Multi-Label Classification

Willie Maddox

Abstract—Machine Learning Engineer Nanodegree Capstone Project

Index Terms—Deep Learning, Machine Learning, Neural Networks, Keras, Tensorflow, VGG16, MS-COCO, NUS-WIDE.

1 Introduction

THIS demo file is intended to serve as a "starter file" for IEEE Computer Society journal papers produced under LaTeX using IEEEtran.cls version 1.8 and later. I wish you the best of success.

mds
December 27, 2012

2 Definition

2.1 Project Overview

In machine learning, image classification is the problem of deciding to which category a particular image belongs or to which categories objects in an image belong. Traditional classifiers, such as binary and multi-class classifiers return as output only one value, the prediction, whereas multilabel classifiers produce a vector of output values. Much of the previous work for solving image classification problems has been focused on using a single label per image to train a classifier. For some of the more popular datasets such as ImageNet [1] and CIFAR-10 [2], each image has associated with it a single label and there are many images per label. This works well for a dataset like MNIST [3] where each instance is a black and white image of a single handwritten digit between 0 and 9. But for images that illustrate the real world such as photographs, there is almost never a single contextual topic in the image. For example, Fig.1 is a picture of bolts, but there are also nuts, washers and a wooden table. So in reality this image has (at least) four tags. The goal of this project is to create a multi-label classifier that can be trained using images that have multiple labels per image.

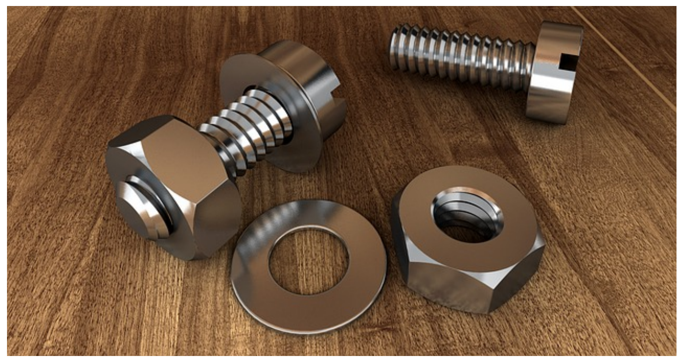


Fig. 1. Nuts n Bolts

2.2 Problem Statement

Classifiers such as AlexNet and VGGNet which are typically trained on single class image data such as the 1000 class ImageNet dataset are also able to classify multiple objects in images even though they weren't specifically trained to do so. This happens because the output of the classifier is simply a list of probabilities for every possible class label. These probabilities range from zero to one and the probability whos value is closest to one is typically regarded as the *winning* prediction. However, if we take the top 3 or even the top 5 highest probable labels we will often find that these *runner-up* predictions can also be found in the image.

• *W. Maddox is currently not affiliated with an academic institution.*

Manuscript received November 21, 2017; revised November 26, 2017.

The problem that we will try to solve is finding multiple features (or labels) in a single image. We would also like to know how much better our trained multi-label classifier is at predicting the top k labels for a given image as compared to a standard single-label classifiers like VGG16 [4] trained on ImageNet? Can we train a classifier using multi-labelled image data to perform at least as well as existing models trained on single-class image data. A model trained on multi-labelled image data should, in theory, require fewer data samples since more information per image is avaliable to training. Is this also true? Let's find out.

2.3 Metrics

After reading through previous literature on multi-label classification, we found that there are quite a few metrics appropriate for this problem. The mean average precision (mAP) is a widely used metric for comparing between trained models and has been regarded as the best metric for classification problems [5]. Other popular metrics include precision, recall, F_1 score, Jaccard index, 0/1 loss and Hamming loss [6], [7], [8].

For multilabel classification, particularly, the precision, recall, and F_1 score have three different variants which we will use [9], [10], [11], [12]. *Macro-averaging* measures the average classification performance across labels.

$$\mathrm{MP} = \frac{1}{L}\sum_{j=1}^{L}\frac{\sum_{i=1}^{N} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N} y_{ij}} \tag{1}$$

$$\mathrm{MR} = \frac{1}{L}\sum_{j=1}^{L}\frac{\sum_{i=1}^{N} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N} \hat{y}_{ij}} \tag{2}$$

$$\mathrm{MF}_1 = \frac{1}{L}\sum_{j=1}^{L}\frac{2\sum_{i=1}^{N} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N} \hat{y}_{ij} + \sum_{i=1}^{N} y_{ij}} \tag{3}$$

This metric treats all classes equal regardless of their sample size, so focusing on getting rare classes right can result in a significant increase in performance. To counterbalance this bias we also perform *instance-averaging* which measures average classification performance across examples,

$$\mathrm{iP} = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{L} y_{ij}\hat{y}_{ij}}{\sum_{j=1}^{L} y_{ij}} \tag{4}$$

$$\mathrm{iR} = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j=1}^{L} y_{ij}\hat{y}_{ij}}{\sum_{j=1}^{L} \hat{y}_{ij}} \tag{5}$$

$$\mathrm{iF}_1 = \frac{1}{N}\sum_{i=1}^{N}\frac{2\sum_{j=1}^{L} y_{ij}\hat{y}_{ij}}{\sum_{j=1}^{L} \hat{y}_{ij} + \sum_{j=1}^{L} y_{ij}} \tag{6}$$

and *micro-averaging* which measures average classification performance across both labels and samples.

