COMP343 Machine Learning
Final Project: Mask detection classifier via TensorFlow
Jiaxuan Chen
May 11th, 2022

Introduction & Data Overview:

Pandemics put everyone on masks. In many public space entrances, it will be crucial to have some live detection software/algorithms to check if people wear masks to stop the pandemic spread. Therefore, I would like to use what I've learned in class (TensorFlow, neural networks) to write a python script that trains a classifier to achieve a mask detection, which can be used further to mask object detection in those scenarios (via the Python module OpenCV).

I secured two image datasets to train and test the classifier: one stratified image dataset and one raw (mixed) dataset with annotation files. The stratified dataset contains over 7000+ images separated by labels into subfolders (with_mask and without_mask). The raw dataset contains 1000+ images with mixed labels. I included this additional raw dataset because the original TensorFlow workstation data is.xml format, which is a type of text delimited file. XML file recorded the labels(either without mask or with mask) and ROI (area of interests) region, which is the binding box that identifies people's faces. Working with this dataset helps me better understand how to preprocess images using raw data in TensorFlow.

I found image classification as an extension of the course material. Since we only worked with number datasets using perceptron or neural networks to perform binary classification tasks, I found using TensorFlow to classify raw image datasets(not tf. data type) an interesting topic for my project.

Algorithm & TensorFlow framework background:

The algorithm I choose is Convolutional Neural Network (CNN) because of the innate traits of the image dataset. Images are an array of pixels. Each pixel is represented as three-element tuples which represent the Red, Green, and Blue values of the pixel (ranging from 0 to 255). Each pixel is one of the features of the image instances. Therefore I need a model to effectively handle massive amounts of data and have well performance on pattern recognition. CNN matches these two criteria by batching the images.

For RGB images (color images like my dataset), It scans through three color channels in the image with a designed filter/batch to get the convoluted feature. (for example, if our batch features size is 3*3, it will shrink down to 1 convoluted feature.) convolution features from each channel (color) add together to come up with the final convoluted feature that can be passed to the next layer. After the convoluted layer, we will have an additional layer called the pooling layer. This further decreases the dimensions of features (pixels) in the dataset and makes our training task less computationally expensive. One of the common methods we used in the pooling layer is called maxpooling. Maxpooling takes the maximum value of a pixel from the image that is covered by the kernel/filter. It also helps de-noising along with dimensionality

reduction. Average pooling is also one approach to decrease dimensionality, but it is not preferred.

Data Preprocessing(xml_parsing.py, image_load.py):

Since the image datasets I acquired are raw and not pre-converted as TensorFlow tensors datasets, the very first steps of my project will be pre-processing/ and converting them to NumPy arrays. First, we need to crop the images in the directory into the same size. After that, we also need to standardize the image array value from 0-255 to 0-1 before I append the image to the image data frame. For the raw image datasets, I implemented an image_load.py which crops the image to 400*400 and use per_image_standardization to rescale the image to 0-1.

In regards to label extraction and matching. For the raw image dataset, xml_parsing.py extracts the picture label recorded in the annotation file and matches the label with the image with the same filename in the image folder as a tuple which is later appended in the image data frame. The main function in the xml_parsing will convert the image data frame created by the xml_to_csv function into the CSV to help me visualize the data frame.

For the stratified image dataset, we have pictures that are already separated by their labels in with_mask and without_mask subfolders, therefore I used the image_dataset_from_directory built-in function in TensorFlow to create the training and validation dataset and map the labels to the images using the folder names (label names).

Training the CNN image classifier (in image_proj.py):

To train the image classifier, I built up a model using the Model. Sequential in TensorFlow (tutorial I followed: https://www.tensorflow.org/tutorials/images/classification). I added up to 3 convolutional layers and 3 pooling layers (tf.Keras.Convex2D, tf.Keras.maxpooling). Then, I created 2 Dense layers which take in convoluted features in those convolutional layers and output the prediction using the activation functions (“relu”, “sigmoid”).

