

# Como funciona um programa de detecção de rosto? (Usando redes neurais)

Guia para iniciantes de detecção de rosto com redes neurais



Chi-Feng Wang

27 de julho de 2018 · 10 min de leitura

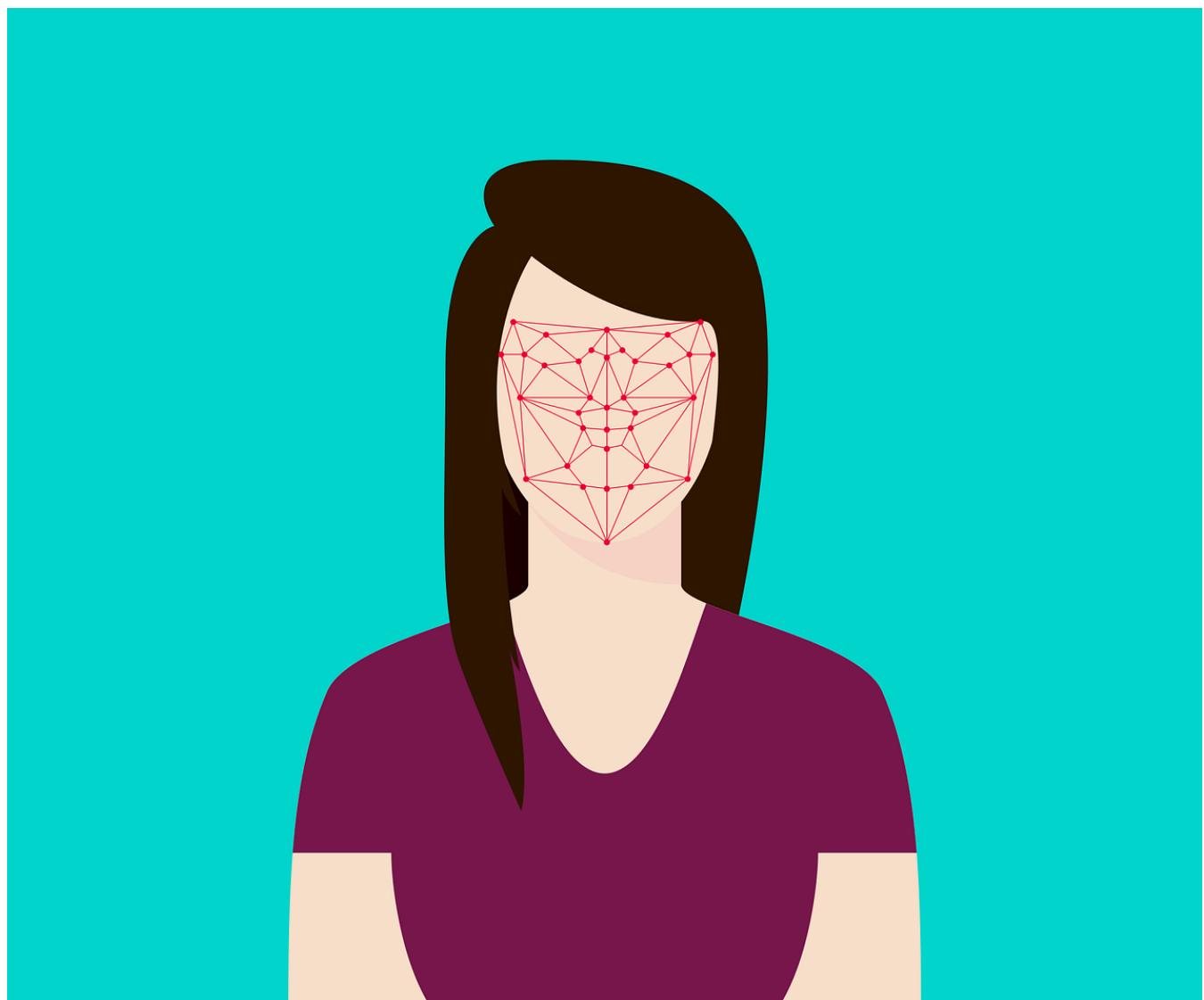


Imagen da capa: Detecção de Rosto // Fonte

Recentemente, tenho brincado com um modelo MTCNN (Rede Convolucional em Cascata Multitarefas) para detecção de rosto. Esse modelo possui três redes convolucionais (P-Net, R-Net e O-Net) e é capaz de superar muitos benchmarks de detecção de rosto, mantendo o desempenho em tempo real. Mas como exatamente isso funciona?

