UNIVERSITY OF CALIFORNIA

Los Angeles

Identifying the Orientation of Medical Images in the Veterinarian Field Using Computer Vision

A thesis submitted in partial satisfaction of the

requirements for the degree Master of Applied Statistics and Data Science

by

Yoonho Kim

2024

ABSTRACT OF THE THESIS

Identifying the Orientation of Medical Images in the Veterinarian Field Using Computer Vision

by

Yoonho Kim

Master of Applied Statistics and Data Science

University of California, Los Angeles, 2024

Professor Yingnian Wu, Chair

With the expansive amount of data, modern technologies including machine learning and deep learning have become extremely complex and accurate. Deep learning models started to gaining popularity in the early 2010. AlexNet is a type of Convolutional Neural Network (CNN) was first introduced and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This model dramatically increased accuracy compared to the previous year by nearly ten percent, resulting in many variations of CNN architectures being introduced based upon this model. In the year of 2015, deep learning reached a new milestone because of a new model architecture called Residual Network (ResNet), which is another variant of CNN architecture. ResNet was trained on ImageNet, a database that contains a wide range of various images, and the ability to accurately predict those images started to outperform human's abilities (Figure 1).

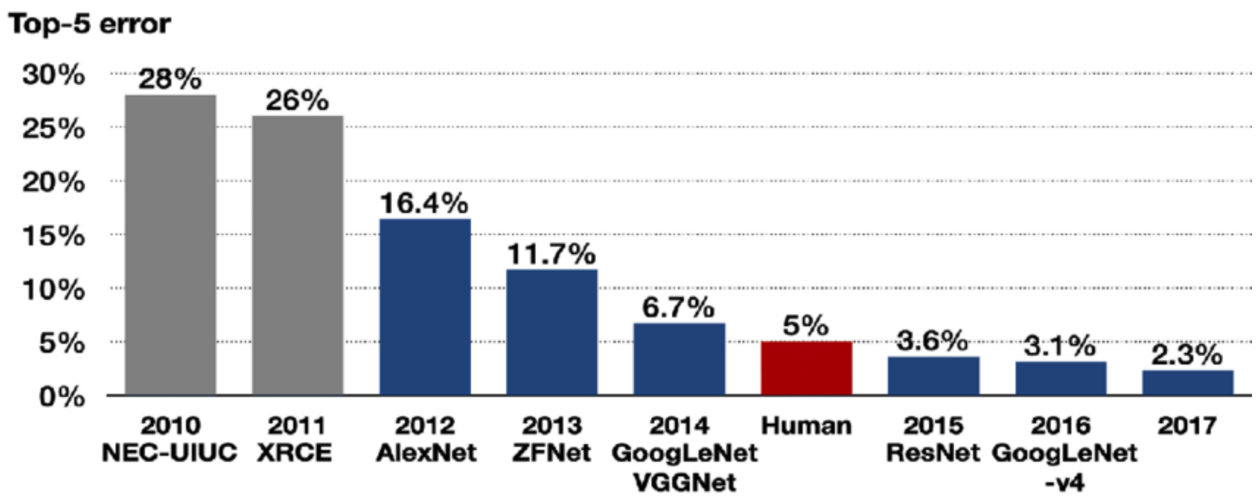


Figure 1: Evolution of Computer Vision

Source: "Application of Deep Learning in Dentistry and Implantology," Kang & et al. 2020, P.151

This significant improvement in Computer Vision models allows many researchers to help implement modern technologies such as self-driving cars and face recognition. After gaining its credibility and accuracy from recent state-of-the-art (SOTA) models, various domains, including health care, have begun to adapt the methodology to better predict one's medical conditions such as the criticalness of tumor and detect if the malignant or benign from medical images including X rays and CT scans. Throughout this paper, we will discuss the application of modern Computer Vision techniques in the Veterinary field with the hope of reducing the number of canine/feline patients who have serious illness and increasing their quality of lives by detecting possible illness in their early stages.

The thesis of Yoonho Kim is approved.

Frederic R. Paik Schoenberg

Mahtash Maryam Esfandiari

Yingnian Wu, Committee Chair

University of California, Los Angeles

2024

List of Figures

List of Tables

1. Overview

Vetology utilizes modern machine learning techniques in order for individuals to better understand the inner symptoms of canine/feline patients. In many cases dogs and cats may suffer from underlying symptoms that we may not been keen on picking up, this has resulted in many furry family members being untreated or treated too late due to the inability to pick up on early symptoms. Vetology aims to help identify and address potential concerns earlier so that vets can make informed decisions on treatment plans that could save the life of your furry family member. The firm utilizes software that efficiently applies Computer Vision (CV), Natural Language Processing, and Large Language Model (LLM) techniques in order to properly predict medical input images and diagnose possible illnesses that canine/feline patients might have developed internally and generate an Artificial Intelligence (AI) screening report based on the input images.

Medical images are the main inputs for checking possible illness and generating the reports, making extremely important for developers to feed the correct orientation of images for this software. Radiologic technicians capture various images and various angles from a specific area in order to help medical professionals better understand the circumstance of given patients since those professionals can investigate at various angles. However, providing those images into a machine learning model can lead to false information as the machine learning model was trained on a specific type of inputs. Hence, in order for the model to generate appropriate outcomes, it is crucial for developers to feed the correct orientation of input images to the machine learning models.

There are four possible angles from the medical images; zero degrees, which is the desired outcome, 90 degrees, 180 degrees and 270 degrees. The goal of this project is to correctly identify the orientation of the given images. By providing the correct orientation, we can provide those images to the machine learning model, which helps the machine learning model predict possible illness more accurately. This orientation model will also reduce human supervision due to its automated process and convert those images to the correct orientation if they are not zero degrees.

We want to develop a machine learning model that detects the medical inputs orientation properly in which we will achieve through experimenting with various data steps including data preprocessing, feature engineering, and various model developments throughout this project. With the development of this model, we will be able to feed the correct input to the model, which helps increase the model performance in the AI generated screening reports.

2. Methodology

The goal of this project is to correctly identify the orientation of the medical inputs, therefore, the focus of the model development is the accuracy of the model, which meant we did not have to emphasize on model expandability. Depending on the potential business issues, there are certain cases where we need to consider model interpretability. In these scenarios, it is often better to utilize data mining techniques such as linear regression or tree-based models. If the problems are more related to how the models accurately predict possible outcomes, we utilize deep learning-based models. Since the goal of the project is to correctly predict the orientation of the images, we have been implementing various deep learning techniques including Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) and more advanced CNN based architectures such as GoogLeNet or EfficientNet, which are pretrained models that are available in the Pytorch libraries in Python, in this project.

This project was developed using a programming language called Python, which is the most widely used in the machine learning and data science community, and deep learning models were developed in the Pytorch library. Converting input images to vectors is performed via the Pillow library. Two popular Python data visualization libraries, Seaborn and Matplotlib, were implemented. Optuna was used for hyper parameter tuning to improve the model performance by changing parameters such as learning rates, batch sizes, or number of layers within the network. Pandas and Numpy were also used for basic data manipulation. In this section, we will cover input images, data preprocessing and feature engineering.

2.1 Input Images

Input images can be defined as features in deep learning which are used for a model to extract information and they can come in a variety of mediums. For the purposes of this project, we have utilized canine/feline patients' medical images including Digital Radiographs (DX) and Computed Radiographs (CR) as input images. Historical medical images are stored in Vetology's Ubuntu server. Sample data, which was a total of 3,298 jpg files from the server, were provided and will be utilized for the deep learning model to learn the specific patterns of each orientation. The images are grayscale, which is an important feature for developing deep learning models. The size of each image is 1,972 by 3,268.

