The George Washington University
Washington D.C.

Machine Learning Report on Image Recognition (Simpsons characters recognition and detection)

By:
Teresalina Paez (G26733064)
Chun-Wei Lo (G39442195)
Wanding Zhang (G4926834)

1. Introduction

In the last few years, the models developed in the field of Machine Learning have allowed us among other things to simulate human vision. Given the wide ranges of being able to recognize and identify images, we decided to carry out this project. The television series The Simpsons, in addition to being extremely famous, provides us an important amount of images of its characters, which allowed us to fulfill the objectives of our project. In this report, we demonstrate a real-world application by identifying the characters using Machine Learning algorithms. Also, we want to compare the advantages and disadvantages of using GPU instead of a local computer. Likewise, we will compare the results of two neural networks models. Finally, we want to recommend based on these two comparisons how is better to run these kind of models and which model gives us the best results.

2. Description of the data set

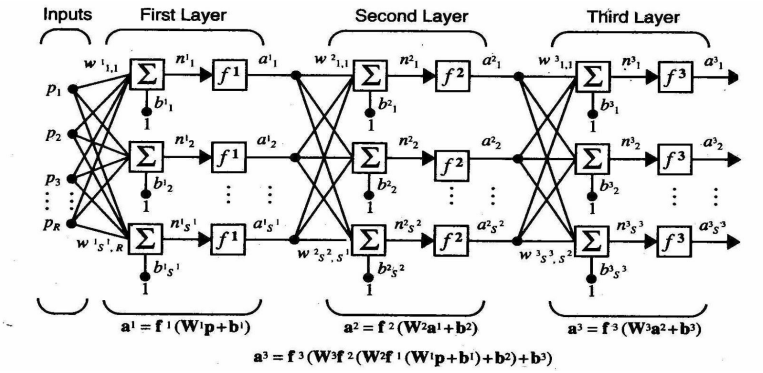
We found our data of labeled pictures of the Simpsons characters on Kaggle. The dataset currently features 18 classes/characters. The pictures are under various size, scenes, could be cropped from other characters and are mainly extracted from episodes (season 4 to 24). The training set includes about 1000 images per character. The character is not necessarily centered in each image and could sometimes be with other characters (but it should be the most important part in the picture). We collected 18,992 images, including training set of 16,143 images and testing 2,849 images respectively.

3. Method

We started exploring and preprocessing the data. Given the process of collection of the information, it was necessary to carry out some steps prior to initial with the model. First, we converted the pictures to the same size. We converted categories to vector, the new image size was 42X42X3. Also, to save memory the format was changed to float32, and finally the normalization[1] to 255. For the purposes of this project the models selected were: Multilayer neural network and convolutional networks.

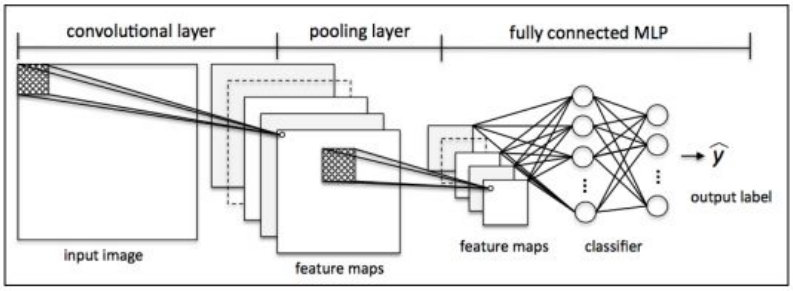
[1] process that changes the range of pixel intensity values.

- **Multilayer Neural Network** can be applied to classification problems. It offers a way to introduce nonlinearities to regression/classification models. It is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs.



Neural Network Design, Hagan, Demuth, Beale, De Jesus.

- **Convolutional Neural Network: This neural network is recommended to image recognition.** we used a feed forward 4 convolutional layers with ReLU activation followed by a fully connected hidden layer. The best explanation found was: "A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m* m*r image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3. The convolutional layer will have k filters (or kernels) of size n* n * q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m−n+1. Each map is then subsampled typically with mean or max pooling over p * p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map."[2]

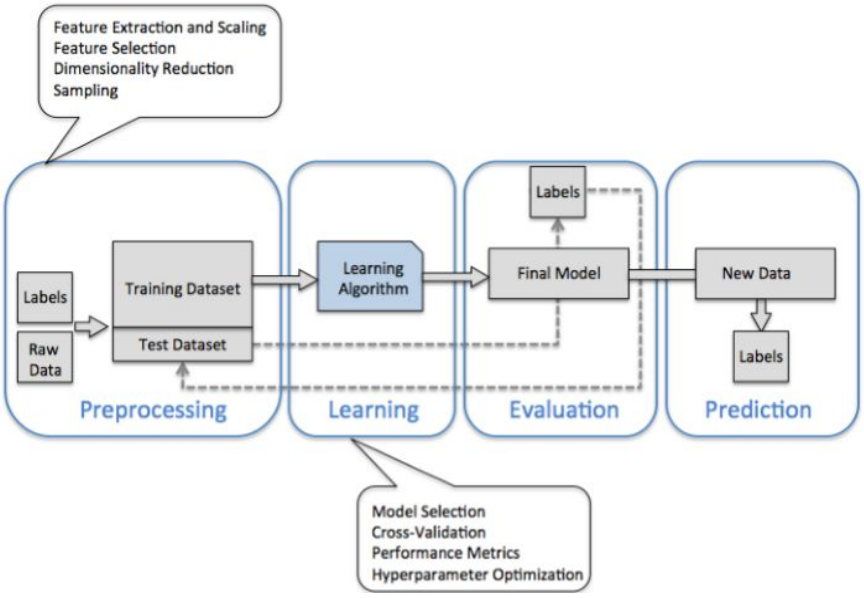


Python Machine Learning. Sebastian Raschka .2015

[2] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

4. Experimental setup

The following flow chart describe the steps that we will be going through:



Python Machine Learning. Sebastian Raschka .2015

In the first stage the data was preprocessed, and finally splitted in training and testing dataset. This step is basically to see if our machine learning algorithm not only performs well on the training set but also generalizes well to new data. We used the training set to train and optimize our machine learning model, while we keeped the test set until the very end to evaluate the final model. The environments used to perform the project were Python, Keras (Tensorflow). Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

For the models we will apply the following code packages:

Action	Packages used and explanations	Command

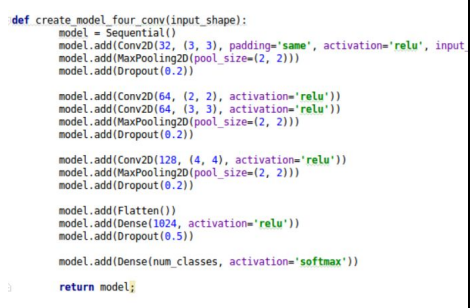
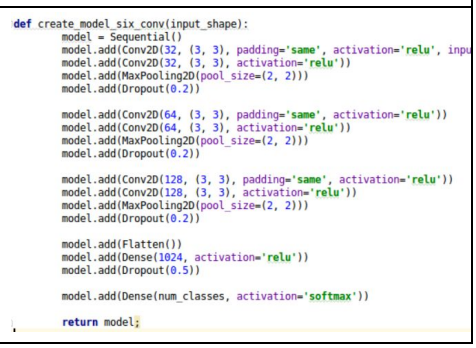
Import the Data	*CV2*
	Allows to read an OpenCV documentation
		`def load_pictures():`
		`  pics = []`
		`  labels = []`
		`  for k, v in map_characters.items(): # k : number v:characters`
		`  pictures = [k for k in glob.glob(imgsPath + "/" + v +`
		`  print v + " : " + str(len(pictures))`
		`  for i, pic in enumerate(pictures):`
		`  tmp_img = cv2.imread(pic)`
		`  tmp_img = cv2.cvtColor(tmp_img, cv2.COLOR_BGR`
		`  tmp_img = cv2.resize(tmp_img, (img_height, im`
		`  pics.append(tmp_img)`
		`  labels.append(k)`
		`  return np.array(pics), np.array(labels)`

