Integrating Contrastive Language-Image Pretraining and Natural Language Similarity towards Robust Classification

Divi Schmidt*
UC Berkeley
divi@berkeley.edu

William Loo*
UC Berkeley
williamloo@berkeley.edu

***Abstract*—We introduce and analyze novel approaches and techniques to improve performance on the Tiny-ImageNet-200 dataset. This paper contributes the implementation of an invented Cross-Entropy-Similarity (CES) Loss along with distillation from CLIP features. We explore the benefits and trade-offs from using CES loss, along with the incorporation of CLIP features to improve classification performance on unseen and potentially adversarial images. Our classification model achieves 50.98% accuracy on the validation set through the use of a robust training approach, novel loss metric, and CLIP.**

I. Introduction

The goal of our research is to build a performant model for the image classification problem on the Tiny-ImageNet-200 dataset. The objective of the work besides achieving outright performance is to develop techniques to strengthen the robustness of our network to random, perhaps even adversarial perturbations. Our approach is centered around first minimizing the effects of adversarial perturbation, and then secondly attempting to reproduce the perturbations themselves. Additionally we develop two novel approaches to guide the model incorporating CLIP and Natural Language Processing components upon the labels during training to yield more relevant and robust label classifications. The evaluation of success in our work is based off model prediction accuracy, as well as how relevant predicted labels are to the ground truth when they are wholly incorrect by traditional metrics.

II. Related Work

Beginning with AlexNet in 2012, deep learning quickly became the default solution to large scale classification problems. GoogLeNet (2014) addressed the issue of propagating gradients through large networks and was able to improve performance with increased model depth. ResNet (2015) took this a step further, adding residual connections represented by the identity transformation to prevent gradients from diverging/vanishing and was able to train networks orders of magnitude larger than previous approaches; with serious performance gains. MobileNet (2017) took the ideas of residual connections and depthwise convolutions to build a parameter efficient network. EfficientNet (2019-2020) evaluates the tradeoff between model scale and performance by creating an easily scalable classification backbone.
In many of these methods, the main focus was the network architecture, which allowed them to perform very well on classification. However most recently, there has been work to re-imagine the rest of the pipeline. This is the main theme of our project, as we adopt these popular architectures, and augment them in several ways in order to improve training signal and generalization.

In re-imaging the image classification pipeline, there has been a big interest in using NLP methods in order to improve upon these classification tasks. Specifically, CLIP [1] performs contrastive learning on image and caption embeddings. This is more of an unsupervised method as they do not explicitly train for classification, but aim to create valuable embeddings of captions (which may or may not contain the image class in them), and images. Amazingly, this strategy allowed for powerful representations and a roundabout method for classification called prompt engineering. We draw much of our inspiration from this method as it has strong generalization capabilities.

III. Methods

A. Approach

1) Base Architecture: The base architecture, while not the focus of our innovative work, was important to ensure a stable and reliable platform for experimentation. Trials were conducted on different network constructions, and ultimately there was an emphasis placed on an architecture that takes into consideration the smaller resolution of the input data, being 64x64 pixel images. We chose U-Net to be the base architecture of our neural network. Our original project report placed emphasis on maintaining features at the original scale through parallel contractive and expansive layers with skip connections, along with a bilinear layer which is shown to speed up training and inference times.[2] Pretrained resnet18 layers are embedded into the architecture in an attempt to improve relative performance. With the original purpose of U-Net being for segmentation, the architecture was re-imagined for classification tasks.[3] Dropout layers with probability 0.5 are placed after every nonlinear activation layer within the

*Equal contribution

combined conv-relu layer groups to reduce overfitting and improve generalization.

B. CES Loss

Cross-Entropy-Similarity (CES) is an invented loss metric that has potential to improve the reliability and generalization of the network. We observed within the data that there are some classes with images with similar appearances, but differing labels (coral reef and brain coral, for example). Predictions which yield labels similar to the training data's ground-truth label should be proportionally rewarded, and we used a generic word2vec implementation to calculate keyword similarities.[4] Since word2vec takes in single words, a preprocessing step is to strip qualifiers from the label ('coral reef' for example, becomes 'reef', and 'brain coral' becomes 'coral' when comparing the labels). For labels with aliases and multiple options, the first option that can succinctly express the class label as a single word is used. Calculating the word2vec similarity between 'reef' and 'coral' yields 0.6339, which indicates some similarity between the two labels. Through word2vec embeddings, an additional metric to include similarity scores weighted with cross-entropy loss forms the CES loss metric:

$$CES\_loss = 0.8(-\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log \hat{y}_i) * 0.2(1 - word2vec\_score) \tag{1}$$

By measuring the word2vec similarity between labels, an additional heuristic is introduced to assist and guide the model by providing language context between labels to make more relevant predictions, which can yield label predictions that make sense to the end-user during inference.

C. Data Perturbations and Augmentations

By modeling potential perturbations as accurately as possible, data augmentation and perturbation are randomly applied to the training data to build robustness within the network. With the implementation of data perturbation (Random Salt and Pepper Noise, Random Gaussian Blur), the classification performance was observed to improve by at least 5 percent on the validation set. Data perturbations and augmentations are not applied to the validation set beyond normalization.

1) Random Salt and Pepper Noise: Salt and Pepper Noise is first randomly applied to 10 percent of training images. The goal of occasionally adding noise to training data is to have the models learn the patterns of the classes which the images are associated with and prevent overfitting on the training data. The choice of using RGB noise rather than traditional black/white noise is to simulate the variability the model might encounter during test time.

2) Random Gaussian Blur: Random Gaussian blur is randomly applied with 66 percent probability for each training image, with the square kernel dimensions within the options of (1,1) or (3,3). Kernel dimensions are kept odd to prevent undesired image shifting. Blurring is a denoising approach used to remove extraneous features and high frequency components which may not be helpful towards training. It is also considered a perturbation because it reduces the detail available. With less details available in the data, the goal is to have the network make the correct generalized predictions and discern pertinent details between images depicting the same class. The possibility of overfitting on training data is also minimized with blurring.

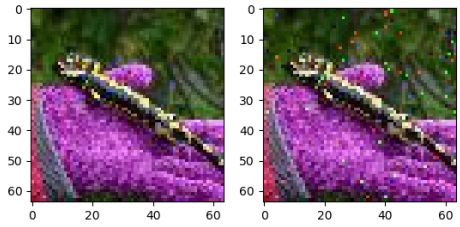

Fig. 1. The result of applying RGB salt-pepper noise perturbation.

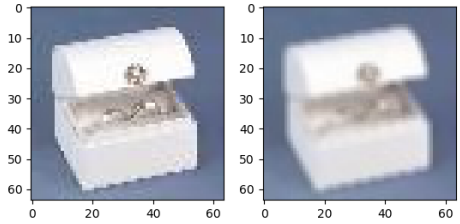

Fig. 2. The result of applying random gaussian blur with kernel size of (3,3).

3) Other Augmentations: Other augmentations used include Horizontal flipping of the training images, which increases the variability in the training data. Brightness Contrast Saturation, Hue, and Contrast Jittering is also applied to the training data in anticipation for potential perturbations. Normalization has been observed to speed up training and improve model generaliztion performance, so it is applied to the data during training, validation, and test phases.[5]

D. CLIP

The previous methods all have to do with augmentations. We augment our inputs using creative image transformations, and we augment our training signal by adding additional losses. Following along this theme, this method aims to augment our data through the use of the CLIP model from OpenAI [1]. Given the impressive representational power of CLIP's image embeddings, we aim to train a new model to

both reproduce these features as well as correctly classify them. We do this by adding a cosine similarity loss between the features obtained from clip and the features extracted from an intermediate layer in our model. In other words, we jointly train an encoder to produce similar features as clip, as well as a small classifier on top of these features. There are several interpretations of this approach. The first is that we are doing network distillation [6] on clip's image encoder. This allows us to "borrow" the insight gained from a very good model. We then add the classification loss on top of this, which allows us to fine tune these features explicitly for our classification objective. We can also see this as an additional augmentation. While modifying our images allows for a more diverse dataset, our images remain limited to image space. This method, however, aims to lift these images into a more meaningful feature space so that our classifier can perform better. We determine this feature space by using CLIP's embeddings and our classification signal.

