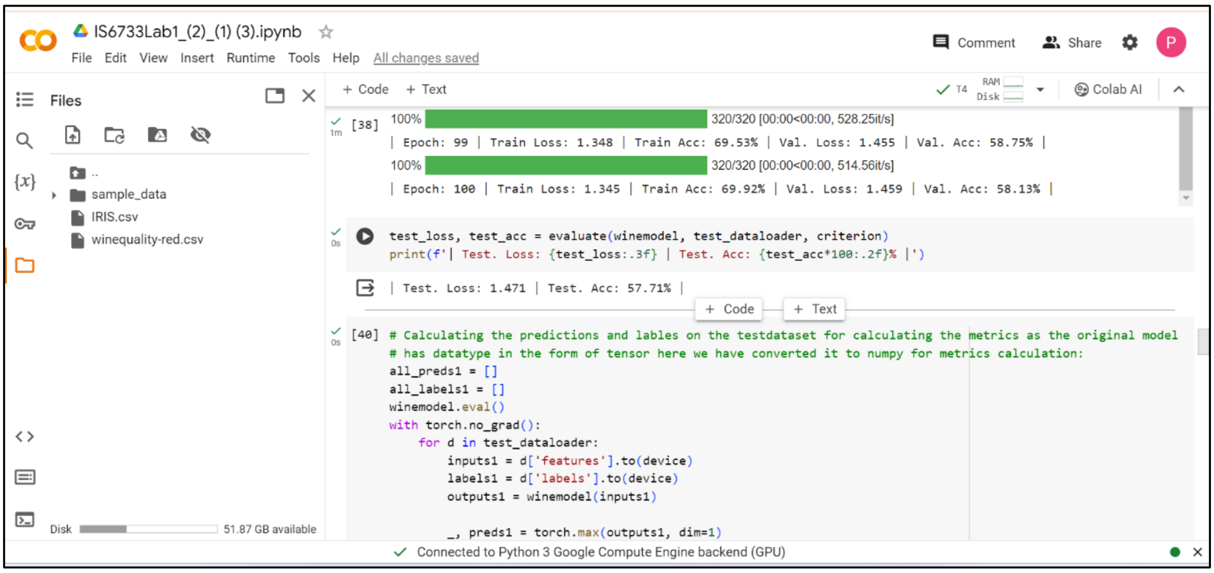
Partha Sai Madallapalli


IS6733Lab1_(2)_(1) (3).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
IRIS.csv
winequality-red.csv
Disk 51.87 GB available
+ Code + Text
T4 RAM Disk
Colab AI
Connected to Python 3 Google Compute Engine backend (GPU)

```
[37]            _, preds = torch.max(outputs, dim=1)
                loss = criterion(outputs, labels)
                acc = accuracy(outputs, labels)

                epoch_loss += loss.item()
                epoch_acc += acc.item()

        return epoch_loss / len(data_loader), epoch_acc / len(data_loader)
```

```
for epoch in range(100):
    train_loss, train_acc = train(winemodel, train_dataloader, optimizer, criterion)
    valid_loss, valid_acc = evaluate(winemodel, val_dataloader, criterion)

    print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss
```

```
100% 320/320 [00:00<00:00, 524.74it/s]
| Epoch: 81 | Train Loss: 1.351 | Train Acc: 69.22% | Val. Loss: 1.456 | Val. Acc: 59.38% |
100% 320/320 [00:00<00:00, 523.97it/s]
| Epoch: 82 | Train Loss: 1.355 | Train Acc: 68.98% | Val. Loss: 1.450 | Val. Acc: 59.38% |
100% 320/320 [00:00<00:00, 482.85it/s]
| Epoch: 83 | Train Loss: 1.354 | Train Acc: 68.98% | Val. Loss: 1.446 | Val. Acc: 60.00% |
```

