

```

def integratedGradients(model,X,Xbase,Steps=50):
    '''
    model: Keras (Tensorflow backend) model
    X: Input vector ~ True image
    Xbase: Baseline Vector
    Steps: How close do you want to approximate integral
    '''

    # Initialize
    X_new = X-Xbase # Construct (X-X') term
    funcinp = []

    # Construct Function Inputs
    for i in range(Steps):
        # Sum over input to F(x):  $x' + (k/m) * (X' - X)$ 
        temp = X_new*(float(i)/Steps)+Xbase
        funcinp.append(temp)

    # Calculate Gradients with TensorFlow backend
    xx = model.inputs[0]
    f = model(xx)
    df_dx = tf.gradients(f,xx)
    # Feed in data to input tensor F(temp)
    a = sess.run(df_dx, feed_dict={xx:funcinp})

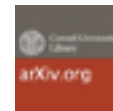
    # Riemann Sum
    summate = [sum(i) for i in zip(*a[0])]
    # Multiply by constant (1/m)
    avg = np.asarray(summate)*(1./Steps)
    # Multiply each value by (X-X') term
    ig = [avg[i]*X_new[i] for i in range(len(avg))]
    return ig

```

## Other Approaches



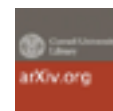
## Pattern Net



*PatternNet and PatternAttribution*

stat.ML (Oct 2017) by Kindermans et al

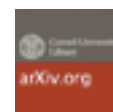
## Influence Functions



*Understanding Black-box Predictions via Influence Functions*

ICML (Jul 2017) by Wei Koh et al

## Explainable Soft Decision Tree



*Distilling a Neural Network Into a Soft Decision Tree*

ICML (Nov 2017) by Frosst et al