HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sign Language Recognition

Computer Vision 20201

Class: ICT-01.K61

Supervisor: Dr. Dinh Viet Sang

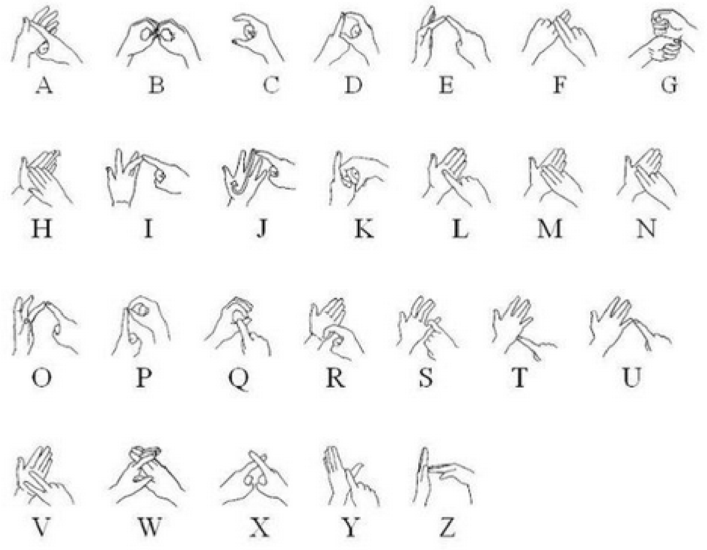
Students
Nguyen Gia Bach - 20160306
Nguyen Tien Dat - 20150853
Nguyen Hoai Nam - 20162821

January 4, 2021

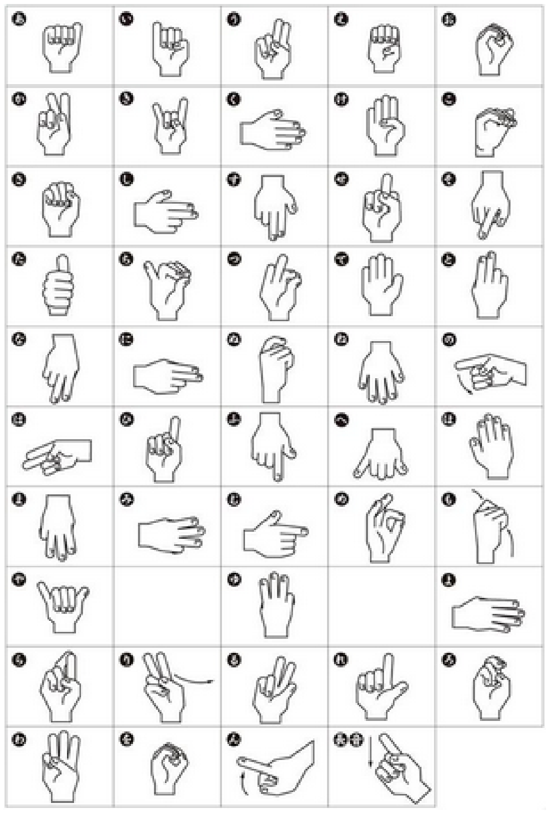
Contents

1 Introduction

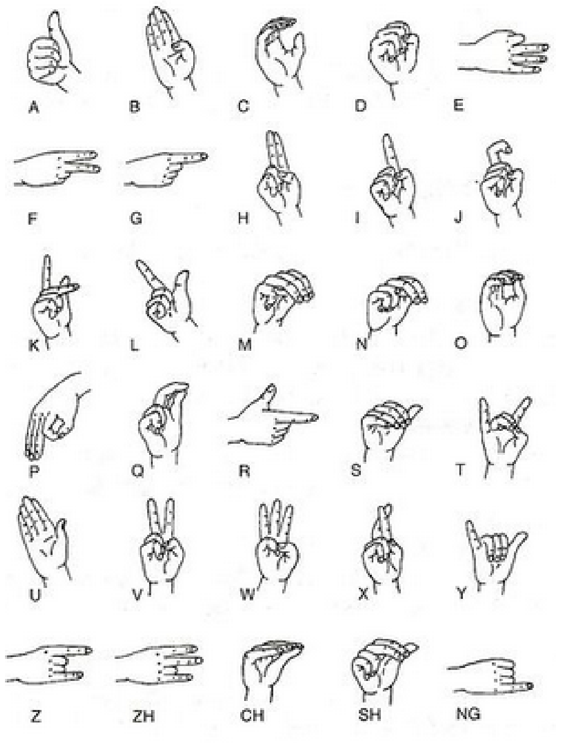
Nowadays, Sign Language is the main form of communication for the Deaf and Hard-of-Hearing community, yet it can be useful for other groups of people as well. People with disabilities including Down Syndrome, Cerebral Palsy, Apraxia of speech, and Autism may also find sign language beneficial for communicating. It is a visual means of communicating through hand signals, gestures, facial expressions, and body language. Many signed language systems represent the written form of the oral language of the area, called manual alphabets or fingerspelling.



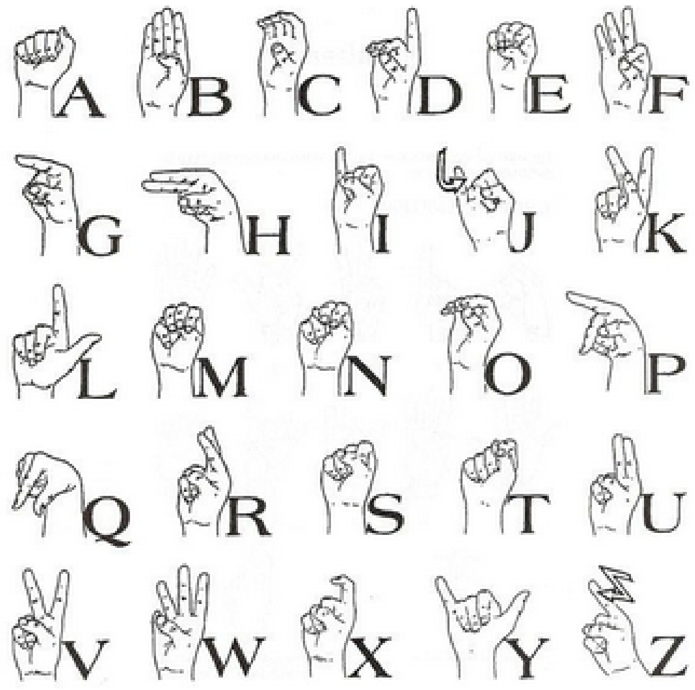
For examples, British Sign Language uses a two-handed manual alphabet



; the Japanese Sign Language (JSL) Syllabary is based on the Japanese alphabet, which is made up of phonetic syllables, also known as Nihon Shuwa in Japan



; Chinese Sign Language uses the hands to make visual representations of written Chinese characters, and has been developing since the 1950s.