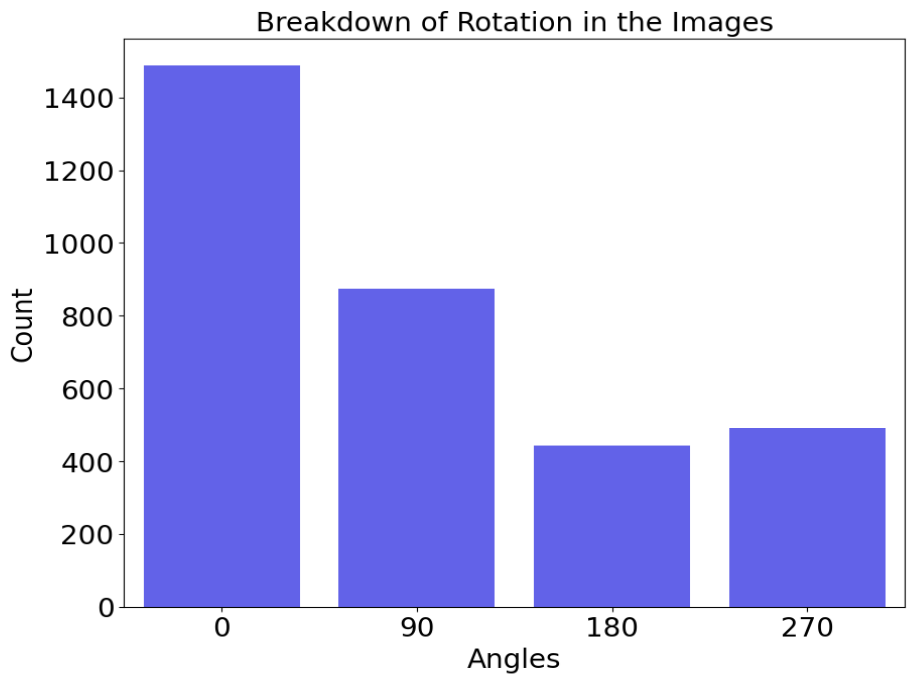


Figure 2: The Distribution of Input Images Based on the Angles

Angle	Count	Percentage
0	1,488	45.12%
90	874	26.5%
180	492	14.92%
270	444	13.46%
Total	**3,298**	**100%**

Table 1: Table of Input Images Based on the Angles

Figure 2 represents the distribution of images based on their angles and Table 1 analyzes the breakdown of each class. Based on the dataset, we could approach business problems in two different ways.

1) Create a binary classification model which predicts if images are at the orientation or not.

2) Create a multiclass classification model which predicts 4 different angles.

The first approach seemed to be easier to implement and might achieve higher accuracy than the second method, however, the second approach better aligns with business purpose because the model would modify to zero degrees when predicted values are not zero. By applying the second approach, we will be able to rotate the input images to zero if they are not fed properly.

In order to train a machine learning model for Computer Vision, input images such as jpg files cannot be trained themselves. Developers need to convert those images as tensors, which support the basic arithmetic operations in deep learning. There are various ways for individuals to convert images to tensors, but we utilized the Pillow library in Python for data conversion and augmentation. Detailed explanations for these input images will be discussed in the next section.

2.2 Data Preprocessing

Data preprocessing is required for us to train a machine learning model. This process will ensure the machine learning model to learn properly and efficiently if done correctly. There are few considerations needed for the data preprocessing in this project. The first consideration is the size of individual inputs. When we train deep learning models, we do not use jpg files directly, instead we first convert the jpg files to tensor, which is a multi-dimensional matrix that is used as the input for deep learning models. Converting more than three thousand jpg files that have a shape of 1,972 by 3,268 will consume a large amount of memory. The second consideration is the channel of the image. Unlike MLP architectures, we specify the input channel for the given image in convolutional layers in the CNN architecture. For grayscale, the input channel is one, while the input channel for Red-Green-Blue (RGB) is three. Although grayscale seemed to be logical for building the machine learning model, many advanced models such as GoogLeNet or

EfficientNet are trained based on ImageNet, which was based on RGB images. For this reason, we developed two different image conversions. The last consideration is the number of inputs in the dataset. Within deep learning, it is often beneficial to have more data. This is because deep learning models tend to have an extensive number of parameters that need to be trained and updated as layers get deeper. Although over three thousand medical images seemed to be sufficient, we aimed to produce more data to better help the training process of model. In this section, we will cover these obstacles and how to overcome them.

2.2.1 Defining Inputs

As mentioned above in section 2.1, individuals cannot directly use actual images by themselves to train a machine learning model. By converting those images to tensors, which are the fundamental data structure of deep learning that allows fast and efficient mathematical calculations, deep learning models can learn and update weights as well as bias from the actual outputs and predicted values. In this process, Pillow and Pytorch libraries in Python were employed to convert images to tensors.

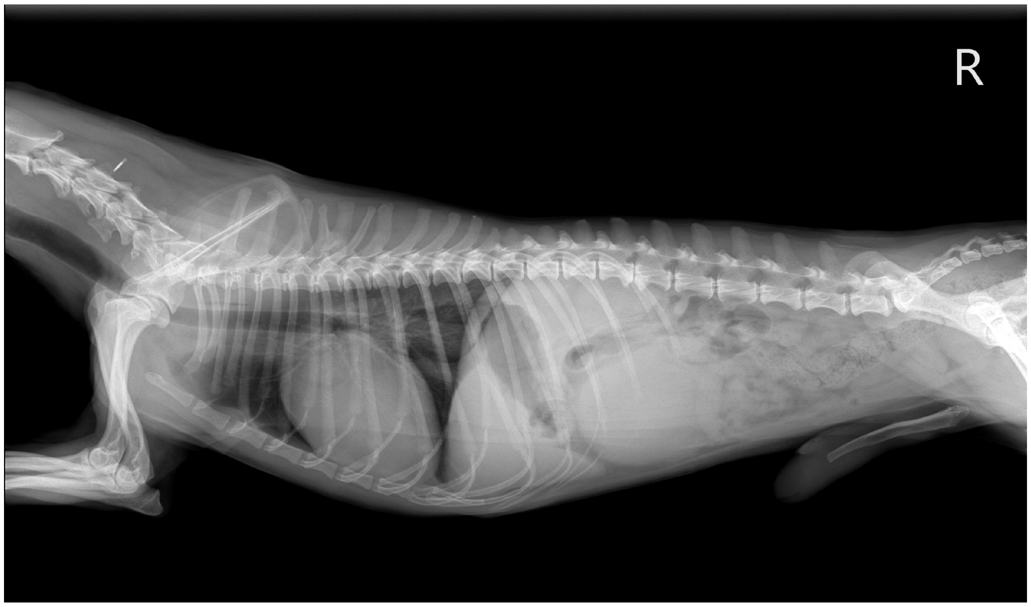


Figure 3: An Example of an Input Image in the Dataset

Figure 3 above is one of the examples in the input images. Each of the pixels in the image is stored as a number in the computer, which typically ranges from zero to 255. The lower the number, the darker it represents and Pillow library in Python helps convert the images to tensors. For example, after converting the image to a tensor using Figure 3 above, different integer values between 0 to 255 will replace each pixel value. After converting the images, it is often beneficial to normalize the input data because it will not only help convergence in the optimization process, but also utilize all the features equally. The ToTensor method under the torchvision.transfroms library helps achieve this goal by changing the images to the tensor forms and normalizing them as well.

2.2.2 Resizing Inputs

Converting these input images to tensor will be computationally intense and expensive since the original inputs typically have higher resolution in our use cases. It is often recommended to resize the input images so that we can have better computation power. However, there are pros and cons when resizing the input images. Higher input sizes will deliver detailed information regarding the inputs but will take more resources to process them which might lead to computational inefficiency. Conversely, lower input sizes will be quick to process the inputs, but the computer vision model will lose a significant amount of information. To meet both retaining sufficient information while computation remains reasonable, proper input images resizing should be performed. We developed two different data sources for different models since one approach is based on shallow network and the other is related to deeper networks. For the shallow network, it will be harder to extract information with higher resolution, while the deeper network will gain more information in the proper resolutions.

Although higher resolution appears to be better in most scenarios, there is a trade-off when resizing the input images. Resizing the input images to higher resolutions will be costly and computationally inefficient, while changing the input images to lower resolutions would lead to information loss. A recent study in Radiology utilizing deep learning proved these results. The author of the paper performed various training process using deep learning models in Radiology with various medical inputs. Based on this paper, the model performance improves as resolution increases, but it declines after reaching a certain point (Figure 4).

