Nooreldean Koteb

Professor Lin

CS484 Data Mining

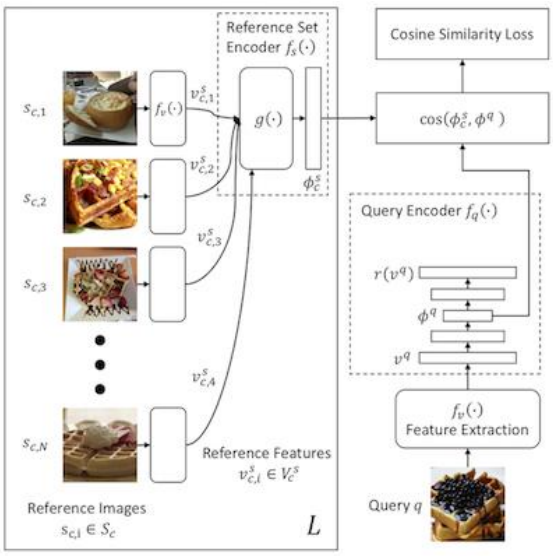
Introduction

The problem we tried to solve for our project was image classification of 101 dishes of food. We thought it would be very interesting to create and train a neural network to find features in food images then classify them to the right dish.

Related Work

We found a couple of previous attempts on this dataset CleanNet, LongReMix, DeepSelf, Inception V3+Bert, and DenseNet-121. These attempts had results of accuracy between 84.59% to 93.26%. CleanNet, LongReMix, and DeepSelf had accuracies of 90.39%, 87.39%, and 85.11% respectively and they all used some form of label noise. Inception V3+Bert Infused text with images and trained using CNNs and Bert this method had an accuracy of 84.59%. Finally, DenseNet-121 a pretrained Pytorch model had an accuracy of 93.26%.

CleanNet



Inception V3+Bert [1]

(a) Real Sashimi (b) Multiple plates

(c) Random image (d) Wrong class

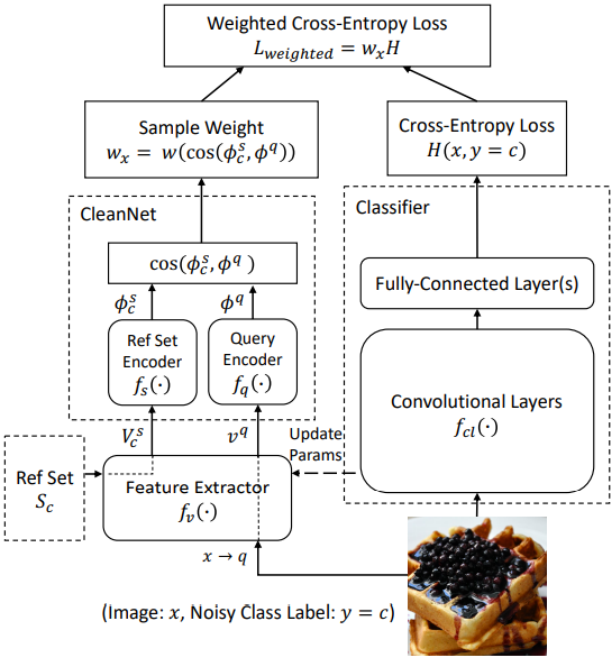
Inception V3+Bert varies wildly from the other solutions because it incorporates text to help the model identify images. All the Images in this dataset are randomly taken by normal people and are not professional. As a result, the dataset contains many images that are not very clear and can easily be mistaken for something else especially when there are multiple dishes that are similar. Inception V3 is used for images while Bert is used with text. They are then fused

[1] https://github.com/kuanghuei/clean-net

[2] https://github.com/artelab/Image-and-Text-fusion-for-UPMC-Food-101-using-BERT-and-CNNs

together to create a final prediction. The disadvantage with this method is that it requires additional information in the form of text, and this data does not come with the dataset.

