White Hat: Hacking Passwords Using ML

Image source: https://www.bearfoxmarketing.com/white-hat-link-building/

tikka (Tikeswar Naik)
28 Oct 2019
tikeswar@gmail.com

Problem Statement

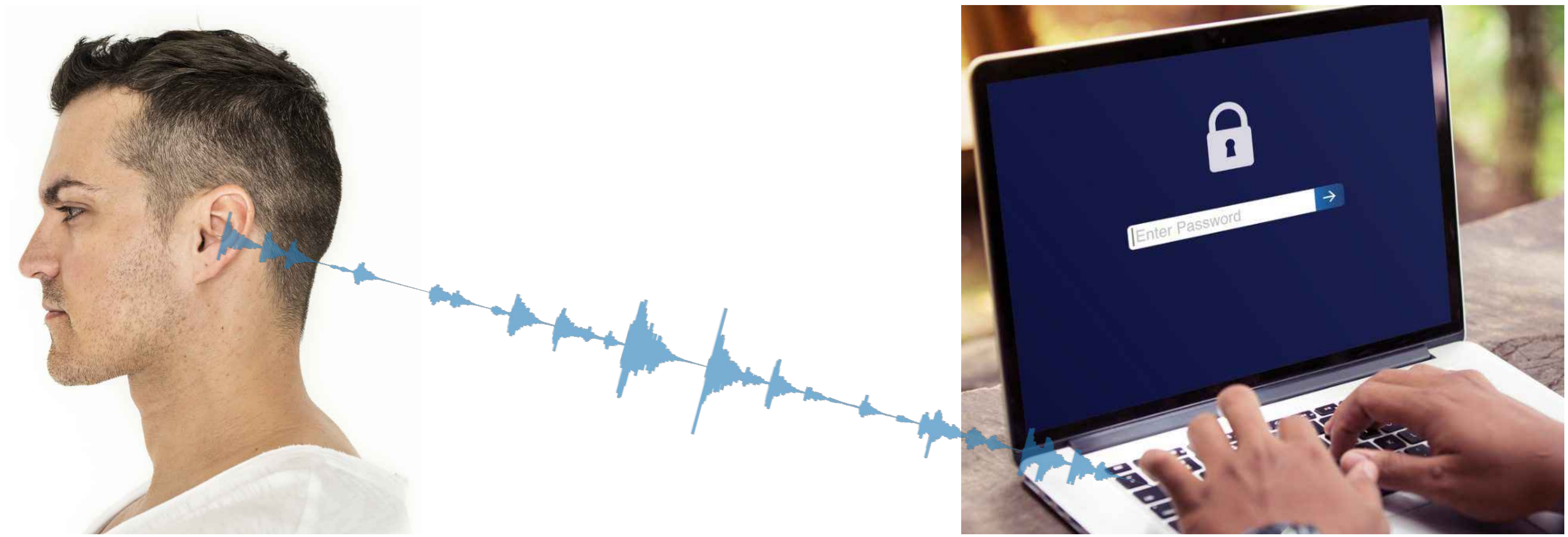


Image source: www.rawpixel.com
through https://www.freepik.com/free-photo/side-portrait-white-man_2825703.htm

Image source: https://www.eacs.com/is-it-time-to-rethink-the-password/

Can we figure out what someone is typing,
just by ***listening to the keystrokes***?

Motivations

- Security implications
  - Typing passwords while people are around, or during teleconferencing
  - Can someone hack your computer microphone and listen to what you type?

`kido` [= Keystroke Decode]
- a ML project to explore this idea

Image source:
https://www.masterfile.com/image/en/621-00746221/executives-teleconferencing

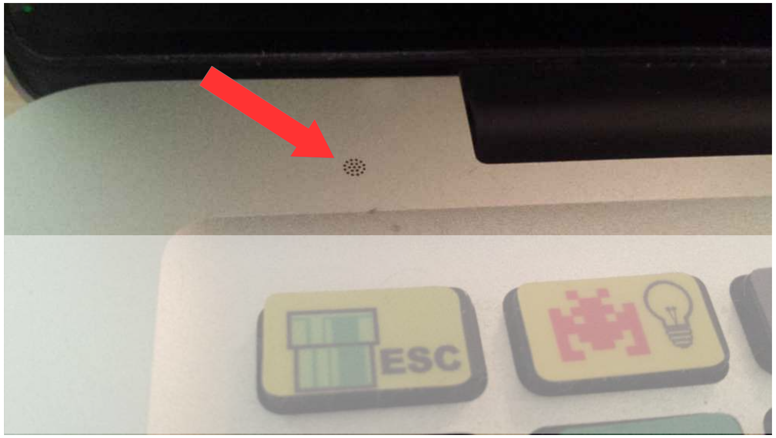


Image source:
https://www.quora.com/Where-is-the-built-in-microphone-located-on-a-MacBook-Pro

Outline

- Data Gathering, and Preparation
- Training and Eval
- Testing and Error Analysis
  - Reality check …
  - How to improve model accuracy?
- Final Remarks; Links

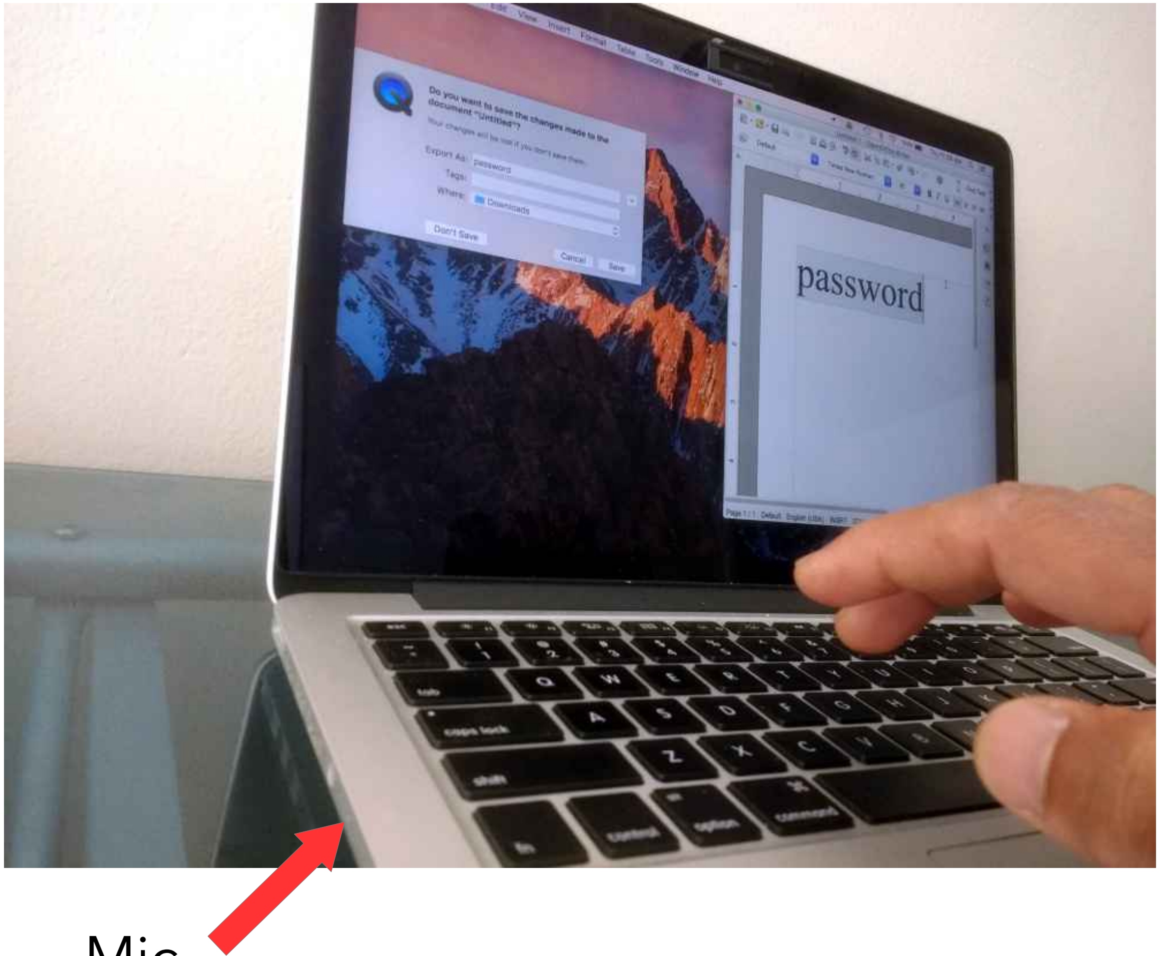
Data Gathering

Used my MacBook Pro:

- Keyboard to type
- QuickTime Player to record audio of typing through the inbuilt mic

Advantages:

- Data has less variability
- And thus it helps us focus on proving (or disproving) the concept without much distraction



