

# Scalable Large Scale Visual Recognition Using Multi-Label Image Classification

by Aaron R. Huang

Submitted to the  
Department of Electrical Engineering and Computer Science  
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

© 2019 Aaron R. Huang. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and to distribute publicly  
paper and electronic copies of this thesis document in whole and in part in any medium  
now known or hereafter created.

Author:

---

Department of Electrical Engineering and Computer Science  
May 20, 2019

Certified by:

---

Fredo Durand, Professor, CSAIL Computer Graphics Group  
May 20, 2019

Accepted by:

---

Katrina LaCurts, Chair, Master of Engineering Thesis Committee

# Scalable Large Scale Visual Recognition Using Multi-Label Image Classification

by

Aaron Huang

Submitted to the Department of Electrical Engineering and Computer Science  
on May, 2019 in Partial Fulfillment of the  
Requirements for the Degree of Master of Engineering in  
Electrical Engineering and Computer Science

## **ABSTRACT**

Today, the amount of image categories and labels for visual recognition is growing at an astonishing rate, and it is becoming increasingly impractical to keep up a high level of accuracy across all categories in addition to retraining these deep networks to classify all these new labels along with previous ones. However, we note that the majority of new labels that are being added now are simply subsets and combinations of existing labels, just an extra step of specificity. In this study, we will be looking at creating a model that can be trained on traditional datasets, either single-class or multi-class, but then can be quickly adapted and trained on new multi-class image datasets, in which the class components are part of the original training set, without losing accuracy on the original dataset and not having to train on it either.

Thesis Supervisor: Fredo Durand

Title: Professor, CSAIL Computer Graphics Group

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Related Work</b>	<b>10</b>
2.1	ILSVRC Models . . . . .	10
2.1.1	AlexNet . . . . .	10
2.1.2	VGG Net . . . . .	10
2.1.3	Inception Net . . . . .	11
2.1.4	ResNet . . . . .	12
2.1.5	Highway Networks . . . . .	12
2.2	DARTS: Optimizing Accuracy Specificity Tradeoff . . . . .	13
2.3	Multi-Label Image Classification (MLIC) . . . . .	13
2.3.1	CNN-RNN . . . . .	14
2.3.2	Weakly Supervised Detection . . . . .	16
2.4	Transfer Learning From Imagenet . . . . .	16
<b>3</b>	<b>Squeeze and Excitation Network</b>	<b>18</b>
3.1	Channel Transformation . . . . .	19
3.2	Squeeze . . . . .	19
3.3	Excitation . . . . .	20
3.4	Comparison to Related Models . . . . .	21
<b>4</b>	<b>Model</b>	<b>23</b>
4.1	SEBlock . . . . .	23
4.1.1	2 Layer SEBlock . . . . .	23
4.1.2	3 Layer SEBlock . . . . .	24
4.2	ResNet Architecture . . . . .	24
4.2.1	18 Layer SENet . . . . .	25

4.2.2	152 Layer SENet . . . . .	26
4.3	Inception-ResNet Architecture . . . . .	27
4.4	Loss Function . . . . .	29
4.4.1	Multi-Label Margin Loss . . . . .	29
4.4.2	Multi-Label Soft Margin Loss . . . . .	30
<b>5</b>	<b>Considerations and Additions</b>	<b>31</b>
5.1	Multi-Label Classification . . . . .	31
5.1.1	Transformation Step . . . . .	32
5.1.2	Excitation Step . . . . .	32
5.1.3	Inception Module . . . . .	32
5.2	Dataset Scaling . . . . .	33
5.2.1	Poor Performance on New Data . . . . .	33
5.2.2	Overfitting to New Data . . . . .	34
<b>6</b>	<b>Implementation</b>	<b>36</b>
6.1	Squeeze and Excitation Block . . . . .	36
6.2	Inception . . . . .	37
6.2.1	Channel Downscaling . . . . .	37
6.2.2	Stem Changes . . . . .	37
<b>7</b>	<b>Dataset</b>	<b>38</b>
7.1	Data Format . . . . .	38
7.2	Data Processing . . . . .	38
<b>8</b>	<b>Results</b>	<b>40</b>
8.1	Experiment Setup . . . . .	40
8.1.1	Models . . . . .	40
8.2	Accuracy . . . . .	41

8.2.1	Multi-Label Classification on Original Set . . . . .	41
8.2.2	Classification Accuracy on Streaming Set . . . . .	44
8.3	Performance . . . . .	46
8.4	Summary . . . . .	47
<b>9</b>	<b>Conclusion</b>	<b>49</b>

## List of Figures

1	Pure Breed Single Label vs. Mixed Breed Multi-Label Dogs . . . . .	8
2	Inception Module . . . . .	12
3	Conventional vs. DARTS Classifier . . . . .	14
4	RNN-CNN Model . . . . .	15
5	SE Block . . . . .	18
6	SENet Epoch Accuracies . . . . .	21
7	2 layer and 3 layer SEBlocks . . . . .	23
8	Detailed Squeeze and Excitation . . . . .	24
9	18 Layer SENet Architecture . . . . .	26
10	Inception Net Modules . . . . .	27
11	Inception Net Architecture . . . . .	28
12	Top 1 and Top 5 Validation Accuracies . . . . .	43
13	Accuracy on Original and Streaming Datasets . . . . .	45

## List of Tables

1	Accuracy of Previous Models . . . . .	21
2	Multi-Label Margin and Soft Margin Loss . . . . .	42
3	Top 1 and Top 5 Validation Accuracy . . . . .	42

4	Accuracy on Original and Streaming Sets . . . . .	44
5	Runtime of Models . . . . .	46

# 1 Introduction

ImageNet is an image dataset consisting of 14 million images annotated with category. Every year there is a classification challenge based on this dataset, ImageNet Large Scale Visual Recognition Competition (ILSVRC), in which competitors try to best categorize images into 1,000 separate object categories. The training set consists of 1.2 million images, with a 50,000 image validation set, and 100,000 test set. Since 2012, deep convolutional neural networks have dominated the challenge, kicking off the deep learning artificial intelligence boom. [7]

Research and progress related to ILSVRC has been done exclusively in increasing the top5 and top1 accuracy on classification of single-class images. The accuracy of these models has already progressed to surpass human annotation accuracy, reaching around 78% for top1 accuracy and 94% for top5 accuracy. As a result a lot of research is now focused on expanding the scope and finding different applications of image classification. For example, starting in 2018 even ILSVRC itself has changed, and is now a competition to classify 3d object models with natural language, with real world applications in robotics and autonomous navigation, two extremely popular topics in fields today and moving forward. Another field of research is in applying top performing imagenet models such as ResNet (Residual Network) in different manners, such as guaranteeing some level of accuracy and instead varying the level of specificity of classification. This direction is especially important since the amount of categories keeps growing and labels get increasingly specific and it is impractical to keep retraining entire networks to the same level of accuracy as before.

In terms of the scalability of models, the way current models are trained and utilized is very inflexible. In large part due to the curated nature of the ILSVRC competition and the ultimate goal only to increase accuracy on given sets, these CNN models are made to only perform well on known categories given in the training set. There is no optimization for how well these models can scale to increasingly larger datasets. In their current form, these networks output a tensor of predicted confidence for each classification, so essentially

a vector of dimension equal to the number of classes. A softmax over this tensor gives us the prediction distribution and then accuracy is determined if the true class is in the top 1 or 5 predicted classes. In this model, if the training set changes, the network will have to be retrained with all these new distinct categories in mind, starting again from the first epoch as now the output tensor, the distribution of predicted classes, will be a completely different dimension.

At this point, we make an important observation. The majority of new labels being added to ImageNet are simply more specific versions of existing labels, essentially subclasses. These labels can thus be represented as combinations of existing labels, and as such we should not have to completely retrain our model to accomodate for new training data which our existing trained model already has collected relevant information on.

For example, if we have a model trained on pure breeds of dogs (like we did in this project), and now would like to predict a dog of mixed breeds from the original pure breed data set, we would want to be able to just add a multi-label target for that dog, with labels for each parent breed, rather than having to add another unique breed to the set of classes and retrain the entire model. We can see an example use case in Figure 1.



**Figure 1:** From left to right: Beagle, Pug, Beagle-Pug mix (puggle). Pure breed Beagle and Pug in original training set, now adding puggle mixed breed as a multi-class image.

In my work, I created a scalable model for training and classifying images, with the capability to expand classification to subclasses with all parents in the existing training set. With this model, we are now able to train in a streaming fashion, adding on new classes and training them without having to retrain on existing data, so our runtime moving forward is



essentially independent of the size of our previous data space. We do this by implementing a modified Squeeze and Excitation network that is capable of multi-target classification. This multi-target classification retains the performance and accuracy of single-target models on traditional single-target datasets and can be trained on new multi-target data without significant loss in accuracy on previously trained data.

## 2 Related Work

### 2.1 ILSVRC Models

The ILSVRC challenge has had many very different variations of convolutional neural networks (CNN's) over its many years, and these networks have progressively built upon each other, also taking advantage of the rapid increase in hardware acceleration for machine learning, coming mainly in the form of GPU acceleration[1]. As a result, from 2012 when GPU accelerated CNN's first started winning the challenge to 2017, the accuracy has increased from around 74% to over 94%.

#### 2.1.1 AlexNet

AlexNet is a CNN that competed in ILSVRC in 2012, achieving a top-5 error of 15.3%, which was around 10% lower than the winner of the previous year[13]. This network revolutionized the way the industry thought about image classification problems. AlexNet consisted of 5 convolutional layers, some of which had max pooling layers, followed by 3 fully connected layers using non-saturating ReLU and finally a 1000 dimension softmax to get the category probabilities. The network was heavily optimized for CUDA and GPU support, as training such a large network without this GPU support would have been impractical.

