Experimentation and Comparison Between Pre-trained and Custom Convolution Neural Networks

Lucas Nguyen, Paul Pan
Department of Computer Science and Engineering
University of California - San Diego
La Jolla, CA 92093
`lnguyen.professional@gmail.com, jpan@ucsd.edu`

Abstract

A common adage in training neural networks says "Don't be a hero", implying it is easier to build neural networks from previous work than to train a model on ones own. Here we experiment with both approaches to see which one achieves better top-1 accuracy on our 20-category classification task. First we train a baseline model and tune the model by modifying architecture and hyperparameters, starting with 25% accuracy and improving to 45%. Next we train two state-of-the-art models, VGG16 and Resnet18, freezing all but the last layer with accuracies of 74.06% and 77.89%.

1 Introduction

In this study we test whether a custom neural network can outperform pre-trained models. Specifically we test for top 1 image classification accuracy on a subset of the Caltech-UCSD Birds Dataset (CUB) which has 20 rather than 200 different classes, or birds. We test two categories of Convolutional Neural Networks; a custom model and two pre-trained models, VGG16 and Resnet18. Pre-trained models have been trained on other image classification tasks and have weights that extract image features. In order to adapt the model to our task we perform transfer-learning, where we use the same architecture and freeze most of the weights except for the end to train the low end features on our task. We experiment with both models by trying different combinations of hyperparameters and model architectures to see which one can improve and achieve highest accuracy on our test set.

2 Related Work

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

Batch Normalization plus Droput: https://arxiv.org/abs/1801.05134

Training tips: http://karpathy.github.io/2019/04/25/recipe/

3 Models

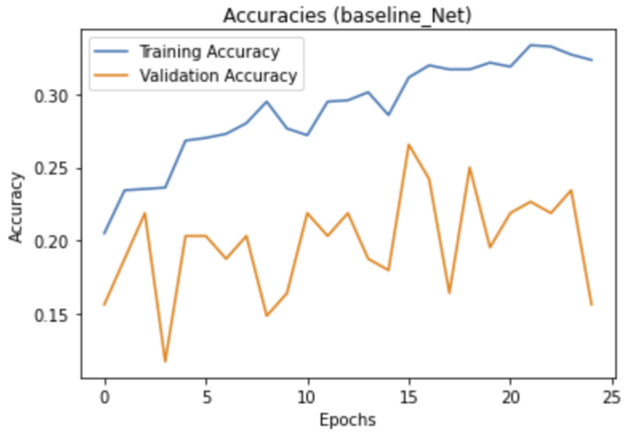
We start with a baseline model consisting of 4 layers of convolution, batch normalization, then ReLU activation. The convolutions have 3x3 filters where the number of filters increases from 64 > 128 > 256. The 4th layer has max pooling before the convolution, followed by average pooling. Finally we attach two dense layers at the end. The custom network architecture extends the baseline model by increasing the amount of convolution layers to 6. Also, three additional max-pooling layers with stride of 1 and dimension of 3x3 are added. This allowed greater spatial invariance which enhances the training. At the last layer of the network, the output dimension of the layer is 3x3, being reduced from 255 by 255. These enhancements allowed the network to more easily learn the non-linear relationships in those images.

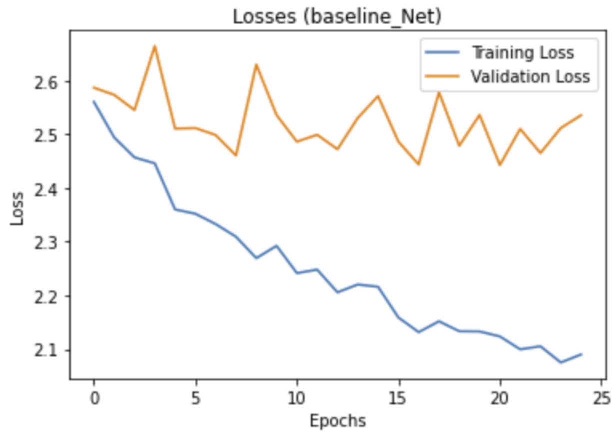
VGG and resnet model architectures were responsible for winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in certain categories for respective years. Both models accomplished state of the art results by using new methods to deal with training deep networks. VGG replaced large filters with 3x3 filters which reduced the total number of parameters and added more non-linear discriminators at each layer. Resnet used passthrough connections that allowed gradients to be propagated backwards through such a deep network by skipping layers.

4 Experimentation

4.1 Baseline Model

Our baseline model has test set accuracy close to that required by this programming assignment. Over the course of the 25 epochs, we have increased the test accuracy to 24.9%, very close to the 25% required. Also, the training accuracy went up by a lot while the validation accuracy tend to not go up as much, implying that the model is not generalizing well on new data. This is to be expected since the baseline model is not the best model.



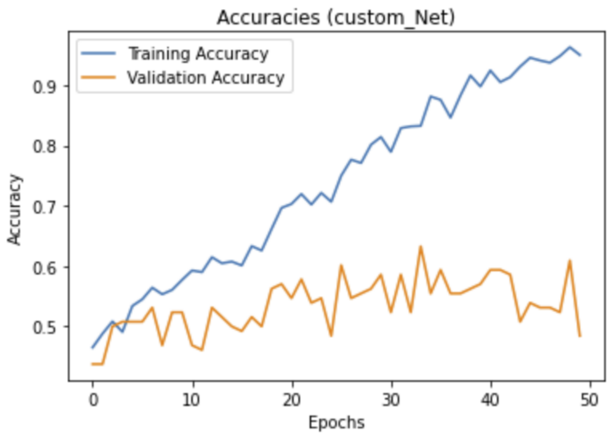


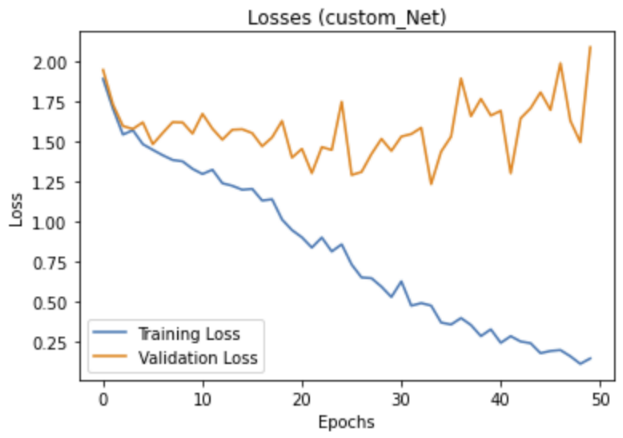
4.2 Custom Model

For the custom model, we added 2 extra convolutional layers to the last convolutional layer. The additional convolutional layers have made a great impact in the network's ability to generalize towards new data. We have also added more max-pooling layers to increase spatial invariance. By increasing the training to 100 epochs, we were able to get the test accuracy all the way up to 40%. However, we realized that the model is being overfitted when over 50 epochs based the validation loss on the graph, so we decided to reduced that training to only 50 epochs.

Tried Model Name	Train	Holdout	Test
No additional max-pooling, 2 additional conv layers after last one	90%	38%	34%
No additional max-pooling, 2 additional conv layers before last one	88%	37%	30%
3 additional max-pooling layers before last one, 2 additional conv layers before last one	95%	60%	40%

Table 1: Custom Model Cases



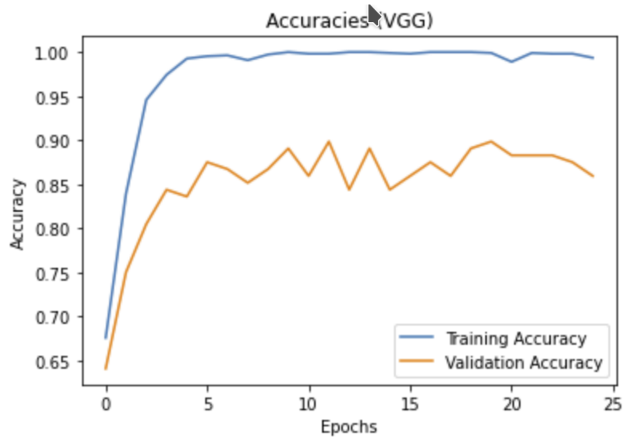


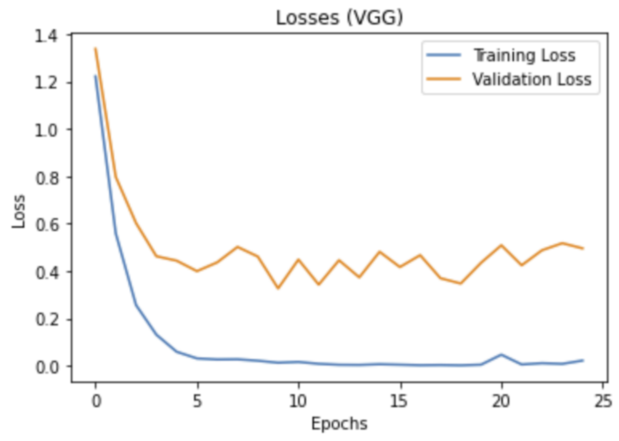
4.3 VGG16

For VGG16 we first tried freezing all but the last fully connected layer, and refitting the layer output to the number of classes in our classification task as a baseline, achieving 74% accuracy. Next we tried unfreezing all fully connected layers, which actually decreased accuracy. We think this is because the model had too many parameters to learn however the fully connected layers couldn't model the features that discriminated between bird classes and hence overfit the training set without generalizing to the test set. Next we tried unfreezing the last 2 convolutional layers along with the last fully connected layer, which improved our accuracy to 80%, suggesting the previous statement was true. For hyperparameter tuning we tried lowering the learning rate, since this is recommended when transfer learning to account for most of the weights being frozen/not learnable. Lowering the learning rate to 1e-4 achieved about the same results of 78%.

Tried Model Name	Train	Holdout	Test
Last Fully Connected layer unfrozen	98%	80%	74.06%
All Fully Connected layers unfrozen	97%	78%	56.67%
Last Fully Connected and 2 Convolution layers Unfrozen	98.5%	73.3%	79.67%

