

The George Washington University

Crowd Counting

Machine Learning II- Deep Learning

Anwasha Tomar, Jingya Gao, Wenyu Zeng
12-1-2020

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.

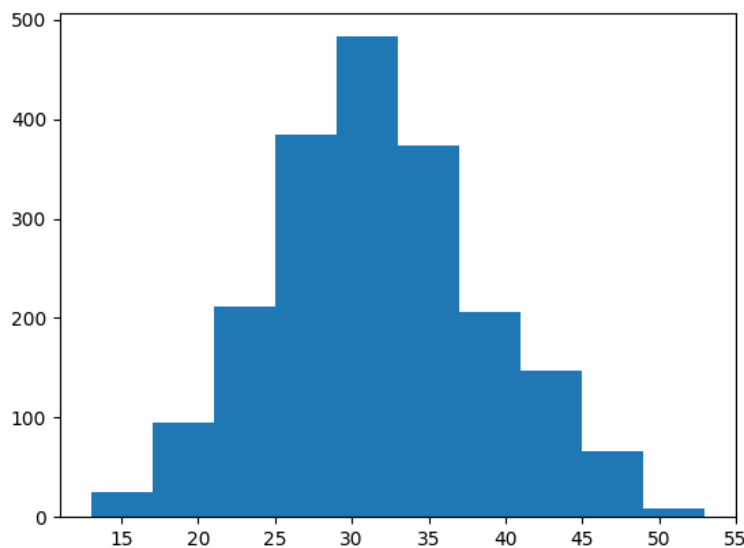


Fig 1. Distribution of labels

The image data is displayed below.

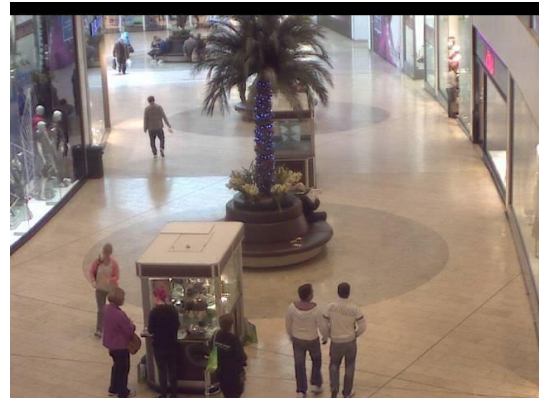
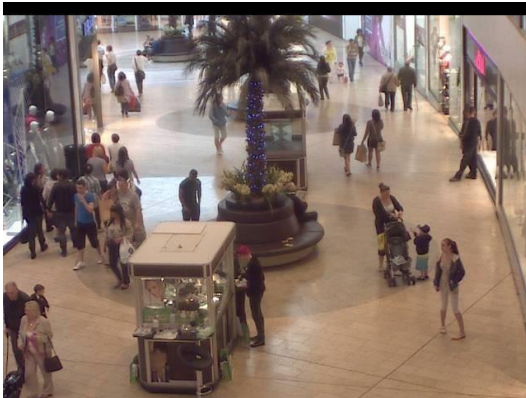


Fig 2. Image data that is input to the training models

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

Architecture of the models:

Background:

For this project, we are using CNN and a pre-trained ResNet 50 model. We are using the CNN model mainly because 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.

A similar reason is used for the pre-trained ResNet 50 model. ResNet 50 is a 50 layers Residual Network. We can also change the output layer to make the model predict the number of people in the image. 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.

