Classification of Caltech-UCSD Birds 200 With Deep Convolutional Networks

Jacky Dam
University of California San Diego
San Diego, California
jdam@ucsd.edu

Dylan Loe
University of California San Diego
San Diego, California
djloe@ucsd.edu

Ali Zaidi
University of California San Diego
San Diego, California
mazaidi@ucsd.edu

Abstract

Image classification on a subset of Caltech-UCSD Birds 200 is conducted using a custom built convolutional neural network architecture with a final test accuracy of about 47%. Our model's final performance and architecture are compared against three other models: a simple baseline four-layer CNN, VGG16, and ResNet-18. Transfer learning, or utilizing pre-trained models to perform a different machine learning task on which the model was trained, is very popular in computer vision; slightly tweaking a model that has already been built and is known to classify images well to suit your machine learning task's needs is usually more efficient than developing one's own model from scratch. Adjusting the final weights in the VGG16 network achieved 76% accuracy on the test set. ResNet-18 performed similarly well after freezing and fine-tuning the final fully connected layer's weights, with a final test performance of 78%.

1 Introduction

The Caltech-UCSD Birds 200 dataset consists of 6,033 color images of birds from 200 different species. The authors of the original paper for this dataset explain that CUB-200 differs from more traditional image datasets like ImageNet by featuring subjects that are objectively difficult for humans to differentiate without prior knowledge, as opposed to a diverse array of different objects that humans can already classify near perfectly, and compensates for its rather small size [1]. Maji also states that this dataset is also notable due to its rigorous annotation of each part of each bird, allowing for the development of frameworks that support fine-grained discrimination between image attributes [2]. Convolutional neural networks have shown great success in computer vision tasks such as image classification due to their ability to dramatically reduce the number of parameters to train when compared to fully connected networks and their consistency in recognizing objects irregardless of size or local position in the image [3]. Due to resource constraints, classification is performed on a subsection of the original dataset with 600 images equally distributed among 20 classes. Model selection is facilitated by two-fold cross-validation with holdout sets of 60 images with special care taken that the class distribution remains completely uniform. As classes are completely balanced, accuracy is used to evaluate performance for our baseline and experimental models.

2 Related Work

As the models in this paper were all implemented in Pytorch, Pytorch documentation was heavily utilized during their implementations. In particular, the data loading process that included instantiating and implementing the Dataset class was aided by the Pytorch tutorial. The visualization of the weight maps and feature maps was helped with a blog post. Additionally, the tutorial on the various pre-trained neural network models available in Pytorch was valuable during the transfer learning on VGG16 and ResNet-18. Other sources served as inspiration into building our custom CNN model for this task.
Pytorch links:
https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
https://debuggercafe.com/visualizing-filters-and-feature-maps-in-convolutional-neural-networks-using-pytorch/
https://pytorch.org/docs/stable/torchvision/models.html
https://medium.com/bird-classification-using-cnn-in-pytorch/bird-classification-using-cnn-in-pytorch-7329c172a508
https://jovian.ai/mitra-krati/bird-classification-using-cnn

2.1 Models

Our baseline model is a simple two-layer convolutional network with a max pooling layer in between the convolutions and two fully connected layers at the end. The first layer consists of three convolutions with a common kernel size of 3 and 64, 128, and 128 output channels, respectively. The subsequent max pooling helps reduce the dimensionality of the input. The second convolutional layer uses a kernel size of 3, and stride of 3, and 256 output channels as well as an average pooling layer to further downsample the data before it is fed into the final two fully connected layers. Rectified linear units are used between layers to introduce non linearity into our network. Batch normalization in the layers helps make the network converge more quickly.

As our baseline model seems too simple to capture much of the underlying patterns in the images, our custom neural network changed the first convolutional layer to have 128 output channels instead of 64 to increase the complexity of our model as well as its number of trainable parameters, and increasing model complexity at this stage of the network is unlikely to introduce overfitting. We similarly gradually increase the number of output channels for each subsequent convolutional layer up to 256 and then 512 to attain further complexity, with max pooling and a ReLU activation function in between to achieve some location invariance and nonlinearity in the features. The last convolutional layer does the bulk of the dimensionality reduction by halving the output channels and using a 3x3 kernel with a stride of 2 before an average pooling layer flattens the data to be fed into the final two fully connected layers. The first fully connected layer has a 50% dropout to provide some regularization and prevent overfitting and a softmax activation function is used in the last layer of the network to create a probability distribution among the 20 bird species. Using another loss function other than cross-entropy in this setting was considered but deemed superfluous.