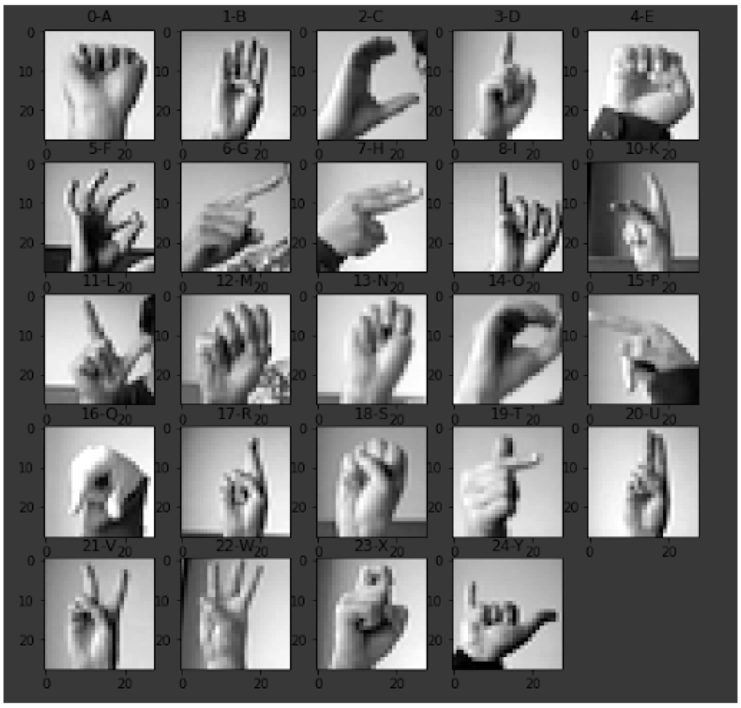


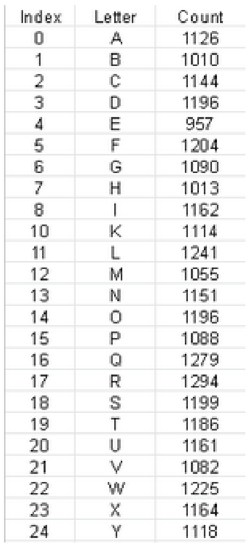
; American Sign Language (ASL) uses 26 one-handed signs to represent the letters of the alphabet used by English speakers, and signs for numbers can be mixed with the letter signs as needed. Twenty-four of the letter signs are static handshapes with specific orientations, and the other two (representing J and Z) have movement components. Except for J and Z, no movement is specified in fingerspelling. Among most commonly used sign languages, ASL has become one of the most popular language classes, ranking fourth in the latest Modern Language Association Survey and nearly shoving German from third place. The number of students taking the language has risen by more than 50 percent in the past decade. In terms of usage, there are 2 types of sign language used daily: word-spelling and finger-spelling. Word-spelling spells common words that have established signs such as “mother”, “father”, “boy”, “girl”, “tree”, etc. However, finger-spelling spells out the letters of each word in unusual occasions like people’s names, places, titles, organizations, brands, etc. For example, “tree” has its specific sign in most sign languages, but “oak” doesn’t, so o-a-k would be finger spelled to convey that specific meaning. Moreover, most people start a sign language journey by learning the A-Z or alphabet equivalent in sign form. Therefore, ASL is chosen in our project as a guiding tool for newcomers to start their sign language journey by practicing alphabet signs, similar to the way a baby learns how to speak.

2 Related Work

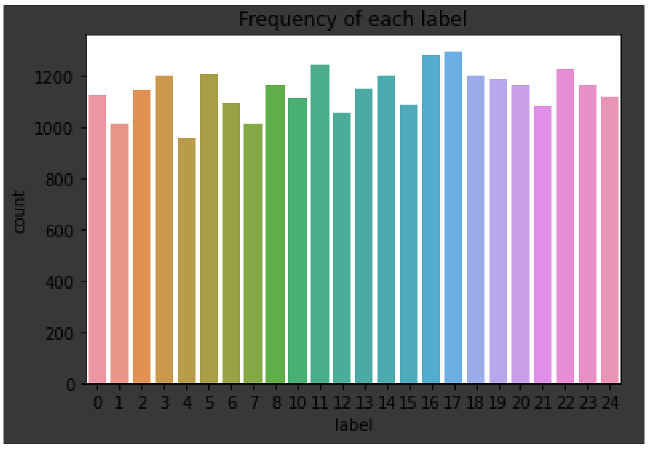
There has been several ASL datasets containing images of hand postures representing each symbol over the years. The ASL Alphabet dataset [1] is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes. Its training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING, which are helpful in real-time applications, and classification. However, the test data set contains a mere 29 images, to encourage the use of real-world test images. In addition, the images are in 3 channels (RGB), leading to 3 times as many features as 1-channel images, yet may not give a better performance in terms of detection.

Another popular dataset is the Sign Language MNIST [2], also known as the drop-in replacement for MNIST [3] for hand gesture recognition tasks. To stimulate the community to develop more drop-in replacements, the Sign Language MNIST is presented and follows the same CSV format with labels and pixel values in single rows. The American Sign Language letter database of hand gestures represents a multi-class problem with 24 classes of letters (excluding J and Z which require motion). Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2. . . .pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255. It is less expensive (computationally) to operate on a single-channel image. The grayscale image is essentially the average of RGB. Grayscale= (R + G + B) / 3





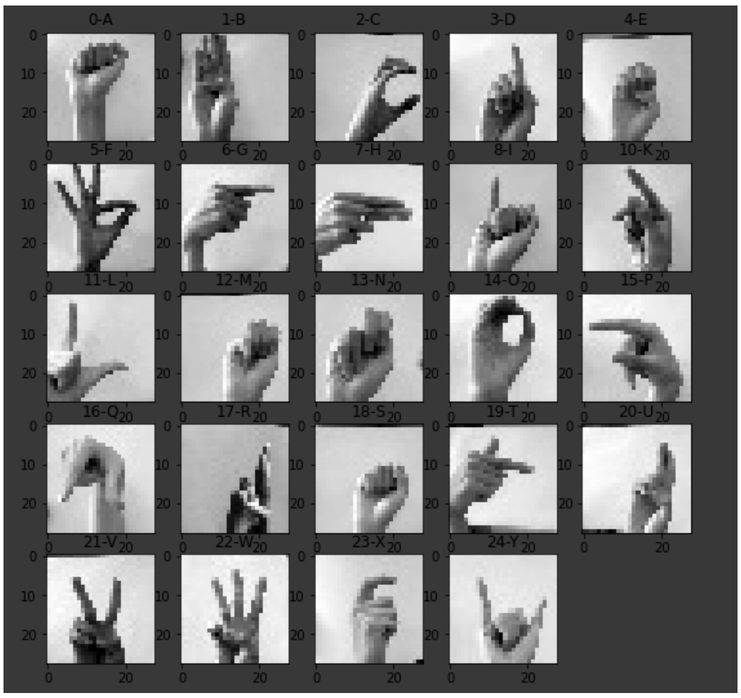
Index	Letter	Count
0	A	1126
1	B	1010
2	C	1144
3	D	1196
4	E	957
5	F	1204
6	G	1090
7	H	1013
8	I	1162
10	K	1114
11	L	1241
12	M	1055
13	N	1151
14	O	1196
15	P	1088
16	Q	1279
17	R	1294
18	S	1199
19	T	1186
20	U	1161
21	V	1082
22	W	1225
23	X	1164
24	Y	1118

