# Creating a Credit Card Number Dataset

### Step 1 - Exploring the Credit Card Font

**NOTE**: Download all code in resources

**Open the file titled: *`Credit Card Reader.ipynb`***

Unfortunately, there isn't an official standard credit card number font - some of the fonts used go by the names **Farrington 7B**, **OCR-B**, **SecurePay**, **OCR-A** and **MICR E13B.** However, in my experience there seem to be two main font variations used in credit cards:

And

Note the differences, especially in the 1, 7 and 8.

Let's open these images in Python using OpenCV

```
import cv2

cc1 = cv2.imread('creditcard_digits1.jpg', 0)
cv2.imshow("Digits 1", cc1)
cv2.waitKey(0)
cc2 = cv2.imread('creditcard_digits2.jpg', 0)
cv2.imshow("Digits 2", cc2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Let's experiment with testing OTSU Binarization. Remember binarization converts a grayscale image to two colors, black and white. Values under a certain threshold (typically 127  out of 255) are clipped to 0, while the values greater than 127 are clipped to 255. It looks like this below.

The code to perform this is:

```
cc1 = cv2.imread('creditcard_digits2.jpg', 0)
_, th2 = cv2.threshold(cc1, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
cv2.imshow("Digits 2 Thresholded", th2)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

### Step 2 - Creating our Dataset Directories

This sets up our training and test directories for the digits (0 to 9).

```
#Create our dataset directories

import os

def makedir(directory):
    """Creates a new directory if it does not exist"""
    if not os.path.exists(directory):
        os.makedirs(directory)
        return None, 0

for i in range(0,10):
    directory_name = "./credit_card/train/"+str(i)
    print(directory_name)
    makedir(directory_name)

for i in range(0,10):
    directory_name = "./credit_card/test/"+str(i)
    print(directory_name)
    makedir(directory_name)
```

### Step 3 - Creating our Data Augmentation Functions

Let's now create some functions to create more data. What we're doing here is taking the two samples of each digit we saw above, and adding small variations to the digit. This is very similar to Keras's Data Augmentation, however, we're using OpenCV to create an augmented dataset instead. We will further use Keras to Augment this even further.

We've created 5 functions here, let's discuss each:

- ***DigitAugmentation()*** - This one simply uses the other image manipulating functions, but calls them randomly. Examine to code to see how it's done.
- ***add_noise()*** **-** This function introduces some noise elements to the image
- ***pixelate()*** - This function re-sizes the image then upscales/upsamples it. This degrades the quality and is meant to simulate blur to the image from either a shakey or poor quality camera.
- ***stretch()*** - This simulates some variation in re-sizing where it stretches the image to a small random amount

- pre_process() - This is a simple function that applies OTSU Binarization to the image and re-sizes it. We use this on the extracted digits. To create a clean dataset akin to the MNIST style format.