Table 2: VGG16 Batch Normalized Transfer Learning



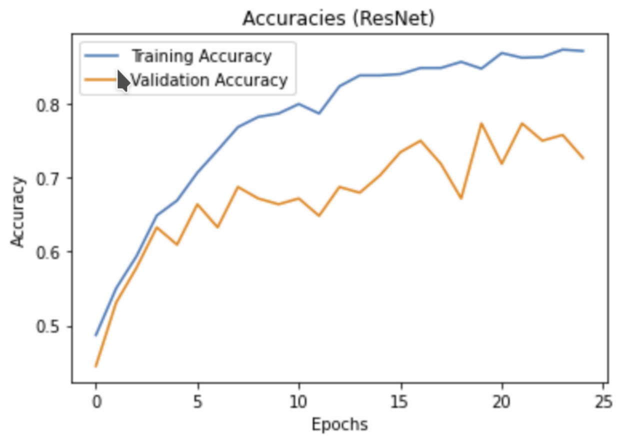


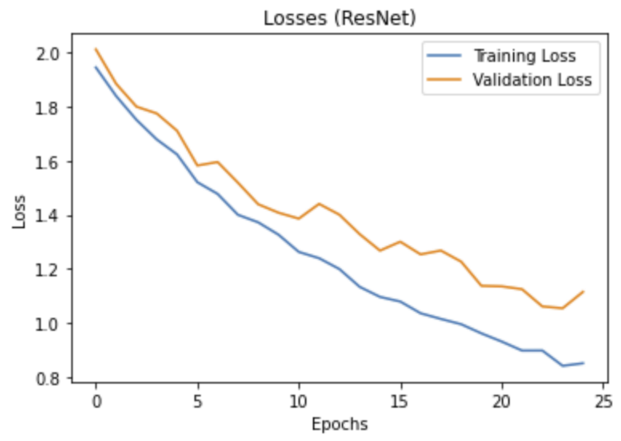
4.4 Resnet18

Like VGG16, we froze all but the last fully connected layer, and refit the layer output to the number of classes, achieving 78% accuracy. We also tried unfreezing the last two convolution layers however upon training results were about the same, 77% accuracy. For hyperparameter tuning we tried decreasing the learning rate for reasons mentioned above however the results were similar.

Tried Model Name	Train	Holdout	Test
Last Fully Connected layer unfrozen	100%	96.6%	77.89%
Last Fully Connected and 2 Convolution layers Unfrozen	100%	98.3%	77.72%

Table 3: Resnet18 Transfer Learning

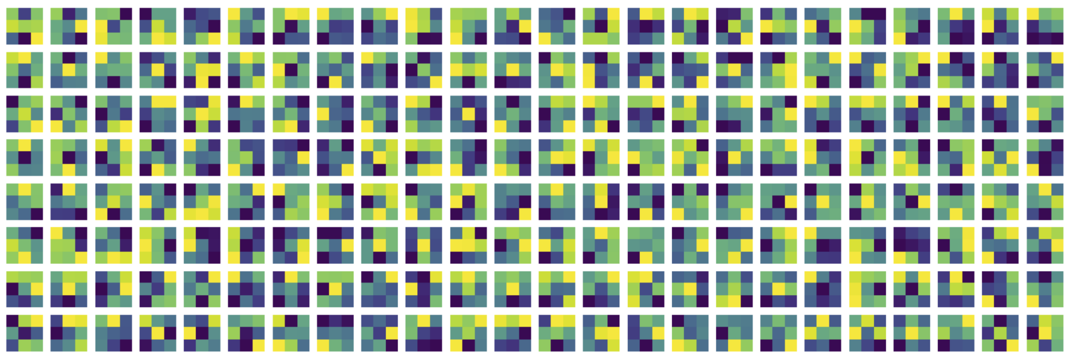




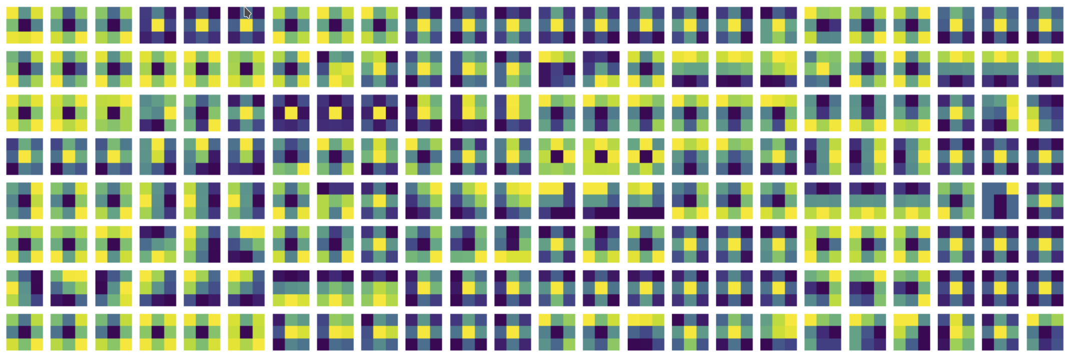
5 Model Feature Maps and Weights

5.1 Weight Visualization

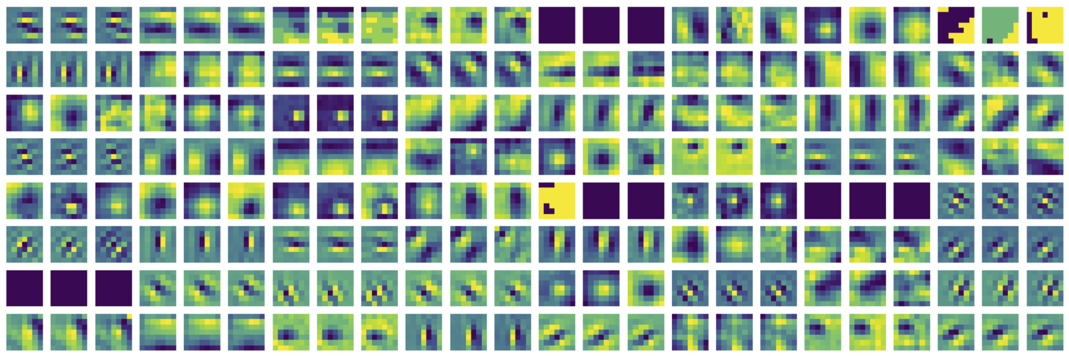
We visualized the weights for the first layer of the custom and pre-trained networks, where all three networks had 64 filters and 3 channels. Note that the custom model and VGG use 3x3 filter shape while resnet uses 7x7. Filters are aligned in column groups of 3 corresponding to RGB channels. Another thing to note is that VGG and Resnet layers dipslayed were trained on the ImageNet dataset and frozen/not trained with our dataset. However there are still similarities between VGG and the custom model, for example both have filters with a light pixel in the center surrounded by dark pixels. Overall VGGs filters seem to be more pattern organized and distinguishable than the custom model's weights, indicating the custom model hasn't been trained enough. In addition, Resnets filters have some familiar patterns such as gabor filters in various directions.