IV. Results

A. CLIP

For our image encoder, we use ResNet18 with an additional fully connected layer that outputs 512 dimensional feature vectors. This was used instead of the base architecture described above in order to speed up training time. Our classifier is a four layer neural network with a hidden dimension of 256. Our method, named Clip Student, is both the encoder and classifier trained with clip loss (cosine similarity between image embeddings) and cross entropy loss. We also trained two additional models for comparison. Clip+Classifier is a 4 layer classifier that is trained using CLIP's image embeddings as input. Classifier is just the 4 layer classifier mentioned above, trained on normal images with no augmentations. All 3 methods are trained for 50 epochs. Results are included in the table below (I). The 4 layer classifier performed the worst by far, only reaching 10% validation accuracy, proving just how powerful a good embedding can be. Clip Student outperformed the vanilla classifier by far, gaining 30.61% validation accuracy. This, however, did not beat Clip only, which reached a validation accuracy of 52.24%. While we did not train the clip model at all for our classification task, it was able to produce meaningful features for our classifier to use. We expected the Clip Student to perform on a similar level as it was trained to not only reproduce these features, but improve upon them, however this proved not to be the case. The gigantic dataset and massive compute used by OpenAI gave the Clip only model a larger advantage over our image encoder, despite the direct supervision for our task. Note that we do not compare to the classification method used by OpenAI [1] as the learned representations were our focus in this project, not prompt engineering.

B. CES Loss

With the addition of the word2vec library, the implementation of the CES loss metric was plagued with cuda and cupy issues. The library could use CPU-only but the training times were suboptimal. Some gpu problems were fixed but the loss calculations still appeared to be CPU-bound, which causes training to be slower. We hope to explore ways to accelerate the loss calculation by completing the integration of the loss metric to work with the GPU.
Our training was originally set for 20 epochs, but we managed to train 12 epochs over 2 days, and have placed the intermediate results in the table below (I). Because of this, the model using the baseline cross-entropy loss was also evaluated based on its result at 12 epochs, despite being able to achieve approximately one percent higher in validation accuracy if it were allowed to train for 8 more epochs. Some improvement might be fairly assumed for the model using the CES loss, but we wouldn't be able to verify this assumption without actually completing the training. Below are some test set predictions from the model using CES loss after 12 epochs of training.

Speaking to the results, we observe an improvement over the baseline at the 12 epoch mark. Additional validation was also performed to quantify the improvement of the model using CES loss by averaging the word2vec distances between labels for misclassifications upon the validation set. For the model using CES loss, this average word2vec distance for misclassified labels would be 0.3139 and and for the baseline model using vanilla cross-entropy, 0.3530. We observe a quantifiable improvement in using the CES loss metric.

Fig. 3. Test Set Predictions for the model using CES loss

C. Training Experiments

Below is a table of our results comparing the 5 strategies. Note that the classifier is mainly for comparison purposes. Further down are images depicting our training accuracy and loss curves.

D. Difficult Classes

Our results analysis wouldn't be complete without recognizing the models' shortcomings. Some of the most difficult classes (see histograms: 9 and 10) appeared to have come from smaller objects. Viewing an example test image belonging to the syringe class for example, shows that these objects are

Strategy	Train Loss	Validation Loss	Train Acc.	Validation Acc.
CE loss	0.9244	1.7809	67.0000%	50.2200%
CES loss	0.6905	1.9986	77.2250%	50.9800%
CLIP Student	0.2185	3.266	59.9720 %	30.61 %
CLIP+Classifier	0.5451	2.505	53.135 %	52.24 %
Classifier	3.552	4.950	12.921 %	10.16 %