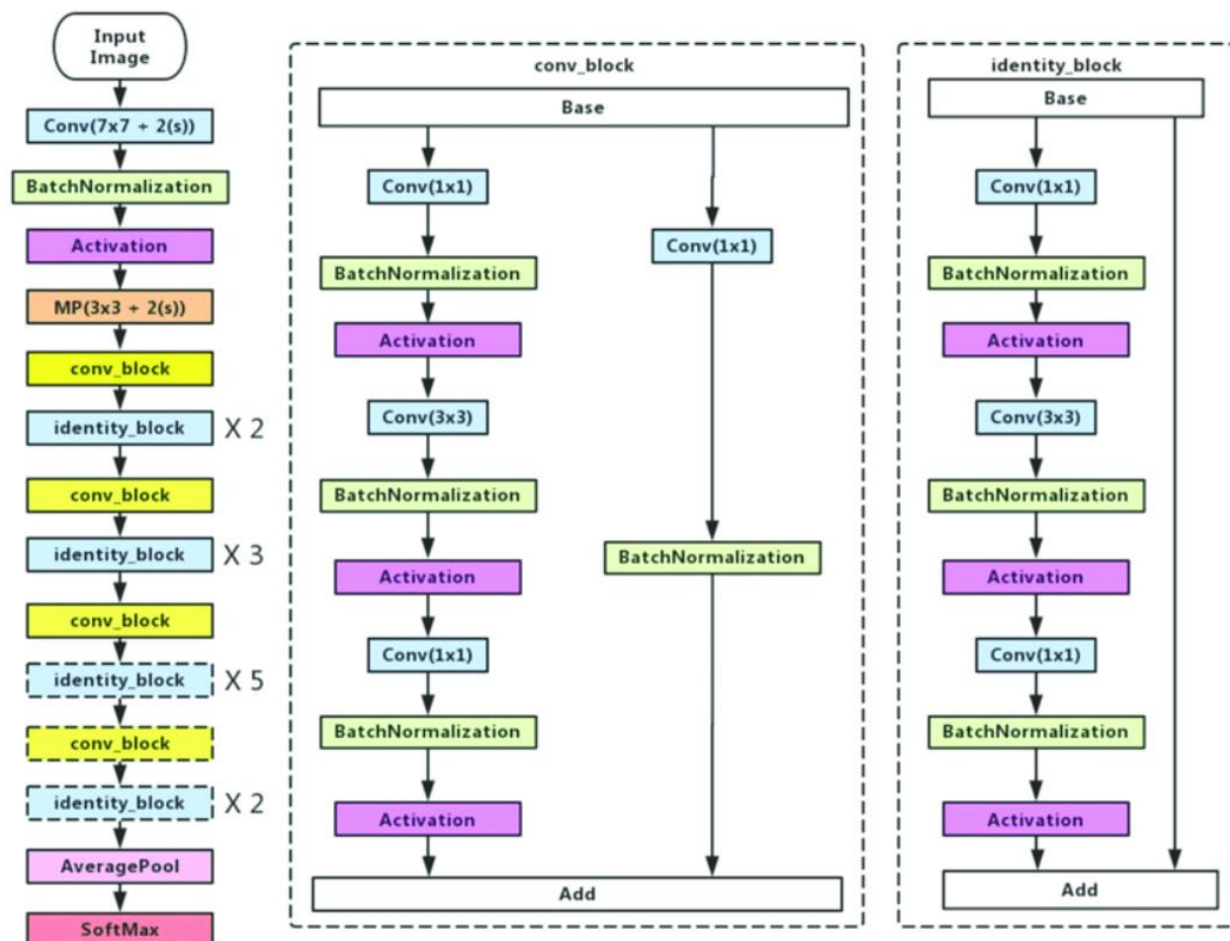


Fig 5. Sample ResNet-50 Architecture

Hyper-parameter selection:

We use various techniques to find the appropriate learning rate, batch size, and numbers of epochs. In order to fine-tune the model, we use a Learning Rate scheduler to find the best learning rate for the model.

```
89
90 lr_monitor = tf.keras.callbacks.LearningRateScheduler(
91     lambda epochs: 1e-8 * 10 ** (epochs / 20))
92
93 model_check = ModelCheckpoint("1205-CNN.hdf5", monitor="val_loss", verbose=1, save_best_only=True)
94
95 learning_rate_reduction = ReduceLROnPlateau(
96     monitor='val_mean_absolute_error',
97     patience=3,
98     verbose=1,
99     factor=0.2,
100     min_lr=0.000001
101 )
102
103 epochs = 100
104 datagen.fit(x_train)
105 # fits the model on batches with real-time data augmentation:
106 history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
107                     steps_per_epoch=len(x_train) / 32, epochs=epochs,
108                     validation_data=(x_test, y_test),
109                     callbacks=[lr_monitor, model_check],
110                     shuffle=True)
111
```

