Histopathologic Cancer Detection
CS 184A/284A Final Report

Pooya Khosravi
pooyak@uci.edu
38127137

Abstract

In this project, I trained different convolutional neural network architectures with different weight initializations to identify the presence of metastasis tissue in digital histopathology images. Two models were trained using random initialization. For the last model, I used transfer learning with densenet169 as the base. Furthermore, I evaluated and compared these neural network architectures in terms of computational complexity and model accuracy.

1 Introduction

1.1 Problem Statement

Breast cancer is one of the main causes of cancer death worldwide. The diagnosis of biopsy tissue with hematoxylin and eosin stained images (figure 1) is non-trivial and specialists often disagree on the final diagnosis. A pathologist's report is often the gold standard in the diagnosis of many diseases and the pathologists require years of training to be able to gain enough expertise and experience to diagnose a patient's case. When it comes to cancer, these diagnosis have profound impact on a patient's therapy and future directions for treatment.

Not only a major problem facing the field of histology for the detection and classification of breast cancer metastases in lymph nodes is how tedious and time-consuming the task is for pathologists, small metastases are also very difficult to detect and sometimes they are missed. Furthermore, even with this extensive training pathologists go through, different pathologists have different diagnostics for the same patient.

This paper reports on different deep neural network based solutions to identify the presence of metastases cancer cell in histopathology images.

1.2 Previous Work

To assist in human decision making for medical diagnostics, a number of models such as support vector machines, convolutional neural networks (CNN), probabilistic neural networks, and recurrent neural networks are being used [1] in different applications. Previous state of the art for cancer detection in histological images have typically consisted of CNNs [2]. Using the training images, CNNs learn low to high level features from image patches. In image classification in different fields, these CNN deep neural network models have achieved excellent performance [3, 4], in particular on histopathology images [5, 6].

1.3 My Contribution

I decided to train 3 architectures designs as in table 1. Then, I evaluated their accuracy and AUC score on the testing dataset provided by Kaggle.

Design #	Similar model	Initialization
1	VGG16	Random
2	AlexNet	Random
3	densenet169	Pre-trained

Table 1: Summary of the architecture designs trained for cancer detection in histological images.

2 Data

I will be using digital histopathological images. These are usually large images that need to be sliced into smaller images. Since a sliced version of the images were available on Kaggle, I decided to use Kaggle Histopathologic Cancer Detection competition data.

2.1 Kaggle Dataset

This data is originally from PCAM dataset [7] which is derived from slicing 400 images in the CAMELYON16 challenge into 96x96px images. The dataset includes a large number of pathology

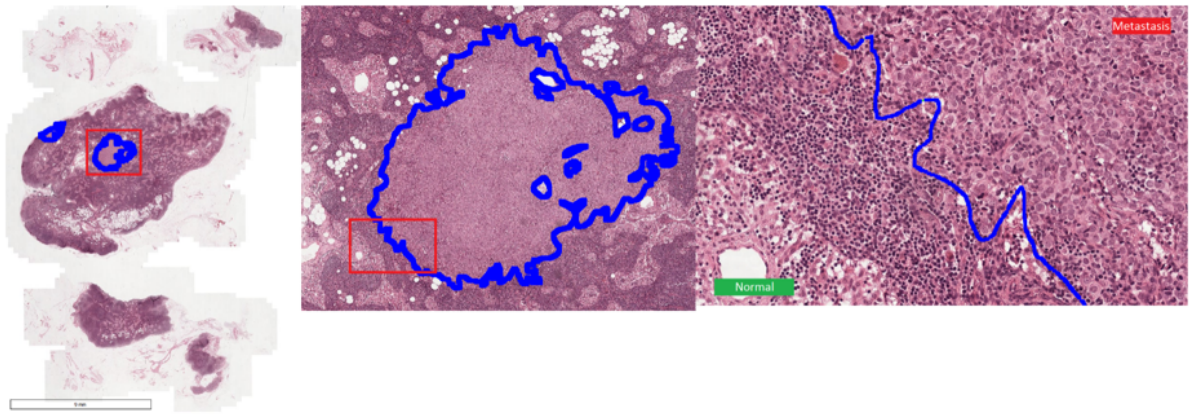

Figure 1: **Glass Slide Sample:** An example of hematoxylin and eosin stained tissue slide under microscope.

images that have been labeled with ground truth. There are 220k training images and labels indicating whether there is at least one pixel of tumor tissue in the center 32x32 region.