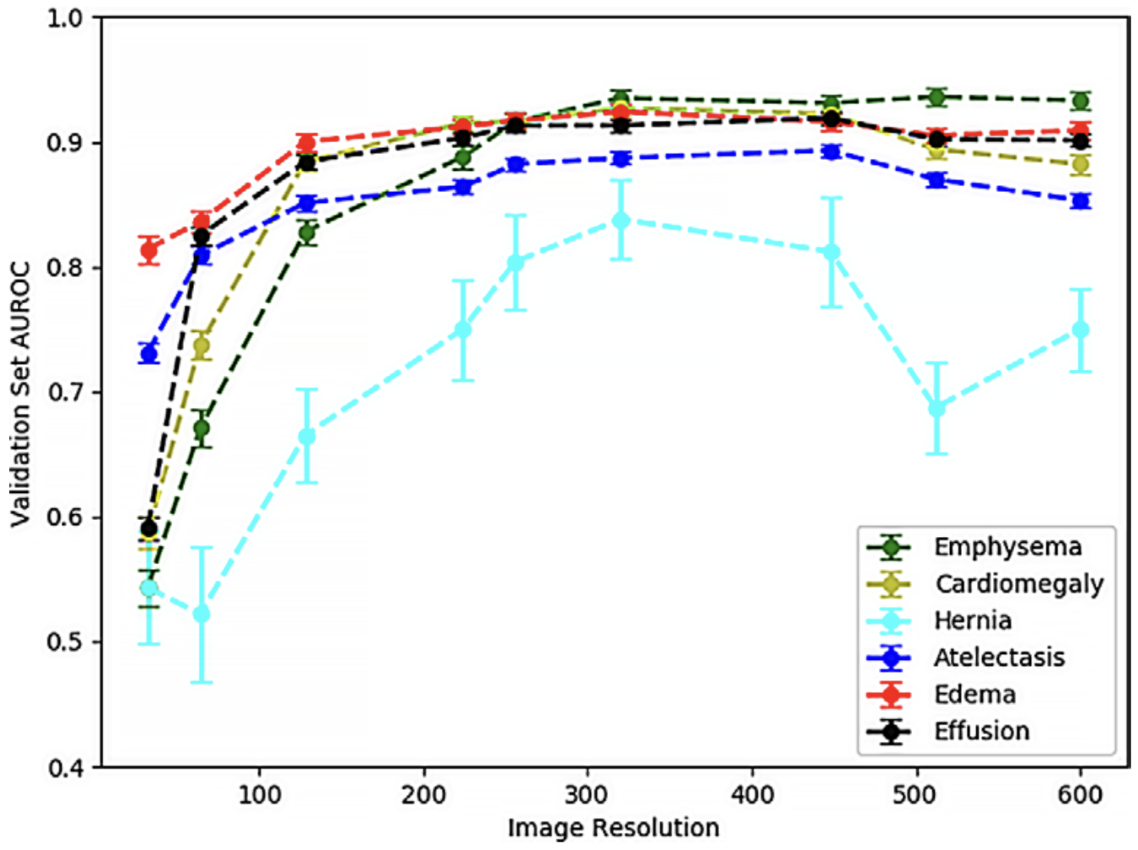


Figure 4: Model Performance Based on Image Resolutions

Source: “The Effect of Image Resolution on Deep Learning in Radiology,” Sabottke & Spieler. 2020, P.4

Figure 4 proves the pros and cons of resizing input images. The X axis shows various resolutions, and the Y axis shows their performance. As the X axis increases, we see that there is an upward trend regarding performance. However, after increasing the resolutions over 400 proves that the model performance does not improve significantly, instead it begins to decline. Therefore, it is important to find an optimal resolution for the given data.

The first dataset was designed for naïve models which often have shallower network architectures compared to the pre-trained models. Because these models often have a smaller number of parameters to tune due to their simplicity, previous research has proven that lower resolution in images can work fine such as LeNet with hand-written digits for classifying different numbers from zero to nine.

The input images for the first dataset were resized from 1,972 by 3,268 to 28 by 28. The second dataset was designed for pre-trained models such as GoogLeNet or EfficientNet. Pre-trained models above often to have more complex model architectures such as a larger number of layers, therefore, we need more information for enhanced tuning. In order to achieve more resources, we can either gather more data or use a higher resolution in the input images. These models were trained in relatively higher resolutions, which both models were trained in images that were at least more than 200 by 200. For the second dataset, we changed the channel of the input images and resized them to 224 by 224.

2.2.3 Changing Input Channels

Images stored in computers will have few different channels. If they are grey scaled images, then the input channel should be one. If they are colored images, then the channel should be three. The original input image is grayscale, which means the input channel for the convolutional layer should be one. As mentioned earlier, many pre-trained models were trained based on ImageNet, which were three channels, Red-Green-Blue (RGB) images; we wanted to increase the size of the input channel. In order to increase the input channel size, two different approaches were designed. The first solution was mapping the same image three times so that the input channel got converted from one to three. The second solution was to convert the input image to RGB using the Pillow library in Python.

2.2.4 Data Augmentation

Data augmentation is a technique where developers transform inputs to generate more data, which helps train a deep learning model. "Previous work has demonstrated the effectiveness of data augmentation through simple techniques, such as cropping, rotating, and flipping input images." (Perez & Wang, 2017) Although over three thousand images appear to be sufficient for training deep learning, it is beneficial for us to have more inputs which allow us to obtain higher accuracy or other metrics. Obtaining more data in the field of Computer Vision is relatively simple. Using pre-existing data, we can rotate the inputs or change the color of the inputs. This process is called data augmentation. Even though we had an option to gain more data naturally, we decided to utilize the data augmentation technique since the dataset was relatively simple and easy to manipulate.

Data augmentation in this project was simply created by following these two steps. First, images that had zero degrees were selected. Rotated angels were another option to consider, therefore, it was easier for us to rotate the zero degreed medical images. Then, we rotated those images to 90, 180 or 270 degrees. After the image rotation, we divided the rotated angles by 90 to convert these angles to more discrete values. With the data augmentation technique, we were able to obtain 4,933 more images which helped increase the accuracy of deep learning models.

This dataset was mainly used in the pre-trained models as they required more data due to the nature of model complexity.

2.2.5 Train Test Split

The goal of machine learning is to better generalize unseen data. In order to achieve this goal, we divided the dataset into three different parts: training data, validation data, test data. The training data is used for the models to update weights and biases based on the training. The validation data works as the new dataset, and we usually test our trained model on the validation set to detect overfitting or under-fitting of the model. When testing the results on the validation set, it is important to freeze the training process and use learned parameters.

In this project, 70% of the data was used for training. For validation and test split, we utilized different approaches for the original dataset and dataset with data augmentation. Since the original dataset does not have sufficient data after the training split, we used 60% of the test data as validation and the rest as test data. For the dataset with data augmentation, we were able to split 50/50 for the validation and test dataset since we had enough samples. Table 2 represents the number of images in each dataset.

Methods	Training Counts	Validation Counts	Test Counts
Original Dataset	2,308	594	396
With Data Augmentation	5,761	1,235	1,235

Table 2: Table of Data Counts in Train/Valid/Test Counts for Different Methods

3. Model Development

The model development occurs after the data preprocessing and feature engineering. This is a step where we decide which models to choose for further developments based on the results we get from individual models. In order to choose the best model, we tested various models in this stage.

For the baseline models, we developed basic models such as MLP and CNN. We tested both dataset methods (Table 2) and the Data Method 1 provided better results for these models. After achieving accuracy of 90%, we proceeded the development step with more advanced architectures such as GoogLeNet or EfficientNet. We tested various pre-trained models in this process, however, those two models tended to have the best results with regards to both the training time and model size. Figure 5 represents the model performances regarding respective model sizes and training time among various models.

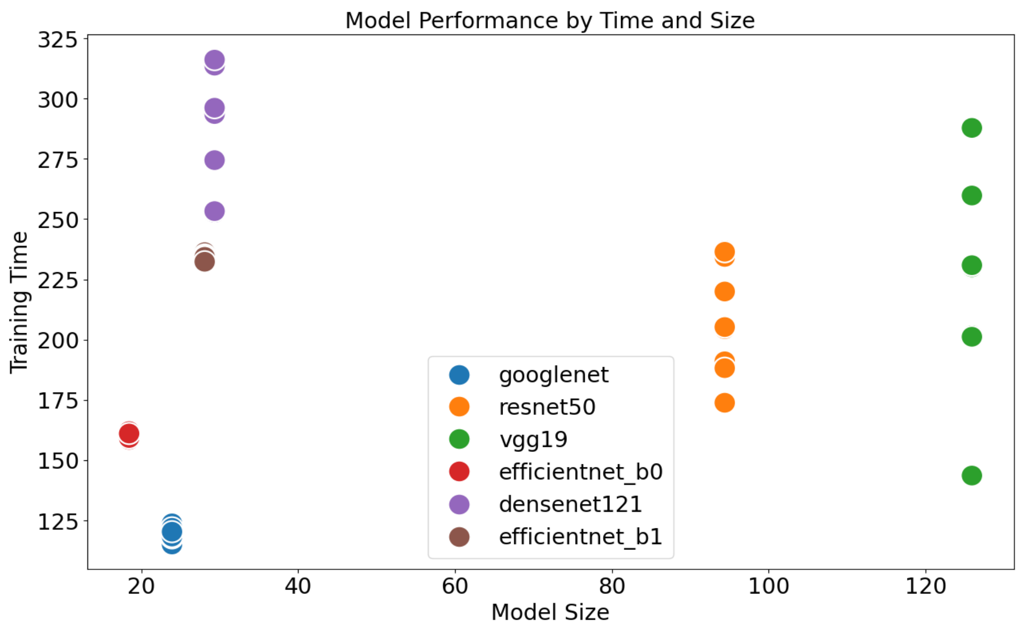


Figure 5: Pre-Trained Models and Their Performances

We attempt to find a that requires less memory and training time so that we can deploy the best model to software. For testing their performances, the same configurations such as batch size and learning rates were used to properly elect the best performing model. Table 3 describes all the parameters that we used via the training process for these models.

