Udacity MLND Capstone Report
On Classification of the STL-10 dataset using Deep Convolutional Networks and Transfer Learning

Tanbir Ahmed

1 Definition

1.1 Domain background and overview

Computer vision is a vast and inter-disciplinary field that seeks to automate tasks performed by human vision-system from object detection to driving a vehicle or flying an aircraft. Many different areas and sub-areas have emerged over the years to cover various aspects of computer vision. Object-detection on images still remains as one of the toughest among the fundamental areas. Datasets continue to pose challenges as researchers continue their effort to overcome the learning barriers. The motivations behind investigating the STL-10 dataset (described in the dataset section) in this project are as follows:

- *Investigating networks from scratch and various pretrained-models for transfer learning*, and
- *Working on a challenging benchmark dataset that presents plenty of room for improvement in performace and developing scalable learning models.*

For historical purpose, the interested readers may consult [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. In this project, we introduce the naming: *Adding Imperfections to Compensate Inadequacies (AICI)* for deep learning in image object detection.

1.2 Problem statement

The main objective of this project is to apply deep-learning techniques to investigate the object-detection on images (image-classification) problem conducting expriments on the STL-10 dataset. In this dataset, the training subset is smaller than the test subset and the state-of-the-art has apparently significant scope for improvement.

Though, getting a state-of-the-art result is not within the scope of this project, obtaining a competitive baseline result using a basic CNN (Convolutional Neural Network) is well within its purpose. The availability of the unlabeled dataset and several pre-trained imagenet models open up possibilities for further improvement using the baseline model on an augmented dataset.

It is highly tempting to try as many different known techniques as possible on the above problem, yet time and computing resources may constrain the options. The Benchmarks section describes more about the possibilities.

1.3 Evaluation metrics

The evaluation metric used is the *multi-class logarithmic loss* (also known as *categorical cross entropy* in machine learning context). It measures the performance of a classification model

whose output is a probability (output of *softmax* activation) between 0 and 1. Let N be the number of images and M be the number of class labels. For $1 \leqslant i \leqslant N$ and $1 \leqslant j \leqslant M$, let $y_{i,j}$ denote the indicator variable which equals to 1 if and only if observation i belongs to class j. Let $p_{i,j}$ denote the predicted probability that observation i belongs to class j. Then the multi-class logarithmic loss is

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\ln\left(p_{i,j}\right).$$

Note that logarithmic loss takes into account the uncertainty of the prediction based on how much it varies from the actual label. On the other hand, metrics such as accuracy only counts the predictions where the predicted value equals the actual value. The log loss function is convex and differentiable, and can be minimized using stochastic gradient discent methods. So in our deep neural network application, we choose this metric for evaluating performance.

2 Analysis

2.1 Data sets and inputs

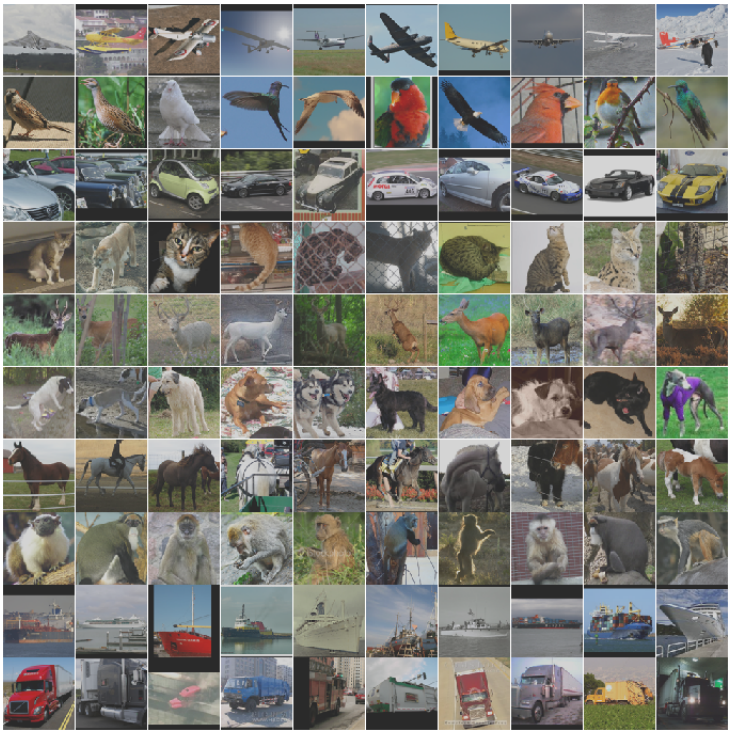


Figure 1: STL-10 dataset example (10 images from each class)

