Flower Recognition Using Deep Learning

Yihao Liu
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
yliu333@jhu.edu

Ruijie Zhu
Johns Hopkins University
500 W University Pkwy, Baltimore
rzhu21@jh.edu

chenxuan xiang
Johns Hopkins University
500 W University Pkwy, Baltimore
cxiang4@jh.edu

Hongchao Shu
Johns Hopkins University
108w 39th, Baltimore
hshu4@jh.edu

Abstract

Image classification is a classic problem in deep learning area. In this paper, we explore different ways to enhance the performance of image classification for five classes of flowers. In general we tried AlexNet, ResNet and VGG. We implemented those structures according to model performance especially in accuracy and generalization.

1 Introduction

1.1 Problem statement

After training the deep learning neural network model by using the standard data set "CIFAR-10". The team decided to move further on building another the picture recognition model with more advanced machine learning framework and more realistic and unprocessed raw data set. The team thought the "flower recognition with five classes" data set from kaggle might be a good data set the final project. Team decided to implement different neural network structure frameworks: AlexNet, ResNet, and VGG11. After that the team will compare all these different frameworks to see which one has the highest accuracy and the team will decide which one to select.

1.2 Data description

This data set contains 4242 images of flowers which divided into five class: chamomile, tulip, rose, sunflower, dandelion. All classes are evenly divided and have roughly 800 pictures for each class. All pictures are media resolution about 320x240 pixels pictures with various dimensions on their sides. Therefore, pictures are required to process and to normalize to single size before putting into the deep learning model.
The data set is from: https://www.kaggle.com/alxmamaev/flowers-recognition

2 Networks

2.1 AlexNet

AlexNet network structure can be seen in Figure 1 [1]. AlexNet has 5 convolution stages and 2 full-connected layers. In conv1, the dimension of the input is 227 by 227 by 3, 227 being the height

Overleaf Template by 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

and length of the image and the 3 being the 3 RGB channels. The kernel size in the stage is 11 by 11 by 3, using a stride of 4 and channel size of 96. Conv1 uses relu activation, followed by a max pooling layer, with a size of 3 by 3 and stride of 2. The output of conv1 is 55 by 55 by 96, 96 being the number of channels. There is also a local response normalization layer size 5 after pooling. It uses the same padding to remain the convoluted image size.

The following blocks, conv2-5 are of the similar format. Parameters are: kernel size 5 by 5, channel 256, stride 1; kernel size 3 by 3, channel 384, stride 1; kernel size 3 by 3, channel 384, stride 1; kernel size 3 by 3, channel 256, stride 1, for conv2, conv3, conv4, conv5, respectively. Conv3 and conv4 do not have maxpooling or batch normalization, and conv5 does not have batch normalization. Followed by the convolution blocks are 2 fully-connected layers, both having drop outs and uses relu activation.

Our implementation had main obstacles with the input and output dimensions, since they are usually not explicitly specified by papers (apparently it is too trivial for the deep learning community). There are open-sourced tools available that can be used to easily derive the sizes. AlexNet is a relatively simplicit network. It is lightweight and easy to train. However, it may not be able to fully learn the information in the provided dataset, compared to the more recent complex networks. The loss came to a very low level after 1000 epochs of training. Some results are shown in the Figure 2 and 3.

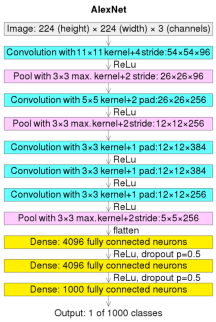


Figure 1: AlexNet structure

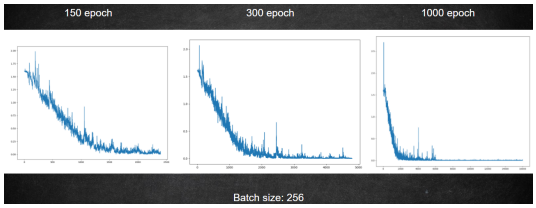


Figure 2: Loss plots for the training of AlexNet

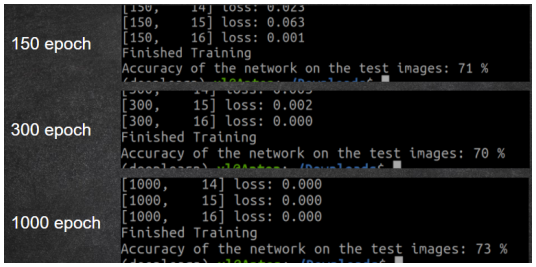


Figure 3: Prediction accuracy of AlexNet on the flower dataset

2.2 ResNet

Resnet is one of the most prominent architecture for complex computer vision problems because of its compelling outcomes. The main idea of ResNet is to utilize skip connections, or shortcuts to jump