Custom Model Weights



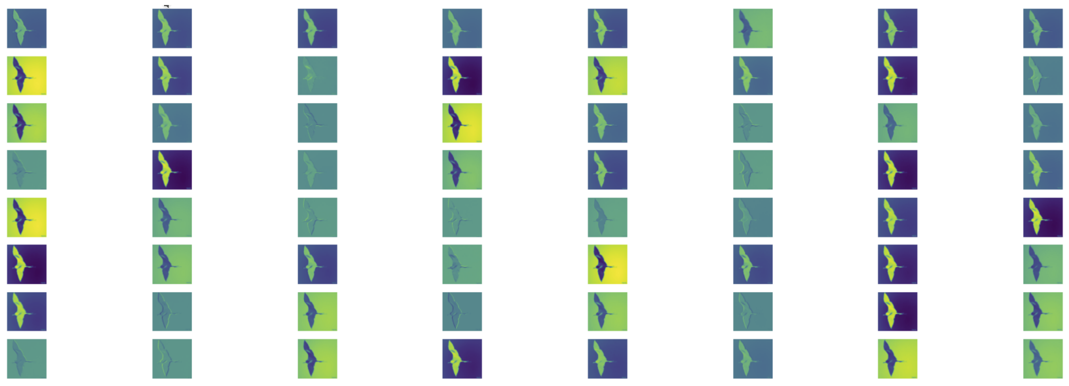
VGG16 Weights



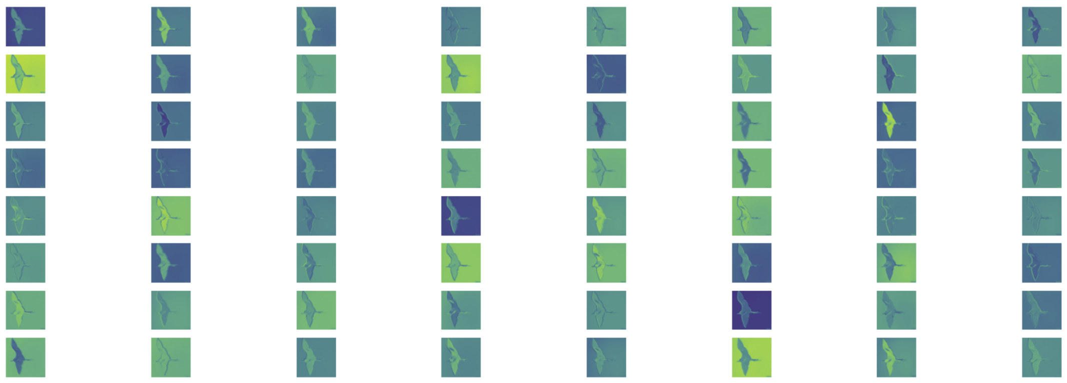
resnet18 Weignts

5.2 Feature Maps

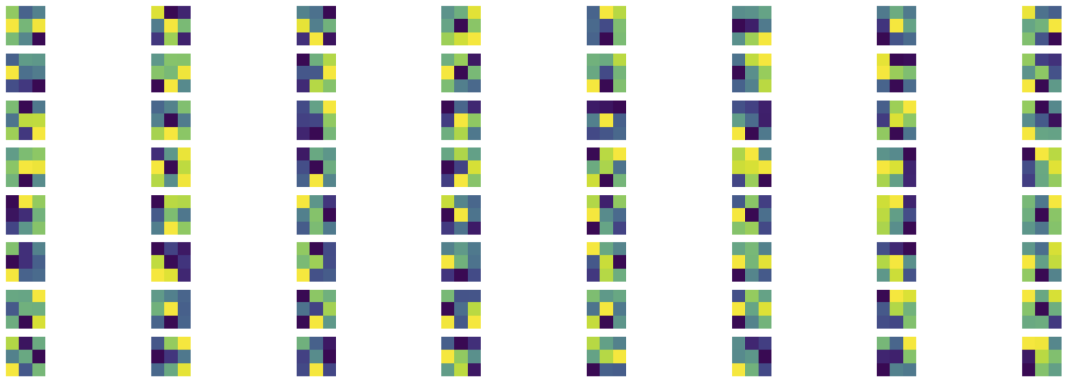
We visualize the activations for beginning, middle, and end convolutional layers for the custom and pre-trained models. Note that layers further into the network have more than 64 filters however for sake of visualization we only plot the first 64. For all models, the first layer activations resemble the image passed, however as we move deeper the resemblance is harder to discern. This follows how CNNs map more and more specific features as the network gets deeper.



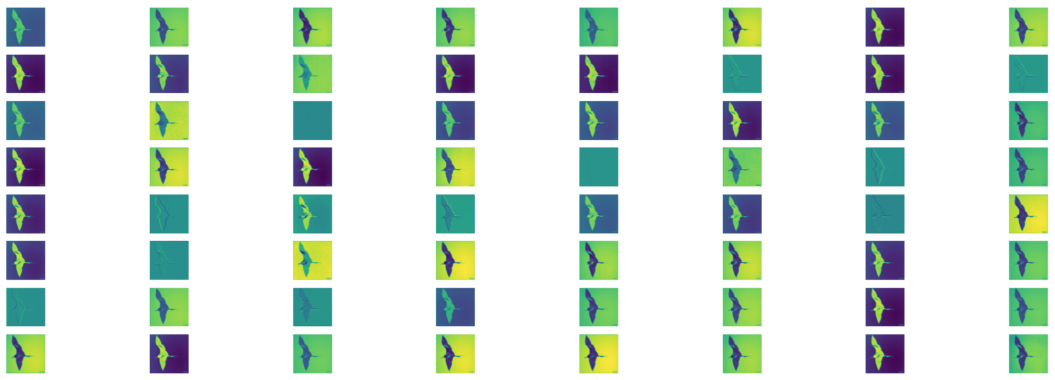
Custom First Layer Feature Map



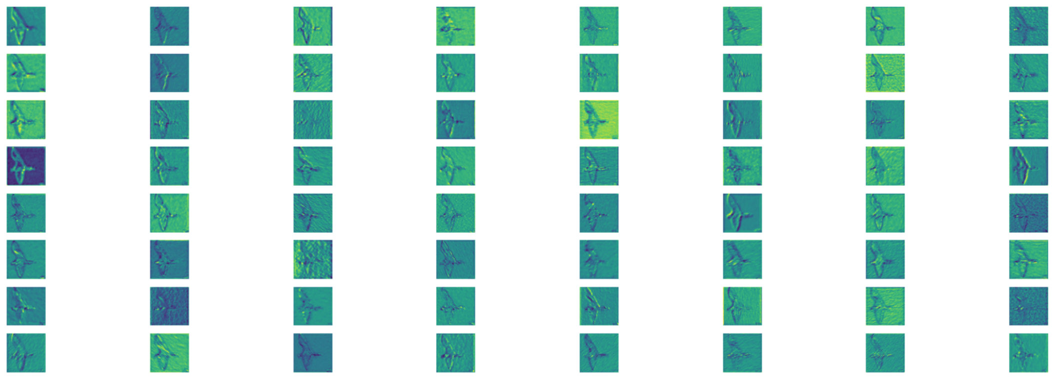
Custom Intermediate Layer Feature Map



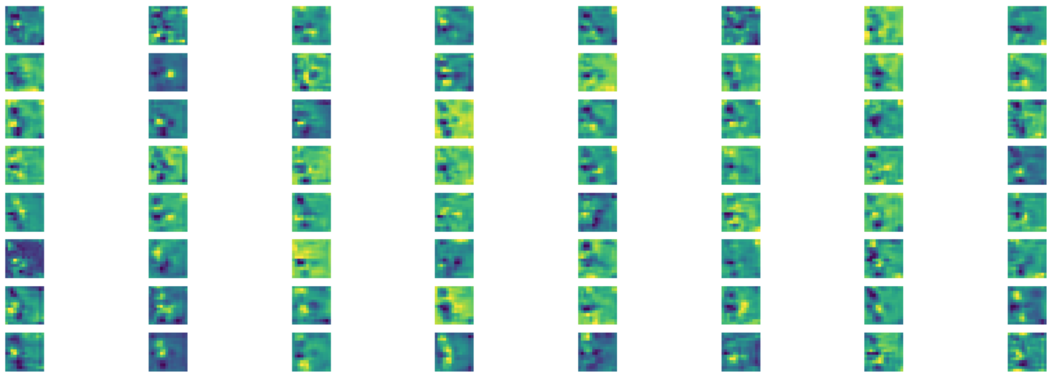
Custom Last Layer Feature Map



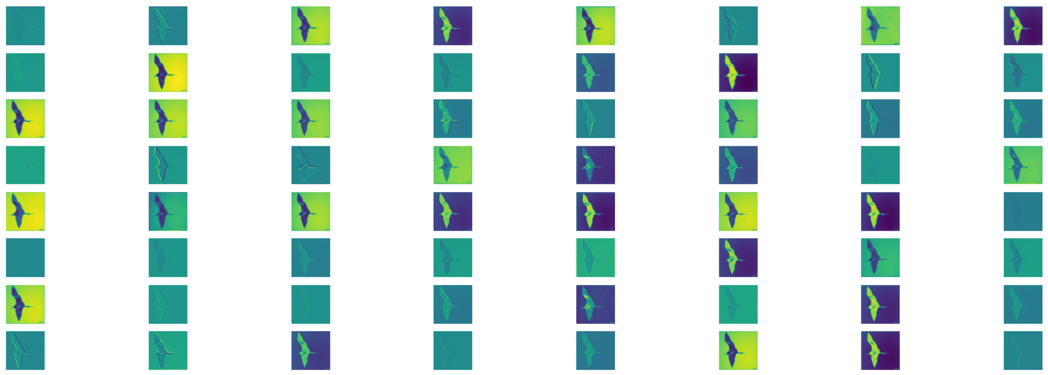
VGG16 First Layer Feature Map



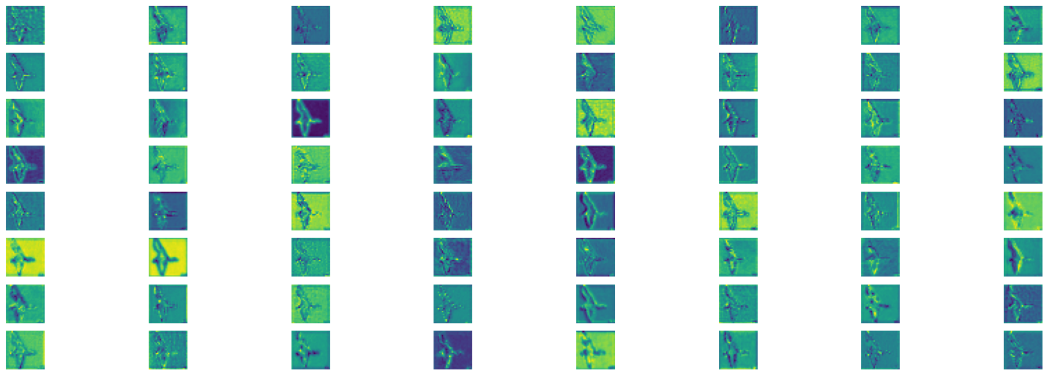
VGG16 Intermediate Layer Feature Map



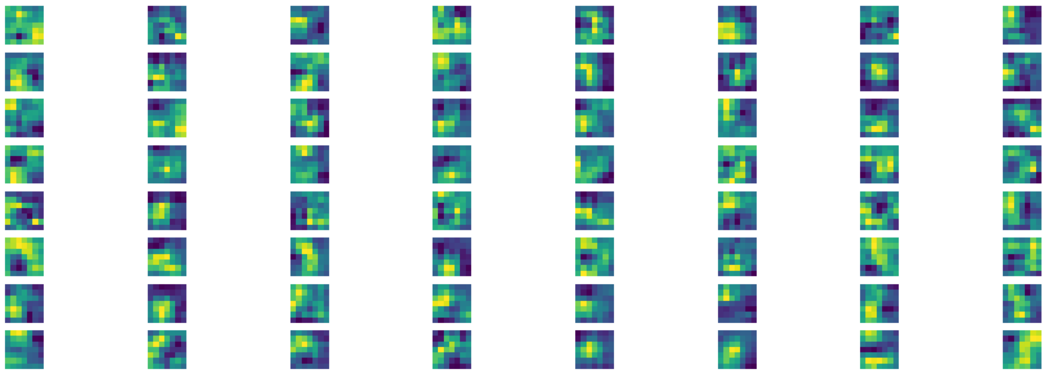
VGG16 Last Layer Feature Map



resnet18 First Layer Feature Map



resnet18 Intermediate Layer Feature Map



resnet18 Last Layer Feature Map

6 Discussion

Our baseline performed much worse than the baseline VGG16 and Resnet18 models, classifying the 20 categories better than chance but one third the accuracy of the pre-trained models. The custom model improvements didn't fare much better, with about half the test accuracy of the pre-trained models base. For the custom model, potential future steps would include more thorough hyperparameter tuning such as with a grid search which would consume a lot of time. We could also perform data augmentation on the dataset to improve any model we train. For the VGG16 and Resnet18 models, we can add more dropout layers to increase the generalizability of the model to unseen data, preventing overfitting of the training set.

In terms of things that worked, the custom model's accuracy on the prediction improved with the additional max pooling layers added in front of many of the convolution layers. Also, the learning rate is increased to increase the speed of learning via fewer epochs (learning rate of 1 x 10^3). However, that learning rate has proven to be slightly too high for VGG16 and Resnet18, so we decided to lower the learning rate back to the original in order to achieve a slight increase in test accuracy. One more thing that increased the accuracies in our models is to make the added convolution layers at the end have more output features.

7 Contributions

Lucas: I preprocessed and loaded the data and wrote the frame for cross-validation training with help from Paul, did transfer learning, weight visualization and feature maps for pre-trained models.

Paul: I implemented the custom model, as well as placing the weight and feature map visualizations correctly so they can be viewed in LaTeX. Also, I am responsible for fixing any dimensionality issues in the training.

We both helped each other debug.