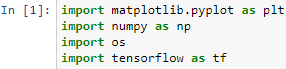
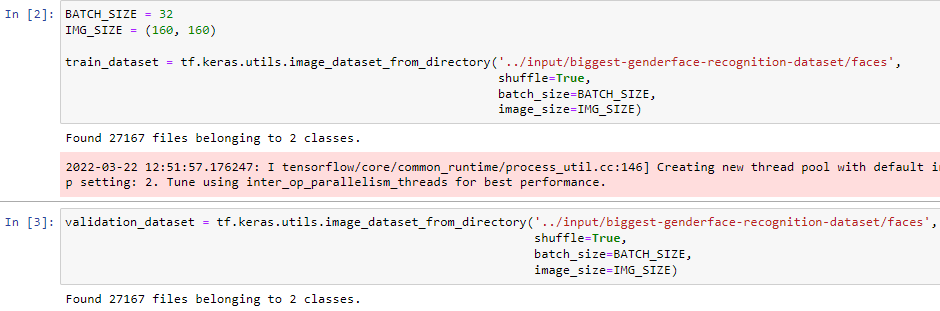
IMPORT NECESSARY LIBRARIES



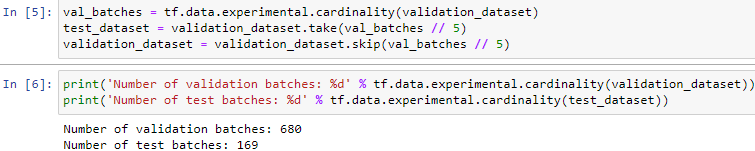
USE tf.keras.utils.image\_dataset\_from\_directory TO LOAD THE BIGGEST GENDER FACE RECOGNITION DATASET



DISPLAY 24 IMAGES FROM THE DATASET

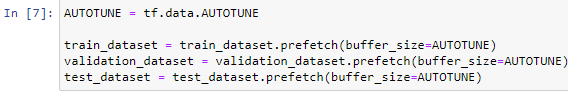


Determine how many batches of data are available in the validation set using cardinality, then move 20% of them to a test set



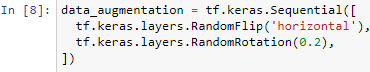
CONFIGURE THE DATASET FOR PERFORM

Use buffered prefetching to load images from disk without having I/O become blocking

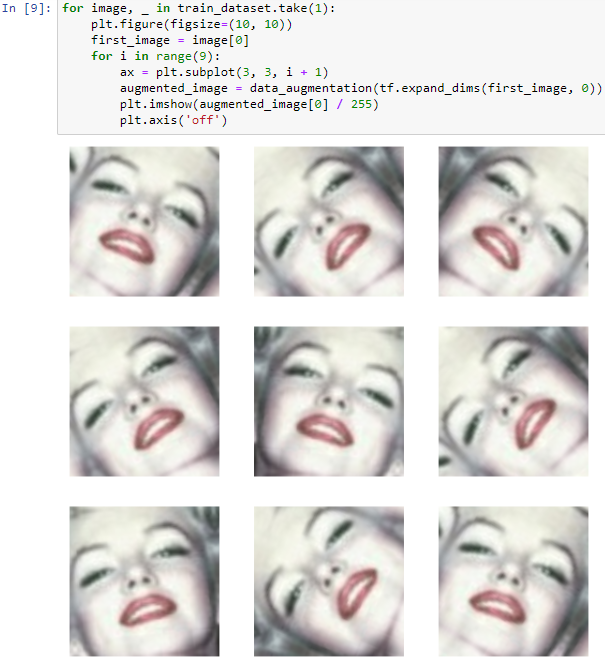


USE DATA AUGMENTATION

Random horizontal flip and Random rotation(0.2)



DATA AUGMENTATION ON A RANDOM IMAGE



Rescale pixel values

MobileNetV2 is the base model. This model expects pixel values in [-1, 1], but at this point, the pixel values in our images are in [0, 255]. To rescale them, use the preprocessing method included with the model.

