Xiyuan Guo (Charles) 400347723

Hua Yao (Edward) 400344368

SFWRTECH 4NN3 Neural Networks and Deep Learning

Final Project

## Load Data:

All data includes the training image set and testing image set, which is provided by the professor. In the training dataset, we have totally 13233 people head images. All the images are uniformly 250\*250 big. When we read each image, we read them as grayscale image and resize the image to 200\*200 to avoid unecessary computation. The images are fully diverse regarding the background, people’s age, people’s gender and cloth etc. This give us a good training data set to obtain a model with more generalizabiltiy. When we read each training testing image, we read them into an array to feed into model later on. In the testing dataset, we have 7 testing images. These are images contains a group of people with varied sizes. Our task is to train a model using the training data and then use the obtained trained model to detect the people faces in each testing image.

## Training:

In the training part, we name the model we trained as Face\_Detection-CNN-{}. Then, we enable a tensorboard tool to monitor our training process.

Next, we select 1 dense layer, 3 convolution layers each with 32 layer size of 7\*7 filter. It satisfied the requirements.

In the main body, we use conv2D to calculate convolution, add an activation layer of rectified linear unit and use maxpooling2D with pool size of (2,2) which is also required. Before we feed it to the neuron networks, we flat it by flatten() function and add the dense layer on it. Finally we use the sigmoid function to as the activattion function.

We use model.compile and model.fit to train the model. Some important parameters we will explained here. We use 32 as batch size, we ran 10 epochs in total, we use 0.3 as validation split which means 70 percent of data will be used for training and 30 percent of data will be used for validation.

Finally, we use the model.save to save this model for the next testing part.

## Testing:

In the testing part, we first use tensorflow.keras.models.load\_model() loaded our saved model from the training part. This will be used to obtain the prediction results from our testing images.

Next , we read in the testing images using cv2.imread() function. To avoid unnecessary computation complexity, we read the images as grayscale format.

To provide the ground truth, we use the website tool <https://onlineimagetools.com/pixelate-image> to get the faces’ coordinates in test images. For example, for the astronauts image, we have 23 coordinates corresponding to each face. The labels of these 23 positions were set to one. All other positions value were labeled as zero.

To detect the faces in the testing images, we first scan the whole testing image to extract window images. Tensorfolow.image.extract\_patches() function is used here. During the extraction, we could set proper image size, extraction stride and extraction rates etc. For example, we could set window size as 60\*60, stride as 5\*5 and rates as 1 on the 640\*280 astronauts image. We then applied the model on each of the window image to detect a face in the window image. The model will give us a binary result as 1 or 0 for face detection.

For the multiple detection of the same face problem, we adjusted the prediction result. We assumed if a face of ground truth has been detected, the prediction of one values nearby the ground truth will be adjusted to zero, except for the ground truth itself. For example, in the astronauts image, we used a range of 5 windows for same face detection. Basically, that means we only detect the face of ground truth only for once in a small area of 5 contentious windows both vertically and horizontally around the ground truth. Other than this, all the other individual prediction results will be kept from the model prediction.

Having this adjusted prediction result array and the labeled ground truth array, we then can utilize the sklearn.metrics.confusion metrix() function to plot our confusion matrix, which give us a clear view of the model performance. Chart, treemap chart

Description automatically generated Text

Description automatically generated