Chapter 5

Implementation

Age and gender play fundamental roles in social interactions. Languages reserve different salutations and grammar rules for men or women, and very often different vocabularies are used when addressing elders compared to young people. Despite the basic roles these attributes play in our day-to-day lives, the ability to automatically estimate them accurately and reliably from face images is still far from meeting the needs of commercial applications. This is particularly perplexing when considering recent claims to super-human capabilities in the related task of face recognition. In this section, we first analyze the influence of each re- gion on the classification of both, gender and age attributes, and compare the component-based approach with respect to the use of the complete face image. We select the most dis- criminative regions and combine them to make the attributes classification, and we compare the result with a state-of- the-art method. Finally we conduct some experiments in order to analyze the robustness of the proposal in front of conclusions.

5.1 Overview

The face images of persons are captured by means of a android camera and some images are collected from net data set images. This paper proposed a novel and effective age group estimation using face features from human face images. This process involves three stages: Preprocessing, Normalization, Feature Extraction, and Classification.

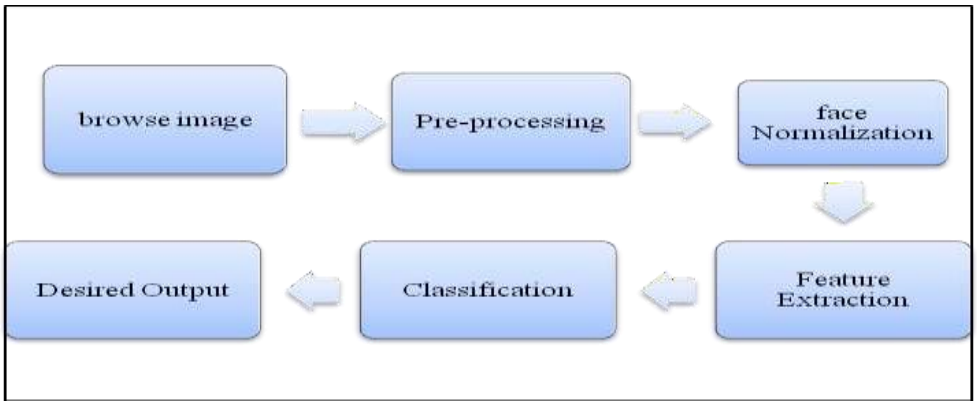

Figure 5.1 Proposed Block Diagram

We propose to subdivide a face image into eight regions of interest that were defined using 68 landmark points detected in the face alignment step. The complete set of regions, where each region corresponds to a functional part of the face (i.e. forehead, eyebrows, eyes, nose, mouth, cheeks, chin and the complete face region). This subdivision allows us to take advantage of the common geometry shared by faces: the regions contain the face parts despite changes in pose, small alignment errors and morphological differences between individuals.

Our regions are somehow similar to those defined. In addition to those shown, they consider a region for the hair zone and also a region for the area between mouth and nose. Since we are only interested on age and gender we propose to simplify this subdivision. Men and women from almost all ages can both have long hair and similar haircuts or hairstyles, so we consider that a hair region is not only poorly discriminative for both gender and age, but could also provide misleading information. Also, we discard the use of an additional region for the area between mouth and nose, instead we enlarge our mouth region enough to cover this face part, in a more compact region. For the cheekbones, unlike the ellipses used, we employ triangular shapes that we believe cover the area better, especially when there are changes in pose and diverse facial expressions. Our regions are automatically obtained based on landmarks estimation, and this increase the robustness against variations in pose and expressions. It should be no- ticed that in the case of eyes, eyebrows and cheeks, each region is composed by two parts corresponding to both sides of the face image. Every region is represented by a different color.

