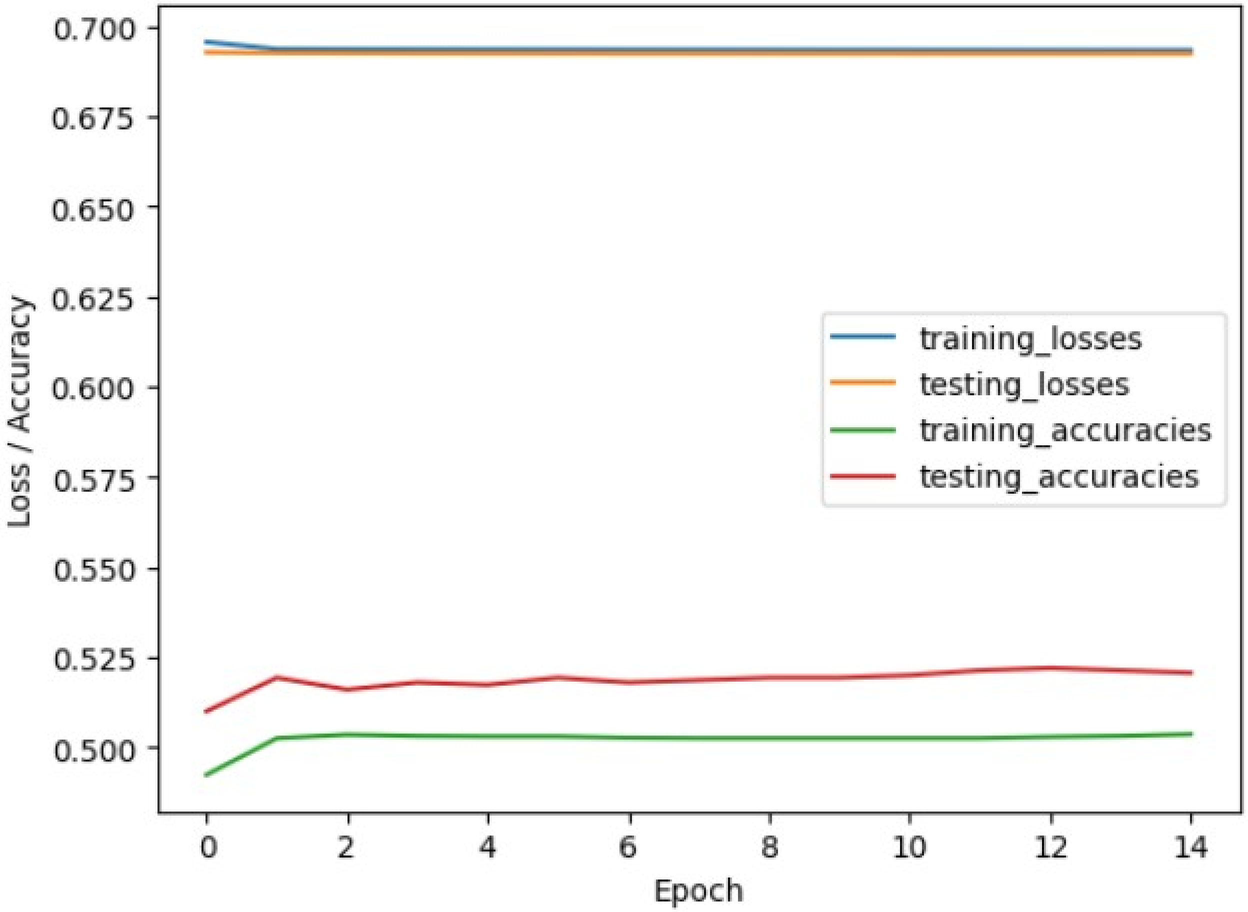
Quark/Gluon Jet Tagger

- Suspiciously good performance with little training data (8,000 jets)

  Usual checks:
  - shuffle labels, does the network learn
    - No! good! Check! (max accuracy: 52.2%)
  - Overtraining, does the network learn the training data by heart
    - Yes, testing/validation performance decreases
- Still a lot more to test, but at first sight, accuracy at ~78% (state of the art: 84% w/ O(2,000,000 jets) for training)