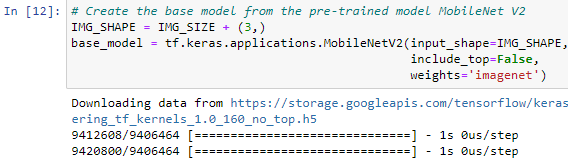


Create the base model from the pre-trained convnets

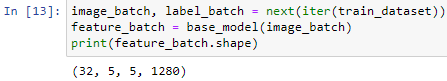
the base model is from the \*MobileNet V2\* model developed at Google. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories.

First, you need to pick which layer of MobileNet V2 you will use for feature extraction. We choose to depend on the very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the final/top layer.

First, instantiate a MobileNet V2 model pre-loaded with weights trained on ImageNet. By specifying the \*include\_top=False\* argument, we load a network that doesn't include the classification layers at the top, which is ideal for feature extraction.



feature extractor converts each 160x160x3 image into a 5x5x1280 block of features



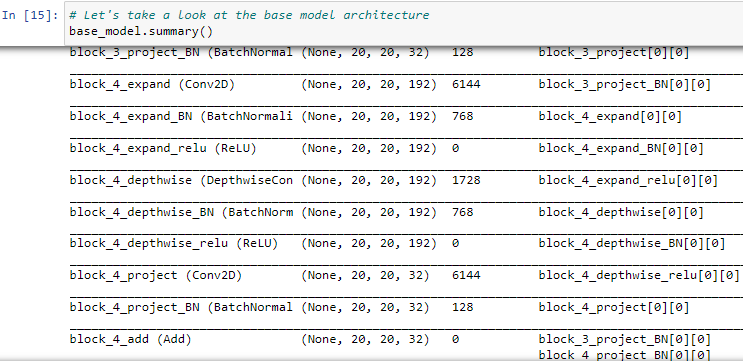
Freeze the convolutional base

It is important to freeze the convolutional base before you compile and train the model. Freezing (by setting layer.trainable = False) prevents the weights in a given layer from being updated during training. MobileNet V2 has many layers, so setting the entire model's trainable flag to False will freeze all of them.



IMPORTANT NOTE ABOUT BATCHNORMALIZATION LAYERS

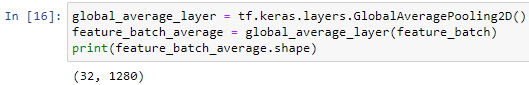
When you set layer.trainable = False, the BatchNormalization layer will run in inference mode, and will not update its mean and variance statistics.



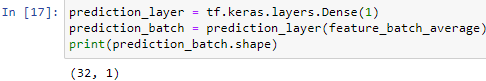
When you unfreeze a model that contains BatchNormalization layers in order to do fine-tuning, you should keep the BatchNormalization layers in inference mode by passing training = False when calling the base model. Otherwise, the updates applied to the non-trainable weights will destroy what the model has learned.

Add a classification head

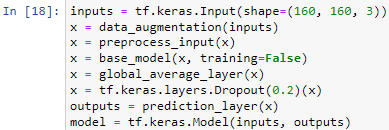
To generate predictions from the block of features, average over the spatial 5x5 spatial locations, using a tf.keras.layers.GlobalAveragePooling2D layer to convert the features to a single 1280-element vector per image.



Apply a tf.keras.layers.Dense layer to convert these features into a single prediction per image. You don't need an activation function here because this prediction will be treated as a logit, or a raw prediction value. Positive numbers predict class 1, negative numbers predict class 0.



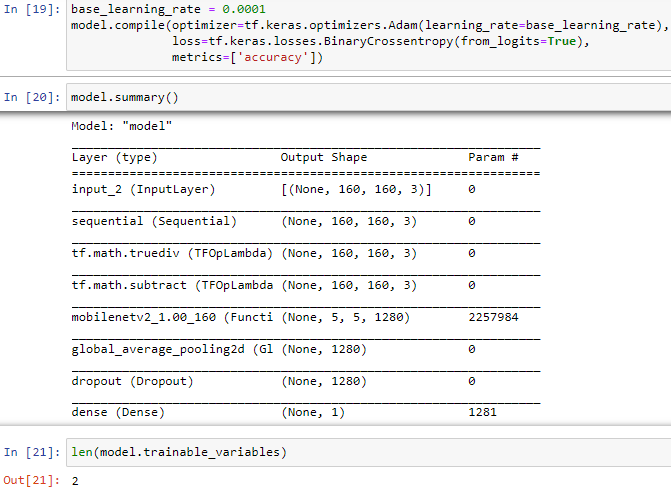
Build a model by chaining together the data augmentation, rescaling, base\_model and feature extractor layers using the Keras Functional API. use training=False as our model contains a BatchNormalization layer.



