Deep Adaptive Learning for Writer Identication based on Single Handwritten Word Images

Pattern Recognition Elsevier, vol.88, pp.64-74

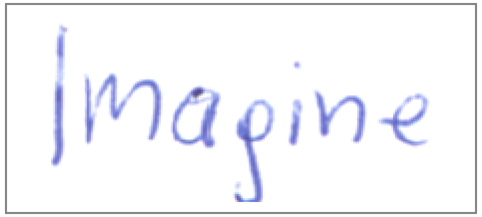
November 2018

6th on Google Schoolar Computer Vision and Pattern Recognition

IF 5.898

Main idea

- Explicit/Implicit information (use as much information as you can)



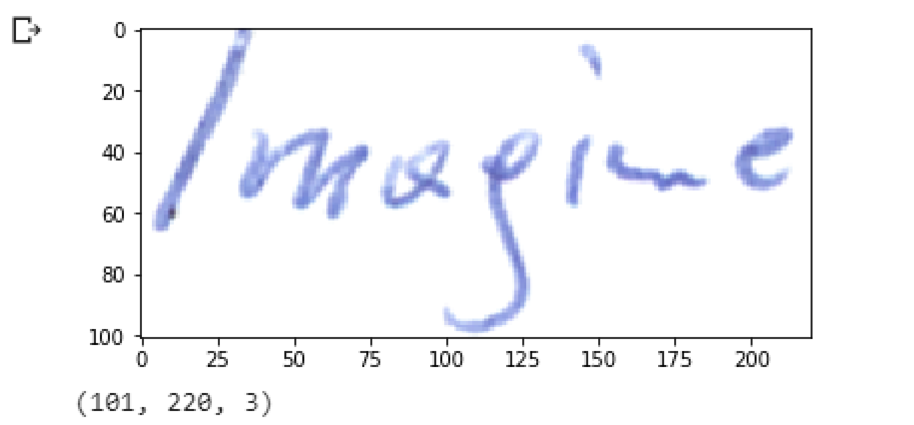
Implicit:
Writer identity: "bob"

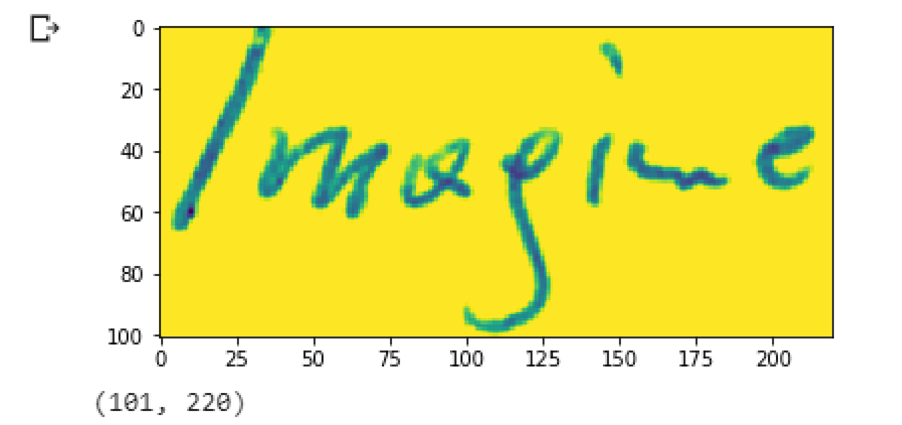
Explicit:
Word content: 'Imagine'
Word length: 7 letters
Character attribute: a, e, g, i, m, n

- Main/Auxiliary task (multi-task learning)
- Regularization?

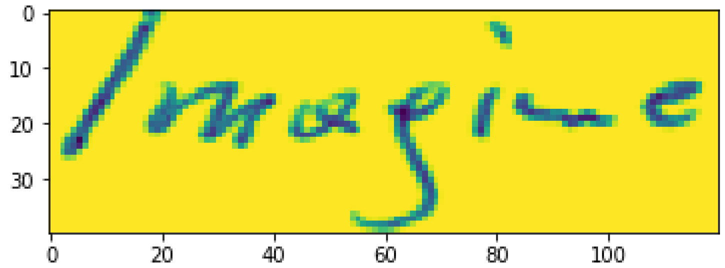
CVL Dataset

- 310 Writers x 5 pages (English/German)
- Document, Line, Word level (two students manually)
- filter(min 20 instances, length>0) : 99,513 -> 28,735/70,788 (70,853/28,660)
- Image Preprocessing?



