Individual Report

Wenyu Zeng

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.

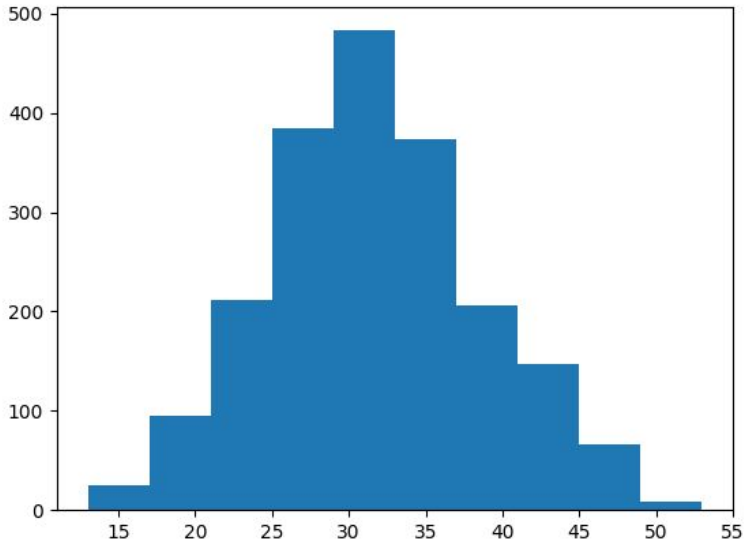


Fig 1. Distribution of labels

The image data is displayed below.

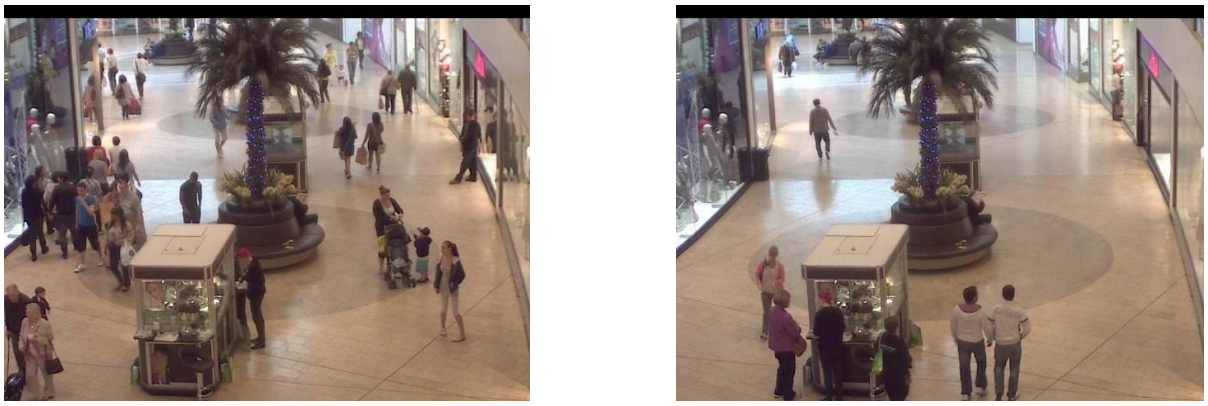


Fig 2. Image data that is input to the training models

Data Cleaning and Preprocessing:

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

```
41
42  x = []
43  for path in [f for f in os.listdir(DATA_DIR) if f[-4:] == ".jpg"]:
44      x.append(cv2.resize(cv2.imread(DATA_DIR + path), (RESIZE_TO, RESIZE_TO)))
45  x = np.array(x)
46  print("x.shape:\n", x.shape)
47
```

Fig 3. Code to pre-process data

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

```
75
76  # add ImageDataGenerator
77  datagen = ImageDataGenerator(
78      featurewise_center=False,
79      samplewise_center=False,
80      featurewise_std_normalization=False,
81      samplewise_std_normalization=False,
82      zca_whitening=False,
83      rotation_range=30,
84      width_shift_range=0.1,
85      height_shift_range=0.1,
86      horizontal_flip=True,
87      vertical_flip=False,
88      shear_range=0.5)
89
```

Fig 4. Code for image augmentation

Description of individual work:

In this project, I work on the ResNet 50 model, ResNet 50 is a 50 layers Residual Network. I change the output layer to make the model predict the number of people in the image, which fits this regression problem. The ResNet 50 model differs from the ordinary model in that it's using skip or shortcut connections. With the shortcut connections, the model measures the residuals calculate by each layer, and make sure the higher layer performs at least as good as the lower layer, and not worse. Below shows an example of the shortcut connection:

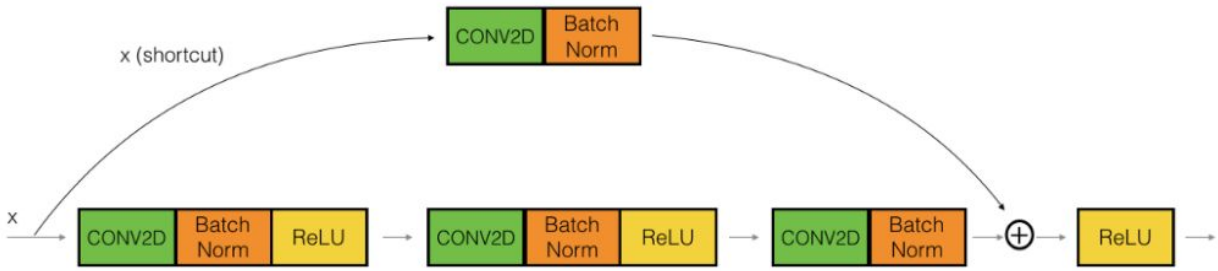


Fig 5. Example of shortcut connection

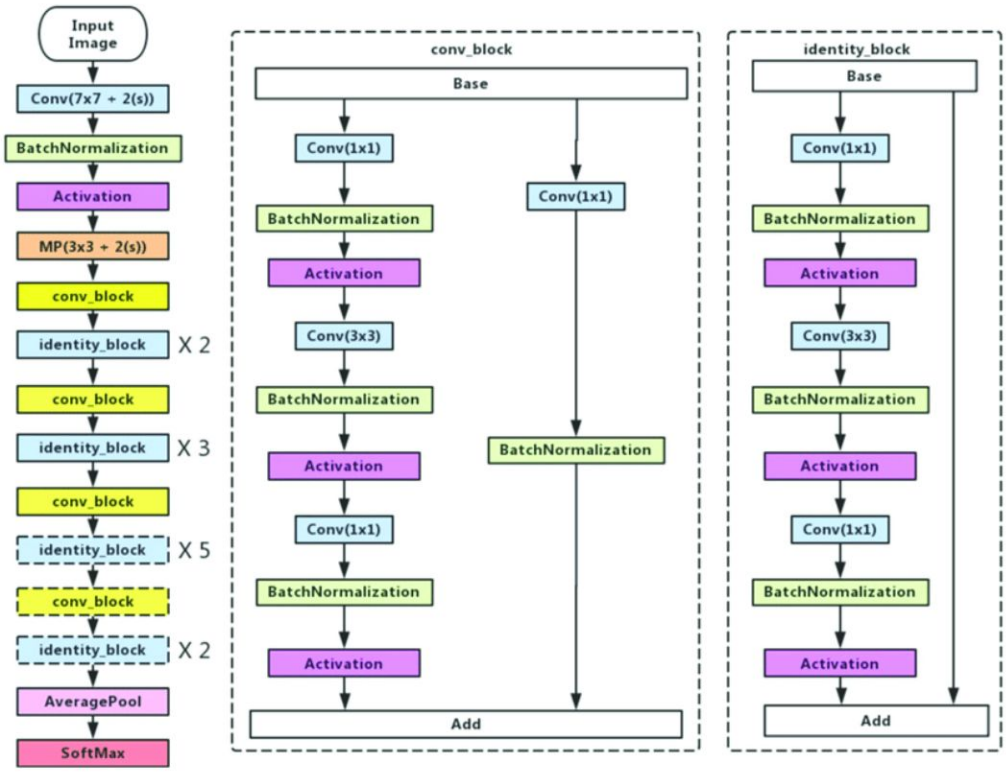


Fig 6. Sample ResNet-50 Architecture

Details:

Here shows the details for training the ResNet 50 model. The data is set up by using the “flow from dataframe” function. The data is split into training and validation.

```
datagen = ImageDataGenerator(
    rescale=1. / 255,
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2,

    preprocessing_function=resnet50.preprocess_input
)

flow_params = dict(
    dataframe=label,
    directory='/home/ubuntu/Deep-Learning/Final_Project/frames/frames',
    x_col="image_name",
    y_col="count",
    weight_col=None,
    target_size=(128, 96),
    color_mode='rgb',
    class_mode="raw",
    batch_size=batch,
    shuffle=True,
    seed=0
)

# The dataset is split to training and validation sets
train_generator = datagen.flow_from_dataframe(
    subset='training',
    **flow_params
)
valid_generator = datagen.flow_from_dataframe(
    subset='validation',
    **flow_params
)
```

