CS 182 Vision Project Report

Arman Tigranyan, John Newsom, Jason Qiao, Jerry Li

Abstract

The goal of this project is design an image classifier which is robust to as many test-time perturbations as possible. We use the Tiny ImageNet dataset for our purposes. It is a dataset having 200 classes, each with 500 training images (100,000 images) that are 64x64 RGB images. The public validation dataset has 50 images per class (10,000 images). Our classifier should perform well on the public validation dataset and be robust to many different perturbations/shifts that occur in the real world. For those purposes, we also construct a corrupted version of Tiny ImageNet. Our best model gains 85.70% top-1 and 96.69% top-5 validation accuracy. We use mean corruption error to evaluate the robustness of our model, giving us a score of mCE=0.50.

Introduction

Since 2010, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is the standard benchmark of image classification, which is the task of assigning an input image one label from a fixed set of categories. It is a fundamental task within Computer Vision that is relatively trivial for human beings to perform, yet challenging for a computer vision algorithm.

Throughout the past decade or so, Convolutional Neural Networks (CNNs) research has dominated in the realm of solutions to the task of image classification. Most notable discoveries have been deep convolutional architectures such as AlexNet in 2012, VGG in 2014, ResNet in 2015, Densenet in 2016, and EfficientNet in 2019. While a full comparison of the aforementioned models is out of scope for this project, we focus mostly on SqueezeNet, DenseNet, ResNet and the different variants of the EfficientNet.

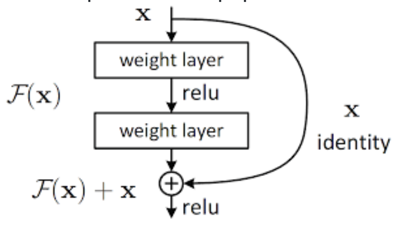
Our goal in this project is to achieve a high validation accuracy model that will also be robust to various corruptions to images.

Related Work

While determining which model would be best fit for our problem, we explored SqueezeNet, DenseNet, and EfficientNet.

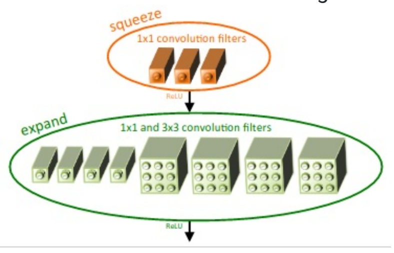
ResNet was published in a seminal paper by Kaiming He et al. in 2015. Before ResNets, convolutional networks could not be trained effectively if they were more than a few layers deep because of issues like vanishing and exploding gradients. He et al. addressed this problem by adding a new structure to their convolutional network called skip connections, which allow layers to take as input the outputs of layers that are not immediately above them in the network structure. He et al. used this strategy to implemenet at 152 layer convolutional network that was both highly performant on ImageNet and had significantly fewer parameters than the then-state-of-the-art VGG.

An example of the skip connection popularized in the ResNet paper


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

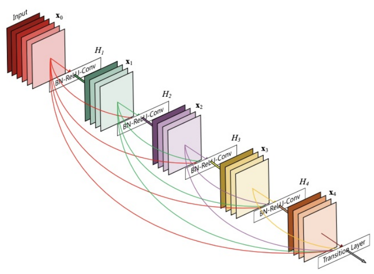
SqueezeNet was first introduced in 2016 by Iandola et al. as an alternative to AlexNet with 2% as many parameters. The authors achieve these results primarily in the early convolutional layers. Changes include replace 3x3 filters with 1x1 filter, decreasing the number of input channels to 3x3 filters, and applying downsampling only very late in the network.

Squeezenet 'fire' module for reducing filter size

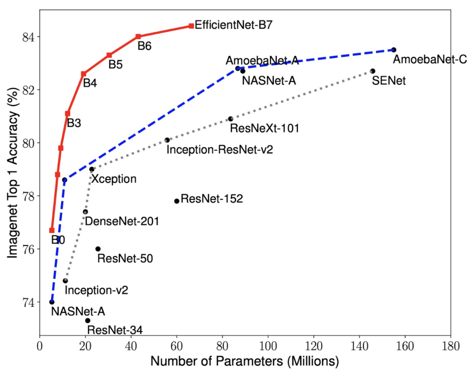


DenseNets, introduced later that year by Huang et al., further improved on results of SqueezeNet by increasing the number of connections between convolutional layers.

A 5 layer dense block



Finally, EfficientNet, a model introduced by Tan and Le in 2019, uses neural architecture search to find an optimal relationship between hyperparameters like depth, width, and resolution, which led to state-of-the-art performance on ImageNet while being both an order of magnitude smaller and faster than the previous best convolutional network.



