IMAGE AND VIDEO PROCESSING MINI PROJECT REPORT

KRISHNA KODALI

17CS01008

SCHOOL OF ELECTRICAL SCIENCES,

COMPUTER SCIENCE & ENGINEERING.

CONVOLUTION NEURAL NETWORK FOR FACE ANALYSIS

ABSTRACT

The idea is to produce a Multipurpose CNN for doing simultaneous tasks which include face detection, gender detection, age detection along with ethnicity of that person using face analysis.

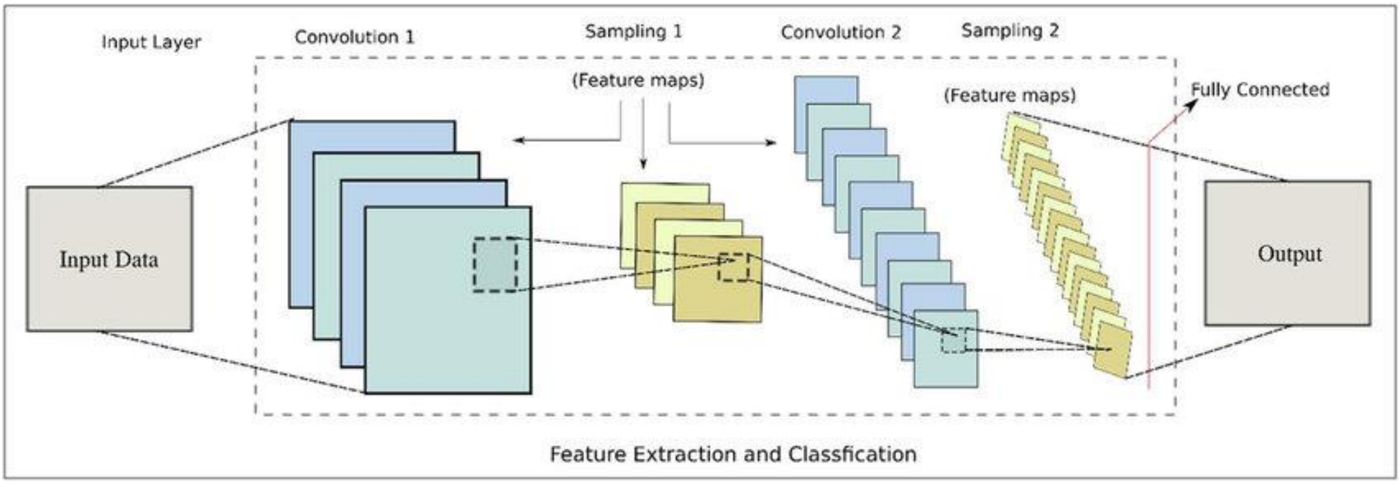
The aim is to make a single deep learning model which takes images as input and give the mentioned features detected in that image. For that we propose a novel CNN architecture for simultaneously performing tasks. The proposed method employs a multi-task learning framework that regularizes the shared parameters of CNN and builds a synergy among different domains and tasks.

THEORY

Neural networks are computing systems that tries to recognize underlying relationships in a set of data through a process that mimics the way neurons operate in the human brain.

An (Artificial Neural Network) ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain.