Fig 7. Code snippet for data splitting

Training the model, the top seven layers, the last parts of the pre-trained model, are made untrainable to avoid overfitting and also to fit this regression problem. The added one dense layer and one final output layer that gives out one parameter, the prediction of the number of people in the image.

```
base_model = resnet50.ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(128, 96, 3),
    pooling='avg'
)

# Change the top (the last parts) of the network.
x = base_model.output
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='linear')(x)
model = Model(inputs=base_model.input, outputs=predictions)
k = -7
for layer in model.layers[:k]:
    layer.trainable = False
print('Trainable:')
for layer in model.layers[k:]:
    print(layer.name)
    layer.trainable = True
```

Fig 8. Compiling ResNet 50 model

Finally to compile the model I make use of the Adam optimizer. For training the model, I apply a learning rate monitor that focuses on the validation mean squared error. For every 3 epochs, if the mean squared error doesn't go down, the learning rate will drop by 20%, the lowest learning rate will not go beyond 1e-6.

```
optimizer = Adam(learning_rate=3e-4)

model.compile(
    optimizer=optimizer,
    loss="mean_squared_error",  # This is a classic regression score - the lower the better
    metrics=['mean_absolute_error', 'mean_squared_error']
)

learning_rate_reduction = ReduceLROnPlateau(
    monitor='val_mean_squared_error',
    patience=3,
    verbose=1,
    factor=0.2,
    min_lr=0.000001
)

model_check = ModelCheckpoint('resnet50_cc.hdf5', monitor='val_loss', verbose=2, save_best_only=T

# Fit the model
history = model.fit(
    train_generator,
    epochs=105,
    validation_data=valid_generator,
    verbose=2,
    callbacks=[learning_rate_reduction, model_check]
)
```

Fig 9. Optimizer and training code

Results:

Model	Training Loss (Mean Squared Error)	Training Mean Absolute Error	Validation Loss (Mean Squared Error)	Validation Mean Absolute Error
ResNet 50	20.26	3.59	36.31	4.92

Fig 10. Table with metrics evaluations

The above table shows the training and validation loss values and the mean absolute error for both sets. The losses are set as mean squared errors. As the table shows, the mean absolute error is pretty small, so we expect we have well trained the model. The last seven layers were also changed back and forth to test if the model has improved. However, if we increased the number of trainable layers, the model will overfit the data because the validation loss will increase after several epochs.

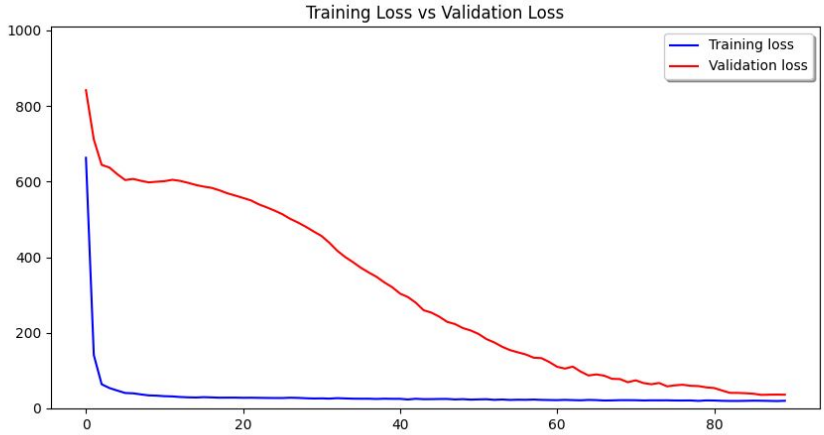
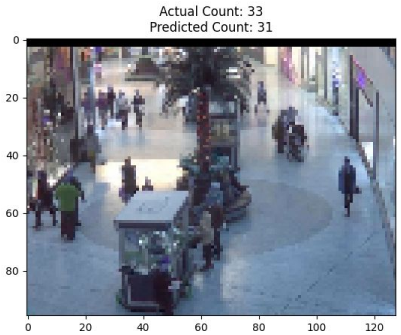


Fig 11. ResNet50 Training loss & validation loss

The above figure shows the training and validation loss after 90 epochs. We can see the validation loss keep decreasing, therefore, we are not overfitting the data, and the model has improved every epoch.

Prediction results:



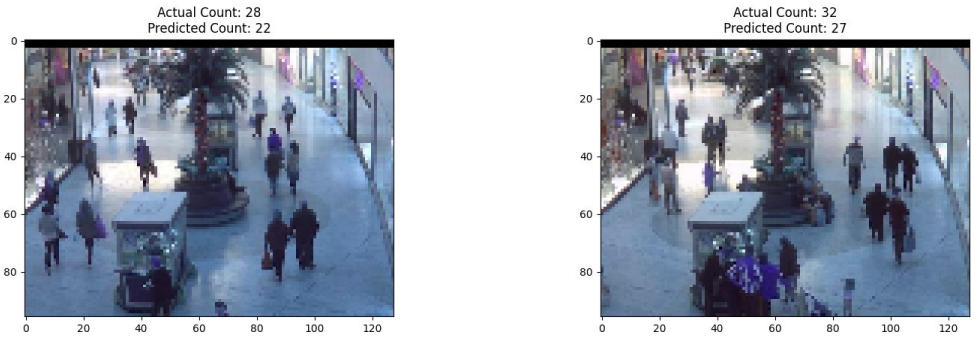


Fig 12. ResNet 50 model predict results vs actual results

The above four figures show the actual count and predicted count of people in the image. We can see that the largest residuals within the four figures are 6, which is not a very big difference.