With regards to accuracy and loss, all these six models tended to have similar results. However, both ResNet50 and VGG-19 appears to have larger sizes than other models. VGG-19 appears to be extremely inconsistent in training as well. Although DenseNet121 and EfficientNet-b1 appear to consume less memory compared to ResNet50 and VGG-19, they generally tend to have a longer training time. Therefore, for the pre-trained model, we decided to experiment with GoogLeNet and EfficientNet_b0 for possible solutions for future deployment. In this section, we will discover the testing process for individual models and their architectures.

Model Parameters	**Values**
Batch Size	64
Learning Rates	0.00001
Optimizer	Adam
Epochs	15
Training Times	10

Table 3: Model Parameters for Pre-Trained Models for Testing Model Performances in Training Time and the Size of the Model

3.1 Baseline Models

Multi-layer perceptron (MLP) models are the foundation of the SOTA models. These models consist of three different layers: input layers, hidden layers and output layers. When inputs are fed into the input layers, weighted sums are calculated and passed to activation functions. Then the output layers attempt to predict the results based on the values from the previous hidden layers. These predicted values from the output layer will be compared to the actual label that we have in the training dataset with the loss function. The loss functions will be optimized with proper optimizers that were selected in the training process. The goal of the training purpose is to minimize the loss functions, so the training process will be iterated until the loss function will reach to the minimum. Figure 6 describes generic structures of MLP as well as its training process.

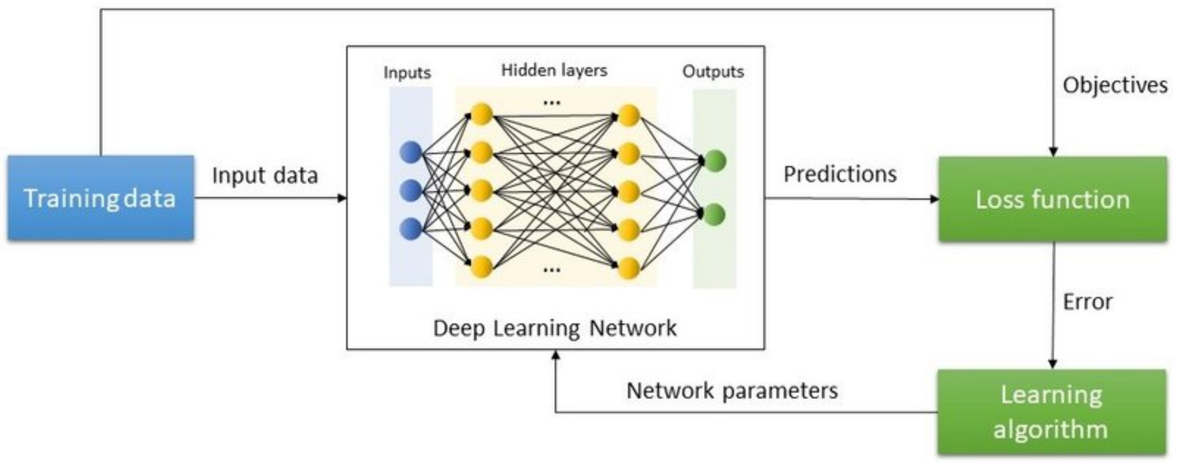


Figure 6: Training Process of Deep Learning

Source: "A Review on Deep Learning in Minimally Invasive Surgery," Rivas-Bianco & et al. 2021, P.3

Although the backbone of the models is the same including the number of layers and neurons, we developed two different MLP models. Both models contain seven layers; one input layer, five hidden layers and one output layer. For the classification layer (output layer), it is important to match the output features as the number of classes in our dataset. After each hidden layer, Rectified Linear Units (ReLU) activation functions are utilized to introduce non-linearity and the output layer utilizes the Softmax activation function to classify the results to different classes. The mathematical formula for ReLU is defined as below.

$$Relu(x) = max(0, x)$$

Formula 1: ReLU Formula

As the formula is shown, the ReLU activation often converges quicker and more efficiently than other activation functions such as Sigmoid since all negative values will be converted to zero. It also helps solve the vanishing gradient descent issue, which happens due to the optimization process of deep learning and its complex architecture. However, if many activations are below zero, the ReLU will provide zero, which might not be beneficial for models to learn from the dataset. Even though there are some disadvantages of utilizing ReLU, this activation function is still widely used and performs extremely well, therefore, we decided to use ReLU as activation functions.

For the output layer, different activation functions should be used based on use cases. When classifying two different labels (Binary Classification), the Sigmoid function can be utilized because this function will provide the results between zero and one, which will be useful for probability. However, when we attempt to classify more than three labels (Multi-class Classification), the output values for the Sigmoid function will be impractical since they will not add up to one, which will be difficult to be considered as probabilities. For multi-class classification, we then import the Softmax activation function, which is described below.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1,2,\dots,K$$

Formula 2: Softmax Activation

After completing the activation functions, the weights are initialized, and the loss will be calculated based on the predicted values and actual labels. For this step, we utilized the Adam optimizer, which is a combination of Momentum and Root Mean Square Propagation (RMSProp), since it takes less tuning requirement, consumes less memory and quicker running time compared to other optimizers. Setting proper learning rates for the optimizer is a crucial part in training deep learning models, therefore, we tried various learning rates with the Adam optimizer. We will discover the high-level overview of the model architectures for the MLP models, which utilize the techniques listed above.

The first model architecture did not utilize any regularization methods, while the other model architecture utilized dropout and batch normalization. “Each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate. Random “dropout” gives big improvements on many benchmark tasks and sets new records for speech and object recognition.” (Hinton & et al, 2012) Dropout randomly drops neurons in different layers after specified probabilities, which prevents models utilizing all neurons in the architecture. “Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout.” (Ioffe & Szegedy, 2015) Batch normalization helps normalize the output from the previous layer so that the next layer can receive normalized values. With batch-normalization, as the quote suggested, higher learning rates can be utilized. This can be beneficial in terms of model training because smaller learning rates will consume extensive time and resources which there might be limitations. These two techniques prevent overfitting and help increase the accuracy during the training process as shown in Figure 7 and 8. Figures 9 and 10 describe the model architecture for each model.

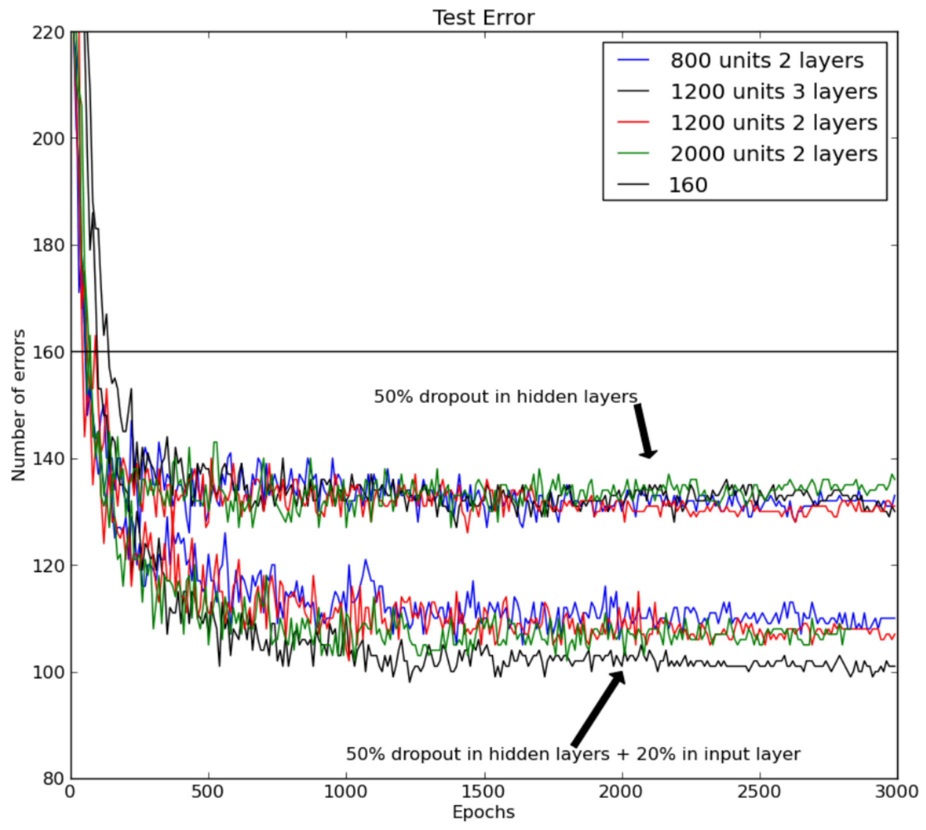


Figure 7: Error Rates for MNIST Dataset for Various MLP Architectures

Source: “Improving neural networks by preventing co-adaptation of feature detectors,” Hinton & et al. 2012, P.3

