Final-Project-Group-2-Indvisual-Report

Jingya Gao

Introduction:

When there is an influx of a large number of people it can cause accidents such as a stampede. During sports events, large stadiums are full of people, this project can be used in order to analyze and predict the number of people in each area and prevent any accidents from taking place. It can also be used to analyze the number of people in a mall in order to prevent overcrowding and avoid mishaps from taking place. The relevance of this project can be seen in 2020, when there is an outbreak of a virus, it is mandatory to remain indoors, this project can be used to alert authorities when the number of people goes above a certain threshold. The goal of this project is to be able to predict the number of people in each image.

In this report, we describe the dataset used, the architecture of the two models, explanations of our results, and finally evaluation of our models.

Dataset:

The dataset consists of a collection of 2000 images, each image has a size of 480x640 pixels and has 3 channels. The images were collected from a still webcam that is located inside a mall. Each image has a different number of people. We are provided with a CSV file containing labels that give the number of people present in each image.

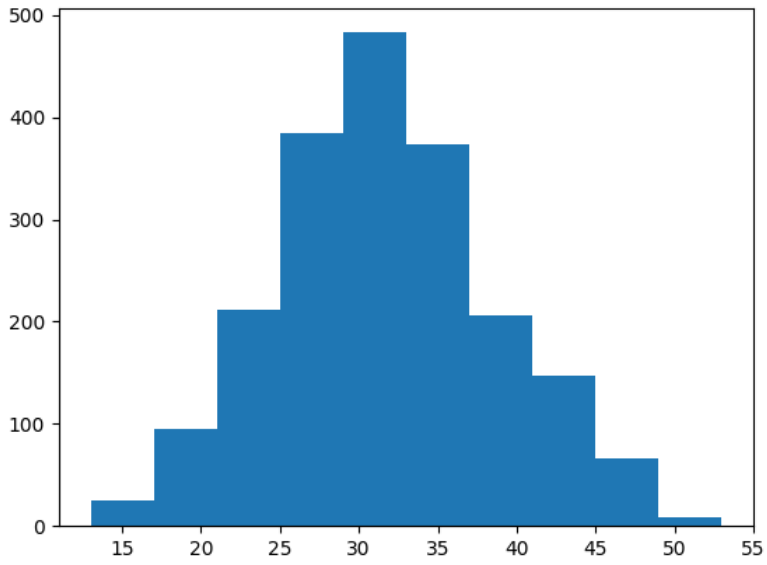


Fig 1. Distribution of labels

The image data is displayed below.

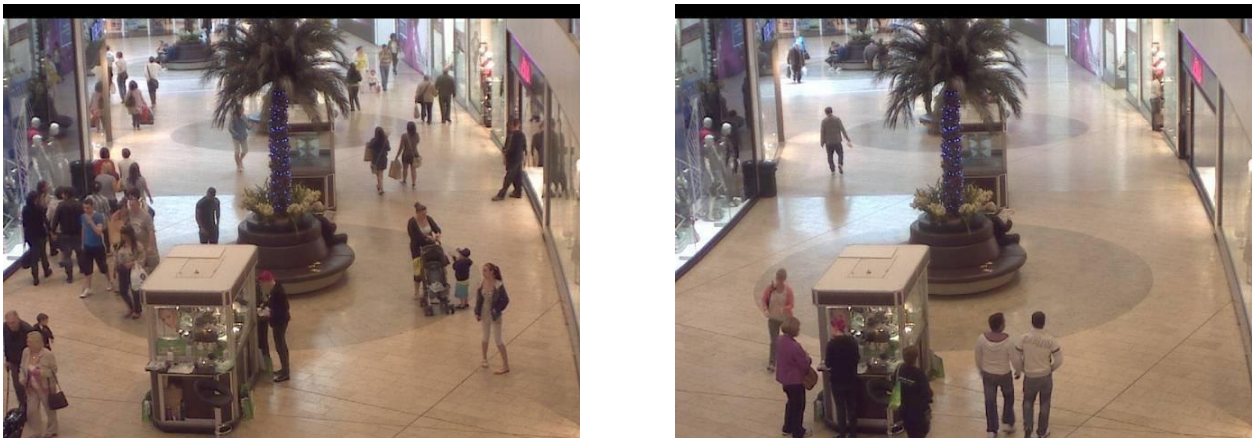


Fig 2. Image data that is input to the training models

Data Cleaning and Preprocessing:
The data is preprocessed and resized to 128x96 in order to store and train the data easily.

```
x = []
for path in [f for f in os.listdir(DATA_DIR) if f[-4:] == ".jpg"]:
    x.append(cv2.resize(cv2.imread(DATA_DIR + path), (RESIZE_TO, RESIZE_TO)))
x = np.array(x)
print("x.shape:\n", x.shape)
```

Fig 3. Code to pre-process data

As the data is not evenly distributed, we introduce an image generator.

```
# add ImageDataGenerator
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False,
    shear_range=0.5)
```

Fig 4. Code for image augmentation

Architecture of the models:

Background:

For this project, we are using CNN and a pre-trained ResNet 50 model. I'm in charge of using the CNN model. it's a fully connected feed-forward neural network. CNN is very effective in reducing the number of parameters and maintain the image quality at the same time. By adding a regression layer at the end of the network, we are able to fit the model with this regression problem.

