A Study on Convolutional Neural Networks Variant for Face Mask Classification

Yang Qixiu*(ID 2175021167), Lu Xu*(ID 2175122023), Sun Zihong[†](ID 2176112709),
Galbiati Tiberio (ID 3120999064), Casarico Massimo[†] (ID 3120999073),
Muhammad Rifki Kurniawan[†] (ID 3120999259), Siriporn Pattamaset[†] (ID 4120999061).
Big Data and Deep Learning Group 39.
*School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an, China
[†]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China

***Abstract*—Deep learning architectures including Convolutional Neural Networks (Convnets) have been broadly applied into diverse numbers of sectors such as surveillance due to high accuracy. Inspired by the recent situation on pandemic outbreaks and, concurrently, surveillance system application involvement in public using AI, we build Convenets architectures for binary image classification task of face mask classification. The aim of the project is to identify human face images wearing a mask and not wearing a mask. The experiments are based on a public dataset available on Kaggle that was collected combining Google searches and the CelebFace dataset. In this report we will discuss the classification performance of the different network architectures based on classification accuracy metrics, such as f1-score, and performance metrics including number of parameters. We are investigating the corollary of pre-trained networks for transfer learning as opposed to learning from random initialization. The experiments resulted that MobileNetV2 with ImageNet pre-trained weight is considerably superior among others with the high accuracy metrics along with its lightweight parameters which sum up its capability to be deployed into mobile or edge devices. Moreover, by simply training the task on simple naive hand-crafted Convnets, we can achieve very competitive high accuracy compared to other state-of-the-art sophisticated backbone.**

***Index Terms*—image classification, deep learning, medical face mask, transfer learning.**

I. Introduction and Theoretical Background

The COVID-19 pandemic has become global pandemic since December 2019, the country and society have advocated wearing masks to prevent infection and protect public health and safety, so testing whether people wear masks has become a new demand. Combining this with the course content, we create an algorithm that can directly detect is a person is wearing a face mask or not.

A. Problem Formulation of Binary Classification

The image classification objective is to distinguish different categories of input images according to the semantic information of the image. It is the core of computer vision and the basis of other high-level vision tasks such as object detection, image segmentation, object tracking, behavior analysis, and face recognition. Yet binary classification means that the classification task categories is binary or having two classes.

The binary task is formally defined as the following illustration. Given X is the dataset — in this case is face-mask dataset — containing sets of image label pairs $\{(x_i, y_i), 1 \leq i \leq N, y_i \in \{1, 0\}\}$ and $x_i \in R^{H \times W \times C}$ where H, W and C denotes the image size referring height, width, and number of channels, respectively. Further we resize all the samples into height and width in the size of 224x224 with 3 number of channels for its RGB channel. In this experiment, we propose to use off-the-shelf backbones and compare with our two hand-crafted Convnets. Therefore, the designated experimentations encompass a set of discriminative models F with sigmoid $\phi(.)$ classifier outputting probability value spreading out $\lfloor 0, 1 \rfloor$.

B. CNN

Convolutional Neural Networks (CNN) are a type of feed-forward neural networks that include convolution calculations and has a deep structure. They are specialized for processing data with a grid-like topology [1]. It is mainly composed of input layer, convolutional layer, activation function, pooling layer, fully connected layer, and loss function. Its essence is feature extraction and decision inference.

C. Transfer Learning on Classification

Transfer learning refers to the transfer of learned and trained model parameters to a new model to help new model training. Considering that most data or tasks are related, we can share the learned model parameters (knowledge learned by the model) to the new model in a certain way through transfer learning to speed up and optimize the learning efficiency of the model instead of starting from scratch.

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems. Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned [2].

From a mathematical prospective, transfer learning can be described as follows. A domain $\mathcal{D}$ consisting of: a feature space § and a marginal probability distribution $P(X)$, where $X{=}x_1, x_2, ..., x_n \in \mathcal{X}$. Given a specific domain, $\mathcal{D}{=}\mathcal{X}, P(X)$, a task consists of two components: a label space $\mathcal{Y}$ and an objective predictive function $f : \mathcal{X} \rightarrow \mathcal{Y}$. The function f is used to predict the corresponding label $f(x)$ of a new instance x. This task, denoted by $\mathcal{T}{=}\mathcal{Y}, f(x)$, is learned from the training data consisting of pairs x_i, y_i, where $x_i \in X$ and $y_i \in \mathcal{Y}$. Going further, given a source domain $\mathcal{D}_s$ and learning task $\mathcal{T}_s$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, where $\mathcal{D}_s \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T()$ in $\mathcal{T}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$.

Is intuitive that the benefits brought from a transfer learning approach are numerous. In the above mentioned book, the authors summarise them in three main blocks: a higher start in the performance curve (the initial skill, before refining the model, on the source model is higher than it otherwise would be), a higher slope (the rate of improvement of skill during training of the source model is steeper than it otherwise would be) and a higher maximum possible performance (the converged skill of the trained model is better than it otherwise would be).

To practically demonstrate the above mentioned advantages, Kafeng Wang et al.,[3] compared transfer learning to traditional compression methods and stated how transfer learning offers the chance for CNNs to learn with limited data samples by transferring knowledge from models pre-trained on large datasets. Transfer learning methodology not only is capable to determine a new model with a higher target task accuracy, but also further accelerates it by computing a subset of channel neurons in each convolutional layers and under a wide range of datasets, the smallest drop in validation accuracy under the same speed-up constraints when compared to traditional compression methods.

D. Deep Learning Frameworks

In order to improve work efficiency, researchers wrote codes into a framework and put them on the Internet for all researchers to use together. The most popular deep learning frameworks in the world are PaddlePaddle, Tensorflow, Caffe, Theano, MXNet, Torch and PyTorch.

In this project we implement the deep learning models using TensorFlow, Keras, and PyTorch.

Google's open source Tensorflow is an open source mathematical calculation software developed in C++ language. It uses the form of Data Flow Graph to perform calculations. It is the most widely used framework in the world and has the largest community.

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with focus on enabling fast experimentation.

PyTorch is developed by the Torch team and is a Python-first deep learning framework. Its predecessor was Torch, but PyTorch is more flexible, supports dynamic graphs, and provides a Python interface.

II. Related Works

Face mask classification. The related studies of face mask classification are predominated by approaches using deep learning, especially Convnets architecture. Generally the study can be categorized into two different tasks: classification [4, 5, 6] and detection task[7, 8, 9, 10, 11, 12]. In [5] was adopted a pre-trained backbone of InceptionV3 with additional both convnets and multi-layer perceptrons for the head classifier while involving 8-variants data augmentations as oversampling strategy to alleviate imbalance dataset distribution. While [4] proposed a similar strategy but exploited ResNet18 features extractor along with attached classifier composing SVM, decision tree, and the ensemble. However, neither of them deeply take the computational cost and complexity of the architecture into consideration even as application study requires comprehensive performance metrics due to potentially hardware limits and energy consumption such as on the edge deployment.

Face mask detection. Contrarily face mask detection goes beyond classification that directs subsequent face localization and classification either wearing mask or not. Therefore given input images the model designates to predict bounding boxes with respect to the objects and determine associated correct labels for each predicted bounding box. Recently, the utilization of pre-trained network alongside developing on-top-of off-the-shelf state-of-the-art object detection framework such as one-stage framework comprises YOLO[13] and its variants [14] and SSD[15] or two-stage approach resembling Faster R-CNN[16]. Among those is [9] that developed a mask detector constructed around YOLOv2 but replacing the darknet backbone with pre-trained ResNet50 which benefited by skip-connections to create more useful representation for YOLOv2 localizer. In contrast, [8] introduced SRCNet assembled from a super-resolution module as input quality enhancer before forwarded into the classifier. The super-resolution module will restore image details from low-resolution images so as the classifier gets restored inputs which significantly improve accuracy on low quality images. While the feature pyramid network was proposed by [7] for enriching the output semantic features by concatenating multiple-level feature maps for the detector inputs. This is resulting in comparable improvement on accuracy metrics relative to the baseline.

Transfer learning. The training regime via transfer learning or so called knowledge transfer appeared to be standard trick in deep learning application. This allows to transfer knowledge either from different or related domain, tasks, and distributions between the source and target [17, 18]. Predominantly, the source domain for computer vision task is using natural images belonging to ImageNet[19] dataset for creating pre-trained model. While the standard techniques in transfer learning for visual task branched into two which include "fixed-feature transfer" — freezing the backbone and replace the classifier with task-specific networks — and "full-network transfer"[20] — fine-tuning the whole networks. Early study on transfer learning employed dimensionality reduction to construct latent features as a proxy for connecting the source with target latent features thru minimizing the distance in this latent space[21]. Yosinski et al., [22], examine the transferability of pre-trained neural networks which deduced that transferability declines as the domain gap between source and target task is high. But, weight initialization from pre-trained resulted in better generalization and accuracy than initializing randomly. Later, layer-wise transferability features problem is resolved via Deep Adaptation Network (DAN) by [23] with projecting those into kernel Hilbert space and diminishing between domains inconsistency. While, [24] studied comprehensively the correlation of the performance of ImangeNet pre-trained on the downstream task and found that training regime such as regularization used is susceptible to downgrading the downstream task and not overly beneficial for fine-grained tasks.

III. Proposed Method

A. Introduction to The Face Mask Data Set

This dataset comprises facial images of people wearing face mask and not wearing face mask for classification purpose. The original data set was already split into training set of 10,000 samples, test of 992 samples and validation of 800 samples. The dataset consists of almost 12,000 images for a total of 328.92MB in size. The images have a minimum size of 25 pixels and a maximum of 563 pixels. The majority length/width is 224 (224 x 224 x 3) pixels as shown in Fig 1. The original dataset can be downloaded from Kaggle (https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset).

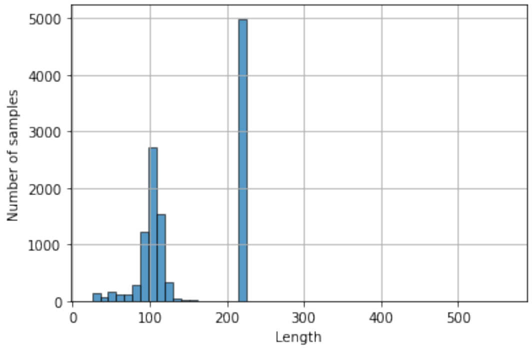

Fig. 1. The distribution of the length of the images in the dataset.

In classification task, the dataset should have balance in the number of instances of each class in order to reduce the bias towards the majority class and prevent false positive and false negative classifications. This data set was already well balanced when splitted into training set, testing set, validation set as shown in Fig 2.

B. Features Transfer in Binary Classification

Binary classification task aims to predict the label $\hat{y}_i$ given input x_i with $\hat{y}_i \in \{0, 1\}$. Let the θ_p denotes the pre-trained weight from the ImageNet pre-trained model for features extraction. We used this pre-trained weight as initialization of the model $f(.; \theta_p)$ while training on the batches of the dataset. While in this experiment, we did not freeze the features extractor and let the training steps fine-tune the weight of both features extractor and the classifier. Then, we has an objective to minimize the loss value to obtained best parameters θ given given image-label pairs in a batch approximately equivalent to dataset distribution D

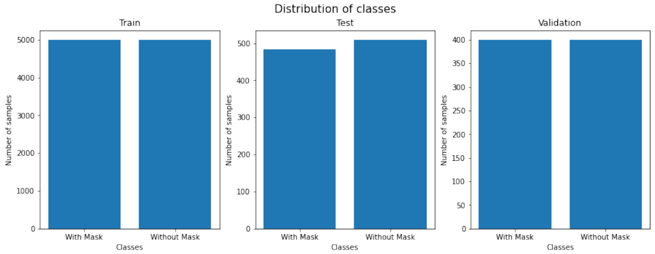

Fig. 2. Class distribution of the dataset.

$$\min_{\theta} E_{(x,y)\sim D} \left[\mathcal{L}_{BCE} \left(f(x; \theta), y \right) \right] \tag{1}$$

. In this context, we employ binary cross entropy loss $\mathcal{L}_{BCE}$ which measure the binary cross entropy between the target y and the output $f(x)$ as following equation Eq.(2) which the gradient with respect to this loss will be used as the learning projection in parameters space

$$\mathcal{L}_{BCE} = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log \left(f\left(x_i\right) \right) + \left(1 - y_i\right) \cdot \log \left(1 - f\left(x_i\right)\right) \tag{2}$$

.

C. Architectures Variation

1) Hand-craft CNN: In this report, we proposed two CNN-based models for face mask detection with different architectures that will be explained in detail in the experiment section.

2) EfficientNet: EfficientNet is a group of convolutional neural network models consisting of eight models from B0 to B7, with each subsequent model number referring to variants with more parameters and higher accuracy. EfficientNet model architecture will have to scale in three stages: (I) Depthwise Convolution and Pointwise Convolution, (II)Inverse ResNet blocks consisting of a layer that squeezes the channels, then a layer that extends the channels and (III) Linear bottleneck that uses linear activation to prevent loss of information from ReLU.[25]

3) ResNet: The deeper the shallow CNN network, the better the effect, but increasing the number of layers after reaching a certain depth will cause the network to converge more slowly and the accuracy rate becomes worse. To solve this problem, the ResNET was born.The basic idea of ResNet is to introduce a ”shortcut connection” that can skip one or more layers to improve performance.

4) Xception: Xception stands for *Extreme version of Inception*, it is a convolutional neural network architecture that relies solely on depthwise separable convolution layers. A depthwise separable convolution splits a kernel into 2 separate ones that do two convolutions:

1) Depthwise Convolution
   This module deals not just with the spatial dimensions, but also with the depth dimension (n. of channels) because the spatial convolution is performed independently over each channel of the input.
2) Pointwise Convolution
   This is so named because it uses a 1x1 kernel, projecting the channels output by the depthwise convolution onto a new channel space. In short, the Xception architecture is a linear stack of depthwise separable convolution layers with residual connections. Xception architecture has overperformed VGG-16, ResNet and Inception V3 in most classical classification challenges [26].

5) MobileNetV2: Improved version of mobile architecture developed by Google Research [27], MobileNetV2 is a multi-task-purpose state-of-the-art backbone specialized for mobile devices. This architecture is built on top of three essential components named inverted residual block, lightweight depth-wise separable convolutional layer, and linear bottlenecks. Additionally, some tricks are used for accuracy improvement including chop off the non-linearity in narrow layers as the effort to build better features representation thru preserving the information. The originated research has proven that this architecture is applicable for various visual tasks such as object detection or object segmentation.

6) MnasNet: Designed specifically for mobile device deployment, MnasNet[28] is developed by a similar group who founded MobileNet. The networks structure was developed via neural architecture search that directly incorporated the performance metrics such as model latency into the searching algorithm objective function. Thereby the NAS algorithm will find out the best architecture by optimizing the trade-off between structure complexity and performance in terms of latency. However, instead of using common FLOPS as metrics performance, MnasNet measure the performance as respects to real-world inference latency of the model deployed on mobile phones.

IV. Experiments

Colaboratory is a Google research project created to help disseminate machine learning education and research. It is a free cloud service and supports free GPU . It's a Jupyter Notebook environment that requires no setup to use and runs entirely in the cloud.

In this experiments, we jointly write and run the code on the Google Colaboratory platform, we created different models based on the same dataset, but using different architectures and hyperparameters as shown in TableI and we compared them. Because we do not have access to high GPU computation resource in order to do grid search. The experimental models consist of two CNN hand-craft models and transfer learning with pre-trained models which are EfficientNetB0, Xception, ResNet18, MobilenetV2, and MnasNet. We also added image augmentation to the data generator as follows: horizontal flip 10 degree, rescale to 1/255, and image normalization.

TABLE I
The summary of parameters setup in each experiments.

Model	Image shape	batch size	Learning rate
Hand-craft CNN1	(224x224x3)	32	0.0003
Hand-craft CNN2	(256x256x3)	64	0.0004
EfficientNetB0	(224x224x3)	16	0.00001
ResNet18	(224x224x3)	32	0.001
Xception	(256x256x3)	64	0.001

A. Model 1 : Hand-craft CNN1

1) Architecture: Hand-craft CNN1 is a hand-craft model with 3 convolutional layers as shown in Fig.3. The model was created as follows:

- The input shape is 224x224 as the majority image shape of the image dataset. The first layer uses 32 filters with kernel size of (3 × 3) and stride of 1.
- Next, rectified linear activation function (ReLU) layer, batch normalization layer was added with 0.90 momentum, and add a max pooling layer which uses a pool size of 2, and then add dropout with 0.2.
- Then,we repeat for 2 times the structure of convolutional layers followed by ReLU activation function layer, batch normalization layer and a max pooling layer. The number of output filters started from 32, 64, 128, and 128.
- Lastly, flatten the input, and add 3 Dense layers and batch normalization layer, and sigmoid activation function layer. The model architecture is shown in Fig.3.

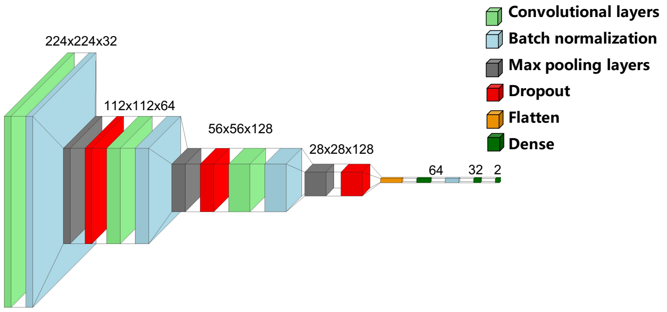

Fig. 3. Hand-craft CNN1 architecture

2) Hyperparamters Tuning:

- Loss function as Binary Crossentropy.
- The learning rate is 3×10^{-4}.
- Using Adam optimizer as the optimizer for training.
- The early stop method was used to prevent overfitting during training with `patience = 5`.

3) Evaluation Metrics: The accuracy and loss are considered as the evaluation metrics.

B. Model 2 : Hand-craft CNN2

1) Architecture: Hand-craft CNN2 was created using a custom architecture as shown in Fig.4, based on [29]
Here follows the description of the layers added:

- The layer uses 64 fairly large filters (7 × 7) with stride equal to 1 because the input images are not so small. Here we set the input shape to 256x256, this could be changed to fit better the distribution of the images of the set, but as we will see later, this value does not make big differences.
- Then we add another convolution layer, similar to the one above but with smaller filters, and without stride.
- Next we have a max pooling layer which uses a pool size of 2, so it divides each spatial dimension by a factor of 2.
- Then we repeat for 3 times the structure of two convolutional layers followed by a max pooling layer. The value 3 is a hyperparameter which can be tuned. It is important to note that the number of filters grows as we climb up the CNN toward the output layer, we started from 64, then 128, 256, 512.
- We finally flatten the input, and add 3 Dense layers. Since we are using binary classification the last activation is sigmoid.

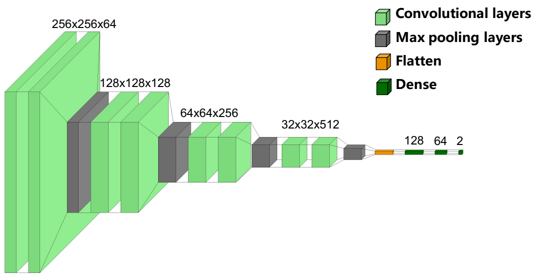

Fig. 4. Hand-craft CNN2 architecture

In TableII is reported the full layer architecture for the model 2.

2) Hyperparameters Tuning:

- Since we are doing binary classification we choose *Binary Cross Entropy* as Loss function.
- For learning rate we used $1E - 4$
- As optimizer we used *Adam* optimization, which is a momentum optimizer, so it uses an exponentially decaying average for past gradients but in addition it also keeps track of the decaying average for past squared gradients.
- In order to avoid overfitting we used the early stop method setting `patience = 5`

C. Model 3: Transfer learning with EfficientNetB0

We utilized EfficientNetB0[25] which is the pre-trained ImageNet model. EfficientNet is built for ImageNet classification contains 1,000 classes.

TABLE II
LAYER ARCHITECTURE

```
Model: "sequential"
_________________________________________________________________
Layer               Output Shape              Param #
=================================================================
conv2d             (None, 256, 256, 64)      9472
_________________________________________________________________
conv2d_1           (None, 256, 256, 64)      36928
_________________________________________________________________
max_pooling2d      (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2           (None, 128, 128, 128)     73856
_________________________________________________________________
conv2d_3           (None, 128, 128, 128)     147584
_________________________________________________________________
max_pool2d_1       (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_4           (None, 64, 64, 256)       295168
_________________________________________________________________
conv2d_5           (None, 64, 64, 256)       590080
_________________________________________________________________
max_pool2d_2       (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_6           (None, 32, 32, 512)       1180160
_________________________________________________________________
conv2d_7           (None, 32, 32, 512)       2359808
_________________________________________________________________
max_pool2d_3.      (None, 16, 16, 512)       0
_________________________________________________________________
flatten            (None, 131072)            0
_________________________________________________________________
dense              (None, 128)               16777344
_________________________________________________________________
dense_1            (None, 64)                8256
_________________________________________________________________
dense_2            (None, 2)                 130
=================================================================
Total params: 21,478,786
Trainable params: 21,478,786
Non-trainable params: 0
```

1) Architecture: This model used EfficientNetB0 as base model made available in Keras applications. To apply this pre-trained model to our purpose, we modified classifier from 1,000 classes to 2 classes by attaching fully connected layer. In the experiment, we tried both pre-trained weights and random weights.

2) Hyperparameters Tuning:

- Loss function as Binary Cross Entropy.
- The learning rate is 1×10^{-5}.
- Using Adam optimizer as the optimizer for training.
- The early stop method was used to prevent overfitting during training with `patience = 5`.

3) Evaluation Metrics: The accuracy and loss are considered as the evaluation metrics.

D. Model 4: Transfer learning with ResNet18, Mnasnet, and MobileNetV2

1) Architecture: The model weight was initialized from ImageNet pre-trained model and then fine-tune the model into down-stream face-mask dataset. The model was implemented on top of PytorchLightning and was trained on single GPU.

2) Hyperparamters Tuning:

- Loss function as Binary Cross Entropy.
- The learning rate is 1×10^{-3}.
- Using Adam optimizer as the optimizer for training.

- number of epochs 20.
- weight decay is 0.0005.

E. Model 5 : Xception with transfer learning

This fifth model is still implemented with Tensorflow, but this time we used transfer learning as explained in Section I-C. We used Xception and to reach the highest score we did some hyperparameters tuning. The parameters we tuned were the learning rate, the batch size, the number and dimension of Dense layers and the patience for early stopping callback. Initially we didn't use Dense layer, but we then notice that adding Dense layers we were able to reduce the gap between training accuracy and validation accuracy.

We tested different version of transfer learning with Xception:

- Cross validation
- Using two consecutive trainings: one with frozen base model and the other with the whole model unfrozen
- Callback to control the variation of learning rate

1) Architecture: We used Xception as base model and we then attach fully connected layers. We find out that doing two consecutive trainings (one with base model frozen and the other with model unfrozen) didn't make much difference in Xception. In the end we made only one training with the model unfrozen to speed up the training time.

2) Evaluation Metrics: The accuracy and loss are considered as the evaluation metrics.

V. Results and Discussion

In this section, we show the experimental results of the proposed models.

A. Transfer learning Compared to Learning from Scratch

Analyzing the influence of transfer learning by utilizing pre-trained from ImageNet model is the one of the objectives that we will discuss in this section.

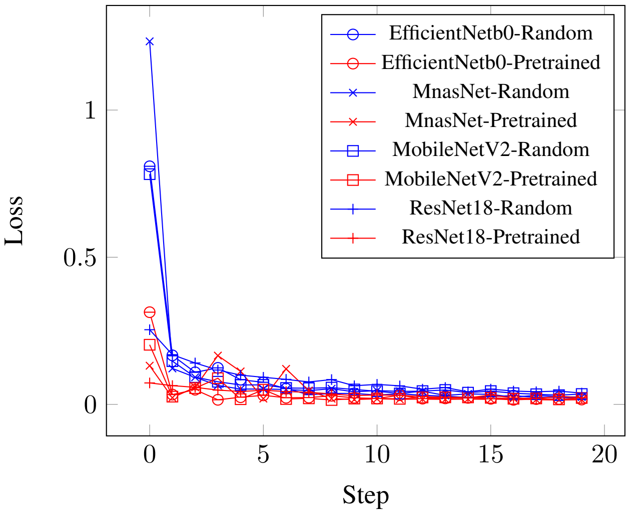

Fig. 5. Training loss profile of our pre-trained models and models trained from random initialization

According to the experiment of various models with transfer learning of ResNet18, MobilenetV2, EfficientNetB0, and MnasNet, training the model with pre-trained weights initialization can yield better than random weights initialization, which corresponds to this paper[31]. It is also evident that transfer learning yields to a high accuracy using less epochs, compared to the custom made CNNs as shown in the Figure 5. Most of the pre-trained-initialization models colored in red lines exhibited earlier convergence while training the downstream task which roughly required 2 epochs training in order to achieve convergence. Meanwhile, the models from random initialization or so called learning from scratch took > 5 epochs to converge.

In Figure 6 we report the graphs of accuracy over epochs

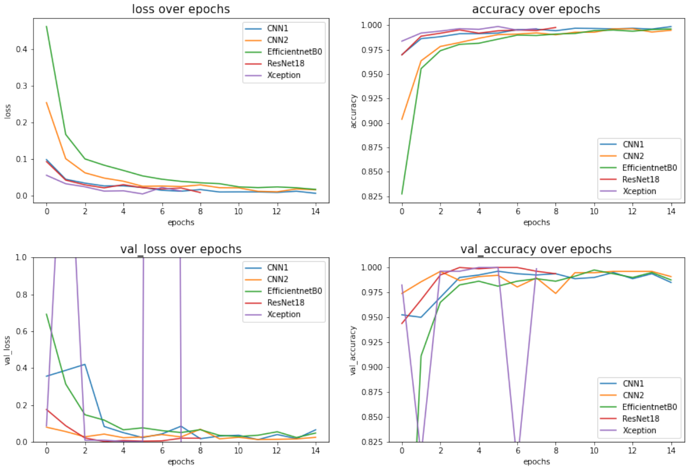

Fig. 6. Showing the accuracy and loss in training and validation over epochs

It is interesting to notice that Xception transfer model had lot of fluctuations in the accuracy and loss of validation set. That was a symptom of overfitting and could be further reduced by adding better regularization.

B. Performance on Various Architectures

In Table III we list the accuracy and performance benchmarks of the different architectures. It is interesting to notice that for this specific task, increasing the number of parameters had not a huge impact on accuracy, because we can see how a 6.5 million parameter model (Hand-craft1) had almost the same performance of a 22.3 million one (Xception). In addition, we can analyze that the majority of the model initialized from ImageNet pre-trained are proven to be more accurate in all the accuracy and F1-Score opposed to training the model from random initialization.

VI. Conclusion

In conformity with the experimental results, the dataset seems well prepared as a consequence all the models performed accurately and achieved over 99% both in accuracy and F1-score. Specifically, the MobilenetV2 model with pre-trained weight can result in the best performance over the other models in terms of accuracy and F1-score. Moreover,

TABLE III
ACCURACY AND PERFORMANCE BENCHMARK ON VARIOUS NETWORKS ARCHITECTURES

Architecture	Weight Initialization	Accuracy (%)				Performance
		Acc	Precision	Recall	F1-Score	Parameters (Millions)
Hand-craft1	random	99.3	99.3	99.3	99.3	6.5
Hand-craft2	random	99.1	99.1	99.1	99.1	19.4
ResNet18 [30]	random	98.9	98.8	99.2	99.0	11.2
ResNet18 [30]	pretrained	99.2	98.6	100.0	99.3	11.2
Xception [26]	pretrained	99.4	99.4	99.4	99.4	22.3
MobilenetV2 [27]	random	99.2	98.6	100.0	99.3	2.2
MobilenetV2 [27]	pretrained	99.7	100.0	99.6	99.8	2.2
Efficientnetb0 [25]	random	99.5	99.2	100.0	99.6	4.0
Efficientnetb0 [25]	pretrained	99.4	99.0	100.0	99.5	4.0
MnasNet [28]	random	98.6	98.8	98.6	98.7	3.1
MnasNet [28]	pretrained	98.6	100.0	97.4	98.7	3.1

compared to other pre-trained models achieving similar accuracy including MnasNet or EfficientNetB0, MobilenetV2 is considerably much more smaller in the number of parameters as such is noticeably applicable to be implemented on any devices specifically edge or mobile as the MobilenetV2 originally designated for with competitive accuracy. However, the simple model of CNNs, such as hand-craft CNN1, provides simple architecture, less complexity with high accuracy as well.

Further developments can be done, such as testing the model with a video live source. In this case it can be useful to use first a segmentation model in order to detect faces in a photo frame and then use our models to detect if a person is wearing a mask or not.