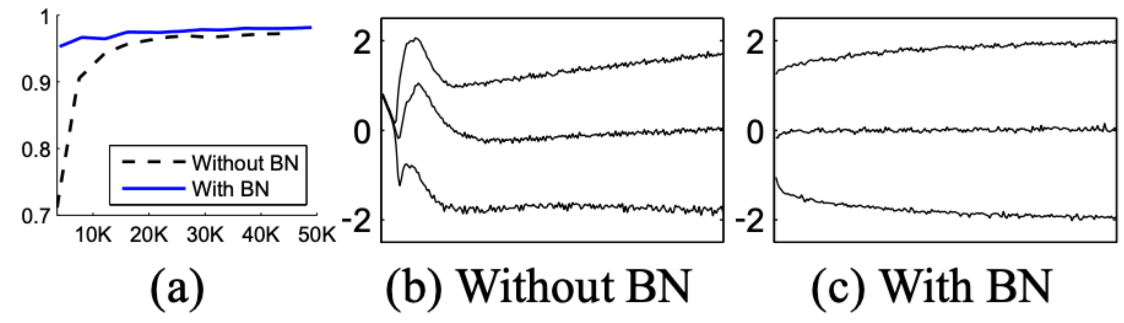
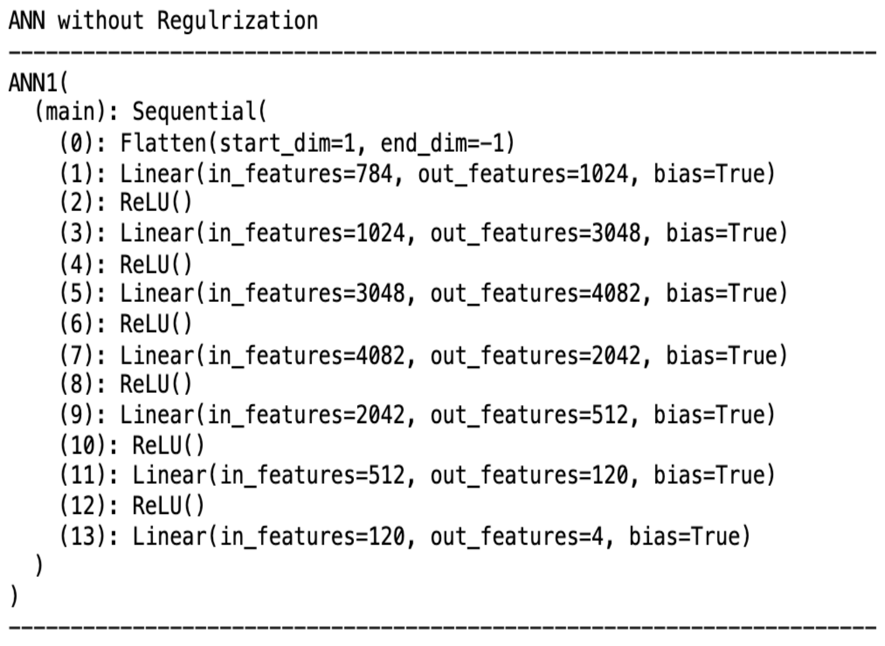


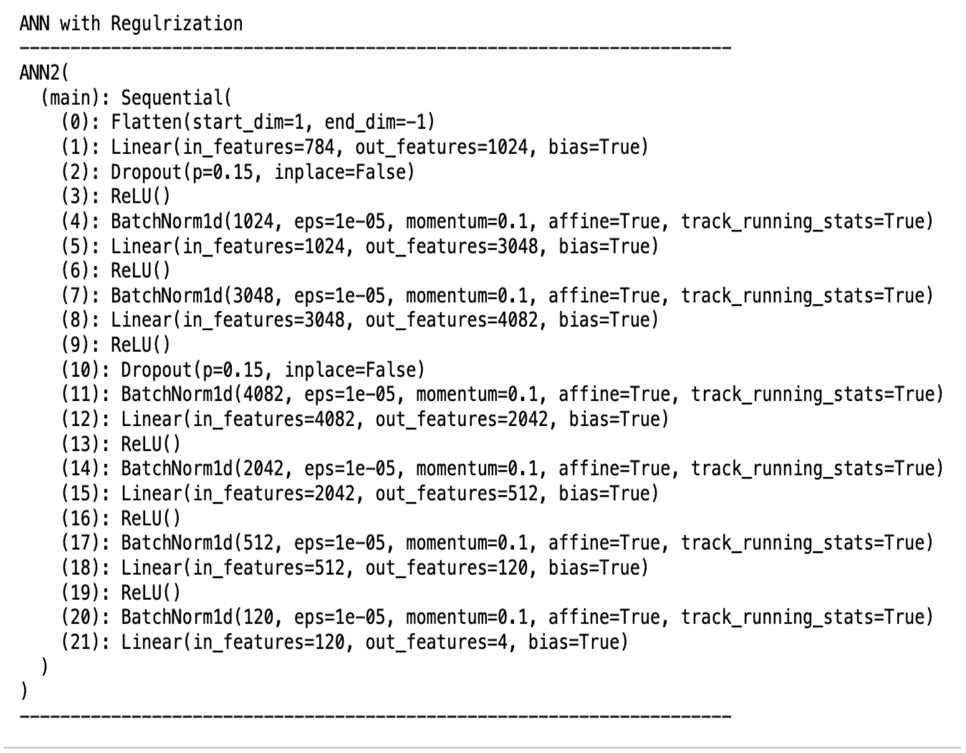
Figure 8: Test Accuracy on MNIST Dataset and Training Steps with Batch Normalization

Source: “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Ioffe & Szegedy. 2015, P.5



```
ANN without Regulrization
----------------------------------------------------------------------
ANN1(
  (main): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=1024, bias=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=3048, bias=True)
    (4): ReLU()
    (5): Linear(in_features=3048, out_features=4082, bias=True)
    (6): ReLU()
    (7): Linear(in_features=4082, out_features=2042, bias=True)
    (8): ReLU()
    (9): Linear(in_features=2042, out_features=512, bias=True)
    (10): ReLU()
    (11): Linear(in_features=512, out_features=120, bias=True)
    (12): ReLU()
    (13): Linear(in_features=120, out_features=4, bias=True)
  )
)
----------------------------------------------------------------------
```

Figure 9: MLP Model Architecture without Regularization Methods.



```
ANN with Regulrization
----------------------------------------------------------------------
ANN2(
  (main): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=1024, bias=True)
    (2): Dropout(p=0.15, inplace=False)
    (3): ReLU()
    (4): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): Linear(in_features=1024, out_features=3048, bias=True)
    (6): ReLU()
    (7): BatchNorm1d(3048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): Linear(in_features=3048, out_features=4082, bias=True)
    (9): ReLU()
    (10): Dropout(p=0.15, inplace=False)
    (11): BatchNorm1d(4082, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): Linear(in_features=4082, out_features=2042, bias=True)
    (13): ReLU()
    (14): BatchNorm1d(2042, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Linear(in_features=2042, out_features=512, bias=True)
    (16): ReLU()
    (17): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): Linear(in_features=512, out_features=120, bias=True)
    (19): ReLU()
    (20): BatchNorm1d(120, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (21): Linear(in_features=120, out_features=4, bias=True)
  )
)
----------------------------------------------------------------------
```

Figure 10: MLP Model Architecture with Regularization Methods.

Although MLP can provide us adequate results, previous papers and research have proven that CNN architectures tend to outperform MLP in the Computer Vision field. The CNN architecture assumes that pixels will be similar if they are next to each other which is referred to as locality. There are two major parts in the CNN architectures, feature extraction and classification. Within feature extraction, kernel, which performs as a sliding window, scans the inputs to extract useful information and pooling layers will be applied to lower dimension of the input data. After the feature extraction, the MLP architecture is applied so that the model can classify different images to individual classes. Figure 11 represents the training process of the CNN architecture.