As presented at Adam Coates' STL-10 page, the STL-10 dataset [4] is an image recognition dataset for developing unsupervised feature learning, deep learning, and self-taught learning algorithms. The dataset has the following properties:

- *There are 10 classes: airplane, bird, car, rat, deer, dog, horse, monkey, ship, truck*

- Total 5000 *training images and* 8000 *test images each with shape* $96 \times 96 \times 3$ *for supervised learning. There are* 500 *training images (*10 *pre-defined folds) and* 800 *test images per class.*

- 100000 unlabeled images for unsupervised learning with images from a similar but a broader distribution of images.

These images were acquired from labeled examples on Imagenet. The dataset is inspired by the CIFAR-10 dataset ($60000 \times 32 \times 32 \times 3$ with 50000 training images and 10000 test images). But higher resolution makes the STL-10 dataset a challenging benchmark for developing scalable learning models.

2.1.1 Exploratory visualization

- Frequency distribution:

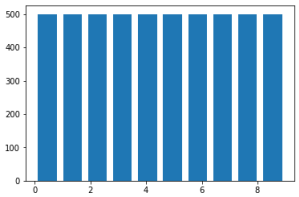
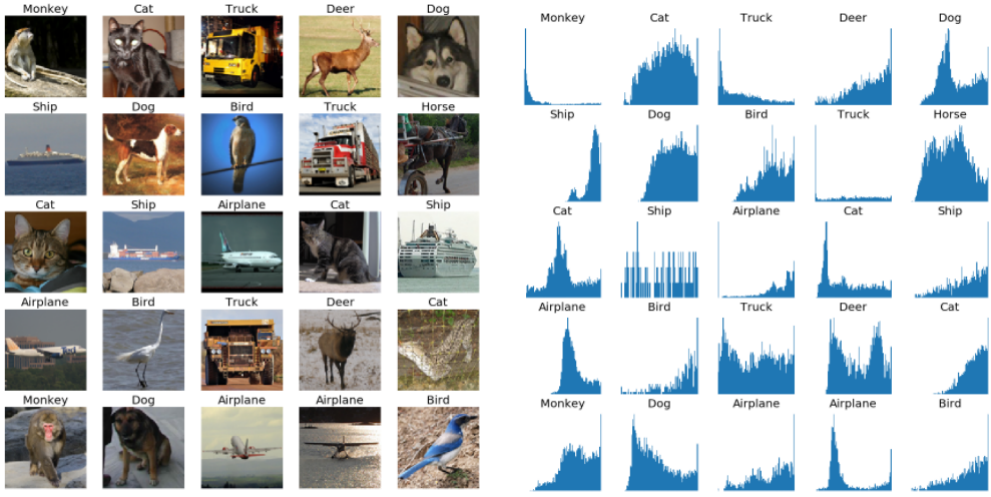


Figure 2: Frequency distribution of the training images (5000 x 96 x 96 x 3)

- Intensity histograms of random training images:



2.2 Solution statement

Deep learning techniques have been extremely successful in classification of large image datasets. One of the fundamental challenges is scalability of the designed models. The STL-10 dataset is a perfect example where obtaining model scalability is a major challenge given the training subset is smaller compared to the test subset. The availability of a large unlabeled subset and several pre-trained imagenet models would allow transfer learning and improvement of performace towards obtaining scalability. In this project, I attempt to explore such a possibility.

2.3 Algorithms and Techniques

2.3.1 Choice of Architecture in designing a Convolutional Neural Network

Designing a neural network is much of an engineering art and like any art there is a certain degree of freedom in how we approach to design one. But based on the problem at hand, a specific design may be tuned to adjust the number, position, type (Convolutional, Fully-Connected, Maxpooling, Batch-normalization, and Drop-outs), and shape (size of filters, number of output channels, and height of fully-connected layers) of layers for better performance. Briefly, the layers have the following properties:

- Convolutional and Maxpooling layers: extracts features through small regions or filters

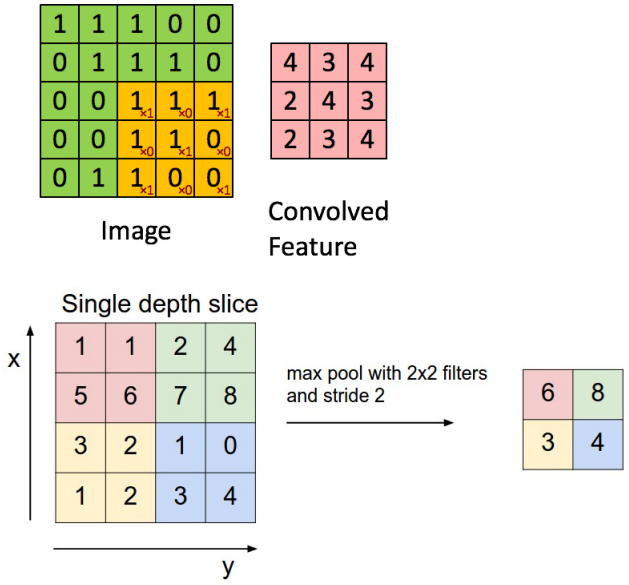


Figure 3: Feature extraction using Convolutional and Maxpool layers (pic source: [18])

- Fully-connected layer: every neuron (unit) in the layer is connected to all activations in the previous layer
- Batch-normalization layer: normalizes the output of the layer before

In this project, I focus on a uniform CNN architecture and tune it accordingly to attain a certain desired test accuracy on the STL-10 dataset.

2.3.2 Transfer learning

Instead of building a CNN from scratch, we may use a pretrained Imagenet model, such as VGG16, VGG19, InceptionV3, ResNet50, etc that have shown state-of-the-art performances on image classification tasks. To save the time-consuming and resource-intensive training process, the best performing weights are saved and made available for us to use. Keras makes it even easier for us. We can load the model with the feature extraction layers and the pretrained weights. Running the images through the network (predicting features) provides us the *bottleneck*

features for the image-set. A set of fine-tuning layers with a classification layer that fits to our purpose would result in an image-classifier that considers

- the general features trained from a larger pool of images as well as
- the specifics of the classes in context.

2.4 Data augmentation with the help of transfer learning

Clearly a classfier based on transfer-learning is more reliable than a model from scratch, and hence can be used to label the unlabeled data. A refinement of that classification based on soft-max probabilities may lead to an augmentation of training data towards improved performance for the model from scratch.

2.5 Benchmark

Classification results including the state-of-the art for the STL-10 dataset are as follows (taken from Rodrigo benenson's web-site at [1]):

Result	Method / Publication title	Reference
74.33%	Stacked What-Where Auto-encoders	Graham [7]
74.10%	Convolutional Clustering for Unsupervised Learning	Springenberg et al. [13]
73.15%	Deep Representation Learning with Target Coding	Yang et al. [16]
72.80%	Discriminative Unsupervised Feature Learning with Convolutional Neural Networks	Dosovitskiy et al. [5]
70.20%	An Analysis of Unsupervised Pre-training in Light of Recent Advances	Le Paine et al. [8]
70.10%	Multi-Task Bayesian Optimization	Swersky et al. [14]
68.23%	C-SVDDNet: An Effective Single-Layer Network for Unsupervised Feature Learning	Wang and Tan [15]
68.00%	Committees of deep feedforward networks trained with few data	Miclut et al. [11]
67.90%	Stable and Efficient Representation Learning with Nonnegativity Constraints	Lin and Kung [9]
64.50%	Unsupervised Feature Learning for RGB-D Based Object Recognition	Bo et al. [2]
62.32%	Convolutional Kernel Networks	Mairal et al. [10]
62.30($\pm$1)%	Discriminative Learning of Sum-Product Networks	R. Gens et al. [6]
61($\pm$0.58)%	No more meta-parameter tuning in unsupervised sparse feature learning	Romero et al. [12]
61.00%	Deep Learning of Invariant Features via Simulated Fixations in Video	Zou et al. [17]
60.10%	Selecting Receptive Fields in Deep Networks	Coates and Ng. [3]

Figure 4: Some best known results on the STL-10 dataset

I would like to obtain a performance that falls within the range in the above list (achieve around 60% accuracy on the test dataset) with a convolutional neural network built from the

scratch with tuned hyperparameters but without preprocessing or augmentation of the training dataset.