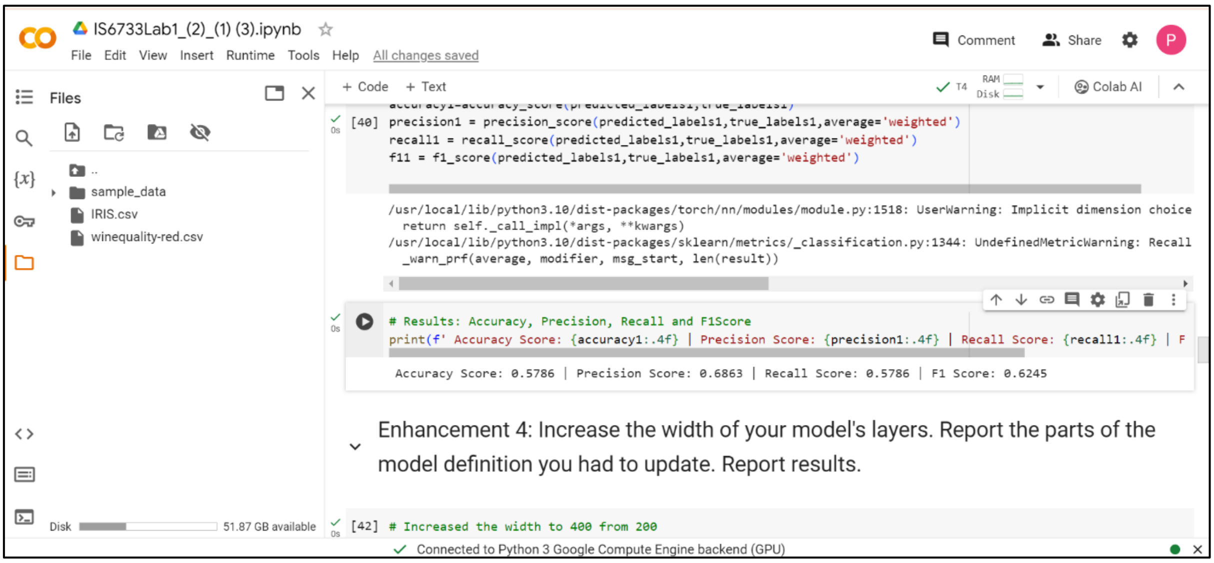

IS6733Lab1_(2)_(1) (3).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
IRIS.csv
winequality-red.csv
Disk 51.87 GB available
+ Code + Text
T4 RAM Disk
Colab AI
Connected to Python 3 Google Compute Engine backend (GPU)

```
[38] 100% 320/320 [00:00<00:00, 528.25it/s]
     | Epoch: 99 | Train Loss: 1.348 | Train Acc: 69.53% | Val. Loss: 1.455 | Val. Acc: 58.75% |
     100% 320/320 [00:00<00:00, 514.56it/s]
     | Epoch: 100 | Train Loss: 1.345 | Train Acc: 69.92% | Val. Loss: 1.459 | Val. Acc: 58.13% |
```

```
test_loss, test_acc = evaluate(winemodel, test_dataloader, criterion)
print(f'| Test. Loss: {test_loss:.3f} | Test. Acc: {test_acc*100:.2f}% |')
```

```
| Test. Loss: 1.471 | Test. Acc: 57.71% |
```

```
[40] # Calculating the predictions and lables on the testdataset for calculating the metrics as the original model
     # has datatype in the form of tensor here we have converted it to numpy for metrics calculation:
     all_preds1 = []
     all_labels1 = []
     winemodel.eval()
     with torch.no_grad():
         for d in test_dataloader:
             inputs1 = d['features'].to(device)
             labels1 = d['labels'].to(device)
             outputs1 = winemodel(inputs1)

             _, preds1 = torch.max(outputs1, dim=1)
```


IS6733Lab1_(2)_(1) (3).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
sample_data
IRIS.csv
winequality-red.csv
Disk 51.87 GB available
+ Code + Text
T4 RAM Disk
Colab AI
Connected to Python 3 Google Compute Engine backend (GPU)

```
     accuracy1=accuracy_score(predicted_labels1,true_labels1)
[40] precision1 = precision_score(predicted_labels1,true_labels1,average='weighted')
     recall1 = recall_score(predicted_labels1,true_labels1,average='weighted')
     f11 = f1_score(predicted_labels1,true_labels1,average='weighted')
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py:1518: UserWarning: Implicit dimension choice
  return self._call_impl(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Results: Accuracy, Precision, Recall and F1Score
print(f' Accuracy Score: {accuracy1:.4f} | Precision Score: {precision1:.4f} | Recall Score: {recall1:.4f} | F
```

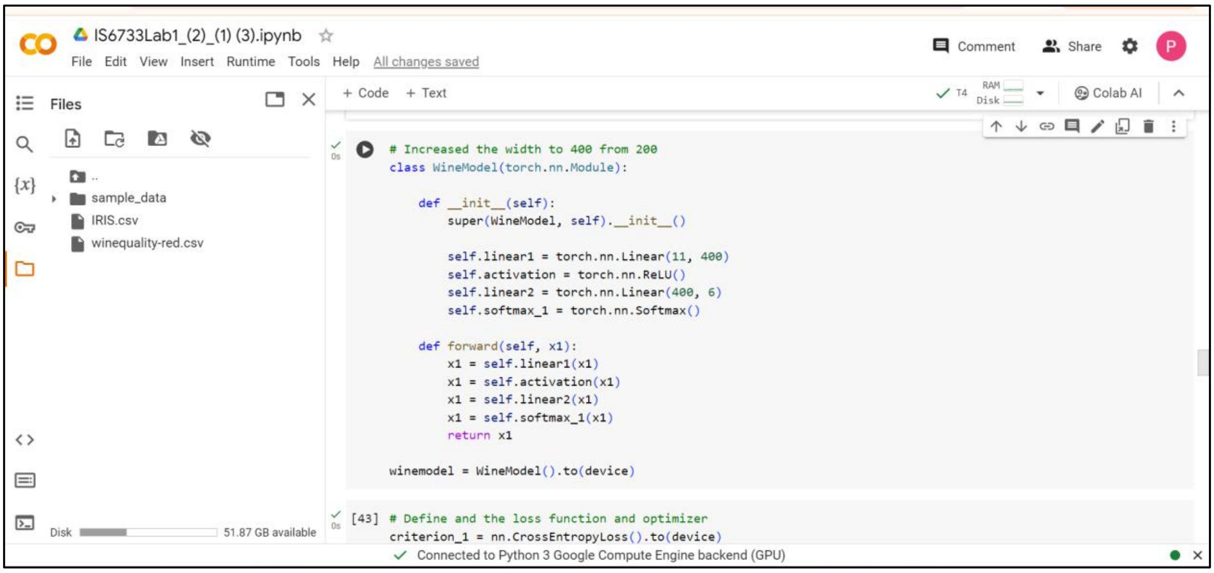
```
 Accuracy Score: 0.5786 | Precision Score: 0.6863 | Recall Score: 0.5786 | F1 Score: 0.6245
```

Enhancement 4: Increase the width of your model's layers. Report the parts of the model definition you had to update. Report results.

```
[42] # Increased the width to 400 from 200
```

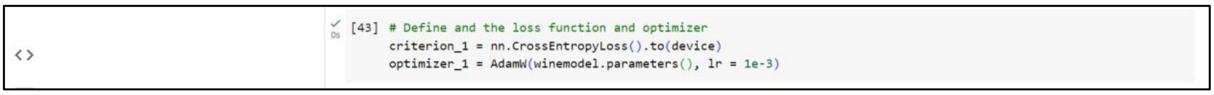
Partha Sai Madallapalli

Enhancement 4: Increase the width of your model's layers. Report the parts of the model definition you had to update. Report results.