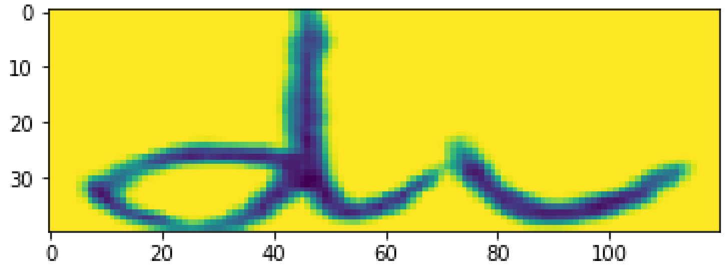


```
Imagine
(40, 120)
```



```
(40, 120)
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         0.9815147  0.7697217
 0.65295357 0.76888514 0.951021   1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.        ]
```

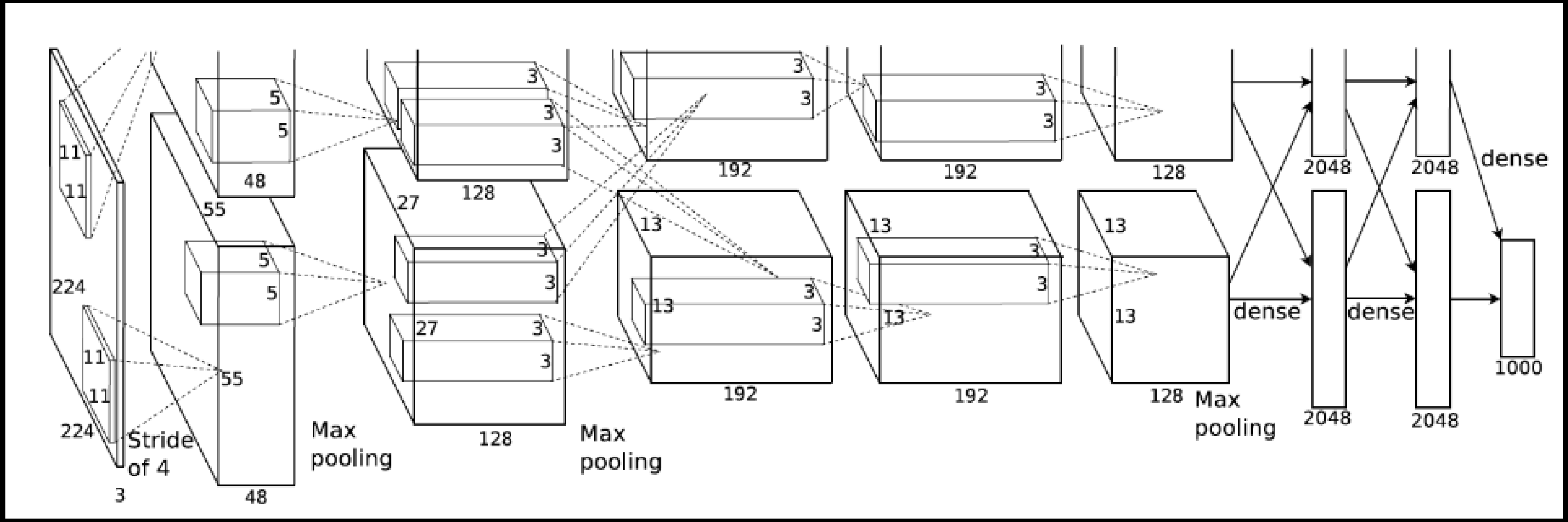
```
du 2
(40, 120)
```



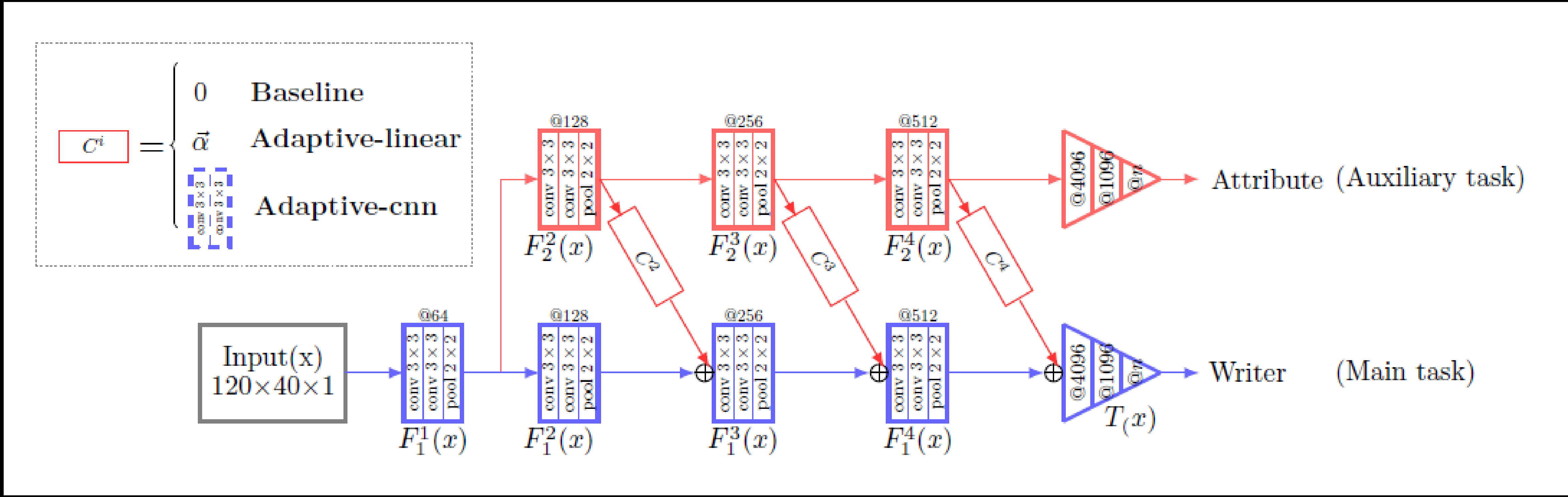
```
(40, 120)
[1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         0.9994528
 0.9960785  0.9237485  0.8464991  0.74871206 0.71982515 0.79017377
 0.870928   0.99407923 0.9986231  1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.        ]
```