Fig 6. Code for finding appropriate learning rate

From this we select the learning rate to be $3e-4$, as it gives us the least loss value. We make use of the Learning Rate Scheduler that reduces the learning rate in order to best fit the model.

```
89
90 lr_monitor = tf.keras.callbacks.LearningRateScheduler(
91     lambda epochs: 1e-8 * 10 ** (epochs / 20))
92
93 model_check = ModelCheckpoint("1205-CNN.hdf5", monitor="val_loss", verbose=1, save_best_only=True)
94
95 learning_rate_reduction = ReduceLROnPlateau(
96     monitor='val_mean_absolute_error',
97     patience=3,
98     verbose=1,
99     factor=0.2,
100     min_lr=0.000001
101 )
102
103 epochs = 100
104 datagen.fit(x_train)
105 # fits the model on batches with real-time data augmentation:
106 history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
107                     steps_per_epoch=len(x_train) / 32, epochs=epochs,
108                     validation_data=(x_test, y_test),
109                     callbacks=[lr_monitor, model_check],
110                     shuffle=True)
111
```

Fig 7. Code for the hyper-parameters

The batch size is selected as 32. As you 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.

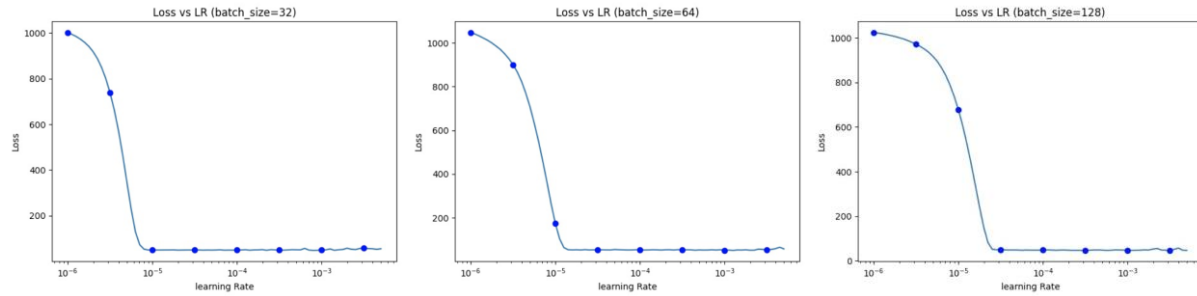


Fig 8. Learning rate vs Loss

Training CNN model:

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

```

1  # %% ----- Model Prep -----
2  # create CNN model
3  model = tf.keras.Sequential([
4
5      tf.keras.layers.Conv2D(16, (5, 5), input_shape=(128, 96, 3), activation=tf.keras.activations.
5      tf.keras.layers.MaxPool2D(2, 2),
7      tf.keras.layers.Conv2D(32, (3, 3), activation=tf.keras.activations.relu),
8      tf.keras.layers.MaxPool2D(2, 2),
9      tf.keras.layers.Dropout(0.2),
9      tf.keras.layers.Flatten(),
1     # tf.keras.layers.Dense(400, activation=tf.keras.activations.relu),
1     # tf.keras.layers.Dropout(0.2),
3     tf.keras.layers.Dense(1)
4
5 ])
6
7 model.compile(loss="mean_squared_error", # This is a classic regression score - the lower the be
8               metrics=['mean_absolute_error'],
9               optimizer=tf.keras.optimizers.Adam(lr=3e-4))

```

Fig 9. Code to compile model

For fitting the model we make use of 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.

```

97
98 epochs = 100
99 datagen.fit(x_train)
100 # fits the model on batches with real-time data augmentation:
101 history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
102                     steps_per_epoch=len(x_train) / 32, epochs=epochs,
103                     validation_data=(x_test, y_test),
104                     callbacks=[lr_monitor, model_check],
105                     shuffle=True)
106

```

Fig 10. Code to fit the model

Training ResNet 50 model:

The data is set up by using the “flow from dataframe” function. The data is split into training and validation.

```

49 datagen = ImageDataGenerator(
50     rescale=1. / 255,
51     featurewise_center=False,
52     samplewise_center=False,
53     featurewise_std_normalization=False,
54     samplewise_std_normalization=False,
55     zca_whitening=False,
56     horizontal_flip=True,
57     vertical_flip=True,
58     validation_split=0.2,
59
60     preprocessing_function=resnet50.preprocess_input
61 )
62
63 flow_params = dict(
64     dataframe=label,
65     directory='/home/ubuntu/Deep-Learning/Final_Project/frames/frames',
66     x_col="image_name",
67     y_col="count",
68     weight_col=None,
69     target_size=(128, 96),
70     color_mode='rgb',
71     class_mode="raw",
72     batch_size=batch,
73     shuffle=True,
74     seed=0
75 )
76
77 # The dataset is split to training and validation sets
78 train_generator = datagen.flow_from_dataframe(
79     subset='training',
80     **flow_params
81 )
82 valid_generator = datagen.flow_from_dataframe(
83     subset='validation',
84     **flow_params
85 )