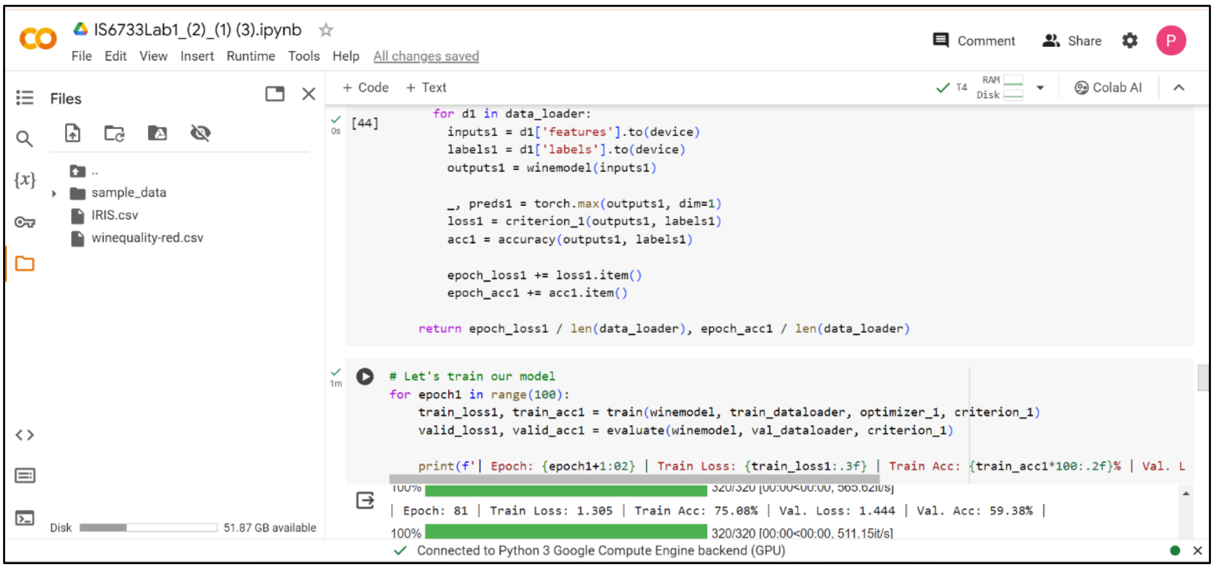

IS6733Lab1_(2)_(1) (3).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
T4 RAM Disk
Colab AI
Files
sample_data
IRIS.csv
winequality-red.csv
Disk 51.87 GB available
Connected to Python 3 Google Compute Engine backend (GPU)

```
# Increased the width to 400 from 200
class WineModel(torch.nn.Module):

    def __init__(self):
        super(WineModel, self).__init__()

        self.linear1 = torch.nn.Linear(11, 400)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(400, 6)
        self.softmax_1 = torch.nn.Softmax()

    def forward(self, x1):
        x1 = self.linear1(x1)
        x1 = self.activation(x1)
        x1 = self.linear2(x1)
        x1 = self.softmax_1(x1)
        return x1

winemodel = WineModel().to(device)
```

```
[43] # Define and the loss function and optimizer
criterion_1 = nn.CrossEntropyLoss().to(device)
```



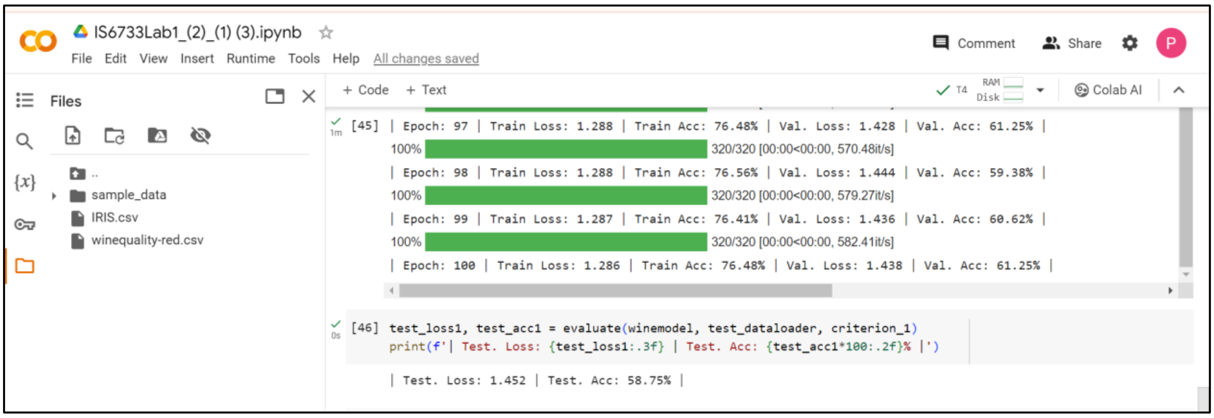
```
[43] # Define and the loss function and optimizer
criterion_1 = nn.CrossEntropyLoss().to(device)
optimizer_1 = AdamW(winemodel.parameters(), lr = 1e-3)
```


IS6733Lab1_(2)_(1) (3).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
T4 RAM Disk
Colab AI
Files
sample_data
IRIS.csv
winequality-red.csv
Disk 51.87 GB available
Connected to Python 3 Google Compute Engine backend (GPU)

```
[44]    for d1 in data_loader:
            inputs1 = d1['features'].to(device)
            labels1 = d1['labels'].to(device)
            outputs1 = winemodel(inputs1)

            _, preds1 = torch.max(outputs1, dim=1)
            loss1 = criterion_1(outputs1, labels1)
            acc1 = accuracy(outputs1, labels1)

            epoch_loss1 += loss1.item()
            epoch_acc1 += acc1.item()

        return epoch_loss1 / len(data_loader), epoch_acc1 / len(data_loader)
```

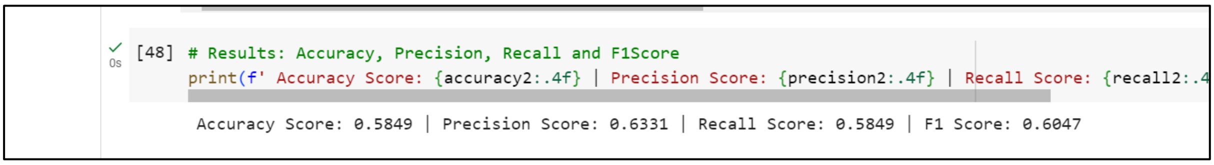
```
# Let's train our model
for epoch1 in range(100):
    train_loss1, train_acc1 = train(winemodel, train_dataloader, optimizer_1, criterion_1)
    valid_loss1, valid_acc1 = evaluate(winemodel, val_dataloader, criterion_1)

    print(f'| Epoch: {epoch1+1:02} | Train Loss: {train_loss1:.3f} | Train Acc: {train_acc1*100:.2f}% | Val. L
```

```
100% 320/320 [00:00<00:00, 565.62it/s]
| Epoch: 81 | Train Loss: 1.305 | Train Acc: 75.08% | Val. Loss: 1.444 | Val. Acc: 59.38% |
100% 320/320 [00:00<00:00, 511.15it/s]
```