I choose stratified datasets for training and validation the image_dataset_from_directory built-in function splits the data into the training dataset and test dataset. After that, I used the model. compile to compile the model. Since this is a binary classification task, I used sigmoid as the activation function in the dense output layer and used BinaryCrossEntropy to calculate the loss.

Testing the CNN image classifier (in image_proj.py, test_data_generator.py):

In the latter part of the script, my first thought is to use the model. Predict to test the model by grabbing the raw image dataset’s dataframe (xml_df) since the label is mixed naturally. However, it is not working properly by feeding in the data frame directly since the folder contains images with mixed labels(both with mask and without mask). However, it doesn’t allow to do and the program complains “cannot convert NumPy. Array to Tf. Tensor”. To fix

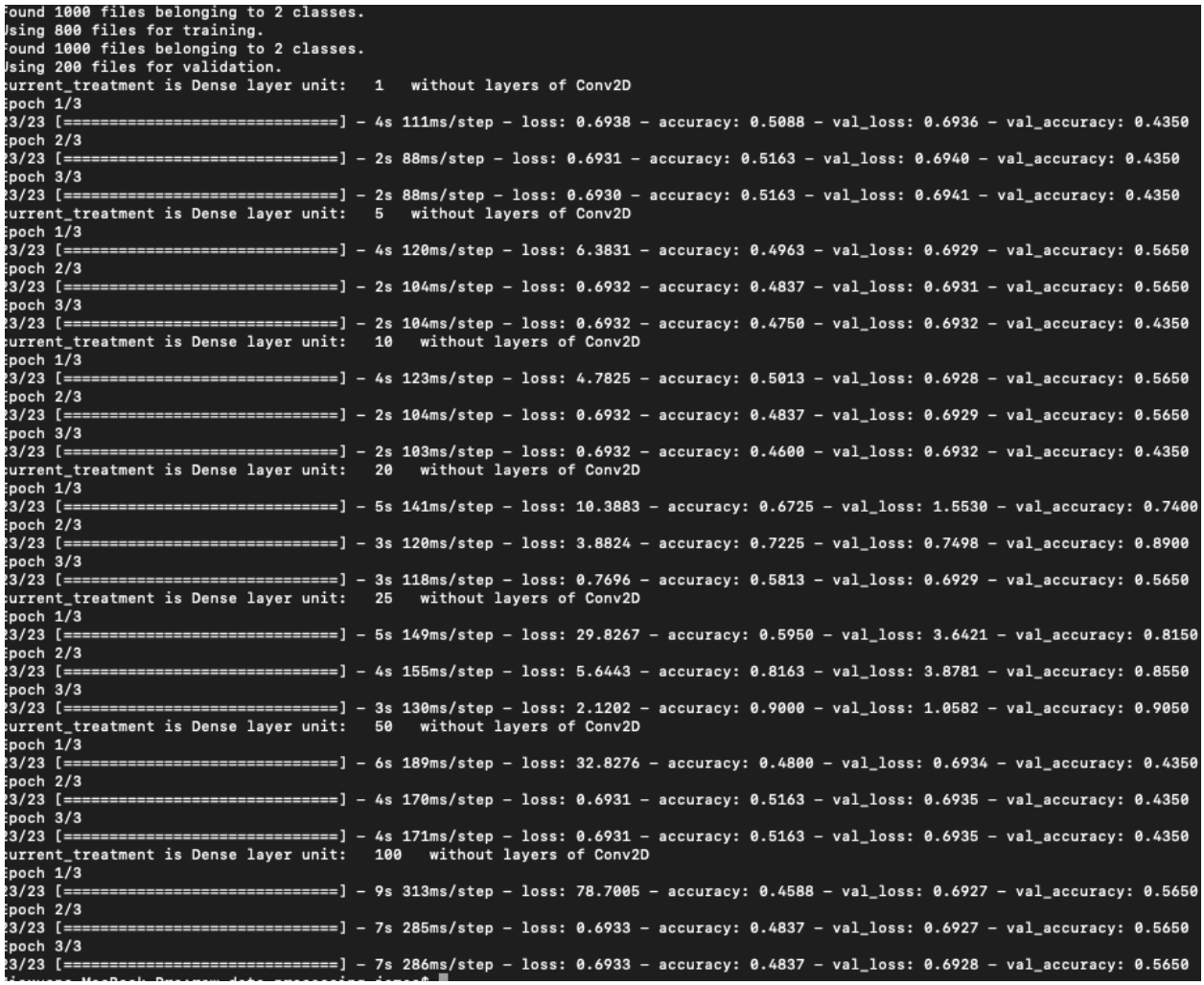
this, I choose to include the brute force for loop which extracts each image from the data frame and predicts the label. At the end of the for loop, I calculate the accuracy by the prediction list. Besides that, I also used the test_data_generator.py to throw some images from both subfolders into a new test data folder and create and panda data frame for the images in the new test data folder.

