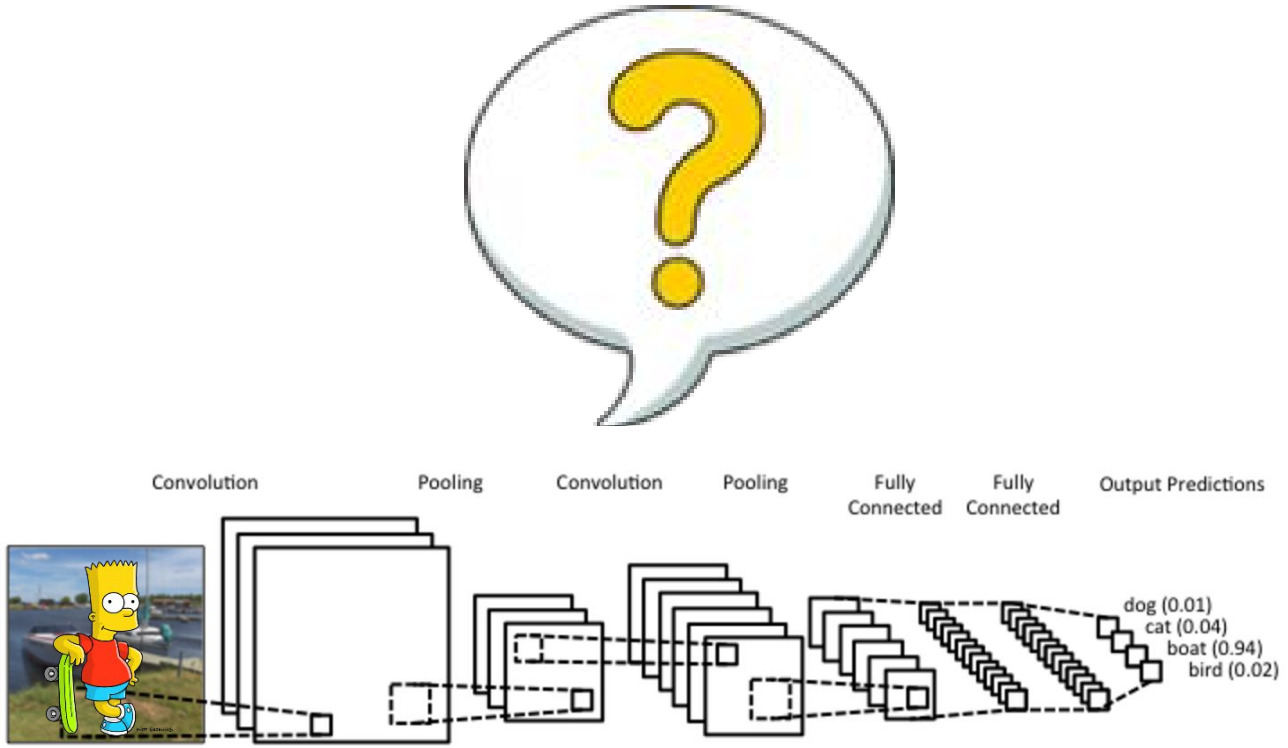
The Simpsons characters: recognition and detection



THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Agenda

1. Introduction
2. The project
3. Results
4. Conclusions

Why Image Recognition is important?

The reasons why we consider this topic so important is because of the number of applications we can imagine. The Image recognition is determinant in the image search engines such as Google. Also, cancer detection in x-ray images to assist doctors and so on and also important in security systems such as face recognition.

How this project is important for other projects?

The process followed to solve this challenge is very similar to other projects that have the same objective. What we pursue is to understand in practice the model used. we could make a hard sufficiently image detection problem to be applied in the future.

Purposes of the project

- Apply the Convolutional Neural Network for image recognition
- Apply the Multilayer Perceptron for image recognition
- Compare CNN and MLP Performance and Results
- Compare GPU and the Local Computer to run the Model

Description of the dataset

Source: Kaggle

Classes : 18 classes

Training set size: About 1,000 images per character. 16,143 images. 2,849.

