UNIVERSITY
OF GÄVLE

FACULTY OF ENGINEERING AND SUSTAINABLE DEVELOPMENT
Department of Computer and Geospatial Sciences

Additional Classes Effect on Model Accuracy using Transfer Learning

Baran Kazan

2020

Degree project, Basic level (Bachelor degree), 15 HE
Computer Science
Study Programme in Computer Engineering

Supervisor: Anders Jackson
Examiner: Goran Milutinovic

Abstract

This empirical research study discusses how much the model's accuracy changes when adding a new image class by using a pre-trained model with the same labels and measuring the precision of the previous classes to observe the changes. The purpose is to determine if using transfer learning is beneficial for users that do not have enough data to train a model. The pre-trained model that was used to create a new model was the Inception V3. It has the same labels as the eight different classes that were used to train the model. To test this model, classes of wild and non-wild animals were taken as samples. The algorithm used to train the model was implemented in a single class programmed in Python programming language with PyTorch and TensorBoard library. The Tensorboard library was used to collect and represent the result. Research results showed that the accuracy of the first two classes was 94.96% in training and 97.07% in validation. When training the model with a total of eight classes, the accuracy was 91.89% in training and 95.40 in validation. The precision of both classes was detected at 100% when the model solely had cat and dog classes. After adding six additional classes in the model, the precision changed to 95.82% of the cats and 97.16% of the dogs.

Keywords — artificial intelligence, machine learning, PyTorch, transfer learning.

Table of Contents

1 Introduction

Machine learning grows more popular in different fields every day, but it has its challenges. Deep learning's popularity is growing more than ever in multiple different fields, such as healthcare, predicting earthquakes, language translations, election prediction, fraud detection, and self-driving cars [1]. Deep Learning resolves human problems and predicts solutions with high accuracy. This popularity is due to deep learning, and providing outputs with training models made up of input data.

Having an accurate model requires a large amount of data to train a new model. Getting data to train a model could be difficult, especially for users that do not have enough resources. It would put them at a disadvantage of not having a model with the minimum accuracy they required. The context of machine learning is to train the weights of the neural networks so that the model can make the correct output out of inputted data [2]. Transfer Learning is a machine learning technique that uses a pre-trained model to transfer its weight into a newly created model [3]. The user can select and download a pre-trained model, with the same or familiar labels to reach the minimum required accuracy, without any additional data to train a model.

1.1 Purpose

The purpose of the research study is to measure how effective it is to use transfer learning to train an imagery model with insufficient data from scratch.

1.2 Goal

The goal is to have a pre-trained imagery model with the same label as the data set. The model will begin with two different classes, and it will iterate until it reaches a total of eight different classes. Before the next iteration, the accuracy of the model and precision of each class will be collected. The data can then be used to analyze the cause of precision or accuracy drop of the model. It will give a better understanding of how beneficial it is to use transfer learning when it comes to imagery prediction, such as the accuracy and precision changes when adding a new class with the same label as the pre-trained model.

1.3 Delimitations

The model is designed to recognize and predict labels only using images. The image file formats that support PyTorch are: .jpg, .jpeg, .png, .ppm, .bmp, .pgm and .tif [4]. The images need to have at least 299x299 resolution to use the Inception V3 (pre-trained model) [5] to train and validate the model. It is possible to use a different pre-trained model that has different labels. The pre-trained model needs to have the same labels as the data set used to train and validate the model.

2 Background

Deep learning uses neural networks to train a model. It requires millions of parameters and data sets to train a model to reach accurate outputs [6]. Models need to recognize the pattern and relations along with the specifics of our problem. The more data there is to train and validate the model, the better accuracy can be expected. However, too much data could cause overfitting.

Google developed a model that identifies 1.28 million images from 1,000 object categories [7]. Complexity in image classification is so powerful that it requires a high number of parameters and a large data set for the training. Those are the challenges of deep learning.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is related to giving machine capability to mimic human behavior, primarily cognitive functions, so that it can learn from the examples in the data set and solve the obstacles [1], [8]. Today AI is being used in many sensitive fields such as healthcare, predicting earthquakes, language translations, election prediction, fraud detection, and self-driving cars [1], [9] to decrease human intervention. AI uses machine learning and deep learning algorithms [8]. There are different AI types, such as Artificial Narrow Intelligence (ANI) and Artificial General Intelligence (AGI) [10]. The ANI is an AI system designed and trained to complete a specific task, while the AGI can replicate a human brain's cognitive abilities [10]. When the AGI is presented with an unfamiliar task, it can use the function to apply knowledge from one domain to another and solve the obstacle autonomously [10]. There are different forms of AI used every day, even though most people are not aware of them. If AI is going to perform a task based on the input data with a neural network model, it is crucial to train the model beforehand [11].

2.2 Machine Learning

Machine learning (ML) is the part of AI that allows the machine to learn and improve from the experience. It allows the computer to learn from the data set without user interaction and can adjust its actions based on the input data [12]. ML is a part of AI that automatically builds a model from the input data and parameters [13]. ML trains a model that can identify specific patterns from the input data set. Today ML learns from previous models and produces more reliable decisions. It is applied in many fields such as self-driving Google cars, robots, facial recognition, traffic recognition of online transportation networks, medical diagnosis, and online fraud detection [1], [8]. It is essential to train the model with training and validation data to make better and more accurate decisions. The ML uses algorithms to analyze the data, identify patterns, and make decisions [2].

There are different types of ML, such as supervised, semi-supervised, and unsupervised learning. Supervised learning is a learning function that maps an input to an output based on an example of input-output pairs [14]. Unsupervised learning is another learning function used to draw inferences from a data set consisting of only inputted data without labeled response [14]. Semi-supervised learning is a combination of supervised and unsupervised learning methods. It uses an algorithm that learns from a data set that includes both labeled and unlabeled data [14].

2.2.1 Supervised Learning

Supervised learning is the most commonly used learning system for training a model [15]. The training model takes on a pair of data. The first is the featured data, and the second is the labeled data. The featured data is the input data used to train the model so that it can recognize the pattern it has and give the same output on a featured data that has a familiar pattern [15]. The labeled data tell the model what category the featured data belong in. It shows the model what output it should generate from the featured data [15]. When the training model is complete, the model will predict the output based on the input data [1]. Training a model with more data will lead to higher accuracy of prediction, which means generating a wrong output will be less over time [2].

2.2.2 Overfitting

Overfitting is a modeling error when a function is too closely fit for a limited set of data points [16]. The model will try to memorize the images instead of learning from it. From the graphical perspective, the loss value will be like a U shaped curve. The loss will have high value from the beginning, but over time it will decrease. Then, the value will start increasing again, and the loss value will worsen. It can happen in any layer in the neural network [16].

