A Deep Learning Approach to Face Classification

Maël Fabien

Télécom ParisTech, 2019

Abstract. The goal of this challenge is to develop a binary classification system which tries to predict the class of an image. One of the challenges of the classification task is that the meaning of the labels (1 or 0) is unknown. We are given a training data set which contains the raw images and their corresponding labels, as well as a validation set. Deep Learning approaches have been explored, and complex architectures such as XCeption architectures tend to outperform other architectures.

1 The training set

The training set contains raw images and their labels. There are 116157 images in this data set. Each image is of size 56 x 56 x 3 (the last dimension encodes the color information: red-green-blue). For each image there is a label, either 0 or 1.

The validation set only contains raw images. There are 27013 images in this data set, where the image format is the same as before.

Due to an imbalance in the classes, the chosen performance criteria is the average class accuracy.

2 Exploratory data analysis

There are 47'269 observations belonging to class 0, and 68'888 belonging to class 1. Having no other information on what the labels mean, it is hard to identify a clear pattern using only the labels visual information.

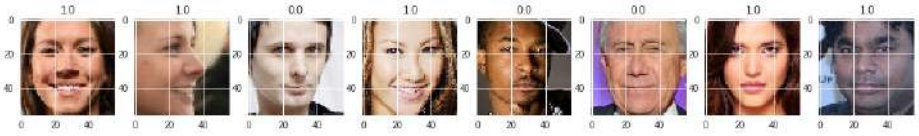

Fig. 1. Sample data

We can take a look at the most important regions using a simple feature importance over a flattened image using XGBoost classification.

The most important features appear to be located around the eyes and the nose :

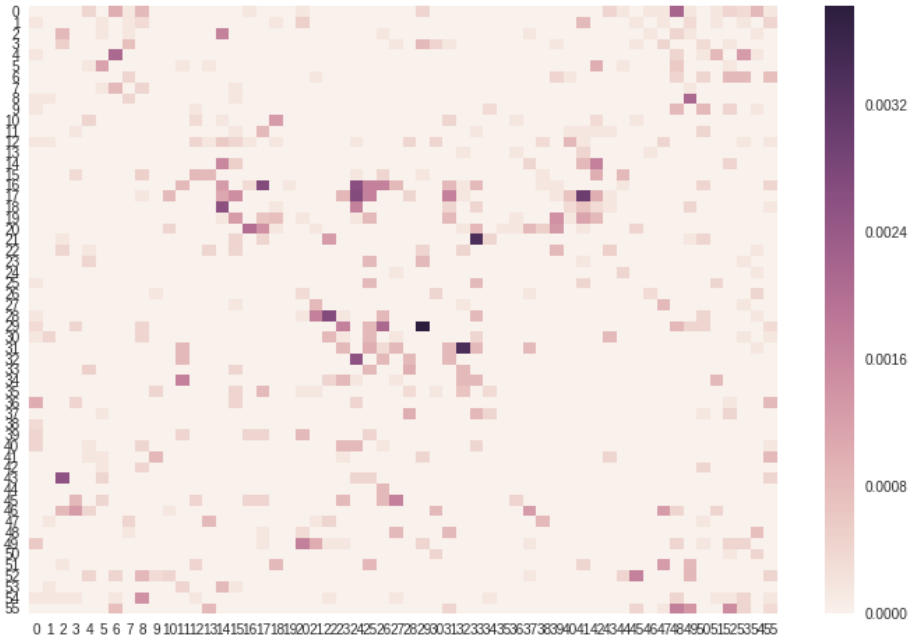

Fig. 2. Pixel Feature Importance

This could mean that the eyes carry an important information for the classification task we are given. The information carried in the eyes and nose is however always relatively important when it comes to computer vision tasks on face images. Cropping the image around the eyes and nose could therefore imply a large overfitting.

To better understand the distinction between classes 0 and 1, we can also take a look at average faces of both classes.