#### 2.1.2 VGG Net

VGG is a network that was introduced in 2014 that used simple 3x3 convolutional, but pushes the depth to 16-19 layers [4]. Because there are so many layers, the effective size of the filters is multiplied, as 3 layers of 3x3 filters has an effective size of 7x7. However, with the reduced filter size, the number of parameters is drastically reduced and thus there is faster convergence and reduced risk of overfitting. VGGNet did particularly well in the 2014 ILSVRC, winning the localization task and placing 2nd in the classification task.

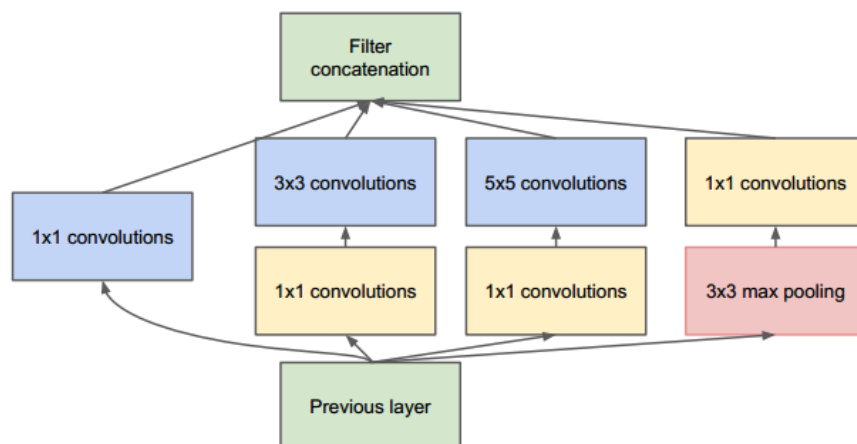
### 2.1.3 Inception Net

While VGG nets improved computational load and performance by separating out large filters with multiple smaller ones, Inception Net, also known as GoogLeNet, worked on the observation that most nodes in deep networks are actually redundant and unnecessary to learning a model[7]. For example, many times we will have layers where the size is around 512 channels, most of which will simply repeat the information correlation found in other channels. Simply pruning out channels and making a sparse network would not work since sparse multiplication is very expensive and even slower than the full dense connection.

Inception net tries instead to use each layer to identify sparse local structures, and then use these correlated areas as clusters to feed into the next layer. The building blocks behind this, Inception modules, contain varied sizes of convolutions, i.e. 1x1, 3x3, 5x5, in order to capture cluster features of different spatial size and spread. Deeper layers capture more abstract features and so larger 5x5 filters become more necessary the farther into the network we get.

In order to prevent the computational blowup of needing these larger filters, Inception Net introduces dimension reductions, which are 1x1 convolutions applied before expensive 3x3 and 5x5 convolutions. To illustrate the performance gain, imagine we have 32 5x5 filters working on an input of 128 channels, giving a computational workload of  $32 \times 5 \times 5 \times 128 = 102,400$ . Now instead if we apply 16 1x1 dimension reduction convolutions before the 5x5 filters, we get a workload of  $16 \times 128 + 32 \times 5 \times 5 \times 16 = 14,848$ , roughly one order of magnitude less.

Using these ideas, Inception net is able to achieve higher accuracy than VGGNet while also being significantly faster. A subsequent iteration of Inception net even introduced the idea of factorization, the process of replacing a large filter i.e. 5x5 with a 5x1 followed by a 1x5. This gives a workload of  $5 \times 1 + 1 \times 5 = 10$  compared to  $5 \times 5 = 25$ .



**Figure 2:** Inception Module: Each layer feeds the input from the previous layer into a mixture of 1x1, 3x3, and 5x5 convolutions. Additionally, 1x1 dimension reduction convolutions are added before the 3x3 and 5x5 filters in order to decrease the workload.

#### 2.1.4 ResNet

Residual Neural Networks (ResNet) utilize a construct known as skip connections/shortcuts in order to jump layers. ResNet won ILSVRC in 2015 with a top-5 error rate of 3.57%. One problem with VGGNet was that when increasing the network depth, accuracy got saturated and then degraded very quickly. To solve this, ResNet introduced skip connections so that instead of having a block output  $y = F(x)$ , residual blocks output  $y = F(x) + x$ . There are two types of shortcuts used in residual blocks, identity shortcuts  $y = F(x, W_i) + x$ , and projection shortcuts  $y = F(x, W_i) + W_s x$  where  $W_i$  is the weight of the normal connection, and  $W_s$  is the weight of the added residual connection [3]. It is found that projection shortcuts have very small gains, and since they add more parameters to backpropagate/train for, projection shortcuts are only used for dimension changes. These shortcuts are also useful for train own even more layers, and the most successful ResNets are on the order of 105 layers.

#### 2.1.5 Highway Networks

Highway networks were introduced in 2015 as an extension off the idea of ResNets with longer skip connections. Since deep networks are very difficult to train as it is hard to

capture information for all layers when using gradient based training. Highway Networks include connections across multiple layers (information highways), regulated by a gating function which controls the information flow, heavily inspired by the ideas from LSTM's [5]. In this way, Highway networks can be hundreds of layers and still be trained using relatively basic SGD methods.

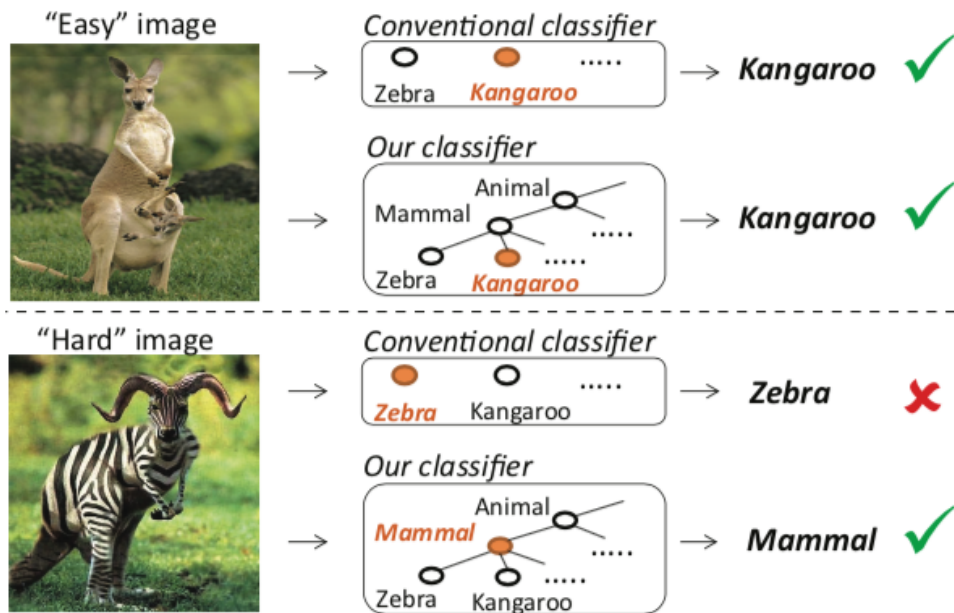
## 2.2 DARTS: Optimizing Accuracy Specificity Tradeoff

In a project with the similar goal of providing a way to scale visual recognition to growing datasets and class labels, Deng et al. created a model, Dual Accuracy Reward Trade-off Search (DARTS) that instead of trying to exactly classify each image, gives the most specific classification the model is capable of while maintaining a set level of accuracy[10]. In this approach, we build up a semantic hierarchy or tree of the classes, and the optimizer can trade specificity for accuracy when the model is uncertain in classification. For example, in our tree we have zebra and kangaroo which are both children of mammal, if our model is unable to confidently classify an image as a kangaroo or zebra, the optimizer will choose to instead just classify an image as a mammal with higher confidence and accuracy.

This approach, while it does account for future scalability in terms of just giving broad, non-specific classifications of new images, does not actually scale well with additional data, and is actually even more difficult to properly scale, since both the CNN model and hierarchy optimizer must be retrained for new classes.

## 2.3 Multi-Label Image Classification (MLIC)

Multi-label image classification is a growing field of study, and it is fundamental to visual understanding. As humans, we don't group every object we see into a set amount of equal and distinct categories, each category completely unrelated to the other, but rather we have a set of base categories that we gradually grow and expand, so that any object can be categorized in one specific category, but also as a mix of multiple other related categories.



**Figure 3:** Conventional vs. DARTS classifier on easy and hard images. We can see that in the hard image, instead of incorrectly classifying the image as the conventional variant does, the DARTS classifier chooses to classify the image at one level of specificity less.

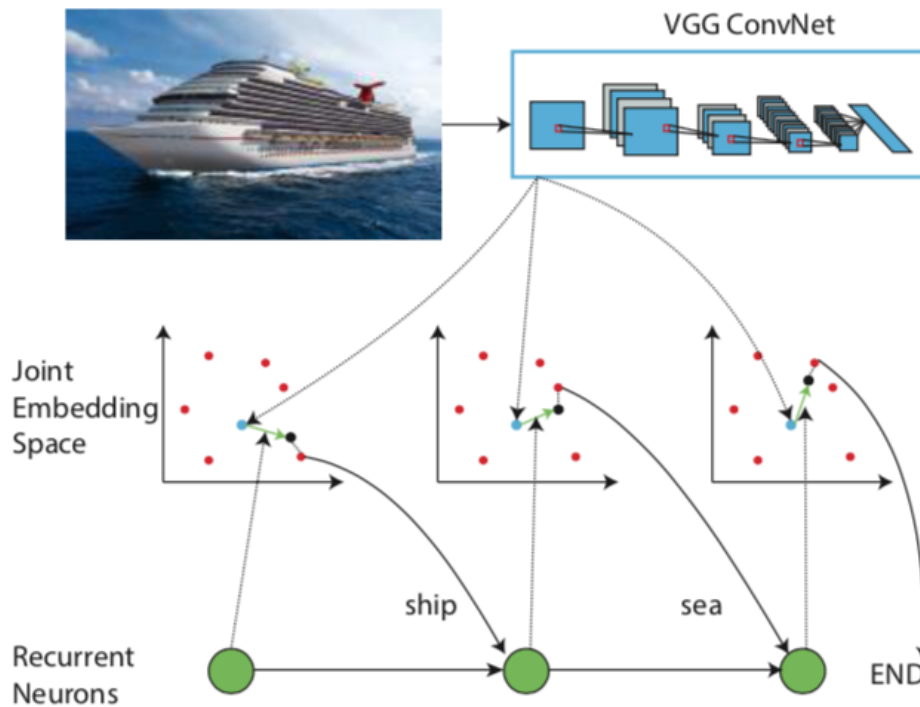
Multi-class recognition is by nature a more challenging task than single-class recognition, as there are more variables to optimize loss for, and currently there is a growing amount of multi-class research. However, the field is not nearly as saturated single-class classification, and there currently is no definitive or standard best approach to the problem.