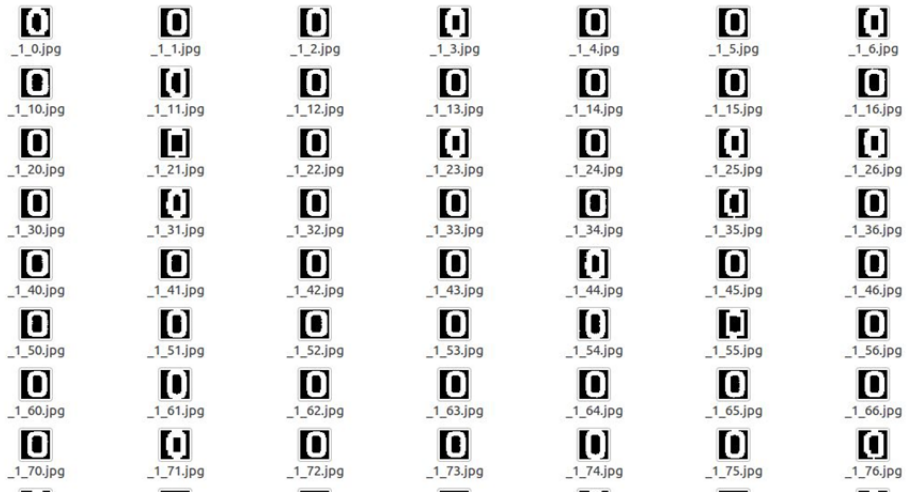
```
import cv2
import numpy as np
import random
import cv2
from scipy.ndimage import convolve

def DigitAugmentation(frame, dim = 32):
    """Randomly alters the image using noise, pixelation and streching image functions"
""
    frame = cv2.resize(frame, None, fx=2, fy=2, interpolation = cv2.INTER_CUBIC)
    frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)
    random_num = np.random.randint(0,9)

    if (random_num % 2 == 0):
        frame = add_noise(frame)
    if(random_num % 3 == 0):
        frame = pixelate(frame)
    if(random_num % 2 == 0):
        frame = stretch(frame)
    frame = cv2.resize(frame, (dim, dim), interpolation = cv2.INTER_AREA)

    return frame

def add_noise(image):
    """Addings noise to image"""
    prob = random.uniform(0.01, 0.05)
    rnd = np.random.rand(image.shape[0], image.shape[1])
    noisy = image.copy()
    noisy[rnd < prob] = 0
    noisy[rnd > 1 - prob] = 1
    return noisy

def pixelate(image):
    "Pixelates an image by reducing the resolution then upscaling it"
    dim = np.random.randint(8,12)
    image = cv2.resize(image, (dim, dim), interpolation = cv2.INTER_AREA)
    image = cv2.resize(image, (16, 16), interpolation = cv2.INTER_AREA)
    return image

def stretch(image):
    "Randomly applies different degrees of stretch to image"
    ran = np.random.randint(0,3)*2
    if np.random.randint(0,2) == 0:
        frame = cv2.resize(image, (32, ran+32), interpolation = cv2.INTER_AREA)
        return frame[int(ran/2):int(ran+32)-int(ran/2), 0:32]
    else:
        frame = cv2.resize(image, (ran+32, 32), interpolation = cv2.INTER_AREA)
        return frame[0:32, int(ran/2):int(ran+32)-int(ran/2)]

def pre_process(image, inv = False):
    """Uses OTSU binarization on an image"""
    try:
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    except:
        gray_image = image
        pass
```

```
56.
57.     if inv == False:
58.         _, th2 = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
59.     else:
60.         _, th2 = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
61.     resized = cv2.resize(th2, (32,32), interpolation = cv2.INTER_AREA)
62.     return resized
```

We can test our augmentation by using this bit of code:

```
1.  cc1 = cv2.imread('creditcard_digits2.jpg', 0)
2.  _, th2 = cv2.threshold(cc1, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
3.  cv2.imshow("cc1", th2)
4.  cv2.waitKey(0)
5.  cv2.destroyAllWindows()
6.
7.  # This is the coordinates of the region enclosing  the first digit
8.  # This is preset and was done manually based on this specific image
9.  region = [(0, 0), (35, 48)]
10.
11. # Assigns values to each region for ease of interpretation
12. top_left_y = region[0][1]
13. bottom_right_y = region[1][1]
14. top_left_x = region[0][0]
15. bottom_right_x = region[1][0]
16.
17. for i in range(0,1): #We only look at the first digit in testing out augmentation functions
18.     roi = cc1[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
19.     for j in range(0,10):
20.         roi2 = DigitAugmentation(roi)
21.         roi_otsu = pre_process(roi2, inv = False)
22.         cv2.imshow("otsu", roi_otsu)
23.         cv2.waitKey(0)
24.
25. cv2.destroyAllWindows()
```

Typically it looks like this:



You can try more adventurous forms of varying the original image. My suggestion would be to try dilation and erosion on these.

### Step 4 - Creating our dataset

Let's create 1000 variations of the first font we're sampling (note 1000 is perhaps way too much, but the data sizes were small and quick to train so why not use the arbitrary number of 1000).

```
# Creating 2000 Images for each digit in creditcard_digits1 - TRAINING DATA

# Load our first image
cc1 = cv2.imread('creditcard_digits1.jpg', 0)

_, th2 = cv2.threshold(cc1, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
cv2.imshow("cc1", th2)
cv2.imshow("creditcard_digits1", cc1)
cv2.waitKey(0)
cv2.destroyAllWindows()

region = [(2, 19), (50, 72)]

top_left_y = region[0][1]
bottom_right_y = region[1][1]
top_left_x = region[0][0]
bottom_right_x = region[1][0]

for i in range(0,10):
    # We jump the next digit each time we loop
    if i > 0:
        top_left_x = top_left_x + 59
        bottom_right_x = bottom_right_x + 59

    roi = cc1[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
    print("Augmenting Digit - ", str(i))
    # We create 200 versions of each image for our dataset
    for j in range(0,2000):
        roi2 = DigitAugmentation(roi)
        roi_otsu = pre_process(roi2, inv = True)
        cv2.imwrite("./credit_card/train/"+str(i)+"./_1_"+str(j)+".jpg", roi_otsu)
cv2.destroyAllWindows()
```

