

Libraries

```
In [1]: import pandas as pd
import numpy as np
import re

In [2]: import spacy

In [3]: from sentence_transformers import SentenceTransformer, util

In [4]: import torch
from PIL import Image
import torchvision.transforms as transforms
from sklearn.preprocessing import LabelEncoder
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Preprocessing

```
In [5]: labels = pd.read_csv('./labels.csv')

Loading dataset

In [6]: labels

Out[6]:
```

Unnamed: 0	image_name	text_ocr	text_corrected	humour	sarcasm	offensive	motivational	overall_sentiment	
0	0	image_1.jpg	LOOK THERE MY FRIEND LIGHTYEAR NOW ALL SOHALIK...	LOOK THERE MY FRIEND LIGHTYEAR NOW ALL SOHALIK...	hilarious	general	not_offensive	not_motivational	very_positive
1	1	image_2.jpeg	The best of #10 Year Challenge Completed in le...	The best of #10 Year Challenge - Sweet Dee Edition	not_funny	general	not_offensive	motivational	very_positive
2	2	image_3.JPG	Sam Thorne @Strippin (Follow Follow Saw every...	Sam Thorne @Strippin (Follow Follow Saw every...	very_funny	not_sarcastic	not_offensive	not_motivational	positive
3	3	image_4.png	10 Year Challenge - Sweet Dee Edition	10 Year Challenge - Sweet Dee Edition	very_funny	twisted_meaning	very_offensive	motivational	positive
4	4	image_5.png	10 YEAR CHALLENGE WITH NO FILTER 47 Hilarious...	10 YEAR CHALLENGE WITH NO FILTER 47 Hilarious...	hilarious	very_twisted	very_offensive	not_motivational	neutral
...
6987	6987	image_6988.jpg	Tuesday is Mardi Gras Wednesday is Valentine's...	Tuesday is Mardi Gras Wednesday is Valentine's...	very_funny	twisted_meaning	very_offensive	motivational	neutral
6988	6988	image_6989.jpg	MUST WATCH MOVIES OF 2017 ITI Chennai memes MA...	MUST WATCH MOVIES OF 2017 ITI Chennai memes MA...	funny	twisted_meaning	not_offensive	not_motivational	neutral
6989	6989	image_6990.png	LESS MORE TALKING PLANNING SODA JUNK FOOD COMP...	LESS MORE TALKING PLANNING SODA JUNK FOOD COMP...	funny	general	slight	not_motivational	positive
6990	6990	image_6991.jpg	When I have time is a fantasy no one has time...	When I have time is a fantasy no one has time...	not_funny	twisted_meaning	not_offensive	motivational	very_positive
6991	6991	image_6992.jpg	The starting point for every good idea is "Wha...	The starting point for every good idea is "Wha...	not_funny	not_sarcastic	not_offensive	motivational	positive

6992 rows x 9 columns

Mapping the labels ('overall_sentiment'). 'very_positive' and 'positive' will be merged to 1 label, same goes for negative labels.

```
In [7]: print(labels['overall_sentiment'].unique())
labels['overall_sentiment'] = labels['overall_sentiment'].replace({'very_positive':'positive', 'very_negative':
labels['humour'] = labels['humour'].replace({'hilarious':1, 'funny':1, 'very_funny':1, 'not_funny':0})
['very_positive' 'positive' 'neutral' 'negative' 'very_negative']

In [8]: labels['offensive'] = labels['offensive'].replace({'very_offensive':1, 'slight':1, 'hateful_offensive':1, 'not_

In [9]: labels['motivational'] = labels['motivational'].replace({'motivational':1, 'not_motivational':0})

In [10]: labels['sarcasm'] = labels['sarcasm'].replace({'general':1, 'twisted_meaning':1, 'very_twisted':1, 'not_sarcasti

In [11]: labels['sarcasm'].unique()

Out[11]: array([1, 0], dtype=int64)
```

Function preprocessor is used to remove punctuation in the captions and convert the text to lower case.

