Embedded Systems Project

Deep Learning Model Optimization for Face Mask Detection

Kartik Chhipa (B20CS084)
Kartik Choudhary (B20CS025)

Introduction

Face mask detection was an important task during the ongoing COVID-19 pandemic. The use of masks was a critical step in preventing the spread of the virus, and being able to automatically detect whether someone is wearing a mask can help ensure compliance with mask mandates in public places. Deep learning models have been successfully used for face mask detection, but these models can be computationally expensive and may not be suitable for use on mobile or embedded devices. In this report, we describe a project that focuses on optimizing a deep learning model for face mask detection to make it more efficient and suitable for deployment on a wide range of devices.

Problem Statement

The goal of the project was to optimize a deep learning model for face mask detection in order to reduce its computational requirements while maintaining high accuracy. Specifically, we aimed to achieve the following objectives:

1. Reduce the number of parameters in the model to make it more lightweight and suitable for deployment on mobile and embedded devices.
2. Reduce the inference time of the model to make it more efficient for real-time applications.
3. Maintain high accuracy in face mask detection.

Approach

We use Haar Cascade Face Detection Algorithm to detect the face from the video stream and then feed the neural network with the cropped face image

We have also used augmentation techniques to artifically increase the size of dataset by randomly rotating, zooming, blurring, shifting the images. This step is essential to increase the accuracy on the testing dataset.

While training we have also created EarlyStopping Callback to stop training if there is no improvement in the validation loss after few epochs

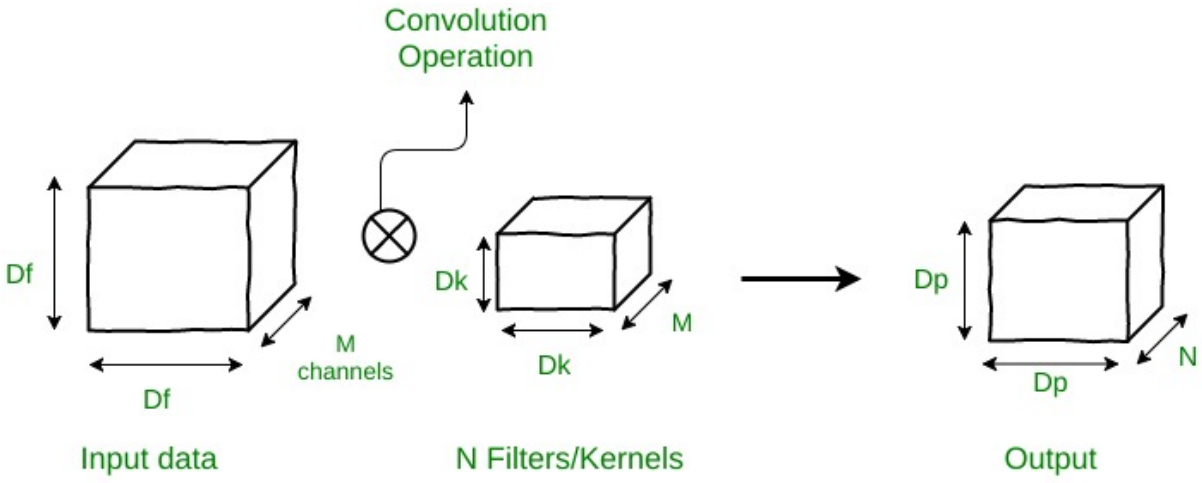
We started with a pre-trained face mask detection model based on the MobileNet architecture. We then used several optimization techniques to improve its efficiency and reduce its computational requirements. The following are the steps we took:

1. Quantization: We used quantization to reduce the precision of the model's weights and activations. We used post-training dynamic quantization, which quantizes the weights and activations to 8-bit integers during inference. This resulted in a reduction in memory requirements and improved inference speed.
2. Pruning: We used pruning techniques to remove unimportant weights from the model. We applied weight pruning, which involves removing the smallest weights in the network, and structured pruning, which removes entire filters or channels. This resulted in a significant reduction in the number of parameters in the model.

