Como é uma rede neural de detecção de rosto?

Chi-Feng Wang
24 de julho de 2018 · 4 min. De leitura

No meu último post , explorei o modelo MTCNN (Rede Convolucional em Cascata Multitarefas), usando-o para detectar rostos com minha webcam. Neste post, examinarei a estrutura da rede neural.

O modelo MTCNN consiste em 3 redes separadas: a P-Net, a R-Net e a O-Net:

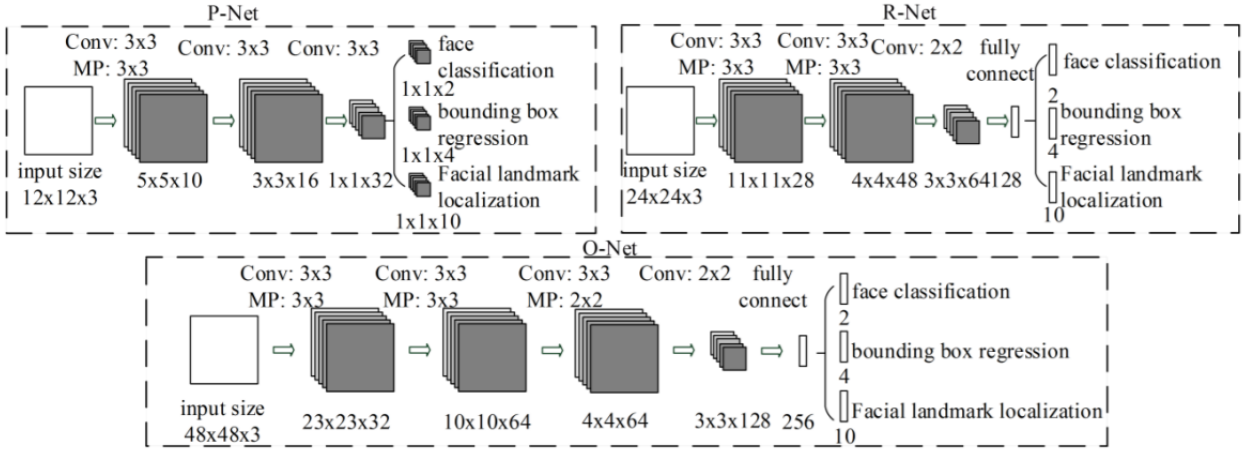


Imagem 1: Estrutura MTCNN // Fonte

Para cada imagem que transmitimos, a rede cria uma pirâmide de imagens: isto é, cria várias cópias dessa imagem em tamanhos diferentes.

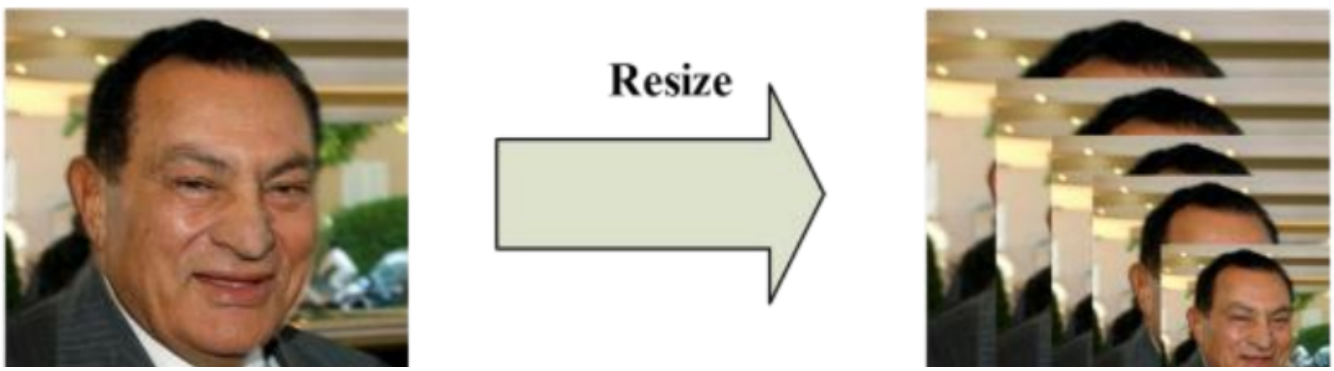


Imagem 2: Pirâmide de Imagem // Fonte

Na P-Net, para cada imagem em escala, um kernel 12x12 percorre a imagem, procurando um rosto. Na imagem abaixo, o quadrado vermelho representa o núcleo, que se move lentamente pela imagem, procurando um rosto.

