Introduction to Deep Learning (I2DL)

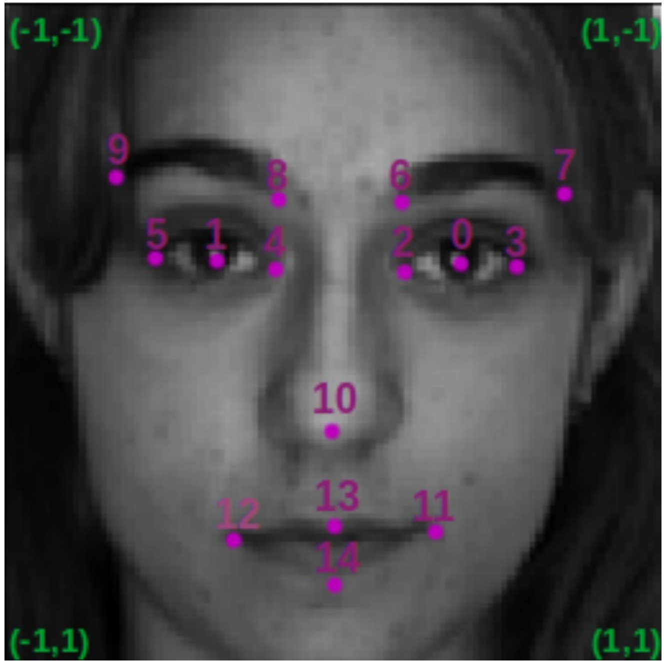
Exercise 9: Convolutional Layers and Q&A

Future Exercises and Transfer Learning

Exercise Overview

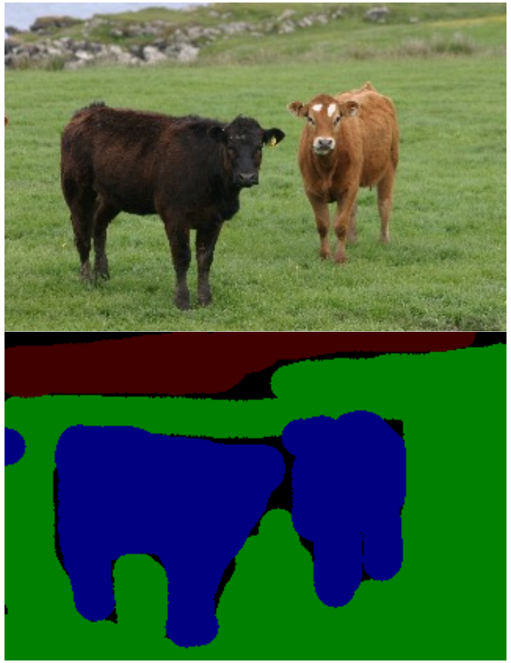
Exercise 01: Organization Exercise 02: Math Recap	Intro
Exercise 03: Dataset and Dataloader Exercise 04: Solver and Linear Regression Exercise 05: Neural Networks Exercise 06: Hyperparameter Tuning	Numpy (Reinvent the wheel)
Exercise 07: Introduction to Pytorch Exercise 08: Autoencoder	Pytorch/Tensorboard
Exercise 09: Convolutional Neural Networks Exercise 10: Semantic Segmentation Exercise 11: Recurrent Neural Networks	Applications (Hands-off)

Exercises 9&10: Scale it up



Exercise 9

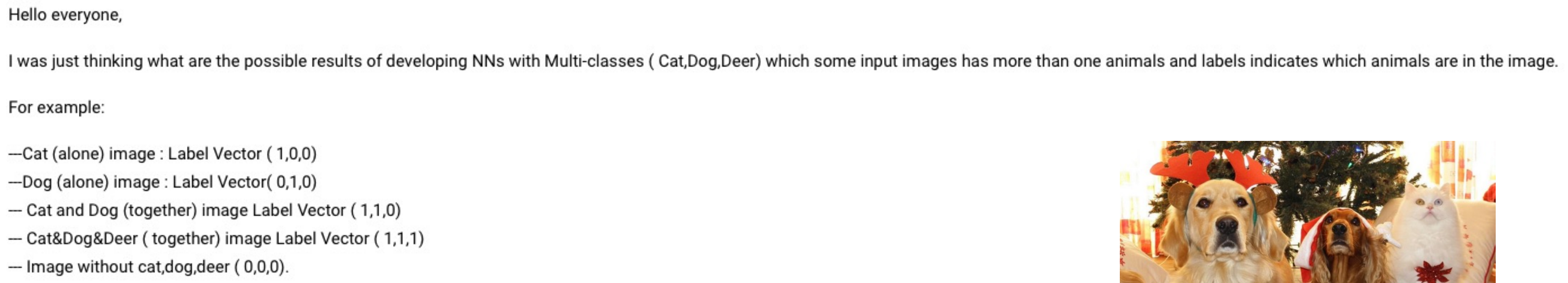
(1, 96, 96) grayscale image



Exercise 10

(3, 224, 224) RGB image

An innocent question…



Hello everyone,

I was just thinking what are the possible results of developing NNs with Multi-classes (Cat,Dog,Deer) which some input images has more than one animals and labels indicates which animals are in the image.

For example:

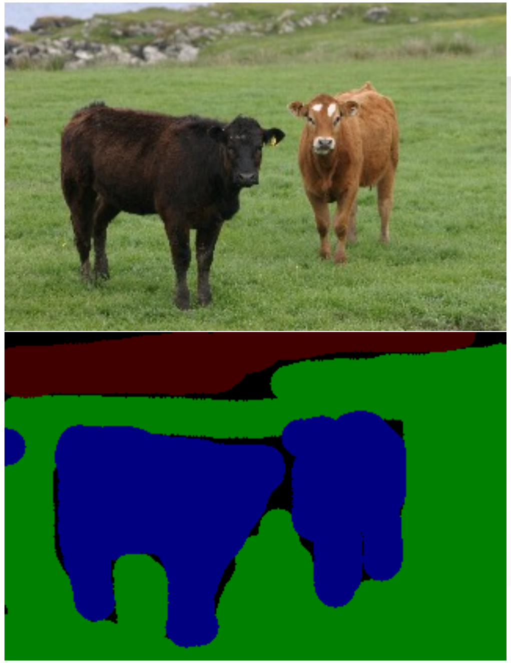
--Cat (alone) image : Label Vector (1,0,0)
--Dog (alone) image : Label Vector(0,1,0)
-- Cat and Dog (together) image Label Vector (1,1,0)
-- Cat&Dog&Deer (together) image Label Vector (1,1,1)
-- Image without cat,dog,deer (0,0,0).

- How would you optimize over it?
  - How should the architecture look like?
- Why should training for this objective give better results?
  - For which objective do you want to utilize this behaviour?
- What should the data look like
  - And most importantly: where/how do you get it?

How do we obtain our data?

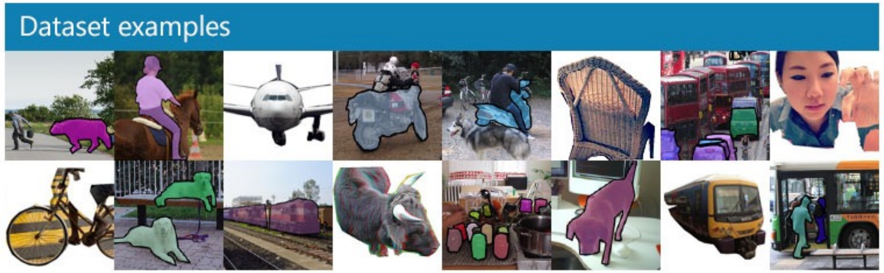
Who generates it?

- A single person
  - E.g., poor PhD candidates
- Crowdsourcing like Amazon Mechanical Turk
  - Harder tasks -> less and worse results



Dataset Size Comparison

- COCO – 200k

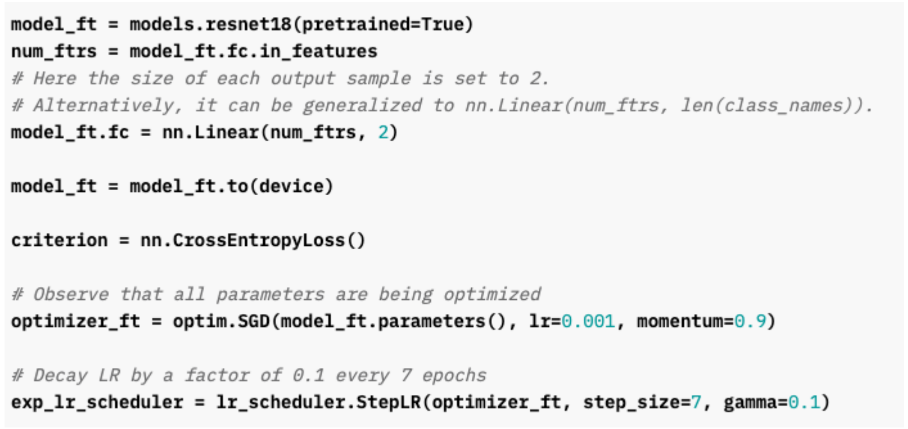


- Imagenet - 14m

Instagram - Billions

Use datasets for best performance

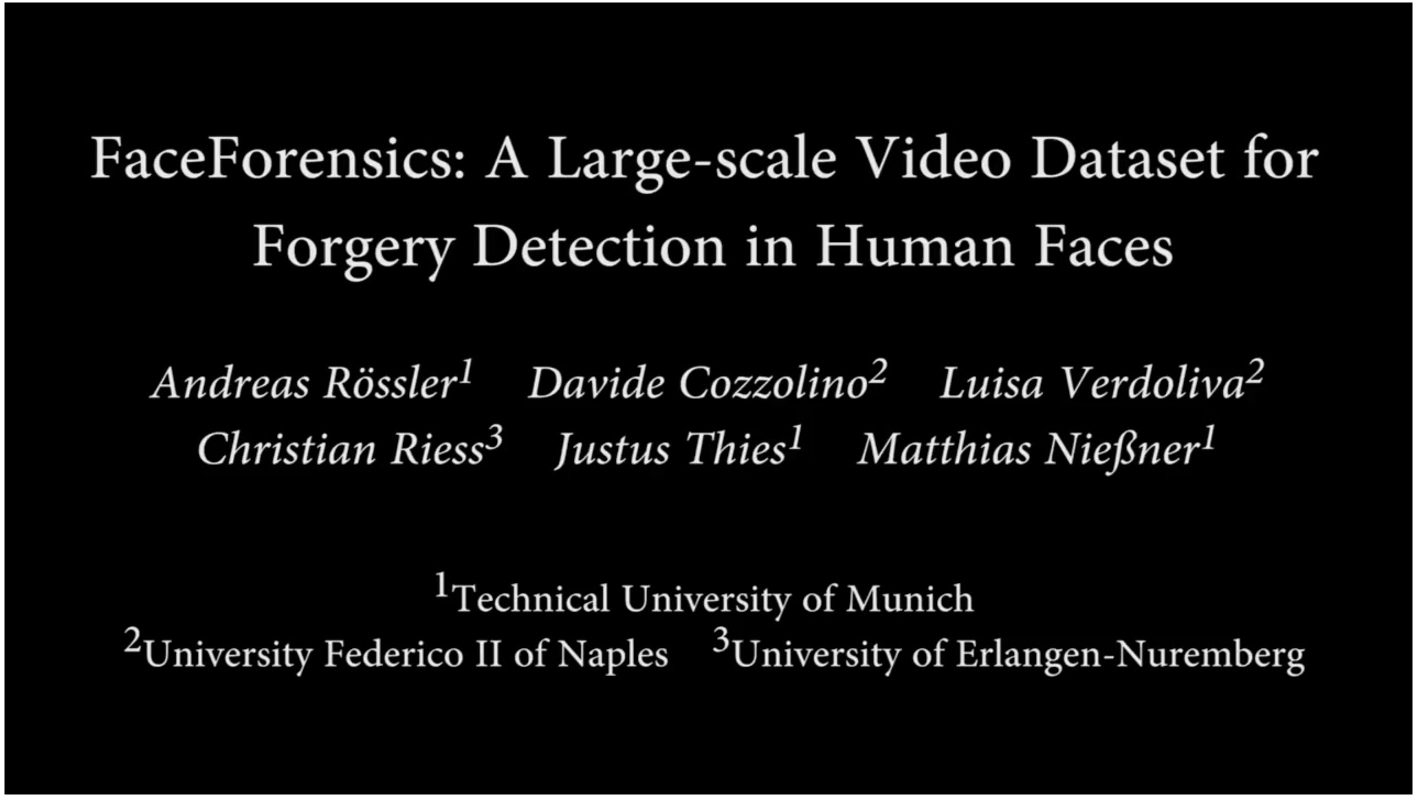
- Simple tutorial: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html



```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

- Multiple possibilities in Pytorch to fix network parts
  - For some layers: Required_grad = False
  - Only feed parts of the network to optimizer

Case Study: FaceForensics



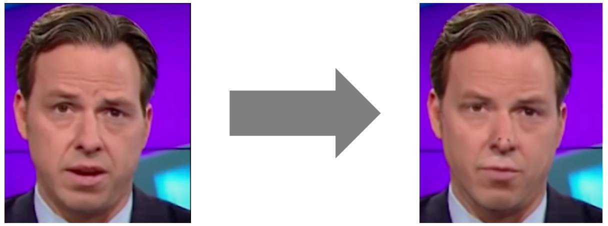
FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces

Andreas Rössler[1] *Davide Cozzolino*[2] *Luisa Verdoliva*[2]
Christian Riess[3] *Justus Thies*[1] *Matthias Nießner*[1]

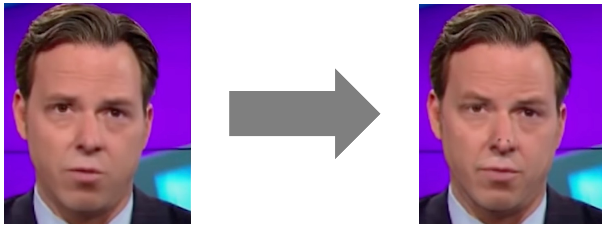
[1]Technical University of Munich
[2]University Federico II of Naples [3]University of Erlangen-Nuremberg

AE Case Study: FaceForensics

- Goal: Refine Face2Face
- Problem: We have non-matching pairs



- Idea: Self-Reenactment
  Run Face2Face again with source == target

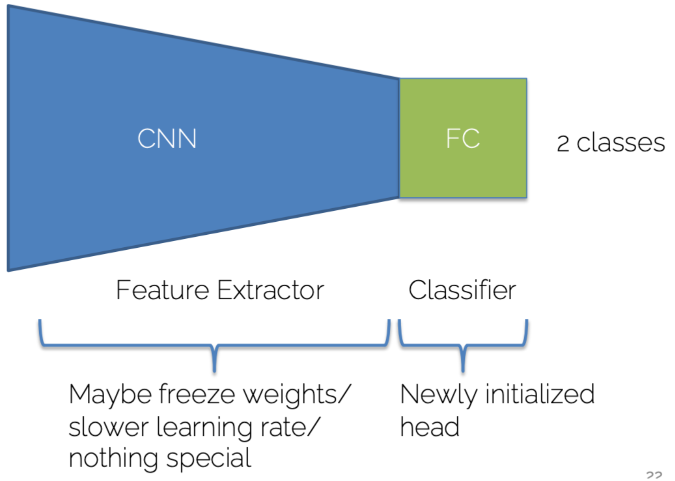


AE Case Study: FaceForensics

- New Problem: We only have 1000 dataset identities
- Idea:
  - Use VGGFace2 dataset
  - Adopt cropping technique from them
  - Pre-train an Autoencoder on VGGFace2
  - Freeze parts, add skip connections etc. to train Refiner on Face2Face data

TL Case Study: FaceForensics

- Crop region around face
- Resize to 224x224
- Use existing Imagenet architecture where we replace the output linear layer with a binary one for real/fake



Exercise 9: Is it always easy?

Input:

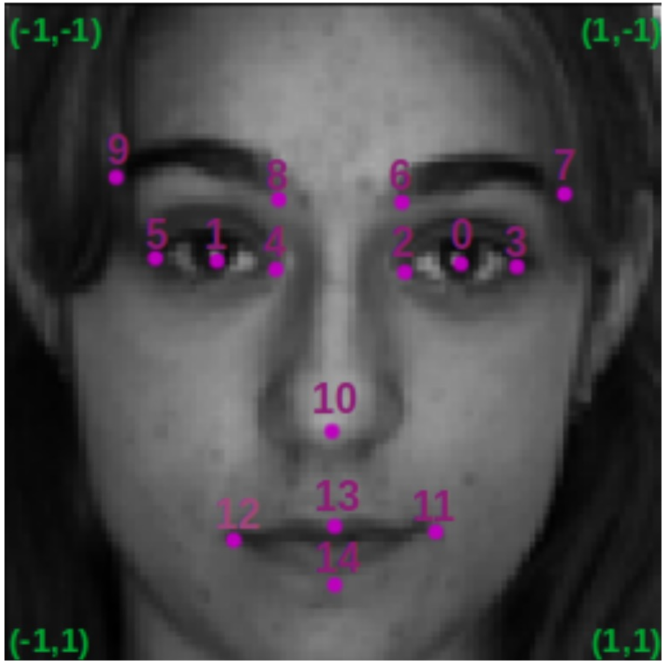
(1, 96, 96) image

- Grayscale, not RGB
- Rather small compared to 224

Output:

coordinates of facial keypoints

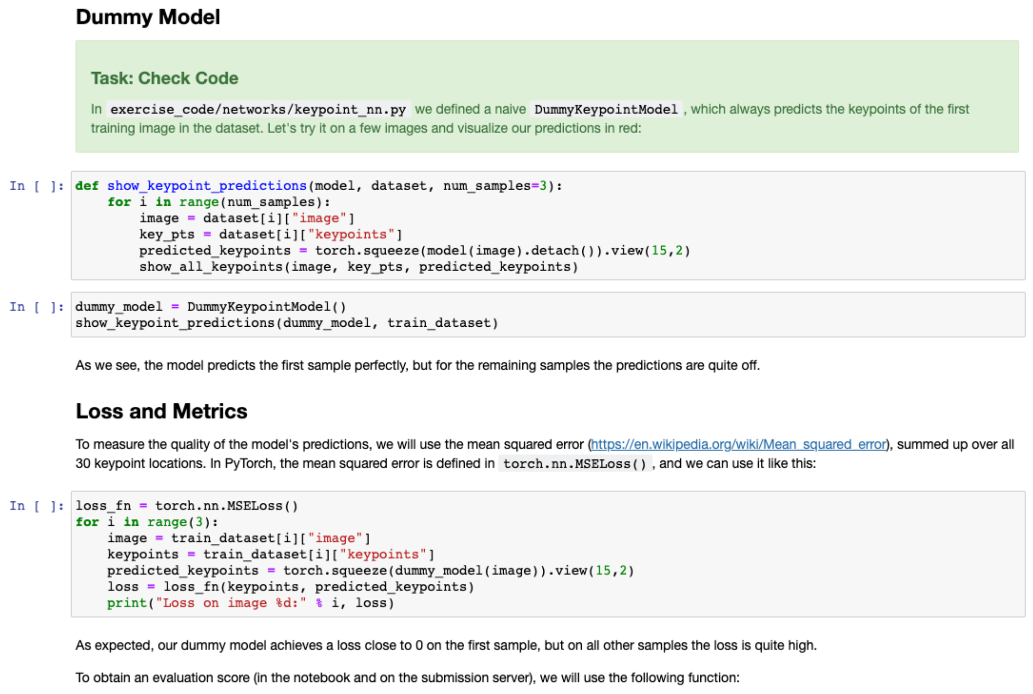
(2, 15)



Hands-Off

Communication
is encouraged!

- Solve it by yourself
  - At least one network of your own design
- Meet up and discuss
  - Campuswire/offline



Dummy Model

Task: Check Code

In `exercise_code/networks/keypoint_nn.py` we defined a naive `DummyKeypointModel`, which always predicts the keypoints of the first training image in the dataset. Let's try it on a few images and visualize our predictions in red:

```
In [ ]: def show_keypoint_predictions(model, dataset, num_samples=3):
            for i in range(num_samples):
                image = dataset[i]["image"]
                key_pts = dataset[i]["keypoints"]
                predicted_keypoints = torch.squeeze(model(image).detach()).view(15,2)
                show_all_keypoints(image, key_pts, predicted_keypoints)
```

```
In [ ]: dummy_model = DummyKeypointModel()
        show_keypoint_predictions(dummy_model, train_dataset)
```

As we see, the model predicts the first sample perfectly, but for the remaining samples the predictions are quite off.

Loss and Metrics

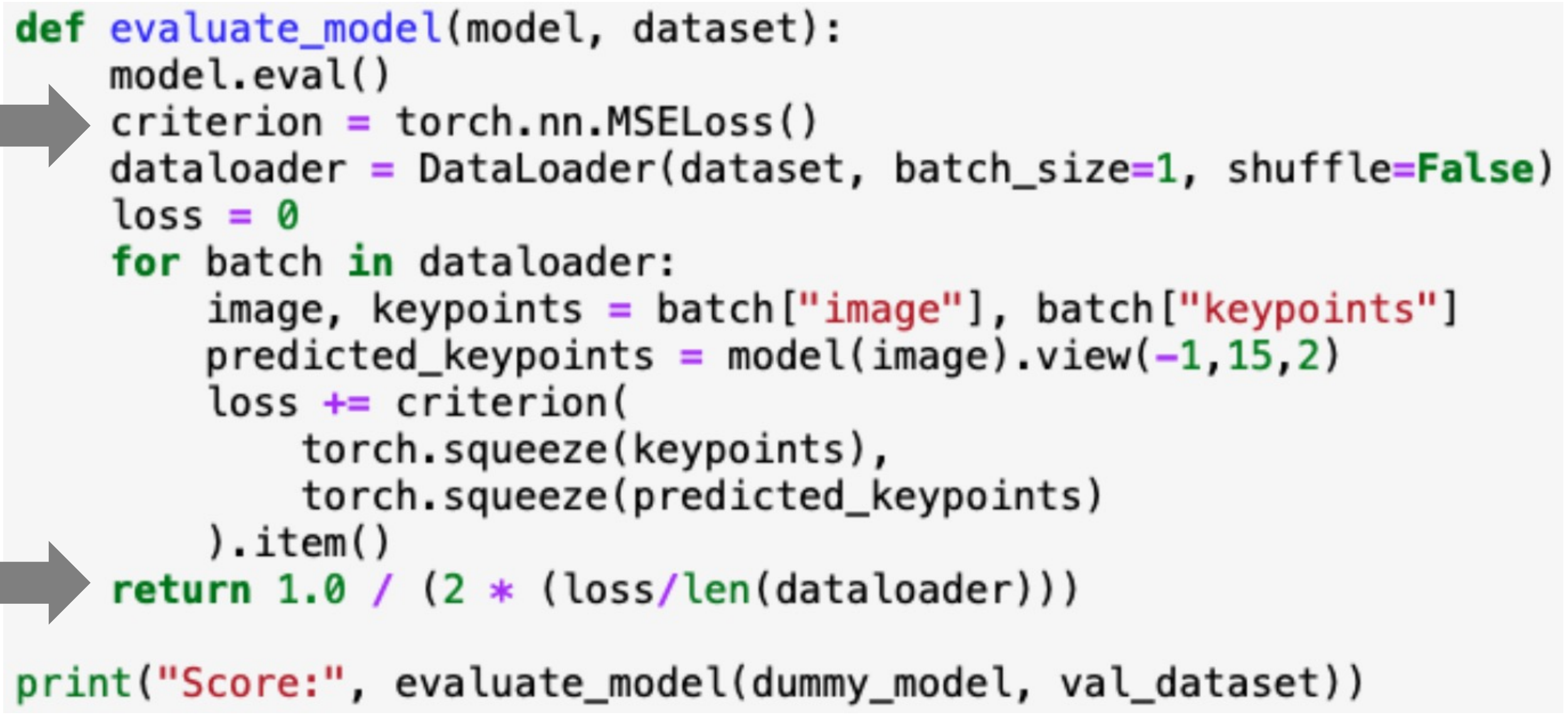
To measure the quality of the model's predictions, we will use the mean squared error (https://en.wikipedia.org/wiki/Mean_squared_error), summed up over all 30 keypoint locations. In PyTorch, the mean squared error is defined in `torch.nn.MSELoss()`, and we can use it like this:

```
In [ ]: loss_fn = torch.nn.MSELoss()
        for i in range(3):
            image = train_dataset[i]["image"]
            keypoints = train_dataset[i]["keypoints"]
            predicted_keypoints = torch.squeeze(dummy_model(image)).view(15,2)
            loss = loss_fn(keypoints, predicted_keypoints)
            print("Loss on image %d:" % i, loss)
```

As expected, our dummy model achieves a loss close to 0 on the first sample, but on all other samples the loss is quite high.

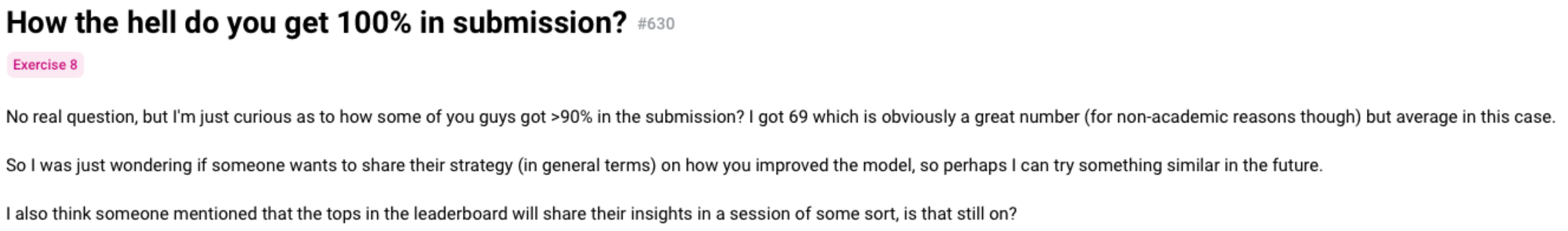
To obtain an evaluation score (in the notebook and on the submission server), we will use the following function:

Submission Metric



```python
def evaluate_model(model, dataset):
    model.eval()
    criterion = torch.nn.MSELoss()
    dataloader = DataLoader(dataset, batch_size=1, shuffle=False)
    loss = 0
    for batch in dataloader:
        image, keypoints = batch["image"], batch["keypoints"]
        predicted_keypoints = model(image).view(-1,15,2)
        loss += criterion(
            torch.squeeze(keypoints),
            torch.squeeze(predicted_keypoints)
        ).item()
    return 1.0 / (2 * (loss/len(dataloader)))

print("Score:", evaluate_model(dummy_model, val_dataset))
```

The Sad Truth



How the hell do you get 100% in submission? #630

Exercise 8

No real question, but I'm just curious as to how some of you guys got >90% in the submission? I got 69 which is obviously a great number (for non-academic reasons though) but average in this case.

So I was just wondering if someone wants to share their strategy (in general terms) on how you improved the model, so perhaps I can try something similar in the future.

I also think someone mentioned that the tops in the leaderboard will share their insights in a session of some sort, is that still on?

Rules for Ex9:

- Don't use the original dataset
- Don't use available networks trained on this dataset
- Otherwise: everything goes
  - Start with non-pretrained version first
  - If you need/want to use an Imagenet network, use MobileNet

Derivatives and Exam

More Exam FAQ

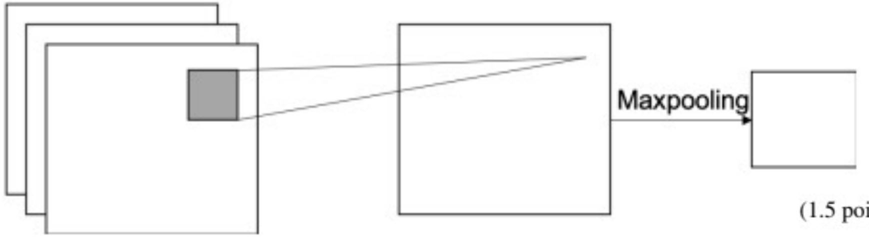
- Do we need to be able to compute derivatives?
  Yes, see mock exam.

- Help! I can’t do crazy math in my head. Let me bring my calculator.
  We hear you. The numbers in the actual exam will be much simpler. A calculator is not allowed.

- Wait, so we also need to be able to backpropagate through a convolutional layer even if we didn’t cover it in the lecture?
  No, not in the general case. You must be able to transfer your knowledge, however.

Example: Backpropagation through CNN

Your friend is excited to try out those "Convolutional Layers" you were talking about from your lecture. However, he seems to have some issues and requests your help for some theoretical computations on a toy example.
Consider a neural network with a convolutional (without activation) and a max pooling layer. The convolutional layer has a single filter with kernel size $(1,1)$, no bias, a stride of 1 and no padding. The filter weights are all initialized to a value of 1. The max pooling layer has a kernel size of $(2,2)$ with stride 2, and 1 zero-padding.



You are given the following input image of dimensions $(3,2,2)$:

$$x = \left(\begin{bmatrix} 1 & -0.5 \\ 2 & -2 \end{bmatrix}, \begin{bmatrix} -2 & 1 \\ -1.5 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \right)$$

(1.5 points) Consider the corresponding ground truth,

$$y = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Calculate the binary cross-entropy with respect to the natural logarithm by summing over all output pixels of the forward pass computed in (a). You may assume $\log(0) \approx -10^9$. (Write down the **equation** and keep the logarithm for the final result.)

(c) (0.5 points) You don't recall learning the formula for backpropagation through convolutional layers but those 1×1 convolutions seem suspicious. Write down the name of a common layer that is able to produce same result as the convolutional layer used above.

Audience Questions

See you next week ☺