$$\mu\mathrm{P} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{L} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{L} y_{ij}} \tag{7}$$

$$\mu\mathrm{R} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{L} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{L} \hat{y}_{ij}} \tag{8}$$

$$\mu\mathrm{F}_1 = \frac{2\sum_{i=1}^{N}\sum_{j=1}^{L} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{L} \hat{y}_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{L} y_{ij}} \tag{9}$$

For both of these, the more frequent classes will be dominant and have a greater impact on performance.

Equations 1 through 9 require the $\hat{y}_{ij}$ values to be binary 1/0 predictions. However, most classifier models including ours return an output of predictions in the form of floating point values on the interval $[0, 1]$. The process of turning these raw predictions into binary predictions is commonly referred to as *label decision* and there are two common approaches to this type of decision: top-k and thresholding. [13]

In the top-k approach, for each sample, the k labels with the highest prediction value are set to 1 and the rest are set to 0. This approach works very well when working with datasets that have nice evenly distributed labels across samples but less so when working with unbalanced datasets. One variation of the top-k approach is to use a *per-sample* top-k, the value of which is determined by the number of ground truth labels in each sample. For example, if a sample has 3 ground truth labels, then we assign a 1 to the top 3 predictions from that sample and 0 to the rest of the predictions. The next sample might only have 1 ground truth label in which case only the highest predicted

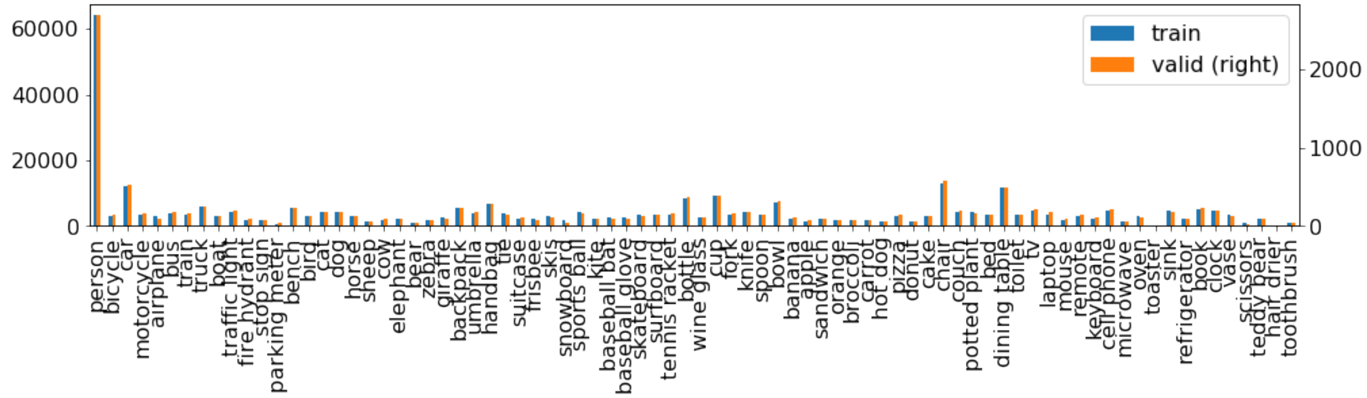


Fig. 2. Number of samples in MS-COCO containing at least one instance of a label.

value will be set to 1. And so on for the rest of the dataset. We have not yet seen this approach in the literature but think it might be interesting to investigate.

The second type of label decision is thresholding. Using this approach the label is predicted as present if the raw prediction exceeds some predefined threshold τ.

$$\hat{y}_{ij} = \begin{cases} 1 & \text{if} \quad h_{ij} \geq \tau \\ 0 & \text{if} \quad h_{ij} < \tau \end{cases} \tag{10}$$

Typically this value is set to 0.5 and is a logical choice for multiclass classification where the raw predictions come from the output of a softmax layer. For multilabel classification it is more common to see a sigmoid function used for the final layer, hence the separation between positive and negative predictions becomes less obvious. To this end, we choose a threshold that minimizes the difference in label cardinality between the ground truth labels y_{ij} and the predictions $\hat{y}_{ij}$. [14]

$$\text{LCard} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} y_{ij} \tag{11}$$

$$\tau = \text{argmin} \left\| \left(\text{LCard} - \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} 1_{h_{ij} \geq \tau} \right) \right\| \tag{12}$$

When comparing our model to the benchmark models, we will use Eqns 1 through 9. When comparing the results of our model to those previously reported in the literature, we will restrict ourselves to using only those metrics for which there is a direct comparison to the previous work.