Recent major advancements in convolutional neural network architectures have drastically changed the way computer scientists design and implement deep networks. VGG16 is a well-known convolutional neural network 19 layers deep that followed AlexNet's success by replacing the 11x11 and 5x5 kernel sizes in the first two convolutional layers with a series of 3x3 convolutions [4]. VGG16 helped introduce the idea that multiple smaller convolutional layers in succession computed better nonlinear features than a single larger kernel and the extra computation it entailed was worthwhile. ResNet-18 was a revolutionary network by being substantially deeper with 152 layers and whose key insight was preserving the gradient with the identity function using skip connections to avoid having it vanish during backpropagation and failing to update initial layers' weights [3, 5].

2.2 Experiments

We trained our baseline model for 25 epochs using a learning rate of 10^{-4}, a batch size of 4, and Adam as the optimizer. The model weights were initialized using the Xavier initialization and

Network Layer	Layer Dimensions (input channels, output channels, kernel size)
Conv Layer 1	(3, 128, 3)
Max Pool 1	Kernel size 2, Stride 2
Conv Layer 2	(128, 128, 3)
Max Pool 2	Kernel size 2, stride 2
Conv Layer 3	(128, 256, 3)
Max Pool 3	Kernel size 2, stride 2
Conv Layer 4	(256, 256, 3)
Max Pool 4	Kernel size 2, stride 2
Conv Layer 5	(256, 512, 3)
Max Pool 5	Kernel size 2, stride 2
Conv Layer 6	(512, 256, 3, stride 2)
Avg Pool 1	(Kernel size 1, stride 1)
Fully Connected 1	(256, 1024) with Dropout
Fully Connected 2	(1024, 20)
Softmax	20

Table 1: Custom CNN Architecture. Layers have a stride of 1 and padding of 0 unless otherwise specified. ReLU activation function is used between each layer as well as batch normalization.

networks were evaluated using cross-entropy loss. Our baseline model accuracy and error on the training and holdout datasets are plotted below in Figure 1. Training and validation accuracies appear to stably increase and top out at 42% and 33%, respectively. The validation loss does not seem to fall as greatly as the training dataset's and rises before the final epoch; however, this is most likely due to the fact that we only averaged over two very small validation sets. The final accuracy obtained on the test dataset was 28%; while still a better performance than a random classifier given 20 classes, a higher accuracy can definitely be achieved using a deeper, more intelligently designed neural network architecture.

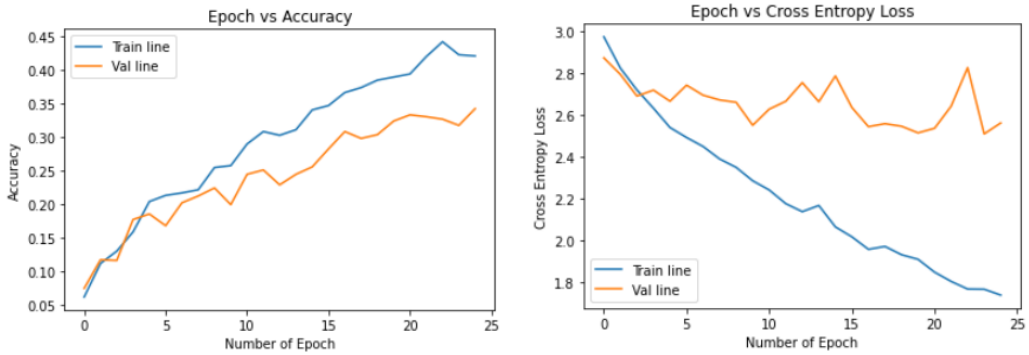

Figure 1: Training and Validation Accuracy and Loss for Baseline Model. Final test accuracy was 28%.

Data Set	Accuracy
Training Set	44.91%
Holdout Set 1	23.00%
Holdout Set 2	23.33%
Test Set	28%

Table 2: Baseline Model (default parameters, 25 epochs)

For the custom model that we created by adding two additional convolutional layers and multiple 2x2 max-pooling layers with strides of 2, we managed to improve the accuracy of the test dataset to about 47% with ± 2 marginal error with 70 epochs, a learning rate of 10^{-4}, and a batch size of 4. Comparing the custom model to the baseline model, we learned that every hyperparameter

that we experimented with had some kind of positive or negative affect on the model performance. With this concept in mind, we selectively made decisions based on if it had a positive change in our model, and acquired inspiration from the ResNet-18 and VGG16 models. Among all the different changes we made for our custom model, we found that increasing the number of output channels for each convolution layer, max-pooling, batch normalizing, and the addition of two convolution layers gave us an acceptable range of accuracy. Model 1 represents no changes in any parameters which is essentially the performance of the baseline model. Model 2 represents the addition of two convolution layers, and increasing output channels. Model 3 represents all the changes in parameters which improved our overall performance, which including, increasing output channels, utilizing max-pooling in more layers, and adding two additional convolution layers.

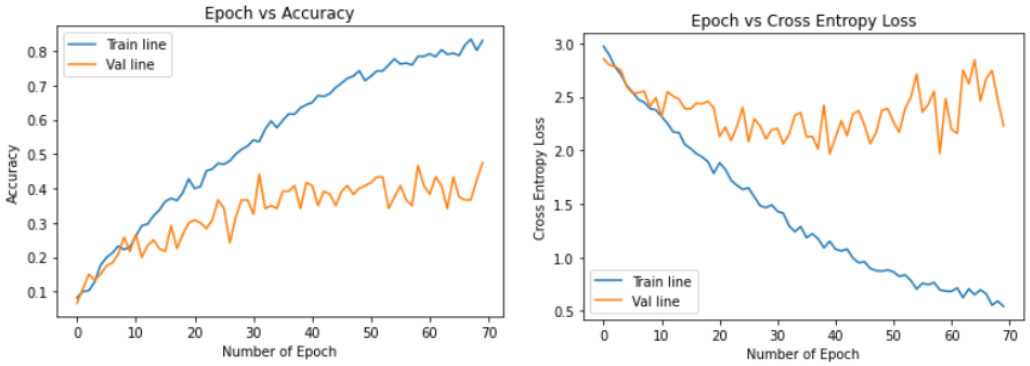

Figure 2: Training and Validation Accuracy and Loss for Custom model. Final test accuracy was 47%.

Data Set	Accuracy
Training Set	75.20%
Holdout Set 1	46.67%
Holdout Set 2	51.67%
Test Set	47%

Table 3: Custom CNN Model With Optimal Parameters

Model Name	Holdout Acc.
Model 1	34.27%
Model 2	42.19%
Model 3	51.67%

Table 4: Process of Building Custom CNN Architecture

For the Vgg16 model which was developed and optimized by some other researchers, we managed to achieve an accuracy of about 76% with $\pm 1\%$ margin of error. This specific model is more complex and sophisticated than our baseline and custom model and is able to converge to optimal performance with less epochs. From the graphs below, we decided to run the algorithm for the same number of epochs as the other models to more clearly understand the difference between the different models. Looking at the graphs, it's evident that as we surpass about 30 epochs, we start overfitting which in the end would degrade the overall performance of the model.

Taking a look at table 8, we we different changes for our Vgg16 model which drastically improved our accuracy. After making minor changes which progressively improved our performance, we were able to achieve an accuracy of about 75% for the test data set.

For the Resnet model developed and optimized by other researchers and scholars, we managed to achieve an accuracy of about 77% with a margin of error of ± 1. Similar to the VGG16 model, this

Data Set	Accuracy
Training Set	88.70%
Holdout Set 1	66.67%
Holdout Set 2	70.67%
Test Set	76%

Table 5: Vgg16 Model With Optimal Parameters

Model Name	Holdout Acc.
Adam + Smaller Learning Rate + More Epochs	67%
Vanilla SGD with Nesterov	69%
Adam + Better Learning Rate	75%

Table 6: Process of Changing VGG16

model is able to converge to an optimal performance with less epochs. From the graphs below, we ran the algorithm for the same number of epochs as the other models to more clearly understand the differences between the models. From the graphs, we can see that we start overfitting after about 30 epochs. Although the ResNet model ran for more epochs than necessary, it's evident from the graphs that it's far more optimal than our custom and baseline model which achieved about half the performance of the ResNet model. In the end, we found that the ResNet model performed the best out of all the models that we tested, but only has a difference of about 2% on the test set when compared with the VGG16 model.

Taking a look at table 8, we we different changes for our Resnet model which drastically improved our accuracy. After making minor changes which helped improve our performance, we were able to achieve an accuracy of about 77% for the test data set.

Data Set	Accuracy
Training Set	87.29%
Holdout Set 1	72.33%
Holdout Set 2	74.33%
Test Set	78%

Table 7: Resnet Model With Optimal Parameters

3 Feature map and weights analysis

The weight maps for our custom model and the pre-trained VGG16 network look much more similar than the weights from ResNet18. VGG16 and the custom model weight maps look to compute many different types of edges and intensities of light in many different orientations; however, these features seem very basic in comparison to ResNet-18's weights. The complexity of all of the different features that ResNet-18 is trying to detect in just the *first* convolutional layer is a testament to the efficacy of deep convolutional neural networks. The deeper the network, the more discriminatory and diverse the features they produce become, even in the preliminary layers.

When we take a look at the feature maps given a specific image from the test data set below in Figures 6, 7, 8, it is evident that the different CNN models detects different features across different convolution layers. Our figures are structured in a way that we show the features map of the specific images across the first convolution layer, middle convolution layer, and last convolution layer. Looking at all three of the models which we analyzed, the first convolution layers all look somewhat similar in which we see the shape of the object with varying amounts of shading. This concept makes sense across the different models because the first convolution layer hasn't gotten a chance to learn and train their weights accordingly. Looking at the middle convolution layers across all three models, we can see that different features of the object becomes more apparent such as the shading of different parts of the bird, and other more apparent shading of different parts of the object. This makes sense because as we progress through the convolutional layers, the algorithm is able to learn

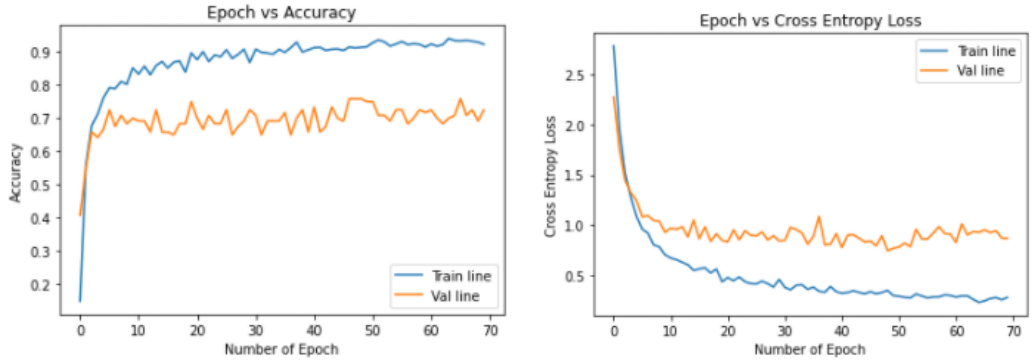

Figure 3: Training and Validation Accuracy and Loss for the Vgg16 model. Final test accuracy was 76%.

Model Name	Holdout Acc.
Adam + Smaller Learning Rate + More Epochs	71%
Vanilla SGD with Nesterov	74%
Adam + Better/Larger Learning Rate	77%

Table 8: Process of Changing ResNet18

and train the weights to better visualize the object. Across the last convolutional layers, we can see the feature map of the final convolutional layer which represents all the different calculations made across the number of convolutional layers. These represent that features which allows of algorithm to properly assign a classification to send as an output. We used hooks to pull out specific convolutional layers from the different models, and this allowed us to display the feature maps after running the model.

3.1 Discussion

Among all four CNN models we experimented with: baseline, custom, vgg16, and resnet18, we found that vgg16 and resnet18 performed significantly better than the baseline and custom model. We found that resnet18 had the best performance overall with only a marginal advantage of a couple percents over vgg16. Although our custom model was able to perform at an accuracy of about 47% with the test data set, we believe that there are more changes such as increasing number of convolution layers, playing with number of output channels, implementing learning rate decay, etc. which could improve our model.

For our custom model, we chose to add two convolution layers, add more max-pooling functions, and increase number of output channels across different convolution layers with batch-normalization. In the beginning, our custom model was based on the baseline model given to us, and we were able to slowly improve the performance as we selectively chose changes which helped our model improve. The reason why these aspects improved our performance is because these characteristics allow our model to learn more complex features after passing images one by one through the algorithm. In the end, the custom model was able to learn more complex features than the baseline model which can be seen when comparing the feature maps.

In addition to the changes that we made for the custom model, we also determined that increasing the number of epochs helped improved our performance. When we looked the loss and accuracy graphs after implementing the change in the custom model, we noticed that the model didn't exactly start overfitting, which allowed us to determine that we could run the algorithm for more epochs and improve the accuracy. Since we managed to meet the acceptable range of accuracy for this custom model, we decided to stop after changing the number of epochs.

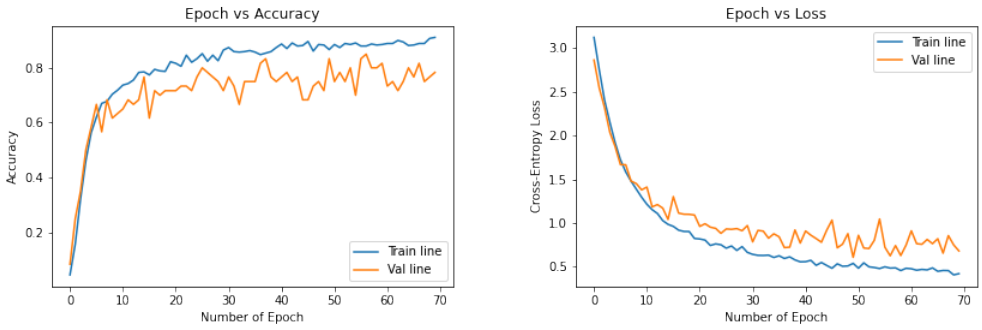

Figure 4: Training and Validation Accuracy and Loss for ResNet-18. Final test accuracy was 78%.

4 Team contributions

4.1 Jacky Dam

In this project, I worked on cleaning and processing all the different data sets to pass through the models. I also worked on figuring out the different parameters that we needed to change in order to optimize the custom model and gave suggestions to the team on the changes necessary to achieve high performance.

4.2 Dylan Loe

In this project, I helped tune the ResNet18 Model and debug the running of the models and generating the accuracy and loss plots. I also wrote the Abstract, Intro, and Models Sections of the report.

4.3 Ali Zaidi

In this project, I helped write and debug code in the training.py (cross_validation, weight maps, and feature maps) and helped create the custom model. I also worked on datasets.py, in getting the data to load and transform.

5 References

[1] Welinder P., Branson S., Mita T., Wah C., Schroff F., Belongie S., Perona, P. "Caltech-UCSD Birds 200". California Institute of Technology. CNS-TR-2010-001. 2010.

[2] Maji, S. Discovering a Lexicon of Parts and Attributes. ECCV Workshops. 2012.

[3] Cottrell, Gary, 2020. Lecture 5, lecture slides, *ConvNets Part 1*, UCSD.

[4] VGG16 - Convolutional Network for Classification and Detection. (2018, November 21). Retrieved from https://neurohive.io/en/popular-networks/vgg16/

[5] Adaloglou, N. (2020, March 22). Intuitive Explanation of Skip Connections in Deep Learning. Retrieved from https://theaisummer.com/skip-connections/

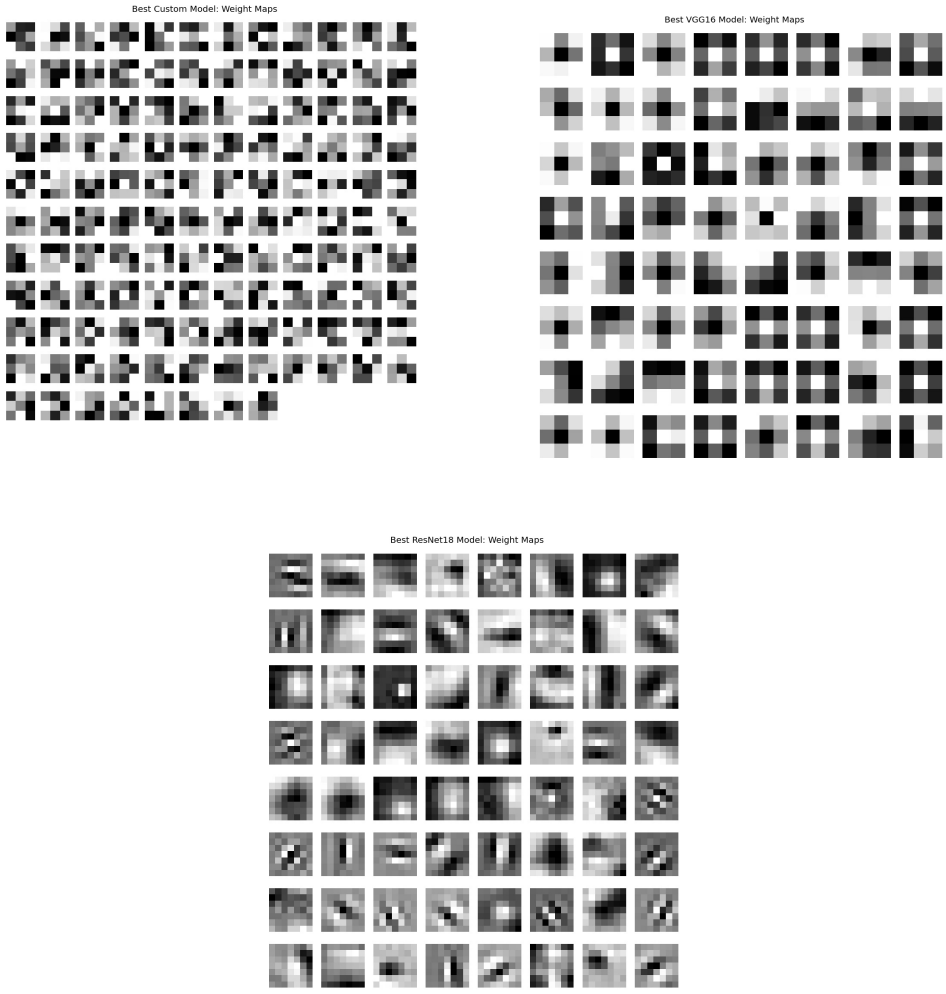

Figure 5: Weight maps from the first convolutional layers going clockwise from our custom model to VGG16 to ResNet18

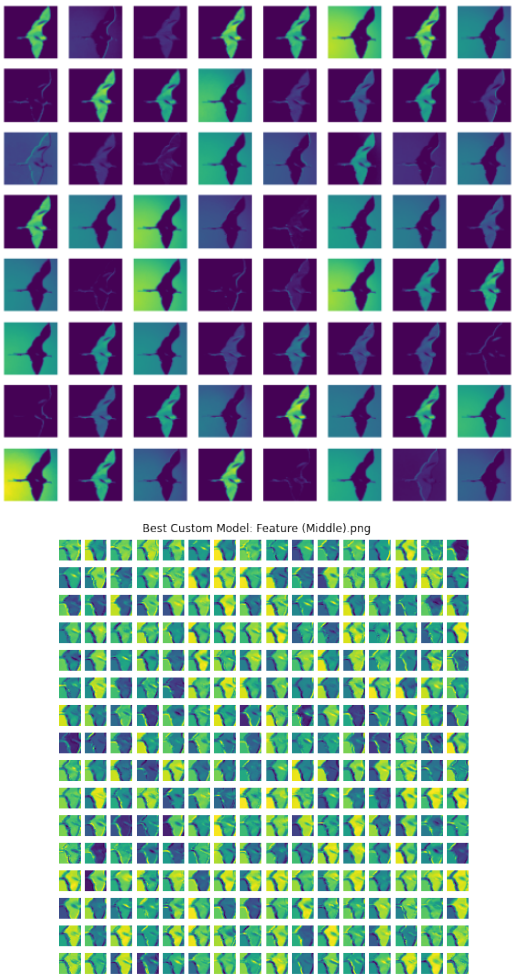

Figure 6: Feature maps from Custom Model going clockwise from upper left: Input to Middle to Output

Figure 7: Feature maps from VGG16 going clockwise from upper left: Input to Middle to Output

Figure 8: Feature maps from ResNet-18 going clockwise from upper left: Input to Middle to Output