The PCam's dataset uses 10x undersampling to increase the field of view giving the resultant pixel resolution of 2.43 microns. The training data (total of 220,025 images) has a class distribution of 60:40 negative and positive samples as seen in image 2 which is fairly balanced. The dataset also includes 57,468 test images used for Kaggle public board where the labels are unknown.

The labels only consider the center region for each image. A positive label indicates that the 32x32px patch in the center region contains at least one pixel of tumor tissue as shown in figure 3.

Existence of tumor tissue in the outer regions of the center patch does not influence the label. As part of the dataset description, it was provided that this outer region is "to enable fully-convolutional models that do not use zero-padding" [7] which ensures that the behavior of those models are consistent when applied to a whole-slide image.

2.2 Preprocessing

Since the images were already in 96x96px patches, there was no need to slice up the original 400 images in the CAMELYON16 challenge. The image labels are only related to the center region of each image. Therefore, I cut the center 32x32px of each image to use for the training. Then, I decided to split the training data provided by Kaggle into train/test (90:10) split. It is important to maintain equal ratios of negative/positive (60/40)

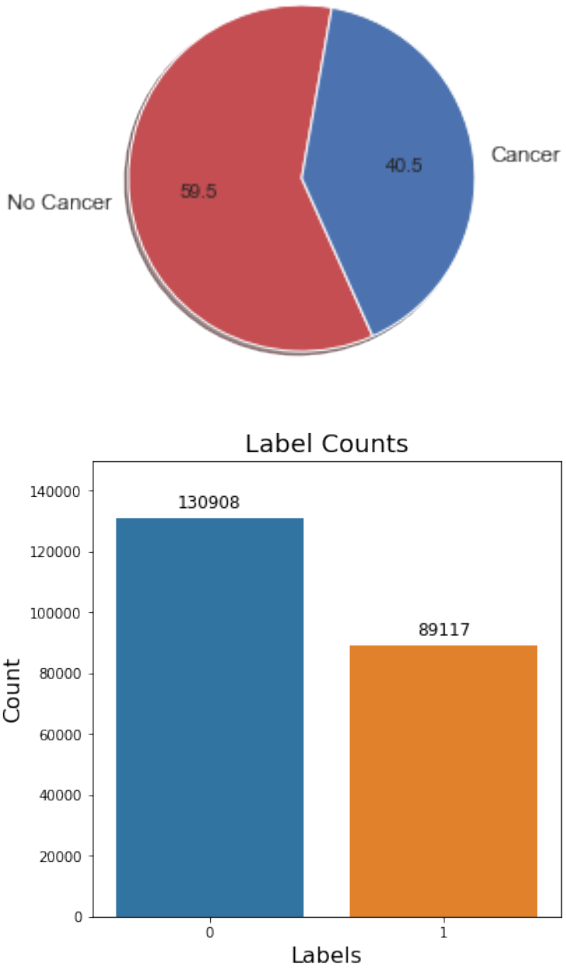

Figure 2: **Dataset Distribution:** Distribution of class labels in the training dataset (top) and the number of images in negative and positive classes (bottom, total = 220,025 images).