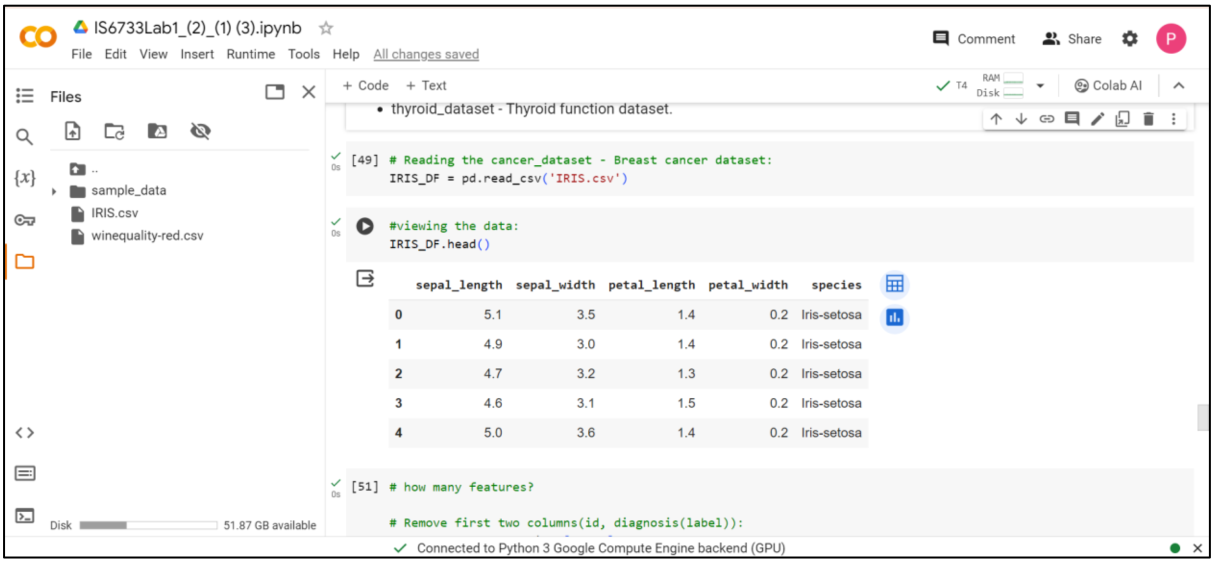
```
[45] | Epoch: 97 | Train Loss: 1.288 | Train Acc: 76.48% | Val. Loss: 1.428 | Val. Acc: 61.25% |
     100% 320/320 [00:00<00:00, 570.48it/s]
     | Epoch: 98 | Train Loss: 1.288 | Train Acc: 76.56% | Val. Loss: 1.444 | Val. Acc: 59.38% |
     100% 320/320 [00:00<00:00, 579.27it/s]
     | Epoch: 99 | Train Loss: 1.287 | Train Acc: 76.41% | Val. Loss: 1.436 | Val. Acc: 60.62% |
     100% 320/320 [00:00<00:00, 582.41it/s]
     | Epoch: 100 | Train Loss: 1.286 | Train Acc: 76.48% | Val. Loss: 1.438 | Val. Acc: 61.25% |
```

```
[46] test_loss1, test_acc1 = evaluate(winemodel, test_dataloader, criterion_1)
     print(f'| Test. Loss: {test_loss1:.3f} | Test. Acc: {test_acc1*100:.2f}% |')

     | Test. Loss: 1.452 | Test. Acc: 58.75% |
```



```
[48] # Results: Accuracy, Precision, Recall and F1Score
     print(f' Accuracy Score: {accuracy2:.4f} | Precision Score: {precision2:.4f} | Recall Score: {recall2:.4

      Accuracy Score: 0.5849 | Precision Score: 0.6331 | Recall Score: 0.5849 | F1 Score: 0.6047
```

Enhancement 5: Choose a new dataset from the list below. Search the Internet and download your chosen dataset (many of them could be available on kaggle). Adapt your model to your dataset. Train your model and record your results.



- thyroid_dataset - Thyroid function dataset.

```
[49] # Reading the cancer_dataset - Breast cancer dataset:
     IRIS_DF = pd.read_csv('IRIS.csv')
```

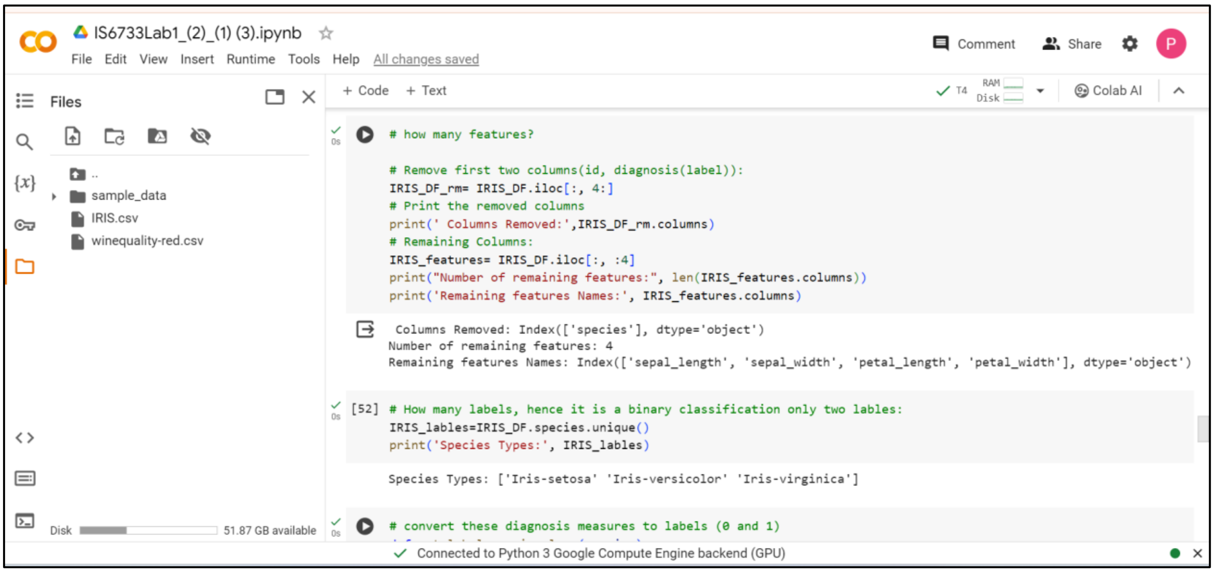
```
#viewing the data:
IRIS_DF.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
[51] # how many features?

     # Remove first two columns(id, diagnosis(label)):
```