Rational behind using MobileNet Model

Understanding Normal Convolution operation

Suppose there is an input data of size $D_f * D_f * M$ where $D_f * Df$ is the size of the image and M is the number of channels in the image (3 for RGB).
Suppose there are N filters/kernels of size $D_f * D_f * M$. If a normal convolution operation is done, then, the output size will be $D_p * D_p * N$



The number of multiplications in 1 convolution operation = size of filter = $D_k * D_k * M$

Since there are N filters and each filter slides vertically and horizontally D_p times

the total number of multiplications becomes $D_p * D_p * N$ **(Multiplications per convolution)**

So for normal convolution operation

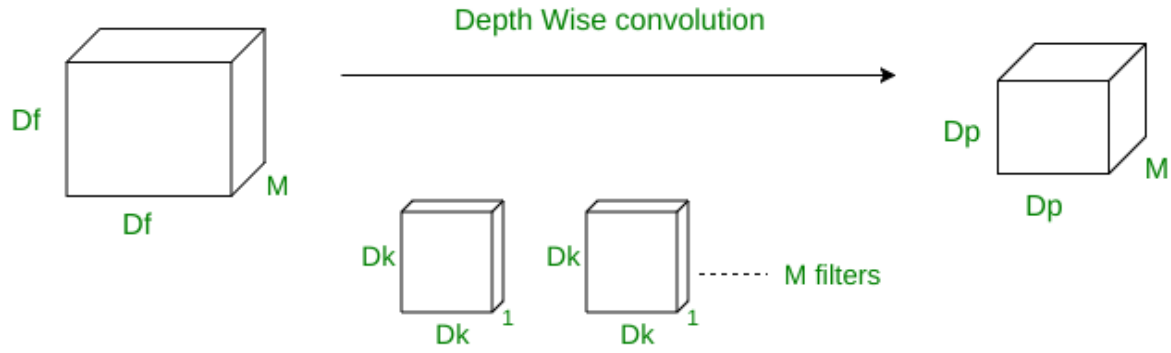
Total no of multiplications = $N * D_p^2 * D_k^2 * M$

Depth-Wise Separable Convolutions

Now lets look at depth-wise separable convolutions. This process is broken down into 2 operations

1. **Depth-wise convolutions**
2. **Point-wise convolutions**

In ***depth-wise operation***, convolution is applied to a **single channel** at a time unlike standard CNN's in which it is done for all the M channels. So here the filters/kernels will be of size $D_k * D_k * 1$. Given there are M channels in the input data, then M such filters are required. Output will be of size $D_p * D_p * M$



Since the filter are slided by $D_p * D_p$ times across all the M channels,

the total number of multiplications is equal to $M * D_p^2 * D_k^2$

In ***point-wise operation***, a **1×1 convolution** operation is applied on the M channels. So the filter size for this operation will be **1 x 1 x M**. Say we use N such

filters, the output size becomes $D_p * D_p * N$

A single convolution operation require 1 x M multiplications. Since the filter is being slided by $D_p * D_p$ times,

the total number of multiplications is equal to $M * D_p * D_p$ (no. of filters)

So for point wise convolution operation

Total no of multiplications = $M * D_p^2 * N$

Therefore, **for overall operation:**

Total multiplications = Depth wise conv. multiplications + Point wise conv. multiplications Total multiplications = $M * D_k^2 * D_p^2 + M * D_p^2 * N = M * D_p^2 * (D_k^2 + N)$

So for depth wise separable convolution operation

Total no of multiplications = $M * D_p^2 * (D_k^2 + N)$

Type of Convolution	Complexity
Standard	N x Dp2 x Dg2 x M
Depth wise separable	M x Dp2 x (Dk2 + N)

The Ratio of Depth wise separable Convolution Layer Operations to the Normal Convolution Layer Operations is

***Ratio(R)* = $1/N + 1/D_k^2$**

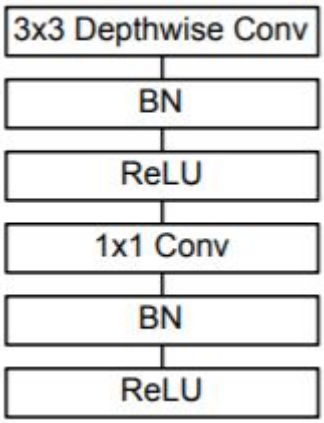
This implies that we can deploy faster convolution neural network models without losing much of the accuracy.

MobileNetModel Architecture

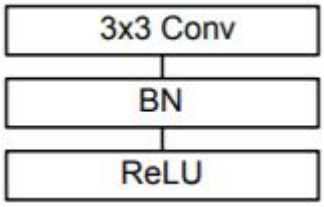
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