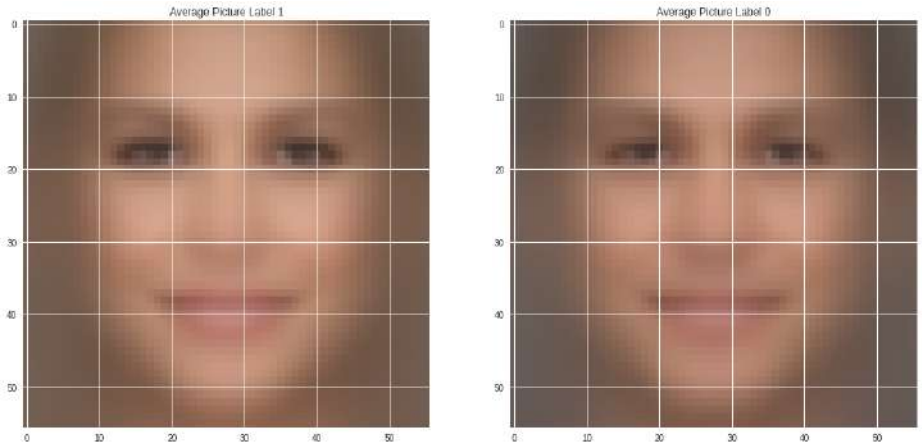

Fig. 3. Average faces per label

There seems to be a lighter shade on the labels 1, and also more feminine traits for labels 1 than for labels 0.

The color difference can be verified by plotting the pixel value distribution, as presented in Figure 4.

Moreover, we can display the average difference face between class 0 and class 1. The average face highlights some traits on the chicks, mouth and eyes.

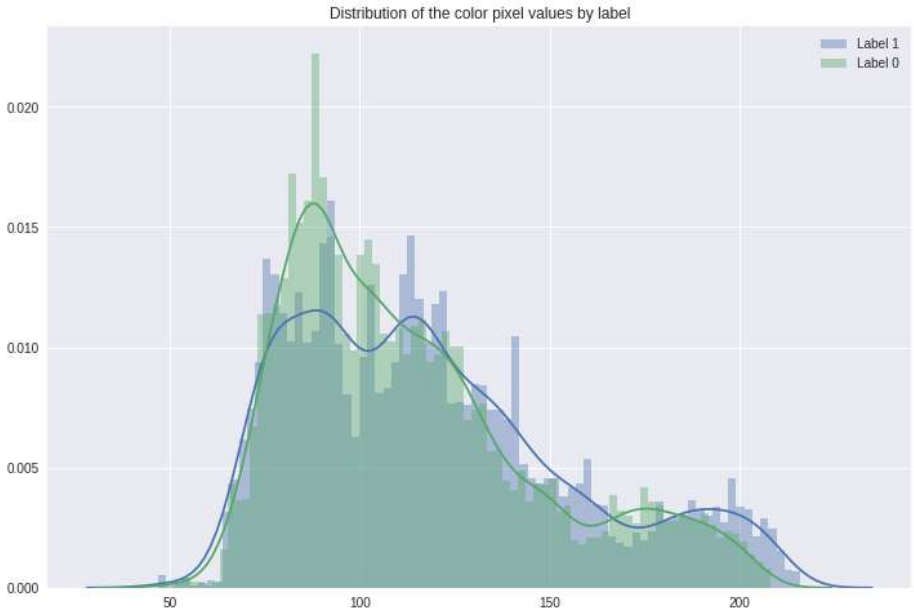

Fig. 4. Pixel value distribution per label

3 Models

In this section, we'll review the main challenges and techniques that have been used, and the performance of the different algorithms.

3.1 Prevent Over-fitting

The validation set provided is a subset of the train set. Therefore, it is impossible to rely on the validation score to estimate the accuracy of the model.

The best approaches remains to :

- apply cross validation, average the score on all folds and identify the methodology that performs the best
- weight the loss function according to the initial class imbalance
- do data augmentation on the initial images
- add regularization and drop out to our deep learning models to avoid co-adaptation

Data augmentation techniques allow a more robust model but bring some concern due to the unknown meaning of the label. If the label is a measure of distance between eyes or a measure of symmetry of the face for example, it then becomes impossible do to data augmentation since they include rotation or cropping of the images.