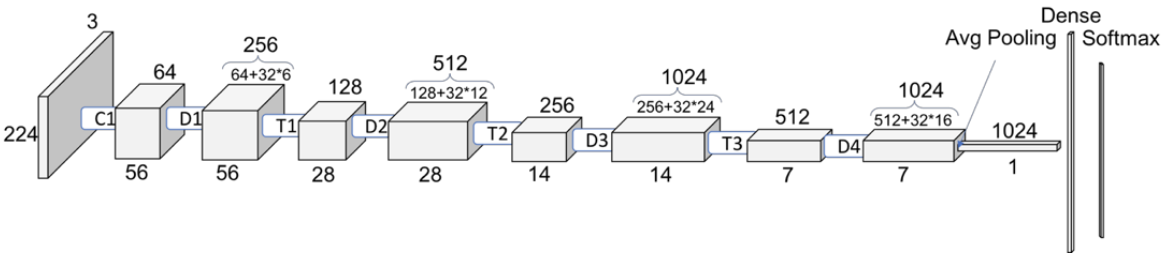
CleanNet, LongReMix, and DeepSelf all used noisy labels in their implementation. This method did much better. We will go over CleanNet's method quickly since they had the highest results. CleanNet used convolutional layers and fully connected layers like any normal neural network Image classifier, however they also used the features extracted to create noise labels and used cosine similarity to create weights for their images. This additional data was then used in the training and prediction. This method preformed fairly well but was a little more complicated compared to DenseNet-121.



[3]

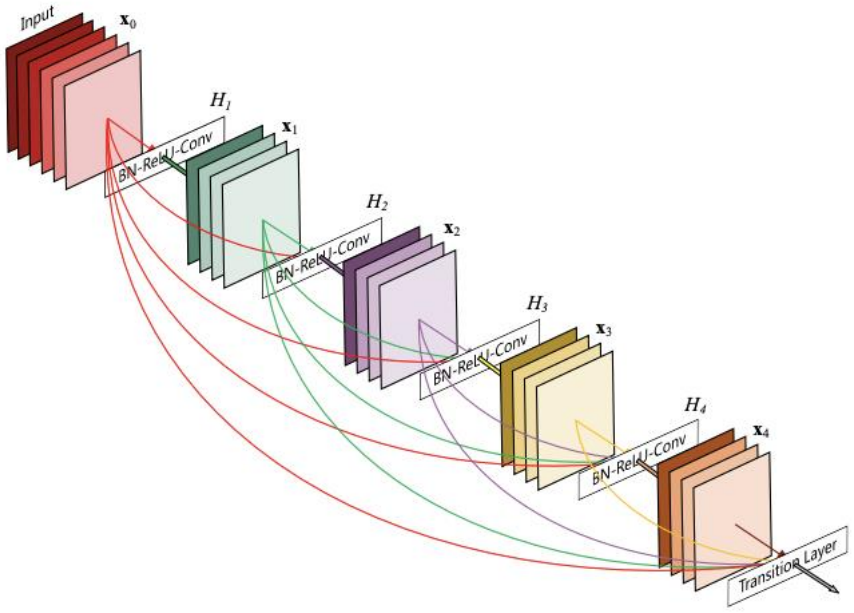
DenseNet-121 was the best preforming and the simplest requiring no additional data. This is a pretrained model in the Pytorch library consisting of many dense block layers of convolutional layers. This model is very successful at extracting features and reducing them over its many convolutions. At the end of these convolutional layers linear classification layers are used to classify the reduced extracted features into classes. Below are some Images of DenseNets, and what they consist of.

[4]



[3] https://arxiv.org/pdf/1711.07131v2.pdf
[4] https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a

5



DenseNets accepts colored 3-dimensional images with a height and width of at least 224 pixels. It also requires that Images be normalized using mean = [0.485, 0.456, 0.406], and std = [0.229, 0.224, 0.225].

Our proposed solution is to create a new neural network that can identify features and classify images from scratch using a similar approach as DenseNets. Although our approach is probably not going to be better than 93.26% accuracy, we wanted to attempt to create our own from scratch, then compare it to DenseNets.

Solution

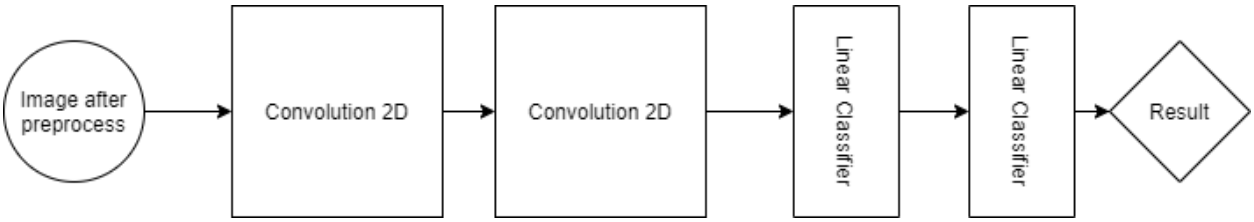
The first thing we did was make sure that we can use our graphics cards on this problem due to the size of the dataset. Using our graphics cards improved training time immensely, however it was still slow due to the amount of data we had to go through. For our solution we created a scalable and flexible class that can automatically create neural networks of various sizes and cross validate them with different parameters.

