S1-Model development

This section contains additional information on the model development section

Processing raw data

The raw curve data (raw_data) in NHANES 2011-12 was available as as a sequence of numeric values representing change in volume in millilitres over a 0.01-second interval during forced expiration. We obtain the time, volume and flow using python library numpy (np) as follows[1]:

```
time = np.arange(0, len(raw_data))*0.01
volume = (np.cumsum(raw_data))/1000
flow= np.diff(vol)/ np.diff(time)
```

where np.arrange (returns evenly spaced values in an interval), np.cumsum (cumulative sum) and np.diff (first order difference) are numpy-based functions. Here, time, volume and flow are 1D arrays

Once the data was processed, we removed any trailing forced inspiratory signal at the end of forced expiration.

Generating pixel matrixes

We generated the pixel matrices of the maximal expiratory flow-volume loop (MEFC) with an aspect ratio of two units of flow for each unit of volume in accordance with the display guidelines of ATSERS for flow-volume curves. We do this as follows:

We first determine the maximum range (yf) to be represented in the pixel matrix

```
yf  = max(flow) - min(flow)
if yf  < 2 x (max(volume) - min(volume)) , then yf = 2 x (max(volume) - min(volume))
```

Then, we initialise an identity matrix of dimension pxl x pxl. In our study, pxl=32

```
pxl_matrix = np.ones((pxl,pxl))
```

Then, we map the flows and volumes to corresponding pixel location

```
I_pxl = (2*((pxl-1)/yf) * volume)
J_pxl = (((pxl-1)/yf) *  flow)
pxl_matrix [J_pxl, I_pxl] = 0
```

Convolutional neural network (CNN) architecture

We use a convolutional neural network that takes the 32x32 MEFVC matrix as input. We first describe the building blocks of our model[2].

Building blocks

a) Convolutional block

   A convolutional block (Figure S1) consists of the following three operations in sequence:

   i) Convolution (CONV): A CONV layer consists of a set of learnable filters, which are spatially small but extends through the depth of the input volume. For example, a typical filter on the first convolutional block may have a size of 3x3x1 (i.e. 3 pixels width and height and 1 because the images have a single channel). During a forward pass, we slide the filter over the width and height of the input volume and produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Similarly, a set of K filters will produce K two-dimensional activation maps, which we stack along the depth dimension and produce the output volume. Intuitively, we can interpret that the network will learn a set of K filters that activate when they see some type of visual feature such as an edge of some orientation on the first layer, or eventually entire honeycomb or wheel-like patterns on later blocks/layers of the network.

Mathematically, a discrete 2D convolution operation can be written as follows:

$$y[m,n,d] = \sum_{u=-\infty}^{\infty} \sum_{v=\infty}^{\infty} f[u,v] \,.\, k_d[u-m\,,\;\; v-n] + b_d \,, d = 1{,}2..K_i$$

where $y[m,n,\ d]$ is the result of the convolutional operation at column m and row n for the d^{th} filter, $f[i,j]$ represents a depth slice at column i and row j of the input volume, k_d is called a filter or kernel which is a square weight matrix of size F_i that is translated over the input volume and b_d is a bias. To summarize, a CONV layer at the i^{th} convolutional block:

- Accepts an input volume X_{1i} of size $W_{1i}\ x\ H_{1i}\ x\ D_{1i}$
- Performs a 2D convolution operation with the hyper-parameters some of which are fixed by us:
    - Number of filters (K_i)
    - The filter width and height ($F_i = 3$)
    - The filter stride ($S_i = 1$)
    - The amount of zero padding ($P_i = 0$)

  Results in a total of $(F_i\ .\ F_i\ .\ D_{1i}\ ).\ K_i$ weights and K_i biases
- Produces an output volume X_{2i} of size $W_{2i}\ x\ H_{2i}\ x\ D_{2i}$ where
    - $W_{2i} = (W_{1i} - F_i + 2P_i)/S_i + 1$
    - $H_{2i} = (H_{1i} - F_i + 2P_i)/S_i + 1$
    - $D_{2i} = K_i$

ii) Rectified linear unit (ReLU): The output volume from the previous step is passed through the ReLU activation function that introduces non-linearity as follows:

$$y = \max(0, x)$$

In other words, the activation is simply thresholded at zero. We can summarize the computations in this step as follows:

- Accepts X_{2i} as input
- Applies the ReLU activation function
- Produces an output volume X_{3i} of size $W_{2i}\ x\ H_{2i}\ x\ D_{2i}$

iii) Max pooling (MAXPOOL): The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The MAXPOOL Layer operates by considering a $MP_i\ x\ MP_i$ region with a stride SP_i over a given 2D slice and outputs a single value, which is the maximum in that region. It operates independently on every depth slice of the input and it has no learnable parameters. We can summarize the computations in this layer as follows:

- Accepts X_{3i} as input
- Performs MAXPOOL that requires the hyper-parameters:
    - The spatial extent ($MP_i = 2$)
    - The stride ($SP_i = 2$)
- Produces an output volume X_{4i} of size $W_{3i}\ x\ H_{3i}\ x\ D_{3i}$ where
    - $W_{3i} = (W_{3i} - MP_i)/SP_i + 1$
    - $H_{3i} = (H_{3i} - MP_i)/SP_i + 1$
    - $D_{3i} = D_{2i}$