3 Analysis

3.1 Data Exploration

For this study we will use the (2017) MS-COCO [15] dataset as our primary dataset. The COCO dataset is freely downloadable from the internet and contains images with multiple labels per image making them ideal for this kind of study. The COCO dataset contains a training set of 118,287 images and a validation set of 5,000 images. The images are colored and the size of each image is about 600 * 400 pixels. The MS-COCO dataset has 80 labels with 2.9 labels per image on average. However, both training and validation sets have images with no labels. This is not so much a problem for training as it is for testing. Specifically, when we evaluate our predictions using the metrics from Section 2.3, empty labeled samples could lead to divide by zero errors. Hence, we will only use labelled samples for testing.

3.2 Exploratory Visualization

In Fig. 2, we can see that the dominating class label is *person* with 64115 occurances in the training set and 2693 in the validation set. The least dominating class labels are the *toaster* and *hair drier* with 217 and 189 in the training set and 8 and 9 in the validation set, respectively. In Fig. 3, we show the number of unique tags per sample. The majority of the samples have

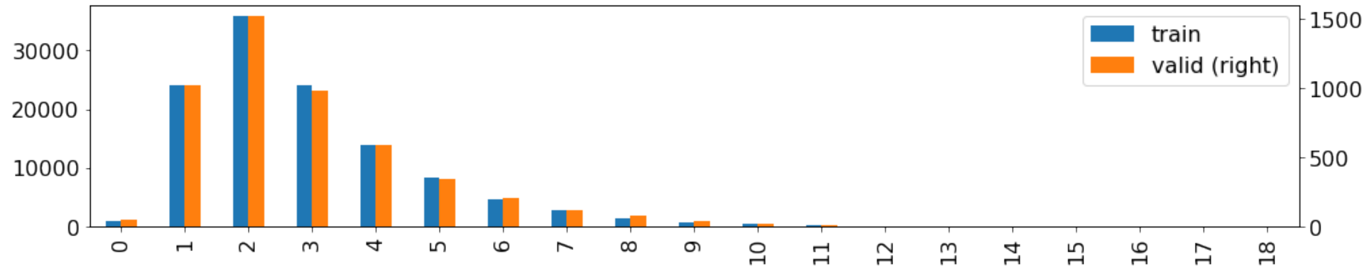


Fig. 3. Number of unique labels per sample in MS-COCO.

at least 1 to 4 unique labels with 2 unique labels being the most dominate. We also notice that the training and testing sets have 1021 and 48 samples with no labels. We will keep this fact in mind when running our evaluation metrics. In both figures, the training set has 20 to 30 times more samples per label than the validation set and the ratios between the training and validation sets are consistent (i.e. the heights of the train and valid bars are about the same.)

3.3 Algorithms and Techniques

To solve this problem, we will model our classifier using a deep convolutional neural net. We will not train our model from scratch, but rather we will perform transfer learning using the VGG16 convnet trained on imagenet. We will read the images from disk and perform data augmentation using worker threads. This will make it so the GPU doesn't have to wait on batches thereby maximizing our training efficiency. We will use sigmoid activation in the output layer [16], binary cross entropy for our loss function [17] and mini-batch stochastic gradient descent with momentum for backpropogation. Table 1 outlines the hyperparameters that will be tuned to optimize the classifier. We will also use some Keras callbacks during our training. ModelCheckpoint to save only the best models at each epoch, EarlyStopping to stop training when we stop making progress and CSVLogger for plotting purposes.

3.4 Benchmark

We were unable to find a VGG16 model trained on COCO to use for benchmarking purposes.

TABLE 1
List of hyperparameters to tune to optimize the classifier. There are 2 options for image preprocessing that is used by both Stage 1 and 2: inet - use a global scaling factor and zmuv - use per-image zero-mean unit-variance. Stage 1 is where the top model gets trained with bottleneck features. We train the entire model (base + top) in Stage 2. Please refer to Sec. 4 for more details. Note: Batch sizes of 256 will be used only for a few select cases.

Stage	Hyper-parameter	Variants
	Image Preprocessing	INET, ZMUV
1	Classifier Design	77, avg, max
1	Number of Hidden FC Layers	1 or 2
1	Number of Nodes per Layer	1024, 2048, 4096
1	Learning Rate	0.1, 0.05
1	Batch Size	32, 64, 128, 256
2	Number of Frozen Layers	15, All
2	Learning Rate	
2	Batch Size	32, 64, 128, 256
2	Data Augmentation	h, w, r, s, z, f

But we did find multiple results of multilabel classifiers trained on COCO in the literature. We will use these results when evaluating the performance of our model in 5.1.

4 METHODOLOGY

4.1 Data Preprocessing

There are 2 types of preprocessing we do on the image data. The first is image resizing and normalizing. Before passing any image through the network it must first be resized to match the height and width of the input layer and its values must be converted (normalized) from uint8's to float32's in the range of -1 and 1.