Mic

Data Prep

.m4a to .wav

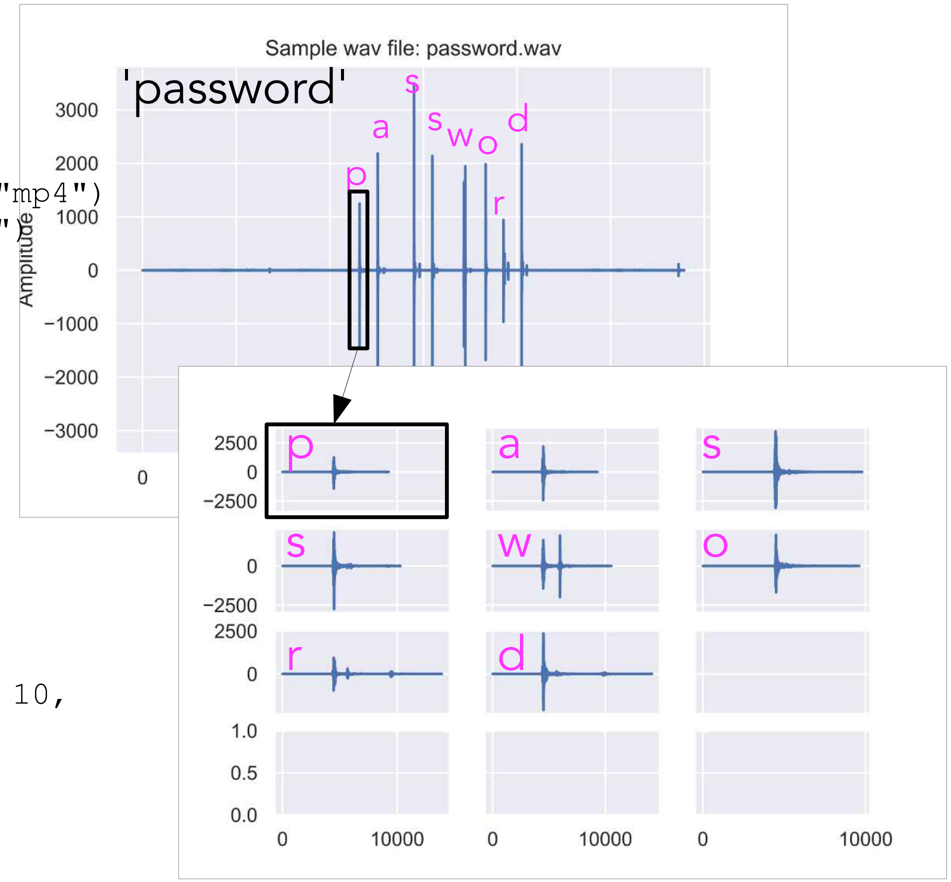
```
audio = AudioSegment.from_file(file, "mp4")
audio.export(file[:-4] + ".wav", "wav")
```

password
.wav

Split

```
chunks = split_on_silence(
        audio,
        min_silence_len = 100,
        silence_thresh = audio.dBFS - 10,
        keep_silence = 100
    )
```

p.wav
a.wav
…
d.wav



Data Prep

.m4a to .wav

```
audio = AudioSegment.from_file(file, "mp4")
audio.export(file[:-4] + ".wav", "wav")
```

password
.wav

Split

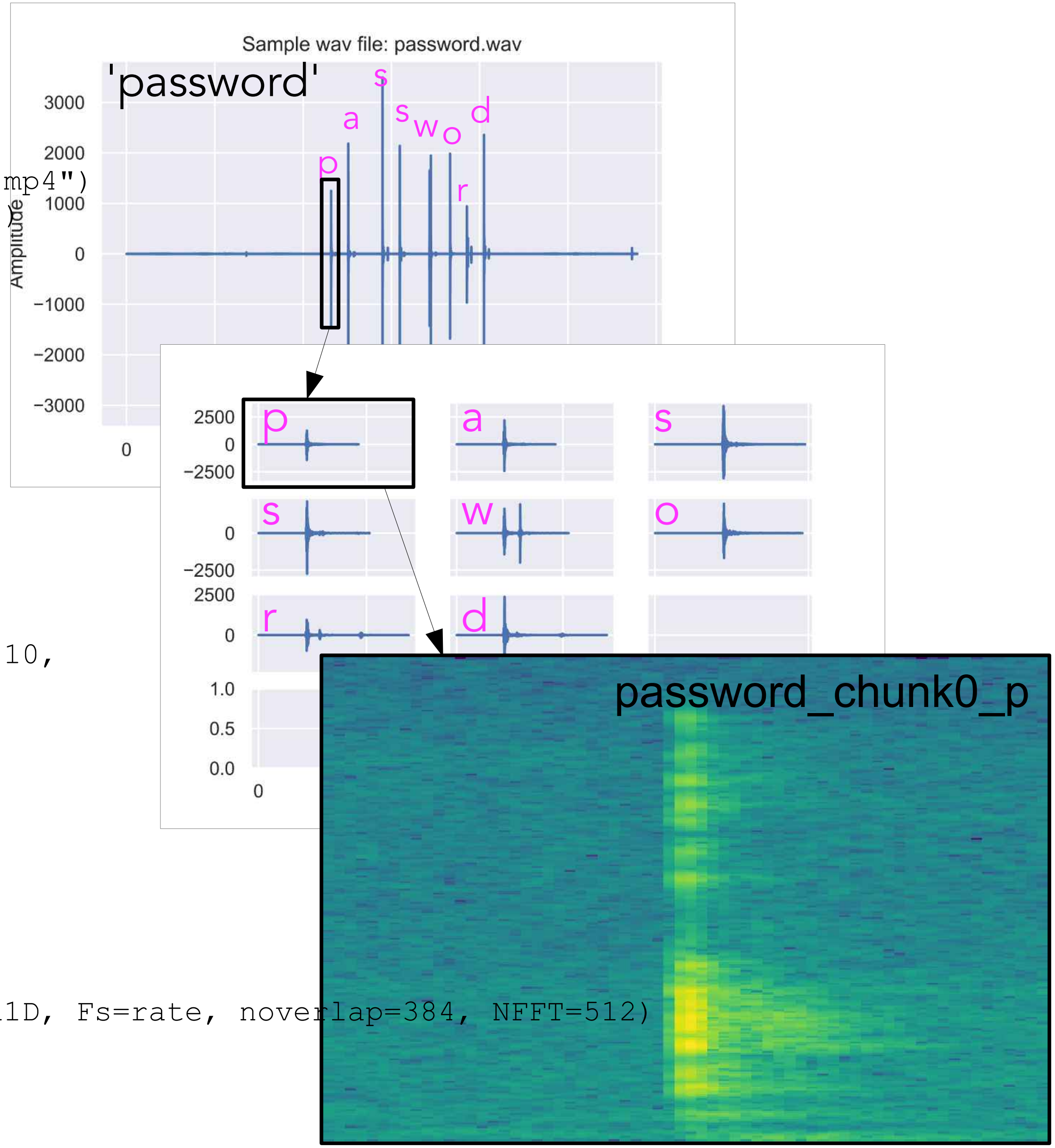
```
chunks = split_on_silence(
        audio,
        min_silence_len = 100,
        silence_thresh = audio.dBFS - 10,
        keep_silence = 100
    )
```

p.wav
a.wav
…
d.wav

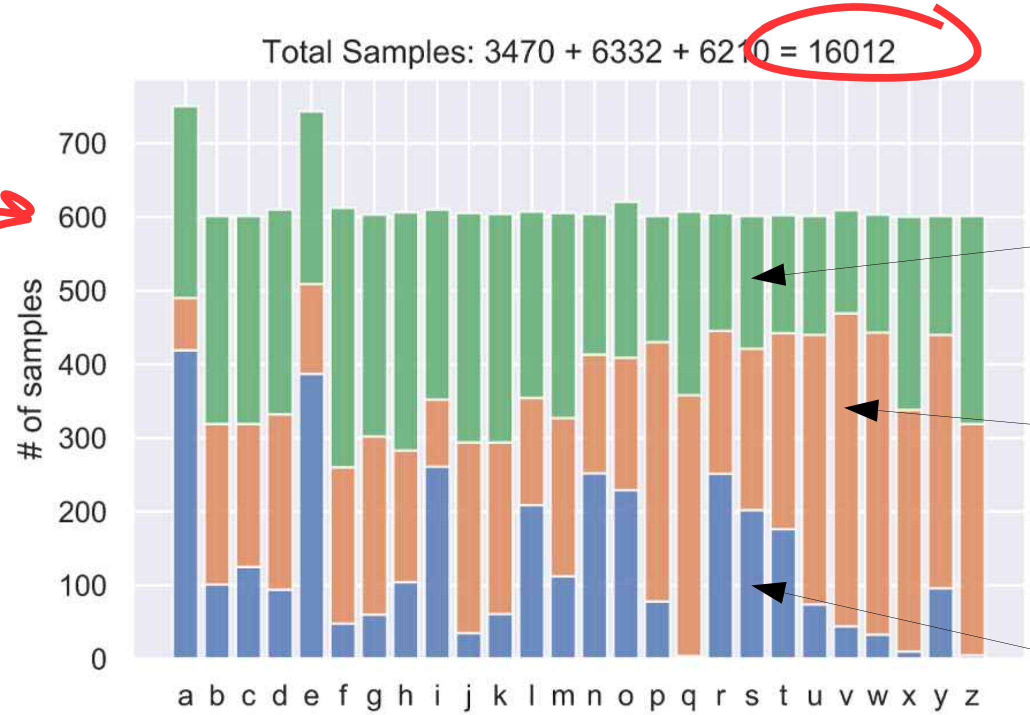
Spectrogram

```
rate, data = wavfile.read(wav_file)
data1D = data[:, 0]
fig, ax = plt.subplots(1)
s2d, f1d, t1d, im = ax.specgram(x=data1D, Fs=rate, noverlap=384, NFFT=512)
```