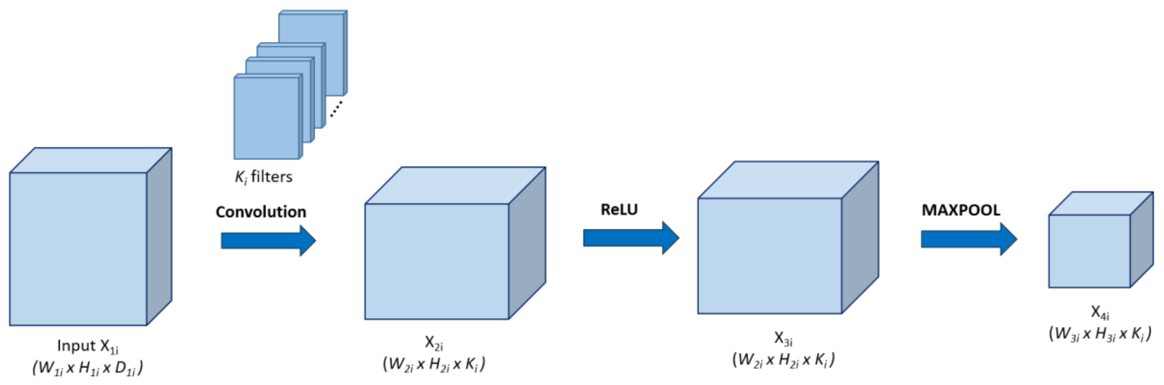
K_i filters
Convolution
ReLU
MAXPOOL
Input X_{1i}
$(W_{1i} \times H_{1i} \times D_{1i})$
X_{2i}
$(W_{2i} \times H_{2i} \times K_i)$
X_{3i}
$(W_{2i} \times H_{2i} \times K_i)$
X_{4i}
$(W_{3i} \times H_{3i} \times K_i)$

Figure S1 Flow of computations inside a convolutional block

b) Fully connected (FC) network (neural network)

A FC network (Figure 6) consists of a standard feed-forward neural network with one hidden layer and a single neuron output layer. The activation of the hidden layer 1 can be expressed as follows:

$$A^1 = ReLU(W^1.A^0 + B^1)$$

where W^1 is a n^1 x n^0 weight matrix where n^1 denotes the number of neurons in layer 1, A^0 are the activations from the previous layer of length n^0 x 1 which in our case would be a flattened output volume, B^l is the bias and ReLU is an activation function defined as:

$$ReLU(x) = \max(0, x)$$

The computation in the final layer is written as:

$$\hat{Y} = sigmoid(W^f.A^1 + B^f)$$

where $\hat{Y}$ is a scalar value between 0 and 1, W^f is a weight matrix of size 1x n^1 and B^f is the single bias term in the final layer. The sigmoid function is defined as:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

To summarize, the FC network:

- Accepts a vector of length n^0 x 1
- Performs a feed-forward computation involving the hyperparameters:
  - The number of hidden layer neurons n^1

  Results in a total of (n^0 . n^1 + n^1) weights and $B^1 + 1$ biases
- Produces a scalar value $\hat{Y}$

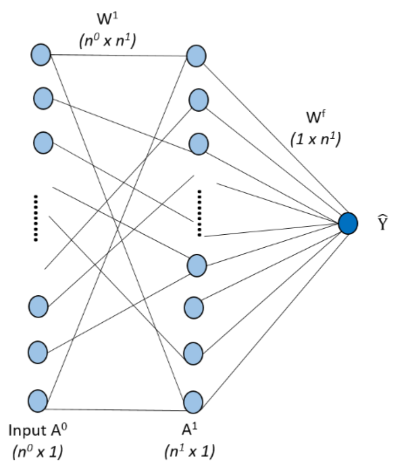

Figure S2 Activations and weight matrices W in a FC network

Model training

The cross-entropy error (Loss) was used to quantify the loss and the cost function J was described as the average cross-entropy error over the training set. Mathematically,

$$Loss(Y_i, \widehat{Y_i}) = -\{Y \log \widehat{Y} + (1 - Y) \ \log \ (1 - \widehat{Y})\}$$

$$J(W) = \frac{\sum_{i=1}^{m_{train}} Loss(Y_i, \widehat{Y_i})}{\mathrm{N_{train}}} + \lambda \sum W.W$$

In the above equation, Y_i refers to the label of the ith training example which is either 1 (acceptable) or 0 (rejected), $\widehat{Y_i}$ refers to the probability calculated by the model corresponding ith training example, W represents the weighs of the entire network and λ is the L2 regularization parameter used to prevent overfitting. W is updated using gradient descent:

$$W \coloneqq W - \alpha \frac{\partial J}{\partial W}$$

where α is the learn rate. In our case, we use batch gradient descent with a batch size of n_{batch} with Adam optimization scheme[3] to train the model. The number of epochs used are n_{epochs} resulting in a total of (m_{train} /n_{batch}) x n_{epochs} iterations.

