Machine Learning Engineer Nanodegree
Capstone Project

Benjamin Rouillé d'Orfeuil
June 7, 2018

I. Definition

Project Overview

Yelp is a social networking site that publishes crowd-sourced reviews about local businesses. About two years ago, Yelp challenged Machine Learning practitioners to build a model that automatically tags restaurants with multiple labels using a dataset of user-submitted photographs. The goal of this project is to develop such a model.

The competition was hosted by Kaggle, a platform where data scientists use their skills to produce the best models for predicting and describing datasets uploaded by companies and users. The Yelp Restaurant Photo Classification competition can be found at this url:

`https://www.kaggle.com/c/yelp-restaurant-photo-classification`

Problem Statement

Back in Spring 2016, when this competition took place, Yelp users were able to upload photographs and write reviews without having to tag the venue with labels. It follows that some restaurants can be left un- or only partially-categorized. There are lot of photographs uploaded on the Yelp site and, for this reason, business labels need to be predicted by a machine learning algorithm if those have not been selected by users during the submission process.

Each photograph belongs to a business and the task is to predict the business attributes purely from the business photographs. Note that this is a multi-instance multi-label classification problem. Indeed, each business has multiple photographs and predictions need to be done at the business level. Likewise, multiple labels can be assigned to the same business and, as a result, potential dependencies among labels need to be accounted for by the classifier. Classifying real world images is a complex endeavor and, often, deep neural networks are best suited to address such problems.

Metrics

The harmonic mean between precision (p) and recall (r), the F_1 score, is used to evaluate the performance of the algorithm:

$$F_1 = 2\frac{p \cdot r}{p + r} \text{ where } p = \frac{tp}{tp + fp} \text{ and } r = \frac{tp}{tp + fn}$$

In the above formula, tp, fp and fn denote the true positive, false positive and false negative counts, respectively. In a classification task, $p = 1$ for class i means that every item labeled as belonging to class i does indeed belong to class i whereas $r = 1$ for class i means that every item from class i is labeled as belonging to class i. Though, precision says nothing about the number of items from class i that are mislabeled (fn) and recall says nothing about the number of items that are incorrectly labeled as belonging to class i (fp).

Both precision and recall are obviously relevant for a multi-label classification problem and, for this reason, the F_1 score appears as being the best evaluation metric for this project. A good retrieval algorithm will maximize precision and recall simultaneously and, consequently, good performance on both is favored over extremely good performance on one and poor performance on the other. Note that all Kagglers participating to this competition use the F_1 score as evaluation metric. Thus, models can directly be compared and participants easily ranked.

II. Analysis

Data Exploration

The various datasets and inputs for this competition are available in the data section of the Kaggle competition webpage[1]. There are 6 different datasets for this competition:

- `train.csv`;
- `train_photo_to_biz_ids.csv`;
- `test_photo_to_biz_ids.csv`;
- `train_photos.tgz`;
- `test_photos.tgz`;
- and `sample_submission.csv`.

There are 9 different business attributes that are encoded as integer ranging from 0 to 8:

0. `good_for_lunch`;
1. `good_for_dinner`;
2. `takes_reservations`;
3. `outdoor_seating`;
4. `restaurant_is_expensive`;
5. `has_alcohol`;
6. `has_table_service`;
7. `ambience_is_classy`;
8. and `good_for_kids`.

The `train.csv` file provides the list of labels for each business id. As mentioned previously, each business has multiple photographs. The correspondence between the photo id and the business id is given in `train_photo_to_biz_ids.csv` (`test_photo_to_biz_ids.csv`) for the training (test) dataset. The photographs for the training (test) dataset are compressed and combined in `train_photos.tgz` (`test_photos.tgz`). All images have `jpg` format and are named after their photo id. There are 234,842 (237,152) photos and 2,000 (10,000) restaurants in the training (test) dataset[2]. Finally, a template file (`sample_submission.csv`) provides the format for the submission. Predictions enclosed in this file are used by Kaggle to calculate the model performance and in turn rank participants.

Table 1 features a small sample of the training dataset. It relates business id, labels and photos id. The number of photos available for each business is calculated and reported in the last column of Table 1. One can see that this statistic varies from one entry to the other. The histogram of the number of photos

	labels	photos id	# photos
business id			
1000	(1, 2, 3, 4, 5, 6, 7)	[438623, 325966, 227692, 407856, 368729, 16319...	54
1001	(0, 1, 6, 8)	[298536, 20346, 8457, 308694, 349310, 407838, ...	9
100	(1, 2, 4, 5, 6, 7)	[338465, 328433, 243861, 361777, 127198, 46652...	84
1006	(1, 2, 4, 5, 6)	[46472, 341947, 396253, 75316, 42330, 244095, ...	22
1010	(0, 6, 8)	[118251, 219940, 27517, 8578, 148347, 433559, ...	11
101	(1, 2, 3, 4, 5, 6)	[13736, 393696, 286907, 86169, 243460, 254663,...	121
1011	(2, 3, 5, 6)	[372371, 116870, 411981, 208597, 127752, 18839...	70
1012	(1, 2, 3, 5, 6)	[287385, 232258, 388225, 151345, 417121, 32754...	37
1014	(1, 2, 4, 5, 6)	[407910, 33911, 269241, 374218, 256236, 296370...	32
1015	(1, 5, 6, 7)	[456770, 44056, 128542, 373344, 87938, 148452,...	145

Table 1: Training dataset. The labels, photo id and number of photos are given for each business id. This table is extracted from the `eda.ipynb` Jupyter notebook.

