digital whisper
Technologic Papers

תקיפת רשתות נוירונים

מאת רון אורבך

הקדמה

רשתות נוירונים מלאכותיות (Artificial Neural Network) הינן אוסף מודלים מתמטיים אשר משמשים מערכות-לומדות. ביחס לשיטות המקבילות הן די מוצלחות ומכך השימוש בהן נפוץ. ישנם תחומים רבים אשר להם אפליקציות הנשענות על רשתות נוירונים: עיבוד שפה טבעית (כגון תרגום, הבנת משפטים ומבנים תחביריים), ראייה ממוחשבת (למשל זיהוי אובייקטים מתמונה, זיהוי פנים, ואף נהיגה אוטונומית) וגם אבטחת מידע (כדוגמת זיהוי התנהגות רשתית חריגה או זיהוי וירוסים שאינם חתומים).

במאמר זה נציג בקצרה את רשתות הנוירונים בראי המערכות-הלומדות המיועדות לזיהוי העצם בתמונה, ולאחר מכן נממש 'תקיפה' עליהן. במסגרת המימוש, ניקח תמונה אשר רשת הנוירונים המאומנת תזהה בהצלחה כ-"משאית", ונערוך את התמונה של המשאית באופן חכם שכמעט ולא נראה לעין בלתי מזוינת - כך שנגרום לרשת הנוירונים להצהיר בביטחון כי מדובר בתמונה של "טוסטר".

יש לציין, שמומלץ מאוד לקרוא את המאמר ובמקביל לנסות ולממש על דוגמא משלכם. כל התוכנות שנשתמש בהן הינן חופשיות לשימוש (קוד-פתוח). אמנם, מימושים העוסקים ברשתות נוירונים לרוב דורשים זמן ויכולות עיבוד חזקות בשל הצורך באימונן, אך זה אינו המקרה שלנו - הרצה של התקיפה עצמה תארך פחות מדקה על מחשב נייד ממוצע.

מערכות-לומדות

כמו על הרבה נושאים שיופיעו במאמר, תחום מדעי זה הינו עשיר מאוד ואפשר היה לרשום רק עליו מאמרים רבים. באופן כללי, תחום זה עוסק בניסיון המידול המתמטי והחישובי של בעיית הלמידה. כיצד נגרום למערכת ממוחשבת להבין משהו על סמך הרבה דוגמאות (ולא ע"י הוראה לשיטה מפורשת) ולהכליל מעל הדוגמאות שהיא למדה לפיהן - כך שגם בהינתן מידע שהיא טרם ראתה כמותו, היא תצליח להתמודד עמו בהצלחה. במקרה שלנו, אנחנו נתמקד בשימוש ספציפי של מערכות לומדות והוא זיהוי התוכן של תמונה. ברמה מאוד כללית, עבור יישום זה הציגו למערכת מספר דוגמאות רב של תמונות טבעיות שונות והתיוג שלהן. למשל תמונה של כלב והתיוג "כלב", תמונה של חצוצרה והתיוג "חצוצרה", וכד'. בתהליך האימון, המערכת מקבלת תמונה כקלט ומנחשת מה יהיה התיוג שלה, ולפי ההצלחה או הכישלון היא מעדכנת פרמטרים פנימיים כך שהיא תמזער את כמות הפעמים בהן היא טועה בניחוש שלה.