ACKNOWLEDGMENT

The face mask dataset was created by Ashish Jangra (https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset) which consists of almost 12,000 images from google search and the CelebFace dataset created by Jessica Li (https://www.kaggle.com/jessicali9530).

In this project we implement the deep learning models using TensorFlow, Keras, and PyTorch. We appreciate all developers involved in those open-source projects.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016, ISBN: 9780262035613. [Online]. Available: https://books.google.co.in/books?id=Np9SDQAAQBAJ.

[2] E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, and A. J. S. Lopez, *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2009, ISBN: 1605667668.

[3] K. Wang, X. Gao, Y. Zhao, X. Li, D. Dou, and C. Xu, "Pay attention to features, transfer learn faster cnns," in *ICLR*, 2020.

[4] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic," *Measurement*, vol. 167, pp. 108 288–108 288, 2020.

[5] G. J. Chowdary, N. S. Punn, S. K. Sonbhadra, and S. Agarwal, "Face mask detection using transfer learning of inceptionv3," in *BDA*, 2020.

[6] S. A. Sanjaya and S. A. Rakhmawan, "Face Mask Detection Using MobileNetV2 in the Era of COVID-19 Pandemic," *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy, ICDABI 2020*, 2020. DOI: 10.1109/ICDABI51230.2020.9325631.

[7] M. Jiang and X. Fan, "Retinamask: A Face Mask Detector," *arXiv*, 2020, ISSN: 23318422. arXiv: 2005.03950.

[8] B. Qin and D. Li, "Identifying facemask-wearing condition using image super-resolution with classification network to prevent COVID-19," *Sensors (Switzerland)*, vol. 20, no. 18, pp. 1–23, 2020, ISSN: 14248220. DOI: 10.3390/s20185236.

[9] M. Loey, G. Manogaran, M. Taha, and N. E. M. Khalifa, "Fighting against covid-19: A novel deep learning model based on yolo-v2 with resnet-50 for medical face mask detection," *Sustainable Cities and Society*, vol. 65, pp. 102 600–102 600, 2020.

[10] A. Chavda, J. Dsouza, S. Badgujar, and A. Damani, "Multi-stage CNN architecture for face mask detection," *arXiv*, 2020, ISSN: 23318422.

[11] "Face Mask Detection by using Optimistic Convolutional Neural Network," *Proceedings of the 6th International Conference on Inventive Computation Technologies, ICICT 2021*, pp. 1084–1089, 2021. DOI: 10.1109/ICICT50816.2021.9358653.

[12] B. Roy, S. Nandy, D. Ghosh, D. Dutta, P. Biswas, and T. Das, "MOXA: A Deep Learning Based Unmanned Approach For Real-Time Monitoring of People Wearing Medical Masks," *Transactions of the Indian National Academy of Engineering*, vol. 5, no. 3, pp. 509–518,

2020, ISSN: 2662-5415. DOI: 10.1007/s41403-020-00157-z. [Online]. Available: https://doi.org/10.1007/s41403-020-00157-z.

[13] J. Redmon, S. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[14] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *ArXiv*, vol. abs/1804.02767, 2018.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. Berg, "Ssd: Single shot multibox detector," in *ECCV*, 2016.

[16] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2015.

[17] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[18] L. A. Torrey and J. Shavlik, "Chapter 11 transfer learning," 2009.

[19] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[20] H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry, "Do adversarially robust imagenet models transfer better?" *ArXiv*, vol. abs/2007.08489, 2020.

[21] S. J. Pan, J. T. Kwok, and Q. Yang, "Transfer learning via dimensionality reduction," *Proceedings of the National Conference on Artificial Intelligence*, vol. 2, pp. 677–682, 2008.

[22] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" In *NIPS*, 2014.

[23] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 97–105. [Online]. Available: http://jmlr.org/proceedings/papers/v37/long15.html.

[24] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2656–2666, 2019.

[25] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," Proceedings of Machine Learning Research, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds., pp. 6105–6114, Sep. 2019. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html.

[26] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, 2017. arXiv: 1610.02357 [cs.CV].

[27] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[28] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2815–2823, 2019.

[29] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017, ISBN: 978-1491962299.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[31] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2661–2671.