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
		`def load_pictures():`
		`  pics = []`
		`  labels = []`
		`  for k, v in map_characters.items(): # k : number v:characters`
		`  pictures = [k for k in glob.glob(imgsPath + "/" + v +`
		`  print v + " : " + str(len(pictures))`
		`  for i, pic in enumerate(pictures):`
		`  tmp_img = cv2.imread(pic)`
		`  tmp_img = cv2.cvtColor(tmp_img, cv2.COLOR_BGR`
		`  tmp_img = cv2.resize(tmp_img, (img_height, im`
		`  pics.append(tmp_img)`
		`  labels.append(k)`
		`  return np.array(pics), np.array(labels)`

Calculate time to run	*Time*
	This package allows us to calculate the time used to run the commands.
		`with tf.device('/gpu:0'):`
		`  start_time=time.time()`
		`  history = model.fit(X_train, y_train,`
		`  batch_size=batch_size,`
		`  epochs=epochs,`
		`  validation_data=(X_test, y_test),`
		`  shuffle=True,`
		`  callbacks=[LearningRateScheduler(lr_schedule),`
		`  ModelCheckpoint('model.h5',save_best_only= True)])`
		`end_time=time.time()`
		`score = model.evaluate(X_test, y_test, verbose=0)`
		`print('Test loss:', score[0])`
		`print('Test accuracy:', score[1])`
		`print "Elapsed Time" + ":" + str(round((start_time - end_time)/60, 2))`
		`import matplotlib.pyplot as plt`

Split the dataset	*sklearn train_test_split*
	Split arrays or matrices into random train and test subsets
		`def get_dataset(save=False, load=False):`
		`  X, y = load_pictures()`
		`  y = pd.get_dummies(y)`
		`  y=y.values`
		`  X_train, X_test, y_train, y_test = train_test_split(X, y, test`
		`  X_train = X_train.astype('float32') / 255.`
		`  X_test = X_test.astype('float32') / 255.`
		`  print "Train", X_train.shape, y_train.shape`
		`  print "Test", X_test.shape, y_test.shape`
		`  return X_train, X_test, y_train, y_test`

Apply the Multilayer Neural Network	*Keras*
	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.
		`# 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=(52`
		`  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'])`

Applying the Convolutional Network 4-layer CNN	*Keras**	
Applying the Convolutional Network 6-layer CNN	*Keras**	

The models specifications are:

- **Convolution Neural Network.** We will use a feed forward 4 convolutional layers with ReLU activation followed by a fully connected hidden layer dropout layers to regularize and avoid overfitting. The output layer uses softmax activation to output the probability for each class.

- **Multilayer Neural Network.** We started with using six-layer (i.e. six-hidden-layer) feed forward neural network with 512 neurons in each hidden layer and used Relu as our activation function.We will experiment with different number of batch size and epochs and compared the performance in terms of accuracy, loss and computing time.

For the model as we said we will use mini batches. To be specific, we will start with 32 and after this we will increase it to see which one is better. In terms of the training parameters, we start with the learning rate in 0.01 and see if we need to change it. To prevent overfitting we use the dropout layer with the dropout layer rate 0.2. Also, for the explanation we use the validation dataset.

5. Results

5.1 Data preprocessing:

There is no doubt that data used during training plays the most important role. Since our data is unstructured data, in the format of image, fetching the right type of data which matches the exact-case of our experiment is a critical task.

1. Resizing the image

The first step for preprocessing picture is resizing them as we need to have all pictures with the same size for training. We set our training image as 42 x 42 x 3. In addition, in order to save some memory, we converted data as *float32* and normalize them by dividing by 255.

2. Convert the categories to vectors

In order to processing target image data, instead of characters name, we need to convert all label names to vectors. Thanks to Keras, we can quickly convert those categories to vectors with to_categorical() function.

3. Splitting the training data into training and validation sets

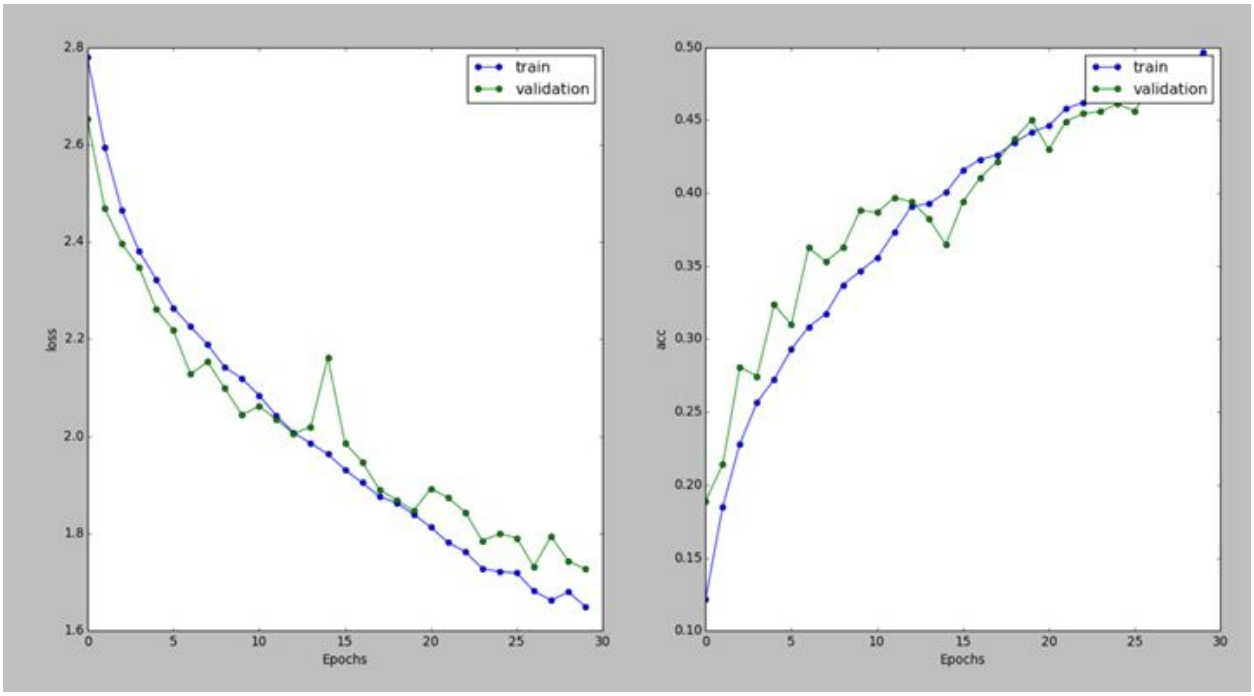
Besides original train and test set, validation dataset is needed as an estimate of model skill before making final predictions. Validation dataset can function as providing an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. In our case, we set 15% of training data as our validation set and 85% of training data used to fit the network.

5.2 Multi-layer Perceptron Neural Network:

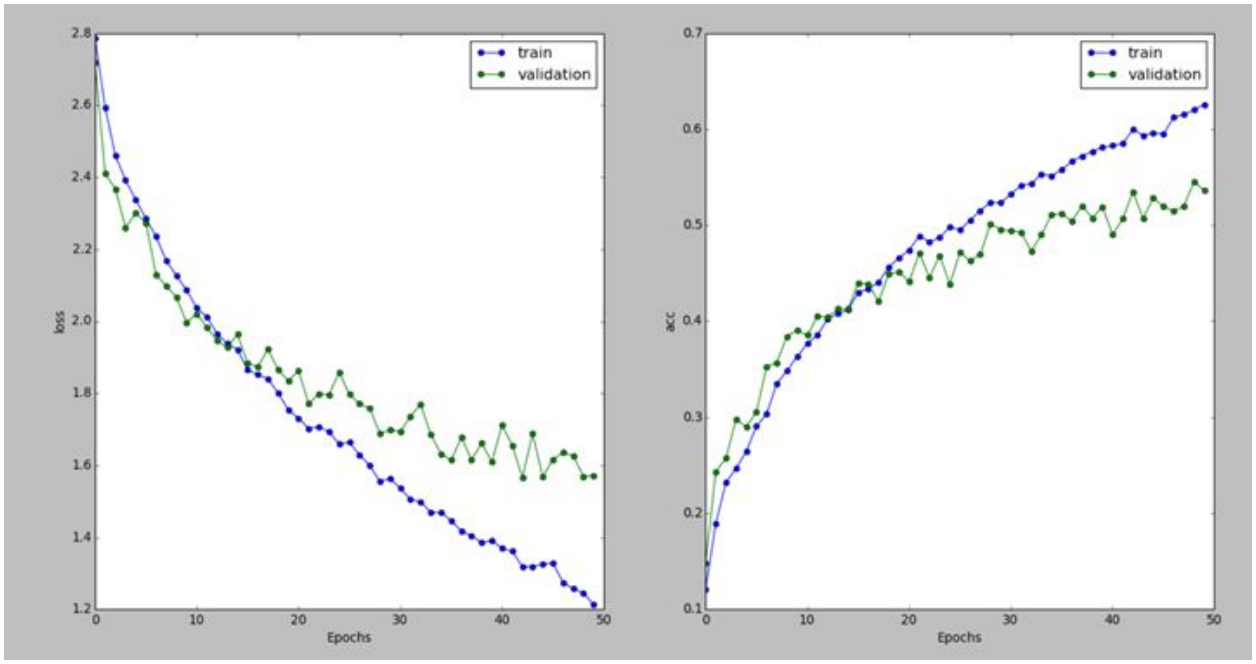
First of all, we employed multilayer perceptron to see the performance of MLP on image recognition. We started with using six-layer (i.e. six-hidden-layer) feed forward neural network with 512 neurons in each hidden layer and used Relu as our activation function.We experimented with different number of batch size and epochs and compared the performance in terms of accuracy, loss and computing time. Based on the table as below, it is apparent that MLP with 64 batch size perform relatively better compared to the others with 53.94 accuracy and 3.24 elapsed time.

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

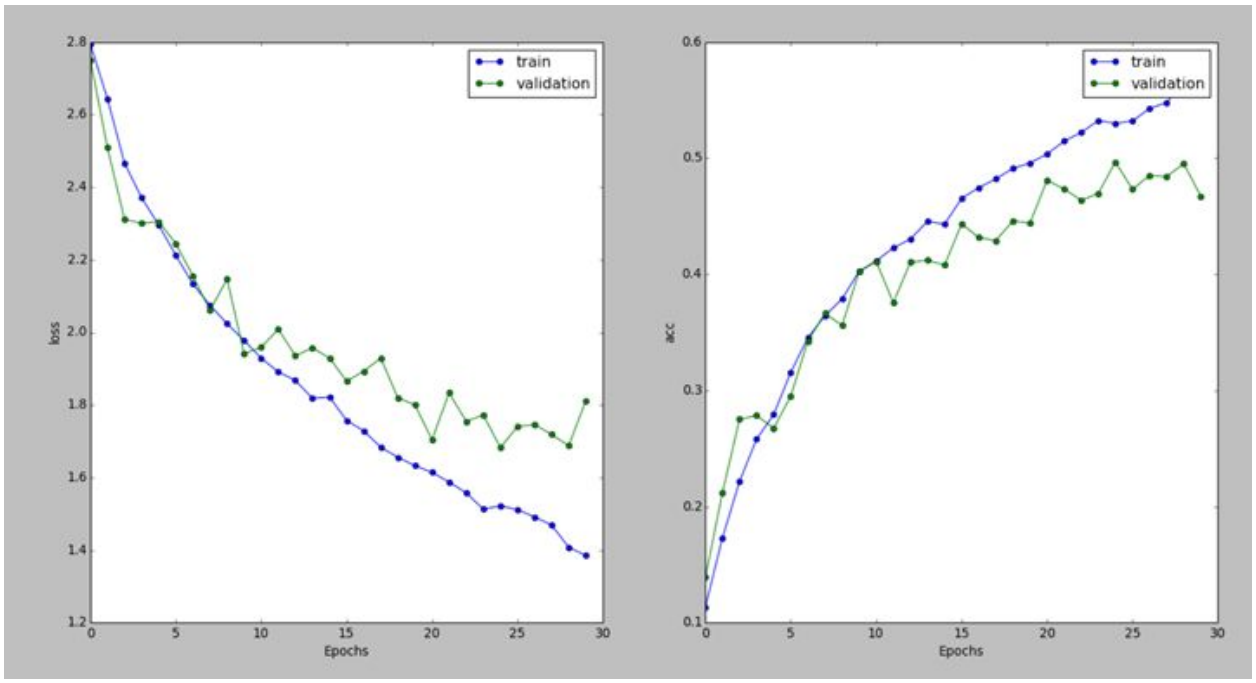
Since we used only the training set to compute gradients and determined weights updates , we need to compute the performance on the validation set at each epoch to see if overfitting issue exist.
For mini-batch size = 32, as we increased the number of epochs, the validation accuracy would ultimately converge to approximately 1.6 in loss and 50 accuracy.



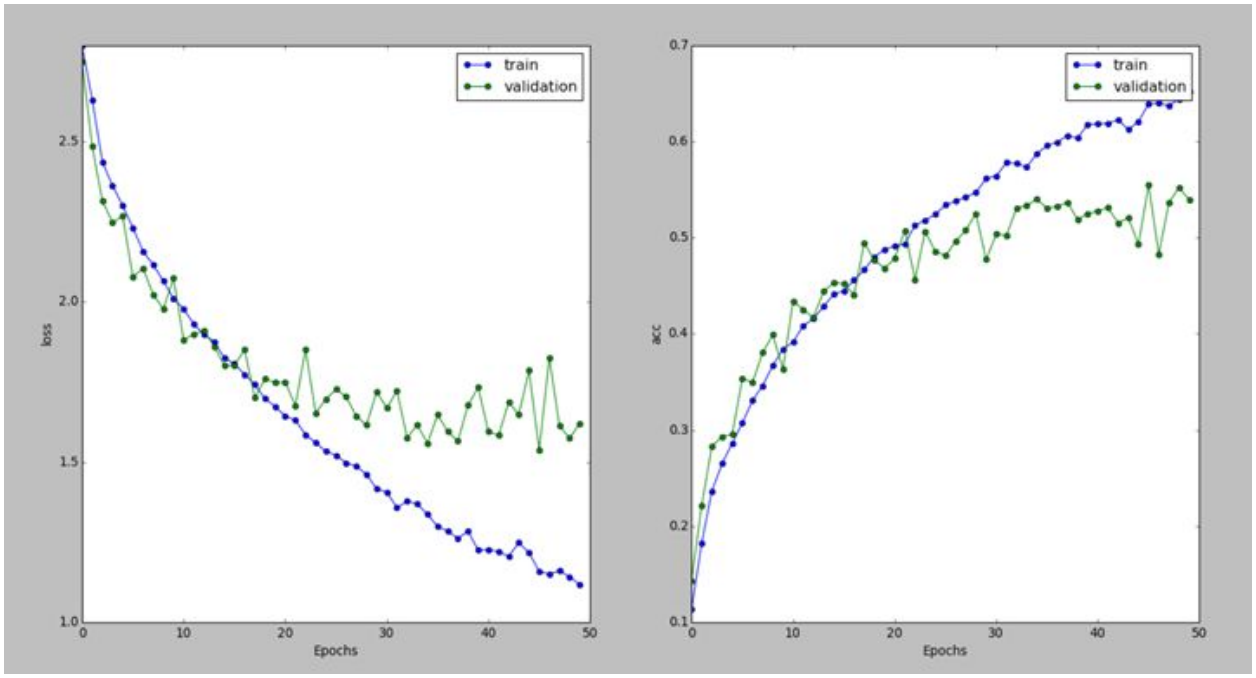
mini-batch size= 32 epochs=30



mini-batch size = 32 epochs =50

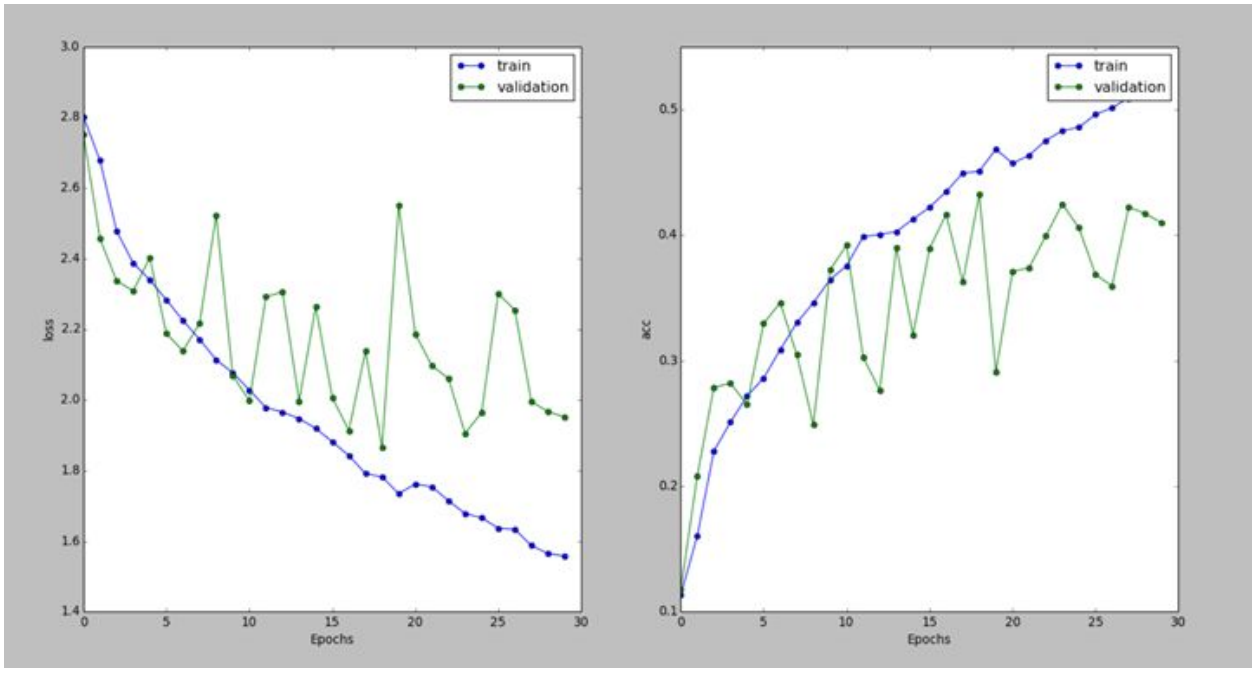


For mini-batch size = 64, the gap between these two line became wider. In addition, as we increased the size of batch, the validation loss and accuracy fluctuate more widely compared with 32 mini batch even though train accuracy and loss increased and decreased respectively.
mini-batch size= 64 epochs=30

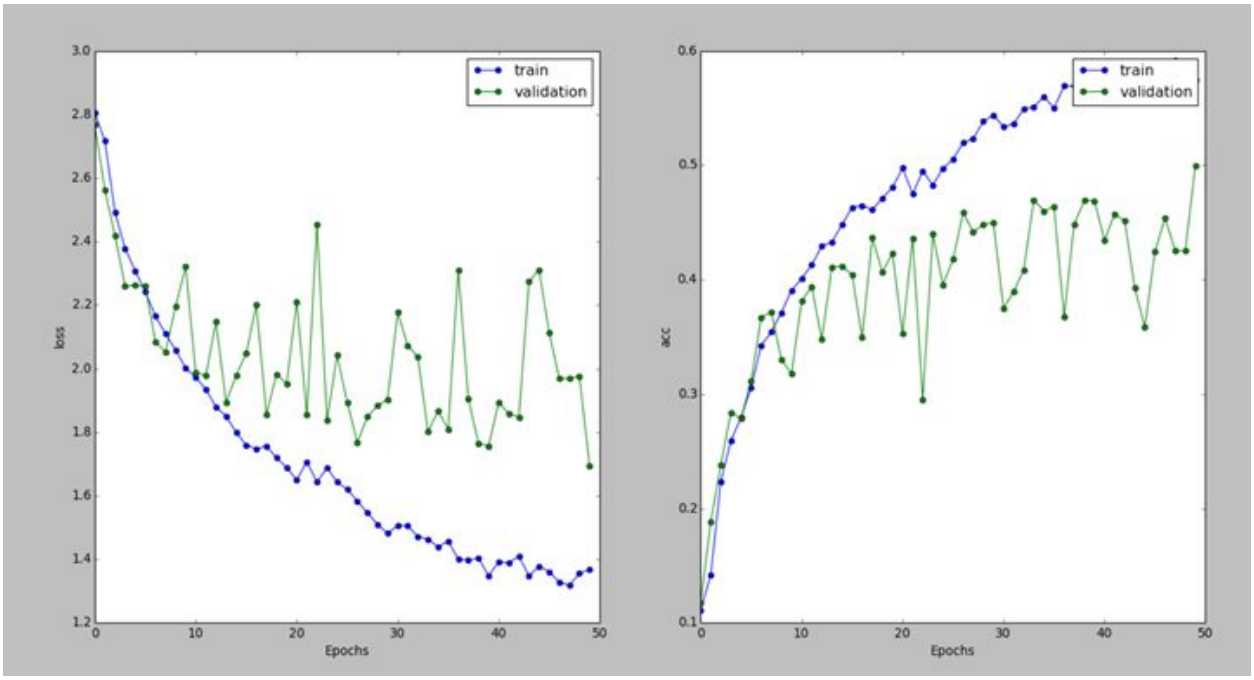


mini-batch size= 64 epochs=50

It is obvious that mini-batch size =128 had the most variable validation loss and accuracy and was unable to converge after epoch 10. There was also an overfitting issue on such network.



mini-batch size= 128 epochs=30



mini-batch size= 128 epochs=50

To sum up, we can conclude that MLP doesn't perform very well when it comes to recognizing our images. Therefore, in next part, we would use Convolutional Neural Network to deal with image detection.

5.3 Convolutional Neural Network

5.3.1 Five-layer CNN

We begun using a feed forwarded five convolutional layers with Relu activation function followed by a fully connected hidden layer. We also use dropout layers after pooling layers to regularize and prevent overfitting. the output layer uses softmax activation to generate the outputs for each class. We set kernel size to 2 X 2 and 4 X 4 in convolutional layer 2 and 4 respectively and the rest convolution layers have kernel size 3 X 3.

Categorical Cross Entropy loss is used for calculating the loss. For the optimizer, we use stochastic gradient descent and momentum metric.

In addition, we also included learning rate scheduler that can adjust the weight of learning rate and make it decay as the epoch increased. Based on the table, we can see the fully connected layer occupied most of the parameters, almost 85% (1,376,402 / 1,180,672) parameters were generated in the first fully connected layer

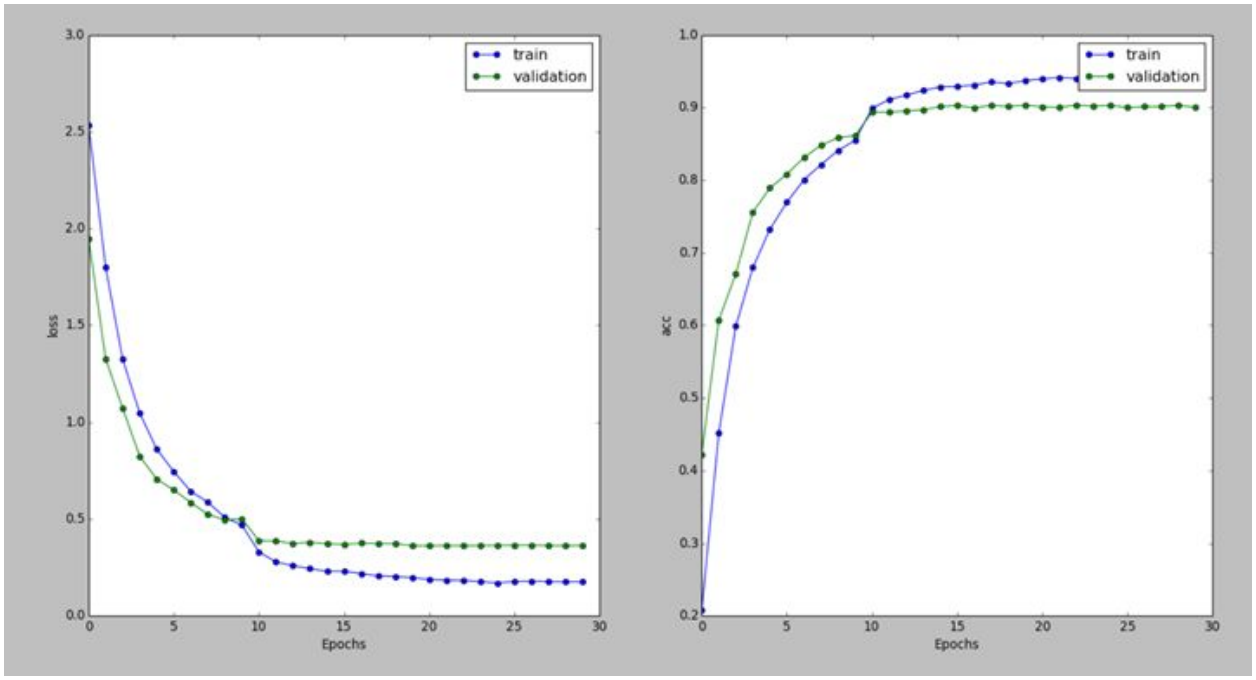
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 42, 42, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 21, 21, 32)	0
dropout_1 (Dropout)	(None, 21, 21, 32)	0
conv2d_2 (Conv2D)	(None, 20, 20, 64)	8256
conv2d_3 (Conv2D)	(None, 18, 18, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 9, 9, 64)	0
dropout_2 (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

