Construction of a Rudimentary Multi-Label Classifier
CS 325 Final Project Report

Nicholas Farkash
Department of Computer Science
CUNY Queens College
Nicholas.Farkash41@qmail.cuny.edu

Abstract

This project seeks to create a multi-label image classification model capable of detecting the presence of five distinct object classes in the PASCAL VOC2012 dataset. The model utilizes the ResNet18 architecture, a Deep Convolutional Neural Network (CNN), modifying the preexisting structure to suit the needs of the classification task. The standard machine learning pipeline (Data Acquisition, Data Cleaning and Transformation, Feature Engineering, Model Selection and Training, Model Validation) is followed to produce a model with an average precision score of 91.1%, average recall score of 87.2%, and an average F_1 score of 88.4%. The model displays a strong performance when classifying images with a single object class present. However, the model struggles to classify images where multiple classes are present. The most probable causes for this are the presence of overlapping objects and limited training data, which are discussed. Solutions to address these shortcomings are explored to suggest potential improvements to the model.

Introduction

Multi-label image classification is process that can be applied to many areas of research and learning. Many medical fields use multi-label image classification to identify and distinguish multiple diseases, and it is crucial that any models used are accurate, as a wrong prediction from the model can have serious consequences. Multi-label classification is a step beyond single label classification, as the additional task of assigning multiple labels to an image results in new challenges that are not easy to work around.

The goal of this project is create a multi-label classifier to observe its performance, identify and explore those challenges, and to propose solutions to account for those challenges. This project builds upon two preexisting frameworks to develop the classifier. The first of these is the ResNet18 model architecture, a CNN structure that will be repurposed for multi-label classification. The second is the PASCAL VOC2012 dataset, which contains images that are used to train and validate the model.

Literature Review

The task of multi-label classification is well documented, and many reports have been written exploring the task. One such publication is "HCP: A Flexible CNN Framework for Multi-label Image Classification" (Wei et al.). This publication notes the power of CNNs as being the state-of-the-art for single label classification, but discusses how these networks often struggle to perform multi-label classification. The reason for this difficulty is the "multi" part of "multi-label classification". The presence of multiple classes in a single image poses a challenge, as "partial visibility and occlusion... increase the complexity of the problem", as the presence of overlapping objects makes its difficult for the model to identify one or both of the objects. To avoid this additional complexity,

the authors take a different approach towards solving the problem, as they utilize a procedure they call Hypothesis-CNN-Pooling (HCP).

This procedure involves taking a series of object segment hypothesis images (images where only the areas that contain a particular class are highlighted), utilizing the same CNN to perform single-label classifications for each hypothesis image, and then aggregating the single-label predictions into one multi-label prediction via pooling. This avoids having to perform multi-label classification altogether, instead performing single-label classification multiple times before combining the results into a single multi-label. This procedure is shown to deliver extremely promising results, as the HCP model significantly outperforms all other models in the trial when attempting to classify each of the the 20 classes in the PASCAL VOC2012 dataset.

Another publication, "Deep Convolution Neural Network sharing for the multi-label images classification" (Coulibaly et al.), presents a similar approach to the challenge of multi-label classification. This project features a "divide and conquer" approach, decomposing a single multi-label classification into multiple single-label classifications. This is done following a framework the authors call "Multi-Branch Neural Network", or MBNN.

This procedure attempts to solve the problem of overlapping images by taking into consideration the correlation between labels. For example, if two classes frequently appear together in images, then the model learns that there is some underlying relationship between the classes. When presented with an image, if the model finds one of these classes, then it will understand that the image is more likely to contain those related classes. This reduces the likelihood of the model not identifying one class due to it being obfuscated by another, and so the model accounts for partial visibility through these relationships, resulting in an increased probability of successfully identifying all classes present in the image. The results show that the authors' "multi-features model", which combines outputs from other pre-trained CNN models, is far superior to other models, including the ResNet50 model, a relative to the ResNet18 model used in this project.