over some layers. Skip connection alleviates the problem of disappearing gradients. The disappearing gradients occur over time when researchers tended to adding more layers in order to build deeper neural networks. The reason for stacking layers is that these layers are able to learn increasingly complicated features as time goes on. However, when we add more layers to the neural network, it gets more difficult to train them, and their accuracy begins to saturate and degrade dramatically. Numerous solutions have been presented, but the problem was not completely solved until ResNet comes to the rescue by allowing the gradient to flow through an additional shortcut channel. These connections also aid the model by allowing it to learn the identity functions, ensuring that the higher layer performs at least as well as the lower layer, if not better.

ResNet contains two basic modules: BasicBlock (using two 3X3 convolutions, followed by BN and ReLU) and Bottleneck (3 convolutions, respectively 1X1, 3X3, 1X1 convolution kernels, these three convolution kernels are used to compress dimensions, convolution processing, and restore dimensions). Figure 4 below shows the structure of the two basic modules and Figure 5 shows the overall structure of different ResNet.

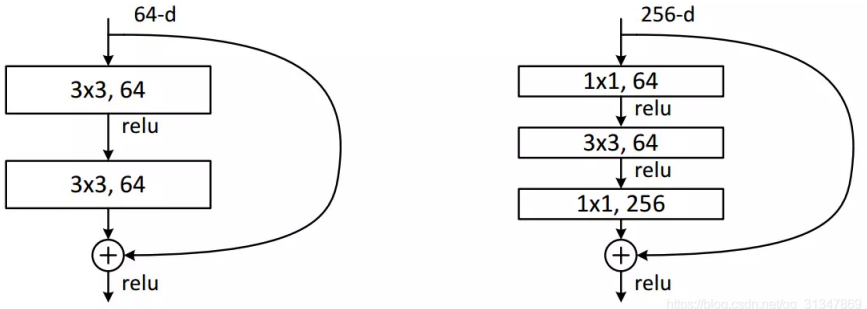


Figure 4: two basic modules of ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 5: overall structure of different ResNet

In this task, the team first implemented the ResNet18, ResNet34 and ResNet50 network, then Choose SGD as the optimizer with cross entropy loss. The result is as below, for the first graph, we use ResNet18 structure without learning rate decay and the accuracy rate fluctuated a lot in Figure 6. Therefore, after implementing the learning rate decay in Figure 7, the final accuracy rate is around 80%. Besides, when we make the net structure deeper like ResNet34 in Figure 8, the difference is not obvious and even a little worse than the ResNet18. Furthermore, we tried the ResNet50 and the result is not as good as ResNet18 either. After checking the accuracy of training sets, because this flower data set is not large enough, this may be due to over-fitting since the model fits the training data well but perform not good enough on the test data.

Furthermore, the team thought the simpler ResNet9 framework might have better result in term of accuracy in the validation data set. It is important to consider the learning rate and optimization

method during the training model process. For example, at the beginning of the training, the model required higher learning rate because overall lower accuracy and processed data from the ResNet9 are much more accuracy than the old model. However, after 10 epochs, the existing model already have a "satisfied" accuracy rate the model might need to learn small from the processed data. Also, the model might need to use different optimization methods like Adam and Stochastic gradient descent during the training process. For example, the first 15 epochs, the Adam method is the optimization method and , the last 5 epochs, the Stochastic gradient descent is the optimization methods. Overall, the team got a satisfied accuracy rate 84 percent 10 by using the ResNet9 framework. This is the accuracy graph 11 and loss graph in training and validation data set 11.

