Energy Clustering in CALIFA Status Update

Tobias Jenegger

Overview Clustering Models

Agglomerative Clustering + Linear Model

SR model ....

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)
under Germany's Excellence Strategy – EXC-2094 – 390783311,
BMBF 05P19WOFN1, 05P21WOFN1
and the FAIR Phase-0 program

TUM Members:
Roman Gernhäuser,Lukas Ponnath,Philipp Klenze,Tobias Jenegger

CALIFA Detector @ R^3B

CALorimeter for the **I**n **F**light detection of γ-rays and light charged p**A**rticles

Endcap:

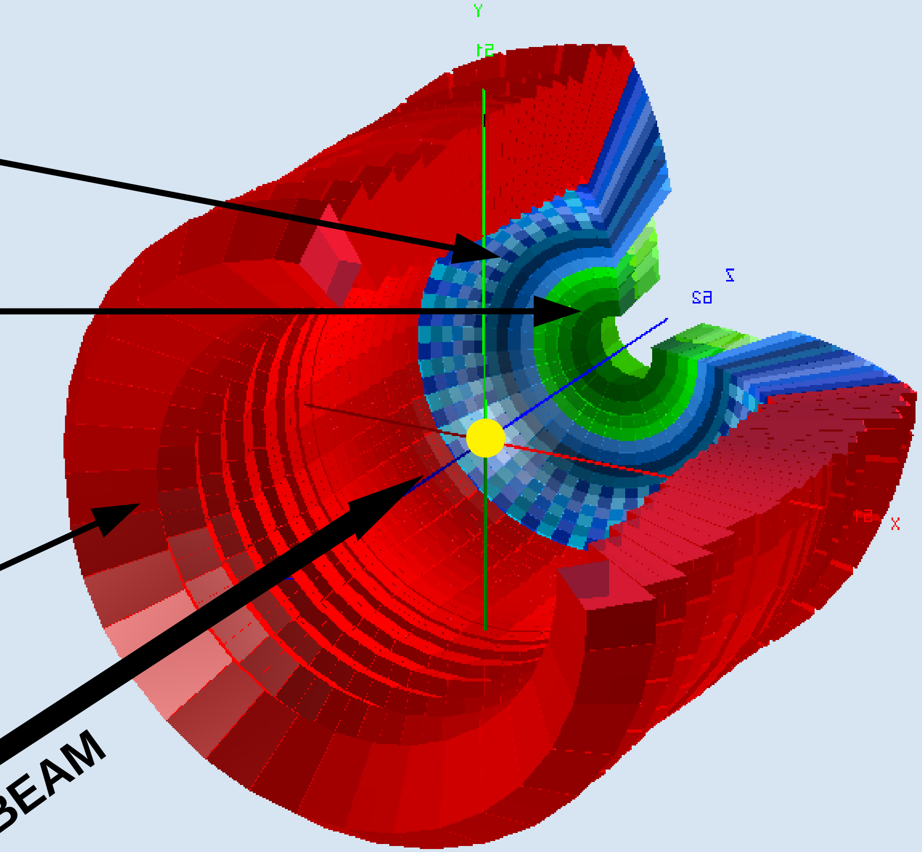
iPhos:

512 CsI(Tl) crystals

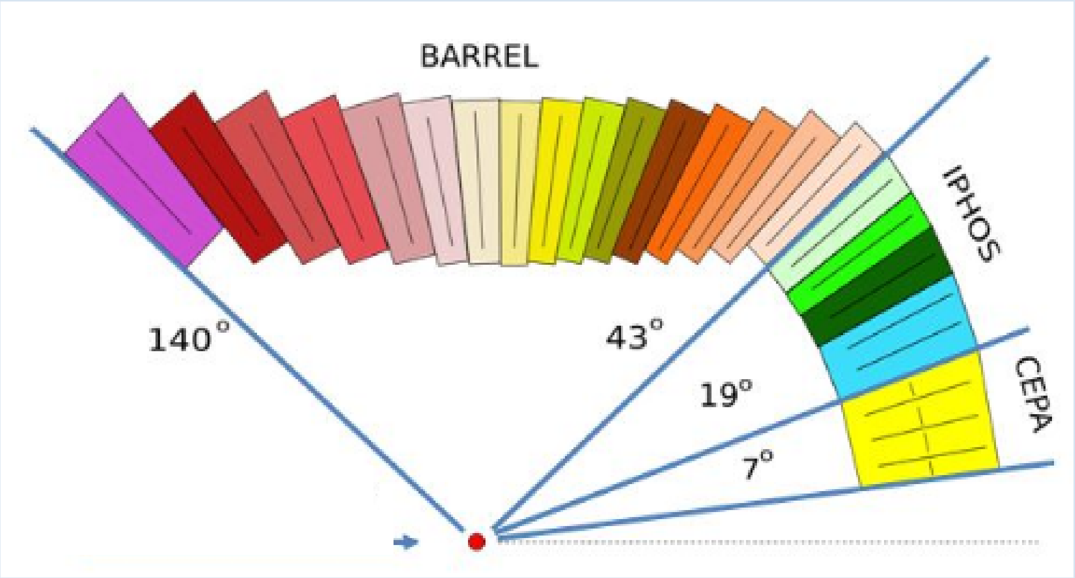
CEPA:

96 LaBr3 & LaCl3 crystals

Barrel:
1952 CsI(Tl) scintillator crystals

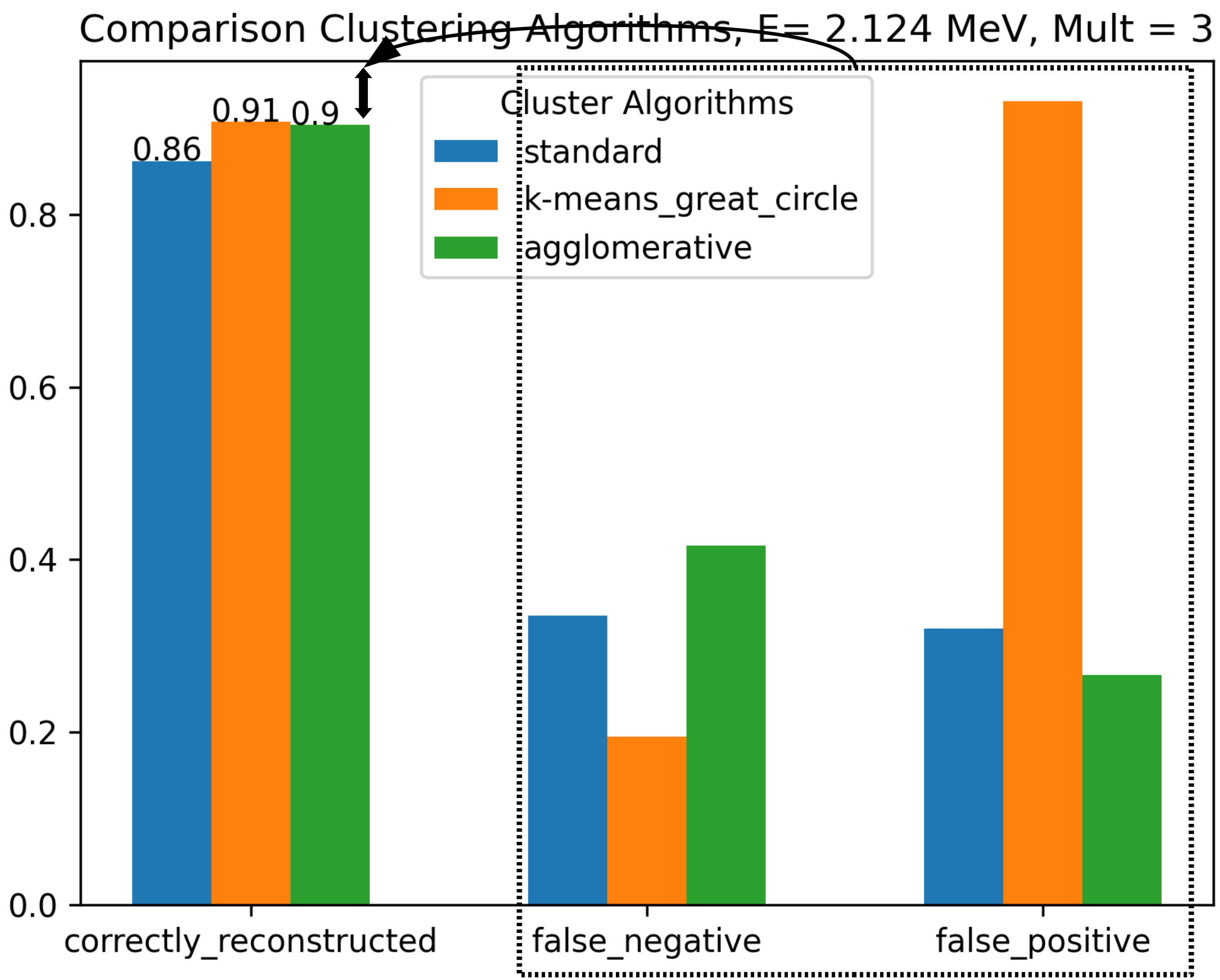


What do we measure?
- **Energy** (100 keV γ-rays – 700 AMeV charged particles)
- **Position**
- **Time**



Over 2500 crystal channels!

Summary Clustering Methods



Agglomerative Clustering + Linear Model

The Idea behind:

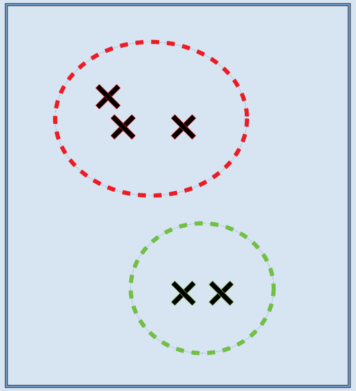
→ **First clusterize the data with agglomerative method**

→ **Select two types of events:**

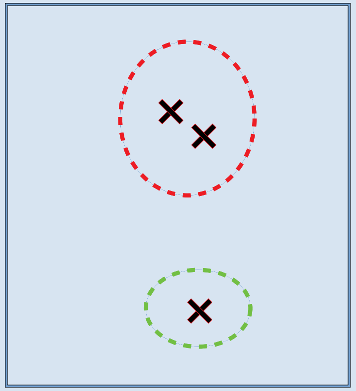
A) events with two subevents and two correctly reconstructed clusters

B) events with only one event but wrongly reconstructed two clusters

case A



case B



Feeding the Linear Model

Input:

Case A : $E_1,\theta_1,\varphi_1,t_1,E_2,\theta_2,\varphi_2,t_2,\mathrm{abs}(\Delta\theta),\mathrm{abs}(\Delta\varphi),\mathrm{abs}(\Delta t),\mathbf{0}$

Case B: $E_1,\theta_1,\varphi_1,t_1,E_2,\theta_2,\varphi_2,t_2,\mathrm{abs}(\Delta\theta),\mathrm{abs}(\Delta\varphi),\mathrm{abs}(\Delta t),\mathbf{1}$

Values φ,θ,t(ime) are the center of mass values

```
class TinyModel(torch.nn.Module):
    def __init__(self):
        super(TinyModel, self).__init__()
        self.linear1 = torch.nn.Linear(11,64)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(64,1)
        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.sigmoid(x)
        return x
~
~
```