2.2.3 Transfer Learning

Transfer learning is a machine learning technique where a model is trained (pre-trained model) on one problem and then reused to solve a similar problem [17]. Millions of parameters and a large amount of labeled data are used to train a model that will lead us to reach the output. Instead of training the other neural network from scratch, we transfer the learned features [18], [19]. Training a model from scratch needs considerable resources. Not every company has these enormous resources, so transfer learning enables them to reuse a pre-trained model for other related tasks. The targeted output can be reached without using considerable resources [17]. If we were to build a self-learning car, it would take years to make a decent image recognition algorithm. Alternatively, there is a model from Google called the Inception model, which was trained on the ImageNet data set to identify objects in those images [3].

Different kinds of pre-trained models can be downloaded from the internet, giving the advantage of having a pre-trained model and its layer that best suits the desired model that can solve a similar problem. As previously mentioned with the Inception model, it has the ImageNet data set with 1,000 different classes with 1,281,167 training and 50,000 evaluation images [20]. There are different versions of Inception models because of the changes made from the layers in each version. The latest Inception model is the third version that has a total of 48 layers [21]. There are also other models to use than the Inception, such as the MobileNet model that has 53 layers with 1,000 different classes [22]. The difference between the MobileNet and Inception is that the MobileNet trains a model faster while Inception provides higher accuracy when training a model [23]. There are different models to choose from that fulfill different tasks. The user can choose the model depending on their necessities.

2.3 Artificial Neural Networks

Artificial Neural Networks, also called neural networks, are computational models [24]. They are brain-inspired systems replicating the way that we humans learn [24]. Like human beings, they learn from historical data or examples. Once neural networks are trained, they learn on their own by considering examples [25]. When the training is done, the user can use the backpropagation technique that allows networks to adjust their hidden layers of neurons if the outcome does not match what the designer was hoping, for instance, a network designed to recognize dogs but misidentifies a cat instead [24]. Neural networks are nonlinear statistical models that find relationships between input data and output data to discover new patterns. They can take sample data rather than the entire data set [26].

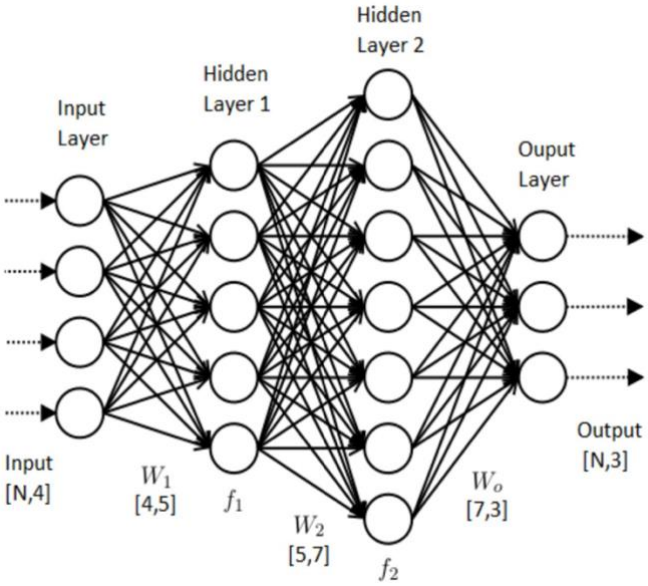

Figure 1. The layers and weights of the neural network [27]

Neural networks have input, hidden, and output layers, as represented in figure 1. The input layer is the first layer of a neural network, which receives input values in image pixels, numbers and audio files, etc. Hidden layers are the in-between layers of the model. There can be single or multiple layers that perform the mathematical computation on the input data and recognize the input data patterns. The output layer is the last. It receives input from the hidden layer and obtains the result [28].

Input, hidden, and output layers have a connection between each node. Each node has a set of inputs, weights, and biased values. Weight controls the strength of the signal between the two neurons, which decides how much influence the input will have on the output [29]. Bias is an additional parameter in the neural networks. It is used to adjust the output with the weighted sum of the inputs of the neuron [29]. The weight is represented as "W" in Figure 1 and shows the strength of the connection between the units. If the weight from node 1 to node 2 (W1) has a higher value, it means node 1 has a high effect over node 2 [28].

$$output = \sum(weight * inputs) + bias$$

The function calculates the output node that input data is multiplied by weights, then bias value is added to the result. The bias value allows the activation function to be shifted to the left or right, to fit the data better [30].

2.3.1 Convolutional Neural Networks

A convolutional neural network (CNN) is similar to artificial neural networks. It is an algorithm that can process the input image and learn the image's specific patterns through filters [31], [32]. This feature allows the model to differentiate one image from the other. For example, the convolutional neural network will learn specific cat's specific features, such as whiskers, pointy ears, and tails, and differentiate it from the dog (figure 2) [32]. CNN has proven very effective in image recognition and classification. Artificial neural networks do not scale the full image well. CNN's main advantage is its ability to pre-process the data by itself, which will prevent us from spending many resources in data pre-processing.

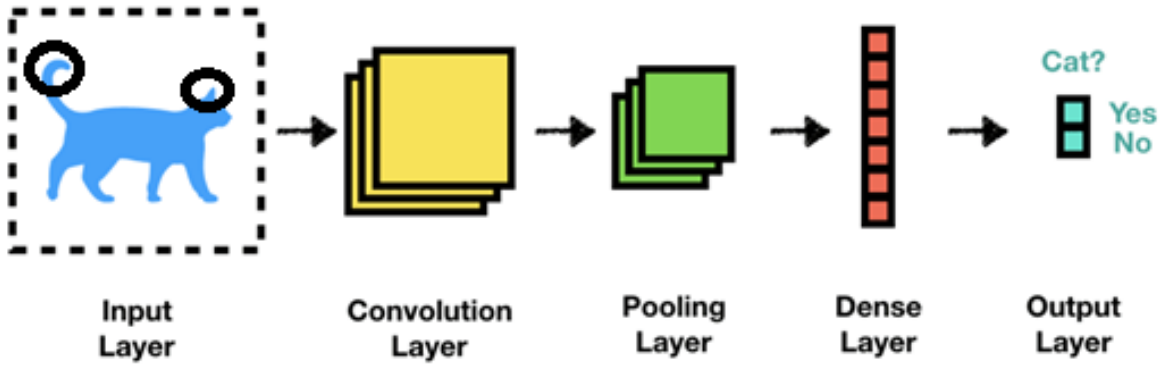

Figure 2. Convolutional neural network structure [33].

The images need to be converted into numerical data so they can be processed after inputting the data; the numerical value is between 0 and 255, which is RGB color model value [31]. Contrary to the linear arrangement of neurons in the artificial neural network model, neurons (nodes) in the CNN model have an overall structure of three dimensions (width, height, and depth). For example, 3x3x1 input data means an input image of width 3, height 3, and 1 color channel greyscale. 3x3x3 convolution filter represents an input image with 3, height 3, and three color channels (red, blue, and green). The convolutional layer is the most crucial in the CNN model because it has learnable filters, which are the network's weights. The CIFAR-10 data set is a collection of images most widely used to train machine learning algorithms. Images are of the size 32x32 and in color, so the first hidden layer of artificial neural networks would have 32*32*3=3072 weights. So, this fully-connected structure does not scale to larger images. This massive number of parameters and hyperparameters would lead to overfitting. In a convolutional neural network, each layer accepts 3D input images and then converts them to 3D output through the differentiable function [31].

3 Process

The model began with training two classes. When the model was in a training or validation set, it saved the accuracy and loss value in each iteration separately, depending on if the model was training or evaluating. The model was trained with ten epochs. It was the optimal value because the graph showed that the accuracy (both training and validation) and loss value made less difference after the seventh epoch [34]. Having a batch size of 32 in each iteration ensured that the graph does not have a sudden accuracy spike. It calculated the average accuracy of those 32 images. Having a smaller batch size value caused a sudden spike because fewer images were used to calculate the average accuracy. Saving the value in each iteration gave accuracy and loss value that could be represented in a graph. The Y-axis represented the accuracy or loss value, and the X-axis contained the number of iterations. The training continued until the graph showed that there was only a very slight difference in accuracy and loss value for each iteration. When the training was complete, the record of the model was saved. That record included data such as the graph of both training and validation for the accuracy and loss value, layers of the model, and precision-recall curve. The record was used to compare the accuracy of the previous model that had one less class.

The Inception v3 was created by using the ImageNet data set. The labels used to train the model were cats, dogs, wolves, beavers, weasels, bears, otters, buffalos, hippopotamuses, and chimpanzees. These labels also include in the ImageNet data set [35] to test the transfer. The weights of the pre-trained model (Inception v3) always have static value when training a new model. However, the value of the neurons from the model could differ each time when training the model.

3.1 Tools

3.1.1 Compute Unified Device Architecture (CUDA)

The training of neural networks will take a long time, especially if the data set is larger (approximately more than 25,000 images). The library, such as TensorFlow and PyTorch, uses the central processing unit (CPU) to do the processing as a default setting. Both of the open-source libraries have a function to run these processes on the graphics processing unit (GPU), which was used to train the models in this research. The GPU has many simple cores that allow parallel computing through thousands of cores computing at a time. The training time is shorter compared to running on a CPU [36]. The requirement of running on a GPU for both libraries is the CUDA [37], [38]. CUDA is an application programming interface (API) and a parallel computing platform created by Nvidia. It enables developers to speed up compute-intensive applications by using the GPU's power for the parallelizable part of the computation [39]. The CUDA installation requires Nvidia GPU that it is compatible with; not all of the Nvidia GPUs support CUDA, so it is essential to check compatibility [40], [41]. CUDA supports Windows, Linux, and Mac operating systems [42], [43].

3.1.2 Python programming language

The majority of data scientists and machine learning developers use Python as their preferred programming language [44]. Around 33% of developers prefer it over any other programming language, and professionals recommend it since the syntax is simple, making the language easy to learn [44]. Python is an object-oriented, interpreted, and high-level programming language with dynamic semantics [45]. It is a high-level built-in data structure combined with a dynamic typing and binding that makes it attractive for programmers that develops an application rapidly [45]. It has no compilation step, and the debugging is easy because a bug or wrong input will never cause a segmentation fault [45]. It will instead raise an exception, or if the exception does not exist, the interpreter prints a stack trace instead [45]. Python is the primary choice not only for the language but also for the wide variety of other open-source libraries such as TensorFlow and PyTorch. TensorFlow and PyTorch are machine learning libraries primarily used to train a neural network to create a model. There are other Python libraries that can help while developing code to train a model, such as Keras and scikit-learn. Libraries related to machine learning or neural networks were useful, but other libraries such as NumPy also helped solve many computation problems, and Panda, for data manipulation and analysis [44].

3.1.3 PyTorch

The popular Torch deep learning framework inspired the name PyTorch. It was written in the Lua programming language. PyTorch is an open-source machine learning library primarily developed by Facebook's AI Research. The library was not complicated to use compared to the other machine learning libraries, such as TensorFlow. The developer of the library believed that they could create a deeper learning framework that is free from complexity. They wanted to build a library that was easy and simple to use, just like the Python programming language [46].

The PyTorch is the main open-source library used in this research to train the neural networks by using a pre-trained model. The library was downloaded with the help of PyCharm that has a package manager than can download different available packages. After downloading the package, the package can be imported by mentioning the package name of the class to use the functionality the library has to offer. The library has the functionalities to download pre-trained models such as Inception v3 and other available pre-trained models that could be used to test other types of models.

3.1.4 TensorBoard

It is essential to analyze the model so that the required adjustments can be made. The adjustments, in turn, can be used to create an improved model. TensorBoard is a web application tool for providing the visualizations and measurements needed during the machine learning workflow. It provides the functionality to track experiment metrics like accuracy and loss value, visualizing an interactable model graph, projecting embedding to a lower-dimensional space, and so forth. TensorBoard supports not only the TensorFlow library but also others such as PyTorch [47].

3.2 CUDA testing

It is crucial to have CUDA as it takes a long time to train or evaluate the model otherwise. There is a separate class that tests if the current CUDA version supports the PyTorch library, the total amount of GPU devices in the computer, and information about those devices (such as names). The result is printed in the console when the processing is done.

3.3 Data set

The data set was stored locally on a computer, in the same folder as the code that implements the transfer learning. Inside of the data set folder, there were two other folders. The first folder was the training folder, and the second folder was validation. The training folder had the images to train the model. The pictures inside the training folder were categorized with a labeled folder. Each number of labeled folders inside the training folder was the number of classes, and they held their featured image data for that class. The validation folder has the images to evaluate the model. It has a sample of the data set to test the model with the images that it has never seen before in each epoch. The validation folder also has a labeled folder, just like the training folder. The only difference between the training and validation folder was that the validation consists of less data than the training folder. In contrast, the training folder consists of the rest of the data set.

The training folder had approximately 4.400 images (1.81 GB), while the validation had around 2.000 images (825 MB). The data in the labeled folder contained images used to train or validate the neural network. The labels used to train the pre-trained model were ten different varieties of wild and non-wild animals. The ten labels were cats, dogs, wolves, bears, otters, buffalos, hippopotamuses, and chimpanzees. There were at least 100 images and a maximum of 1.400 images in each labeled folder in the training folder. The validation folder had a minimum of 50 images and a maximum of 700 images in each of the labeled folders.

The data type of each image was JPG. Each image can have a different dimension, but each image needed to have at least 299x299 for reaching the requirement of using a pre-trained model [5]. Each image, both in the training and validation folder were unique. Some of the images could contain one or more of the same label, and the label's whole body may not be visible.