3 Methodology

3.1 Data processing and Implementation

We normalize data by dividing the pixel values by 255
- Training data mean before (after) normalizing: 109.919187146 (respectively 0.431055635865)
- Test data mean before (after) normalizing: 109.80558034 (respectively 0.4306101189820

3.1.1 Design of a Convolutional Neural Network from Scratch

We intend to design a simple but uniform architecture of a Convolutional Neural Network (CNN) from the scratch with the following properties:

- Equal number of channels in all the convolutional layers
- Same Kernel shape for all the convolutional layers
- Each convolutional layer is followed by a Batch normalization and 2x2 MaxPool layers
- One fully connected layer before the output layer and the height of this layer is proportional to the number of output channels in the convolutional layers
- Each Flattened or fully-connected layer (except the softmax output layer) contains a dropout layer with the same keep-probability

We define the following variables:

- N = number of convolutional layers
- m = number of channels in each of the N convolutional layers
- t for the kernel shape ($t \times t$)
- $m \times 16$ = size of the only fully-connected layer before the output layer

Upon tuning over the following values

- $N \in \{3, 4\}$; $m \in \{32, 48, 64, 80, 96\}$, and $t \in \{3, 4, 5\}$,

we have found consistantly above 60% test accuracy with $N = 3$, $m = 80$, and $t = 4$, which was one of the primary goals of this project.

3.1.2 Feature extraction and fine-tuning using pre-trained Imagenet model

We use the ResNet50 pre-trained Imagenet model to extract features from STL-10 training dataset and then finetune it to use it as a classifier of the STL-10 dataset using the steps given below:

- Computed the bottleneck features of the original training subset (5000 x 96 x 96 x 3), which included the following steps:

◇ Loading the feature-extraction part of the ResNet50 model with pretrained weights

◇ Resizing the images to (197 x 197 x 3) which is the minimum requirement for ResNet50

◇ Running the training subset through the network to predict features (5000 x 2048)

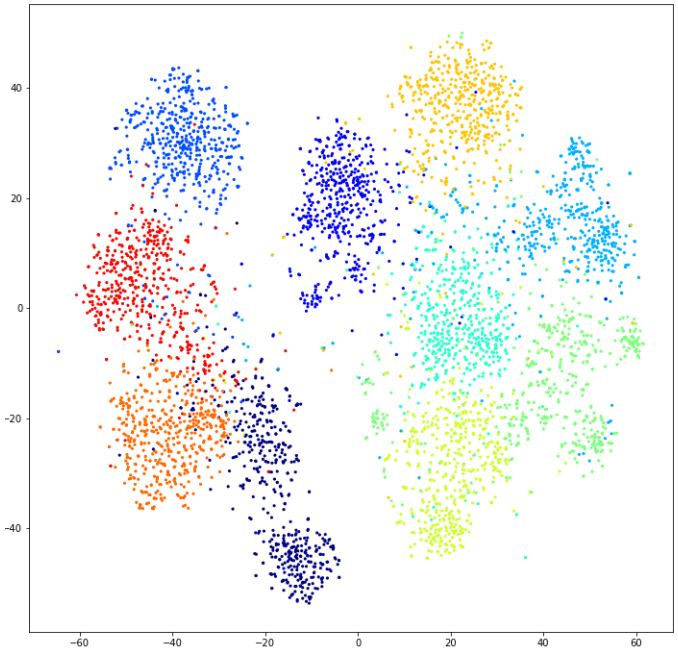


Figure 5: TSNE embedding of bottleneck features for Training set (5000 x 96 x 96 x 3)

The bottleneck features of ResNet50 provide encouraging result as shown in the plot above where points representing same-class objects are in the neighborhood on the 2D plot. Thus training a classifier on the bottleneck features would provide better validation and test performance.

- Used fine-tuning layers with bottleneck features as input and softmax-classifier as output

  ◇ One fully-connected layer (500) followed by batch normalization and dropout (0.5)

  ◇ Six fully-connected layers (each of size 256) each followed by batch normalization

Fitting the network for 100 epochs result in 93.85% accuracy on the bottleneck features of the test subset (8000 x 96 x 96 x 3).

3.1.3 Labelling unlabeled STL-10 data using finetuned ResNet50 model

Since the unlabeled dataset contains objects beyond the 10 object classes available in the labeled STL-10 dataset, it is natural that there are serveral approximations done by the ResNet50-based classifier (which is also only 93.85% correct against the test subset of STL-10 dataset). Now, the important question is how close is the approximation with respect to the features of the corresponding STL-10 label when the softmax probability for the labelling is close to 1.0, say more than 0.95. We list a few cases:

- Cat: Mongoose, Cougar, Raccoon, Owl, Rat, Squirrel, Lioness, Iguana, Turtle

- Horse: Donkey, Raindeer (with horns undistinguishable from background), Byson
- Deer: Goat, Alpaka, Bull with horns
- Dog: Bear, Wolf
- Monkey: Chimpanzee, Baboon, Gorilla

It appears that the softmax probability is higher if features are more and more similar to the corresponding STL-10 class.

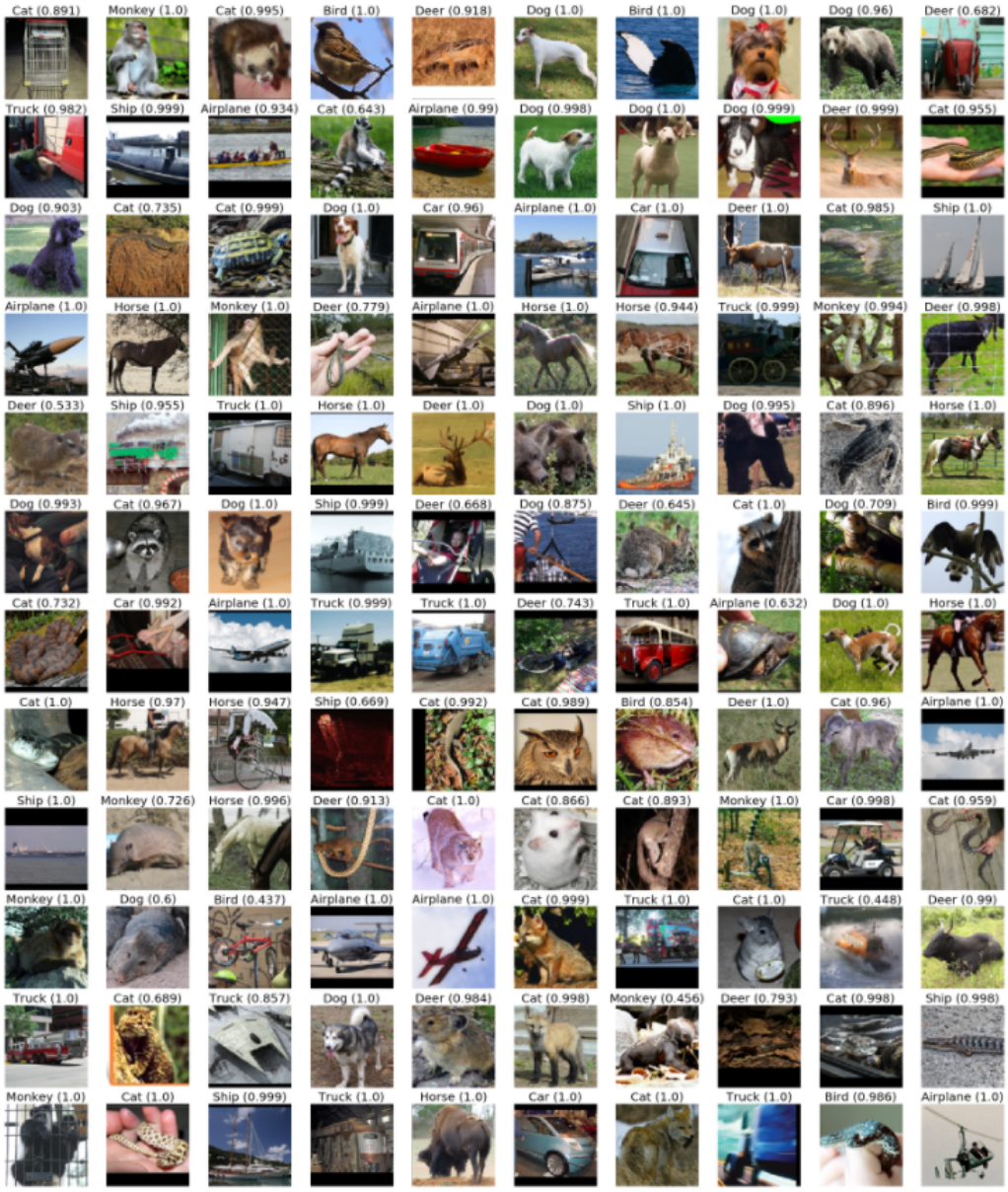


Figure 6: Labelling random images from STL-10 unlabeled subset

3.1.4 Labelling unlabeled STL-10 data using softmax threshold 0.999 or more

Even though the image distribution is similar but more diverse in the unlabeled STL-10 dataset, we have a number of approximations with very high probability (0.999 or more).