*All the other MobileNet Architectures are similar with slight modifications

Depthwise Separable Convolution Layer



Standard Convolution Layer

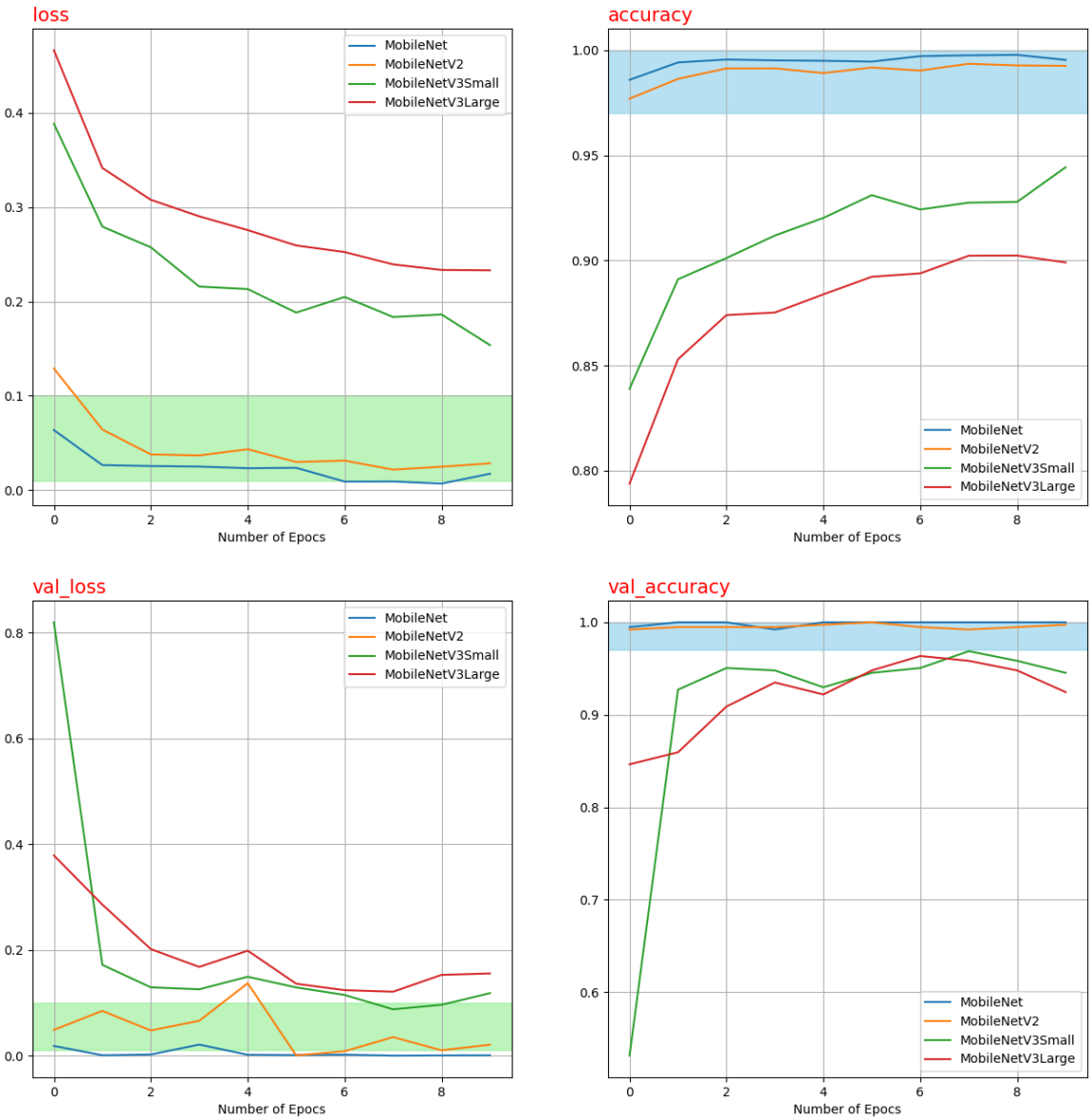


For the output I have used a Dense Layer with 2 Neuron Since It is a binary classificaiton problem there are only two classes either Mask or No Mask.