The method by which the images are normalized became a tunable hyperparameter. Keras comes shipped with a preprocess_image[1] function that is intended to be used for data normalization on VGG16. If using Keras with the Tensorflow backend this normalization boils down to dividing by 127.5 and then subtracting by 1 for every image. This is the INET option in Table 1. As an alternate option, the various flavors of model generators in Keras provide to option to normalize images by either featurewise or samplewise zero-mean unit-variance (or ZMUV). These three normalization methods each provide slightly different results and so we leave the decision as to which one is best as a tunable hyperparameter.[2]

The other type of image preprocessing involves various forms of data augmentation. Going by the codes in Table 1 we will augment the image data by (h) shifting along the height direction, (w) shift along the width direction, (r) performing random rotation, (s) skew transform, (z) multiplying by a zoom factor, and (f) performing horizontal flips. We will also shuffle the training data after each epoch. We will only perform data augmentation during training, not during top model training as the input images to the top model aren't images but rather features of a deep layer.

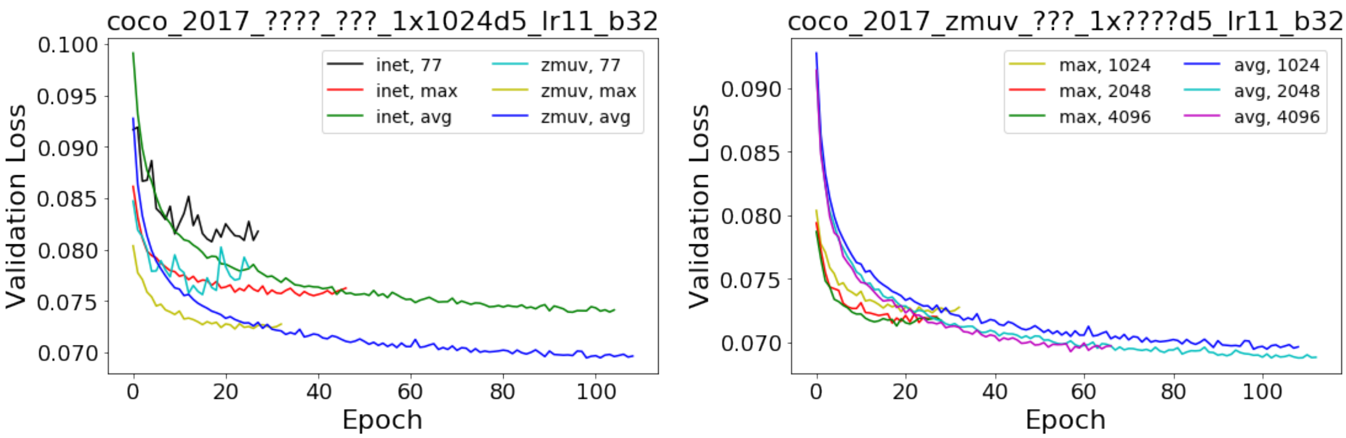
4.2 Implementation

The implementation is divided into 2 stages. In the first stage we load the VGG16 model but without the last 3 fully connected layers. These layers are specific to the 1000 ImageNet classes that the VGG16 model was originally trained on. We then run a single prediction calculation the both the training and validation sets. But instead of getting the typical label predictions, we get the output of the last VGG16 convolutional layer. Depending on our Classifier Design (See Table 1), the output will either be (77) the raw $7 \times 7 \times 512$ features, (avg) the 512 features that result from applying global average pooling to the raw features, or (max) the 512 features that result from applying global max pooling to the raw features. We refer to these as bottleneck features and save them (along with their ground truth labels) as python numpy arrays. We then construct a small[3] classifier model that we can train using the bottleneck features. The goal of Stage 1 is to train this "top" model as best we can before attaching it to the VGG16 base for fine tuning. There are a few reasons for pretraining the classifier. The top model has fewer layers and the bottleneck features are much smaller $7 \times 7 \times 512 = 25088$ than the original raw images $224 \times 224 \times 3 = 150528$. This leads to fewer parameters to learn and faster training. We will still need to fine tune the full model in Stage 2, but pretraining the top model in this way provides a much better starting guess than simply initializing the top model with random weights. Another happy side effect of pretraining the classifier is we can make faster progress with tuning many of our hyperparameters. Specifically, the optimal classifier design, regularization method, and number of nodes per layer can each be found much faster.

Once the optimal Stage 1 hyperparameters are found then we can proceed to Stage 2 and attach our freshly trained classifier to our base VGG16 model. We can choose to freeze a certain number of layers (in the base model) before we start fine tuning or we can freeze them all. For this project we either freeze all layers or just the ones up to the last convolutional block. Weights on frozen layers do not update during backpropogation. Since we are now training the final model on the raw images it's safe to augment our data as prescribed in Section 4.1. We vary the learning rate for various batch sizes as presented in Table 1 and choose the model which minimizes the loss function.

1. https://github.com/keras-team/keras/blob/master/keras/applications/imagenet_utils.py

2. This part of the study raised an interesting question. Should the input image normalization method be part of the (or be supplied as) supplemental information to the saved model? In other words, if I download and use a trained model, should I not also expect the model to somehow tell me in what format the model expects the input data to be? There seemed to be conflicting opinions as to which normalizing method should be used.



