Face Mask Classifier **Project**

This problem is designed to detect whether a person in an image is wearing a face mask or not.

By Basel Mather

Available on

github.com/baselhusam/Face-Mask-Classifier

basel-mather/

Face Mask Classifier:

Approach:

This problem is designed to detect whether a person in an image is wearing a face mask or not. It is based on deep learning models trained on the Face Mask Classification dataset from Kaggle.

This solution consists of two models: Transfer Learning with the ResNet50 model and a custom model built using TensorFlow. Both models trained on the same dataset have achieved high accuracy in detecting whether a person is wearing a mask.

This solution includes multiple Python files:

- **resnet_train.py**: script for training the ResNet50 model.
- **tf_model_train.py**: script for building and training a custom model using TensorFlow.
- **evaluate.py**: script for evaluating the performance of the models on a test dataset.
- **test.py**: script for testing the models on a single image and showing it with the predicted class.
- ***face_mask_classifier_package.py***: which has the wrapper class that does everything mentioned above.
- ***main.py***: which is the main file called the class that makes predictions and prints the result.

With the ***main.py*** file, I called the class with "***from face_mask_classifier_package import FaceMaskClassifier***" which is the name of the class. This file has a code for the **argparse** library just to run the script in an easy way. When initializing the class instance, we pass the image path and the model path as the parameters to the class, and after that call the predict function that returns the prediction, then print the results.

In addition, a Jupyter notebook file called **FaceMaskDetection.ipynb** includes the code for all four Python files to showcase the result and have all the codes in one place.

The trained models are also included in the solution:

- **resnet.h5**: trained ResNet50 model (Transfer Learning)
- **tf_model.h5**: trained custom model using TensorFlow (Build Model from Scratch)

Figure 1 - ResNet50

Figure 2 - ResNet50

Figure 1 - TF model

Figure 2 - TF model

Dataset:

The dataset used for training the models is the Face Mask Detection dataset from Kaggle. The dataset consists of approximately 12,000 images, including 6,000 images with people wearing masks and 6,000 images with people not wearing masks.

The dataset contains 3 files, train, validation, and test file. The training file has 10K images (5K with mask, and 5K without mask), the validation file has 800 images (400 with mask, and 400 without mask), and the testing file has 800 images (400 with mask, and 400 without mask).

Training:

The ResNet50 and custom models were trained on the same dataset using the Keras API in TensorFlow. The ResNet50 model was fine-tuned using transfer learning from pre-trained ImageNet weights, while the custom model was built from scratch.

The resnet_train.py script can be run to train the ResNet50 model. The tf_model_train.py script can be run to build and train the custom model using TensorFlow. Both scripts take the path to the dataset as input.

ResNet50:

This script trains a machine learning model to detect whether a person is wearing a face mask or not using a pre-trained **ResNet50** model. The ResNet50 model is a powerful image classification model that has already been trained on a large dataset of images.

To fine-tune the pre-trained ResNet50 model for our specific use case, the script adds several new classification layers on top of the base model. It then compiles the new model using the **Adam** optimizer and **categorical cross-entropy** loss, with accuracy, precision, and recall metrics.

The script uses the **ImageDataGenerator** from the **Keras** library to generate batches of training and validation data from directories containing images. These images are resized and normalized to have pixel values between 0 and 1.

The script then trains the new model on the training data and validates it on the validation data for a specified number of epochs. It saves the trained model to a specified file name.

To run this script, you need to provide four arguments in the command line. The first two arguments should be the **paths to the training and validation** data directories, respectively. The third argument should be the **number of epochs** to train the model, and the fourth argument should be the **file name** to save the trained model.

Custom Trained Model:

The first step is to define the architecture of the model. In this case, the model is created using the Keras Sequential API. The model is a series of convolutional and max pooling layers, followed by two fully connected layers. The final layer outputs a probability distribution over the two classes using the **softmax** activation function.

The next step is to compile the model. Here, the model is compiled with **binary cross-entropy** loss and the **Adam** optimizer. The program then creates an instance of the **ImageDataGenerator** class, which is used for data augmentation. The program then loads the **training and validation** data sets using the **flow_from_directory** method of the **ImageDataGenerator** class.

Finally, the program trains the model using the **fit** method of the model object. With the **tf_model_train.py** file, the **training data** and **validation data** are passed in as arguments, along with the **number of epochs** to train for. Once training is complete, the model is saved to a file specified by the user.

Evaluation:

The performance of the models was evaluated on a test dataset using **accuracy**, **precision**, **recall**, and **F1-score** metrics. The **evaluate.py** script can be run to calculate the performance metrics for either the ResNet50 or the custom model. The script takes the **path to the test dataset** and the **path to the pre-trained model** as inputs.

After applying the evaluation step to our models, I got the following values for the metrics.

Models \ Metrics	Accuracy	Precision	Recall	F1-score
ResNet50 model	0.91	0.93	0.87	0.9
Custom model	0.98	0.97	0.98	0.98

As we can see, the custom model performed better than the ResNet50 for transfer learning, maybe it is because the custom model has 19,264,322 parameters, or maybe because the testing data is small, and when we move this solution to production, the ResNet50 maybe will better result that the custom model.

I created a wrapper class that loads the model and uses it to make predictions on images. The input image is first load the model then the image using **Keras preprocessing** library and then resized to the required size (the size that the model trained on), then the image is normalized and passed to the loaded model, which returns the predicted class.

How to Run the Scripts

1. Download the Face Mask Detection dataset from Kaggle and extract it to a folder.

2. Install the required packages by running pip install -r requirements.txt

3. To train and save the ResNet50 model, run the command ***python resnet_train.py <path to the training data> <path to the validation data> <number of epochs> <saved model name>***

4. To train and save the custom TensorFlow model, run the command ***python tf_model_train.py <path to the training data> <path to the validation data> <number of epochs> <saved model name>***

5. To evaluate the model performance on test data, run the command ***python evaluate.py <path to the test dataset folder> <path to the saved model file>***

6. To test the model on a single image, run the command ***python test.py <path to the test image> <path to the saved model file>***

Using the Class wrapper way:

1. run the following command in cmd after running ***pip install -r requirements.txt*** :
python main.py --img_path path/to/img --model_path path/to/model

Python Version:

The script was developed using Python version **3.9.13**

Future Enhancement:

In the future, we can further improve the performance of the model by using data augmentation and hyperparameter tuning. We can also explore other pre-trained models like InceptionV3 and EfficientNet, which might give better performance. Furthermore, we can use object detection models to detect the presence of masks and other objects in the image.