Connected to Python 3 Google Compute Engine backend (GPU)

Partha Sai Madallapalli



```
# how many features?

# Remove first two columns(id, diagnosis(label)):
IRIS_DF_rm= IRIS_DF.iloc[:, 4:]
# Print the removed columns
print(' Columns Removed:',IRIS_DF_rm.columns)
# Remaining Columns:
IRIS_features= IRIS_DF.iloc[:, :4]
print("Number of remaining features:", len(IRIS_features.columns))
print('Remaining features Names:', IRIS_features.columns)
```

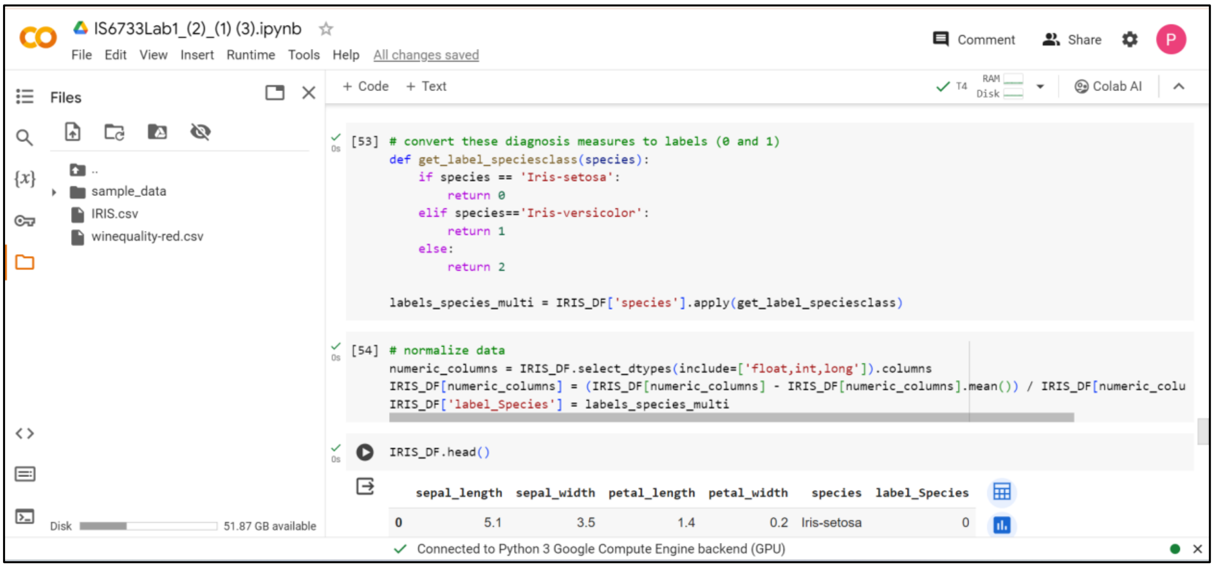
```
 Columns Removed: Index(['species'], dtype='object')
Number of remaining features: 4
Remaining features Names: Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width'], dtype='object')
```

```
[52] # How many labels, hence it is a binary classification only two lables:
     IRIS_lables=IRIS_DF.species.unique()
     print('Species Types:', IRIS_lables)
```

```
Species Types: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

```
# convert these diagnosis measures to labels (0 and 1)
```

Connected to Python 3 Google Compute Engine backend (GPU)



```
[53] # convert these diagnosis measures to labels (0 and 1)
     def get_label_speciesclass(species):
         if species == 'Iris-setosa':
             return 0
         elif species=='Iris-versicolor':
             return 1
         else:
             return 2