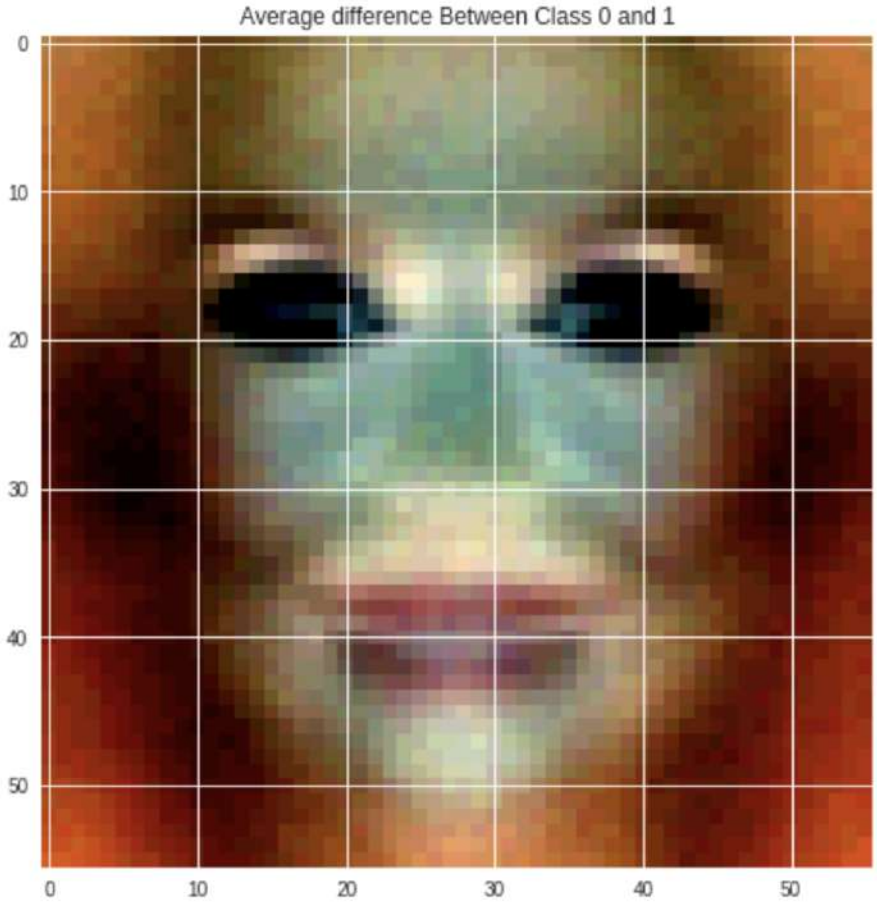

Fig. 5. Average face difference between classes 0 and 1

For all deep learning models, we'll weight the loss function according the the initial class imbalance and try a version with and without data augmentation.

Visually, the validation loss and accuracy curves should more or less follow the ones of the training loss and accuracy.

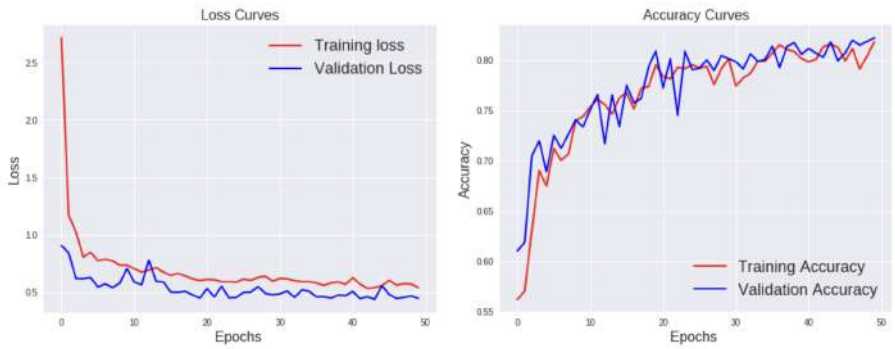

Fig. 6. Example of a model fitting process

3.2 Approaches

We'll explore several strategies.