Zip file: 600Mb

	name	train	test	total
1	abraham_grampa_simpson	913	48	961
2	apu_nahasapeemapetilon	623	50	673
3	bart_simpson	1342	50	1392
4	charles_montgomery_burns	1193	48	1241
5	chief_wiggum	986	50	1036
6	comic_book_guy	469	49	518
7	edna_krabappel	457	50	507
8	homer_simpson	2246	50	2296
9	kent_brockman	498	50	548
10	krusty_the_clown	1206	50	1256
11	lisa_simpson	1354	50	1404
12	marge_simpson	1291	50	1341
13	milhouse_van_houten	1079	49	1128
14	moe_szyslak	1452	50	1502
15	ned_flanders	1454	49	1503
16	nelson_muntz	358	50	408
17	principal_skinner	1194	50	1244
18	sideshow_bob	877	47	924

Description of the dataset

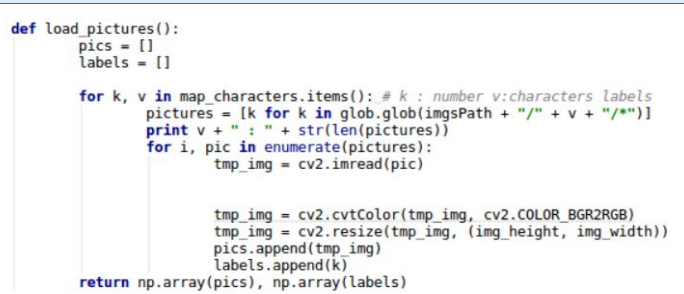
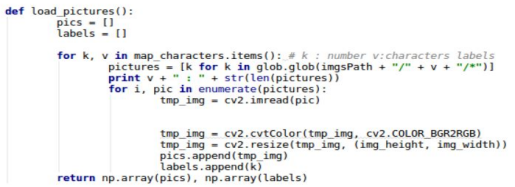
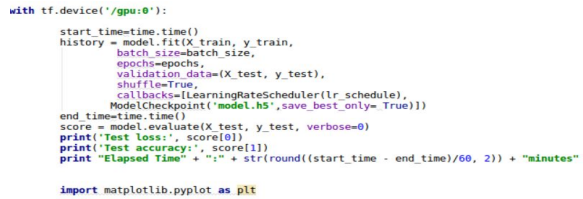
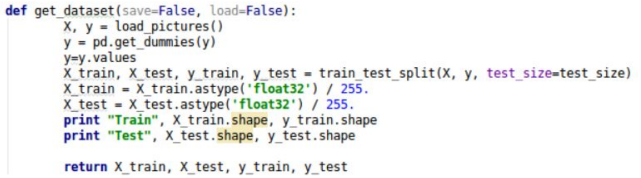
Data Preprocessing

Resize the images

1. image size: 42X42X3
2. convert categories to vectors
3. Save memory: float32
4. Normalization: /255

```
abraham_grampa_simpson : 913
apu_nahasapeemapetilon : 623
bart_simpson : 1342
charles_montgomery_burns : 1193
chief_wiggum : 986
comic_book_guy : 469
edna_krabappel : 457
homer_simpson : 2246
kent_brockman : 498
krusty_the_clown : 1206
lisa_simpson : 1354
marge_simpson : 1291
milhouse_van_houten : 1079
moe_szyslak : 1452
ned_flanders : 1454
nelson_muntz : 358
principal_skinner : 1194
sideshow_bob : 877
Train (16143, 42, 42, 3) (16143, 1
Test (2849, 42, 42, 3) (2849, 18)
```

Packages used:

Action	Packages used and explanations	Command
Import the Data	*CV2* *Allows to read an OpenCV documentation*	
Read the Folders	*Glob* *The glob module finds all the pathnames matching a specific pattern according to the rules used by the Unix shell, although results are returned in arbitrary order*	
Calculate time to run	*Time* *This package allows us to calculate the time used to run the commands.*	
Split the dataset	*sklearn train_test_split :Split arrays or matrices into random train and test subsets*	

Environment used:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

Recommendations:

Use Keras if you need a deep learning library that:

Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).

Supports both convolutional networks and recurrent networks, as well as combinations of the two.

Runs seamlessly on CPU and GPU.

Multi-layer Perceptron

Six-layer feed forward neural network

512 neurons in each hidden layer

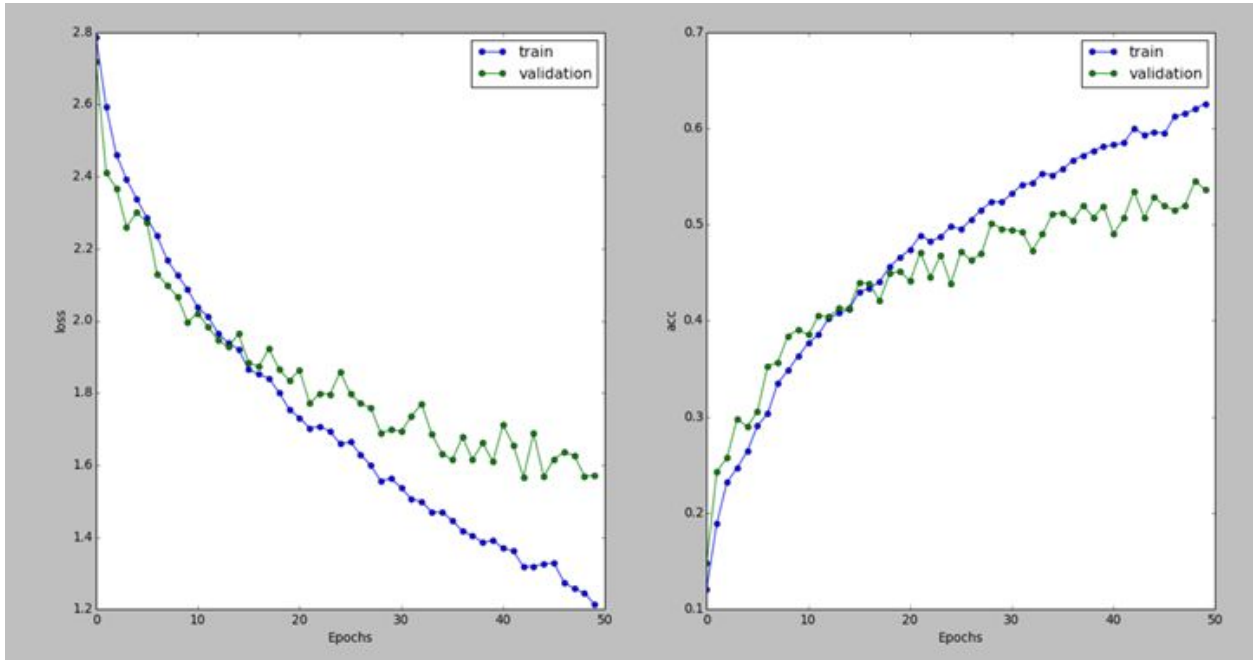
Relu activation function

Dropout layer

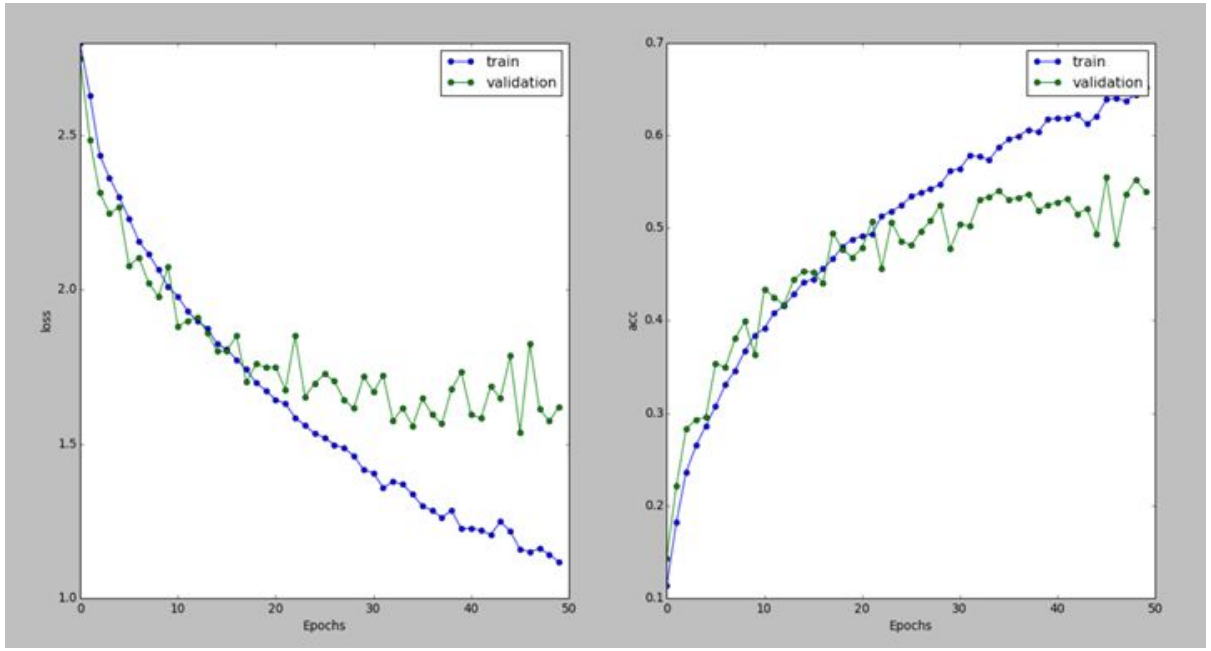
```
# MLP for multi-class softmax classification
with tf.device('/gpu:0'):
    start_time=time.time()
    model_MLP= Sequential()

    model_MLP.add(Dense(512,activation='relu', input_shape=(5292,)))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dense(512, activation='relu'))
    model_MLP.add(Dropout(0.2))
    #drop-out layer for avoid over-fitting
    model_MLP.add(Dense(512, activation='relu'))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dense(512, activation='relu'))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dense(512, activation='relu'))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dense(512, activation='relu'))
    model_MLP.add(Dropout(0.2))
    model_MLP.add(Dense(18,activation='softmax'))

    sgd=SGD(lr=0.01,decay=1e-6, momentum=0.9, nesterov=True)
    model_MLP.compile(loss='categorical_crossentropy',
                      optimizer=sgd,
                      metrics=['accuracy'])
```