Dataset

For this project, we trained our networks on Tiny Imagenet, a lightweight version of ImageNet. It consists of 200 classes, each having 500 training and 50 validation images of size 64x64 pixels, as opposed to ImageNet's 1000 classes with 1.2 million training images of size 256x256 pixels.

In order to add extra robustness against adverserial corruptions, we also perform some training over the Tiny ImageNet-C dataset, created by Dan Hendrycks and Thomas Dietterich. Tiny ImageNet-C contains the same images as Tiny ImageNet, but corrupts each of them using one of fifteen different possible corruptions, including Gaussian noise, motion blur, brightness distortion, and JPEG compression. A version of each image under each corruption is available at five different levels of intensity.

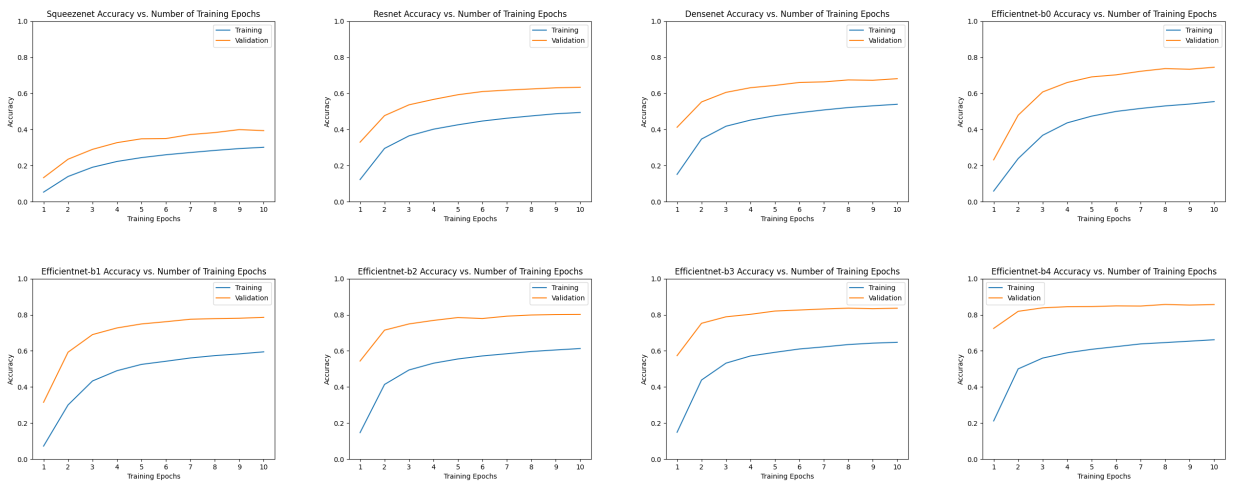
Approaches

Models

As mentioned above, we trained versions of SqueezeNet, DenseNet, and EfficientNet for this problem. We pick these three models because they were all state-of-the-art on the original ImageNet problem when they were originally released; EfficientNet still holds this position.

After fine-tuning versions of each of these three models for 10 epochs, SqueezeNet performed the worst, reaching 39.4% validation accuracy. ResNet beat SqueezeNet significantly, scoring 63.4% on the validation set. DenseNet barely edged out ResNet, scoring 68.1% accuracy. EfficientNet, however, blew both of these numbers out of the water, reaching over 85.7% validation accuracy. Specifically, we reached these numbers with EfficientNet-b4. We would like to use a higher-resolution version of EfficientNet, like b7, but we were limited by GPU memory on Google Colab and Google Cloud.

Graphs of model performance over TinyImagenet-C



Environment and Framework

We used PyTorch as our deep learning framework. For computational resources, we have been making use of Google Colab (Nvidia Tesla T4), Kaggle Notebooks (30 hours/week of Nvidia Tesla P100) and Google Colab (Nvidia Tesla K80).

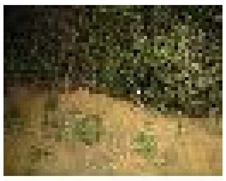
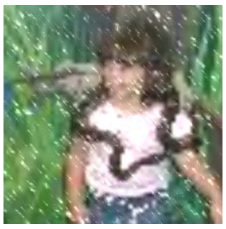
Finetuning

Given ImageNet pretrained models, we limit ourselves to 10 epochs per model for finetuning. We apply SGD optimizer with 0.001 learning rate and 0.9 momentum.