- build an auto-encoder to reduce the dimension of the data and apply a convolutional neural network on top
- apply a convolutional neural network (CNN) architecture on the input image
- build more complex architectures using neural networks (Inception, XCeption, VGG16...)
- extract manually HOG and sliding HOG features and apply a Support Vector Machine Classifier

For deep learning models, we'll always end the model by a Dense layer with 2 neurons, one for each class, and apply a softmax activation layer, to transform the output into a probability that an observation belongs to each class. This way, in prediction, we simply choose the class with the highest probability.

3.3 Auto-Encoding the input images to reduce the dimension

One way to overcome the memory issue is to reduce the dimension of the initial problem. There are several ways to do this :

- using Principal Component Analysis, which can be proven to have converging results with Auto-Encoders
- delete the color dimension by applying gray scale. This option relies on the hypothesis that the color does not have much importance, which is quite restrictive.
- apply Auto-Encoding and fetch the reduced dimension bottleneck, which is the method chosen here

The structure of the Auto-Encoder is described in figure 5.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 56, 56, 3)	0
conv2d_5 (Conv2D)	(None, 56, 56, 2)	56
max_pooling2d_1 (MaxPooling2	(None, 28, 28, 2)	0
conv2d_6 (Conv2D)	(None, 28, 28, 1)	19
conv2d_7 (Conv2D)	(None, 28, 28, 1)	10
conv2d_8 (Conv2D)	(None, 28, 28, 2)	20
up_sampling2d_2 (UpSampling2	(None, 56, 56, 2)	0
conv2d_9 (Conv2D)	(None, 56, 56, 3)	57

Total params: 162
Trainable params: 162
Non-trainable params: 0

Fig. 7. Auto-Encoder architecture

During the fitting of the auto-encoder, we can plot the loss over the number of epochs :

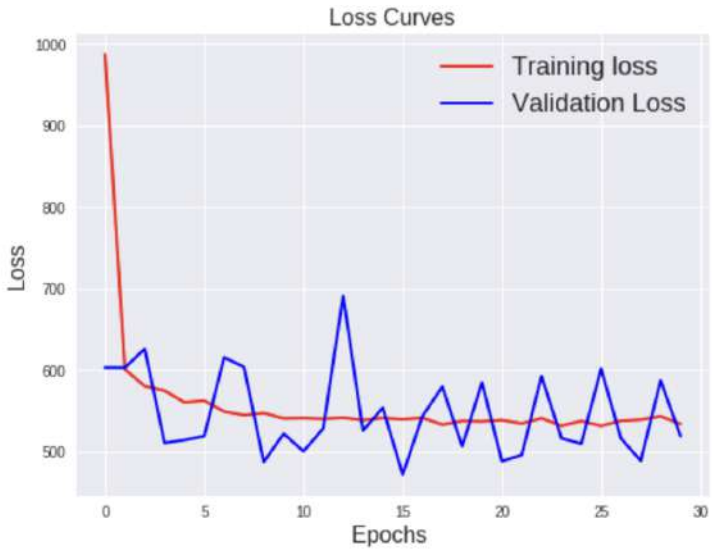

Fig. 8. Model Loss during training

We then define the bottleneck dimension (28*28*1, which is only 8% of the original dimension). We build a model that takes as input the image, and as output the encoded bottleneck :

```
encoder = Model(inputs = input_img, outputs = encoded)
X_train_enc = encoder.predict(X_train)
```

We can visualize the encoded images in their reduced dimension (28*28*1).

Fig. 9. Original, Encoded and Decoded images

We lose some information regarding the image color mainly, by reducing the input image to a 28*28*1.

We can reduce again the dimension of the encoded images using T-SNE for visualization purposes.