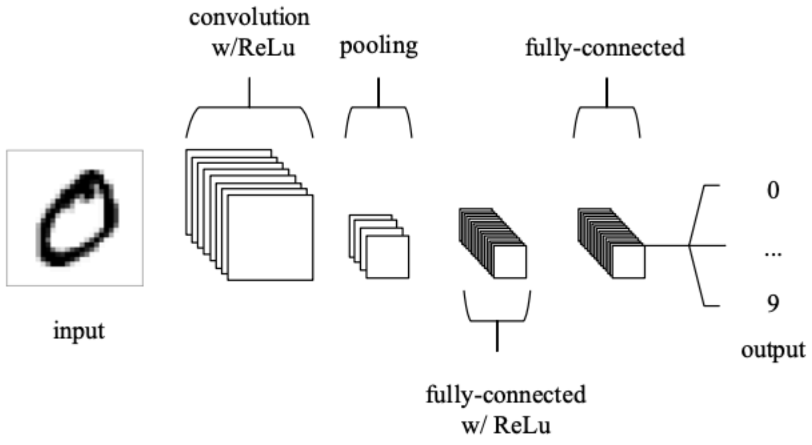
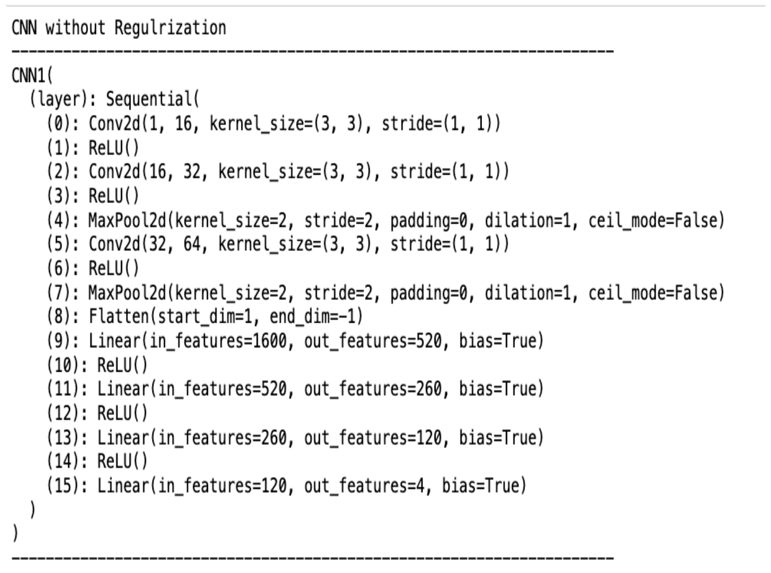


Figure 11: Training Process of the CNN Architecture

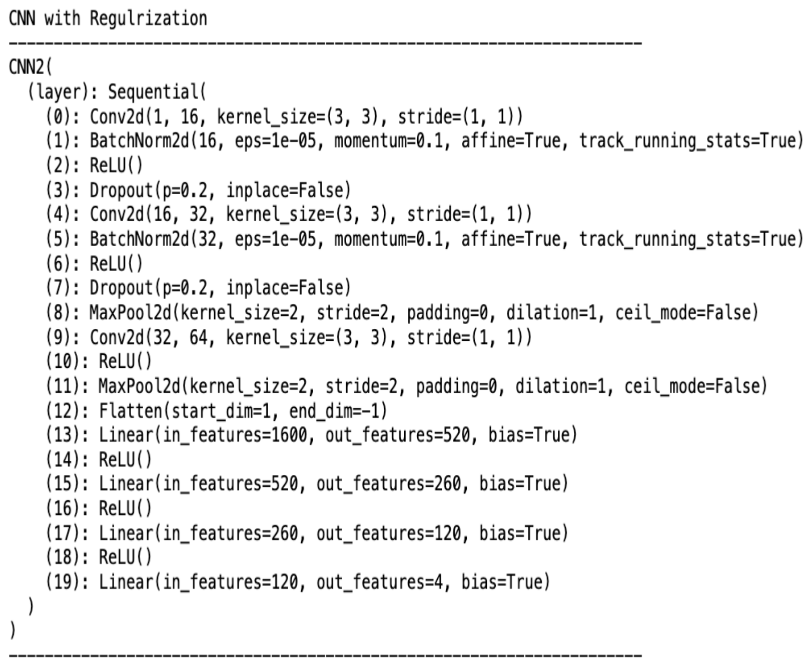
Source: “An Introduction to Convolutional Neural Networks,” O’Shea & Nash. 2015, P.4

Similar to the MLP architectures, we developed two CNN based models. The first CNN architecture does not utilize any regularization methods, while the second CNN architecture, which is built on top of the first CNN architecture, utilizes batch normalization and drop out to ensure that the models will not overfit during the training process. One of the distinctions from CNN to MLP is that the CNN architecture perceives the input channel, therefore, developers need to specify the number of channels when importing the input features. Figures 12 and 13 describe the architectures of these two models.



```
CNN without Regulrization
----------------------------------------------------------------------
CNN1(
  (layer): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=1600, out_features=520, bias=True)
    (10): ReLU()
    (11): Linear(in_features=520, out_features=260, bias=True)
    (12): ReLU()
    (13): Linear(in_features=260, out_features=120, bias=True)
    (14): ReLU()
    (15): Linear(in_features=120, out_features=4, bias=True)
  )
)
----------------------------------------------------------------------
```

Figure 12: CNN Model Architecture without Regularization Methods.



```
CNN with Regulrization
----------------------------------------------------------------------
CNN2(
  (layer): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.2, inplace=False)
    (4): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): Dropout(p=0.2, inplace=False)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Flatten(start_dim=1, end_dim=-1)
    (13): Linear(in_features=1600, out_features=520, bias=True)
    (14): ReLU()
    (15): Linear(in_features=520, out_features=260, bias=True)
    (16): ReLU()
    (17): Linear(in_features=260, out_features=120, bias=True)
    (18): ReLU()
    (19): Linear(in_features=120, out_features=4, bias=True)
  )
)
----------------------------------------------------------------------
```

Figure 13: CNN Model Architecture with Regularization Methods.

After creating model architectures, we attempted to find a model that had the highest performance. Using the same parameters in Table 3, Figure 14 describes the model performance with regards to accuracy. The X axis represents epochs, and the Y axis shows the accuracy for different models. In general, the accuracy of models is subject to increase as the number of epochs increases. We also tried both first and second data methods. The model architectures for each data method needed to be modified since input channels are important in terms of training the CNN based models. After checking the results, we validated that simpler models (MLP or CNN models) tended to perform better with the simpler method (data method 1). For the base line models, it was relatively fast to train the model since the data size was small and the model architecture was not complex. Thus, computation for updating parameters was relatively low compared to pre-trained models.

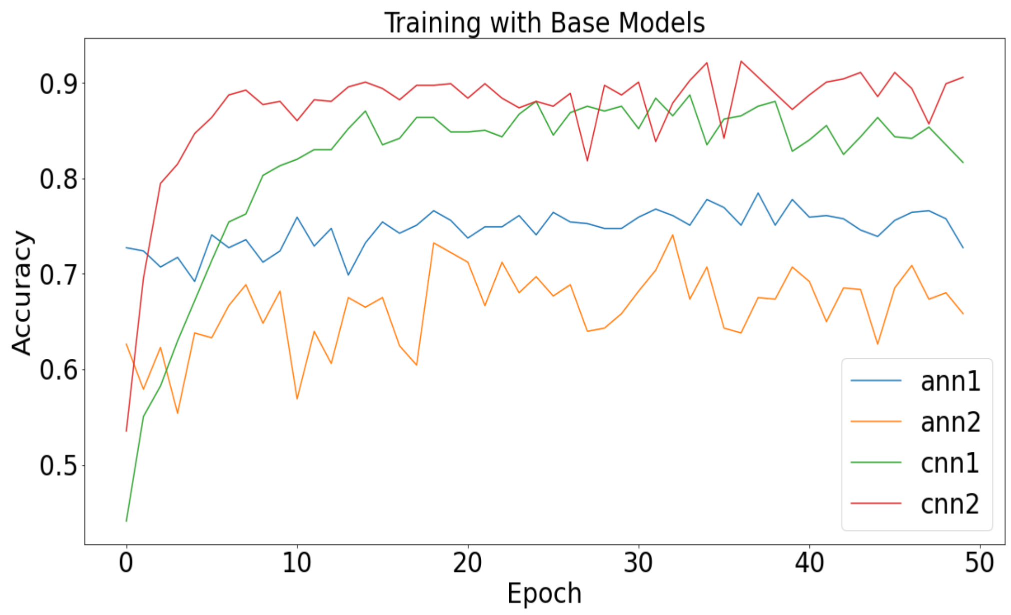


Figure 14: Training Base Models with Batch Size 64 and Input Channel One.

Based on the results above, it is evident that the second model in the CNN architecture outperformed other models. We proceeded with this model by changing different parameters such as learning rates and batch size. Figure 15 describes various learning rates from the second CNN model and their accuracy. It appeared that 0.001 as the learning rates provided the best results for our model. “While the use of large mini-batches increases the available computational parallelism, small batch training has been shown to provide improved generalization performance and allows a significantly smaller memory footprint, which might also be exploited to improve machine throughput.” (Masters & Luschi, 2018) In order for us to find an optimal batch size, we decided to test various mini batches and monitor their performance. We attempted to set a higher number than 200 for testing the optimal batch size, however, GPU did not have sufficient computation power. Table 4 shows different batch sizes and their performances. Batch size did not seem to play a critical role as learning rates but setting batch size too low would impair the model performance.