(a) (b)

Fig. 4. Top model validation losses as a function of epoch. Each panel explores a different set of hyper-parameters. The yellow and blue curves are of the same data in both plots.

4.3 Refinement

At this point we would like to refine our model by tuning some of the Stage 1 hyper-parameters. We begin by defining some rules for how we will format our log and top model weights filenames[4]. This is best explained with a simple example. Consider the following string, ″coco_2017_zmuv_avg_1x2048d5_lr11_b32″. The first field, ″coco″, refers to the name of the dataset and ″2017″ is the subset (or version) of the dataset. The other fields follow directly from Table 1. ″ZMUV″ indicates zero-mean unit-variance scaling was used to normalize the input images. The 4^{th} field, ″avg″ denotes the use of GlobalAveragePooling on the final layer of the base VGG16 model. This also describes the shape of the input to the top model. The next term, ″1x2048d5″ is a combination of 3 hyper-parameters that describe the fully connected layers in the top model. The ″1x2048″ indicates that there is 1 hidden layer and that it has ″2048″ nodes. All FC layers are immediately followed by a ReLU activation layer which is then followed by either a dropout (″d″) layer or a batch normalization (″bn″) layer. For dropout, the number after the ″d″ indicates the percentage of nodes that get dropped. So ″d3″ would mean that only 30% of the nodes are dropped. The second to last field gives information about the learning rate in the form ″lrXY″ which should be interpreted as $\mathrm{lr} = \mathrm{X} * 10^{-\mathrm{Y}}$. So, lr53 would indicate a learning rate of 0.005, lr11 would be 0.1, and so forth. The final field, ″b32″ denotes the batch size.

The first hyper-parameter we looked at was image preprocessing method. We compared the results between each method by looking at how they compared for each of the different classifier designs. In all cases, the ZMUV preprocessing method led to lower validation losses as seen in Fig. 4a. We also looked at how well the different classifier designs compared to each other. Looking again at Fig. 4a we can see that the 77 design performed much worse than the global max and global average pooling designs. To determine whether global max or global average pooling is better, we compare them against different layer sizes as shown in Fig. 4b. In all three cases the validation loss was lower when using global average pooling. In light of these preliminary results we will restrict the remaining hyperparameter tuning calculations to using only ZMUV to preprocess the images and global average pooling to cap off the end

3. Small in comparison to the base VGG16 model which is 16 layers deep. Our classifier will have a single hidden layer between the input layer and the ourput layer.

4. Logs will be saved as csv's and the weights will be saved in hdf format using the hdf5 extension. The files can be found from the root of the project directory in data/bottleneck_top_models.

of the VGG16 pretrained model. A consequence of transfer learning is that the input shape of our top model will need to match the output shape of our base model. With global average pooling, this simplifies the input shape from $512 \times 7 \times 7$ features to $512 \times 1 \times 1$ features. This will greatly reduce the training time of our top model classifier and enable faster progress while we tune the remaining hyperparameters.

The results of Fig. 4b also suggest that there may exist a relationship between the validation loss and the layer size, namely that increasing the number of nodes per layer brings about lower validation losses. It worth noting that up until now we have been using a somewhat high learning rate of 0.1. The higher the learning rate the more the validation loss fluctuates from one training epoch to the next. This fact makes it hard to say one way or another which layer size is better.

In the rest of this section, we present the results from a grid search that we performed on the remaining top model hyperparameters, namely those listed under Table 1 stage 1. The original VGG16 model has 3 dense fully connected layers. The first 2 layers are hidden and have 4096 nodes each. The last output layer has 1000 nodes equal to the number of classes in ImageNet. Since the COCO dataset consists of only 80 unique classes we will test the performance of using only a single hidden layer versus using two hidden layers. We would also like to see how different layer sizes (1024, 2048, and 4096) affect performance. The size of the hidden layers in VGG16 is 4096. Again, since we are working with only 80 classes, we may be able to get as good performance with smaller layer sizes. As we did before in Fig. 4 we will use a learning rate of 0.1 but also compare how the performance changes when we cut the learning rate in half (e.g. 0.05). In general, the higher the learning rate, the faster the training time. If it appears from these preliminary results that we need to explore further smaller learning rates, we will do so on a case by case basis. The final hyperparameter for tuning our top model is the batch size. We will look at 3 sizes, 32, 64, and 128. The grid search on these four hyperparameters yields $2*3*2*3 = 32$ pretraining runs. For brevity,

