model

January 10, 2022

1 CNN for positive, negative and neutral classification

QuaternairCombinedPN is the **RAVDESS**, **CREMA-D**, **TESS** and **SAVEE** dataset combined **RAVDESS** and **CREMA-D** has equal *male* and *female* sample **TESS** has only *female* samples **SAVEE** has only *male* samples ***

Dataloaders that are used **Epochs: 5** *Following results are with a unbalanced dataset* * **NormalCrema** - *Anger, Disgust, Fear, Happy, Neutral, & Sad* * Train Acc: 49.46% * Validate Acc: 47.98% * Precision: 49.46% * **BinairCrema** - *Positive & Negative* * Train Acc: 68.26% * Validate Acc: 72.92% * Precision: 72.92% * **MaleSplitCremaBinair** - *Male samples Positive, Negative & Neutral* * Train Acc: 59.16% * Validate Acc: 59.46% * Precision: 59.46% * **FemaleSplitCremaBinair** - *Female samples Positive, Negative & Neutral* * Train Acc: 62.01% * Validate Acc: 66.31% * Precision: 66.31% * **QuaternairCombinedPN** - *Positive, Negative & Neutral* * Train Acc: 74.51% * Validate Acc: 74.23% * Precision: 74.51% * **QuaternairCombinedMaleSplitPN** - *Male samples Positive, Negative & Neutral* * Train Acc: ?% * Validate Acc: ?% * Precision: ?% * **QuaternairCombinedFemaleSplitPN** - *Female samples Positive, Negative & Neutral* * Train Acc: ?% * Validate Acc: ?% * Precision: ?%

[1]:
```
import os
import numpy as np
import pandas as pd
import time
```

2 Fetch Dataset

[2]:
```
# %run /data/emo/notebooks/source/pipeline/generator.ipynb
# train_ds, valid_ds = Generator.generate(MaleSplitCremaBinair, Spectrogram).
 ↪to_dataset()

# train_loader = utils.DataLoader(train_ds, batch_size=batch_size, shuffle=True␣
 ↪, num_workers=4)
# valid_loader = utils.DataLoader(valid_ds, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=4)
```

```
# print("Using", len(train_ds), "files for training.")
# print("Using", len(valid_ds), "files for validation.")

# for i in range(4):
#     print(train_ds[i][0].shape)
#     plt.imshow(train_ds[i][0].permute(1, 2, 0))
#     plt.axis('off');
#     plt.show()
```

3 Create Model

[3]:
```
import torch
import torchmetrics
import torch.nn as nn
import torch.nn.functional as fn
# from torchsummary import summary
import pytorch_lightning as pl

class Convolutional(pl.LightningModule):
    def __init__(self, cnn_output:int):
        super().__init__()
        self.cnn_output = cnn_output
        self.configure_metrics()
        self.loss_func = nn.CrossEntropyLoss()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.pool2 = nn.MaxPool2d(4, 4)
        self.pool3 = nn.MaxPool2d(4, 4)
        self.fc1 = nn.Linear(64*3*7, 64)
        self.fc2 = nn.Linear(64, cnn_output)

    def forward(self, x):
        x = self.pool1(fn.relu(self.conv1(x)))
        x = self.pool2(fn.relu(self.conv2(x)))
        x = self.pool3(fn.relu(self.conv3(x)))
        x = x.view(-1, 64*3*7) # Flatten layer
        x = fn.relu(self.fc1(x))
        x = fn.log_softmax(self.fc2(x), dim=1)
        return x

    def configure_metrics(self):
        self.train_acc = torchmetrics.Accuracy()
        self.valid_acc = torchmetrics.Accuracy()
```

```
        self.valid_precision = torchmetrics.Precision(num_classes=self.
↪cnn_output)
        self.valid_recall = torchmetrics.Recall(num_classes=self.cnn_output)

    def configure_optimizers(self):
        optimizer = torch.optim.AdamW(self.parameters(), lr=0.001,␣
↪weight_decay=0.01)
#         optimizer = torch.optim.Adam(self.parameters(), lr=0.001)
        return optimizer

    def validation_step(self, val_batch, batch_idx):
        x, y = val_batch
        output = self(x)
        loss = self.loss_func(output, y)
        self.valid_precision(output, y)
        self.valid_recall(output, y)
        self.valid_acc(output, y)
        self.log("precision", self.valid_precision, on_step=False,␣
↪on_epoch=True, logger=True)
        self.log("recall", self.valid_recall, on_step=False, on_epoch=True,␣
↪logger=True)
        self.log('val_acc', self.valid_acc, on_step=False, on_epoch=True,␣
↪logger=True)
        self.log('val_loss', loss, on_step=False, on_epoch=True, logger=True)

    def training_step(self, train_batch, batch_idx):
        x, y = train_batch
        output = self(x)
        loss = self.loss_func(output, y)
        self.train_acc(output, y)
        self.log('train_acc', self.train_acc, on_step=False, on_epoch=True,␣
↪logger=True)
        self.log('train_loss', loss, on_step=False, on_epoch=True, logger=True)
        return loss

    def on_train_start(self):
        self.log("hp/metric_1", 0.001)
        self.log("hp/metric_2", 32)

# network = Convolutional()
# summary(network, (3, 128, 256))
```

```
using gpu 3
```

3.1 Callback function

Should only contain logging and visualization logic

[4]:
```python
from pytorch_lightning.callbacks import ModelCheckpoint

checkpoint_callback = ModelCheckpoint(monitor="val_loss")
```