     labels_species_multi = IRIS_DF['species'].apply(get_label_speciesclass)
```

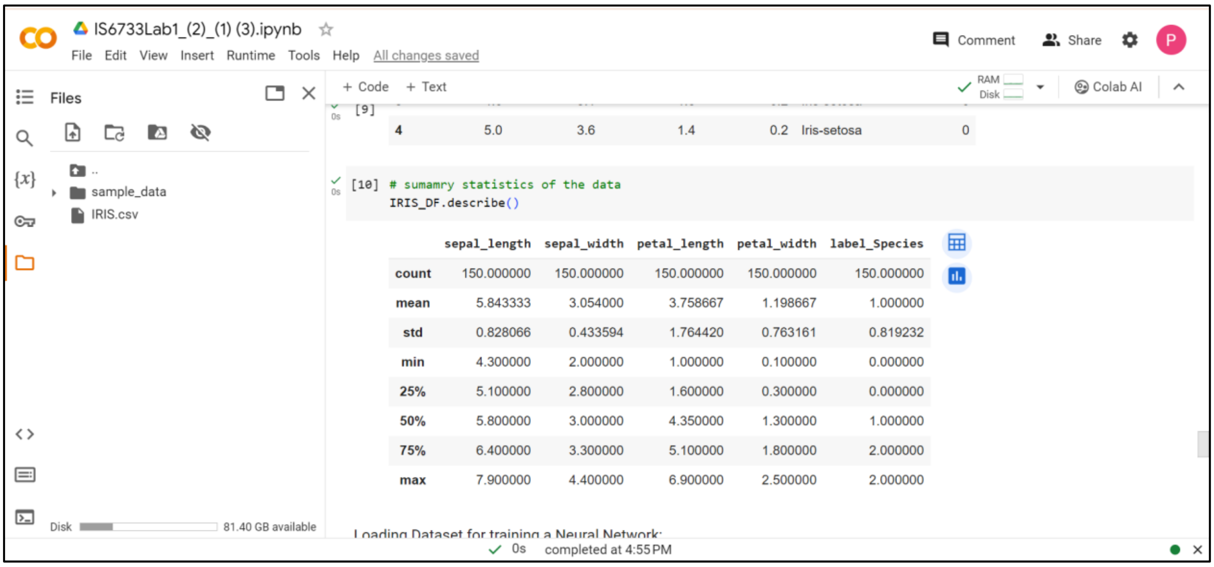
```
[54] # normalize data
     numeric_columns = IRIS_DF.select_dtypes(include=['float,int,long']).columns
     IRIS_DF[numeric_columns] = (IRIS_DF[numeric_columns] - IRIS_DF[numeric_columns].mean()) / IRIS_DF[numeric_colu
     IRIS_DF['label_Species'] = labels_species_multi
```

```
IRIS_DF.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	label_Species
0	5.1	3.5	1.4	0.2	Iris-setosa	0

Connected to Python 3 Google Compute Engine backend (GPU)

Partha Sai Madallapalli



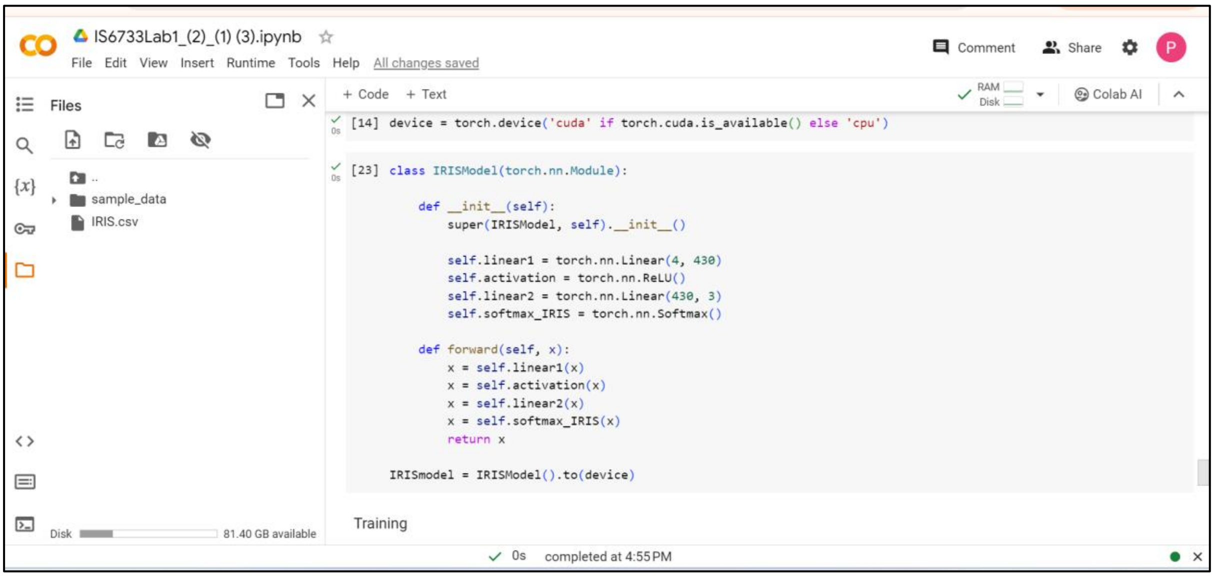
```
[9]
4    5.0    3.6    1.4    0.2  Iris-setosa    0
```

```
[10] # sumamry statistics of the data
     IRIS_DF.describe()
```

	sepal_length	sepal_width	petal_length	petal_width	label_Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667	1.000000
std	0.828066	0.433594	1.764420	0.763161	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Loading Dataset for training a Neural Network:

✓ 0s completed at 4:55 PM



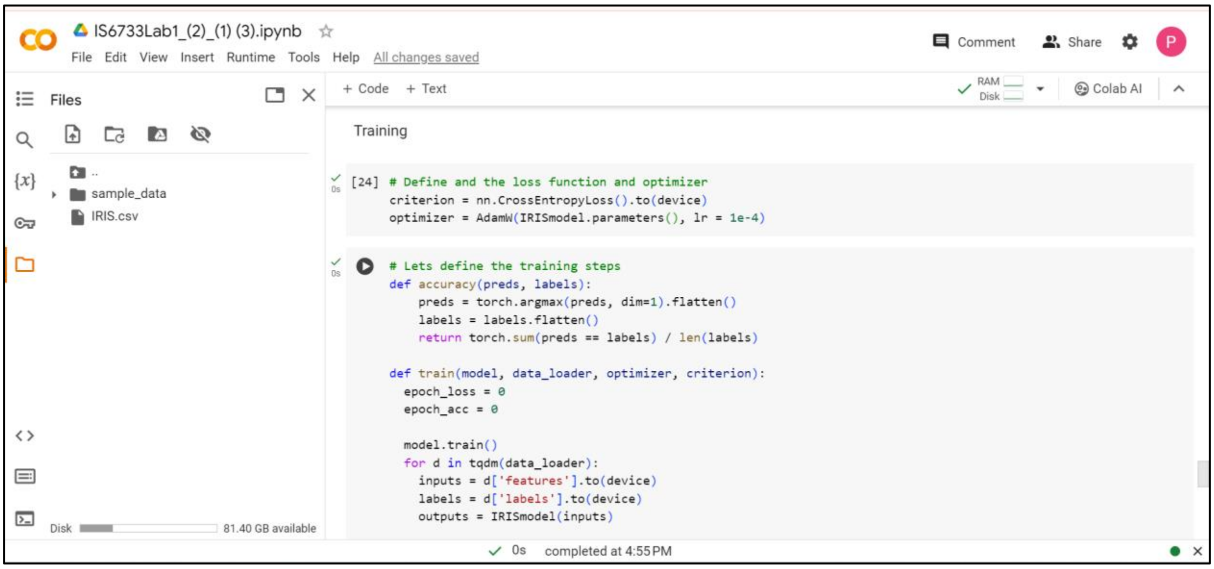
```
[14] device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
[23] class IRISModel(torch.nn.Module):

         def __init__(self):
             super(IRISModel, self).__init__()

             self.linear1 = torch.nn.Linear(4, 430)
             self.activation = torch.nn.ReLU()
             self.linear2 = torch.nn.Linear(430, 3)
             self.softmax_IRIS = torch.nn.Softmax()

         def forward(self, x):
             x = self.linear1(x)
             x = self.activation(x)
             x = self.linear2(x)
             x = self.softmax_IRIS(x)
             return x

     IRISmodel = IRISModel().to(device)
```