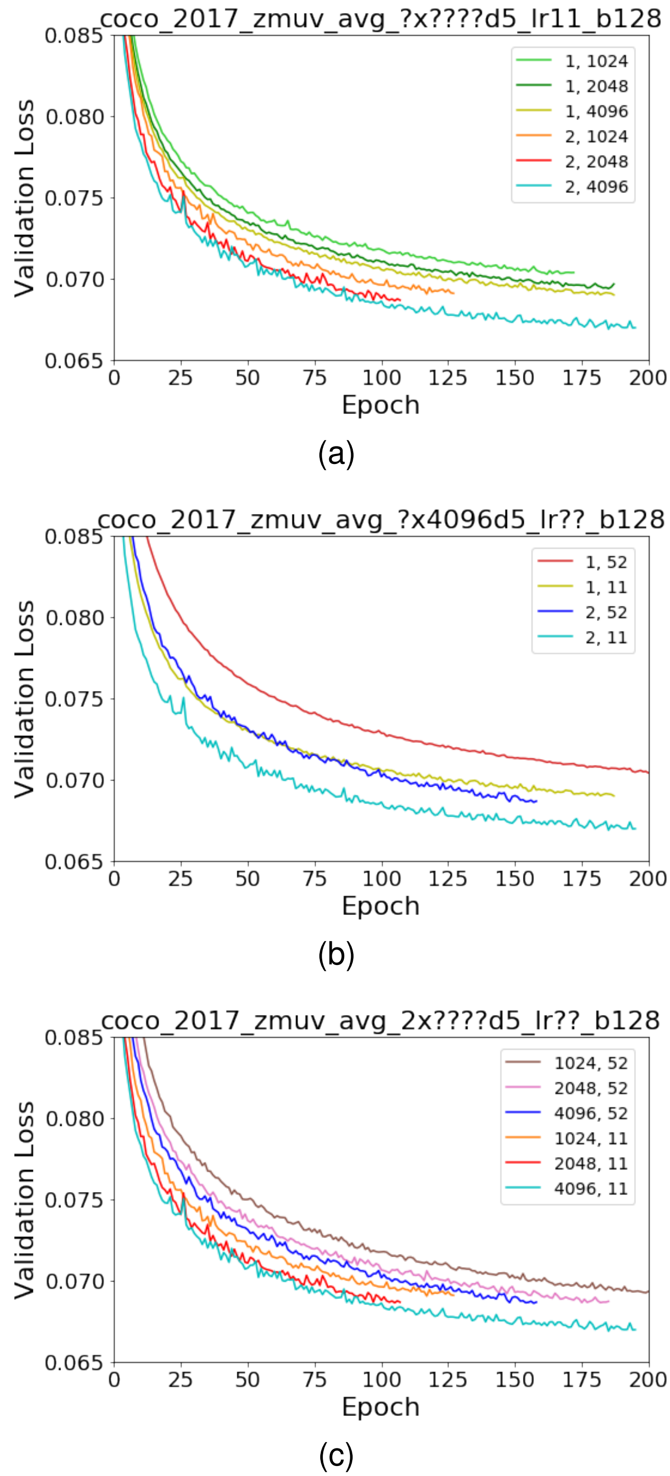
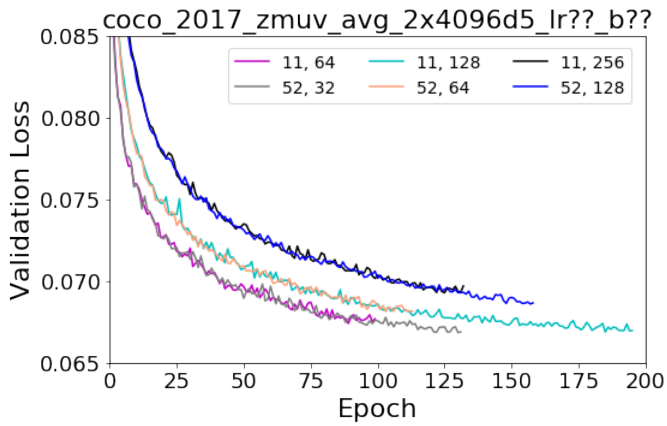
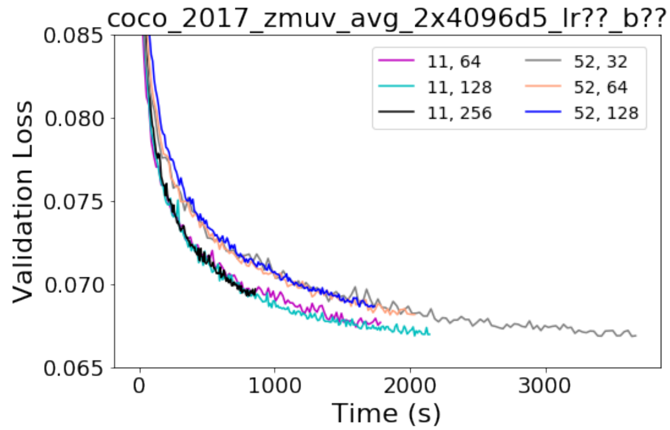


Fig. 5. Top model validation losses as a function of epoch. Each panel explores a different set of hyper-parameters. All results are for a mini batch size of 128. The colors are the same for same data in each plot. For example, The teal curves in all three plots are of the same data. The shorter curves are a result of early stopping. Refer to Section 4.3 for further details.