```

Fig 11. 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. We add one dense layer and one final output layer that gives out one parameter, the prediction of the number of people in the image.

```

88
89 base_model = resnet50.ResNet50(
90     weights='imagenet',
91     include_top=False,
92     input_shape=(128, 96, 3),
93     pooling='avg'
94 )
95
96 # Change the top (the last parts) of the network.
97 x = base_model.output
98 x = Dense(1024, activation='relu')(x)
99 predictions = Dense(1, activation='linear')(x)
100 model = Model(inputs=base_model.input, outputs=predictions)
101 k = -7
102 for layer in model.layers[:k]:
103     layer.trainable = False
104     print('Trainable:')
105 for layer in model.layers[k:]:
106     print(layer.name)
107     layer.trainable = True

```

Fig 12. Compiling ResNet 50 model

Finally to compile the model we make use of the Adam optimizer. For training the model, we 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.

```

110
111 optimizer = Adam(learning_rate=3e-4)
112
113 model.compile(
114     optimizer=optimizer,
115     loss="mean_squared_error", # This is a classic regression score - the lower the better
116     metrics=['mean_absolute_error', 'mean_squared_error']
117 )
118
119 learning_rate_reduction = ReduceLROnPlateau(
120     monitor='val_mean_squared_error',
121     patience=3,
122     verbose=1,
123     factor=0.2,
124     min_lr=0.000001
125 )
126
127 model_check = ModelCheckpoint('resnet50_cc.hdf5', monitor='val_loss', verbose=2, save_best_only=True)
128
129 # Fit the model
130 history = model.fit(
131     train_generator,
132     epochs=105,
133     validation_data=valid_generator,
134     verbose=2,
135     callbacks=[learning_rate_reduction, model_check]
136 )

```

Fig 13. Optimizer and training code

Evaluation:

Models	Mean Absolute Error	Mean Square Error
CNN	5.55	50.29
ResNet 50	4.92	36.31

Fig 14. Table with metrics evaluations

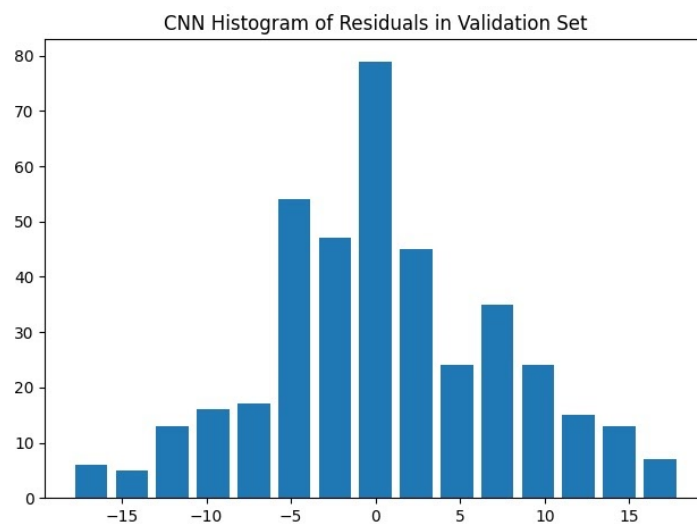
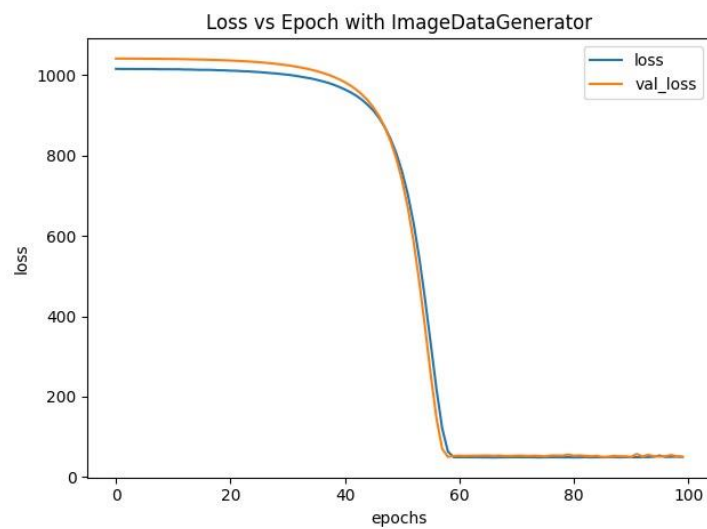