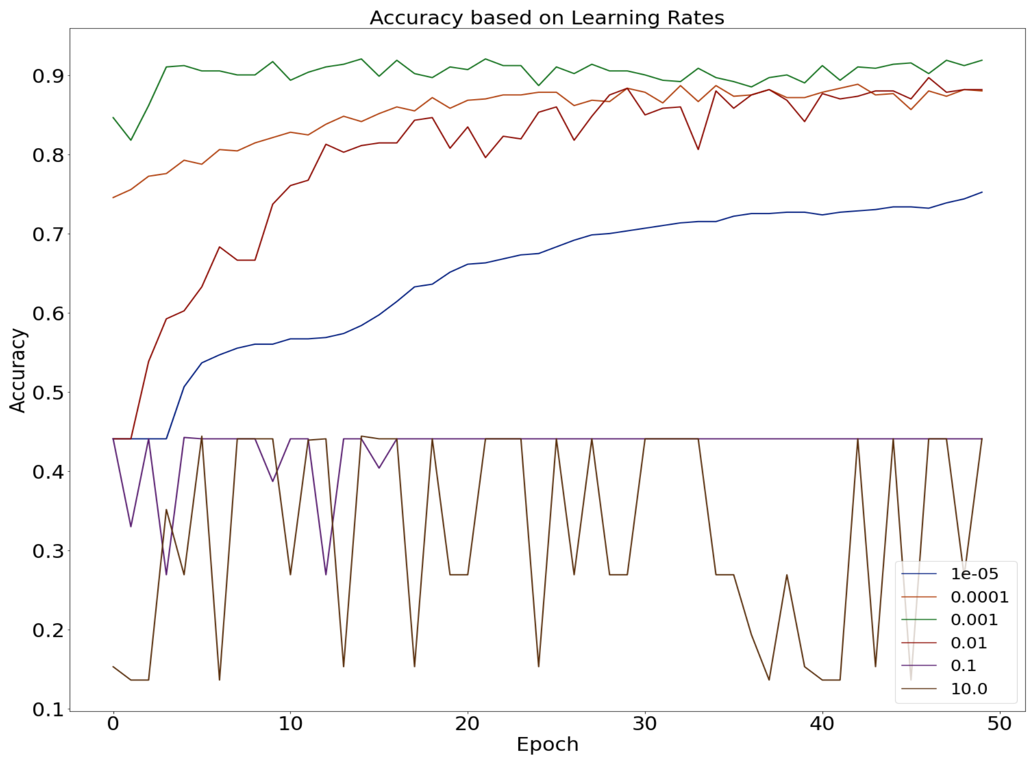


Figure 15: Different Learning Rates and Accuracy Using the Second CNN Model Architecture

Batch Size	Mean	Std Dev	25%	50%	75%	Max
16	0.863165	0.034332	0.855219	0.865320	0.886785	0.914141
32	0.879158	0.023345	0.870370	0.880471	0.893939	0.920875
64	0.906094	0.011610	0.897727	0.905724	0.914141	0.934343
100	0.896768	0.018187	0.890993	0.901515	0.908670	0.924242
128	0.903367	0.014849	0.897727	0.905724	0.912458	0.927609
200	0.914781	0.007566	0.910774	0.914141	0.919192	0.934343

Table 4: Table of Different Batch Sizes for the Second CNN Architecture and Summary Statistics of Accuracy

Table 4 describes performance based on different batch sizes for the CNN architecture with regularization. Each model was trained with different batch sizes for 50 epochs to represent reasonable results. In terms of increasing the performance of the model, learning rates are the most important factor. However, as the table describes, there are some benefits of using different batch sizes. Mini batch with 200 seems to provide the best results from both the average and max accuracy perspective. It also appears that the mini batch 200 has the least variation compared to other batch sizes. For the final base model, we decided to utilize the second CNN architecture with batch size as 200, learning rates as 0.001 with Adam optimizer.

Although the second CNN architecture provided a reasonable performance with 0.93 accuracy, the production model needs to be more accurate. In order to achieve this goal, we decided to test pre-trained models. These models are trained on a larger dataset which can be fine-tuned for specific use cases. As Figure 5 presents, we developed a few different pre-trained models under Pytorch libraries. Considering the performance benefits from both memory and training size, we decided to fine-tune GoogLeNet and EfficientNet for the production model. In the next section, we will discover the idea behind these models and their architectures.

3.2 Pre-Trained models

GoogLeNet was invented in 2014, which won the ILSVRC 2014 competition. The idea of this model was to overcome computational bottlenecks using an Inception architecture. “The Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components.” (Szegedy & et al, 2014) When dealing with multi-channel computer vision problems, the number of parameters increases dramatically. This leads to not only an issue with over fitting the computer vision model, but also with an inefficient usage of computation power.

In order to overcome these issues, the Inception architecture utilizes 1 by 1 convolutional layers which perform as dimension reductions. Utilizing these 1 by 1 convolution layers allows the model to minimize the usage of previous layers that have zero weights which will not be used in the backpropagation step. In addition to this, the model utilizes dropout before the fully connected layer to prevent the model from overfitting. The plot below describes the Inception modules in a nutshell (Figure 16).

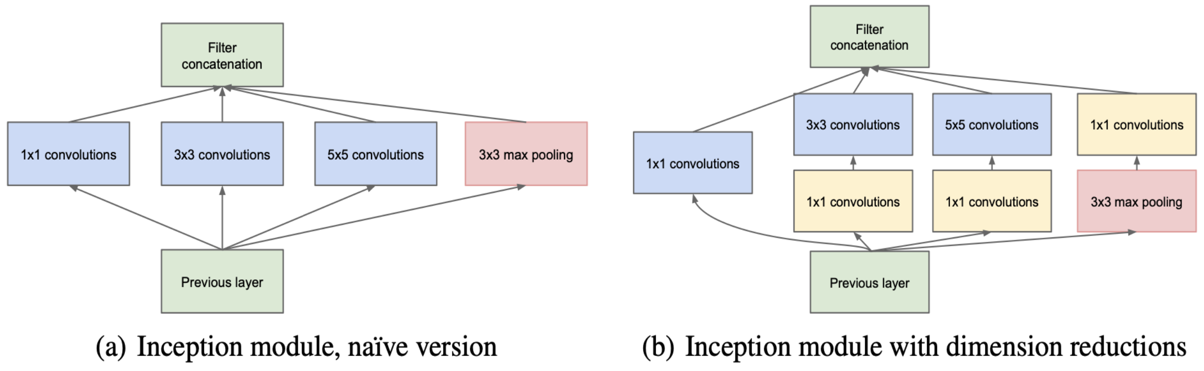


Figure 16: Inception Module in a Nutshell

Source: “Going Deeper with Convolutions,” Szegedy & et al. 2014, P.5

EfficientNet is a relatively new architecture compared to GoogLeNet. This model architecture won the ILSVRC 2019 competition. As the name represents, the model tries to achieve better accuracy and efficiency by utilizing compound model scaling. Previously, other scaling methods focused on width and depth sides, while EfficientNet was the first architecture to consider the channel side so that larger models can scale up network width, depth and resolutions which lead to better model performances.

Training for EfficinetNet required a longer time frame than GoogLeNet, but the model size was smaller. It also appears that EfficientNet seemed to be more consistent and have less variance in terms of training. We chose both GoogLeNet and EfficientNet were selected for hyper parameter tuning because these models exceled others in the experimental stage (Figure 5). The Optuna library was utilized to optimize the number of layers and neurons in the fully connected layers along with optimizers, learning rates and batch size. For both models, the range for the learning rates was between 0.000001 and 0.1. Batch size was picked between 64 and 200. For the optimizer, we selected four different optimizers: Adam, RMSprop, Adagrad and SGD. For the fully connected layers, we choose integer values from 500 to 2,400 for the L1 and between 300 and 2,400 for the L2 layers. Although we can choose the number of layers as a hyper parameter tuning step, we decided to stick with 2 layers since these models provided outstanding results even before the hyper parameter tuning step. Below table shows the best hyper parameter for each model (Table 5).

Model	Best Parameter	Best Value
GoogLeNet	L1	682
	L2	1,701
	Optimizer	Adam
	Batch Size	200
	Learning Rates	0.0001959
EfficientNet	L1	1,328
	L2	318
	Optimizer	RMSprop
	Batch Size	74
	Learning Rates	0.0005046

Table 5: Best Performing Parameters for GoogLeNet and EfficientNet

4. Model Evaluation

Evaluation is a critical step for the machine learning cycle since the goal of machine learning is to generalize unseen data well. We began this stage after we fine-tuned the model architectures of the pre-trained models and performed hyper parameter tunings to get the best results. The previous steps utilized train and validation sets and this step utilized test data, which acted as unseen data. Both GoogLeNet and EfficientNet were utilized based on the results from Figure 5 to test their performance on the test data. Figure 17 presents the best GoogLeNet model with optimal parameters and figure 18 shows the best EfficientNet model. Table 6 describes the best performance for each model with the hyper parameters tuned.

