Images are self-made, part of the IAM Dataset, or credited in the slides.

Horizontally Variable OCR

Horizontally Variable OCR

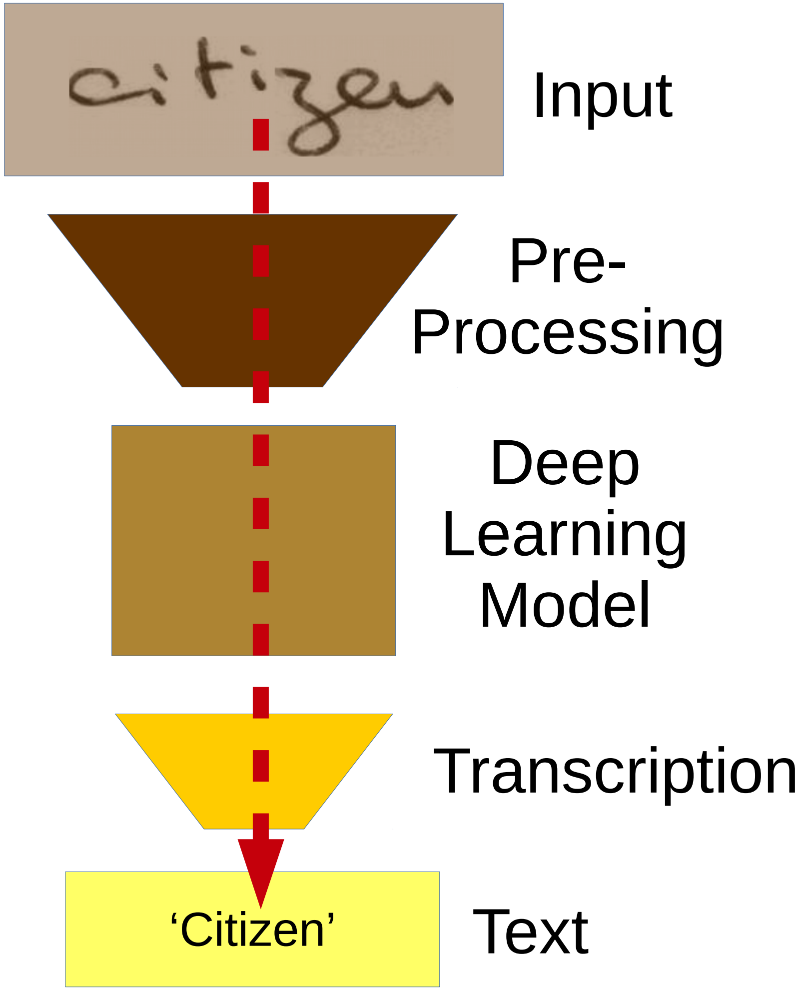
Or something….

Goal: Transcription of Images to Text, without letter segmentation

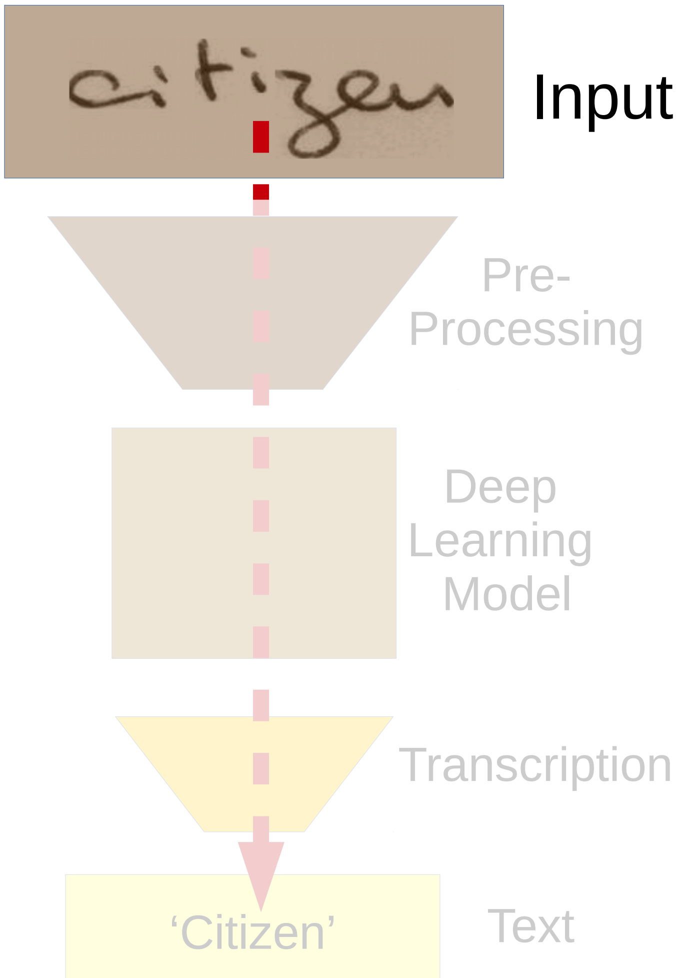
- Examine a lot of different datasets, and found one which had the closest resemblance to my problem.
- Ended up with the IAM (offline, i.e. handwritten) dataset with **word segmentation***
- I’ll skip over the first few months of research :)
- First let’s see what the ML model is supposed to do:

*The IAM dataset contains variations with word, sentence and line segmentation.

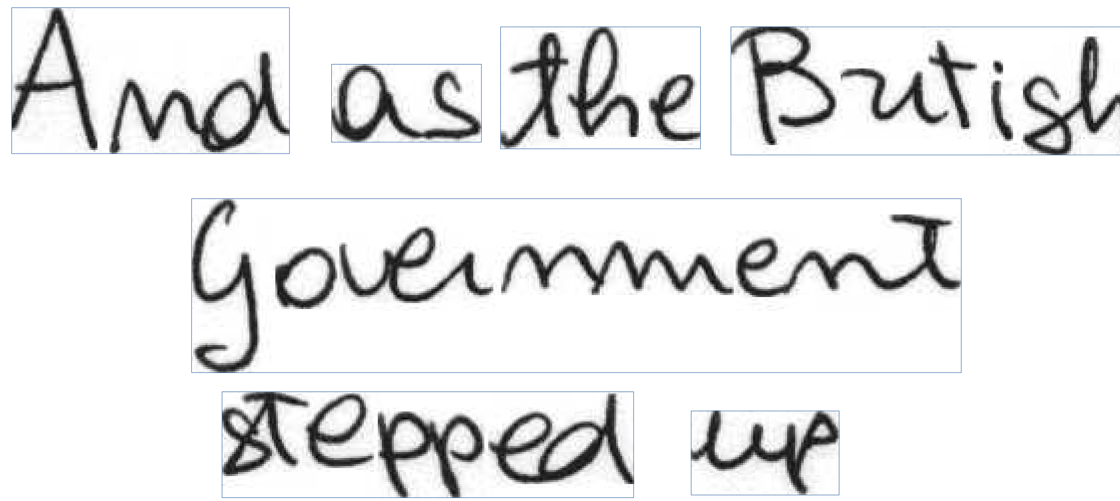
Black Box Illustration



Black Box Illustration



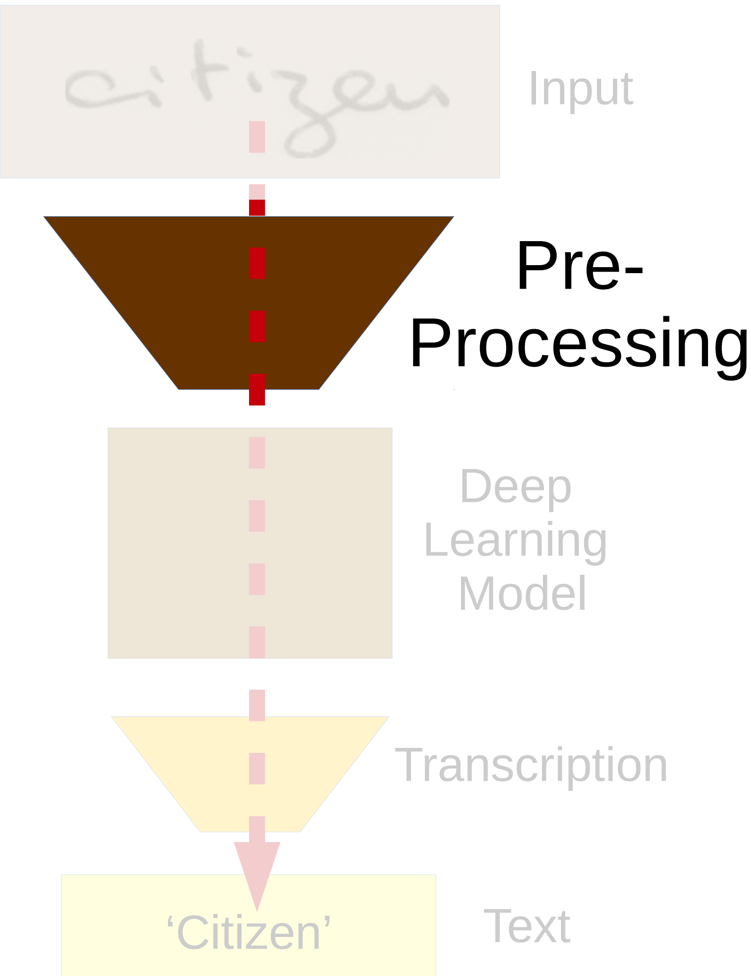
Text Examples from IAM:

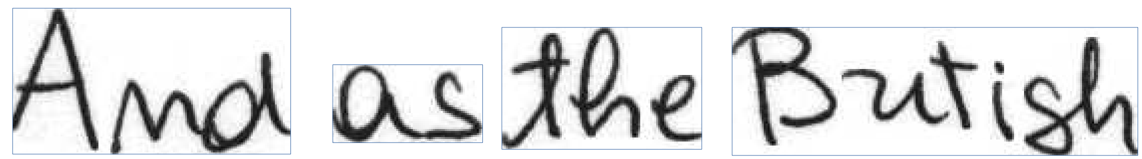


Labels from the Dataset:
'And' 'as' 'the' 'British'
'Government'
'stepped' 'up'

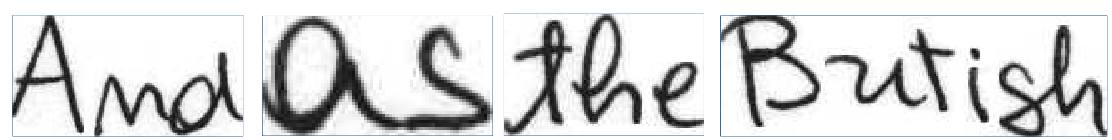
Images from the IAM dataset: http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

Black Box Illustration



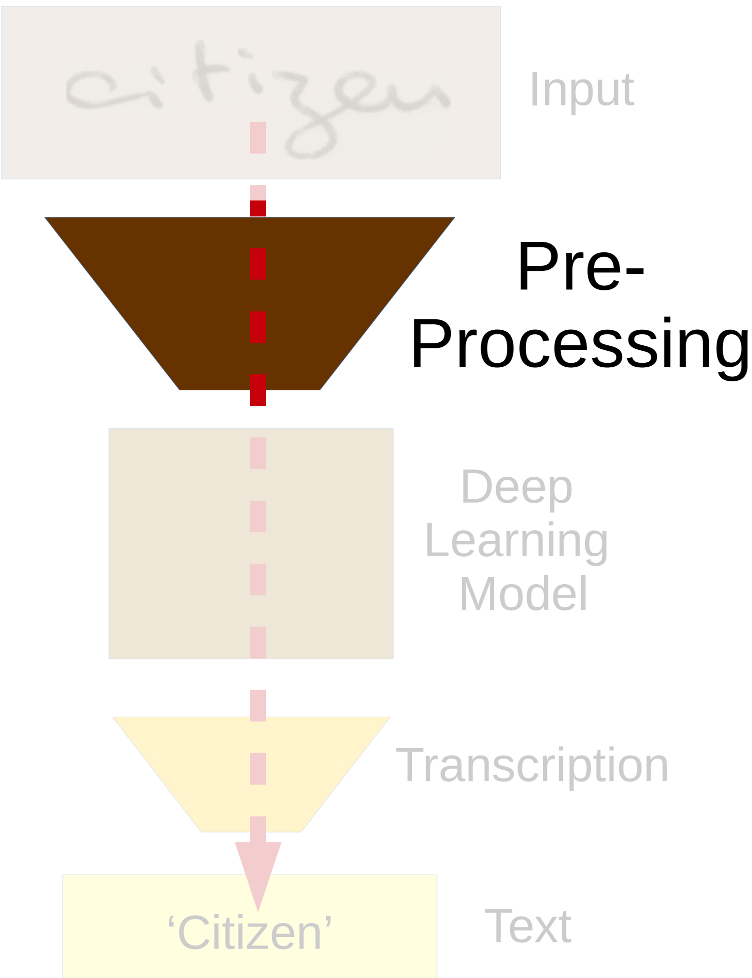


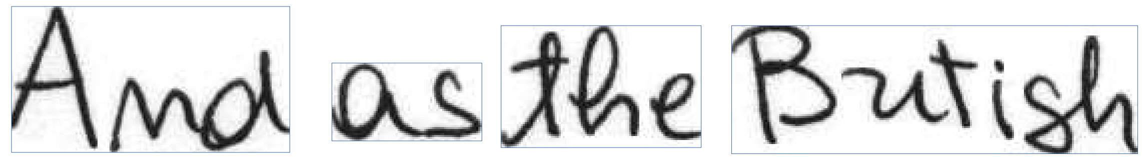
Text had to be normalized to a fixed height:



Not ideal, but the letters should be recognized regardless of size anyway.

Black Box Illustration

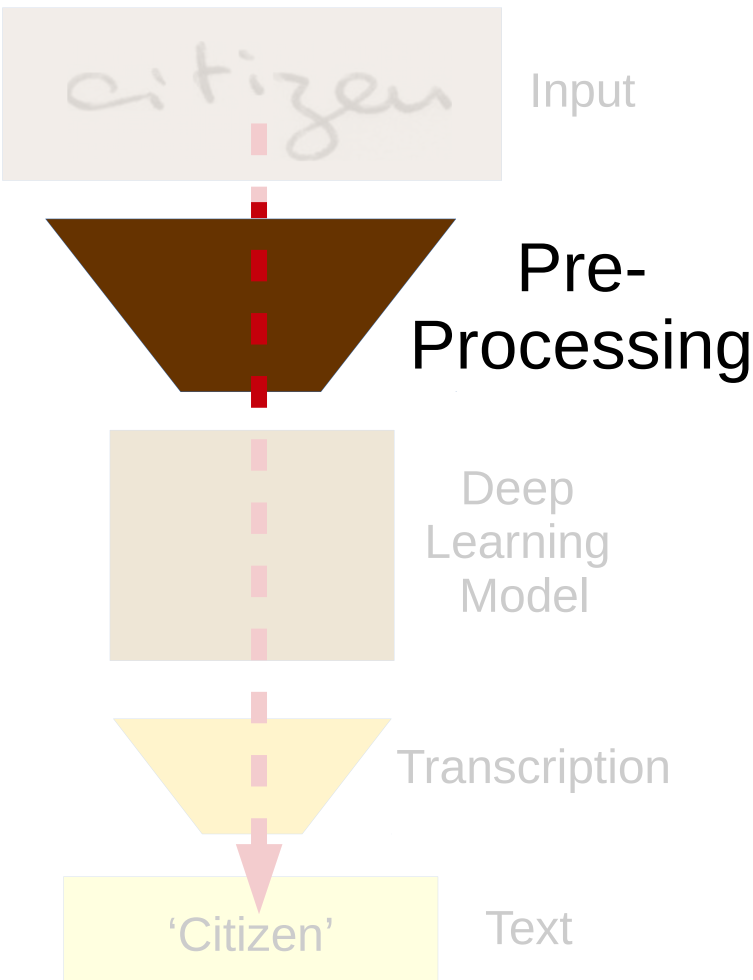


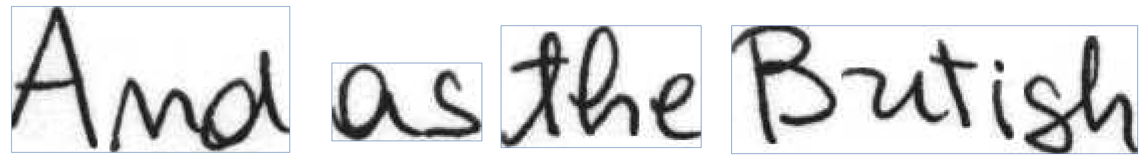


<u>Why not just pad the words?</u>

Due to the large variation between the images padding might end up being very large, which would cause a visible break between the images and the padding. This is already a problem in some of the images:

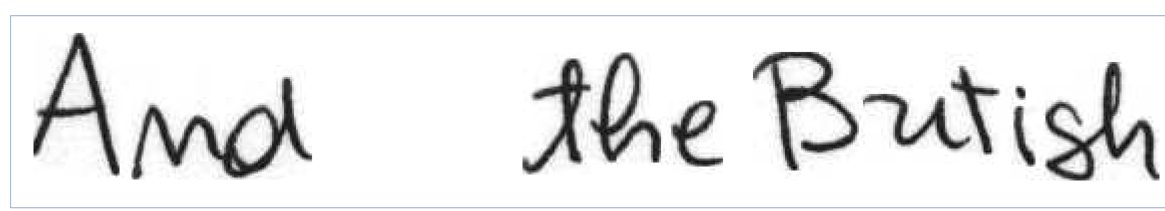
Black Box Illustration





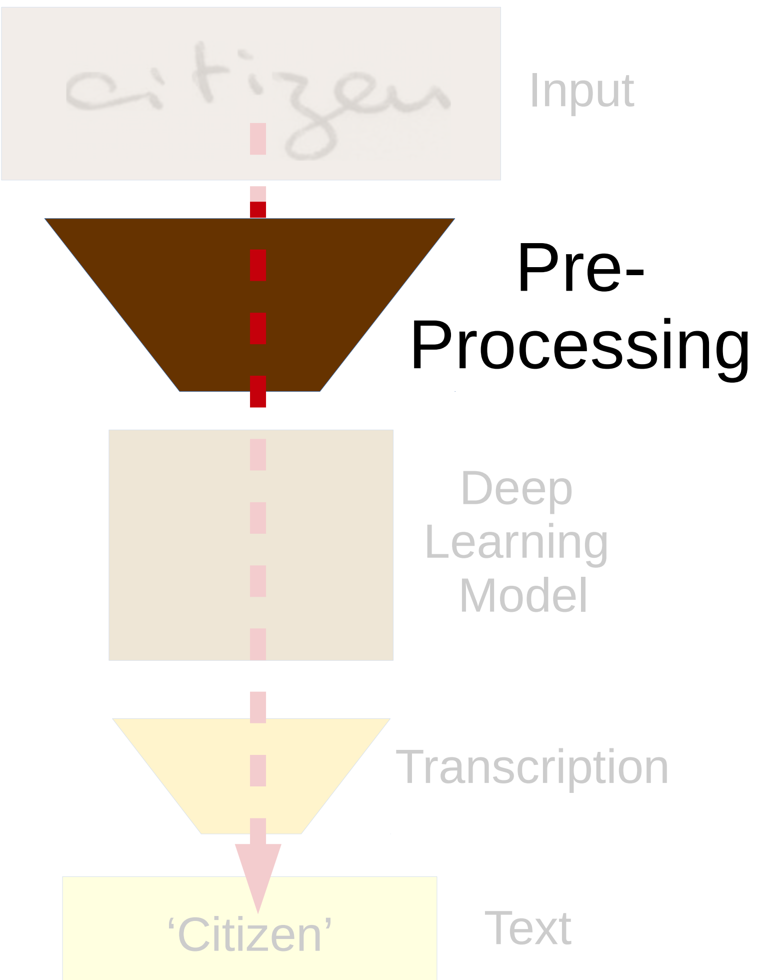
Different checks were made, the dataset was cleaned of the biggest errors in labels or images:

Some words were not segmented so that an entire sentence was labeled as one word, and sometimes the labels were wrong.



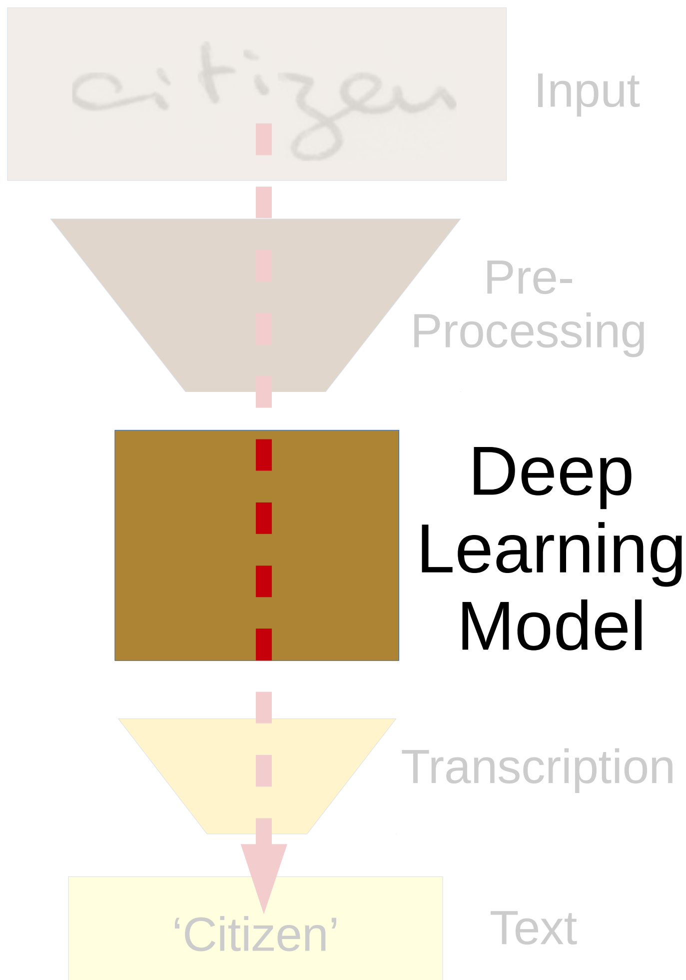
'And'

Black Box Illustration



Images were pickled after preprocessing for easier and faster loading, as well as for the convenience of loading a single file instead of 115.276

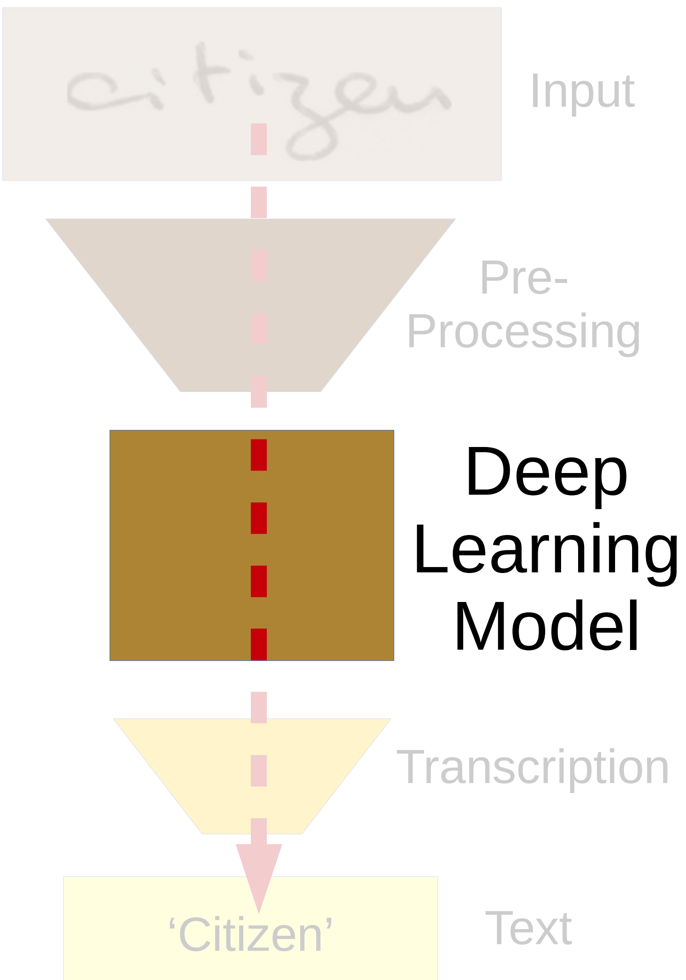
Black Box Illustration

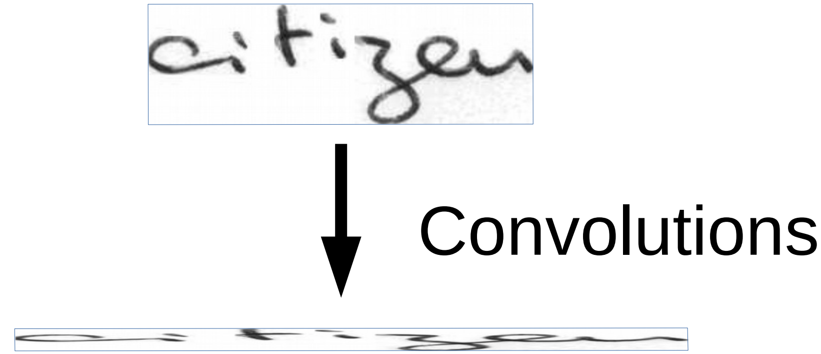


After some research, I found this paper:
'An End-to-End Trainable NN for Image-based Sequence Recognition and it's application to Scene Text Recognition'.

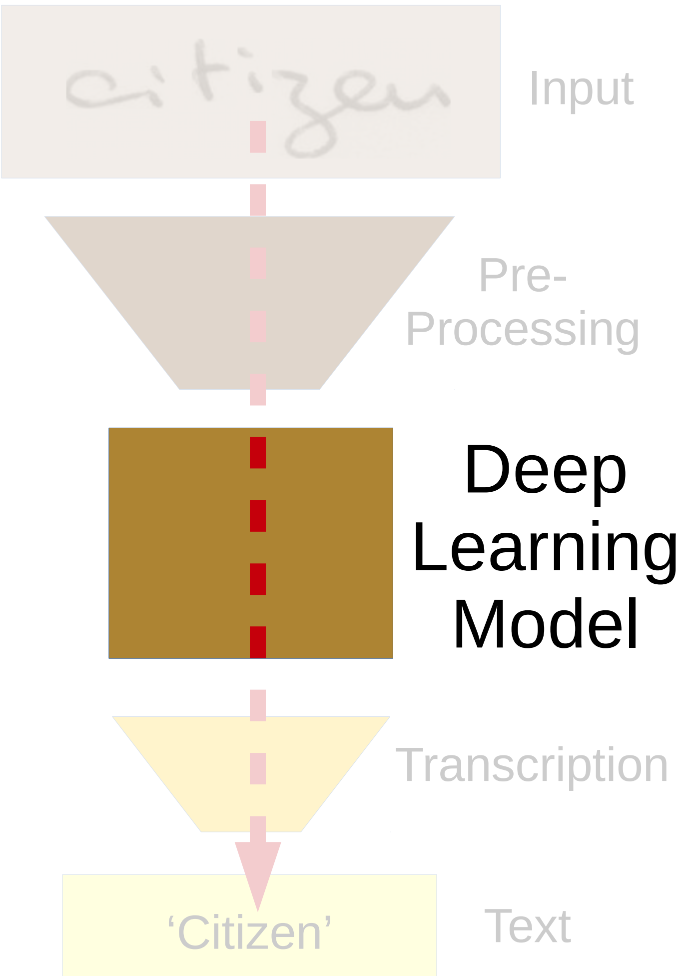
What this paper, in a rather complicated fashion, describes, is a model where Convolutional Layers are used to extract information from an image, and then a RNN is used to process that data to be transcribed into text.

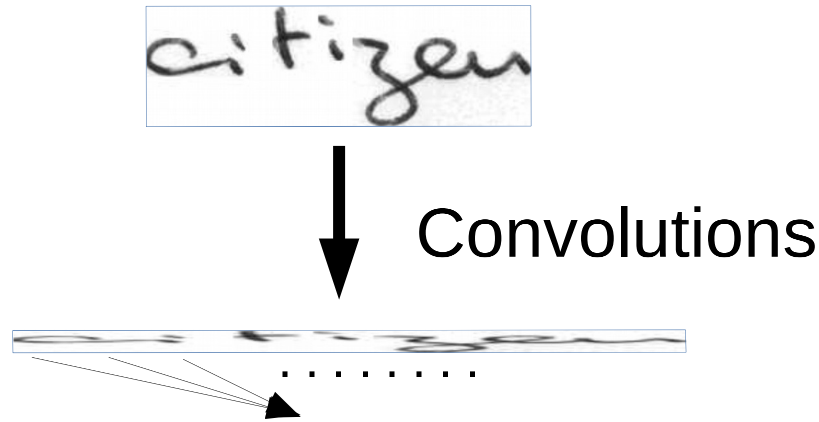
Black Box Illustration





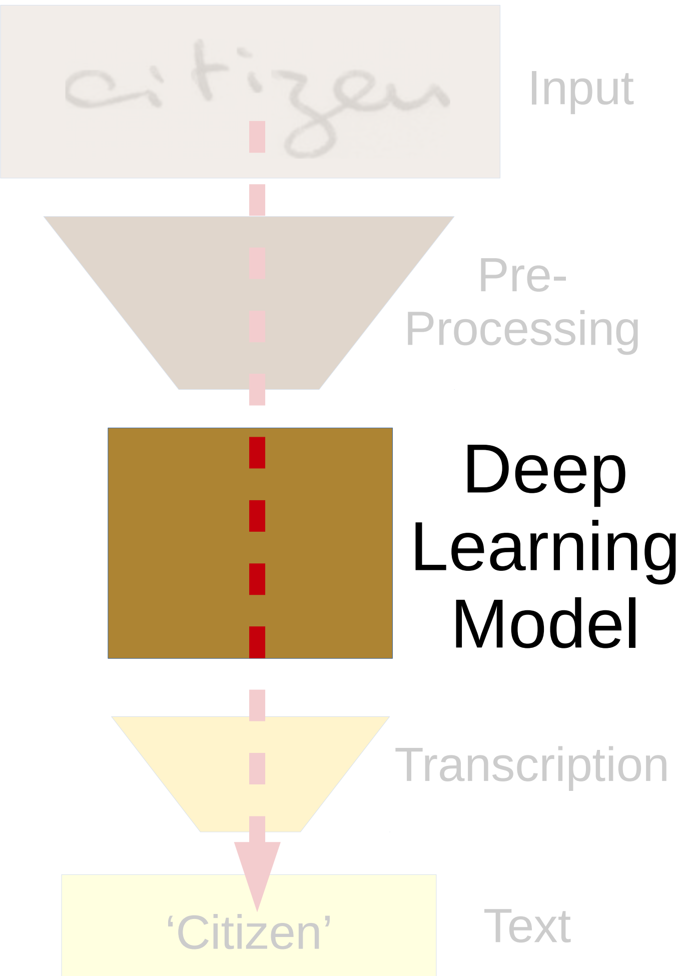
Black Box Illustration

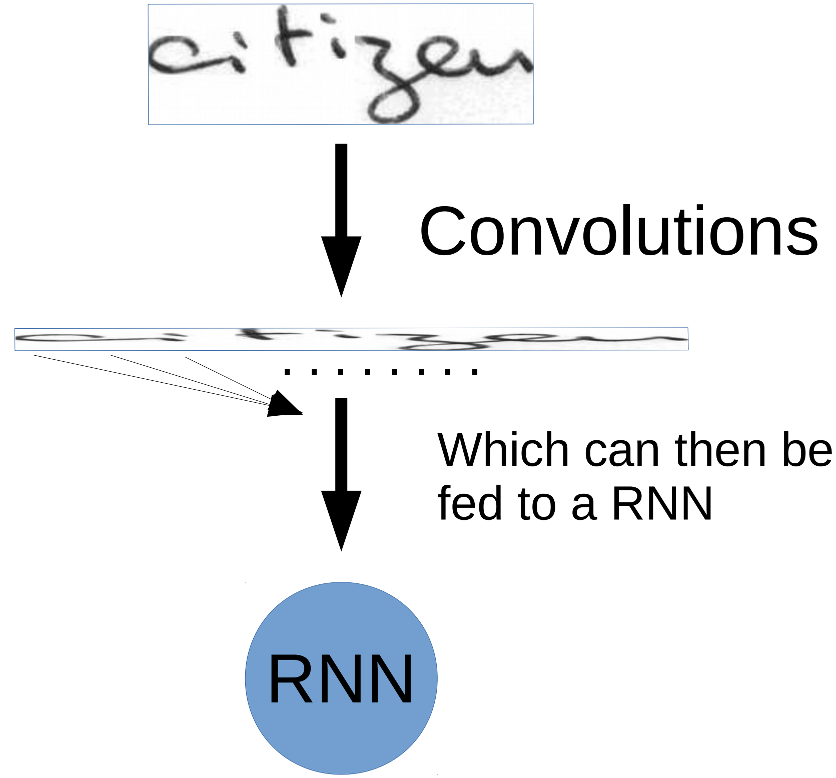




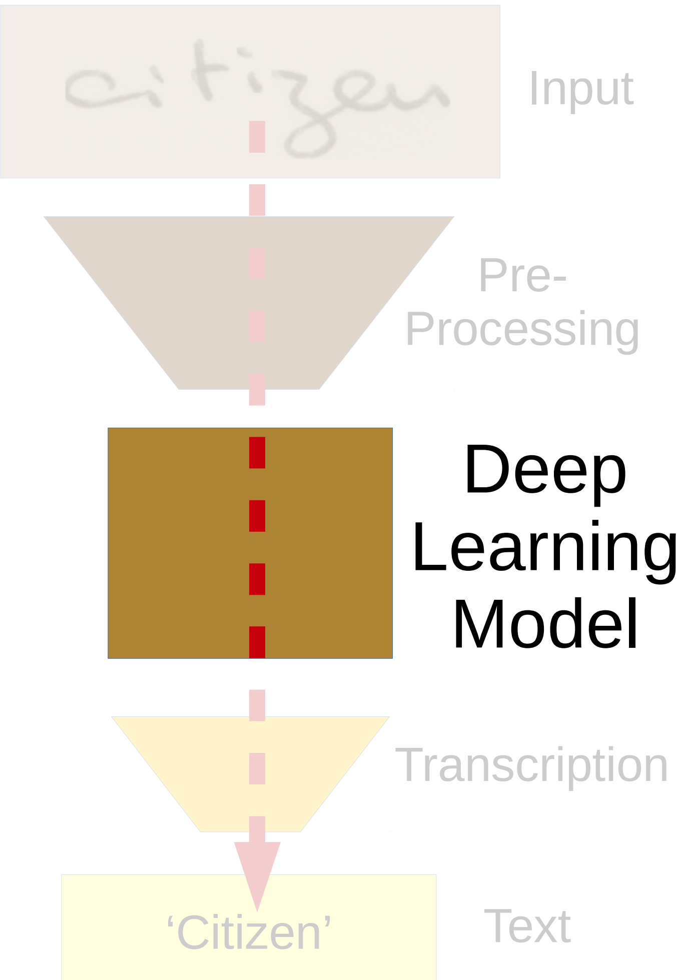
Each column of pixels is treated as a separate data point in a sequence

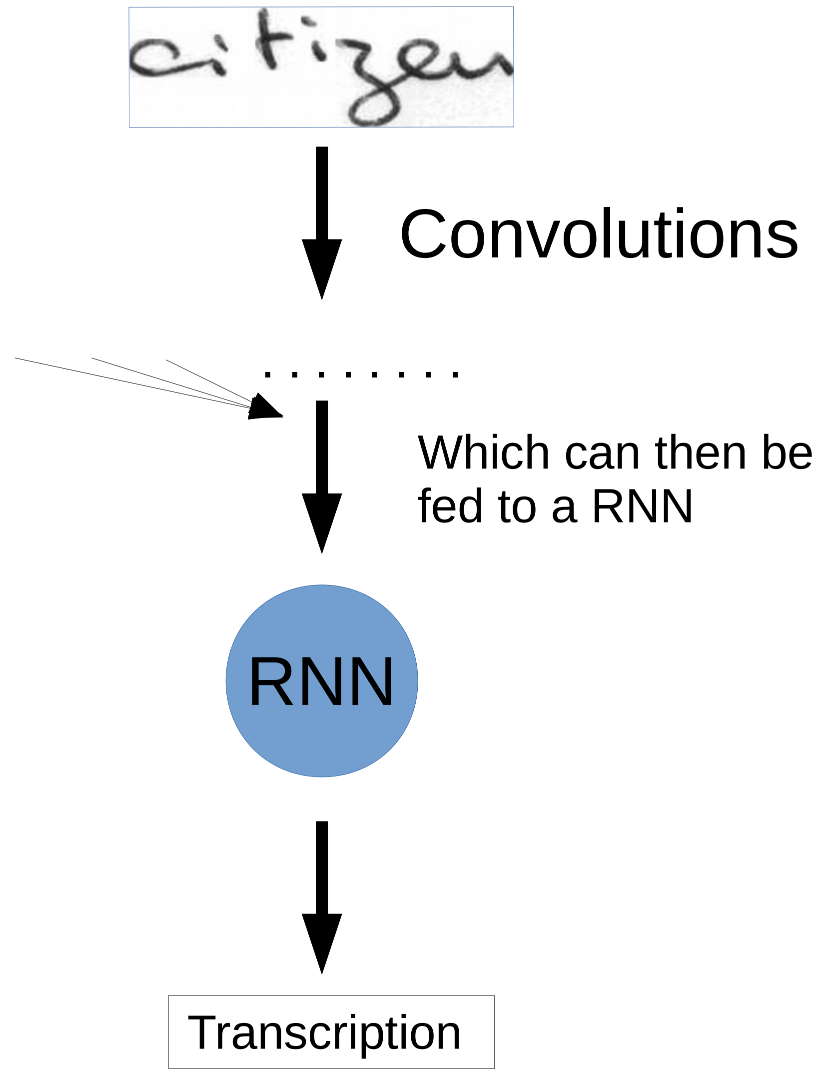
Black Box Illustration



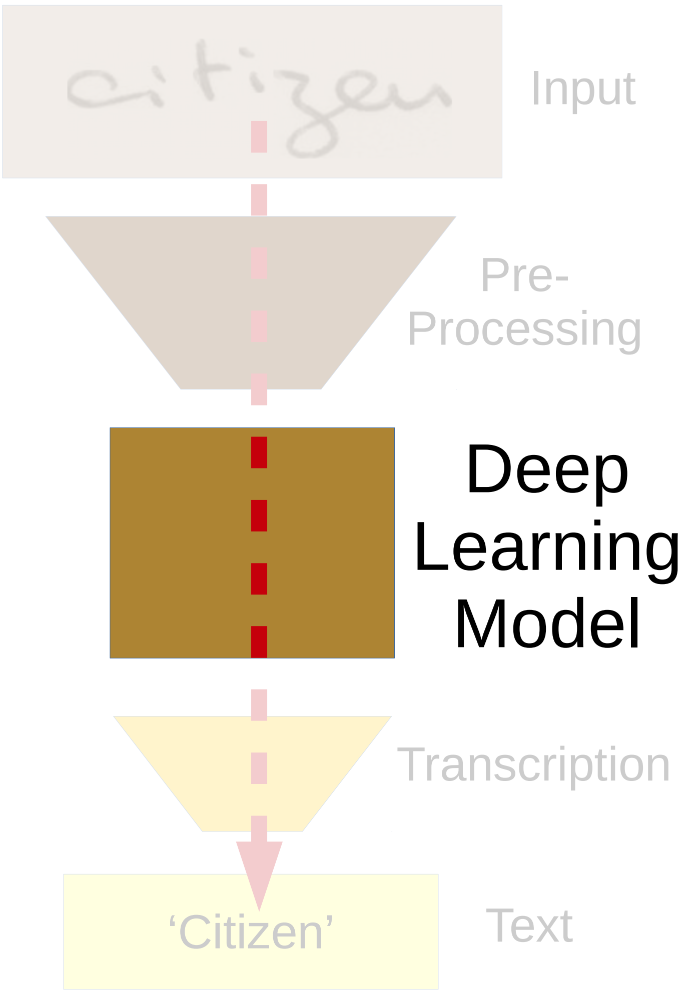


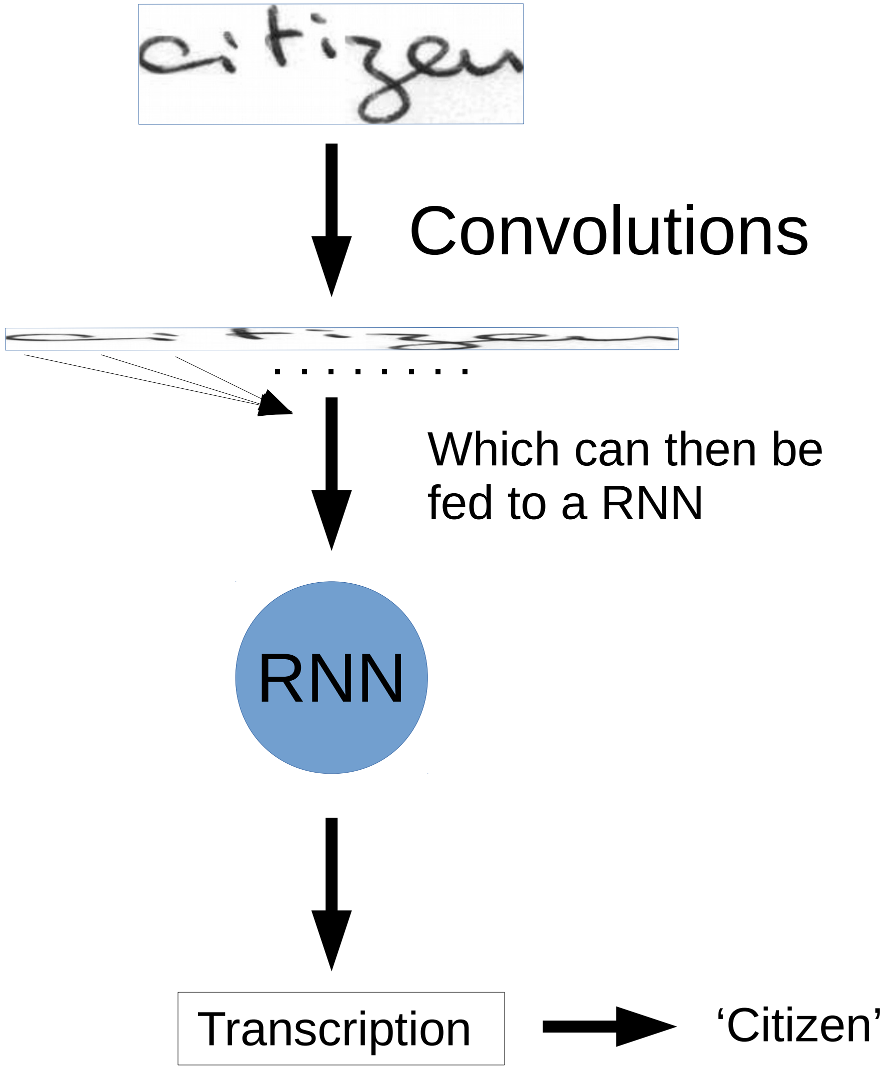
Black Box Illustration



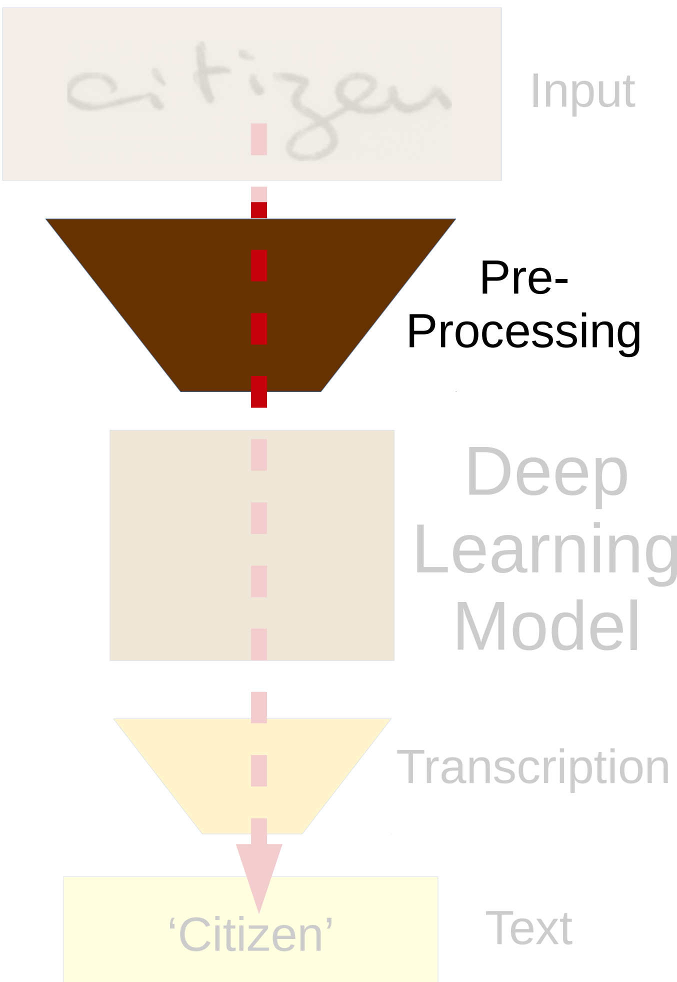


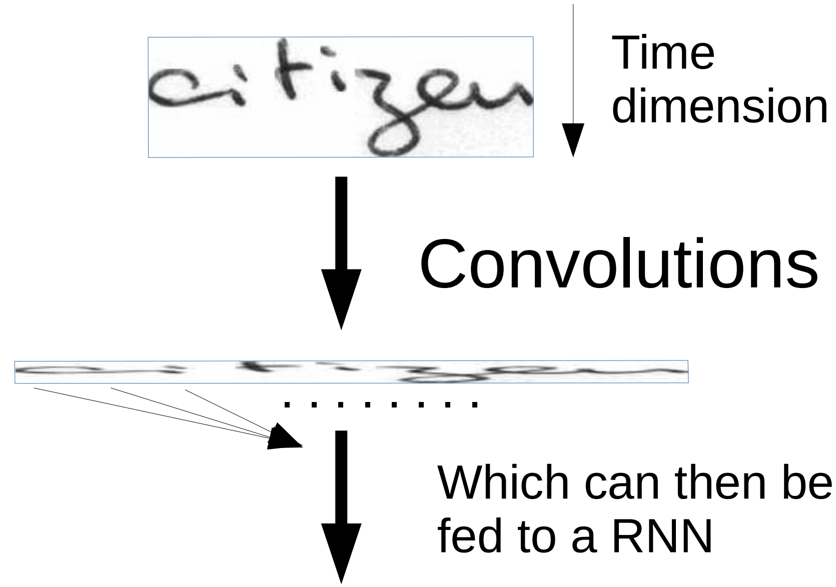
Black Box Illustration





Aside:

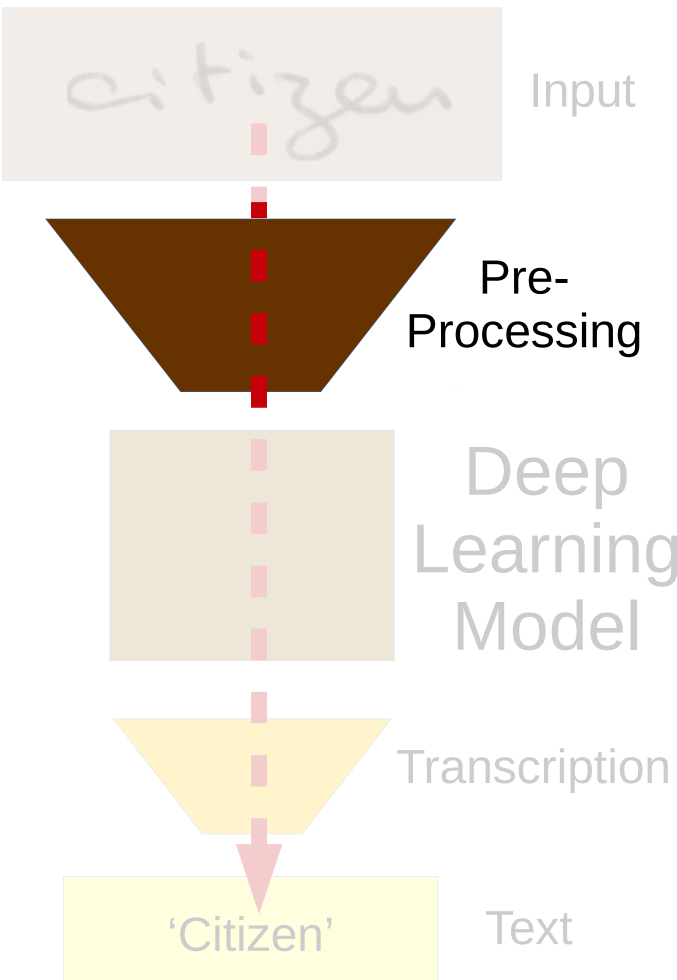


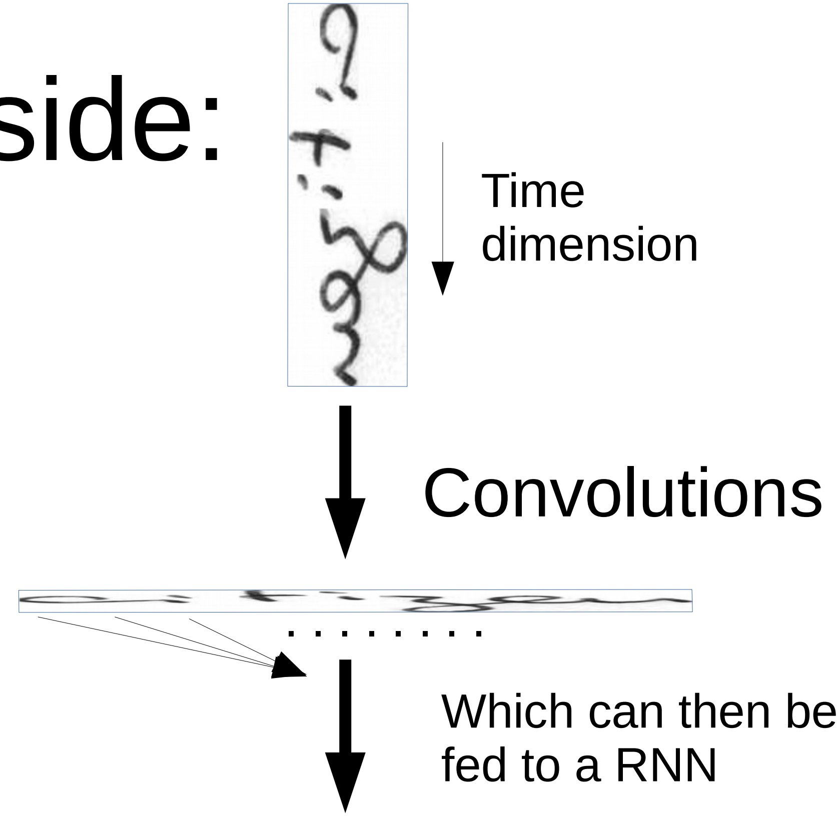


Since the RNNs take data in which the 'Time Dimension'* is the first dimension, the dimensions for the images were flipped.

*See https://keras.io/layers/recurrent/

Aside:

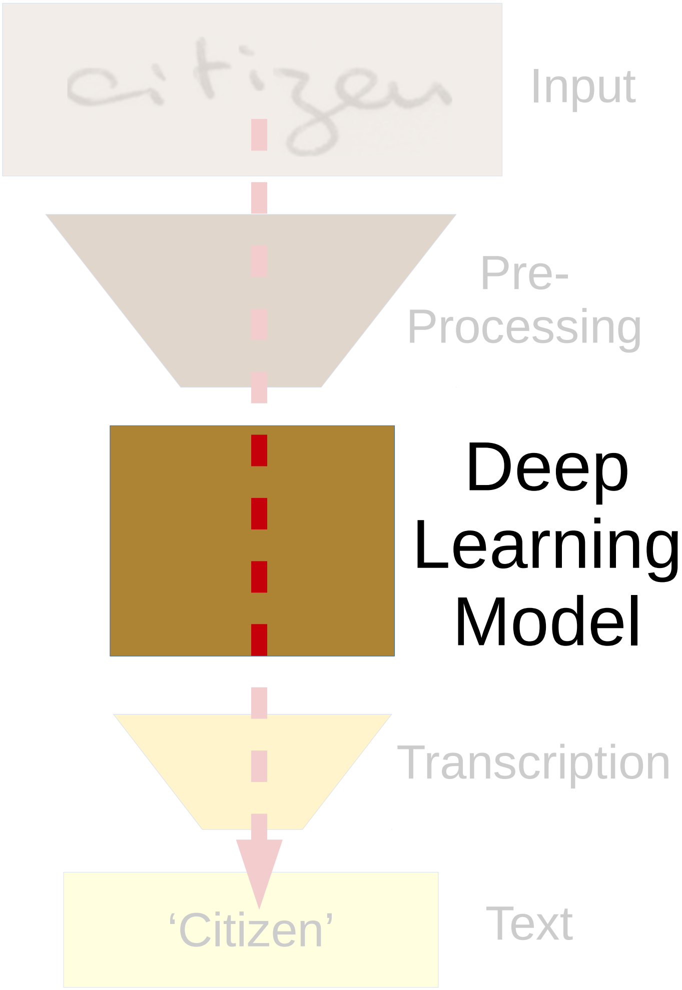




Since the RNNs take data in which the ‘Time Dimension’* is the first dimension, the dimensions for the images were flipped.

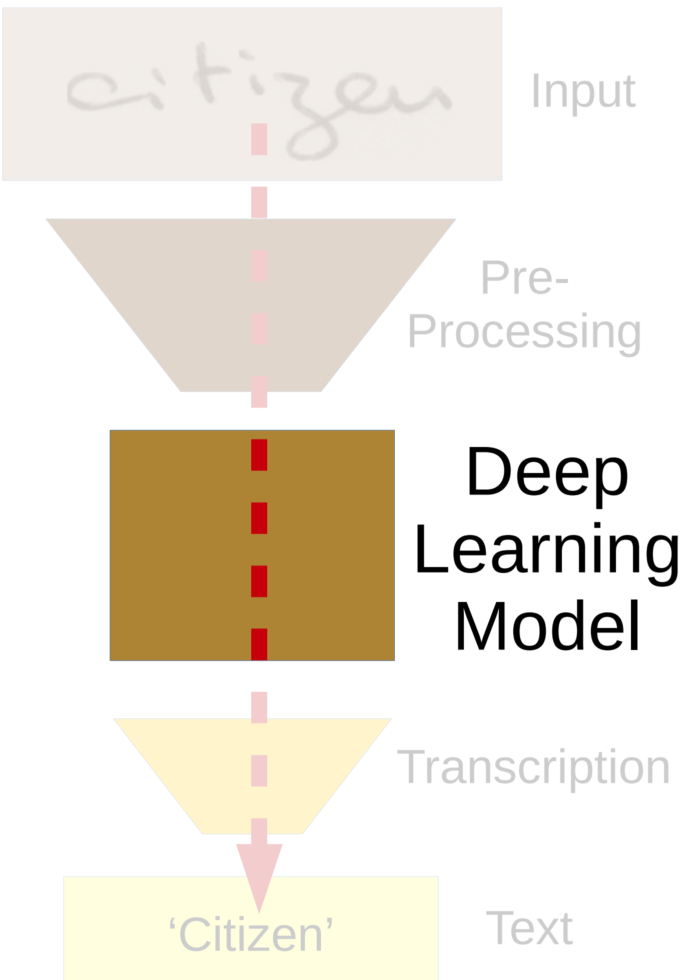
*See https://keras.io/layers/recurrent/

Black Box Illustration

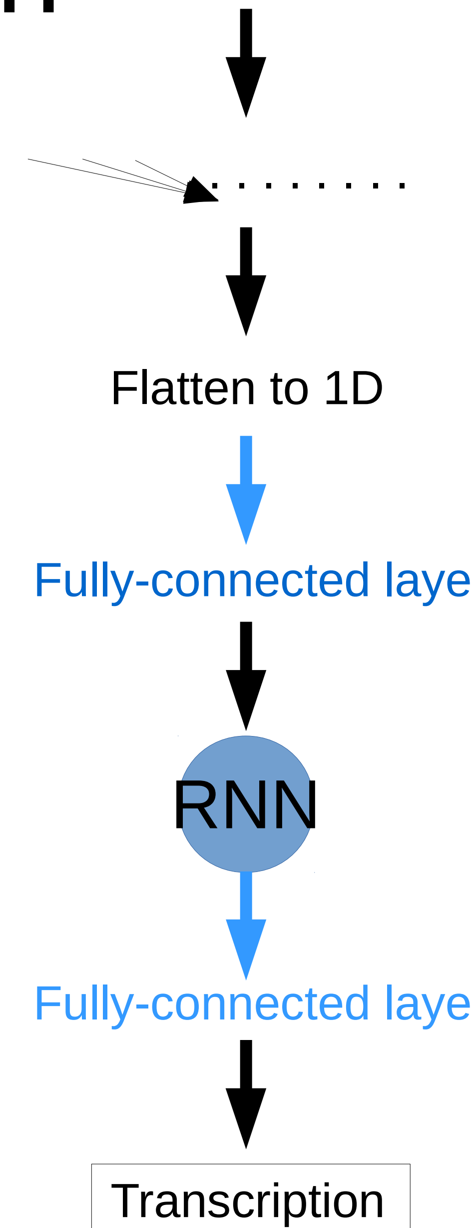


Since the paper was coded in Torch I looked for any implementations that used Tensorflow and ideally the Keras frontend. While the architecture was relatively easy to understand, the puzzle piece missing was how to calculate the loss function for this type of output.

Black Box Illustration

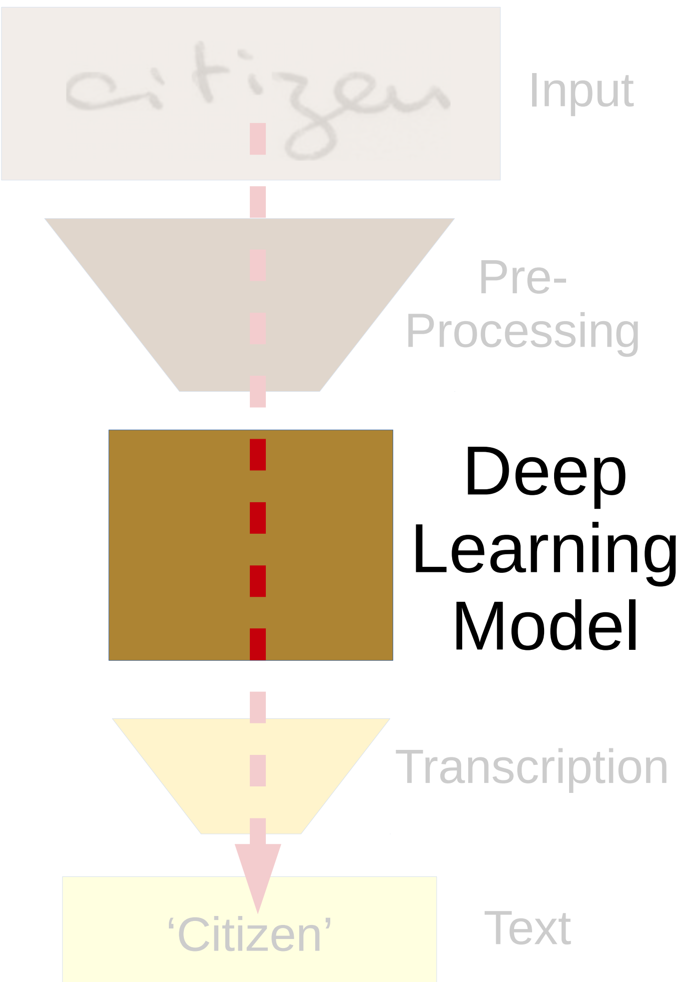


I found the keras-image-ocr repository* which (as a demonstration) used a similar architecture.



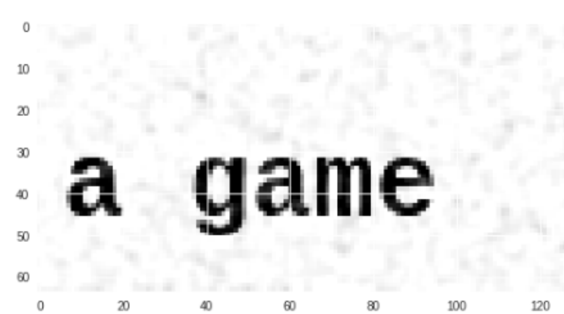
*https://github.com/Tony607/keras-image-ocr

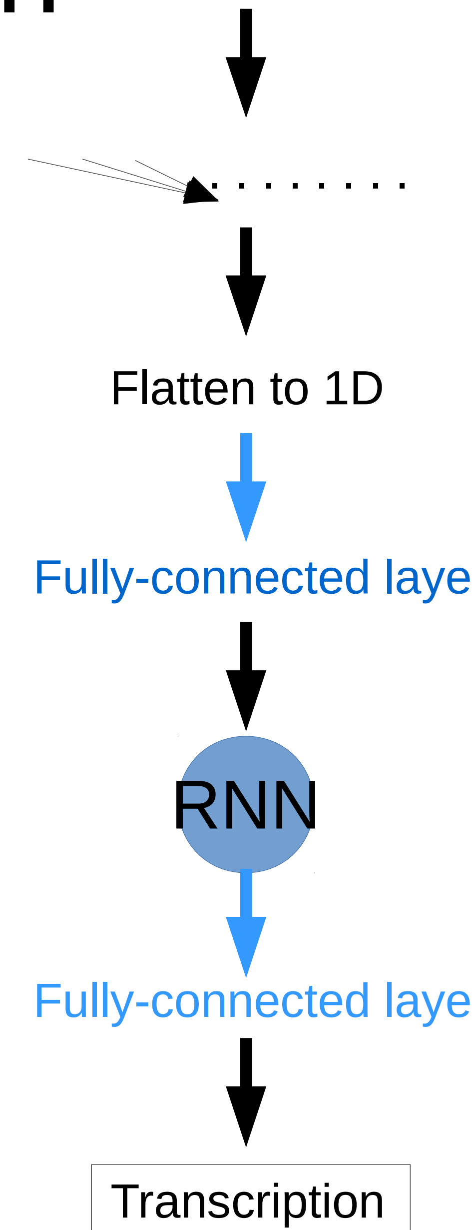
Black Box Illustration



I found the keras-image-ocr repository which (as a demonstration) used a similar architecture. This demo used Automatically generated images to train the network.

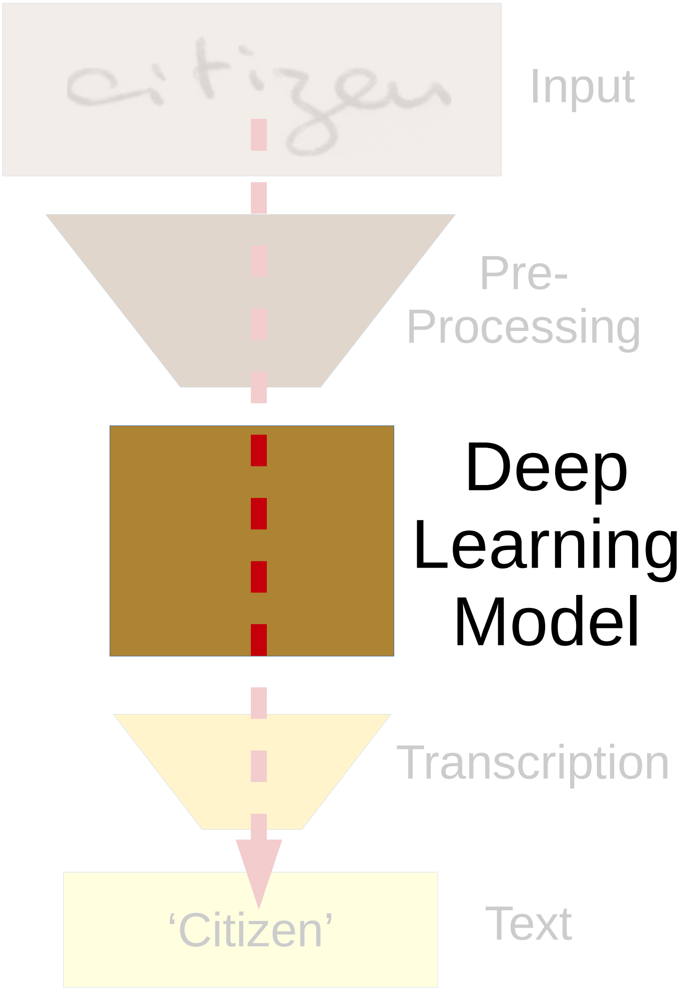
Example image*:





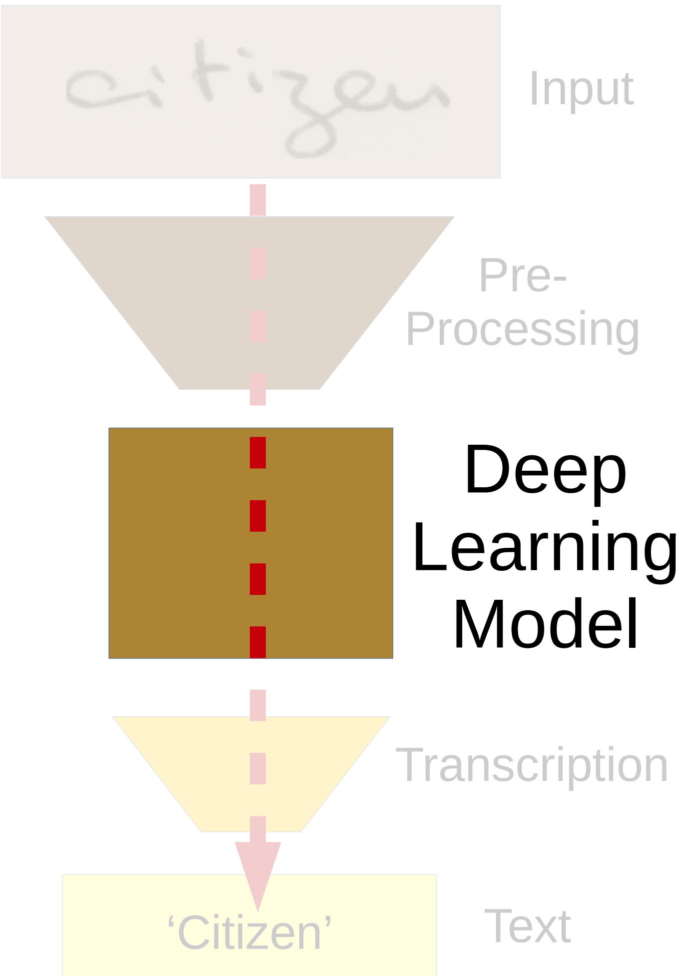
*https://github.com/Tony607/keras-image-ocr/blob/master/image-ocr.ipynb

Black Box Illustration



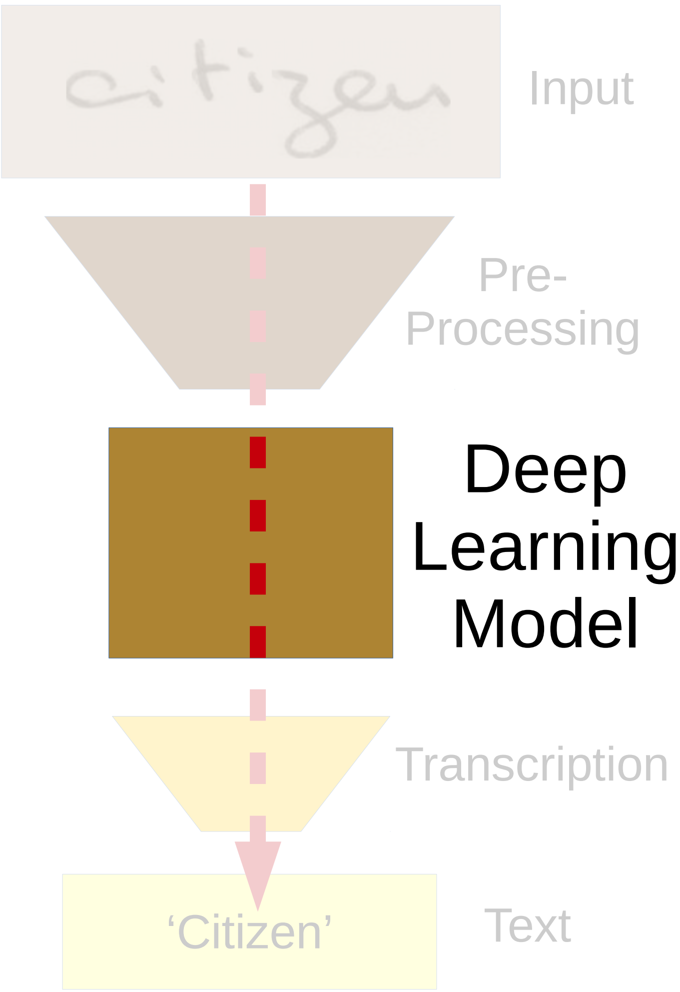
Most importantly it used a working implementation of the CTC loss function.

Black Box Illustration



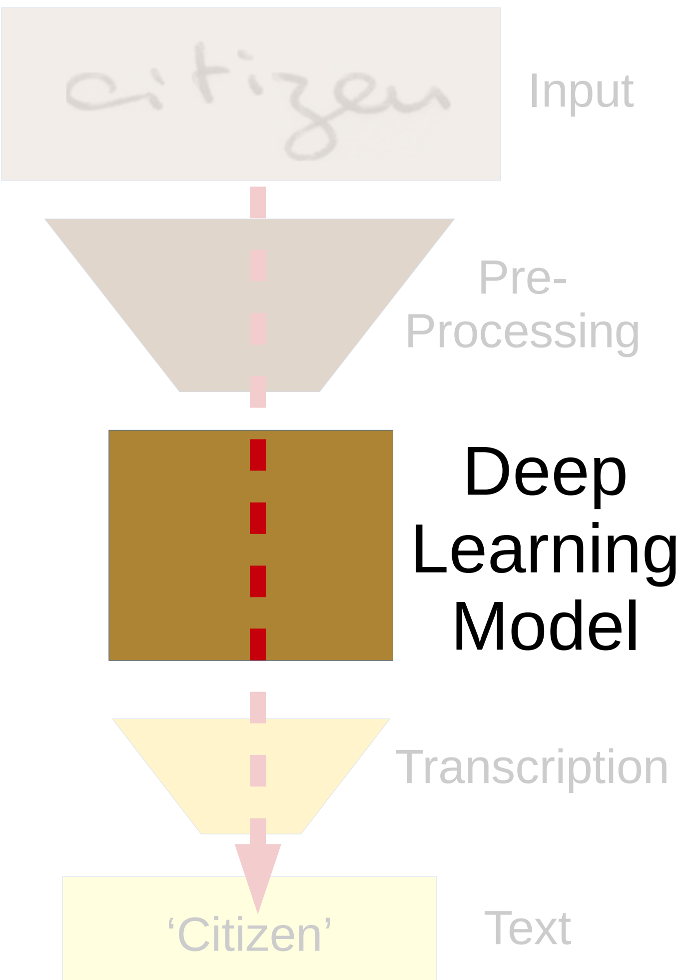
The CTC loss function is able to calculate Loss for unsynchronized input and output.
Since every letter is not the same number of pixels wide, you cannot apply a n-number of pixels per letter algorithm.
This is where CTC comes into play. It was originally developed for audio data, but works for any kind of sequential data.
I won't go into details as to how it works, see the paper at http://www.cs.toronto.edu/~graves/icml_2006.pdf
for more. (warning: Math!)

Black Box Illustration



While CTC Loss has been implemented in Tensorflow, the documentation is not particularly clear, so I had to take the data generator from the demo apart to see the data it was generating, then format the IAM dataset into that format.

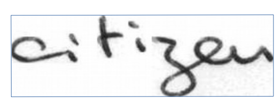
Black Box Illustration



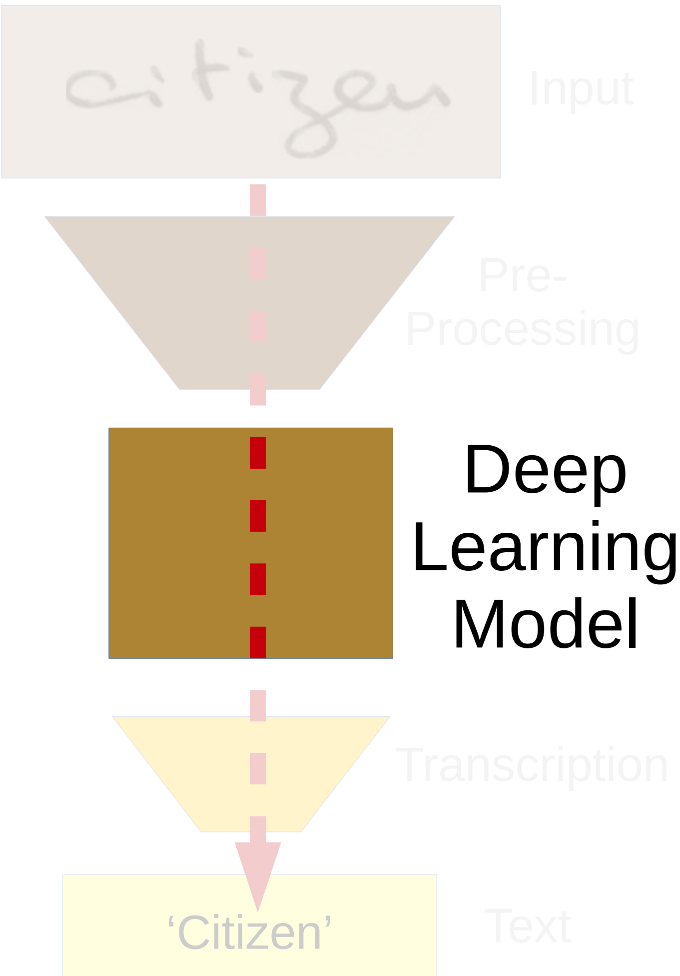
CTC loss requires for each sample:

- the output Data from the model,
- Labels encoded in integer form and padded with an encoding for ‘blank’ to a certain max length,
- The size of the input in the time dimension,
- The length of the label without the ‘blank’ padding.

For example:

 (processed through the network),
‘[5, 11, 22, 11, 28, 7, 16,-1,-1,-1]’,
30 pixels,
7 letters long

Black Box Illustration



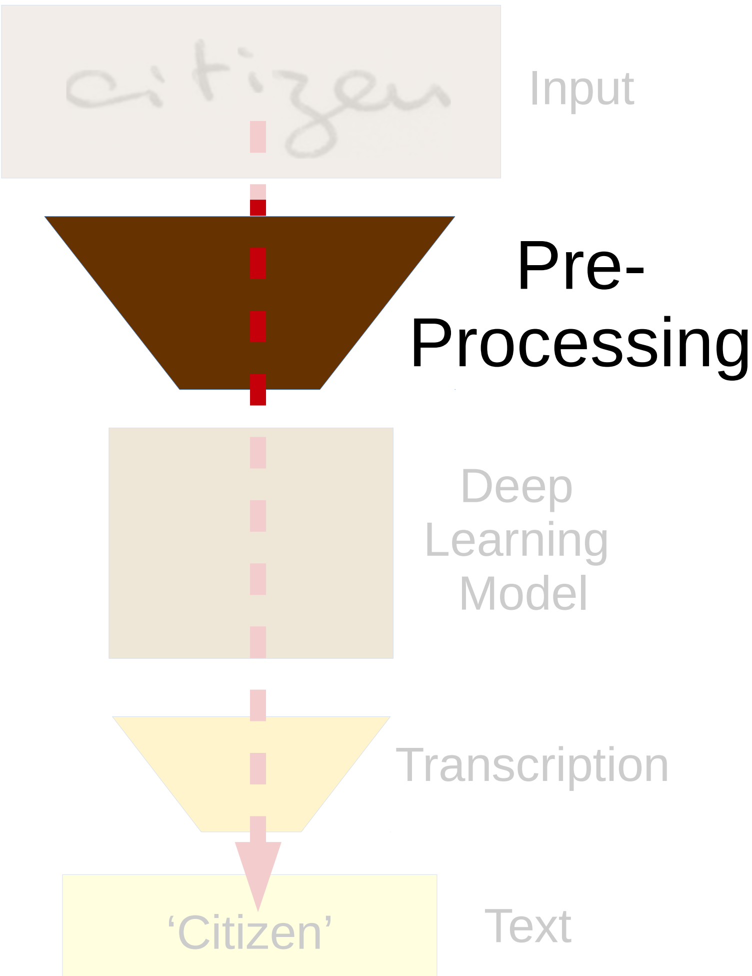
Now the model can process images of variable width, however during training, image dimensions have to be defined, this is a restriction since gpu memory has to be reserved as well as computation optimizations needing a defined input format.

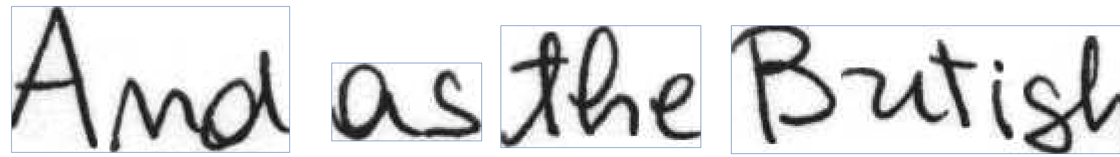
Solution: Padding?

Nope!

Back to data Pre-processing!

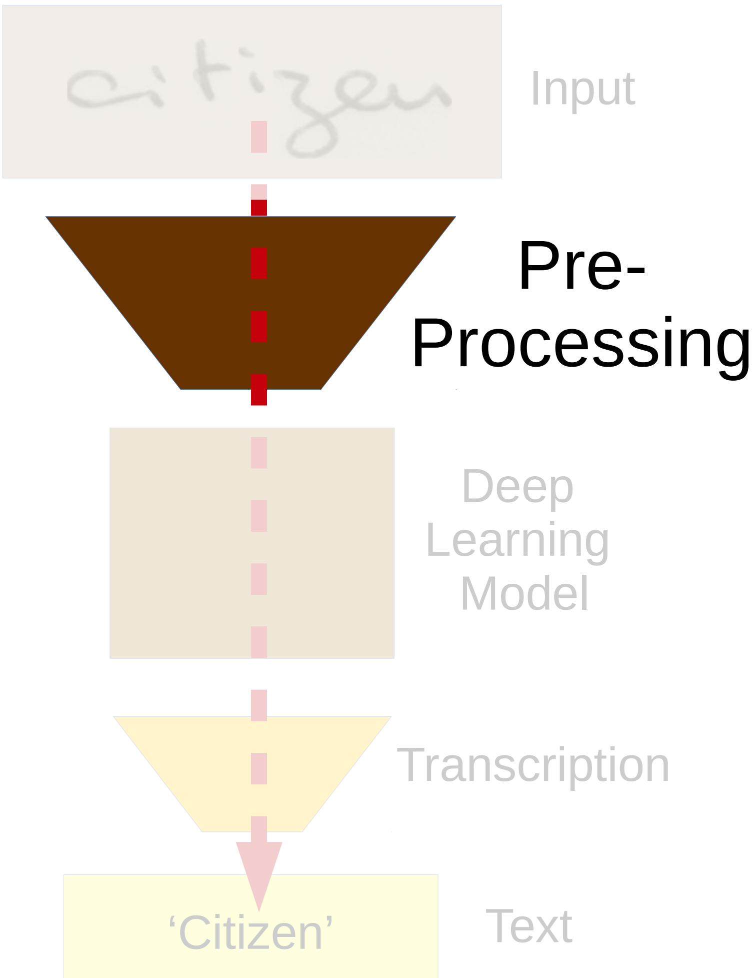
Buckets!





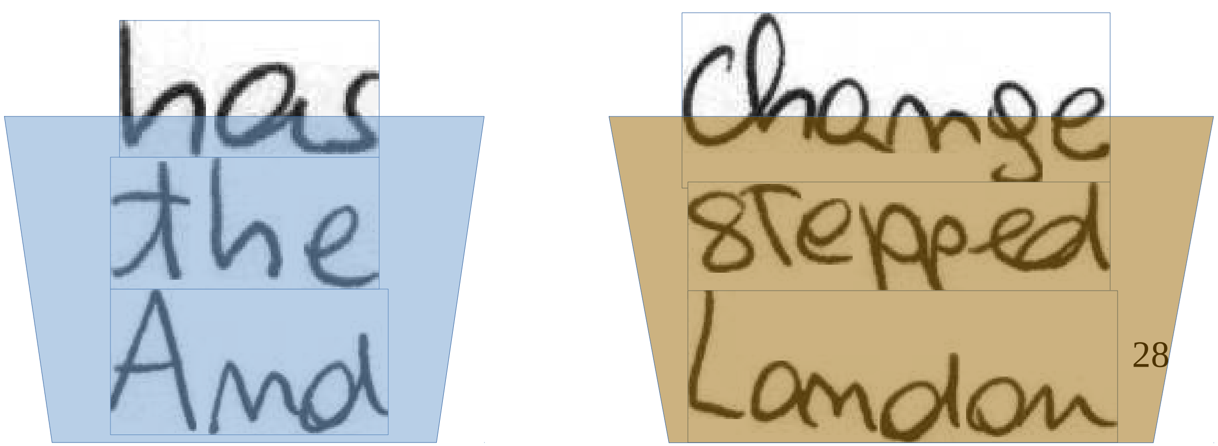
Padding was not necessary since **Bucketing** was implemented. **Bucketing** is the grouping of images based on a criterion, in this case <u>**Image Length**</u>. Image height was already normalized as explained before.

Buckets!

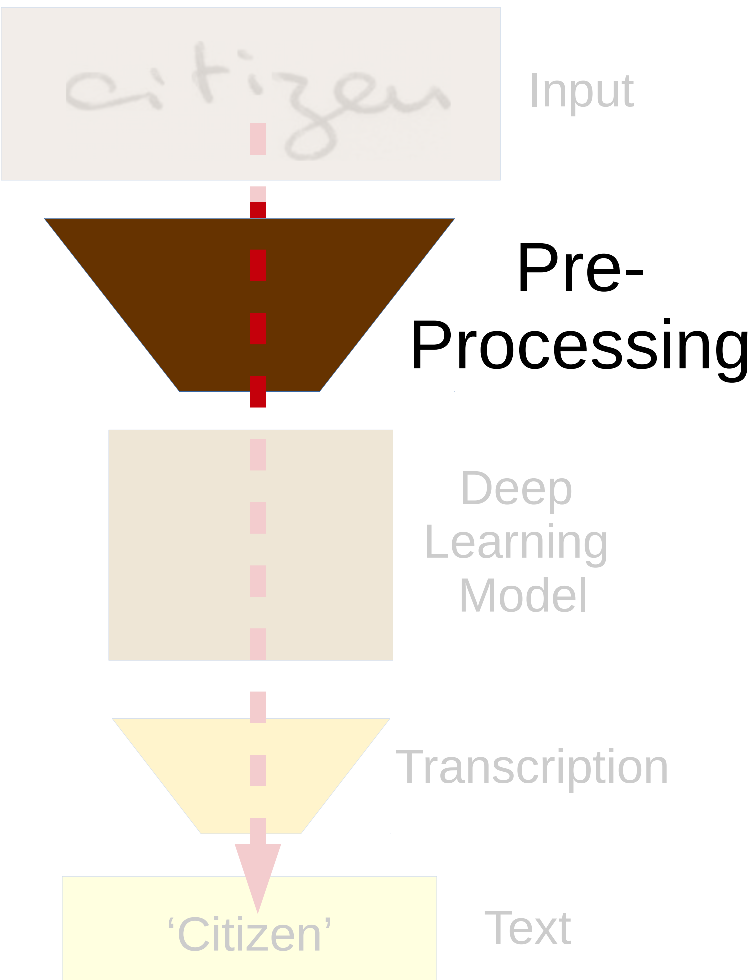


Now we can feed a bucket at a time to the model and since all the images have the same width, the model can be trained on a bucket at a time.

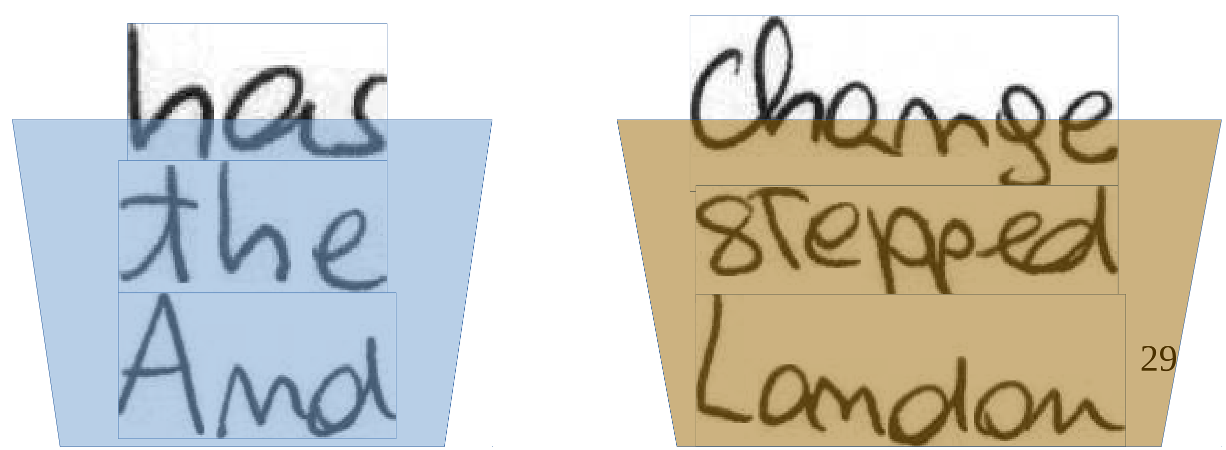
All that is left is writing a generator which supplies the model with a bucket at a time. This can probably be improved with Queues, but it works well enough for now.



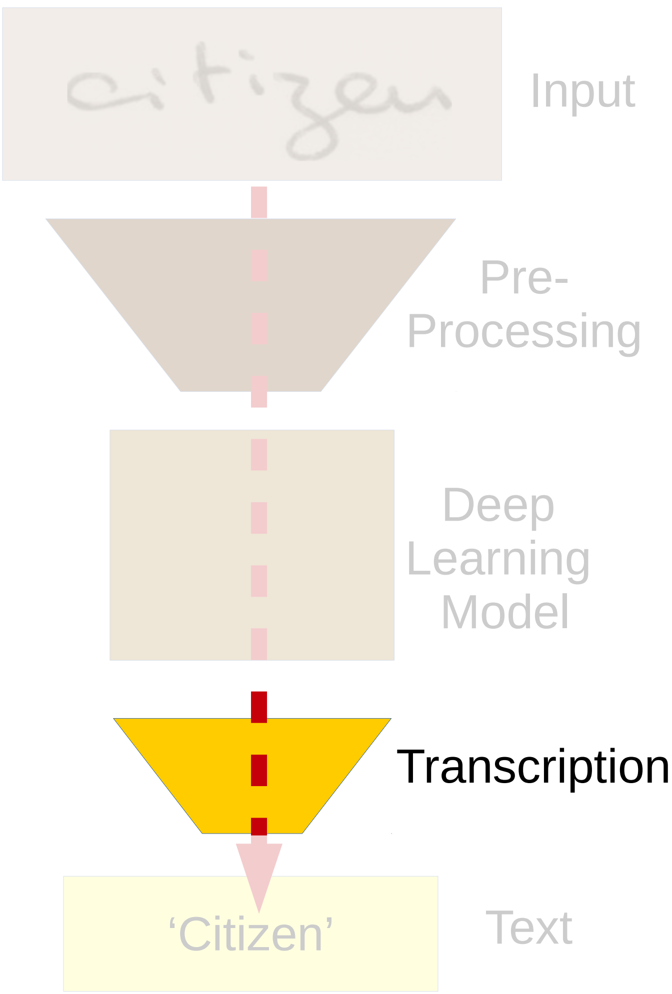
Buckets!



One last thing about the generator: Keras runs the generator in the background on the CPU, while the Model is training. This saves time by pre-fetching data. So even if the generator only supplies a single bucket,



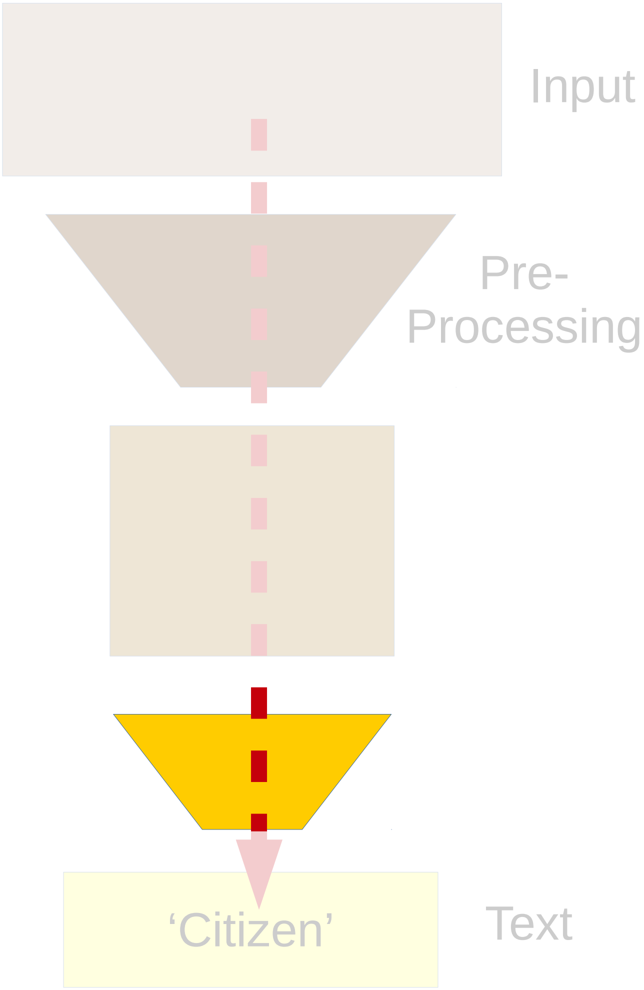
Black Box Illustration



Transcription is done via a CTC decode function from the Keras frontend, which uses Beam Search.

Beam search is a transcription method where the most probable paths are explored and the highest scoring end path is used.

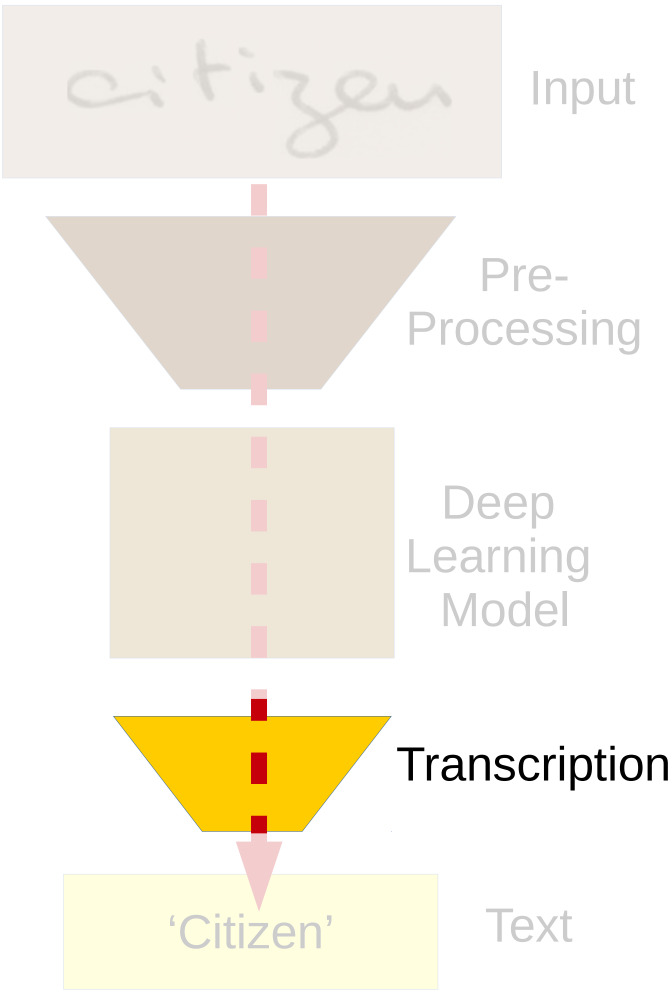
Beam Search, top_paths = 3

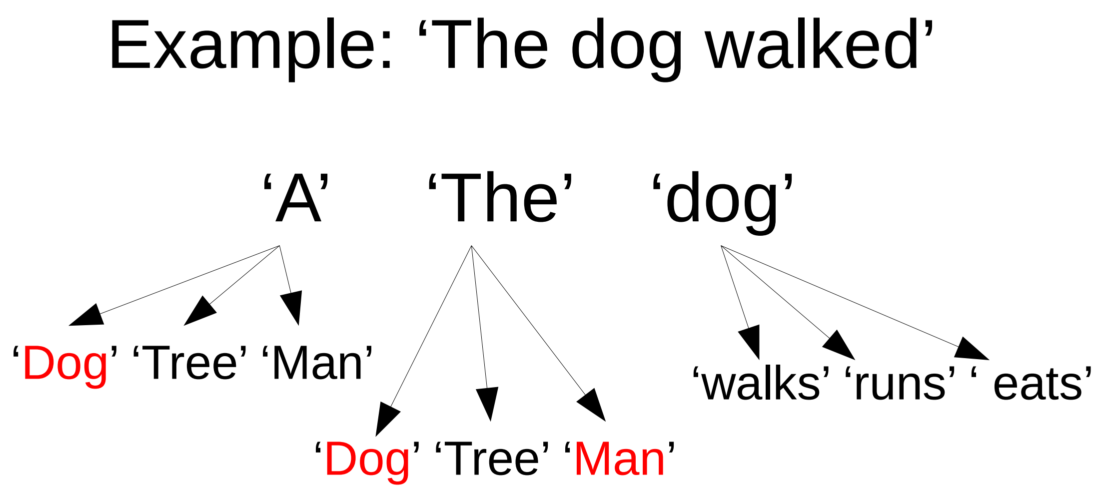


Example: 'The dog walked'

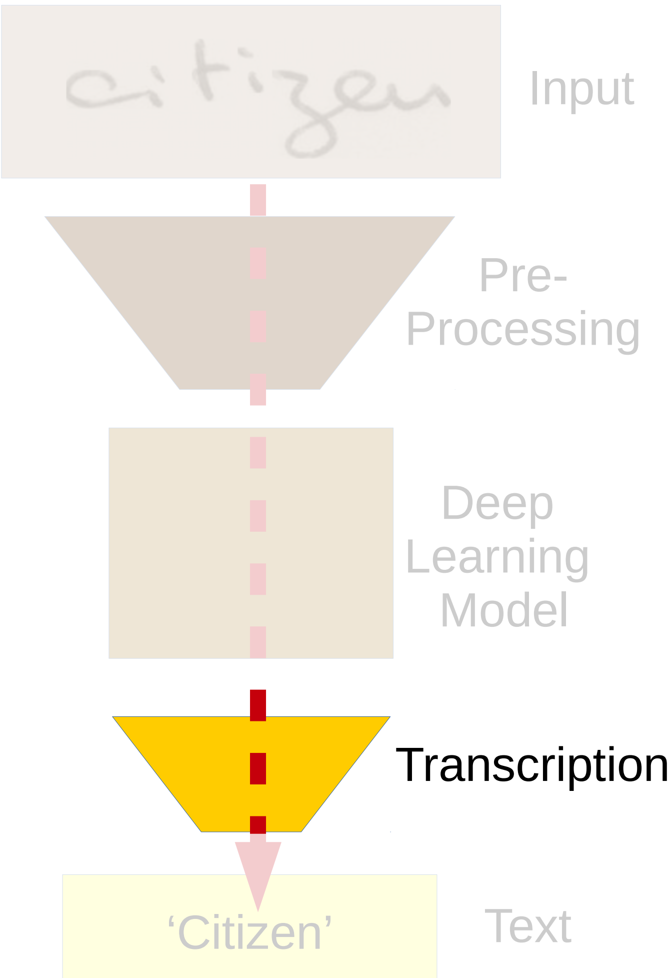
'A' 'The' 'dog'

Beam Search, top_paths = 3

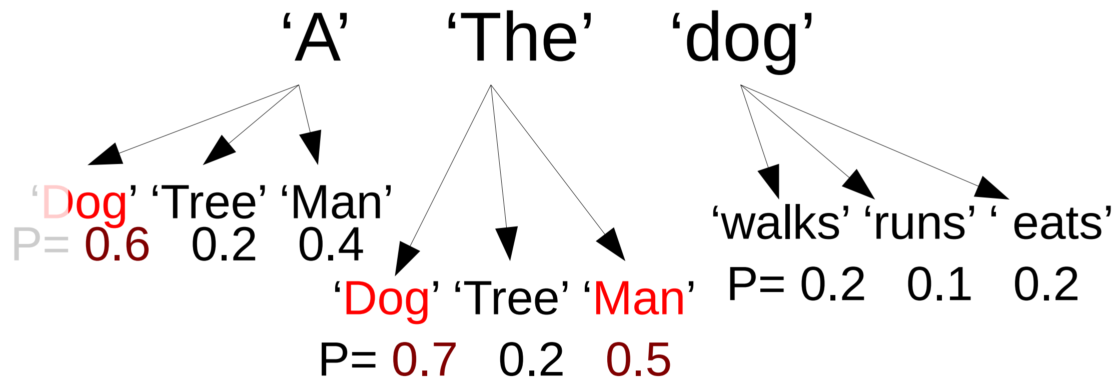




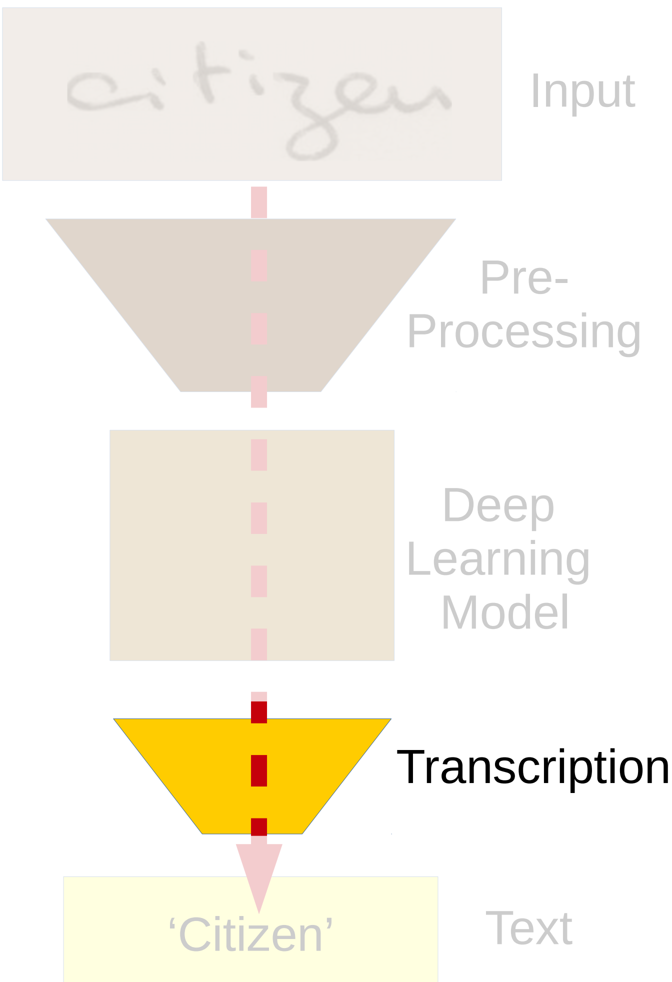
Beam Search, top_paths = 3



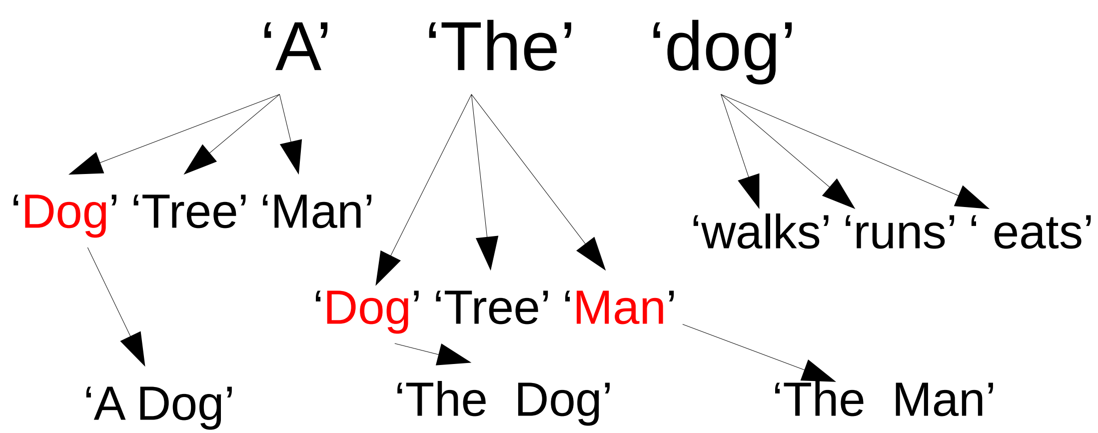
Example: 'The dog walked'



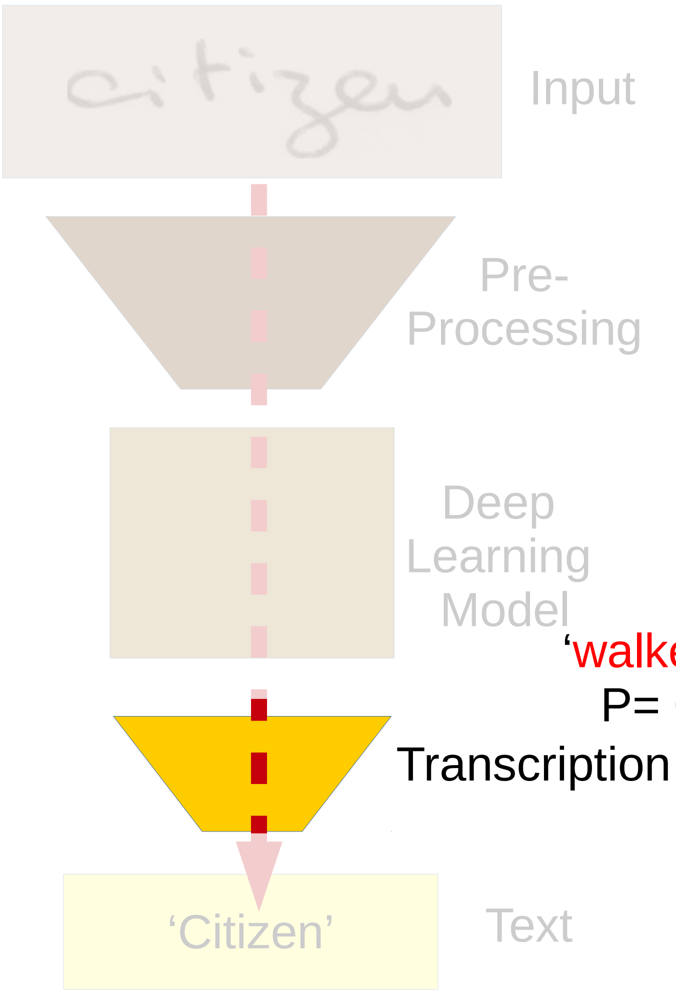
Beam Search, top_paths = 3



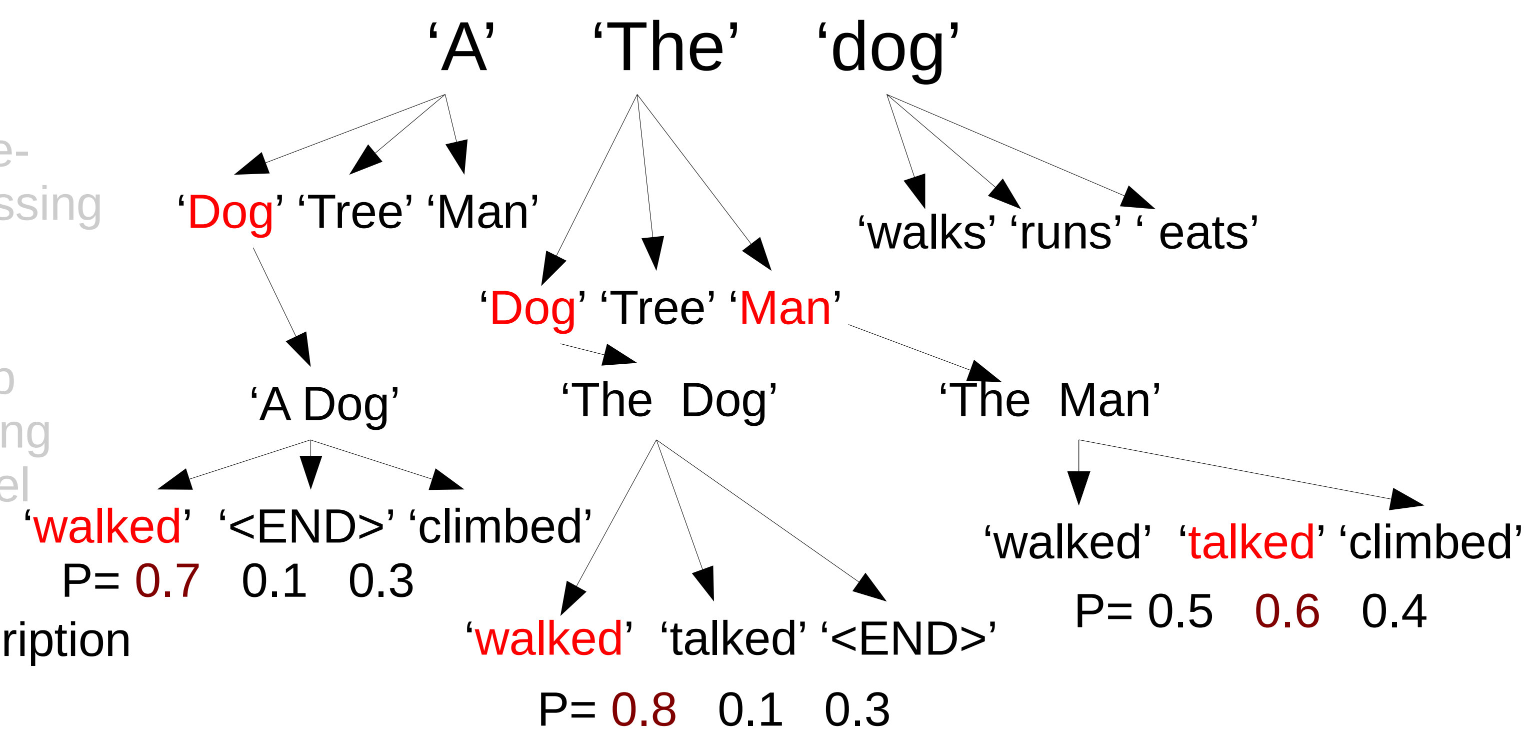
Example: 'The dog walked'



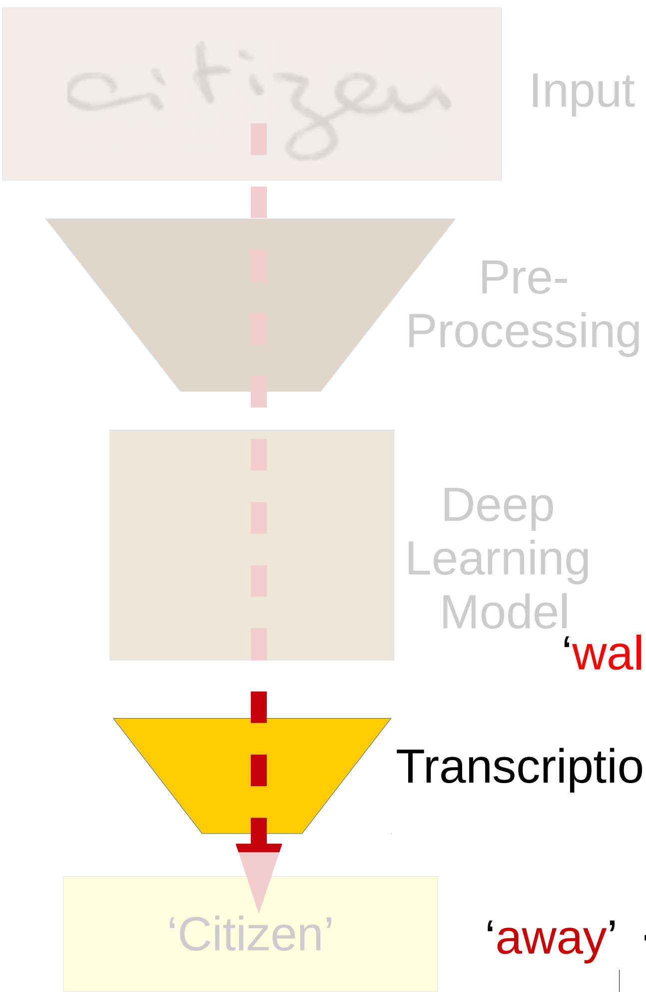
Beam Search, top_paths = 3



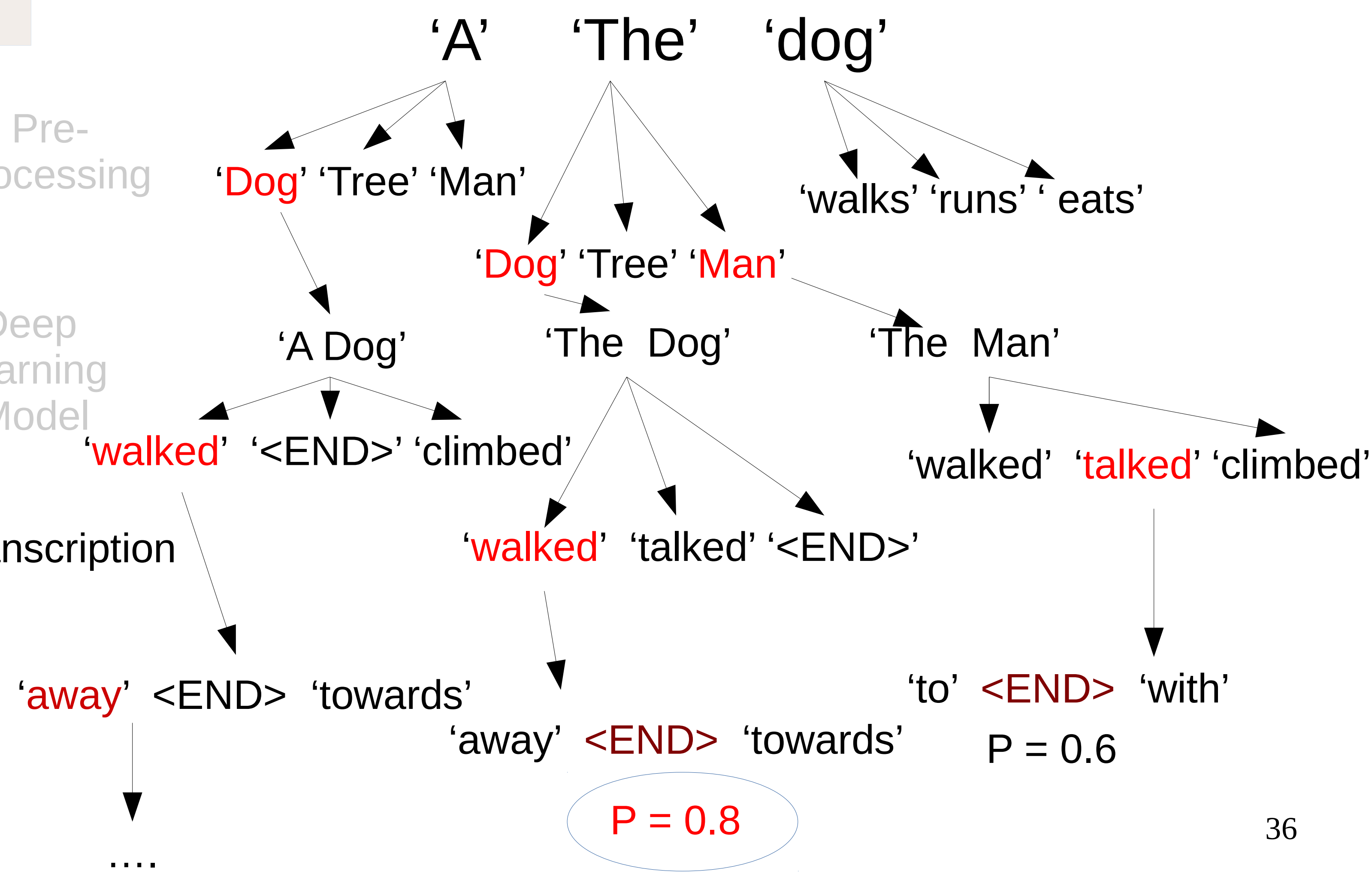
Example: 'The dog walked'



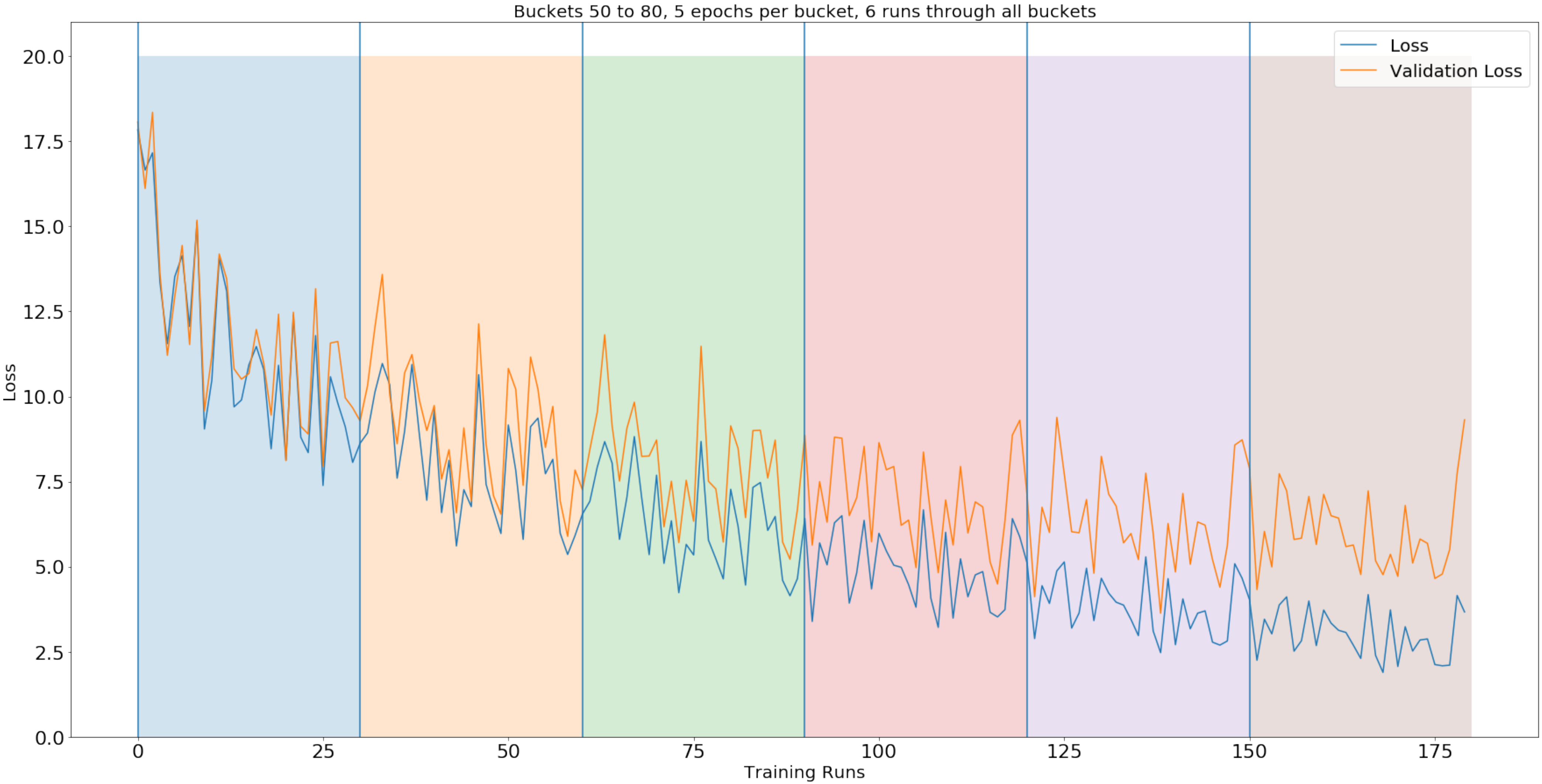
Beam Search, top_paths = 3



Example: ‘The dog walked’



Training

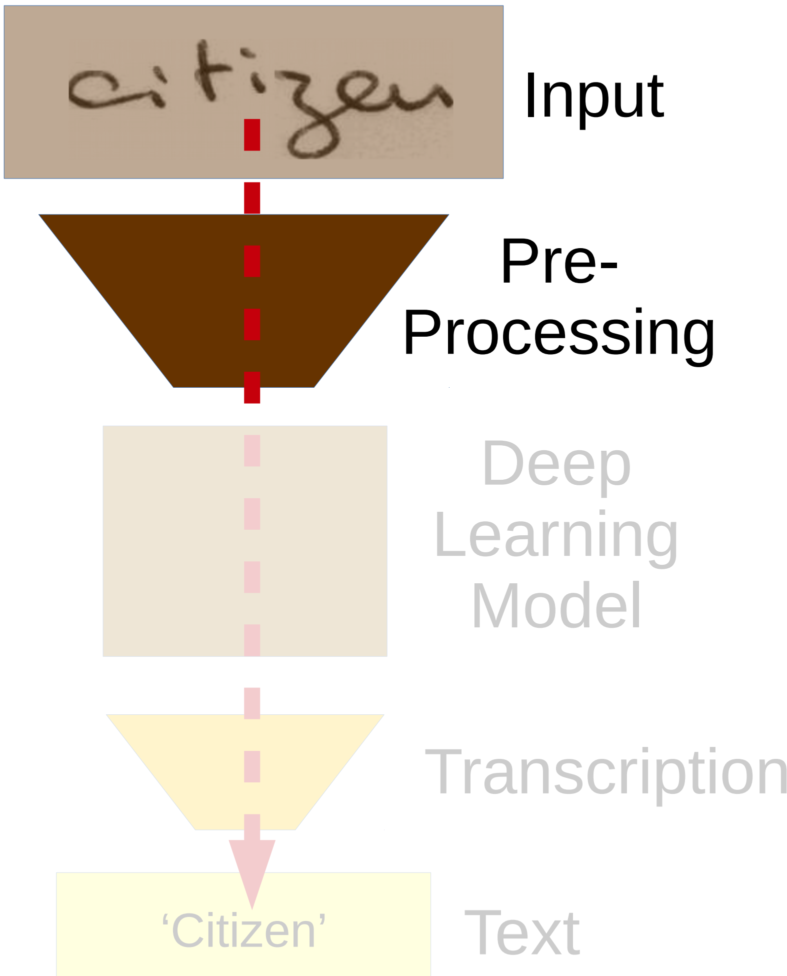


Small bucket number causes the model to quickly begin to overfit

Training

Only 70 of 100 buckets were used, since 30 did not contain enough data for validation. Larger diversity of data and fewer epochs combat overfitting.

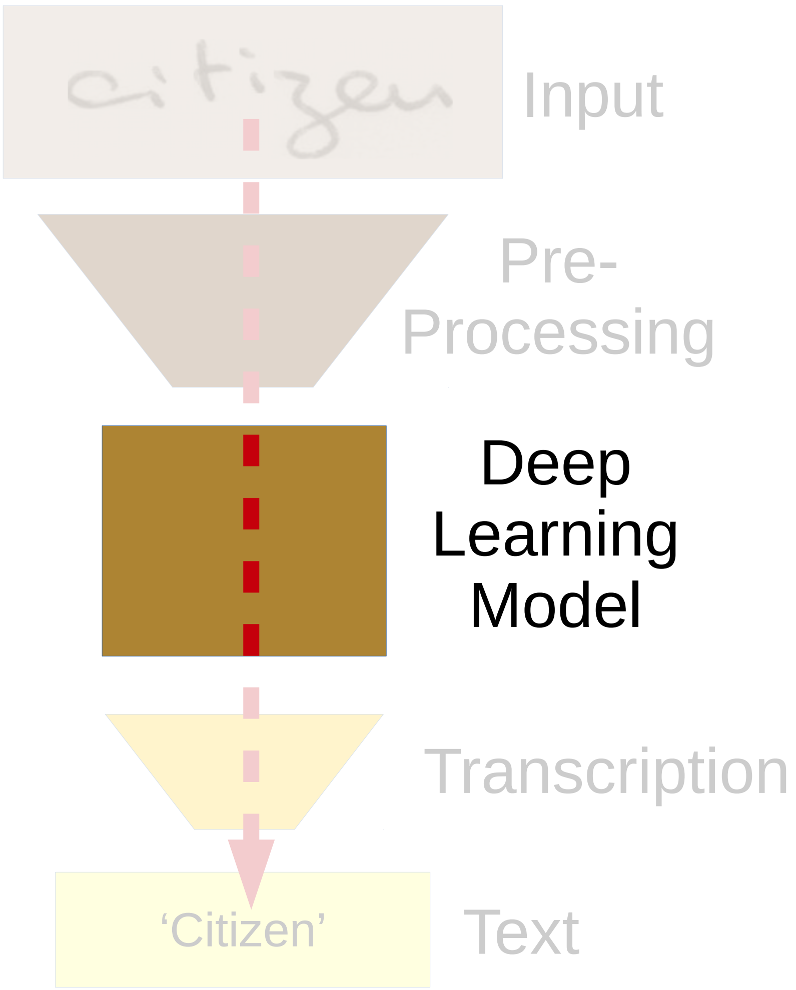
QUESTIONS ?



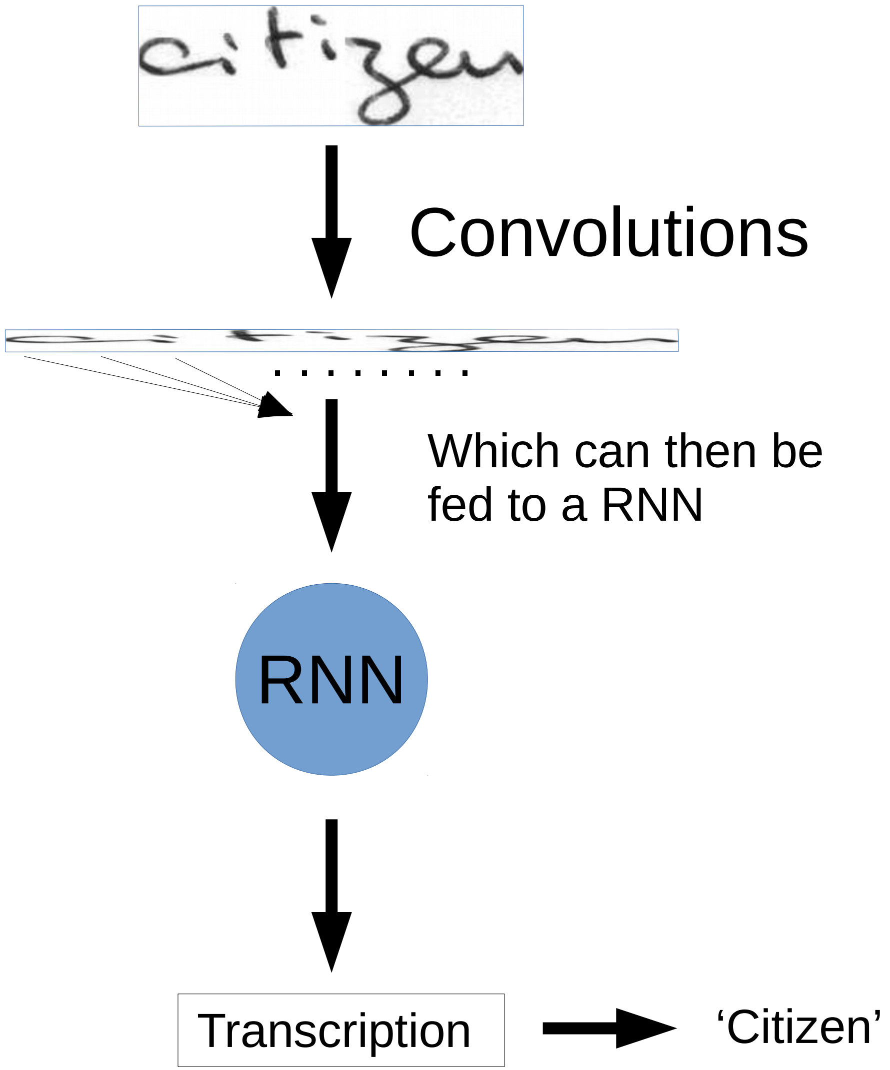
citizen
Input
Pre-Processing
Deep Learning Model
Transcription
'Citizen'
Text

British
And
the British
has
the
And

QUESTIONS ?

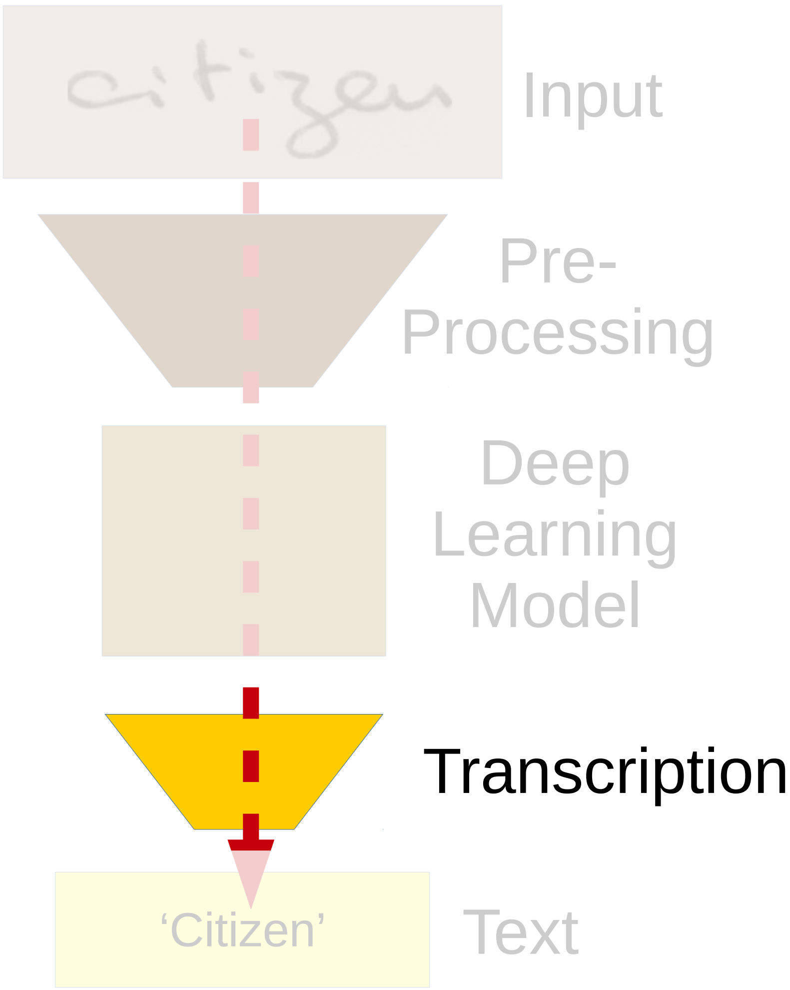


Input
Pre-
Processing
Deep
Learning
Model
Transcription
'Citizen'
Text



Convolutions
Which can then be
fed to a RNN
RNN
Transcription
'Citizen'

QUESTIONS ?



Beam Search