Before creating or loading the model, the image data in both the training and validation folders were transformed during the execution time. The Inception model requires to resize the image data into 299x299 [5]. Before creating or loading the model, the data set was transformed into the given image size of 299 pixels, both in height and width. It also transformed each image into a tensor that each pixel will have an RGB value from 0 to 255, and it even normalized the tensors with given vectors. The training transformation rotated the images for data augmentation, efficiently increasing the model's accuracy and loss value without having additional data.

3.4 Create or load the model

There is a class that is capable of transforming the data set, creating and loading the model, training the neural network of that model, and saving the records. This functionality is implemented in a single class. The class can take in arguments when running the code from the terminal. The user can change the parameter of the class running from the terminal, such as giving the amount of epoch it should run in the training process, if the class loads a model from an existing local dictionary or if the fine-tuning is enabled. The user can see from the terminal what parameters are available as inputs by typing in the class name and then writing "--help". If the user does not give any inputs, it will run with the given default values. Getting the terminal's input value or the default value calls a method that creates or loads the model based on the given value. Creating a new model from scratch creates an inception version three models that are a pre-trained that has thousands of classes. Creating a new model from a pre-trained model requires some slight modifications, such as updating the number of classes from thousands of available labels in the training folder. Afterward, when the model has been created, several things are required before training the model. The criterion is necessary to calculate the loss value from the given output value from the model after inputting the featured data and the labeled data to check if the model has guessed right or wrong. The optimizer updates the weight of the parameters to minimize the loss function. The loss function acts as a guide for the optimizer to ensure that it moves towards the right direction to lower loss values. There is also a scheduler for the optimizer that reduces the learning rate every seventh epoch by 0.1. Reducing the learning rate makes fewer changes to the weights of the layers in the model. If the learning rate is too high, it will cause undesirable divergent behavior of the loss function [48].

Loading the model instead of creating it from scratch, is another approach. Instead of creating the pre-trained model, it only copies the layers it has and sets the amount labels available in the training folder. It restores the variables such as weights and bias from the previously saved model. It also regains criterion, optimizer, scheduler, and the iteration step from both training and validation. The iteration step must be separate from training and validation when plotting a graph.

Training the model using a small data set, especially when CovNet in the model has a massive parameter, significantly affects the CovNet's ability to generalize, often resulting in overfitting. There is a fine-tuning functionality that could be activated before training the model. Fine-tuning increases the training speed because only the last layer's weights are being adjusted in the model. If the data-set's labels are drastically different from the labels that were used to train the pre-trained model and the sample size of the data set is less than a thousand, it is not recommended to use the fine-tuning functionality. Training a model using a data set with the same labels as the pre-trained model generates familiar weights. However, training a model that has an unfamiliar data set as a pre-trained model generates different weights. Having a commonly labeled data set as a pre-trained model only slightly differs from the model's accuracy and loss value. Suppose the data set labels are unfamiliar with the pre-trained model. In that case, the accuracy and loss value improvements are not as good as having a common data set with the same labels. The model should then be trained without the fine-tuning functionality to adjust all the weights of the model's layers. The study was performed without turning on the fine-tuning because all the classes from the pre-trained model (Inception V3) were not included [35]. Having a small data set causes overfitting when using fine-tuning [49].

3.5 Training process

After all the preparation, such as creating a model, criterion, optimizer, and scheduler, the training process could finally begin. The first thing to do was to create an instance of a timer that could keep the training time. Before starting the training, it was essential to maintain a deep copy of the model's parameters dictionary using a deserialized state_dict method. The state_dict is a python dictionary object that maps each layer to its parameter tensor [50]. Keeping a copy of the model's parameters dictionary ensures having a model with the best validation accuracy. If another model surpasses the saved model's accuracy, it would replace it with that model.

There was a loop that iterates until it reaches the given max epoch value from the user. Each epoch trains the model with the whole data set available in the local dictionary and evaluates the model. The process continued until it reaches the end of the loop. There was an additional loop in the epoch loop. The second loop considered if the model is training or evaluating. It means the loop iterated two times. The first one was training, and the second was evaluating. It made sure to tell the model if it was evaluating or training and not using the validation images for training the model. There was also a third and final loop; iterated every feature and label available in the local data set, depending on whether the model is in the training or evaluation mode. If the model had been in the training mode, it would have used the training data set. If the model had been in the evaluation mode, it would have used the validation data set. There was a counter that counted the number of steps separately for training and evaluation. It was used to plot the x-axis of the graph to check the accuracy in each iteration. Before inputting the model with features data to train the model, the grad had to be enabled if it was in a training state. Enabling the grad made sure to clear the intermediate value for evaluations. It was needed to backpropagate during the training and saving memory space.

The model could then be inputted with featured data to train or evaluate the prediction of the model. When the model had generated an output (the prediction), it was used to calculate the accuracy and loss value with the label's help. When the accuracy and loss value calculation was done, it saved both values and iteration steps in the records file. Before the evaluation mode was done, it compared the validation accuracy if it was better than the previous epoch. If the accuracy was higher than the previous epoch, then the model's parameters dictionary should be deep copied. It will ensure having the best model with the highest validation accuracy. Before moving on to the next epoch, it will save the training and validation steps, model's parameters dictionary, optimizer, scheduler, and then criterion when training the same model for the next time. When all of the loops are made, it will print out the total time it took to finish the model's training.

3.6 Records

While the model was in a training or validation state, it would save the model's records in each iteration (batch size of four). The records were kept in a local historical dictionary in the project folder as a file. The file contains a graph, model's construction, and precision-recall curve saved from the Tensorboard library.

Two separate graphs share two lines each. The first graph was the accuracy with training and validation line, and the second was loss value with training and validation line. The accuracy and loss value were added in a single file in each iteration and the current amount of global step (iteration step).

When the training was complete, it saved the structure of the model. The user could then interact with the model to further inspect, such as the layers it contained and the output from the layers (figure 3). It was a useful functionality to check if the model behaved as expected or needed some slight modifications.

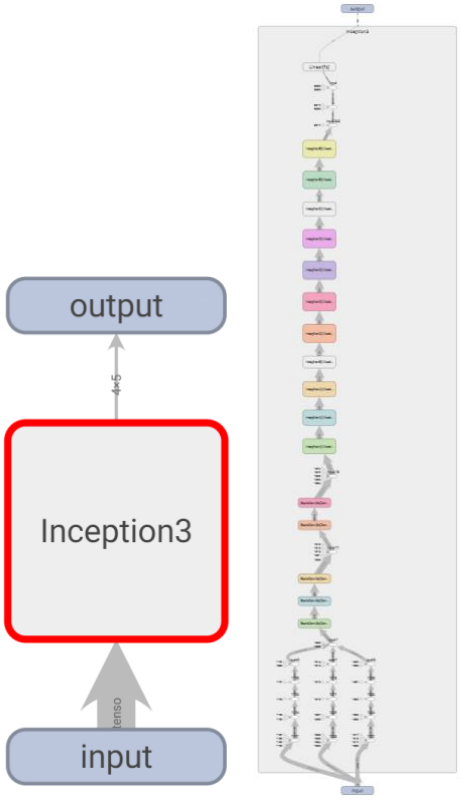

Figure 3. Inception model, both closed and open.

