Deep Learning Lab (MCEN-394)

MASTER OF TECHNOLOGY

IN

COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING AND TECHNOLOGY

JAMIA MILLIA ISLAMIA, NEW DELHI - 110025

Submitted To

Dr. Sarfaraz Masood

Dr. Faiyaz Ahmed

Submitted By

Muhammad Mahad (23MCS015)

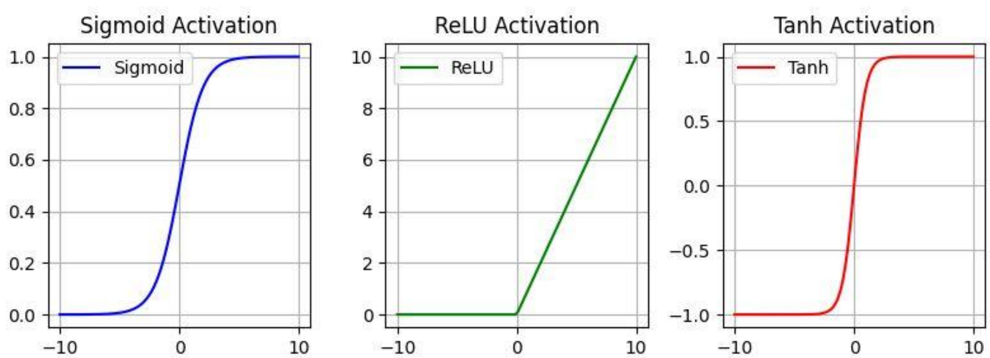
INDEX

1.	Implementation of most commonly used activation functions in Deep Neural Networks (Sigmoid, ReLU, tanh).
2.	Implement a single filter convolution over a variable-sized image using a variable-sized filter with varying stride.
3.	Implement a MULTI Filter convolution over a variable-sized image using a variable-sized filter with varying stride and padding.
4.	Implement the CNN architecture on the MNIST dataset.
5.	Implement the CNN architecture on the CIFAR dataset.
6.	WAP to calculate the number of parameters at each layer of a Conv network and finally sum them to present the total no of params for the network. The program must be generic, i.e. the user may provide a different number of input channels, a different number of Conv layers with individual filter sizes as well as varying dense network architectures.
7.	WAP to implement the POOLING OPERATION of a CNN. Consider a Square image and a square filter size, with variable stride(slide) value and padding. Images should be randomly initialized between (0,255) while the filters should be initialized randomly between (- 1,1).The user should only provide the image size, filter size, stride value, and padding value.
8.	Implement the task of Watermark removal using autoencoders.
9.	Implement the LeNet-5 CNN using tensor flow.
10.	Implement the Alex Net CNN using tensor flow.

1. **Implementation of most commonly used activation functions in Deep Neural Networks (Sigmoid, ReLU, tanh).**

```
[ ] import numpy as np
    import matplotlib.pyplot as plt

    # Activation Functions
    def sigmoid(x):
        """Sigmoid activation function."""
        return 1 / (1 + np.exp(-x))

    def relu(x):
        """ReLU activation function."""
        return np.maximum(0, x)

    def tanh(x):
        """Tanh activation function."""
        return np.tanh(x)

    # Test input range
    x = np.linspace(-10, 10, 100)

    # Apply activation functions
    sigmoid_output = sigmoid(x)
    relu_output = relu(x)
    tanh_output = tanh(x)
```

Output:



2. **Implement a single filter convolution over a variable-sized image using a variable-sized filter with varying stride.**

```
import numpy as np

def convolve2d(image, filter_kernel, stride=1, padding=0):
    # Add padding to the input image
    if padding > 0:
        image = np.pad(image, ((padding, padding), (padding, padding)), mode='constant', constant_values=0)

    # Get dimensions of the image and the filter
    image_h, image_w = image.shape
    filter_h, filter_w = filter_kernel.shape

    # Calculate output dimensions
    output_h = (image_h - filter_h) // stride + 1
    output_w = (image_w - filter_w) // stride + 1

    # Initialize the output
    output = np.zeros((output_h, output_w))
```

```
    # Initialize the output
    output = np.zeros((output_h, output_w))

    # Perform the convolution
    for i in range(0, output_h):
        for j in range(0, output_w):
            # Extract the region of the image
            region = image[i * stride:i * stride + filter_h, j * stride:j * stride + filter_w]
            # Element-wise multiplication and summation
            output[i, j] = np.sum(region * filter_kernel)

    return output

# Example usage
if __name__ == "__main__":
    # Define a sample image (5x5)
    image = np.array([
        [1, 2, 3, 4, 5],
        [6, 7, 8, 9, 10],
        [11, 12, 13, 14, 15],
        [16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25]
    ])
```

```
    # Define a sample filter (3x3)
    filter_kernel = np.array([
        [1, 0, -1],
        [1, 0, -1],
        [1, 0, -1]
    ])

    # Convolve with stride=2 and padding=0
    stride = 2
    padding = 0
    output = convolve2d(image, filter_kernel, stride, padding)

    print("Input Image:\n", image)
    print("\nFilter Kernel:\n", filter_kernel)
    print("\nConvolution Output:\n", output)
```

Output:

```
Input Image:
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]

Filter Kernel:
 [[ 1  0 -1]
 [ 1  0 -1]
 [ 1  0 -1]]

Convolution Output:
 [[-6. -6.]
 [-6. -6.]]
```

3. **Implement a MULTI Filter convolution over a variable-sized image using a variable-sized filter with varying stride and padding.**

```python
import numpy as np

def multi_filter_convolve2d(image, filters, stride=1, padding=0):

    # Add padding to the input image
    if padding > 0:
        image = np.pad(image, ((padding, padding), (padding, padding)), mode='constant', constant_values=0)

    # Get dimensions of the image and the filters
    image_h, image_w = image.shape
    num_filters, filter_h, filter_w = filters.shape

    # Calculate output dimensions
    output_h = (image_h - filter_h) // stride + 1
    output_w = (image_w - filter_w) // stride + 1

    # Initialize the output
    output = np.zeros((num_filters, output_h, output_w))
```

```python
    # Perform the convolution for each filter
    for f in range(num_filters):
        for i in range(0, output_h):
            for j in range(0, output_w):
                # Extract the region of the image
                region = image[i * stride:i * stride + filter_h, j * stride:j * stride + filter_w]
                # Apply the filter
                output[f, i, j] = np.sum(region * filters[f])

    return output

# Example usage
if __name__ == "__main__":
    # Define a sample image (5x5)
    image = np.array([
        [1, 2, 3, 4, 5],
        [6, 7, 8, 9, 10],
        [11, 12, 13, 14, 15],
        [16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25]
    ])
```

```python
if __name__ == "__main__":
    # Define a sample image (5x5)
    image = np.array([
        [1, 2, 3, 4, 5],
        [6, 7, 8, 9, 10],
        [11, 12, 13, 14, 15],
        [16, 17, 18, 19, 20],
        [21, 22, 23, 24, 25]
    ])

    # Define multiple filters (3 filters, each 3x3)
    filters = np.array([
        [[1, 0, -1], [1, 0, -1], [1, 0, -1]],  # Filter 1
        [[1, 1, 1], [0, 0, 0], [-1, -1, -1]],  # Filter 2
        [[0, 1, 0], [1, -4, 1], [0, 1, 0]]    # Filter 3 (Laplacian)
    ])

    # Convolve with stride=2 and padding=1
    stride = 2
    padding = 1
    output = multi_filter_convolve2d(image, filters, stride, padding)

    print("Input Image:\n", image)
    print("\nFilters:\n", filters)
    print("\nConvolution Output:\n", output)
```

Output

```
Input Image:
 [[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]

Filters:
 [[[ 1  0 -1]
  [ 1  0 -1]
  [ 1  0 -1]]

 [[ 1  1  1]
  [ 0  0  0]
  [-1 -1 -1]]

 [[ 0  1  0]
  [ 1 -4  1]
  [ 0  1  0]]]

Convolution Output:
 [[[ -9.  -4.  13.]
  [-36.  -6.  42.]
  [-39.  -4.  43.]]

 [[-13. -24. -19.]
  [-20. -30. -20.]
  [ 33.  54.  39.]]

 [[  4.   2.  -6.]
  [-10.   0. -16.]
  [-46. -28. -56.]]]
```