Results

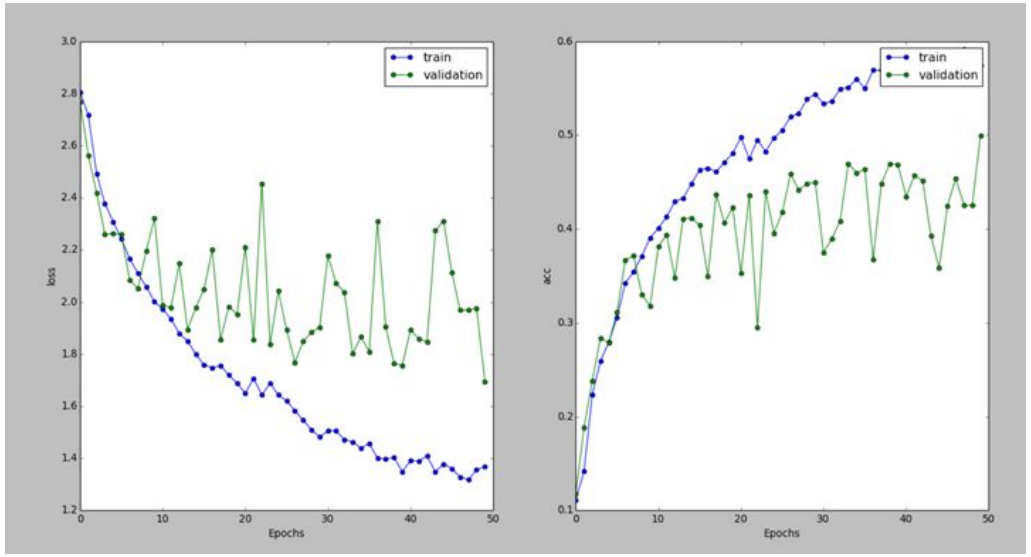
Batch-Size	Epochs	Accuracy	Loss	Elapsed Time
32	30	48.16%	1.72	3.41 minutes
32	50	48.15%	1.57	5.63 minutes
64	30	46.72%	1.81	1.8 minutes
64	50	**53.94%**	1.62	3.24 minutes
128	30	40.96%	1.95	1.09 minutes
128	50	49.94%	1.69	1.79 minutes



mini-batch size = 32 epochs =50



mini-batch size = 64 epochs =50

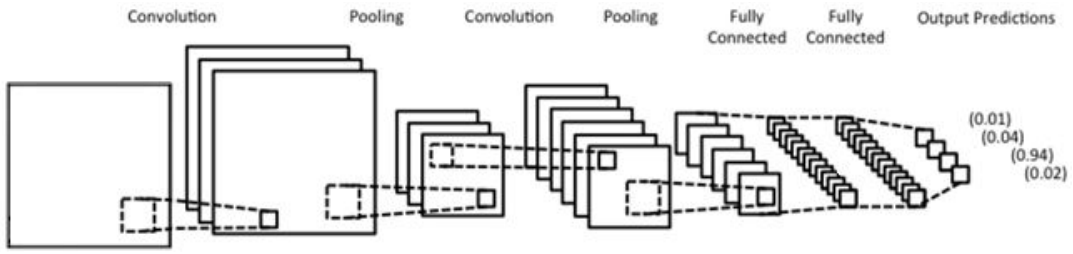


mini-batch size = 128 epochs =50

increase the size of batch→ validation loss and accuracy fluctuate much more widely

5-layers CNN

```
def create_model_four_conv(input_shape):
        model = Sequential()
        model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_shape))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Conv2D(64, (2, 2), activation='relu'))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Conv2D(128, (4, 4), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Flatten())
        model.add(Dense(1024, activation='relu'))
        model.add(Dropout(0.5))

        model.add(Dense(num_classes, activation='softmax'))

        return model;
```

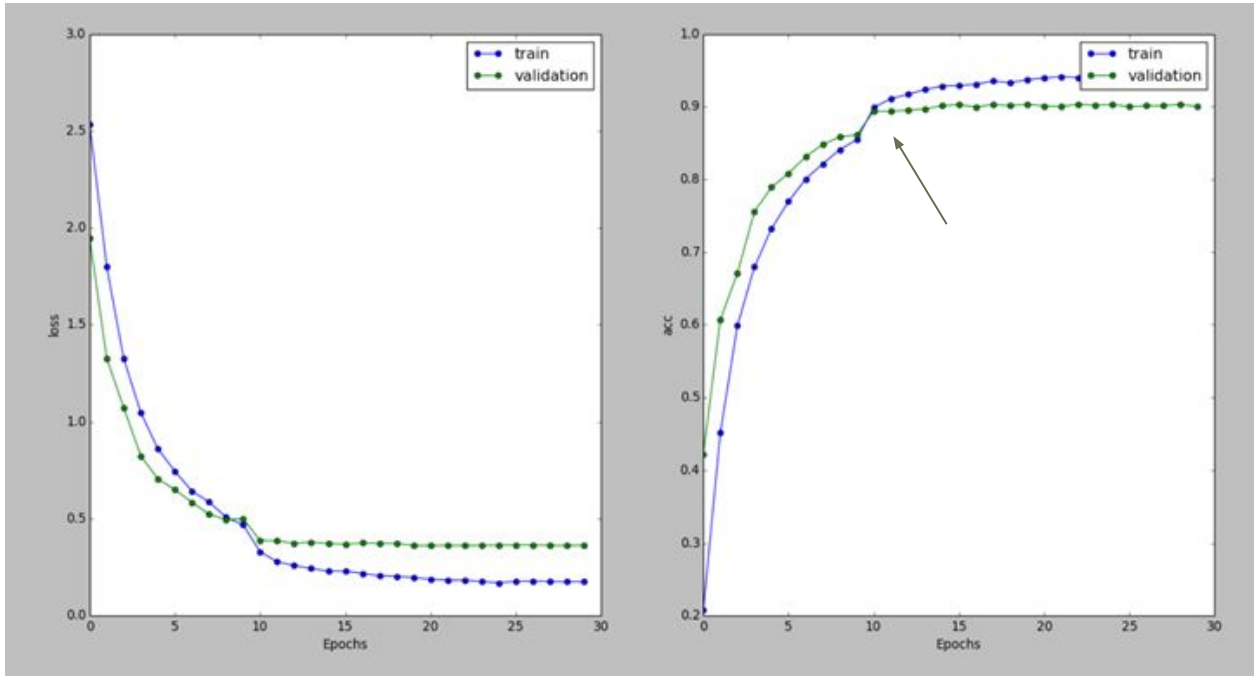


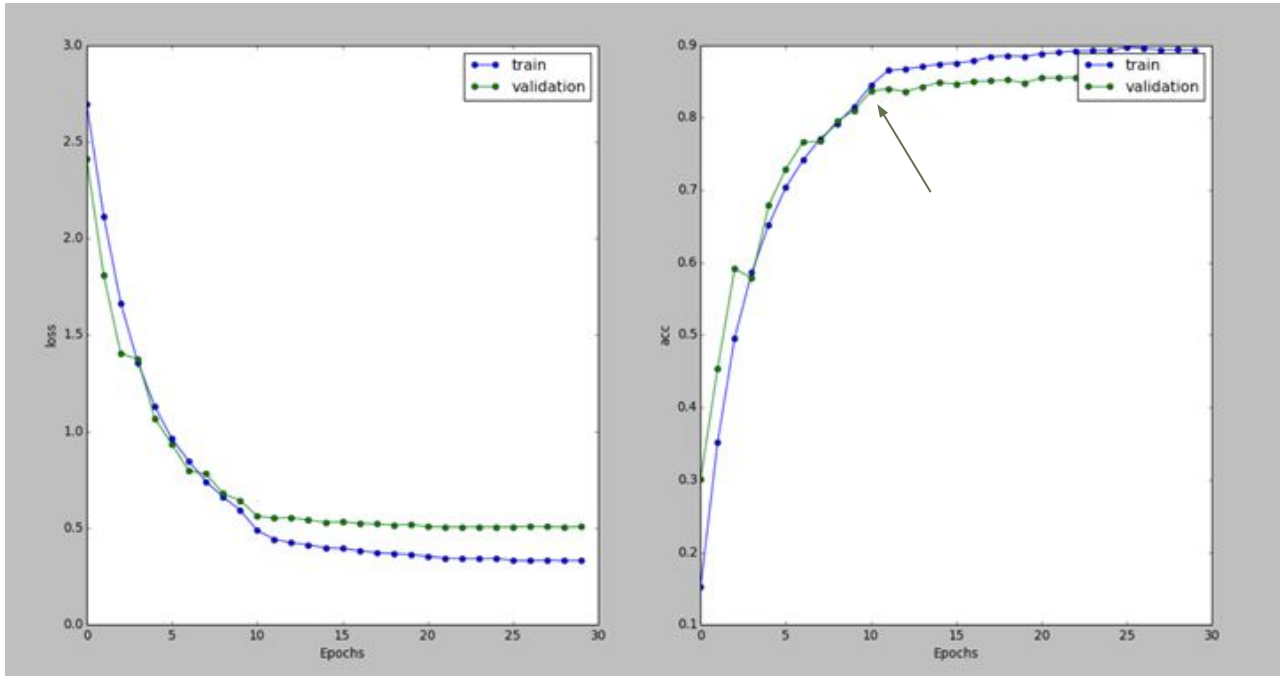
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 42, 42, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 21, 21, 32)	0
dropout_1 (Dropout)	(None, 21, 21, 32)	0
conv2d_2 (Conv2D)	(None, 20, 20, 64)	8256
conv2d_3 (Conv2D)	(None, 18, 18, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 9, 9, 64)	0
dropout_2 (Dropout)	(None, 9, 9, 64)	0
conv2d_4 (Conv2D)	(None, 6, 6, 128)	131200
max_pooling2d_3 (MaxPooling2	(None, 3, 3, 128)	0
dropout_3 (Dropout)	(None, 3, 3, 128)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 1024)	1180672
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 18)	18450

Total params: 1,376,402
Trainable params: 1,376,402
Non-trainable params: 0

Results

Batch-Size	Epochs	Accuracy	Loss	Elapsed Time
32	30	**90.03%**	0.3635	4.65 minutes
64	30	85.64%	0.5079	3.18 minutes





6-layers CNN

```
def create_model_six_conv(input_shape):
        model = Sequential()
        model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_shape))
        model.add(Conv2D(32, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
        model.add(Conv2D(128, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Flatten())
        model.add(Dense(1024, activation='relu'))
        model.add(Dropout(0.5))

        model.add(Dense(num_classes, activation='softmax'))

        return model;
```

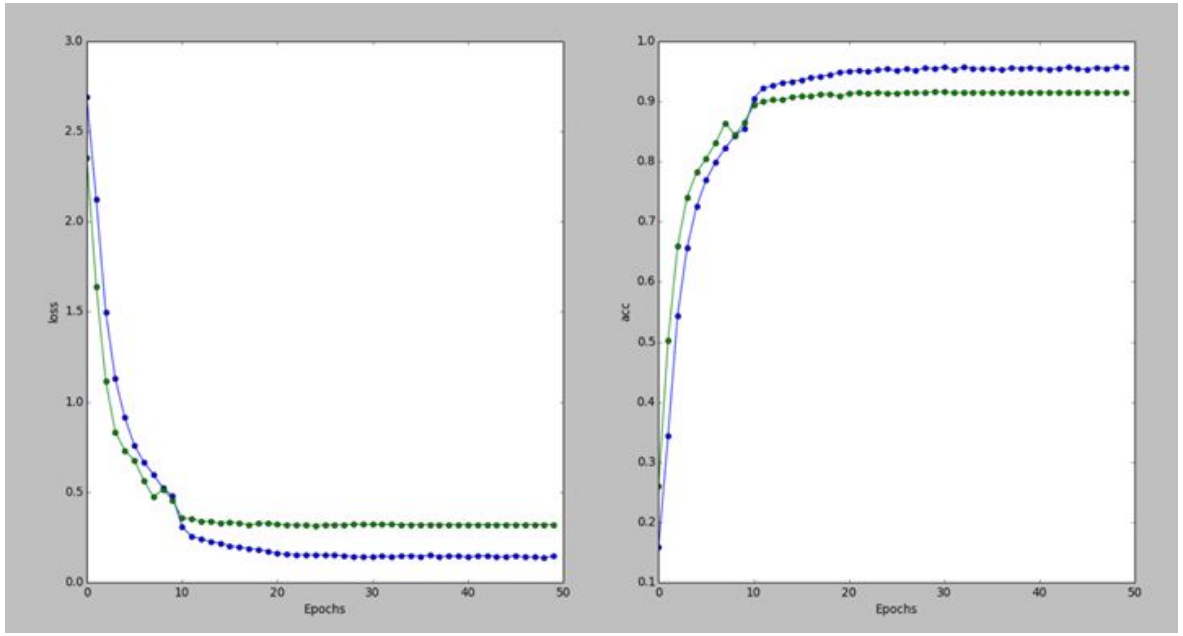
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 42, 42, 32)	896
conv2d_2 (Conv2D)	(None, 40, 40, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 20, 20, 32)	0
dropout_1 (Dropout)	(None, 20, 20, 32)	0
conv2d_3 (Conv2D)	(None, 20, 20, 64)	18496
conv2d_4 (Conv2D)	(None, 18, 18, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 9, 9, 64)	0
dropout_2 (Dropout)	(None, 9, 9, 64)	0
conv2d_5 (Conv2D)	(None, 9, 9, 128)	73856
conv2d_6 (Conv2D)	(None, 7, 7, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 3, 3, 128)	0
dropout_3 (Dropout)	(None, 3, 3, 128)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 1024)	1180672
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 18)	18450