When the structure of the model was saved, it also kept the precision-recall curve (PR-curve). The PR-curve is a plot of the precision in the y-axis and the recall on the x-axis. Both precision and recall sampled the data points from the validation folder to test the model. The precision refers to the fraction of the relevant instance of the total of the retrieved instances. The recall refers to the fraction of relevant instances retrieved in the total amount of the relevant instance. In the other view of point, precision and recall are measurements of relevance [51].

$$precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + individuals\ incorrectly\ labeled\ as\ terroists}$$

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + terrorists\ incorrectly\ labeled\ as\ not\ terrorists}$$

Figure 4, the formula for both precision and recall [51]

True positive (TP) is when the model correctly predicts the positive class, false positive (FP) is when the model incorrectly predicts the positive class and false-negative (FN) is when the model incorrectly predicts the negative class [52]. If the precision value has generated a value of 1.0, that means the model has not produced any false positives. If the recall also gets the same value, that means the model has not produced any false negatives [53]. The formula for precision and recall can be observed in figure 4.

4 Result

The first model began training with the first two classes, cats and dogs. The result after training the model with ten epoch was 94.96% training and 97.07% validation accuracy. The PR-curve showed that the value of precision from both labels was 100%. It means that neither label got any false positives (Table 1). The data could then be used to compare it with the next model with an additional class.

The wolves were the next class to be added to the model. The model was trained with the same amount of epoch, but it had more iterations due to the fact that class were to be added into the model. Adding wolves into the model changed the accuracy roughly to 93.36% training and 95.66% validation. The training accuracy was lost by 1.6%, and the validation was lost by 1.41% compared to the previous model. The cat and dog class from the previous model dropped their precision by 2%. The wolves did not get any false negatives (Table 1). The labels (cats and dogs) from the previous model did not have any false positives. Just from adding wolves into the model caused cats and dogs to lose their precision slightly. The model was better in predicting wolves than cats and dogs because they had a better precision than the other classes.

Adding bears into the model changed the accuracy by approximately 93.07% training and 95.72% validation. Comparing it to the previous model, training accuracy was lost by 0.29%, and the validation accuracy was increased by 0.06%. Adding the bears into the model changed the precision of all the classes to 100% (Table 1).

The model's accuracy was changed to 92.53% on training and 95.46% on validation after adding otters to the model. Compared to the previous model, the training accuracy was lost by 0.54% and 0.26 on validation. The otters had a precision of 99.32%. Adding otters to the model reduced the cat's precision to roughly 99.12%, 96.82% of the dogs, 99.21% of the wolves, and 96.45% of the bears (Table 1).

Adding buffalos into the models changed the training's accuracy to 92.62% and 95.68% for the validation. The accuracy of the training model was improved by 0.09% and 0.22% for the validation, compared to the previous model. The buffalo's precision was 99.44%, and the otters and wolves were improved to 100%. The dogs also were improved to 97.80%. The decreased classes are bear to 92.72% and 97.33% of the cats (Table 1).

Adding the hippopotamus to the model changed the accuracy to 92.69% of the training and about 95.28% for the validation. The accuracy of the training was reduced by 0.07%, and validation increased by 0.40%. The precision of the hippopotamus was 97.96%. The precision of bears improved to 97.38%, while precision for buffalos was decreased to 96.22%, 95.48% for the otters, 96.32% of the dogs, and 96.22% of the cats (Table 1).

Adding chimpanzee into the model changed the training accuracy to 91.89% and 95.40% for the validation. The training's accuracy was reduced by 0.80%, and the validation was increased by 0.12%. The precision of the chimpanzee was 98.68%. The classes that were improved in precision were buffalos to 96.77%, 97.87% of the otters, and 97.16% of the dogs. The hippopotamus's precision decreased to 94%, 96.41% of the bears, and 95.82% of the cats (Table 1).

Table 1, the result of the first model with at least 250 images in each class.

Number of Classes	Training iteration	Validation iteration	Time	Training accuracy	Validation accuracy	Cat precision	Dog precision	Wolf precision	Bear precision	Otter precision	Buffalo precision	Hippopotamus precision	Chimpanzee precision
2	640	290	6:35	94.96%	97.07%	100%	100%	N/A	N/A	N/A	N/A	N/A	N/A
3	720	340	7:12	93.36%	95.66%	98.23%	98.06%	100%	N/A	N/A	N/A	N/A	N/A
4	900	390	8:29	93.07%	95.72%	100%	100%	100%	100%	N/A	N/A	N/A	N/A
5	1020	440	9:16	92.53%	95.46%	99.12%	96.82%	99.21%	96.45%	99.32%	N/A	N/A	N/A
6	1150	500	10:14	92.62%	95.68%	97.33%	97.80%	100%	92.72%	100%	99.44%	N/A	N/A
7	1250	540	11:00	92.69%	95.28%	96.22%	96.32%	100%	97.38%	95.48	96.22%	97.96%	N/A
8	1350	590	11:56	91.89%	95.40%	95.82%	97.16%	100%	96.41%	97.87%	96.77%	94%	98.68%

The accuracy dropped when an additional class was added into the model (figure 5). Sometimes the accuracy increased after adding a new class instead of decreasing. The pre-train model always has the same weights, but training the model can give different weights compared to the previous training session.

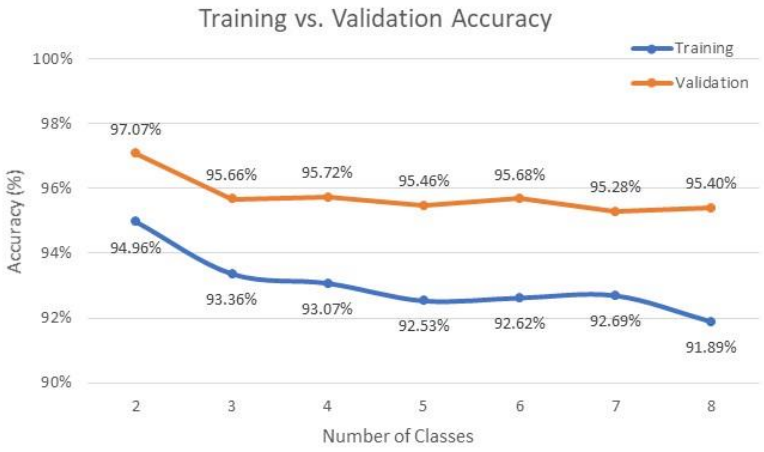

Figure 5. Accuracy changes of the first model.

