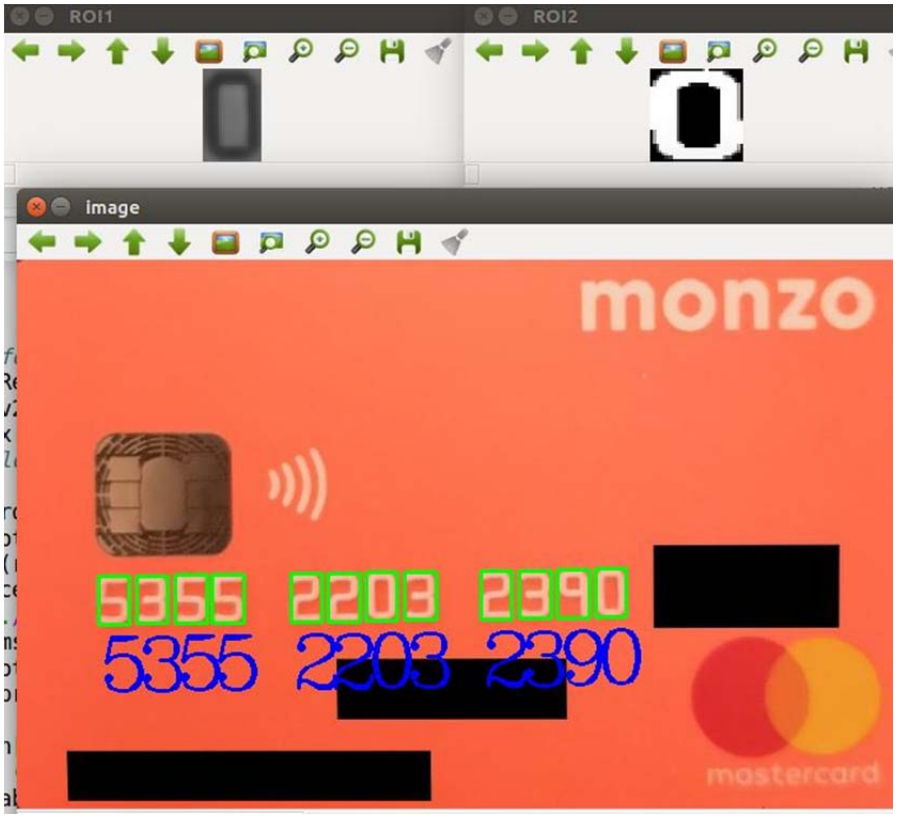
Classifying Digits on a Real Credit Card



In this section we're going to write the code to make the above!

Firstly, let's load the model we created by:

```
1. from keras.models import load_model
2. import keras
3.
4. classifier = load_model('/home/deeplearningcv/DeepLearningCV/Trained Models/creditcard.
   h5')
5. In this section we're going to write the code to make the above!
```

Now let's do some OpenCV magic and extract the digits from the image below:

The algorithm we follow to do this is really quite simple:

1. We first load our grayscale extracted image and the original color (note we could have just loaded the color and grayscaled it)
2. We apply the Canny Edge algorithm (typically we apply blur first to reduce noise in finding the edges).
3. We then use findCountours to isolate the digits
4. We sort the contours by size (so that smaller irrelevant contours aren't used)
5. We then sort it left to right by creating a function that returns the x-cordinate of a contour.
6. Once we have our cleaned up contours, we find the bounding rectange of the contour which gives us an enclosed rectangle around the digit. (To ensure these contours are valid we do extract only contours meeting the minimum width and height expectations). Also because I've created a black square around the last 4 digits, we discard contours of large area so that it isn't fed into our classifier.
7. We then take each extracted digit, use our pre_processing function (which applies OTSU Binarization and re-sizes it) then breakdown that image array so that it can be loaded into our classifier.

The full code to do this is shown below:

```
def x_cord_contour(contours):
    #Returns the X cordinate for the contour centroid
    if cv2.contourArea(contours) > 10:
        M = cv2.moments(contours)
        return (int(M['m10']/M['m00']))
    else:
        pass

img = cv2.imread('credit_card_extracted_digits.jpg')
orig_img = cv2.imread('credit_card_color.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2.imshow("image", img)
cv2.waitKey(0)

# Blur image then find edges using Canny
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
#cv2.imshow("blurred", blurred)
#cv2.waitKey(0)

edged = cv2.Canny(blurred, 30, 150)
#cv2.imshow("edged", edged)
#cv2.waitKey(0)

# Find Contours
_, contours, _ = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM
PLE)

#Sort out contours left to right by using their x cordinates
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:13] #Change this to 16
to get all digits
contours = sorted(contours, key = x_cord_contour, reverse = False)

# Create empty array to store entire number
full_number = []

# loop over the contours
```

```
for c in contours:
    # compute the bounding box for the rectangle
    (x, y, w, h) = cv2.boundingRect(c)
    if w >= 5 and h >= 25 and cv2.contourArea(c) < 1000:
        roi = blurred[y:y + h, x:x + w]
        #ret, roi = cv2.threshold(roi, 20, 255,cv2.THRESH_BINARY_INV)
        cv2.imshow("ROI1", roi)
        roi_otsu = pre_process(roi, True)
        cv2.imshow("ROI2", roi_otsu)
        roi_otsu = cv2.cvtColor(roi_otsu, cv2.COLOR_GRAY2RGB)
        roi_otsu = keras.preprocessing.image.img_to_array(roi_otsu)
        roi_otsu = roi_otsu * 1./255
        roi_otsu = np.expand_dims(roi_otsu, axis=0)
        image = np.vstack([roi_otsu])
        label = str(classifier.predict_classes(image, batch_size = 10))[1]
        print(label)
        (x, y, w, h) = (x+region[0][0], y+region[0][1], w, h)
        cv2.rectangle(orig_img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(orig_img, label, (x , y + 90), cv2.FONT_HERSHEY_COMPLEX, 2, (255, 0
, 0), 2)
        cv2.imshow("image", orig_img)
        cv2.waitKey(0)

cv2.destroyAllWindows()
```