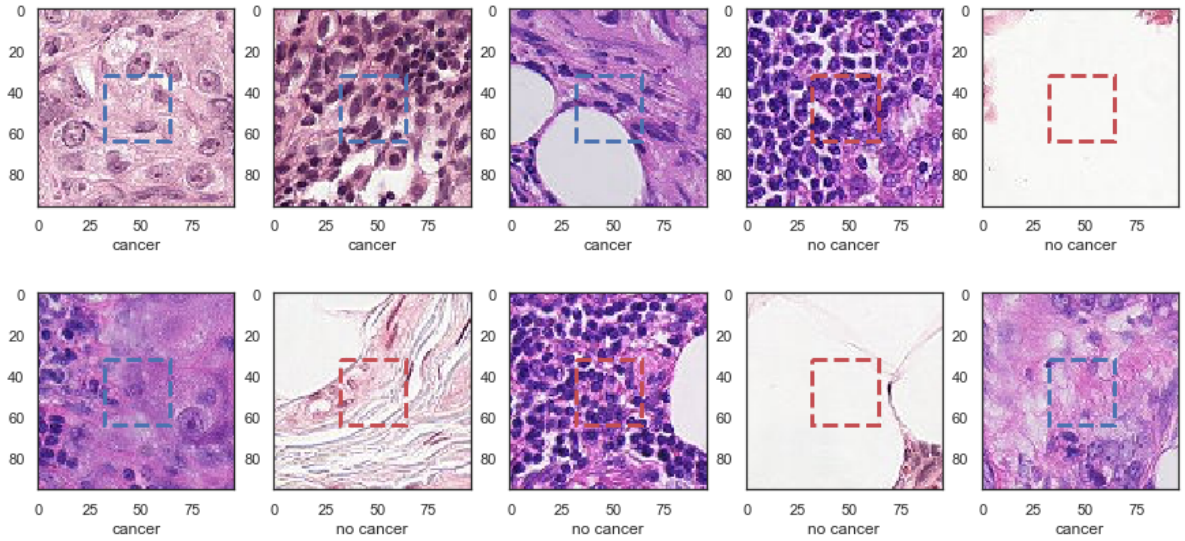

Figure 3: **Dataset Examples:** Example of some the images in the dataset. As described in the text, the classification is only for the center marked regions.

in both training and test splits, but since the data is fairly balanced, a random split would be unlikely to cause severe under representation.

2.3 Augmentation

To help generalize the models, I implemented the on the fly augmentations summarized in Table 2. These augmentations are used to train our model to be more agnostic to placement and rotation of each image. Figure 7 in the appendix illustrates these augmentations on an image.

Desired Flexibility	Augmentation
Translation	Random Crop
Rotation	Random Flips
Rotation	Random Rotations
Lighting/Color	Random Lighting

Table 2: Summary of the augmentations used for images.

3 Approach

I will be using deep learning approaches to accomplish binary classification on histopathological images to detect presence of cancer. There will different iterations of model architecture to compare and contrast simple and complex models on the dataset. Each model is further explained below.

3.1 Architecture Design

Design 1 (VGG16) The network layers is based on VGG16 [8], but is not pre-trained on the ImageNet Large Scale Visual Recognition Competition (ILSVRC) classification task [3]. This model is implemented using PyTorch similar to figure 8. This model contains 12 layers of CNN filters with 3 Dense layers at the end for predictions. Each layer uses ReLU as activation function with an addition of batch normalization. After every 3rd layer, there is an additional max pooling layer.

Design 2 (AlexNet) This network model is similar to AlexNet [4] (Figure 9) and is trained using random initialization. This model include 5 layers of CNN filters with 3 Dense layers at the end for predictions. Similar to previous design, ReLU is used as activation function with the addition of batch normalization to each layer. Different from AlexNet, this deign has a max pooling layer after each CNN layer.

Design 3 (Transfer Learning) I used Fast.ai V1 that is built on PyTorch. This library has out of the box solutions for different pre-trained network with amazing tutorials to follow through. For this task, I specifically chose densenet169 as the base model and decided to not add any additional layers. This model uses pre-trained weights of densenet169.

3.2 Transfer Learning with densenet169

The densenet169 architecture expects 224x224 input images and has 1000 output neurons (for the 1000 classes of ImageNet). For the PCAM Kaggle dataset, we need to adapt the first part of the con-

volutional network since the images are 32x32px after being cropped. Fortunately, the Fast.ai library takes care of that by providing a transformation function for the images which we can simply provide it with the final size of the image (32x32px).

When it comes to training models using transfer learning, different strategies (figure 11) are recommended that involves either freezing or not freezing weights of the pre-trained model. Freezing the weight can ensure that the model doesn't overfit to the dataset that we're using. In this implementation, I decided to freeze every layer other than the last classification layer of the densenet169 model.