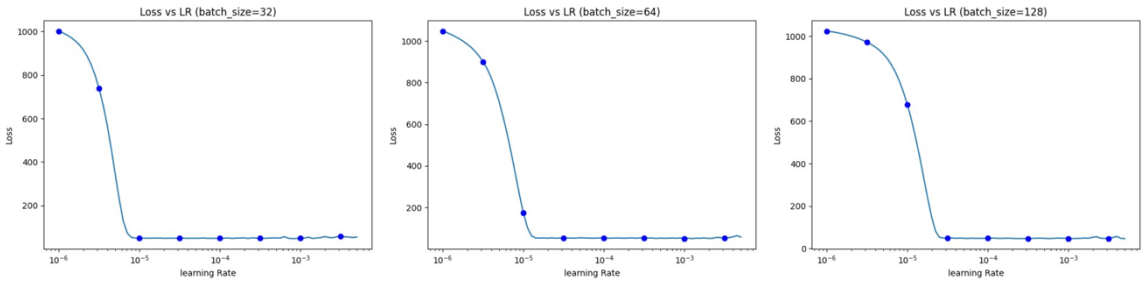


Fig 5. Learning rate vs Loss

The batch size is selected as 32. As we can see in the figures below the loss value drops quickly and considerably for batch size 32, while for batch size 64 and 128 it takes longer for the loss values to drop.

Training CNN model:

While training the CNN model I make use of relu as the activation function and Adam as the optimizer function.

```
52  # create CNN model
53  model = tf.keras.Sequential([
54
55      tf.keras.layers.Conv2D(16, (5, 5), input_shape=(128, 96, 3), activation=tf.keras.activations.
56      tf.keras.layers.MaxPool2D(2, 2),
57      tf.keras.layers.Conv2D(32, (3, 3), activation=tf.keras.activations.relu),
58      tf.keras.layers.MaxPool2D(2, 2),
59      tf.keras.layers.Dropout(0.2),
60      tf.keras.layers.Flatten(),
61      # tf.keras.layers.Dense(400, activation=tf.keras.activations.relu),
62      # tf.keras.layers.Dropout(0.2),
63      tf.keras.layers.Dense(1)
64
65  ])
66
67  model.compile(loss="mean_squared_error",  # This is a classic regression score - the lower the be
68                metrics=['mean_absolute_error'],
69                optimizer=tf.keras.optimizers.Adam(lr=3e-4))
```

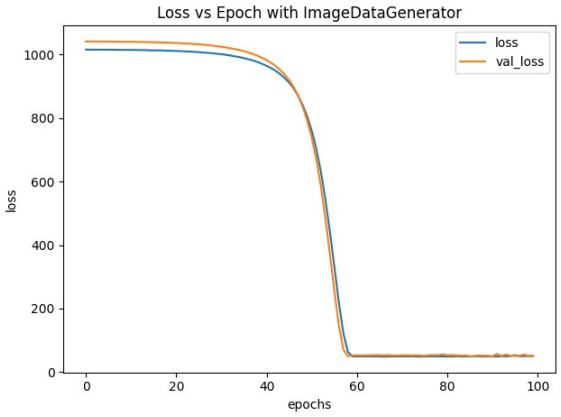
Fig 6. Code to compile model

For fitting the model, I use 100 epochs, increasing the number of epochs cause the model to overfit the data, therefore we chose 100 epochs. The final model was trained on a batch size of 32.

```
epochs = 100
datagen.fit(x_train)
# fits the model on batches with real-time data augmentation:
history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
                    steps_per_epoch=len(x_train) / 32, epochs=epochs,
                    validation_data=(x_test, y_test),
                    callbacks=[lr_monitor, model_check],
                    shuffle=True)
```

Fig 7. Code to fit the model

Evaluation:



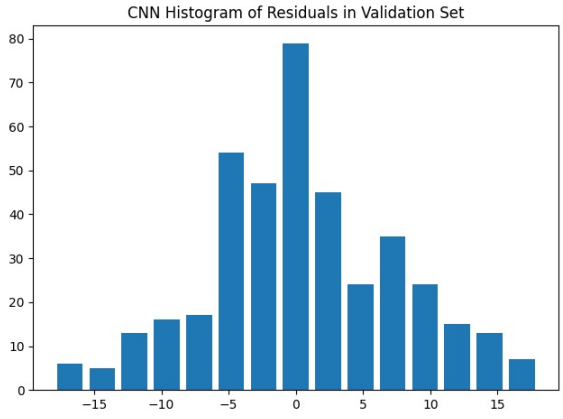


Fig 8. CNN Training loss & validation loss

Fig 9. CNN hist of residuals in validation set

In Fig 8. the blue line represents the performance of the model in the training data, and the orange line represents the training results of the model in the test data. It can be known that the model successfully achieved the learning function during training, and after 100 epochs, the training loss and testing loss were reduced to about 49.

