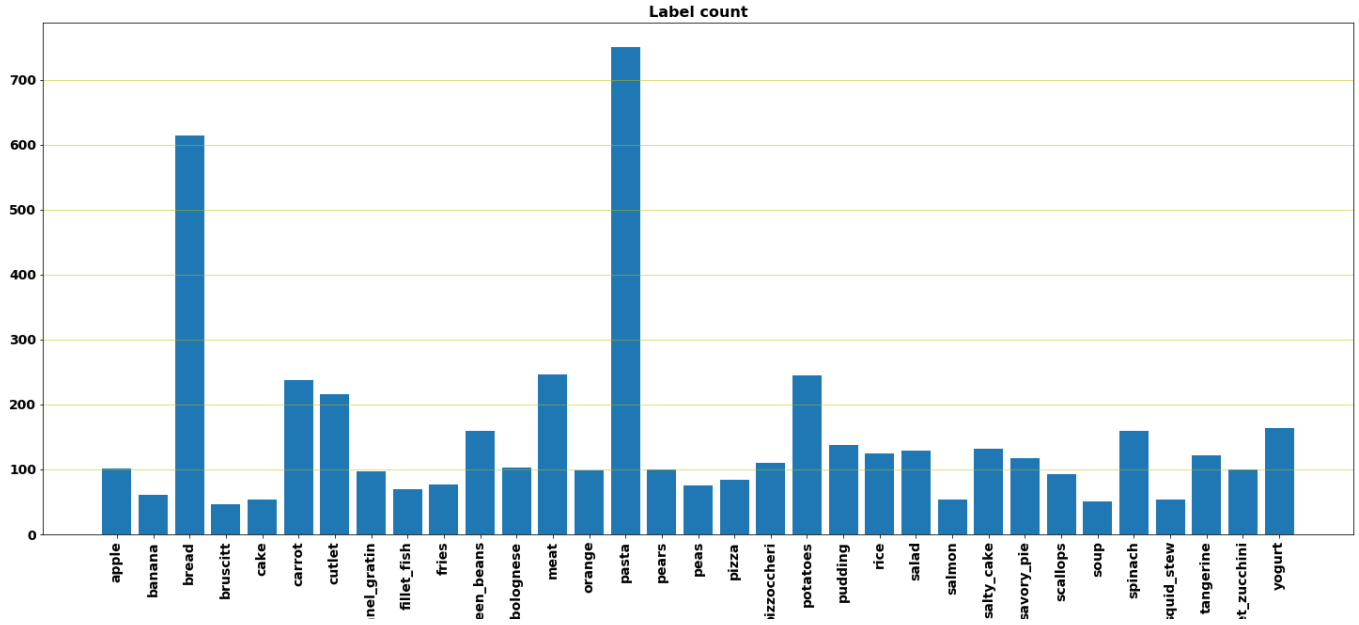
Exploratory Data Analysis

There are a total of 1400 samples and the task is of multi-label classification, with 33 unique classifications in total.



Above barplot shows the total counts of each label in the dataset. The data is pretty much balanced but two classes, bread and pasta seem to have a very high frequency of occurence. Special care needs to be taken while training on skewed data.

All images have 224x224 pixels with 3 color channels. Also there are no missing values. Given the task, the pixel values can be directly used as input features without applying any major image processing operations.

Each image has 150528 features. So, using a vanilla neural network is out of the question as the number of weights in the networkwill be staggering. A convolutional neural network will be an ideal choice. All the feature values will be scaled between 0 and 1 so that the compuations are faster.

To obtain the target features, the labels of each image will be one-hot encoded. Each target array will have 33 values, corresponding to each of the food items. Presence of a food item (independent of its quantity) will translate to 1 in the target array and the absent food items will translate to zero.

Convolutional Neural Network Architecture

For a multi-class classification problem, each image can be classified as only one of the many classes. Generally, in such a case, we have as many output neurons as the number of classes, and a softmax activation function which converts final scores into probabilities, so that all the final values sum up to 1. The neuron with the highest probability is taken as the class predicted by the model.

In a mutli-label classification probelem, each image can have more than one label. So, we can't use a softmax activation function. Instead we use sigmoid activations at the output layer, so that each neuron can output a value between 0 and 1. We then have to keep a threshold value above which the final value is taken as 1 and values below it are taken as 0. For the purpose of training, we can use 'binary cross-entropy' as both the target values and the predicted values are binary in nature.