3.3 Training

Since the data set includes about 220k images with a size of approximately 6 Gb, I ran into issues training on my personal laptop. My personal laptop has a Nvidia GTX 1050 graphics cards and the training took longer than expected. Hence, I decided to do the large portion of training using Kaggle Notebooks. Kaggle provides 30 hours of GPU compute time for free every week that I was able to take advantage of. Using my computer, design 1 and 2 both approximately took about 3 hours each. On the other hand, both took less than an hour to train when using Kaggle GPU. For design 3, the model took about 5 hours on my local computer, but only taking 75 minutes on Kaggle GPU.

Model #	Batch size	# of Epochs
Design 1	128	20
Design 2	256	6
Design 3	128	8

Table 3: Summary of batch size and number of epochs for each model.

As seen in Figure 4, design 1 training and validation loss decreased after each epoch, suggesting that the model is learning. This metric, of course, is not complete and the data might be overfitting to the dataset, but due to limited time and gpu resources, no additional data sets were tested.

4 Software

I used Python this project. For data analysis, I used Pandas framework. For the machine learning portion of the project, I used PyTorch and Fast.ai libraries. For visualizations

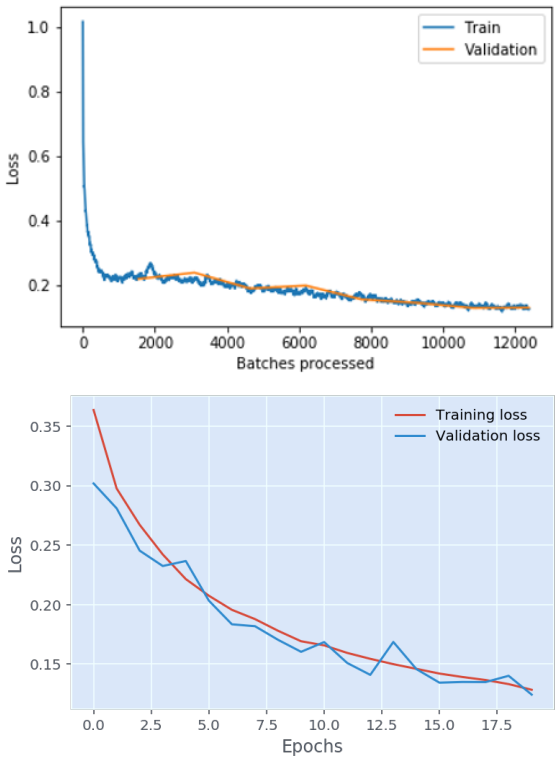

Figure 4: **Training and Validation Losses:** Training vs Validation Loss of the Kaggle training for design 1 (bottom) and transfer learning (top).

and plotting, I used seaborn and matplotlib libraries. The code for training can be found here (https://github.com/pooyakhosravi/cs184a). The repository contains different notebooks for each model and the data analysis that has been done on the data set.

4.1 Code I Wrote

I implemented the data analysis, the code for model designs 1 and 2, in addition to the code for evaluation. For design 3, I heavily referenced a Kaggle notebook as I wasn't able to make my own fast.ai implementation working probably without issues.

4.2 Code Referenced

For the implementations of the Fast.ai model using densenet169, I referenced the following material which contained amazing tutorials on how to use fast.ai:

1. Most up voted notebook (link) on the Kaggle Competition

2. Fast.ai course series videos (link)

3. Fast.ai tutorial (link)

5 Results

5.1 Evaluation Metrics

AUC is the metric that Kaggle uses for evaluating each submission. For each model, I used the validation set for several metric including accuracy of prediction and area under the ROC curve. The model accuracy is the simply the portion of the images in the validations set that are predicted correctly.

5.2 Model Performances

As seen in figures 4 and 5, all three models had no issue with learning. All three performed with an accuracy higher than 94% on the training set. The accuracy and auc score (from kaggle submission) of each model can be found in table 4. As seen in the table 4 the model with pre-trained weights performed better than the two models with random weight initialization in terms of both accuracy and area under the curve.

Model #	Accuracy(%)	AUC
Design 1	0.934	0.975
Design 2	0.923	0.968
Design 3	0.953	0.988

Table 4: Summary of accuracy and AUC score from Kaggle submission of each model.