*Nota: Se você quiser um exemplo concreto de como processar uma rede neural de detecção de rosto, anexamos os links de download do modelo MTCNN abaixo. Após o download, abra ./mtcnn/mtcnn.py e role até a função detect\_faces. Essa é a função que você chamaria ao implementar esse modelo; portanto, passar por essa função forneceria uma noção de como o programa calcula e restringe as coordenadas da caixa delimitadora e dos recursos faciais. No entanto, não decifrarei o código linha por linha: não apenas tornaria este artigo desnecessariamente longo, mas também seria contraproducente para a maioria dos leitores, pois muitos dos parâmetros no código são adequados apenas para este modelo específico.*

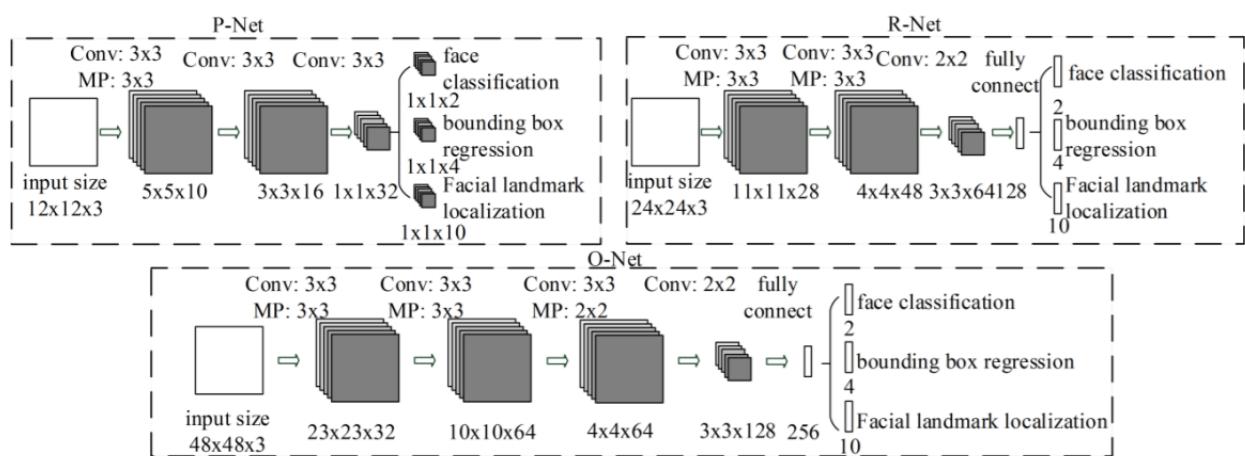


Imagen 1: Estrutura MTCNN // Fonte

• • •

## Estágio 1:

Obviamente, a primeira coisa a fazer seria passar uma imagem para o programa. Neste modelo, queremos criar uma **pirâmide de imagens**, a fim de detectar faces de todos os tamanhos diferentes. Em outras palavras, queremos criar cópias diferentes da mesma imagem em tamanhos diferentes para procurar rostos de tamanhos diferentes na imagem.

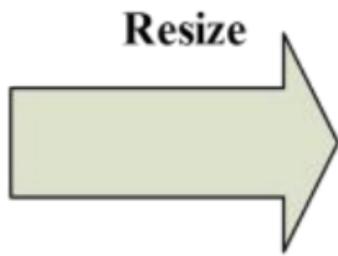
**Test image****Image pyramid**

Imagen 2: Pirâmide de Imagem // Fonte

Para cada cópia em escala, temos um **kernel**  $12 \times 12$  estágio 1 que percorre todas as partes da imagem, procurando rostos. Começa no canto superior esquerdo, uma seção da imagem de  $(0,0)$  a  $(12,12)$ . Essa parte da imagem é passada para o P-Net, que retorna as coordenadas de uma caixa delimitadora, se notar uma face. Em seguida, repetiria esse processo com as seções  $(0 + 2a, 0 + 2b)$  para  $(12 + 2a, 12 + 2b)$ , deslocando os  $12 \times 12$  kernel 2 pixels para a direita ou para baixo de cada vez. O deslocamento de 2 pixels é conhecido como **passo**, ou quantos pixels o kernel move sempre.

Ter um passo 2 ajuda a reduzir a complexidade da computação sem sacrificar significativamente a precisão. Como os rostos na maioria das imagens são significativamente maiores que dois pixels, é altamente improvável que o kernel perca um rosto apenas porque mudou 2 pixels. Ao mesmo tempo, seu computador (ou qualquer máquina que esteja executando esse código) terá um quarto da quantidade de operações para calcular, tornando o programa mais rápido e com menos memória.

A única desvantagem é que temos que recalcular todos os índices relacionados ao avanço. Por exemplo, se o kernel detectou uma face após mover um passo para a direita, o índice de saída nos diria que o canto superior esquerdo desse kernel está em  $(1,0)$ . No entanto, como a passada é 2, temos que multiplicar o índice por 2 para obter a coordenada correta:  $(2,0)$ .

Cada kernel seria menor em relação a uma imagem grande, portanto, seria possível encontrar faces menores na imagem em escala maior. Da mesma forma, o kernel seria maior em relação a uma imagem de tamanho menor, portanto, seria capaz de encontrar faces maiores na imagem em menor escala.

**MIT CNN Process Images**

Vídeo: os núcleos podem encontrar rostos menores em imagens maiores e rostos maiores em imagens menores.

Depois de passar a imagem, precisamos criar várias cópias em escala da imagem e passá-la para a primeira rede neural - P-Net - e reunir sua saída.

<b>Kernel Coordinates</b>	<b>Bounding box: x1</b>	<b>Bounding box: y1</b>	<b>Bounding box: x2</b>	<b>Bounding box: y2</b>	<b>Confidence</b>
(0,5)	0.50	0.25	0.80	0.58	0.98
(0,6)	0.45	0.17	0.75	0.49	0.96
(0,7)	0.52	0.08	0.73	0.41	0.94
(1,4)	0.42	0.34	0.67	0.66	0.92
(1,8)	0.40	0.01	0.66	0.32	0.96
(3,5)	0.32	0.22	0.49	0.59	0.88
(3,6)	0.25	0.20	0.52	0.53	0.91
(6,6)	0.01	0.18	0.25	0.50	0.89
(6,8)	0.02	0.00	0.22	0.33	0.90

Imagen 3: Exemplo de saída para P-Net. Observe que a saída real tem 4 dimensões, mas por simplicidade, eu a combinei em uma matriz bidimensional. Além disso, as coordenadas para as caixas delimitadoras são valores entre 0 e 1: (0,0) seria o canto superior esquerdo do kernel, enquanto (1,1) seria o canto inferior direito do kernel.

Os pesos e preconceitos do P-Net foram treinados para gerar uma caixa delimitadora relativamente precisa para cada kernel 12 x 12. No entanto, a rede está mais confiante em algumas caixas em comparação com outras. Portanto, precisamos analisar a saída da P-Net para obter uma lista de níveis de confiança para cada caixa delimitadora e excluir as caixas com menor confiança (ou seja, as caixas que a rede não tem certeza de conter uma face)

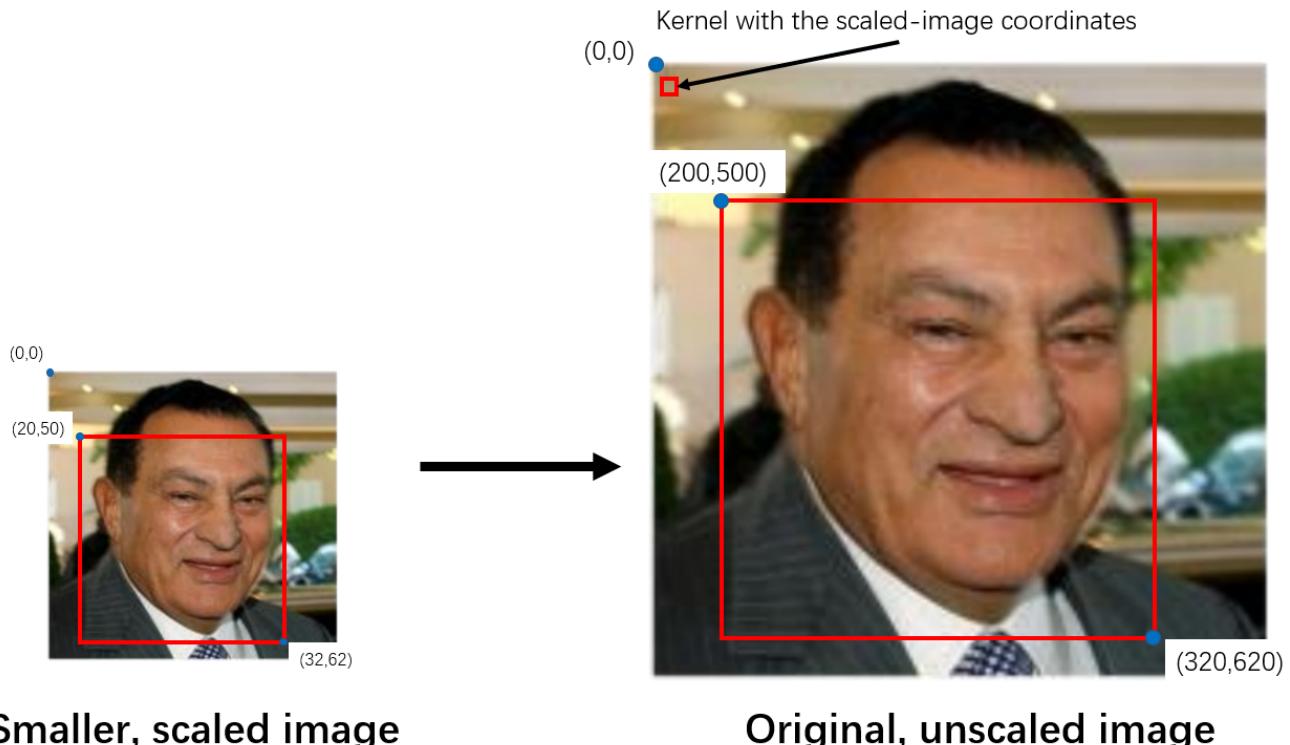


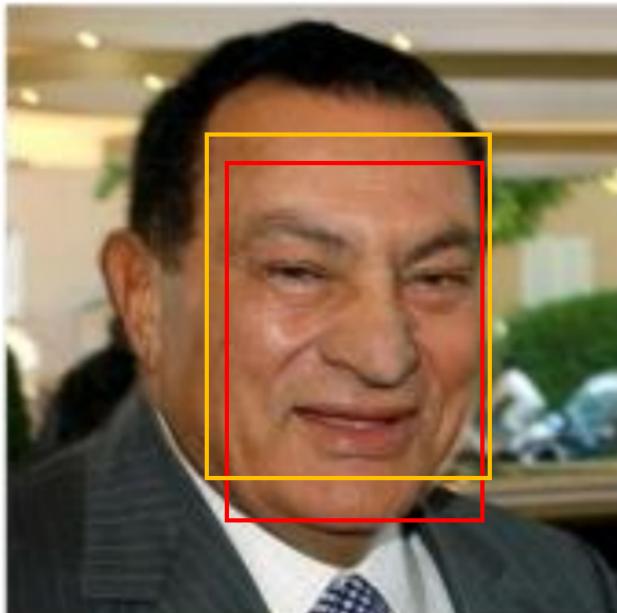
Imagen 4: Padronizando as coordenadas do kernel multiplicando-o pela escala

Depois de selecionar as caixas com maior confiança, teremos que padronizar o sistema de coordenadas, convertendo todos os sistemas de coordenadas para o da imagem real e não escalada. Como a maioria dos kernels está em uma imagem reduzida, suas coordenadas serão baseadas na imagem menor.

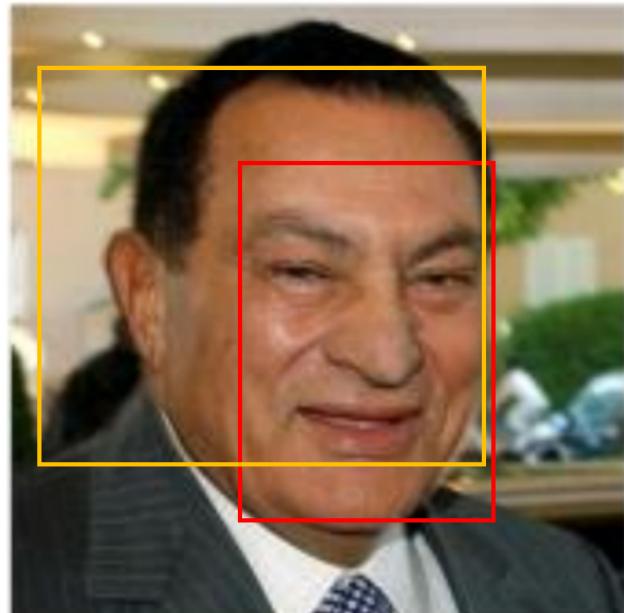
No entanto, ainda existem muitas caixas delimitadoras, e muitas delas se sobrepõem. A **supressão não máxima**, ou NMS, é um método que reduz o número de caixas delimitadoras.

Nesse programa em particular, o NMS é conduzido primeiro classificando as caixas delimitadoras (e seus respectivos 12 x 12 núcleos) por sua confiança ou pontuação. Em alguns outros modelos, o NMS leva a maior caixa delimitadora, em vez daquela em que a rede está mais confiante.

Posteriormente, calculamos a área de cada um dos núcleos, bem como a área sobreposta entre cada núcleo e o núcleo com a pontuação mais alta. Os kernels que se sobrepõem muito ao kernel com alta pontuação são excluídos. Por fim, o NMS retorna uma lista das caixas delimitadoras "sobreviventes".



Large overlap, yellow box gets deleted



Small overlap, yellow box remains

Imagen 5: Supressão não máxima

Conduzimos o NMS uma vez para cada imagem em escala e, mais uma vez, com todos os kernels sobreviventes de cada escala. Isso elimina as caixas delimitadoras redundantes, permitindo restringir nossa pesquisa a uma caixa precisa por face.

Por que não podemos simplesmente escolher a caixa com a maior confiança e excluir todo o resto? Existe apenas uma face na imagem acima. No entanto, pode haver mais de um rosto em outras imagens. Nesse caso, acabaríamos excluindo todas as caixas delimitadoras das outras faces.

Depois, convertemos as coordenadas da caixa delimitadora em coordenadas da imagem real. No momento, as coordenadas de cada caixa delimitadora são um valor entre 0 e 1, com (0,0) no canto superior esquerdo do kernel 12 x 12 e (1,1) no canto inferior direito (consulte a tabela acima). Ao multiplicar as coordenadas pela largura e altura reais da imagem, podemos converter as coordenadas da caixa delimitadora nas coordenadas padrão da imagem em tamanho real.

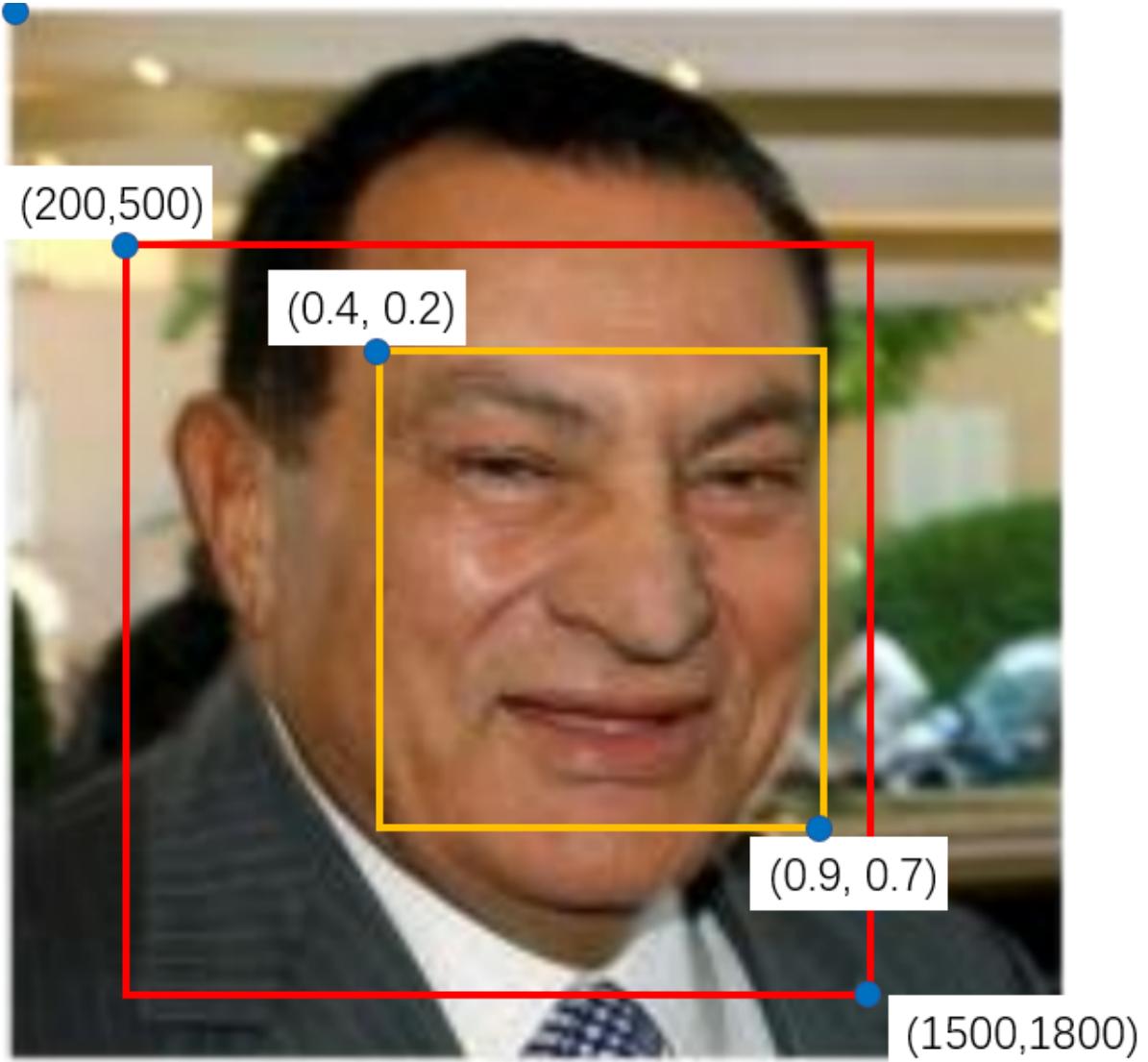


Imagen 6: Aqui, a caixa vermelha é o kernel  $12 \times 12$ , enquanto a caixa amarela é a caixa delimitadora dentro dele.

Nesta imagem, a caixa vermelha representa o kernel  $24 \times 24$ , redimensionado para a imagem original. Podemos calcular a largura e a altura do kernel:  $1500 - 200 = 300$ ,  $1800 - 500 = 300$  (Observe como a largura e a altura não são necessariamente 12. Isso é porque estamos usando as coordenadas do kernel no original. A largura e a altura que chegamos aqui são a largura e a altura do kernel quando redimensionadas para o tamanho original.) Depois, multiplicamos as coordenadas da caixa delimitadora por 300:  $0.4 \times 300 = 120$ ,  $0.2 \times 300 = 60$ ,  $0.9 \times 300 = 270$ ,  $0.7 \times 300 = 210$ . Por fim, adicionamos a coordenada superior esquerda do kernel para obter as coordenadas da caixa delimitadora:  $(200 + 120, 500 + 60)$  e  $(200 + 270, 500 + 210)$  ou  $(320, 560)$  e  $(470, 710)$ .

Como as caixas delimitadoras podem não ser quadradas, modificamos as caixas delimitadoras para um quadrado alongando os lados mais curtos (se a largura for

menor que a altura, expandimos para o lado; se a altura for menor que a largura, expandimos verticalmente).

Por fim, salvamos as coordenadas das caixas delimitadoras e passamos para o estágio 2.

• • •

## Etapa 2:

Às vezes, uma imagem pode conter apenas uma parte do rosto espiando pela lateral do quadro. Nesse caso, a rede pode retornar uma caixa delimitadora parcialmente fora do quadro, como o rosto de Paul McCartney na foto abaixo:

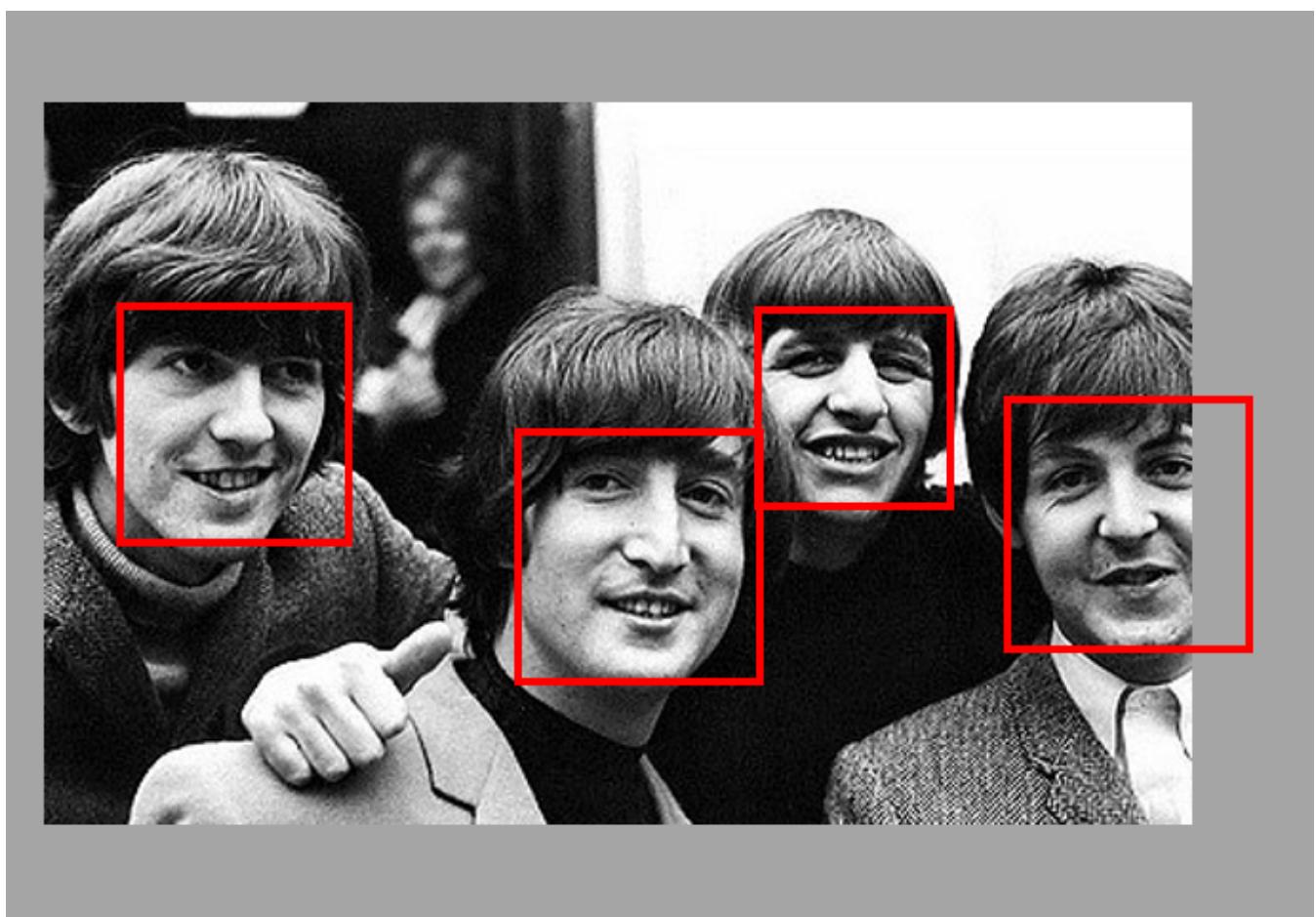


Imagen 7: Os Beatles e suas caixas delimitadoras. A caixa de Paul McCartney está fora dos limites e requer preenchimento. // Origem

Para cada caixa delimitadora, criamos uma matriz do mesmo tamanho e copiamos os valores de pixel (a imagem na caixa delimitadora) para a nova matriz. Se a caixa delimitadora estiver fora dos limites, copiaremos apenas a parte da imagem na caixa delimitadora para a nova matriz e preencheremos todo o resto com um 0. Na imagem

acima, a nova matriz para o rosto de McCartney teria valores de pixel em no lado esquerdo da caixa e várias colunas de 0 perto da borda direita. Esse processo de preenchimento de matrizes com 0s é chamado de **preenchimento**.

Depois de preencher as matrizes da caixa delimitadora, as redimensionamos para 24 x 24 pixels e normalizamos para valores entre -1 e 1. Atualmente, os valores de pixel estão entre 0 e 255 (valores RGB). Subtraindo cada valor de pixel pela metade de 255 (127,5) e dividindo por 127,5, podemos manter seus valores entre -1 e 1.

Agora que temos inúmeras matrizes de imagem 24 x 24 (o número de caixas delimitadoras que sobreviveram ao Estágio 1, uma vez que cada uma delas foi redimensionada e normalizada nesses kernels), podemos alimentá-las no R-Net e coletar sua saída.

O resultado da R-Net é semelhante ao da P-Net: inclui as coordenadas das novas caixas delimitadoras mais precisas, bem como o nível de confiança de cada uma dessas caixas delimitadoras.

Mais uma vez, nos livramos das caixas com menor confiança e executamos o NMS em todas as caixas para eliminar ainda mais as caixas redundantes. Como as coordenadas dessas novas caixas delimitadoras são baseadas nas caixas delimitadoras da P-Net, precisamos convertê-las nas coordenadas padrão.

Após padronizar as coordenadas, remodelamos as caixas delimitadoras para um quadrado a ser passado para o O-Net.

• • •

### **Etapa 3:**

Antes de podermos passar nas caixas delimitadoras da R-Net, precisamos primeiro preencher todas as caixas que estão fora dos limites. Depois de redimensionar as caixas para 48 x 48 pixels, podemos passar as caixas delimitadoras para o O-Net.