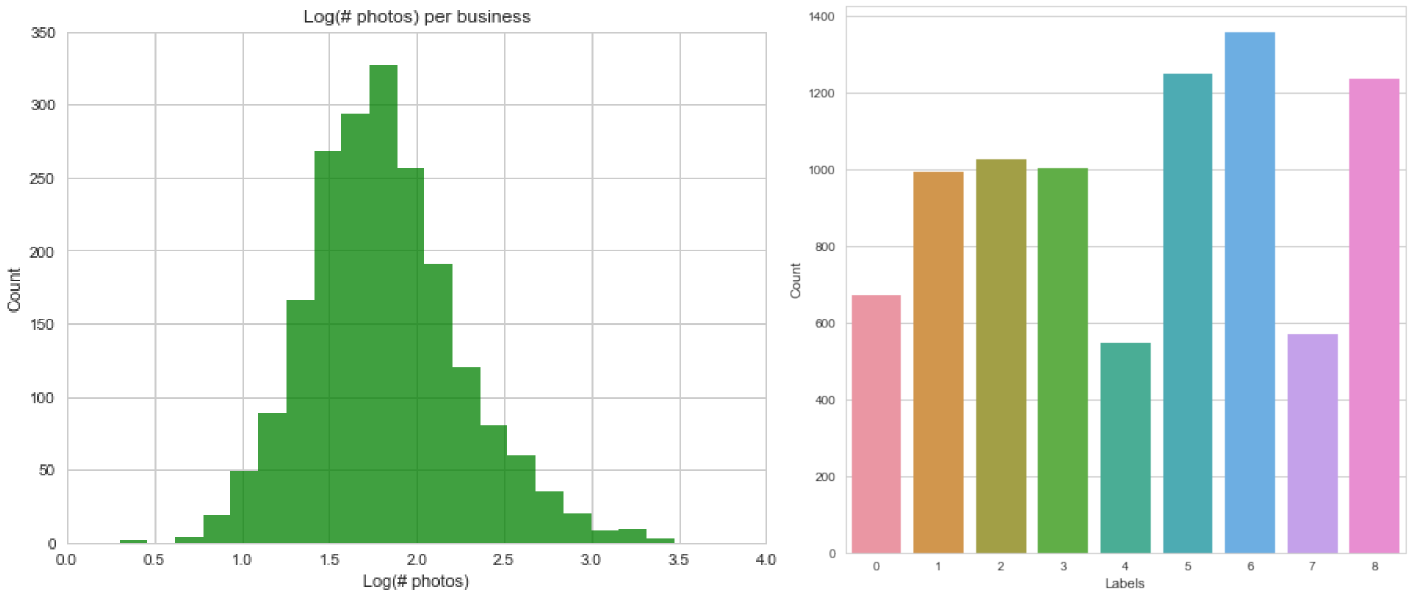

Figure 1: Training dataset statistics. Left: histogram of the number of photos per business. Right: bar plot of the label frequency. These figures are extracted from the `eda.ipynb` Jupyter notebook.

per business for the training dataset is shown in the left panel of Figure 1. Some businesses have very few photos (as low as 2 photos) whereas other have thousands of photos. There is on average 117 photographs per business. The label frequency is shown in the right panel of Figure 1. The least used attributes are `good_for_lunch` (#0), `restaurant_is_expensive` (#4) and `ambience_is_classy` (#7) whereas the most used attributes are `has_alcohol` (#5), `has_table_service` (#6) and `good_for_kids` (#8).

Exploratory Visualization

Figure 2 shows 24 randomly chosen photos that has been tagged as `good_for_kids` (#8). One can see that these photos provide rich local business information. Some photos capture the ambiance/decor of a place whereas other exhibit the food and drinks that are served. Teaching a computer to understand the context of these photos is clearly not an easy task. Objects can easily be misinterpreted and

[1] Visit `https://www.kaggle.com/c/yelp-restaurant-photo-classification/data`.
[2] The datasets are quite large. Both the training and test tar archive files have a size of about 7 GB.