[5]:
```python
from pytorch_lightning.callbacks import Callback

class ConvolutionalCallback(Callback):
#     def __init__(self):
#         self.conv_layers = []

#     def on_train_start(self, trainer, pl_module):
#         self.writer("hp/metric_1", 0.001)
#         self.writer("hp/metric_2", 32)

    def on_fit_start(self, trainer, pl_module):
        """Callback function that gets executed before the fit starts

        Parameters
        ----------
        trainer : pl.Trainer
            The trainer of the CNN module (pl_module)
        pl_module : pl.LightningModule
            The model we want to use to retrieve information
        """
        self.writer = pl_module.logger.experiment

        # Connect the hooks to the CNN
        pl_module.fc1.register_forward_hook(self.activation_hook)
        pl_module.fc2.register_forward_hook(self.activation_hook)

    def activation_hook(self, inst, inp, out):
        """Run activation hook

        Parameters
        ----------
        inst : torch.nn.Module
            The layer we want to attach the hook to.
        inp : torch.Tensor
            The input to the `forward` method.
        out : torch.Tensor
            The output of the `forward` method.
        """
        # Create histogram of layer weights
```

```
        self.writer.add_histogram(repr(inst), out)

        # Create a grid with filterd images
        img_grid = torchvision.utils.make_grid(inp[0])
        self.writer.add_image('Forward Input images', img_grid)

        # Create a grid with filterd images
        img_grid = torchvision.utils.make_grid(out)
        self.writer.add_image('Forward Output images', img_grid)
```

4 Train Model

[6]:
```
%run /data/emo/notebooks/source/pipeline/generator.ipynb

batch_size = 32
n_epochs = 5
```

[7]:
```
def execute_training(dataloader, cnn_output:int):
    print("Get dataset")
#    train_ds, valid_ds = Generator.generate(dataloader, LogSpectrogram).
 ↪to_dataset()
    train_ds, valid_ds = Generator.generate(dataloader, Spectrogram).
 ↪to_dataset()

    train_loader = utils.DataLoader(train_ds, batch_size=batch_size,␣
 ↪shuffle=True , num_workers=4)
    valid_loader = utils.DataLoader(valid_ds, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=4)

    print("Using", len(train_ds), "files for training.")
    print("Using", len(valid_ds), "files for validation.")

    # ssh -L 6006:127.0.0.1:6006 18082920@datascience.hhs.nl
    # tensorboard --logdir ./tb_logs
    logger = pl.loggers.TensorBoardLogger('tb_logs', name='convolutional_pn')

    trainer = pl.Trainer(
        max_epochs=n_epochs,
        weights_summary=None,
        logger=logger,
        callbacks=[ConvolutionalCallback(), checkpoint_callback]
    )
```

```
    print("Start start training")
    network = Convolutional(cnn_output)

    trainer.fit(network, train_loader, valid_loader)
```

4.1 NormalCrema

[8]:
```
# execute_training(NormalCrema, 8)
```

4.2 BinairCrema

[9]:
```
# execute_training(BinairCrema, 2)
```

4.3 MaleSplitCremaBinair

[10]:
```
# execute_training(MaleSplitCremaBinair, 3)
```

4.4 FemaleSplitCremaBinair

[11]:
```
# execute_training(FemaleSplitCremaBinair, 3)
```

4.5 QuaternairCombinedPN

[12]:
```
execute_training(QuaternairCombinedPN, 3)
```

```
Get dataset

/opt/jupyterhub/anaconda/lib/python3.9/site-
packages/matplotlib/axes/_axes.py:7723: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
/opt/jupyterhub/anaconda/lib/python3.9/site-
packages/matplotlib/axes/_axes.py:7723: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
GPU available: True, used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs

Using 8145 files for training.
Using 2037 files for validation.
Start start training
```

```
/opt/jupyterhub/anaconda/lib/python3.9/site-
packages/pytorch_lightning/trainer/trainer.py:1303: UserWarning: GPU available
but not used. Set the gpus flag in your trainer `Trainer(gpus=1)` or script
`--gpus=1`.
  rank_zero_warn(

Validation sanity check: 0it [00:00, ?it/s]

Training: -1it [00:00, ?it/s]

Validating: 0it [00:00, ?it/s]

Validating: 0it [00:00, ?it/s]

Validating: 0it [00:00, ?it/s]

IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)


Validating: 0it [00:00, ?it/s]
```

4.6 QuaternairCombined

[13]:
```
execute_training(QuaternairCombined, 4)
```

```
Get dataset

/opt/jupyterhub/anaconda/lib/python3.9/site-
packages/matplotlib/axes/_axes.py:7723: RuntimeWarning: divide by zero
encountered in log10
  Z = 10. * np.log10(spec)
GPU available: True, used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs

Using 7544 files for training.
Using 1886 files for validation.
Start start training

Validation sanity check: 0it [00:00, ?it/s]

Training: -1it [00:00, ?it/s]

Validating: 0it [00:00, ?it/s]
```

```
Validating: 0it [00:00, ?it/s]
Validating: 0it [00:00, ?it/s]
Validating: 0it [00:00, ?it/s]
```

4.7 MaleSplitRAVDESSBinair

[14]:
```
#execute_training(MaleBinairRavdess, 3)
```

4.8 FemaleSplitRAVDESSBinair

[15]:
```
#execute_training(FemaleBinairRavdess, 3)
```

[ ]:
```

```