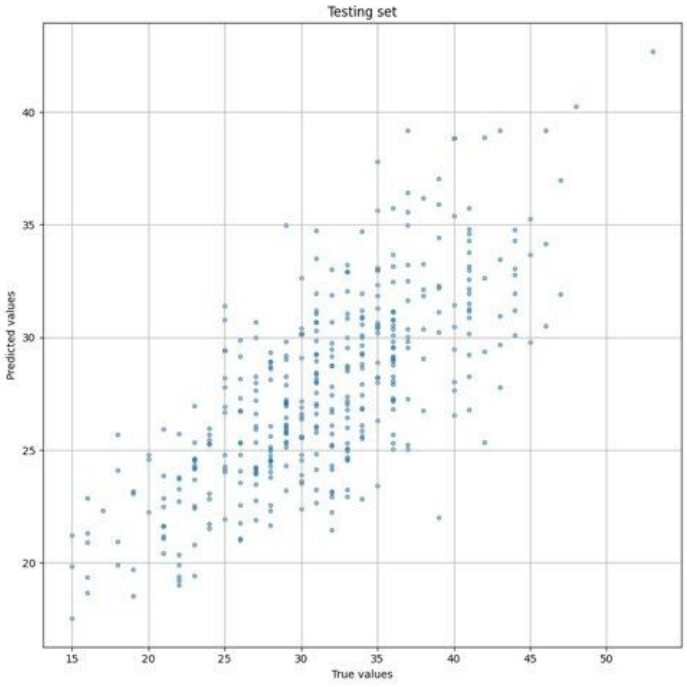


Fig 13. Predicted values vs. True values in validation set

The above scatter plot shows a comparison between the true counts and the predicted counts. The process is done by resetting the validation generator. We can see a clearly strong positive correlation between the two values, which means the model is well trained, and are able to produce results with high accuracy.

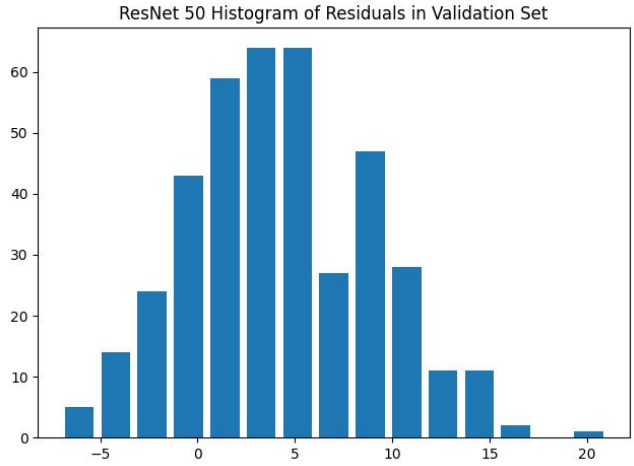


Fig 14. Residuals of True Values and Predicted Values in validation set

From the above histogram, we can see that the residuals are relatively normally distributed, with a mean around 5. Most of the values are spread out between 0 and 10. The model accurately predicted 24 images in the validation set with small differences between the true counts and the predicted counts.

Conclusion:

Based on the results, the pre-trained ResNet 50 model acts pretty well for this image regression problem. For future improvements, the MCNN (Multi-Scale Convolutional Neural Networks) model could be a better model because it's designed to count a small number of people inside an image. This dataset is predicting numbers of people in the mall from one camera, we can also see from the distribution of the labels, the countings are not very big. Therefore, the MCNN model might be a better fit for the dataset.

Percentage of the code found from the internet: 100*(88-13)/(88+23) = 67.57%

References:

1. Dwivedi, P. (2019, March 27). Understanding and Coding a ResNet in Keras. Retrieved December 07, 2020, from https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
2. Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images - Scientific Figure on ResearchGate. Retrieved December 07, 2020, from https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be_fig3_331364877

3. T, S. (2020, August 18). Crowd Counting - Keras pretrained ResNet50 CNN. Retrieved December 07, 2020, from https://www.kaggle.com/shovalt/crowd-counting-keras-pretrained-resnet50-cnn