Ex 1: Object detection with a linear model

In this exercise we develop a linear model for object detection in images. Specifically, we will train the model to detect the area of an image in which a hand is shown to gesture a sign language sign.

We will develop a linear model that, given an image $\mathbf{x}$ (an array of pixels), produces a bounding box $\mathbf{y}$. The bounding box is a rectangle defined by its top-left and bottom-right corners. Hence $\mathbf{y}=(y_1, y_2, y_3, y_4)$. The detected object -- a hand gesturing a sign -- is to be inside this bounding box.

Data. This is a supervised learning problem, as we have a set of images with their corresponding bounding boxes. We will use a sign language dataset that includes both images and bounding box examples. The data is available as a zip file at https://github.com/yoavram/Sign-Language/raw/master/Dataset.zip. After extracting the zip file to the folder *data/sign-lang*, we have 7 folders, one for each person (user). Each folder has 10 images of that person gesturing one of 24 signs: the ABC letters, not including J and Z. This is the file structure (the hyperlinks don't lead anywhere):

(see in the next page)

```
├── user_3
│   ├── A0.jpg
│   ├── A1.jpg
│   ├── A2.jpg
│   ├── ...
│   ├── B0.jpg
│   ├── B1.jpg
│   ├── B2.jpg
│   ├── ...
│   ├── Y8.jpg
│   ├── Y9.jpg
│   └── user_3_loc.csv
├── user_4
│   ├── A0.jpg
│   ├── A1.jpg
│   ├── A2.jpg
│   ├── ...
│   ├── Y8.jpg
│   ├── Y9.jpg
│   └── user_3_loc.csv
...
└── user_10
    ├── A0.jpg
    ├── A1.jpg
    ├── A2.jpg
    ├── ...
    ├── Y8.jpg
    ├── Y9.jpg
    └── user_10_loc.csv
```

The *jpg* files are images, for example these are *user_3/A0.jpg* and *user_9/K7.jpg*:


The image name has the sign (A and K in these examples) and the repetition number.

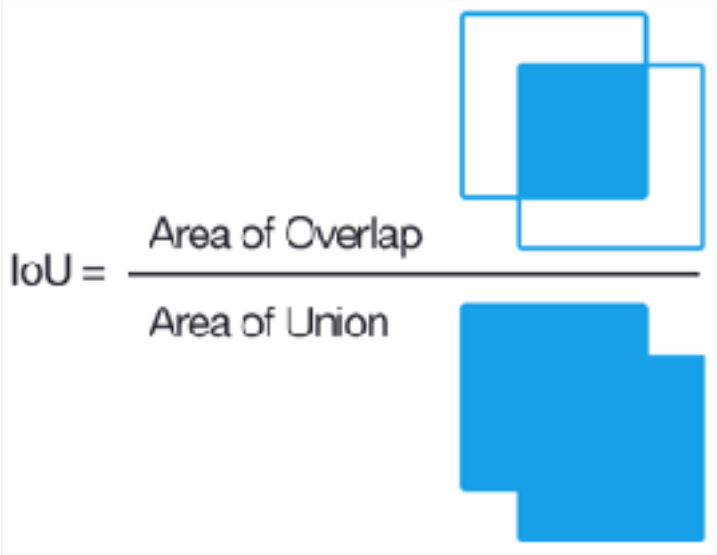
In addition, each folder contains a metadata file; for example, the first 5 rows in *user_3/user_3_loc.csv* are

	image	top_left_x	top_left_y	bottom_right_x	bottom_right_y
0	user_3/A0.jpg	124	18	214	108
1	user_3/A1.jpg	124	18	214	108
2	user_3/A2.jpg	123	19	213	109
3	user_3/A3.jpg	122	21	212	111
4	user_3/A4.jpg	122	20	212	110

This table has 5 columns. The first column **image** provides the name of the image. The other columns provide the **bounding box** *y* for that image. For example, the bounding box for *user_3/A0.jpg* is a box with corners at (124, 18) (214, 108):