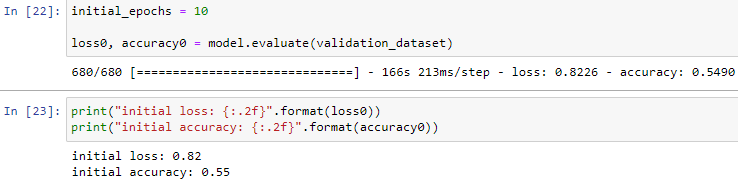
Compile the model

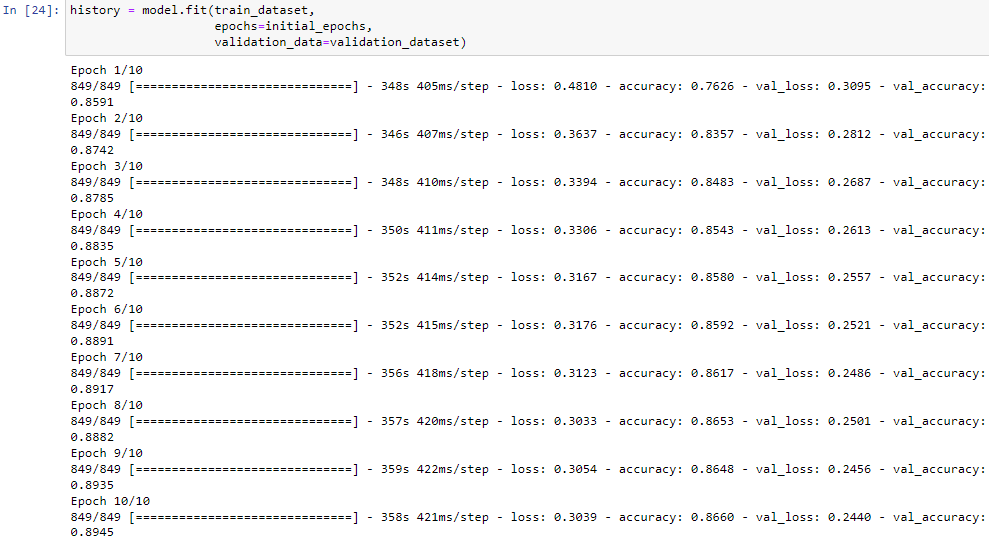
Compile the model before training it. Since there are two classes, use the tf.keras.losses.BinaryCrossentropy loss with from\_logits=True since the model provides a linear output.

(((The 2.5 million parameters in MobileNet are frozen, but there are 1.2 thousand trainable parameters in the Dense layer. These are divided between two tf.Variable objects, the weights and biases.)))



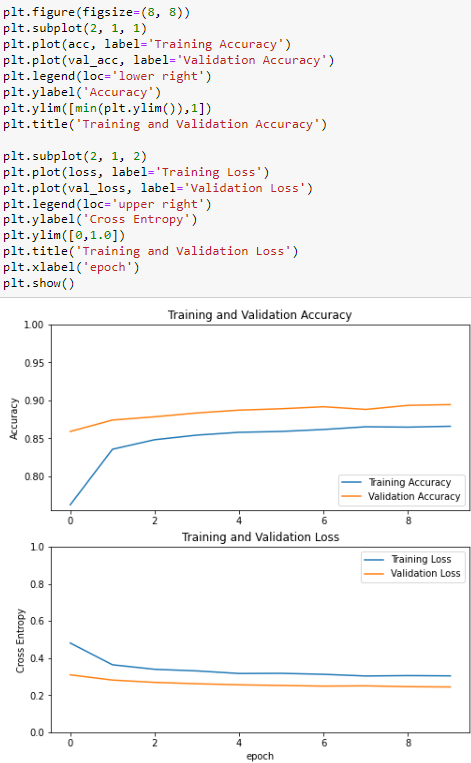
TRAINING THE MODEL 10 EPOCHS 89.45% VALIDATION ACCURACY





Learning curves

Learning curves of the training and validation accuracy/loss when using the MobileNetV2 base model as a fixed feature extractor.