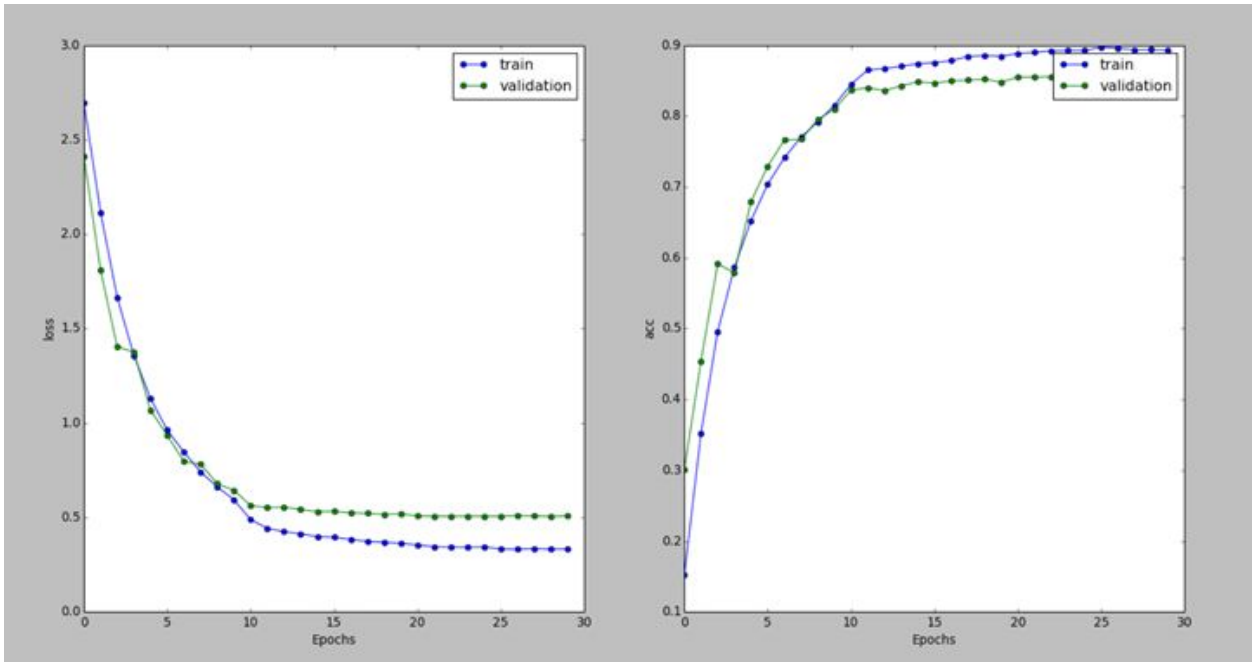
For the training, the model is iterating over batches of training set. We experimented with 32 and 64 batch-size for 30 epochs. Generally we need to increased the number of epochs and compared the difference. However, as we can see from the figures as below, the validation loss and accuracy begun to converge after 10 epochs and hence there is no need to increase the number of epoch.

We can conclude that Five-layer CNN had a significant improvement using regarding accuracy rate and there was no apparently overfitting issue based on the validation loss and accuracy figures.

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



mini-batch size= 32 epochs=30



mini-batch size= 32 epochs=30

5.3.2 Six-layer CNN

We further tried using a feed forwarded six convolutional layers with Relu activation followed by a fully connected hidden layer. We also use dropout layers after pooling layers to regularize and prevent overfitting. The output layer uses softmax activation to generate the outputs for each

class. The only different part is that all the kernel size is 3 X 3 in order to maintain the total parameters to roughly 1.3M generated by previous network so that we can make comparison between those two models.

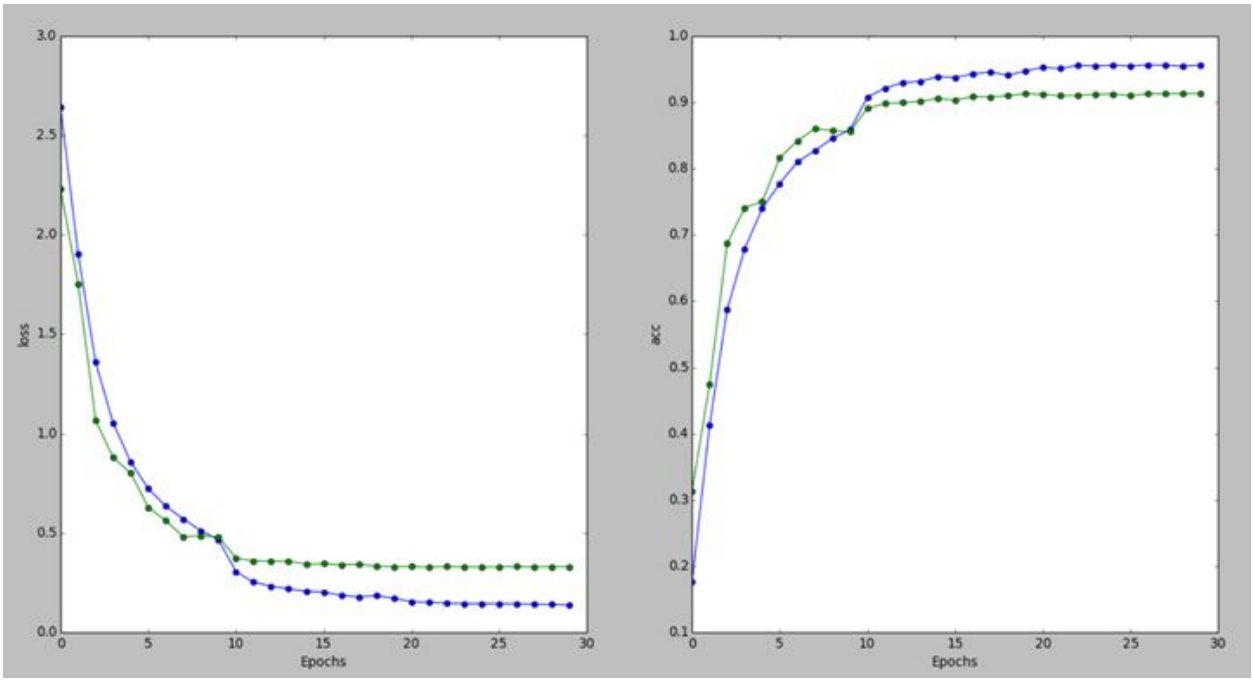
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 42, 42, 32)	896
conv2d_2 (Conv2D)	(None, 40, 40, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 20, 20, 32)	0
dropout_1 (Dropout)	(None, 20, 20, 32)	0
conv2d_3 (Conv2D)	(None, 20, 20, 64)	18496
conv2d_4 (Conv2D)	(None, 18, 18, 64)	36928
max_pooling2d_2 (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