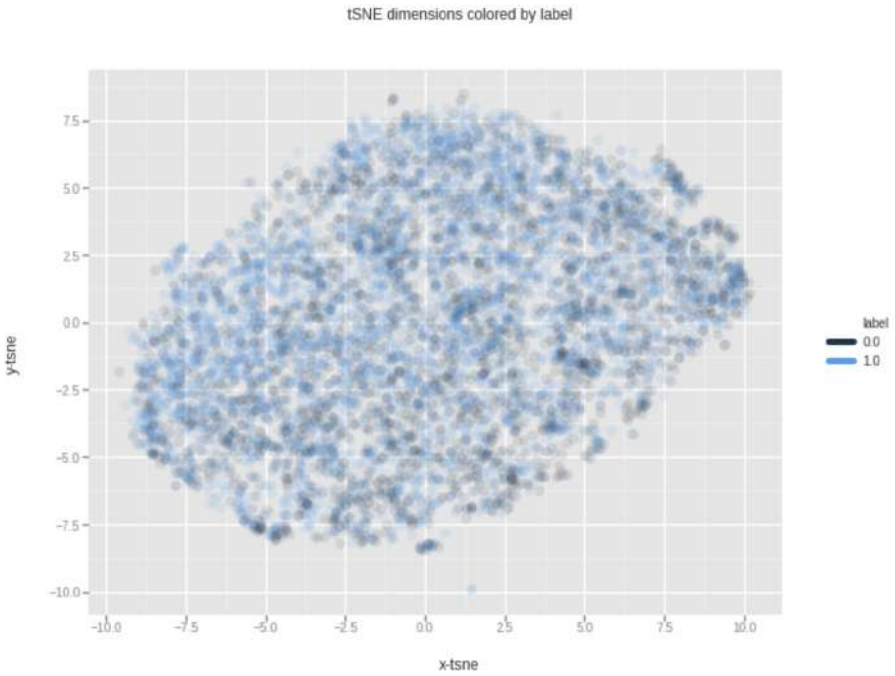

Fig. 10. Original, Encoded and Decoded images

There is no clear pattern to identify the image clusters. The approach chosen here is to apply a Convolutional Neural Network on the encoded images. The encoded images can still be treated as images, and CNNs can be used.

Applying a simple architecture made of convolution layers, batch normalization, dropout and max pooling, the validation accuracy is 80.5% which means that we did not improve the accuracy of the classifier.

Fig. 11. Model fitting on auto-encoded images

3.4 Large Scale Kernel Methods

Kernel methods offer flexibility when it comes to such complex problem, since the feature space includes an infinite dimensional mapping of the problem. However, this is nearly impossible to apply when we deal with large amount of data and a large number of features. One way to overcome this issue is to apply large scale kernel methods such as Random Kernel Features SVM. This idea is to pre-process the data so that we are able to fit a simple linear Support Vector Classifier on top.

```
def random_features(X_train, gamma, c, seed=42):
    rng = np.random.RandomState(seed)
    n_samples, n_features = X_train.shape

    W = np.random.normal(0, np.sqrt(2*gamma), (n_features, c))
    b = np.random.uniform(0, 2*math.pi, (1,c))

    X_new_train = np.sqrt(2/n_features) * np.cos(np.dot(X_train, W) + b)

    return X_new_train
```

This method heads an average cross validation accuracy of : 74.5%

3.5 Simple Deep Learning Architecture

Deep Learning Architectures involving Convolutional Neural Networks typically perform well on such tasks. We will first of all try to build a simple deep learning architecture involving simple sequential layers.

```
def createModel():
  model = Sequential()

  model.add(Conv2D(4, (3, 3), padding='same', input_shape=(56,56,3)))
  model.add(Conv2D(8, (3, 3), padding='same', activation='relu'))
  model.add(Conv2D(16, (3, 3), padding='same', activation='relu'))
  model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2, 2), strides=None, padding='same'))
  model.add(Dropout(0.20))

  model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
  model.add(Dropout(0.20))

  model.add(Flatten())

  model.add(Dense(512))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.25))

  model.add(Dense(256))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.25))

  model.add(Dense(96))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.25))

  model.add(Dense(2))
  model.add(Activation('softmax'))

  return model
```

The Batch Normalization layer controls for co-adaptation of successive layers, and prevents from observing vanishing gradient issues. The Dropout layer is an-

other protection against overfitting and co-adaptation of layers. The MaxPooling layer is a way to reduce the dimension of the problem. This creates a total of more than 13'000'000 parameters to train.

