ECE 228 GROUP 3: FACE MASK CLASSIFICATION VIA DEEP LEARNING

Benjamin Chang, Joseph Chang, Nikhil Pathak

Department of Electrical and Computer Engineering, University of California, San Diego

ABSTRACT

Prominent health officials clearly stipulate the benefits of face mask usage and stress wearing them correctly during the pandemic. As such, we train and compare various different CNN models to detect if people are wearing masks correctly or not.

1. INTRODUCTION

In the midst of the COVID-19 pandemic, the CDC recommends everyone wear face masks when in public to prevent spread of the virus [1]. Many states like California have mandated mask usage in "public and workplace settings where there is high risk of exposure" [2]. Hence, it is important to identify individuals who are not or incorrectly wearing masks to reduce viral spread and protect those at higher risk. Store owners, restaurant managers, and other executives need to be informed when people are not following public mandates in gathering hotspots. Our project solves this through a deep learning approach that can classify a face in one of three classes: masked, unmasked, masked incorrectly.

The input to our algorithm is an image of any size with any number of faces. We use a face detector [3] to extract all faces in an image, resize them into standard 35x35 pixels, and pass them into our model. Then, we use a neural network to output a class prediction of each face as masked, unmasked, or masked incorrectly. Our code then bounds colored squares around faces in the original image indicating the correctness of mask usage for each face.

In this project, we reuse some classification training code from Nikhil Pathak's ECE 176 Final Project.

2. RELATED WORK

Mask classification is a relevant topic due to COVID-19 and many previous approaches have been used to try to solve it. One such method is Transfer Learning which uses pretrained feature extractors of highly accurate models such as Resnet [4] or VGG [5]. These models are trained on large datasets for many epochs and have shallow feature-extracting layers which can be repurposed for different classification tasks. Transfer learning is a common and effective approach, requiring less computational resources during training. While VGG networks use deep architectures with "small (3x3) convolution filters" [5], Resnet networks use even deeper architectures that learn residuals through skip connections to tackle vanishing and exploding gradient issues. Both perform very well at classification tasks and can be repurposed for our mask classification project. Some methods which have specifically been used to solve the face mask classification problem include a 2 stage CNN [6] as well as a VGG-16 network composed of convolutional layers [7].

Another method to detect masked or unmasked faces is YOLOv3 [8] which consists of blocks containing 1x1, 3x3 convolutions followed by residual layers. The input of each block is added to a processed layer's output which applies transfer learning using a pretrained model and freezing all layers except the last 3. Alternatively, Faster R-CNN [9] is a widely used state-of-the-art object detection algorithm that generates bounding boxes by extracting features with a region proposal network (RPN) algorithm. Then, it passes features through CNN layers and classifies objects in the image. The key is that RPN is used to extract features. Unlike YOLO, localization and classification happen in separate networks making Faster R-CNN slower. It uses simple convolution layers with 3x3 kernels followed by the RPN and then a detector network. Both algorithms can be modified to detect if people are wearing masks[10].

3. DATASET AND FEATURES

We use a Face Mask Detection dataset obtained from Kaggle [11] which has 853 images with potentially several people in each image. Each image has ground truth bounding boxes for every person and ground truth labeling if they are correctly masked, unmasked, or masked incorrectly. This dataset is useful for tasks such as face detection and face-mask classification which our project focused on. We choose this task for its feasibility and how thorough we can be with our experiments. Though face detection is more difficult than image classification, we still researched the task and used some elements of it in our overall algorithm. While solving face mask classification, we also improve on the existing Kaggle CNN work [12] that inspired this project. The many experiments we ran give us a lot to learn from–specifically, what elements of a network benefit classification accuracy.