6 Discussion

As seen in table 4, the pre-trained model using transfer learning performed better than the other two models. With transfer learning, the model doesn't start the learning process from scratch, it starts from patterns that have been learned. These patterns have been thought to represent lower level features that are common to all images. This way, we are leverage previous learnings and avoid starting from scratch.

Another reason why design 3 might have performed better is the length of training. As mentioned before, the training of design 3 includes took almost twice as long as the other 2 designs although the number of epochs was less than design 1.

7 Conclusion

This report evaluates different alterations of CNN architecture to compare and contrast the results in the detection of breast cancer metastases in lymph nodes.

Three different models were trained with different architecture and weight initialization strategies to identify the presence of metastases from 3x96x96px digital histopathology images. Through these experiments, it is suggested that using pre-trained weights from models trained on very large data sets will perform better than random initialization.

References

[1] H. Kordylewski, D. Graupe, and Kai Liu. A novel large-memory neural network as an aid in medical diagnosis applications. *IEEE Transactions on Information Technology in Biomedicine*, 5(3):202–209, Sep. 2001. ISSN 1558-0032. doi: 10.1109/4233.945291.

[2] Teresa Araújo, Guilherme Aresta, Eduardo Castro, José Rouco, Paulo Aguiar, Catarina Eloy, António Polónia, and Aurélio Campilho. Classification of breast cancer histology images using convolutional neural networks. *PLOS ONE*, 12(6):1–14, 06 2017. doi: 10.1371/journal.pone.0177544. URL `https://doi.org/10.1371/journal.pone.0177544`.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] 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 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] Geert Litjens, Clara I Sánchez, Nadya Timofeeva, Meyke Hermsen, Iris Nagtegaal, Iringo Kovacs, Christina Hulsbergen-Van De Kaa, Peter Bult, Bram Van Ginneken, and Jeroen Van Der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6:26286, 2016.

[6] Korsuk Sirinukunwattana, Shan e Ahmed Raza, Yee-Wah Tsang, David RJ Snead, Ian A Cree, and Nasir M Rajpoot.

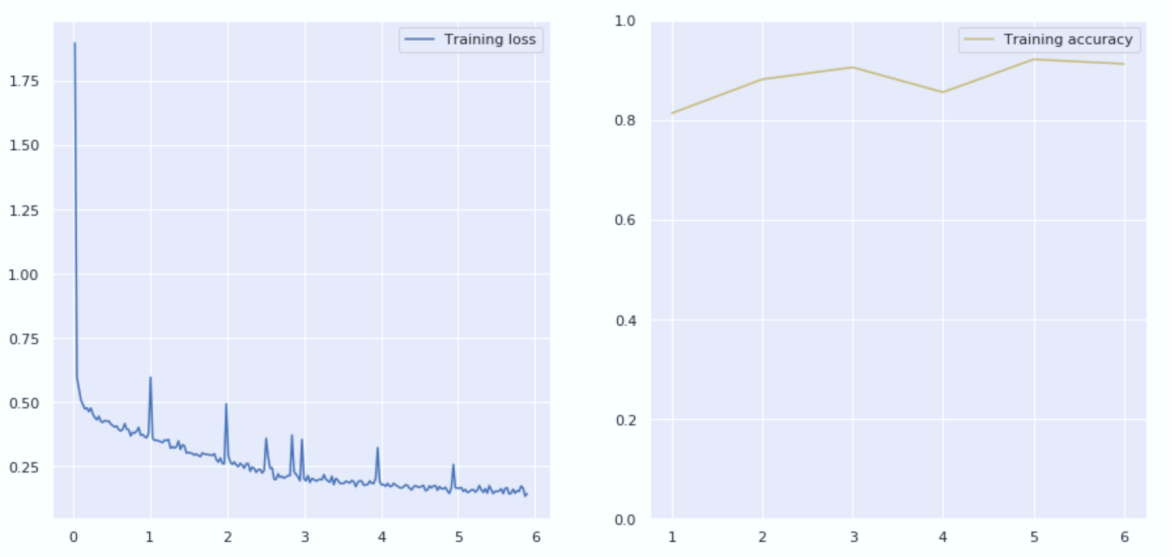

Figure 5: **Training and Accuracy of Design 2:** This figure shows the accuracy and training loss of design 2.