```
In [12]: def preprocessor(text):
text = re.sub('[\W\s]',' ',text)
text = text.lower()
return text

Removing words that have null values (there were not many).
```

```
In [13]: labels = labels.dropna()
labels = labels.reset_index()

In [14]: for i in range(len(labels)):
labels['text_corrected'][i] = preprocessor(labels['text_corrected'][i])

<ipython-input-14-c51ce5353a3>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
labels['text_corrected'][i] = preprocessor(labels['text_corrected'][i])

Selecting the columns needed (images, captions, and label).
```

```
In [15]: data_used = labels[['image_name', 'text_corrected', 'humour', 'sarcasm', 'offensive', 'motivational', 'overall_se

In [16]: data_used

Out[16]:
```

	image_name	text_corrected	humour	sarcasm	offensive	motivational	overall_sentiment
0	image_1.jpg	look there my friend lightyear now all sohalik...	1	1	0	0	positive
1	image_2.jpeg	the best of 10 yearchallenge completed in less...	0	1	0	1	positive
2	image_3.JPG	sam thorne strippin follow follow saw everyon...	1	0	0	0	positive
3	image_4.png	10 year challenge sweet dee edition	1	1	1	1	positive
4	image_5.png	10 year challenge with no filter 47 hilarious ...	1	1	1	0	neutral
...
6825	image_6988.jpg	tuesday is mardi gras wednesday is valentines ...	1	1	1	1	neutral
6826	image_6989.jpg	must watch movies of 2017 iti chennai memes ma...	1	1	0	0	neutral
6827	image_6990.png	less more talking planning soda junk food comp...	1	1	1	0	positive
6828	image_6991.jpg	when i have time is a fantasy no one has time ...	0	1	0	1	positive
6829	image_6992.jpg	the starting point for every good idea is what...	0	0	0	1	positive

6830 rows x 7 columns

```
In [17]: le = LabelEncoder()
data_used['overall_sentiment'] = le.fit_transform(data_used['overall_sentiment'])

<ipython-input-17-648562a388ae>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
data_used['overall_sentiment'] = le.fit_transform(data_used['overall_sentiment'])

Checking the maximum length of captions for padding
```

```
In [18]: lens = []
for i in data_used['text_corrected']:
lens.append(len(i.split()))
max(lens)

Out[18]: 187
```

DataLoader

```
In [19]: import os
from skimage import io
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

Using transformer.
```

```
In [20]: sentence_model = SentenceTransformer('all-MiniLM-L6-v2')

Function for padding text.
```

```
In [21]: def pad_tensor(t):
t = torch.tensor(t)
padding = max(lens) - t.size()[0]
t = torch.nn.functional.pad(t, (0, padding))
return t

Load data using custom dataloader class. It returns image int form of tensor, transformed text, and transformed label.
```

```
In [22]: #reference: https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/Basics/custom_
class MemeSentimentDataset(Dataset):
def __init__(self, csv_file, root_dir, transform = None, freq_threshold = 5):
self.df = csv_file.copy()
self.img = self.df['image_name']
self.captions = self.df['text_corrected']
self.root_dir = root_dir
self.transform = transform

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

def __getitem__(self, index):
image = Image.open(os.path.join(self.root_dir, self.df.iloc[index, 0])).convert("RGB")
y_label1 = torch.tensor(int(self.df.iloc[index, 2]))
y_label2 = torch.tensor(int(self.df.iloc[index, 3]))
y_label3 = torch.tensor(int(self.df.iloc[index, 4]))
y_label4 = torch.tensor(int(self.df.iloc[index, 5]))
y_label5 = torch.tensor(int(self.df.iloc[index, 6]))
caption = self.captions[index]
if self.transform:
image = self.transform(image)

padded_text_tensor = pad_tensor(sentence_model.encode(caption, convert_to_tensor=True))

return (image, padded_text_tensor, y_label1, y_label2, y_label3, y_label4, y_label5)

Transforming the read data.
```

```
In [23]: mean = [0.4, 0.5, 0.5]
std = [0.4, 0.5, 0.5]

transform = transforms.Compose(
[transforms.Resize((100, 100)),
transforms.ToTensor(),
transforms.Normalize(torch.Tensor(mean), torch.Tensor(std))
])

