This is a report of the assignment 04, of the course MA-INF 2218 Video Analytics SS21.
Group: Mayara Bonani - Ismail Wahdan

1. Single Stage TCN

The model single stage TCN is implemented in the file ‘’model.py’’ by the class TCN().
The model is described as follow:

```
TCN(
  (conv_1x1): Conv1d(2048, 64, kernel_size=(1,), stride=(1,))
  (layers): ModuleList(
    (0): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(1,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (1): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(2,), dilation=(2,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (2): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(3,), dilation=(3,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (3): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(4,), dilation=(4,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (4): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(5,), dilation=(5,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
```

```
    (5): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(6,), dilation=(6,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (6): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(7,), dilation=(7,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (7): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(8,), dilation=(8,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (8): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(9,), dilation=(9,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
    (9): DilatedResidualLayer(
      (block): Sequential(
        (0): Conv1d(64, 64, kernel_size=(3,), stride=(1,), padding=(10,), dilation=(10,))
        (1): ReLU(inplace=True)
        (2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
      )
    )
  )
  (last_conv): Conv1d(64, 48, kernel_size=(1,), stride=(1,))
)
```

2. Multi Stage TCN

The model multi stage TCN is implemented in the file ‘’model.py’’ by the class MultiStageTCN(). The model is described as follow:

- The building units of the model are the single stage TCNs from above, all with the same configurations of layer number, dilation, etc…

- However, after the first stage, the prediction of one stage is concatenated with the original features (resulting in a vector of size 2048 + 48) and passed as input to the next stage.
- The model consists of 4 stages. The prediction of the last stage is the prediction of the whole model.

3. Multi Stage TCN with Video Loss

The video loss was implemented in file ‘’main.py’’ by the functions:

- **get_target_vector(labels):** It transforms the label tensor, with shape labels [batch size, number of frames] to a target vector, with a dimension equals the number of classes in the dataset, [batch size, number of classes].The i-th element of this vector is 1 if the i-th class is present in the video, otherwise it should be 0.

- **get_video_level_prediction(out):** It transforms the output of the model tensor, with shape [batch size, number of classes, number of frames]. To get the video level prediction apply a max pooling on the temporal dimension of the predicted frame-wise logits. Since the elements of the input of the binary cross entropy should be between 0 and 1, we apply a softmax function. The output has a shape [batch size, number of classes].

- **video_level_loss(out,labels):** This function returns the binary cross entropy between the video level prediction and the target vector, calculated by the functions **get_video_level_prediction(out)** and **get_target_vector(labels)** respectively.

The loss during the training is the sum of the cross entropy with the video level loss calculated by the function **video_level_loss(out,labels).**

4. Multi-scale model using the single-stage TCN with 10 layers

The model multi stage TCN is implemented in the file ‘’model.py’’ by the class ParallelTCNs(). The model is described as follow:

- The model uses 3 single stage TCNs in parallel.
- The first one takes the input as it is, the second the input downsampled by factor 4, the third by factor 8.
- In the end, after the residual layers inside each stage, the features are upsampled again to the original size, and a prediction is made using 1x1 convolutions.
- While there are many ways to upsample and downsample, e.g. linear interpolation and nearest neighbor, we perform both actions using extra convolutions. This enables the model to learn the right sampling technique, and it is the preferred way to do sampling in many of the current literature (see Transposed Convolutions)
- The model prediction is the average of the prediction of the branches

- The total loss (used in training) is the loss of each branch (summed) plus the loss of final model output: loss = loss_branch1 + loss_branch2 + loss_branch3 + loss_model

5. Results

In order to study the model behavior, we evaluate the model not only in the test set, but also in the training set. For training the Multi Stage TCN we used a batch of size 3 due to constraints related to the Cuda memory while executing the code. For the Single TCN and Parallel TCN (Multi-scale model using the single-stage TCN with 10 layers) we used a batch size of 4.

- **Training set:**

	Single TCN	**Multi Stage TCN**	**Multi Stage TCN with video loss**	**Parallel TCN***
Accuracy	89.2601	89.3809	81.4521	85.0545

- **Test set:**

	Single TCN	**Multi Stage TCN**	**Multi Stage TCN with video loss**	**Parallel TCN***
Accuracy	49.2666	49.2330	40.7909	49.4156
Edit	6.3570	6.1355	7.2946	6.9475
F1@0.10	4.7232	4.3712	4.0155	4.7735
F1@0.25	3.6205	3.4540	3.1422	3.8756
F1@0.50	2.0951	1.7435	1.8690	2.4646

*** on epoch 49, since epoch 50 was an anomaly against the previous 20 epochs**

6. Conclusion

The model is clearly able to learn something about the classes in the video, as a performance reaching 50% exceeds by far an expected random guess accuracy of 2% for 48 classes. However, we are unable to match the results in the paper.

Some factors that can contribute to this:

- Possible differences in the dataset. The model is deep and has a large capacity. It has large data needs.
- The huge difference in the dilation factor between what used in the paper (exponential) and here (linear). 10 is much smaller than 2^10, which means the perceptive field of the later layer is much smaller.
- Not using smoothing loss to avoid over segmentation.
- Overfitting: The accuracy on the training set grows disproportionately to the accuracy on the testing set (nearing a 35% difference in some iterations, like epochs 29 and 36). This also can be a good argument for needing more data, and for investigating proper regularization techniques.