CS 182 Vision Project - Generalizable Classifiers

Spring 2020

Kevin Liu
University of California, Berkeley
kdliu00@berkeley.edu

Oscar Ortega
University of California, Berkeley
oscar.g.ortega.5@berkeley.edu

Sofia Paganin
University of California, Berkeley
paganin@berkeley.edu

Abstract

We attempt to construct a generalizable classifier on the Tiny ImageNet dataset. Our approach consists of constructing two candidate models that will serve as members in a final stacking ensemble. We experiment with data augmentation to improve generalizability and optimization tuning to improve overall performance. Our results demonstrate that good data augmentation is the strongest singular method to improve model performance and generalizability.

1 Problem Statement

Although modern computer vision has become quite adept at discerning and classifying objects from image data, state of the art systems often require large data sets to train from and are difficult to generalize. The human vision system learns faster and is far more resilient to environment perturbations. The latter weakness has been amplified with the emergence of adversarial machine learning, where artificial pixel perturbations, indiscernable by the human eye, can cause an image to be misclassified entirely by a neural network.

Our goal is to build a robust image classifier on top of the Tiny ImageNet data set. This data set contains 200 classes of 64x64 RGB images. There are a total of 100,000 training images, 10,000 validation images, and 10,000 test images, split evenly among the classes. The training and validation sets have accompanying true labels, whereas the test set does not. Performance will be gauged with cross-validation accuracy on the validation set and the difference in accuracy between the training and validation sets. We also consider the length of training time an important factor due to our lack of compute resources.

Successful solutions will have a great impact on the training and real world performance of computer vision systems. A generalizable classifier will require less training data, be faster to train, be more transferable to other contexts, and be more consistently accurate in edge case scenarios. Many use cases would benefit from such improvements, for example, automated medical imaging [7] and facial recognition [12].

2 Tools

We decided to use the Keras library on top of Tensorflow. The main benefits of Keras are its modularity and ease of use. These two factors allow for rapid iteration. Our goal was to experiment with several existing architectures before combining a select few in a stacking ensemble. A wide variety of prebuilt architectures are readily available in Keras, including ResNet [2], VGG [8], InceptionNet [10], etc.

Keras also supports a wide variety of data augmentation through its ImageDataGenerator class, which performs image modifications during training, eliminating the need for heavy preprocessing. In addition, Keras' functional API allows for trivial construction and training of ensemble models with a DAG[1] structure.

[1] Directed acyclic graph

We also decided to use PyTorch for our GAN implementations. PyTorch is similarly modular to Keras, but uses lower level building blocks that make it possible to create very custom architectures that do not necessarily operate in a standard feed-forward fashion, such as GANs.

3 Data Augmentation

Figure 1: Augmentation Examples

The ImageDataGenerator class in Keras features various forms of data augmentation. The data augmentations we decided to perform include brightness, rotation, shear, zoom, flipping, shifting, channel shifting, and channel shuffling. We used the reflection fill mode to fill any empty space caused by these augmentations. Channel shuffling involves randomly swapping the RGB channel values across an entire image for a proportion of the images. See figure 10 for more examples.

3.1 Augmentation Classes

We experimented with several combinations of augmentations, the following of which we used in our fully trained candidate models. For augmentation classes A and B, the image pixel values were normalized to the range $[0, 1]$ during training time and not during preprocessing. As such, the channel shift ranges for these augmentation classes are in terms of RGB values in $[0, 255]$.

3.1.1 Class A

This augmentation class was only used on the first iteration of AlphaNet. We initially decided not to use any augmentation techniques that affected the relative orientation of pixels, such as rotation and shear, out of concern for the noise it would result in due to the low resolution of the images.

- Width Shift Range: $\pm 20\%$
- Height Shift Range: $\pm 20\%$
- Channel Shift Range: $+60$
- Channel Shuffling Proportion: 100%

3.1.2 Class B

The range of augmentation was increased as an attempt at addressing an overfitting issue during initial experiments with BravoNet. We found that this class of data augmentation noticeably improved the performance of AlphaNet with virtually no changes in architecture, thus resulting in AlphaNet_v2. We briefly experimented with upscaling the images from 64x64 to 72x72 using Lanczos resampling in AlphaNet_v3 and BravoNet. This resulted in very little improvement but much longer training time. We addressed this by decreasing the size of the output dense layers from 2048 to 1024 in AlphaNet_v3 and subsequent versions with no noticeable effect on performance.

- Rotation Range: $\pm 30°$
- Brightness Range: $[0.3, 0.7]$
- Shear Range: $\pm 0.2°$
- Zoom Range: $\pm 20\%$
- Horizontal Flipping Proportion: 50%
- Width Shift Range: $\pm 20\%$
- Height Shift Range: $\pm 20\%$
- Channel Shift Range: $+90$
- Channel Shuffling Proportion: 100%

3.1.3 Class C

We surmised that more generalizable features would be learned from stronger shear and zoom. To balance this out we removed or reduced some

other data augmentations. Upon further inspection, it appeared that brightness and channel shifting resulted in similar effects, and thus we opted to remove brightness shifting. This augmentation class significantly improved performance with no architecture changes.

- Shear Range: $\pm 20°$
- Zoom Range: $\pm 40\%$
- Horizontal Flipping Proportion: 50%
- Width Shift Range: $\pm 20\%$
- Height Shift Range: $\pm 20\%$
- Channel Shift Range: $+90$
- Channel Shuffling Proportion: 20%

3.2 DCGAN

Figure 2: Generated Images from DCGAN

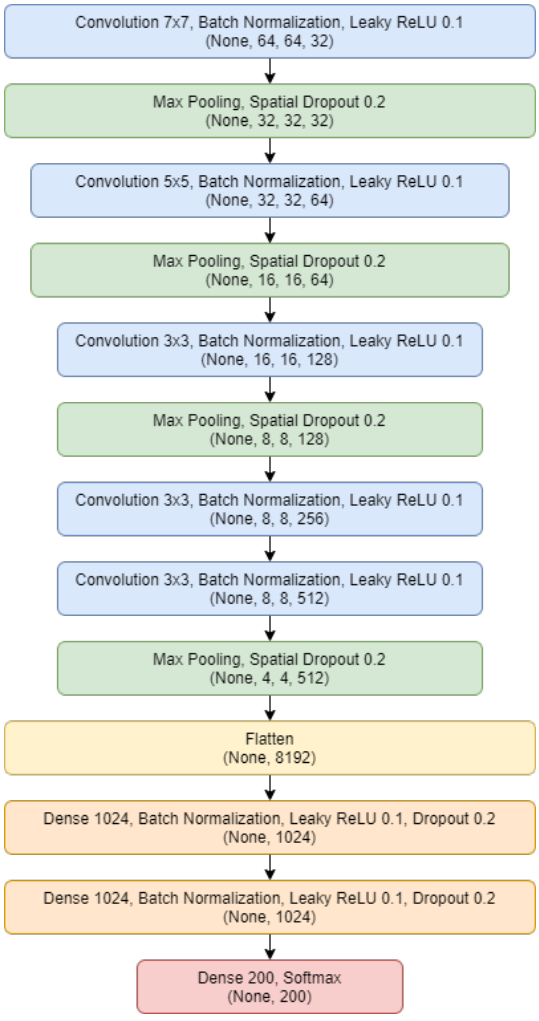
Because the Tiny ImageNet data set only had a limited amount of images, we wanted to try generating more images. We decided to use a Deep Convolutional Generative Adversarial Network [6] to do so. We based our implementation off of the model presented by Sebastian Nowozin[2] in his September 2018 Madrid lecture. The generator and discriminator have similar architectures of convolution layers with batch normalization layers sandwiched in between. However, the generator starts off with a single feed-forward layer and has 4 deconvolution layers before ending with regular convolution, and the discriminator architecture consists of 4 convolutions and ends with 2 feed-forward layers with another batch normalization layer in between.