p, a, …, d
.png



Data Samples



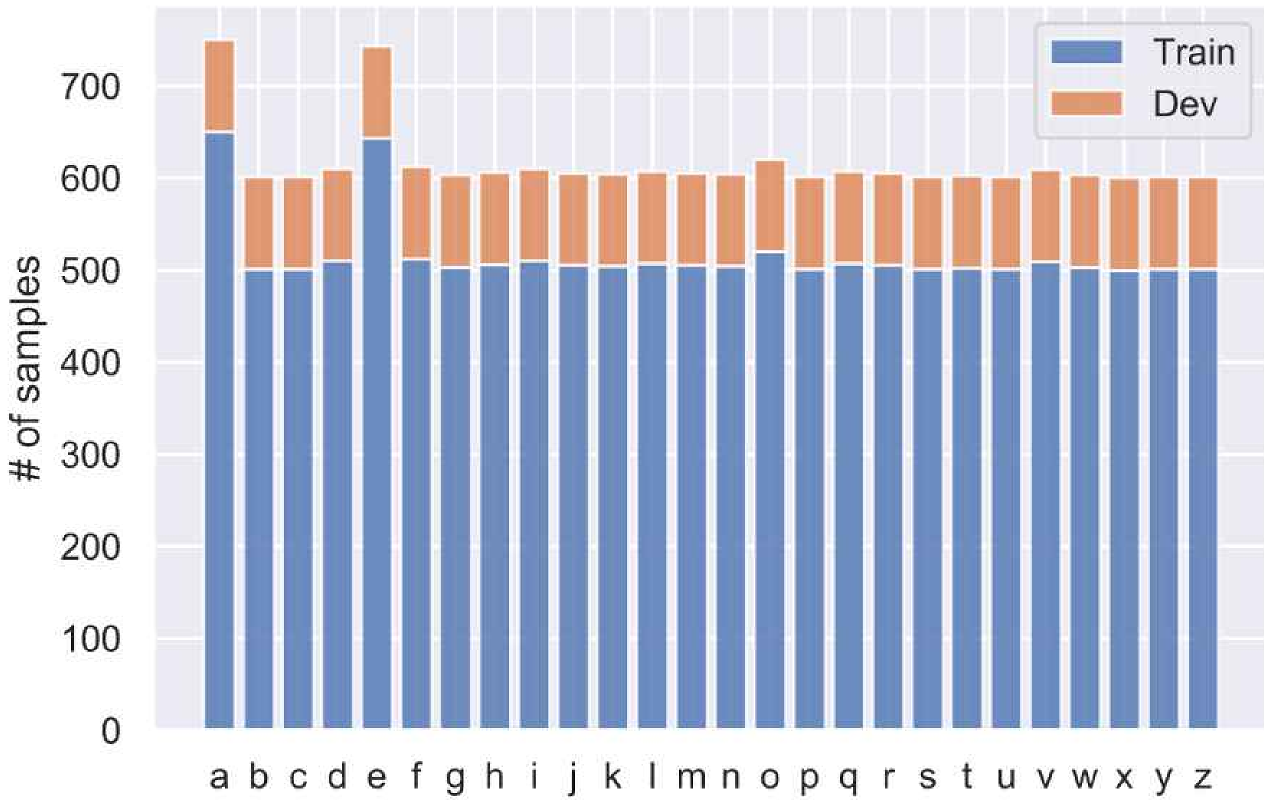
Typing letters individually
(e.g., 'mmmmmmmmmm')

Typing (random) texts
(e.g., 'fwtkuqwgmp')

From the 'rockyou' passwords
https://www.kaggle.com/wjburns/common-password-list-rockyoutxt#rockyou.txt
(e.g., 'cheese')

Total # of samples: ~ **16,000**
# of samples / letter: ~ **600**

Train-Validation Split



Validation (17%)

~ 100 / letter

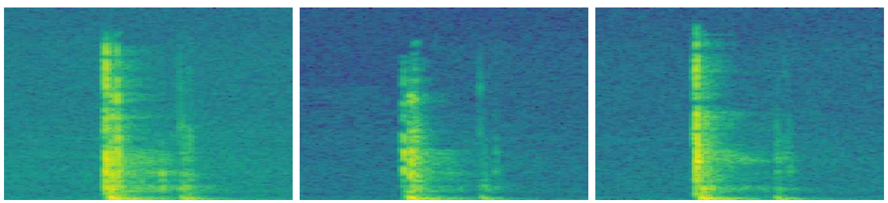
Training(83%)

~ 500 / letter

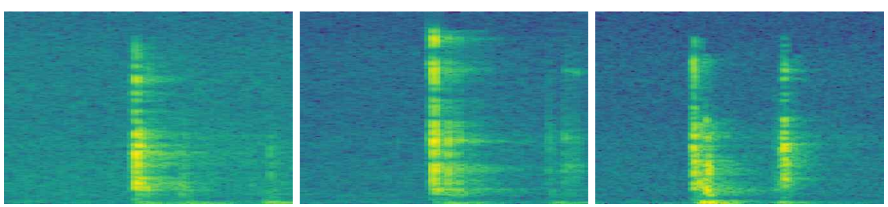
The ML Problem in a Nutshell …

Given	Guess?