Input dim: 11

Output dim: 1

See: https://github.com/jenegger/agglo_linear_model

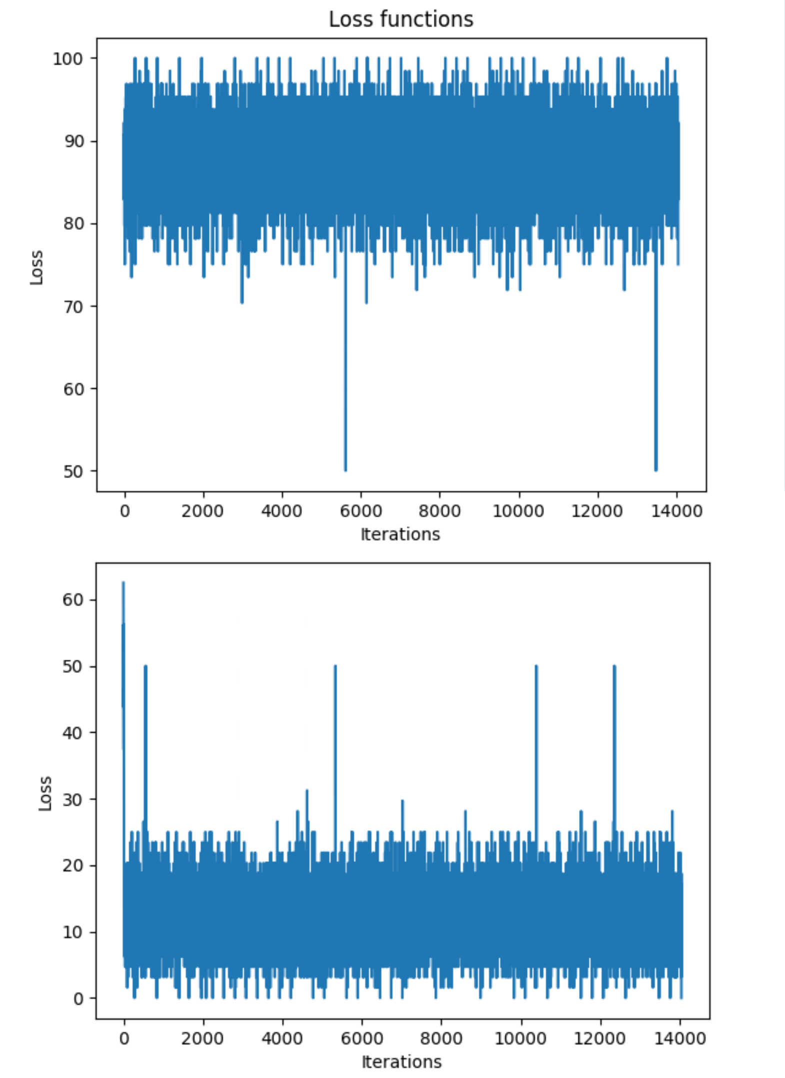
Model Parameters:

Input features: 11
nn.Linear- features: 64/32/16/8/4
Output features: 1

Batch-size: 64/32/16
Learning-rate: 3e-4
Optimizer: SGD/Adam
num_epochs: 50/500

Data Input:
sample size: 17924 from which
→ 2127 case B (bad reco)
→15797 case A (well reco)