The first two classes (cats and dogs) from the first model were used to compare the precision when adding new classes in the model. Most of the time, adding a new class into the model would drop the precision of cats and dogs (figure 6). Both cats and dogs had more data compared to other classes. However, there were different classes in the model that had better precision than cats and dogs. That might be because the other classes possibly had common features with cats and dogs. There are also different type of cats and dogs. Some could be ten times smaller than other dogs.

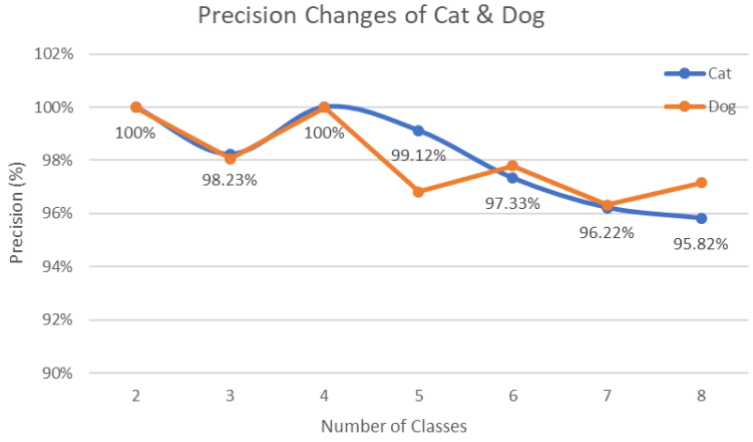

Figure 6. The precision changes for the cat and dog class in the first model.

The second model had beavers and weasels instead of hippopotamuses and chimpanzees. The beavers and weasels had less than 120 unique training images and 70 validation images. Giving fewer data to the beavers and weasels than the other classes reduced the precision of both and some other classes. It also reduced the accuracy of the model. The pre-trained model was the first model before training the bears. The result can be viewed from the table 2.

Table 2, the result of the second model that each class has at least 200 images, besides beaver and weasel. The beaver and weasel class has approximately three times fewer data compared to the other classes.

Number of Classes	Training iteration	Validation iteration	Time	Training accuracy	Validation accuracy	Cat precision	Dog precision	Wolf precision	Beaver precision	Weasel precision	Bear precision	Otter precision	Buffalo precision
2	640	290	6:35	94.96%	97.07%	100%	100%	N/A	N/A	N/A	N/A	N/A	N/A
3	720	340	7:12	93.36%	95.66%	98.23%	98.06%	100%	N/A	N/A	N/A	N/A	N/A
4	750	350	7:22	90.31%	94.21%	93.39%	97.87%	100%	78.26%	N/A	N/A	N/A	N/A
5	780	370	7:46	88.04%	88.15%	92.59%	96.61%	100%	78.26%	80%	N/A	N/A	N/A
6	970	430	8:47	89.26%	91.78%	95.22%	96.81%	98.48%	55.56%	97.62%	94.09%	N/A	N/A
7	1080	470	11:33	86.89%	89.22%	93.52%	95.88%	100%	55.56%	72.22%	96.52%	91.49%	N/A
8	1210	530	12:59	88.55%	90.65%	90.57%	96.52%	97.01%	55.56%	65.85%	96.60%	97.26%	98.88%

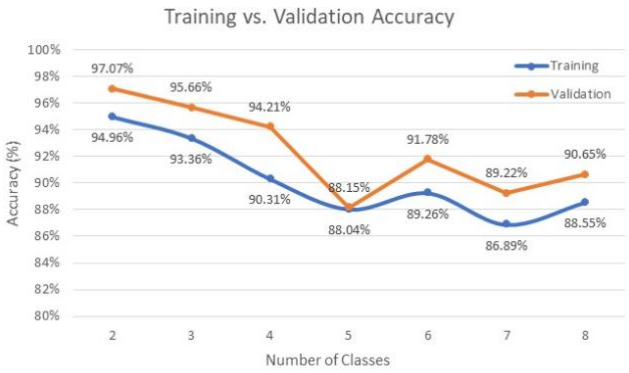

Figure 10. The changes to the accuracy of the second model.

The accuracy after the fourth class (beaver) dropped significantly, while after the fifth class (weasel), it fell much more dramatically (figure 10). The beaver and weasel precision was lower than the other classes in the model (Table 2). The other classes, such as the otter, had less precision than the first model because they might share the same feature. The cause of this precision was the lack of data for the beaver and weasel class.

5 Discussion

The purpose of the research was to determine how effective it is to use transfer learning for users with insufficient data to train an imagery model from scratch. The comparison between the first and second models was that if there were fewer data in one class than the others, it would negatively affect the model's accuracy and other class precision. The second model had two classes with approximately three times less data than the other classes. The result was that both the training and validation had less accuracy than the first model (Table 1 and 2). The two classes (beaver and weasels) from the second model had worse precision than the other classes. It affected other class's precision, such as cats. Having enough data for each class is essential to train a model when using a pre-trained model. The consequences of not having enough data for each class caused less accuracy for the model, and precision dropped for the classes with less data and some of the other classes.

The cats and dogs had the most data compared to the other classes in the model. Having more data than the other classes did not cause higher precision than the other classes. Checking the data for both of the classes clarified the reason behind it. There were different types of cats and dogs, such as the Pekingese and German Shepherd. Both of them are classed as dogs, but they do not look alike. The Pekingese is much smaller, and their ears and tail are almost not visible compared to the German Shepherd. The wolves class had less data than cats and dogs, but it still had more precision than both classes because there was only a single type of wolves in the data set. The classes with different types such as cats and dogs should be treated with the same amount of data as a single class if they do not look alike as the other types. It is also essential to have good enough data for other classes to improve the precision of the first two classes. The two classes replaced in the second model gave the majority of the classes less precision.

5.1 Previous attempts

There were different attempts to create a transfer learning project. The first attempt included using the TensorFlow library instead of the PyTorch library. However, there were some issues with restoring the weight and bias value, which was a vital part of the transfer learning. Not being able to transfer the weight and bias value from the pre-trained model would give the model random weight values.

The second attempt included using the later version of the TensorFlow. There is a script that can automatically upgrade the previous version code to the latest version of TensorFlow [54]. The script did not work because the critical module was missing in the latest version of the TensorFlow. The name of the module is tf.contrib, and it was deprecated when TensorFlow 2 was released [55]. It was an essential module from the previous code to achieve Transfer Learning.

5.2 Alternative tools

Different tools could have been used to train the neural network or even to represent the results. One of the tools that could have been used is to train the neural network is Keras library. There are also different kinds of libraries that could be used to represent the output of the model, such as Matplotlib and OpenCV library. The TensorBoard was chosen because it was easy to use and had the requirements (such as creating PR-curve and interacting with the graph) that were necessary to collect data.