Our neural networks consisted of convolutional layers for feature extraction/reduction, and fully connected layers for classification. We randomly scaled, horizontally flipped, and rotated images to create a larger variety of images so that our neural networks do not become over fitted. We used cross entropy as our loss function and the Adam optimizer. We wanted to test with SGD as well, however we did not have the time to do that. Early stopping was implemented to validate the model every 5 epochs to make sure that our model was improving

[5] https://pytorch.org/hub/pytorch_vision_densenet/

and not just wasting time. Multiple tests were executed to find the best parameters in terms of preprocessing and model parameters.

Unfortunately, we were unable to run all our tests, and were restricted with what we can test due to the size of the dataset. Some tests required more GPU memory than was available, and since a single test could take at least 2 hours to complete. As a result, we were only capable of running few experiments on this dataset. Bigger models were restricted by GPU memory, so this is what our model looked like for our tests:



Experiments

Data:

For this project we used the food101 dataset that we found on Kaggle: https://www.kaggle.com/dansbecker/food-101. This dataset is 5.6 gigabytes when unzipped and consists of 101,000 images, and 101 classes. Each class contains 1000 images of a specific food dish (ex: hotdog, cheesecake, bruschetta). These images are not professionally taken and vary in size, resolution, angles, lighting, and dimensions.






Experimental setup:

For this project, the data was split into 3 parts, training, validation, and testing data. We used torchvision to read in all the images from the folders and apply transformations to them, then we used data.random_split to split the data into 3 parts. Training data was 70% of the overall data with validation and testing data being 15% each. Since our project is image classification accuracy was our performance metric. Models were validated every 5 epochs to make sure progress was being made, accuracy was the metric used for checking.

A cross validation method was implemented in our class that automatically trained, tested, and saved the models, and used the best model for prediction. However, we were not able to do too many cross-validation tests due to the amount of time it took to just run one test. For comparison we switched out our own convolutional layers in the neural network model with DenseNet-121 to see the difference.

Experimental results and analysis:

(Epochs is where the validation decided there was no more improvement and stopped.)

Resize	Rotation	Learning rate	Epochs	Accuracy
128	0	0.0001	30	15.83%
32	0	0.0001	15	14.13%
80	15	0.001	20	13.00%
32	0	0.001	15	12.73%
64	0	0.001	15	12.47%
64	30	0.001	15	11.87%
100	0	0.001	35	11.65%
128	0	0.001	15	8.57%

Our results did not turn out too well, we feel a big part of that is because of how long it took to run one cross validation test, and how much GPU memory was required to run our models. Some of our results were also lost halfway if a session was dropped with google collab. With that being said we did notice that a lower learning rate increased the accuracy. Our assumption is if we were able to create larger convolutional network like DenseNet, we could have increased our scores immensely since more features would have been detected and reduced.

	Resize	Epochs	Learning Rate	Time	Accuracy
Best Model	128	30	0.0001	~4.5hrs	15.83%
Baseline(DenseNet)	224	45	0.001	~8hrs	51.87%

Replacing our convolutional neural network with the pretrained DenseNet-121 convolutional network (we used our own classifier) resulted in much better results. We were hoping to run

another test on DenseNet with a lower learning rate, however it took much longer than expected and there was no time for another run.

Challenges, Lessons, and Conclusion

This project was extremely challenging. We underestimated the amount of time, and resources that would have been required to achieve good results on this dataset. We had to purchase google collab pro to increase our GPU memory since we would constantly get errors of not having enough memory whenever we increased the size of images above 80 pixels or increased our starting hidden layers above 8. In addition to memory issues, a cross validation test took anywhere between 2 and 5 hours depending on where our validation test would stop the training session. This is the time it took using a 16-gigabyte graphics card on google collab pro.

We learned that when it comes to data mining projects resources and time should be a major consideration before taking on a project. Some datasets require a lot more time, effort, and resources to achieve good results. Although we did not get amazing results on this project, we learned a lot of new methods, techniques, and theories. Although this project didn't turn out as planned, we are really interested in doing another image classification project this summer.