Fig 15. CNN Outputs

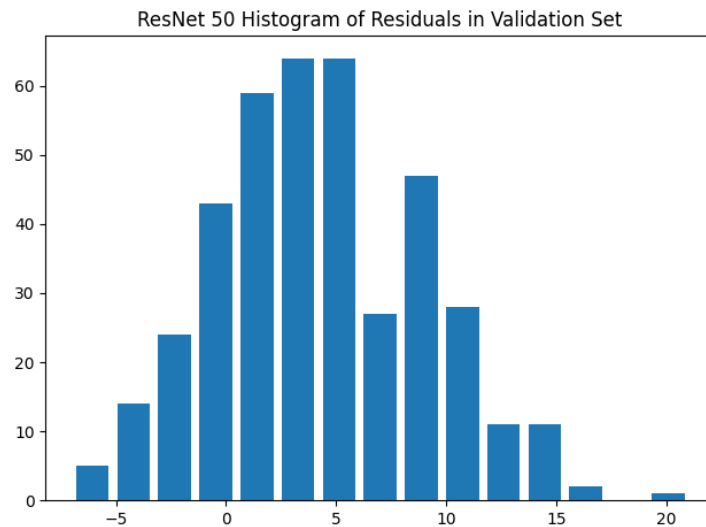
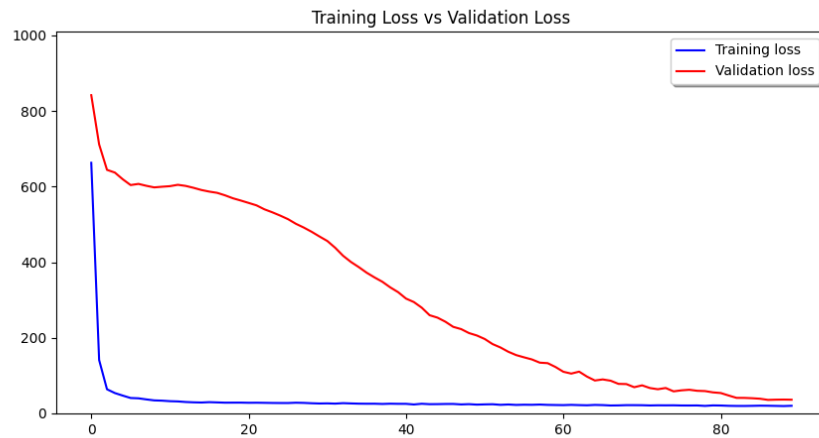


Fig 16. ResNet50 Outputs

From the above two histograms, we can see that the residuals are relatively normally distributed for both models, with a mean around 0 and 5, respectively. For CNN, most of the values are spread out between -5 and 5. For ResNet 50, most of the values are spread out between 0 and 5. CNN model accurately predicts 31 images, and ResNet 50 model accurately predicts 24 images. For the remaining pictures where count is not accurately predicted, the error between the value of count predicted by these two models and the actual value is mostly not large. Compare the two models with respect to the residuals, ResNet 50 has smaller differences between the true counts and predicted counts.

Prediction results:

- CNN model:



Fig 17. CNN model predict results vs actual results

- ResNet 50 model:



Fig 18. ResNet 50 model predict 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. However, from the picture shown in Fig 17. 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.

We can see that the ResNet50 model performed better than the CNN model in Fig 18, but there is still room for improvement. In the future, we would look to collect more data and improve the performance of our model. As of now, we are limited because the data collected does not have any images for more than 55 people, our next step would be to find such data points. Also, 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. Since this dataset is predicting numbers of people in the mall from one camera, we can also see from the distribution of the labels, the counts are not very big. Therefore, the MCNN model might be a better fit for the dataset.