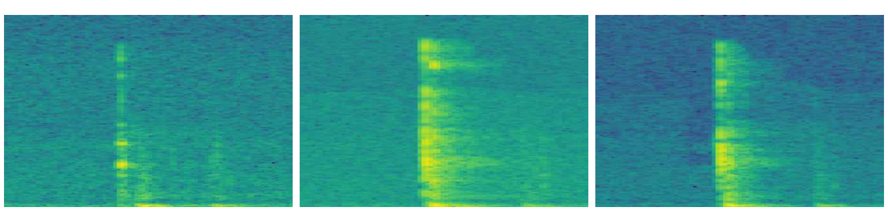
a

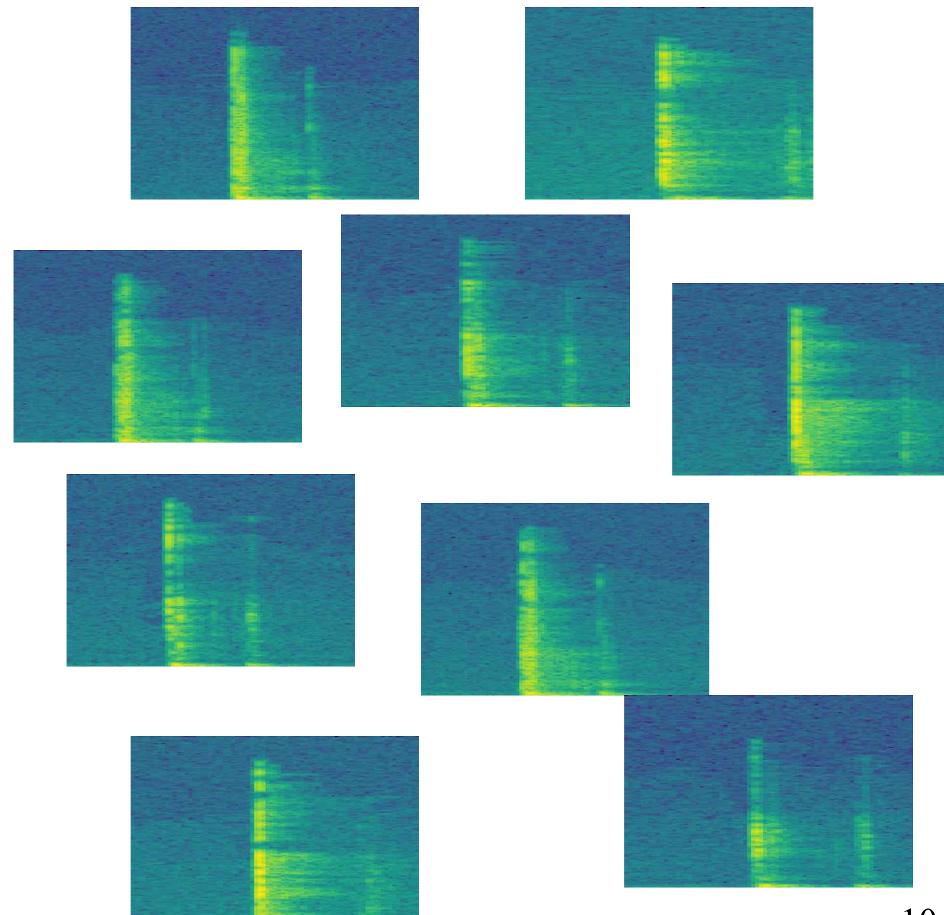


m

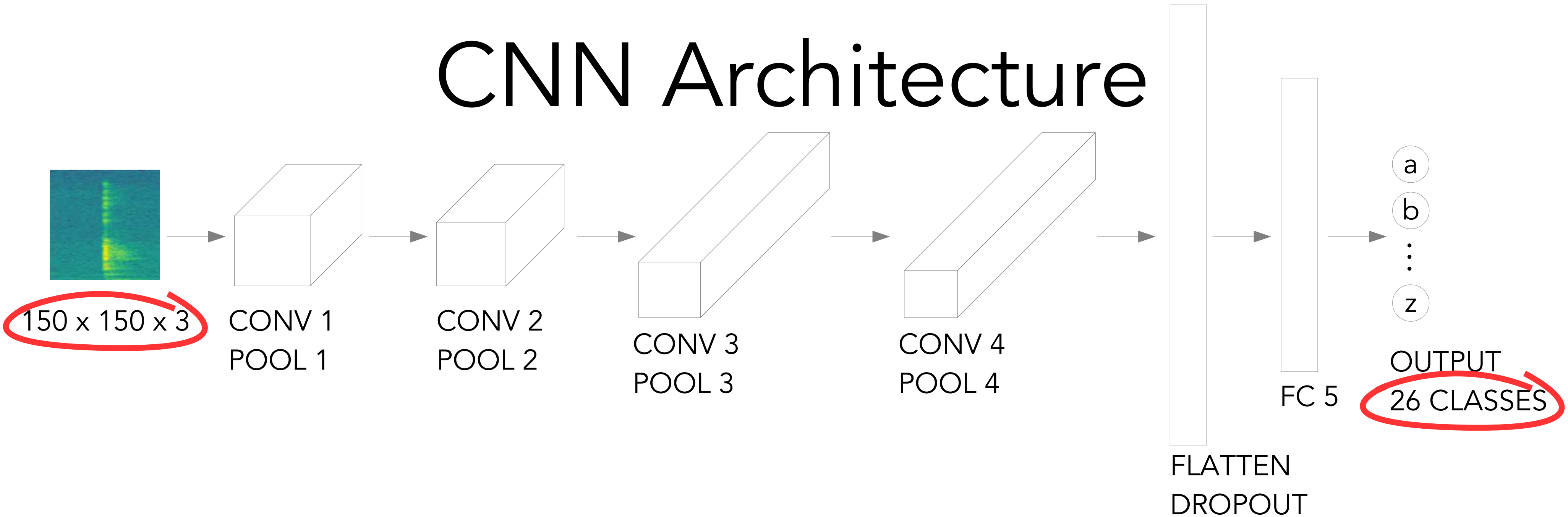


r





CNN Architecture

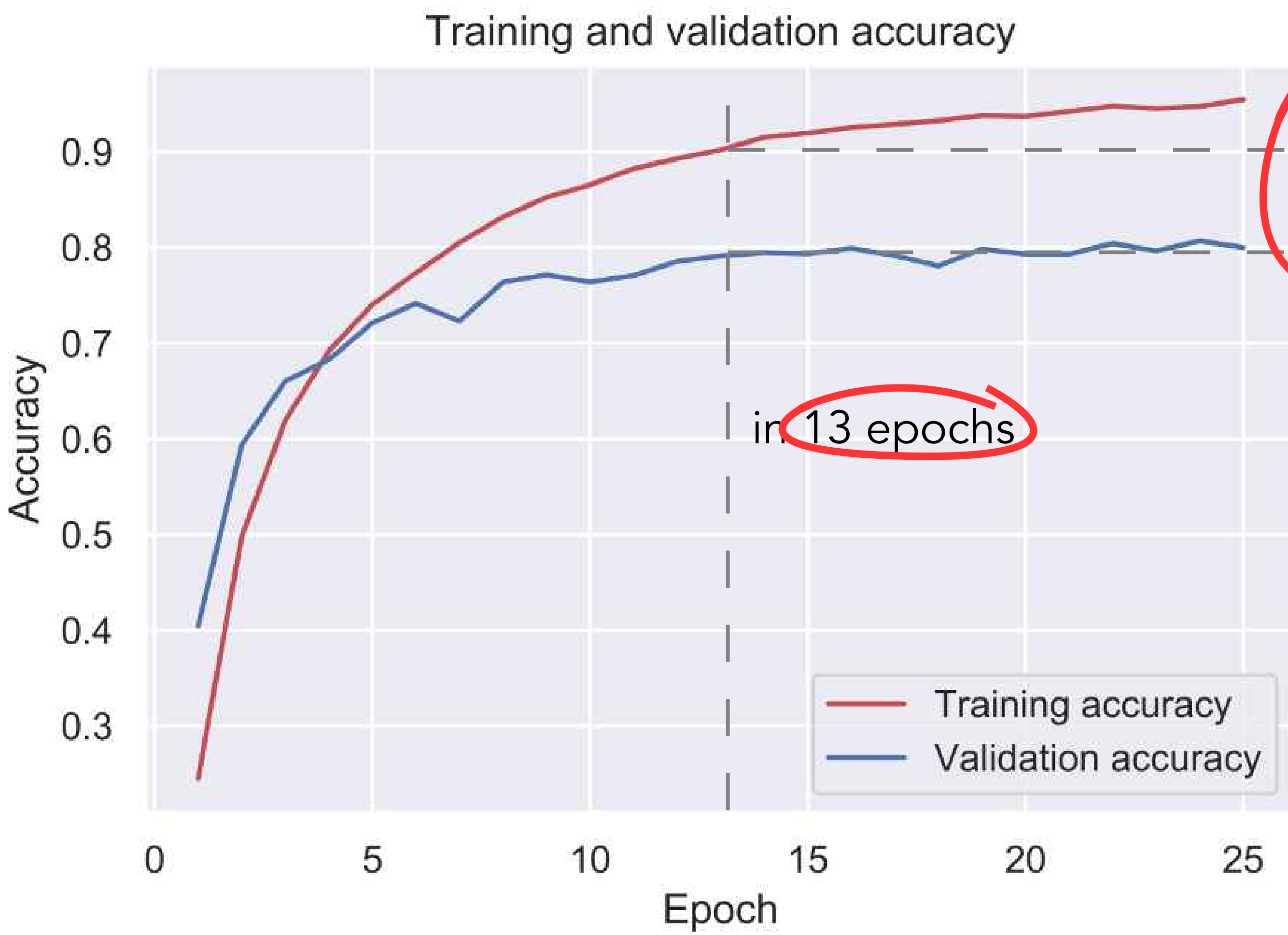



```
model = tf.keras.models.Sequential([
    # 1st convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    # 2nd convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # 3rd convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # 4th convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),

    # FC layer
    tf.keras.layers.Dense(512, activation='relu'),

    # Output layer
    tf.keras.layers.Dense(26, activation='softmax')
])
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 148, 148, 64)	1792
max_pooling2d_4 (MaxPooling2	(None, 74, 74, 64)	0
conv2d_5 (Conv2D)	(None, 72, 72, 64)	36928
max_pooling2d_5 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_6 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_6 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_7 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_7 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dropout_1 (Dropout)	(None, 6272)	0