### 2.3.1 CNN-RNN

Common approaches for MLIC are to simply extend single-class CNN’s by transforming the task into multiple single-label problems, i.e. including each multi-label example twice, once with each label. However this approach fails to capture the inherent dependency between labels, as there are very strong co-occurrence dependencies such as water and ocean. In order to solve this issue, other methods try to model the dependencies with pairwise compatibility probabilities and Markov random fields to infer joint probabilities, but this is impractical to do on all pairs of labels of a large dataset.

CNN-RNN proposes a recurrent neural network with convolutional layers, a CNN and

RNN combined as the name suggests [6]. The RNN functions to capture label relationships through its multiple recurrences, as the image embeddings are trained in the same space as the label embeddings, so at each step the output of the recurrent neuron is combined with the image embedding to find the probability of matching to a label embedding. Through the steps the label embeddings are also moved accordingly, modeling the label co-occurrence dependency. Additionally, this model is able to use the RNN framework to adapt the attentional area of the CNN in each step to focus on different regions in the image, so if there is a ship and a sea, the model is able to classify both, while also classifying ocean, due to the shared label dependency with sea. Since it is not optimizing probabilities, this CNN-RNN method is computationally tractable unlike the previous Markov based models.



**Figure 4:** RNN-CNN model[6]. The red points are label embeddings, while the blue point is the image embedding. The black point is the sum of the image embedding and the output of the recurrent neuron at that step, and thus the predicted label would be the closest blue point to the black point. The predicted label is then fed into the next neuron and a new classification is made

This CNN-RNN approach is nice and definitely captures more information than tradi-

tional CNN approaches, which cannot capture the label dependencies, however due to its complex nature it is not very practical to train and use.

### **2.3.2 Weakly Supervised Detection**

Weakly supervised detection for image recognition is typically a process in which a picture is annotated with a set of labels, but the bounding box annotations associated with each of the labels is not given, and it is the job of the WSD model to recognize the semantic Regions. Liu et al. combined a WSD model with a classification model using knowledge distillation, a process in which a large teacher network, the WSD model, teaches a smaller student network, the classification model[11]. In this case, the WSD provides the semantic bounding box regions with proper labels to the classification model, whose objective is to learn the mutual class dependencies in the image. Eventually, the student model will improve to a point where it does not need the WSD teacher model anymore.

## **2.4 Transfer Learning From Imagenet**

Kornblith et al. performed a study on how well ImageNet performance transfers to other vision tasks[9]. While most image classification research is aimed at improving the accuracy on a specific set of data, it is implicitly assumed that achieving good performance on the specified task and data generalizes to good performance on other data and image tasks. The study found that, with ImageNet trained Models, there is a strong correlation between the performance on ImageNet and the performance on other tasks, transfer accuracy. However one key finding is that many fine-tuning techniques used to improve accuracy on ImageNet such as regularization, even in slight quantities, throw off this correlation to transfer accuracy and indeed hurt transfer performance.

Building off this knowledge, Recht et al. attempted to train current models on data based on ImageNet and CIFAR-10, and see how well these newly trained models performed and generalized. The authors copied the data curation process and applied it to a larger image



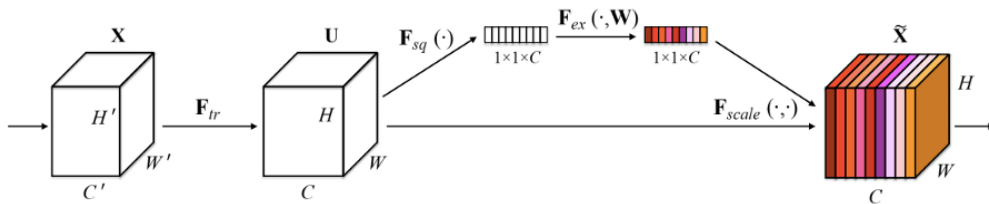
space in order to build their new unbiased datasets. It was found that there is a significant drop in accuracy when training the same models on this new data, around 10 – 15%, and since model performance differences are preserved (higher performing models on ImageNet will still perform better on this new dataset). Interestingly it is also found that better performing models in the original dataset experienced a lower proportional drop in accuracy when trained on the new dataset. This suggests that higher performance models are not overfitted and over adapted to their training set, but rather even more robust than their lower performing counterparts. Since the drop in accuracy is not from over-adaptivity then, the authors claim that the drop is from human annotation being biased towards easier images, and thus original models are unable to generalize to "harder" images found in the real world.

### 3 Squeeze and Excitation Network

For our model, we will be using a network known as a Squeeze and Excitation Network (SENet) which is similar to ResNet, but instead of focusing on the spatial aspect, as ResNet and many previous architectures do, SENet introduces a new aspect, the inter channel relationship. This network uses Squeeze and Excitation block as the architectural unit, which attempts to explicitly model the channel dependencies in order to perform feature recalibration, in which we use channel information in order select for features on each channel. [8]

There are three steps in these blocks, first a transformation of spatial and channel dimension, and then a squeeze step that produces a channel descriptor by combining the feature maps of each channel. Once we have this embedding of the global distribution of features, we perform an excite step, which is essentially just a gating step that takes the channel descriptor embedding and produces a set of weights for each channel in order to modulate the features of that channel.

SE blocks can be used to replace any traditional residual block one for one, and can be stacked in order to create and SE network. In earlier layers, SE blocks serve to filter desired features regardless of classification/category, while SE blocks in later layers are much more specialized, filtering in a very class specific manner.



**Figure 5:** SE block with transformation  $F_{tr}$ , squeeze  $F_{sq}$ , excitation  $F_{ex}$ , and scaling  $F_{scale}$

### 3.1 Channel Transformation

For each SE block, we first need to transform an input  $X \in R^{H' \times W' \times C'}$  to an output in a different space,  $U \in R^{H \times W \times C}$ , where  $H \times W$  is the height and width, and  $C$  is the number of channels. Now let us say we have  $V = [v_1, v_2, \dots, v_C]$  ( $C$  here is the number of channels in the target output) be the set of learned filter kernels, and the layer would have outputs,  $U = [u_1, u_2, \dots, u_C]$  where

$$u_c = v_c * X = \sum_{s=1}^{C'} v_c^s * x^s$$

where  $*$  denotes convolution,  $X^s$  represents the  $H' \times W'$  spatial input of channel  $s$ , and  $v_c^s$  is the selected 2D filter of specified kernel size for that channel. [8] More simply, each channel  $c \in C$  in the output is calculated by applying a filter  $v_c^s$  (there is a separate 2D filter for each output channel/input channel combination) to each each channel  $s \in C'$  of the input and summing the convolutions across the  $C'$  channels. Since we are summing across channels, we can see that  $v_c$  contains information on the input channel dependencies, but since we are applying the filters to each of the output channels independently, we still preserve the local spatial relationship of the input. This lets us make sure that the network remains sensitive to local features even as we keep transforming the dimensions/channels.

### 3.2 Squeeze

In the previous calculation of output  $U$ , we saw that each of the channel filters  $v_c$  operated on a local receptive field and thus the output  $U$  contained the entangled spatial and channel information of  $X$ , but did not have the inter-channel information of itself that we want. Thus, we perform a squeeze step in which we combine the global spatial information of  $U$  into a channel descriptor. In order to do this, we do a global average pooling on the spatial output of each channel, getting a descriptor we denote  $z \in R^C$  where  $C$  is the number of channels. Formally, we get  $Z$  by collapsing  $U$  by its spatial dimension  $H \times W$ , where each

channel  $z_c$  is given by

$$z_c = F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j)$$

Another way of looking at this is that each channel of  $U$  was a collection of local descriptors of  $X$ , and by average pooling each of the channels, we now have a statistic for each channel that is expressive of the whole image of  $X$ .

### 3.3 Excitation

Now that we have the statistic from the squeeze operation, we can now capture the full channel-wide dependencies. This function should be able to learn a complex, nonlinear relationship between the channels, and must not become a mutually exclusive relationship that only depends on one channel at a time. We can do so using a gating function with sigmoid activation as follows,

$$s = F_{ex}(z, W) = \sigma(W_2 ReLU(W_1 z))$$

where  $W_1 \in R^{C/r \times C}$  and  $W_2 \in R^{C \times C/r}$  for some reduction ratio  $r$ , thus giving us a scalar  $s$  representing the gating value. [8] The reason we parametrize  $W$  into  $W_1, W_2$  is in order to prevent our gating mechanism from overfitting, as we make a bottleneck of 2 fully connected layers: a dimensionality reduction layer applied to  $z$  using  $W_1$ , then apply a ReLU, and then apply a dimensionality increasing layer using  $W_2$ .