After successive layers, we then apply dense layers, reduce the dimension to 2, and apply a final softmax activation layer to assign probabilities to both classes.

The overall accuracy of this approach is 84.5% on a 20% validation set. There is a large limit in the model performance since it seems to overfit after only 5 epochs.

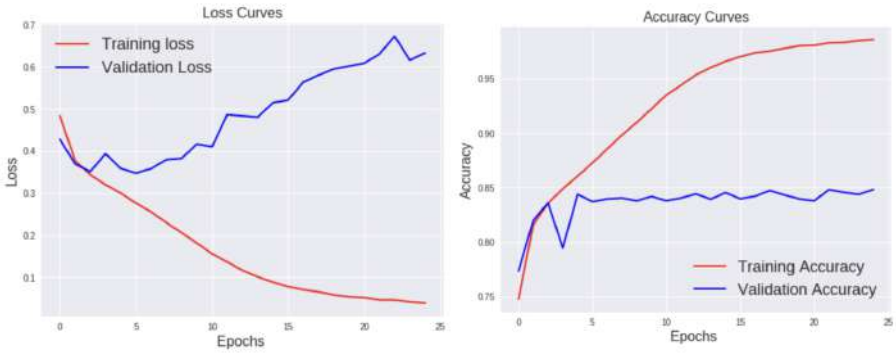

Fig. 12. Model Performance on 20% Validation set

3.6 XCeption

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. It was developped by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads them to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions. The architecture of XCeption networks, is made of an entry, middle and exit module.

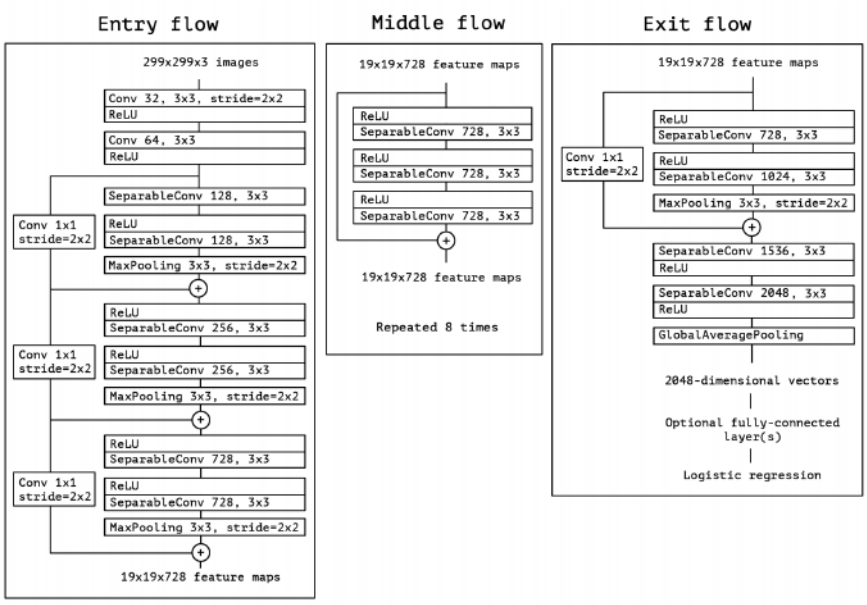

Fig. 13. XCeption Architecture

XCeption Architecture achieves the best accuracy reached so far, at 87.0% on the validation split, but tend to overfit quite easily on such task. We can illustrate the overfitting through the graph in Figure 13. To make the model more robust, we also train in on augmented images.