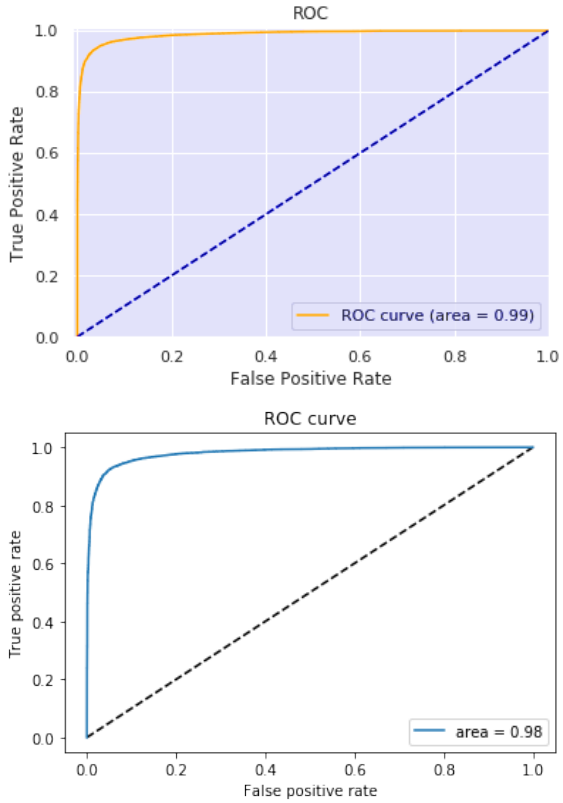

Figure 6: **ROC Evaluations:** This figure shows the AUC score of the design 1 (bot) and transfer learning (top) on the training data.

Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. *IEEE Trans. Med. Imaging*, 35(5):1196–1206, 2016.

[7] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes van Diest, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. van der Laak, , and the CAMELYON16 Consortium. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. *JAMA*, 318(22):2199–2210, 12 2017. ISSN 0098-7484. doi: 10.1001/jama.2017.14585. URL `https://doi.org/10.1001/jama.2017.14585`.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Appendix A Figures

Figures are included in the following pages.

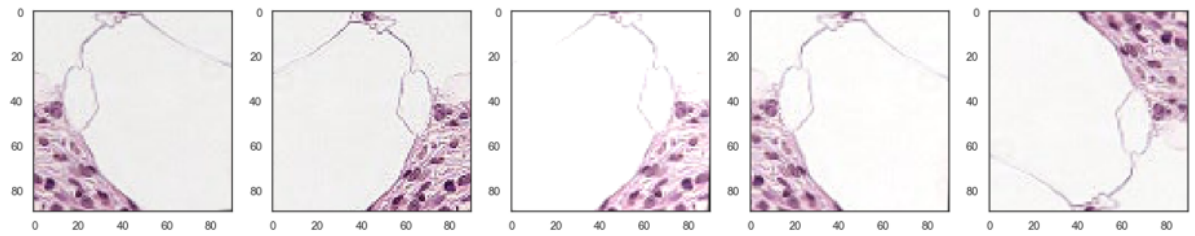

Figure 7: **Augmentations:** This figure shows on the fly augmentations applied to an image.

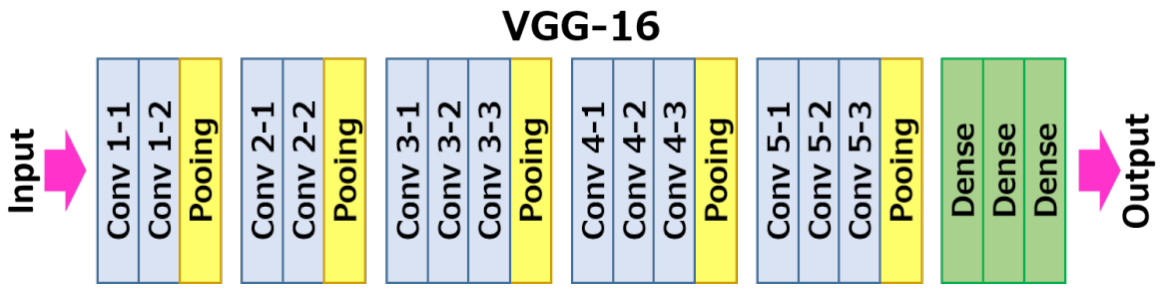

Figure 8: **VGG16 Architecture:** This figure shows the architecture for VGG16.