6 Conclusions

The accuracy changed when new image classes were added into a model using a pre-trained model. Transfer learning enabled the recognition of eight classes of images data with a minimum of 95.4% validation and 91.89% training accuracy. The precision accuracy for each class was higher than 94% (figure A1). Each class had at least 250 unique images in the training folder and 100 in the validation folder. The average accuracy dropped by 0.28% for the training and 0.51% for the validation. The second model was created to test the minimum requirements of unique images. The hippopotamus and chimpanzees were replaced with beaver and weasel with a maximum of 120 unique training images and 70 unique images for validation. The beaver's precision dropped to 55.65%, and the weasels to 65.85%. The second model's accuracy dropped to 90.65% for the validation and to 88.5% of the training (figure A2). The second model proved that if there were classes with insufficient data to train the model, it would harm all the class precision, and the accuracy of the model would also be affected. Two classes of eight had less than 120 unique training and 70 unique validation images. The other classes had 250 unique training and 100 unique validation images, which was enough not to affect the accuracy of the model negatively. The average accuracy drop for the model was 1.07% for both training and validation.

The precision of the first two image classes when adding a new class into the model using a pre-trained model was as follows; the average precision drop for the first class, cats, is 2.09%, and for the second class, dogs, is 1.42%. Both cat's and dog's precision when adding a new class can be observed from figure A1. The dogs had much more data (unique images in the training and validation folder) compared to the other classes. The cats had the second most data. Even with that much data, they did not have the best precision compared to the other classes in the model. The classes that had better precision than the dogs and cats was otters, buffalos, and wolves. So having more data for a specific class did not mean that it will have higher precision compared to the other classes. The cause of the dog's lack of precision was likely that they had common features with other classes, such as wolves because they are the same animal type. It could also be that there were different types of dogs. It would be hard for the model to predict dogs if different breeds did not have widespread features to each other. Having more data for both kinds (Pekingese and German Shepherd) will help the model to predict both types as dogs [56].

References

[1] Cogito Tech LLC, "Where is Artificial Intelligence Used Today?," *Becoming Human*, 2019. https://becominghuman.ai/where-is-artificial-intelligence-used-today-3fd076d15b68#:~:text=Currently AI is Used is Following Things%2FFields%3A&text=Retail%2C Shopping and Fashion,Manufacturing and Production (accessed Mar. 13, 2020).

[2] Z. Kodelja, "Is Machine Learning Real Learning?," vol. 9, 2019, doi: 10.26529/cepsj.709.

[3] D. Gupta, "Master Transfer learning by using Pre-trained Models in Deep Learning," *Analytics Vidhya*, Jun. 01, 2017. https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/ (accessed May 05, 2020).

[4] "torchvision.datasets.folder — PyTorch master documentation," *PyTorch*. https://pytorch.org/docs/0.4.0/_modules/torchvision/datasets/folder.html#DatasetFolder (accessed Jul. 31, 2020).

[5] "Advanced Guide to Inception v3 on Cloud TPU | Google Cloud," *Google*, Jun. 30, 2020. https://cloud.google.com/tpu/docs/inception-v3-advanced (accessed Jul. 31, 2020).

[6] A. Şeker, "Evaluation of Fabric Defect Detection Based on Transfer Learning with Pre-trained AlexNet," Jan. 2019, doi: 10.1109/IDAP.2018.8620888.

[7] A. Esteva *et al.*, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, Feb. 2017, doi: 10.1038/nature21056.

[8] R. Margaret, "What is Artificial Intelligence (AI)?," *TechTarget*. https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence (accessed Mar. 17, 2020).

[9] I. Mihajlovic, "How Artificial Intelligence Is Impacting Our Everyday Lives," *Medium*, 2019. https://towardsdatascience.com/how-artificial-intelligence-is-impacting-our-everyday-lives-eae3b63379e1#:~:text=There are so many,get music or movie recommendations.&text=Email communications,Web searching (accessed Mar. 13, 2020).

[10] J. Naveen, "7 Types Of Artificial Intelligence," *Forbes*, Jun. 19, 2019. https://www.forbes.com/sites/cognitiveworld/2019/06/19/7-types-of-artificial-intelligence/#35821846233e (accessed Aug. 03, 2020).

[11] "Types of AI (K-Mean, NN-Type, Tree)," *University of Alcala*, Mar. 08, 2020. https://master-artificialintelligence.com/types-artificial-intelligence/ (accessed Jun. 12, 2020).

[12] S. Ray, "A Quick Review of Machine Learning Algorithms," in *Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Prespectives and Prospects, COMITCon 2019*, Feb. 2019, pp. 35–39, doi: 10.1109/COMITCon.2019.8862451.

[13] "Difference between Machine Learning, Deep Learning and Artificial Intelligence," *Medium*, Mar. 29, 2018. https://medium.com/@UdacityINDIA/difference-between-machine-learning-deep-learning-and-artificial-intelligence-e9073d43a4c3 (accessed May 31, 2020).

[14] L. Serafeim, "What is Machine Learning: Supervised, Unsupervised, Semi-Supervised and Reinforcement learning methods," *Towards Data Science*, Jun. 10, 2020. https://towardsdatascience.com/what-is-machine-learning-a-short-note-on-supervised-unsupervised-semi-supervised-and-aed1573ae9bb (accessed Aug. 17, 2020).

[15] A. Wilson, "A Brief Introduction to Supervised Learning," *Towards Data Science*, 2019. https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590 (accessed Mar. 13, 2020).

[16] W. Kenton, "Overfitting Definition," *Investopedia*, Jul. 02, 2019. https://www.investopedia.com/terms/o/overfitting.asp (accessed May 21, 2020).

[17] L. Shao, F. Zhu, and X. Li, "Transfer learning for visual categorization: A survey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 26, no. 5, pp. 1019–1034, May 2015, doi: 10.1109/TNNLS.2014.2330900.

[18] H. Zuo, G. Zhang, W. Pedrycz, and J. Lu, "Domain Selection of Transfer Learning in Fuzzy Prediction Models," in *IEEE International Conference on Fuzzy Systems*, Jun. 2019, vol. 2019-June, doi: 10.1109/FUZZ-IEEE.2019.8858992.

[19] S. Shao, S. McAleer, R. Yan, and P. Baldi, "Highly Accurate Machine Fault Diagnosis Using Deep Transfer Learning," *IEEE Trans. Ind. Informatics*, vol. 15, no. 4, pp. 2446–2455, Apr. 2019, doi: 10.1109/TII.2018.2864759.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database." Accessed: May 05, 2020.

[Online]. Available: http://www.image-net.org/papers/imagenet_cvpr09.pdf.