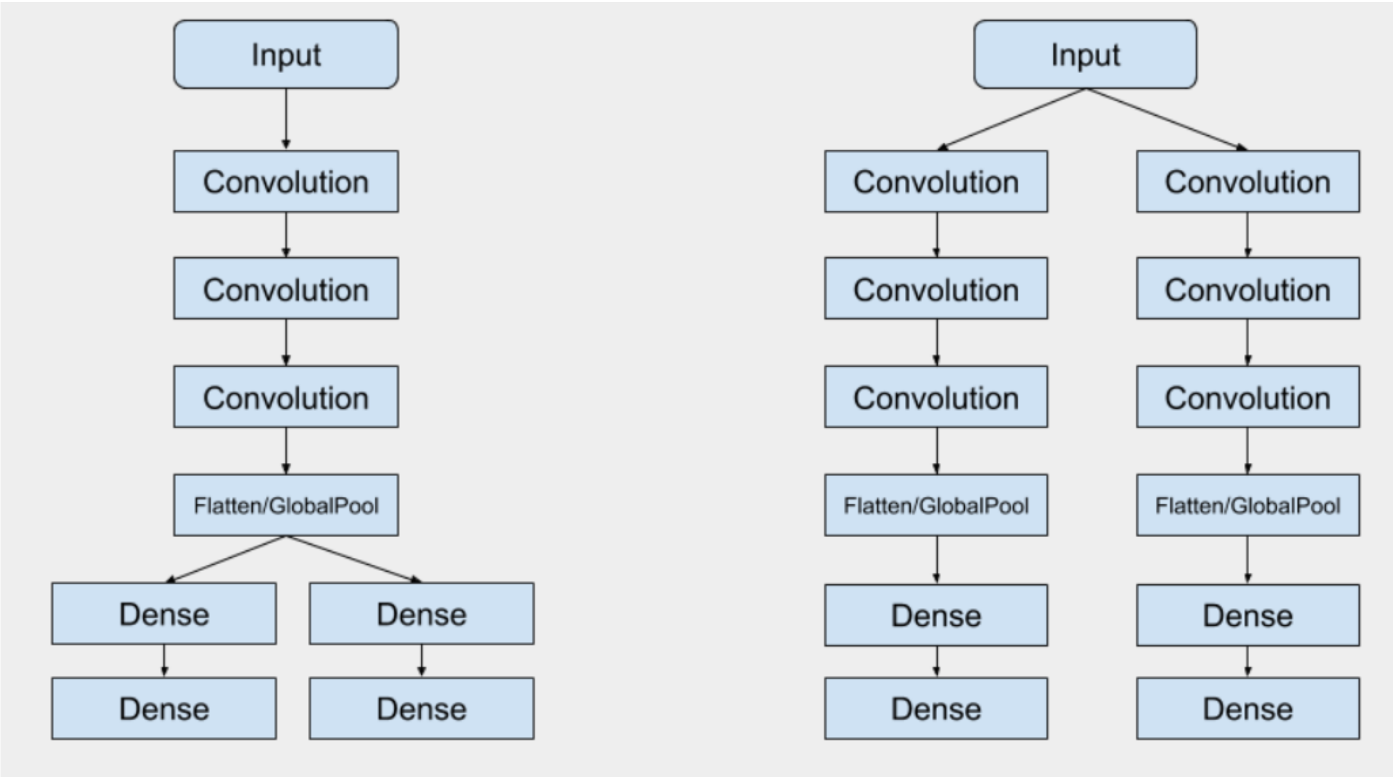
A Convolutional Neural Network conventionally called as CNN are a class of neural networks which deal with analysing visual imagery i.e. to understand the common features in a set of images and the correlations between them.



Layers in CNN

A neural network takes a data as input and learns from them by running epochs. By each epoch the model does back propagation to adjust the weights between the nodes in the hidden layers which helps to increase the accuracy on an overall basis.

Neural Networks can also produce multiple outputs at the same time i.e. to achieve multiple tasks simultaneously. Consider an example, where you want to detect age, gender, ethnicity and to determine some other features of the person in a given picture. The conventional way is to run multiple models on the dataset for getting the correlations of the corresponding attributes. By using Multi Task Learning we can develop a single model which can give us multiple outputs and the corresponding correlations.



Multi Taking Model vs Conventional CNN

In this way, the lower layers learn general representation common to all the tasks, whereas upper layers are more specific to the given task, this helps us to reduce the over-fitting layers. Thus, our model is able to learn robust features for distinct tasks. Employing multiple tasks enables the network to learn the correlations between data from different distributions in an effective way.