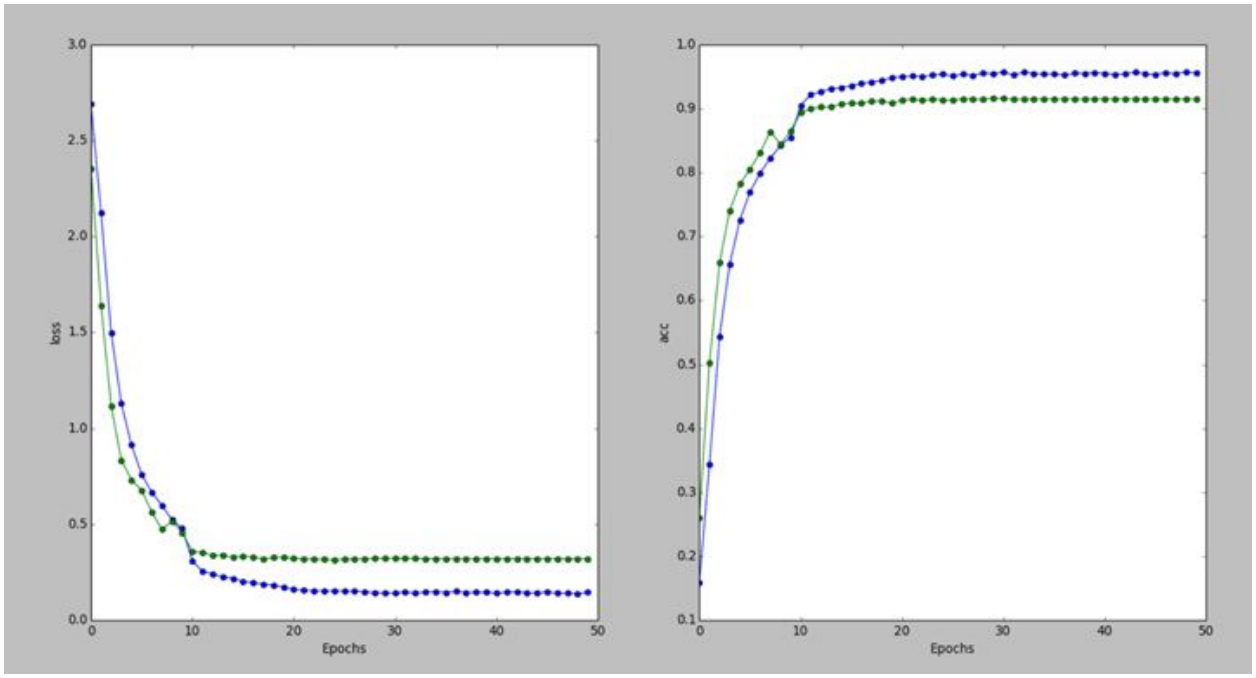
Based on the results, we can conclude that six-layer CNN with minibatch size =32 and epochs= 50 performed the best among other three network in terms of accuracy and loss. However, it took us more than 10 minutes to train such network. It seems that the network started to converge after epochs=10, which mean we are not able to increase the number of epoch for improving the model. Therefore, we went back to the data-preprocessing process and involved other data augmentation techniques, including setting width_shift_range to 0.1, height_shift_range to 0.1 and rotation_range to 10 , on the currently best network.

However, it is obvious that data augmentation mades huge improvement on the accuracy while increasing our computing time.

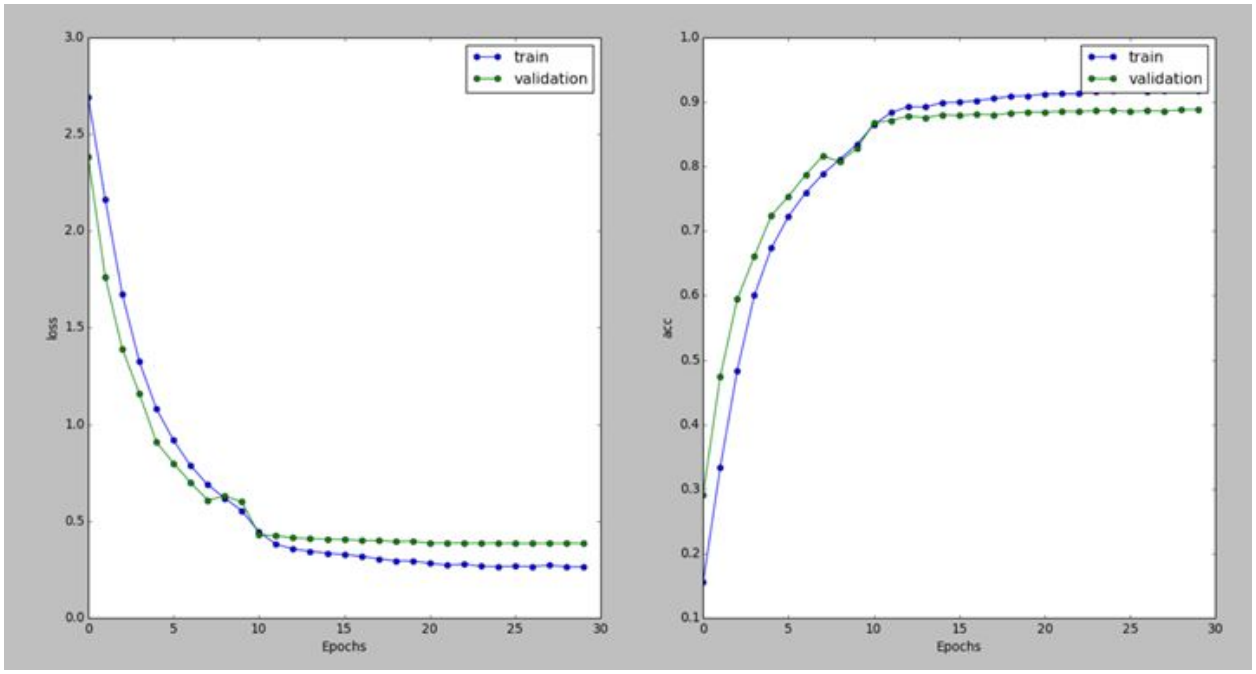
Batch-Size	Epochs	Accuracy	Loss	Elapsed Time
32	30	91.30%	0.3312	5.83 minutes
32	50	91.47%	0.3206	10.42 minutes
64	30	87.93%	0.4315	4.95 minutes
64	50	87.99%	0.4329	7.81 minutes
Data Augmentation 32	50	90.87%	0.3375	11.01minutes