[2]Sebastian.Nowozin@microsoft.com

We trained for 200 epochs at about 320s per epoch using Google Cloud Platform's Nvidia GPU. The images produced were not outstanding; it was difficult to see what they might correspond to in the real world. Unfortunately, the DCGAN was missing the essential class-conditioned aspect for the images to be used in training. As a result, we were unable to add generated images to our training set and did not have enough time to retrain our DCGAN.

4 Standard Convolution

Figure 3: AlphaNet_v4 Architecture



Our first design method was to use standard convolution layers with batch normalization, dropout, leaky ReLU, and max pooling as a baseline model. The architecture has largely remained the same across versions. Inspiration was taken from the AlexNet [4] architecture.

4.1 AlphaNet

Our first attempt used class A data augmentation. Training time was about 173s per epoch for a total of 300 epochs. Validation accuracy converged to about 36%, although for the vast majority of epoch results, training accuracy was noticeably worse than validation accuracy. Figure 11 shows that the validation loss suffered from severe fluctuations. We believe this is reflective of low confidence in the model. Dropout was set to 0.4 for convolution layers and 0.5 for dense layers. See figure 11 for full results.

4.2 AlphaNet_v2

This version used class B data augmentation, early stopping with a patience[3] of 15 epochs, and learning rate reduction with a factor of 0.1 and a patience of 10 epochs. The model initially had difficulty learning with slow convergence. Thus, we lowered the dropout to 0.2 across all layers for this and subsequent versions. Training time was about 123s per epoch for a total of 133 epochs. Validation accuracy converged to about 38.4%. Learning rate was reduced on epochs 81 and 128, but only the first reduction showed a noticeable jump in accuracy. See figure 12 for full results.

4.3 AlphaNet_v3

This version used class B data augmentation and upscaled 72x72 images. Early stopping and learning rate reduction were identical to AlphaNet_v2. Learning rate was reduced on epochs 88, 113, and 131, but only the first reduction showed a noticeable jump in accuracy. Training time was about 390s per epoch for a total of 136 epochs. Validation accuracy converged to about 40%. See figure 12 for full results.

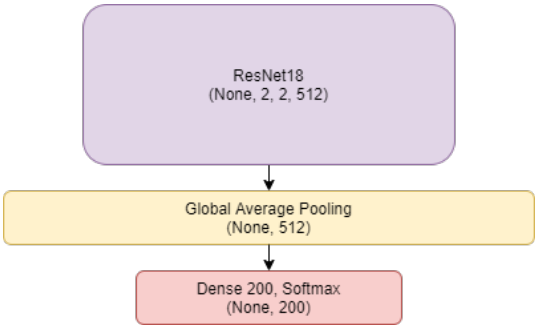
4.4 AlphaNet_v4

This version used class C data augmentation, early stopping with a patience of 10 epochs, and learning rate reduction with a factor of 0.316 and a patience of 5 epochs. Learning rate was reduced on epochs 47, 67, 81, 91, and 97, but only the first reduction showed a noticeable jump in accuracy. Training time was about 95s per epoch for a total of 102 epochs. Validation accuracy converged to about 47.7%. We observed a noticeable gap between the training and validation accuracy, with training accuracy roughly 8% worse overall. See figure 14 for full results.

[3]Number of epochs observed with no improvement in validation accuracy before triggering callback

5 Residual Networks

Figure 4: BravoNet_v2 Architecture



Our second design method was to use an existing architecture and employ transfer learning based on ImageNet weights. We decided to use the ResNet18 [2] architecture, one of the best performing architectures on image classification and readily available in Keras. However, we observed poor convergence with the use of ImageNet weights despite retraining several output layers. We surmised that the convolution layers, which were trained on higher resolution images, had a hard time with 64x64 Tiny ImageNet images. Retraining the weights from scratch demonstrated improvements in training time and performance. The architecture is identical across all versions.

5.1 BravoNet

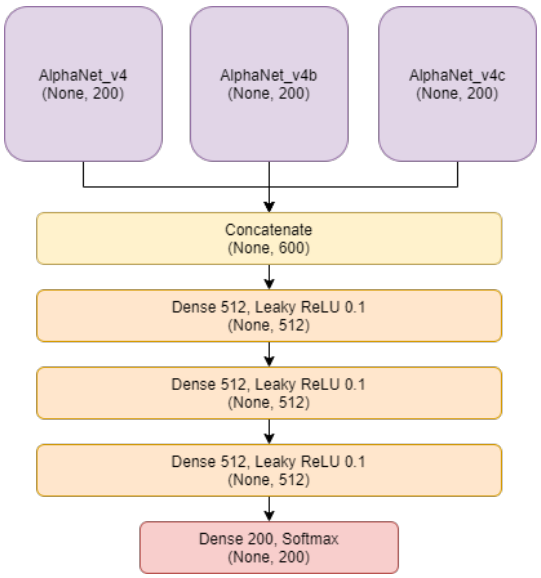
Our first attempt used class B data augmentation and upscaled 72x72 images. Early stopping and learning rate reduction were identical to AlphaNet_v2. Learning rate was reduced on epochs 67, 81, 92, and 105, but only the first reduction showed a noticeable jump in accuracy. Training time was about 163s per epoch for a total of 110 epochs. Validation accuracy converged to about 41.6%. We observed severe overfitting by the model, implying that class B data augmentation was not enough to learn generalizable features. See figure 18 for full results.

5.2 BravoNet_v2

This version used class C data augmentation. Early stopping and learning rate reduction were identical to AlphaNet_v4. Learning rate was reduced on epochs 21 and 41. Both reductions showed a noticeable jump in training accuracy but only the first showed a noticeable jump in validation accuracy. Training time was about 118s per epoch for a total of 46 epochs. Validation accuracy converged to about 43.5%. Although the overfitting was reduced, the validation accuracy did not improve by much, indicating that the model learned features less specific to the training set but not enough to be very generalizable. See figure 19 for full results.

6 Stacking Ensemble

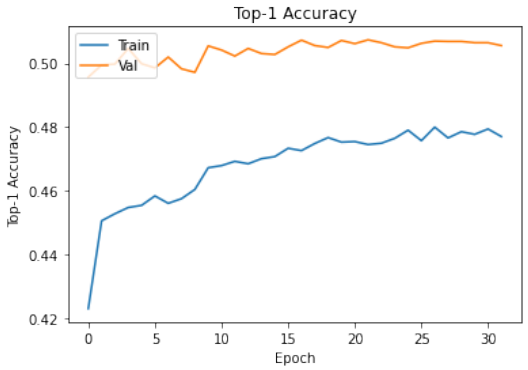
Figure 5: AlphaStack Architecture



For our final model, we selected the best performing candidate models and combined them into a stacking ensemble model. This involves concatenating the raw outputs from member models and feeding them through several dense layers before returning a final classification. Our initial trials used the best performing version from each series, AlphaNet_v4 and BravoNet_v2. However, this appeared to only average the candidate performances rather than improving upon them as expected. In addition, modifying the data augmentation scheme only worsened performance.

6.1 AlphaStack

Figure 6: AlphaStack Accuracy



We took our best architecture, AlphaNet_v4, and retrained it twice under the same parameters, resulting in AlphaNet_v4b and AlphaNet_v4c. See figure 15 and 16 for full results. The average validation accuracy among the v4's was 48.2%. We combined them as seen in figure 5, froze the model weights, and trained the ensemble for 32 epochs at about 97s per epoch. Our final validation accuracy reached 50.7%, measurably better than the best performing member model. See figure 17 for full results.

7 Optimizer Tuning

Table 1: Tuning Results for BravoNet_v3

Optimizer	Val. Accuracy
Adam	28.6%
RMSprop	36.3%
Adadelta	40.7%
SGD	**40.8%**

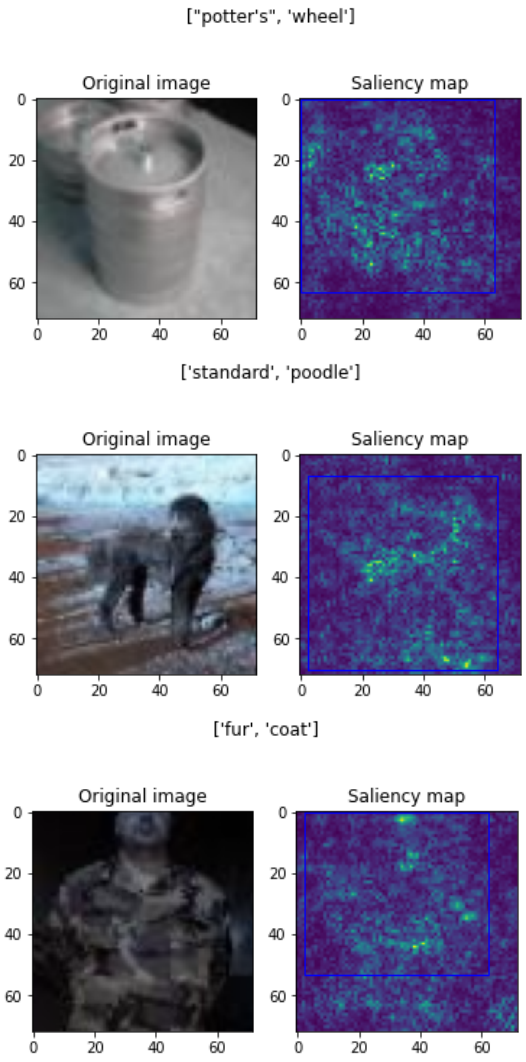
We used random search in the range $[0.01, 0.0001]$ to tune learning rate on Adam [3], RMSprop, Adadelta [13], and SGD. Performance was measured with final validation accuracy after 5 epochs. Our results showed that SGD and Adadelta had the best performance overall, but this did not hold true when we retrained our models with them. We believe this error is due to the small number of epochs per trial as well as not clearing the training session between trials. As a result, all of our models were trained with Adam and an initial learning rate of 0.001.

8 Explainable AI

Since many methods for classification are black box systems where not even the designers can produce reasoning for the model's decision, it can be difficult to understand model predictions. We used saliency, PCA, and t-SNE to help visualize how our models may interpret the image data.

8.1 Saliency

Figure 7: Saliency and Bounding Boxes



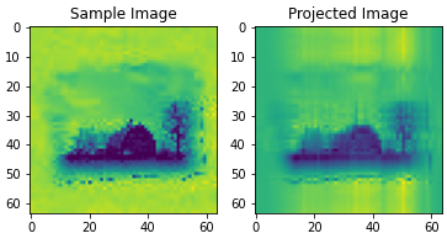
One method of explaining the AI's decision is to use a saliency map. The saliency map highlights areas of the images that the model pays more attention to. These should correspond to areas that correspond to where the item to be classified lies in the image space.

There are many different implementations of saliency maps. The ones here use the keras-vis library. We've also plotted the given bounding boxes onto the saliency maps to see if they correspond. Here, it is clear that the bounding boxes were not well-defined (this is not surprising because they were machine-generated), so we did not use them in our classification.

In the first two images, the saliency map demonstrates a general focus on the item we want to classify. In the third image, we can see that the model focuses explicitly on certain areas which correspond to the face and neck. The model incorrectly classifies this military man as a fur coat. This might be due to the fact that the model recognized a human (by the face) wearing clothing.

8.2 PCA

Figure 8: PCA on Red Color Channel



Principal Component Analysis is a dimensionality reduction technique used to emphasize the variation within the feature space of a set of points by bringing out the strong features within a dataset by retaining the features that "explain" the highest proportion of variance from point to point. Formally given a data matrix $X \in \mathbb{R}^{n,f}$ choosing to retain f' of the features of highest variance is equivalent to solving for the following optimization problem.

$$\min_{X'} \|X - X'\|_F^2 : \text{ s.t. } \text{rank}(X') = f' \qquad (1)$$

We can compute the percentage of retained variance from X' by computing the proportion of retained singular values of X' to the proportion of singular values of X. More formally, if we let $\sigma_1, ..., \sigma_f$ be the singular values of X, we can compute the retained variance as follows.

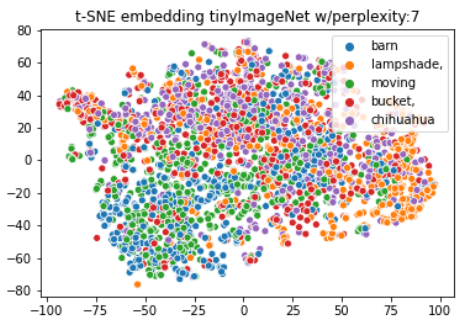
$$\text{RetVar}(X') = \frac{\sum_{i=1}^{f'} \sigma_i}{\sum_{i=1}^{f} \sigma_i} \qquad (2)$$

We decided to perform PCA on our training image data to verify whether we could retain a

high proportion of the variance while reducing the dimensionality of our data set by a significant margin. As it is the case our Tiny ImageNet images belong in a 12888 dimensional space and a 4096 dimensional space across color channels, we decided that this would be valuable in future training. Promisingly, across all color channels retaining only about 500 dimensions led to a retention of about 90% of the variance and the same held across red, blue and green color channels. The results of our experimentation can be seen in the appendix in figure 20. Given the set of features with high variance after performing PCA, we can perform a more detailed analysis of what these features correspond to. For individual images across individual color channels, the results appeared promising as well as we only needed to retain 10% of the components to retain 95% of the variance within the color channel as shown in figure 8. This tells us that 95% of the information in this particular image was encoded in the 6 component directions specified, which gives us insight into the portions of an input that are more valuable to the network we train.

8.3 t-SNE

Figure 9: t-SNE Visualization



To help visualize the feature-space of the Tiny ImageNet dataset, we performed a t-distributed Stochastic Neighbor Embedding, or t-SNE [5] of a subset of the images. We first performed PCA on the sample of the Tiny ImageNet dataset, retaining 90% of the explained variance of the dataset to speed up the time to perform the embedding. We then performed two grid searches through the dataset, first between the values $[1, 50]$ and then between values $[1, 10]$, before deciding on the particular embedding. For the purposes of clarity, we have also decided to to run the embedding on a random sample of 5 of the 200 Tiny ImageNet classes. Figure 9 shows that the sample classes have very rough feature clusters which suggests the continuous nature of the feature space. This also suggests the inherent challenge of performing classification on this dataset as the t-SNE embedding suggests there is a high-degree of similarity among the feature spaces of different classes.

9 Results

Table 2: Model Performance Comparison

Model	Training Time	Val. Accuracy
AlphaNet	14.4 hrs.	36%
AlphaNet_v2	4.5 hrs.	38.4%
AlphaNet_v3	14.7 hrs.	40%
AlphaNet_v4	2.7 hrs.	47.7%
AlphaNet_v4b	2.5 hrs.	47.6%
AlphaNet_v4c	3.2 hrs.	**49.4%**
AlphaStack	**0.9 hrs.**	**50.7%**
BravoNet	5 hrs.	41.6%
BravoNet_v2	**1.5 hrs.**	43.5%

Most notably, our models with the best validation accuracies are also some of the fastest to train. This leads us to believe that good convergence does not necessarily require longer training, and that the right architecture and data augmentation can often allow a model to reach a better optima sooner. We also observed that combining separately trained models with identical parameters and architectures into a stacking ensemble provided appreciable performance improvements with no substantial increase in computational cost.

10 Lessons Learned

The Tiny ImageNet data set is tricky to deal with given its low image resolution. Many of the images were difficult to classify, even by our own eyes. The results of t-SNE and PCA are reflective of the vast dimensionality of the image space and the heavy presence of noise. Data augmentation proved to be the best way to improve model performance. This was exemplified by AlphaNet_v4's validation accuracy improvement of 9.3% over AlphaNet_v2, with which it shared the same architecture but used better data augmentation.

Our experiments with stacking ensembles demonstrate having member models with differing performance levels and architectures may hinder the overall performance of the ensemble. We observed the most improvement in performance when member models had identical architectures and were trained under the same parameters. In addition, ensemble top-3 and top-5 accuracy had much more noticeable improvements than top-1 accuracy, indicating that the ensemble learned to rank the outputs from member models to more likely weight the correct class higher.

10.1 Future Work

This project was completed over the course of three months, on top of other coursework and the COVID-19 situation. In addition, our accessibility towards computing resources was limited to 1 GPU on a local machine and free GPU cloud computing. Given these constraints, we were not able to experiment and iterate as much as we had hoped.

10.1.1 Hyperparameter Tuning

Because our approach involved iterating over multiple architectures and data augmentation classes, we did not have the time to fine tune the hyperparameters of each model. If given more time, we would have experimented with using the Keras Tuner library, hypermodels, batch size tuning, and more data augmentations.

10.1.2 Candidate Architectures

We were only able to successfully train two candidate models: our custom AlphaNet and ResNet18-based BravoNet. Our unsuccessful architecture candidates include VGG [8], Xception [1], NASNet [14], InceptionNetV3 [11], and InceptionResNetV2 [9]. For the most part, these models had longer training times due to increased parameter count. Due to our time constraints, we did not fully train these architectures.

10.1.3 PCA

PCA showed that the Tiny ImageNet images belong in a vast dimensional space, so any potential dimensionality reduction would be valuable in future training. Due to the promise of our experiments and given the time, this would have been included as a preprocessing step for our training pipeline on our final ensemble. However, performing PCA across the entire data set proved too computationally expensive for the time allotted.

10.1.4 GAN-based Data Augmentation

A next step with general adversarial networks would be to use a class-conditioned GAN such as a BiGAN to augment the data with labeled, computer-generated training examples. However, there is a chance that this would not help the model accuracy on natural images, which is the main intent of this classifier. Therefore, this would be a computationally expensive step that may or may not improve the model.

11 Team Contributions

11.1 Kevin Liu - 38%

- Abstract
- Sections 1, 2, 3.1, 4-7, 9, 10.1.1, 10.1.2
- AlphaNet series
- BravoNet series
- AlphaStack ensemble
- Data augmentation
- Model tuning
- Training framework

11.2 Oscar Ortega - 31%

- Sections 8.2, 8.3, 10.1.3
- Experiments with t-SNE
- Experiments with PCA
- Research into GANs
- GAN implementation

11.3 Sofia Paganin - 31%

- Section 3.2, 8.1 and 10.1.4
- Saliency visualizations
- Research on different model architectures
- GCP for GAN training
- Research on data augmentation

References

[1] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

[2] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[3] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[5] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[6] Alec Radford & Luke Metz and Soumith Chintala. "Unsupervised Representation Learning Learning with Deep Convolution Generative Adversarial Network". In: (2016). arXiv: 1511.06434 [cs.CV].

[7] Seelwan Sathitratanacheewin and Krit Pongpirul. "Deep Learning for Automated Classification of Tuberculosis-Related Chest X-Ray: Dataset Specificity Limits Diagnostic Performance Generalizability". In: *CoRR* abs/1811.07985 (2018). arXiv: 1811.07985. URL: http://arxiv.org/abs/1811.07985.