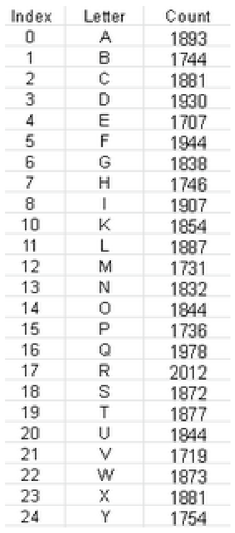


3 Proposed Solution

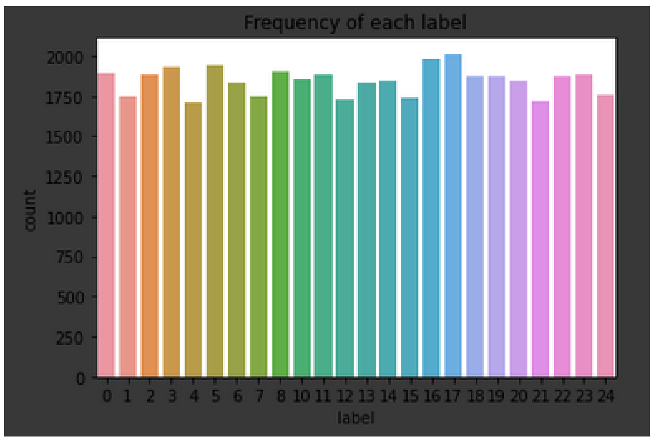
3.1 Dataset

In this project, we propose a custom dataset, as an expansion to the Sign Language MNIST. For the training dataset, an addition of about 700 images for each character in ASL (except for J and Z) were gathered manually in different locations with various lightning schemes and the background made of A4 paper: in-door capturing (4 lightning schemes) + out-door capturing (1 lighting scheme)





Index	Letter	Count
0	A	1893
1	B	1744
2	C	1881
3	D	1930
4	E	1707
5	F	1944
6	G	1838
7	H	1746
8	I	1907
10	K	1854
11	L	1887
12	M	1731
13	N	1832
14	O	1844
15	P	1736
16	Q	1978
17	R	2012
18	S	1872
19	T	1877
20	U	1844
21	V	1719
22	W	1873
23	X	1881
24	Y	1754



For the testing dataset, we collected 300 images of hand gestures for each character on the classroom environment

Sign-language-mnist-train.csv limitations:

- Images of only 1 hand from 1 person
- Imbalanced data distribution for letters

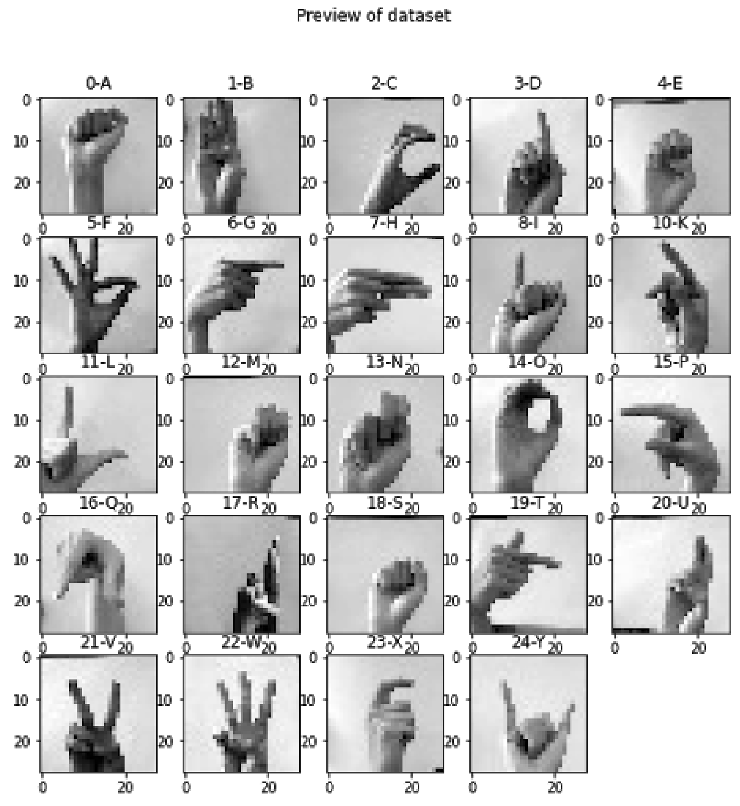
Custom dataset gains:

- Images of hands from 4 people (including original)
- More balanced distribution (letters with similar gestures having more images than others)

3.2 Preprocessing

3.2.1 Raw image to Grayscale pixels

The data first is captured using our PC's webcam, using open-cv library. The original image resolution is . . . with the RGB color mode. Then the image is cropped using a fixed windows of size (28x28) Since the data's important feature is the shape of the hand, the original image is then converted into Grayscale image.

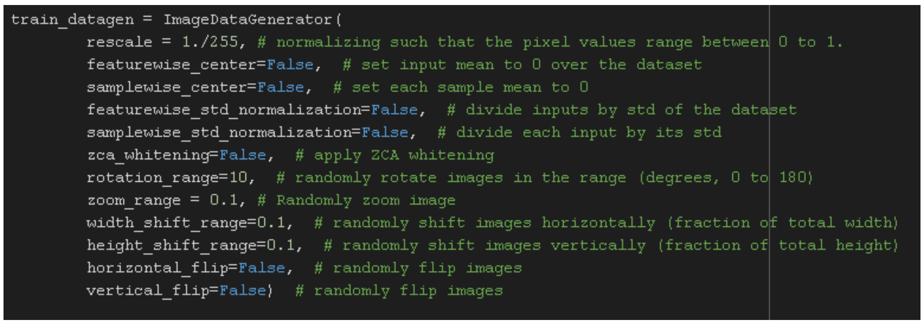


Using the cv2 reading image library, the image is translated to an array. Next, using numpy to convert it into numpy array with 1 dimension only and then feed to the model

```
(7172, 28, 28, 1) #for CNN architecture
(7172, 784) #for Random Forest Classifier and Logistic Regression
```

