

Real Time Bare Skinned Images Filtering Using CNN

Pratik Jain
Tina Shah
Kshitija Shah

Guide: Prof. Jaya Gupta

Introduction



Skin colour detection has been used in numerous computer vision applications like face detection, nudity recognition, hand gesture detection and person identification. Skin colour detection is a challenging task as the skin colour in an image is sensitive to various factors like illumination, camera characteristics, ethnicity, individual characteristics such as age, sex and body parts and other factors like makeup, hairstyle and glasses. All these factors affect appearance of skin colour. Another problem is that there is a significant overlap between the skin and non-skin pixels. However when these techniques are used in real-time, it is crucial to follow time deadlines and memory constraints. Sometimes, accuracy may need to be sacrificed when the skin detection strategy is used only as a preprocessing step to face detection, particularly in real time applications. In this study we have focused on the problem of developing an accurate and robust model for the human skin.

Objective



The model will offer child proof surfing on the internet without parent intervention. So that parents do not have to worry about their children coming across nude images at such an early age without knowing the actual meaning of it. That is it will monitor every page and filter all the images, hiding their details from the children and disabling their activation upon any click. Even if the people in the image is not completely nude, may it be just the upper or lower half of the person's body; the model will still blur that image once it reaches the minimum percentage of human pixel colour set by the classifier, thus ensuring guaranteed protection.

Neural Networks



Neural Networks is a machine learning algorithm, which is built on the principle of the organization and functioning of biological neural networks. Neural networks consist of individual units called neurons. Neurons are located in a series of groups — layers. Neurons in each layer are connected to neurons of the next layer. Data comes from the input layer to the output layer along these compounds. Each individual node performs a simple mathematical calculation. Then it transmits its data to all the nodes it is connected to.

Different types of neural network architecture are as follows:

- 1) Perceptrons
- 2) Convolutional Neural Networks
- 3) Recurrent Neural Networks
- 4) Long / Short Term Memory
- 5) Gated Recurrent Unit
- 6) Hopfield Network

Why Convolution Neural Networks

Convolutional neural networks (CNN) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon – for generating product recommendations and Google – for search through among users' photos. The main task of image classification is acceptance of the input image and the following definition of its class. This is a skill that people learn from their birth and are able to easily determine that the image in the picture is an elephant. But the computer sees the pictures quite differently:

What I see

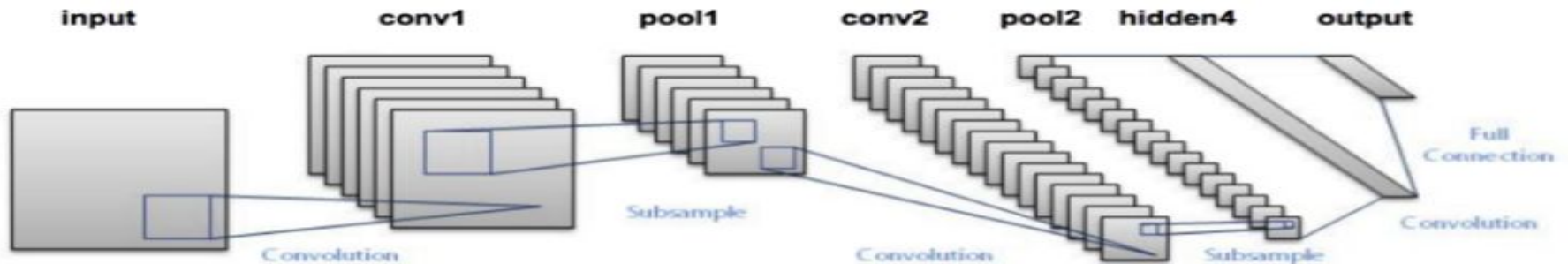


What a computer sees


08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	43	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	95	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	41	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	34
01	70	54	71	83	51	54	69	16	92	33	48	41	43	52	01	89	19	47	48

Convolution Neural Network cont.....

Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point. To solve this problem the computer looks for the characteristics of the base level. In human understanding such characteristics are for example the trunk or large ears. For the computer, these characteristics are boundaries or curvatures. And then through the groups of convolutional layers the computer constructs more abstract concepts. In more detail: the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.



Output of Cat and Dog Tutorial with 4 Layers



```
Epoch 11/20
32/32 [=====] - 229s 7s/step - loss: 0.6069 - acc: 0.6576 - val_loss: 0.6123 - val_acc: 0.6475
Epoch 12/20
32/32 [=====] - 228s 7s/step - loss: 0.6024 - acc: 0.6834 - val_loss: 0.6115 - val_acc: 0.6625
Epoch 13/20
32/32 [=====] - 227s 7s/step - loss: 0.6050 - acc: 0.6580 - val_loss: 0.6072 - val_acc: 0.6600
Epoch 14/20
32/32 [=====] - 226s 7s/step - loss: 0.6050 - acc: 0.6761 - val_loss: 0.5730 - val_acc: 0.7000
Epoch 15/20
32/32 [=====] - 230s 7s/step - loss: 0.6051 - acc: 0.6700 - val_loss: 0.6301 - val_acc: 0.6250
Epoch 16/20
32/32 [=====] - 230s 7s/step - loss: 0.5850 - acc: 0.6917 - val_loss: 0.6203 - val_acc: 0.6450
Epoch 17/20
32/32 [=====] - 227s 7s/step - loss: 0.6045 - acc: 0.6427 - val_loss: 0.5674 - val_acc: 0.7250
Epoch 18/20
32/32 [=====] - 230s 7s/step - loss: 0.5942 - acc: 0.6790 - val_loss: 0.5915 - val_acc: 0.6575
Epoch 19/20
32/32 [=====] - 230s 7s/step - loss: 0.5915 - acc: 0.6861 - val_loss: 0.5500 - val_acc: 0.7575
Epoch 20/20
32/32 [=====] - 225s 7s/step - loss: 0.5653 - acc: 0.7142 - val_loss: 0.5567 - val_acc: 0.7275
>72.750
```

Using VGG16 model



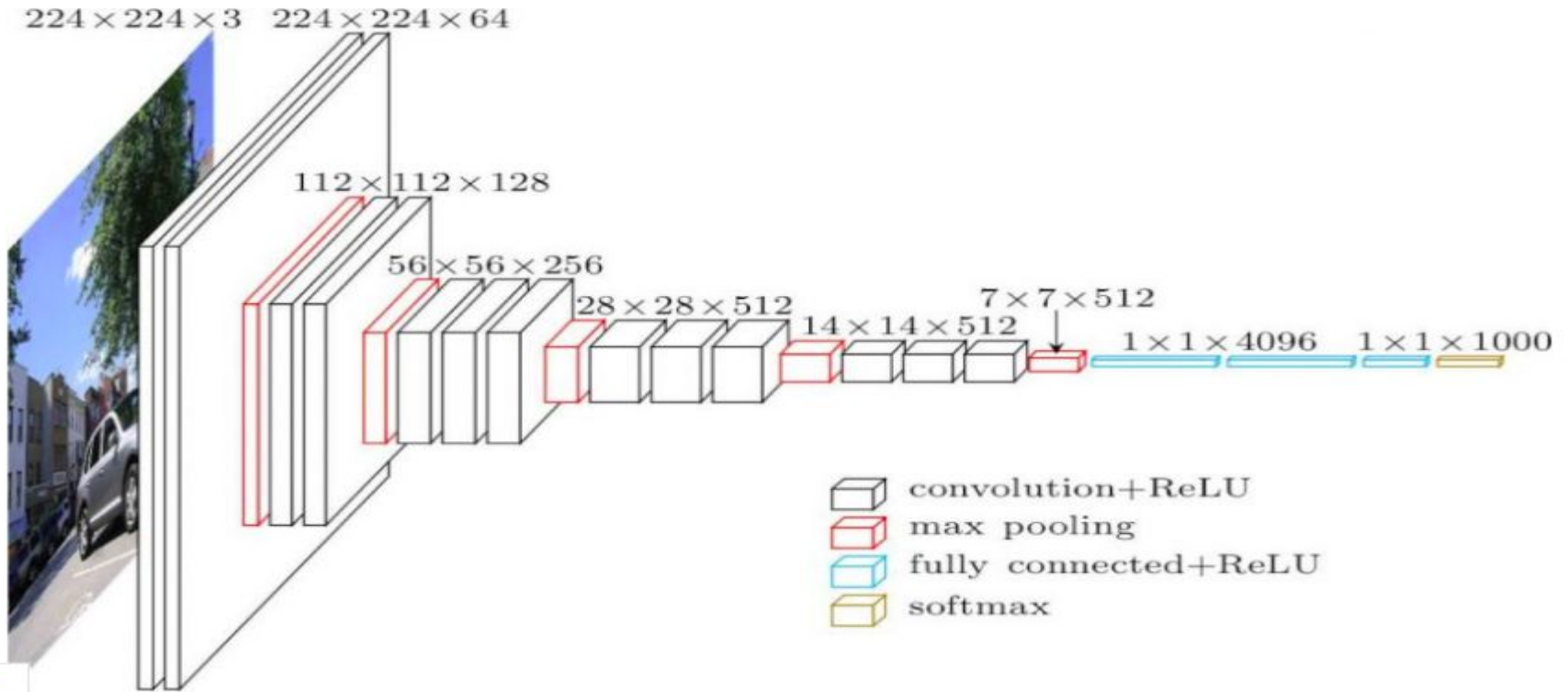
```
Epoch 1/10
32/32 [=====] - 1192s 37s/step - loss: 0.9444 - acc: 0.9191 - val_loss: 0.5645 - val_acc: 0.9500
Epoch 2/10
32/32 [=====] - 1181s 37s/step - loss: 0.3874 - acc: 0.9700 - val_loss: 0.3456 - val_acc: 0.9750
Epoch 3/10
32/32 [=====] - 1187s 37s/step - loss: 0.3833 - acc: 0.9719 - val_loss: 0.6194 - val_acc: 0.9525
Epoch 4/10
32/32 [=====] - 1192s 37s/step - loss: 0.3094 - acc: 0.9751 - val_loss: 0.5087 - val_acc: 0.9600
Epoch 5/10
32/32 [=====] - 1185s 37s/step - loss: 0.1624 - acc: 0.9858 - val_loss: 0.2661 - val_acc: 0.9750
Epoch 6/10
32/32 [=====] - 1189s 37s/step - loss: 0.1257 - acc: 0.9902 - val_loss: 0.2903 - val_acc: 0.9750
Epoch 7/10
32/32 [=====] - 1200s 37s/step - loss: 0.1794 - acc: 0.9873 - val_loss: 0.6117 - val_acc: 0.9525
Epoch 8/10
32/32 [=====] - 1205s 38s/step - loss: 0.1362 - acc: 0.9902 - val_loss: 0.4027 - val_acc: 0.9725
Epoch 9/10
32/32 [=====] - 1207s 38s/step - loss: 0.1283 - acc: 0.9902 - val_loss: 0.2983 - val_acc: 0.9800
Epoch 10/10
32/32 [=====] - 1190s 37s/step - loss: 0.1441 - acc: 0.9897 - val_loss: 0.3136 - val_acc: 0.9800
>98.000
```


Predictions



```
/content/drive/My Drive/major projects/cat and dog/test/dog.4024.jpg  
[1.]  
/content/drive/My Drive/major projects/cat and dog/test/cat.4134.jpg  
[0.]  
/content/drive/My Drive/major projects/cat and dog/test/dog.4025.jpg  
[1.]  
/content/drive/My Drive/major projects/cat and dog/test/cat.4183.jpg  
[0.]  
/content/drive/My Drive/major projects/cat and dog/test/dog.4166.jpg  
[1.]  
/content/drive/My Drive/major projects/cat and dog/test/cat.4138.jpg  
[0.]  
/content/drive/My Drive/major projects/cat and dog/test/cat.4040.jpg  
[0.]  
/content/drive/My Drive/major projects/cat and dog/test/cat.4162.jpg  
[0.]  
/content/drive/My Drive/major projects/cat and dog/test/dog.4102.jpg  
[1.]  
/content/drive/My Drive/major projects/cat and dog/test/dog.4134.jpg  
[1.]  
/content/drive/My Drive/major projects/cat and dog/test/cat.4014.jpg  
[0.]  
/content/drive/My Drive/major projects/cat and dog/test/dog.4014.jpg  
[1.]  
/content/drive/My Drive/major projects/cat and dog/test/dog.4033.jpg  
[1.]
```

VGG16 Architecture



VGG16



The input to cov1 layer is of fixed size 224×224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

VGG16 Layers



Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808

VGG16 Layers



block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
=====		
Total params: 138,357,544		
Trainable params: 138,357,544		
Non-trainable params: 0		

Bare Image Classification

```
↳ Found 806 images belonging to 2 classes.  
Found 200 images belonging to 2 classes.  
Epoch 1/10  
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:716: UserWarning: This ImageDataGenerator specifies  
warnings.warn('This ImageDataGenerator specifies '  
13/13 [=====] - 652s 50s/step - loss: 1.8829 - acc: 0.8087 - val_loss: 8.7702 - val_acc: 0.4250  
Epoch 2/10  
13/13 [=====] - 642s 49s/step - loss: 0.9601 - acc: 0.9238 - val_loss: 8.8420 - val_acc: 0.4250  
Epoch 3/10  
13/13 [=====] - 639s 49s/step - loss: 0.8273 - acc: 0.9382 - val_loss: 6.0052 - val_acc: 0.5900  
Epoch 4/10  
13/13 [=====] - 640s 49s/step - loss: 0.8191 - acc: 0.9387 - val_loss: 9.9569 - val_acc: 0.3550  
Epoch 5/10  
13/13 [=====] - 635s 49s/step - loss: 0.5197 - acc: 0.9616 - val_loss: 8.3309 - val_acc: 0.4600  
Epoch 6/10  
13/13 [=====] - 634s 49s/step - loss: 0.4111 - acc: 0.9684 - val_loss: 7.1673 - val_acc: 0.5250  
Epoch 7/10  
13/13 [=====] - 635s 49s/step - loss: 0.4449 - acc: 0.9703 - val_loss: 6.7312 - val_acc: 0.5450  
Epoch 8/10  
13/13 [=====] - 636s 49s/step - loss: 0.4037 - acc: 0.9739 - val_loss: 7.6236 - val_acc: 0.4850  
Epoch 9/10  
13/13 [=====] - 637s 49s/step - loss: 0.3788 - acc: 0.9747 - val_loss: 8.5476 - val_acc: 0.4500  
Epoch 10/10  
13/13 [=====] - 639s 49s/step - loss: 0.3963 - acc: 0.9711 - val_loss: 9.2629 - val_acc: 0.4100  
>41.000
```

Bare Image Classification



```
Epoch 8/20
13/13 [=====] - 643s 49s/step - loss: 0.6071 - acc: 0.9559 - val_loss: 8.4083 - val_acc: 0.4550
Epoch 9/20
13/13 [=====] - 642s 49s/step - loss: 0.3916 - acc: 0.9723 - val_loss: 8.7273 - val_acc: 0.4300
Epoch 10/20
13/13 [=====] - 641s 49s/step - loss: 0.3892 - acc: 0.9735 - val_loss: 8.6571 - val_acc: 0.4400
Epoch 11/20
13/13 [=====] - 639s 49s/step - loss: 0.4021 - acc: 0.9727 - val_loss: 8.9160 - val_acc: 0.4250
Epoch 12/20
13/13 [=====] - 642s 49s/step - loss: 0.4010 - acc: 0.9739 - val_loss: 8.8522 - val_acc: 0.4150
Epoch 13/20
13/13 [=====] - 639s 49s/step - loss: 0.4105 - acc: 0.9744 - val_loss: 8.4253 - val_acc: 0.4500
Epoch 14/20
13/13 [=====] - 643s 49s/step - loss: 0.3911 - acc: 0.9756 - val_loss: 8.3093 - val_acc: 0.4550
Epoch 15/20
13/13 [=====] - 658s 51s/step - loss: 0.3911 - acc: 0.9756 - val_loss: 8.2767 - val_acc: 0.4550
Epoch 16/20
13/13 [=====] - 672s 52s/step - loss: 0.3666 - acc: 0.9771 - val_loss: 8.2689 - val_acc: 0.4550
Epoch 17/20
13/13 [=====] - 659s 51s/step - loss: 0.3666 - acc: 0.9771 - val_loss: 8.2671 - val_acc: 0.4550
Epoch 18/20
13/13 [=====] - 647s 50s/step - loss: 0.3788 - acc: 0.9763 - val_loss: 8.2677 - val_acc: 0.4550
Epoch 19/20
13/13 [=====] - 653s 50s/step - loss: 0.3666 - acc: 0.9771 - val_loss: 8.2671 - val_acc: 0.4550
Epoch 20/20
13/13 [=====] - 650s 50s/step - loss: 0.3788 - acc: 0.9763 - val_loss: 8.2689 - val_acc: 0.4550
>45.500
```


Creation Of Our Own Model

```
model = Sequential()
model.add(Conv2D(input_shape=(200,200,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

model.add(Flatten())

model.add(Dense(units=1024,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=500,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=128,activation="relu"))
model.add(Dense(units=64,activation="relu"))
model.add(Dense(units=1, activation="tanh"))
```


VGG19 Model With Some Additional Dense Layers



```
1 import sys
2 from matplotlib import pyplot
3 from keras.utils import to_categorical
4 from keras.applications.vgg19 import VGG19
5 from keras.models import Model
6 from keras.layers import Dense
7 from keras.layers import Flatten
8 from keras.optimizers import SGD
9
10 model=VGG19(include_top=False,input_shape=(200,200,3))
11 for layer in model.layers:
12     layer.trainable=False
13 flat1=Flatten()(model.layers[-1].output)
14 class1=Dense(500,activation='relu',kernel_initializer="he_uniform")(flat1)
15 class2=Dense(500,activation='relu',kernel_initializer="he_uniform")(class1)
16 class3=Dense(500,activation='relu',kernel_initializer="he_uniform")(class2)
17 class4=Dense(500,activation='relu',kernel_initializer="he_uniform")(class3)
18 output=Dense(1,activation='sigmoid')(class4)
19 model=Model(inputs=model.inputs,outputs=output)
```

Output of Our Model With 5604 training images

```
[ ] 1 opt=SGD(lr=0.001,momentum=0.9)
    2 model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    3 model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=15,verbose=1)
    4 #model.save_weights("weights.h5")
    5 #model.save("final.h5")
```

```
Train on 5604 samples, validate on 999 samples
Epoch 1/15
5604/5604 [=====] - 40s 7ms/step - loss: 0.6825 - acc: 0.5803 - val_loss: 0.6212 - val_acc: 0.7518
Epoch 2/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.6601 - acc: 0.6089 - val_loss: 0.7298 - val_acc: 0.3233
Epoch 3/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.6681 - acc: 0.5805 - val_loss: 0.7753 - val_acc: 0.3293
Epoch 4/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.6332 - acc: 0.6363 - val_loss: 0.5746 - val_acc: 0.6937
Epoch 5/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5637 - acc: 0.7136 - val_loss: 0.4926 - val_acc: 0.7898
Epoch 6/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5376 - acc: 0.7404 - val_loss: 0.4925 - val_acc: 0.7848
Epoch 7/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5592 - acc: 0.7243 - val_loss: 0.5506 - val_acc: 0.7317
Epoch 8/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5184 - acc: 0.7575 - val_loss: 0.4492 - val_acc: 0.8318
Epoch 9/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5239 - acc: 0.7514 - val_loss: 0.5181 - val_acc: 0.7708
Epoch 10/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5199 - acc: 0.7521 - val_loss: 0.4678 - val_acc: 0.8078
Epoch 11/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.5038 - acc: 0.7630 - val_loss: 0.4547 - val_acc: 0.8438
Epoch 12/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.4857 - acc: 0.7741 - val_loss: 0.3581 - val_acc: 0.8629
Epoch 13/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.4804 - acc: 0.7741 - val_loss: 0.3733 - val_acc: 0.8539
Epoch 14/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.4822 - acc: 0.7703 - val_loss: 0.5174 - val_acc: 0.7808
Epoch 15/15
5604/5604 [=====] - 38s 7ms/step - loss: 0.4780 - acc: 0.7784 - val_loss: 0.3424 - val_acc: 0.8729
<keras.callbacks.History at 0x7f4ce2324860>
```

Output of VGG19 with 5604 training images

```
[ ] 1 opt=SGD(lr=0.001,momentum=0.9)
    2 model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    3 model.fit(X_train,Y_train,validation_data=(X_test,Y_test),batch_size=128,epochs=15,verbose=1)
    4 #model.save_weights("weights.h5")
    5 model.save("final.h5")
```

Train on 5604 samples, validate on 999 samples

```
Epoch 1/15
5604/5604 [=====] - 21s 4ms/step - loss: 0.0065 - acc: 0.9995 - val_loss: 0.7341 - val_acc: 0.8368
Epoch 2/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0065 - acc: 0.9995 - val_loss: 0.6669 - val_acc: 0.8589
Epoch 3/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0067 - acc: 0.9993 - val_loss: 0.6752 - val_acc: 0.8559
Epoch 4/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0065 - acc: 0.9995 - val_loss: 0.7129 - val_acc: 0.8478
Epoch 5/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0066 - acc: 0.9995 - val_loss: 0.6376 - val_acc: 0.8669
Epoch 6/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0069 - acc: 0.9993 - val_loss: 0.6841 - val_acc: 0.8539
Epoch 7/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0064 - acc: 0.9993 - val_loss: 0.6529 - val_acc: 0.8609
Epoch 8/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0065 - acc: 0.9995 - val_loss: 0.7065 - val_acc: 0.8478
Epoch 9/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0065 - acc: 0.9993 - val_loss: 0.6958 - val_acc: 0.8529
Epoch 10/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_loss: 0.6650 - val_acc: 0.8599
Epoch 11/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_loss: 0.7135 - val_acc: 0.8478
Epoch 12/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_loss: 0.6784 - val_acc: 0.8579
Epoch 13/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_loss: 0.7408 - val_acc: 0.8438
Epoch 14/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0064 - acc: 0.9995 - val_loss: 0.6634 - val_acc: 0.8619
Epoch 15/15
5604/5604 [=====] - 16s 3ms/step - loss: 0.0065 - acc: 0.9993 - val_loss: 0.7027 - val_acc: 0.8539
```

Comparison



OWN MODEL

- 1] It has 15 Convolution Layers with 5 blocks
- 2] It has 5 MaxPooling Layers with pool size (3,3) and strides(2,2).
- 3] It has 7 Fully Connected Layers having :
1024 units in 1st layer, 1000 units in 2nd layer, 500 units in 3rd layer, 128 units in 4th and 5th layer, 64 units in 6th layer and 1 unit in 7th i.e output layer.
- 4] Accuracy of Model with 15 epochs: 87.29

VGG19 MODEL

- 1] It has 16 Convolution Layers with 5 blocks.
- 2] It has 5 MaxPooling Layers.
- 3] It has 5 Fully Connected Layers having:
500 units in all the 4 layers and 1 unit in 5th i.e output layer.
- 4] Accuracy of Model with 15 epochs : 85.39

Few of the Common Parameters



- Optimizer Used : Stochastic Gradient Descent(SGD)
- Activation Function : Sigmoid
- Learning Rate : 0.001
- Momentum : 0.9
- Batch Size : 128
- Loss : Binary Crossentropy
- Training Images : 5604
- Testing Images : 999
- No. of Epochs : 15

Predictions



```
/content/drive/My Drive/major projects/nude and non-nude/test/No444.jpg
[0.]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes486.jpg
[0.9999287]
/content/drive/My Drive/major projects/nude and non-nude/test/No498.jpg
[2.7357288e-05]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes500.jpg
[0.99979466]
/content/drive/My Drive/major projects/nude and non-nude/test/No402.jpg
[0.47208062]
/content/drive/My Drive/major projects/nude and non-nude/test/No460.jpg
[0.]
/content/drive/My Drive/major projects/nude and non-nude/test/No462.jpg
[0.00023309]
/content/drive/My Drive/major projects/nude and non-nude/test/No480.jpg
[9.990355e-08]
/content/drive/My Drive/major projects/nude and non-nude/test/No458.jpg
[0.]
/content/drive/My Drive/major projects/nude and non-nude/test/No484.jpg
[0.00032475]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes409.jpg
[0.9999999]
/content/drive/My Drive/major projects/nude and non-nude/test/No428.jpg
[0.6779908]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes434.jpg
[0.9999193]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes465.jpg
[0.96150035]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes457.jpg
[0.9998764]
/content/drive/My Drive/major projects/nude and non-nude/test/No447.jpg
[1.8960392e-05]
/content/drive/My Drive/major projects/nude and non-nude/test/No419.jpg
[0.98673356]
/content/drive/My Drive/major projects/nude and non-nude/test/No442.jpg
[0.00457988]
/content/drive/My Drive/major projects/nude and non-nude/test/No424.jpg
[0.00057627]
/content/drive/My Drive/major projects/nude and non-nude/test/Yes477.jpg
[0.99999917]
```

Future Scope



To Create Google Chrome Extension Using the Trained model which will blur the Bare Skinned Images on the Webpage At RunTime as soon as the page gets loaded .

Also the hyperlink will be disabled if any with such type of images.



Thank You