[8] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[9] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261 (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261.

[10] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[11] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[12] Xiaoguang Tu et al. "Learning Generalizable and Identity-Discriminative Representations for Face Anti-Spoofing". In: *CoRR* abs/1901.05602 (2019). arXiv: 1901.05602. URL: http://arxiv.org/abs/1901.05602.

[13] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

[14] Barret Zoph et al. "Learning Transferable Architectures for Scalable Image Recognition". In: *CoRR* abs/1707.07012 (2017). arXiv: 1707.07012. URL: http://arxiv.org/abs/1707.07012.

Appendix

Figure 10: Data augmentations

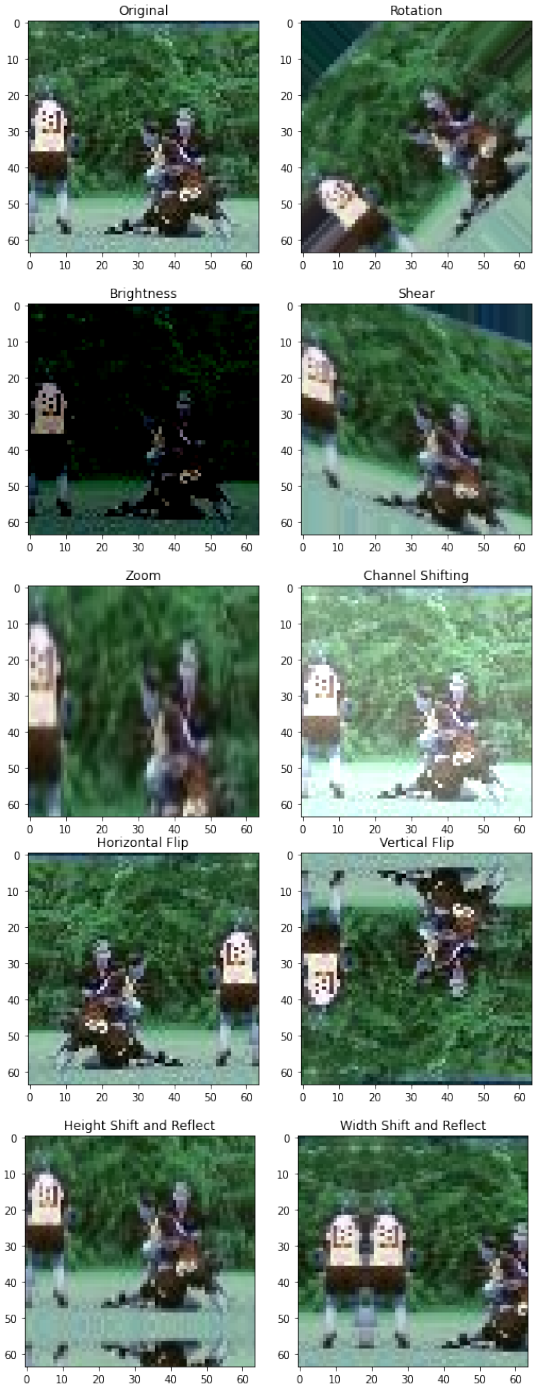


Figure 11: AlphaNet Results

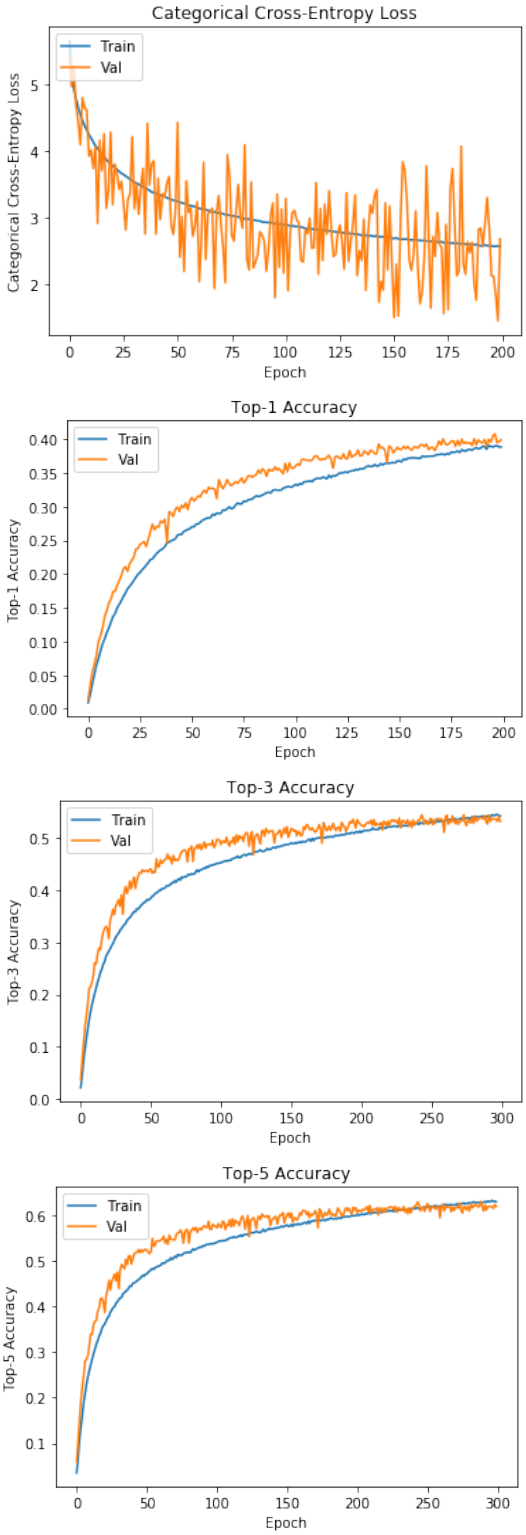


Figure 12: AlphaNet_v2 Results

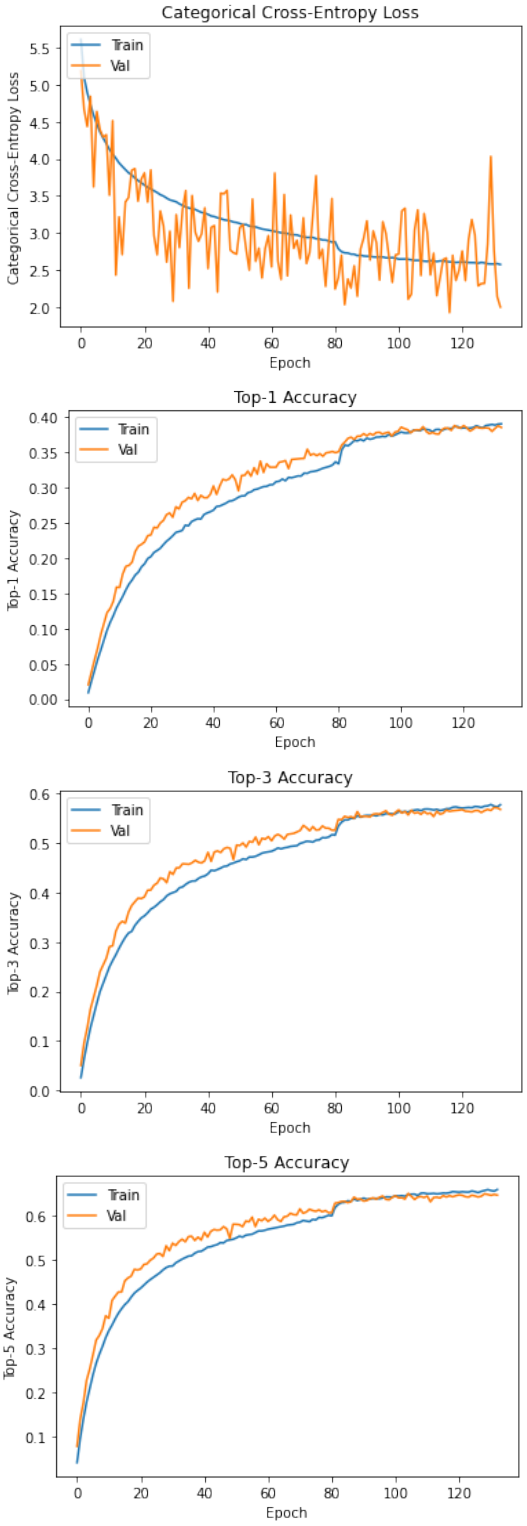


Figure 13: AlphaNet_v3 Results

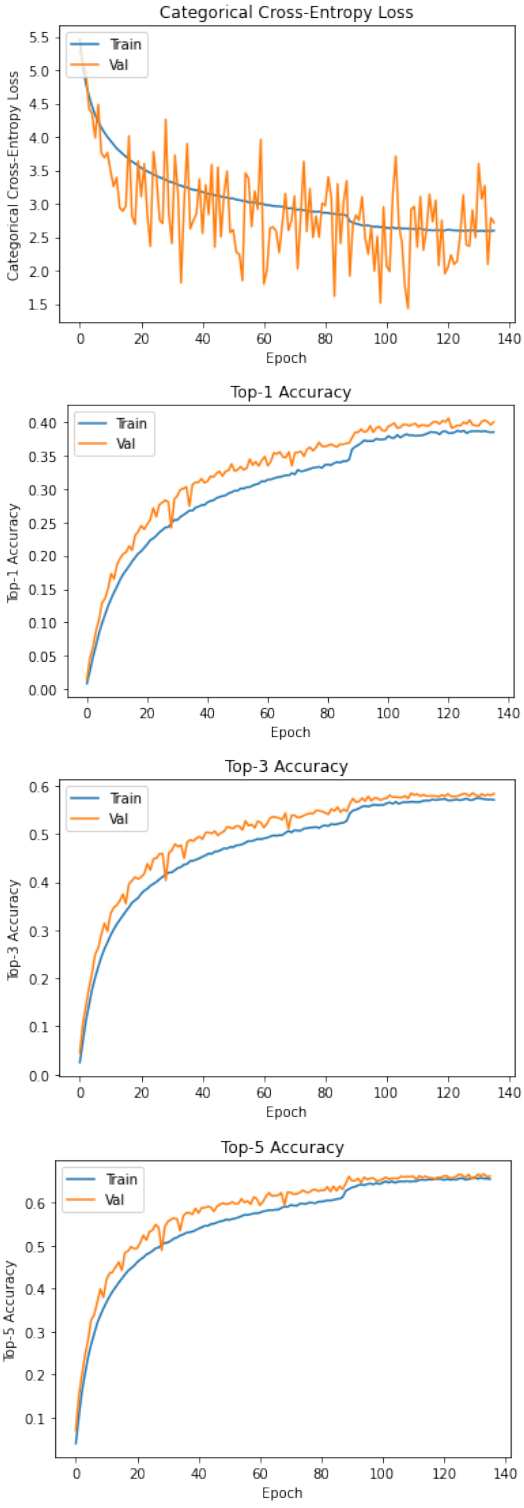


Figure 14: AlphaNet_v4 Results

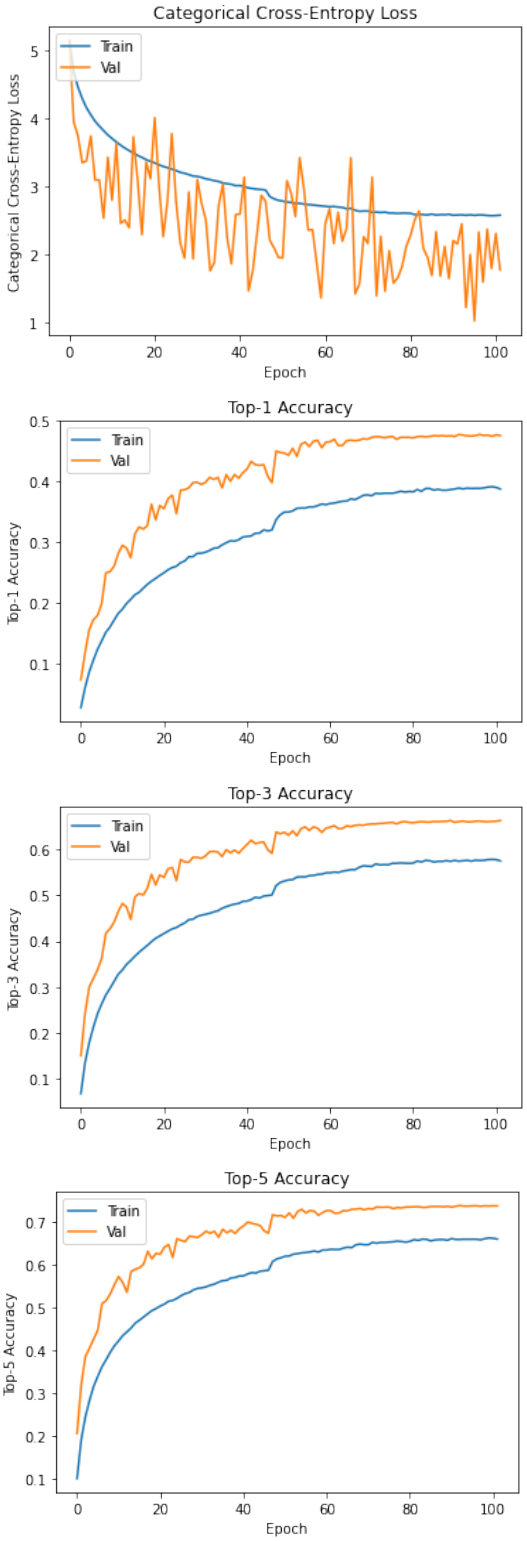


Figure 15: AlphaNet_v4b Results