From the histogram in Fig 9, we can see that the residuals are relatively normally distributed for both models, with a mean around 0. It accurately predicts 31 images. For the remaining pictures without accurate prediction of count, the error between the value of count predicted by this model and the actual value is mostly relatively small.

Prediction results:

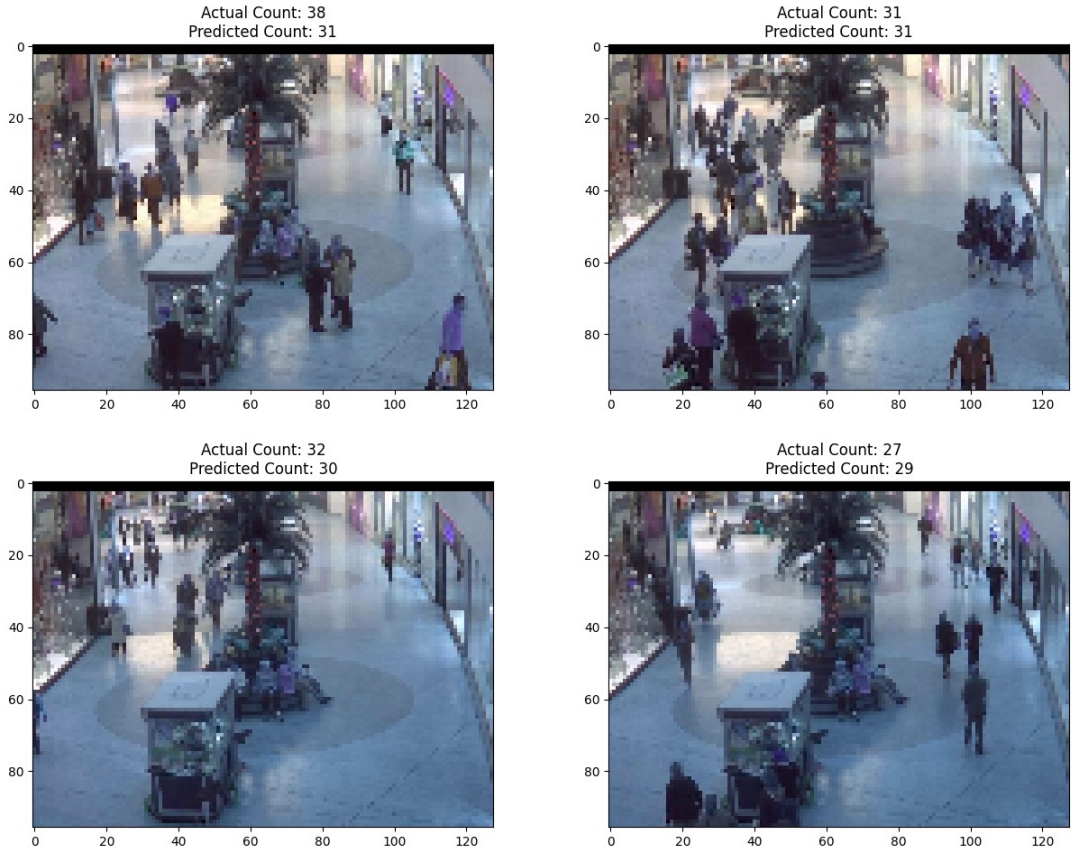


Fig 10. CNN model predicts results vs actual results

Conclusion:

It can be seen from the results that the CNN model designed this time can successfully predict the number of people in the picture. In Fig. 8. CNN Training loss & validation loss, the blue line represents the performance of the model in the training data, and the orange line represents the training results of the model in the test data. From the trends of the two curves, it can be known that the model successfully achieved the learning function during training, and after 100 epochs, the training loss and testing loss were reduced to about 49. However, from the picture shown in Fig 10. CNN model predicts results vs actual results, we can also find that the predicted results still have partial deviations, so this model has room for improvement. This improvement can be used as a future research direction.

percentage of the code that found or copied from the internet:
100*(78-33)/(78+41) = 37.8%

References:

https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be_fig3_331364877
https://www.kaggle.com/shovalt/crowd-counting-keras-pretrained-resnet50-cnn
https://www.kaggle.com/rmishra258/counting-crowd-with-cnn-social-distancing-project
https://www.mathworks.com/help/deeplearning/ref/resnet50.html#:~:text=ResNet%2D50%20is%20a%20convolutional,%2C%20pencil%2C%20and%20many%20animals.