(((Note: Validation metrics are clearly better than the training metrics, the main factor is because layers like tf.keras.layers.BatchNormalization and tf.keras.layers.Dropout affect accuracy during training. They are turned off when calculating validation loss.

To a lesser extent, it is also because training metrics report the average for an epoch, while validation metrics are evaluated after the epoch, so validation metrics see a model that has trained slightly longer.)))

Fine tuning

In the feature extraction experiment, you were only training a few layers on top of an MobileNetV2 base model. The weights of the pre-trained network were not updated during training.

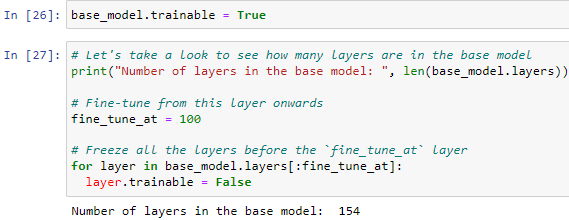
One way to increase performance even further is to train (or "fine-tune") the weights of the top layers of the pre-trained model alongside the training of the classifier you added. The training process will force the weights to be tuned from generic feature maps to features associated specifically with the dataset.

Note: This should only be attempted after training the top-level classifier with the pre-trained model set to non-trainable. If you add a randomly initialized classifier on top of a pre-trained model and attempt to train all layers jointly, the magnitude of the gradient updates will be too large (due to the random weights from the classifier) and your pre-trained model will forget what it has learned.

Also, you should try to fine-tune a small number of top layers rather than the whole MobileNet model. In most convolutional networks, the higher up a layer is, the more specialized it is. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.

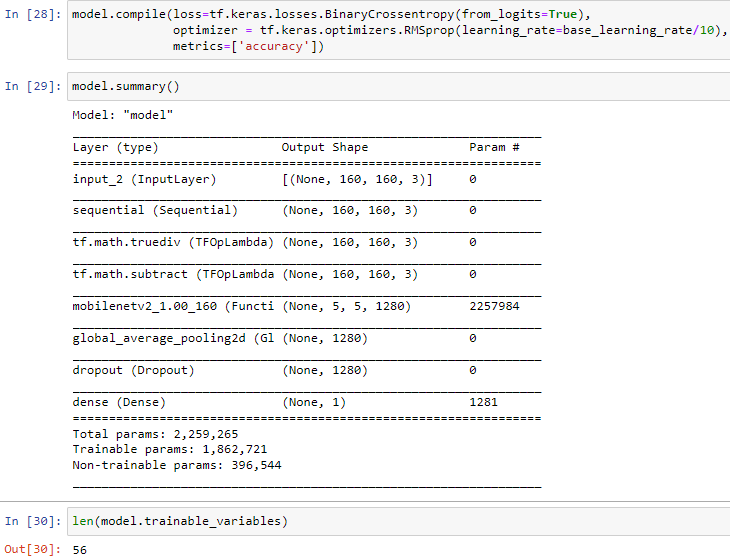
Un-freeze the top layers of the model

unfreeze the base\_model and set the bottom layers to be un-trainable. Then, you should recompile the model (necessary for these changes to take effect), and resume training.