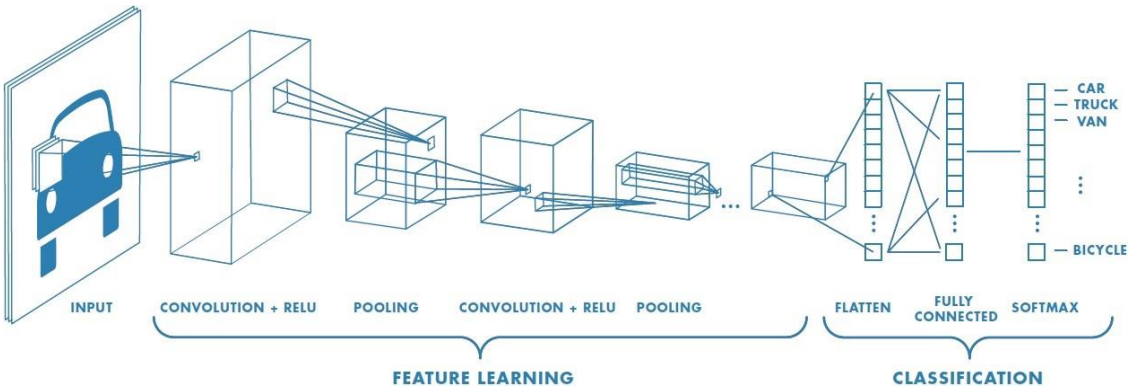
רשתות נוירונים מלאכותיות

כאמור, רשתות נוירונים מלאכותיות הינן אוסף מודלים מתמטיים אשר משמשים מערכות-לומדות. לצורך המאמר לא צריך להכיר לעומק את המודל, אך כן ארצה להתמקד במספר דברים שעתידים להיות רלוונטיים. היחידות הבסיסיות במודל זה הינם **נוירונים מלאכותיים**. נוירון מלאכותי הוא פונקציה f המקבלת וקטור (רצף מספרים), כופלת כל כניסה בוקטור במשקולת התואמת לה, סוכמת את המכפלות, מוסיפה לסכום הנ"ל מספר שנקרא bias ומפעילה פונקציית אקטיבציה שאינה לינארית (הנוירון מחזיר מספר). אם כן, כל f שכזו מאופיינת במשקולות שלה, והיא תיראה כך:

$$f_{\overrightarrow{\theta_{weights}}}(\overrightarrow{x}) = \varphi_{activation}\left(\sum_{i=1}^{n} x_i \cdot \theta_i + \theta_{bias}\right)$$

רשתות נוירונים מכילות **שכבות** של נוירונים. בין השכבות הנוירונים מחוברים ביניהם, כך שנוירון משכבה מסוימת מקבל את וקטור הקלט שלו שהוא למעשה אוסף פלטים של הרבה נוירונים מהשכבה הקודמת. רשתות הנוירונים שמכילות הרבה שכבות נקראות רשתות עמוקות, ומכאן הכינוי Deep Learning שניתן לתת-התחום של למידה המבוסס רשתות נוירונים מלאכותיות עמוקות.

במאמר אנחנו נתקוף **רשת נוירונים קונבולוציוניות** (Convolutional Neural Networks), או בקיצור CNN, שהן רשתות נוירונים בעלות ארכיטקטורה דומה לזו המתוארת באיור הבא:



[ארכיטקטורה טיפוסית ל-CNN, מקור: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html]

באיור זה אנו רואים כי ניתן למעשה לחלק את תהליך זיהוי האובייקט המתפרש ע"פ השכבות לעומק הרשת - לשלב של חילוץ התכונות, ושלב הסיווג. כלומר, בהתחלה רשת הנוירונים מחלצת מתוך התמונה איזה תכונות רלוונטיות יש בה (בזמן אימון רשת הנוירונים, היא 'בחרה' איזה תכונות הן רלוונטיות לטובת הזיהוי). למשל, אם יש בתמונה צורה של גלגל סביר להניח שיש כלי רכב בתמונה. רק בשכבות האחרונות של הרשת, מבוצע הסיווג שיביא להחלטה איזה אובייקט רואים בתמונה. הסיווג הזה נעשה ע"פ התכונות שחולצו מתמונת הקלט, ולא מתמונה עצמה ישירות. רשתות נוירונים מסוג זה הוכיחו את יעילותן עבור שימושים רבים בראייה ממוחשבת ובפרט עבור זיהוי האובייקט בתמונה.

כעת נרחיב על **הפלט** מרשת נוירונים זו. בפועל, מה שחוזר מהרשת הוא וקטור באורך כמות המחלקות האפשריות. אם נניח לדוגמה כי ישנן 1000 קטגוריות אפשריות של סוג האובייקט בתמונה (כלב, חתול,

שולחן, אופניים, פסנתר, תפוז, וכו') אז הוקטור המוחזר יהיה בגודל 1000 (יכיל אלף מספרים) כאשר כל מספר הוא מ-0 ל-1 ומתאר את ההסתברות (מבחינת רשת הנוירונים) שבתמונה יש את האובייקט התואם. עניין זה יהיה רלוונטי עבורנו בהמשך, שכן אנו נרצה להשפיע על הרשת כך שהיא תפלוט הסתברות גבוהה עבור מחלקה (=קטגוריה) אחרת לחלוטין מזו הנכונה, שאכן מופיעה בתמונה.