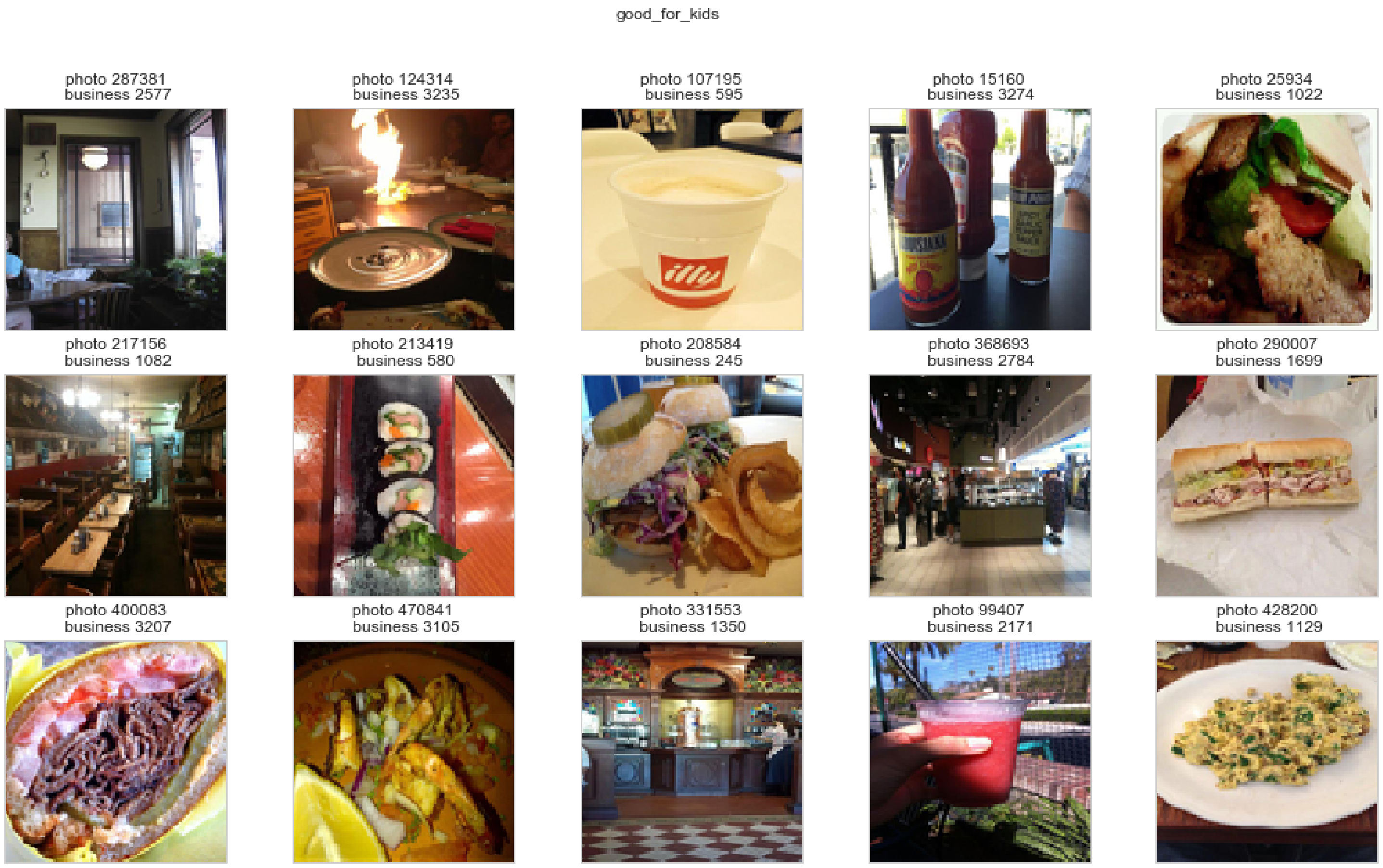
good_for_kids
photo 287381 business 2577
photo 124314 business 3235
photo 107195 business 595
photo 15160 business 3274
photo 25934 business 1022
photo 217156 business 1082
photo 213419 business 580
photo 208584 business 245
photo 368693 business 2784
photo 290007 business 1699
photo 400083 business 3207
photo 470841 business 3105
photo 331553 business 1350
photo 99407 business 2171
photo 428200 business 1129

Figure 2: Photos of the training dataset. Each business has been tagged as `good_for_kids` (#8). These images are extracted from the `eda.ipynb` Jupyter notebook.

subsequently miclassified. For example, condiment bottles that can be seen on photo #15160 (1st row and 3rd column) could easily be interpreted as alcohol bottles by a model. Also, it is worth noting that labels are annotated by Yelp users and hence are subjective. To illustrate, some people think that Japanese cuisine is appropriate for kids while other might not. It follows that an establishment that serves sushis (2nd row and 2nd column), for instance, will likely not be labeled as `good_for_kids` (#8) by every Yelp users who have eaten there and submitted photographs.

Algorithms and Techniques

Neural networks have proven to be incredibly efficient at classifying images and often outperform other machine learning algorithms at this task. It comes then as no surprise that deep learning models are used extensively in this project. One now faces two options: i) build and train a deep neural network from scratch or ii) use transfer learning[3]. The properties of the dataset such as its size and nature usually dictate the type of approach to adopt. The Yelp dataset being both large and complex, it would be unrealistic to train a deep neural network model from scratch given this task would require fine expertise and enormous resources. Also, deep neural networks that have been pre-trained on large and diverse datasets like ImageNet[4] capture universal features in its early layers that are relevant and useful for most computer vision problems. Thus, leveraging such features allows to reach a better accuracy than any method that would rely only on the available data. For those reasons, transfer learning is a better approach for this project

[3]Machine learning technique where a model trained on one task is re-purposed on a second related task.
[4]Large visual database designed for use in visual object recognition software research.

The next step entails the selection of the most relevant pre-trained model for the problem domain. Four state-of-the-art deep learning models whose weights have been pre-trained on the ImageNet database are considered here. For each model, the bottleneck features[5] are extracted and used as inputs of a very simple classifier. Each classifier is independently trained and their performance is evaluated on an unseen set of features. Based on these results, the best pre-trained deep learning model is selected and used as a fixed features extractor.

Before feeding the bottleneck features to a classifier, one needs to address the multi-instance aspect of this project. There are essentially two options: i) derive a feature vector for each instance and combine them accordingly to get one feature vector per restaurant or ii) assign to each instance the label of its corresponding restaurant, proceed to classification and average the output probabilities of each label in order to obtain one set of nine probabilities per business. Both scenarios are investigated in this project.

Finally, a classifier is trained and predictions are made. Two models are considered for the classification task: i) a multi-layer neural network with a final layer containing one node for each label and ii) gradient boosted decision trees. It is worth noting that a neural network automatically accounts for eventual dependencies among labels because it shares weights for the different label learning tasks. For the other model, label dependencies are handled through classifier chains.