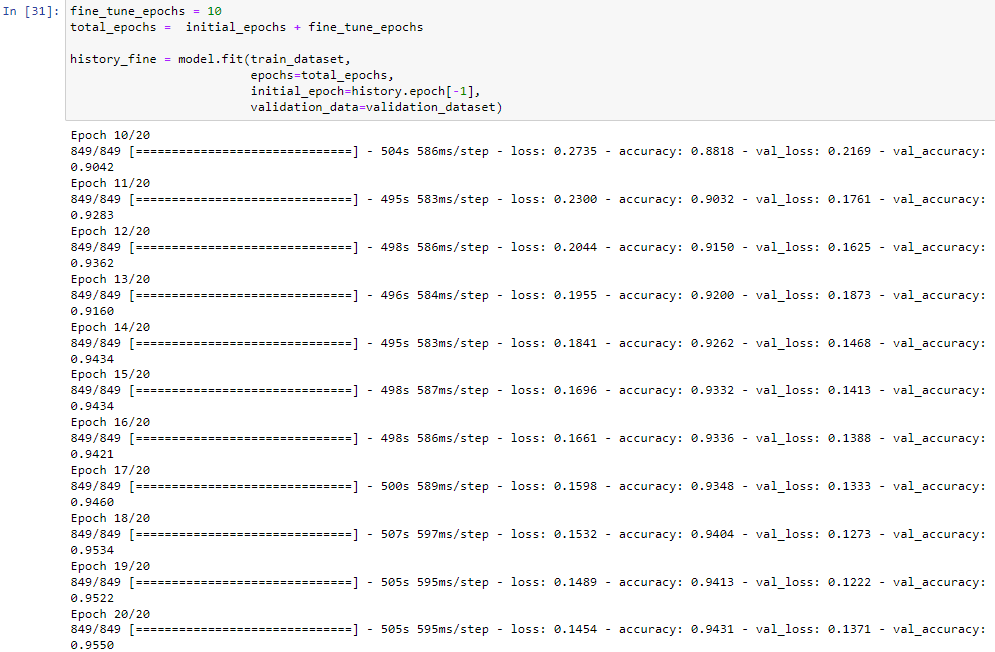
Compile the model

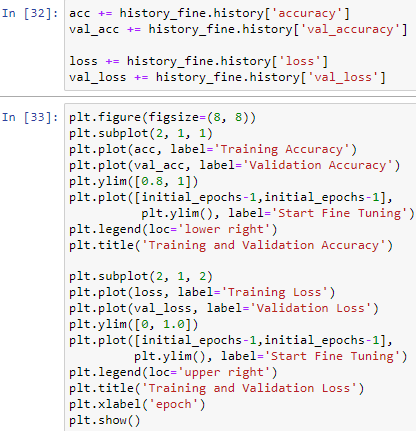
As you are training a much larger model and want to readapt the pretrained weights, it is important to use a lower learning rate at this stage. Otherwise, your model could overfit very quickly.

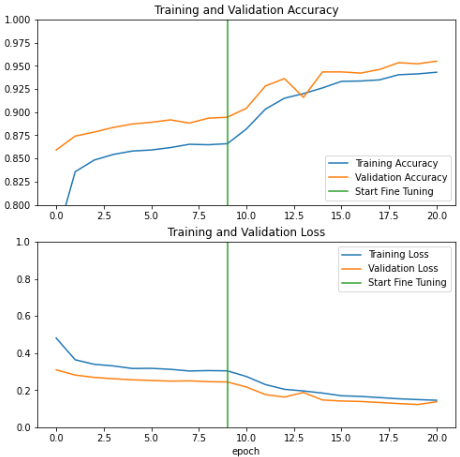


Continue training the model

We trained convergence earlier, this step improves our accuracy from 89.45% to 95.50% validation accuracy

learning curves of the training and validation accuracy/loss when fine-tuning the last few layers of the MobileNetV2 base model and training the classifier on top of it. The validation loss is much higher than the training loss, there is some overfitting.

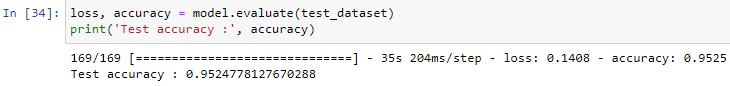




After fine tuning the model nearly reaches 98% accuracy on the validation set.

EVALUATION AND PREDICTION

TEST ACCURACY 95.24%



PREDICTIONS ON TEST DATA