References:

1. 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
2. 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>
3. Mishra, R. (2020, June 18). Counting crowd with CNN- social distancing project. Retrieved December 07, 2020, from <https://www.kaggle.com/rmishra258/counting-crowd-with-cnn-social-distancing-project>
4. DAGNetwork. (n.d.). Retrieved December 07, 2020, from <https://www.mathworks.com/help/deeplearning/ref/resnet50.html>

Appendix- Code:

Resnet_train 50:

```
import os
import cv2
import random
import scipy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.optimizers import Adam
from keras.models import *
from keras.layers import *
from sklearn.metrics import mean_squared_error
from tensorflow.keras.applications import resnet50
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.python.keras.callbacks import ModelCheckpoint

SEED = 42
os.environ['PYTHONHASHSEED'] = str(SEED)
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
# references https://www.kaggle.com/rmishra258/counting-crowd-with-cnn-social-distancing-project
# %% ----- Load Data -----
-----
```

```

DATA_DIR = os.getcwd() + "/frames/"
x, y = [], []
label = pd.read_csv("labels.csv")
label['image_name'] = label['id'].map('seq_{:06d}.jpg'.format)
lb = label['id']
count = label['count']

for path in [f for f in os.listdir(DATA_DIR) if f[-4:] == ".jpg"]:
    x.append(cv2.resize(cv2.imread(DATA_DIR + path), (128, 96)))
    for j in range(len(lb)):
        if lb[j] == int(path[4:-4]):
            y.append(count[j])
x, y = np.array(x), np.array(y)

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=SEED,
test_size=0.2)
x_train, x_test = x_train / 255, x_test / 255

resize = 64
batch = 256

# ImageDataGenerator - with defined augmentations
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=DATA_DIR,
    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
)

batch = next(train_generator)

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

optimizer = Adam(learning_rate=3e-4)

model.compile(
    optimizer=optimizer,
    loss="mean_squared_error",
    metrics=['mean_absolute_error']
)

learning_rate_reduction = ReduceLROnPlateau(
    monitor='val_mean_absolute_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=True)

# Fit the model
history = model.fit(
    train_generator,

```



```
epochs=90,  
validation_data=valid_generator,  
verbose=2,  
callbacks=[learning_rate_reduction, model_check]  
)  
print('\nTraining Done')
```

```
# Plot the loss and accuracy curves for training and validation  
fig, ax = plt.subplots(1, 1, figsize=(10, 5))  
ax.plot(history.history['loss'], color='b', label="Training loss")  
ax.plot(history.history['val_loss'], color='r', label="Validation loss", axes=ax)  
ax.set_ylim(top=np.max(history.history['val_loss']) * 1.2, bottom=0)  
legend = ax.legend(loc='best', shadow=True)  
ax.set_title('Training Loss vs Validation Loss')
```

```
plt.show()
```

```
# Predict on entire validation set, to be able to review the predictions  
manually
```

```
valid_generator.reset()  
all_labels = []  
all_pred = []  
for i in range(len(valid_generator)):  
    x = next(valid_generator)  
    pred_i = model.predict(x[0])[0]  
    labels_i = x[1]  
    all_labels.append(labels_i)  
    all_pred.append(pred_i)
```

```
test_labels = np.concatenate(all_labels)  
test_pred = np.concatenate(all_pred)  
df_predictions = pd.DataFrame({'True values': test_labels, 'Predicted  
values': test_pred})  
ax = df_predictions.plot.scatter('True values', 'Predicted values', alpha=0.5,  
s=14, figsize=(9, 9))
```

```

# ax.plot(df_predictions['True values'], df_predictions['Predicted values'],
color='r')
ax.grid(axis='both')
ax.set_title('Testing set')

plt.show()

# ----- Evaluation -----
mse = mean_squared_error(*df_predictions.T.values)
pearson_r = scipy.stats.pearsonr(*df_predictions.T.values)[0]

#print(f'MSE on testing: {mse:.1f}\nPearson r between True Values and
Predicted Values: {pearson_r:.1f}')
print("MSE Train Set Loss value", history.history['loss'][-1])
print("MSE Validation Set Loss value", history.history['val_loss'][-1])
print("MAE Test Set Loss value", history.history['mean_absolute_error'][-1])
print("MAE Validation Set Loss
value", history.history['val_mean_absolute_error'][-1])

```

Cnn train :

```

# %% ----- Imports -----
-----

import random
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras_preprocessing.image import ImageDataGenerator