(a)



(b)

Fig. 6. Top model validation losses as a function of (a) Epoch and (b) Time. The lr11 and lr52 correspond to 0.1 and 0.05 learning rates respectively. In (b), We scaled the x-axis for each run by multiplying through the average training time per epoch.

only a selection of these results are shown[5].

The results shown of Fig. 5 are all for a mini batch size of 128. In Fig. 5a, we compare how the number of hidden layers and the size of the hidden layers affects performance. For both the single and double layer cases validation loss decreases with increased layer size. Also, the losses for the double layer models are all lower than those of the single layer models. Next, in Fig. 5b, we hold the layer size fixed at 4096 nodes per layer and compare relationships between the number of layers and the learning rate. Independent of the learning rate, the double layer runs perform much better than the single layer runs. We also observe that the training runs with the higher learning rate start to flatten out a little faster than the others. This is typical as higher learning rates tend to find a local minimia quickly. In Fig. 5c, we hold the number of hidden layers fixed at 2 layers and compare the realtionship between number of nodes per layer and learning rate. As the number of nodes per layer increases the validation loss decreases. The validation loss was also lower in all cases where the larger learning rate was used.

These results suggest better performance should be expected using 2 hidden layers with 4096 nodes per layer. Also, we can afford to use a little higher learning rate than we usually would. Typically, high learning rates tend to lead to overfitting. However, since the entropic capacity of the top model classifier is fairly low it does not have as strong a tendency to overfit.

The final hyperparameter to optimize is the batch size. Using the results above, we set the number of layers to 2 and the number of nodes per layer to 4096. We plot the relationship between learning rate and batch size in Fig. 6. In Fig. 6a, we see an interesting relationship. The curves of training runs tend to overlap when both the learning rate and the batch size are both multiplied by the same constant factor (here 2 or 12). The loss curves in Fig. 6b are the same as those in Fig. 6a but plotted with respect to training time. From this perspective it appears that in order to train fast one needs only focus on setting the learning rate as high as possible without overfitting. Increasing the batch size can help smooth out the validation losses between epochs and consequently reduce overfitting.

It is not clear at this point what will be the best choice for our final model in terms of learning rate and batch size. In the next section we fine-tune a few different models and compare the results using our evaluation metrics.

5. See the Jupyter notebook for all 32 pretraining plots.

5 Results

5.1 Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the models solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?

Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?

Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?

Can results found from the model be trusted?

5.2 Justification

In this section, your models final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

Are the final results found stronger than the benchmark result reported earlier?

Have you thoroughly analyzed and discussed the final solution?

Is the final solution significant enough to have solved the problem?

6 Conclusion

6.1 Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

Have you visualized a relevant or important quality about the problem, dataset, input data, or results?

Is the visualization thoroughly analyzed and discussed?

If a plot is provided, are the axes, title, and datum clearly defined?

6.2 Reflection

To summarize the project we used transfer learning to create a multilabel classifier trained on the MS-COCO dataset. We started with a VGG16 neural network trained on ImageNet as our base model and replaced the final fully connected layers with a classifier pretrained using bottleneck features.

During both training and pretraining the validation accuracy was always high between

TABLE 2
Comparisons on MS-COCO validation dataset for $k = 3$. The macro precision MP, macro recall MR, and macro F score MF_1 are evaluated for each label class before averaging. The micro precision $\mu\mathrm{P}$, micro recall $\mu\mathrm{R}$, and micro F score $\mu\mathrm{F}_1$ are averaged over all instance-label pairs.

Method	MP	MR	MF_1	μP	μR	$\mu\mathrm{F}_1$
KNN [18]	32.6	19.3	24.3	42.9	53.4	47.6
WARP [12]	59.3	52.5	55.7	59.8	61.4	60.7
Softmax [19]	59.0	57.0	58.0	60.2	62.1	61.1
BCE [19]	59.3	58.6	58.9	61.7	65.0	63.3
No RNN [19]	65.3	54.5	59.3	68.5	61.3	65.7
CNN-RNN [19]	66.0	55.6	60.4	69.2	66.4	67.8
ResNet-101 [20]	84.3	57.4	65.9	86.5	61.3	71.7
ResNet-107 [21]	84.4	57.6	66.1	86.4	61.4	71.8
ResNet-101-s [21]	84.3	57.7	66.2	86.3	61.8	72.0
ResNet-SRN-a [21]	85.8	57.5	66.3	88.1	61.1	72.1
ResNet-SRN [21]	85.2	58.8	67.4	87.4	62.5	72.9
	56.7	55.7	54.1	60.3	61.3	60.8
	56.8	55.6	54.1	60.4	61.3	60.9
	56.0	55.6	53.9	60.1	61.0	60.6
	55.2	54.3	52.8	59.6	60.5	60.0
	52.7	52.2	50.5	58.2	59.1	58.6
	55.5	54.6	53.0	59.7	60.6	60.1
	56.1	55.2	53.7	59.9	60.8	60.4
	56.0	54.7	53.2	59.9	60.8	60.4
	54.5	55.8	53.2	60.0	60.8	60.3
	54.9	55.4	53.1	60.2	61.1	60.7
	56.8	56.0	54.5	60.4	61.1	60.9
	59.2	58.5	57.6	69.4	69.4	69.4
	65.2	45.4	51.4	75.4	51.5	61.2

96% and 98%. Accuracy is (true posititives + true negatives) / (total population). There are $5000 * 80 = 400000$ total possibilities for positive labels in the population but there are only 14631 ground truth positives. This means that in the worst case scenario, if we predicted all negatives, we would still wind up with an accuracy of 96.34%. This is a great example of how accuracy can be a very misleading metric. As such, we did not use this metric in the project.