The convoluational neural network will consist of both convoltuional layers and fully connected layers. As the sample size is very small, and with 30% of it with-held as validation and test sets, the number of sample we actually have available for training is small.

And with 33 labels in total with such a small number of training samples, to obtain a useful model, the choice of hyper-parameters will be very important. The first obvious assertion is that we would require many convolutions stacked after one another, so that we can capture enough patterns to be able to get the desired results. The risks of vanishing gradients and overfitting increase as we go too deep. For the second problem, we will use Batch Normalization before each convolutional layer, and a dropout of 0.5 after every fully connected hidden layer. For the first problem, we will use the ReLU activation function after every layer, except for the output layer.

First, convolutions with filters of varying sizes is applied and the resulting convolved layers are all stacked together. Zero padding is for each convolution so that the convolutions with different filter sizes lead to ouput layers that have the same dimensions as the original dimensions of each input channel. This is done so that all the output layers can be stacked together. The idea is that the small filter convolutions (3x3 and 5x5) will help in detecting food items with fine texture like pasta, soup, pasta etc, and the bigger filter convolutions (7x7 and 11x11) will help detect larger whole food items like apples, pears, pizza etc.

The remaining deep convolution layers all have filter sizes of 3x3 but we kreep increasing the number of filters as we go deep. Maxpooling is also applied so that the feature size gets smaller and smaller and only the relevant features get filtered ahead, before we can put in a fully connected layer. After multiple convolutions and poolinsgs, we are left with 1600 features that we feed to a fully connected layer. There is just one hidden layer before the output layer of size 33, after which sigmoid activation is applied.

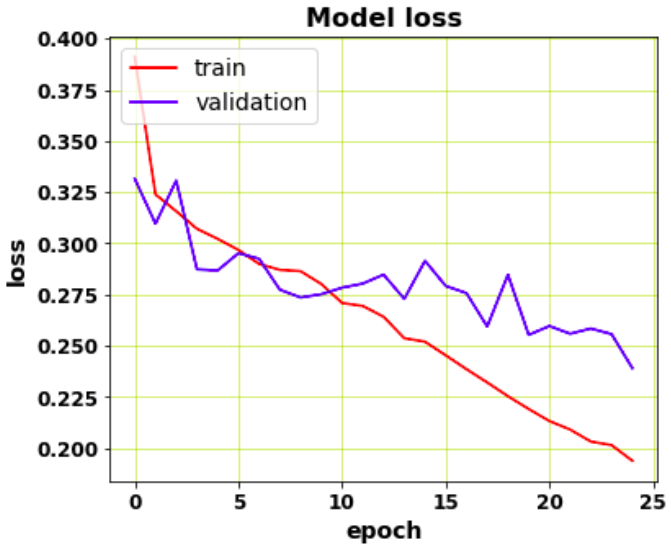
The complete network can be viewed in the model.png image.

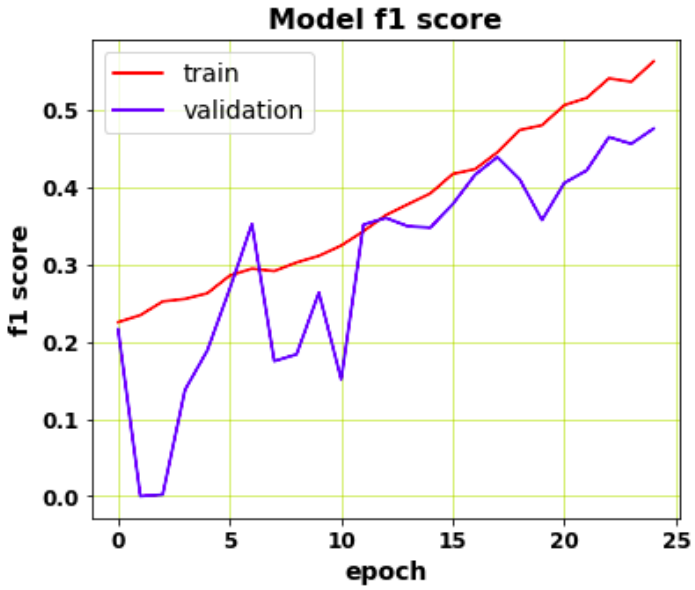
Training

As the samples are few, we can keep a small batch size and avoid getting stuck in some local minima. A large batch size does reduce the training time but that's not a concern here. With a smaller batch size, it's usually better to keep a small learning rate and run for a good number of epochs.