Methodology

The process of creating and training the multi-label classifier used in this project follows the standard machine learning pipeline:

Data Acquisition

The PASCAL VOC2012 dataset was downloaded from the PASCAL VOC Challenge 2012 website[3]. Under the "Development Kit" section, clicking the link in the first item, "Download the training/validation data (2GB tar file)", will begin the download of the VOC2012 Dataset. The downloaded VOC2012 dataset ZIP file is large and contains many elements that are irrelevant for the purposes of this experiment. To minimize the space taken up by the project, all irrelevant files will be deleted.

The *ImageSets/Main* and *JPEGImages* directories are the only relevant parts of the dataset. Furthermore, located within *ImageSets/Main* is a series of .txt files. All files of the form "<classname>_trainval.txt" are deleted, as these files contain a combination of training and validation data points for each class. They are not needed for this experiment. Furthermore, the files named "train.txt", "val.txt", and "trainval.txt" are also deleted. All of these files contain information that would be beneficial if the model were being trained on all 20 classes at once. Finally, to establish a naming convention for ease, the *JPEGImages* directory is renamed to *VOC2012 Images* and the adjusted *ImageSets/Main* directory is renamed to *VOC2012 Sets*.

At this point, *VOC2012 Images* contains all images in the dataset and *VOC2012 Sets* contains all .txt files of the form "<classname>_train.txt" or "<classname>_val.txt" for a total of 40 .txt files.

Data Cleaning and Transformation

There are too many images in the dataset to look through to see if there are any repeated images, but since the dataset follows a strict file naming structure, it is assumed that there are no duplicate images. The dataset does contain some null values, where the label on an image is "0". These images will not be used to train the model, so they will be removed. Images that will also be removed are images labeled as "-1". The model should not be trained on what a class is not. It should be trained on what

each is class is. Therefore, all images not labeled as "1" (labels "0" and "-1") will be removed from the dataset.

Furthermore, the dataset will be reformatted to be more easily used. Images will be organized into a "train" and a "val" directory, each of which will store the training and validation images for each class, respectively. Each .txt file in the *VOC2012 Sets* directory is looked at. If and only if the .txt file belongs to a class that the model will be trained upon, then the contents of that .txt file are parsed. Within the .txt file is a list of (image name, label) pairs. For each pair, if the label is "1", meaning that the image belongs to the class, then that image is added to that class's "training" or "validation" directory, depending on the name of the .txt file being parsed. In addition, the image is also added to a directory called "all", which stores all images used in training or validation.

The result of this process is three new directories:

- "train" - stores all images that will be used to train the model. Contains a subdirectory for each class.
- "val" - stores all images that will be used to validate the model. Contains a subdirectory for each class.
- "all" - stores all images located in "train" and "val".

Feature Engineering

The newly formatted dataset is now ready to have its contents prepared for utilization. The *PyTorch*[4] package will be very helpful in accomplishing this task.

The first step is to take each image file and assign to it a multi-class label. Each image in the "all" directory is evaluated. If the image is found within a class subdirectory in the "train" directory, then it will have a "2" appended to its label, corresponding to that class. If the image is found within a class subdirectory located in the "val" directory, a "1" will be appended to its label for that class. If the image is not found in either a class's "train" or "val" directory, then it will have a "0" appended to its label for that class. This produces a list of either "2"s and "0"s or "1"s and "0"s that define which classes are present in each image, creating a multi-label for each image. The image name and labels are stored in a tuple pair, making it easy to see which (image name, labels) pairs belong to each of the training and validation datasets. It is worth noting that there cannot be "2"s and "1"s in the same label, as the training and validation data are disjoint.

Classification models generally operate on batches of (image, labels) pairs, as the model processes multiple data points at a time. As such, the data input to the model must match the format of the model's expected input. As such, each (image name, labels) pair undergoes the following procedure:

1. The image name is used to open the corresponding image.
2. The image is transformed to match the model's preferences and converted to a *PyTorch Tensor*. As will be discussed shortly, the ResNet18 model will be used for this experiment. This model expects to receive images of dimension (256, 256) pixels, crops them to dimension (224, 224) pixels, and normalizes the image with a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225]. Information on this transformation can be found on PyTorch's page for the ResNet18 model.
3. The labels are evaluated for sorting. If any of the labels' entries contains a "2", then the data point belongs to the training dataset, as discussed above. Otherwise, it belongs to the validation dataset. These (image, labels) pairs are then saved to their respective datasets.