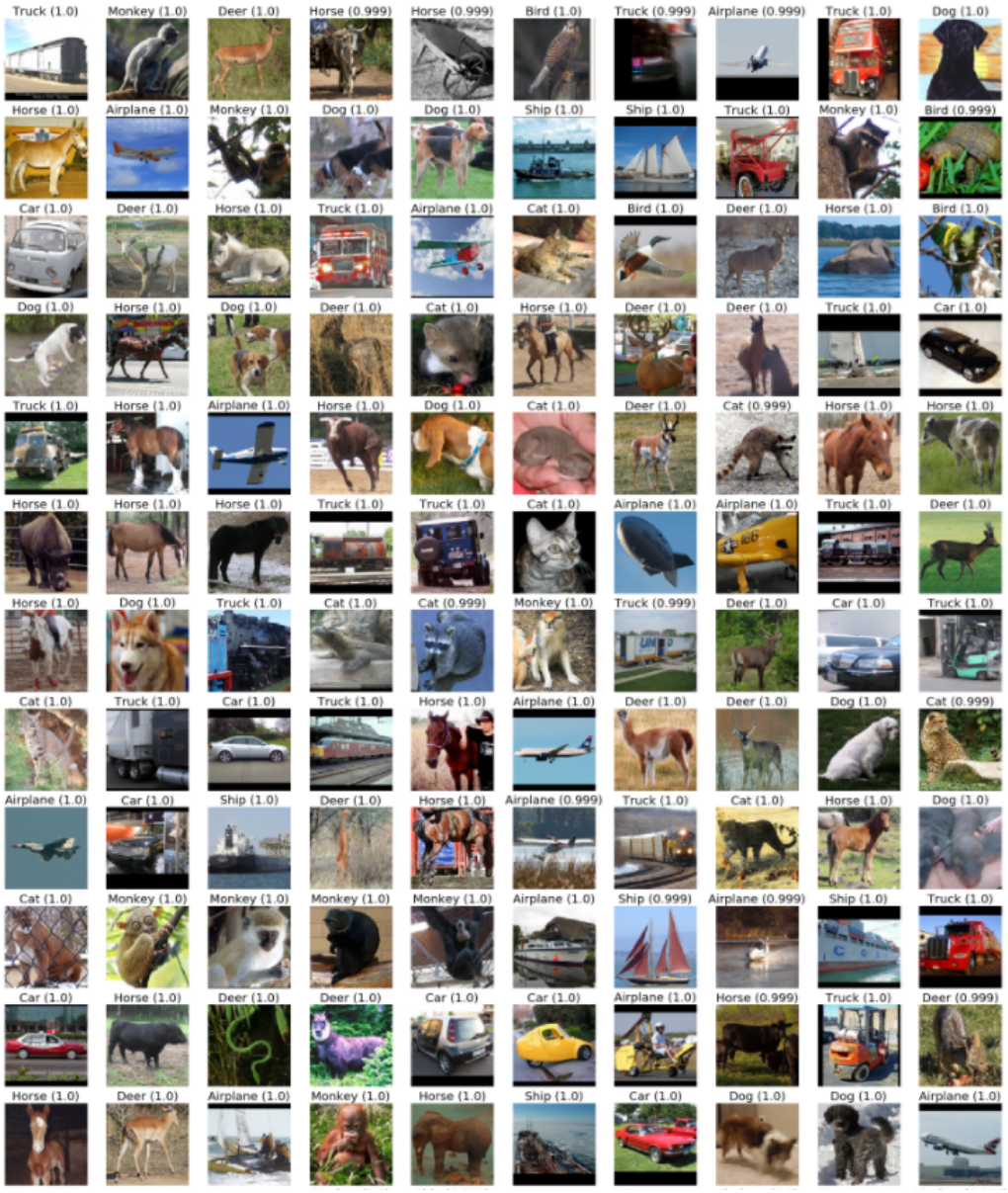


Figure 7: Labelling STL-10 unlabeled subset with softmax probability 0.999 or more

Let us look at a few such cases:
- Cat: Cheetah, Leopard, Raccoon, Lioness, Cougar
- Deer: Goat, Sheep, Alpaka
- Dog: Bear, Wolf, Panda
- Horse: Cow, Donkey, Rhino
- Monkey: Chimpanzee, Baboon, Gorilla