By this method we can reduce the amount of time required to train the model and also the memory consumed in relative to achieving the same in multiple models, since it can simultaneously solve the tasks and requires the storage of a single CNN model instead of separate CNN for each task.

METHODOLOGY

- **Data Pre-processing:** The dataset used is UTKFace. UTKFace is a large-scale dataset which consists of around 24,000 images with age, gender also the ethnicity of the person.
  The images are embedded with filename formatted in the form of [age]_[gender]_[race].
  [age] – Integer – (0-116)
  [gender] – Integer – 0(male) or 1(female)
  [race/ethnicity] – Integer – (0-4)
  0 – white, 1 – Black, 2 – Asian, 3 – Indian, 4 – Others
  We pre-process this data into a dataframe with columns named age, gender, ethnicity along with image. All the images are re-sized to (198, 198, 3) and normalized for obtaining better accuracies.

- **CNN:** We apply our multi task learning here. The initial layers remain the same but the final layers i.e. after convolution and pooling, we flatten the pixel values of the image and use these as the nodes for training our neural network.
  Filter size = 32, Kernel size = [3x3], activation = relu – For each Convolution Layer
  Pool Size = [2x2] – Max Pooling is applied to the output obtained after the convolution.

- **Training:** The dataset is split into training set, validation set and test set. The model is trained with 20 epochs at a batch size of 64.

- **Output:** Here we used functional API model of the keras instead of the Sequential model because in sequential model we only add one output layer by using Dense module at the end whereas by using the functional API model we can simultaneously add multiple output layers. For each output we use different activation functions depending on the different types of values.

  Age – Activation Function = Sigmoid

  Gender – Activation Function = Softmax

  Ethnicity – Activation Function = Softmax

  After adding the output layers, we compile our model using adam optimizer with different loss functions for different output layers. The Loss functions are as follows.

  Age: mse

  Gender: categorical_crossentropy

  Ethnicity: categorical_crossentropy

EXPERIMENT RESULTS

Results obtained after running the all the epochs.

```
Classification report for race
             precision    recall  f1-score   support


          0       0.84      0.83      0.83        58
          1       0.69      0.93      0.79        29
          2       0.82      0.88      0.85        16
          3       0.70      0.44      0.54        16
          4       0.20      0.11      0.14         9

avg / total       0.74      0.76      0.74       128



Classification report for gender
             precision    recall  f1-score   support


          0       0.84      0.99      0.91        77
          1       0.97      0.73      0.83        51

avg / total       0.90      0.88      0.88       128
```

The model is 87 percent in predicting the gender and 77 percent in predicting the ethnicity. Exact age of the person cannot be predicted but the range of his/her age is predicted accurately using this model.

DISCUSSION

With the help of models like Multi Task Learning, we can obtain correlations for multiple outcomes from a single neural network. In this model we were able to identify a face in the image and then were able to determine the age, gender and ethnicity of that person from the obtained face analysis. All these factors were predicted using a single network.

CONCLUSION

Here using the CNN model, we were able to compute the age, gender and ethnicity of that person using face analysis. The dataset used here is UTKFace dataset and the age of the person is predicted accurately i.e. the range of age group a person belongs to. The accuracy for gender prediction is around 87 and ethnicity is around 77. For increasing the accuracy, we need to get a dataset with a greater number of pixels and also train our model using high-performance GPUs.