4. **WAP to calculate the number of parameters at each layer of a Conv network and finally sum them to present the total no of params for the network. The program must be generic, i.e. the user may provide a different number of input channels, a different number of Conv layers with individual filter sizes as well as varying dense network architectures.**

```
def calculate_conv_params(input_channels, filter_height, filter_width, num_filters):

    # Each filter has weights and a bias
    params = (filter_height * filter_width * input_channels + 1) * num_filters
    return params


def calculate_dense_params(input_units, output_units):

    # Each unit has weights and a bias
    params = (input_units + 1) * output_units
    return params


def main():
    print("Generic Parameter Calculator for ConvNet\n")

    # Input layer configuration
    input_height = int(input("Enter input height: "))
    input_width = int(input("Enter input width: "))
    input_channels = int(input("Enter number of input channels: "))

    # Convolutional layers
    conv_layers = int(input("\nEnter the number of convolutional layers: "))
    total_params = 0
    prev_channels = input_channels
```

```
    for i in range(conv_layers):
        print(f"\nConvolutional Layer {i+1}:")
        filter_height = int(input("Enter filter height: "))
        filter_width = int(input("Enter filter width: "))
        num_filters = int(input("Enter number of filters: "))

        # Calculate parameters for this layer
        conv_params = calculate_conv_params(prev_channels, filter_height, filter_width, num_filters)
        total_params += conv_params

        print(f"Parameters in Convolutional Layer {i+1}: {conv_params}")

        # Update previous channels for the next layer
        prev_channels = num_filters

    # Fully connected layers
    dense_layers = int(input("\nEnter the number of dense layers: "))
    prev_units = int(input("Enter the number of units in the last Conv layer (flattened): "))

    for i in range(dense_layers):
        print(f"\nDense Layer {i+1}:")
        output_units = int(input("Enter number of output units: "))

        # Calculate parameters for this layer
        dense_params = calculate_dense_params(prev_units, output_units)
        total_params += dense_params
```

```python
    for i in range(dense_layers):
        print(f"\nDense Layer {i+1}:")
        output_units = int(input("Enter number of output units: "))

        # Calculate parameters for this layer
        dense_params = calculate_dense_params(prev_units, output_units)
        total_params += dense_params

        print(f"Parameters in Dense Layer {i+1}: {dense_params}")

        # Update previous units for the next layer
        prev_units = output_units

    # Output the total parameters
    print("\n======================================")
    print(f"Total Parameters in the Network: {total_params}")
    print("======================================")

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

Output

```
Enter input height: 32
Enter input width: 32
Enter number of input channels: 3

Enter the number of convolutional layers: 2

Convolutional Layer 1:
Enter filter height: 3
Enter filter width: 3
Enter number of filters: 16
Parameters in Convolutional Layer 1: 448

Convolutional Layer 2:
Enter filter height: 3
Enter filter width: 3
Enter number of filters: 32
Parameters in Convolutional Layer 2: 4640

Enter the number of dense layers: 2
Enter the number of units in the last Conv layer (flattened): 1024

Dense Layer 1:
Enter number of output units: 128
Parameters in Dense Layer 1: 131200

Dense Layer 2:
Enter number of output units: 10
Parameters in Dense Layer 2: 1290

======================================
Total Parameters in the Network: 137578
======================================
```

5. **WAP to implement the POOLING OPERATION of a CNN. Consider a Square image and a square filter size, with variable stride(slide) value and padding. Images should be randomly initialized between (0,255) while the filters should be initialized randomly between (1,1).The user should only provide the image size, filter size, stride value, and padding value**

```python
import numpy as np

def pooling_operation(image, filter_size, stride, padding, mode='max'):

    # Add padding to the image
    if padding > 0:
        image = np.pad(image, ((padding, padding), (padding, padding)), mode='constant', constant_values=0)

    # Get dimensions of the image
    image_h, image_w = image.shape

    # Calculate output dimensions
    output_h = (image_h - filter_size) // stride + 1
    output_w = (image_w - filter_size) // stride + 1

    # Initialize the output
    output = np.zeros((output_h, output_w))

    # Perform pooling
    for i in range(output_h):
        for j in range(output_w):
            # Extract the region of the image
            region = image[i * stride:i * stride + filter_size, j * stride:j * stride + filter_size]
            if mode == 'max':
                output[i, j] = np.max(region)
            elif mode == 'average':
                output[i, j] = np.mean(region)

    return output
```

Output

```
Enter the size of the square image: 5
Enter the size of the square filter: 2
Enter the stride value: 1
Enter the padding value: 1
Enter pooling type ('max' or 'average'): max

Original Image:
 [[194 186  17 210 174]
 [ 93 114 253 117  79]
 [120  42 164  24  42]
 [  2 117  17 126 151]
 [146  99  25 239 116]]

Pooled Image (Max Pooling):
 [[194. 194. 186. 210. 210. 174.]
 [194. 194. 253. 253. 210. 174.]
 [120. 120. 253. 253. 117.  79.]
 [120. 120. 164. 164. 151. 151.]
 [146. 146. 117. 239. 239. 151.]
 [146. 146.  99. 239. 239. 116.]]
```

6. Implement the task of Watermark removal using autoencoders.

```
from keras.layers import Input, Conv2D, UpSampling2D, BatchNormalization, Concatenate
from keras.models import Model
from keras.optimizers import Adam

def create_model(img_x, img_y):
    x = Input(shape=(img_x, img_y, 3))

    # Encoder - compresses the input into a latent representation
    e_conv1 = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    batchnorm_1 = BatchNormalization()(e_conv1)

    e_conv2 = Conv2D(64, (3, 3), activation='relu', padding='same', strides=(2, 2))(batchnorm_
1)  # Downsample with stride
    batchnorm_2 = BatchNormalization()(e_conv2)

    e_conv3 = Conv2D(32, (3, 3), activation='relu', padding='same')(batchnorm_2)
    h = Conv2D(32, (3, 3), activation='relu', padding='same', strides=(2, 2))(e_conv3)  # Downsa
mple with stride
```

```
    # Decoder - reconstructs the input from a latent representation
    d_conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(h)
    up1 = UpSampling2D((2, 2))(d_conv1)

    d_conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(up1)
    up2 = UpSampling2D((2, 2))(d_conv2)

    d_conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(up2)

    r = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(d_conv3)

    model = Model(x, r)
    model.compile(optimizer=Adam(learning_rate=0.0005), loss='mse')
    return model
```

Output:

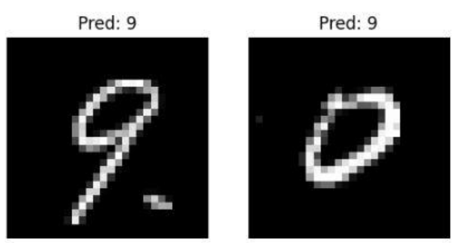
7. **Implement the CNN architecture on the MNIST dataset.**

```
# Load the datasets
df_train = pd.read_csv('/kaggle/input/digit-recognizer/train.csv')
df_test = pd.read_csv('/kaggle/input/digit-recognizer/test.csv')

# Prepare features and labels
df_features = df_train.iloc[:, 1:785]  # Features are the columns from index 1 to 785
df_label = df_train.iloc[:, 0]  # Labels are the first column (0)

# Prepare the test dataset
X_test = df_test.iloc[:, 0:784].values  # Features from test set (flattened)

# Split training data into train and validation sets
X_train, X_cv, y_train, y_cv = train_test_split(df_features, df_label, test_size=0.2, random_state=1212)

# Reshape the data to be 2D arrays (samples, features)
X_train = X_train.values.reshape(-1, 784)
X_cv = X_cv.values.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)

# Normalize the features (scale to range 0-1)
X_train = X_train.astype('float32') / 255.0
X_cv = X_cv.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One-hot encode the labels
num_digits = 10
y_train = keras.utils.to_categorical(y_train, num_digits)
y_cv = keras.utils.to_categorical(y_cv, num_digits)
```

```
# Convolutional layers
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)
x = Dropout(0.25)(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)
x = Dropout(0.25)(x)

# Flatten for fully connected layers
x = Flatten()(x)

# Dense layers
x = Dense(300, activation='relu')(x)
x = Dropout(0.3)(x)

x = Dense(100, activation='relu')(x)
x = Dropout(0.3)(x)

x = Dense(100, activation='relu')(x)
x = Dropout(0.3)(x)

x = Dense(200, activation='relu')(x)
x = Dropout(0.3)(x)

# Output layer
output = Dense(num_digits, activation='softmax')(x)

# Create the model
model = Model(Inp, output)
```

```
# Compile the model
learning_rate = 0.001  # Using a smaller learning rate for stability
optimizer = Adam(learning_rate=learning_rate)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Print model summary
model.summary()

# Set up early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the model
history = model.fit(X_train, y_train,
                    batch_size=100,
                    epochs=20,
                    verbose=2,
                    validation_data=(X_cv, y_cv),
                    callbacks=[early_stopping])
```

Output



8. Implement the CNN architecture on the CIFAR dataset.

```
# Importing the CIFAR-10 dataset from Keras
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

```
# Normalizing
X_train = X_train/255
X_test = X_test/255

# One-Hot-Encoding
y_train_en = to_categorical(y_train,10)
y_test_en = to_categorical(y_test,10)
```

```python
# Model_3 with Batch Normalization
model = Sequential()
model.add(Conv2D(264,(4,4),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(4,4),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(512,(4,4),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(4,4),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(128,(4,4),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(4,4),input_shape=(32,32,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.35))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

9. Implement the LeNet-5 CNN using tensor flow.

```python
# LeNet-5 model
class LeNet(Sequential):
  def __init__(self, input_shape, nb_classes):
    super().__init__()

    self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=input_shape, padding="same"))
    self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
    self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
    self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
    self.add(Flatten())
    self.add(Dense(120, activation='tanh'))
    self.add(Dense(84, activation='tanh'))
    self.add(Dense(nb_classes, activation='softmax'))

    self.compile(optimizer='adam',
                 loss=categorical_crossentropy,
                 metrics=['accuracy'])
```

```
Model: "le_net"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

10. Implement the Alex Net CNN using tensor flow.

```
class AlexNet(Sequential):
  def __init__(self, input_shape, num_classes):
    super().__init__()

    self.add(Conv2D(96, kernel_size=(11,11), strides= 4,
                    padding= 'valid', activation= 'relu',
                    input_shape= input_shape, kernel_initializer= 'he_normal'))
    self.add(MaxPooling2D(pool_size=(3,3), strides= (2,2),
                          padding= 'valid', data_format= None))

    self.add(Conv2D(256, kernel_size=(5,5), strides= 1,
                    padding= 'same', activation= 'relu',
                    kernel_initializer= 'he_normal'))
    self.add(MaxPooling2D(pool_size=(3,3), strides= (2,2),
                          padding= 'valid', data_format= None))

    self.add(Conv2D(384, kernel_size=(3,3), strides= 1,
                    padding= 'same', activation= 'relu',
                    kernel_initializer= 'he_normal'))

    self.add(Conv2D(384, kernel_size=(3,3), strides= 1,
                    padding= 'same', activation= 'relu',
                    kernel_initializer= 'he_normal'))

    self.add(Conv2D(256, kernel_size=(3,3), strides= 1,
                    padding= 'same', activation= 'relu',
                    kernel_initializer= 'he_normal'))

    self.add(MaxPooling2D(pool_size=(3,3), strides= (2,2),
                          padding= 'valid', data_format= None))
```

```
self.add(Flatten())
self.add(Dense(4096, activation= 'relu'))
self.add(Dense(4096, activation= 'relu'))
self.add(Dense(1000, activation= 'relu'))
self.add(Dense(num_classes, activation= 'softmax'))

self.compile(optimizer= tf.keras.optimizers.Adam(0.001),
             loss='categorical_crossentropy',
             metrics=['accuracy'])
```

```
Model: "alex_net"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 384)       885120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 384)       1327488
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 256)       884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              37752832
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2002
=================================================================
Total params: 62,380,346
Trainable params: 62,380,346
Non-trainable params: 0
```