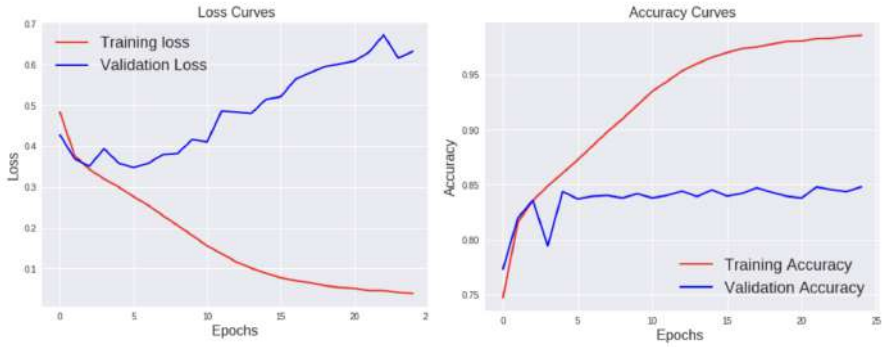

Fig. 14. Model fitting of XCeption

The Best XCeption model has been achieved using Data Augmentation.

```
datagen = ImageDataGenerator(
        zoom_range=0.20,
        rotation_range=10,
        width_shift_range=0.10,
        height_shift_range=0.10,
        horizontal_flip=True,
        vertical_flip=False)

epochs = 35
batch_size = 128
```

3.7 DenseNet

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. Dense Convolutional Network (DenseNet), introduced in 2016, connects each layer to every other layer in a feed-forward fashion.

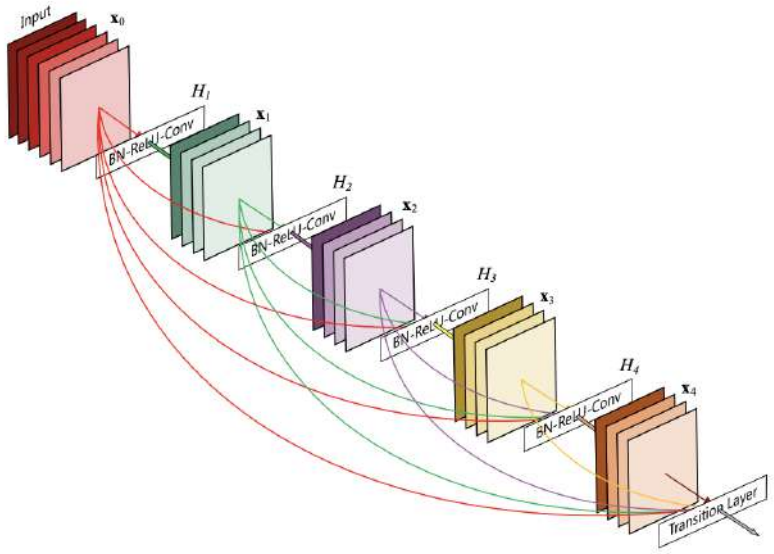

Fig. 15. DenseNet Architecture

DenseNet was originally developped for the ImageNet challenge, an image classification challenge with 1024 classes. Since our challenge is simpler, we need to add a Dense layer with 2 neurons and a softmax activation function.

There are substantially less parameters to estimate than in other architecture, which makes the training process quite computationally efficient. DenseNet heads an accuracy of 84.5% on the validation set.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 56, 56, 3)	0
batch_normalization_v1 (Batc	(None, 56, 56, 3)	12
densenet121 (Model)	(None, 1024)	7037504
flatten (Flatten)	(None, 1024)	0
fc2 (Dense)	(None, 2)	2050