In [24]: dataset = MemeSentimentDataset(csv_file = data_used, root_dir = './images', transform = transform)

Data split into train and test parts.
```

```
In [25]: train_set, test_set = torch.utils.data.random_split(dataset, [5000, 1830])
train_loader = DataLoader(dataset = train_set, batch_size = 32, shuffle=True)
test_loader = DataLoader(dataset = test_set, batch_size = 32, shuffle=True)

In [26]: # for data in train_loader:
# image, text, lab1, lab2, lab3, lab4, lab5 = data
# print(image.size())

In [ ]:

In [ ]:
```

```
In [27]: # for i, data in enumerate(train_loader):
# i, t, l = data
# print(t.shape)
```

Neural Networks

Neural Networks for images.

```
In [28]: class NN(nn.Module):
def __init__(self):
super(NN, self).__init__()
self.flatten = nn.Flatten(1)
self.linear_relu_stack = nn.Sequential(
nn.Linear(100*100*3, 256),
nn.ReLU(inplace=False),
nn.Linear(256, 64),
nn.ReLU(inplace=False),
nn.Linear(64, 32),
nn.ReLU(inplace=False),
nn.Linear(32, 3)
)

def forward(self, x):
x = self.flatten(x)
logits = self.linear_relu_stack(x)
return logits
```

Neural Network for text

```
In [29]: class NN_text(nn.Module):
def __init__(self):
super(NN_text, self).__init__()
self.flatten = nn.Flatten(1)
self.linear_relu_stack = nn.Sequential(
nn.Linear(187, 256),
nn.ReLU(inplace=False),
nn.Linear(256, 64),
nn.ReLU(inplace=False),
nn.Linear(64, 32),
nn.ReLU(inplace=False),
nn.Linear(32, 3)
)

def forward(self, x):
x = self.flatten(x)
logits = self.linear_relu_stack(x)
return logits
```

Combining both image and text Neural Networks.

```
In [30]: class Combined_model(nn.Module):
def __init__(self, modelA, modelB):
super(Combined_model, self).__init__()
self.modelA = modelA
self.modelB = modelB
self.classifier1 = nn.Linear(6, 128)
self.classifier2 = nn.Linear(128, 64)
self.classifier3 = nn.Linear(64, 32)
self.classifier4 = nn.Linear(32, 16)

self.out1 = nn.Linear(16, 2)
self.out2 = nn.Linear(16, 2)
self.out3 = nn.Linear(16, 2)
self.out4 = nn.Linear(16, 2)
self.out5 = nn.Linear(16, 3)

def forward(self, x1, x2):
x1 = self.modelA(x1)
x2 = self.modelB(x2)
x = torch.cat((x1, x2), dim=1)

x = self.classifier1(x)
x = self.classifier2(x)
x = self.classifier3(x)
x = self.classifier4(x)

out1 = self.out1(x)
out2 = self.out2(x)
out3 = self.out3(x)
out4 = self.out4(x)
out5 = self.out5(x)

return out1, out2, out3, out4, out5

In [31]: net = NN()
net_text = NN_text()

In [32]: combined_model = Combined_model(net, net_text)

In [33]: image = torch.rand(1, 3, 100, 100)
text = torch.rand(1, 187)

In [34]: out1, out2, out3, out4, out5 = combined_model(image, text)

In [ ]:

In [ ]:
```

Data training and testing. This also shows losses and accuracies after each epoch.

```
In [39]: #Reference: AI lab task
def train_loop(dataloader, model, loss_fn, optimizer):
size = len(dataloader.dataset)
test_loss = 0
correct1 = 0
correct2 = 0
correct3 = 0
correct4 = 0
correct5 = 0
for batch, (X, T, y1, y2, y3, y4, y5) in enumerate(dataloader):
# Compute prediction and loss
pred1, pred2, pred3, pred4, pred5 = model(X, T)

loss1 = loss_fn(pred1, y1)
loss1 += loss_fn(pred2, y2)
loss1 += loss_fn(pred3, y3)
loss1 += loss_fn(pred4, y4)
loss1 += loss_fn(pred5, y5)