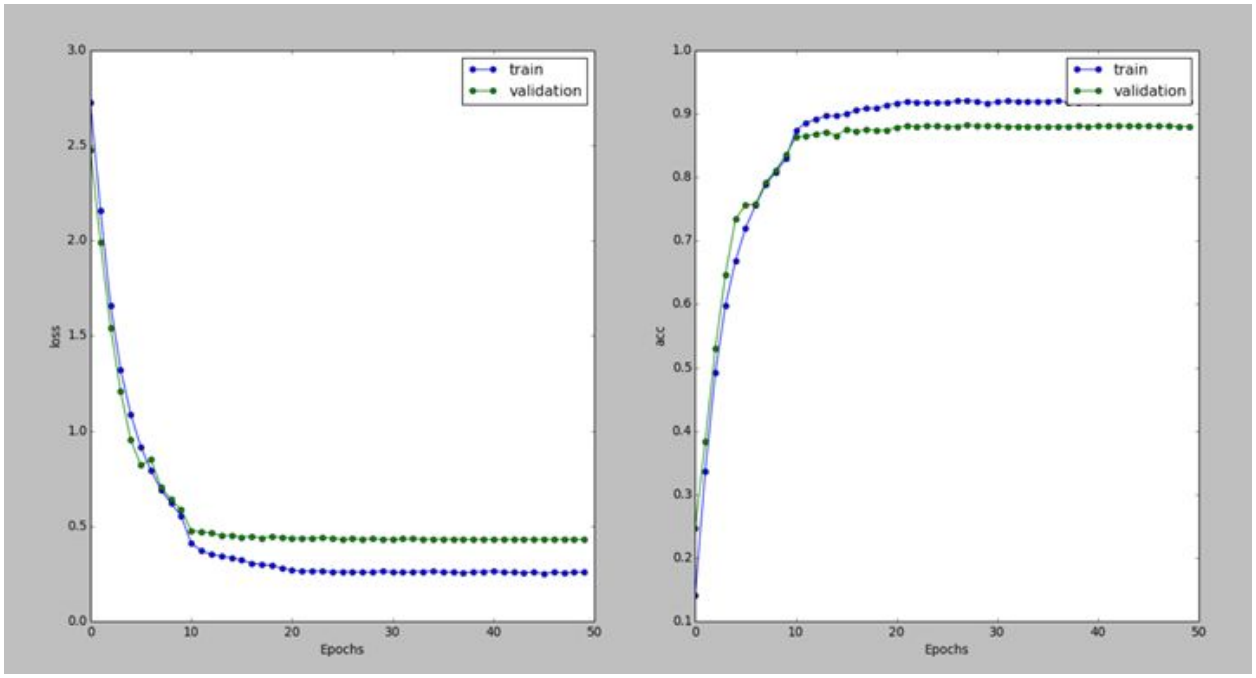
mini-batch size= 32 epochs=30



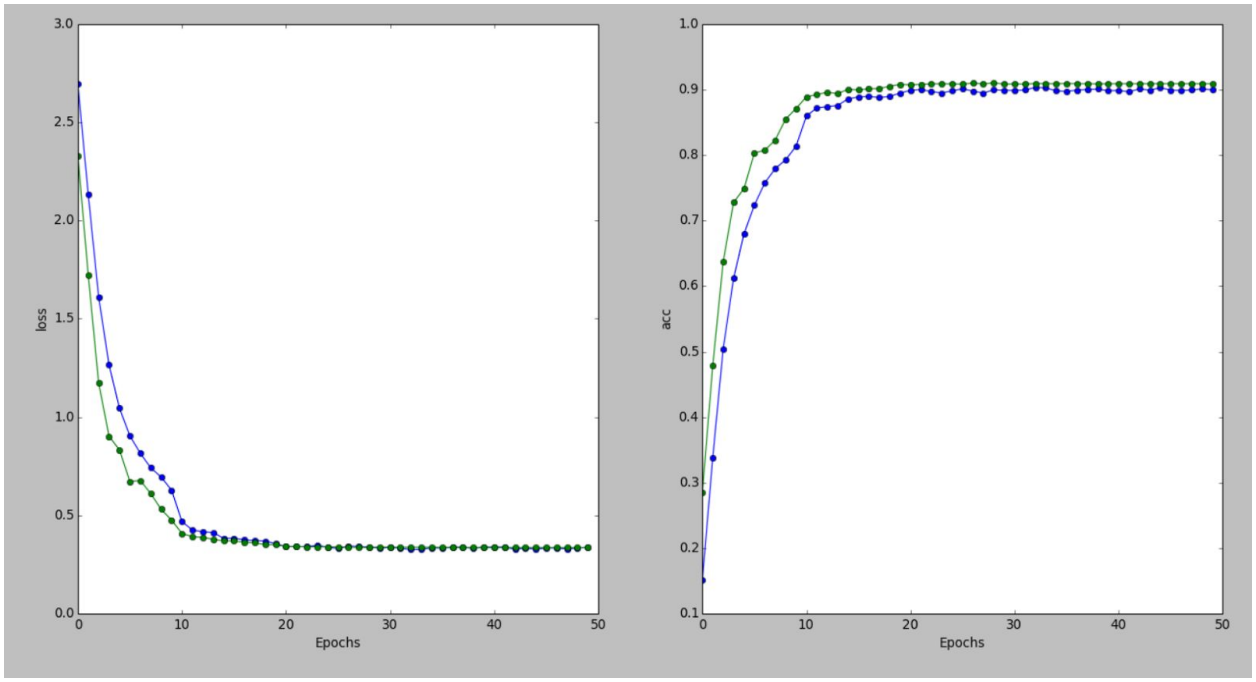
mini-batch size= 32 epochs=50



mini-batch size= 64 epochs=30



mini-batch size= 64 epochs=50



Data Augmentation mini-batch size= 64 epochs=50

5.4 Prediction and Precision

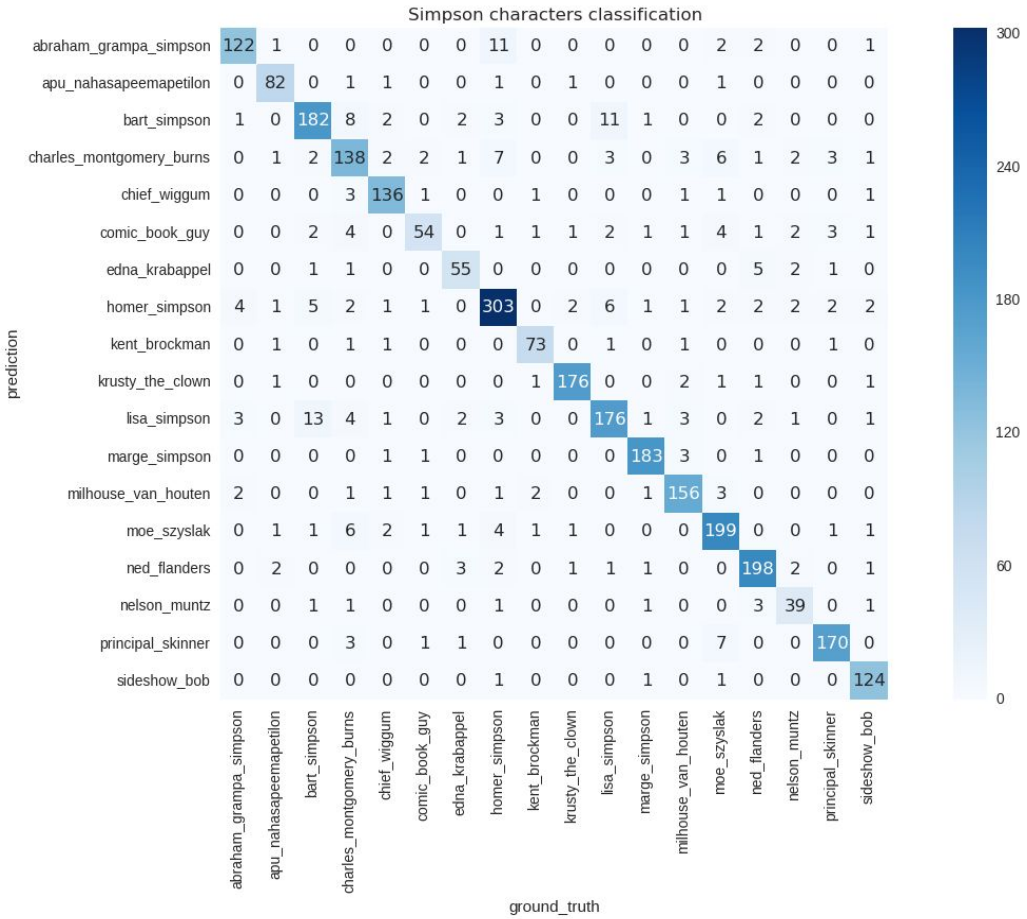
	precision	recall	f1-score	support
abraham_grampa_simpson	0.92	0.87	0.89	154
apu_nahasapeemapetilon	0.90	0.93	0.92	90
bart_simpson	0.88	0.87	0.87	187
charles_montgomery_burns	0.88	0.78	0.83	179
chief_wiggum	0.93	0.90	0.91	138
comic_book_guy	0.96	0.81	0.88	90
edna_krabappel	0.90	0.80	0.85	65
homer_simpson	0.90	0.90	0.90	344
kent_brockman	0.96	0.97	0.96	67
krusty_the_clown	0.88	0.97	0.92	175
lisa_simpson	0.85	0.88	0.87	199
marge_simpson	0.94	0.97	0.96	188
milhouse_van_houten	0.96	0.96	0.96	187
moe_szyslak	0.94	0.94	0.94	206
ned_flanders	0.92	0.96	0.94	235
nelson_muntz	0.89	0.78	0.83	60
principal_skinner	0.88	0.93	0.91	153
sideshow_bob	0.92	0.96	0.94	132
avg / total	0.91	0.91	0.91	2849

The precision reports shown above can provide a clearer scores for every simpson character in our test data. We can see an averagely 91% precision rate for all characters, and the rate can range from 88% to 96%.

- The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
- The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
- The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.
- The F-beta score weights recall more than precision by a factor of beta. beta == 1.0 means recall and precision are equally important.
- The support is the number of occurrences of each class.

We randomly chose 16 pictures from the test data and we feed to the model. We labeled each picture with its true class and the top three predictions our model produced.


