Transfer Learning Using Convolutional Neural Networks to Detect Spoofs in Fingerprints

Chris Denniston

December 14, 2016

This paper was written by the author in their own words and all external knowledge and literature has been referenced properly in it's respective text.

Abstract

Detection of fake fingerprints is an important task in the field of biometrics. Classifiers must be able to distinguish not only fingerprints from other fingerprints, but real fingerprints from fake fingerprints. These fake fingerprints are called "spoofs". A neural network is proposed as one solution to detect spoof fingerprints from images. Neural networks that utilize image classification are time consuming to train. For this reason, it is proposed that a pre-trained image classification neural network be used as a feature extractor and a simpler trainable neural network be used on those extracted features to produce a classifier. This classifier produces a .96 testing AUC [1a] when trained against the full dataset and at least a .95 testing AUC [2a] [2b] when a spoof material it has not seen is introduced.

1 Introduction

Detection of fake fingerprints are an important part of a fingerprint based biometric system. These fake fingerprints are commonly called "spoofs." It is not enough that the system can tell apart one person from another, but also must be able to tell apart fake fingerprints from real fingerprints to inspire real confidence in it's users. Spoof fingerprints can be made from common household materials and more material's are appearing which can be used to make these spoofs. Because of this it is also important that a spoof detection system also can detect spoofs when novel types of materials are introduced to it.

A neural network may be used to solve this complex classification problem as it has the ability to learn from a large variety of samples and draw complex decision boundaries. ([1] Section 2.2.1) Neural networks are expensive to train for complex problems involving large dimensional data, such as images. It is possible to take a pre-trained general image classification neural network and reapply it to this classification task. This enables a much quicker training time and more accurate results than could be achieved by a small team.

2 Methods

The LivDet11 Database provides four types of detectors and five spoof materials. For training purposes two of the detectors were used: The Digital and Sagem detectors. Alexnet was used as a general image feature extractor. Alexnet has been trained on the ImageNet database, which is a general purpose image database.([2] 1) Alexnet has achieved a top-5 test error rate of 15.3% in the ILSVRC-2012 competition ([2] 1). AlexNet "consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax" ([2] 1) Convolutional Layers are particularly well suited for image processing because they take into account the spatial structure of the image. ([3] 1)

Alexnet is trained as a general image classifier, which is unsuitable for this experiment. In order to use it as a general image feature extractor, five of it's final layers were removed. These layers in reverse order were a softmax classification layer, a dropout layer, a fully connected layer with Relu activation, a dropout layer, and a fully connected layer with Relu activation. This modified Alexnet was then activated on each of the images and the activations were saved to files.

A simple neural network with 4096 inputs (The output size of the last layer of the modified Alexnet), a variable hidden layer size, and a single output was trained. All of the activations in that classifier were hypertangent. The chosen objective function was mean squared error, as it is the most common criterion. ([1] Section B.6)

For inference on unknown materials the same method and classifier were applied, except the classifier was trained on all but 10% of the live samples and all but one of the spoof types. The classifier was then tested on the remaining 10% of the live samples and all of the spoof samples for that particular material.

3 Results

It was found that 50 hidden layer neurons worked well when trained with stochastic gradient descent with a learning rate of 0.01 and a momentum of 0.01. Validation data was taken as 10% of the training data. It achieved a validation AUC of .99 [1b] and a testing AUC of .96 [1a] after training for 100 epochs on the training data. A higher momentum was useful because it helped to speed up the stabilize the convergence of training. ([1] Section 4.3.1)

For testing with the introduction of novel materials it was found that Silicone produced the lowest AUC at .92 [2d]. This was followed by Gelatine [2a] and Latex [2b] which both achieved an AUC of .95. Playdoh had an AUC of .96 [2c]. WoodGlue was the most easy to determine when it was introduced with an AUC of .99 [2e].

Experiments with the number of hidden layer neurons showed that a small amount of neurons could classify spoofs well. It is believed this is because the number of parameters explodes with the number of hidden layer nodes ([1]

Section 4.5.1) and the difficulty of classification on extracted features is low. For contrast, the Alexnet network has about 4 million parameters. A rule of thumb states that there should be 10 times the number of parameters as data. ([1] Section 4.5.1) To fully train Alexnet for this classification purpose there would need to be 40 million images, which is unrealistic for a smaller company or research team.

Momentum was useful when training the network because it allowed the system to overcome local minima in the performance surface. ([1] Section 4.3.1) The AUCs were much more repeatable when this momentum constant was larger.

Because the classification neural network is only a traditional feed forward neural network, it is very efficient to train. Running each test took less than a minute on a Nvidia GTX 780, a consumer lower end graphics card.

4 Conclusion

The classifier produces good results with a relatively short training period. The classifier also performs well when exposed to spoof material types it has not seen. Feature extraction using a general purpose image classification neural network produces simpler features which can be more readily trained for specialized image classification tasks. In the future other pretrained image classification networks could be used for feature extraction or multiple could be used to produce different kinds of features.

5 Reference

References

[1] e. a. J. C. Prncipe, *Neural and Adaptive Systems: Fundamentals Through Simulations*. New York: Wiley, 1999.

[2] G. E. H. Alex Krizhevsky, Ilya Sutskever, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105.

[3] M. Nielsen, "Chapter 6: Deep learning." electronic.

6 Appendix

```
from keras.optimizers import SGD
from keras import *
from keras.layers import *
from keras.models import *
from convnetskeras.convnets import preprocess_image_batch, convnet
import os
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import random
import pickle


def make_extractor_model():
    model = convnet('alexnet',weights_path="alexnet_weights.h5", heatmap=False)
    model.layers = model.layers[:-4]
    model = Model(input=model.input,output=model.layers[-1].output)
    return model

def make_model(hidden_layer_neurons):
    return Sequential([
        Dense(hidden_layer_neurons, input_dim=4096,name="cust_dense_1",init="uni
        Activation('tanh'),
        Dense(1),
        Activation('tanh')])



def flatten_once(l):
    return [item for sublist in l for item in sublist]

def split_train_validate(l,fraction):
    shuffled_data = sorted(l,key=lambda k: random.random())
    return (shuffled_data[int(fraction * len(shuffled_data)):], shuffled_data[:i

# Plot data
def plot_results(y_test, y_score, title,location,show=False):
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)
    if roc_auc < .7:
        return #IgIgIgIgnore it
    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.05])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    full_title = 'ROC,' + ('AUC: %0.2f' % roc_auc) +  str(title)
    plt.title(full_title)
    try:
        os.mkdir("./roc_curves/" + location)
    except Exception:
```

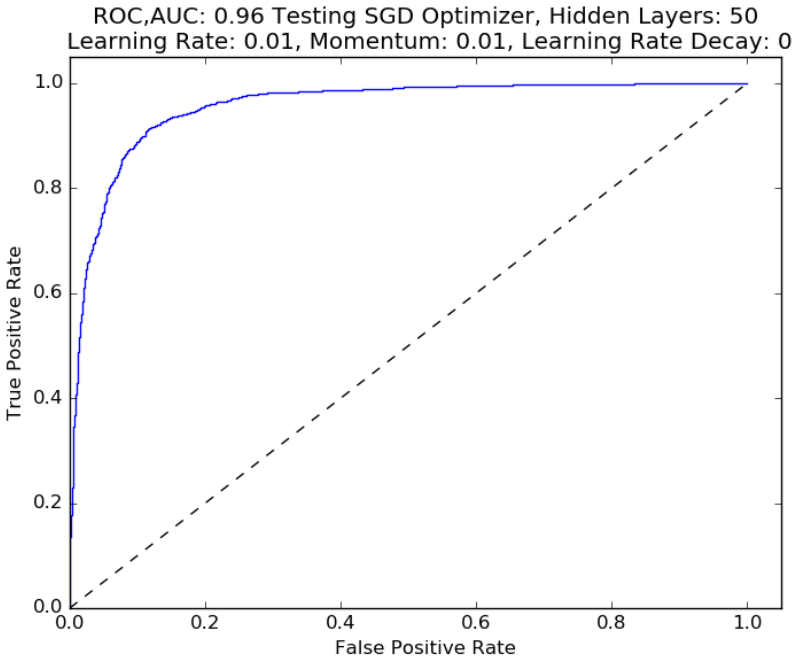
```
            #It prolly exists
            pass
        plt.savefig("./roc_curves/" +location+ "/" + full_title +".png".replace("\n"
        if(show):
            plt.show()
        plt.close()


class CustTrainingResults:
    def __init__(self,test_targets,test_outputs,validation_targets,validation_ou
        self.test_targets = test_targets
        self.test_outputs = test_outputs
        self.validation_targets = validation_targets
        self.validation_outputs = validation_outputs
        self.training_targets = training_targets
        self.training_outputs = training_outputs

def train_all_images(model,epochs,validation_split_fraction):
    BATCH_SIZE = 100
    training_imgs = []
    training_targets = []
    testing_imgs = []
    testing_targets = []
    for root, dirs, files in os.walk("./LiveDet11AlexnetExtracted"):
        for f in files:
            fname = os.path.join(root,f)
            if "Training" in fname:
                training_imgs.append(fname)
                if "Spoof" in fname:
                    training_targets.append([1])
                else:
                    training_targets.append([0])
            if "Testing" in fname:
                testing_imgs.append(fname)
                if "Spoof" in fname:
                    testing_targets.append([1])
                else:
                    testing_targets.append([0])
    fraction_using = 1
    training = list(zip(training_imgs,training_targets))
    testing = list(zip(testing_imgs,testing_targets))
    training = random.sample(training,int(len(training) / fraction_using))
    testing = random.sample(testing,int(len(testing) / fraction_using))
    training,validation = split_train_validate(training,validation_split_fractio

    cur_images = []
```

```
    for f in map(lambda a:a[0],training):
        with open(f, 'rb') as f:
            cur_images.append(pickle.load(f))
    cur_images = np.array(cur_images)
    cur_targets = list(map(lambda a:a[1], training))
    print(cur_images)

    validation_images = []
    for f in map(lambda a:a[0],validation):
        with open(f, 'rb') as f:
            validation_images.append(pickle.load(f))
    validation_images = np.array(validation_images)
    validation_targets = list(map(lambda a:a[1], validation))


    model.fit(cur_images, cur_targets, nb_epoch=100,shuffle=True,validation_data

    testing_imgs = []
    for f in map(lambda a:a[0],testing):
        with open(f, 'rb') as f:
            testing_imgs.append(pickle.load(f))
    testing_imgs = np.array(testing_imgs)

    print(testing_imgs)
    return  CustTrainingResults(list(map(lambda a: a[1], testing)),model.predict
                list(map(lambda a: a[1], validation)),model.predict(validation_in
                list(map(lambda a: a[1], training)),model.predict(cur_images,bat



    return (list(map(lambda a: a[1], testing)),model.predict(testing_imgs,batch_


def preprocess_alexnet(imgs):
    return preprocess_image_batch(imgs,img_size=(256,256), crop_size=(227,227))


def extract_features(model):
    images = []
    for root, dirs, files in os.walk("."):
        for f in files:
            fname = os.path.join(root,f)
            if "Digital" in fname or "Sagem" in fname:
                images.append(fname)
    features = model.predict(preprocess_alexnet(images),verbose=1)
    for fname,feature in zip(images,features):
```

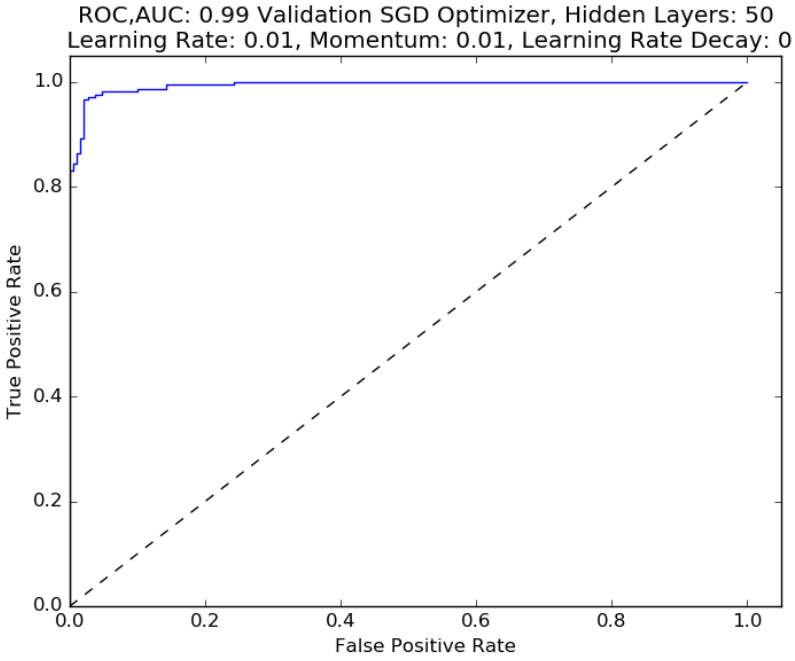
```
        with open("./LiveDet11AlexnetExtracted" + (fname[10:].replace("/","")),"
            pickle.dump(feature,f)


def train_and_plot(model,epochs,validation_split_fraction,title):
    results =  train_all_images(model,epochs,validation_split_fraction)
    plot_results(results.training_targets,results.training_outputs,
"Training",title)
    plot_results(results.validation_targets,results.validation_outputs, "Validat
    plot_results(results.test_targets,results.test_outputs, "Testing",title)

EPOCHS = 100
VALIDATION_SPLIT = 1/10

def test_sgd(learning_rate,momentum,decay, hidden_layer_neurons):
    model = make_model(hidden_layer_neurons)
    sgd = SGD(lr=learning_rate, momentum=momentum,decay=decay)
    model.compile(optimizer=sgd, loss="mse")
    title = "SGD Optimizer, Hidden Layers: " + str(hidden_layer_neurons) +"\n Le
    train_and_plot(model,EPOCHS,VALIDATION_SPLIT,title)

#test_sgd(0.01,1e-2,0,50)
#test_sgd(0.1,1e-2,0,50)
#test_sgd(0.001,1e-2,0,50)
#test_sgd(0.0001,1e-2,0,50)
#test_sgd(0.00001,1e-2,0,50)




class CustLeaveOutResults:
    def __init__(self,left_out_material,training_targets,training_outputs,left_ou
        self.left_out_material = left_out_material
        self.training_targets = training_targets
        self.training_outputs = training_outputs
        self.left_out_targets = left_out_targets
        self.left_out_output = left_out_output

#Begin Leave out tests
def train_leave_one_out(model,out_type):
    BATCH_SIZE = 100
    training_imgs = []
    training_targets = []
    testing_imgs = []
```

```
    testing_targets = []
    for root, dirs, files in os.walk("./LiveDet11AlexnetExtracted"):
        for f in files:
            fname = os.path.join(root,f)
            if out_type in fname:
                testing_imgs.append(fname)
                if "Spoof" in fname:
                    testing_targets.append([1])
                else:
                    testing_targets.append([0])
            else:
                training_imgs.append(fname)
                if "Spoof" in fname:
                    training_targets.append([1])
                else:
                    training_targets.append([0])

    training = list(zip(training_imgs,training_targets))
    testing = list(zip(testing_imgs,testing_targets))

    training, extra_testing_samples = split_train_validate(training,1/10)
    #Only include non spoof
    testing += list(filter(lambda a: a[1] == [0], extra_testing_samples))


    cur_images = []
    for f in map(lambda a:a[0],training):
        with open(f, 'rb') as f:
            cur_images.append(pickle.load(f))
    cur_images = np.array(cur_images)
    cur_targets = list(map(lambda a:a[1], training))

    model.fit(cur_images, cur_targets, nb_epoch=100,shuffle=True,batch_size=BATC

    testing_imgs = []
    for f in map(lambda a:a[0],testing):
        with open(f, 'rb') as f:
            testing_imgs.append(pickle.load(f))
    testing_imgs = np.array(testing_imgs)
    return CustLeaveOutResults(out_type,
            list(map(lambda a: a[1], training)),model.predict(cur_images,batch_s
            list(map(lambda a: a[1], testing)),model.predict(testing_imgs,batch_

def test_leave_out(learning_rate,momentum,decay, hidden_layer_neurons):
    types = ["Gelatine","Latex","Playdoh","Silicone","WoodGlue"]
    for type in types:
```

```
        model = make_model(hidden_layer_neurons)
        sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay)
        model.compile(optimizer=sgd, loss="mse")
        title = "Leave Out " + type
        results = train_leave_one_out(model,type)
        title = " Training without type" + type + " "
        plot_results(results.training_targets,results.training_outputs,title,typ
        title = " Testing when " + type + " is introduced"
        plot_results(results.left_out_targets,results.left_out_output,title,type


test_leave_out(.01,.001,0,50)
```

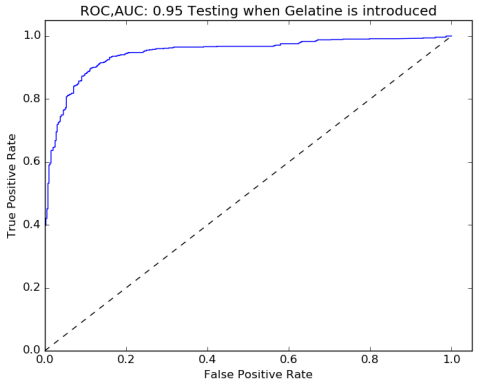


(a) Testing

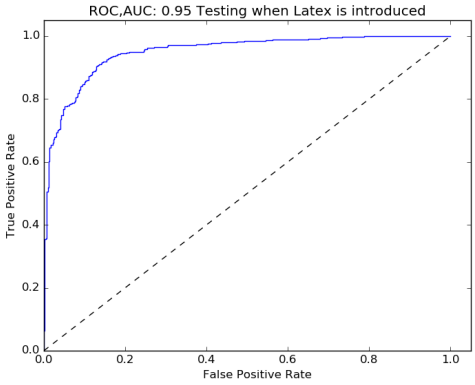


(b) Validation

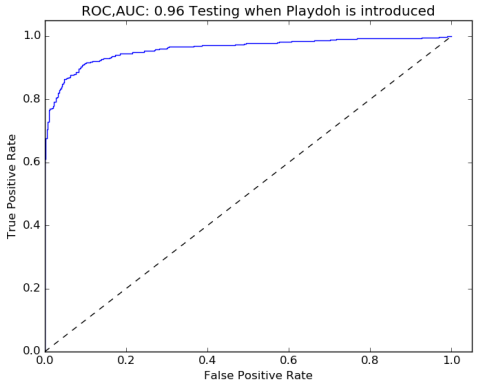
Figure 1: ROC Curves For Training On All Samples



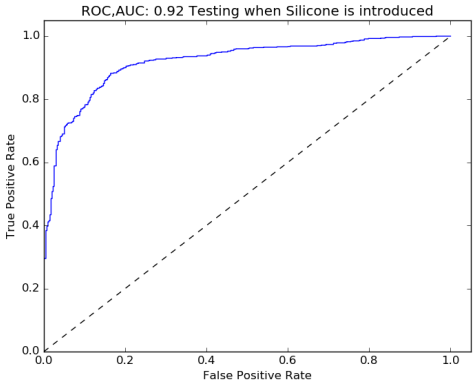
(a) Gelatine



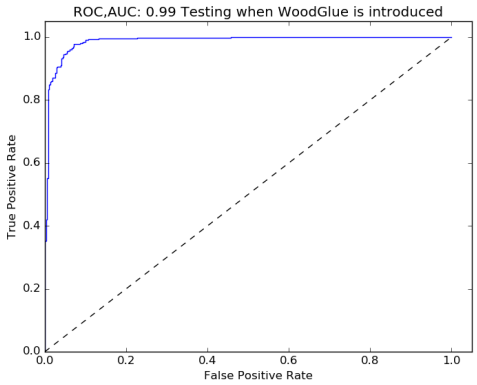
(b) Latex



(c) Playdoh



(d) Silicone



(e) Wood Glue

Figure 2: ROC Curves When Novel Spoofs Are Introduced