Data augmentation

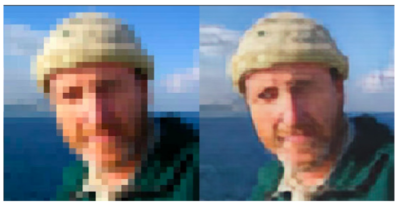
In order to achieve a better performance and avoid potential overfitting, we make use of PyTorch's built-in data augmentation. When an image from the training set is loaded, we upsample it to the expected input size of the given pretrained network (224 for everything except EfficientNet-b4, which uses 380), flip it horizontally with probability 0.5, rotate it by a number of degrees chosen at random from [-15, 15], and finally normalize it.

Additionally, we make use of various corruption transformations to make our model more robust. More specifically, following Hendrycks 2019, we corrupt tiny imagenet to train on. We implement 19 different corruptions (gaussian noise, fog, motion blur) and randomly select one corruption to apply to each image in the training set. These corruptions emulate corruptions in images we may encounter in the real world. Here are some examples of these corruptions:

n01742172_2	n01882714_14	n02074367_78	 n02125311_15
 Snow Corruption	Shot Noise Corruption	Frost Corruption	Zoom Blur Corruption

GAN deblurring

We attempted to use GAN for image deblurring to achieve better result. We first blur the images and use generator to generate deblurred images to fool discriminator which judges whether an input is artificial or not. The dataset we use to train is the GOPRO dataset which contains both the original and blurred images and we code in Python with keras. The outcome was not as desirable as expected. One probability is that the GOPRO's blurred images are images blurred due to some motion of the camera but our images are not. Thus, we finally didn't apply deblurGAN on the pictures. Here is an example of the deblurred image.

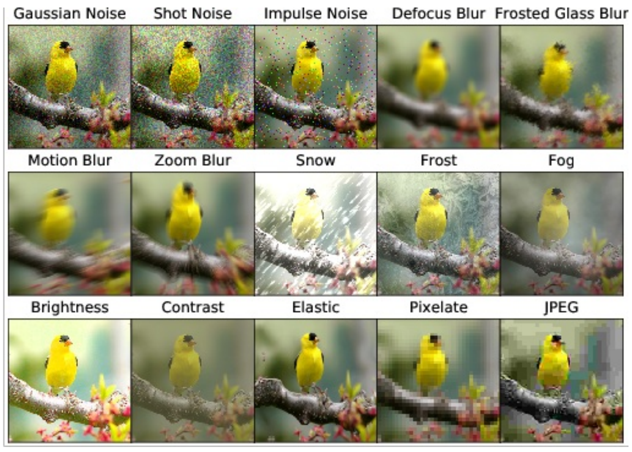


Robustness

Imagenet-C

Training/validation dataset hosted here: https://archive.org/details/tiny-imagenet-c-200

Imagenet-C is a dataset developed by Hendrycks and Dietterich to benchmark various networks in their robustness against real world corruptions. Each image in Imagenet-C is corrupted using one of the 19 corruptions described above at a severity level out of 5.



Hendrycks and Dietterich also developed a second dataset, Imagenet-P, which benchmarks robustness against perturbations such as translations, zooms, etc. We focused on Imagenet-C because perturbations are easily emulated using common data preprocessing techniques.

mCE

Mean corruption error, or mCE, is one of the metrics devised by Hendrycks to benchmark neural networks against robustness. For every corruption, we calculate an error rate at each level to get a corruption error value. However, different corruptions affect performance differently (e.g. fog might have higher error rates than a gaussian blur corruption), so we divide each corruption error by a baseline network's corruption errors. Hendrycks uses AlexNet as a baseline so we will do that same. Averaging these normalized errors gives us our network's mCE.

Results

Subscript "{MODEL}_b" indicates an ImageNet-pretrained model finetuned for 10 epochs on the vanilla tiny-imagenet. Subscript "{MODEL}_c" indicates an ImageNet-pretrained model finetuned for 10 epochs on the corrupted tiny-imagenet-c dataset.

Val Acc (%) / Model	Squeezenet_b	Resnet_b	Densenet_b	EfficientNet-b0_b	EfficientNet-b1_b	EfficientNet-b2_b	EfficientNet-b3_b	EfficientNet-b4_b
Top-1	51.93	71.29	75.22	80.62	84.16	85.54	88.56	89.43
Top-5	77.26	90.09	92.13	94.28	96.30	96.49	97.80	97.92
Val Acc (%) / Model	**Squeezenet_c**	**Resnet_c**	**Densenet_c**	**EfficientNet-b0_c**	**EfficientNet-b1_c**	**EfficientNet-b2_c**	**EfficientNet-b3_c**	**EfficientNet-b4_c**
Top-1	39.93	63.37	68.14	74.47	78.57	80.19	83.66	85.70
Top-5	66.46	85.26	88.12	91.58	93.62	94.58	95.79	96.69

Below is the mCE and per-class corruption error for various models we trained. We use AlexNet as our baseline to calculate mCE.

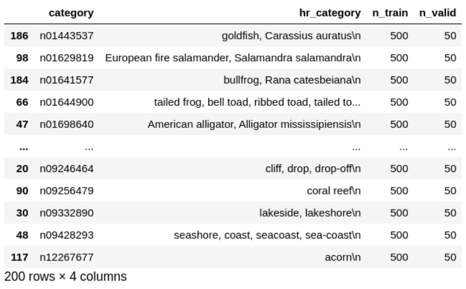
Model	mCE	Bright.	Contrast	Defocus	Elastic	Fog	Frost	Gauss.	Glass	Impulse	jpeg	Motion	Pixel.	Shot	Snow	Zoom
AlexNet	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
EfficientNet-b1	0.55	0.59	0.70	0.61	0.62	0.50	0.44	0.55	0.65	0.50	0.57	0.53	0.51	0.49	0.46	0.54
EfficientNet-b2	0.53	0.58	0.64	0.58	0.59	0.48	0.43	0.52	0.63	0.47	0.55	0.50	0.50	0.46	0.45	0.51
EfficientNet-b4	0.50	0.57	0.66	0.57	0.55	0.48	0.40	0.46	0.61	0.41	0.52	0.48	0.43	0.41	0.42	0.48
EfficientNet-b4 trained on tiny-imagenet	0.56	0.49	0.79	0.61	0.58	0.57	0.40	0.64	0.72	0.55	0.47	0.52	0.48	0.55	0.45	0.55

Visualization

Credits: https://github.com/WillKoehrsen/pytorch_challenge/blob/master/Transfer%20Learning%20in%20PyTorch.ipynb

Tiny Imagenet

Let us first explore our dataset. There are 200 classes total, with 500 training and 50 validation images per category.



	category	hr_category	n_train	n_valid
186	n01443537	goldfish, Carassius auratus\n	500	50
98	n01629819	European fire salamander, Salamandra salamandra\n	500	50
184	n01641577	bullfrog, Rana catesbeiana\n	500	50
66	n01644900	tailed frog, bell toad, ribbed toad, tailed to...	500	50
47	n01698640	American alligator, Alligator mississipiensis\n	500	50
...	...	...	...	...
20	n09246464	cliff, drop, drop-off\n	500	50
90	n09256479	coral reef\n	500	50
30	n09332890	lakeside, lakeshore\n	500	50
48	n09428293	seashore, coast, seacoast, sea-coast\n	500	50
117	n12267677	acorn\n	500	50

200 rows × 4 columns

Each picture is 64x64 pixels in size.

['king penguin, Aptenodytes patagonica\n', 'thatch, thatched roof\n', 'wok\n', 'seashore, coast, seacoast, sea-coast\n']
0
100
200
300
0
200
400
600
800
1000
1200
1400

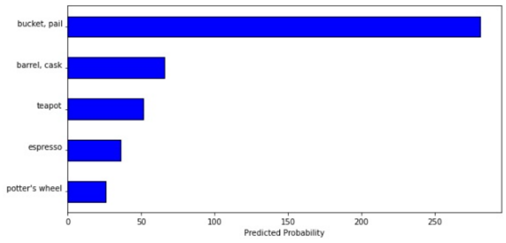
Classification

Let us now explore our best performing model's classification results. Here are some randomly selected imaged from the validation set:

path: ./tiny-imagenet-200/val/n07614500/val_6147.JPEG;
predicted: ice cream, icecream
;
actual: ice cream, icecream

As one can notice, the model seems to be classifying the validation images fairly well. The last image got misclassified as a bucket, which is still close enough. For a human being, distinguishing between a bucket and a water jug could potentially be challenging too.

It is also interesting to look at the top5 predictions for the misclassified image above:



As we can see, even the top-5 predicted classes are close enough to each other to potentially fool a human observer.

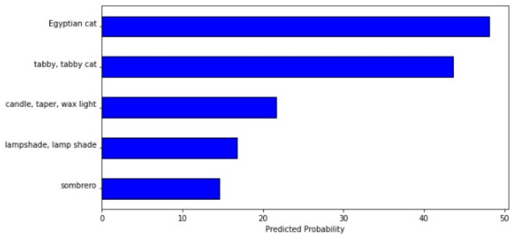
But let us try to see just how good our model's results are on the validation dataset.

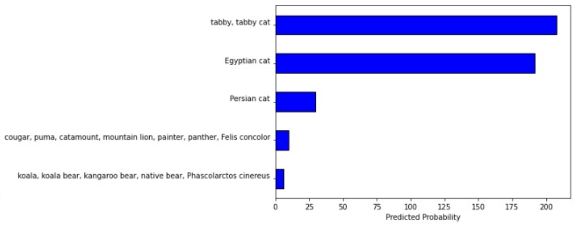
The 10 worst top-1(left) and top-5(right) categories are:

	class	hr_class	top1	top5	loss
132	n03976657	pole\n	42.0	90.0	-4.793992
30	n02123045	tabby, tabby cat\n	44.0	88.0	-5.525692
159	n04376876	syringe\n	44.0	70.0	-3.290758
172	n04560804	water jug\n	52.0	90.0	-4.948066
175	n04597913	wooden spoon\n	54.0	84.0	-3.913039
168	n04507155	umbrella\n	56.0	78.0	-3.983121
131	n03970156	plunger, plumber's helper\n	56.0	80.0	-3.945476
190	n07871810	meat loaf, meatloaf\n	58.0	90.0	-5.802985
135	n03983396	pop bottle, soda bottle\n	60.0	94.0	-5.177217
182	n07711569	mashed potato\n	62.0	94.0	-6.134476

	class	hr_class	top1	top5	loss
159	n04376876	syringe\n	44.0	70.0	-3.290758
168	n04507155	umbrella\n	56.0	78.0	-3.983121
131	n03970156	plunger, plumber's helper\n	56.0	80.0	-3.945476
175	n04597913	wooden spoon\n	54.0	84.0	-3.913039
79	n02906734	broom\n	74.0	86.0	-4.205380
67	n02795169	barrel, cask\n	68.0	86.0	-4.697035
139	n04067472	reel\n	68.0	86.0	-4.294508
9	n01774750	tarantula\n	82.0	86.0	-5.995072
99	n03250847	drumstick\n	74.0	88.0	-4.449768
62	n02730930	apron\n	72.0	88.0	-5.106379

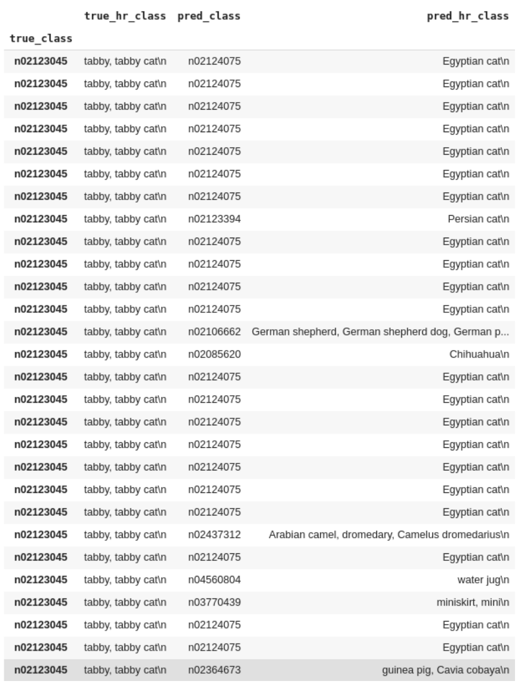
Let us take a look at the model's predictions of a couple of sample images whose class is "tabby, tabby cat":



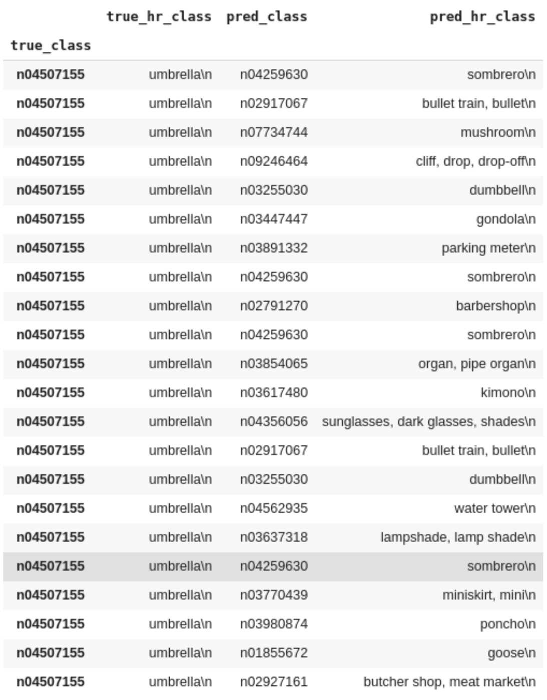


Looks like the model tends to include class "Egyptian cat" in its predictions. Let us take a look at just how frequent those types of misclassifications are.