Results & Analysis (find_best_parameter in image_proj.py):

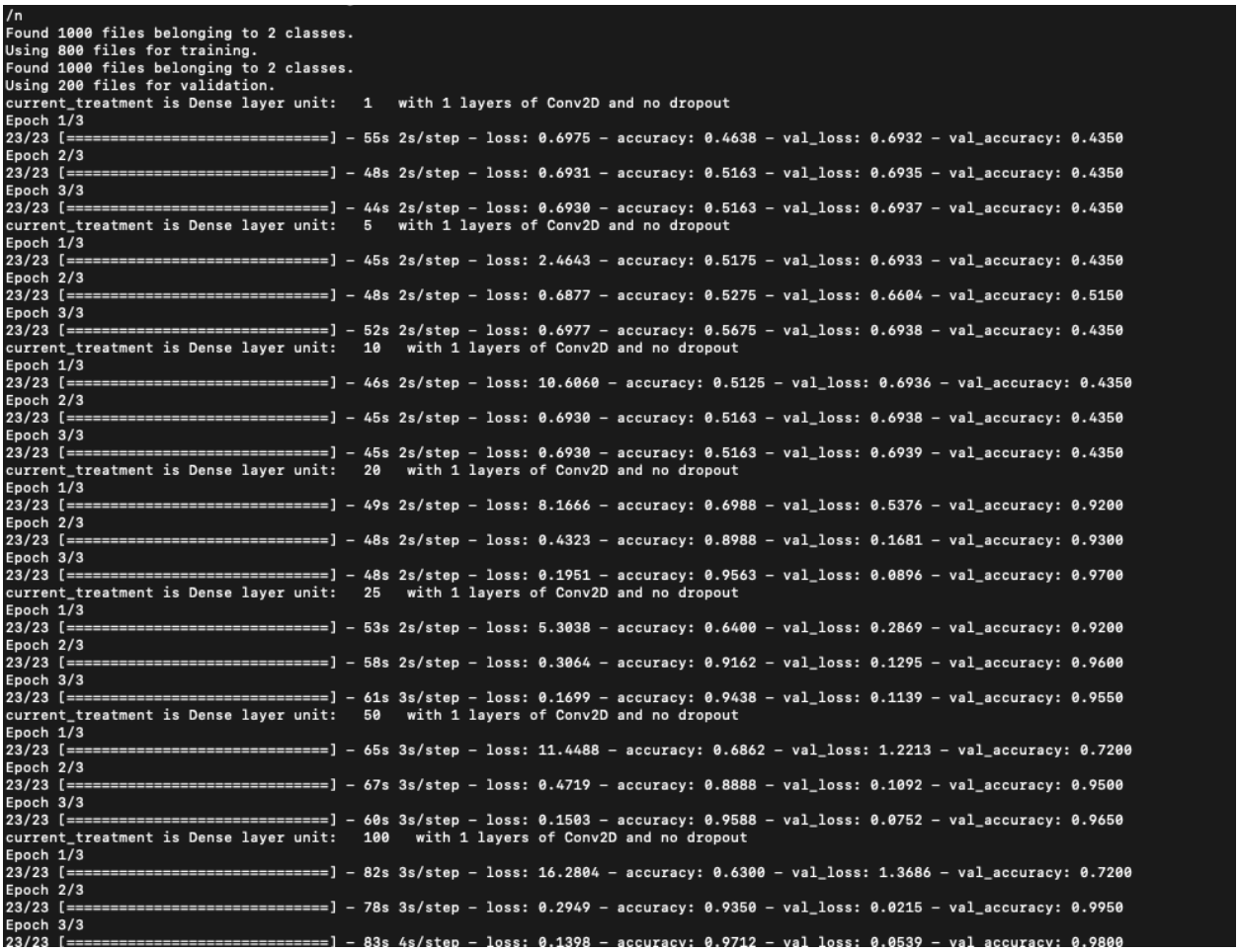
1. The validation accuracy is very high even without a lot of units in the dense layer or just dense. Optimal validation accuracy can be found when the dense unit is around 50.

```
Found 1000 files belonging to 2 classes.
Using 800 files for training.
Found 1000 files belonging to 2 classes.
Using 200 files for validation.
current_treatment is Dense layer unit:   1   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 69s 3s/step - loss: 0.6947 - accuracy: 0.4825 - val_loss: 0.6933 - val_accuracy: 0.4350
Epoch 2/3
23/23 [==============================] - 72s 3s/step - loss: 0.6931 - accuracy: 0.5163 - val_loss: 0.6938 - val_accuracy: 0.4350
Epoch 3/3
23/23 [==============================] - 75s 3s/step - loss: 0.6930 - accuracy: 0.5163 - val_loss: 0.6939 - val_accuracy: 0.4350
current_treatment is Dense layer unit:   5   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 68s 3s/step - loss: 7.5007 - accuracy: 0.4938 - val_loss: 0.6934 - val_accuracy: 0.4350
Epoch 2/3
23/23 [==============================] - 67s 3s/step - loss: 0.6931 - accuracy: 0.5163 - val_loss: 0.6935 - val_accuracy: 0.4350
Epoch 3/3
23/23 [==============================] - 69s 3s/step - loss: 0.6931 - accuracy: 0.5163 - val_loss: 0.6937 - val_accuracy: 0.4350
current_treatment is Dense layer unit:   10   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 65s 3s/step - loss: 4.0843 - accuracy: 0.5450 - val_loss: 0.6935 - val_accuracy: 0.4350
Epoch 2/3
23/23 [==============================] - 63s 3s/step - loss: 0.6930 - accuracy: 0.5163 - val_loss: 0.6937 - val_accuracy: 0.4350
Epoch 3/3
23/23 [==============================] - 85s 4s/step - loss: 0.6930 - accuracy: 0.5163 - val_loss: 0.6939 - val_accuracy: 0.4350