לפני שנמשיך הלאה, ארצה לדון בנקודה אחרונה והיא **תהליך האימון**. הלמידה הלכה למעשה מתבטאת בשינוי הערכים של המשקולות בכל נוירון שבשכבות השונות, כך שהרשת תצליח לתת את התשובה הרצויה על קלט כלשהו (רשת הנוירונים היא פונקציה שנקבעת ע"פ המון המשקולות שהיא למדה). כפי שהוסבר קודם לכן, רשת הנוירונים מחזירה וקטור של הסתברויות לכל מחלקה. מגדירים פונקציית הפסד עבור תהליך הלמידה - כך שככל שוקטור ההסתברויות הנ"ל "רחוק" מהתיוג האמיתי (שהוא וקטור שמכיל 1 במקום של המחלקה האמיתית, ואפסים בשאר המקומות), כך הקנס יגדל. איני רוצה לפרט יתר על המידה בהיבט המתמטי, אך נציין שהלמידה מתבצעת ע"י חישוב וקטור הגרדיאנט (מעין הכללה של נגזרת למימדים מרובים) של פונקציית ההפסד. וכאשר משנים את המשקולות בכיוון מנוגד לכיוון שקובע וקטור הגרדיאנט ממזערים את פונקציית ההפסד. שיטה זו נקראת Gradient Descent, ואמנם היא אינה מבטיחה הגעה למינימום גלובאלי, אך כאשר מצליחים להתכנס למינימום מקומי (במזעור ההפסד) הוא יכול להוות כזה שנותן תוצאות טובות מספיק.

כלים

את התקיפה נממש בשפת התכנות **Python** (גרסה 3 ומעלה). Python הינה שפת תכנות סקריפטים (בפרט, אינה עוברת הידור) שקלה מאוד לשימוש.

אנחנו נשתמש בספריה **Tensorflow** לטובת הרצת רשת הנוירונים. Tensorflow הינה ספריה שפותחה ע"י Google והיא נותנת תשתית נוחה למדי לעיצוב מודלים של מערכות לומדות (ובייחוד רשתות נוירונים), אימונן, והרצתן.

רשת הנוירונים שנתקוף תהיה מסוג CNN והיא נקראת **AlexNet**. AlexNet עוצבה ע"י קבוצת SuperVision אשר השתתפה בתחרות ImageNet לזיהוי ויזואלי של אובייקטים בתמונה. AlexNet התחרתה בשנת 2012, כאשר התחום היה פחות נפוץ, והגיעה ל-15.3% שגיאת top-5 (שגיאה זו מתירה לכל רשת לנחש 5 מחלקות אפשריות לכל תמונה ולצדוק באחת), שזה שיפור של יותר מ-10.8% מהיכולות שהיו באותה תחרות. זה נחשב הישג משמעותי ביותר (עד כה הצליחו לשפר באחוזים בודדים לכל היותר), והיווה פריצת דרך משמעותית לתחום. במקור, AlexNet לא עוצבה ע"ג tensorflow, ואנחנו נשתמש בארכיטקטורה והמשקולות שהוסבו לשימוש בtensorflow שפורסם באתר תחת אוניברסיטת טורנטו, בקישור להלן: http://www.cs.toronto.edu/~guerzhoy/tf_alexnet. חשוב לציין כי יש באגים במימוש הנ"ל, לכן מומלץ לעקוב אחרי הקוד שאצרף בשלבים השונים במאמר, שעבר עריכה ותיקון של הבאגים.

לבסוף, אציין עוד 3 ספריות שנעשה בהן שימוש: skimage (באמצעותה נקרא קובץ תמונה), matplotlib (באמצעותה נציג את התמונות המזוייפות), numpy (באמצעותה נבצע מספר חישובים אריתמטיים על המטריצות המייצגות את התמונה).

שימוש ב-AlexNet

בשלב זה אציג את הקוד שיאפשר לנו להשתמש ברשת הנוירונים לזיהוי האובייקטים בתמונה. לשם ההפעלה יש להוריד מהקישור http://www.cs.toronto.edu/~guerzhoy/tf_alexnet את הקבצים caffe_classes.py (מכיל רשימה של שמות הקטגוריות, מה שמאפשר המרה בין מספר הקטגוריה לשם הטקסטואלי שלה) ואת bvlc_alexnet.npy (מכיל את המשקולות של הרשת AlexNet המאומנת, כי אנחנו לא נרצה לאמן מחדש את הרשת אלא להשתמש ברשת מוצלחת שעברה אימון). צריך לוודא שהקבצים הללו יימצאו באותו מקום ממנו מריצים את הקוד שנבנה לאורך המאמר.