Once all (image, labels) pairs have been placed in the appropriate datasets, a *PyTorch DataLoader* is created for each of the training and validation datasets. The DataLoader organizes the data points into batches of 32 data points, a hyperparameter that was experimented with to provide more optimal results. After batching is complete, all of the data has been properly formatted and is ready to be given to the model.

Model Selection and Training

The model that was used in this experiment is *PyTorch's* ResNet18 model[5]. It is an 18-layer model following the ResNet architecture, placing it on the small side of models. However, it is powerful to address the multi-label classification problem faced in this project.

The ResNet18 model is imported, has its final layer stripped away, and replaced with a new final layer that will be modified for the purpose of classifying images into the five designated classes. This last layer is comprised of two components. *PyTorch's* nn.Linear() is a neural network module that takes incoming data and applies a linear transformation to it. The linear transformation added takes the data output from previous layer and fits it to exactly five features - one for each class of the model. The result is a "score" for each class that represents the model's "belief" in the presence of the associated class within the image. In addition to this, an nn.Sigmoid() module is added as well. This is the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

which provides a smooth mapping of real numbers to the interval (0, 1). This takes the model's "belief" in a class, found in each of the five linear nodes, and converts it into a probability.

To train the model, a loss function is needed to measure the performance of the model so adjustments can be made to the model's weights. The loss function used on this model is *PyTorch's BCEWithLogitsLoss()*, which is a module that assigns loss according to the formula:

$$l_n = -w_n \left[y_n \cdot log(\sigma(x_n)) + (1 - y_n) \cdot log(1 - \sigma(x_n)) \right] \tag{2}$$

where n represents the parameter, w represents the weight, and (y, x) represents a (actual label, predicted label) pair.

An optimizer is also needed to perform weight adjustment according to the loss calculation. For that, *PyTorch's Adam*[6] optimizer is used. The Adam optimizer performs a variation of gradient descent that adjusts the learning rate (hyperparameter initialized to 0.0001) as it progresses.

The process for training the model using the loss function and the optimizer is as follows. For each batch of (image, labels) pairs in the training DataLoader:

1. The optimizer's weights are reset for the loss pass.
2. The model processes the batch and produces its output.
3. Loss is calculated by comparing the predicted labels to the actual labels according to the loss function.
4. The loss is back-propagated to the weights through the backward pass.
5. The weights of the model are adjusted through the *Adam* optimizer, according to the back-propagated values.

This repeats until all batches of (image, labels) pairs in the training dataset have been processed. At that point, the model has completed its training.

Model Evaluation

The model is validated by providing the validation data as input to the model. The model outputs predictions for each batch of (image, labels) pairs and stores these outputs in a set. The predicted labels are compared to the actual labels to measure the validity of the model. Three different scores are calculated to determine the model's performance - precision, recall, and F_1 score. Each of these three scores are calculated through the use of the *sklearn.metrics*[7] package and are explained in detail in the following section's "Validation Process" section.

Experimentation

Description of the Dataset

The dataset used in this experiment was the PASCAL VOC2012 Dataset. Specifically, only the *Main* dataset is used. This *Main* dataset features 20 classes: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor. Each image in the dataset features some combination of these classes. The full dataset contains 17,125 images, which are further categorized into two sub-datasets for classification - training and validation.

The images do not come presorted. Instead, all of the .jpg images are located within a single directory, *JPEGImages*. Alongside these *JPEGImages* is another directory called *ImageSets*. Within the *ImageSets* directory is a series of .txt files. Each class has 3 .txt files associated with it; a training set, a validation set, and a training/validation set, which combines the two. Within each .txt file is a list of pairs (image name, label). The file name corresponds to the name of a .jpg file in the *JPEGImages* directory, and the label is either 1, 0, or -1, stating whether that class is (1) or is not (-1) present in the image, or has not been labeled (0).