current_treatment is Dense layer unit:   20   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 72s 3s/step - loss: 12.5113 - accuracy: 0.5700 - val_loss: 2.0218 - val_accuracy: 0.7850
Epoch 2/3
23/23 [==============================] - 82s 3s/step - loss: 0.6127 - accuracy: 0.8975 - val_loss: 0.0704 - val_accuracy: 0.9700
Epoch 3/3
23/23 [==============================] - 90s 4s/step - loss: 0.1745 - accuracy: 0.9450 - val_loss: 0.0654 - val_accuracy: 0.9750
current_treatment is Dense layer unit:   25   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 80s 3s/step - loss: 4.1961 - accuracy: 0.7550 - val_loss: 1.2019 - val_accuracy: 0.8600
Epoch 2/3
23/23 [==============================] - 78s 3s/step - loss: 0.4397 - accuracy: 0.9187 - val_loss: 0.1881 - val_accuracy: 0.9400
Epoch 3/3
23/23 [==============================] - 78s 3s/step - loss: 0.2119 - accuracy: 0.9575 - val_loss: 0.0570 - val_accuracy: 0.9800
current_treatment is Dense layer unit:   50   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 90s 4s/step - loss: 12.0519 - accuracy: 0.6562 - val_loss: 0.4850 - val_accuracy: 0.8950
Epoch 2/3
23/23 [==============================] - 86s 4s/step - loss: 0.3366 - accuracy: 0.9300 - val_loss: 0.0518 - val_accuracy: 0.9750
Epoch 3/3
23/23 [==============================] - 87s 4s/step - loss: 0.1255 - accuracy: 0.9688 - val_loss: 0.0377 - val_accuracy: 0.9900
current_treatment is Dense layer unit:   100   with one layer of Conv2D
Epoch 1/3
23/23 [==============================] - 138s 6s/step - loss: 10.1474 - accuracy: 0.7387 - val_loss: 0.9573 - val_accuracy: 0.9350
Epoch 2/3
23/23 [==============================] - 142s 6s/step - loss: 0.7385 - accuracy: 0.9362 - val_loss: 0.8551 - val_accuracy: 0.8950
Epoch 3/3
23/23 [==============================] - 90s 4s/step - loss: 0.4764 - accuracy: 0.9475 - val_loss: 0.1177 - val_accuracy: 0.9600
```