3.2.2 Data Augmentation

In order to avoid overfitting problem, we need to expand artificially our dataset. We can make existing dataset even larger. The idea is to alter the training data with small transformations to reproduce the variations. Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques. Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more. For this custom dataset, augmentations we use are scaling, translations, rotations, zooming. We randomly shift the images horizontally and vertically by 10%, randomly rotate images by 10%, and randomly zoom images by 10%. Flipping method is not used since the gestures are sensitive and easily classified. By applying just a couple of these transformations to our training data, we can easily double or triple the number of training examples and create a very robust model.



```
train_datagen = ImageDataGenerator(
        rescale = 1./255, # normalizing such that the pixel values range between 0 to 1.
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

3.3 Model

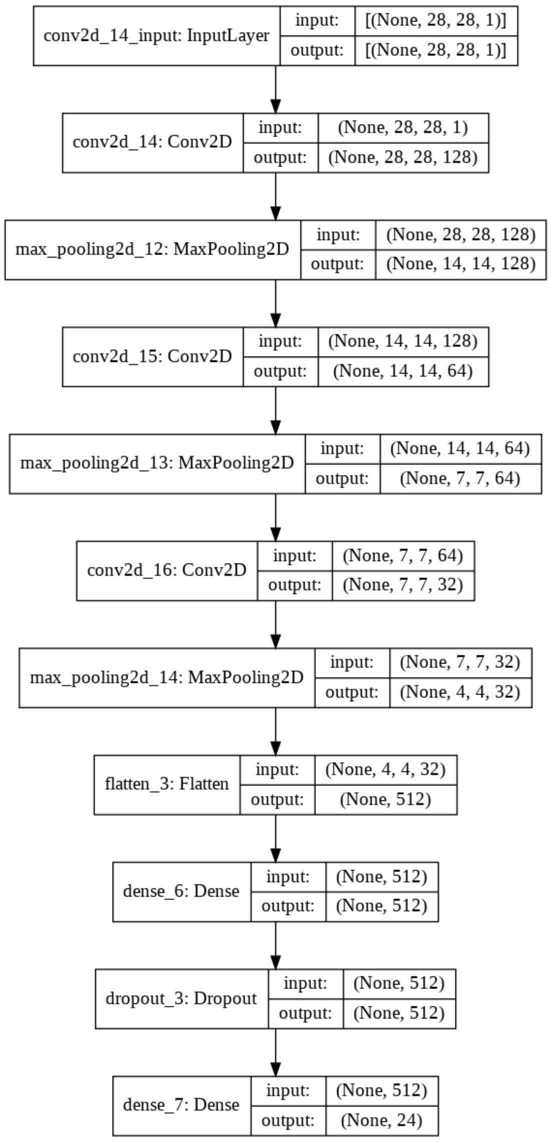
Here we want to try out different techniques, ranging from Machine Learning to Deep Convolutional Neural Network, with the aim of achieving the balance between performance/computational resources. First of all, logistic regression is the most basic technique in Machine Learning, simplest to implement with little to no tuning required. Secondly, Random Forest is a famous method for classification based on Entropy theory and probability. This give very good result in general (statistically) even with complex data structure. Finally, Convolutional Neural Network remains as the state-of-the-art model, giving out the best accuracy, with big margin comparing to other methods.

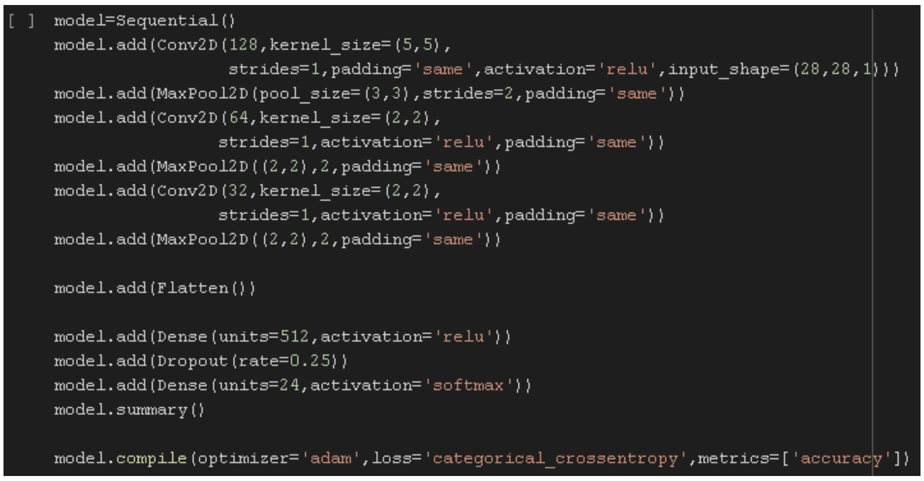
3.3.1 Convolutional Neural Network

The architecture of our model is represented in the figures below. A Convolutional Neural Network is a special type of an Artificial Intelligence implementation which uses a special mathematical matrix manipulation called the convolution operation to process data from the images.

- A convolution does this by multiplying two matrices and yielding a third, smaller matrix.
- The Network takes an input image, and uses a filter (or kernel) to create a feature map describing the image.
- In the convolution operation, we take a filter (usually 2x2 or 3x3 matrix) and slide it over the image matrix. The coresponding numbers in both matrices are multiplied and and added to yield a single number describing that input space. This process is repeated all over the image.This working can be seen in the following figure

Activation layers introduce element-wise operations to introduce non-linearities, increasing the representational power of the model. The rectification non-linearity (ReLU) uses the max(0, x) activation function, which provides the non-linearity without introducing a problem with vanishing gradients . To avoid over-fitting, pooling layers are used to reduce the spatial size of the representation, reducing the amount of parameters. Drop-out layers provide additional regularization, by only keeping each neuron active with some probability. The final layer of the CNN is a softmax classifier, producing scores that are interpreted as unnormalized log probabilities for each class.


conv2d_14_input: InputLayer
input: [(None, 28, 28, 1)]
output: [(None, 28, 28, 1)]
conv2d_14: Conv2D
input: (None, 28, 28, 1)
output: (None, 28, 28, 128)
max_pooling2d_12: MaxPooling2D
input: (None, 28, 28, 128)
output: (None, 14, 14, 128)
conv2d_15: Conv2D
input: (None, 14, 14, 128)
output: (None, 14, 14, 64)
max_pooling2d_13: MaxPooling2D
input: (None, 14, 14, 64)
output: (None, 7, 7, 64)
conv2d_16: Conv2D
input: (None, 7, 7, 64)
output: (None, 7, 7, 32)
max_pooling2d_14: MaxPooling2D
input: (None, 7, 7, 32)
output: (None, 4, 4, 32)
flatten_3: Flatten
input: (None, 4, 4, 32)
output: (None, 512)
dense_6: Dense
input: (None, 512)
output: (None, 512)
dropout_3: Dropout
input: (None, 512)
output: (None, 512)
dense_7: Dense
input: (None, 512)
output: (None, 24)



