Jerk, Snap, Crackle & Pop.

Big Picture:

Map the problem onto an *image recognition/classification problem* that we know how to solve using a convolutional neural network or some other deep learning technique.

Key Insight: The following analogy gives a mapping

telematics problem $\mapsto$ handwriting association/classification' problem.

A trip can be represented as a *trajectory* on an image contained in a grid of $P \times P$ pixels.	A handwriting sample is represented by an image contained in a grid of $P \times P$ pixels.
Each trajectory/image represents a collection of time-stamped coordinates $\{(x_t, y_t)\}$ of the positions of the car collected during a trip.	Each image represents a collection of time-stamped coordinates $\{(x_t, y_t)\}$ of the positions of the pen collected while the handwriting sample was made.
Consider a dataset that consists of trips taken by N individuals, indexed by i.	Consider a data set that consists of handwriting samples taken from N individuals, indexed by i.
Each individual, i, makes K trips, with each trip indexed by (i, k). So $k = 1, \cdots, K$, and $i = 1, \cdots, N$, giving a total of KN trips.	Each individual, i, provides K handwriting samples, with each sample indexed by (i, k). So $k = 1, \cdots, K$, and $i = 1, \cdots, N$, giving a total of KN samples.
A demon goes through the each individual's trips and randomly removes a fraction α_i and replaces them with decoys.	A demon goes through the each individual's samples and randomly removes a fraction α_i and replaces them with forgeries.
Your task is to build a system which, given an image, reveals the identity of the trip's driver.	Your task is to build a system which, given an image, reveals the identity of the image's author.

We know that a version of the handwriting problem consisting of the first three elements of the list on the right can be solved by a deep neural network. So, we can be confident that placing a similar constraint on the *telematics* problem should allow a deep neural network solution.

So the real twist here is how to deal with the problem posed by the nasty demon.

Proposed strategy:

Assume that the dataset is close to being ideal in the sense that the demon's actions are a small enough perturbation to a probabilistic solution to the ideal problem.

In other words, find a probabilistic solution to the $\alpha = 0$ problem, and then interpret any samples with low probability outcomes as outliers corresponding to the demon's perturbation.

Notation

- Let $N =$ total number of drivers, and $K =$ total number of trips per driver.
- Let T denote the duration of the longest trip in the data-set.
- Let $\mathbb{i} = 1, \cdots, N$, $\mathbb{k} = 1, \cdots, K$, and $t = 1, \cdots, T$.
- We will implement a mapping

$$\begin{array}{c}\text{data corresponding to} \\ \text{the } \mathbb{k}^{\text{th}} \text{ trip} \\ \text{made by driver } \mathbb{i}\end{array} \quad \leftrightarrow \quad \text{a } \textit{multi-channel} \text{ image } D^{(\mathbb{i},\mathbb{k})}$$

- Each multi-channel image has C channels representing C categories of features, and each feature, c, is represented by T data-points.
  - Thus each multi-channel image $D^{(\mathbb{i},\mathbb{k})}$ has the structure $\left[D_{c,t}^{(\mathbb{i},\mathbb{k})}\right]_{c=1,\,t=1}^{C \quad T}$
  - We will represent each $D^{(\mathbb{i},\mathbb{k})}$ as a `numpy ndarray` of shape $(1, C, T)$.
- Finally, we will pack all trips by a given driver, say driver $\mathbb{i}$ into a data structure $D^{(i)} = \left[D^{(\mathbb{i},\mathbb{k})}\right]_{k=1}^{K}$ represented by a `numpy ndarray` of shape (K, C, T).

Feature Engineering

- Each "example" in the training set consists of a collection of "time-stamped" spatial coordinates $(x(t), y(t))$, with $0 \le t \le T - 1$. We will always transform our data so that $\sqrt{T}$ is an integer.
- Begin by first mapping "time into space" by defining a transformation that maps each time $t = 0, \cdots, T - 1$ to spatial coordinates (p_t, q_t) given by

$$p_t \equiv \frac{t - (t \mod \sqrt{T})}{\sqrt{T}}, \qquad q_t \equiv t \mod \sqrt{T}$$

- We will implement a neural network with an input layer of CT neurons, where
  - C is the number of feature categories extracted from the data, and
  - each feature category will be labeled using a "channel index" $c = 0, \cdots, C - 1$.
- An input neuron representing feature category c at time t is represented by a unit in channel c located at $(p(t), q(t))$ and receives input $I_c(t)$ where

$$I_c(t) = \begin{cases} \dfrac{d^c}{dt^c} x(t) & \text{if } c \bmod 2 \equiv 0 \\ \dfrac{d^c}{dt^c} y(t) & \text{if } c \bmod 2 \equiv 1 \end{cases}$$

- The 0th & 1st features are positions, the 2nd & 3rd features are velocities, the 4th & 5th features are accelerations, the 6th & 7th features are jerks, the 8th & 9th features are snaps, the 10th & 11th features are crackles, and the 12th & 13th features are pops. A sample of the data format is given below.

t	x	y	v_x	v_x	a_x	a_y	j_x	j_y	s_x	s_y	c_x	c_y	p_x	p_y
0	-0.05	-3.56	0.24	0.24	0.27	0.63	0.29	0.17	0.32	0.04	0.30	0.31	0.27	0.53
1	-0.04	-3.57	0.25	0.31	0.28	0.33	0.32	-0.02	0.33	0.28	0.30	0.59	0.29	0.34
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

1. So, each driving record $D^{(i,k)}$ is associated with $C = 14$ channels, with each channel represented by a vector with T non-zero components corresponding to non-zero input neuron activations in the input image map associated with that channel.

2. During training, the set of GPS coordinates $\{\boldsymbol{x}^{(i,k)}(t)\}_{t=1}^{T}$ gives rise to activity in 14 channels that are to be classified as belonging to driver i. In other words, each class label i has 200 sample images indexed by k, with each image having 14 channels. Of these images, a fraction α_i are mislabeled, and the task is to identify them.

3. At the outset, we arbitrarily split the data into a training set and a test set in a 4:1 ratio.

4. The pair of objects $(D^{(i,k)}, i)$ will serve as the input,target pairs to our network during training.

5. These are fed into the input layer of a neural network with the structure

$$\begin{tikzcd}
\begin{array}{c}\text{input}\\ \text{layer}\\ \text{(CT units)}\end{array} \arrow[r] & \begin{array}{c}\text{four cascaded}\\ \text{convolution+pooling}\\ \text{layers}\end{array} \arrow[r] & \begin{array}{c}\text{two consecutive}\\ \text{dense}\\ \text{layers}\end{array} \arrow[r] & \begin{array}{c}\text{softmax}\\ \text{output}\\ \text{layer}\\ \text{(N units)}\end{array}
\end{tikzcd}$$

6. The network utilizes dropout during training.

7. In the testing phase, we interpret the response of the most active network output as the response to the input, i.e. if we use $\{O_j\}$ to denote output activities, then on feeding an image $D^{(i,k)}$ as input, we interpret $\underset{1\leq j\leq N}{\text{argmax}}\{O_j\}$ as the network's prediction for the class label.

.... To Be Completed ....