5.2 Implementation steps

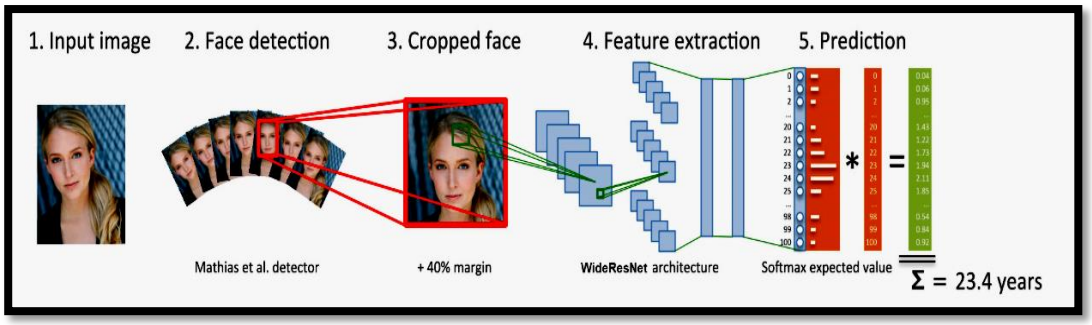

Figure 5.2 An overview of how it works

Implementation of this model is shown in the below steps using the following techniques.

1.Input image:

First, the photo is taken from the webcam stream live by the cv2 module. To capture live stream with camera, OpenCV is used which provides a very simple interface to this. To capture a video, a VideoCapture object is created. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected. So it is simply passed with 0 (or -1). After that, we can capture frame-by-frame.

2.Face Detection:

Second, we turn the image to grayscale and use the cv2 module's Cascade Classifier class to detect faces in the image. The variable faces return by the detectMultiScale method is a list of detected face coordinates [x, y, w, h]. cvtColor() method is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV. The above method is passed with 2 arguments where one is frame(input image) and another is type of conversion. Since we are converting image to grayscale (or gray level) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel.

3.Cropped face:

After known the faces' coordinates, we need to crop those faces before feeding to the neural network model. We add the 40% margin to the face area so that the full head is included. Face is cropped using crop_face method giving frame as the input. cv2. rectangle() method is used to draw a rectangle on any image color: It is the color of border line of rectangle to be drawn. For BGR, we pass a tuple. Eg: (255, 0, 0) for blue color.

4.Feature Extraction**:**

Then we are ready to feed those cropped faces to the model, it's as simple as calling the predict method. Here we are using Wide Resnet architecture to predict age.For the age prediction, the output of the model is a list of 101 values associated with age probabilities ranging from 0~100, and all the 101 values add up to 1 (or what we call softmax).The image is processed using wide resnet technique. So we multiply each value with its associated age and sum them up resulting final predicted age.In hidden layer ReLU is used as activation function and in output layer softmax function as activation function. The feature extraction part of the neural network uses the WideResNet architecture, short for Wide Residual Networks. It leverages the power of Convolutional Neural Networks to learn the features of the face. From less abstract features like edges and corners to more abstract features like eyes and mouth.

5.Prediction

Last but not least we draw the result and render the image. The output of wide resnet technique outputs the value of age ranges and on applying binary classification on it the gender is recognized.

5.3 Algorithm

The convolutional neural network (CNN) is a class of deep learning neural networks. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

A CNN convolves (not convolutes…) learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing 2D images. Compared to other image classification algorithms, CNNs actually use very little preprocessing. This means that they can learn the filters that have to be hand-made in other algorithms. CNNs can be used in tons of applications from image and video recognition, image classification, and recommender systems to natural language processing and medical image analysis.