Actual : Abraham
Sideshow : 82.6%
Principal : 4.6%
Comic : 2.4%
Actual : Apu
Ned : 17.5%
Apu : 36.7%
Sideshow : 18.3%
Actual : Bart
Marge : 53.1%
Ned : 10.9%
Homer : 8.3%
Actual : Charles
Krusty : 58.7%
Sideshow : 20.7%
Comic : 12.2%
Actual : Chief
Charles : 16.9%
Krusty : 15.0%
Sideshow : 14.6%
Actual : Comic
Bart : 66.7%
Homer : 14.6%
Charles : 3.3%
Actual : Edna
Krusty : 36.2%
Milhouse : 16.8%
Ned : 13.0%
Actual : Homer
Krusty : 38.6%
Apu : 27.7%
Ned : 17.2%
Actual : Kent
Krusty : 86.9%
Comic : 6.4%
Chief : 3.3%
Actual : Krusty
Marge : 48.5%
Milhouse : 17.6%
Apu : 8.6%
Actual : Lisa
Homer : 41.6%
Krusty : 19.4%
Ned : 12.9%
Actual : Marge
Homer : 36.0%
Ned : 19.4%
Marge : 10.9%
Actual : Milhouse
Milhouse : 33.1%
Homer : 24.1%
Marge : 19.7%
Actual : Moe
Ned : 15.1%
Principal : 15.0%
Charles : 14.7%
Actual : Ned
Principal : 51.1%
Edna : 9.1%
Comic : 7.8%
Actual : Nelson
Krusty : 36.5%
Homer : 12.9%
Charles : 11.2%
0
50
100
150
200
250
300
350
0 50 100 150 200 250 300 350


Simpson characters classification
prediction
ground_truth
abraham_grampa_simpson 122 1 0 0 0 0 0 11 0 0 0 0 0 2 2 0 0 1
apu_nahasapeemapetilon 0 82 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0
bart_simpson 1 0 182 8 2 0 2 3 0 0 11 1 0 0 2 0 0 0
charles_montgomery_burns 0 1 2 138 2 2 1 7 0 0 3 0 3 6 1 2 3 1
chief_wiggum 0 0 0 3 136 1 0 0 1 0 0 0 1 1 0 0 0 1
comic_book_guy 0 0 2 4 0 54 0 1 1 1 2 1 1 4 1 2 3 1
edna_krabappel 0 0 1 1 0 0 55 0 0 0 0 0 0 0 5 2 1 0
homer_simpson 4 1 5 2 1 1 0 303 0 2 6 1 1 2 2 2 2 2
kent_brockman 0 1 0 1 1 0 0 0 73 0 1 0 1 0 0 0 1 0
krusty_the_clown 0 1 0 0 0 0 0 0 1 176 0 0 2 1 1 0 0 1
lisa_simpson 3 0 13 4 1 0 2 3 0 0 176 1 3 0 2 1 0 1
marge_simpson 0 0 0 0 1 1 0 0 0 0 0 183 3 0 1 0 0 0
milhouse_van_houten 2 0 0 1 1 1 0 1 2 0 0 1 156 3 0 0 0 0
moe_szyslak 0 1 1 6 2 1 1 4 1 1 0 0 0 199 0 0 1 1
ned_flanders 0 2 0 0 0 0 3 2 0 1 1 1 0 0 198 2 0 1
nelson_muntz 0 0 1 1 0 0 0 1 0 0 0 1 0 0 3 39 0 1
principal_skinner 0 0 0 3 0 1 1 0 0 0 0 0 0 7 0 0 170 0
sideshow_bob 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 124
abraham_grampa_simpson
apu_nahasapeemapetilon
bart_simpson
charles_montgomery_burns
chief_wiggum
comic_book_guy
edna_krabappel
homer_simpson
kent_brockman
krusty_the_clown
lisa_simpson
marge_simpson
milhouse_van_houten
moe_szyslak
ned_flanders
nelson_muntz
principal_skinner
sideshow_bob
300
240
180
120
60
0

The classification heatmap shown above display correlation rate between pairs of characters. We can spot the high correlation between characters in the Simpson's family, father and childrens to be specific, which is understandable since families look alike. Additionally, Bart Simpson and Lisa Simpson have high correlation perhaps because they always appear together in our data pictures, which we want to have further clean of the data in favor of a better model and predictions.

For the predictions we applied the following packages:

Action	Packages used and explanations	Command

Precision table	*keras.models*
	Load model
	sklearn
	Generate metrics classification report
		`from keras.models import load_model`
		`import sklearn`
		
		`model2 = load_model('/home/wandingz/Deep-Learning/simpson/weights.hdf5')`
		`y_pred = model2.predict(X_test)`
		`acc = np.sum(y_pred==np.argmax(y_test, axis=1))/np.size(y_pred)`
		`print("Test accuracy = {}".format(acc))`
		
		`print u'''.join([unicode(u'\n'), unicode(sklearn.metrics.classification_report(np.where(y_test > 0)[1], np.argmax(y_p`

Test prediction lookup	*mpl_toolkits.axes_grid1*
	Generate grid
		`from mpl_toolkits.axes_grid1 import AxesGrid`
		
		`F = plt.figure(1, (15, 20))`
		`grid = AxesGrid(F, 111, nrows_ncols=(4, 4), axes_pad=0, label_mode=u"1")`
		`for i in xrange(16):`
		`  char = map_characters[i]`
		`  image = cv2.imread(np.random.choice([k for k in glob.glob(u'data/test/*.*') if char in k]))`
		`  img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)`
		`  pic = cv2.resize(image, (42, 42)).astype(u'float32') / 255.`
		
		`  a = model2.predict(pic.reshape(1, 42, 42, 3))[0]`
		`  actual = char.split(u'_')[0].title()`
		`  text = sorted([`
		`  [u'{:s} : {:.1f}%'.format(map_characters[k].split(u'_')[0].title(), 100 * v) for k, v in enumerate(a)]`
		`  key=lambda x: float(x.split(u':')[1].split(u'%')[0]), reverse=True)[:3]`
		`  img = cv2.resize(img, (352, 352))`
		`  cv2.rectangle(img, (0, 260), (215, 352), (255, 255, 255), -1)`
		`  font = cv2.FONT_HERSHEY_SIMPLEX`
		`  cv2.putText(img, u'Actual : %s' % actual, (10, 280), font, 0.7, (0, 0, 0), 2, cv2.CV_AA)`
		`  for k, t in enumerate(text):`
		`  cv2.putText(img, t, (10, 300 + k * 18), font, 0.65, (0, 0, 0), 2, cv2.CV_AA)`
		`  grid[i].imshow(img)`
		
		`plt.show()`

Classification map	*seaborn*
	Generate heatmap
	sklearn.metrics
	Generate confusion matrix
		`import seaborn as sns; sns.set()`
		`import pandas as pd`
		`from sklearn.metrics import confusion_matrix`
		
		`conf_mat = confusion_matrix(np.where(y_test > 0)[1], np.argmax(y_pred, axis=1))`
		`classes = list(map_characters.values())`
		`df = pd.DataFrame(conf_mat, index=classes, columns=classes)`
		
		`fig = plt.figure(figsize = (10,10))`
		`sns.heatmap(df, annot=True, square=True, fmt=u'.0f', cmap=u"Blues")`
		`plt.title(u'Simpson characters classification')`
		`plt.xlabel(u'ground_truth')`
		`plt.ylabel(u'prediction')`
		
		`plt.show()`

6. Summary and Conclusions

The Convolutional Neural Network has a better result than MLP. (Accuracy : 50% Vs. 91%).The local computer allows us to run the model but in a very slow speed. (50 mins vs. 5 mins). We think that if we increase the inputs the results will be better. Also, we want to apply this model to another dataset created by us in the future. We think that in the future more models can be applied to the dataset. This work was too important for us because, we learned about how to apply the algorithms to the real world dataset to classify images.

References

1. A. (2015, February 10). R: a language and environment for statistical computing. Retrieved May 01, 2017, from http://www.gbif.org/resource/81287

2. Boser, B.E., Guyon, I.M., & Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. Proceeding sof the fifth annual workshop on Computational learning theory - COLT '92.doi:10.1145/130385.130401

3. T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag, NY, USA, 2001.

4. P. Cortez. RMiner: Data Mining with Neural Networks and Support Vector Machines using R. In R. Rajesh (Ed.), Introduction to Advanced Scientific Softwares and Toolboxes, In press.

5. Hagan, M. T., Demuth, H. B., Beale, M. H., & Jesús, O. D. (2016). Neural network design. S. l.: S. n.

6.Raschka, S. (2016). Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics. Birmingham: Packt Publishing.

7. Keras Documentation. https://keras.io/

8.A.(2017,November)
https://www.kaggle.com/wcukierski/the-simpsons-by-the-data?sortBy=null&group=competition