Benchmark

As mentioned above, the evaluation metric for this Kaggle competition is the F_1 score. Participants ranking works as follow: predictions are uploaded to the Kaggle platform, the F_1 score is calculated for each submission and, subsequently, a leaderboard is generated. There are two different leaderboards on Kaggle. A public leaderboard that is calculated using approximately 30% of the test data and a private leaderboard that is based on the other 70%. As long as the competition is ongoing, participants can upload their predictions to Kaggle and consult their standing on the public leaderboard. Once the competition has completed, the private leaderboard reflects the standing. Note that having two leaderboards ensures that participants do not tune their model with the sole purpose of improving their ranking on the public leaderboard. Machine learning is all about building models that generalize well and this property is addressed via the private leaderboard.

As soon as the competition ends both leaderboards are accessible. The performance achieved by other models and reported on the private leaderboard[6] is used as a benchmark in this project. There are 355 participants in this competition and the 144 best performing models all achieve $F_1 > 75\%$. Note that the best model obtains $F_1 = 83.177\%$.

III. Methodology

Data Preprocessing

Images of the training (test) dataset do not have the same size (height and/or width, scaling) and need to be reshaped/rescaled before being fed to the pre-trained deep learning models. The subtlety is that not all models use the same convention. To illustrate, some models use images with pixel values ranging from 0 to 1 whereas others expect pixel values to be centered but not normalized. That said, there is no need to worry about because Keras, the deep learning framework used for this project, provides a `preprocess_input` function that transform the image to the format required by each model. Apart from that, the data are tidy and require no other preprocessing.

[5] Last feature map before the fully connected layer.

[6] See `https://www.kaggle.com/c/yelp-restaurant-photo-classification/leaderboard`

Before starting the analysis, the training dataset has been split in three dataset: a training, a validation and a test dataset. The models developed in this project are trained on 176,131 images (75% of the original training dataset), validated on 29,355 images (12.5% of the original training dataset) and tested on 29,356 images (12.5% of the original training dataset). These datasets are produced in the `split_data.ipynb` Jupyter notebook.

Implementation

There are two distinct parts in the analysis. The first part consists in selecting the most relevant pre-trained deep learning model for the problem domain. The second part entails developing a model that predicts attribute labels for each business. The various steps that are undertaken to achieve these goals are thoroughly presented below.

Fixed Features Extractor

As discussed earlier, transfer learning is undoubtedly the best approach for this project. Several deep learning models for image classification with weights trained on ImageNet are made available with Keras. Four models are considered here: VGG16[7], Xception[8], ResNet50[9] and InceptionV3[10]. All have achieved excellent results in the ILSRVC[11]. To illustrate, the Xception model achieves 79% (top-1) and 94.5% (top-5) accuracy on the ImageNet validation dataset. It is common practice today to use these models for feature extraction. Only a small fraction of the datasets is used to select the best fixed feature extractor for this competition. The training, validation and test datasets are now made of 20,000, 2,000 and 2,000 image-target pairs, respectively. Note that each image in these datasets inherit the labels (9-tuple of 0 or 1) from the parent business.

The VGG16, Xception, ResNet50 and InceptionV3 bottleneck features are extracted for the reduced training, validation and test datasets. Images in each dataset are first preprocessed and stacked as 4D-arrays or tensors: `(#images,#rows,#columns,#channels)`, where `#images` is the number of images that is fed to the model, `#rows` (`#columns`) is the width (height) of the images in pixel unit and `#channels` is the depth of the images (RGB = 3, grayscale = 1). As noted previously, the image preprocessing is handled by Keras and depends on each model. The tensors associated to each dataset are then fed to the pre-trained deep learning models and the feature maps located before the last fully connected layer, the bottleneck features, are extracted[12] and saved on disk. Note that the shape of the bottleneck features is specific to each model since the penultimate layer is different from one model to the other. To illustrate, bottleneck features calculated with VGG16 have shape `(7,7,512)` whereas those extracted with ResNet50 have `(1,1,2048)`. The various bottleneck features are calculated in the `bottleneckFeaturesExtraction.ipynb` Jupyter notebook.

An extremely naive neural network is used for classification. A pooling layer spatially reduces the input feature maps which are then flatten using a fully connected layer. The latter contains 9 nodes, one for each label, and is equipped with a `sigmoid` activation function to output probabilities. This classifier is trained on the VGG16, Xception, ResNet50 and InceptionV3 training bottleneck features independently. The loss function is set to `binary_crossentropy` and a stochastic gradient descent (`SGD`) optimizer with a learning rate of 10^{-4} is used to minimize the weights of the neural networks. At each epoch, when a new set of weights is derived, the loss function is calculated for the corresponding

[7]See `https://arxiv.org/abs/1409.1556`.

[8]See `https://arxiv.org/abs/1610.02357`.

[9]See `https://arxiv.org/abs/1512.03385`.

[10]See `https://arxiv.org/abs/1512.00567`.

[11]ImageNet Large Scale Visual Recognition Competition: `http://www.image-net.org/challenges/LSVRC`.