נתחיל מייבוא הספריות הרלוונטיות עבורנו, והגדרת 2 קבועים:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
import tensorflow as tf
from caffe_classes import class_names

VALID_PAD = 'VALID'
SAME_PAD = 'SAME'
```

כעת ארשום את הקוד שבונה את הארכיטקטורה של הרשת תוך טעינת המשקולות מהקובץ שרשמנו. הוא אולי יראה ארוך, אך כל מה שהוא עושה זה לבנות את רשת הנוירונים שכבה אחר שכבה (כיום רשתות נוירונים מודרניות מגיעות לעומק של מאות שכבות, כך שהקוד הזה אפילו קצר יחסית).

```
def conv(input, kernel, biases, k_h, k_w, c_o, s_h, s_w, padding=VALID_PAD, group=1):
    """ generates a convolutional layer for the neural network """
    c_i = input.get_shape()[-1]
    convolve = lambda i, k: tf.nn.conv2d(i, k, [1, s_h, s_w, 1], padding=padding)
    if group == 1:
        conv = convolve(input, kernel)
    else:
        input_groups = tf.split(input, group, 3)
        kernel_groups = tf.split(kernel, group, 3)
        output_groups = [convolve(i, k) for i, k in zip(input_groups, kernel_groups)]
        conv = tf.concat(output_groups, 3)
    return tf.reshape(tf.nn.bias_add(conv, biases), [-1] + conv.get_shape().as_list()[1:])


def alexnet(net_data, x):
    """ defines the architecture of Alexnet """
    with tf.name_scope('conv1'):
        conv1W = tf.Variable(net_data["conv1"][0])
        conv1b = tf.Variable(net_data["conv1"][1])
        conv1_in = conv(x, conv1W, conv1b, 11, 11, 96, 4, 4, padding=SAME_PAD, group=1)
        conv1 = tf.nn.relu(conv1_in)
    with tf.name_scope('lrn1'):
        lrn1 = tf.nn.local_response_normalization(conv1, depth_radius=2, alpha=2e-05, beta=0.75,
bias=1.0)
    with tf.name_scope('maxpool1'):
        maxpool1 = tf.nn.max_pool(lrn1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding=VALID_PAD)
```

```
    with tf.name_scope('conv2'):
        conv2W = tf.Variable(net_data["conv2"][0])
        conv2b = tf.Variable(net_data["conv2"][1])
        conv2_in = conv(maxpool1, conv2W, conv2b, 5, 5, 256, 1, 1, padding=SAME_PAD, group=2)
        conv2 = tf.nn.relu(conv2_in)
    with tf.name_scope('lrn2'):
        lrn2 = tf.nn.local_response_normalization(conv2, depth_radius=2, alpha=2e-05, beta=0.75,
bias=1.0)
    with tf.name_scope('maxpool2'):
        maxpool2 = tf.nn.max_pool(lrn2, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding=VALID_PAD)
    with tf.name_scope('conv3'):
        conv3W = tf.Variable(net_data["conv3"][0])
        conv3b = tf.Variable(net_data["conv3"][1])
        conv3_in = conv(maxpool2, conv3W, conv3b, 3, 3, 384, 1, 1, padding=SAME_PAD, group=1)
        conv3 = tf.nn.relu(conv3_in)
    with tf.name_scope('conv4'):
        conv4W = tf.Variable(net_data["conv4"][0])
        conv4b = tf.Variable(net_data["conv4"][1])
        conv4_in = conv(conv3, conv4W, conv4b, 3, 3, 384, 1, 1, padding=SAME_PAD, group=2)
        conv4 = tf.nn.relu(conv4_in)
    with tf.name_scope('conv5'):
        conv5W = tf.Variable(net_data["conv5"][0])
        conv5b = tf.Variable(net_data["conv5"][1])
        conv5_in = conv(conv4, conv5W, conv5b, 3, 3, 256, 1, 1, padding=SAME_PAD, group=2)
        conv5 = tf.nn.relu(conv5_in)
    with tf.name_scope('maxpool5'):
        maxpool5 = tf.nn.max_pool(conv5, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding=VALID_PAD)
    with tf.name_scope('fc6'):
        fc6W = tf.Variable(net_data["fc6"][0])
        fc6b = tf.Variable(net_data["fc6"][1])
        maxpool5_flat = tf.reshape(maxpool5, [-1, int(np.prod(maxpool5.get_shape()[1:]))])
        fc6 = tf.nn.relu_layer(maxpool5_flat, fc6W, fc6b)
    with tf.name_scope('fc7'):
        fc7W = tf.Variable(net_data["fc7"][0])
        fc7b = tf.Variable(net_data["fc7"][1])
        fc7 = tf.nn.relu_layer(fc6, fc7W, fc7b)
    with tf.name_scope('fc8'):
        fc8W = tf.Variable(net_data["fc8"][0])
        fc8b = tf.Variable(net_data["fc8"][1])
        fc8 = tf.nn.xw_plus_b(fc7, fc8W, fc8b)
    prob = tf.nn.softmax(fc8)
    return prob, fc8
```

הדבר האחרון שנותר לעשות הוא לרשום פונקציה שמריצה את הרשת על תמונה מסוימת, ומחזירה את 5 הניחושים הטובים ביותר של הרשת עבור התמונה (כלומר, את 5 הקטגוריות שקיבלו את ההסתברות הגבוהה ביותר לניחוש מוצלח, מבחינת רשת הנוירונים):

```
def inference(img_path, net_data, top_amount=5):
    """ runs the alexnet on the given image from path (img_path) with the pretrained weights (net_data)
"""
    img = (imread(img_path)[:, :, :3]).astype(np.float64)
    img -= np.mean(img)
    img = img[:, :, ::-1]
    input_shape = (227, 227, 3)
    x = tf.placeholder(tf.float32, (None,) + input_shape)
    prob, fc8 = alexnet(net_data, x)
    init = tf.global_variables_initializer()
    sess = tf.Session()
    sess.run(init)
    output = sess.run(prob, feed_dict={x: [img]})  # could run on many images
    for input_im_ind in range(output.shape[0]):
        inds = np.argsort(output)[input_im_ind, :]
        print("Image", input_im_ind)
        for i in range(top_amount):
            print(inds[-1 - i], class_names[inds[-1 - i]], output[input_im_ind, inds[-1 - i]])
```

בהקשר ה-inference, נבחין כי רשת הנוירונים אומנה על תמונות קלט בגודל 227 על 227 עם 3 ערוצי צבע. לכן גם התמונה שנבחר צריכה להיות צבעונית (RGB) ובגודל זה (ניתן במסגרת הקוד לבצע התאמה אוטומטית, למשל למרכז התמונה או מתיחה של הקלט וכד').

לבלבל את רשת הנוירונים

נתחיל מתיאור כללי של **תהליך התקיפה:** ניקח תמונה של משאית, שרשת הנוירונים מזהה בהצלחה ובביטחון גבוה כמשאית. נמצא קטגוריה שמקבלת הסתברות לא זניחה על התמונה, ונבצע כמו אימון לרשת הנוירונים - אך שבמקום לעדכן את המשקולות של השכבות הפנימיות של רשת הנוירונים (כפי שתיארנו כשהסברנו את תהליך האימון של הרשת) הפעם נתייחס לתמונת הקלט בתור אוסף של משקולות בפני עצמה. למעשה, אנחנו נרשה עריכה ושינוי של תמונת הקלט בלבד, בכיוון המנוגד לגרדיאנט, כך שתתבצע אופטימיזציה שנועדה להגביר את ההסתברות של הקטגוריה המוחלשת שזיהינו בהתחלה. למען הבהירות, אם בתהליך האימון הרגיל הרשת שינתה את המשקולות הפנימיות שלה כדי למזער את פונקציית ההפסד שבגדול הענישה על ניחושים לא נכונים של הרשת, הפעם אנחנו מגדירים את פונקציית ההפסד בתור מינוס האקטיבציה של הקטגוריה שבחרנו. בכך, כאשר אנו מבצעים אופטימיזציה למזעור מינוס האקטיבציה, אנחנו למעשה ממקסמים את האקטיבציה של המחלקה הנ"ל (וכמובן שזה על חשבון הקטגוריות האחרות ובפרט הקטגוריה המקורית - משאית).

אם כך, נתחיל בבחירת התמונה. זו התמונה שבחרתי, לאחר חיתוך לגודל 227 על 227 (יובהר כי ניתן לבחור בכל תמונה שרוצים, מכל קטגוריה, שרשת הנוירונים מזהה בביטחון גבוה נכונה):

[תמונה של משאית, מקור: https://mobilefleetsolutions.com/transportation-and-logistics/tractor-trailer-on-highway]

נריץ לראשונה את רשת הנוירונים על התמונה של המשאית ונראה אם היא מצליחה לזהות אותה כהלכה.