Total params: 1,486,130
Trainable params: 1,486,130
Non-trainable params: 0

Maintain the total parameters→
adjust the kernel size for each convolution layers

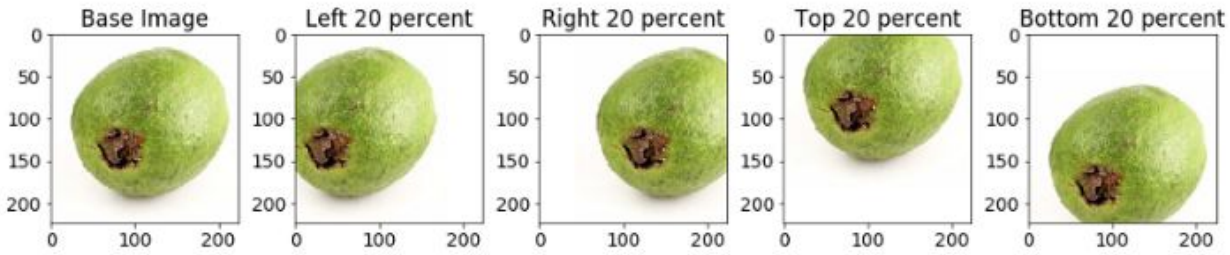
Results

Batch-Size	Epochs	Accuracy	Loss	Elapsed Time
32	30	91.30%	0.3312	5.83 minutes
32	50	**91.47%**	0.3206	**10.42 minutes**
64	30	87.93%	0.4315	4.95 minutes
64	50	87.99%	0.4329	7.81 minutes



Data Augmentation

```
datagen = ImageDataGenerator(
            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)
            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=True,  # randomly flip images
            vertical_flip=False)  # randomly flip images
```



Background color white of image blends with added background color white

Results

Batch-Size	Epochs	Accuracy	Loss	Elapsed Time
32	30	91.30%	0.3312	5.83 minutes
32	50	**91.47%**	0.3206	10.42 minutes
64	30	87.93%	0.4315	4.95 minutes
64	50	87.99%	0.4329	7.81 minutes
Data Augmentation 32	50	**90.87%**	0.3375	11.01minutes

Precision Report

	precision	recall	f1-score
abraham_grampa_simpson	0.92	0.87	0.89
apu_nahasapeemapetilon	0.90	0.93	0.92
bart_simpson	0.88	0.87	0.87
charles_montgomery_burns	0.88	0.78	0.83
chief_wiggum	0.93	0.90	0.91
comic_book_guy	0.96	0.81	0.88
edna_krabappel	0.90	0.80	0.85
homer_simpson	0.90	0.90	0.90
kent_brockman	0.96	0.97	0.96
krusty_the_clown	0.88	0.97	0.92
lisa_simpson	0.85	0.88	0.87
marge_simpson	0.94	0.97	0.96
milhouse_van_houten	0.96	0.96	0.96
moe_szyslak	0.94	0.94	0.94
ned_flanders	0.92	0.96	0.94
nelson_muntz	0.89	0.78	0.83
principal_skinner	0.88	0.93	0.91
sideshow_bob	0.92	0.96	0.94
avg / total	0.91	0.91	0.91

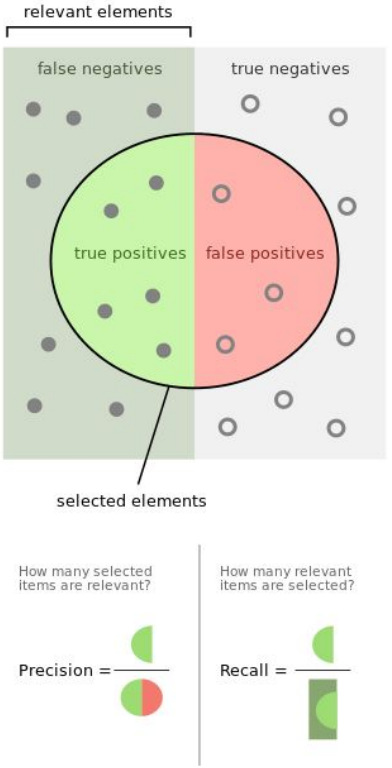
Precision Report Interpretation

Precision: is the number of correct positive results divided by the number of all positive results