Description of Performance Measure

Three scores are calculated to measure the performance of the model:

1. Precision (P) - Number of True Positives divided by Number of Predicted Positives (True Positives + False Positives). This score answers the question, "Out of all images that were predicted to contain the class, how many of them truly did contain the class?"

2. Recall (R) - Number of True Positives divided by Number of Actual Positives (True Positives + False Negatives). This score answers the question, "Out of all images that contained the class, how many of them were correctly predicted?".

3. F_1 Score - $\frac{2PR}{P+R}$ - A harmonic mean of Precision and Recall used to create a score that takes both measures into account.

Validation Process

In order to validate the trained model, a validation dataset is evaluated by the model. This validation dataset comes from those images whose file names and label are located within the *ImageSets* directory and of the form <classname>_train.txt.

Once this validation dataset has been created and formatted according to ResNet18's preferences (*see Methodology, Feature Engineering*), the batches composed of 32 (image, labels) pairs are given to the model. The model outputs predictions for each (image, labels) pair in the batch. The model's predictions are a series of five floating point numbers on the interval (0, 1). Each floating point prediction score corresponds to the model's belief in the presence of a particular class in the image. This means that, if the model outputs as score of "0.9842" for the class "cat", then the model is 98.42% confident that there is a cat present in the image.

Each prediction value is then compared to a threshold, a hyperparameter that was trained upon to produce more optimal results. This value was found to be 0.75. If the model's prediction for a particular class matches or exceeds the threshold, then that prediction is overwritten to be "1", meaning that the model believes that the class is present in the image. If the prediction falls short of the threshold, then the prediction is overwritten to be "0", meaning that the model believes that the class is not present in the image. This is done for each of the five classes, for each image in the validation dataset.

Once the labels have been converted to a binary form, they are compared to the actual labels, which were created in *Methodology, Feature Engineering*. The scores discussed in the previous section are then calculated according to these label comparisons and the results are displayed.

Results

To ensure consist performance in the model, 10 models were created, trained, and validated. The average score for each validation performance measure is as follows:

Table 1: Summary of Validation Results

Performance Measure	Average Score
Precision	0.861
Recall	0.822
F_1 Score	0.835

For each performance measure, no single trial's results produced a score further than 2% from its respective average. Thus, it can be concluded that the model's validation is reasonably consistent, and that the model is successful in performing multi-label classification on images over 80% of the time.

Discussion and Conclusion

Although the model produced promising results upon validation, it is important to understand what those results truly reflect. Although the model is a multi-label classifier, many of the images in the VOC2012 dataset featured only a single class. The majority of validations featured the model identifying the presence of just a single class, more closely resembling a single-label classifier. This is a much easier task to succeed at than detecting multiple classes at once. As such, a separate, external test was performed to evaluate the model's ability to classify images with a single class present and compare that performance to the model's ability to classify images with multiple classes present.

To test the model's ability to classify images of a single class, the following trial was performed. For each class the model was trained upon, five images were found online that contained only one class. These were images of just the class object and nothing else. These images were ran through the model to measure the model's ability to classify images where only one class was present. Out of the 25 images tested, 24 of them were classified correctly, and of those 24 correct classifications, 23 of them had a belief value of over 95%. The model was very successful in classifying images of a single class.

To test the model's ability to classify images of multiple classes, the following trial was performed. Four combinations of two classes were selected, and images containing those combinations were ran through the model. These were images where instances of the two classes were present and nothing else; pictures such as "Bicycle and Person" or "Sofa and Cat". It should be noted that these images had more noise present compared to the previous trial, as the images featured one class instance overlapping/obstructing the other, such as the person blocking part of the bike or the cat blocking part of the sofa. This was a common feature of the image search results, and so evaluation of the results must take into consideration this noise. It would be expected to receive results that were lower than the previous trial's results. However, the results of this test were extremely disappointing. Out of the 20 images tested on, only 6 images were classified correctly. Out of the 14 incorrect classifications, 13 of them saw the model predicting the presence of one class very strongly and the other not strong enough to exceed the threshold.