```
def main():
    net_data = np.load(open("bvlc_alexnet.npy", "rb"), encoding="latin1").item()
    inference("truck.png", net_data)


if __name__ == "__main__":
    main()
```

הרצת הקוד נותנת את הפלט הבא:

```
867 trailer truck, tractor trailer, trucking rig, rig, articulated lorry, semi 0.893489
675 moving van 0.0550013
717 pickup, pickup truck 0.0139656
569 garbage truck, dustcart 0.0118674
757 recreational vehicle, RV, R.V. 0.0108393
```

מה שאנו רואים זה את חמשת המחלקות שקיבלו את הניחוש עם ההסתברות הגבוהה ביותר. בתחילת השורה מופיע מספר הקטגוריה, לאחר מכן התיאור הטקסטואלי שלה, ולבסוף ההסתברות. אכן, מקום ראשון עם כ-89% ביטחון, מופיעה הקטגוריה של trailer truck (השמות בהמשך זה כינויים נוספים שמשתתפים באותה המחלקה). נשים לב שבאופן משמעותי יש לה יותר ביטחון בקטגוריה משאית מהקטגוריה הבאה בתור, שהיא moving van עם רק כ-5% ביטחון.

בחירת מחלקת יעד לזיוף

כעת נרצה לבחור את הקטגוריה שאותה נרמה את רשת הנוירונים לזהות. לא נרצה משהו עם הסתברות אפסית לחלוטין, כי זה כנראה יקשה עלינו, אך גם נוכל 'להתפרע' ולקחת משהו עם הסתברות ביטחון נמוכה מאוד. לשם כך, נדפיס את 20 התוצאות הגבוהות ע"י השינוי הבא בקריאה ל-inference:

```
    inference("truck.png", net_data, 20)
```

אני אחסוך מלרשום את כל הפלט, אך אציג את המחלקה ה-18 בדירוג ההסתברותי היא toaster:

```
859 toaster 0.000226374
```

נבחין כי מבחינת רשת הנוירונים, הסיכוי שבתמונה של המשאית שלנו יש למעשה טוסטר הוא כ-0.02%, שזו כבר הסתברות קטנה למדי. למרות זאת, התקיפה שלנו תצליח ונגרום לרשת לחשוב בביטחון גבוה כי בתמונה יש טוסטר.

מימוש התקיפה

נתחיל בתקיפה עצמה כמו ע"פ התהליך שתואר בתחילה. 'נאמן' את רשת הנוירונים כך שהיא תשנה אך ורק את התמונה המקורית ע"מ למקסם את האקטיבציה על המחלקה של הטוסטר. להלן הקוד שיאמן את הרשת באופן הנ"ל:

```
def attack(original_img, net_data, fake_class, original_class, train_iterations=50):
    """ fools the neural network by optimizing input toward fake_class classification """
    input_shape = list(original_img.shape)
    init_var_input_image = tf.constant(original_img.astype(np.float32).reshape([1] + input_shape))
    input_image_var_unchanged = tf.get_variable("input_image_variable",
initializer=init_var_input_image)
    black_img = np.zeros(input_shape, dtype=np.float64)
    x = tf.placeholder(tf.float32, [None]+ input_shape)
    initial_image = tf.get_variable("initial_image_variable", initializer=init_var_input_image)
    opt_im_var = tf.Variable(initial_image)
    opt_x = x + opt_im_var
    prob, fc8 = alexnet(net_data, opt_x)
    loss = -fc8[0, fake_class]  # optimize into fake class