[12]The top layer is easily removable with Keras

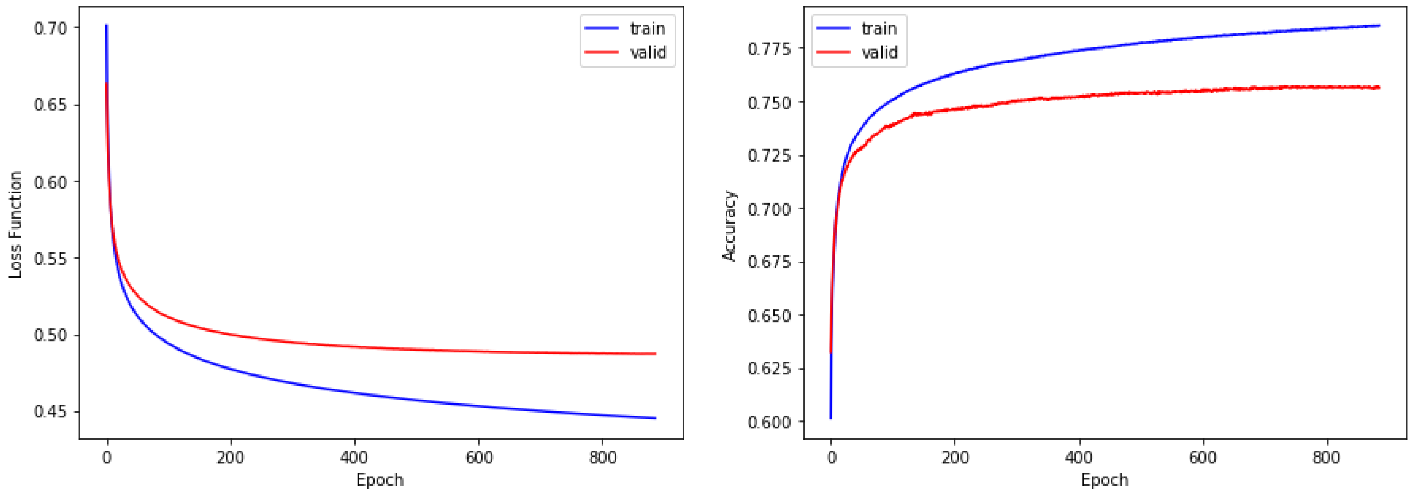

Figure 3: Loss function (left) and accuracy (right) across epoch for the ResNet50 training (blue) and validation (red) bottleneck features. These figures are extracted from the `pretrainedModelsComparison.ipynb` Jupyter notebook. Similar plots were produced for the neural networks trained and validated with the VGG16, Xception and InceptionV3.

validation bottleneck features. To avoid overfitting, an `EarlyStopping` function is implemented and the training is stopped if the loss function calculated on the validation dataset, the `val_loss`, does not improve after 20 consecutive epochs. The weights of the best model are automatically saved on disk. This procedure is carried out in the `pretrainedModelsComparison.ipynb` Jupyter notebook. Figure 3 shows the value of the loss function (left) and the accuracy metric (right) across epoch for the ResNet50 training (blue) and validation (red) bottleneck features. Once the training is completed, the weights of the best performing model, i.e., the one that minimizes the loss function, are loaded and predictions are made on the test bottleneck features. The output probabilities as returned by the fully connected layer are rounded and the F_1 score is calculated using the `f1_score` function implemented in the `common.py` file[13]. Note that the predictions are made at the instance level. In other words, neither the training, validation and test bottleneck features or the output probabilities were averaged in order to only get one feature map or one set of predictions per restaurant. The goal of this analysis is to test the ability of the VGG16, Xception, ResNet50 and InceptionV3 deep learning models to provide relevant features for this competition and hence at this stage of the project, there is no need to worry about calculating the F_1 score at the business level.

The F_1 scores calculated at the instance level on the test datasets are **0.75284**, **0.74756**, **0.76744** and **0.74086** for VGG16, Xception, ResNet50 and InceptionV3 bottleneck features, respectively. It seems that all four deep learning models can safely be used as fixed features extractor. The ResNet50 model is picked for two reasons: i) it creates features that yield to a better result and ii) ResNet50 is the model that produces the lowest number of features. In turn the training time of the classifier will be reduced. The ResNet50 bottleneck features of the full training, validation and test datasets are calculated using the `bottleneckFeaturesExtraction_resnet50.py` script.

First Classifier: Neural Network

The first classifier developed for this project is is a multi-layer neural network. Its architecture is as follows. A pooling layer first down-samples the ResNet50 bottleneck features. Then, three fully connected layers are used in order to create a feature hierarchy while increasing feature learnability. These layers have 1024, 512 and 256 nodes and are equipped with a `relu` activation function.

[13]This file encloses all the functions that are used throughout the various notebooks.

In order to prevent overfitting, L_2 regularization is used. This method, also called *weight decay*, aims to decrease the complexity of the model by penalizing weights that have large magnitudes. This is done by adding to the loss function multiple of the sum of the square of the weights. The hyperparameter λ controls the degree of penalization and hence needs to be properly chosen. For too low values, the effect of the regularizer will be negligible whereas for too high values, the optimal model will set all the weights to zero. Taking this into account, λ is set to 10^{-4} for this project.

Besides L_2 regularization, a couple of dropout layers are used to further keep overfitting in check. Both have a probability of 0.3 and are inserted between the fully connected layers. It follows that only 30% of the nodes will be randomly activated in the first and second fully connected layers at each epoch. The network is then forced to learn the same concept with different neurons and, as a result, the model will generalize better. Of course, the fraction of units to drop needs to be chosen properly.

Another technique used in this model is batch normalization. As the signal propagates through the network, it may well end up completely skewed in both mean and variance properties in the hidden layers even if the inputs have been properly normalized. This effect is called *internal covariance shift* and potentially leads to severe discrepancies between gradient updates across layers. Batch normalization layers solve this issue by normalizing the activations of the previous layer to zero mean and unit variance at each batch. It is worth noting that batch normalization also acts as a great regularizer due to the fact that the output of the network for a single entry is no longer deterministic since it depends on the entire batch within which it is contained. As a result, the model will generalize better. A batch normalization layer is inserted following fully connected layers.

