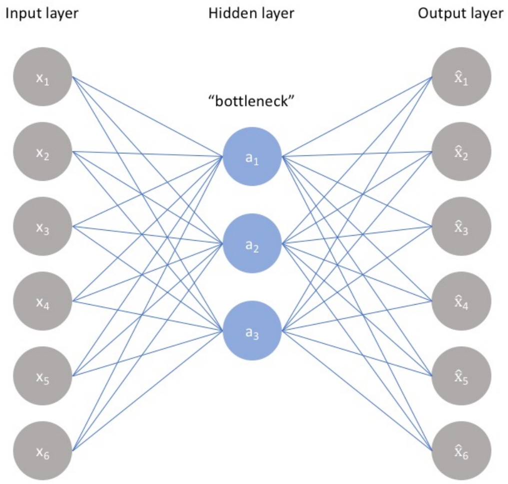
Introduction to Deep Learning (I2DL)

Exercise 8: Autoencoder

Today's Outline

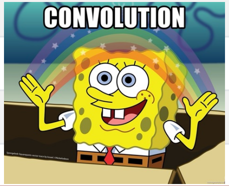
- Exercise 07: Example Solutions
- Exercise 08
  - Batch Normalization & Dropout
  - Transfer Learning
  - Autoencoder



Exercise 7: Solutions

Leaderboard: Ex7

#	User	Score
1	u0139	78.71
2	u1574	77.34
3	u0342	72.76
4	u0797	66.50
5	u1605	60.18
6	u0798	59.78
7	u0337	58.70
8	u1313	58.43
9	u1652	56.49
10	u0071	56.09
11	u0078	56.04
12	u0943	55.55
13	u1469	54.36



Solution 1: 60,18%

```python
self.model = nn.Sequential(
    nn.Linear(self.hparams["input_size"], self.hparams["nn_hidden_Layer1"]),
    nn.ReLU(),
    nn.Linear(self.hparams["nn_hidden_Layer1"], self.hparams["num_classes"]),
    nn.ReLU()
    )
```

Manual Transforms:

- Gaussian filter
- Rotation
- etc

```python
my_transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize(mean, std)])
```

```python
# Note: you can change the splits if you want :)
split = {
    'train': 0.9,
    'val': 0.05,
    'test': 0.05
}
split_values = [v for k,v in split.items()]
assert sum(split_values) == 1.0
```

```python
def configure_optimizers(self):

    optim = None
    ########################################################################
    # TODO: Define your optimizer.                                         #
    ########################################################################

    optim = torch.optim.Adam(self.model.parameters(), self.hparams["learning_rate"], weight_decay=self.hparams['weight_decay'])
    StepLR = torch.optim.lr_scheduler.MultiStepLR(optim, milestones=[30],gamma=0.5)

    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
    return optim
```

```python
hparams["loading_method"] = 'Memory'
hparams['num_workers'] = 1
hparams['input_size'] = 3 * 32 * 32
hparams['batch_size'] = 1000
hparams['learning_rate'] = 5e-5
hparams['weight_decay'] = 1e-3
hparams['nn_hidden_Layer1'] = 1500
hparams['num_classes'] = 10
```

Manual Initialization: Kaiming with ReLU

Solution 2: 59,78%

```python
self.model = nn.Sequential(
    nn.Linear(hparams["input_size"], hparams["hidden_size"][0]),
    nn.BatchNorm1d(hparams["hidden_size"][0]),
    nn.ReLU(),
    nn.Dropout(),
    nn.Linear(hparams["hidden_size"][0], hparams["hidden_size"][1]),
    nn.BatchNorm1d(hparams["hidden_size"][1]),
    nn.ReLU(),
    nn.Dropout(),
    nn.Linear(hparams["hidden_size"][1], hparams["num_classes"])
)
```

```python
my_transform = transforms.Compose([
    transforms.AutoAugment(transforms.AutoAugmentPolicy.CIFAR10),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)])
```

```python
# Note: you can change the splits if you want :)
split = {
    'train': 0.9,
    'val': 0.05,
    'test': 0.05
}
split_values = [v for k,v in split.items()]
assert sum(split_values) == 1.0
```

```python
def configure_optimizers(self):

    optim = None
    ########################################################################
    # TODO: Define your optimizer.                                         #
    ########################################################################

    optim = torch.optim.SGD(MyPytorchModel.parameters(self), lr=self.hparams["learning_rate"], momentum=0.9)

    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
    return optim
```

```python
hparams = {
    "learning_rate": 0.3,
    "input_size": 3 * 32 * 32,
    "batch_size": 512,
    "hidden_size": [1332, 666],
    "num_classes": 10,
    "num_workers": 8,
    "loading_method": "Memory"
}
```

Solution 3: 58,43%

```python
self.model = nn.Sequential(
    nn.Linear(self.hparams["input_size"], self.hparams["hidden_size"]),
    nn.ReLU(),
    nn.Linear(self.hparams["hidden_size"], self.hparams["hidden_size"]),
    nn.ReLU(),
    nn.Linear(self.hparams["hidden_size"], self.hparams["hidden_size"]),
    nn.ReLU(),
    nn.Linear(self.hparams["hidden_size"], self.hparams["num_classes"])
)
```

```python
# Note: you can change the splits if you want :)
if 'split' not in self.opt.keys():
    split = {
        'train': 0.6,
        'val': 0.2,
        'test': 0.2
    }
else:
    split = self.opt['split']

split_values = [v for k,v in split.items()]
assert sum(split_values) == 1.0
```

```python
# Init weights with kaiming init
nn.init.kaiming_normal_(self.model[0].weight, nonlinearity='relu')
nn.init.kaiming_normal_(self.model[2].weight, nonlinearity='relu')
nn.init.kaiming_normal_(self.model[4].weight, nonlinearity='relu')
```

```python
my_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean, std),
    transforms.RandomHorizontalFlip(0.5),
    transforms.GaussianBlur(5, (0.01, 2))])
```

```python
def configure_optimizers(self):

    optim = None
    ########################################################################
    # TODO: Define your optimizer.                                         #
    ########################################################################


    optim = torch.optim.Adam(self.model.parameters(), lr=self.hparams["learning_rate"],
        weight_decay=self.hparams['weight_decay'])

    lr_scheduler =   lr_scheduler = {'scheduler': ReduceLROnPlateau(optimizer=optim, mode='min', factor=self.hparams["lr_decay"], patience=2),
        'monitor': 'loss' }
    ########################################################################
    #                             END OF YOUR CODE                         #
    ########################################################################
    return {"optimizer": optim, "lr_scheduler": lr_scheduler}
```

```python
hparams = {
    # Model
    "input_size": 3 * 32 * 32,
    "num_classes": 10,
    "hidden_size": 320,
    # Dataloader
    "loading_method": 'Memory',
    "batch_size": 64,
    "num_workers": 2,
    "split": {
        'train': 0.8,
        'val': 0.1,
        'test': 0.1,
    },
    # Optimizer
    "learning_rate": 5e-04,
    "lr_decay": 0.5, # ReduceLROnPlateau with val_loss monitor
    "weight_decay": 1.15e-03, # L2 regularization
}
```

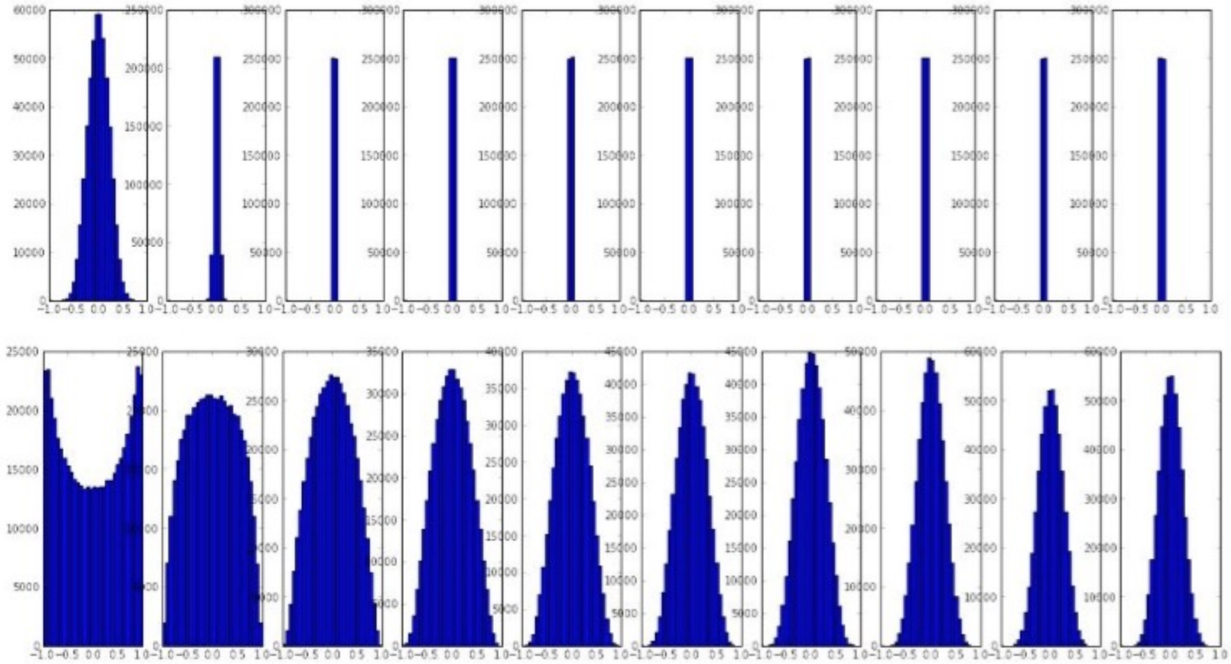
Summary

- Network: Linear + ReLU (Depth: 2-4)
- Initialization of Network Weights
- Optimizer: SGD or Adam, LR Scheduler
- Data Augmentation

Improve your training!

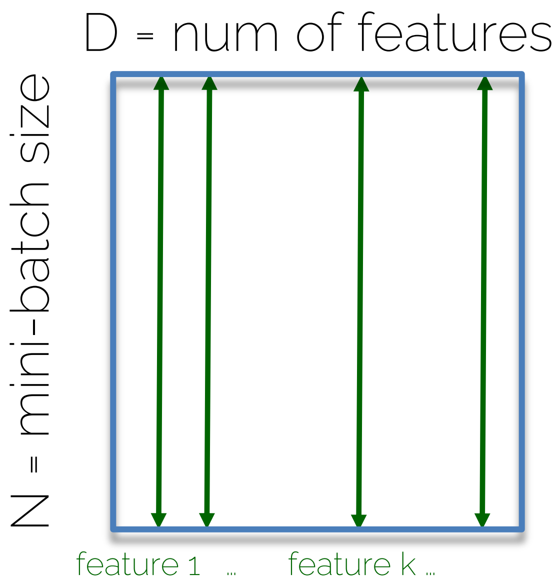
Batch Normalization

- All we want is that our activations do not die out



Batch Normalization

- **Wish: Unit Gaussian activations**



Mean of your mini-batch examples over feature k

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$

Unit gaussian

Batch Normalization

- 1. Normalize

$$\hat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - E[\boldsymbol{x}^{(k)}]}{\sqrt{Var[\boldsymbol{x}^{(k)}]}}$$

- 2. Allow the network to change the range

$$\boldsymbol{y}^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

backprop

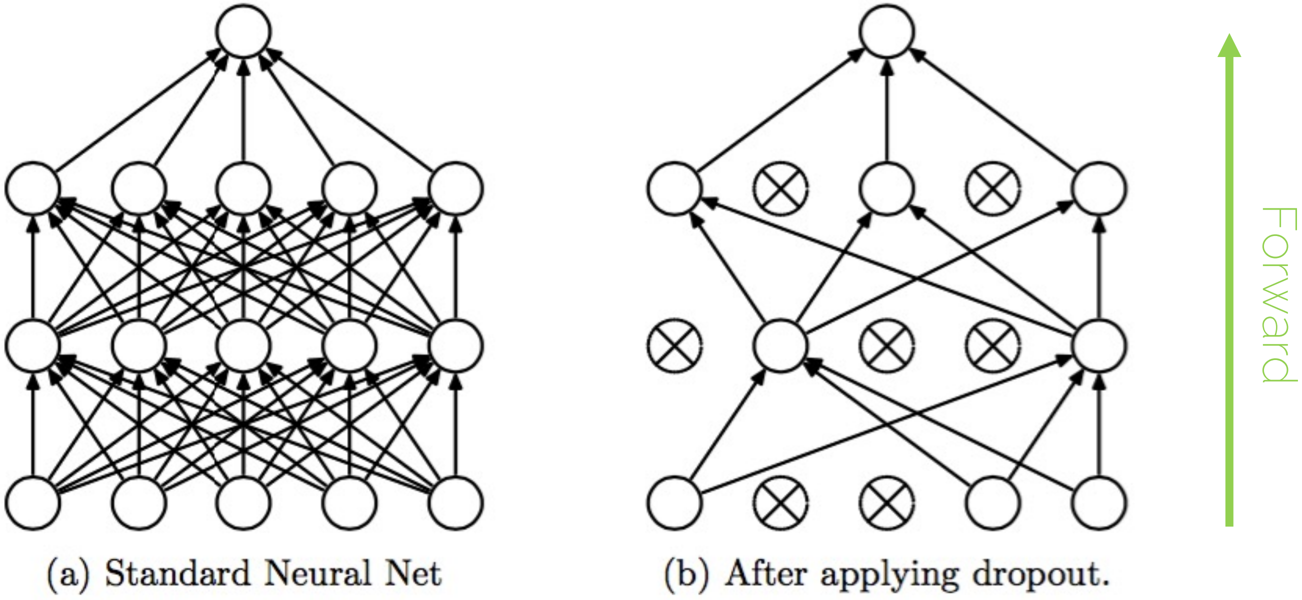
The network can learn to undo the normalization

$$\gamma^{(k)} = \sqrt{Var[\boldsymbol{x}^{(k)}]}$$

$$\beta^{(k)} = E[\boldsymbol{x}^{(k)}]$$

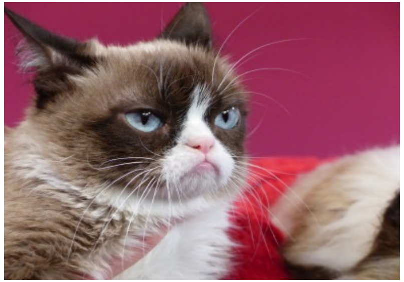
Dropout

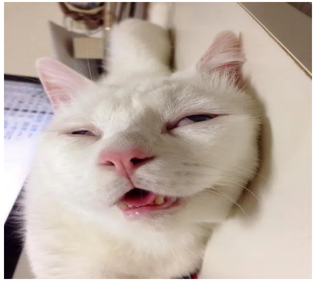
- Using half the network = half capacity



Transfer Learning

Transfer Learning: Example Scenario



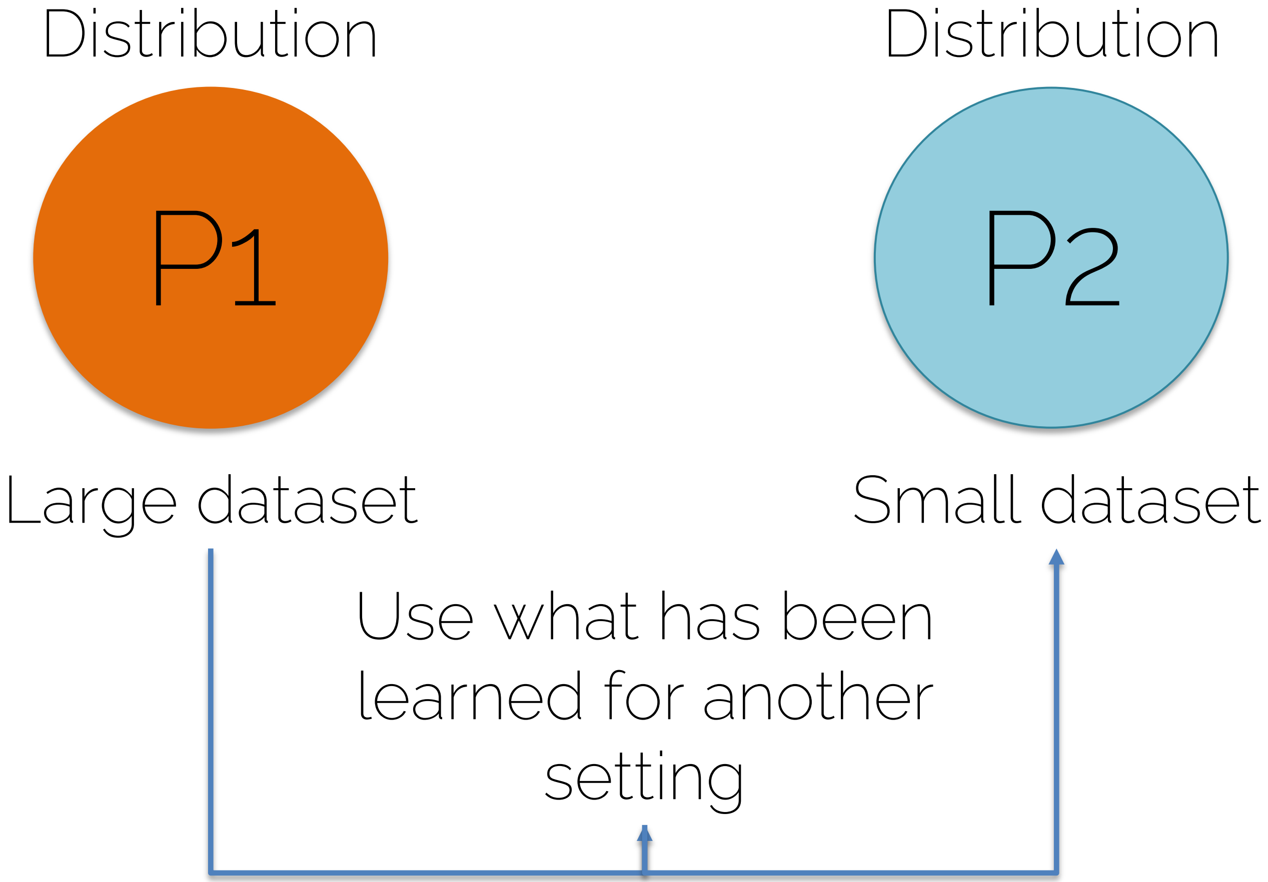
- Need to build a Cat classifier
- Only have a few images ~10 000

Transfer Learning

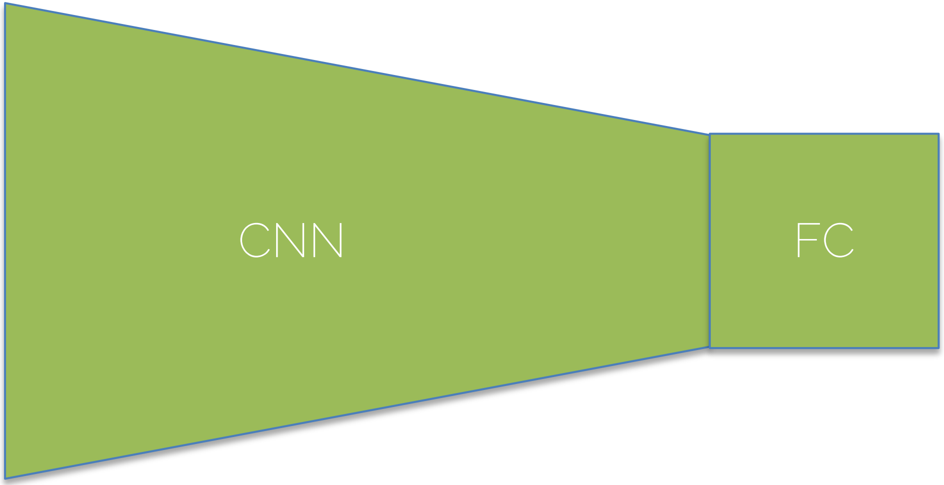
- Problem Statement:
  – Training a Deep Neural Network needs a lot of data
  – Collecting much data is expensive or just not possible

- Idea:
  – Some problems/ tasks are closely related
  – Can we transfer knowledge from one task to another?
  – Can we re-use (at least parts of) a pre-trained network for the new task?

Transfer Learning



Transfer Learning



2 classes

Feature Extractor

Classifier

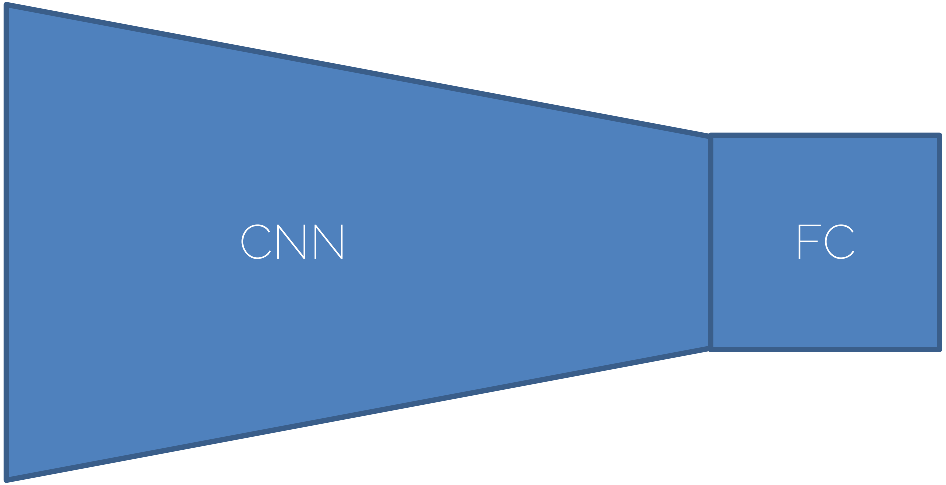
Trainable Parameters

Coloring Legend:

Untrained

Trained

Transfer Learning

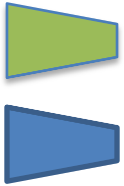


1000 classes

Feature Extractor

Classifier

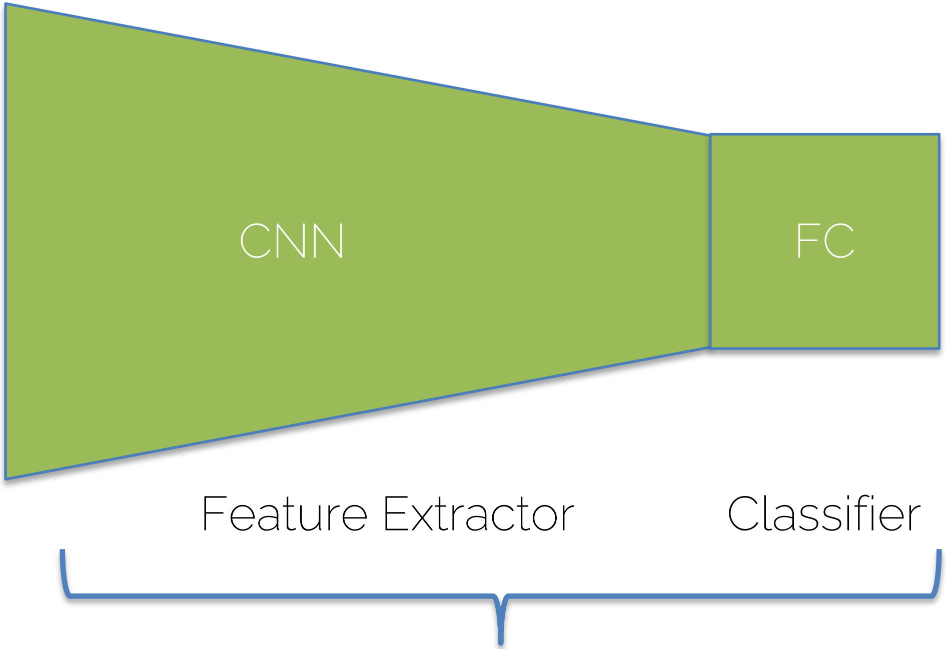
Coloring Legend:



Untrained

Trained

Transfer Learning



2 classes

Feature Extractor

Classifier

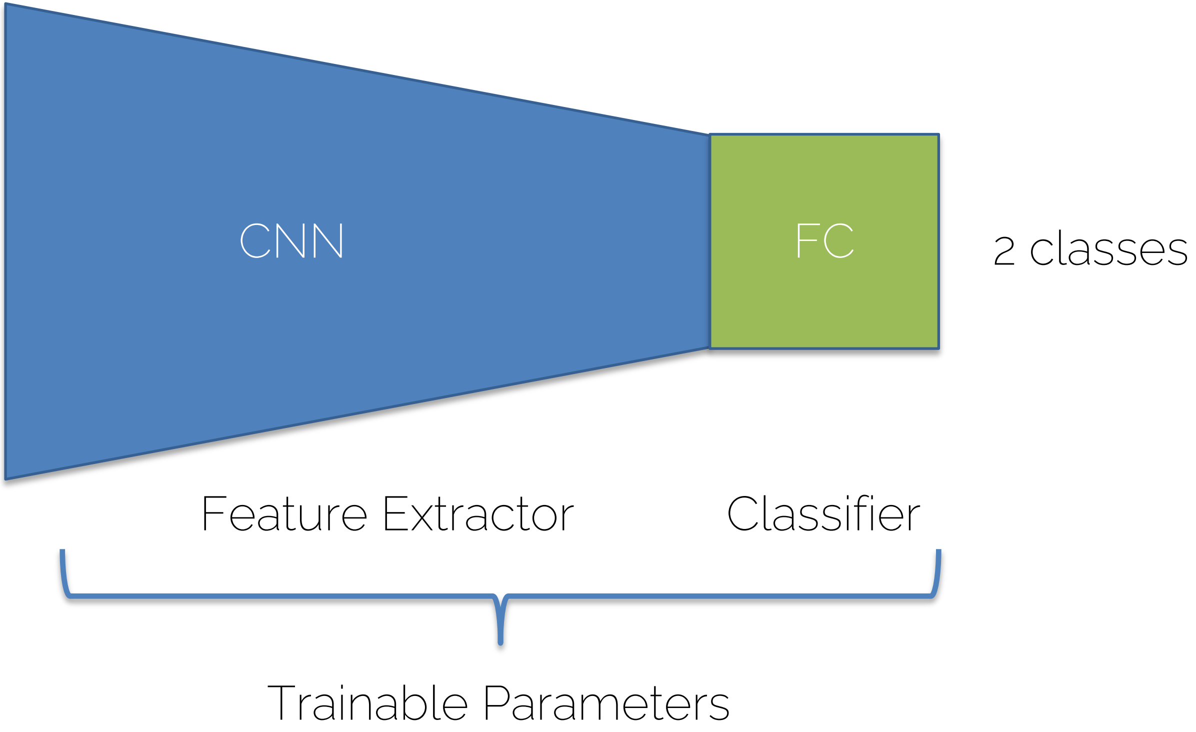
Trainable Parameters

Coloring Legend:

Untrained

Trained

Transfer Learning

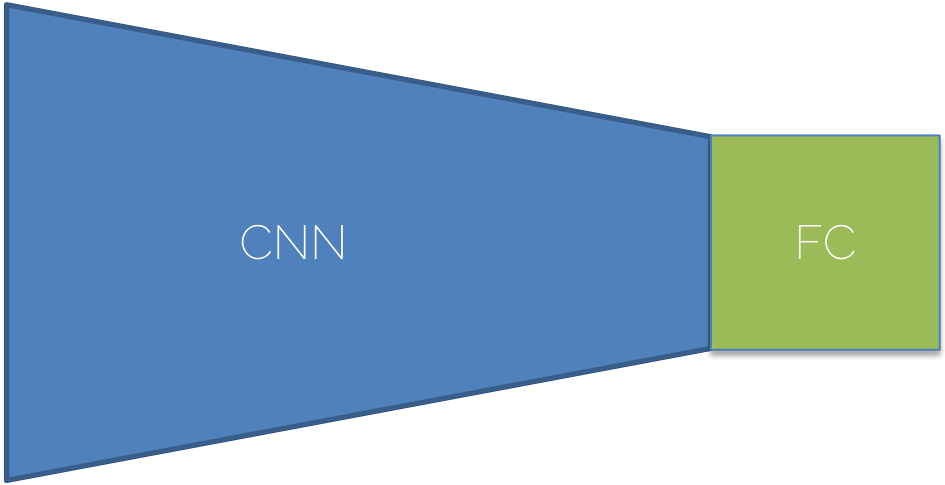


Coloring Legend:

Untrained

Trained

Transfer Learning



2 classes

Feature Extractor

Classifier

Maybe freeze weights/
slower learning rate/
nothing special

Newly initialized
head

Coloring Legend:

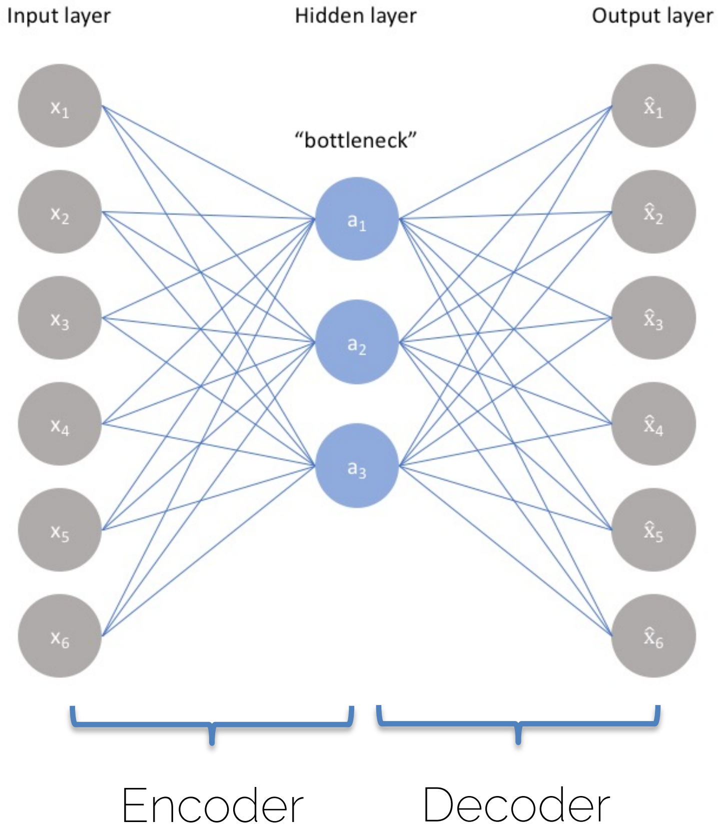
Untrained

Trained

Application: Autoencoder

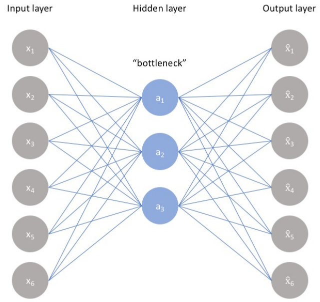
Autoencoder

- Task
  - Reconstruct the input given a lower dimensional bottleneck
  - Loss: L1/L2 per pixel
- Actually need no labels!
- Without non-linearities: similar to PCA

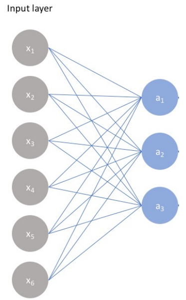


Transfer Using an Autoencoder

- Step 1:
  - Train an Autoencoder on a large (maybe unlabelled) dataset very similar to your target dataset



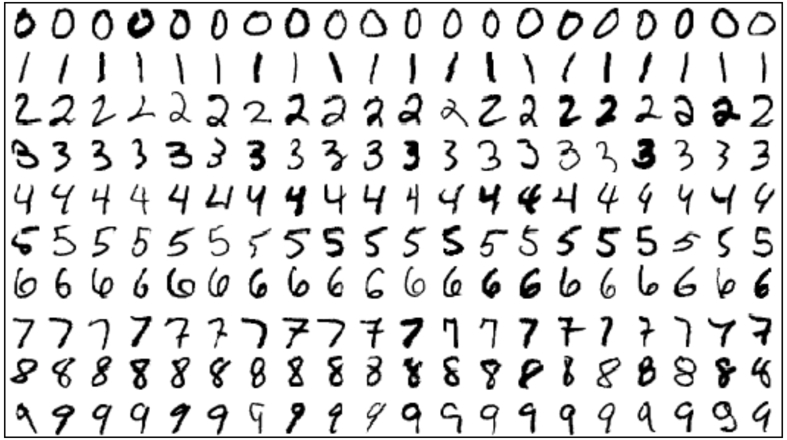
- Step 2:
  - Take pre-trained Autoencoder and use it as the first part of a classification architecture for your target dataset

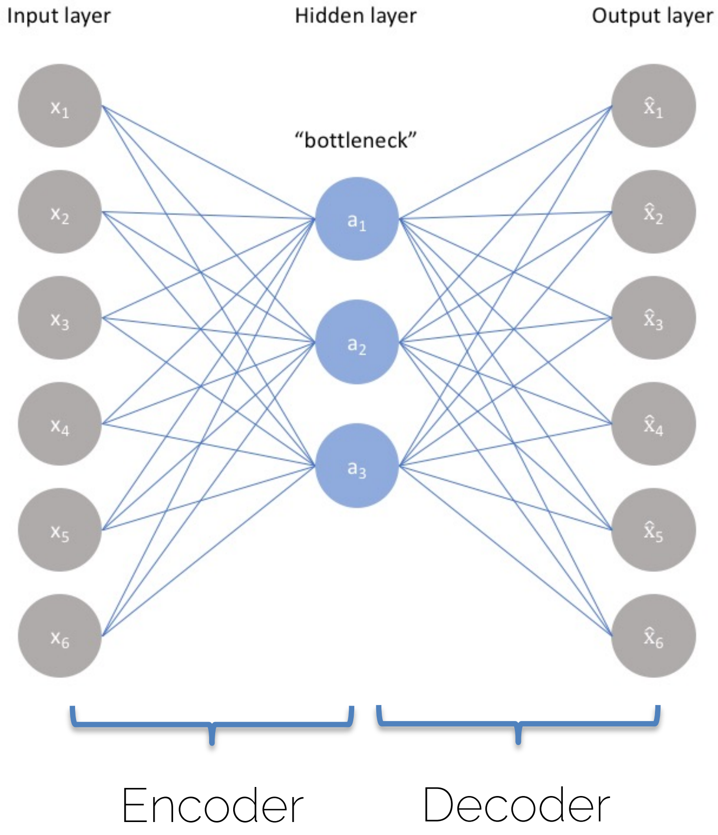


Exercise 8

Autoencoder

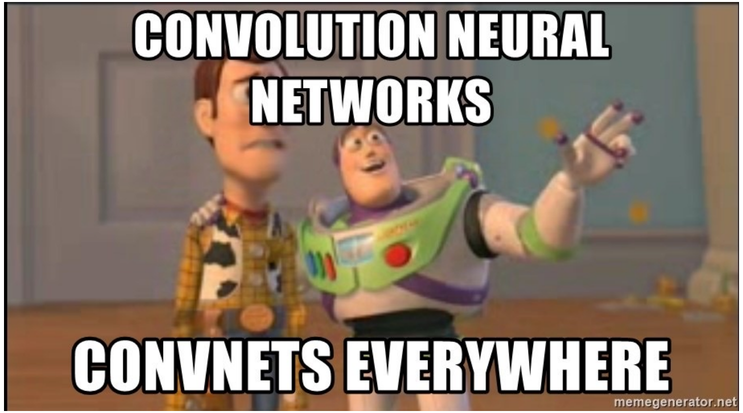
- Exercise Task:
  - 60 000 Images
  - Only 300 with labels





We get there…

No convolutions yet,
but be prepared…



Next week will be the week.

But that means for now, we stick (one last time) with our linear layers.

Summary

- Monday 04.07.22: Watch Lecture 9
  - Convolutional Neural Networks
- Monday 04.07.22: Exercise 8 Submission
  - Autoencoder: 04.07.2022 23.59
- Tuesday 05.07: Tutorial Session (On-Site Q&A)
  - Ex09: Facial Keypoint Detection

See you next week!