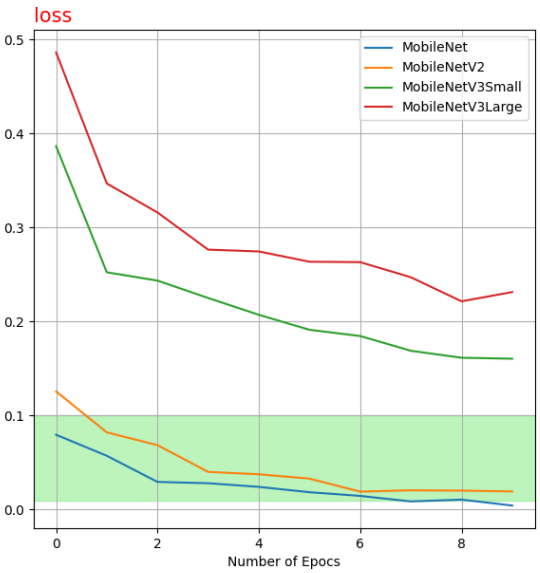
Results

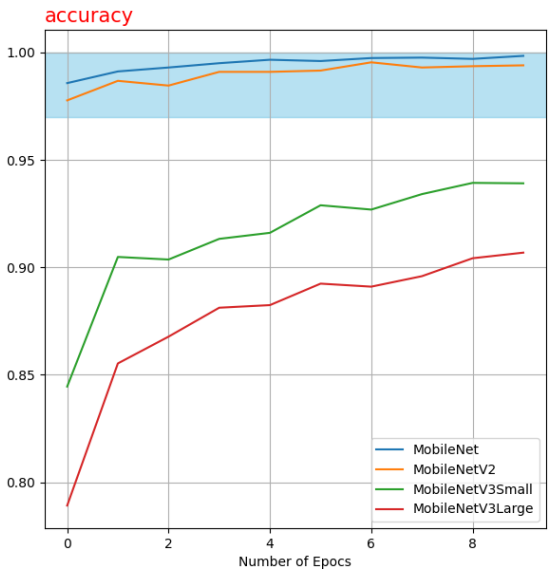
Training Plots

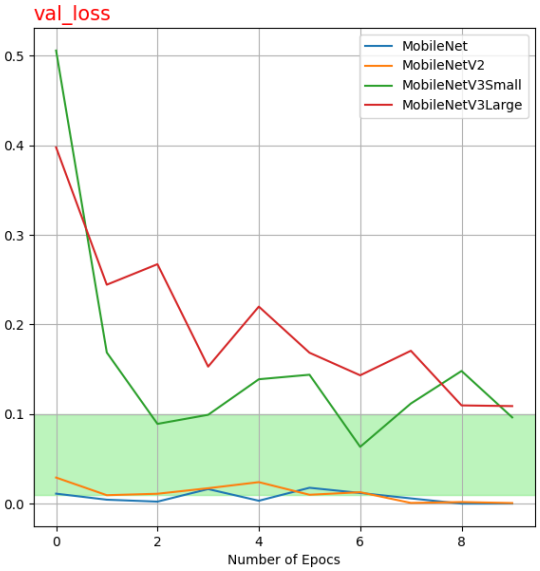
With 64 Neurons in the Middle Dense Layer