dense_2 (Dense)	(None, 512)	3211776
dense_3 (Dense)	(None, 26)	13338

Total params: 3,485,274
Trainable params: 3,485,274
Non-trainable params: 0

CNN Training

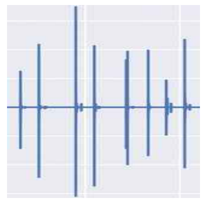


90% training accuracy

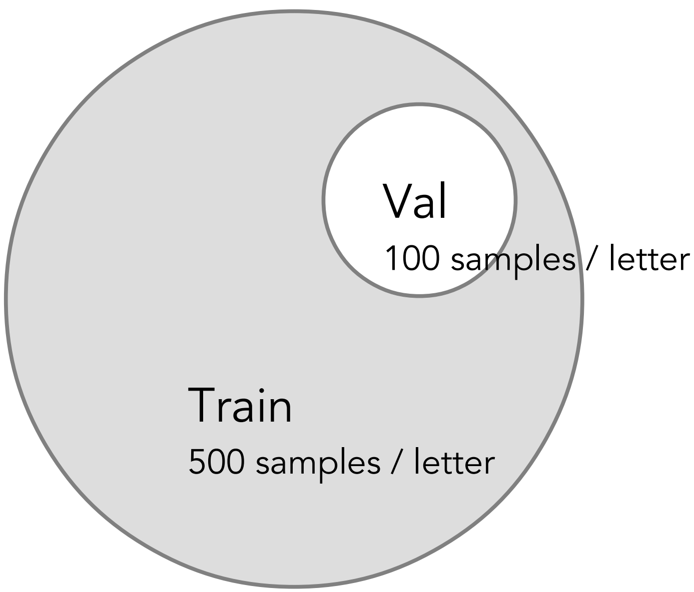
80% validation accuracy

Note:
This is chr-level accuracy,
NOT word level accuracy!

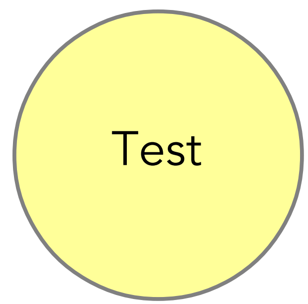
Testing



? ? ? ? ? ? ? ?



Total # of samples: ~ **16,000**



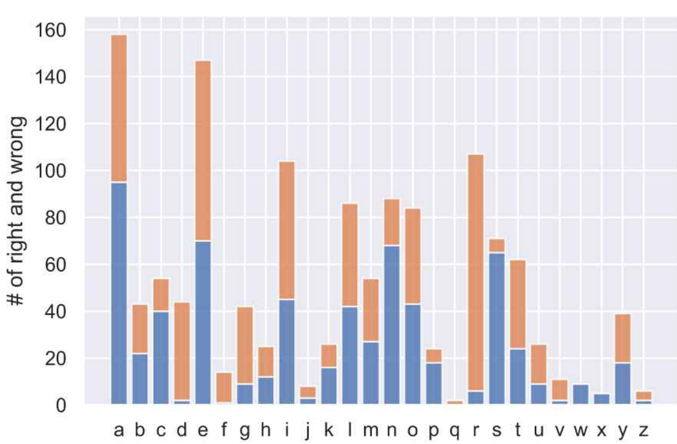
200 ***different*** [pass]words
from 'rockyou'

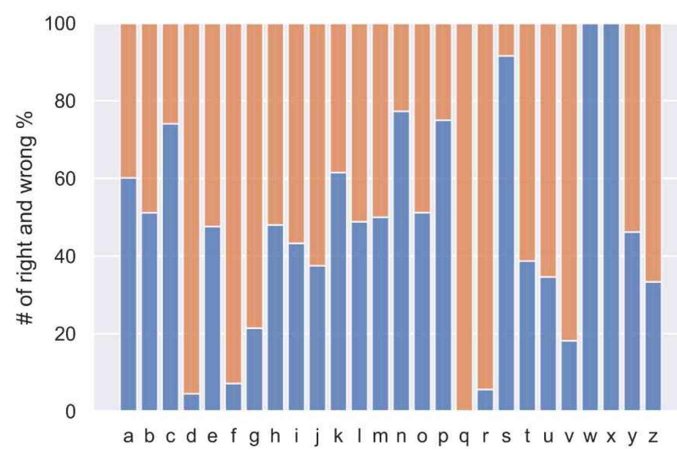
Testing (contd.)

Testing (contd.)

Test Examples

Actual	Predicted	Correct Ones
a a r o n	s s i o b	_ _ _ o _
c a n a d a	c a n z w a	c a n _ _ a
l o k i t a	l o k i t a	l o k i t a





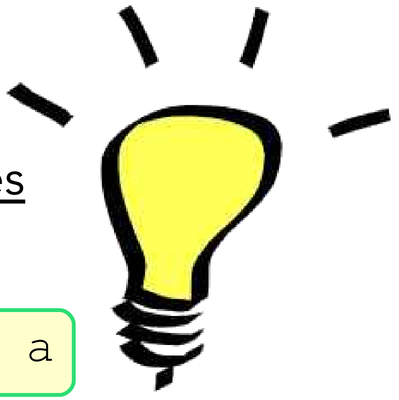
Accuracy %

Chr	Word
48.8	1.5

Error Analysis

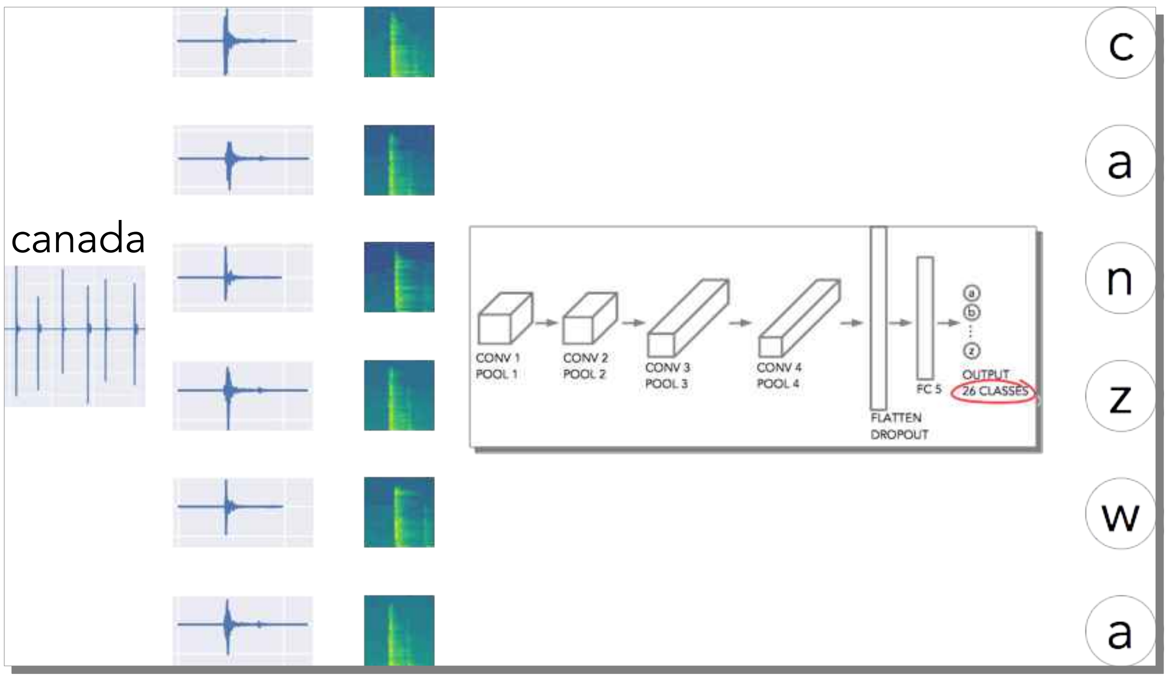
Test Examples

Actual	Predicted	Correct Ones
a a r o n	s s i o b	_ _ _ o _
c a n a d a	c a n z w a	c a n _ _ a
l o k i t a	l o k i t a	l o k i t a



What if we pass it through a spellchecker?!

Improving Model Accuracy

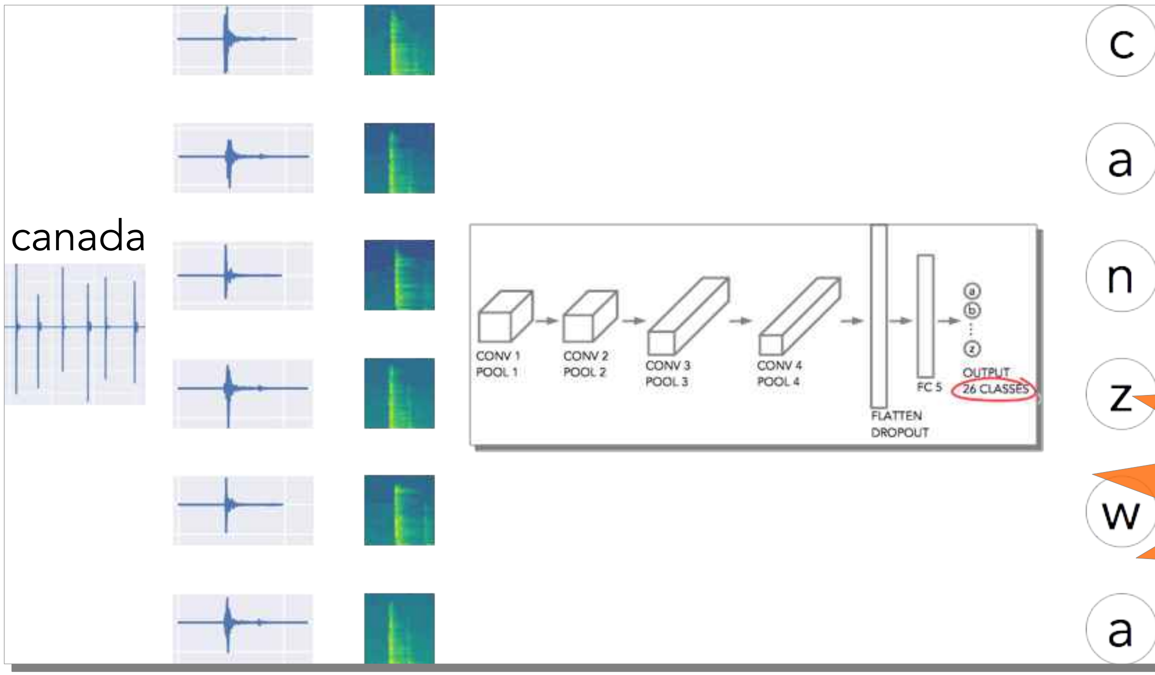