Now that we have a scalar  $s_c$  for each of the output channels  $c$ , we can apply these scalar weights to the output  $U$  we got after the transformation, giving

$$y_c = F_{scale}(u_c, s_c) = s_c \cdot u_c$$

and thus we now have an output  $Y \in R^{H \times W \times C}$  that encapsulates the spatial and channel interdependencies of the original input  $X$  of different channel dimension.

### 3.4 Comparison to Related Models

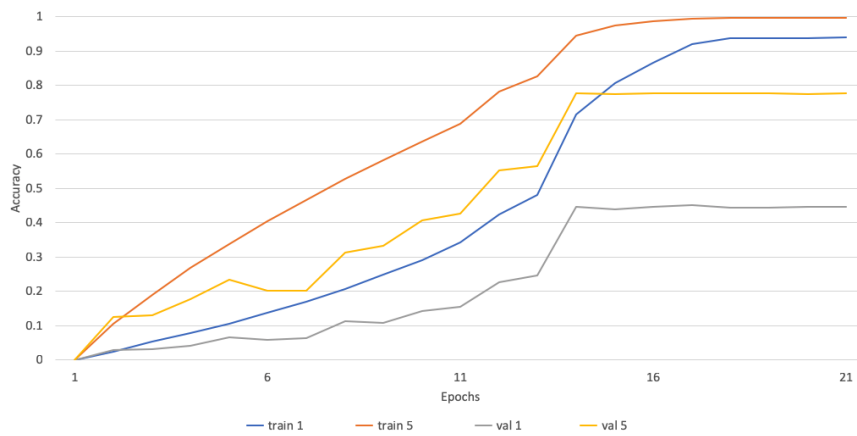
The great thing about SE blocks is that we can keep the general architecture of ResNets and other CNN’s, we simply substitute in SE blocks for residual blocks. Thus we can use a relatively standard 18 layer SE ResNet, with all blocks being SE blocks.

We tested normal single category classification (top1 and top5) accuracy of popular models to get a baseline for comparison later. We trained SENet for 15 epochs, the final top1 and top5 validation accuracies are shown in Table 1, as compared to standard ResNet and VGGNet trained over the same number of epochs.

**Table 1:** Accuracy of Previous Models

	Top1 Acc	Top5 Acc
SENet18	44.7%	79.7%
ResNet18	42.1%	75.8%
VGGNet16	41.3%	73.6%

We can see then that the squeeze and excitation blocks do really well at helping to progress training while improving validation accuracy, as there is a 2% and 4% increase in top1 and top5 validation accuracy, respectively, over ResNet18, which is identical architecturally. In addition to this, we hypothesize that a lot of inter-class relationships can be captured by the extra information gain in the squeeze step, especially in later layers where channel expansion is more pronounced.



**Figure 6:** Epoch accuracies for training and validation on standard 18 Layer SE Network

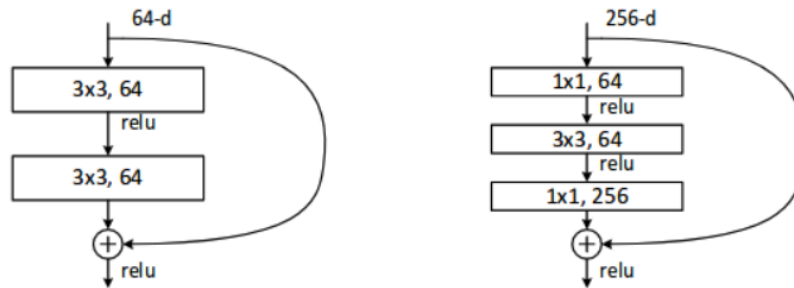
Looking at a graph of the training and validation error over epochs in Figure 6, we can see that overfitting seems to become more of a problem as the epochs progress. Due to our limited resources, we can only train and run 18 layer SENets in our research, however comparing with standard implementations of SENet18 and SENet152 indicate that increasing the layers to something like 152 should allow us to reach top5 accuracy around 95% and top1 accuracy around 80%.

## 4 Model

In this section I will go over some of the basic structures and architectural variations of the model that we will be implementing. The point of this research is to develop a model that is as simple as possible while achieving our scalability goals, so that current models can be moved over onto this framework as easily as possible and previous and concurrent research ideas do not need to be drastically modified to be applied on top of this model.

### 4.1 SEBlock

Each Squeeze and Excitation Block (SEBlock) is a multi-layer computational unit that performs a squeeze and excitation as described in Section 3. There are two layer and three layer SEBlocks as shown in Figure 7, but both variants follow the same basic idea.

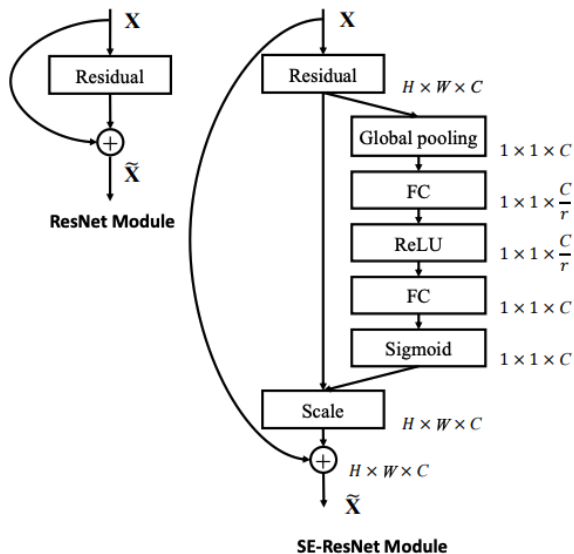


**Figure 7:** 2 layer and 3 layer SEBlocks: 2 layer blocks consist of two 3x3 filters, 3 layer blocks consist of a 1x1 filter followed by a 3x3 followed by a 1x1.

#### 4.1.1 2 Layer SEBlock

The 2 layer SEBlock consists of two convolutional blocks with 3x3 filters. When computing the output of these blocks, we perform the first convolution, apply batch normalization and ReLU to the result, then feed it into the second convolution, and then apply batch normalization. However, now we perform the squeeze step shown in Figure 8, in which we average pool each channel, giving us an output with size equal to the number of channels. We

then do the excitation, in which we apply a ReLu and Sigmoid gating, and then weight the original output from the convolution with this new descriptor. We then add this full output to the residual, the input from before the 2 convolutions, which comes from the shortcut pathway.



**Figure 8:** Detailed Squeeze and Excitation: After getting the residual from the normal convolutions as we would in a typical ResNet, we perform a squeeze with a Global pooling, and then an excitation with a ReLu and Sigmoid gating.

### 4.1.2 3 Layer SEBlock

The 3 layer SEBlock functions in much the same way, except now we have 3 convolutional blocks. The first is a 1x1 filter with the given number of planes, the second is 3x3 with the given number of planes, and the third is 1x1 again but now with 4 times the planes. The squeeze step thus now gives back 4 times the channel descriptors, which we then weight the output and add to the residual in the excite step as before.

## 4.2 ResNet Architecture

For the overall layer architecture of our model, we used a traditional 18 layer ResNet model as our base, in order to make this model as familiar and easily expandable as possible. We



replace all ResNet blocks with SENet blocks, and since this is only an 18 layer SENet, we only use SENet blocks that are 2 layers deep. If this were a 50 layer or 152 layer model, we would use 3 layer SENet blocks.

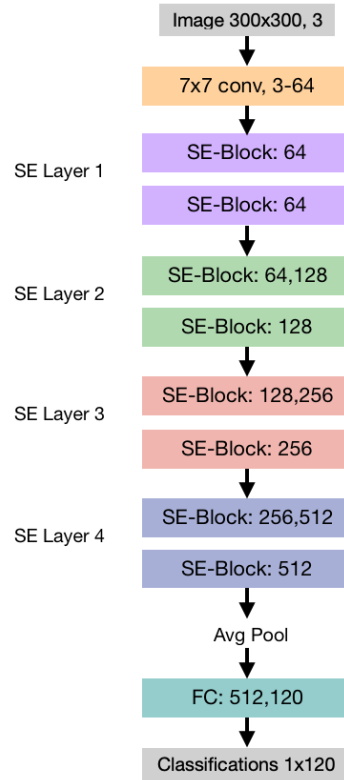
The general layout of our network, no matter how many layers, will always follow this breakdown which can also be seen in Figure 9.

1. **Standard Convolutional Layer** with filter size 7 and stride 2, that takes in 3 planes (3 RGB channels) and outputs a 64 plane convolution
2. **SELayer 1:** Some given number of SEBlocks, each 2 or 3 layer, with stride 1. This layer takes in the 64 plane convolution and gives a 64 plane output.
3. **SELayer 2:** Some given number of SEBlocks, each 2 or 3 layer, with stride 2. This layer takes in the 64 plane convolution and gives a 128 plane output.
4. **SELayer 3:** Some given number of SEBlocks, each 2 or 3 layer, with stride 2. This layer takes in the 128 plane convolution and gives a 256 plane output.
5. **SELayer 4:** Some given number of SEBlocks, each 2 or 3 layer, with stride 2. This layer takes in the 256 plane convolution and gives a 512 plane output.
6. **Fully Connected Layer** that turns the 512 plane output of the final SELayer into an output with dimension equal to the number of classes, this is the distribution of predicted classes. We perform a sigmoid function to scale these predictions as opposed to the softmax that would be applied in the case of single-label classification.

Note that the first block in each SELayer that expands the planes (i.e. 64 to 128) does the expansion and all subsequent blocks maintain the planes (i.e. 128 to 128).

