The Simpsons Recognition Application

Fazliddin Naimov

Daniel Fischer

Peter Kim

Kevin Freehill

Dahmane Skendraoui

Muhammad Habib

Deep Learning: Training a convolutional neural network to recognize The Simpsons characters

The approach used to develop this application is based on the convolutional neural networks (CNNs): A multi-layered feed forward neural networking module which is able to learn various types of features and behaviors through training the model.

Process

Training the Model

Classification Evaluation

Improving the CNN

Visualizing Predicted Character

Flask App & HTML/CSS

Technology stack used:

- **Python**
- **HTML/CSS**
- **Bootstrap/Javascript**
- **Keras**
- **Tensorflow**
- **CNN**
- **Flask**

Process

Convolutional Neural Network (CNNs)

The Simpsons dataset was retrieved from Kaggle which provided data on 40+ Simpsons characters and pictures. For training the model, we only selected characters which had more than 290 pictures in the dataset.

We used a feed forward network with 4 convolutional layers and with a ReLU activation set followed by a fully connected hidden layer. The model iterated batches of training sets (batch size : 32) for 200 epochs. We also used data augmentation which calculated a number of random variations on the pictures so the trained model would never see the same picture twice. This helped prevent overfitting and helped train the model to generalize the data better.

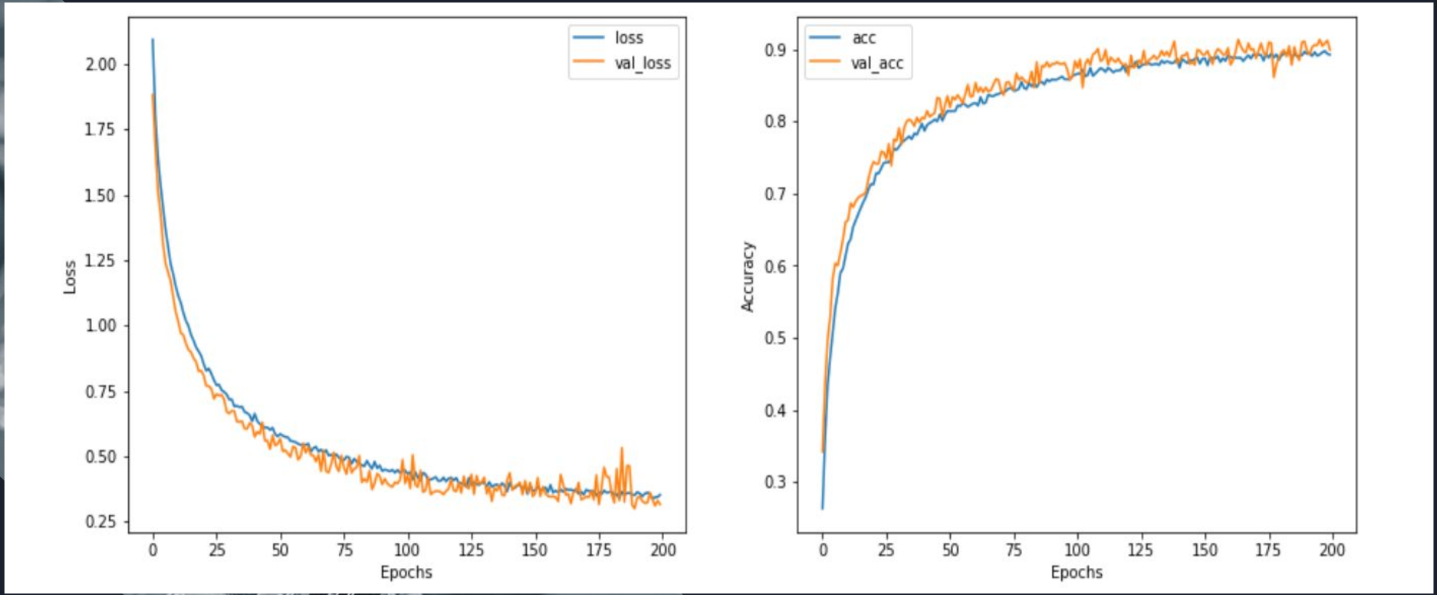
Training the Model

Splitting the data to Train and Test using get dataset function from train.py.

```
imp.reload(train)
X_train, X_test, y_train, y_test = train.get_dataset(save=True)
```

```
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
 samplewise_std_normalization=False, # divide each input by its std
 rotation_range=0, # randomly rotate images in the range
 width_shift_range=0.1, # randomly shift images horizontally
 height_shift_range=0.1, # randomly shift images vertically
 horizontal_flip=True, # randomly flip images
 vertical_flip=False) # randomly flip images
```

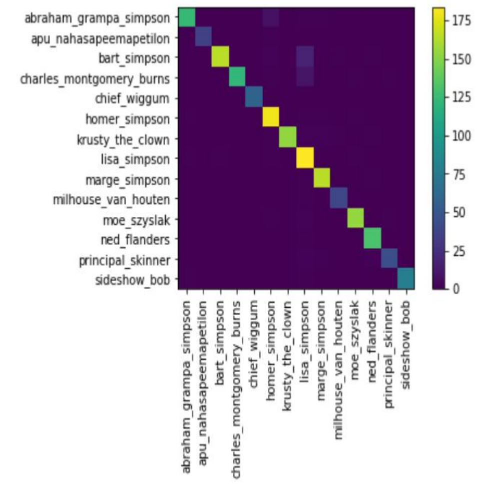
Loss and Accuracy during training



The accuracy (f1-sport) worked very well for us while training the model to recognize The Simpsons characters. The output was above 90 % correct for every character except for Lisa Simpson. The precision for Lisa was 82%.