Loss Function- BCELoss



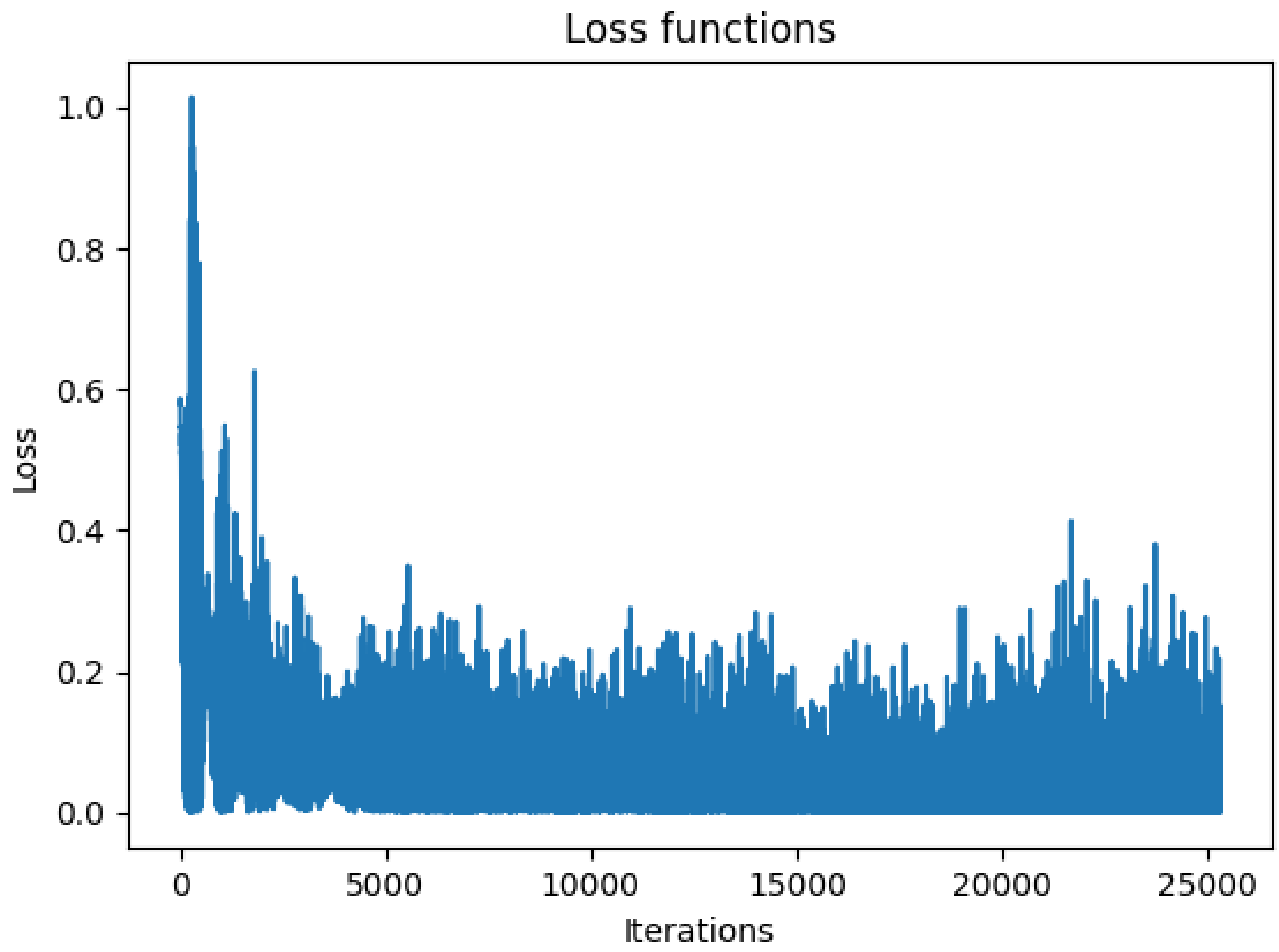
- Loss does not seem to decrease

- Changing different parameters , eg. nn.Linear features, lr, batchsize does not change behaviour

- Is the data size to small? Since there is no tendency of decreasing loss, this does not seem to be the bottleneck

Now with time normalization

Best hyper-parameters
Learning rate: 8e-5
nn.Linear features : 64
Batchsize: 64
Optimizer: SGD
Num. Epochs: 100



Other Idea:

Input data:

Events, with 3 true clusters and e.g. 9 hits

→

Compute all different combinations of the hits & appropriate mask

511 combinations for 9 hits

Mask:

- hit_i & hit_j do not belong to same true cluster → 0
- hit_i & hit_j belong together and are the complete cluster → 1
- hit_i & hit_j belong together and are part of bigger cluster → E_i+E_j /E_true

→

Feed the combinations to a Model

Issue: multi-dimensional model,
CNN is needed

Appendum - MLSegmenter Model

Clustering of CALIFA Data with MLSegmenterTorch Model

Adapt input shape, clusternr. Etc.

mlsegmenter = MLSegmenterTorch(**image_shape=(1,27,112)**, **nslots_max=3**, encoder_output_shape=(256,1,7), nlatent=16, encoder_class=**Encoder_ae,** decoder_class=DecoderCNN_Resnet5, slotmaker_class=**Slotmaker_ae**)

Q: Should I use nslots_max= 3 or should I use a higher number so that the algorithm tries to find the best cluster number?