On the left, we have the list of top-1 misclassifications for class "tabby, tabby cat". On the right, we have the list of top-5 misclassifications for class "umbrella":



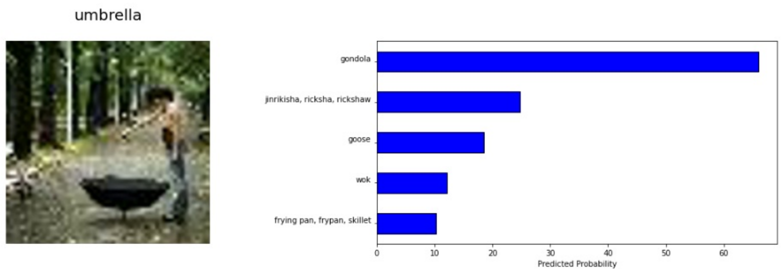
true_class	true_hr_class	pred_class	pred_hr_class
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02123394	Persian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02106662	German shepherd, German shepherd dog, German p...
n02123045	tabby, tabby cat\n	n02085620	Chihuahua\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02437312	Arabian camel, dromedary, Camelus dromedarius\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n04560804	water jug\n
n02123045	tabby, tabby cat\n	n03770439	miniskirt, mini\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02124075	Egyptian cat\n
n02123045	tabby, tabby cat\n	n02364673	guinea pig, Cavia cobaya\n



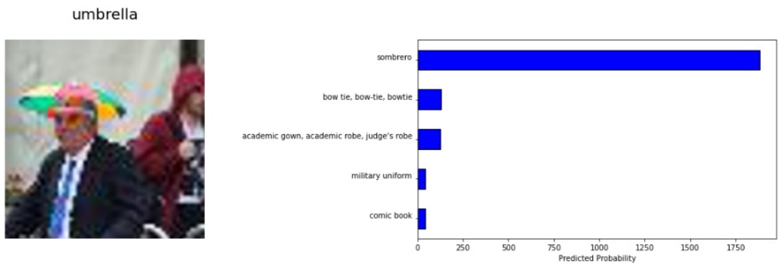
true_class	true_hr_class	pred_class	pred_hr_class
n04507155	umbrella\n	n04259630	sombrero\n
n04507155	umbrella\n	n02917067	bullet train, bullet\n
n04507155	umbrella\n	n07734744	mushroom\n
n04507155	umbrella\n	n09246464	cliff, drop, drop-off\n
n04507155	umbrella\n	n03255030	dumbbell\n
n04507155	umbrella\n	n03447447	gondola\n
n04507155	umbrella\n	n03891332	parking meter\n
n04507155	umbrella\n	n04259630	sombrero\n
n04507155	umbrella\n	n02791270	barbershop\n
n04507155	umbrella\n	n04259630	sombrero\n
n04507155	umbrella\n	n03854065	organ, pipe organ\n
n04507155	umbrella\n	n03617480	kimono\n
n04507155	umbrella\n	n04356056	sunglasses, dark glasses, shades\n
n04507155	umbrella\n	n02917067	bullet train, bullet\n
n04507155	umbrella\n	n03255030	dumbbell\n
n04507155	umbrella\n	n04562935	water tower\n
n04507155	umbrella\n	n03637318	lampshade, lamp shade\n
n04507155	umbrella\n	n04259630	sombrero\n
n04507155	umbrella\n	n03770439	miniskirt, mini\n
n04507155	umbrella\n	n03980874	poncho\n
n04507155	umbrella\n	n01855672	goose\n
n04507155	umbrella\n	n02927161	butcher shop, meat market\n

Looks like most of "tabby, tabby cat"s top-1 misclassifications are of class "Egyptian cat", which is fortunate - at least the model tends to recognize the fact that it's a cat, even in misclassifications. Notice that "tabby, tabby cat" is not in the 10 worst top-5 misclassifications - that means that the model is good at including the right prediction in the top 5 classes, but has troubles distinguishing between a tabby cat and an Egyptian cat, resulting in poor top-1 performance.

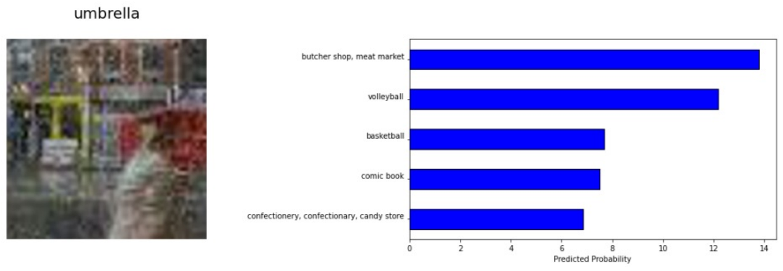
On the other hand, the list of top-5 misclassifications for class "umbrella" is much more diverse. For example, looks like the model thinks the upside-down umbrella looks more like a gondola.



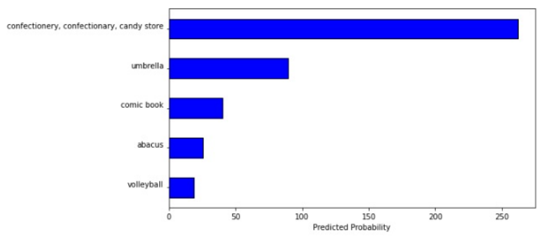
Here, the umbrella is worn like a hat, so the model thinks it's a sombrero with very high confidence.



The model seems to be confused on what exactly to be looking at. Supposedly it focuses its attention on the building, giving "butcher shop, meat market" and "confectionery, confectionary, candy store" in its top-5 classifications.



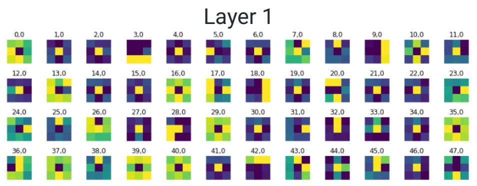
Again, the model is confused what to be looking at. The image looks colorful so the model thinks it's a candy store with colorful candy.

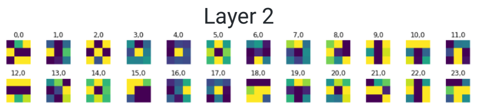


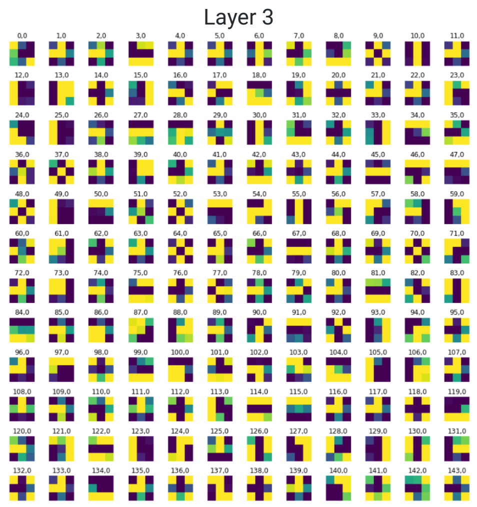
Filters

Credits: https://github.com/Niranjankumar-c/DeepLearning-PadhAI/blob/master/DeepLearning_Materials/6_VisualizationCNN_Pytorch/CNNVisualisation.ipynb

The first three depthwise_conv layers of our efficientnet-b4 are:



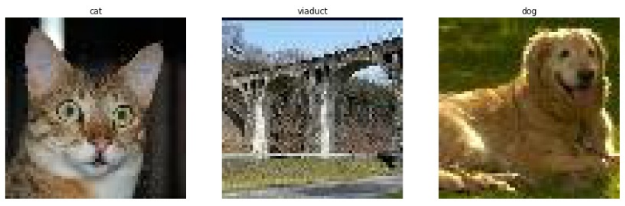




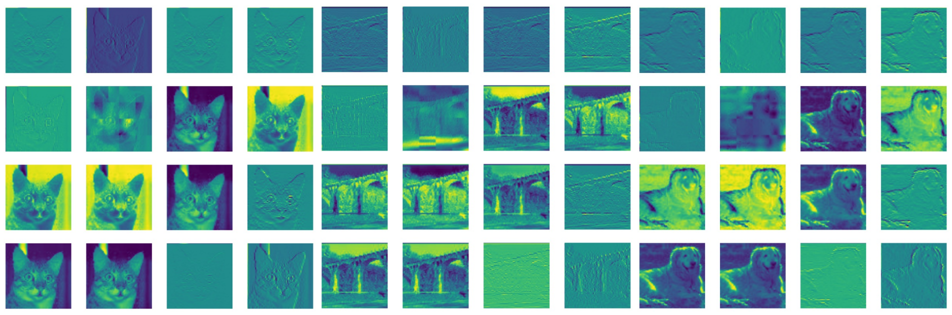
Weights and Activations

Credit: https://arxiv.org/pdf/1610.02391.pdf; https://github.com/FrancescoSaverioZuppichini/A-journey-into-Convolutional-Neural-Network-visualization-; https://www.kaggle.com/meaninglesslives/efficientnet-eb0-eb5-model-comparisons

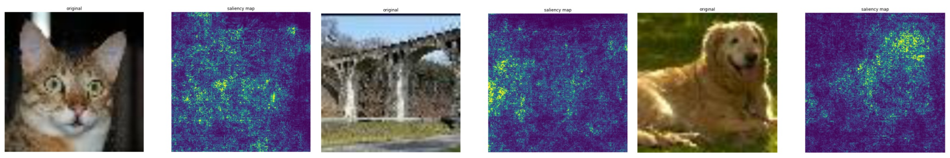
Here are some images from our dataset:



Let's plot the first layer's weight for efficientnet-b0:

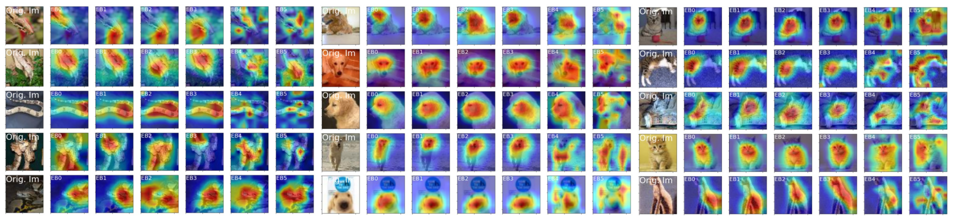


Next, let us look at efficientnet-b0's saliency maps for those same images:



Looks like the model looks closely at the cat's eyes, nose and ears, the dog's face and the viaduct's leftmost arch.

Now let us visualize the Grad-CAMs of different efficientnets and see what we can infer. The images below correspond to boa constrictor, dog and cat, from left to right.



Looks like that, the more complex the efficientnet, the more abstractly it looks at the images. Earlier imagenets focus on specific parts of the image like faces, while later efficientnets look at various parts of the image.

Conclusion

In summary, we set out to find a model that could be both performant on the Tiny ImageNet problem and resilient against a wide range of perturbations and corruptions. To do so, we expiremented with a number of models that were known to be performant on the original ImageNet problem. Each was trained over a modified version of Tiny ImageNet called Tiny ImageNet C that contains over fifteen different types of corruptions at various intensities. To compare their performance, we calculate three metrics: top-1 accuracy, top-5 accuracy, and mean corruption error. Our best model ends up having 85.70% top-1 and 96.69% top-5 validation accuracy and 0.5 mCE.

Perhaps unsurprisingly, of the models we considered, the best according to the three aforementioned metrics was also the most performant on the original ImageNet problem, EfficientNet-b4. We believe that this is a reflection of the model's effective scaling structure as introduced in the original paper. As a result, we also believe that one could see better performance in EfficientNets with higher input resolution like EfficientNet-b7.

Team Contributions

Arman Tigranyan (SID: 3032106429): model training, visualizations (25%)

John Newsom (SID: 3033896038): model design, data transformations (25%)

Jason Qiao (SID: 3032724788): dataset augmentation/corruption and mCE calculations (25%)

Jerry Li (SID: 3031882582): image processing (deblurGAN, etc), cloud platform exploration (25%)

You can find our full source code here: https://github.com/jqiao2/sp20-cs182-final-project

References

ILSVRC: http://www.image-net.org/challenges/LSVRC/

AlexNet: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

VGG: https://arxiv.org/pdf/1409.1556.pdf

ResNet: https://arxiv.org/pdf/1512.03385.pdf

Squeezenet: https://arxiv.org/pdf/1602.07360.pdf

Densenet: https://arxiv.org/pdf/1608.06993.pdf

Efficientnet: https://arxiv.org/abs/1905.11946.pdf

Survey of CNNs: https://arxiv.org/pdf/1803.01164.pdf

Advprop: https://arxiv.org/pdf/1911.09665.pdf

DeblurGAN: https://arxiv.org/pdf/1711.07064.pdf

Robustness: https://arxiv.org/pdf/1903.12261.pdf; https://github.com/hendrycks/robustness

Visualization: https://github.com/WillKoehrsen/pytorch_challenge/blob/master/Transfer%20Learning%20in%20PyTorch.ipynb; https://github.com/FrancescoSaverioZuppichini/A-journey-into-Convolutional-Neural-Network-visualization-; https://www.kaggle.com/meaninglesslives/efficientnet-eb0-eb5-model-comparisons

Preprocessing: https://github.com/tjmoon0104/Tiny-ImageNet-Classifier