- Truck: Train, Jeep, Bus

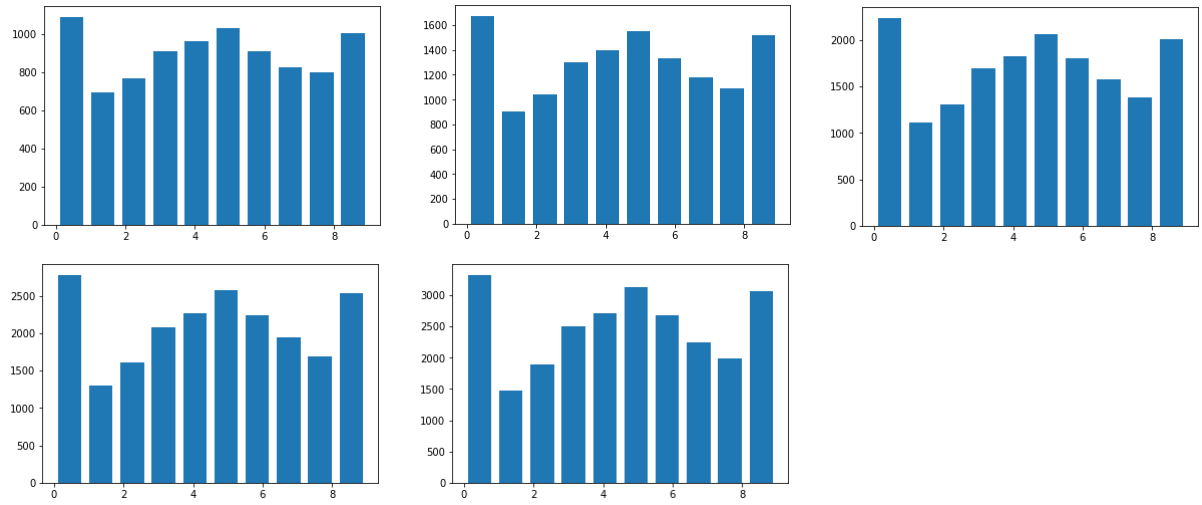
The test subset does not contain any Cheetah, Leopard, Raccoon, Lioness, or Cougar, but adding these images as cats in the training dataset may enhance and diversify the features of cats and improve performance on the test subset. A similar assumption may be applicable for the classes: dog, horse, monkey, deer, truck, etc.

Let us call this process: *Adding Imperfections to Compensate Inadequacies (AICI).*

3.2 Augmentation of training set and Refinement

Since we have a large unlabeled image dataset, we restrict our augmentation of the training dataset on adding images selected from that unlabeled dataset based on the assumptions discussed in the previous section. We gradually increase the training dataset size as follows:

$$(9000\text{x}96\text{x}96\text{x}3), (13000\text{x}96\text{x}96\text{x}3), (17000\text{x}96\text{x}96\text{x}3), (21000\text{x}96\text{x}96\text{x}3), (25000\text{x}96\text{x}96\text{x}3)$$



Surprisingly, the distributions are quite similar.

Fitting the CNN model from scratch on the augmented datasets is a resource-intensive undertaking. Fortunately, the availability of a GPU (GTX 1060) made it possible to run 50 epochs in each of the above cases within hours. But memory soon became an issue. As a result, popular augmentation techniques such as rotation, flip, and random zoom, shear and shift are not considered (not even on-the-fly augmentation through fit-generators due to limitations in time). Instead, the mission has been to make better use of the dataset itself, in particular, the unlabeled subset.

4 Results

4.1 Evaluation against benchmark models

Training set-size	Epochs	Training acc.	Training loss	Valid. acc.	Valid. loss	Test acc.
5000 (original)	50	96.100%	0.1259	63.300%	1.1916	61.600%
9000	50	96.85%	0.0939	65.667%	1.1868	67.200%
13000	50	96.15%	0.1165	70.425%	0.9556	68.925%
17000	50	97.07%	0.0880	72.059%	0.9687	71.638%
21000	50	95.09%	0.1491	73.667%	0.8308	73.238%
25000	50	96.95%	0.0887	74.200%	0.9341	75.638%

- The model obtains a test accuracy more than 60% on the original training subset
- Validation accuracy on the original as well as the augmented training datasets generalize really well on the training subset
- Test accuracy increases as more relevant images from the unlabeled dataset is added to the training subset
- Use of dropouts prevent overfitting as we see that the training accuracies remain between 95% and 97%. Tendency to overfit may be reduced by further tuning the learning rate and and regulerization constants.
- Since the training accuracy is not growing while achieving monotonously better test accuracy, we can consider the model as reasonably robust.