2. By removing the Convoluted layer, we see a significant drop in validation accuracy for all treatments of dense unit



3. Removing the dropout layer but retaining 1 convolution layer seems to have not much effect on the model. The validation accuracy is still higher compared to training accuracy.



4. By adding an additional convolutional layer, the accuracy significantly increased starting from lesser dense units Dense layers compared to the result with only 1 convolutional layer.

```
Epoch 1/3
23/23 [==============================] - 111s 5s/step - loss: 0.6931 - accuracy: 0.5038 - val_loss: 0.6936 - val_accuracy: 0.4350
Epoch 2/3
23/23 [==============================] - 108s 5s/step - loss: 0.6930 - accuracy: 0.5163 - val_loss: 0.6938 - val_accuracy: 0.4350
Epoch 3/3
23/23 [==============================] - 107s 5s/step - loss: 0.6930 - accuracy: 0.5163 - val_loss: 0.6941 - val_accuracy: 0.4350
current_treatment is Dense layer unit:   5   with 2 layers of Conv2D
Epoch 1/3
23/23 [==============================] - 87s 4s/step - loss: 0.7584 - accuracy: 0.4700 - val_loss: 0.6931 - val_accuracy: 0.5650
Epoch 2/3
23/23 [==============================] - 81s 3s/step - loss: 0.6932 - accuracy: 0.4837 - val_loss: 0.6931 - val_accuracy: 0.5650
Epoch 3/3
23/23 [==============================] - 81s 4s/step - loss: 0.6932 - accuracy: 0.4888 - val_loss: 0.6933 - val_accuracy: 0.4350
current_treatment is Dense layer unit:   10   with 2 layers of Conv2D
Epoch 1/3
23/23 [==============================] - 93s 4s/step - loss: 1.4562 - accuracy: 0.5512 - val_loss: 0.6916 - val_accuracy: 0.4550
Epoch 2/3
23/23 [==============================] - 80s 3s/step - loss: 0.5857 - accuracy: 0.7600 - val_loss: 0.4766 - val_accuracy: 0.8150
Epoch 3/3
23/23 [==============================] - 79s 3s/step - loss: 0.4384 - accuracy: 0.9087 - val_loss: 0.3690 - val_accuracy: 0.9450
current_treatment is Dense layer unit:   20   with 2 layers of Conv2D
Epoch 1/3
23/23 [==============================] - 81s 3s/step - loss: 1.5907 - accuracy: 0.7375 - val_loss: 0.1079 - val_accuracy: 0.9750
Epoch 2/3
23/23 [==============================] - 79s 3s/step - loss: 0.1433 - accuracy: 0.9563 - val_loss: 0.0761 - val_accuracy: 0.9750
Epoch 3/3
23/23 [==============================] - 79s 3s/step - loss: 0.1171 - accuracy: 0.9712 - val_loss: 0.0920 - val_accuracy: 0.9800
current_treatment is Dense layer unit:   25   with 2 layers of Conv2D
Epoch 1/3
23/23 [==============================] - 78s 3s/step - loss: 1.5551 - accuracy: 0.7175 - val_loss: 0.2647 - val_accuracy: 0.9150
Epoch 2/3
23/23 [==============================] - 79s 3s/step - loss: 0.1783 - accuracy: 0.9375 - val_loss: 0.0821 - val_accuracy: 0.9650
Epoch 3/3
23/23 [==============================] - 78s 3s/step - loss: 0.1224 - accuracy: 0.9625 - val_loss: 0.1185 - val_accuracy: 0.9700
current_treatment is Dense layer unit:   50   with 2 layers of Conv2D
Epoch 1/3
23/23 [==============================] - 80s 3s/step - loss: 2.8168 - accuracy: 0.6625 - val_loss: 0.1836 - val_accuracy: 0.9300
Epoch 2/3
23/23 [==============================] - 81s 4s/step - loss: 0.1800 - accuracy: 0.9463 - val_loss: 0.0759 - val_accuracy: 0.9750
Epoch 3/3
23/23 [==============================] - 82s 4s/step - loss: 0.1206 - accuracy: 0.9588 - val_loss: 0.0785 - val_accuracy: 0.9800
current_treatment is Dense layer unit:   100   with 2 layers of Conv2D
Epoch 1/3
23/23 [==============================] - 87s 4s/step - loss: 3.9077 - accuracy: 0.5838 - val_loss: 0.5443 - val_accuracy: 0.7300
Epoch 2/3
23/23 [==============================] - 81s 4s/step - loss: 0.4345 - accuracy: 0.8062 - val_loss: 0.5138 - val_accuracy: 0.7400
Epoch 3/3
23/23 [==============================] - 86s 4s/step - loss: 0.2489 - accuracy: 0.8975 - val_loss: 0.1526 - val_accuracy: 0.9450
```