```
[ ]  model=Sequential()
     model.add(Conv2D(128,kernel_size=(5,5),
                      strides=1,padding='same',activation='relu',input_shape=(28,28,1)))
     model.add(MaxPool2D(pool_size=(3,3),strides=2,padding='same'))
     model.add(Conv2D(64,kernel_size=(2,2),
                     strides=1,activation='relu',padding='same'))
     model.add(MaxPool2D((2,2),2,padding='same'))
     model.add(Conv2D(32,kernel_size=(2,2),
                     strides=1,activation='relu',padding='same'))
     model.add(MaxPool2D((2,2),2,padding='same'))

     model.add(Flatten())

     model.add(Dense(units=512,activation='relu'))
     model.add(Dropout(rate=0.25))
     model.add(Dense(units=24,activation='softmax'))
     model.summary()

     model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

3.3.2 Random Forest Classifier

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Preliminaries: decision tree learning
Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie et al., "because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate".

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Forests are like the pulling together of decision tree algorithm efforts. Taking the teamwork of many trees thus improving the performance of a single random tree. Though not quite similar, forests give the effects of a K-fold cross validation.

Bagging

Main article: Bootstrap aggregating

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

- Sample, with replacement, n training examples from X, Y; call these Xb, Yb.
- Train a classification or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

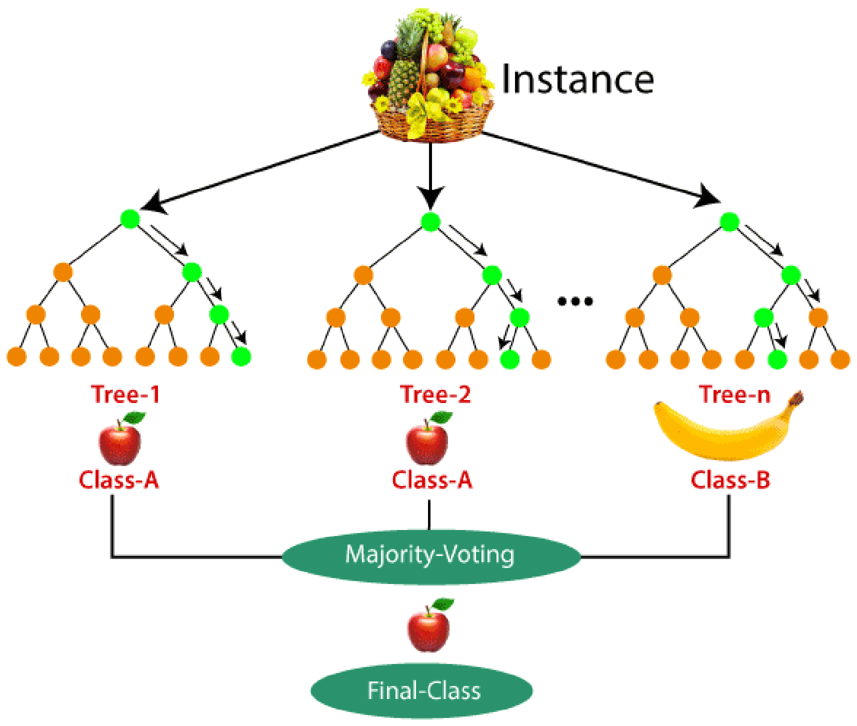
$$\sigma = \sqrt{\frac{\sum_{b-1}^{B}\left(f_b(x') - \hat{f}\right)^2}{B-1}}$$

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample x, using only the trees that did not have x in their bootstrap sample.[14] The training and test error tend to level off after some number of trees have been fit.

From bagging to random forests
Main article: Random subspace method The above procedure describes the

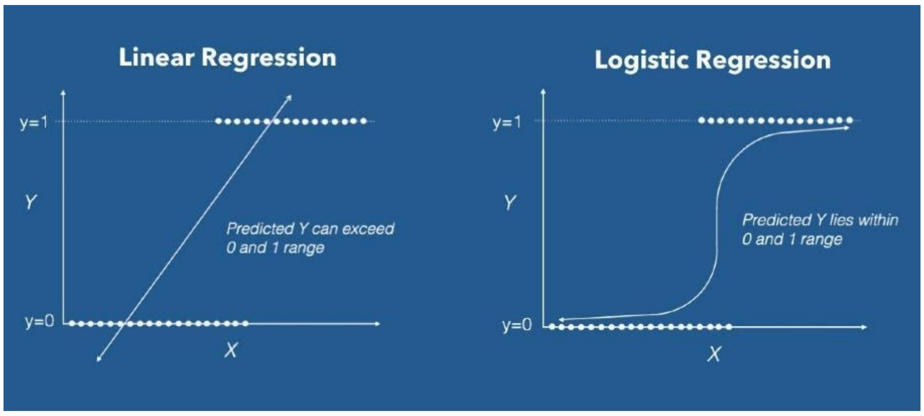
original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

Typically, for a classification problem with p features, p (rounded down) features are used in each split. For regression problems the inventors recommend p/3 (rounded down) with a minimum node size of 5 as the default.In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.



3.3.3 Logistic Regression

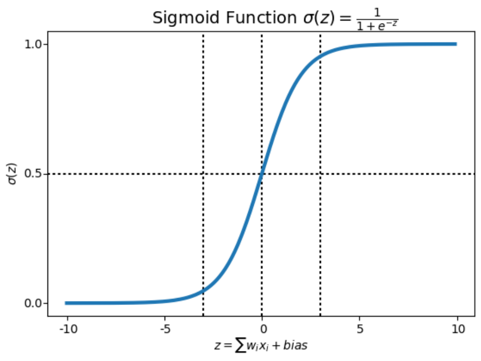
Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.