[21] "Inception-v3 convolutional neural network - MATLAB inceptionv3," *MathWorks*. https://www.mathworks.com/help/deeplearning/ref/inceptionv3.html (accessed May 05, 2020).

[22] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." Accessed: May 05, 2020. [Online]. Available: https://arxiv.org/pdf/1704.04861.pdf 05/05.pdf.

[23] I. Mansouri, "Computer Vision Part 4: An overview of Image Classification architectures," *Medium*, Mar. 27, 2019. https://medium.com/@ilias_mansouri/part-4-image-classification-9a8bc9310891 (accessed May 05, 2020).

[24] L. Dormehl, "What is an artificial neural network? Here's everything you need to know," *Digital Trends*, Jan. 06, 2019. https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/ (accessed May 08, 2020).

[25] "Real-Life and Business Applications of Neural Networks," *Smartsheet*. https://www.smartsheet.com/neural-network-applications (accessed May 08, 2020).

[26] "Artificial Neural Networks for Machine Learning - Every aspect you need to know about," *Data Flair*, Aug. 06, 2019. https://data-flair.training/blogs/artificial-neural-networks-for-machine-learning/ (accessed May 08, 2020).

[27] J. B. Ahire, "The Artificial Neural Networks handbook: Part 1 - Coinmonks," *Medium*, Aug. 24, 2018. https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4 (accessed May 11, 2020).

[28] G. Ognjanovski, "Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun," *Towards Data Science*, Jan. 14, 2019. https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a (accessed May 05, 2020).

[29] D. Kobran and D. Banys, "Weights and Biases," *AI Wiki*, Jan. 2020. https://docs.paperspace.com/machine-learning/wiki/weights-and-biases (accessed Aug. 17, 2020).

[30] J. Collis, "Glossary of Deep Learning: Bias - Deeper Learning," *Medium*, Apr. 14, 2017. https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2 (accessed May 16, 2020).

[31] "CS231n Convolutional Neural Networks for Visual Recognition," *Standford University*. https://cs231n.github.io/ (accessed May 08, 2020).

[32] "Convolutional Neural Networks tutorial - Learn how machines interpret images," *Data Flair*, Aug. 16, 2019. https://data-flair.training/blogs/convolutional-neural-networks-tutorial/ (accessed May 08, 2020).

[33] "Convolutional Neural Network: A Step By Step Guide," *Towards Data Science*, Mar. 17, 2019. https://towardsdatascience.com/convolutional-neural-network-a-step-by-step-guide-a8b4c88d6943 (accessed May 08, 2020).

[34] "How long and how many epochs does it take to train a net?," *International Computer Science Institute*, Aug. 07, 2000. https://www1.icsi.berkeley.edu/Speech/faq/nn-epochs.html (accessed Aug. 05, 2020).

[35] "text: imagenet 1000 class idx to human readable labels (Fox, E., & Guestrin, C. (n.d.). Coursera Machine Learning Specialization.)," *GitHub*, Jan. 2019. https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a (accessed May 19, 2020).

[36] S. Shah, "Do we really need GPU for Deep Learning? - CPU vs GPU," *Medium*, Mar. 26, 2018. https://medium.com/@shachishah.ce/do-we-really-need-gpu-for-deep-learning-47042c02efe2 (accessed May 09, 2020).

[37] "GPU support," *TensorFlow*. https://www.tensorflow.org/install/gpu (accessed May 09, 2020).

[38] "CUDA semantics — PyTorch 1.5.0 documentation," *PyTorch*. https://pytorch.org/docs/stable/notes/cuda.html (accessed May 09, 2020).

[39] M. Heller, "What is CUDA? Parallel programming for GPUs," *InfoWorld*, Aug. 30, 2018. https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html (accessed May 09, 2020).

[40] "CUDA GPUs," *Nvidia Developer*. https://developer.nvidia.com/cuda-gpus (accessed May 09, 2020).

[41] "Installation Guide Windows :: CUDA Toolkit Documentation," *Nvidia Developer*, Nov. 28, 2019. https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html (accessed May 09, 2020).

[42] O'Hara Andrew, "Apple says these Mac Pro graphics cards are Mojave compatible, reveals Boot Camp support limited for iMac," *Appleinsider*, 2019. https://appleinsider.com/articles/18/09/24/apple-says-these-mac-pro-graphics-cards-are-mojave-compatible-reveals-boot-camp-support-limited-for-imac (accessed May 25, 2020).

[43] "Installation Guide Mac OS X :: CUDA Toolkit Documentation," *Nvidia Developer*, Nov. 28, 2019. https://docs.nvidia.com/cuda/cuda-installation-guide-mac-os-x/index.html (accessed May 09, 2020).

[44] H. Bansal, "Best Languages For Machine Learning in 2020!," *Becoming Human*, 2019. https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd242 (accessed Mar. 15, 2020).

[45] "What is Python? Executive Summary," *Python*. https://www.python.org/doc/essays/blurb/ (accessed Aug. 18, 2020).

[46] "Features," *PyTorch*. https://pytorch.org/features/ (accessed Mar. 15, 2020).

[47] "Visualisation with TensorBoard," *Databricks*. https://databricks.com/tensorflow/visualisation (accessed May 07, 2020).

[48] J. Jordan, "Setting the learning rate of your neural network.," *Jeremy Jordan*, Mar. 01, 2018. https://www.jeremyjordan.me/nn-learning-rate/ (accessed May 14, 2020).

[49] F. Yu, "A Comprehensive guide to Fine-tuning Deep Learning Models in Keras (Part I)," *Felix Yu*, Oct. 03, 2016. https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html (accessed May 15, 2020).

[50] "Saving and Loading Models — PyTorch Tutorials 1.5.0 documentation," *PyTorch*. https://pytorch.org/tutorials/beginner/saving_loading_models.html#what-is-a-state-dict (accessed May 16, 2020).

[51] K. Will, "Beyond Accuracy: Precision and Recall," *Towards Data Science*, Mar. 03, 2018. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c (accessed May 13, 2020).

[52] "Classification: True vs. False and Positive vs. Negative," *Google Developers*, Feb. 10, 2020. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative (accessed May 13, 2020).

[53] "Classification: Precision and Recall | Machine Learning Crash Course," *Google Developers*, Feb. 10, 2020. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (accessed May 13, 2020).

[54] "Automatically upgrade code to TensorFlow 2," *TensorFlow*, May 19, 2020. https://www.tensorflow.org/guide/upgrade (accessed May 24, 2020).

[55] "Module: tf.contrib," *TensorFlow*, May 05, 2020. https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/contrib (accessed May 24, 2020).

[56] S. Ray, "8 Proven Ways for boosting the 'Accuracy' of a Machine Learning Model," *Analytics Vidhya*, Dec. 29, 2015. https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/ (accessed May 28, 2020).