Training

✓ 0s completed at 4:55 PM

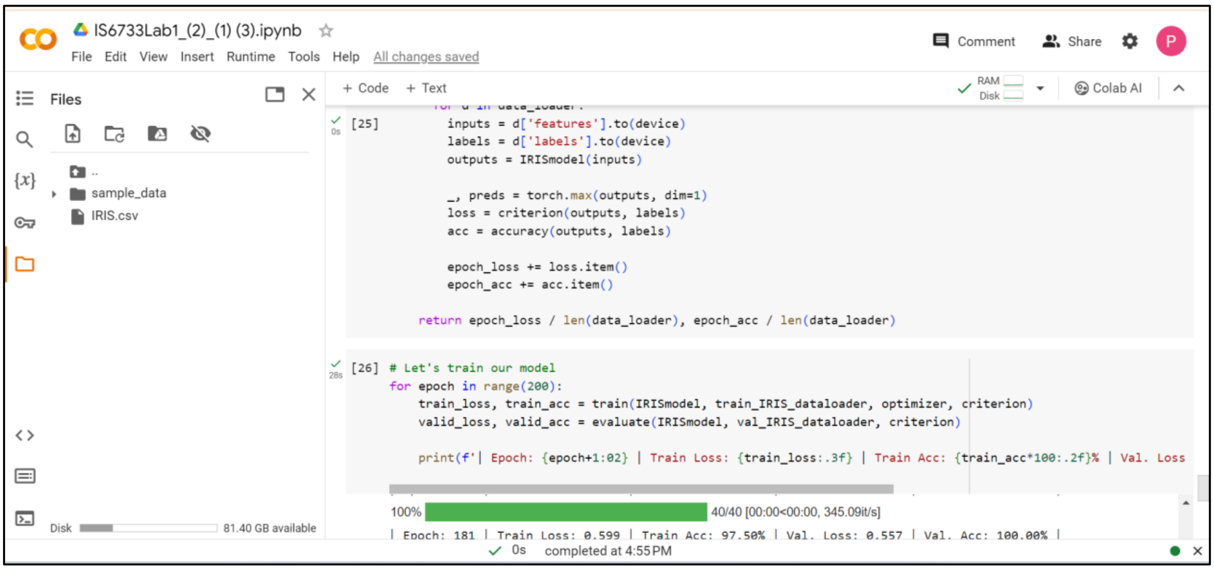
Partha Sai Madallapalli



Training

```
[24] # Define and the loss function and optimizer
     criterion = nn.CrossEntropyLoss().to(device)
     optimizer = AdamW(IRISmodel.parameters(), lr = 1e-4)
```

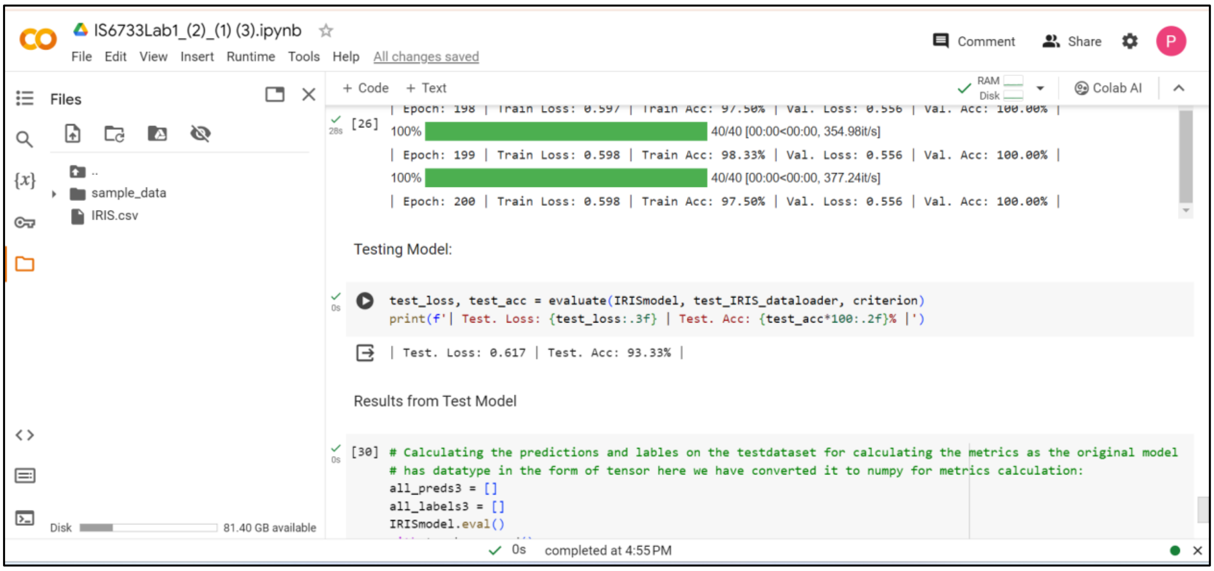
```
# Lets define the training steps
def accuracy(preds, labels):
    preds = torch.argmax(preds, dim=1).flatten()
    labels = labels.flatten()
    return torch.sum(preds == labels) / len(labels)

def train(model, data_loader, optimizer, criterion):
  epoch_loss = 0
  epoch_acc = 0

  model.train()
  for d in tqdm(data_loader):
    inputs = d['features'].to(device)
    labels = d['labels'].to(device)
    outputs = IRISmodel(inputs)
```



```
[25]       inputs = d['features'].to(device)
           labels = d['labels'].to(device)
           outputs = IRISmodel(inputs)

           _, preds = torch.max(outputs, dim=1)
           loss = criterion(outputs, labels)
           acc = accuracy(outputs, labels)

           epoch_loss += loss.item()
           epoch_acc += acc.item()

       return epoch_loss / len(data_loader), epoch_acc / len(data_loader)
```

```
[26] # Let's train our model
     for epoch in range(200):
         train_loss, train_acc = train(IRISmodel, train_IRIS_dataloader, optimizer, criterion)
         valid_loss, valid_acc = evaluate(IRISmodel, val_IRIS_dataloader, criterion)

         print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss
```

```
100% 40/40 [00:00<00:00, 345.09it/s]
| Epoch: 181 | Train Loss: 0.599 | Train Acc: 97.50% | Val. Loss: 0.557 | Val. Acc: 100.00% |
```



```
| Epoch: 198 | Train Loss: 0.597 | Train Acc: 97.50% | Val. Loss: 0.556 | Val. Acc: 100.00% |
100% 40/40 [00:00<00:00, 354.98it/s]
| Epoch: 199 | Train Loss: 0.598 | Train Acc: 98.33% | Val. Loss: 0.556 | Val. Acc: 100.00% |
100% 40/40 [00:00<00:00, 377.24it/s]
| Epoch: 200 | Train Loss: 0.598 | Train Acc: 97.50% | Val. Loss: 0.556 | Val. Acc: 100.00% |
```