Califa DataSet Class

```
class califa_Dataset(Dataset):
    def __init__(self, steps_per_epoch, batch_size = 64):

        self.batch_size = batch_size
        self.size = steps_per_epoch
        self.cluster_gen_obj = make_batch(N_events=self.batch_size)

    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        event_images, _  = self.cluster_gen_obj

        event_images= torch.unsqueeze(event_images,dim=1)
        #print("type of event_images")
        #print(type(event_images))

        #event_images = torch.tensor(event_images, dtype=torch.float32)
        event_images_out = event_images.clone().detach()
        #print(event_images_out.shape)
        #print(event_images.shape)

        return event_images, event_images_out
```

This function selects randomly three clusters from my simulated data and merges them to one event

The final training steps:

```
num_epochs = 30
losses =[]

for epoch in range(num_epochs):
    mlsegmenter.train()
    mlsegmenter.to(device)
    total_epoch_loss = 0
    dataset_from_gen = califa_Dataset(steps_per_epoch=90)
    train_loader = DataLoader(dataset_from_gen, batch_size=None, shuffle=True, num_workers=20)
    for batch_idx, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)
        opt_full.zero_grad()
        reconstruction = mlsegmenter(inputs)

        loss = binary_loss_internal_summation(inputs, reconstruction)

        loss.backward()
        opt_full.step()

        total_epoch_loss += loss.item()
        if batch_idx == 0:
            plot_realistic(0,inputs,reconstruction,losses)

    epoch_loss = total_epoch_loss / len(train_loader)
    losses.append(float(epoch_loss))
    #plot_realistic(0,

    print(f"Epoch {epoch+1}/{num_epochs} => loss: {epoch_loss:.4f}")
```

How well does the Model train/perform?

Looking at the loss function progress it seems to converge:

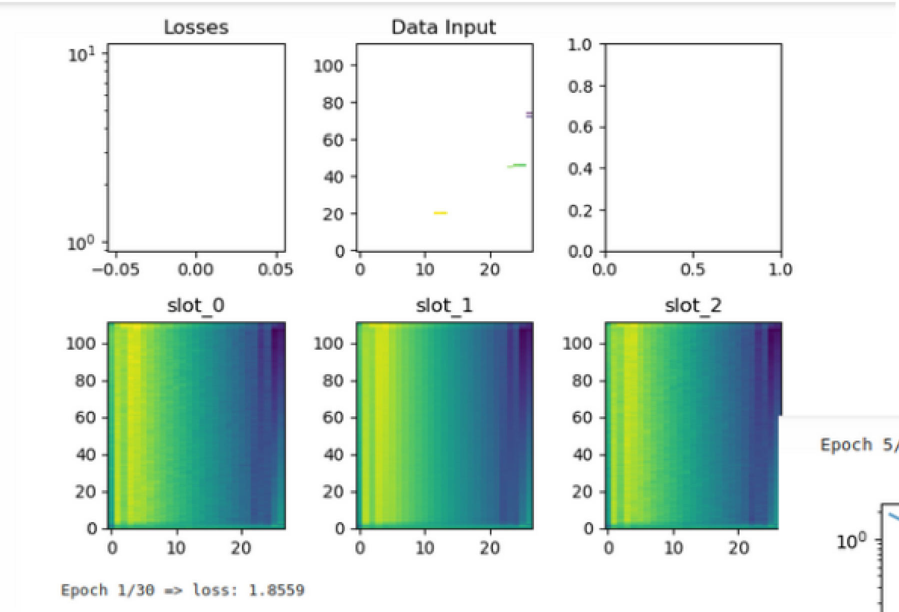
Epoch 1/30 => loss: 2.0170
Epoch 2/30 => loss: 1.3786
Epoch 3/30 => loss: 0.2274
Epoch 4/30 => loss: 0.0186
Epoch 5/30 => loss: 0.0057
Epoch 6/30 => loss: 0.0028
Epoch 7/30 => loss: 0.0017
Epoch 8/30 => loss: 0.0011
Epoch 9/30 => loss: 0.0008
Epoch 10/30 => loss: 0.0006
Epoch 11/30 => loss: 0.0005
Epoch 12/30 => loss: 0.0004
….