Although this is disheartening, it should be noted that in all but 2 of the 14 incorrect classifications, the model still predicted the presence of that second class stronger than any other class not present in the image. The top two results from the model were almost always the two objects present, but the scores for one of those classes was not high enough to pass the threshold. In spite of this positive outlook, it should also be noted that in these scenarios, the score for the second class varied significantly, sometimes being only a few percentage points higher than the third place class. This hints that, for the most part, the model was able to detect the presence of both classes. However, it lacked the confidence to declare its presence and also varied in belief strength significantly.

Despite these unfortunate results, there is hope for this model in the future. While the VOC2012 dataset is reasonably sized, limiting the model to only 5 classes greatly reduces the number of training data points for the model. While the entire dataset features over 17,000 images, after filtering out all the images unrelated to the 5 classes in use, just over 3,000 of them are actually used to train the

five classes. If the model were to be trained on more images (specifically more images that featured combinations of classes), then the model would gain more confidence in classifying the presence of all classes present in the image.

Fortunately, there is a straightforward solution to this dilemma. One part of the *Feature Engineering* process that was omitted in this project is data augmentation. In the context of this model, this would involve taking images in the training dataset and applying various transformations, such as rotations and crops. This allows the model to learn more generally, as it can train on a wider variety of data, resulting in an increase in the model's ability to detect the presence of classes in images where one object is partially obscured. The cropping, in particular, would be very helpful, as it would contribute to the issue of training on partially obscured classes. If data augmentation were added to the model's creation process, then it would not be surprising to see the model's ability to classify multiple classes in a single image improve noticeably. This would be the first step to take towards improving upon this model.

The next logical step would be to use a model with more layers in the CNN, such as the ResNet50 model. However, adding more layers will not necessarily improve the model's ability to perform multi-label classification. Instead, it should be argued that the next step in creating a multi-label classifier should instead be to use an entirely different model. Taking the other literature's results into consideration, the next model should perform a series of single-label classifications, one for each class the model is trained on. Then, the model should aggregate these results into one label containing all of the separate single-label classifications. This avoids the complications of multi-label classification while still performing the task.

Although the model was very successful in detecting the presence of a single class in an image, it does not sufficiently classify multiple classes at once. This is likely due to a small training dataset size and a lack of training on images where objects are partially obscured. More training data can be acquired through performing data augmentation on the current training dataset, which would be the first step taken to improve the model. The next step in making a multi-label classifier would be to switch to a model that performs multiple single-label classifications and later combines them into a single multi-label. If these steps are taken, it is expected that the model's success rate in performing multi-label classification would increase noticeably.

References

[1] Y. Wei et al., HCP: A Flexible CNN Framework for Multi-Label Image Classification, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 38, no. 9, pp. 1901-1907, 1 Sept. 2016, doi: 10.1109/TPAMI.2015.2491929.

[2] Solemane Coulibaly, Bernard Kamsu-Foguem, Dantouma Kamissoko, Daouda Traore, Deep Convolution Neural Network sharing for the multi-label images classification, Machine Learning with Applications, Volume 10, 2022, 100422, ISSN 2666-8270, https://doi.org/10.1016/j.mlwa.2022.100422.

[3] PASCAL VOC Challenge. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit.
http://host.robots.ox.ac.uk/pascal/VOC/voc2012/#devkit, Accessed: December 4, 2024.

[4] PyTorch. PyTorch Documentation.
https://pytorch.org/docs/stable/index.html, Accessed: December 4, 2024.

[5] torchvision. torchvision.models.resnet18.
https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html, Accessed: December 4, 2024.

[6] PyTorch. torch.optim.Adam.
https://pytorch.org/docs/stable/generated/torch.optim.Adam.html, Accessed: December 4, 2024.

[7] scikit-learn. Model Evaluation in scikit-learn.
https://scikit-learn.org/1.5/modules/model_evaluation.html, Accessed: December 4, 2024.