Proposed Method

- Single word images
- CNN for multi task learning (writer identification main task, auxiliary task?)
- Two pathways
- ~AlexNet, NIPS, 2012



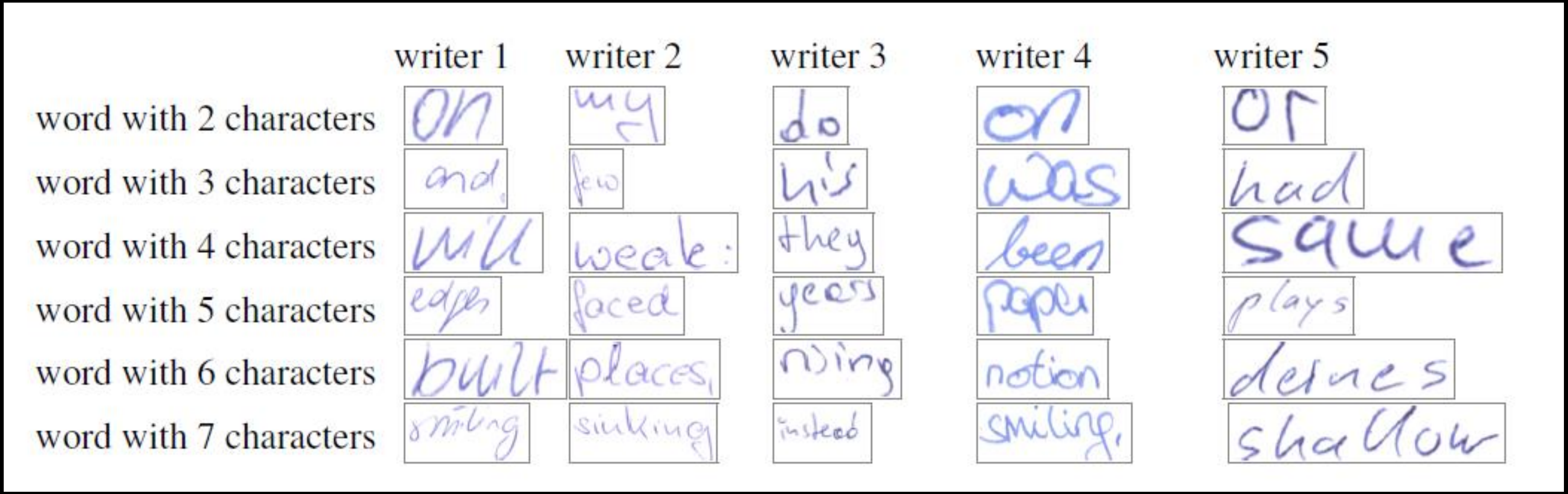
Architecture



Auxiliary tasks

- Word recognition
- Character atribute recognition
- Word length estimation (1-13)

Model	Writer Identification			
	CVL		IAM	
	Top-1	Top-5	Top-1	Top-5
Baseline	75.3	92.5	66.0	82.9
Linear-adaptive	75.9	92.7	65.4	83.1
Deep-adaptive	**79.1**	**94.3**	**68.3**	**85.2**



Model?