Linear Regression
y=1
Y
y=0
X
Predicted Y can exceed 0 and 1 range
Logistic Regression
y=1
Y
y=0
X
Predicted Y lies within 0 and 1 range

We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function.

$$0 \leq h_\theta(x) \leq 1$$

The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.



$$h(x) = \theta^T x$$

$$h(x) \geq 0 \Leftrightarrow f(h_\theta(x)) \geq 0.5$$

$$h(x) < 0 \Leftrightarrow f(h_\theta(x)) < 0.5$$

Cost function:

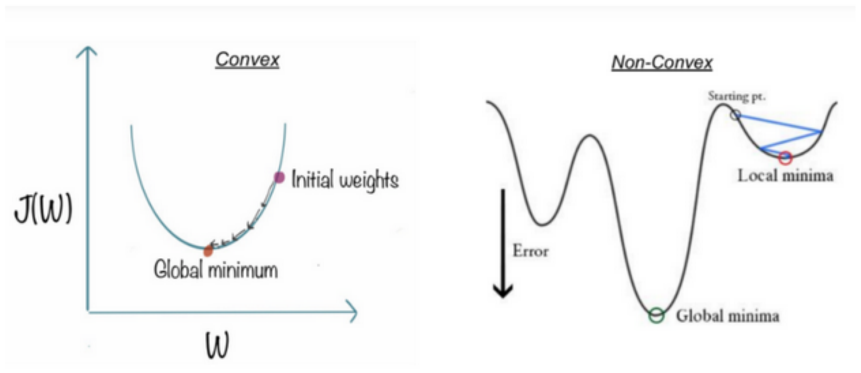
We apply cross entropy to calculate the cost function. It can be divided in to 2 parts: y = 1 and y =0:

$$cost(h_\theta(x), y) = \begin{cases} -y \cdot \log(\hat{y}) & \text{y} = 1 \\ -(1-y) \cdot \log(1-\hat{y}) & \text{y} = 0 \end{cases}$$

Finally, we can combine and have the cost function as follow:

$$J(\Theta) = \frac{1}{m} \sum_{i=0}^{n} (-y^{(i)} \cdot \log(f(h_\theta(x^i))) - (1-y^{(i)})(1 - \log(f(h_\theta(x^{(i)})))))$$

To explain why we don't use Mean Squared Error for cost function, it will cause finding the global minimum to become much harder when using with gradient descent.



4 Results

4.1 Logistic Regression

4.1.1 Original MNIST dataset

		precision	recall	f1-score	support
	0	0.88	1.00	0.93	331
	1	1.00	0.90	0.95	432
	2	0.91	0.84	0.87	310
	3	0.82	0.82	0.82	245
	4	0.83	0.86	0.85	498
	5	0.72	0.91	0.81	247
	6	0.81	0.65	0.72	348
	7	0.80	0.76	0.78	436
	8	0.67	0.67	0.67	288
	10	0.47	0.25	0.33	331
	11	0.61	0.89	0.72	209
	12	0.55	0.57	0.56	394
	13	0.66	0.43	0.52	291
	14	0.89	0.66	0.76	246
	15	0.81	0.91	0.86	347
	16	0.58	0.84	0.69	164
	17	0.22	0.42	0.29	144
	18	0.25	0.33	0.28	246
	19	0.33	0.36	0.34	248
	20	0.50	0.48	0.49	266
	21	0.83	0.48	0.61	346
	22	0.51	0.70	0.59	206
	23	0.49	0.57	0.53	267
	24	0.75	0.57	0.65	332
	accuracy			0.67	7172
Macro	macro avg	0.66	0.66	0.65	7172
Weighted	weighted avg	0.70	0.67	0.67	7172

4.1.2 Custom dataset

		precision	recall	f1-score	support
	A	0.47	0.42	0.44	331
	B	0.42	0.56	0.48	432
	C	0.37	0.48	0.42	310
	D	0.27	0.29	0.28	245
	E	0.23	0.09	0.13	498
	F	0.42	0.39	0.40	247
	G	0.23	0.29	0.26	348
	H	0.42	0.52	0.46	436
	I	0.38	0.32	0.35	288
	K	0.27	0.35	0.31	331
	L	0.50	0.53	0.51	209
	M	0.23	0.31	0.26	394
	N	0.24	0.24	0.24	291
	o	0.23	0.15	0.18	246
	P	0.51	0.41	0.45	347
	Q	0.40	0.43	0.41	164
	R	0.44	0.25	0.32	144
	S	0.23	0.24	0.23	246
	T	0.48	0.48	0.48	248
	U	0.33	0.45	0.38	266
	V	0.26	0.26	0.26	346
	W	0.30	0.37	0.33	206
	X	0.32	0.24	0.28	267
	Y	0.28	0.19	0.23	332
accuracy				0.34	7172
macro	avg	0.34	0.34	0.34	7172
weighted	avg	0.34	0.34	0.34	7172

4.2 Random Forest

4.2.1 Original MNIST dataset

		precision	recall	F1-score	support
	0	0.74	0.96	0.83	331
	1	0.79	0.88	0.83	432
	2	0.76	0.90	0.82	310
	3	0.62	0.78	0.69	245
	4	0.80	0.89	0.84	498
	5	0.68	0.81	0.74	247
	6	0.72	0.82	0.77	348
	7	0.89	0.80	0.84	436
	8	0.45	0.49	0.47	288
	10	0.55	0.47	0.51	331
	11	0.70	0.82	0.76	209
	12	0.62	0.45	0.52	394
	13	0.47	0.28	0.35	291
	14	0.84	0.69	0.76	246
	15	0.86	0.84	0.85	347
	16	0.81	0.93	0.87	164
	17	0.23	0.45	0.30	144
	18	0.50	0.61	0.55	246
	19	0.55	0.60	0.57	248
	20	0.46	0.44	0.45	266
	21	0.66	0.43	0.52	346
	22	0.48	0.43	0.45	206
	23	0.71	0.54	0.61	267
	24	0.77	0.43	0.55	332
	accuracy			0.67	7172
macro	avg	0.65	0.66	0.64	7172
weighted	avg	0.68	0.67	0.66	7172

4.2.2 Custom dataset

		precision	recall	f1-score	support
	A	0.64	0.95	0.76	331
	B	0.75	0.82	0.79	432
	C	0.78	0.83	0.81	310
	D	0.56	0.91	0.69	245
	E	0.75	0.83	0.79	498
	F	0.64	0.85	0.73	247
	G	0.65	0.78	0.71	348
	H	0.91	0.78	0.84	436
	I	0.52	0.48	0.49	288
	K	0.50	0.44	0.46	331
	L	0.76	0.77	0.76	209
	M	0.63	0.49	0.55	394
	N	0.42	0.32	0.36	291
	O	0.72	0.56	0.63	246
	P	0.84	0.93	0.89	347
	Q	0.79	0.87	0.83	164
	R	0.27	0.62	0.38	144
	S	0.51	0.48	0.50	246
	T	0.61	0.60	0.60	248
	U	0.56	0.51	0.53	266
	V	0.66	0.32	0.43	346
	W	0.40	0.37	0.38	206
	X	0.79	0.55	0.65	267
	Y	0.81	0.44	0.57	332
accuracy				0.65	7172
macro	avg	0.64	0.65	0.63	7172
weighted	avg	0.67	0.65	0.65	7172

4.3 Convolutional Neural Network

4.3.1 Original MNIST Dataset

		precision	recall	f1-score	support
	0	0.95	1.00	0.98	331
	1	1.00	1.00	1.00	432
	2	1.00	1.00	1.00	310
	3	1.00	0.94	0.97	245
	4	1.00	1.00	1.00	498
	5	1.00	1.00	1.00	247
	6	0.97	0.97	0.97	348
	7	1.00	0.97	0.99	436
	8	0.95	1.00	0.97	288
	9	0.00	0.00	0.00	0
	10	0.00	0.00	0.00	331
	11	0.00	0.00	0.00	209
	12	0.02	0.02	0.02	394
	13	0.00	0.00	0.00	291
	14	0.00	0.00	0.00	246
	15	0.00	0.00	0.00	347
	16	0.98	1.00	0.99	164
	17	0.98	1.00	0.99	144
	18	1.00	0.85	0.92	246
	19	1.00	1.00	1.00	248
	20	1.00	0.98	0.99	266
	21	0.95	1.00	0.97	346
	22	1.00	1.00	1.00	206
	23	0.97	1.00	0.99	267
	24	1.00	1.00	1.00	332
accuracy				0.74	7172
macro	avg	0.71	0.71	0.71	7172
weighted	avg	0.74	0.74	0.74	7172

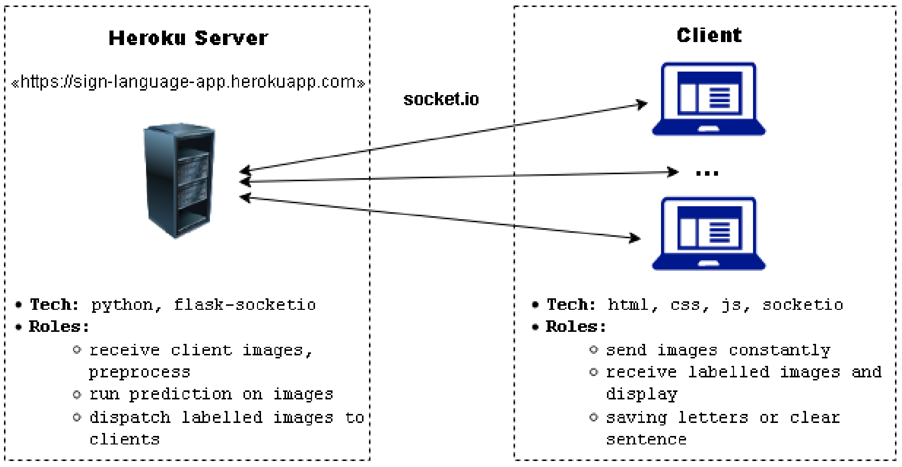
4.3.2 Custom dataset

		precision	recall	f1-score	support
	A	1.00	1.00	1.00	331
	B	1.00	1.00	1.00	432
	C	1.00	0.97	0.99	310
	D	1.00	1.00	1.00	245
	E	1.00	1.00	1.00	498
	F	0.98	1.00	0.99	247
	G	1.00	0.98	0.99	348
	H	1.00	0.99	1.00	436
	I	1.00	1.00	1.00	288
	K	1.00	1.00	1.00	331
	L	1.00	1.00	1.00	209
	M	1.00	1.00	1.00	394
	N	1.00	1.00	1.00	291
	o	0.96	1.00	0.98	246
	P	1.00	1.00	1.00	347
	Q	1.00	1.00	1.00	164
	R	1.00	1.00	1.00	144
	S	0.99	1.00	0.99	246
	T	1.00	1.00	1.00	248
	U	1.00	0.91	0.95	266
	V	0.94	1.00	0.97	346
	W	1.00	1.00	1.00	206
	X	1.00	1.00	1.00	267
	Y	1.00	1.00	1.00	332
	accuracy			0.99	7172
Macro	avg	0.99	0.99	0.99	7172
Weighted	avg	0.99	0.99	0.99	7172

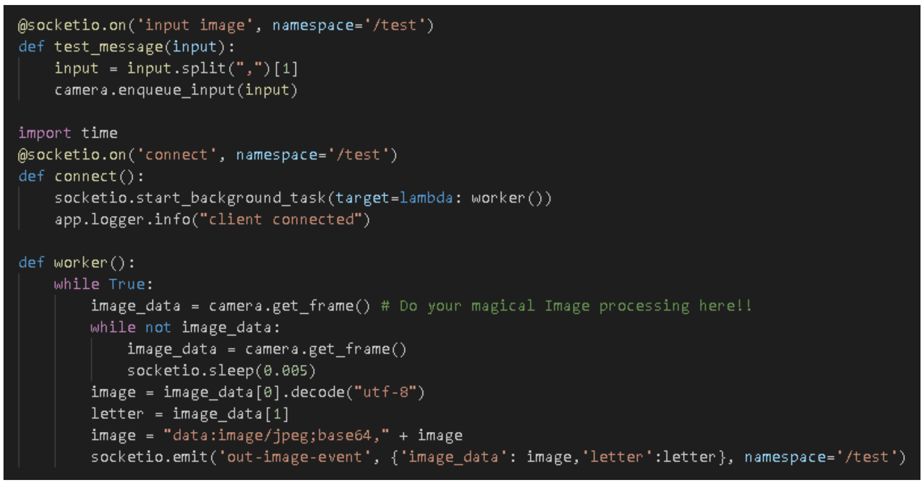
4.3.3 Evaluation result analysis

With logistic regression, it comes as no surprise that this model performance is the lowest. Due to high dimension large number of training samples, logistic regression can correctly predict 30% of the real dataset. This is not acceptable in real world instance. Random forest brings about better performance, at around 66% in custom dataset. This is a huge improvement comparing with logistic regression model but still not applicable to practical situation. With CNN, this is a game changer with the accuracy of approximately 100%. This outperform the former 2 models while consuming a bit higher computational resource/training time. Therefore we choose to implement this model in the application.

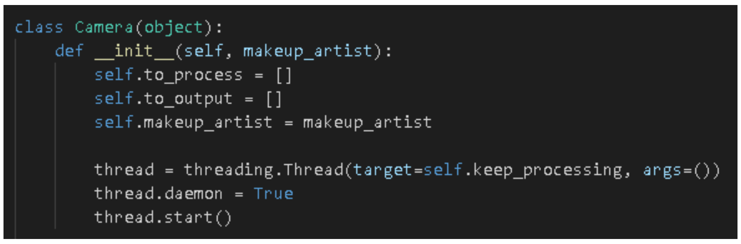
5 Deploy


Heroku Server
«https://sign-language-app.herokuapp.com»
socket.io
Client
...
• Tech: python, flask-socketio
• Roles:
∘ receive client images, preprocess
∘ run prediction on images
∘ dispatch labelled images to clients
• Tech: html, css, js, socketio
• Roles:
∘ send images constantly
∘ receive labelled images and display
∘ saving letters or clear sentence