Recall: is the number of correct positive results divided by the number of positive results that should have been returned

F-1 Score:

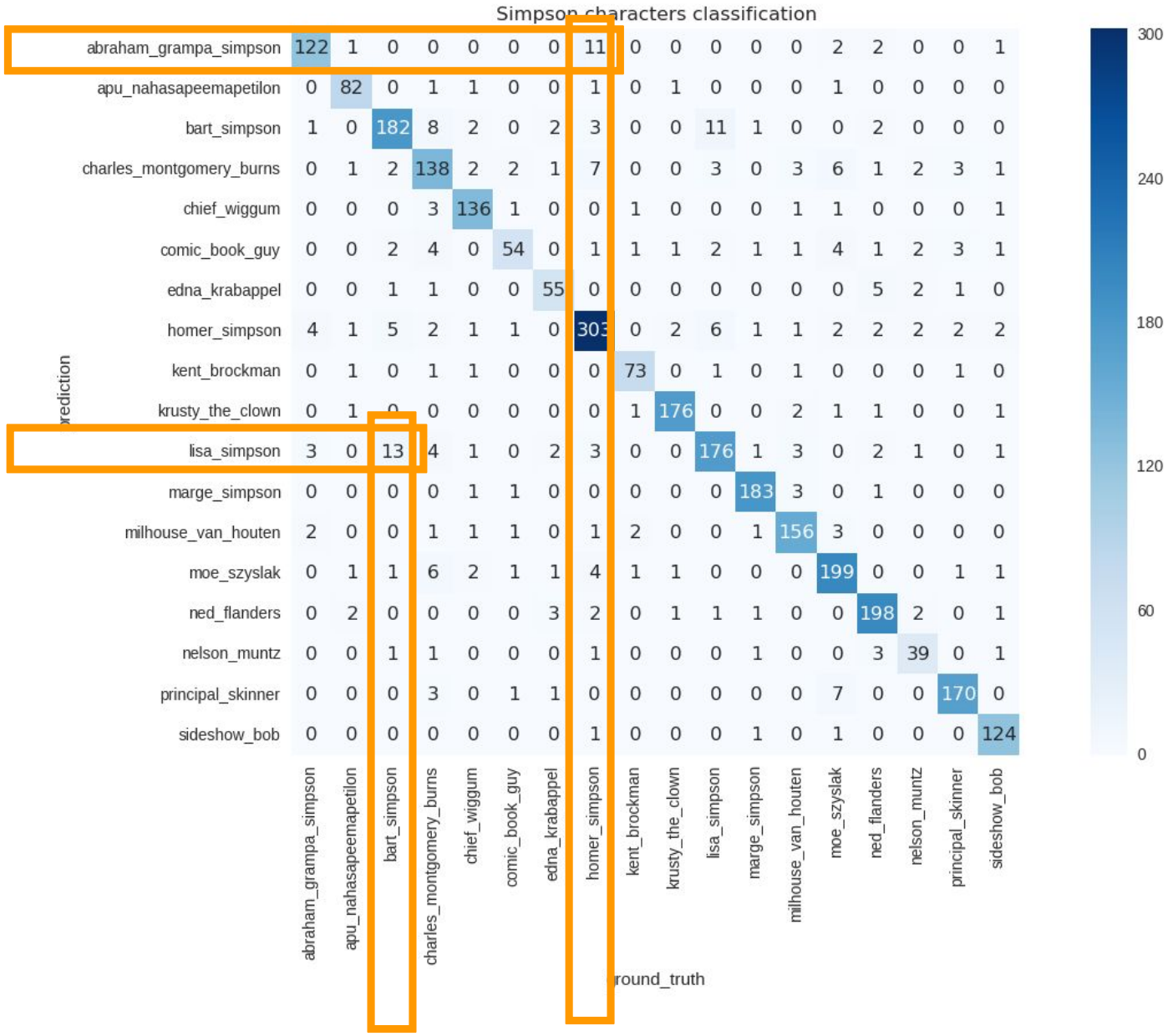
$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$



Test Prediction

Characters Classification

deeper color = high accuracy



Description of the dataset

Conclusions

1. The Convolutional Neural Network has a better result than MLP. (Accuracy : 50% Vs. 91%).
2. The local computer allows us to run the model but in a very slow speed. (50 mins vs. 5 mins).
3. Increase the inputs will increase the results.
4. We want to apply this model to another dataset created by us in the future.