Testing accuracy:

For the testing, I used the raw data set and the stratified dataset to test the accuracy. Stratified test dataset data is from the same dataset that trains the classifier while the raw test dataset data is from a different dataset compared to training data. Surprisingly, the accuracy is very low for both test datasets (0.5 accuracies for the stratified dataset and 0 for the raw data set). The output in the terminal is saved at test_accuracy_output.txt. (check the text file in the description file folder)

Conclusion & Future work:

What I learned from this project is machine learning for image classification is hard (especially working with raw image data) in generalization even though we can get a very high validation accuracy but the fact of low testing accuracy shows that there might be some generalization issues assuming the NumPy image dataset is loading properly. Two potential mistakes that might cause the misloading:

1. the label from image_dataset_from_directory is autogenerated (to 0 and 1 according to which folders come first in the directory)
2. Noise from the raw dataset itself. Since I found several errors in the raw dataset (for example maksssksksss24.png in the raw image dataset is masked but labeled as

unmasked). It is ambiguous because he is still wearing a mask in the image but the annotation says he is wearing it incorrectly (so labeled as unmasked). Besides, some images in the raw datasets have multiple faces with different backgrounds may also cause the dataset unable to generalize well.

Besides, I found from the result session that the convolution layers (Convex2D and Maxpooling) significantly increase the validation accuracy and training accuracy. This indicates that CNN is a good model overall for image classification. I attempted a lot of methods trying to fix and diagnose why the testing accuracy is so low compared to the validation accuracy including switching labels in the image_dataset_from_directory, and trying to change the activation function in the output layer and the hidden layers as well as the loss function. However, the accuracies maintain very low. I believe there is also a chance that the test dataset is not properly loaded to the CNN model I created.

For future work, I may need to work with the testing dataset and make sure it loads properly to the CNN network. In regards to the model, due to high validation accuracies (so the model is mostly correct), I will only make some slight tweaks to the batch size while I am training the model to see how the batch size is going to affect the model. At the current stage of the project, I only studied the effect of dropout, convolution layers, and the number of dense units in the hidden layer. If the testing accuracies are fixed, I may move on to the next stage of the project: use my saved model to do live mask detection via OpenCV.