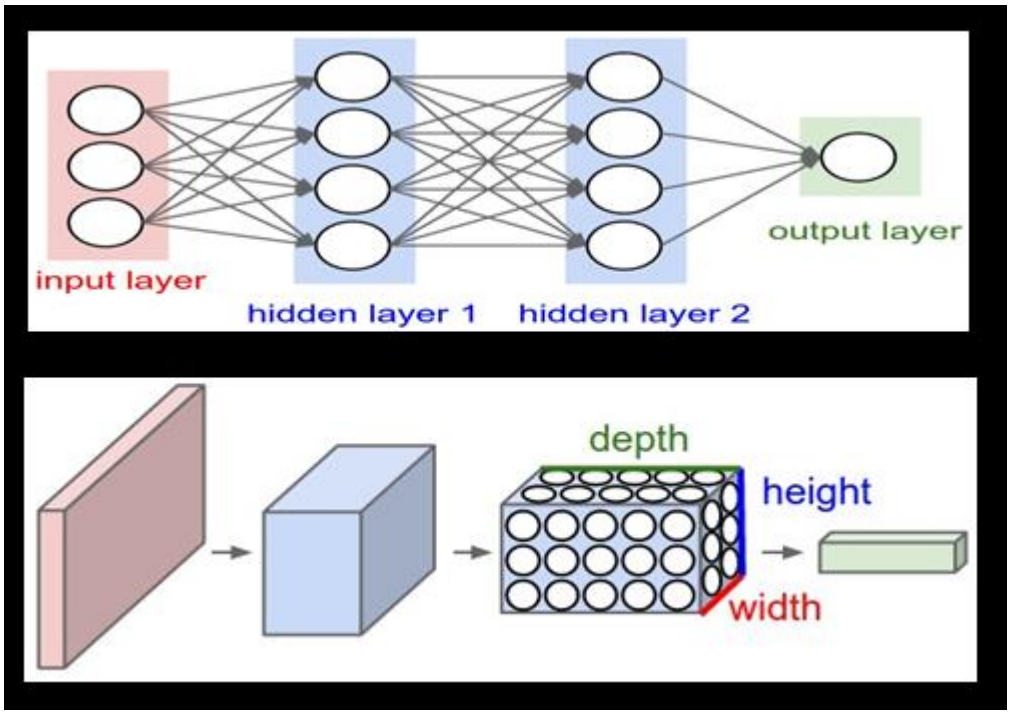

Figure 5.3 Structure of CNN

CNNs have an input layer, and output layer, and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

- Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
- Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.
- Fully connected layers connect every neuron in one layer to every neuron in the next layer.

In a convolutional layer, neurons only receive input from a subarea of the previous layer. In a fully connected layer, each neuron receives input from *every* element of the previous layer. A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained. They're learned while the network trains on a set of images. This makes deep learning models extremely accurate for

computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

- INPUT [32x32x3] will hold the raw pixel values of the image.
- CONV layer will compute the output of neurons that are connected to local regions in the input
- This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero.
- POOL layer will perform a downsampling [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10].
- The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

For example, we demonstrate that even a simple 16-layer-deep wide residual network outperforms in accuracy and efficiency all previous deep residual networks, including thousand-layer-deep networks. We further show that WRNs achieve incredibly good results (e.g., achieving new state-of-the-art results on CIFAR-10, CIFAR-100, SVHN, COCO and substantial improvements on ImageNet) and train several times faster than pre-activation ResNets. Wide Residual networks simply have increased number of channels compared to ResNet. Otherwise the architecture is the same. Deeper ImageNet models with bottleneck block have increased number of channels in the inner 3x3 convolution.

In WRNs, plenty of parameters are tested such as the design of the ResNet block, how deep (deepening factor l) and how wide (widening factor k) within the ResNet block. When k=1, it has the same width of *ResNet*. While k>1, it is k time wider than *ResNet*. WRN-*d*-*k*: means the WRN has the depth of d and with widening factor k.

- Pre-Activation ResNet is used in CIFAR-10, CIFAR-100 and SVHN datasets. Original *ResNet* is used in ImageNet dataset.

- The major difference is that *Pre-Activation ResNet* has a structure of performing batch norm and ReLU before convolution (i.e. BN-ReLU-Conv) while original *ResNet* has a structure of Conv-BN-ReLU. And *Pre-Activation ResNet* is generally better than the original one, but it has no obvious improvement in ImageNet when layers are only around 100.

ReLU (Rectified Linear Unit) :

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.Instead of sigmoids, most recent deep learning networks use rectified linear units (ReLUs) for the hidden layers. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. ReLUs' machinery is more like a real neuron in our body .The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.

$$F(x)=max(x,0)$$

Max Pooling:

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers.

Softmax Function:

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1. The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true. Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes.

Binary Classification:

The goal of a binary classification problem is to make a prediction that can be one of just two possible values. The gender prediction is a binary classification task. The model outputs value between 0~1, where the higher the value, the more confidence the model think the face is a male. The output of wide resnet is used and using binary qualification gender is recognized from the image captured.