TABLE I
TRAINING RESULTS ACROSS DIFFERENT STRATEGIES

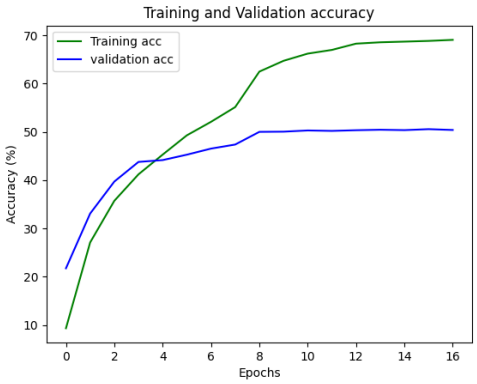

Fig. 4. Train Accuracy with Cross-Entropy Loss

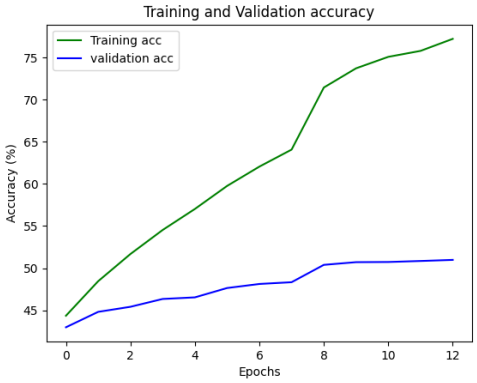

Fig. 5. Train Accuracy with Cross-Entropy-Similarity Loss

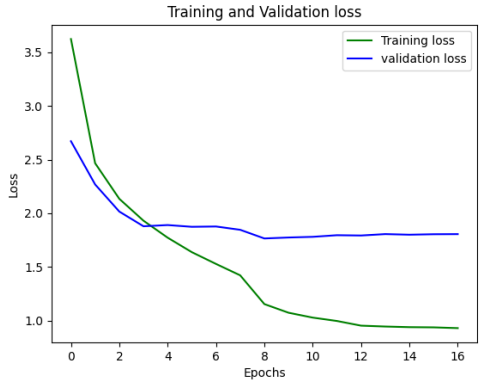

Fig. 6. Train Loss with Cross-Entropy Loss

Fig. 7. Train Loss with Cross-Entropy-Similarity Loss

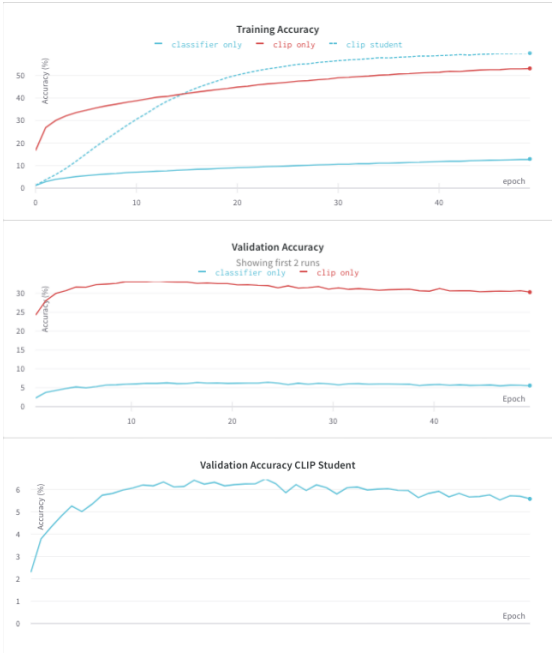

Fig. 8. Train and Validation Accuracy Across our Three CLIP strategies.

usually a small part of the image. Accompanied with the syringe is usually another entity in the scene, which can cause a classifier with no spatial context to become confused. This situation could be improved by incorporating the bounding boxes and defining a region of interest (ROI) to focus the classifier.

V. CONCLUSION, LESSONS LEARNED

With consideration to the improvements from applying augmentations and perturbations upon the data, there was a noticeable increase in validation accuracy, which shows that our approaches for guiding the model predictions to be generalizable have been working. In practice, we didn't observe a huge difference between using the CES loss metric as opposed to the vanilla cross-entropy metric, but we also made fewer passes over the training set than we would've liked

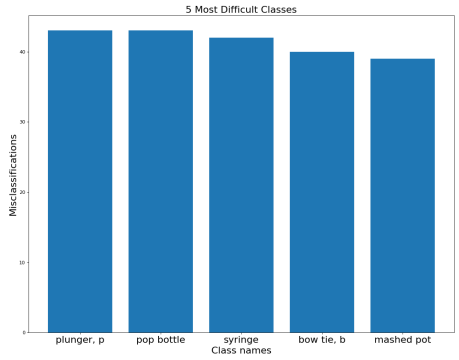

Fig. 9. Five Most Misclassified with Cross-Entropy-Similarity Loss

Fig. 10. Five Most Misclassified with Cross-Entropy Loss

due to the GPU issues we encountered with the CES metric. In its current state, the CES loss is shown to be a performant metric in terms of classification performance, although the training times have made using it infeasible at least until it is able to utilize GPUs effectively.

In regards to our explorations with CLIP, our novel method showed large improvements over our baseline classifier, proving that CLIP features are helpful training signal. However, the model we chose to learn from CLIP was not of the same caliber as CLIP itself. Although this was expected, we did hope that the classification signal would help improve our model for our specific classification task, which did not seem to be the case. The high performance of CLIP itself on our task is a testament to the impressive compute and data storage of OpenAI, as well as the novel training strategy. In addition, our experiments with this model show just how important good features can be.

VI. Team Contributions

The percentage breakdown is 65-35 between William and Divi. William mainly focused on the CES Loss and data augmentation methods, while Divi worked on the methods relating to CLIP.

References

[1] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[2] R. K. Pandey, A. Vasan, and A. G. Ramakrishnan, "Segmentation of liver lesions with reduced complexity deep models," 2018.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[5] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Normalization techniques in training dnns: Methodology, analysis and application," 2020.

[6] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.