REFERENCES

https://ieeexplore.ieee.org/abstract/document/7961718 - Reference IEEE Conference Paper

https://susanqq.github.io/UTKFace - UTFFace Dataset

https://en.wikipedia.org/wiki/Convolutional_neural_network - Convolutional Neural Networks

https://en.wikipedia.org/wiki/Multi-task_learning - Multi Task Learning

https://keras.io/guides/functional_api - For building CNN Layers

APPENDIX(Python Code)

```python
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import os
import glob

#Specifying he directory of the dataset
Image_Dir = r"E:\Sem 7\IVP\Mini Project\Python-1\UTKFace"
Train_Size = 0.7
I_W = I_H = 198

Gender = {0: 'male', 1: 'female'}
Gender_Map = dict((g, i) for i, g in Gender.items())
Race = {0: 'white', 1: 'black', 2: 'asian', 3: 'indian', 4: 'others'}
Race_Map = dict((r, i) for i, r in Race.items())

def parse_filepath(filepath):
    ''''' For Extracting Images in the mentioned Directory'''
    global Gender, Race
    try:
        path, filename = os.path.split(filepath)
        filename, ext = os.path.splitext(filename)
        age, gender, race, _ = filename.split("_")
        return int(age), Gender[int(gender)], Race[int(race)]
    except Exception as e:
        print(filepath, e)
        return None, None, None

files = glob.glob(os.path.join(Image_Dir, "*.jpg"))

attributes = list(map(parse_filepath, files))

#Making a dataframe with images as input and corresponding features as output.
df = pd.DataFrame(attributes)
df['file'] = files
df.columns = ['age', 'gender', 'race', 'file']
df = df.dropna()
df.head()

df.groupby(by=['race', 'gender'])['age'].count().plot(kind='bar')

#Making a permutation of the dataset and splitting into training and test set.
Len = len(df)
Permutation = np.random.permutation(Len)
Training_Size = int((Len)*(Train_Size))
Training_Set = Permutation[:Training_Size]
Test_Set = Permutation[Training_Size:]

Training_Size = int(Train_Size * 0.7)
Training_Set, Validation_Set = Training_Set[:Training_Size], Training_Set[Training_Size:]

df['gender_id'] = df['gender'].map(lambda gender: Gender_Map[gender])
```

```python
df['race_id'] = df['race'].map(lambda race: Race_Map[race])

Max_Age = df['age'].max()

from keras.utils import to_categorical
from PIL import Image

def get_data_generator(df, indices, for_training, batch_size=16):
    ''''Normalizing the image pixels and converting into categorical variables.'''
    global I_W, I_H, Max_Age
    images, ages, races, genders = [], [], [], []
    while True:
        for i in indices:
            r = df.iloc[i]
            file, age, race, gender = r['file'], r['age'], r['race_id'], r['gender_id']
            im = Image.open(file)
            im = im.resize((I_W, I_H))
            im = np.array(im) / 255.0
            images.append(im)
            ages.append(age / Max_Age)
            races.append(to_categorical(race, len(Race_Map)))
            genders.append(to_categorical(gender, 2))
            if len(images) >= batch_size:
                yield np.array(images), [np.array(ages), np.array(races), np.array(genders)]
                images, ages, races, genders = [], [], [], []
        if not for_training:
            break

#Importing libraries necessary to build the CNN
import tensorflow as tf
from keras.layers import Input, Dense, BatchNormalization, Conv2D, MaxPool2D, GlobalMaxPool2D
from keras.optimizers import SGD
from keras.models import Model

#Specifying the sizeof the input images so that they can be reshaped to the specified size.
input_layer = Input(shape=(I_H, I_W, 3))

#Convolution Layers followed by a max pooling
C1 = Conv2D(32, kernel_size = (3,3), strides = (1,1), activation = 'relu')(input_layer)
M1 = MaxPool2D(pool_size=(2,2), strides=(2,2))(C1)

C2 = Conv2D(32*2, (3,3), activation = 'relu')(M1)
M2 = MaxPool2D(pool_size=(2,2))(C2)

C3 = Conv2D(32*4, (3,3), activation = 'relu')(M2)
M3 = MaxPool2D(pool_size=(2,2))(C3)

C4 = Conv2D(32*4, (3,3), activation = 'relu')(M3)