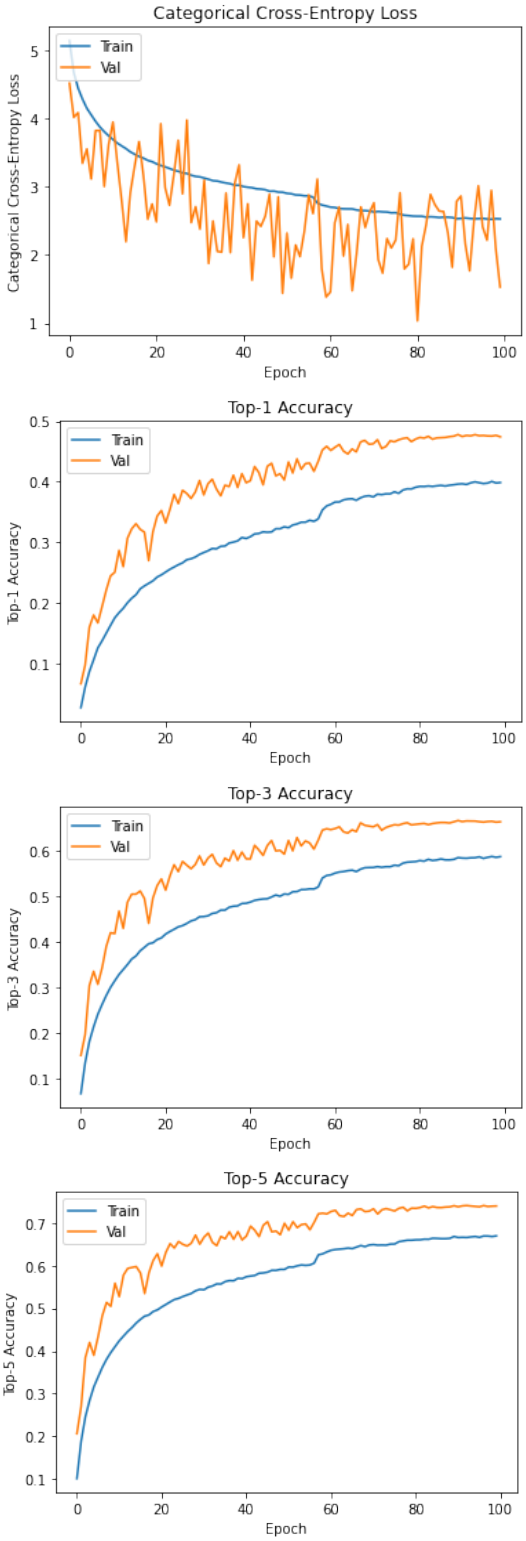


Figure 16: AlphaNet_v4c Results

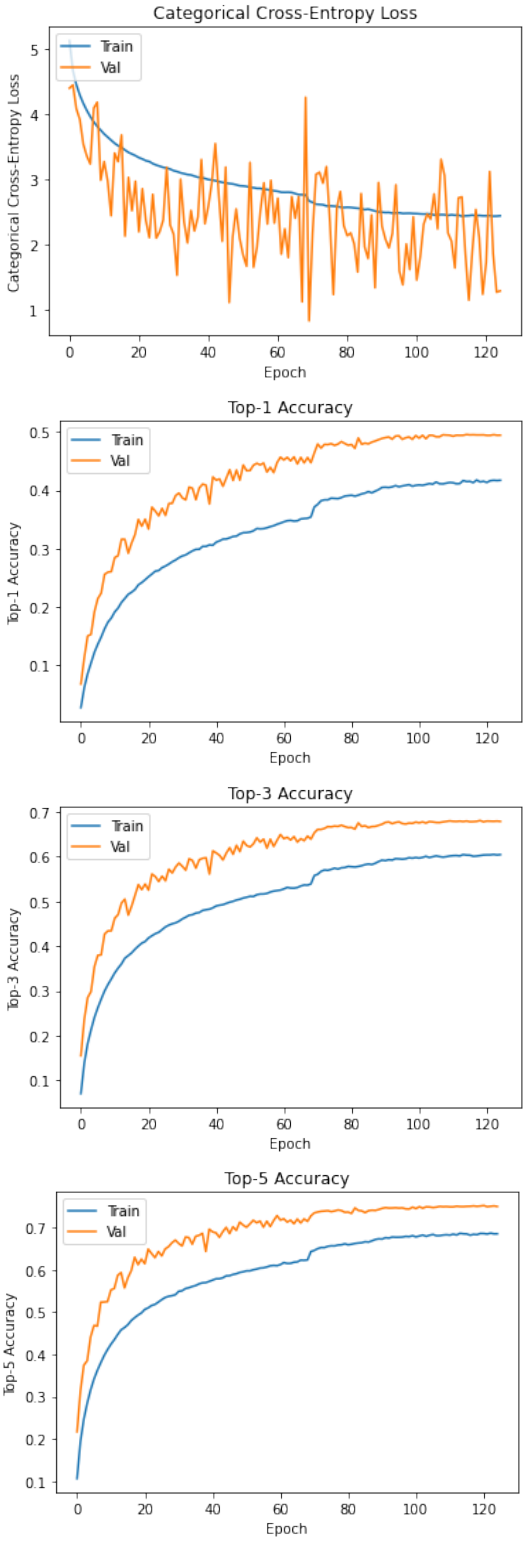


Figure 17: AlphaStack Results

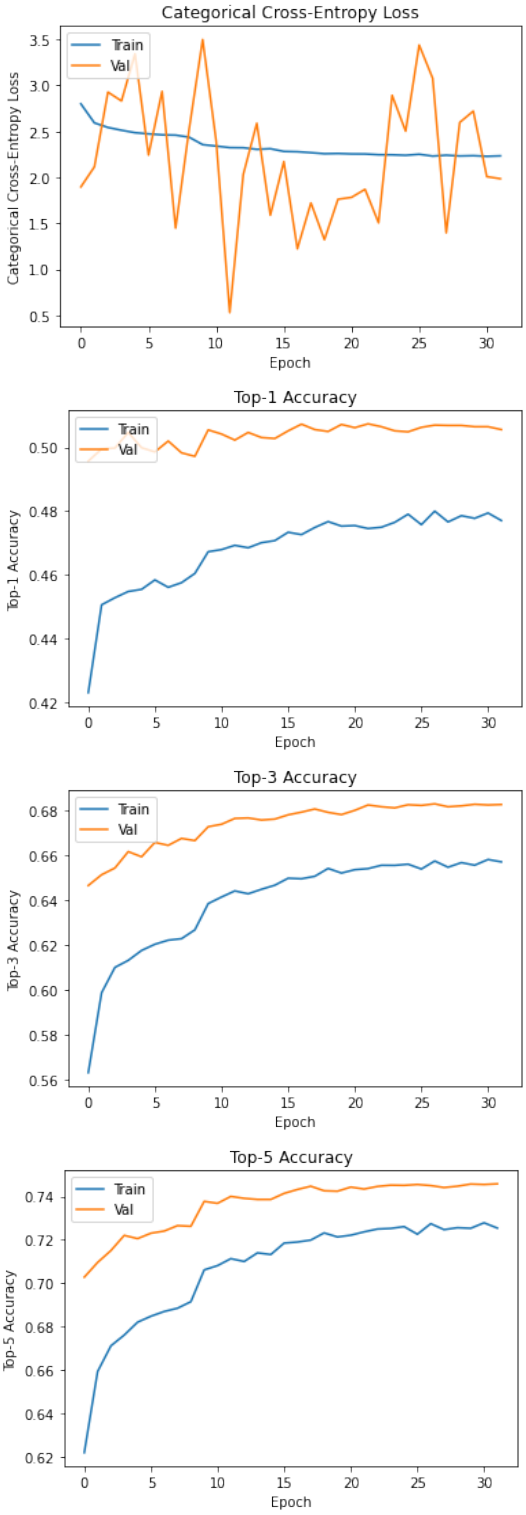


Figure 18: BravoNet Results

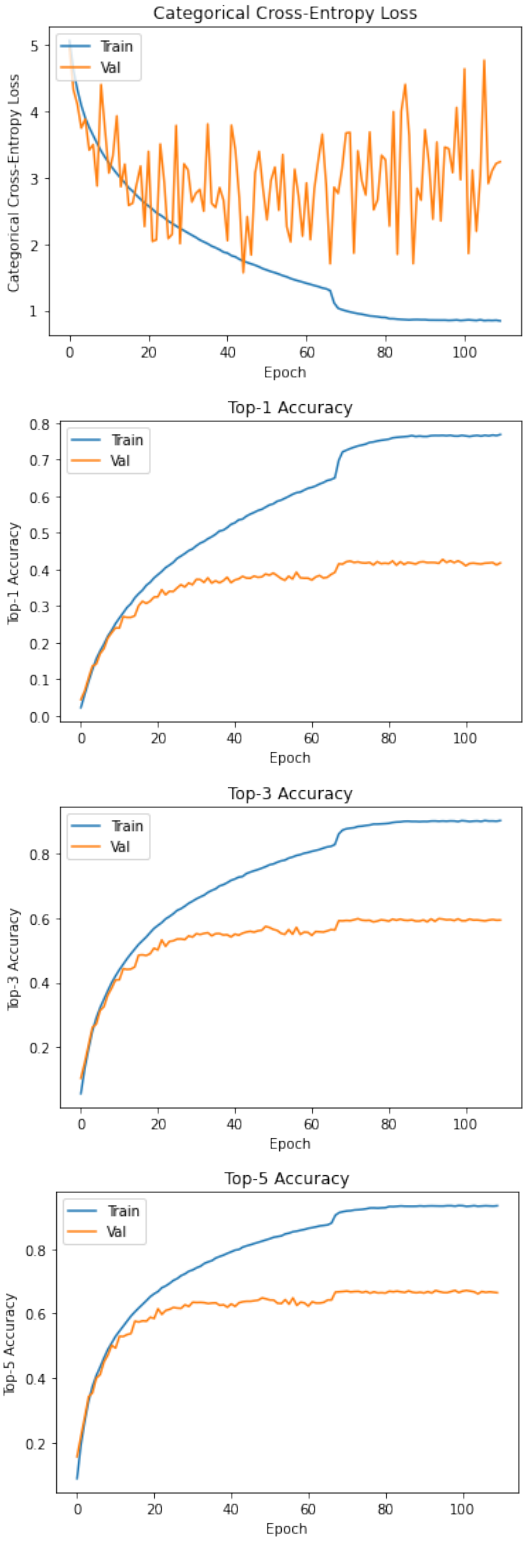


Figure 19: BravoNet_v2 Results

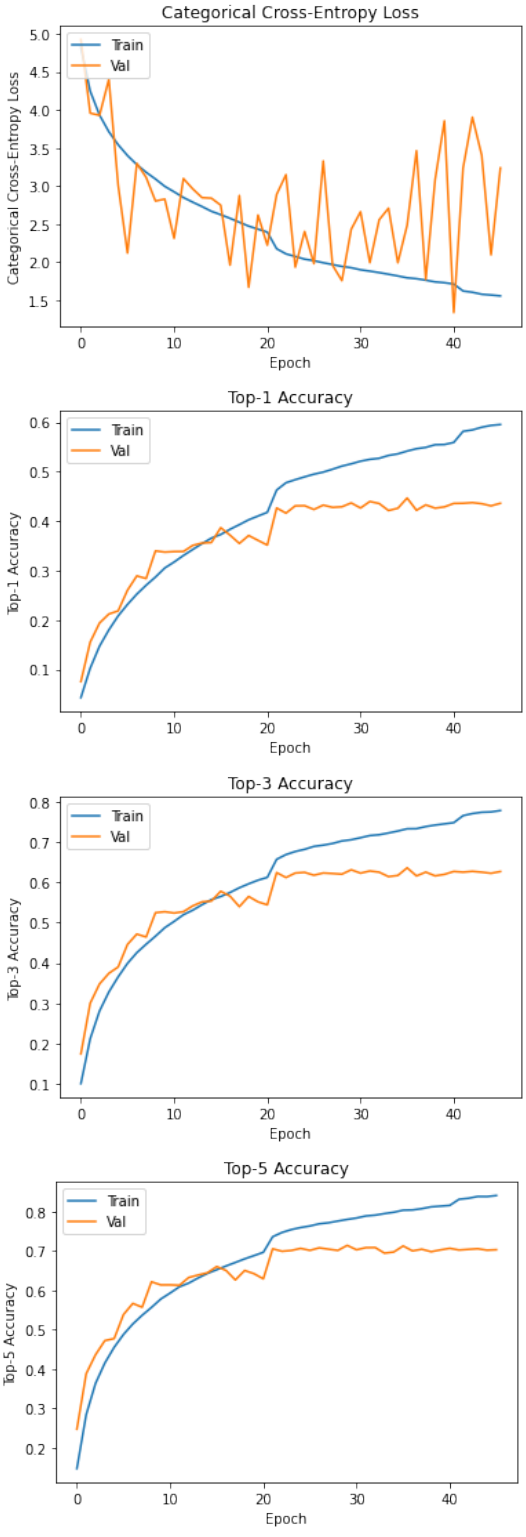


Figure 20: PCA Feature Reduction