#### 4.2.1 18 Layer SENet

In our 18 layer network, we have the first 3-64 convolutional layer, 2 two layer 64-64 SEBlock2 for SELayer 1, 2 two layer 64-128 SEBlock2 for SELayer 2, 2 two layer 128-256 SEBlock2 for



**Figure 9:** 18 Layer SENet Architecture The 18 layer SENet contains 4 SE Layers, each containing two 2 layer SEBlocks, which can be seen in Figures 7 and 8. The SEBlocks each contain two 3x3 convolutional filters, giving us a total of 18 layers when including the initial 7x7 filter and final fully connected layer.

SELayer 3, 2 two layer 256-512 SEBlock2 for SELayer 4, and finally the fully connected layer. That is 1 convolutional layer, 4 SELayers of two 2 layer blocks each, and 1 fully connected layer, giving us the 18 total layers in the end.

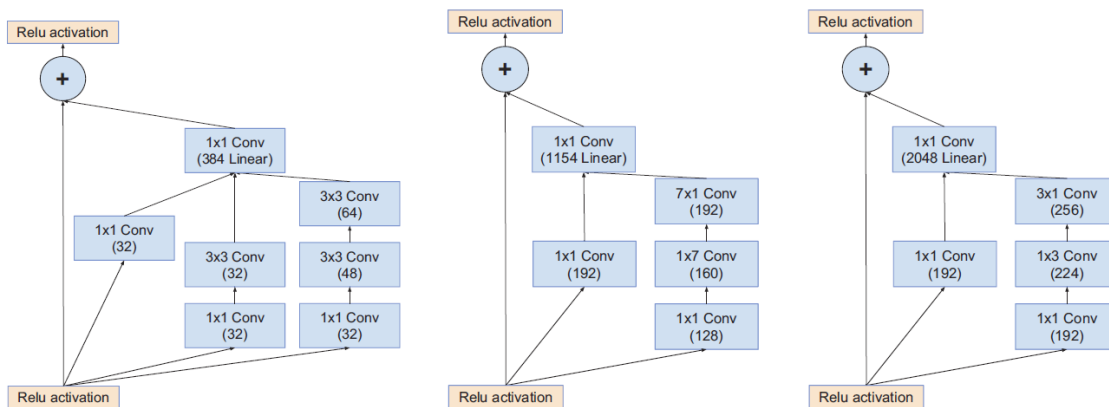
#### 4.2.2 152 Layer SENet

On the other hand, a 152 layer network would have the first convolutional layer and last fully connected layer, 3 three layer SEBlock-3 for SELayer 1, 8 three layer SEBlock-3 for SELayer 2, 36 three layer SEBlock-3 for SELayer 3, and 3 three layer SEBlock-3 for SELayer 4. The plane expansion in this network is multiplied by 4, so the SELayers 1 to 4 would now be 256-256, 256-512, 512-1024, 1024-2048, respectively. The reason we only want three 256-256

and three 1024-2048 SE-Block-3 is that the first SELayer blocks do not have enough planes to need many layers to capture the information and the last SELayer blocks have too many planes, and the information capture is miniscule in comparison to the 2 middle layers, and thus we want the majority of work to be done in 512-1024 SE-Layer 3, followed by 256-512 SELayer 2.

### 4.3 Inception-ResNet Architecture

Inception Resnets are just what the name implies, Residual networks mixed with the ideas of Inception networks[2]. A Inception Resnet consists of 3 main component layers, each made of a slightly different inception module, aptly named Inception-A, Inception-B, Inception-C. Each of these module layers is like a residual layer in that it adds the residual to the output of some convolution, but in this case the convolution is the combination of multiple branches of filters of various sizes. These modules can be seen in Figure 10.

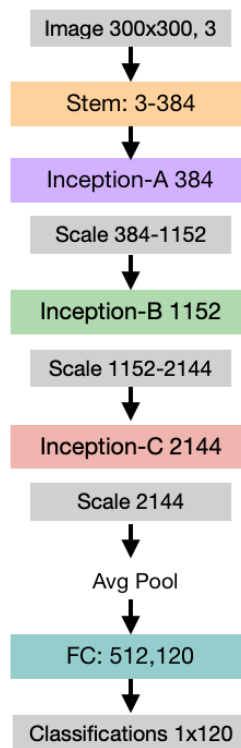


**Figure 10:** Inception Net Modules: Inception-A, Inception-B, Inception-C from left to right, respectively. Inception-A contains branch 1 of 1x1, branch 2 of 3x3, and branch 3 of two 3x3's. Inception-B contains branch 1 of 1x1, branch 2 of 7x1 and 1x7 (factorized 7x7). Inception-C contains branch 1 of 1x1, and branch 2 of 3x1 and 1x3 (factorized 3x3). Note that the channels increase from Inception-A to Inception-C.

In Figure 10, we can see that each branch with a relatively large filter (i.e. 3x3) has a 1x1 dimension reducing filter before in order to control the computational workload. In addition, large convolutional filters are factorized into smaller parts, i.e. a 7x7 filter is factorized into

a 7x1 and a 1x7. Putting these smaller serial filters is much cheaper than the larger filter yet still captures the same spatial features.

Now, to put all the modules together, we have a general structure of a stem section, Inception-A section, Inception-B section, Inception-C section, and a fully connected section. Between each of these sections is a scaling layer, in order to transfer between the different channel dimensions of the inception sections. We can see an overview of the architecture in Figure 11.



**Figure 11:** Inception Net Architecture: Inception Net is broken into a stem, 3 inception layers, A,B,C, and finally a fully connected layer

A detailed breakdown of these five sections is given below.

1. **Stem:** Takes in 3 channel (RGB) input image (300 x 300 pixels) and applies standard convolutions (including 7 x 7 filter). Outputs 384 channels.
2. **Inception A:** Takes in 384 channel input and applies 4 layers of Inception-A modules.

Output channel width is tripled to 1152 with scaling.

3. **Inception B:** Takes in 1152 channel input and applies 7 layers of Inception-B modules.

Output channel width is increased to 2144 with scaling.

4. **Inception C:** Takes in 2144 channel input and applies 3 layers of Inception-C modules.

Output channel width is kept at 2144, and then average pooling is applied in order to bring this down to 512.

5. **Fully Connected:** Fully connected layer of 512 input channels, and number of classes as the output dimension (120 in our case).

## 4.4 Loss Function

For our loss function, we tested two possible functions, Multi-label margin loss based on hinge loss, and Multi-label soft margin loss based on cross entropy loss. Both loss functions are essentially based on their single-label loss counterparts, but just adapted to sum across the multiple components

### 4.4.1 Multi-Label Margin Loss

We define  $X$  to be the 2d output tensor of our network, with dimension  $B \times C$  where  $B$  is the batch size and  $C$  is the number of classes. Then let  $Y$  be the 2d target tensor of the same dimension, a one hot tensor with 1's at each target class index and 0's at every other index.

The multi-class hinge loss for this becomes,

$$loss(x, y) = \sum_{ij} \frac{max(0, 1 - (y[j] - x[i]))}{|x[0]|}$$

which is essentially the average of hinge loss summing all incorrect predictions in  $X$ , whether false positive or false negative.

A small note is that in PyTorch, the *MultiLabelMarginLoss* function does not take in a one hot target tensor, but rather a tensor in which the target class indices are given in the first contiguous block and all subsequent indexes are -1. For example, if we had a 4 class distribution and had a target of class 0 and class 2, our one hot vector would be  $Y = \langle 1, 0, 1, 0 \rangle$ , but our vector for pytorch would be  $Y = \langle 0, 2, -1, -1 \rangle$ . The equation for the loss function also becomes

$$loss(x, y) = \sum_{ij} \frac{\max(0, 1 - (x[y[j]] - x[i]))}{|x[0]|}$$

in order to account for this change.

#### 4.4.2 Multi-Label Soft Margin Loss

We define output  $X$  and target  $y$  to be two tensors of size  $B \times C$ , where  $B$  is the batch size and  $C$  is the number of classes.  $Y$  is a one hot target vector across classes for each example in the batch, with 1's at the index of each target class. Normal single target binary cross entropy loss is given by

$$loss_{CE}(x, y) = -(y \log(x) + (1 - y)(\log(1 - x)))$$

The multi-class cross entropy loss is thus given as an average of cross entropy across the multiple classes, with an additional sigmoid gating on each prediction to help scale the losses better.

$$loss_{MCE}(x, y) = -\frac{1}{C} \sum_i y[i] \log \frac{1}{1 + \exp -x[i]} + (1 - y[i]) \log \frac{\exp -x[i]}{1 + \exp -x[i]}$$

The advantage of a cross entropy based loss function over the previous hinge loss based function is that convergence should occur quicker since the loss will get exponentially larger the farther from target we are so initial steps in training will move much faster.

## 5 Considerations and Additions

In this section I will describe in detail some of the key considerations and obstacles we must be aware of, as these are the points we must optimize for in order to reach our goals scalability performance while still maintaining acceptable accuracy performance.

### 5.1 Multi-Label Classification

In terms of the actual multi-label image classification (MLIC), since we are using pretty standard models that were original made for single-label classification, it is not a very simple task to adapt them to multi-label tasks while still maintaining accuracy and computational tractability. There are two fields of MLIC that models usually optimize for: first to learn class correlations, i.e. sea and blue; and second to be able to predict classes in disjoint segments of an image, such as an image with two dogs of different breed. For the scope of our research, we will be focusing on the first point, class correlations, since disjoint segments are not a huge concern for model scalability. A simple solution to this would be to use current bounding box detection models, which are very accurate, and just divide the image into multiple new images to train on.

In order to learn correlations between classes, it is obvious that we must make a model that is capable of capturing more information, in order to learn the label embeddings. Two very straightforward ways of doing this are to increase the number of layers in the network (i.e. 18 to 152), or increase the plane/channel expansion in the network (i.e. 64 to 128). Apart from being obvious, these methods are relatively brute force and require increasing computation power of hardware, defeating the whole objective of finding a scalable model for image classification. Also, we are operating under the assumption that we are already testing on the state of the art model (152 layer) and expanding layers beyond this either hurts performance or has no significant effect at all.