    train_step = tf.train.AdamOptimizer(0.95).minimize(loss, var_list=[opt_im_var])
    init = tf.global_variables_initializer()
    sess = tf.Session()
    sess.run(init)
    for i in range(train_iterations):
        var_mean = np.mean(opt_im_var.eval(session=sess))
        output = sess.run(prob, feed_dict={x: [black_img - var_mean]})
        sess.run(train_step, feed_dict={x: [black_img]})
    print("fake prob:",output[0, fake_class], "\toriginal prob:",output[0, original_class])
    fake = opt_im_var.eval(session=sess)[0, :, :, ::-1]
    for channel in range(3):
        fake[:,:,channel] -= fake[:,:,channel].min()
        fake[:,:,channel] /= fake[:,:,channel].max()
    plt.imshow(fake)
    plt.show()
```

הקוד עובד כפי שתואר תהליך התקיפה: היא מתייחס לתמונת הקלט (המשאית שלנו) בתור אוסף משקולות שהוא יכול לשנות ע"מ למזער את פונקציית ההפסד החדשה שלנו, שהיא מינוס האקטיבציה של הנוירון של טוסטר בשכבה האחרונה. בכוונה לא לקחתי את ההסתברות של טוסטר אלא את האקטיבציה הלקוחה מתוך fc8 השכבה האחרונה של רשת הנוירונים שממנה ישירות נגזרת ההסתברות אחריה הפעלת softmax, כי אחרת כדי למזער את פונקציית ההפסד שהגדרנו, תהליך האימון היה גורם לרשת הנוירונים לשנות את הפיקסלים כך שהרשת תזהה בפחות הצלחה את כל המחלקות האחרות, במקום לזהות ביתר הצלחה את המחלקה של טוסטר כפי שאנחנו רוצים.

ובכן, נעדכן את פונקציית main, נריץ ונצפה בתוצאה (לאחר שהרצנו 50 איטרציות של אימון ושינינו את תמונת הקלט):

```
def main():
    net_data = np.load(open("bvlc_alexnet.npy", "rb"), encoding="latin1").item()
    img = (imread('truck.png')[:, :, :3]).astype(np.float64)
    img -= np.mean(img)
    img = img[:, :, ::-1]
    attack(img, net_data, 859, 867)  # 859=toaster class id, 867=truck class id
```

מה שמייצר את הפלט:

```
fake prob: 1.0  original prob: 2.89078e-33
```

בהצלחה יתרה, גרמנו לרשת לחשוב שמדובר בתמונה של "טוסטר" בביטחון של 100%. כמובן שלא באמת מדובר ב-100% אלא מספר מאוד קרוב לזה, אך בשל אילוצי דיוק וייצוג מספרים ממשיים במחשב הדיגיטלי, הקרבה התעגלה להסתברות 1. כלומר, בעוד שבמקור הרשת חשבה שהיא "טוסטר" רק בסיכוי של כ-0.02% כעת היא בטוחה לחלוטין שהיא רואה "טוסטר" בתמונה (ב-100%). כעת, על התמונה המזויפת שלנו, הרשת חושבת שהיא רואה "משאית" בסיכוי של כ-$\frac{3}{10^{31}}$%. קשה לתפוס כמה קטן מספר זה.

אך לא סיימנו. יש כאן בעיה לא קטנה, אם נתבונן בתמונה המזויפת שקיבלנו לאחר התקיפה, היא נראית כך:

קל לראות את השינויים שהתקיפה שעשינו ביצעה לתמונה. לעין אנושית זה נראה כאילו התמונה מלאה במריחות צבעוניות מוזרות.

תקיפה שקטה יותר

אנחנו רוצים לעשות את התקיפה באופן שקט, שלא יצליחו לזהות אותה אם יתבוננו בתמונה. לשם כך, נוסיף רגולריזציה לפונקציית ההפסד שלנו. לא רק נמזער את מינוס האקטיבציה של "טוסטר", אלא גם נעניש על כל שינוי שנעשה בתמונה המקורית. במילים אחרות, נחשב את סכום ריבועיי ההפרשים בין הפיקסלים בתמונה המקורית לזו הערוכה, ואת ערך זה נוסיף (חיבור) לפונקציית ההפסד (מכיוון שאותה אנו נרצה למזער, תוספת חיובית תגרור הענשה על הסטייה מהתמונה המקורית). בכך נגרום לרשת מצד אחד לשנות את התמונה כדי להגביר את האקטיבציה של הטוסטר ומצד שני לא לשנות באופן קיצוני (או, ניכר לעין) את התמונה.

נעדכן את פונקציית האימון כך (שימו לב לתוספת *2* השורות שאחרי ההגדרה של פונקציית ההפסד *loss*):