M4 = MaxPool2D(pool_size=(2,2))(C4)

C4 = Conv2D(32*5, (3,3), activation = 'relu')(M3)
M4 = MaxPool2D(pool_size=(2,2))(C4)
```

```
    C5 = Conv2D(32*6, (3,3), activation = 'relu')(M3)
    M4 = MaxPool2D(pool_size=(2,2))(C5)

    bottleneck = GlobalMaxPool2D()(M4)

    #Creating output layers
    C = Dense(units=128, activation='relu')(bottleneck)
    age_output = Dense(units=1, activation='sigmoid', name='age_output')(C)

    C = Dense(units=128, activation='relu')(bottleneck)
    race_output = Dense(units=len(Race_Map), activation='softmax', name='race_output')(C)

    C = Dense(units=128, activation='relu')(bottleneck)
    gender_output = Dense(units=len(Gender_Map), activation='softmax', name='gender_output')(C)

    model = Model(inputs=input_layer, outputs=[age_output, race_output, gender_output])

    #Compiling our model with adam optimizer.
    model.compile(optimizer='adam',
                  loss={'age_output': 'mse', 'race_output': 'categorical_crossentropy', 'gender_output': 'categorical_crossentropy'},
                  loss_weights={'age_output': 2., 'race_output': 1.5, 'gender_output': 1.},
                  metrics={'age_output': 'mae', 'race_output': 'accuracy', 'gender_output': 'accuracy'})

    #Extracting training and test sets from the dataframe.
    batch_size = 64
    valid_batch_size = 64
    train_gen = get_data_generator(df, Training_Set, for_training=True, batch_size=batch_size)
    valid_gen = get_data_generator(df, Validation_Set, for_training=True, batch_size=valid_batch_size)

    from keras.callbacks import Callback

    class MyLogger(Callback):
        ''''For logging during each epoch'''
        def on_epoch_end(self, epoch, logs=None):
            with open('log.txt', 'a+') as f:
                f.write('%02d %.3f\n' % (epoch, logs['loss']))

    mylogger = MyLogger()

    #Training our model upon training set and applying the correlations on validation set.
    output = model.fit(train_gen,
                        steps_per_epoch=len(Training_Set)//batch_size,
                        epochs=20,
                        verbose = 1,
                        validation_data=valid_gen,
                        validation_steps=len(Validation_Set)//valid_batch_size)

    test_gen = get_data_generator(df, Test_Set, for_training=False, batch_size=128)
```

```python
    dict(zip(model.metrics_names, model.evaluate_generator(test_gen, steps=len(Test_Set)//128)))

    test_gen = get_data_generator(df, Test_Set, for_training=False, batch_size=128)
    x_test, (age_true, race_true, gender_true)= next(test_gen)
    age_pred, race_pred, gender_pred = model.predict_on_batch(x_test)

    race_true, gender_true = race_true.argmax(axis=-1), gender_true.argmax(axis=-1)
    race_pred, gender_pred = race_pred.argmax(axis=-1), gender_pred.argmax(axis=-1)
    age_true = age_true * Max_Age
    age_pred = age_pred * Max_Age

    #For getting the accuracies achiened dur
    from sklearn.metrics import classification_report
    print("Classification report for race")
    print(classification_report(race_true, race_pred))

    print("\nClassification report for gender")
    print(classification_report(gender_true, gender_pred))

    #Applying our correlations on some of the images for comparing actual values with predicted values
    import math
    n = 30
    random_indices = np.random.permutation(n)
    n_cols = 5
    n_rows = math.ceil(n / n_cols)
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 20))

    for i, img_idx in enumerate(random_indices):
        ax = axes.flat[i]
        ax.imshow(x_test[img_idx])
        ax.set_title('a:{}, g:{}, r:{}'.format(int(age_pred[img_idx]), Gender[gender_pred[img_idx]], Race[race_pred[img_idx]]))
        ax.set_xlabel('a:{}, g:{}, r:{}'.format(int(age_true[img_idx]), Gender[gender_true[img_idx]], Race[race_true[img_idx]]))
        ax.set_xticks([])
        ax.set_yticks([])
```