Next, let's make 1000 variations to each digit of the second font type.

```
# Creating 2000 Images for each digit in creditcard_digits2 - TRAINING DATA

cc1 = cv2.imread('creditcard_digits2.jpg', 0)
_, th2 = cv2.threshold(cc1, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
cv2.imshow("cc1", th2)
cv2.waitKey(0)
cv2.destroyAllWindows()

region = [(0, 0), (35, 48)]

top_left_y = region[0][1]
bottom_right_y = region[1][1]
top_left_x = region[0][0]
bottom_right_x = region[1][0]
```

```
15.
16. for i in range(0,10):
17.     if i > 0:
18.         # We jump the next digit each time we loop
19.         top_left_x = top_left_x + 35
20.         bottom_right_x = bottom_right_x + 35
21.
22.     roi = cc1[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
23.     print("Augmenting Digit - ", str(i))
24.     # We create 200 versions of each image for our dataset
25.     for j in range(0,2000):
26.         roi2 = DigitAugmentation(roi)
27.         roi_otsu = pre_process(roi2, inv = False)
28.         cv2.imwrite("./credit_card/train/"+str(i)+"./_2_"+str(j)+".jpg", roi_otsu)
29.         cv2.imshow("otsu", roi_otsu)
30.         print("-")
31.         cv2.waitKey(0)
32.
33. cv2.destroyAllWindows()
```

### Making our Test Data

- Note is a VERY bad practice to create a test dataset like this. Even though we're adding random variations, our test data here is too similar to our training data. Ideally, you'd want to use some real life unseen data from another source. In our case, we're sampling for the same dataset.

```
1.  # Creating 200 Images for each digit in creditcard_digits1 - TEST DATA
2.
3.  # Load our first image
4.  cc1 = cv2.imread('creditcard_digits1.jpg', 0)
5.
6.  _, th2 = cv2.threshold(cc1, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
7.  cv2.imshow("cc1", th2)
8.  cv2.imshow("creditcard_digits1", cc1)
9.  cv2.waitKey(0)
10. cv2.destroyAllWindows()
11.
12. region = [(2, 19), (50, 72)]
13.
14. top_left_y = region[0][1]
15. bottom_right_y = region[1][1]
16. top_left_x = region[0][0]
17. bottom_right_x = region[1][0]
18.
19. for i in range(0,10):
20.     # We jump the next digit each time we loop
21.     if i > 0:
22.         top_left_x = top_left_x + 59
23.         bottom_right_x = bottom_right_x + 59
24.
25.     roi = cc1[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
26.     print("Augmenting Digit - ", str(i))
27.     # We create 200 versions of each image for our dataset
28.     for j in range(0,2000):
29.         roi2 = DigitAugmentation(roi)
30.         roi_otsu = pre_process(roi2, inv = True)
```

```
        cv2.imwrite("./credit_card/test/"+str(i)+"./_1_"+str(j)+".jpg", roi_otsu)
cv2.destroyAllWindows()
# Creating 200 Images for each digit in creditcard_digits2 - TEST DATA

cc1 = cv2.imread('creditcard_digits2.jpg', 0)
_, th2 = cv2.threshold(cc1, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
cv2.imshow("cc1", th2)
cv2.waitKey(0)
cv2.destroyAllWindows()

region = [(0, 0), (35, 48)]

top_left_y = region[0][1]
bottom_right_y = region[1][1]
top_left_x = region[0][0]
bottom_right_x = region[1][0]

for i in range(0,10):
    if i > 0:
        # We jump the next digit each time we loop
        top_left_x = top_left_x + 35
        bottom_right_x = bottom_right_x + 35

    roi = cc1[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
    print("Augmenting Digit - ", str(i))
    # We create 200 versions of each image for our dataset
    for j in range(0,2000):
        roi2 = DigitAugmentation(roi)
        roi_otsu = pre_process(roi2, inv = False)
        cv2.imwrite("./credit_card/test/"+str(i)+"./_2_"+str(j)+".jpg", roi_otsu)
        cv2.imshow("otsu", roi_otsu)
        print("-")
        cv2.waitKey(0)
cv2.destroyAllWindows()
```