Imagem 3: kernel 12x12 no canto superior direito. Depois de digitalizar este canto, ele se desloca lateralmente (ou para baixo) em 1 pixel e continua fazendo isso até passar pela imagem inteira.

Dentro de cada um desses kernels 12x12, são realizadas três convoluções (se você não souber o que são convoluções, consulte meu outro artigo ou este site) com kernels 3x3. Após cada camada de convolução, uma camada preliminar é implementada (quando você multiplica cada pixel negativo com um determinado número 'alfa'. 'Alfa' deve ser determinado por meio de treinamento). Além disso, uma camada maxpool é inserida após a primeira camada de prelu (maxpool remove todos os outros pixels, deixando apenas o maior na vizinhança).

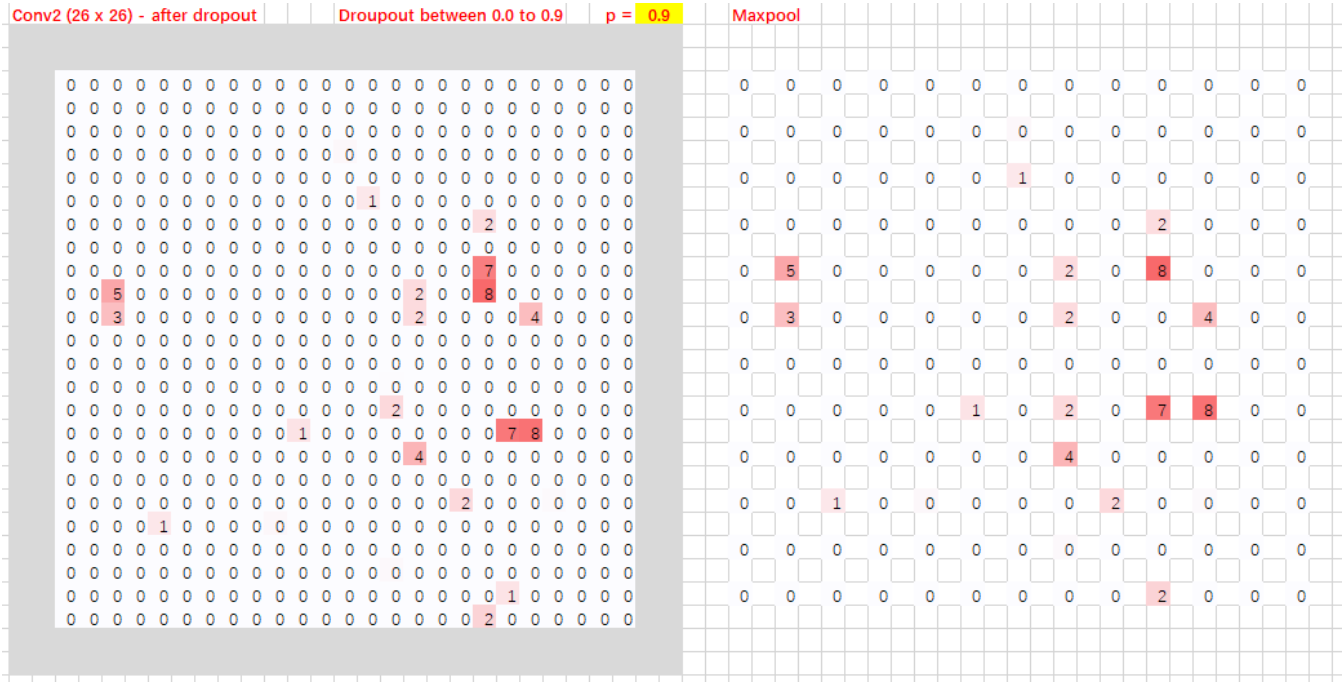
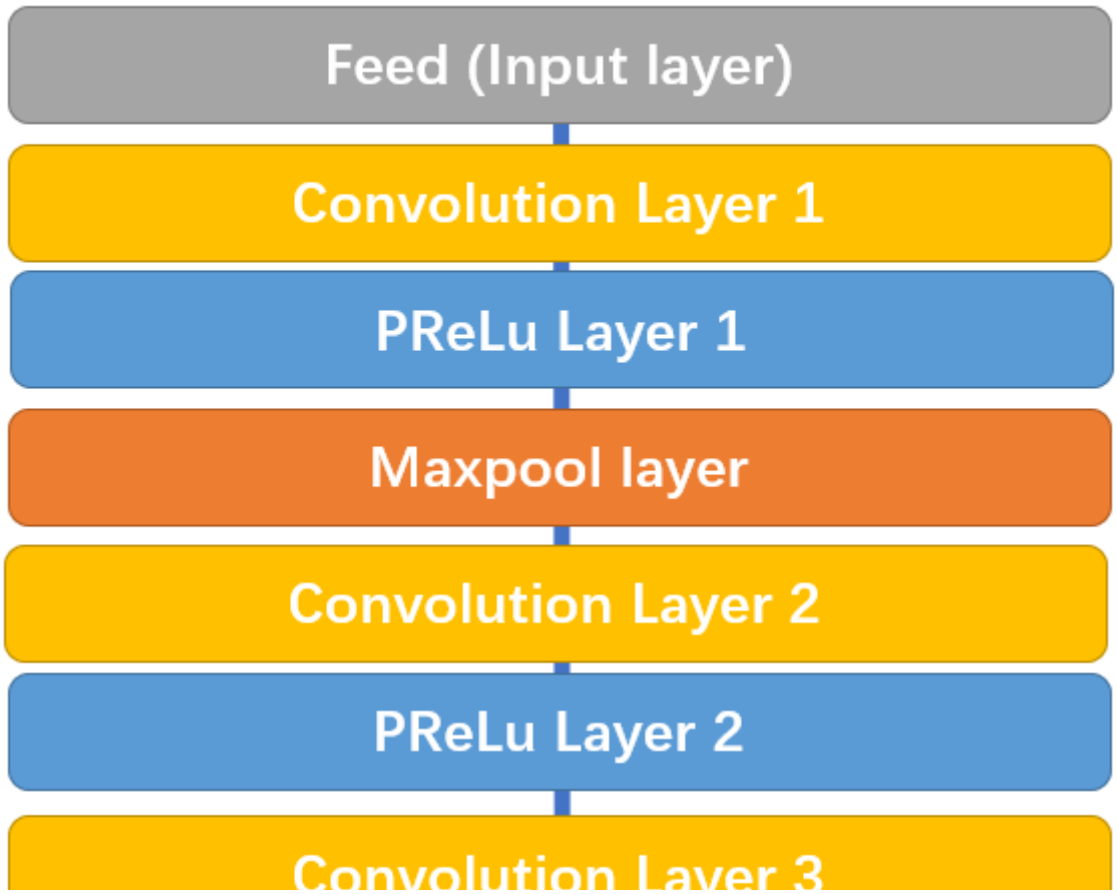


Imagem 4: Max-pool // Origem

Após a terceira camada de convolução, a rede se divide em duas camadas. As ativações da terceira camada são passadas para duas camadas de convolução separadas e uma camada softmax após uma dessas camadas de convolução (o softmax atribui probabilidades decimais a todos os resultados e as probabilidades somam 1. Nesse caso, gera 2 probabilidades: a probabilidade de que não **é** um rosto na área e a probabilidade de que lá **não é** um rosto).



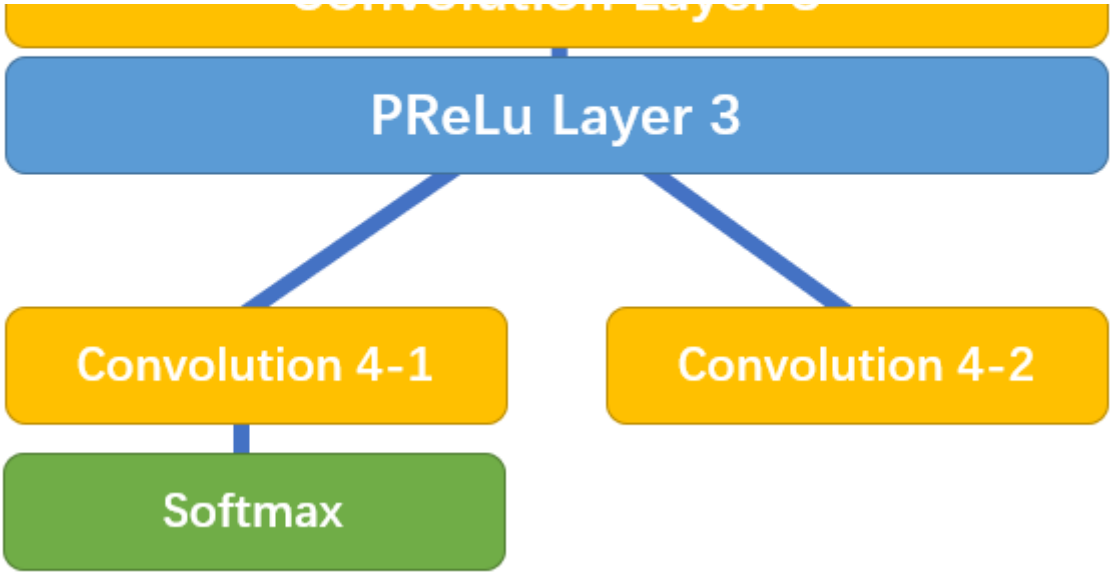


Imagem 5: P-Net

A convolução 4–1 gera a probabilidade de uma face estar em cada caixa delimitadora e a convolução 4–2 gera as coordenadas das caixas delimitadoras.

Dando uma olhada no mtcnn.py, você mostrará a estrutura da P-Net:

```
classe PNet (Rede):

def _config (self):

layer_factory = LayerFactory (self)

layer_factory.new_feed (name = 'data', layer_shape = (None, None,
None, 3))

layer_factory.new_conv (name = 'conv1', kernel_size = (3, 3),
channels_output = 10, stride_size = (1, 1),

padding = 'VALID', relu = False)

layer_factory.new_prelu (name = 'prelu1')

layer_factory.new_max_pool (name = 'pool1', kernel_size = (2, 2),
stride_size = (2, 2))

layer_factory.new_conv (name = 'conv2', kernel_size = (3, 3),
channels_output = 16, stride_size = (1, 1),

padding = 'VALID ', relu = False)

layer_factory.new_prelu (name =' prelu2 ')

layer_factory.new_conv (name =' conv3 ', kernel_size = (3, 3),
channels_output = 32, stride_size = (1, 1),
```

```
    padding = 'VÁLIDO', relu = False)

    layer_factory.new_prelu (name = 'prelu3')

    layer_factory.new_conv (name = 'conv4-1', kernel_size = (1, 1),
    channels_output = 2, stride_size = (1, 1), relu = False)

    layer_factory.new_softmax ( nome = 'prob1', eixo = 3)

    layer_factory.new_conv (name = 'conv4-2', kernel_size = (1, 1),
    channels_output = 4, stride_size = (1, 1),

    input_layer_name = 'prelu3', relu = Falso)
```

O R-Net tem uma estrutura semelhante, mas com ainda mais camadas. Ele pega as caixas delimitadoras da P-Net como entradas e refina suas coordenadas.

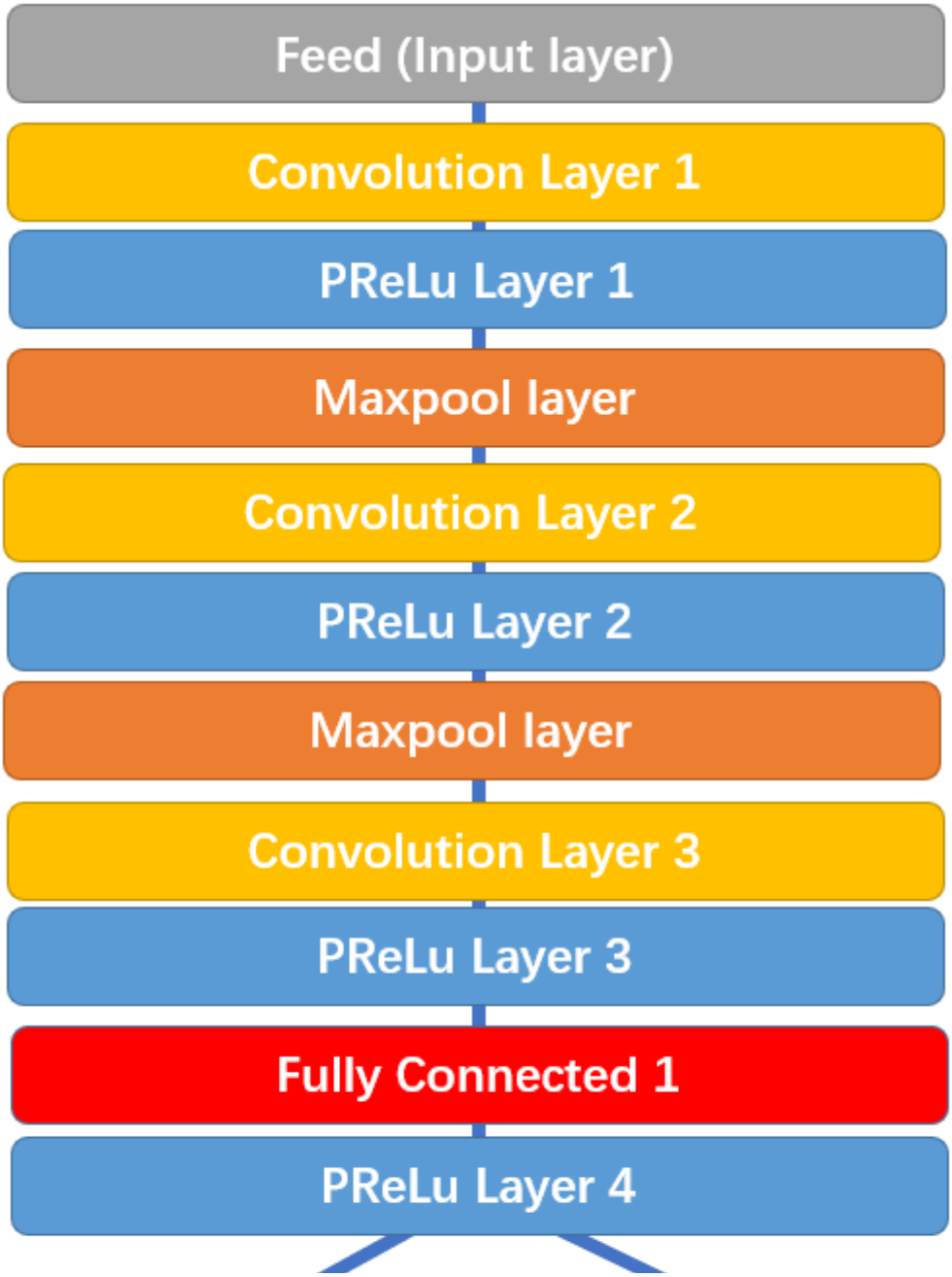


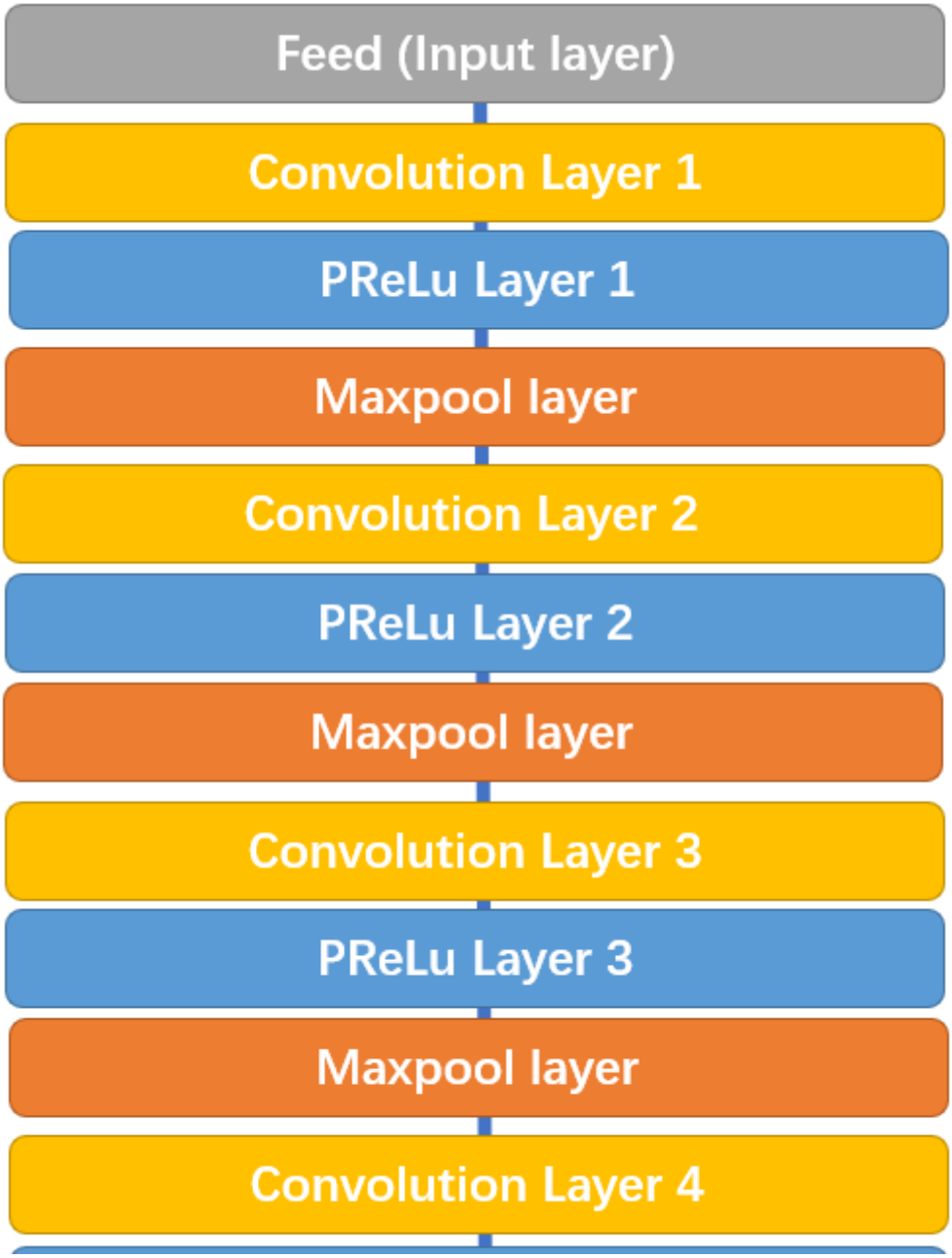
Imagem 6: R-Net

Da mesma forma, o R-Net se divide em duas camadas no final, fornecendo duas saídas: as coordenadas das novas caixas delimitadoras e a confiança da máquina em cada caixa delimitadora. Novamente, mtcnn.py inclui a estrutura da R-Net:

```
classe RNet (Rede):

def _config (self):

layer_factory = LayerFactory (self)

layer_factory.new_feed (name = 'data', layer_shape = (None, 24, 24,
3))

layer_factory.new_conv (name = 'conv1', kernel_size = (3, 3),
channels_output = 28, stride_size = (1, 1),

padding = 'VALID', relu = False)

layer_factory.new_prelu (name = 'prelu1')

layer_factory.new_max_pool (name = 'pool1', kernel_size = (3, 3),
stride_size = (2, 2))

layer_factory.new_conv (name = 'conv2', kernel_size = (3, 3),
channels_output = 48, stride_size = (1, 1),

padding = 'VALID ', relu = False)

layer_factory.new_prelu (name =' prelu2 ')

layer_factory.new_max_pool (name =' pool2 ', kernel_size = (3, 3),
stride_size = (2, 2), padding =' VALID ')

layer_factory.new_conv (name = 'conv3', kernel_size = (2, 2),
channels_output = 64, stride_size = (1, 1),

padding = 'VALID', relu = False)
```

```
layer_factory.new_prelu (name = 'prelu3')

layer_factory.new_fully_connected (name = 'fc1', output_count = 128,
relu = False)

layer_factory.new_prelu (name = 'prelu4')

layer_factory.new_fully_connected (nome = 'fc2-1', output_count = 2,
relu = False)

layer_factory. new_softmax (nome = 'prob1', eixo = 1)

layer_factory.new_fully_connected (nome = 'fc2-2', output_count = 4,
relu = False, input_layer_name = 'prelu4')
```

Finalmente, o O-Net toma as caixas delimitadoras do R-Net como entradas e marca as coordenadas dos pontos de referência faciais.



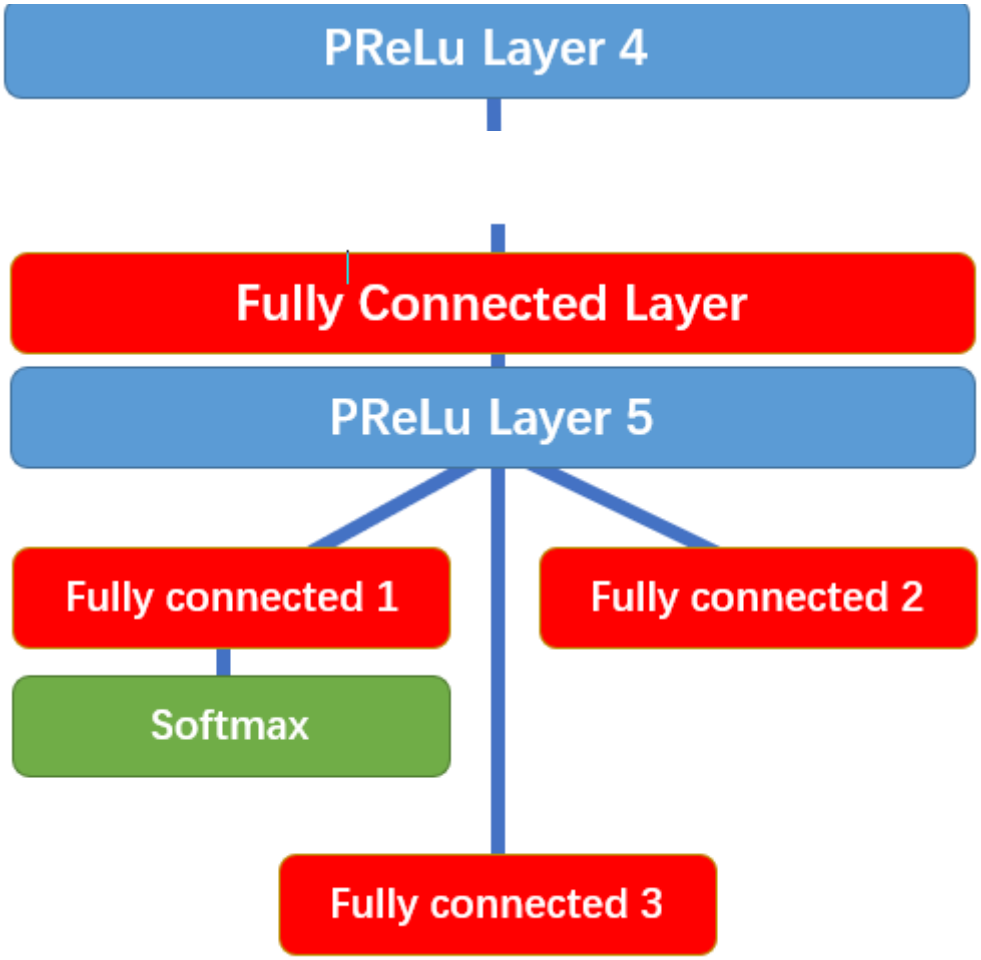


Imagem 7: O-Net

O-Net se divide em 3 camadas no final, fornecendo 3 saídas diferentes: a probabilidade de um rosto estar na caixa, as coordenadas da caixa delimitadora e as coordenadas dos pontos de referência faciais (localizações dos olhos, nariz e boca). Aqui está o código para O-Net:

```
classe ONet (Rede):

def _config (self):

layer_factory = LayerFactory (self)

layer_factory.new_feed (name = 'data', layer_shape = (None, 48, 48,
3))

layer_factory.new_conv (name = 'conv1', kernel_size = (3, 3),
channels_output = 32, stride_size = (1, 1),

padding = 'VALID', relu = False)

layer_factory.new_prelu (name = 'prelu1')
```

```
layer_factory.new_max_pool (name = 'pool1', kernel_size = (3, 3),
stride_size = (2, 2))

layer_factory.new_conv (name = 'conv2', kernel_size = (3, 3),
channels_output = 64, stride_size = (1, 1),

padding = 'VALID ', relu = False)

layer_factory.new_prelu (name =' prelu2 ')

layer_factory.new_max_pool (name =' pool2 ', kernel_size = (3, 3),
stride_size = (2, 2), padding =' VALID ')

layer_factory.new_conv (name = 'conv3', kernel_size = (3, 3),
channels_output = 64, stride_size = (1, 1),

padding = 'VALID', relu = False)

layer_factory.new_prelu (name = 'prelu3')

layer_factory.new_max_pool (name = 'pool3', kernel_size = (2, 2),
stride_size = (2, 2))