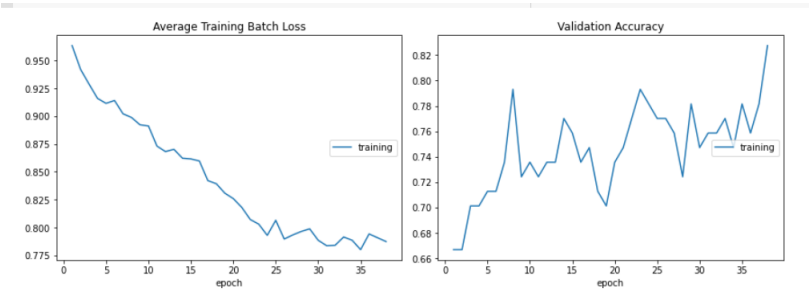


Figure 6: Accuracy and loss of ResNet18(without learning rate decay)

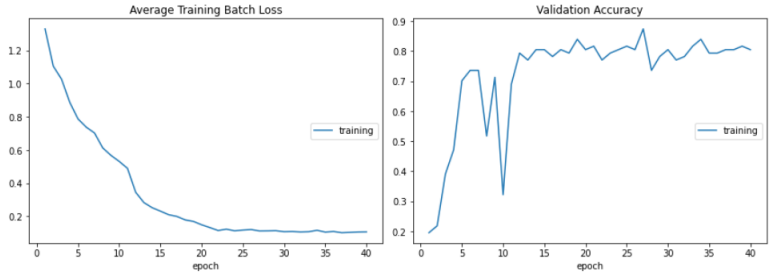


Figure 7: Accuracy and loss of ResNet18(with learning rate decay)

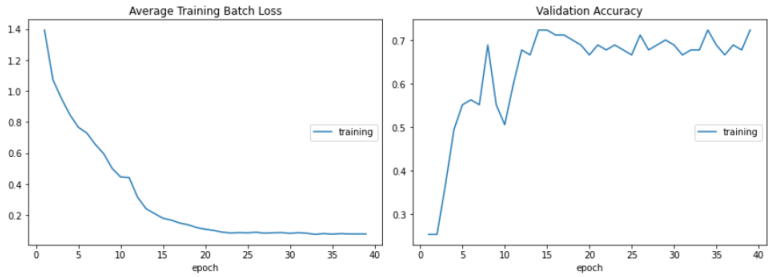


Figure 8: Accuracy and loss of ResNet34

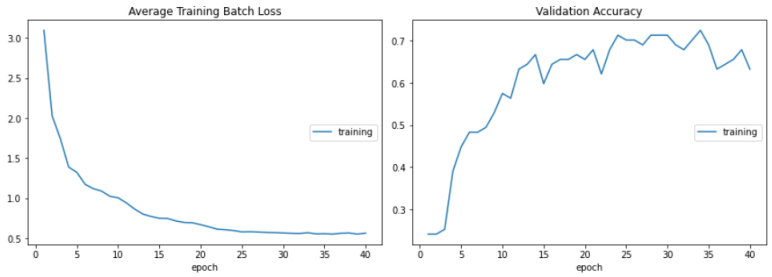


Figure 9: Accuracy and loss of ResNet50

```
history += fit(5,0.001,model , train_dl,valid_dl, torch.optim.Adam)

Epoch [0], train_loss: 1.4444, val_loss: 1.6752, val_acc: 0.4847
Epoch [1], train_loss: 0.9512, val_loss: 1.1238, val_acc: 0.5765
Epoch [2], train_loss: 0.8752, val_loss: 0.9057, val_acc: 0.6511
Epoch [3], train_loss: 0.7880, val_loss: 0.8909, val_acc: 0.6924
Epoch [4], train_loss: 0.7500, val_loss: 0.8008, val_acc: 0.7217

[40] history += fit(5,0.001,model , train_dl,valid_dl, torch.optim.Adam)

Epoch [0], train_loss: 0.8575, val_loss: 1.0037, val_acc: 0.6472
Epoch [1], train_loss: 0.6952, val_loss: 1.1974, val_acc: 0.5884
Epoch [2], train_loss: 0.6708, val_loss: 0.8431, val_acc: 0.6883
Epoch [3], train_loss: 0.5770, val_loss: 0.7941, val_acc: 0.7567
Epoch [4], train_loss: 0.5381, val_loss: 0.7675, val_acc: 0.7128

[41] history += fit(5,0.0001,model , train_dl,valid_dl, torch.optim.Adam)

Epoch [0], train_loss: 0.3975, val_loss: 0.5821, val_acc: 0.7913
Epoch [1], train_loss: 0.3573, val_loss: 0.5885, val_acc: 0.8117
Epoch [2], train_loss: 0.3395, val_loss: 0.6008, val_acc: 0.8209
Epoch [3], train_loss: 0.3161, val_loss: 0.6066, val_acc: 0.8025
Epoch [4], train_loss: 0.3055, val_loss: 0.5864, val_acc: 0.8248

history += fit(5,0.0001,model , train_dl,valid_dl, torch.optim.SGD)

Epoch [0], train_loss: 0.2668, val_loss: 0.5939, val_acc: 0.8243
Epoch [1], train_loss: 0.2711, val_loss: 0.5791, val_acc: 0.8137
Epoch [2], train_loss: 0.2757, val_loss: 0.5950, val_acc: 0.8137
Epoch [3], train_loss: 0.2706, val_loss: 0.6042, val_acc: 0.8151
Epoch [4], train_loss: 0.2735, val_loss: 0.5884, val_acc: 0.8360
```