- **Heroku Server**



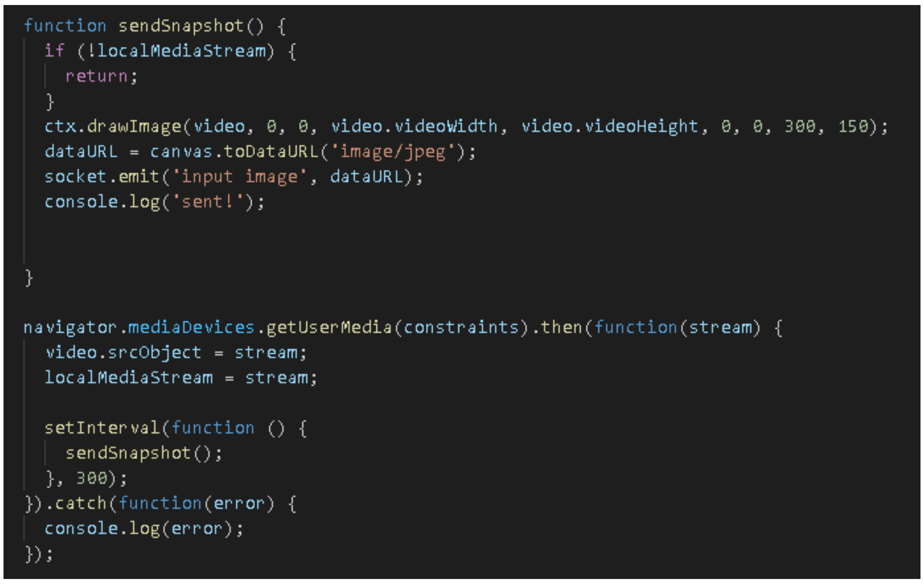
```
@socketio.on('input image', namespace='/test')
def test_message(input):
    input = input.split(",")[1]
    camera.enqueue_input(input)

import time
@socketio.on('connect', namespace='/test')
def connect():
    socketio.start_background_task(target=lambda: worker())
    app.logger.info("client connected")

def worker():
    while True:
        image_data = camera.get_frame() # Do your magical Image processing here!!
        while not image_data:
            image_data = camera.get_frame()
            socketio.sleep(0.005)
        image = image_data[0].decode("utf-8")
        letter = image_data[1]
        image = "data:image/jpeg;base64," + image
        socketio.emit('out-image-event', {'image_data': image,'letter':letter}, namespace='/test')
```



```
class Camera(object):
    def __init__(self, makeup_artist):
        self.to_process = []
        self.to_output = []
        self.makeup_artist = makeup_artist

        thread = threading.Thread(target=self.keep_processing, args=())
        thread.daemon = True
        thread.start()
```

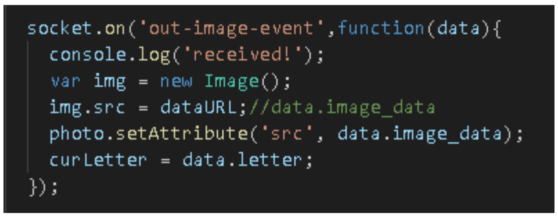
Storing incoming images into input_queue and yielding labelled images into output_queue. Broadcasting images in output_queue one by one

- **Client**



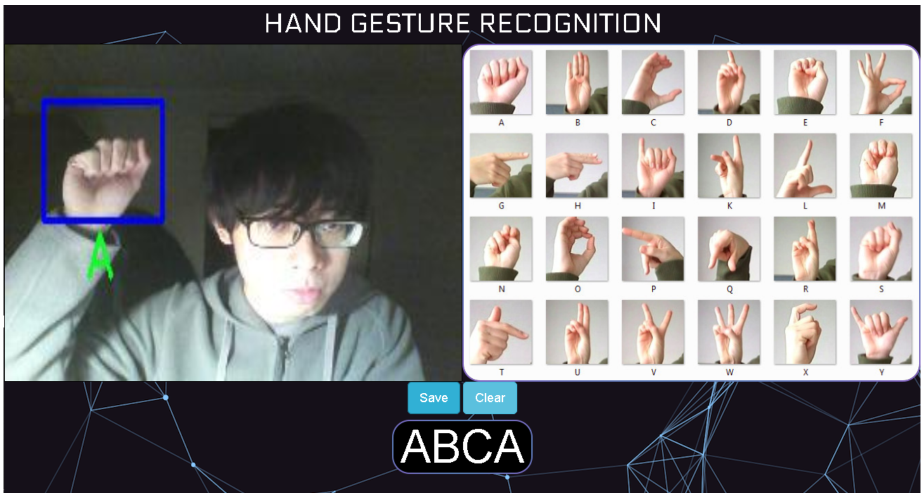
```
function sendSnapshot() {
  if (!localMediaStream) {
    return;
  }
  ctx.drawImage(video, 0, 0, video.videoWidth, video.videoHeight, 0, 0, 300, 150);
  dataURL = canvas.toDataURL('image/jpeg');
  socket.emit('input image', dataURL);
  console.log('sent!');


}

navigator.mediaDevices.getUserMedia(constraints).then(function(stream) {
  video.srcObject = stream;
  localMediaStream = stream;

  setInterval(function () {
    sendSnapshot();
  }, 300);
}).catch(function(error) {
  console.log(error);
});
```

Send images of client's webcam every 300 ms



```
socket.on('out-image-event',function(data){
  console.log('received!');
  var img = new Image();
  img.src = dataURL;//data.image_data
  photo.setAttribute('src', data.image_data);
  curLetter = data.letter;
});
```

Display the received image on event



6 Roles

Nguyen Gia Bach: Data capture, deep learning model build, web deploy

Nguyen Tien Dat: Data capture, build website, Random Forest classifier

Nguyen Hoai Nam : Website building and deployment.

7 Conclusion

Deep learning models are proven to be far superior than traditional machine learning algorithms by a large margin with little to no extra manual tuning from experts. I believe deep learning has been a great benefactor to many scientific and industrial field by providing automated tasks with great accuracy. This project is just an implementation of a small network architecture, yet it achieve a near absolute accuracy, which suggests an enormous potential for application in the future.

8 Future work

- Automatic detect hand gesture (localization)
- Classify complex gestures (time series data) or hand animations using LSTM, GRU

9 Appendix

9.1 Source code

The project git repository link is:

https://github.com/bachzz/hand-gesture-recognition

References

[1] Kaggle, “Kaggle asl-alphabet, image data set for alphabets in the american sign language,” Available at https://www.kaggle.com/grassknoted/asl-alphabet (accessed 22 Dec. 2020).

[2] ——, “Kaggle sign-langeuage-mnist - drop-in replacement for mnist for hand gesture recognition tasks,” Available at https://www.kaggle.com/datamunge/sign-language-mnist (accessed 22 Dec. 2020).

[3] Wikipedia, “Wikipedia - mnist database,” Available at https://en.wikipedia.org/wiki/MNIST_database (accessed 22 Dec. 2020).