Epoch 30/30 => loss: 0.0001

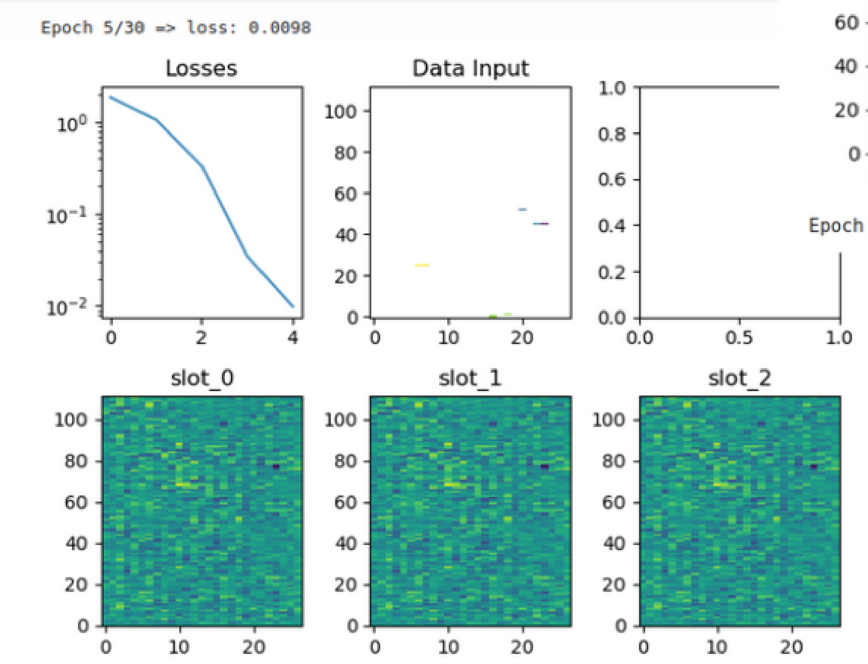
Seems good….

But when plotting the input data vs the reconstructed slots:

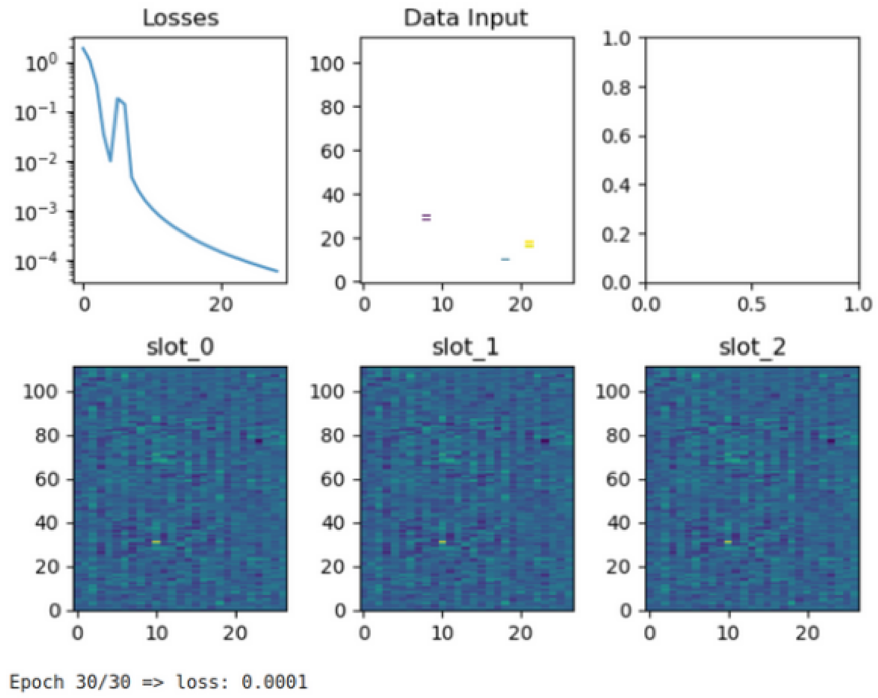
First epoch



Epoch 5



Epoch 30



Short Summary

I tried to implement the MLSegmenter Model (only spatial info and energy info, time info not yet)
→ the losses seem to converge
→ however clustering doesn't seem to work. It seems so that the algorithm converges to a state where all pixels in the slots go to 0

If you have any advices, spotted bugs in my code or ideas for improvements please let me know!

Thank you!

CALIFA @ Technical University of Munich (TUM)

Roman Gernhäuser, Lukas Ponnath, Philipp Klenze, Tobias Jenegger

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung