Real Time Bare Skinned Images Filtering Using CNN

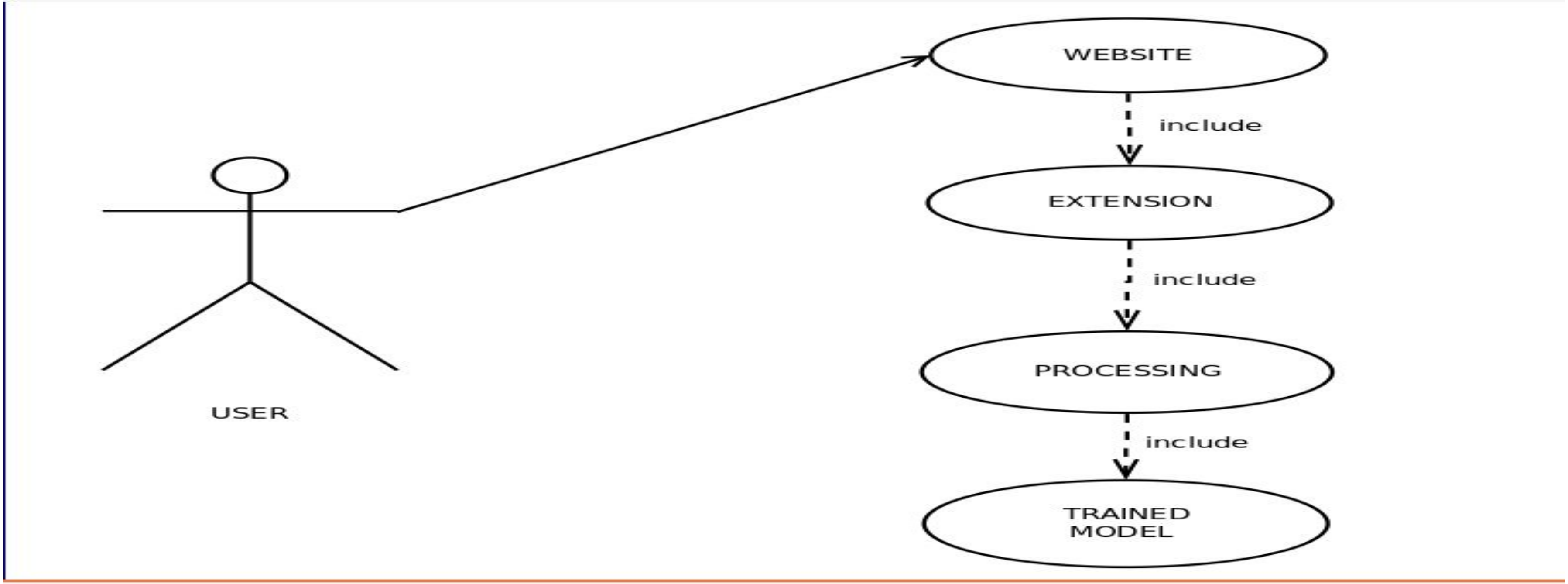
Introduction

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.

Used Case Diagram



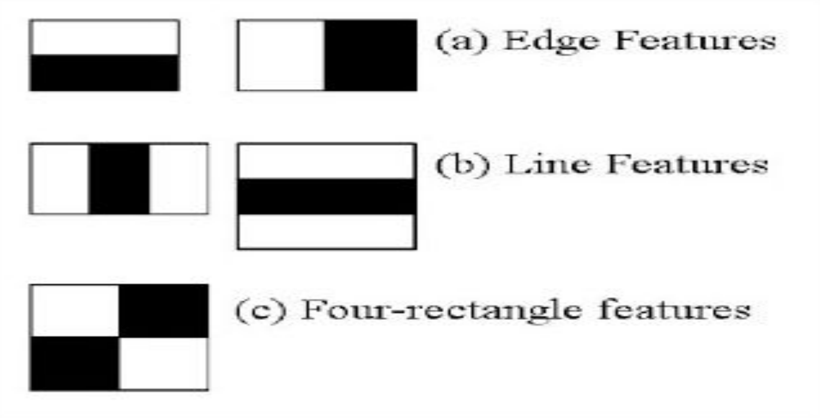
Face Detection

Face Detection is a type of application classified under “computer vision” technology. It is the process in which algorithms are developed and trained to properly locate faces or objects in images.These can be in real time from a video camera or from photographs. On social media apps like Snapchat, face detection is required to augment reality which allows users to virtually wear dog face masks using fancy filters. Another use of face detection is in smartphone face ID security.

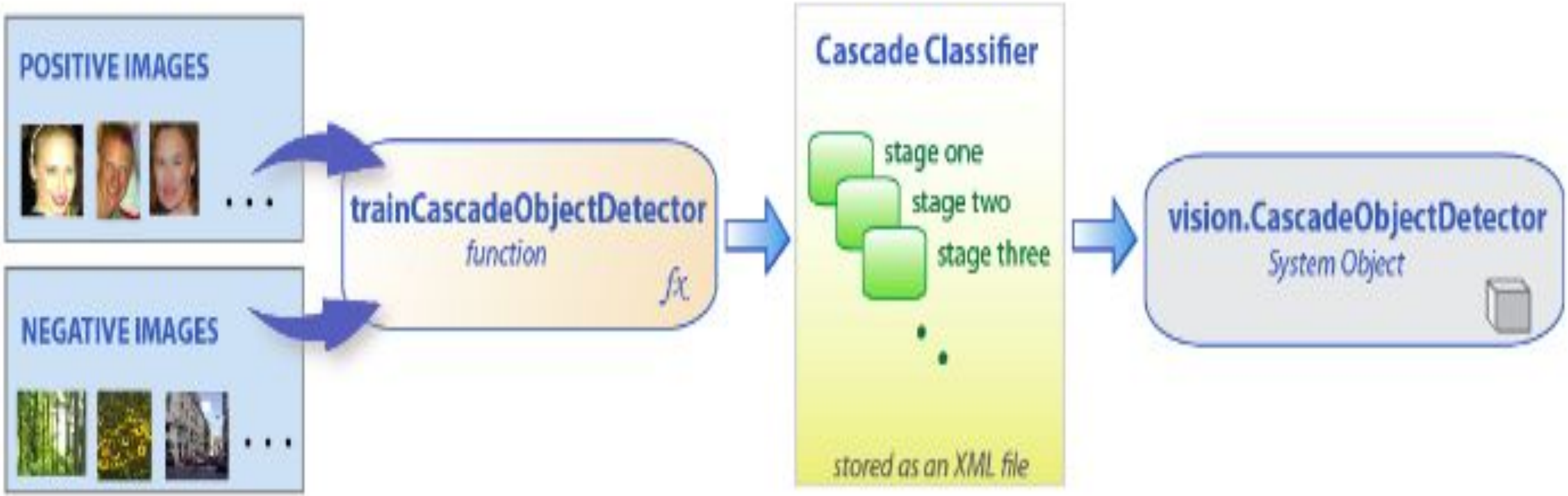
So the first aim of aur project is to detect the whether there is human face in the image or not.Face Detection is implemented in OpenCv using 2 techniques:
1)Haar Cascade Classifier
2)Deep Neural Network

Haar Cascade Classifier

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones.It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it.They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.



Haar Cascade Classifier

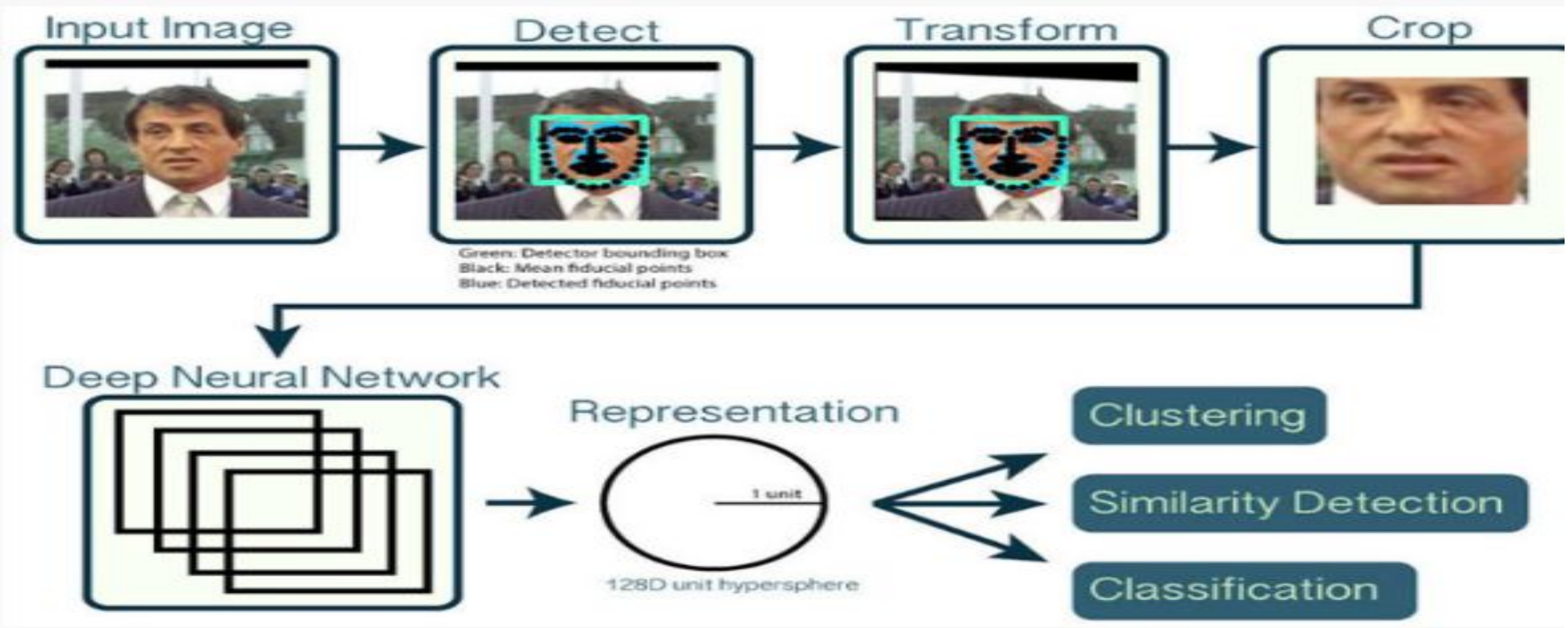


Deep Neural Network

OpenCV (Open Source Computer Vision) is a library with functions that mainly aiming real-time computer vision. OpenCV supports Deep Learning frameworks Caffe, Tensorflow, Torch/PyTorch.With OpenCV we can perform face detection using pre-trained deep learning face detector model which is shipped with the library.When OpenCV 3.3 was officially released, it has highly improved deep neural networks (dnn) module.
We have used Caffe based framework model for face detection.To use OpenCV Deep Neural Network module with Caffe models we will need two files .
1)prototxt file which defines model architecture
2).caffemodel file which contains the weights for the actual layers

OpenCV’s face detector is based on the Single Shot Detector framework with a ResNet base network.

Deep Neural Network



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

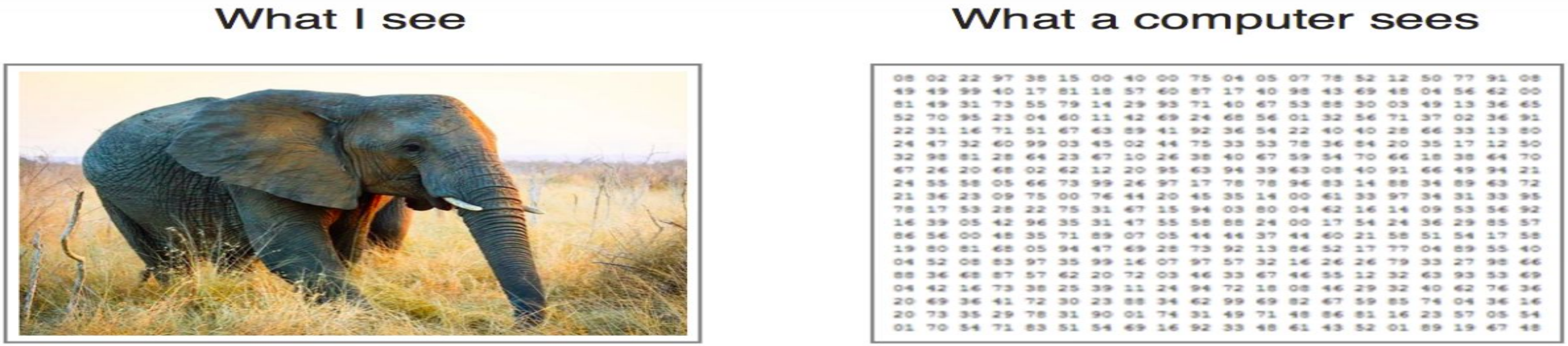
4)Long / Short Term Memory

5)Gated Recurrent Unit

6)Hopfield Network

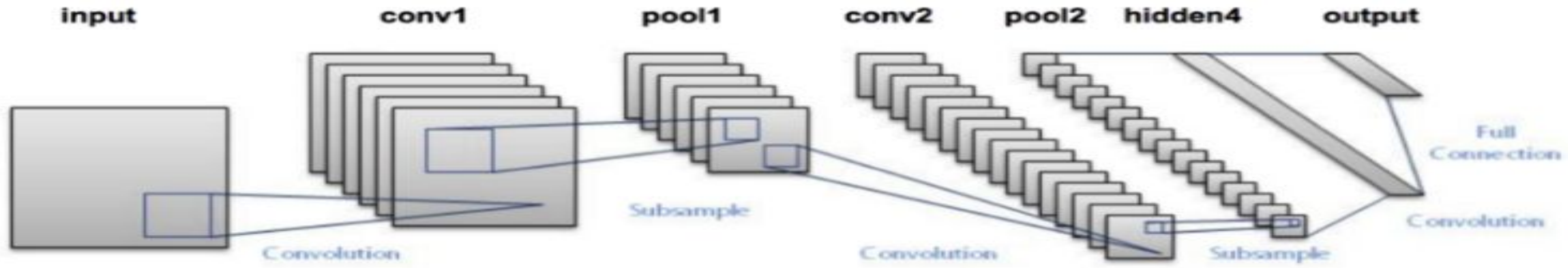
Why Convolution Neural Networks

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:



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
```

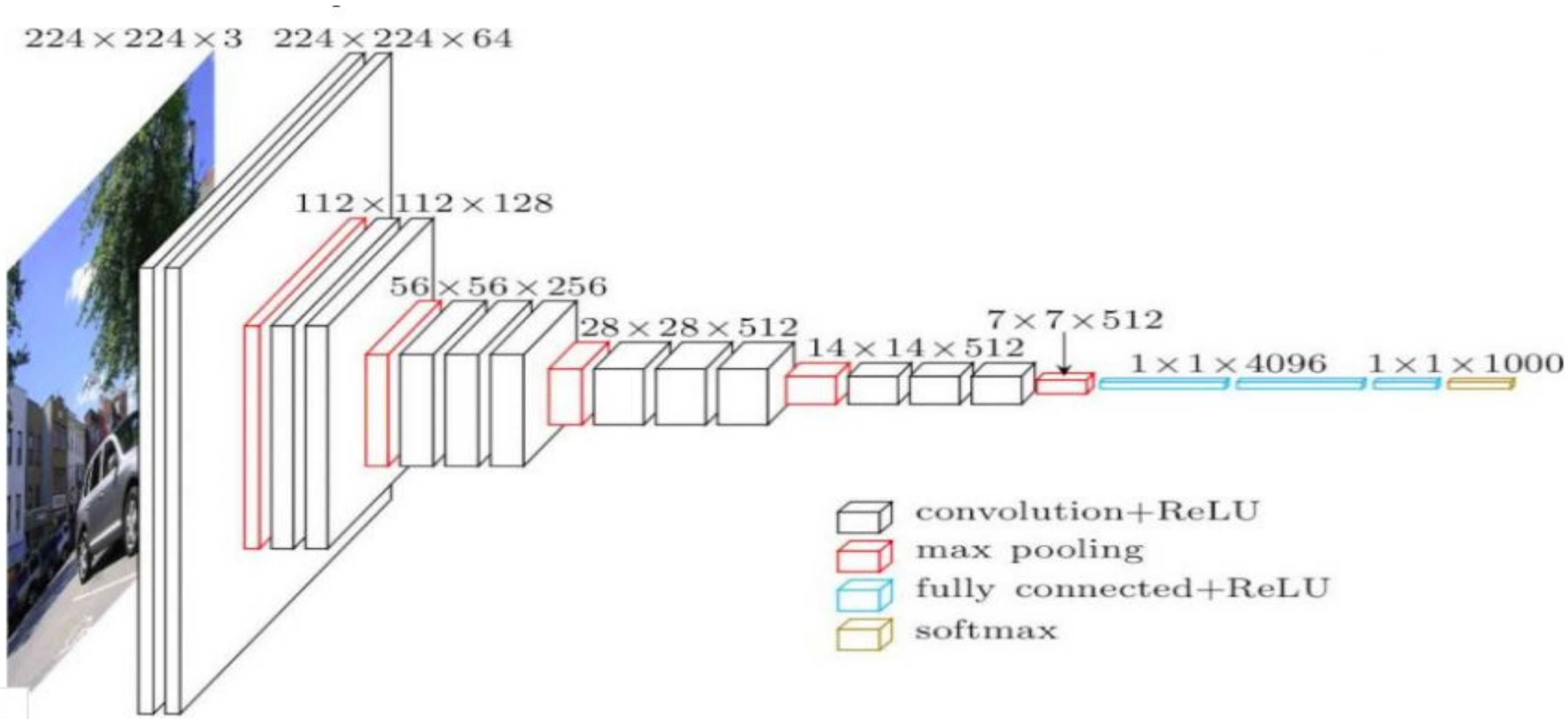
Using VGG16 model

```
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 x 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 1000channels (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 speci
  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
```

Thank You