Handwriting Recognition

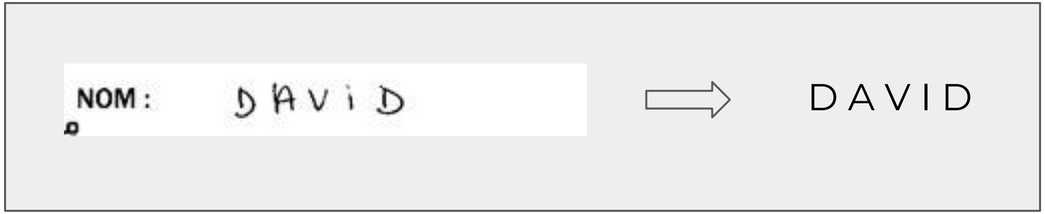
Deep Learning Final Project

Amee Tan & Evan Chen

Agenda

1. Recap of Project
2. Results
3. Failures / Methods Attempted
4. Successes
5. Future Directions

Project Recap



331,059 training images

41,382 validation images

Handwriting Recognition using CRNN in Keras

Python notebook using data from Handwriting Recognition · 10,602 views · 1y ago · gpu

Pytorch

Model

1

CNN

3 conv layers

2

RNN

Bidirectional LSTM with 2 layers

3

Loss Function

CTC Loss

Scoring Metrics

Correct Characters Predicted

On average, how many characters did we get correct in each name?

Prediction	**A**	**I**	**E**	**K**
Actual	**A**	**L**	**E**	**X**

50%

Correct Words Predicted

How many of them did we correctly predict?

Actual	Prediction
JOSE	JOSB
ANNIE	ANNIE
MARCOS	NAKOOS
MARIANNA	MARLANNA

1 **out of 4**

How did we do?

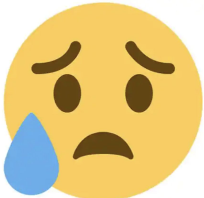
Correct characters predicted

20%

Correct words predicted

5

out of 41,382



```
Correct characters predicted : 82.16%
Correct words predicted      : 69.10%
```

= 28,553 correct words

Failures / Methods Attempted

- Could not successfully load data into Google Collab without timing out
    - Trained in Kaggle instead
- Kaggle GPUs timed out when running overnight

Failures / Methods Attempted

- Could not successfully load data into Google Collab without timing out
    - Trained in Kaggle instead
- Kaggle GPUs timed out when running overnight

```
Epoch:  14
Train:
CTC Loss: 13.4237
Percent correct characters per word: 0.1979
Number of correct words: 5
Valid
CTC Loss 13.0708
Percent correct characters per word 0.2007
Number of correct words 1
```

```
Epoch 14/60
235/235 [==============================]
loss: 5.6524 - val_loss: 5.2620
```

Failures/Methods Attempted

- Negative CTC loss values
    - Take LogSoftMax of predictions before feeding into CTC loss
- NaNs for loss when training on entire dataset

```
Epoch:  0
/opt/conda/lib/python3.7/site-packages/ipyker
ated. Change the call to include dim=X as an
Train:
CTC Loss: nan
Percent correct characters per word: 0.082
Number of correct words: 10
Valid
CTC Loss nan
Percent correct characters per word 0.1246
Number of correct words 5
```

```
train_loss = one_pass(model, dl_train, optimizer)
train_loss
```

```
tensor(-5.0396, grad_fn=<MeanBackward0>)
tensor(-4.1961, grad_fn=<MeanBackward0>)
tensor(6.1593, grad_fn=<MeanBackward0>)
tensor(22.7922, grad_fn=<MeanBackward0>)
tensor(-6.9500, grad_fn=<MeanBackward0>)
tensor(18.8461, grad_fn=<MeanBackward0>)
tensor(21.2995, grad_fn=<MeanBackward0>)
tensor(11.2788, grad_fn=<MeanBackward0>)
tensor(-6.8725, grad_fn=<MeanBackward0>)
tensor(10.3967, grad_fn=<MeanBackward0>)
tensor(20.5246, grad_fn=<MeanBackward0>)
tensor(18.8454, grad_fn=<MeanBackward0>)
tensor(14.8141, grad_fn=<MeanBackward0>)
tensor(6.6474, grad_fn=<MeanBackward0>)
tensor(-1.7920, grad_fn=<MeanBackward0>)
tensor(1.4204, grad_fn=<MeanBackward0>)
Avg loss 8.01088497042656
```

Learning Successes

Batch Size Experiments

	32	64
CTC Loss	24.9271	30.5298
Correct Characters Predicted	6.91%	6.36%
Correct Words Predicted	0	0

Learning Successes

Learning Rate Experiments

	0.01	0.001	Cosine Annealing Scheduler
CTC Loss	37.3579	24.5227	30.3909
Correct Characters Predicted	6.15%	6.91%	7.43%
Correct Words Predicted	0	0	1

Learning Successes

- MiSH Activation function
  - Longer to train vs. ReLU due to derivative calculation when x < 0
  - Better results

	ReLU	MiSH
Correct Characters Predicted	8.8%	20%
Correct Words Predicted	0	5

```python
class Mish(nn.Module):
    def __init__(self):
        super().__init__()
    def forward(self,x):
        return (x*torch.tanh(F.softplus(x)))
```

Keras ➞ Pytorch Translation Guide

Keras	Pytorch
(batch_size, img_length, img_width, channels)	(batch_size, channels, img_length, img_width)
`inner = Conv2D(32, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(input_data)`	`self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1)`
`inner = Dense(num_of_characters, kernel_initializer='he_normal', name='dense2')(inner)`	`self.linear2 = nn.Linear(in_features=1024, out_features=30)`
`inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm1')(inner)` `inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm2')(inner)`	`self.lstm1 = nn.LSTM(input_size=64,hidden_size=512, batch_first=True, bidirectional=True, num_layers=2)`

Future Directions

- Train for more than 15-20 epochs on USF's GPUs
- Revise percent correct metric to be more forgiving
- Deeper dive into what was wrong with our metrics and loss function
- Try using a pre-trained model

NVIDIA DEVELOPER

Text Recognition

Recognizes text from an image.

VIEW MODELS ›

THANKS