One assumption we had was that Lisa Simpsons data could be mixed up with other Simpsons characters data making the output a little skewed.

	precision	recall	f1-score	support
abraham_grampa_simpson	0.97	0.88	0.92	159
apu_nahasapeemapetilon	0.97	0.93	0.95	82
bart_simpson	0.85	0.86	0.85	186
charles_montgomery_burns	0.90	0.88	0.89	190
chief_wiggum	0.96	0.92	0.94	146
comic_book_guy	0.94	0.74	0.83	68
edna_krabappel	1.00	0.85	0.92	53
homer_simpson	0.83	0.85	0.84	185
kent_brockman	0.93	0.90	0.92	61
krusty_the_clown	0.95	0.98	0.96	166
lisa_simpson	0.72	0.86	0.78	153
marge_simpson	0.95	0.93	0.94	179
milhouse_van_houten	0.91	0.90	0.91	114
moe_szyslak	0.93	0.88	0.90	162
ned_flanders	0.93	0.96	0.94	181
nelson_muntz	0.85	0.74	0.79	46
principal_skinner	0.80	0.94	0.87	150
sideshow_bob	0.95	0.97	0.96	133
avg / total	0.90	0.90	0.90	2414



Improving the CNN model

To train the Neural Network to understand more details, complexities and specific behaviors, we need to get deeper into the program and add more convolutional layers to the model. We improved the model with a total of 6 convolutional layers (dimensions of the output spaces were 32, 64, 512 vs 32, 64, 256, 1024) . It improved the accuracy, precision and recall of the trained data as depicted in the graph below The lower precision is 0.89 for Nelson Muntz since we only had 300 training examples for this character. Moreover, this model can converge quicker than our previous model: 40 epochs vs 200

	precision	recall	f1-score	support
abraham_grampa_simpson	0.97	0.93	0.95	120
apu_nahasapeemapetilon	0.99	0.99	0.99	80
bart_simpson	0.94	0.93	0.93	174
charles_montgomery_burns	0.96	0.92	0.94	193
chief_wiggum	0.99	0.97	0.98	145
comic_book_guy	0.95	0.92	0.93	77
edna_krabappel	0.94	0.90	0.92	73
homer_simpson	0.91	0.96	0.93	173
kent_brockman	0.95	0.93	0.94	76
krusty_the_clown	0.99	0.98	0.98	190
lisa_simpson	0.93	0.93	0.93	176
marge_simpson	1.00	0.97	0.98	185
milhouse_van_houten	0.96	1.00	0.98	152
moe_szyslak	0.92	0.93	0.92	166
ned_flanders	0.98	0.98	0.98	173
nelson_muntz	0.89	0.96	0.93	53
principal_skinner	0.94	0.99	0.96	164
sideshow_bob	1.00	1.00	1.00	140
avg / total	0.96	0.96	0.96	2510

Visualizing Predicted Characters

Created two functions which used the trained model to predict the image and URL

```
def file_predict(image_path, all_perc=False):
    image = cv2.imread(image_path)
    img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.show()
    pic = cv2.resize(image, (64,64))
    a = model.predict_proba(pic.reshape(1, 64, 64,3))[0]
    if all_perc:
        print('\n'.join(['{} : {}%'.format(map_characters[i], round(k*100)) for i,k
in sorted(enumerate(a), key=lambda x:x[1], reverse=True)]))
    else:
        return map_characters[np.argmax(a)].replace('_',' ').title()
def url_predict(url, all_perc=False):
    image = url_to_image(url)
    img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.show()
    pic = cv2.resize(image, (64,64))
    a = model.predict_proba(pic.reshape(1, 64, 64,3))[0]
    if all_perc:
        print('\n'.join(['{} : {}%'.format(map_characters[i], round(k*100)) for i,k
in sorted(enumerate(a), key=lambda x:x[1], reverse=True)]))
    else:
        return map_characters[np.argmax(a)].replace('_',' ').title()
```

```
image_path = os.path.join(".", "characters","krusty_the_clown","pic_0019.jpg")
file_predict(image_path)
```

```
'Krusty The Clown'
```

```
url = "https://deadhomersociety.files.wordpress.com/2011/06/amilhousedivided6.png"
url_predict(url)
```

```
'Lisa Simpson'
```

Flask app

The Flask app connected the python server to JavaScript. Test Simpsons image were converted to array values which were then passed to JavaScript.

@app.route("/") def index(): return render_template("index.html")

@app.route('/predict', methods=['GET', 'POST']) def predict():

HTML/CSS/JS

JavaScript converted the arrays to base64 strings for transport to the server for the prediction output. The result was then passed through JavaScript then to HTML

```
// Predict
function getPrediction() {

    var imageInput = $('#imagePreview').attr('style').split(",")[1];

    var base64ImageData = imageInput.substring(0,imageInput.length-3);

    $(this).hide();
    $('.loader').show();

    fetch("/predict",{
        method: "POST",
        body: JSON.stringify({image:base64ImageData}),
        headers: {
            'Content-Type': 'application/json'
            // 'Content-Type': 'application/x-www-form-urlencoded',
        },
    }
```


THE
END
IS
NEAR


The Simpsons
Guess Who?
Choose An Image...
Predict!
AND THE PREDICTION IS ...
Krusty The Clown


THANK YOU
memegenerator.net