layer_factory.new_conv (nome = 'conv4', kernel_size = (2, 2),
channels_output = 128, stride_size = (1, 1),

padding = 'VALID', relu = False)

layer_factory.new_prelu (name = 'prelu4')

layer_factory.new_fully_connected (name = 'fc1', output_count = 256,
relu = False)

layer_factory.new_prelu (name = 'prelu5')

layer_factory.new_fully_connected (name = 'fc2-1', output_count = 2,
relu = False)

layer_factory.new_softmax (nome = 'prob1', eixo = 1)

layer_factory.new_fully_connected (name = 'fc2-2', output_count = 4,
relu = False, input_layer_name = 'prelu5')

layer_factory.new_fully_connected (nome = 'fc2-3', output_count =
10, relu = False, input_layer_name = 'prelu5 ')
```

Observe que todo o código para P-Net, R-Net e O-Net importam uma classe chamada “LayerFactory”. Em essência, o LayerFactory é uma classe - criada pelos fabricantes deste modelo - para gerar camadas com configurações específicas. Para mais informações, você pode conferir o layer_factory.py.

. . .

Clique aqui para ler sobre a implementação do modelo MTCNN!

Clique aqui para ler sobre como o modelo MTCNN funciona!

. . .

Faça o download do documento e recursos do MTCNN aqui:

Download do Github: https://github.com/ipazc/mtcnn

Artigo de pesquisa: http://arxiv.org/abs/1604.02878

Inteligência artificial Redes neurais Detecção de rosto Mtcnn Aprendizagem Profunda