Total params: 7,039,566
Trainable params: 6,955,912
Non-trainable params: 83,654

Fig. 16. DenseNet Model Summary

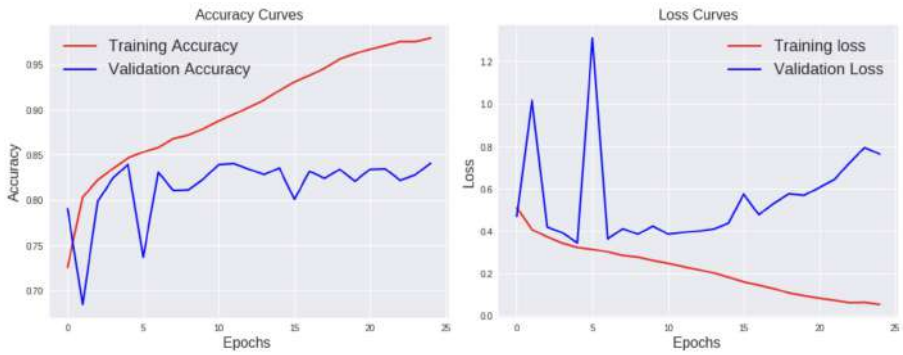

Fig. 17. Model Performance on 20% Validation set

3.8 VGG16

VGG16 usually refers to a deep convolutional network for object recognition developed and trained by Oxford's renowned Visual Geometry Group (VGG), which achieved very good performance on the ImageNet dataset.

I have chosen to duplicate the architecture of VGG16 for our classification task, by adding a final dense layer of 2 classes :

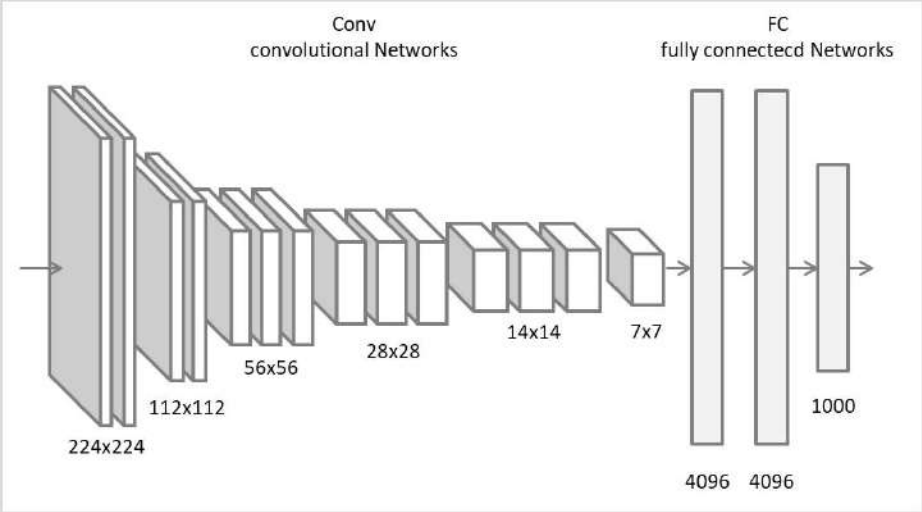

Fig. 18. VGG16 Architecture

VGG16 architecture achieves an accuracy of 82.5% on a 20% validation set.

4 Ensemble Model

An ensemble model has also been built as follows : we combine the soft probabilities of the different classifiers. We then train a random forest classifier on top of the concatenated values. This heads an accuracy of 85.1% in prediction.

5 Models Performance Summary

We can summarize the performance of the different models on a 20% validation set and on augmented images. The validation accuracy will is designated by "Val. Accuracy", and the accuracy on the whole augmented images is designated by "Aug. Accuracy". The accuracy in prediction is also presented.

Table 1. Classification accuracy per model

Classifier	Val. Accuracy	Aug. Accuracy	Prediction Accuracy
Auto-Encoder + CNN	80.5 %	79.8	-
CNN	85.0 %	84.6%	84.7%
XCeption	-	87%	87.2%
DenseNet	84.7%	-	-
ResNet	84.5%	-	-
VGG16	82.5%	80.5%	-
Ensemble Model	-	-	85.1%

For the prediction on the final set of data, the XCeption model trained on augmented images has been chosen. An accuracy in prediction of 87.2% was achieved.

The ensemble model is made of a random forest trained on the outputs of the XCeption model and the CNN Architecture.

6 Conclusion

We have seen that in the context of face classification, deep learning architectures seem to outperform classical approaches in terms of accuracy scores. Complex architectures with typically over 5 million parameters react well to data augmentation and provide more robust models. XCeption model was selected in our case and performed better than other models. Preventing overfitting for such tasks is crucial.