We explored a wide range of hyperparameters and while some of them were presented a greater challenge in terms of optimization, we did not feel that the overall task was necessarily difficult. Our final model We were able to do better than some groups but not every group. It seems that the best models for multi-label classification are those which incorporate region proposal networks and/or some flavor of RNN.

6.3 Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

We take a very naive approach to creating the labels for the COCO dataset. Specifically each image has associated with it an 80 element vector of 0's or 1's where the 1's denote that the particular class is found in the image. However the label vector does not take into account multiple instances of the same class, nor does it take into account how much of the actual image the labeled object occupies. It might be interesting to use segmentation maps or object bounding boxes to obtain a percent coverage of the image and use those values instead of a simple binary vector.

It would have been interesting to explore the label-to-label hierarchial relationships using WordNet and use those relationships to incorporate either a word2vec type embedding layer or an RNN based on the hierarchial sequences. Also, incorporation of region proposal networks using either bounding boxes or segmentation maps would have been useful. It would also have been interesting to see how zcf whitening might affect our training. However, with such a large dataset as COCO, we didn't have the computational capacity to perform the SVD.

If you used your final solution as the new benchmark, do you think an even better solution exists?

Before submitting, ask yourself...

- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?

Acknowledgments

The authors would like to thank...

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[2] Krizhevsky, A. and Hinton, G., "Learning Multiple Layers of Features from Tiny Images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. of the IEEE*, vol. 86, 1998, pp. 2278–2324.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[5] V. Lavrenko, "Evaluation 12: mean average precision," https://www.youtube.com/watch?v=pM6DJ0ZZee0&list=PLBv09BD7ez_6nqE9YU9bQXpjJ5jJ1Kgr9&index=12, 2014.

[6] G. Tsoumakas and I. Katakis, "Multi-Label Classification: An Overview," *International Journal of Data Warehousing and Mining (IJDWM)*, 2007. [Online]. Available: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jdwm.2007070101

[7] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427 – 437, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306457309000259

[8] F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus, *Multilabel Classification*. Springer International Publishing, 2016.

[9] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Deroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition*, vol. 45, no. 9, pp. 3084 – 3104, 2012, best Papers of Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'2011). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320312001203

[10] X. Wu and Z. Zhou, "A unified view of multi-label performance measures," *CoRR*, vol. abs/1609.00288, 2016. [Online]. Available: http://arxiv.org/abs/1609.00288

[11] O. Koyejo, P. Ravikumar, N. Natarajan, and I. S. Dhillon, "Consistent multilabel classification," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 3321–3329. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969442.2969610

[12] Y. Gong, Y. Jia, T. Leung, A. Toshev, and S. Ioffe, "Deep convolutional ranking for multilabel image annotation," *CoRR*, vol. abs/1312.4894, 2013. [Online]. Available: http://arxiv.org/abs/1312.4894

[13] Y. Li, Y. Song, and J. Luo, "Improving pairwise ranking for multi-label image classification," *CoRR*, vol. abs/1704.03135, 2017. [Online]. Available: http://arxiv.org/abs/1704.03135

[14] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Mach. Learn.*, vol. 85, no. 3, pp. 333–359, 12 2011. [Online]. Available: http://dx.doi.org/10.1007/s10994-011-5256-5

[15] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[16] G. Kurata, B. Xiang, and B. Zhou, "Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence," *Proceedings of NAAL-HLT*, pp. 521–526, 6 2016.

[17] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, Feb 2005. [Online]. Available: https://doi.org/10.1007/s10479-005-5724-z

[18] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "NUS-WIDE: A Real-World Web Image Database from National University of Singapore," in *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., July 8-10, 2009. [Online]. Available: http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

[19] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "CNN-RNN: A unified framework for multi-label image classification," *CoRR*, vol. abs/1604.04573, 2016. [Online]. Available: http://arxiv.org/abs/1604.04573

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[21] F. Zhu, H. Li, W. Ouyang, N. Yu, and X. Wang, "Learning spatial regularization with image-level supervisions for multi-label image classification," *CoRR*, vol. abs/1702.05891, 2017. [Online]. Available: http://arxiv.org/abs/1702.05891

[22] K. Sechidis, G. Tsoumakas, and I. Vlahavas, *On the Stratification of Multi-label Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 145–158. [Online]. Available: https://doi.org/10.1007/978-3-642-23808-6_10

PLACE PHOTO HERE

Willie Maddox Biography text here.