Figure 10: Model training result by using ResNet9

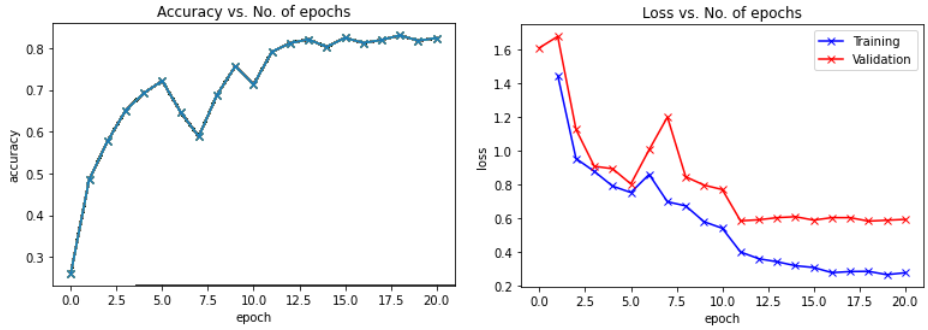


Figure 11: ResNet9 accuracy and loss between train/valid dataset

2.3 VGG11

VGG is a neural network model that uses convolutional neural network (CNN) layers and was suitable for image classification challenge. VGG has many configurations 12, here we use configuration "A", which has 11 layers with weights known as VGG11. The core idea of VGG is to repeatedly use 3x3 convolutional layer and 2x2 pooling to increase depth of network.

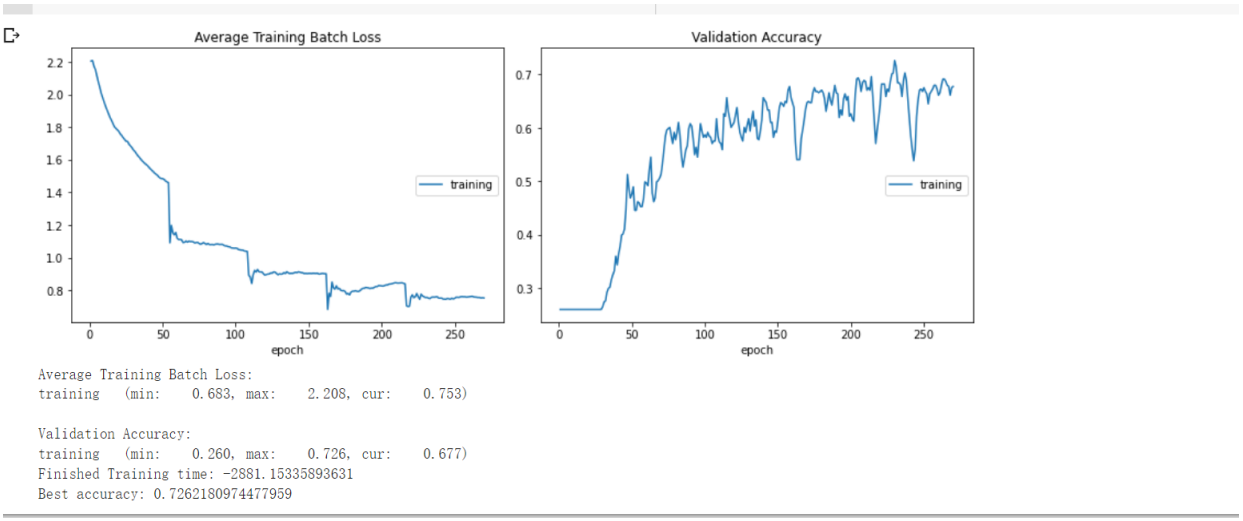
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 12: Architecture of configurations of VGG

First, we tried to implement VGG11 net and VGG16 net. In data processing, we tried many useful data augmentation methods, such as random crop, random horizontal and vertical flip. Choosing SGD as the optimizer with cross entropy loss, we successfully complete the training process though the result is not so good 13.