### 5.1.1 Transformation Step

This is where the idea behind squeeze and excitation comes into play. SE blocks have marginal work increase over the standard residual blocks, but the information gain is huge. If we increase the channel expansion in the transformation step  $F_{tr}$ , the squeeze step  $F_{sq}$  should be able to capture more granular information and could help us learn the inter-class dependencies. Additionally, due to SIMD (Single Instruction Multiple Data) parallelism, the average pooling on more channels scales relatively well, so increasing the channel expansion in the squeeze could give us this increased information gain at a relatively low computational cost. We can even average channels together in the squeeze step, so that our expanded channels from the transformation step only affects the squeeze step computation, so the excitation step and subsequent steps have the same channel size and computational workload, but the squeeze was still able to capture more information.

### 5.1.2 Excitation Step

In the excitation step, when gating the channel descriptors, we apply a dimensionality reduction with  $W_1$  before applying ReLU, and then a dimensionality increase with  $W_2$  before the sigmoid activation. This was done in order to prevent the ReLU gating function from overfitting, however now that we are performing a multi-label classification task, it is inherently harder to overfit. As a result, we can lower the reduction ratio, and this will help us maintain more of the information gain contained in the channel descriptors. When doing this however, we must keep careful watch of the training to validation difference to make sure that we are not overfitting since it is still a significant concern.

### 5.1.3 Inception Module

In addition to changes we can make in the SE blocks, we can also implement the idea of combining varying filter sizes from Inception net. In multi-label classification, it would be very advantageous to allow earlier layers to learn the more obvious localized features, and



allow later layers to try to learn the more higher level abstract features, which is likely where the class dependency relation will be learned.

## 5.2 Dataset Scaling

In this section, we will discuss the concerns with the streaming aspect of training the model. Since we are using streaming approach, we will never be retraining on older datasets, and thus there is a chance that we overtrain our model on newly streamed data, and lose performance on old data. On the other hand there is also a chance that we put too little weight on new training examples, and can never reach acceptable accuracy on the new examples. This is a very delicate tradeoff that our model must seek to optimize.

### 5.2.1 Poor Performance on New Data

One of the key advantages of a streaming approach to image classification is that since the label space is staying constant, the more data we add and stream training, the more information on the label space we should get, and eventually the faster training should get, assuming that our model is large enough to handle the inherent entropy of the data and label correlations.

Since the label space is constant, and the images being streamed should be comparable to the images in the original dataset, we should not have to worry too much about the weights not being able to change enough to fit the new model. Our channel width for both the layers and SE blocks should be wide enough if we were able to achieve good accuracy on the original images, but we can add varying filter sizes and pool the result like Inception net if we are not getting fast enough training on streamed data. This should just help us accelerate training on the streamed set as we will be able to better separate out local differences in the earlier layers, which should be closely matched in both original and streamed sets, and rather have the majority of learning occur for high level class dependencies in later layers.

### 5.2.2 Overfitting to New Data

Probably the main concern with a streaming approach is the degradation of performance on previous data as we train on new data. While an important half of the problem is to achieve high accuracy on new data, an arguably more important half to the problem is maintaining that performance on existing data moving forward. Otherwise it would almost be the same as training a completely new model, focused purely on the new data set.

One simple method to help prevent overfitting is introducing dropout layers in the network. Dropout layers choose a random set of neurons to deactivate during training, so that the network is forced to learn a more general set of features. The set of deactivated neurons changes randomly each batch, so all neurons are trained on the same space. By not relying on the complete set of neurons, we avoid the risk of using the higher dimensionality available to overfit on the niche biases and features of the training set. When performing validation however, dropout rate is set to 0 meaning all neurons are used.

Another method that is usually used along with the dropout is regularization, such as L1 and L2 regularization. Regularization adds a term to the loss function based on the magnitudes of the current weights, so that we essentially bias the weights against changing too drastically when updating weights.

L1, or LASSO regression, is the most intuitive approach, summing the absolute value of all weights and adding this sum as a penalty to the loss function. For a loss function based on least squared error, the L1 regularization can be given by

$$loss_{L1} = \sum_{n=1}^N (y_n - w \cdot \phi(x_n))^2 + \lambda \sum_{m=1}^M |w_m|$$

where  $y$  is the true label,  $w$  is the vector of weights,  $\phi(x_n)$  is the predicted label, and  $\lambda$  is the regularization term.

L2, or ridge regression, is similar to L1 regularization, but instead of summing the absolute value of weights, it adds the squared sum of weights as the penalty term to the loss

function. For a least squared error loss function just like before, the L2 regularization can be given by

$$loss_{L1} = \sum_{n=1}^N (y_n - w \cdot \phi(x_n))^2 + \lambda \sum_{m=1}^M w_m^2$$

where  $y$  is the true label,  $w$  is the vector of weights,  $\phi(x_n)$  is the predicted label, and  $\lambda$  is the regularization term.

Additionally, we can apply Inception net ideas of dimension reduction, adding 1x1 convolutions before our larger filters, in order to prevent overfitting, as these filters will now essentially be trying to learn on a compressed set of features. We can also apply factorization, separating large filters such as 3x3 into two smaller ones, 3x1 and 1x3, which still preserves input and output dimensions, but compresses the amount of features we are able to learn.

## 6 Implementation

As described in the architecture section, our base model and benchmark will be the standard 18 layer ResNet. In this section, we will discuss the various additions and modifications that we will apply to our model in order to make it suitable for multi-label image classification and scalability.

### 6.1 Squeeze and Excitation Block

In our Squeeze and Excitation block, we have two additions we want to implement and test. The first is to increase the channel expansion in the transformation before the squeeze and the second is to try to lower the reduction ratio in the excitation step.

Normally in implementations of SENet, the transformation step is not used, and the output of the last convolution before the squeeze determines the channels used throughout the squeeze and excitation. Now if we changed the expansion of the original convolution to increase the size of the output channel this would be the same as just increasing the channel width of the entire layer. Instead we add a 1x1 convolutional filter with some specified expansion rate of the transformation, which is inexpensive and transforms our number of channels. The increase in number of channels has marginal effect on our average pooling squeeze and then excitation since these are relatively inexpensive operations compared to the convolutions in the first place.

For the reduction ratio, we try both lowering the ratio across the board, but also having an adaptive ratio. Most of our overfitting concerns will come in later layers when the number of channels is high. While a constant ratio does scale linearly with channel size, having an adaptive rate such as  $\sqrt{c}$  or  $\frac{c}{\log c}$  will ensure that the larger channel input will be scaled down more, to a reduced dimension of  $\sqrt{c}$  and  $\log c$  respectively.

## 6.2 Inception

For our inception addition, we implemented an Inception-ResNet model based loosely on the Inception-Resnet-v2 model. We use the same architecture of layers of 3 different blocks, Inception-A, Inception-B, and Inception-C, respectively. However, in order to make the model computationally tractable for our resources, we downscaled the model.

### 6.2.1 Channel Downscaling

First, we decreased the channels in all layers by roughly a factor of 4, proportionally scaling channels of intermediary reduction/expansion convolutions. The standard Inception-Resnet had 3 inception layers of 384, 1152, and 2144 channels. Our new model uses 96, 288, and 576 channels. Additionally, we constrain the number of layers to 2 layers in each inception group (A,B,C) in order to make this an analogous comparison to our implemented 18 layer ResNet and SENet.

### 6.2.2 Stem Changes

We also changed the initial stem of the Inception Resnet to be more like the normal Resnet initial convolutions. This is a convolution of filter size 7 and stride 2, followed by a ReLU and maxpool, and then fed into an SEBlock. Based on the channels we used, the initial 7x7 filter expanded channels from 3 to 64, and then the SEBlock would expand from 64 to 96. Then we would have an input vector that could be fed into our three inception layers.

We tested this stem against a traditional inception stem if we had just scaled down the filter sizes in the inception stem. We found that performance with the simplified SEBlock stem was significantly better in both training time and accuracy, which is most likely due to the fact that the stem structure is inherently tied to the channel dimensions, and many of the layers are unnecessary if we are not trying to expand to so many dimensions anyway.

## 7 Dataset

For our model, in order to mimic real life use cases, we want a large original dataset, trained on single-label images, and a smaller multi-label dataset to perform streaming training on. The multi-label dataset must contain mixtures of labels that are in the original single-label dataset.

### 7.1 Data Format

We decided to use dog images for our dataset, with the labels being dog breeds. For our multi-label images then, we have two choices, mixed breed dog images where the labels are each of the pure breed parents, and images with 2 different breeds of dogs, to test our image segmentation accuracy.

For our large training set, we used the Stanford Dogs Dataset, a set of 20,580 annotated images of pure breed dogs, taken from 120 breeds (roughly 170 dogs for each breed)[12]. Since the images in this dataset are of non uniform size, we re-sized all images to pixel size  $300 \times 300$ , which is also necessary in order to batch images together to improve training efficiency.

In order to get our mixed breed set, we wrote a scraper to download dog images of multiple mixed breeds of dogs that we chose, choosing mixed breeds in which the visual characteristics of the parents were apparent as this would not be applicable to real world use cases otherwise, nor would the performance be very good. In total, our small mixed breed set was a collection of 10 mixed breeds, with around 100 images for each breed.

### 7.2 Data Processing

Our data preprocessing was relatively simple. To curate our multi-label set, we used python and built in APIs in order to scrape images down from various searches, then filtered examples by hand. Then we ran all images through a python script to resize/rescale all images to

300x300 without changing aspect ratio, padding all dimensions that were below 300 pixels but above 250 pixels, and discarding any image with a dimension below 250 pixels.