Hyperparameter tuning

Parameters that are not calculated by gradient descent are called hyperparameters. The following hyper-parameters were considered for optimization in our study:

- The learn rate (α)
- Batch size (n_{batch})
- Number of epochs (n_{epochs})
- L2 regularization (λ)
- Dropout rate between flattened volume and first hidden layer (*dr*)
- No of filters in the first convolution block of both branches (K_1)
- No of filters in the second convolution block of both branches (K_2)
- No of filters in the third convolution block of both branches (K_3)

For each set of hyper-parameters, the model is trained and prediction probabilities are calculated on the validation set. A threshold of 0.5 is used to assign a prediction label i.e. a manoeuvre with a probability of greater than 0.5 is assigned an acceptable label (1) and with a probability of lesser than 0.5 is assigned a rejected label (0).

The cost function J_{hp} for optimizing the hyper-parameters of the CNN is an average of validation sensitivity and specificity.

$$Sensitivity = \frac{TP}{TP + FN}, \qquad Specificity = \frac{TN}{TN + FP},$$

$$J_{hp} = -0.5 \ (Sensitivity + \ Specificity)$$

where TP is true positives, FN is false negatives, TN is true negatives and FP is false positives. We using a Bayesian search scheme [4] with 100 evaluations to solve the following optimization problem:

$$\min_{\alpha,\, n_{batch}, n_{epochs}, \lambda, dr, F_1, F_2, n^l} J_{hp}(\alpha,\ n_{batch}, n_{epochs}, \lambda, dr, F_1, F_2, F_3\ )$$

Model interpretability

A direct interpretation of CNN is difficult; therefore we used a game-theoretic approach called Shapley values to interpret our model's output[5]. The Shapley value of a feature value is its contribution to the output, weighted and summed over all possible feature value combinations. The formula for Shapley value is:

$$\Phi_j = \sum_{S \in \{x_1,\dots x_p\} \backslash \{x_j\}} \frac{|S|!\,(p - |S| - 1)!}{p!} \left(f(S\,U\,\{x_j\} - f(S)\right)$$

Where $x_1....x_p$ represents the set of p input features for function f, x_j is the desired feature, Φ_j is the Shapley value of feature j, S is a combination of inputs without including x_j (done by replacing value of x_j with a background value from the dataset), $S\,U\,\{x_j\}$ includes feature x_j

To calculate Shapley values in our study, we first divided the MEFVC pixel matrix into 2 regions: MEFVC<1s containing the MEFVC pixels from the point of maximum inspiration until 1 second after time 0, and MEFVC>1s containing the rest remaining curve. This division can be observed in figure s3 where a vertical line on the 20th pixel divides the matrix into two regions. Then using a technique called Shapley additive explanation (SHAP) [6], which allows a faster but approximate Shapley value calculation, we estimated how MEFVC<1s, MEFVC>1s, BEV criteria, tFE>6s, EOP and tPEF contributed to the model's output. To simulate the effect of background value, we replaced these inputs by randomly drawing instances from our training dataset. For MEFVC<1s and MEFVC>1s, we replaced all the pixels for corresponding regions from a randomly drawn pixel matrix. In this example, this would amount to replacing all pixels between vertical lines 0 and 20 for MEFVC<1s and between vertical lines 20 and 32 for MEFVC>1s.

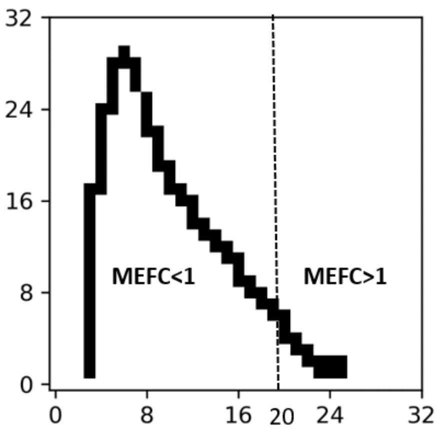

Figure S3 Maximal expiratory flow volume loop pixel matrix partitioned into two regions, MEFVC<1s and MEFVC>1s, at a location corresponding to FEV1

References

1. Klein B, Klein B. NumPy. *Einführung Python 3* 2014.

2. LeCun Y, Bengio Y, Hinton G, Y. L, Y. B, G. H. Deep learning. *Nature* 2015; 521: 436–444.

3. Kingma DP, Ba JL. Adam: a Method for Stochastic Optimization. *Int. Conf. Learn. Represent. 2015* 2015; : 1–15.

4. Bergstra J, Yamins D, Cox DD. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. *12th PYTHON Sci. CONF. (SCIPY 2013)* [Internet] 2013; : 13–20Available from: http://hyperopt.github.io/hyperopt/%5Cnhttps://github.com/jaberg/hyperopt%5Cnhttp://www.youtube.com/watch?v=Mp1xnPfE4PY.

5. Shapley LS. A value for n-person games. *The Shapley value* 2009.

6. Lundberg SM, Lee SI. A unified approach to interpreting model predictions. *Adv. Neural Inf. Process. Syst.* 2017.