Finally, a fully connected layer outputs probabilities. It has 9 units, i.e., one for each label, and is equipped with a `sigmoid` function. The loss function is set to `binary_crossentropy` and a stochastic gradient descent (SGD) optimizer with a learning rate of 10^{-4} is used to minimize the weights of the network. The model is trained/validated/tested on the 176,131/29,355/29,356 ResNet50 bottleneck features extracted previously in `bottleneckFeaturesExtraction_resnet50.py`. An `EarlyStopping` function is implemented and training is stopped if `val_loss` does not improve after 5 consecutive epochs. Also, the weights of the best model are automatically saved on disk. Each time `val_loss` reaches a new minimum, the file enclosing the weights of the best model on disk is updated. After training this file is loaded and predictions are made on the test ResNet50 bottleneck features. Note that the predictions are made at the instance level. The output probabilities of each label are then averaged to obtain one set of nine probabilities per restaurant. These values are then rounded and the F_1 score is derived. This classifier is trained, validated and tested in the `nn.ipynb` Jupyter notebook.

Second Classifier: Boosted Trees

Boosted trees is the second classifier employed in this project. The general idea is to compute a sequence of simple trees (weak learners), where each successive tree is built for the prediction residuals of the preceding tree. It can be shown that this additive expansion of simple trees can eventually produce an excellent fit of the predicted values to the observed values, even if the specific nature of the relationships between the predictor variables and the dependent variable of interest is very complex. The gradient boosting framework used in this project is XGBoost[14] software library. It has been developed with both deep consideration in terms of systems optimization and principles in machine learning.

As noticed at many occasions in this document, multiple photographs can belong to the same restaurant (multi-instance aspect of the project). This time, a single feature vector is derived for each business by averaging the ResNet50 bottleneck features for the training, validation and test datasets. Note that there is no guarantee that the training, validation and test datasets enclose all 2,000 businesses. Indeed,

[14]Short for extreme gradient boosting. See `https://github.com/dmlc/xgboost`.

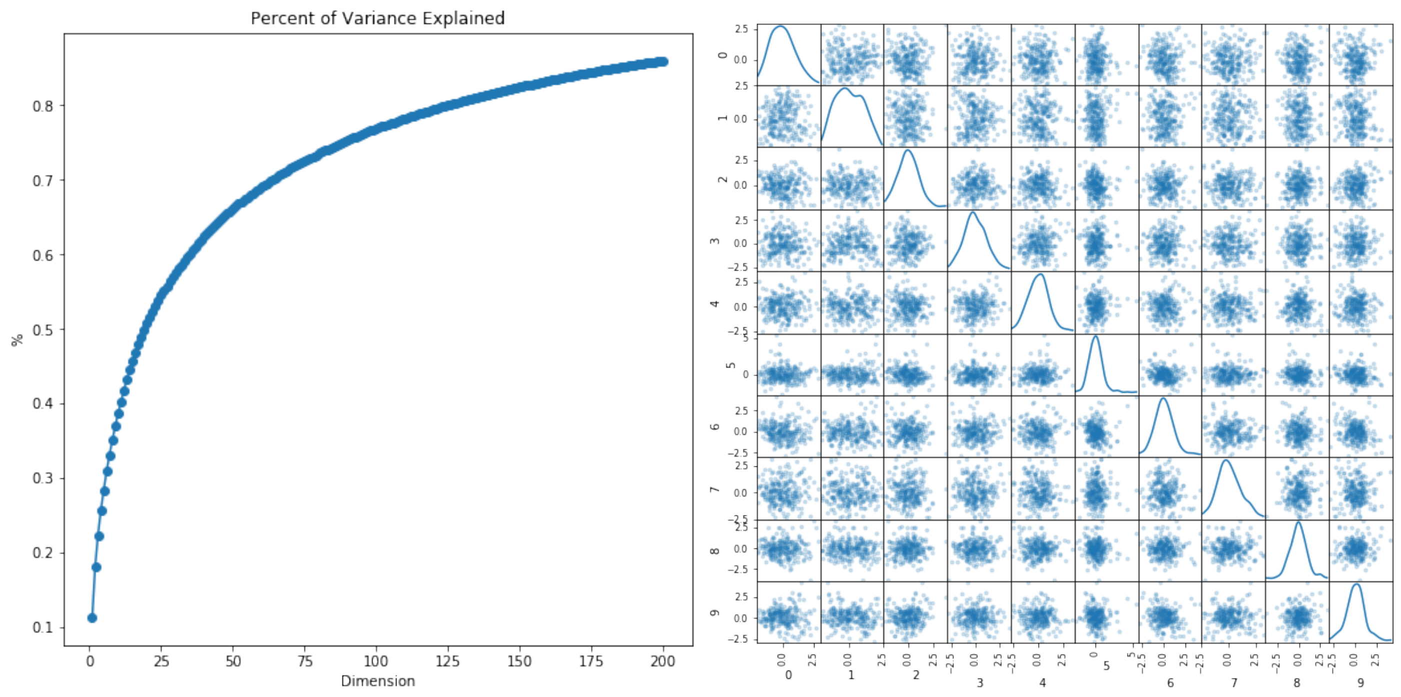

Figure 4: Principal component analysis. Left: Amount of variance explained. This is a cumulative representation. Right: Correlations between the first ten principal components. Both the scatter plots (off-diagonal) and the distribution (diagonal) are shown. These figures are extracted from the `xgboost.ipynb` Jupyter notebook.

these datasets originate from a split of original dataset that have 117 photographs on average for each 2,000 restaurants. The shape of the training, validation and test datasets after the averaging operation are: `(2000,2048)`, `(1946,2048)`, `(1953,2048)`. These features are then standardized by removing the mean and scaling to unit variance. The scaler is first derived on the training data and then applied on the training, validation and test data. Besides, a principal component analysis (PCA) is carried out in order reduce the number of features by an order of magnitude. The plot in the left panel of Figure 4 shows that the first 200 principal components capture about 85% of the variance in the training data. The pair-wise scatter plots of the the first 10 principal components are shown in the right panel of Figure 4. One can see that these variables are uncorrelated as expected by construction. The PCA transformation is applied to the validation and test data before proceeding to classification.

Boosted decision trees are used for classification. The following parameters have been considered for the algorithm[15]: i) learning rate, $\eta = 0.01$; ii) the maximum depth of a tree. An increase in this value makes the model more complex. Here, `max_depth` = 10; iii) `subsample` is set to 0.5 so that XGBoost randomly collects only half of the data instances to grow trees and, thus, prevent overfitting; iv) the fraction of features to consider to be randomly sampled when looking for the best split in a tree. Here `colsample_bytree` = 0.3. The objective function and the evaluation metric are set to v) the logistic regression for binary classification, `objective` = `binary:logistic`, to output probabilities and vi) the negative log-likelihood, `eval_metric` = `logloss`, respectively.

First, potential correlations among labels are ignored and a prediction model is produced for each of the 9 labels. For each label, a maximum of `num_round` = 2000 decision trees with the above parameters are considered and trained. The minimization is controlled by the validation data and the training is stopped when either the number of trees in the sequence reaches 2000 or `logloss` does not improve after adding 10 new trees to the existing sequence of trees in the model. After training, predictions are

[15]These parameters are specific to the XGBoost software library. The list of parameters can be found at the following url: `http://xgboost.readthedocs.io/en/latest/parameter.html`.

made on the test data. The probabilities are averaged at the label level and then rounded in order to obtain a set of 9 integers comprised of 0 or 1 for each business. The F_1 score can then be calculated. This analysis is carried out in the `xgboost.ipynb` Jupyter notebook.

To exploit the potential correlations among labels, eight classifier chains[16] are created. Each classifier chain contains a boosted trees model for each of the 9 labels. The models in each chain are ordered randomly. In addition to the 200 features in the dataset, each model gets the predictions of the preceding models in the chain as features. These additional inputs allow each chain to exploit correlations among the classes and the F_1 score for each chain tends to be greater than the one calculated in the independent case. Presumably there is an optimal ordering of the classes in a chain that will yield the best performance. However, that ordering is not known in advance. For this reason, an ensemble of classifier chains averaging the binary predictions of the chains is constructed. Each of the 9 probabilities are then rounded and the F_1 score is calculated. Note that this score will be greater than the one calculated with independent models and is likely to exceed the score of each chain in the ensemble. Classification using ensemble of classifier chains is undertaken in `xgboost.ipynb`.

Refinement

The value of the threshold used for rounding the output probabilities – as returned by the neural network or the boosted trees – plays a significant role in this analysis. Setting the threshold to 0.5 is probably a good option. However, there is no evidence that this value produces a set of 9 labels, which in turn maximizes the F_1 score. In fact, it is very likely that each label has its own optimal threshold.

Matthews correlation coefficients (MCC) are used to determine the best set of thresholds. The MCC is a correlation coefficient between the observed and predicted binary classifications. It returns a value between -1 and $+1$. A coefficient of $+1$ indicates a perfect prediction, 0 no better than random whereas -1 indicates a total disagreement between prediction and observation. The MCC takes into account true and false positives and negatives.

A simple experiment is carried out in `findBestThreshold.ipynb`. The training dataset is divided in 14 equal sets. For each set, predictions are made using the multi-layer neural network developed in `nn.ipynb` and averaged to obtain a single set of 9 probabilities per business. First, a unique threshold is used to round all 9 probabilities. Values ranging from 0.4 to 0.6 are considered in this experiment. The F_1 score is calculated for each dataset and threshold combination. A box plot is used to interpret the results (see `findBestThreshold.ipynb`). It is found that a threshold of about **0.5** maximizes the F_1 score. The case of several thresholds, i.e., one for each output probability has also been tested. Using the same range of threshold values, the best combination of thresholds is obtained for each sample through the calculation of MCC. Finally, the median values give the overall best set of thresholds. Those are: **(0.43, 0.55, 0.535, 0.525, 0.545, 0.54, 0.55, 0.47, 0.5)**.

IV. Results

Model Evaluation and Validation

Two models have been considered to classify the bottleneck features extracted from the ResNet50 pre-trained deep learning model: a multi layer neural network and boosted trees. The former (latter) classifier is trained, validated and tested on 176,131 (2,000), 29,355 (1,946) and 29,356 (1,953) bottleneck features, respectively. Figure 5 depicts the training phase of the neural network. The value of the loss function (left panel) and the accuracy metric (right panel) across epoch is shown for the training (blue)

[16]See `https://en.wikipedia.org/wiki/Classifier_chains`.

and validation (red) datasets. One can see that overfitting is kept in check. The series of dropout and batch normalization layers as well as the the L_2 regularizers implemented throughout the network do an excellent job here. The best set of weights are loaded and the F_1 score is calculated on the test dataset: $F_1 = 0.82807$ using a unique threshold of 0.5 and $F_1 = 0.83160$ for the tailored set of thresholds. Two results are reported in `xgboost.ipynb` for the boosted trees. For the case where a unique threshold of 0.5 is utilized and the potential dependencies among labels are ignored: $F_1 = 0.76503$. When applying the classifier chains rule: $F_1 = 0.78695$. Figure 6 illustrates these results. One can see that the F_1