```

```
from tensorflow.python.keras.callbacks import ReduceLROnPlateau,  
ModelCheckpoint
```

```
# references https://www.kaggle.com/rmishra258/counting-crowd-with-cnn-social-distancing-project
```

```
# %% ----- Set-Up -----  
-----
```

```
SEED = 64  
os.environ['PYTHONHASHSEED'] = str(SEED)  
random.seed(SEED)  
np.random.seed(SEED)  
tf.random.set_seed(SEED)
```

```
# %% ----- Load-Data -----  
-----
```

```
# read the labels.csv to get the labels and set columns name: 'image_id' ,  
'num_of_people'  
df_of_labels = pd.read_csv('labels.csv')  
df_of_labels.columns = ['image_id', 'num_of_people']
```

```
# get the label_num of the people  
labels = np.array(df_of_labels['num_of_people'])  
print("labels.shape:\n", labels.shape)
```

```
# read the data of image, check the dimension  
DATA_DIR = os.getcwd()+"/frames/"
```

```
# set image size to (128, 96)  
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), (128, 96)))  
x = np.array(x)  
print("x.shape:\n", x.shape)
```

```
# %% ----- Data Prep -----  
-----
```

```
x_train, x_test, y_train, y_test = train_test_split(x, labels, test_size=0.2)  
x_train, x_test = x_train.astype('float') / 255, x_test.astype('float') / 255  
print("x_train.shape:\n", x_train.shape, "\nx_test.shape:\n", x_test.shape)  
print("y_train.shape:\n", y_train.shape, "\ny_test.shape:\n", y_test.shape)
```

```
# %% ----- Model Prep -----  
-----
```

```
# create CNN model
```

```
model = tf.keras.Sequential([
```

```
    tf.keras.layers.Conv2D(16, (5, 5), input_shape=(128, 96, 3),  
activation=tf.keras.activations.relu),  
    tf.keras.layers.MaxPool2D(2, 2),  
    tf.keras.layers.Conv2D(32, (3, 3), activation=tf.keras.activations.relu),  
    tf.keras.layers.MaxPool2D(2, 2),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(1)
```

```
])
```

```
model.compile(loss="mean_squared_error",  
              metrics=['mean_absolute_error'],  
              optimizer=tf.keras.optimizers.Adam(lr=3e-4))
```

```
# 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)
```

```
lr_monitor = tf.keras.callbacks.LearningRateScheduler(  
    lambda epochs: 1e-8 * 10 ** (epochs / 20))
```

```
model_check = ModelCheckpoint("keras-CNN-128-96.hdf5",  
    monitor="val_loss", verbose=1, save_best_only=True)
```

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

```
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)
```

```
# %% ----- Training Loop -----
```

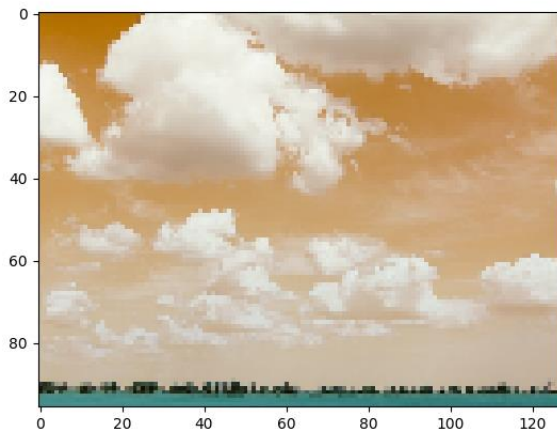
```
-----
```

```
# plot loss
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'])

plt.xlabel('epochs')
plt.ylabel('loss')
plt.title('Loss vs Epoch with ImageDataGenerator')
plt.show()
print("MSE Train Set Loss value", history.history['loss'][-1])
print("MSE Validation Set Loss value", history.history['val_loss'][-1])
print("MAE Test Set Loss value", history.history['mean_absolute_error'][-1])
print("MAE Validation Set Loss value", history.history['val_mean_absolute_error'][-1])
```

Appendix- added on Professors suggestion



The true label: 0

Predicted Label: 27

From this we can see that the model fails when there are no people in the image. We believe this is because the image was not trained to recognize these images, we could improve this by adding more images with 0 people.

Correctly predicted values:

ResNet:24/400

CNN: 31/400

From this we can see that our inference was wrong the CNN model performs much better which means that our loss values do not give us much information and we should use accuracy, precision and recall for future comparisons.