Below are some examples of images from the Stanford Dogs dataset as well as from our scraping.

## 8 Results

In this section we will discuss the setup and results of our experiment. We will be evaluating our models in two metrics: accuracy achieved between the original and new datasets, and computational performance (training time) on both.

### 8.1 Experiment Setup

For our experiment, we implemented all models in PyTorch, training all from scratch. When comparing to pretrained models for ResNet, our own trained model had similar performance, around 1 – 2% less for top 5 accuracy and 0.5 – 1% less for top 1. We used a batch size of 100 and a train/test split of 0.8 training, 0.2 test of our 20,580 original images as well as our additional 500 images. All models were trained for 15 epochs on the single label original set for uniformity of benchmarking, and all seemed to be converging at this point. We then trained for 4 epochs on the streaming set after training converged and finished on the original set.

All our training was performed on Amazon AWS p2.xlarge servers, which contained a NVIDIA Tesla K80 accelerator along with two NVIDIA GK210 GPUs for a total of 12 GB of ecc VRAM, and 2496 cuda cores. The CPU was a Intel Xeon E5-2686 v4 (Broadwell), using 4 vCPU cores and 61 GB of ecc memory.

#### 8.1.1 Models

For our experiment, we tested three main architectures, ResNet, SENet, and Inception-Resnet. For Inception-resnet, we used our modified 18 layer network in order to have a meaningful comparison. Specifically, we only really compared against our version utilizing SEBlocks, since the traditional scaled down Inception-Resnet did not have close to comparable accuracy due to the significant decrease in channels without changing overall structure.

For our ResNet, we used a standard PyTorch implementation, which has one 3 to 64



channel initial 7x7 SE convolutional layer, then three sets of layers, 64 to 128 channel, 128 to 256 channel, and 256 to 512 channel, each containing two 3x3 layers. Finally we have a fully connected sigmoid layer that takes the 512 channels and gives us an output with dimension of the number of classes, 120 in our case.

For our SENet, we used the same ResNet structure, and replaced each residual block with an SEBlock that had an extra squeeze and excitation layer that was added onto the residual at the end of the block. We tried 1x1 channel expansions of 2x and 4x for our our transform expansion before the squeeze and we tried reduction ratios of powers of 8, 16 (default), 32, and square root of channel size.

For our Inception-resnet, we implemented our described model, with a similar initial 3 to 64 channel 7x7 SE convolutional layer stem, and then two each of A,B,C layers: 96, 288, and 576 channel, respectively. Since this model is still more computationally expensive than our basic 18 layer ResNet/SENet, we also tested this on 48, 144, 288 channels.

For all models, we tested both multi label margin loss as well as multi label soft margin loss. While we used a learning rate scheduler (PyTorch ReduceLROnPlateau), we adjusted learning rate manually between epochs when training got stuck. We found it best to run a learning rate of 0.1 up to epoch 10, then 0.01 up to epoch 12 and 0.001 up to 15. We used an SGD optimizer with momentum 0.9 and weight decay  $5^{-4}$ .

## 8.2 Accuracy

### 8.2.1 Multi-Label Classification on Original Set

First we want to train and test our three models under different parameters on the initial training dataset containing only single label images. The first parameter we look at is the difference between multi-label margin loss (based on hinge loss) and multi-label soft margin loss (based on cross entropy loss), and we can see the results in Table 2.

Looking at the differences in accuracies, we note that multi-label margin loss performs significantly better than multi-label soft margin loss. Moreover, while multi-label margin loss

Model	Loss	Top 5	Top 1
ResNet	Margin Loss	77.4%	41.1%
	Soft Margin Loss	69.3%	37.6%
SENet	Margin Loss	87.8%	44.8%
	Soft Margin Loss	64.7%	38.8%

**Table 2:** Multi-Label Margin Loss compared to Multi-Label Soft Margin Loss on ResNet and SENet

consistently converged to some loss/error rate, many times soft margin loss would get stuck in exploding errors/loss and we would have to terminate training and restart, either from the beginning or from the last plausible checkpoint. This difference is most likely due to the image/label space being difficult to learn at times, and the scaled cross entropy loss in soft margin loss will explode exponentially if we cannot consistently correct our model direction. Margin loss uses linear hinge loss so large losses will only linearly affect our learning and not cause explosions but rather give flexibility to correct over multiple epochs. For this reason, all subsequent experiments we ran used only Multi-Label Margin Loss.

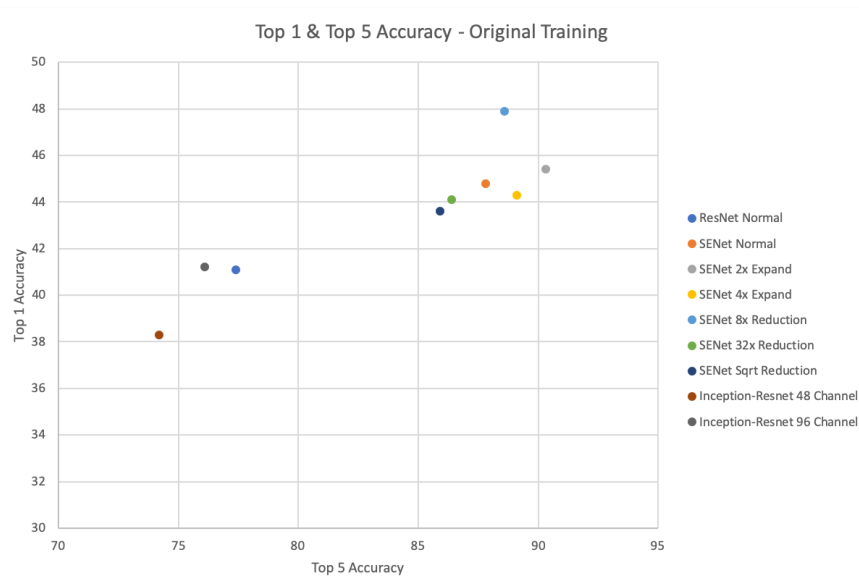
Next we want to look at the actual accuracies achieved on the original single-label training set, shown in Table 3.

Model	Variant	Top 5	Top 1
ResNet	Normal	77.4%	41.1%
SENet	Normal	87.8%	44.8%
	2x Expand	90.3%	45.4%
	4x Expand	89.1%	44.3%
	8x Reduction	88.6%	47.9%
	32x Reduction	86.4%	44.1%
	Sqrt Reduction	85.9%	43.6%
Inception-Resnet	48 Channel	74.2%	38.3%
	96 Channel	76.1%	41.2%

**Table 3:** Top 1 and top 5 validation accuracy after training on original single label dataset. For SENet variants, Expand refers to the factor of expansion we perform in the transform before the squeeze step. Reduction refers to the reduction ratio within the the excitation step.

We can see that adding Squeeze and Excitation expansion actually helps the prediction accuracy on this original set, although we should not place too much emphasis on this since

the training set is all single label targets, so this may be overfitting. Looking at a visual representation in Figure 12 makes it easier to identify trends and relations.



**Figure 12:** Top 1 and Top 5 validation accuracies listed in Table 3. Vertical axis is top 1 accuracy, horizontal axis is top5, the closer to the upper right corner, the better the general accuracy.

If we look at the reduction rates, it seems that all three perform pretty similarly, which is a good sign since these reduction ratios are used to calibrate against overfitting when we start training our next streaming set. Note that reducing the ratio does give us some accuracy gain over the baseline SENet, but again it may be due to overfitting so we cannot put too much weight on this result.

Looking at the Inception Resnet, these two models were slow and difficult to train so the lower accuracies are not surprising. This inception-resnet based model is really more of a proof of concept to be compared to how it reacts to streaming training, and in real life applications we would want to have more compute power and use the higher channel models. In any case, the accuracies are still decent and significant.

Since all these top 5 accuracies and top 1 accuracies are more or less in line with the accuracy benchmarks from single target image classification, we have proven that migrating from a single-label image classification model to a multi-label image classification model does

not have a detrimental effect on the accuracy of single-target classification tasks.

### 8.2.2 Classification Accuracy on Streaming Set

Now we would like to see how well our different models adapt to the new streaming set. Two key metrics we are looking for are to maintain accuracy performance on the original set while still achieving desirable accuracy on the new streaming set. We can see the accuracies in Table 4.

Model	Variant	Base	Original	Streaming
ResNet	Normal	77.4%	56.2%	27.3%
SENet	Normal	87.8%	76.3%	69.8%
	2x Expand	90.3%	84.3%	76.7%
	4x Expand	89.1%	63.7%	92.6%
	8x Reduction	88.6%	79.6%	89.3%
	32x Reduction	86.4%	85.8%	71.9%
	Sqrt Reduction	85.9%	82.3%	71.6%
SENet	2x Expand/32x Reduce	86.8%	86.2%	75.4%
Inception-Resnet	48 Channel	74.2%	76.9%	69.2%
	96 Channel	76.1%	69.1%	74.4%

**Table 4:** Top 5 validation accuracy on original set and streaming set after streaming training. For SENet variants, Expand refers to the factor of expansion we perform in the transform before the squeeze step. Reduction refers to the reduction ratio within the the excitation step. The column **Base** is the top 5 accuracy on the original set (copied from Table 3 for convenience). **Original** and **Streaming** is the validation accuracy on the original and streaming set, respectively, after training on the streaming set.

We can see that the normal ResNet takes a significant hit to original accuracy while also having pretty low accuracy on the streaming set. The normal SENet does much better, generally preserving original dataset accuracy while also achieving comparable streaming accuracy.

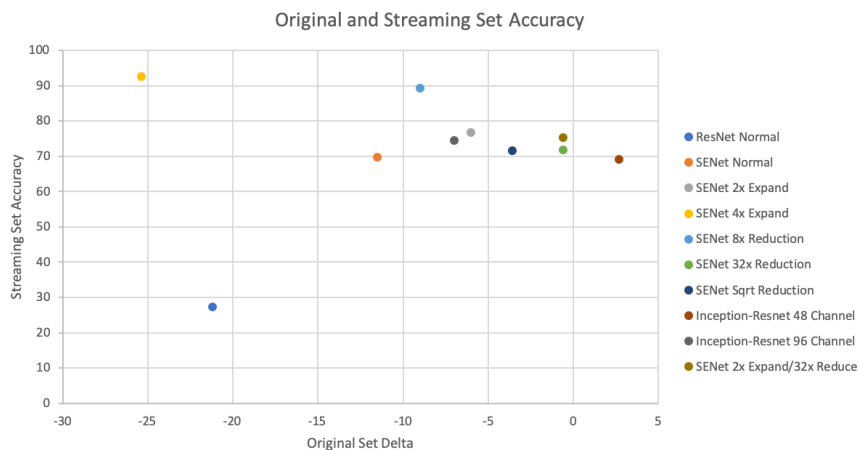
For the 4x expanded and 8x reduced variants of the SENet, we can see that there is a large factor of overfitting occurring, as the accuracy on the streaming set is very good, but the accuracy on the original set falls much lower.

For the 2x expanded and 32x reduced variants of SENet, we see that the performance

is actually very good, both preserving decent performance on the original set while also achieving good performance on the new streaming set. We thus implemented an SENet using both variant modifications, and found a very good balance between original and streaming set accuracy as shown.

While the Inception-Resnet performance lagged behind SENet performance, when comparing the performance after streaming to its own performance before, it actually appears to perform decently well, with 48 channel seemingly outclassing the more complex 96 channel variant. With more compute resources, an interesting additional experiment could be to implement a full Inception-Resnet with the Squeeze and Excitation blocks, modified with expanded channels and various reduction ratios (i.e. 2x expansion and 32x reduction, although these exact figures may need to be returned and tested on the inception-Resnet model).

We can see the relations more clearly in a graphical representation in Figure 13.



**Figure 13:** Accuracy on original and streaming datasets as seen in Table 4. The horizontal axis is the change in top 5 accuracy of the original dataset after training on the streaming set, and the vertical axis is the top 5 accuracy on the new streaming set.

The overfitting of the SENet 4x Expanded and Resnet Normal are very clear, as the negative delta in original set is huge. The other models have slightly more clustered performance, with SENet 8x Reduction achieving the highest streaming set performance but taking a slight hit (little less than 10%) in original set performance. The inception-resnet 48 channel actually had a slightly positive delta, the only one to do so, and had a streaming

set performance of 69% which is comparable to its normal performance on the original set. The positive delta may be a slight anomaly, as subsequent streaming training runs yielded a mix of positive or negative deltas, but this model is still the only one to do so, showing its resilience to overfitting which we expect from inception networks. The best overall performer was the SENet 2x Expand, 32x Reduce, which maintained a 86.2% original set accuracy and achieved a 75.4% streaming set accuracy. This indicates that our expansion and reduction variations to our SE networks work well together.

### 8.3 Performance

We measure the performance measured in training time of our various multi-label image classification models as compared to single-label image classification using traditional ResNet. In Table 5, we can see the training times for each epoch on the 20,000 image original single label set as well as the the epoch training times on the 1000 image multi label streaming set.

Model	Variant	Original Epoch	Streaming Epoch
Single Target ResNet	Normal	395.7 sec	394.3 sec
	Normal	403.7 sec	20.9 sec
SENet	Normal	417.2 sec	25.3 sec
	2x Expand	421.4 sec	26.7 sec
	4x Expand	425.6 sec	24.2 sec
	8x Reduction	408.2 sec	25.8 sec
	32x Reduction	406.4 sec	20.6 sec
	Sqrt Reduction	402.7 sec	19.7 sec
Inception-Resnet	96 Channel	587.4 sec	37.1 sec
	48 Channel	305.6 sec	16.8 sec

**Table 5:** Epoch Runtime of various models. These runtimes are the average time per epoch over 10 epochs for each model. The runtime for training on the original single label set is shown alongside the runtime for training on the streaming dataset.

We can see that the squeeze and excitation addition to the model adds a slight increase in training time, around a 5% increase. This additional runtime becomes more pronounced with SENet 2x and 4x expansions, as expected, since we are squeezing more elements, but is a very marginal increase because the squeeze is one of the least expensive operations. We

see that increasing the reduction ratio seems to slightly decrease the runtime, which is also expected since the convolutions in the excitation step become smaller.

For the Inception addition, it is a little more difficult to compare to normal ResNet, since the Inception Module architecture is fundamentally different. However, we see that the epoch runtimes are not drastically different, roughly 40% slower for 96 channel and 25% faster for 48 channel. Memory is the main computational need for inception-resnets, and the reason we could not test higher channel width models, so even though the runtimes look comparable, the workload was much greater than our more standard ResNet and SENet architectures.

Since the single label classifier needs to retrain the entire model when the class space changes (adding 10 new classes so 120 to 130 classes), the training time once we add the streaming set more than doubles. Compare this to the training time of the 1000 image model, which only needs to train on the smaller streaming dataset, and validate on some validation split of the original set. A large majority of the time actually comes from the original set validation since it is so large. Thus in Table 5 we see that using our multi-label approach, each epoch is around 8 times faster than it would be using a traditional single-label approach. Actually the time savings would be even greater, since the single-label approach would have to retrain basically from scratch, so another 15 epochs, each epoch being 8 times slower, whereas in our multi-label approach we only need 4 of the faster epochs, so in total for this example, we would have around a 30x speedup by using our scalable approach as compared to traditional models.

## 8.4 Summary

Of the various model additions that we tested, it seemed that the SENet channel expansion worked well for allowing the model to capture higher dimensional features, specifically the label interdependencies for our multi-label classification task. However, increasing the channel expansion too far (4x in our tests) caused drastic overfitting, and so we found 2x

expansion to be the best overall performer. Increasing the channel reduction worked well to help prevent against overfitting on the streaming set relative to the original set, and we found that 32x reduction worked best. Square root reduction, while interesting, showed little to no benefit over 32x reduction, and just introduced more computational overhead and complexity to our models, which is why we ultimately chose 32x.

The inception-resnet models showed a lot of promise, and with more compute resources, testing out a 384 channel model as opposed to our 48 and 96 channel models could possibly yield very good results. Specifically, Inception-Resnet 384 might perform a lot closer to its 50 and 152 layer ResNet and SENet counterparts than our scaled down models did, all while maintaining the good generalization to streaming sets without losing original set accuracy that we saw in our smaller models.

Based on the potential results of a 384 layer Inception-Resnet, the best performing model might be this Inception-Resnet with blocks all replaced with SEBlocks using 2x expansion and 32x reduction, aptly named Inception-SENet. The performance of this would depend on how well squeeze and excitation works with the multiple convolutions of inception modules, but since the squeeze and excitation steps are separate from the convolutional steps, there is no reason why they would be adversarial, although this still should be tested.

Based on the results of our research, we are able to consistently achieve general multi-label accuracy (over both original and streaming sets) within 3% of single-label models, while being up to 30x faster in real world scenarios of adding new training data to an already trained model.



## 9 Conclusion

In our research, we were able to create a multi-label image classification model that allows us to perform large scale visual recognition in a streaming fashion. While traditional supervised models are focused on training on the complete space of unique classes, our approach instead trains on a fixed space of inter-dependent classes, and allows new data to be trained as combinations of the existing known classes. In this fashion, we can scale our visual recognition models indefinitely. Our training at each time step is independent of the size of the set of previously trained examples, but rather our workload is dependent on the size of the new data added, hence we coin the term "streaming training".

We were able to build a model that sacrifices marginal overall classification accuracy for drastically faster training times (one order of magnitude) when applied to real world training scenarios. Our techniques are easily applied to current models and only require simple modifications to translate traditional models to "streaming" models.

## References

- [1] K. Simonyan, A. Vedaldi, and A. Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *arXiv preprint arXiv:1312.6034* (2014).
- [2] C. Szegedy et al. “Going deeper with convolutions”. In: *arXiv preprint arXiv:1409.4842* (2014).
- [3] K. He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv preprint arXiv:1512.03385* (2015).
- [4] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv preprint arXiv:1409.1556* (2015).
- [5] R. K. Srivastava, K. Greff, and J. Schmidhuber. “Highway Networks”. In: *arXiv preprint arXiv:1505.00387* (2015).
- [6] J. Wang et al. “CNN-RNN: A Unified Framework for Multi-label Image Classification”. In: *arXiv preprint arXiv:1604.04573* (2016).
- [7] C. Szegedy et al. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: *AAAI Conference on Artificial Intelligence (AAAI)* (2017).
- [8] J. Hu et al. “Squeeze-and-Excitation Networks”. In: *arXiv preprint arXiv:1709.01507* (2018).
- [9] S. Kornblith, J. Shlens, and Q. Le. “Do Better ImageNet Models Transfer Better?” In: *arXiv preprint arXiv:1805-08974* (2018).
- [10] H. Liu, K. Simonyan, and Y. Yang. “DARTS: Differentiable Architecture Search”. In: *arXiv preprint arXiv:1806-09055* (2018).
- [11] Y. Liu et al. “Multi-Label Image Classification via Knowledge Distillation from Weakly-Supervised Detection”. In: *arXiv preprint arXiv:1809.05884* (2019).
- [12] A. Khosla et al. *Stanford Dogs dataset for Fine-Grained Visual Categorization*.

- [13] A. Krizhevsky, I. Sutskever, and G. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *nips* ().