- Main formula

$$in(F_1^{i+1}) = r(F_1^i) + C^i\left(r(F_2^i)\right)$$

- Baseline

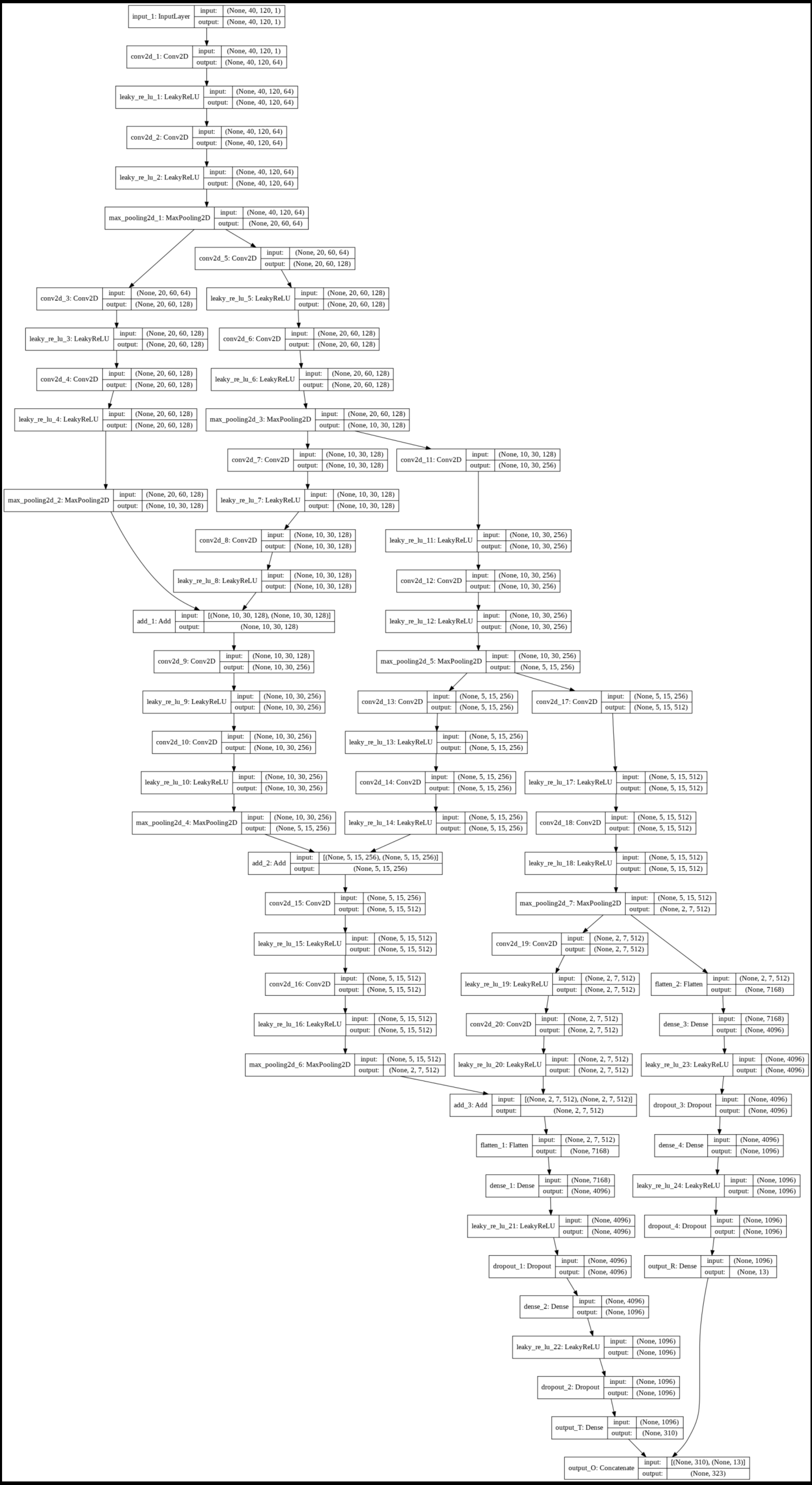
$$C^i(\cdot) = 0$$

- Linear

$$in_j(F_1^{i+1}) = \alpha_j \cdot r_j(F_1^i) + (1 - \alpha_j) \cdot r_j(F_2^i)$$

- Deep Adaptive

$$C^i\left(r(F_2^i)\right) = in(F_1^{i+1}) - r(F_1^i)$$



input_1: InputLayer input: (None, 40, 120, 1) output: (None, 40, 120, 1)
conv2d_1: Conv2D input: (None, 40, 120, 1) output: (None, 40, 120, 64)
leaky_re_lu_1: LeakyReLU input: (None, 40, 120, 64) output: (None, 40, 120, 64)
conv2d_2: Conv2D input: (None, 40, 120, 64) output: (None, 40, 120, 64)
leaky_re_lu_2: LeakyReLU input: (None, 40, 120, 64) output: (None, 40, 120, 64)
max_pooling2d_1: MaxPooling2D input: (None, 40, 120, 64) output: (None, 20, 60, 64)
conv2d_5: Conv2D input: (None, 20, 60, 64) output: (None, 20, 60, 128)
conv2d_3: Conv2D input: (None, 20, 60, 64) output: (None, 20, 60, 128)
leaky_re_lu_5: LeakyReLU input: (None, 20, 60, 128) output: (None, 20, 60, 128)
leaky_re_lu_3: LeakyReLU input: (None, 20, 60, 128) output: (None, 20, 60, 128)
conv2d_6: Conv2D input: (None, 20, 60, 128) output: (None, 20, 60, 128)
conv2d_4: Conv2D input: (None, 20, 60, 128) output: (None, 20, 60, 128)
leaky_re_lu_6: LeakyReLU input: (None, 20, 60, 128) output: (None, 20, 60, 128)
leaky_re_lu_4: LeakyReLU input: (None, 20, 60, 128) output: (None, 20, 60, 128)
max_pooling2d_3: MaxPooling2D input: (None, 20, 60, 128) output: (None, 10, 30, 128)
conv2d_7: Conv2D input: (None, 10, 30, 128) output: (None, 10, 30, 128)
conv2d_11: Conv2D input: (None, 10, 30, 128) output: (None, 10, 30, 256)
max_pooling2d_2: MaxPooling2D input: (None, 20, 60, 128) output: (None, 10, 30, 128)
leaky_re_lu_7: LeakyReLU input: (None, 10, 30, 128) output: (None, 10, 30, 128)
conv2d_8: Conv2D input: (None, 10, 30, 128) output: (None, 10, 30, 128)
leaky_re_lu_11: LeakyReLU input: (None, 10, 30, 256) output: (None, 10, 30, 256)
leaky_re_lu_8: LeakyReLU input: (None, 10, 30, 128) output: (None, 10, 30, 128)
conv2d_12: Conv2D input: (None, 10, 30, 256) output: (None, 10, 30, 256)
add_1: Add input: [(None, 10, 30, 128), (None, 10, 30, 128)] output: (None, 10, 30, 128)
leaky_re_lu_12: LeakyReLU input: (None, 10, 30, 256) output: (None, 10, 30, 256)
conv2d_9: Conv2D input: (None, 10, 30, 128) output: (None, 10, 30, 256)
max_pooling2d_5: MaxPooling2D input: (None, 10, 30, 256) output: (None, 5, 15, 256)
leaky_re_lu_9: LeakyReLU input: (None, 10, 30, 256) output: (None, 10, 30, 256)
conv2d_13: Conv2D input: (None, 5, 15, 256) output: (None, 5, 15, 256)
conv2d_17: Conv2D input: (None, 5, 15, 256) output: (None, 5, 15, 512)
conv2d_10: Conv2D input: (None, 10, 30, 256) output: (None, 10, 30, 256)
leaky_re_lu_13: LeakyReLU input: (None, 5, 15, 256) output: (None, 5, 15, 256)
leaky_re_lu_10: LeakyReLU input: (None, 10, 30, 256) output: (None, 10, 30, 256)
conv2d_14: Conv2D input: (None, 5, 15, 256) output: (None, 5, 15, 256)
leaky_re_lu_17: LeakyReLU input: (None, 5, 15, 512) output: (None, 5, 15, 512)
max_pooling2d_4: MaxPooling2D input: (None, 10, 30, 256) output: (None, 5, 15, 256)
leaky_re_lu_14: LeakyReLU input: (None, 5, 15, 256) output: (None, 5, 15, 256)
conv2d_18: Conv2D input: (None, 5, 15, 512) output: (None, 5, 15, 512)
add_2: Add input: [(None, 5, 15, 256), (None, 5, 15, 256)] output: (None, 5, 15, 256)
leaky_re_lu_18: LeakyReLU input: (None, 5, 15, 512) output: (None, 5, 15, 512)
conv2d_15: Conv2D input: (None, 5, 15, 256) output: (None, 5, 15, 512)
max_pooling2d_7: MaxPooling2D input: (None, 5, 15, 512) output: (None, 2, 7, 512)
leaky_re_lu_15: LeakyReLU input: (None, 5, 15, 512) output: (None, 5, 15, 512)
conv2d_19: Conv2D input: (None, 2, 7, 512) output: (None, 2, 7, 512)
conv2d_16: Conv2D input: (None, 5, 15, 512) output: (None, 5, 15, 512)
leaky_re_lu_19: LeakyReLU input: (None, 2, 7, 512) output: (None, 2, 7, 512)
flatten_2: Flatten input: (None, 2, 7, 512) output: (None, 7168)
leaky_re_lu_16: LeakyReLU input: (None, 5, 15, 512) output: (None, 5, 15, 512)
conv2d_20: Conv2D input: (None, 2, 7, 512) output: (None, 2, 7, 512)
dense_3: Dense input: (None, 7168) output: (None, 4096)
max_pooling2d_6: MaxPooling2D input: (None, 5, 15, 512) output: (None, 2, 7, 512)
leaky_re_lu_20: LeakyReLU input: (None, 2, 7, 512) output: (None, 2, 7, 512)
leaky_re_lu_23: LeakyReLU input: (None, 4096) output: (None, 4096)
add_3: Add input: [(None, 2, 7, 512), (None, 2, 7, 512)] output: (None, 2, 7, 512)
dropout_3: Dropout input: (None, 4096) output: (None, 4096)
flatten_1: Flatten input: (None, 2, 7, 512) output: (None, 7168)
dense_4: Dense input: (None, 4096) output: (None, 1096)
dense_1: Dense input: (None, 7168) output: (None, 4096)
leaky_re_lu_24: LeakyReLU input: (None, 1096) output: (None, 1096)
leaky_re_lu_21: LeakyReLU input: (None, 4096) output: (None, 4096)
dropout_4: Dropout input: (None, 1096) output: (None, 1096)
dropout_1: Dropout input: (None, 4096) output: (None, 4096)
output_R: Dense input: (None, 1096) output: (None, 13)
dense_2: Dense input: (None, 4096) output: (None, 1096)
leaky_re_lu_22: LeakyReLU input: (None, 1096) output: (None, 1096)
dropout_2: Dropout input: (None, 1096) output: (None, 1096)
output_T: Dense input: (None, 1096) output: (None, 310)
output_O: Concatenate input: [(None, 310), (None, 13)] output: (None, 323)

This is still seminary work

```
O = Concatenate(axis=1, name='output_O')([T, R])
```

```
model2 = Model(inputs=x_input, outputs=[T, R, O], name='Model2')
```

```
def CustomLoss(y_true, y_pred):
  y_true1 = y_true[:, :MAX_WRITER]
  y_true2 = y_true[:, MAX_WRITER:]

  y_pred1 = y_pred[:, :MAX_WRITER]
  y_pred2 = y_pred[:, MAX_WRITER:]

  return alpha*K.categorical_crossentropy(y_true1, y_pred1)+(1-alpha)*K.categorical_crossentropy(y_true2, y_pred2)
```

```
alphaChanger = LambdaCallback(on_epoch_end=lambda epoch, _: K.set_value(alpha, 0.5+(epoch+1)*0.01))
```

```
model2.compile(optimizer=Adam(lr=0.0001), \
               loss={'output_T' : categorical_crossentropy, 'output_R': categorical_crossentropy, 'output_O': CustomLoss}, \
               loss_weights={'output_T': 0, 'output_R': 0, 'output_O':1}, \
               metrics=['acc'])
```

Implementation details

- Conv2D: kernel_size = (3, 3)
- MaxPool2D: pool_size = (2, 2)
- LeakyRelu(0.1)
- Dropout(0.5)
- cross-entropy loss (softmax activation/sigmoid activation?)
- Xavier initialization
- Adam(0.001)
- batch_size=100, n_iter=40 000 ~ 40 epoch! (My problem?)
- 83,589,227 params

Training

- Combine two losses (O -> T, R)

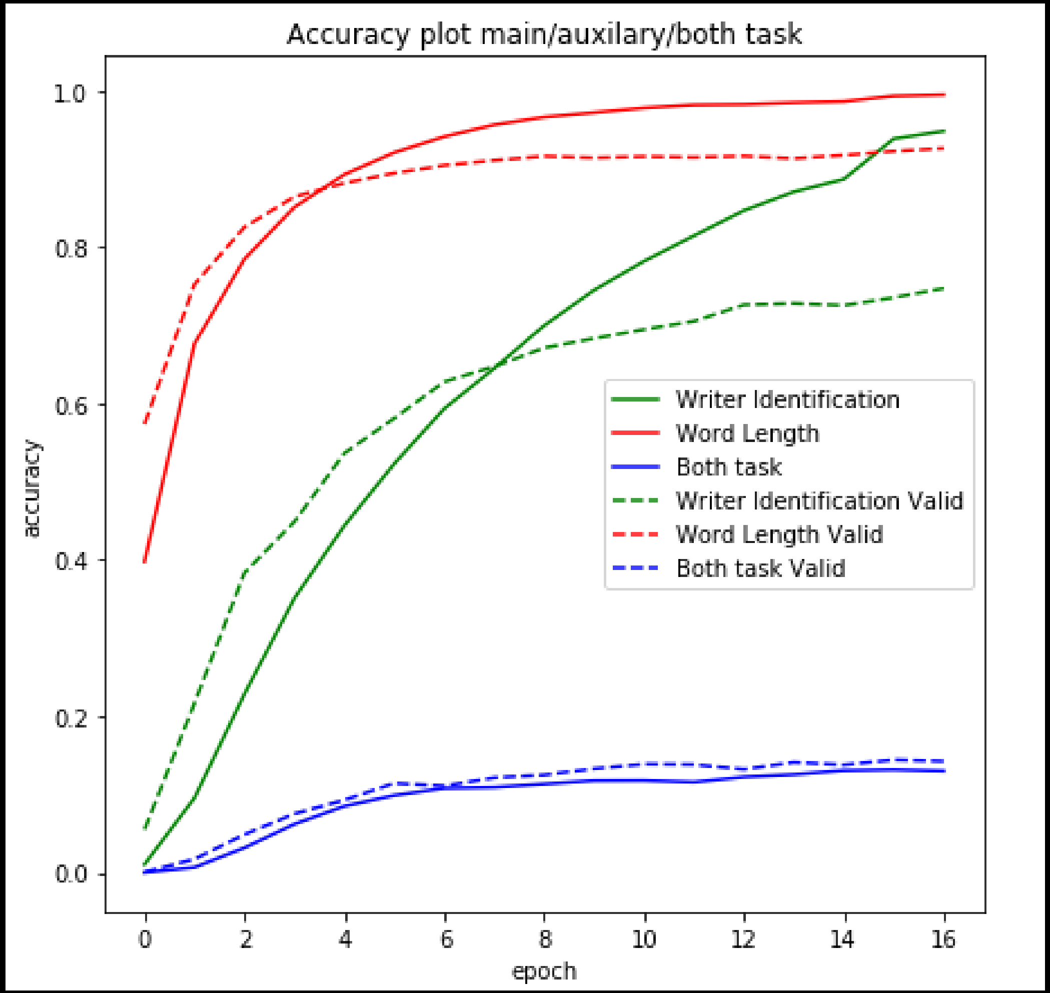
$$\text{Loss}_{total} = (1-\lambda)\text{Loss}_{au} + \lambda\text{Loss}_{wi}$$

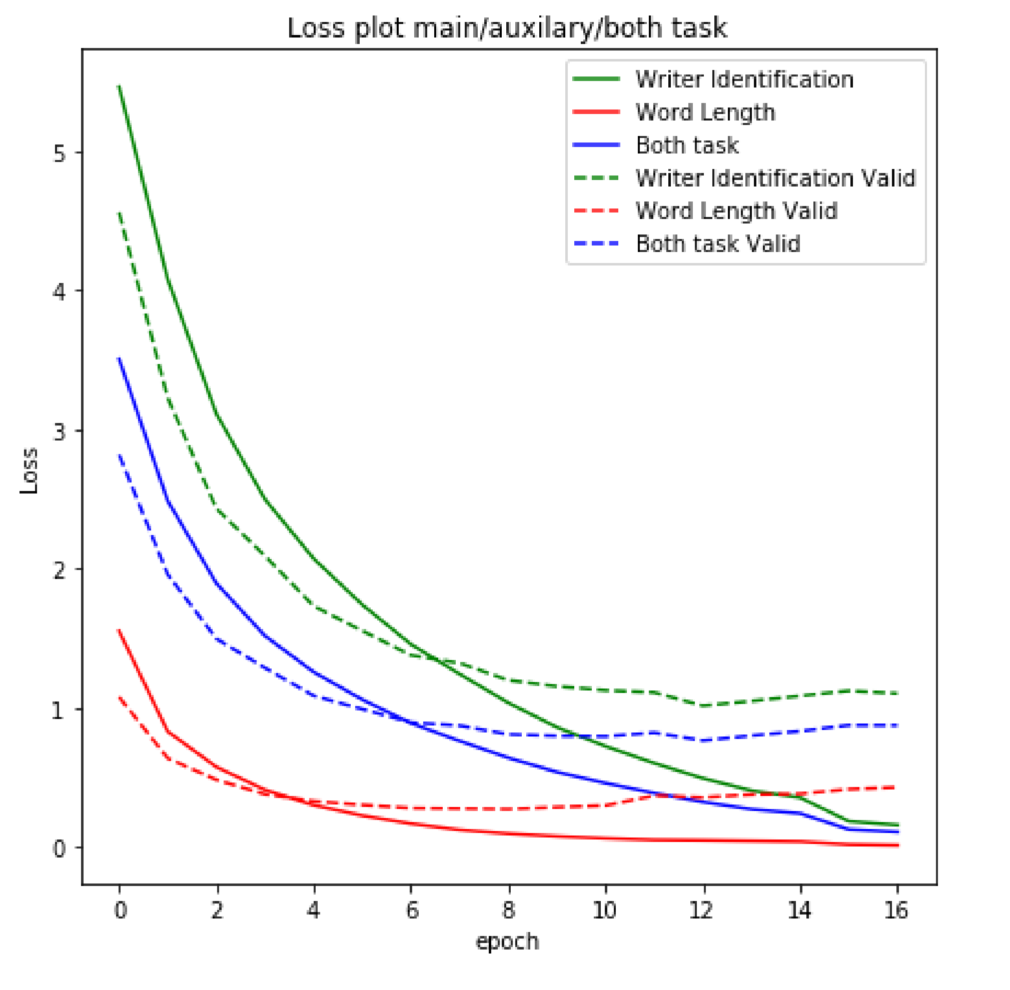
- λ = 0.5 (10 epoch) + 0.066 (5 epoch) - increasing lambda effect?
- 0.896

- λ = 0.5 (0 epoch) + 0.01 (every epoch) - better one
- 0.9

- Weird lambda?

Early stopping? Overfitting? 15/40? Split?





Top1/Top5?

- Weird evaluation?
- Identifcation based on N word images from the same writer
- Randomly selected N word images for each writer
- Put them into CNN
- Use average response of the last CNN layer to recognize writer (x20 times, avg)

$$y = \frac{1}{N} \sum_{i}^{N} \mathrm{CNN}(x_i)$$

Results (Paper)

Model	Input size: 40×120×1			
	CVL		IAM	
	W.I.	W.L.E.	W.I.	W.L.E.
Baseline	75.3	94.3	66.0	91.5
Linear-adaptive	75.9	92.7	65.4	90.4
Deep-adaptive	79.1	93.6	68.3	91.6
Training Time	8.5 hours			

Baseline: 68.62%
Deep-adaptive: 72.37%/92.02% (T/R)

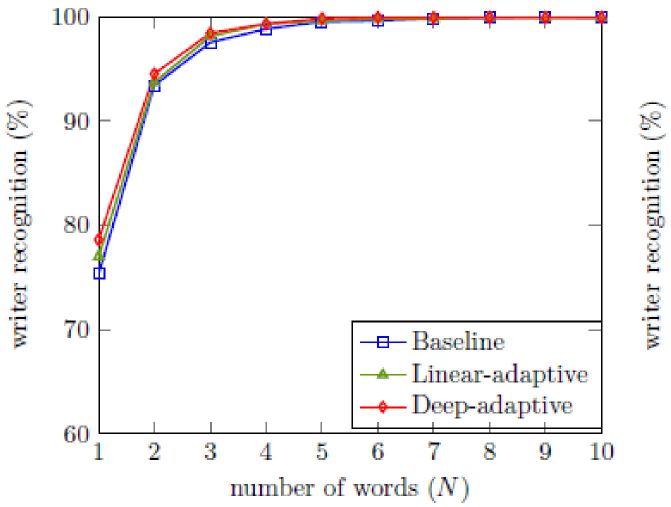
Method	CVL	
	Top1	Top5
Hinge [1]	25.8	48.0
Quill [2]	29.4	52.6
Chain Code Pairs [3]	22.4	44.6
Chain Code Triplets [3]	28.8	51.4
COLD [34]	12.8	29.6
QuadHinge [28]	30.0	52.4
CoHinge [28]	25.9	46.9
CNN [6]	75.3	92.6
CNN+Adaptive	79.1	93.7

???

Table 2: Performance of writer identification using different adaptive learning methods with **word recognition** as the auxiliary task on the **CVL** and **IAM** datasets.

Model	Writer Identification				Word Recognition (*aux.*)			
	CVL		**IAM**		**CVL**		**IAM**	
	Top1	Top5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Baseline	75.3	92.4	65.7	83.5	95.1	99.1	93.5	98.7
Linear-adaptive	77.0	93.1	68.0	84.7	94.1	98.9	91.3	98.1
Deep-adaptive	**78.6**	**93.7**	**69.5**	**86.1**	94.5	99.0	92.6	98.4

Misunderstanding?



(a) **CVL** data set

From Fig. 4 we can see that writer-identification performance increases with more word images from the same writer. The **Deep-adaptive** model achieves the best results with different numbers of words for writer identification. The Top-1 performance for writer identification using the **Deep-adaptive** model was 79.1% and 68.3% when using one word, and this increases to 99.8% and 92.0% when using five words on **CVL** and **IAM**, respectively. For the special-

Figure 4: Performance (Top1) of writer identification using different numbers of words (from 1 to 10 words), using CNN models trained with word recognition as the auxiliary task on the **CVL** dataset (Figure (a)) and the **IAM** dataset (Figure (b)).

My improvements

- Next-To-Last layer
- Baseline:
  - Last: Top1/Top5/Top10: 67.90%/95.03%/98.87%
  - Next-To-Last: Top1/Top5/Top10: 66.99%/98.75%/99.73% (on Top1 is the small sample)

    +3.72%
- Deep:
  - Last: Top1/Top5/Top10: 72.11%/97.20%/99.50% VS 79.10%/99.80%??
  - Next-To-Last: Top1/Top5/Top10: 70.53%/99.29%/99.98%

    +2.09%

More ideas?

- Babysitting the learning process
  - (O? Better guidance through loss surface?)
- Preprocessing?
  - (Why n is the best auxiliary?)
- One word cropping?
  - (Augmentation some kind? Histogram with respect to n? ~Top3/5)

- Baseline (Hinton) to 75.3% and Next-To-Last = Top5 Improvements publication!

Questions?

Milan M. Čugurović
milan_cugurovic@matf.bg.ac.rs