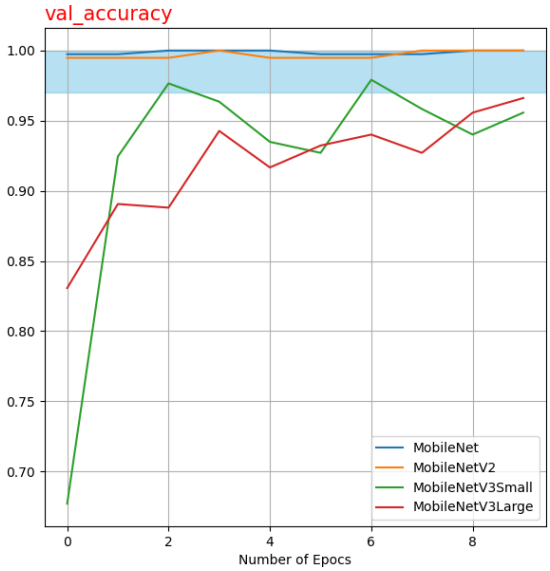


With 128 Neurons in the Middle Dense Layer

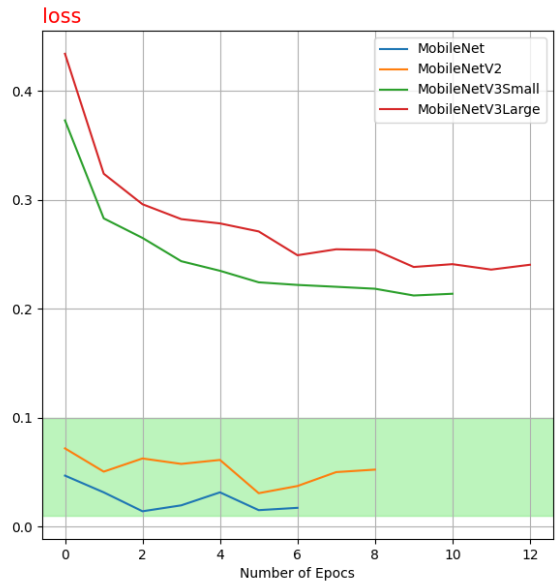


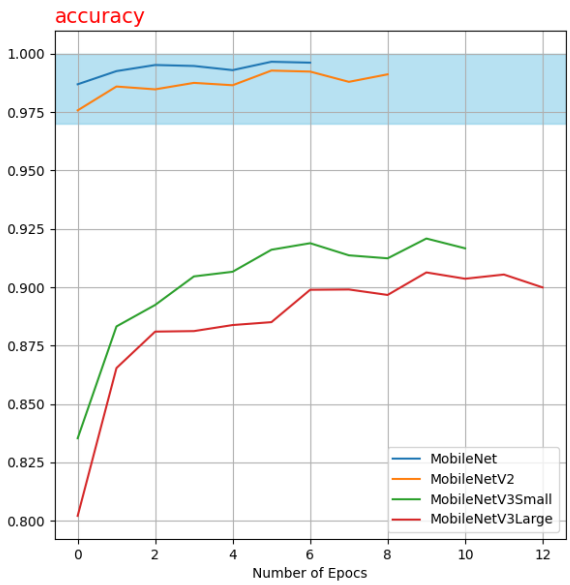


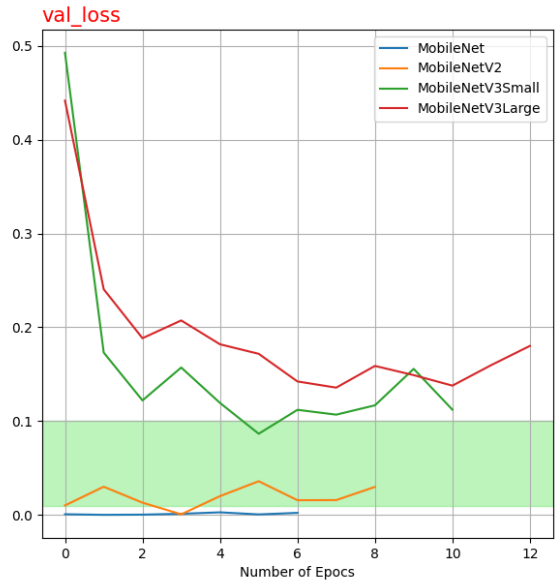


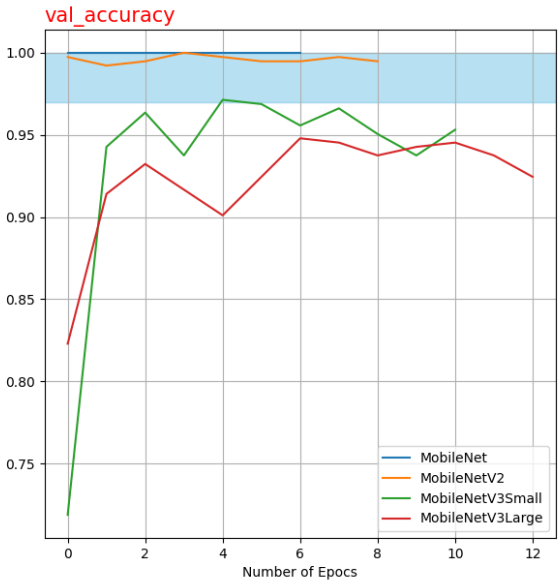


With no Middle Dense Layer









Accuracy on the Testing Data

Model	64 Neuron in Middle Dense Layer	128 Neurons in Middle Dense Layer	No Middle Dense Layer
MobileNet	99.90	99.90	99.50
MobileNetV2	99.09	98.89	99.40
MobileNetV3Small	96.88	97.28	96.17
MobileNetV3Large	95.06	94.56	93.85

Loss on Testing Data

Model	64 Neuron in Middle Dense Layer	128 Neurons in Middle Dense Layer	No Middle Dense Layer
MobileNet	0.0128	0.0161	0.0167
MobileNetV2	0.0498	0.0506	0.0455
MobileNetV3Small	0.1019	0.0802	0.1078
MobileNetV3Large	0.1396	0.1439	0.1610

Model Size Comparision

Model	64 Neuron in Middle Dense Layer	No Middle Dense Layer
MobileNet	24.9MB	13.3MB
MobileNetV2	24.5MB	10.0MB
MobileNetV3Small	10.9MB	4.37MB
MobileNetV3Large	23.5MB	12.6MB

Quantization

After performing post training Quantization we were able to reduce the Model Size of ResNet Model With No Middle Dense Layer from 13.3 MB to 3.38MB. The Face Detection Model took 700KB of space.

Dataset Link

FaceMask Dataset

References

Pruning in Keras

Quantization Aware Training

Post Training Quantization

MobileNetV1 Architecture Paper

ResNet Architecture Paper

Convolutional Neural Networks Explained - Coursera