Other than validation loss, f1-score is another metric that is used to monitor the training process. As it is the harmonic mean of recall and precision, it's a good metric to measure performance on a multi-label classification problem.

Following figures show the comparison of loss and f1-scores on the training and validation sets.





The performance on the validation set slowly gets better and catches up with the performance on the training set at the 10th epoch. The performance keeps getting better on the validation set even after that although it remains slightly less than that on the training set. This indicates that the model hasn't overfit yet. We keep training for more epochs since we have set the model callbacks to save the weights based on performance on the validation set.

After the 20th epoch the gap begins to widen, and even though we have set tight regularization constraints with batch normaization and dropouts, the risk of overfitting keeps increasing as we keep training for more and more epochs. The performance is the best at the 25th epoch although the rate of improvement over the validation set appears to be stalling.

Interpreting the Results

To properly assess our trained neural network, we use it on the test set (15% of the total available data).

The predictions on the test input features are values between 0 and 1. To convert them to neat binary values, we loop over a range of thresholds that we can apply. Then we evaluate the f1 scores on all the thresholded predictions and choose the threshold that gave the best f1 score.

We obtain the best f1 score of 0.47 when we keep the threshold of 0.37. After thresholding, we convert the one-hot encoded arrays back to lists of labels, and compare with the true values. Let's pick 3 samples at random from the predictions made on the test set.

actual_labels: ['pizzoccheri'], classified_labels: ['pizzoccheri']

Just one food item to classify, and the model does it well this time.

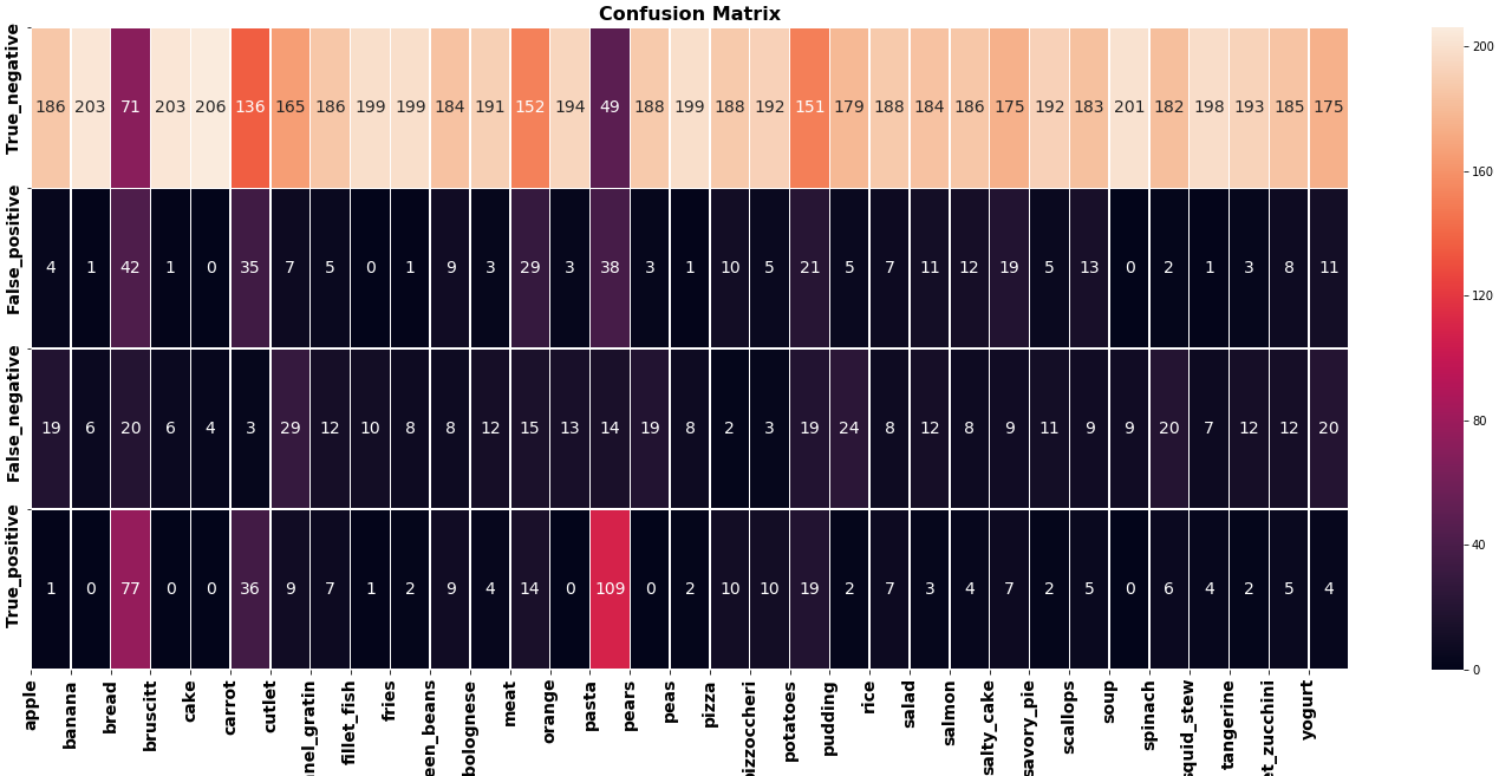
actual_labels: ['apple' 'bread' 'carrot' 'pasta'], classified_labels: ['bread' 'carrot' 'meat' 'pasta' 'yogurt']