```
def attack(original_img, net_data, fake_class, original_class, train_iterations=50):
    """ fools the neural network by optimizing input toward fake_class classification """
    input_shape = list(original_img.shape)
    init_var_input_image = tf.constant(original_img.astype(np.float32).reshape([1] + input_shape))
    input_image_var_unchanged = tf.get_variable("input_image_variable",
initializer=init_var_input_image)
    black_img = np.zeros(input_shape, dtype=np.float64)
    x = tf.placeholder(tf.float32, [None]+ input_shape)
    initial_image = tf.get_variable("initial_image_variable", initializer=init_var_input_image)
    opt_im_var = tf.Variable(initial_image)
    opt_x = x + opt_im_var
    prob, fc8 = alexnet(net_data, opt_x)
    loss = -fc8[0, fake_class]  # optimize into fake class
    penalty_on_input_change = (tf.reduce_sum(tf.square(tf.subtract(opt_im_var,
input_image_var_unchanged))))
    loss += 0.00003 * penalty_on_input_change  # regularize

    train_step = tf.train.AdamOptimizer(0.95).minimize(loss, var_list=[opt_im_var])
    init = tf.global_variables_initializer()
    sess = tf.Session()
    sess.run(init)
    for i in range(train_iterations):
        var_mean = np.mean(opt_im_var.eval(session=sess))
        output = sess.run(prob, feed_dict={x: [black_img - var_mean]})
        sess.run(train_step, feed_dict={x: [black_img]})
    print("fake prob:",output[0, fake_class], "\toriginal prob:",output[0, original_class])
    fake = opt_im_var.eval(session=sess)[0, :, :, ::-1]
    for channel in range(3):
        fake[:,:,channel] -= fake[:,:,channel].min()
        fake[:,:,channel] /= fake[:,:,channel].max()
    plt.imshow(fake)
    plt.show()
```

נריץ, ונקבל את התוצאה הבאה:

```
fake prob: 0.996366      original prob: 0.00278609
```

ויתרנו במעט על הביטחון הגבוה של הרשת המותקפת ב-"טוסטר". כעת היא חושבת שמדובר ב-"טוסטר" בסיכוי של כ-99% "בלבד" (נזכיר, שאת התמונה המקורית של המשאית היא זיהתה בריצה תקינה בסיכוי של כ-89%). כמו כן, ניתן לציין כי על התמונה שייצרנו הרשת תגיד שמדובר ב-"משאית" בסיכוי של כ-0.2% בלבד. וכיצד נראית התמונה הערוכה שלנו הפעם?

הרבה יותר קשה להבחין בתקיפה. חדי העין אולי יראו בשמיים שמצד שמאל למעלה את הרעש בסגנון מה שהיה כאשר לא הוספנו רגולריזציה. בנוסף, יכולנו להגביר את הרגולריזציה, על חשבון הביטחון של הרשת במחלקה המזויפת, ולהגביר את נראות התמונה הערוכה כתמונה המקורית.

סיכום

תחילה למדנו על קצה המזלג מה היא רשת נוירונים לזיהוי אובייקטים בתמונה, ואז ראינו כיצד ניתן לתקוף אותה ולגרום לה לראות עצם שונה ממה שיש בפועל בתמונה, לבחירתנו.

כפי שהוצג בהקדמה למאמר, רשתות נוירונים משמשות היום במגוון רחב של תחומים. חלקן אף מובילות להחלטות הרות גורל, למשל לזיהוי מחלות רפואיות וקבלת ההחלטה אם צריך לבצע ניתוח או לא, או לנסיעה אוטונומית אם מדובר ברמזור אדום או לא, וכד'. צריך לזכור שהן ניתנות לתקיפה, ויחסית בקלות (הן לא יציבות מאוד לשינויים בקלט). אמנם, הסבירות שרשת הנוירונים תטעה על תמונה טבעית הוא נמוך יותר (לעומת תמונה שעברה שינוי מכוון לשם התקיפה), אך תמיד הסיכוי קיים. ישנם עבודות נוספות בתחומים האלו, למשל תקיפה דומה לזו שהוצגה אך שמשנה פיקסל בודד בלבד, או מדבקה פיזית בגודל של כף-יד (שפיתחו חוקרים מ-Google) שכשהיא מופיעה בתמונה עד כדי גודל מסוים הרשת מזהה שמדובר בטוסטר בביטחון גבוה. אציין גם שראיתי עבודה שמנסה להגן על רשתות הנוירונים, באמצעות רשת נוירונים אחרת שמנסה להסיר 'רעש' בסגנון זה שנוצר בעקבות תקיפות כמו אלו (כמו המריחות הצבעוניות שהבחנו בהן), כך שתמונות שמגיעות כקלט לרשת 'נוקו' או 'שוחזרו' מהתקיפה שבוצעה, לו היא בוצעה.

על המחבר

שמי רון אורבך ואני מתעניין בפיתוח תוכנה, ראייה ממוחשבת ואבטחת מידע. לשאלות והערות ניתן לפנות אליי בכתובת ron.urbach.mail@gmail.com.