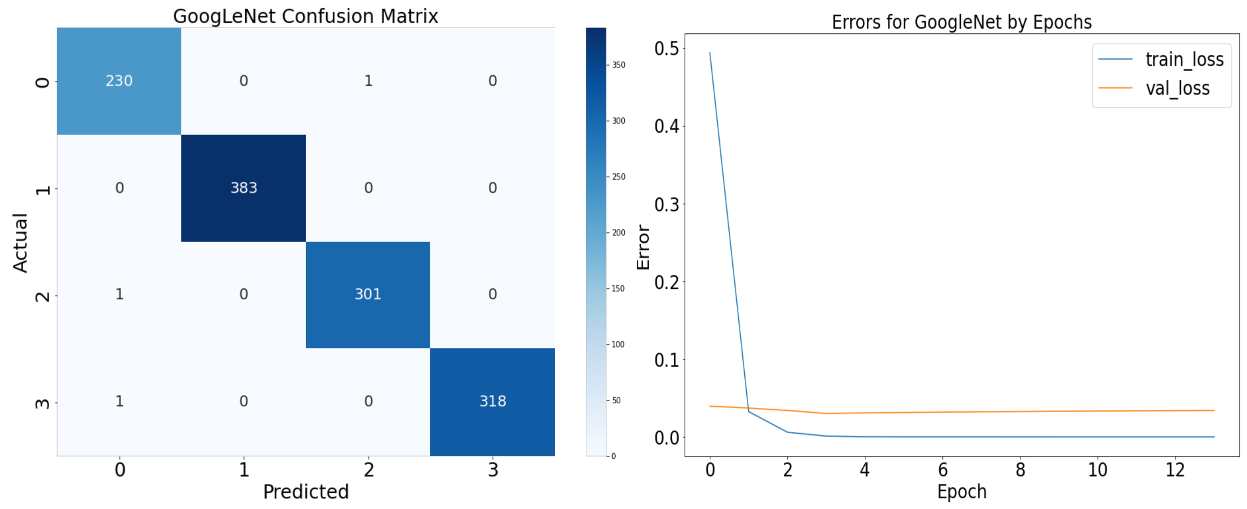


Figure 17: Confusion Matrix for GoogLeNet (left), and Training and Validation Loss (right)

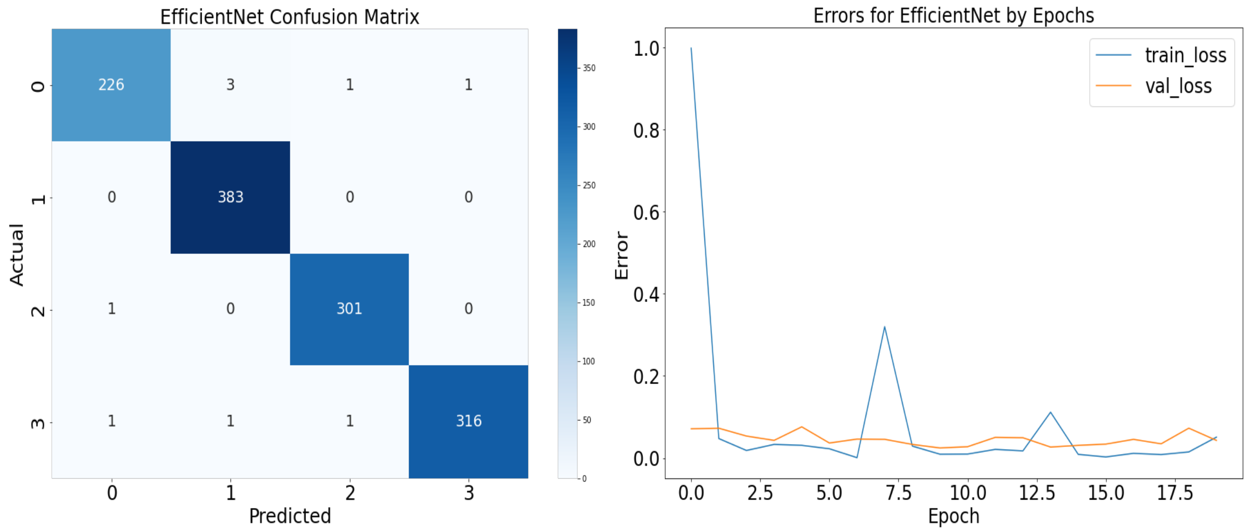


Figure 18: Confusion Matrix for EfficientNet (left), and Training and Validation Loss (right)

Model	Data	Accuracy	Recall	Precision	F1 Score
GoogLeNet	Training	1.0	1.0	1.0	1.0
	Validation	0.993522	0.993522	0.993535	0.993522
	Testing	0.997571	0.997571	0.997578	0.997573
EfficientNet	Training	0.996528	0.996528	0.996541	0.996525
	Validation	0.994332	0.994332	0.994336	0.994330
	Testing	0.992713	0.992713	0.992725	0.992699

Table 6: Model Performances for GoogLeNet and EfficientNet

Both confusion matrix and error plots are great indicators for developers to better understand the performance of machine learning models. The confusion matrix presents both actual values and predicted output from the given machine learning models. Confusion matrix is essential for working with classification models since it is not only verifying the accuracy of the models, but also visualizing type one (false positive) and type two (false negative) errors, so that developers can modify the threshold which will improve the performance of the models. All the values within the plot should sum up to the total number of rows in the testing dataset. The horizontal values indicate the actual number of images in each class. For example, based on the performance of GoogLeNet, there are 231 medical input images that have zero angle. Out of these 231 inputs, the model identified the correct orientation of 230 medical inputs.

Although both models performed excellently in all data sources, GoogLeNet appeared to outperform EfficientNet in our use case. The confusion matrix proved that GoogLeNet correctly identified 1,232 out of 1,235 inputs while EfficientNet identified 1,226 images correctly. The loss plot also suggested that the loss for GoogLeNet was steady and lower compared to EfficientNet. These metrics are useful to analyze the initial performance of each model. For classification problems, it is extremely crucial to have proper metrics due to data imbalance. Certain fields including medical tend to have higher chances of having imbalance data because of the nature of the fields. The percentage of having unique diseases will be extremely rare, which will lead to an extreme data imbalance issue. In the case of imbalance data, accuracy will not be an ideal indicator. Accuracy does not take the quality of the model into account since it ignores type one and two errors. Thus, it is more beneficial to examine other metrics such as recall, precision and F1 score. Recall and F1 score for GoogLeNet seemed to surpass

EfficientNet's performance. Therefore, with all this supporting evidence, GoogLeNet was chosen for the final development.

5. Conclusion

Increasingly, many individuals make furry friends a part of their families but living with these companions comes with many responsibilities. Unlike human beings, canine/feline patients are unable to communicate their symptoms and hide their pain more effectively. Experienced individuals who are educated in vet diseases might be able to catch early symptoms, but it is often hard to find internal issues. Technology has increasingly improved catching early signs and administering treatment and as technology evolves these symptoms can be addressed at a much quicker pace. With the help from deep learning models such as Computer Vision, we hope to identify canine/feline patients' internal issues as early as possible so that they can live happier and healthier lives. Preventative care is crucial for living longer healthier lives for both humans and pets, so identifying potential concerns early can save the lives of our furry companions.

6. References

[1] Kang, D.-Y., Duong, H. P., & Park, J.-C. (2020). Application of deep learning in dentistry and Implantology. *The Korean Academy of Oral and Maxillofacial Implantology*, *24*(3), 148–181. https://doi.org/10.32542/implantology.202015

[2] Perez, L., & Wang, J. (2017, December 13). *The effectiveness of data augmentation in image classification using Deep Learning*. arXiv.org. https://arxiv.org/abs/1712.04621

[3] Sabottke, C. F., & Spieler, B. M. (2020). The effect of image resolution on Deep Learning in radiography. *Radiology: Artificial Intelligence*, *2*(1). https://doi.org/10.1148/ryai.2019190015

[4] Rivas-Blanco, I., Perez-Del-Pulgar, C. J., Garcia-Morales, I., & Munoz, V. F. (2021). A review on Deep Learning in minimally invasive surgery. *IEEE Access*, *9*, 48658–48678. https://doi.org/10.1109/access.2021.3068852

[5] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012, July 3). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv.org. https://arxiv.org/abs/1207.0580

[6] Ioffe, S., & Szegedy, C. (2015, March 2). *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv.org. https://arxiv.org/abs/1502.03167

[7] O'Shea, K., & Nash, R. (2015, December 2). *An introduction to Convolutional Neural Networks*. arXiv.org. https://arxiv.org/abs/1511.08458

[8] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2015.7298594

[9] Masters, D., & Luschi, C. (2018, April 20). *Revisiting small batch training for Deep Neural Networks*. arXiv.org. https://arxiv.org/abs/1804.07612

[10] Tan, M., & Le, Q. V. (2020, September 11). *EfficientNet: Rethinking model scaling for Convolutional Neural Networks*. arXiv.org. https://arxiv.org/abs/1905.11946