4.2 Justification

Based on the project outcome, we make the following observations:

- Adding imperfections in the training dataset does not make any negative impact on the model from scratch. As anticipated, imperfections have actually helped to diversify features for the dataset classes:
  - Example: Identifying a bear or a wolf as a dog
  - Example: Identifying a donkey or a cow as a horse
  - Example: Identifying a cheetah or a leopard as a cat

With only abount 75% accuracy, it is expected that on randomly selected images, there will be misclassifications. Yet, the model seems to perform quite well as in the set of internet pictures (resized to 96 x 96 x 3)

Figure 8: Classification of some random Internet pictures (resized to 96 x 96 x 3)

Note that we have a state-of-the-art performance using a model from scratch on the STL-10 dataset but only with help of transfer learning while understanding the unlabeled data. We have obtained this performance without applying popular preprocessing and image augmentations techniques.

5 Conclusion

5.1 Reflection

Due to limitations in time and computing resources, we could not perform further experiments, but we have achieved the goals of this project and found a constructive way to improve performance of a model from scratch to perform consistantly better on the STL-10 dataset. In other words, we have determined a scalable method to obtain better performance on the dataset.

5.2 Improvements

Since we have devised a process involving a basic model from scratch, we may consider the following to improve performance further:

- How we initialize the weights, learning rates, and other parameters? (say, we restart fitting with the previous best weight)
- Tune the structure of the uniform model from scratch with fit generators and grid-search
- Apply other pre-trained models (say, InceptionV3) to possibly obtain better classifier for the unlabeled dataset
- Given required resources are available, popular preprocessing and augmentation techniques can be used

References

[1] R. Benenson, Discover the current state of the art in objects classification. (Retrieved: 2018-03-27) *http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html*.

[2] L. Bo, X. Ren, D. Fox, Unsupervised Feature Learning for RGB-D Based Object Recognition, *ISER*, (2012).

[3] A. Coates, A. Y. Ng, Selecting Receptive Fields in Deep Networks, *NIPS*, (2011).

[4] A. Coates, H. Lee, A. Y. Ng, An Analysis of Single Layer Networks in Unsupervised Feature Learning, *AISTATS*, (2011).

[5] A. Dosovitskiy, J. T. Springenberg, M.n Riedmiller and T. Brox, Discriminative Unsupervised Feature Learning with Convolutional Neural Networks, *NIPS*, (2014).

[6] R. Gens and P. Domingos, Discriminative Learning of Sum-Product Networks, *NIPS*, (2012).

[7] B. Graham, Fraxtional Max-Pooling, *http://arxiv.org/abs/1412.6071*, (2015).

[8] T. Le Paine, P. Khorrami, W. Han, T. S. Huang, An Analysis of Unsupervised Pre-training in Light of Recent Advances, *ICLR*, (2015).

[9] T. H. Lin, H. T. Kung, Stable and Efficient Representation Learning with Nonnegativity Constraints, *ICML*, (2014).

[10] J. Mairal, P. Koniusz, Z. Harchaoui, C. Schmid, Convolutional Kernel Networks, *https://arxiv.org/abs/1406.3332*, (2014).

[11] B. Miclut, T. Kaester, T. Martinetz, E. Barth, Committees of deep feedforward networks trained with few data, *https://arxiv.org/abs/1406.5947*, (2014).

[12] A. Romero, P. Raveda, C. Gatta, No more meta-parameter tuning in unsupervised sparse feature learning, *https://arxiv.org/abs/1402.5766*, (2014)

[13] J. T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for Simplicity: The All Convolutional Net, *https://arxiv.org/pdf/1412.6806.pdf*, (2015).

[14] K. Swersky, J. Snoek, R. P. Adams, Multi-Task Bayesian Optimization, *NIPS*, (2013).

[15] D. Wang, X. Tan, Unsupervised Feature Learning with C-SVDDNet, *https://arxiv.org/abs/1412.7259*, (2014).

[16] S. Yang, P. Luo, C. C. Loy, K. W. Shum, X. Tang, Deep Representation Learning with Target Coding, *AAAI*, (2015).

[17] W. Y. Zou, S. Zhu, A. Y. Ng, K. Yu, Deep Learning of Invariant Features via Simulated Fixations in Video, *NIPS*, (2012).

[18] http://deeplearning.stanford.edu/wiki/index.php