As saídas do O-Net são ligeiramente diferentes das do P-Net e R-Net. O-Net fornece 3 saídas: as coordenadas da caixa delimitadora (fora de [0]), as coordenadas dos 5 marcos faciais (saída [1]) e o nível de confiança de cada caixa (saída [2]).

Mais uma vez, eliminamos as caixas com níveis de confiança mais baixos e padronizamos as coordenadas da caixa delimitadora e as coordenadas do marco facial. Finalmente, nós os executamos no último NMS. Nesse ponto, deve haver apenas uma caixa delimitadora para cada rosto na imagem.

• • •

## Entregando Resultados:

O último passo é empacotar todas as informações em um dicionário com três chaves: 'caixa', 'confiança' e 'pontos-chave'. 'caixa' contém as coordenadas da caixa delimitadora, 'confiança' contém o nível de confiança da rede para cada caixa e 'pontos-chave' contém as coordenadas de cada ponto de referência facial (olhos, nariz e pontos finais da boca).

Quando queremos implementar esse modelo em nosso próprio código, tudo o que precisamos fazer é chamar `detect_faces` e receberíamos esse dicionário com todas as coordenadas necessárias para desenhar as caixas delimitadoras e marcar os recursos faciais.

• • •

## Aqui está um breve resumo de todo o processo:

### Estágio 1:

1. Passe na imagem
2. Crie várias cópias em escala da imagem
3. Alimente imagens em escala no P-Net
4. Reúna a saída P-Net
5. Excluir caixas delimitadoras com baixa confiança
6. Converta coordenadas de 12 x 12 do kernel em coordenadas de "imagem não escalada"
7. Supressão não máxima para kernels em cada imagem dimensionada

8. Supressão não máxima para todos os kernels
9. Converter coordenadas da caixa delimitadora em coordenadas de “imagem não dimensionada”
10. Remodelar caixas delimitadoras para quadrado

**Etapa 2:**

1. Pad fora de caixas de limite
2. Alimente imagens em escala no R-Net
3. Reúna a saída R-Net
4. Excluir caixas delimitadoras com baixa confiança
5. Supressão não máxima para todas as caixas
6. Converter coordenadas da caixa delimitadora em coordenadas de “imagem não dimensionada”
7. Remodelar caixas delimitadoras para quadrado

**Etapa 3:**

1. Pad fora de caixas de limite
2. Alimente imagens em escala no O-Net
3. Reunir saída O-Net
4. Excluir caixas delimitadoras com baixa confiança
5. Converta as coordenadas da caixa delimitadora e do marco facial em coordenadas de “imagem não dimensionada”
6. Supressão não máxima para todas as caixas

**Entregando Resultados:**

1. Empacote todas as coordenadas e níveis de confiança em um dicionário
2. Retornar o dicionário

• • •

## Considerações finais:

Este modelo é bastante diferente dos que encontrei antes. Não é apenas uma simples rede neural: utiliza algumas técnicas interessantes para obter alta precisão com menos tempo de execução.

Baixa complexidade computacional resulta em um tempo de execução rápido. Para obter desempenho em tempo real, ele usa um passo de 2, reduzindo o número de operações para um quarto do original. Ele também não começa a encontrar pontos de referência faciais até a última rede (O-Net), depois que as coordenadas da caixa delimitadora foram calibradas, o que restringe as coordenadas dos recursos faciais antes mesmo de iniciar seus cálculos. Isso torna o programa muito mais rápido, pois só precisa encontrar pontos de referência faciais nas poucas caixas que passam pelo O-Net.

Alta precisão é alcançada com uma rede neural profunda. Ter três redes - cada uma com várias camadas - permite maior precisão, pois cada rede pode ajustar os resultados da anterior. Além disso, este modelo emprega uma pirâmide de imagem para encontrar rostos grandes e pequenos. Embora isso possa produzir uma quantidade impressionante de dados, o NMS, assim como o R-Net e o O-Net, ajudam a rejeitar um grande número de caixas delimitadoras.

Olhando para o alto desempenho deste modelo, não posso deixar de me perguntar para que mais podemos utilizar esse modelo. Seria capaz de reconhecer certos animais? Certos carros? Pode ser ajustado para reconhecimento facial e detecção facial? Esse modelo - e suas aplicações - nos deu inúmeras aplicações para uso futuro.

• • •

- Clique aqui para ler sobre a implementação do modelo MTCNN!
- Clique aqui para ler sobre a estrutura de rede do modelo MTCNN!

• • •

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>

Graças a Chamin Nalinda .

Inteligência artificial

Redes neurais

Detecção de rosto

Pirâmide de imagem

Rede Convolucional

Sobre a Ajuda Jurídica