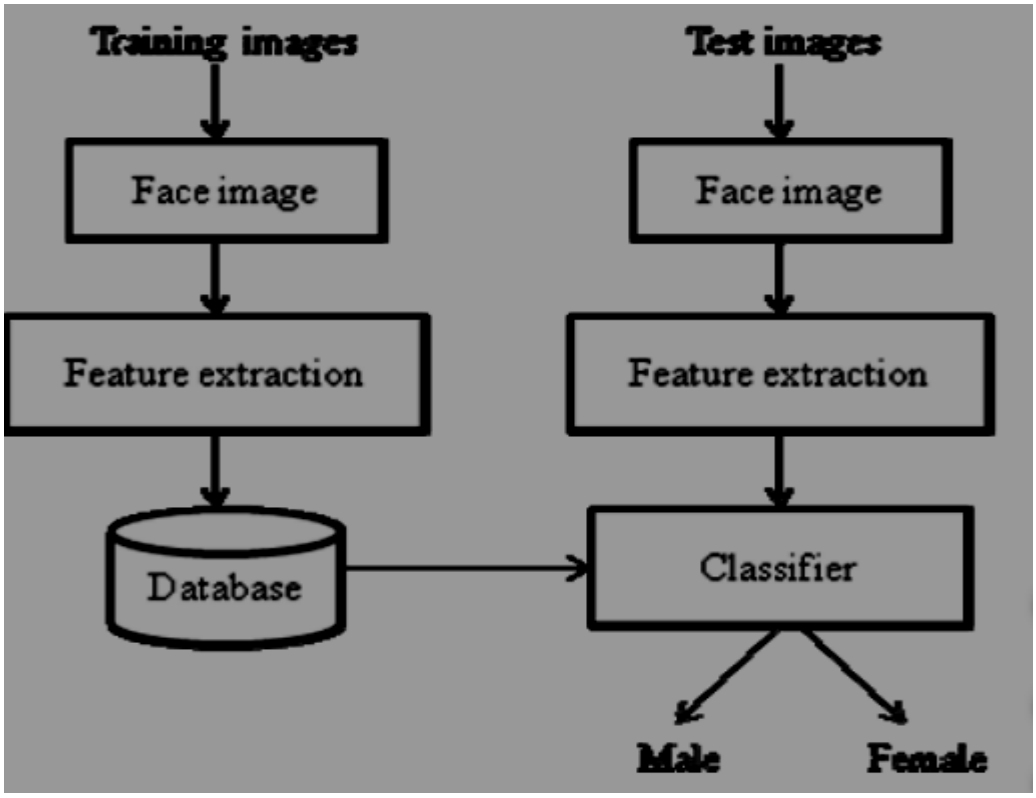

Figure 5.4 Binary classification dataflow diagram

Keras:

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Keras also supplies ten well-known models, called Keras Applications, pretrained against ImageNet: Xception, VGG16, VGG19, ResNet50, InceptionV3, InceptionResNetV2, MobileNet, DenseNet, NASNet, MobileNet V2TK. You can use these to predict the classification of images, extract features from them, and fine-tune the models on a different set of classes. By the way, fine-tuning existing models is a good way to speed up training. For example, you can add layers as you wish, freeze the base layers to train the new layers, then unfreeze some of the base layers to fine-tune the training. You can freeze a layer with by setting layer trainable = False.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. While deep neural networks are all the rage, the complexity of the major frameworks has been a barrier to their use for developers new to machine learning.

There have been several proposals for improved and simplified high-level APIs for building neural network models, all of which tend to look similar from a distance but show differences on closer examination. Keras is one of the leading high-level neural networks APIs. It is written in Python and supports multiple back-end neural network computation engines. The biggest reasons to use Keras stem from its guiding principles, primarily the one about being user friendly. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options, integration with at least five back-end engines (TensorFlow, CNTK, Theano, MXNet, and PlaidML), and strong support for multiple GPUs and distributed training.

Keras does not do its own low-level operations, such as tensor products and convolutions; it relies on a back-end engine for that. Even though Keras supports multiple back-end engines, its primary (and default) back end is TensorFlow, and its primary supporter is Google. The Keras API comes packaged in TensorFlow as tf.keras, which as mentioned earlier will become the primary TensorFlow API as of TensorFlow 2.0.

To change back ends, simply edit your $HOME/.keras/keras.json file and specify a different back-end name, such as theano or CNTK. Alternatively, you can override the configured back end by defining the environment variable KERAS_BACKEND, either in your shell or in your Python code using the os.environ["KERAS_BACKEND"] property. The Model is the core Keras data structure. There are two main types of models available in Keras: the Sequential model, and the Model class used with the functional API.