# Backpropagation
with torch.autograd.set_detect_anomaly(True):
optimizer.zero_grad()
loss1.backward()
optimizer.step()

if batch % 10 == 0:
loss, current = loss1.item(), batch * len(X)
print(f"loss: {loss1:>7f} [{current:>5d}/{size:>5d}]"
```

```
In [40]: def train_loop(dataloader, model, loss_fn):
size = len(dataloader.dataset)
num_batches = len(dataloader)

test_loss = 0
correct1 = 0
correct2 = 0
correct3 = 0
correct4 = 0
correct5 = 0

with torch.no_grad():
for X, T, y1, y2, y3, y4, y5 in dataloader:
pred1, pred2, pred3, pred4, pred5 = model(X, T)
test_loss += loss_fn(pred1, y1).item()
test_loss += loss_fn(pred2, y2).item()
test_loss += loss_fn(pred3, y3).item()
test_loss += loss_fn(pred4, y4).item()
test_loss += loss_fn(pred5, y5).item()

correct1 += (pred1.argmax(1) == y1).type(torch.float).sum().item()
correct2 += (pred2.argmax(1) == y2).type(torch.float).sum().item()
correct3 += (pred3.argmax(1) == y3).type(torch.float).sum().item()
correct4 += (pred4.argmax(1) == y4).type(torch.float).sum().item()
correct5 += (pred5.argmax(1) == y5).type(torch.float).sum().item()

test_loss /= num_batches
correct1 /= size
correct2 /= size
correct3 /= size
correct4 /= size
correct5 /= size

print(f"\n Humour Accuracy: {(100*correct1)>0.1f}%")
print(f"\n Sarcasm Accuracy: {(100*correct2)>0.1f}%")
print(f"\n Offensive Accuracy: {(100*correct3)>0.1f}%")
print(f"\n Motivational Accuracy: {(100*correct4)>0.1f}%")
print(f"\n Overall Accuracy: {(100*correct5)>0.1f}%")
```

I used Adam optimizer and cross entropy loss.

```
In [41]: #model=Combined_model(M1,M2)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(combined_model.parameters(), lr=0.001)

In [42]: epochs = 1
for t in range(epochs):
print(f"Epoch {t+1}\n-----")
train_loop(train_loader, combined_model, loss_fn, optimizer)
train_loop(test_loader, combined_model, loss_fn)
print("Done!")

Epoch 1
-----
<ipython-input-21-03565fe0dd97>:2: UserWarning: To copy construct from a tensor, it is recommended to use source
Tensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(source
Tensor).
t = torch.tensor(t)

loss: 3.331372 [ 0/ 5000]
loss: 2.956993 [ 320/ 5000]
loss: 3.459740 [ 640/ 5000]
loss: 3.223244 [ 960/ 5000]
loss: 3.061769 [ 1280/ 5000]
loss: 3.354167 [ 1600/ 5000]
loss: 3.184566 [ 1920/ 5000]
loss: 3.313868 [ 2240/ 5000]
loss: 3.441355 [ 2560/ 5000]
loss: 3.434478 [ 2880/ 5000]
loss: 3.630612 [ 3200/ 5000]
loss: 3.178700 [ 3520/ 5000]
loss: 3.067171 [ 3840/ 5000]
loss: 3.031541 [ 4160/ 5000]
loss: 3.217144 [ 4480/ 5000]

Humour Accuracy: 76.0%
Sarcasm Accuracy: 78.1%
Offensive Accuracy: 59.8%
Motivational Accuracy: 66.6%
Overall sentiment Accuracy: 59.8%
Done!
```

Visualization

```
In [43]: from torchviz import make_dot

In [44]: import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'

In [45]: network = next(iter(train_loader))
graph=combined_model(network[0],network[1])
make_dot(graph)

<ipython-input-21-03565fe0dd97>:2: UserWarning: To copy construct from a tensor, it is recommended to use source
Tensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(source
Tensor).
t = torch.tensor(t)
```



Text reading

```
In [ ]:
```