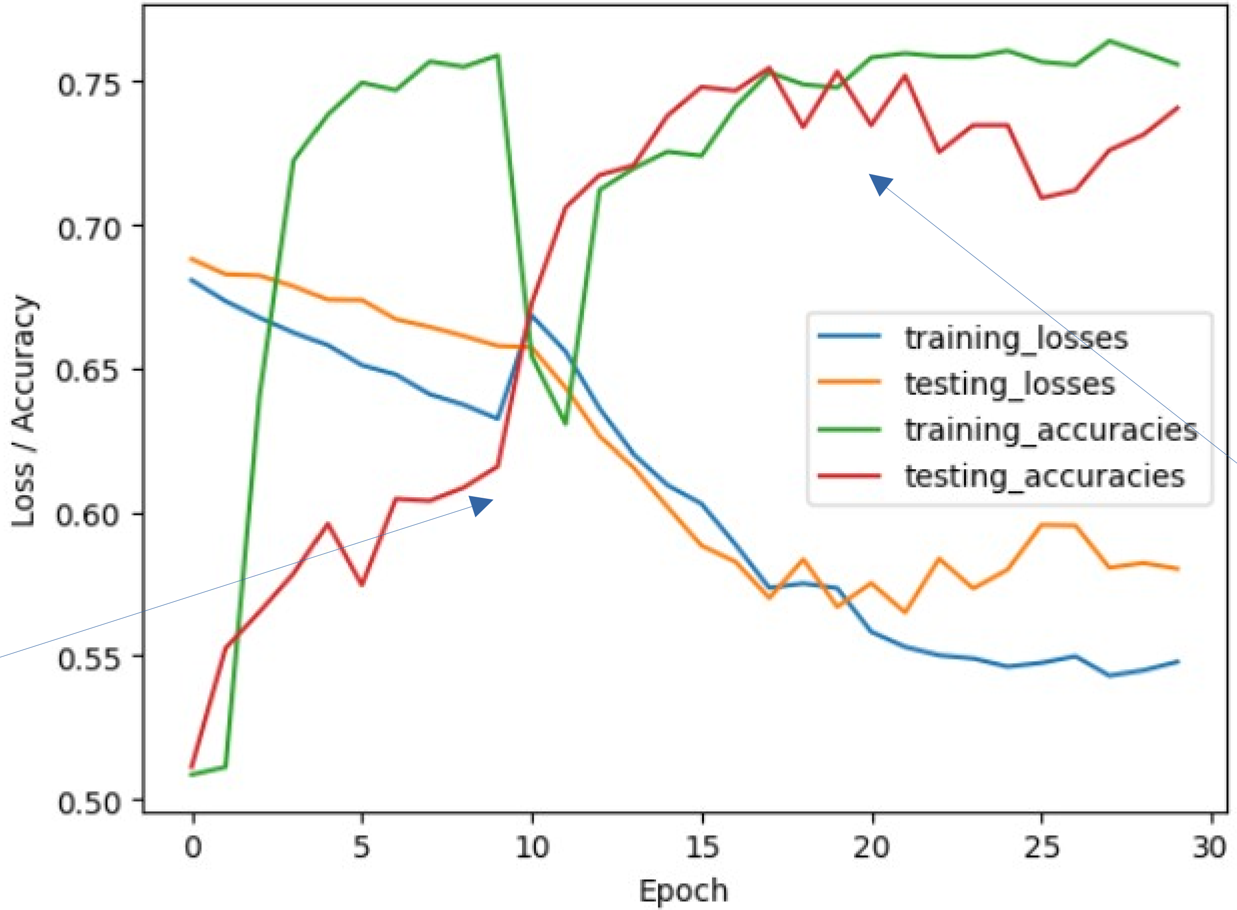
Training & Testing on shuffled labels



- 49% gluons
- 51% quarks

→ balanced dataset

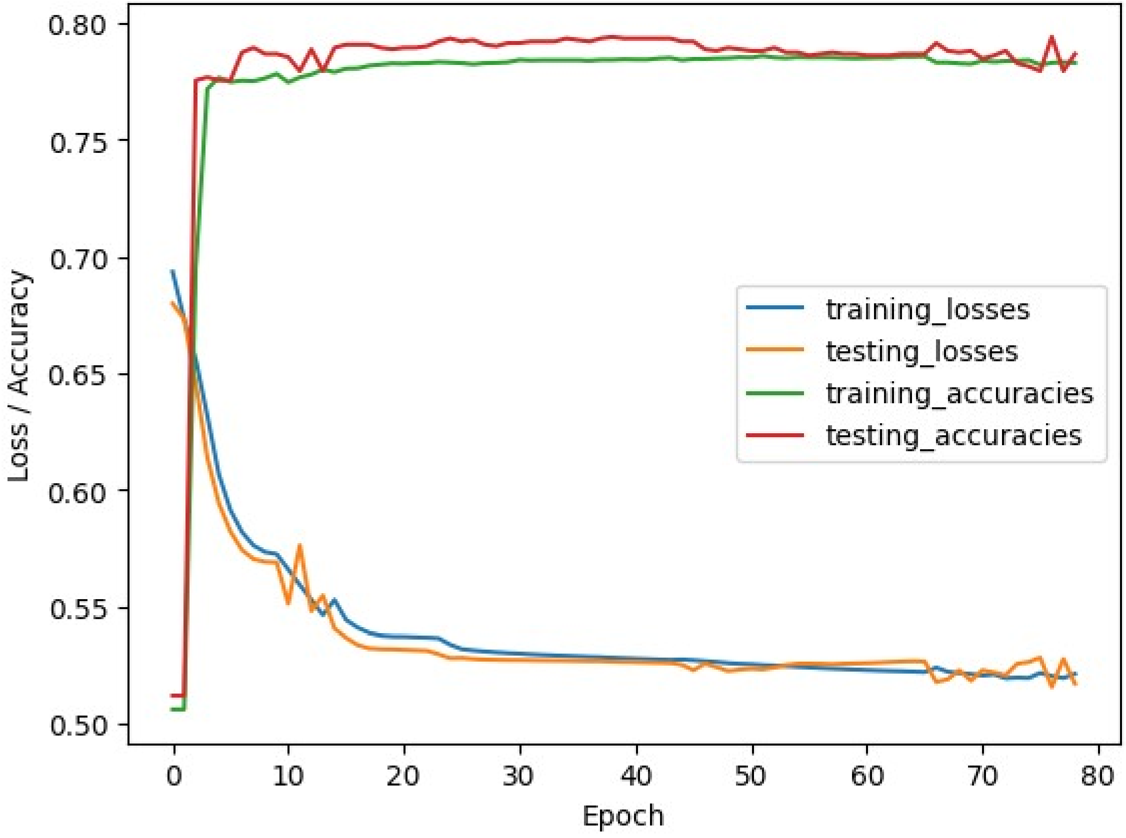
Training on non-shuffled dataset



LR * 10

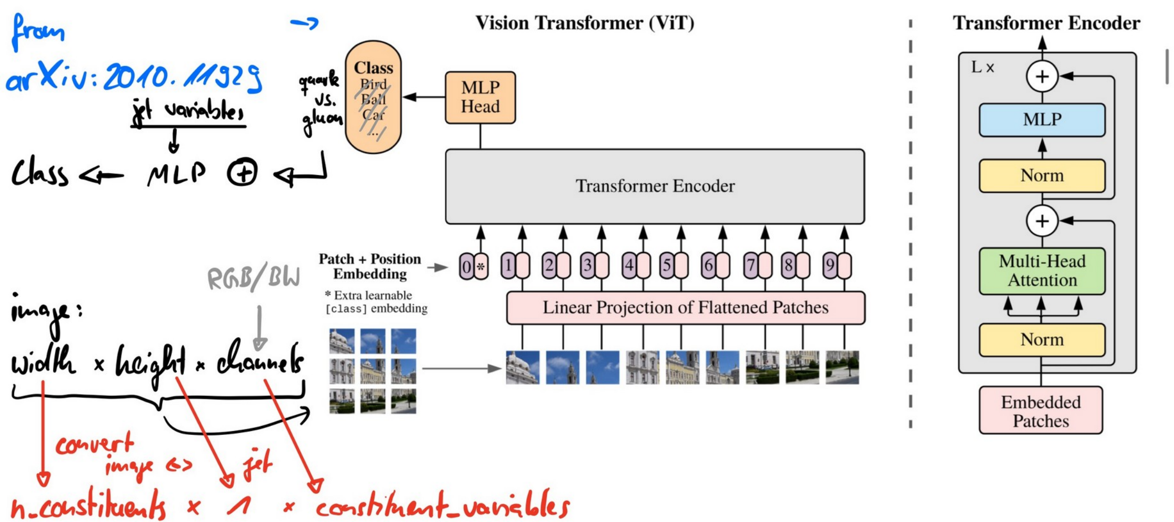
LR * 0.75

Overtraining



JetViT Architecture

```python
self.bs, sequence_length, self.dimensionality = jet_constituents.shape

padding_mask = torch.tensor(create_padding_mask(jet_constituents.cpu().detach().numpy()).any(axis=2))

# embedding of the input -> (bs, sequence_length, hidden_d)
tokens = self.linear_mapper(jet_constituents)

# adding classification token to the tokens -> (bs, sequence_length+1, hidden_d)
tokens = torch.cat((self.class_token.expand(self.bs, 1, self.hidden_d), tokens), dim=1)
tensor_true = torch.full((self.bs, 1), True, dtype=torch.int)
padding_mask = torch.cat((tensor_true, padding_mask), dim=1)
padding_mask = padding_mask[:, :, None] @ padding_mask[:, None, :]
padding_mask = padding_mask.bool().to(self.device)

# adding a positional embedding (based on order of particles in jet)
if pos_embed:
    pos_embed = self.pos_embed.repeat(self.bs, 1, 1)
    tokens = tokens + pos_embed

# transformer encoder:
for block in self.blocks:
    tokens = block(tokens, padding_mask)

# Getting the classification token only
tokens = tokens[:, 0]

# Processing classification tokens
tokens = self.mlp(tokens) # Map to output dimension, output category distribution

# Processing global jet features
jet_tokens = self.jet_mlp(jets)

# combining tokens
comb_tokens = torch.cat((tokens, jet_tokens), dim=1)

return self.token_combiner(comb_tokens)
```



How important are the jet constituents / is the global jet info?

```
Training ...
100%|██████████| 250/250 [00:59<00:00,  4.21it/s]
Training loss: 0.62
Testing ...
100%|██████████| 1500/1500 [00:06<00:00, 232.61it/s]
Testing accuracy: 0.7466666666666667 best accuracy: 0.7466666666666667
```

```python
# print the final mlp token combiner params:
for name, param in model.named_parameters():
    if param.requires_grad:
        if (name == 'token_combiner.0.weight') or (name == 'token_combiner.0.bias'):
            print(name, param.data)
```

```
token_combiner.0.weight tensor([[-0.2955, -0.0300, -0.8453,  0.3120],
        [-0.2995, -0.1621,  0.7126, -0.0198]], device='cuda:0')
token_combiner.0.bias tensor([0.3255, 0.0582], device='cuda:0')
```

Note to self:
I still need to calculate the token combination matrix for many trainings and compare!

```python
constituents_tokens = torch.tensor([1., 0.])
jet_tokens = torch.tensor([0., 1.])
```

```python
combiner_weights = torch.tensor([
    [-0.2955, -0.0300, -0.8453,  0.3120],
    [-0.2995, -0.1621,  0.7126, -0.0198]
])

combiner_bias = torch.tensor([0.3255, 0.0582])
```

```python
tokens = torch.cat((constituents_tokens, jet_tokens))
```

```python
F.softmax(combiner_weights @ tokens + combiner_bias, dim=0)
```

```
tensor([0.6464, 0.3536])
```

Now: also added Jet Angularities

- +1% in testing accuracy

$$\lambda_\alpha^\kappa = \sum_{i\in\text{jet}} \left(\frac{p_{T,i}}{\sum_{j\in\text{jet}} p_{T,j}}\right)^\kappa \left(\frac{\Delta_i}{R}\right)^\alpha .$$