```
autocorrected
= spell(predicted)
```

c a n a d a

	Chr	Word
CNN Predicted accuracy:	48.8	1.5
+ Autocorrected accuracy:	49.5	8.0

Improving Model Accuracy




```
autocorrected
= spell(predicted)
```

How about a RNN instead??

c
a
n
a
d
a

	Chr	Word
CNN Predicted accuracy:	48.8	1.5
+ Autocorrected accuracy:	49.5	8.0

More Error Analysis

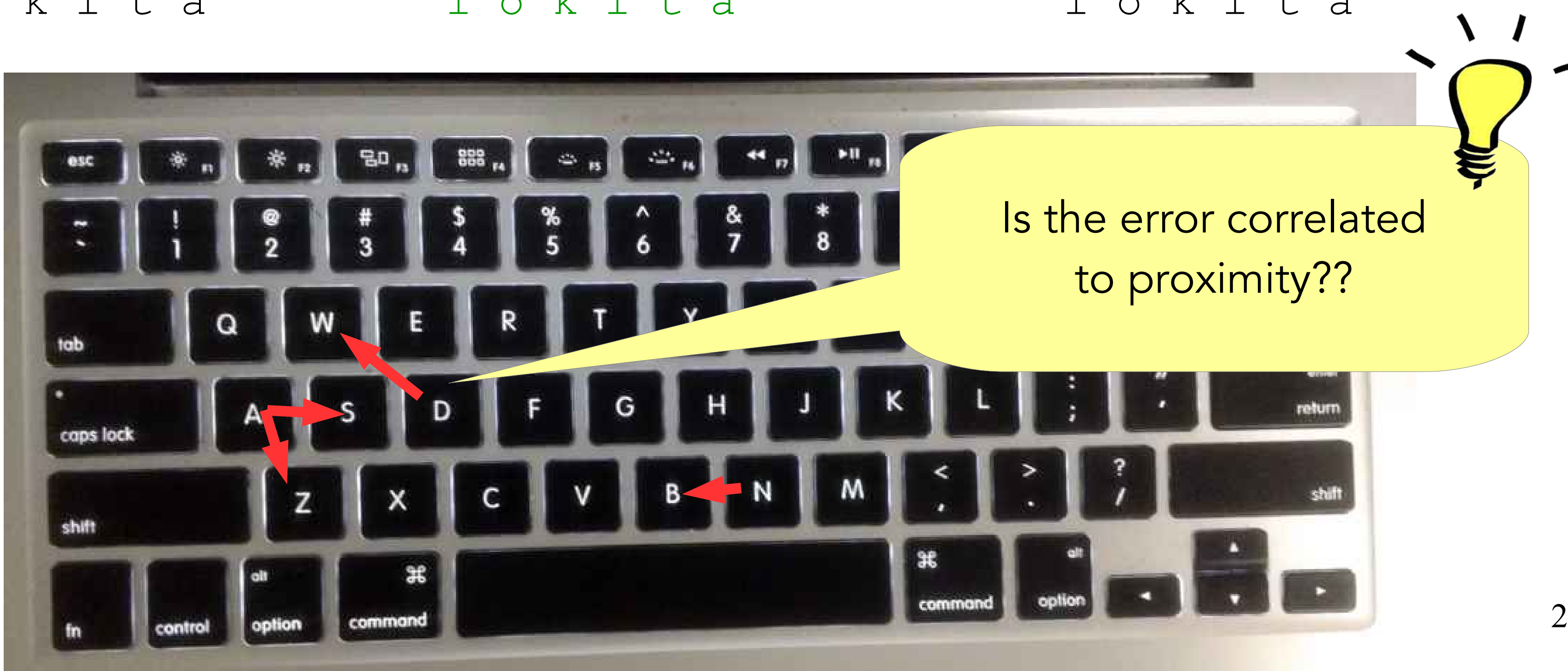
Test Examples

Actual	Predicted	Correct Ones
a a r o n	s s i o b	_ _ _ o _
c a n a d a	c a n z w a	c a n _ _ a
l o k i t a	l o k i t a	l o k i t a

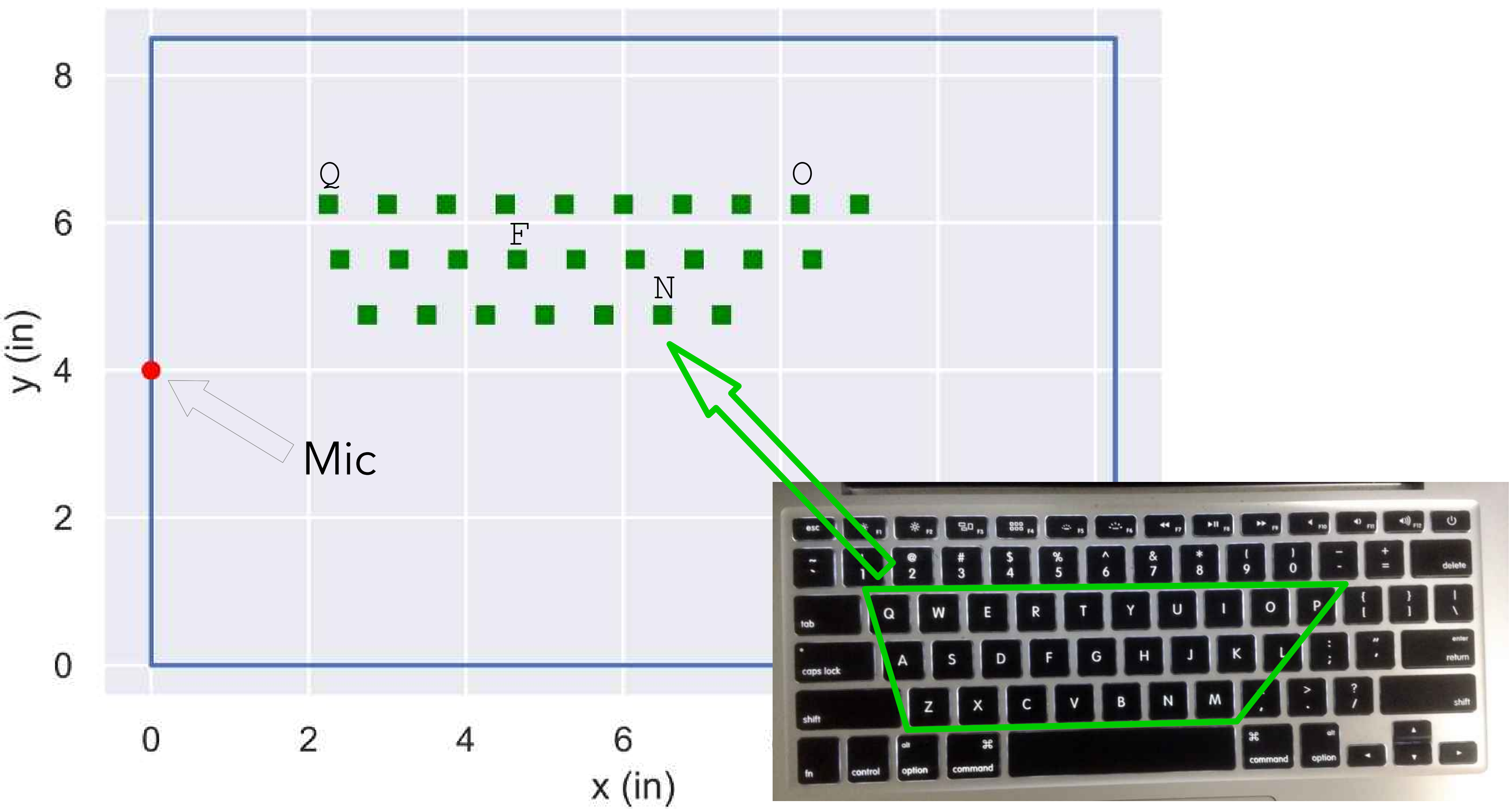
More Error Analysis

Test Examples

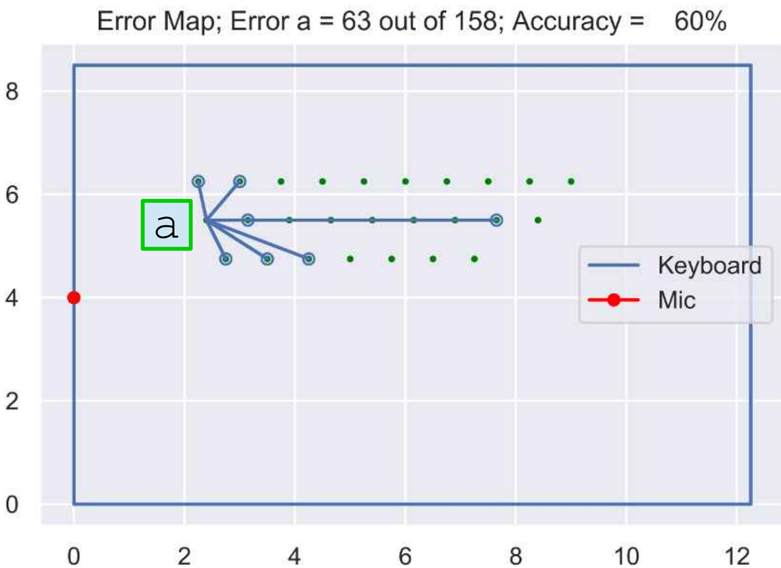
Actual	Predicted	Correct Ones
a a r o n	s s i o b	_ _ _ o _
c a n a d a	c a n z w a	c a n _ _ a
l o k i t a	l o k i t a	l o k i t a

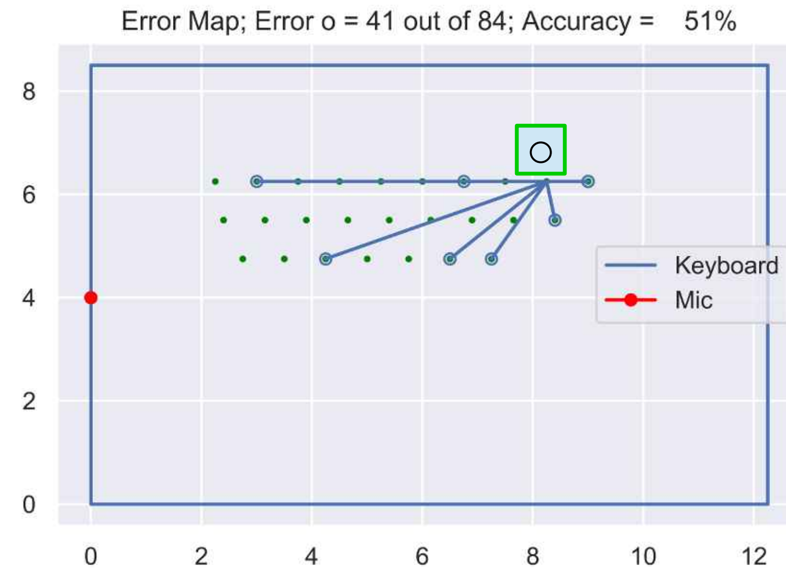


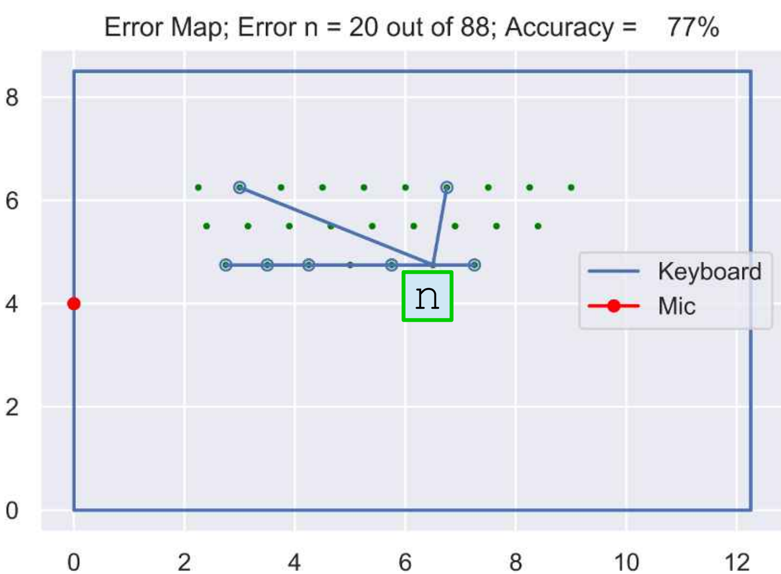
More Error Analysis

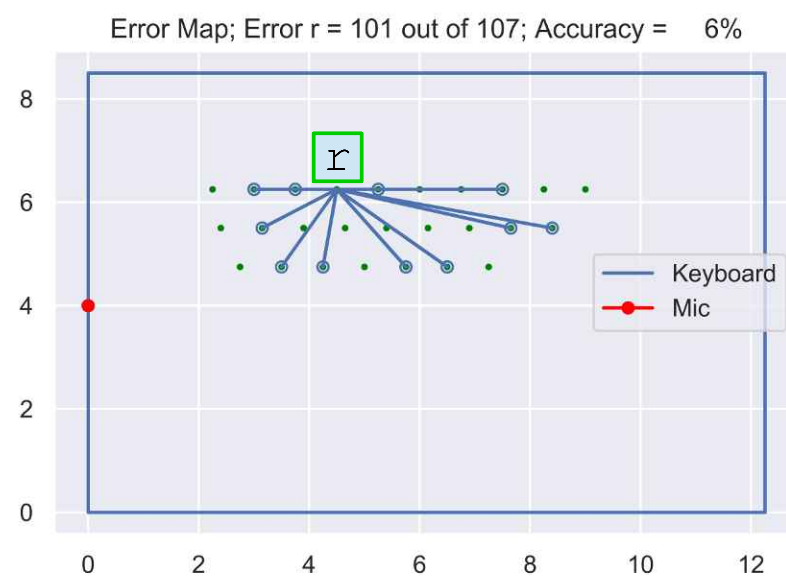


Error Map (Samples)

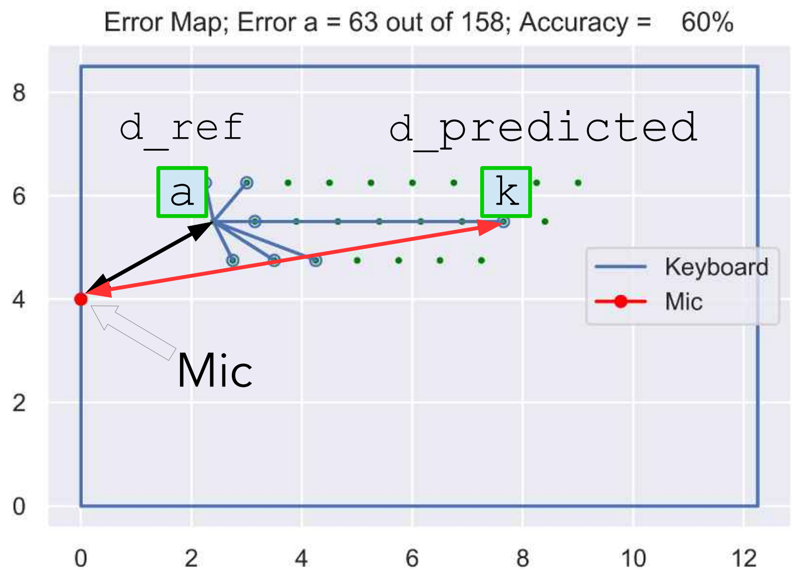