Testing Model:

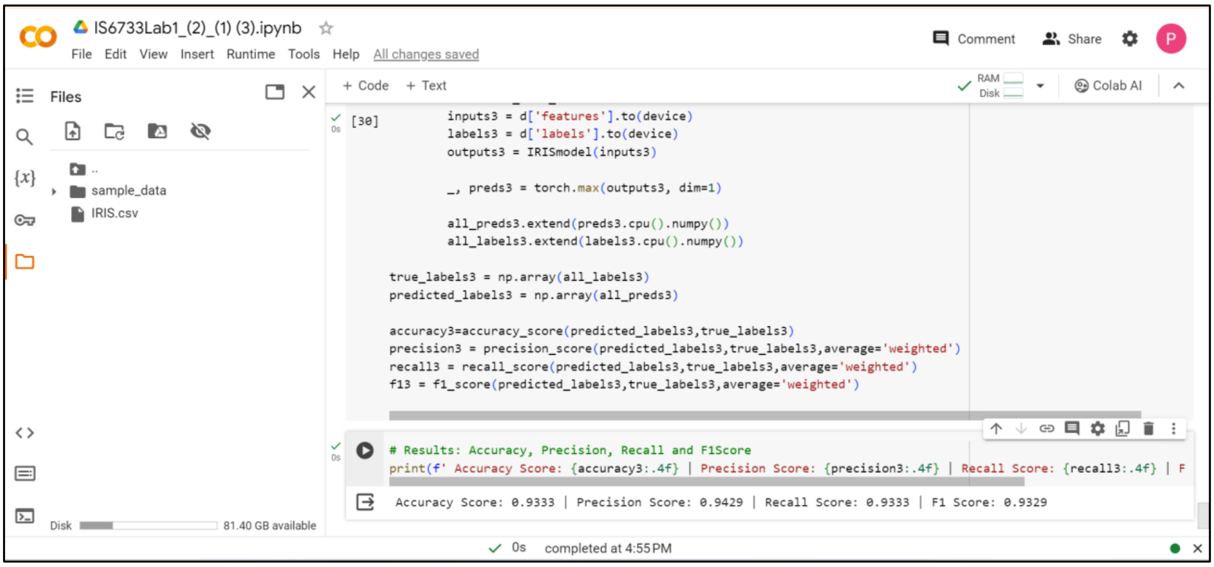
```
test_loss, test_acc = evaluate(IRISmodel, test_IRIS_dataloader, criterion)
print(f'| Test. Loss: {test_loss:.3f} | Test. Acc: {test_acc*100:.2f}% |')
```

```
| Test. Loss: 0.617 | Test. Acc: 93.33% |
```

Results from Test Model

```
[30] # Calculating the predictions and lables on the testdataset for calculating the metrics as the original model
     # has datatype in the form of tensor here we have converted it to numpy for metrics calculation:
     all_preds3 = []
     all_labels3 = []
     IRISmodel.eval()
```

Partha Sai Madallapalli



```
[30]        inputs3 = d['features'].to(device)
            labels3 = d['labels'].to(device)
            outputs3 = IRISmodel(inputs3)

            _, preds3 = torch.max(outputs3, dim=1)

            all_preds3.extend(preds3.cpu().numpy())
            all_labels3.extend(labels3.cpu().numpy())

    true_labels3 = np.array(all_labels3)
    predicted_labels3 = np.array(all_preds3)

    accuracy3=accuracy_score(predicted_labels3,true_labels3)
    precision3 = precision_score(predicted_labels3,true_labels3,average='weighted')
    recall3 = recall_score(predicted_labels3,true_labels3,average='weighted')
    f13 = f1_score(predicted_labels3,true_labels3,average='weighted')
```

```
# Results: Accuracy, Precision, Recall and F1Score
print(f' Accuracy Score: {accuracy3:.4f} | Precision Score: {precision3:.4f} | Recall Score: {recall3:.4f} | F
```

```
 Accuracy Score: 0.9333 | Precision Score: 0.9429 | Recall Score: 0.9333 | F1 Score: 0.9329
```