The Kaggle Face Mask Detection [11] shown in Fig. 1 contains 853 images. Since the images have no set resolu-

Fig. 1: Original Images from Face Mask Detection Dataset

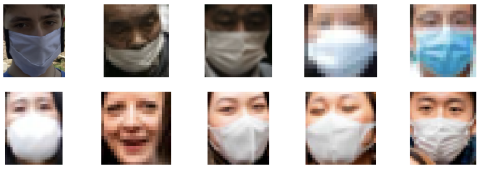

Fig. 2: Extracted Faces from Face Mask Detection Dataset

tion or aspect ratio, we preprocessed them. To train our classification model, we isolated each face in all the images to form our own face dataset. The 853 images have 4072 labeled faces given in the dataset annotations. Each annotation contains bounding box information of each face in an image. We form training/validation/testing splits of 80%/10%/10% totalling 3257, 407, and 408 faces. The faces of this dataset come from different angles, sizes, and distances so the size of each extracted face is not constant as seen in Fig. 2. However, CNNs require fixed input shape for our classification use case. To account for this, we calculated the average width and height of the faces in the dataset to be 31.15 pixels wide, 35.00 pixels tall. Then, for pre-processing, we resize all face images to 35x35 and normalize them according the mean and standard deviation of ImageNet [13]. Using ImageNet normalization constants is commonplace because ImageNet is a massive dataset of 14,197,122 images.

4. METHODS

To solve this 3-way mask classification task, we use CNNs, an approach widely used across image tasks due to its strength in parameter sharing and translational invariance [14]. Specifically, we use transfer learning which applies knowledge gained while solving one problem to a different, but related problem" [15]. We use models pretrained on ImageNet [13] and modify the final layer which leverages the strong feature extraction portions of state-of-the-art models.

The Kaggle submission [12] only trained a VGG-19 model on a dataset that contained 2 classes, masked and unmasked, by applying the pretrained model to images in the Face Mask Detection dataset. This was done by extracting the faces with an OpenCV face detector and running them through a binary classifier CNN. For our project, we solved a 3 class classification problem with classes: masked, unmasked, and masked incorrectly. We train, evaluate, and test our models using the Face Mask Detection dataset [11] mentioned in Section 3.

For training our models, we used CrossEntropyLoss which is standard for classification tasks with a set number of classes. However, one caveat in our task is that our training data had serious class imbalance issues. Of the 3257 training faces, there are 2551 masked, 608 unmasked, and only 98 masked incorrectly. Based on some early results, we found that using an unweighted CrossEntropyLoss resulted in high overall accuracy, but at the expense of individual class accuracy of "masked incorrectly." Since there were few training images of incorrect masks, the model effectively ignored them in favor of maximizing the other two class accuracies which had more samples. One primary, new aspect of our transfer learning approach was tackling the "masked incorrectly" class. To solve this, we used a weighted CrossEntropyLoss to optimize our models (taken from Pytorch documentation [16]):

$$\text{loss}(\mathbf{x}, class) = (-\mathbf{w}[class]) \log\left(\frac{e^{\mathbf{x}[class]}}{\sum_j e^{\mathbf{x}[j]}}\right)$$

$$\text{loss for batch size } N = \frac{\sum_{i=1}^{N} \text{loss}(\mathbf{pred}[i], class[i])}{\sum_{i=1}^{N} \mathbf{w}[class[i]]}$$

The weights **w** were assigned to each class such that more weight was placed on classes with less images. This gave weights as such: Masked = 98/2551, Unmasked = 98/608, Masked Incorrectly = 98/98.

For each model, we trained using PyTorch [17], minimized training loss, and tuned weights based on validation loss. We also employed early saving techniques to avoid overfitting to the training set by always saving the model with the lowest validation loss in the case that validation loss starts increasing in later epochs. To compare different models, we used the average test set accuracy across all 3 classes. This is a self-designed metric that emphasizes the equal importance of all 3 classes in our dataset instead of those with most images in the dataset. The disproportionate distribution of classes will skew overall accuracy, but our priority is classifying all 3 classes well instead of just 2 classes at the expense of the 3rd. Thus, this self-designed metric is justified. We can then decide which model is best by comparing their test set average-class-accuracy (**henceforth called ACA**).

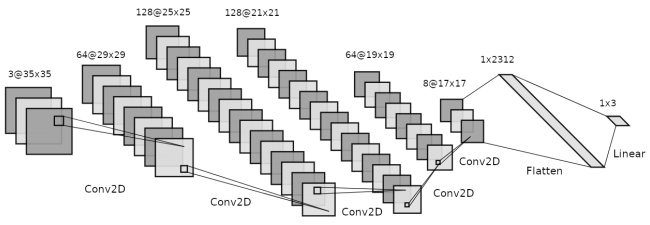

Fig. 3: Custom CNN Architecture (Figure from [18])

We test a variety of different transfer-learning models: ResNet18, ResNet34, VGG16, VGG19, DenseNet121, DenseNet161, and DenseNet201 [19]. In addition to these,

we also design our own Custom CNN whose architecture is seen in Fig. 3. This custom architecture allows us to conduct even more experiments on different types of models including using BatchNormalization and different activation functions.

5. EXPERIMENTS/RESULTS/DISCUSSION

	Accuracies			
Model	Mask	No Mask	Inc. Mask	**ACA**
ResNet18	96.03	80.95	23.08	**66.69**
ResNet18 w/ 2 FC Layers	96.03	83.33	15.39	**64.92**
ResNet34	89.80	90.48	30.77	**70.35**
Custom CNN with ReLU	90.94	88.10	46.15	**75.06**
Custom CNN with ReLU + BatchNorm	93.77	92.86	69.23	***85.29***
Custom CNN with TanH	86.69	90.48	53.85	**77.01**
Custom CNN with TanH + BatchNorm	86.97	95.24	46.15	**76.12**
VGG16	93.48	88.10	69.23	**83.60**
VGG16 with BatchNorm	90.94	71.43	24.18	**62.18**
VGG19	92.92	95.24	61.54	**83.23**
VGG19 with BatchNorm	90.94	85.71	30.77	**69.14**
DenseNet201	98.87	95.24	23.10	**72.40**
DenseNet161	98.58	95.24	46.15	**79.99**
DenseNet121	95.75	90.48	30.77	**72.33**

Fig. 4: Model Experiments and Corresponding ACA

For our experiments, we wanted to improve on the previous Kaggle work by experimenting with different transfer learning models and our own custom model which uses a targeted loss function that handles class imbalance issues in our training dataset with 3 classes. We did some individual hyperparameter tuning for each experiment, but for the transfer learning models (ResNet, VGG, and DenseNet), we trained for fewer epochs (20) since the weights were pretrained. This is because the model will converge faster and begin to overfit if it is trained for too many epochs. For our custom model with fresh weights, we had to train for 40 epochs before overfitting was clearly apparent. As mentioned before, we employed early saving techniques to fight this issue because training for more epochs when validation loss is clearly increasing can be harmful. We also used a standard regularization of 1e-2 across most of the experiments because this gave us the best validation loss results at the end of training.

Additionally, we used the Adam optimizer with a learning rate of 1e-5 for all the experiments including the Custom CNN. We expected to only need a small learning rate for the pretrained models because they are already highly optimized as feature extractors so their weights and biases only need minimal tuning. Surprisingly, the fresh custom CNNs also performed best on validation loss with this low learning rate. Finally, we used a batch size of 128 to ensure we would not run out of GPU memory and to accurately approximate the distribution of classes in the entire training set. This way, the batch updates would be a good approximation of the entire dataset. We drive our network training with a weighted CrossEntropyLoss and save the models at the epoch with smallest validation set loss. To evaluate the performance of the experiments against each other, we use the self-designed ACA metric, which is the average accuracy of the individual class accuracies. This helps equally emphasize each class's importance despite the dataset imbalance.

Results from our experiments are shown in Fig. 4 including individual class accuracies and overall ACA performance metric. We notice that of the transfer learning models, ResNet architectures perform worst while VGG architectures perform much better than DenseNet and ResNet. The normal VGG models performed best on average, but the Custom CNN with ReLU and BatchNorm actually produced the highest ACA. Our custom model was the best with equal emphasis on each class! The problem of class imbalance in the dataset is clearly seen in the trends of "Incorrect Mask" which consistently has the lowest accuracy of the 3 classes. This is expected, but we were able to bring the accuracy to as high as 69.23% in our custom model which is quite good given the minimal training data we have for that class. With our custom model, we also tested different activation functions and BatchNormalization. It was clear ReLU working with BatchNormalization gave the best results. It is important to note that out of thoroughness, we tested the transfer learning models *without* pretrained weights as well, but those always gave worse ACA performance than the pretrained models.

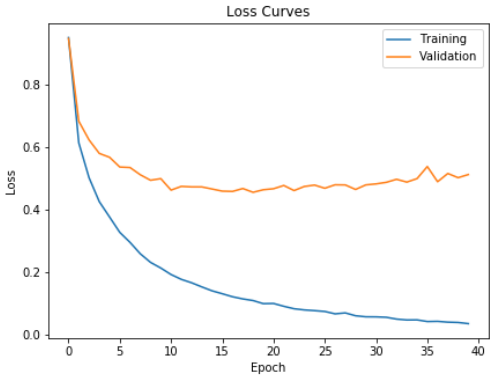

Fig. 5: Custom CNN with ReLU+BatchNorm Loss Curves

In Fig. 5, we see training and validation loss curves for our best model–the Custom CNN with ReLU and BatchNorm. One can observe there is some clear overfitting occuring as the validation loss begins to rise near the end of training. However, with our early saving implementation, our best model is taken from the training epoch with lowest validation loss which happens before the overfitting occurs!

In Fig. 6, we see the results of our best model experiment on the test set. Specifically, this normalized confusion matrix

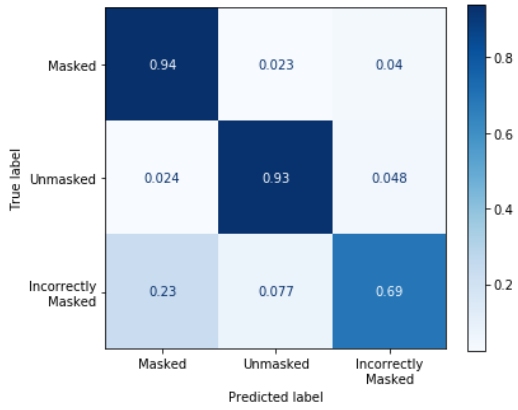

Fig. 6: Custom CNN with ReLU+BatchNorm - Normalized Confusion Matrix on Test Set

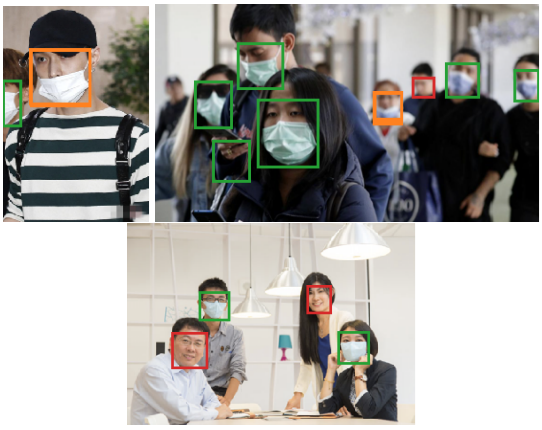

Fig. 7: Strong Predictions

Fig. 8: Some Poor Predictions

illustrates where the model is going wrong in its final predictions. For the masked and unmasked classes, the predictions are quite strong with high accuracy and very few misclassifications. The problematic case is predicting the Incorrectly Masked class and this is expected due to the lack of training data for it. The confusion matrix paints a very clear and logical picture for the very low accuracy on this class–the model often confuses Incorrect Masks with Correct Masks. 23% of the incorrectly masked faces in the test were misclassified as correctly masked. This is easily understood because as humans, we sometimes struggle to see if a mask is fully covering the nose and being worn properly. With low resolution images, we can understand that differentiating a face with a mask just below the nose from a face with a mask just above the nose is a very difficult problem to learn from such small sample sizes of one of those classes.

With our trained classification model on faces, we use a pretrained face-detector [3] (that specializes in identifying faces with masks because traditional face detectors struggle with this) to extract the faces in a given image. All these extracted faces are passed through our custom CNN and classified into one of our 3 classes. We then have an algorithm to draw colored bounding-boxes around these detected faces in the original images with following key: Green=Masked, Red=Unmasked, Orange=Masked Incorrectly.

In Fig. 7 and 8, we show some examples of our algorithm applied on test images. From Fig. 7, we see strong examples of face predictions where all faces are properly classified. When the images are clear and the faces are high-enough resolution, the model tends to be very good at the classifying task. However, there do exist some cases where the model struggles. Particularly, as mentioned with the confusion matrix, the model sometimes thinks an incorrectly masked face is correctly masked. We see one such example with the middle face in the second image of Fig. 8. Moreover, in the first image of Fig. 8, we see most of the faces properly classified, but there are some wrong classifications–particularly of the 4 orange boxes, only 1 of them is actually an incorrectly masked face. This is an example of the difficulties the model sometimes faces with distinguishing correct and incorrect masks–when faces are small and thus will be low resolution inputs to the model.

6. CONCLUSION/FUTURE WORK

From our experiments, we found the Custom CNN with ReLU and BatchNormalization produced the highest ACA. This was an interesting discovery because we expected the pretrained models to do much better due to their powerful feature extractions. However, since those models are trained via a balanced CrossEntropyLoss on a different dataset of 1000 unique classes, it makes sense that the custom CNN was able to outperform the transfer learning models. For future work, we plan to explore using a FasterRCNN model to directly detect faces and experiment with more model layers. We would also like to collect more incorrectly masked data to train our models to improve accuracy on that class.

7. TEAM MEMBER CONTRIBUTION

7.1. Benjamin Chang

I wrote code to train pretrained and non-pretrained Dense201 models for face mask classification. This was done on the "Face Mask Detection 12k Images Dataset" as well as the 3 class "Facemask Dataset" such that bounding boxes are drawn around input image face masks. I also performed evaluation of the classification accuracy of these networks on DenseNet networks of varying number of layers in addition to investigating the effect of augmented data and dropout layers on classification accuracy. Lastly, I worked on the final report.

7.2. Joseph Chang

I trained VGG-16, VGG-19 network model and modified them to detect whether a face has mask on, off, or incorrectly using the "Face Mask 12K dataset". The mask detector itself has high accuracy, but is dependent on a good face detector. I used Haarcascade to detect faces and bounding box coordinates. This detector is then prepended to the mask detector so the region of interest is more specific. After tests, I found Haarcascade performed rather poorly so I trained a Faster RCNN for face detection as a replacement. Computed output results, training loss, and worked on the final report.

7.3. Nikhil Pathak

I have written the training and Dataset/Dataloader code for our classification model. I also conducted the experiments with ResNet and our custom CNN, and worked on the class-imbalance issue we were facing. I also wrote the code that integrated the work end-to-end by first extracting all the faces from an image using the FasterRCNN model, and then running each extracted face through the trained 3-way classification model in order to get a label for each face that is then bounded by a color-coordinated box for the user to inspect the faces and their classifications in an output image. I also wrote the code for retrieving the confusion matrix for the model on the test set. Finally, I worked on the final report.

References

[1] Centers for Disease Control and Prevention, "Your guide to masks," https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/about-face-coverings.html, Apr 2021.

[2] "Masks," https://covid19.ca.gov/masks-and-ppe/, Apr 2021.

[3] Daniel, "Fasterrcnn," https://www.kaggle.com/daniel601/pytorch-fasterrcnn/notebook, May 2019.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," 2015.

[5] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[6] Amit Chavda, Jason Dsouza, Sumeet Badgujar, and Ankit Damani, "Multi-stage cnn architecture for face mask detection," https://www.researchgate.net/publication/344276976_Multi-Stage_CNN_Architecture_for_Face_Mask_Detection, Sep 2021.

[7] Christoph Lippert, Akhyar Ahmed, and Raza Ali, "Face mask detector," https://www.researchgate.net/publication/344173985_Face_Mask_Detector, Jul 2020.

[8] Joseph Redmon and Ali Farhadi, "Yolov3: An incremental improvement," 2018.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[10] Sunil Singh, Umang Ahuja, Munish Kumar, Krishan Kumar, and Monika Sachdeva, "Face mask detection using yolov3 and faster r-cnn models: Covid-19 environment," https://doi.org/10.1007/s11042-021-10711-8, Febuary 2021.

[11] Andrew Maranhão, "Face mask detection," https://www.kaggle.com/andrewmvd/face-mask-detection, May 2020.

[12] Nagesh Singh Chauhan, "Mask and social distancing detection using vgg19," https://www.kaggle.com/nageshsingh/mask-and-social-distancing-detection-using-vgg19, Feb 2021.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "Imagenet large scale visual recognition challenge," 2015.

[14] Peter Gerstoft, "Lecture 7 - ece 228 machine learning for physical applications," April 2021.

[15] Wikipedia, "Transfer learning — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Transfer\%20learning&oldid=1022394104, 2021.

[16] PyTorch, "Crossentropyloss," https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html, 2019.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[18] Alexander Lenail, "Nn-svg," https://alexlenail.me/NN-SVG/, 2019.

[19] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger, "Densely connected convolutional networks," 2018.