The model has detected bread and pasta correctly. Not surprising since they are the two most frequent labels that it has been trained on. However, it probably misclassifies carrot as meat, and the pickle packet as yogurt. It makes sense since the yougurt it has seen comes in packets similar to this one but pickle is not included in any of the food labels. It fails to detect the apple too. However, in multi-label classification problems, we can not distinguish cases of mis-classification and failure of detection with certainity.

actual_labels: ['bread' 'cutlet' 'pasta' 'tangerine'], classified_labels: ['bread' 'green_beans' 'meat' 'pasta' 'potatoes']

Again, accurate detection of bread and pasta. The cutlet is mis-classified as potatoes, probably owing to similar texture and color, and similar sizes The tangerine is missclassifed as meat.

To get a clear picture, let's look at the confusion matrix heat map.



As expected, True positives for bread and pasta are high, and the false negatives are very few. But, they also have the highest False Positives. The skewness in the training data on their side has spilled over considerably into the trained weights.

Bruscitt had the lowest overall count in the data. It also reflects in the results on the test set. The model failed to detect all of its six occurences in the test set. On the plus side, it only made one false positve out of the remaining 200 samples in which it was actually absent. Same is the case with other rare items like soup and stew. The model may have had few training samples to learn to detect these food items, but it stiil doesn't make too many false detections or misclassify other food items as them.

The model seems to do well on items like Pizza and Pizzoccheri where the false negatives and false positives are very few compared to the true positives and true negatives. It's probably because the model had enough of their true instances to train on and more importanlty, they are distinct enough and large enough for the model to mis-classify them or miss them.

For items like apples, oranges, and pears, the numbers of false negatives are very high and true postives are very very low. This suggests that the model hasn't learnt to detect them at all. This is because their counts were low to begin with, and their small size coupled with the fact that they mostly appear on plates as side items with many other items also present made it difficult for the model to learn them.

For items like cutlet and potatoes, even though their counts were probably enough for the model to learn to detect them, as evident form a good number of True Positives, they still have a high number of false positives and false negatives. This suggests that the model hasn't learnt well enough to distinguish correctly between items of comparable sizes and colors. This seems to be true for many other items like meat and carrot as well.

The precision, recall, and f1 scores of individual food items are summed up in statistics.csv file.

Observations

The trained model seems to work well on food items that are present in a large proportion of training samples.

It also does well on the items that were present in a good enough number of samples given the items were very distinct from the other items.

The model fails to learn about items that are small in size and are usually present with other large items on the plate.

For many items, the true postives are very few compared to the false negatives. This suggests that the model has not learned to detect these food items.

The false postives are mostly few for most items. But it is high for items that are very similar and are present in similar number of instances in the training samples.

The inference that can be drawn is that due to the tight regularization constraints and proper shuffling of training data, our model did not overfit. But for a total of 33 labels, and with highly random order in which those labels are present in the samples, we require far more samples for our model to train on.

Data Augmentation techniques like rotating, flipping, shifting images can artificially generate new samples but that will only provide slight improvement. What's actually required is more instances where the rare food items are more. That can be done by cropping the images for the food items that the model is failing to detect and train on those cropped images.

For better interpretation of the model's results, we can also include the bounding points of food items as part of training, so that the model also gives pixel locations of the boundaries of objects that it is detecting. That will make it easier to interpret the model's results and henceforth take decisions to improve the model.