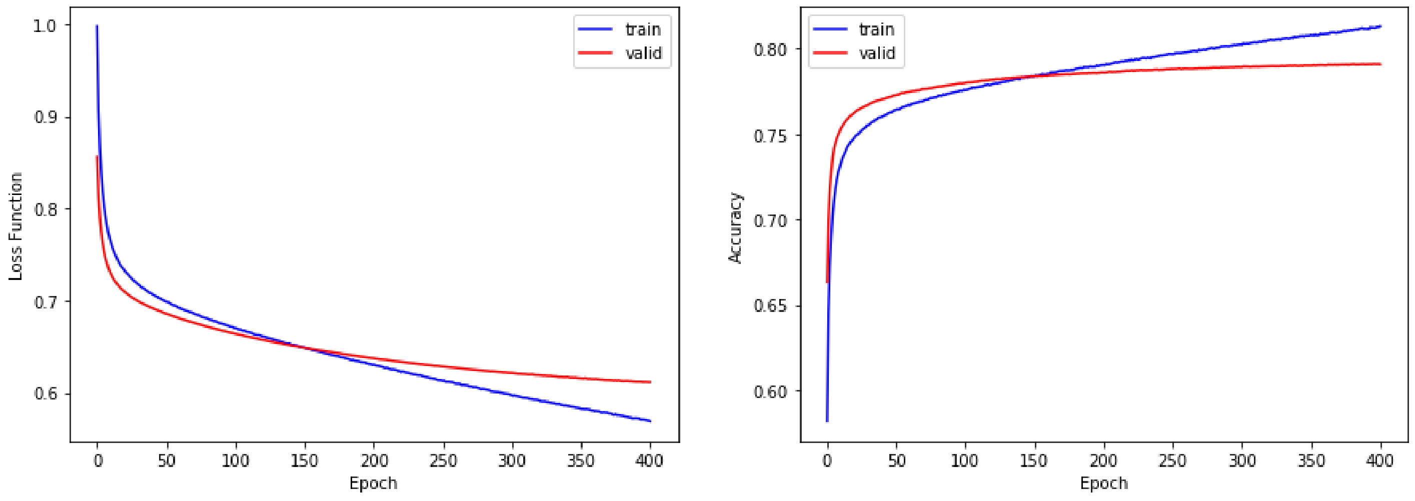

Figure 5: Training of the multi-layer neural network. Loss function (left) and accuracy across epoch (right) for the training (blue) and validation (red) datasets These figures are extracted from the `nn.ipynb` Jupyter notebook.

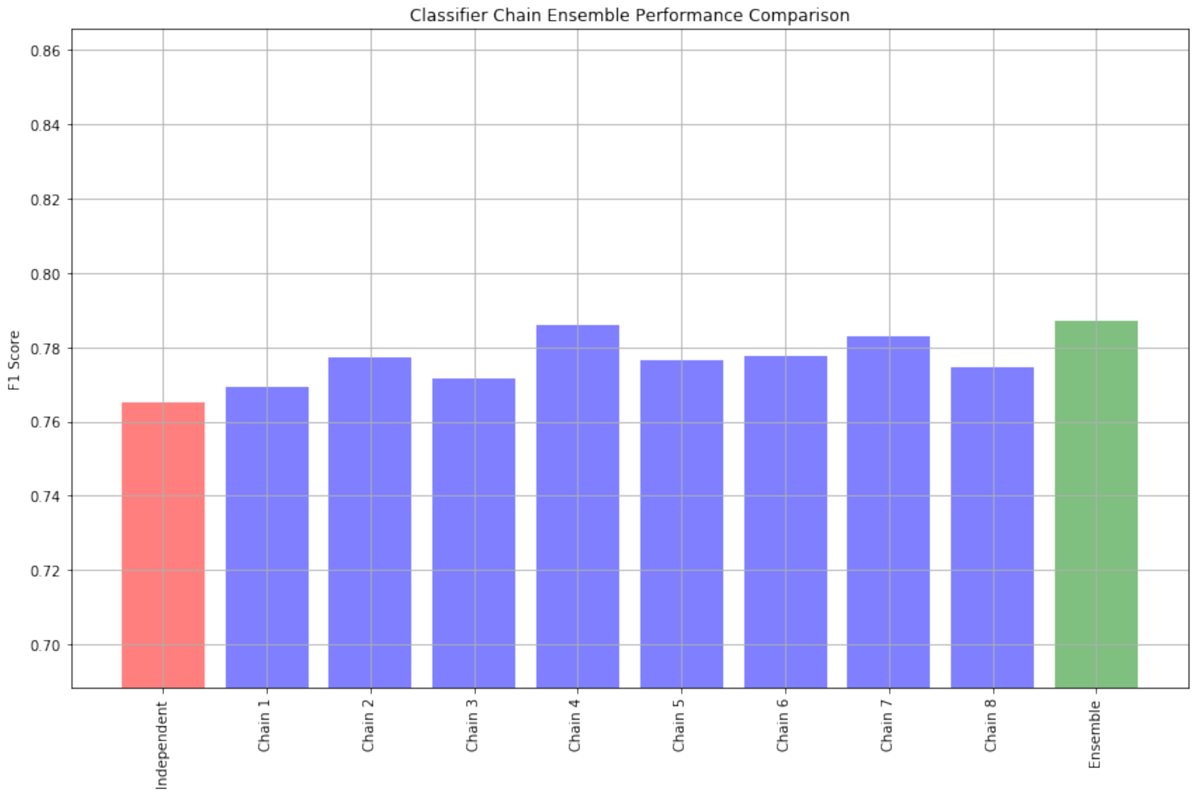

Figure 6: Classifier chain ensemble performance prediction. This figure is extracted from the `xgboost.ipynb` Jupyter notebook.

score calculated.with the ensemble of classifier (green bar) exceeds the score derived for the independent models (red bar).

V. Conclusion