Keras Sequential models: The Sequential model is a linear stack of layers, and the layers can be described very simply. Here is an example from the Keras documentation that uses model.add() to define two dense layers in a Sequential model: In the functional API you define the layers first, and then create the Model, compile it, and fit (train) it. Evaluation and prediction are essentially the same as in a Sequential model, so have been omitted in the sample code below.

```
import keras
from keras.models import Sequential
from keras.layers import Dense

#Create Sequential model with Dense layers, using the add method
model = Sequential()

#Dense implements the operation:
#        output = activation(dot(input, kernel) + bias)
#Units are the dimensionality of the output space for the layer,
#     which equals the number of hidden units
#Activation and loss functions may be specified by strings or classes
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

#The compile method configures the model's learning process
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#The fit method does the training in batches
# x_train and y_train are Numpy arrays --just like in the Scikit-Lear
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Keras has a wide selection of predefined layer types, and also supports writing your own layers. Core layers include Dense (dot product plus bias), Activation (transfer function or neuron shape), Dropout (randomly set a fraction of input units to 0 at each training update to avoid overfitting), Lambda (wrap an arbitrary expression as a Layer object), and several others. Convolution layers (the use of a filter to create a feature map) run from 1D to 3D and include the most common variants, such as cropping and transposed convolution layers for each dimensionality. 2D convolution, which was inspired by the functionality of the visual cortex, is commonly used for image recognition. Pooling (downscaling) layers run from 1D to 3D and include the most common variants, such as max and average pooling.

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Locally connected layers act like convolution layers, except that the weights are unshared. Recurrent layers include simple (fully connected recurrence), gated, LSTM, and others; these are useful for language processing, among other applications. Noise layers help to avoid over fitting.

Wide resnets:

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, in this work we conduct a detailed experimental study on the architecture of ResNet blocks, based on which we propose a novel architecture where we decrease depth and increase width of residual networks. We call the resulting network structures wide residual networks **(**WRNs**)** and show that these are far superior over their commonly used thin and very deep counterparts

```
class WideResNet:
    def __init__(self, image_size, depth=16, k=8):
        self._depth = depth
        self._k = k
        self._dropout_probability = 0
        self._weight_decay = 0.0005
        self._use_bias = False
        self._weight_init = "he_normal"

        if K.image_data_format() == "channels_first":
            logging.debug("image_dim_ordering = 'th'")
            self._channel_axis = 1
            self._input_shape = (3, image_size, image_size)
        else:
            logging.debug("image_dim_ordering = 'tf'")
            self._channel_axis = -1
            self._input_shape = (image_size, image_size, 3)

    # Wide residual network http://arxiv.org/abs/1605.07146
    def _wide_basic(self, n_input_plane, n_output_plane, stride):
        def f(net):
            # format of conv_params:
            #               [ [kernel_size=("kernel width", "kernel height"),
            #               strides="(stride_vertical,stride_horizontal)",
            #               padding="same" or "valid"] ]
            # B(3,3): orignal <<basic>> block
            conv_params = [[3, 3, stride, "same"],
                           [3, 3, (1, 1), "same"]]
```

5.3 Code fragments

5.3.1 Image preprocessing

Converting image to gray scale image and then to binary image to reduce the noise.

Gaussian filtering method can be used for noise reduction.

```
def __init__(self, image_size, depth=16, k=8):
    self._depth = depth
    self._k = k
    self._dropout_probability = 0
    self._weight_decay = 0.0005
    self._use_bias = False
    self._weight_init = "he_normal"

    if K.image_data_format() == "channels_first":
        logging.debug("image_dim_ordering = 'th'")
        self._channel_axis = 1
        self._input_shape = (3, image_size, image_size)
    else:
        logging.debug("image_dim_ordering = 'tf'")
        self._channel_axis = -1
        self._input_shape = (image_size, image_size, 3)
```

5.3.2 Normalization

Detecting the features of faces and cropping the rectangular features of faces.