Error Groups



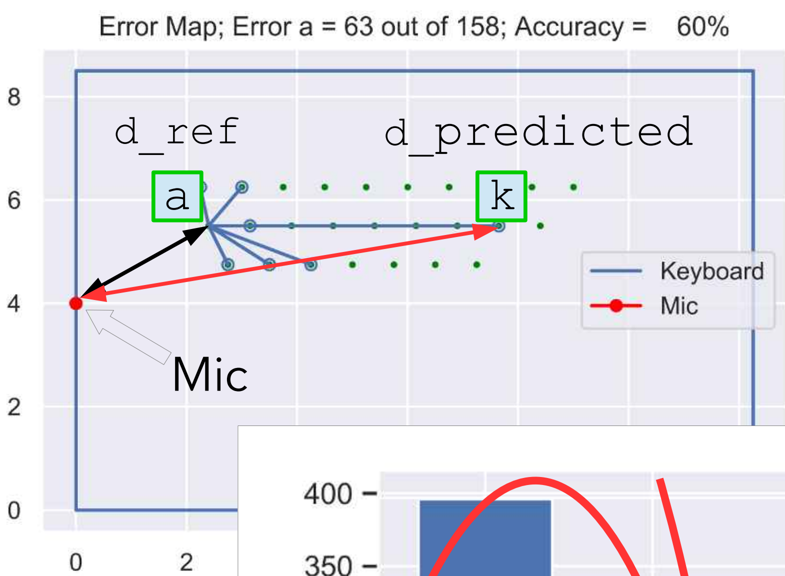
d_ref = dist. of the reference letter from the mic

d_predicted = dist. of the predicted letter from the mic

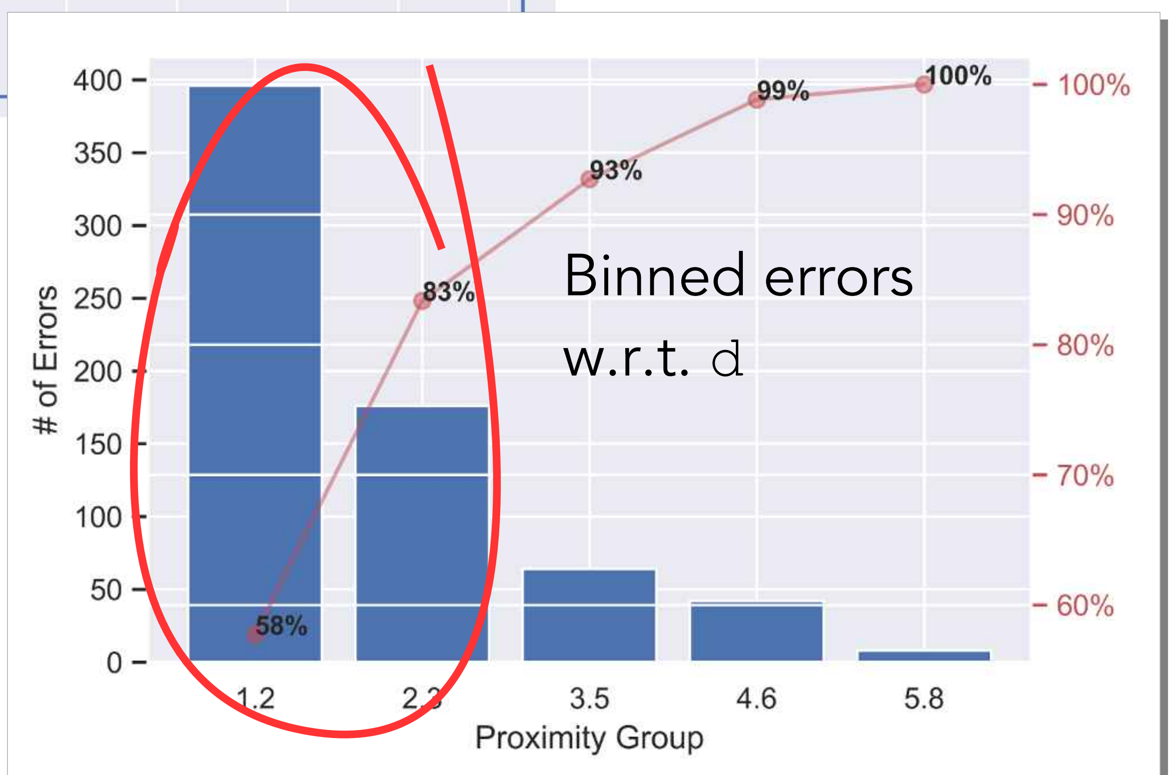
d = abs(**d_predicted** - **d_ref**)

Bin the errors w.r.t. **d**

Error Groups



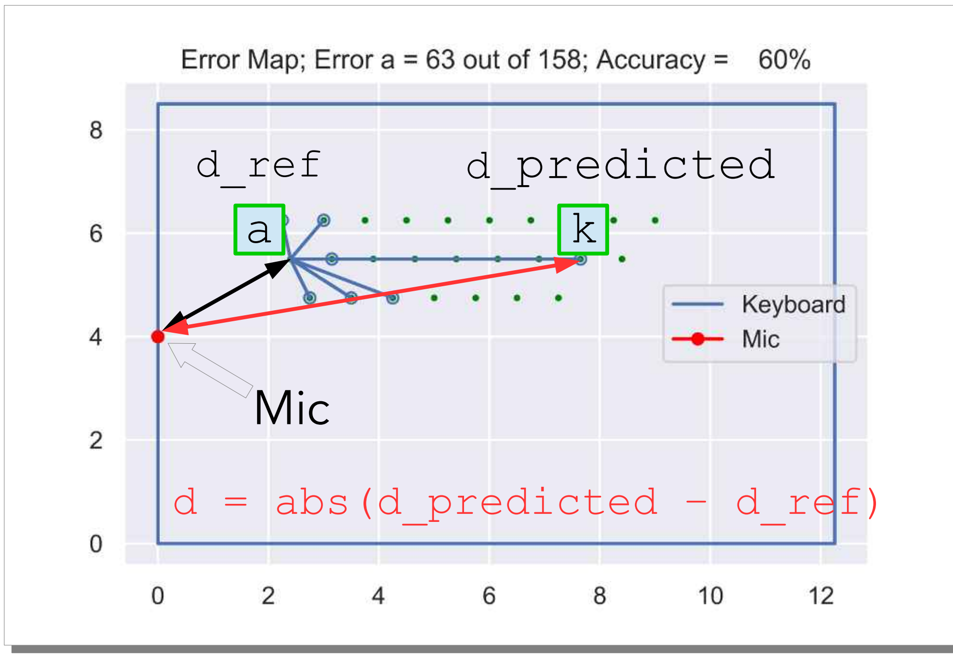
d_ref = dist. of the reference letter from the mic
d_predicted = dist. of the predicted letter from the mic
d = abs(**d_predicted** – **d_ref**)

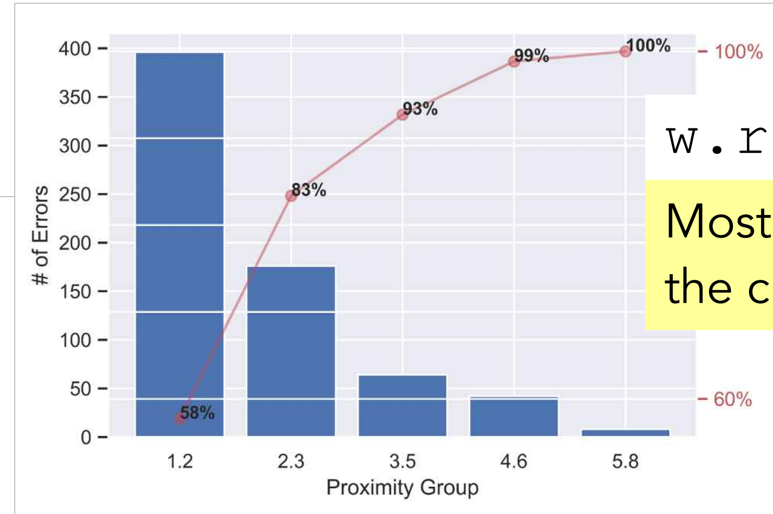


=>
Most errors are from the close proximity

=>
- More data
- Bigger network
- Network architecture that can capture this better

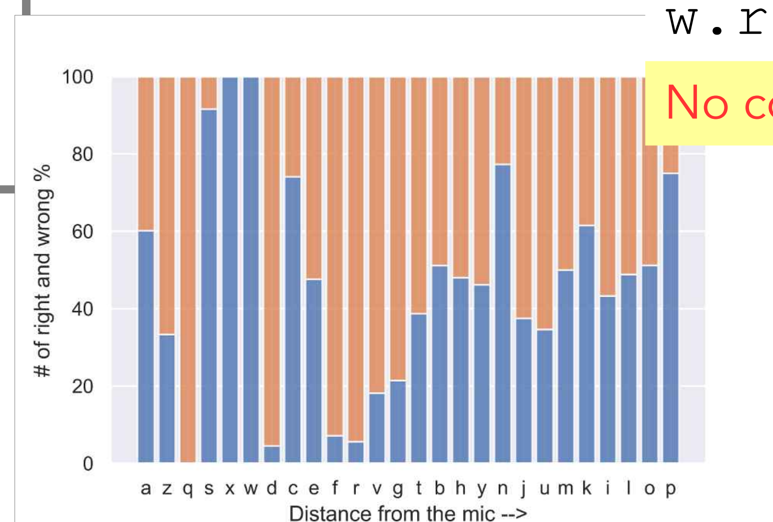
One more thing ...





`w.r.t. d`

Most errors are from the close proximity



`w.r.t. d_ref`

No correlation

Summary

- It seems possible to hack the keystroke sounds
- With a fairly small amount of data and a simple CNN architecture + spell check, we can get a non-trivial word-level accuracy

	Chr	Word
CNN Predicted accuracy:	48.8	1.5
+ Autocorrected accuracy:	49.5	8.0

Model Enhancements ? Thoughts

- Normal typing speed ? challenging signal processing (to isolate individual keystrokes)
  - Here I had typed slow one letter at a time
- Any keystrokes ? challenging signal processing (Caps Lock on?, Shift?, …)
  - Here I had used only lower-case letters (no upper case letters, digits, special characters, special keystrokes, etc. were included)
- Background noise ? add noise
  - Here during the data recording some simple and light background noise of a car passing by were present in some cases, but no complex background noise (cafeteria background noise for example)
- Different keyboards and microphone settings + different persons typing ? more data + data augmentation + bigger network + different network architecture
- Can we use vibration signature instead of audio signature?

Thank You!

Code, data, resources:

https://github.com/tikeswar/kido

Contact:

tikka / tikeswar@gmail.com