When we want to optimize the accuracy, we had to take deeper configuration of VGG, like VGG16, however, the deeper the network is, the more computational resources are needed, especially for GPU and memory. Also the change in optimizer and loss function have less to do with the result than totally change the network. To make the training process faster with higher test accuracy, we tired transfer learning, using the pre-trained VGG11 net.

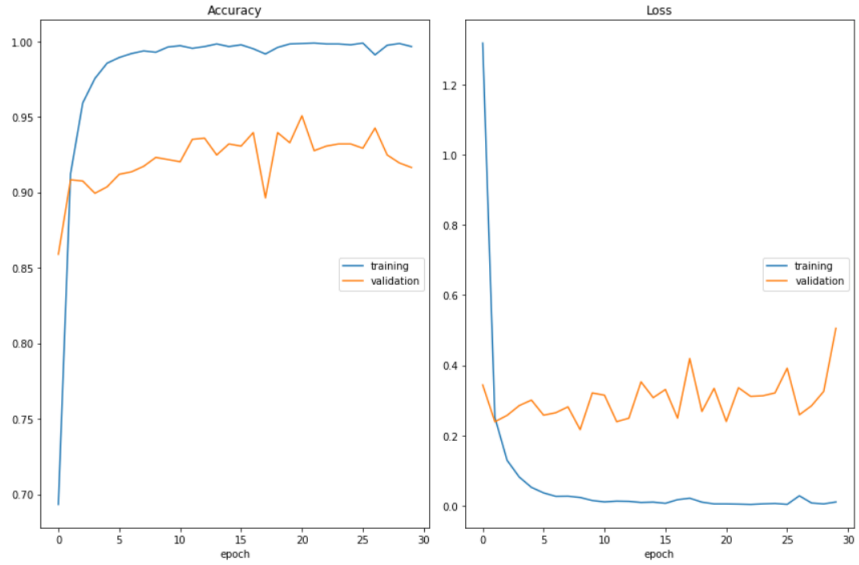
VGG11 and pre-trained VGG11 have the same net architecture, the only difference is that we take random weights to initialize the VGG11, while in pre-trained VGG11 with parameters that have already been trained for a certain task. Again, we used the same cross entropy loss, change the optimizer to Adam which can automatically change the learning rate. Taking the batch size as 64, learning with 30 epochs, we got pretty good validate accuracy of about 95 % in fairly short time 14.




```
Average Training Batch Loss:
training   (min:    0.683, max:    2.208, cur:    0.753)

Validation Accuracy:
training   (min:    0.260, max:    0.726, cur:    0.677)
Finished Training time: -2881.15335893631
Best accuracy: 0.7262180974477959
```

Figure 13: Accuracy and loss of VGG11 in 250 epochs




```
Accuracy
        training                (min:    0.693, max:    0.999, cur:    0.997)
        validation              (min:    0.859, max:    0.951, cur:    0.916)
Loss
        training                (min:    0.003, max:    1.318, cur:    0.010)
        validation              (min:    0.217, max:    0.505, cur:    0.505)
Epoch: 30 | Epoch Time: 1m 8s
        Train Loss: 0.010 | Train Acc: 99.65%
         Val. Loss: 0.505 |  Val. Acc: 91.64%
```

Figure 14: Accuracy and loss of pre-trained VGG11 in 30 epochs

3 Conclusion

In sum, the team did a great job on improving the accuracy of picture recognition after trying all three convolutional neural network framework: AlexNet, ResNet, and VGG11. For AlexNet, the team encounter the difficulties like implement correct dimension for input and output of the network. The team managed to lower the loss to one extremely low level after 1000 epochs and got maximum accuracy around 73 percent. For ResNet, the team implemented the ResNet9, ResNet18, ResNet34 and ResNet50. When the team implemented more layers like from RN18 to RN34 and RN50, RN34 and RN50 actually have lower maximum accuracy than that of RN18. The optimal accuracy for RN18 is roughly over 80 percent. Furthermore, after implementing Adam optimizer, the ResNet9 reached

84% of accuracy. For VGG11, the team implemented both VGG16 for the accuracy optimization, pre-trained VGG11 and VGG11 for the main framework. After lowing the revolution of pictures, the team got a really high accuracy like 95 percent on the validation data set.Therefore, the team decide to choose the pre-trained VGG11 neural network framework for the "flower recognition with five classes" project.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.