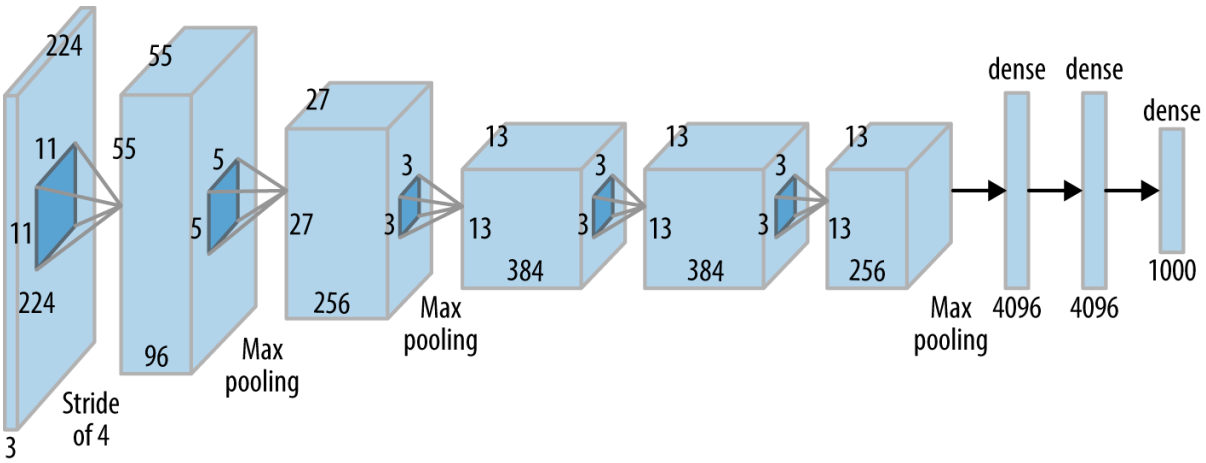

Figure 9: **AlexNet Architecture:** This figure shows the architecture for AlexNet. In the implementation used in this paper, the input size of the image is 32x32.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 10: **Densenet169 Architecture:** This figure shows the architecture for Densenet169.

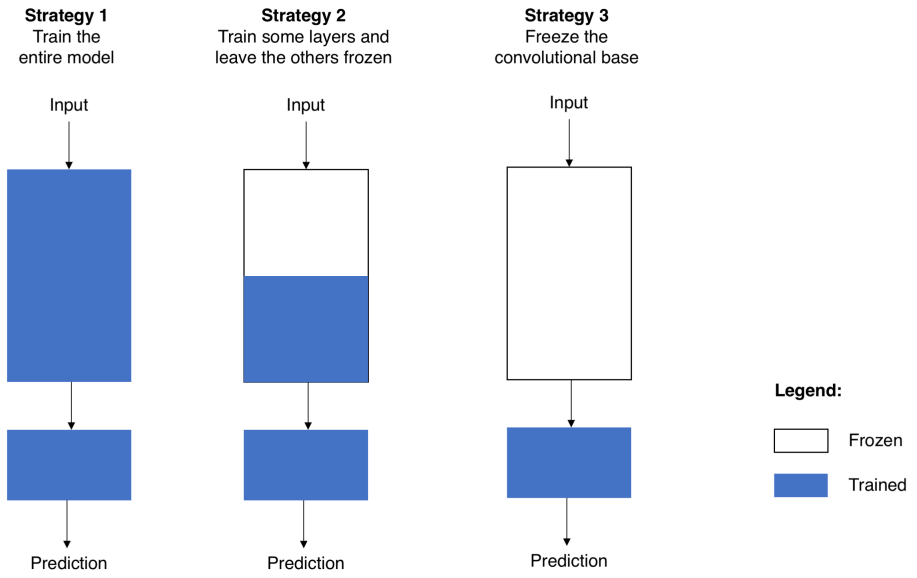

Figure 11: **Transfer Learning Training Suggestions:** This figure shows suggestions about freezing/unfreezing weights when it comes to transfer learning to avoid overfitting.