```
for i, v in enumerate(conv_params):
    if i == 0:
        if n_input_plane != n_output_plane:
            net = BatchNormalization(axis=self._channel_axis)(net)
            net = Activation("relu")(net)
            convs = net
        else:
            convs = BatchNormalization(axis=self._channel_axis)(net)
            convs = Activation("relu")(convs)

        convs = Conv2D(n_bottleneck_plane, kernel_size=(v[0], v[1]),
                               strides=v[2],
                               padding=v[3],
                               kernel_initializer=self._weight_init,
                               kernel_regularizer=l2(self._weight_decay),
                               use_bias=self._use_bias)(convs)
    else:
        convs = BatchNormalization(axis=self._channel_axis)(convs)
        convs = Activation("relu")(convs)
        if self._dropout_probability > 0:
            convs = Dropout(self._dropout_probability)(convs)
        convs = Conv2D(n_bottleneck_plane, kernel_size=(v[0], v[1]),
                               strides=v[2],
                               padding=v[3],
                               kernel_initializer=self._weight_init,
                               kernel_regularizer=l2(self._weight_decay),
                               use_bias=self._use_bias)(convs)
```

5.3.3 Feature Extraction

A combination of global and grid features are extracted by geometric based and appearance based extraction methods.

```
def crop_face(self, imgarray, section, margin=40, size=64):
    """
    :param imgarray: full image
    :param section: face detected area (x, y, w, h)
    :param margin: add some margin to the face detected area to include a full head
    :param size: the result image resolution with be (size x size)
    :return: resized image in numpy array with shape (size x size x 3)
    """
    img_h, img_w, _ = imgarray.shape
    if section is None:
        section = [0, 0, img_w, img_h]
    (x, y, w, h) = section
    margin = int(min(w,h) * margin / 100)
    x_a = x - margin
    y_a = y - margin
    x_b = x + w + margin
    y_b = y + h + margin
    if x_a < 0:
        x_b = min(x_b - x_a, img_w-1)
        x_a = 0
    if y_a < 0:
        y_b = min(y_b - y_a, img_h-1)
        y_a = 0
    if x_b > img_w:
        x_a = max(x_a - (x_b - img_w), 0)
        x_b = img_w
    if y_b > img_h:
        y_a = max(y_a - (y_b - img_h), 0)
        y_b = img_h
    cropped = imgarray[y_a: y_b, x_a: x_b]
    resized_img = cv2.resize(cropped, (size, size), interpolation=cv2.INTER_AREA)
    resized_img = np.array(resized_img)
    return resized_img, (x_a, y_a, x_b - x_a, y_b - y_a)
```

5.4 Implementation Support

Feature Detection and its Extraction is considered to be an indispensable process which is needed to be followed and implemented to accomplish the task of gender detection and its classification. The facial feature which has been identified for extraction is the ‘lip’. The process of extraction of the selected feature involves the following steps. The Region of Interest (ROI) principle is followed. A specific window size is selected and is applied to the input facial image. The desired location is identified using the window dimension from where the required feature can be extracted.

The feature is then detected and extracted from the selected location using the imcrop () function. Each extracted image is then resized into a single vector of dimension 1581. A matrix containing the features of training images in its column wise is formed along with the class labels which is associated with each image. A class label of +1 is assigned to the female image and a class label -1 is assigned to the male image.

One of the first applications of convolutional neural networks (CNN) is perhaps the LeNet-5 network described for optical character recognition. Compared to modern

deep CNN, their network was relatively modest due to the limited computational resources of the time and the algorithmic challenges of training bigger networks. Though much potential laid in deeper CNN architectures (networks with more neuron layers), only recently have they became prevalent, following the dramatic increase in both the computational power (due to Graphical Processing Units), the amount of training data readily available on the Internet, and the development of more effective methods for training such complex models. One recent and notable examples is the use of deep CNN for image classification on the challenging Image net benchmark. Deep CNN have additionally been successfully applied to applications including human pose estimation, face parsing, facial key point detection, speech recognition and action classification. To our knowledge, this is the first report of their application to the tasks of age and gender classification from unconstrained photos.

Gathering a large, labeled image training set for age and gender estimation from social image repositories requires either access to personal information on the subjects appearing in the images (their birth date and gender), which is often private, or is tedious and time-consuming to manually label. Data-sets for age and gender estimation from real-world social images are therefore relatively limited in size and presently no match in size with the much larger image classification data-sets (e.g. the Imagenet dataset [45]). Over fitting is common problem when machine learning based methods are used on such small image collections. This problem is exacerbated when considering deep convolutional neural networks due to their huge numbers of model parameters. Care must therefore be taken in order to avoid over fitting under such circumstances.