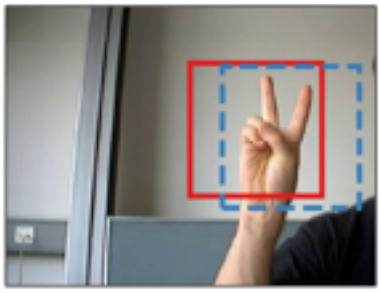
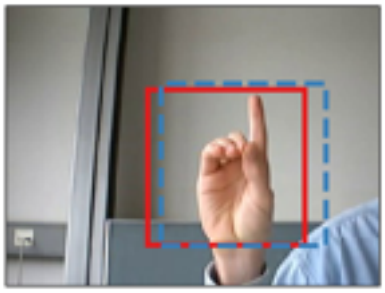
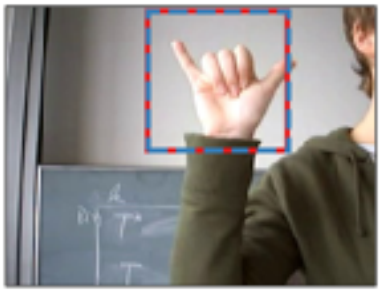
Goal. We want to collect all images into an array *X* and all bounding boxes into an array *Y*. Then, we want to train a model that, given an example image *x* produces the bounding box *y* for that image, such that the bounding box will contain a gesturing hand.
We would like to have a simple, rather than complex model: if we can produce good results with a linear model, than we prefer them over neural networks.
We also need to evaluate and visualize our model. Visualization is easy enough – we plot the image, together with the true bounding box and the predicted bounding box. To evaluate the model, however, we need to define a metric called IoU (intersection over union): the ratio of the intersection and the union of the true and predicted bounding boxes. The intuition is given by this illustration:

The definition is illustrated here:



You can find more details on IoU, including an implementation, here. However, you should implement IoU using NumPy such that it works on arrays of images, rather than just using the implementation in the link. Hint: use *np.minimum* and *np.maximum*.

Results. First, **train a generalized linear model** on a training set.
Second, **Visualize the model performance** by plotting the image, the original bounding box, and the predicted bounding box, for example:

Plot the predictions for 9 random images from the test set and print the real and predicted bounding boxes, together with the IoU.
Third, **summarize model performance** by computing the IoU over all test images, plotting the IoU histogram, together with the mean, the 5^{th} percentile, and the 95^{th} percentile. Also **print** the mean and the percentiles.

Ex 2: Classification of sign language hand gestures with CNN

This exercise is a continuation of Ex 1, but can be solved.
In Ex 1 we developed a linear model to detect the region of an image in which there is a hand that gestures a sign.
In this exercise we will develop a convolutional network model to classify the hand gestures. We will develop a model that, given an image $\mathbf{x}$ (an array of pixels), produces a classification $(y_A, y_B, \ldots, y_Y)$ such that y_i is the probability that the image is of the hand sign i and $\sum y_i = 1$.
This is a supervised learning problem, as we have a set of images with their corresponding hand signs.

Data. The data is the same as in Ex 1 (see above). However, we are going to work on cropped versions of the original images: we will use the bounding boxes, given in the metadata *csv* files, to crop the images. The idea is that the model from Ex 1 and the model we develop here will consist of a pipeline: the first model looks at an image and finds a bounding box, the second model only looks inside the bounding box and classifies the hand gesture.

For example, here's the image *user_3/A0.jpg* with a bounding box, and then its cropped version:

Original **Cropped**

The classes or labels are given in the filename (which also appears as a column in the *csv* files): for example, *A0.jpg* has the class *A*, whereas *H8.jpg* has the class *H*.

Goal. We want to collect all cropped images into an array *X* and all classes into an array *Y* (don't forget one-hot encoding the classes). Then, we want to train a **convolutional neural network** that, given an example cropped image *x* produces the predicted class probabilities *y* for that image.
Notes:

- Each cropped image has a different shape. **Resize or rescale** them (using *scipy.ndimage* or *skimage.transform* or whatever) to a common shape: this is a requirement for convolutional neural networks, which in principle don't work on variable array sizes.
- Cropping and resizing must be done in a ***preprocess*** function that will be shipped together with the model so that users can use it on their own images; the *preprocess(image, box)* function should take the bounding box as an argument.
- You should also prepare a ***decode*** function that translates the model output $\hat{y}$ to a string of the most likely hand sign.
- The convolutional neural network can be similar to that from the CNN session. If they do not perform well, increase their complexity (depth and/or width). You can also use *BatchNormalization* as a replacement for *Dropout*, or use *transfer learning*, or any other technique you want (whatever gets the job done).
- You can work on *Google Colaboratory*, which provides free access to GPU.

Results. First, **train a Keras CNN** on a training set. Then **print** its accuracy on the test set. You should be able to achieve an accuracy of at least 85%.
Second, **visualize** the model performance by randomly selecting 9 images, plotting them, and printing their real and predicted classes (as text strings like 'H' and 'B').
Third, identify which classes are more commonly mis-classified: **compute and plot** the accuracy of each separate class (hand sign), and illustrate which classes have relatively lower accuracy. These should be the classes we attempt to deal with next (if we were to continue this exercise), maybe by checking what they are confused with.