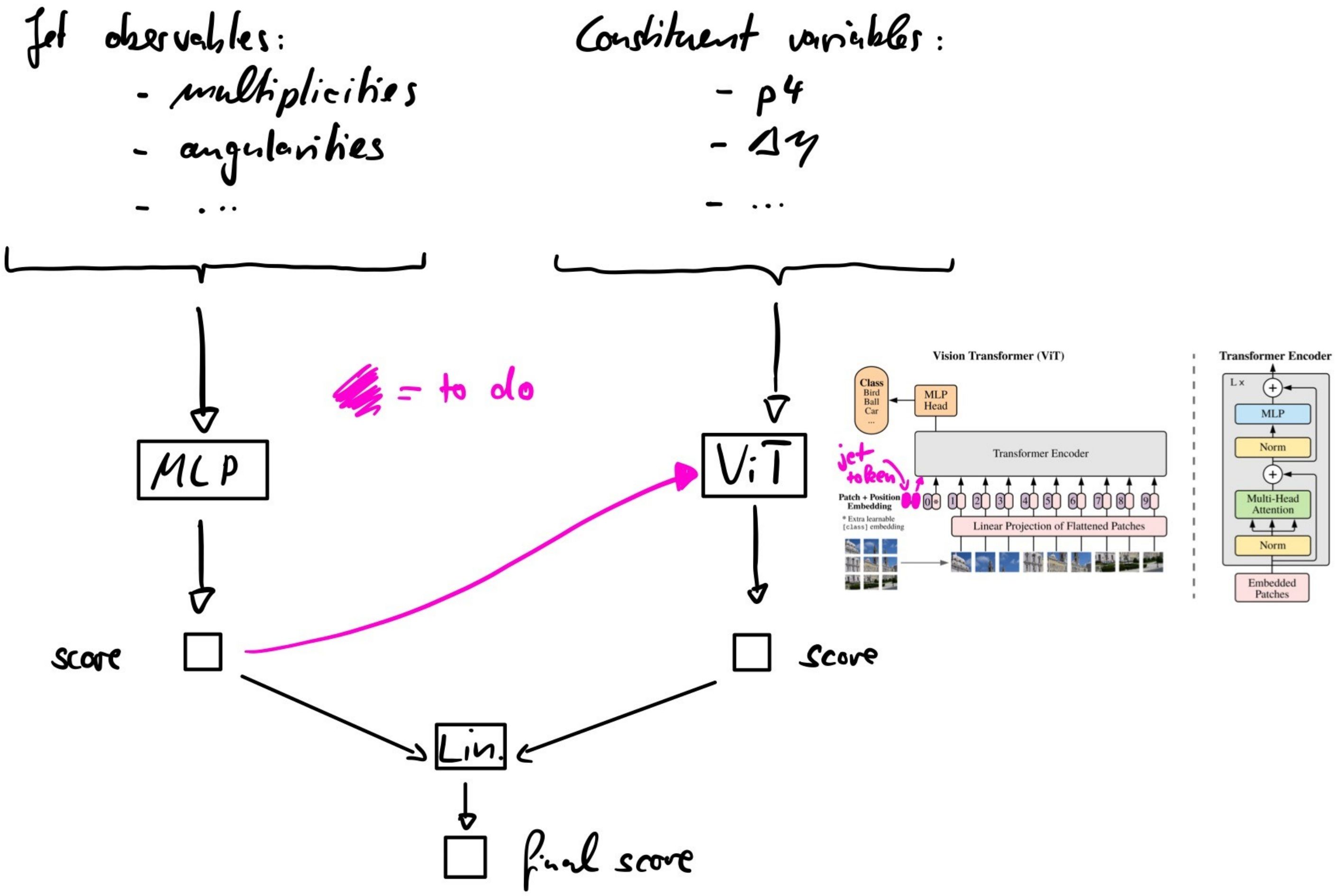
Here

$$\Delta_i = \sqrt{(y_i - y_{\text{jet}})^2 + (\phi_i - \phi_{\text{jet}})^2}\,,$$

Ideas/Next Steps

- Implement same metrics as ParT arXiv:2202.03772v2
- Still suspicious: generate some MC samples with Sherpa → validate performance
- Implement ‘Jet bias’

Jet Bias



Jet observables:
- multiplicities
- angularities
- ...
Constituent variables:
- p4
- Δy
- ...
= to do
MLP
ViT
jet token
score
score
Lin.
final score
Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches