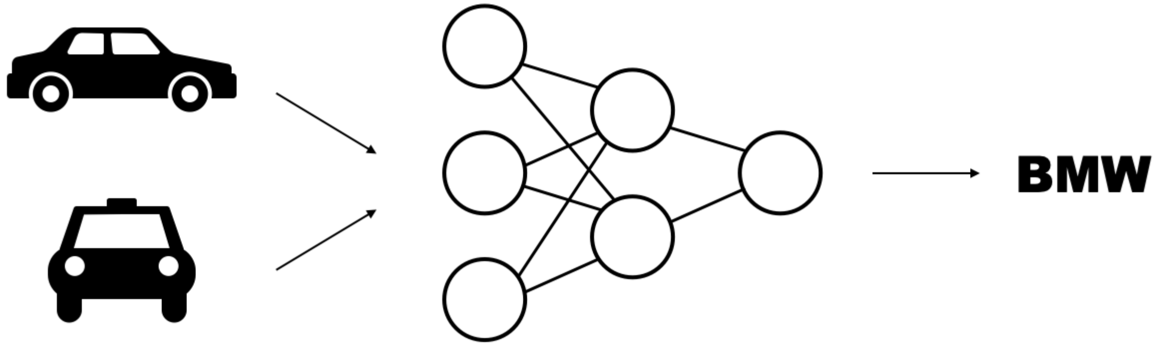
Multi-input models

INTERMEDIATE DEEP LEARNING WITH PYTORCH

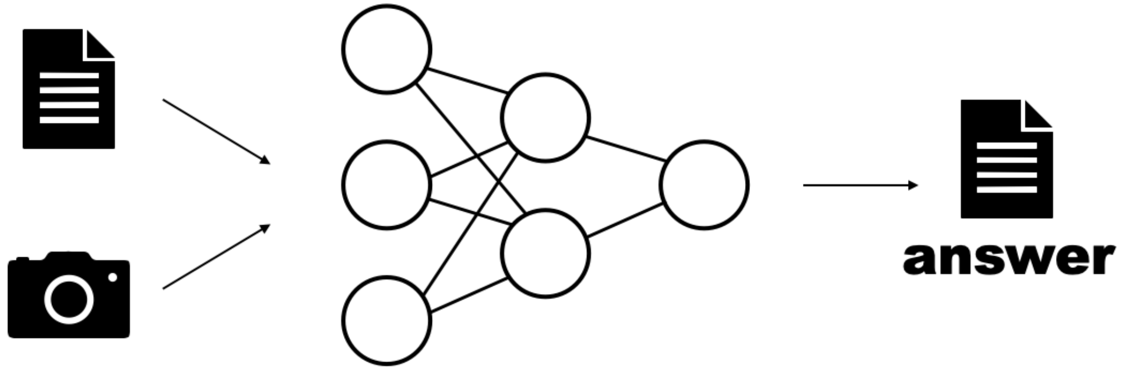
Michal Oleszak
Machine Learning Engineer

Why multi-input?

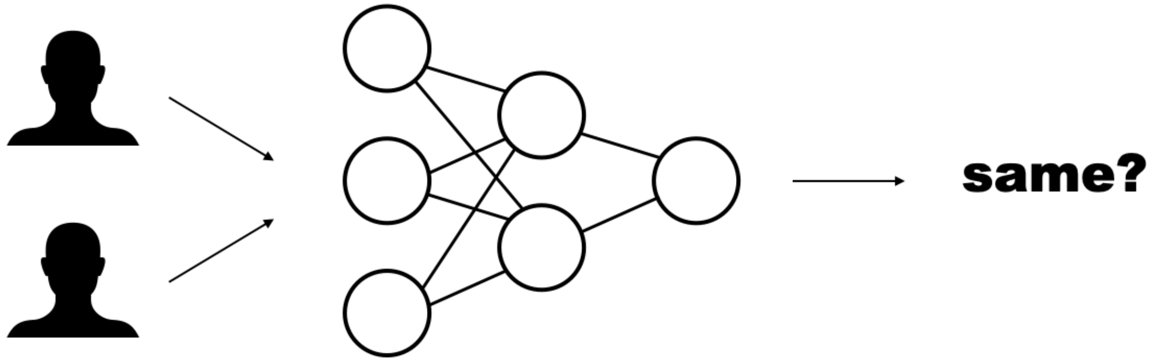
Using more information



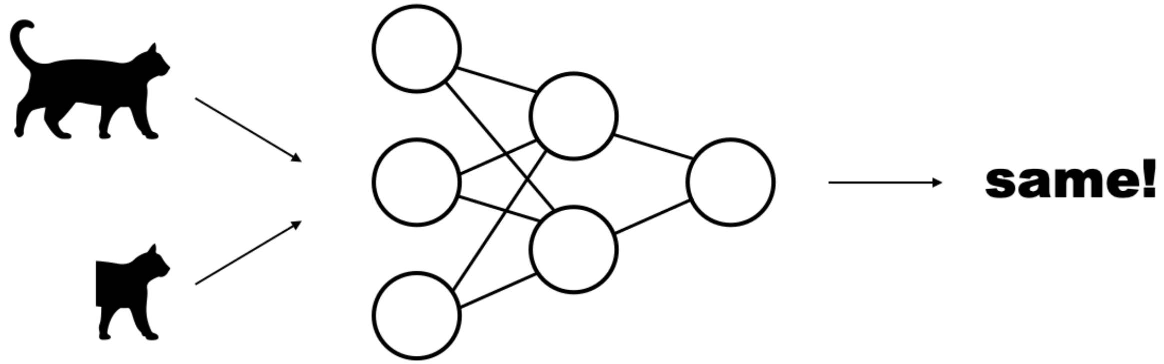
Multi-modal models



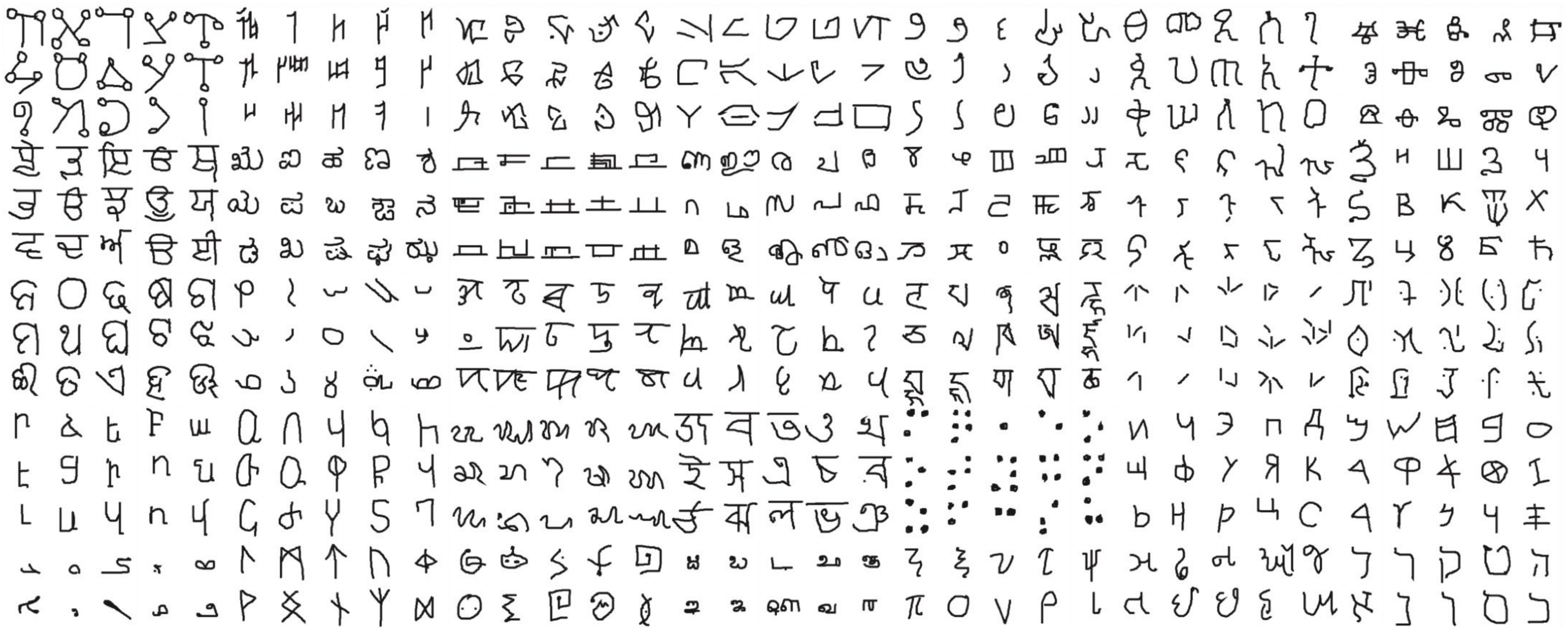
Metric learning



Self-supervised learning

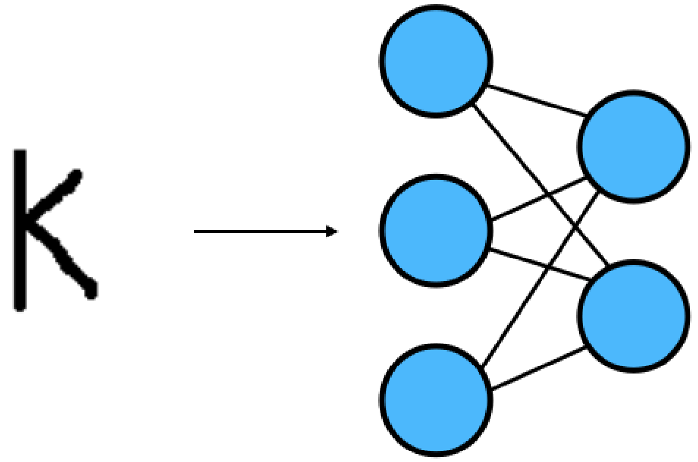


Omniglot dataset

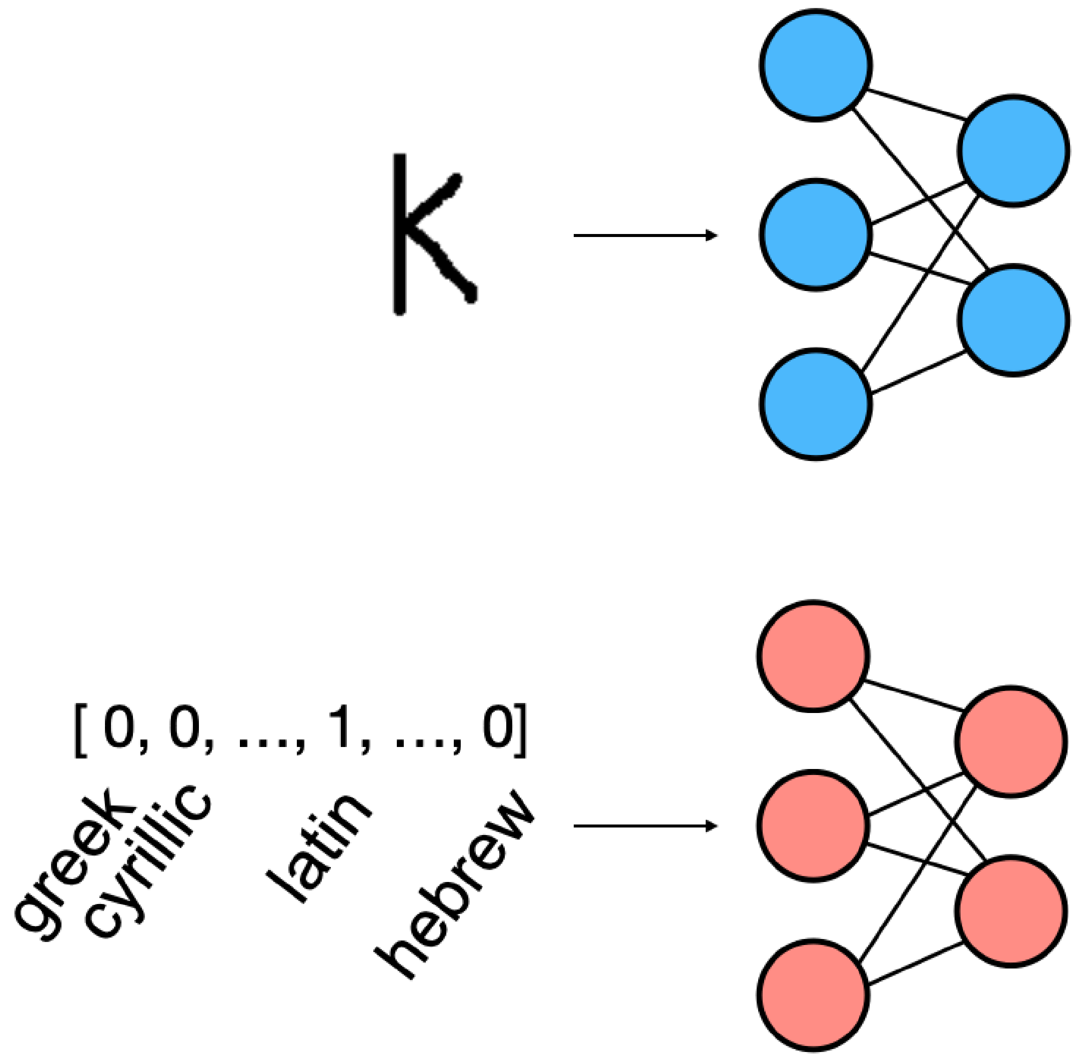


[1] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. Science, 350(6266), 1332-1338.

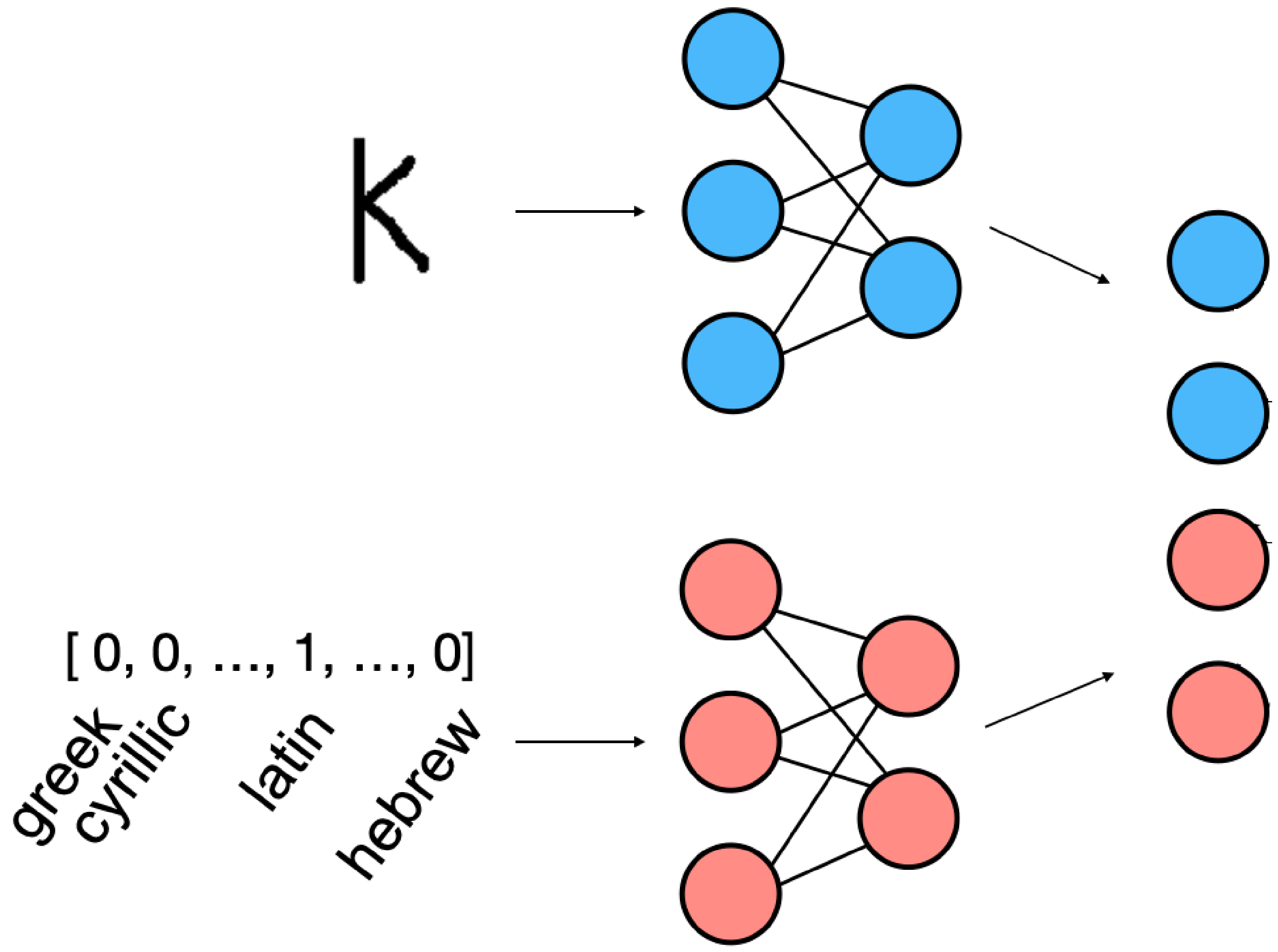
Character classification



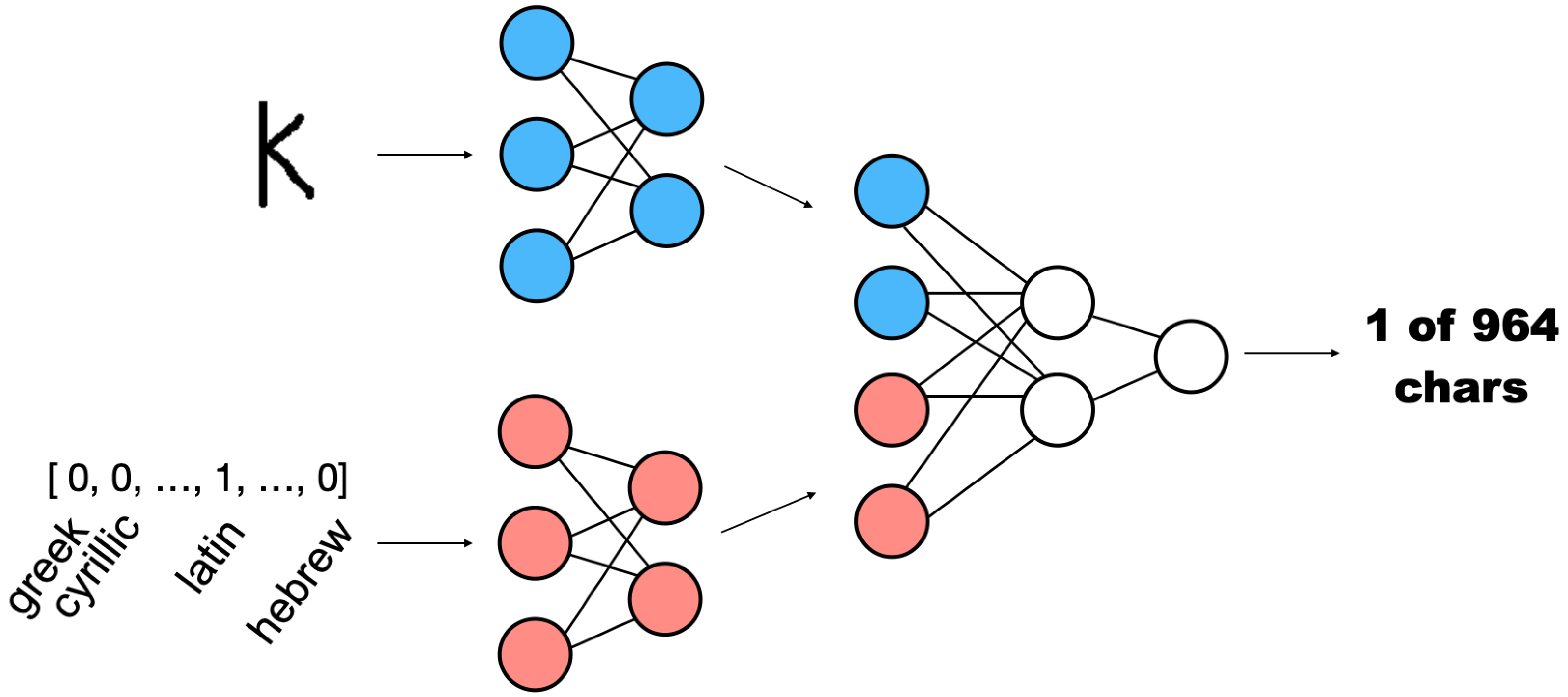
Character classification



Character classification



Character classification



Two-input Dataset

```python
from PIL import Image

class OmniglotDataset(Dataset):
    def __init__(self, transform, samples):
        self.transform = transform
        self.samples = samples

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        img_path, alphabet, label = self.samples[idx]
        img = Image.open(img_path).convert('L')
        img = self.transform(img)
        return img, alphabet, label
```

- Assign samples and transforms

```python
print(samples[0])
```

```
[(
  'omniglot_train/.../0459_14.png',
   array([1., 0., 0., ..., 0., 0., 0.]),
   0
 )]
```

- Implement `__len__()`
- Load and transform image
- Return both inputs and label

Tensor concatenation

```
x = torch.tensor([
  [1, 2, 3],
])

y = torch.tensor([
  [4, 5, 6],
])
```

Concatenation along axis 0

```
torch.cat((x, y), dim=0)
```

```
[[1, 2, 3],
 [4, 5, 6]]
```

Concatenation along axis 1

```
torch.cat((x, y), dim=1)
```

```
[[1, 2, 3, 4, 5, 6]]
```

Two-input architecture

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.image_layer = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),
            nn.MaxPool2d(kernel_size=2),
            nn.ELU(),
            nn.Flatten(),
            nn.Linear(16*32*32, 128)
        )
        self.alphabet_layer = nn.Sequential(
            nn.Linear(30, 8),
            nn.ELU(),
        )
        self.classifier = nn.Sequential(
            nn.Linear(128 + 8, 964),
        )
```

- Define image processing layer
- Define alphabet processing layer
- Define classifier layer

Two-input architecture

```python
def forward(self, x_image, x_alphabet):
    x_image = self.image_layer(x_image)
    x_alphabet = self.alphabet_layer(x_alphabet)
    x = torch.cat((x_image, x_alphabet), dim=1)
    return self.classifier(x)
```

- Pass image through image layer
- Pass alphabet through alphabet layer
- Concatenate image and alphabet outputs
- Pass the result through classifier

Training loop

```
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01)

for epoch in range(10):
    for img, alpha, labels in dataloader_train:
        optimizer.zero_grad()
        outputs = net(img, alpha)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

- Training data consists of three items:
  - Image
  - Alphabet vector
  - Labels
- We pass the model images and alphabets

Let's practice!

INTERMEDIATE DEEP LEARNING WITH PYTORCH

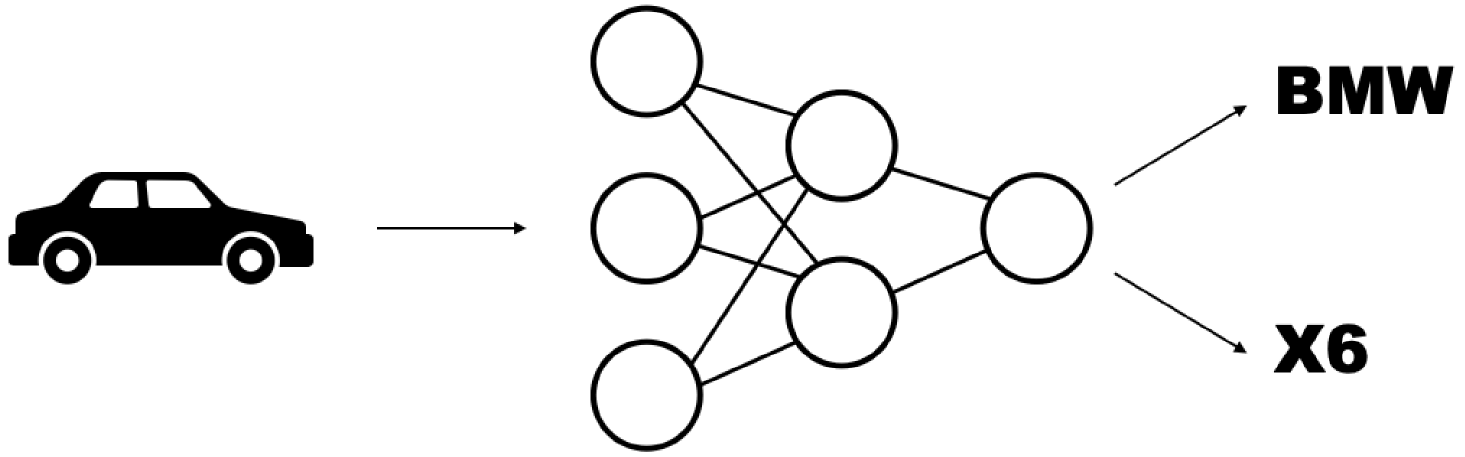
Multi-output models

INTERMEDIATE DEEP LEARNING WITH PYTORCH

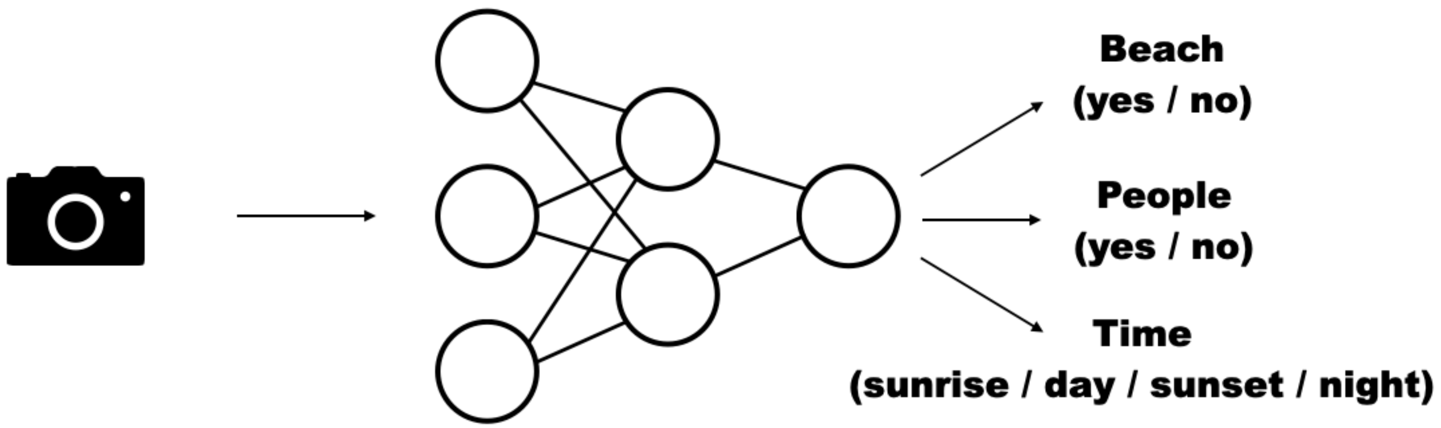
Michal Oleszak
Machine Learning Engineer

Why multi-output?

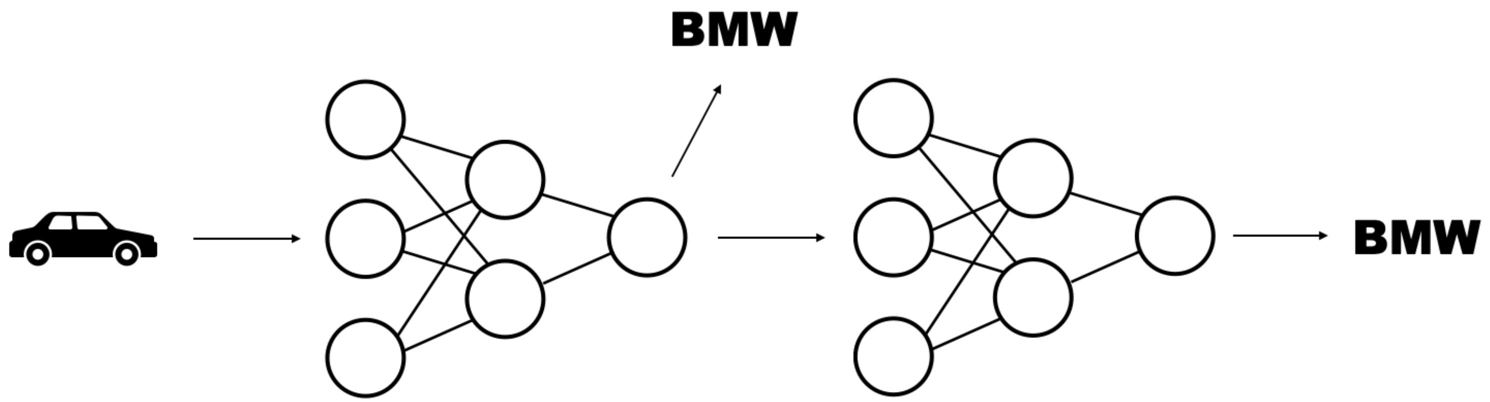
Multi-task learning



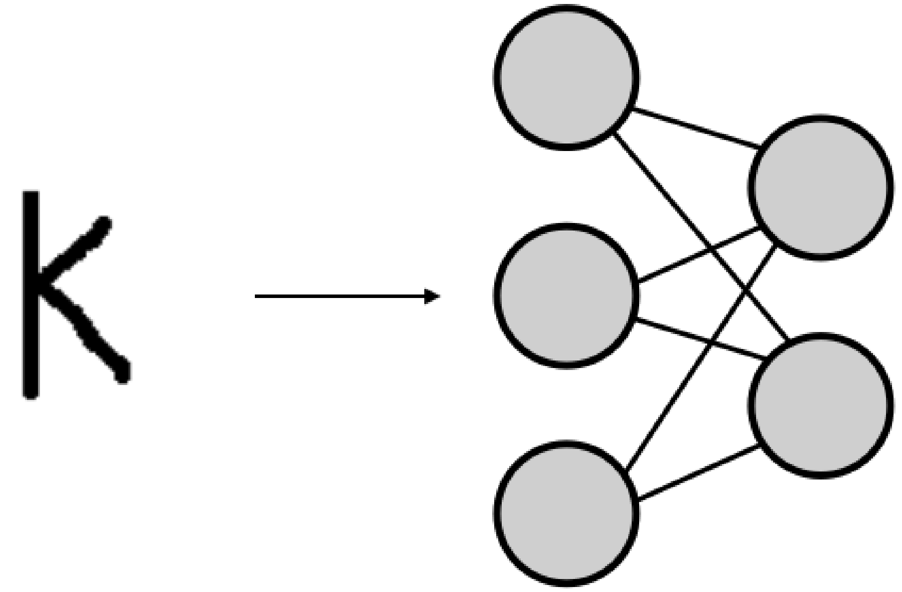
Multi-label classification



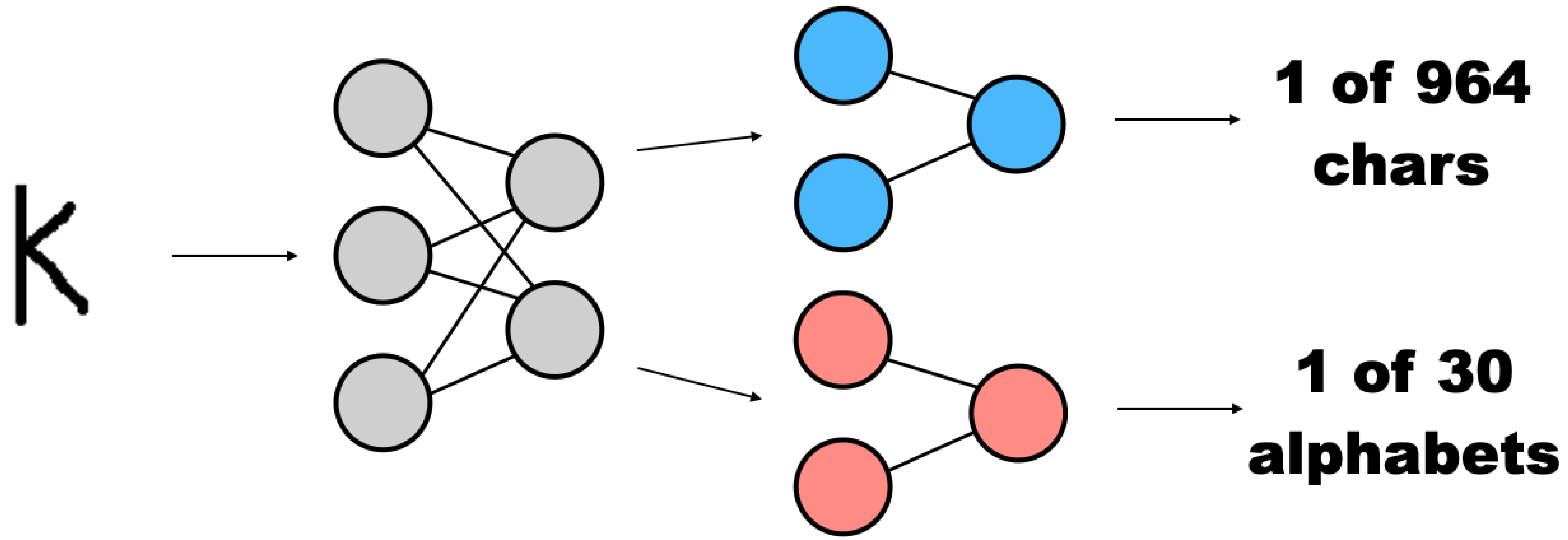
Regularization



Character and alphabet classification



Character and alphabet classification



Two-output Dataset

```
class OmniglotDataset(Dataset):
    def __init__(self, transform, samples):
        self.transform = transform
        self.samples = samples

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        img_path, alphabet, label = \
            self.samples[idx]
        img = Image.open(img_path).convert('L')
        img = self.transform(img)
        return img, alphabet, label
```

- We can use the same Dataset...
- ...with updated samples:

```
print(samples[0])
```

```
[(
  'omniglot_train/.../0459_14.png',
   0,
   0,
 )]
```

Two-output architecture

```
class Net(nn.Module):
    def __init__(self, num_alpha, num_char):
        super().__init__()
        self.image_layer = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),
            nn.MaxPool2d(kernel_size=2),
            nn.ELU(),
            nn.Flatten(),
            nn.Linear(16*32*32, 128)
        )
        self.classifier_alpha = nn.Linear(128, 30)
        self.classifier_char = nn.Linear(128, 964)

    def forward(self, x):
        x_image = self.image_layer(x)
        output_alpha = self.classifier_alpha(x_image)
        output_char = self.classifier_char(x_image)
        return output_alpha, output_char
```

- Define image-processing sub-network
- Define output-specific classifiers
- Pass image through dedicated sub-network
- Pass the result through each output layer
- Return both outputs

Training loop

```
for epoch in range(10):
    for images, labels_alpha, labels_char \
    in dataloader_train:
        optimizer.zero_grad()
        outputs_alpha, outputs_char = net(images)
        loss_alpha = criterion(
          outputs_alpha, labels_alpha
        )
        loss_char = criterion(
          outputs_char, labels_char
        )
        loss = loss_alpha + loss_char
        loss.backward()
        optimizer.step()
```

- Model produces two outputs
- Calculate loss for each output
- Combine the losses to one total loss
- Backprop and optimize with the total loss

Let's practice!

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Evaluation of multi-output models and loss weighting

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

Model evaluation

```
acc_alpha = Accuracy(
    task="multiclass", num_classes=30
)
acc_char = Accuracy(
    task="multiclass", num_classes=964
)

net.eval()
with torch.no_grad():
    for images, labels_alpha, labels_char \
    in dataloader_test:
        out_alpha, out_char = net(images)
        _, pred_alpha = torch.max(out_alpha, 1)
        _, pred_char = torch.max(out_char, 1)
        acc_alpha(pred_alpha, labels_alpha)
        acc_char(pred_char, labels_char)
```

- Set up metric for each output
- Iterate over test loader and get outputs
- Calculate prediction for each output
- Update accuracy metrics
- Calculate final accuracy scores

```
print(f"Alphabet: {acc_alpha.compute()}")
print(f"Character: {acc_char.compute()}")
```

```
Alphabet: 0.3166305720806122
Character: 0.24064336717128754
```

Multi-output training loop revisited

```
for epoch in range(10):
    for images, labels_alpha, labels_char \
    in dataloader_train:
        optimizer.zero_grad()
        outputs_alpha, outputs_char = net(images)
        loss_alpha = criterion(
          outputs_alpha, labels_alpha
        )
        loss_char = criterion(
          outputs_char, labels_char
        )
        loss = loss_alpha + loss_char
        loss.backward()
        optimizer.step()
```

- Two losses: for alphabets and characters
- Final loss defined as sum of alphabet and character losses: `loss = loss_alpha + loss_char`
- Both classification tasks deemed equally important

Varying task importance

Character classification 2 times more important than alphabet classification

- Approach 1: Scale more important loss by a factor of 2

```
loss = loss_alpha + loss_char * 2
```

- Approach 2: Assign weights that sum to 1

```
loss = 0.33 * loss_alpha + 0.67 * loss_char
```

Warning: losses on different scales

- Losses must be on the same scale before they are weighted and added
- Example tasks:
  - Predict house price -> MSE loss
  - Predict quality: low, medium, high -> CrossEntropy loss
- CrossEntropy is typically in the single-digits
- MSE loss can reach tens of thousands
- Model would ignore quality assessment task
- Solution: Normalize both losses before weighting and adding

```
loss_price = loss_price / torch.max(loss_price)
loss_quality = loss_quality / torch.max(loss_quality)
loss = 0.7 * loss_price + 0.3 * loss_quality
```

Let's practice!

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Wrap-up

INTERMEDIATE DEEP LEARNING WITH PYTORCH

Michal Oleszak
Machine Learning Engineer

What you learned

1. Training robust neural networks

- PyTorch and OOP
- Optimizers
- Vanishing and exploding gradients

2. Images and convolutional neural networks

- Handling images with PyTorch
- Training and evaluating convolutional networks
- Data augmentation

3. Sequences and recurrent neural networks

- Handling sequences with PyTorch
- Training and evaluating recurrent networks (LSTM and GRU)

4. Multi-input and multi-output architectures

- Multi-input models
- Multi-output models
- Loss weighting

What's next?

What you might consider learning next:

- Transformers
- Self-supervised learning

Courses:

- Deep Learning for Text with PyTorch
- Deep Learning for Images with PyTorch

Congratulations and good luck!

INTERMEDIATE DEEP LEARNING WITH PYTORCH