The Effects of Hyperparameter Choices on Convolutional Neural Networks

To what extent are the accuracy and time taken by a Convolutional Neural Network (CNN) affected by the kernel size and activation function for image classification?

Computer Science EE

Word Count: 3991

Contents

1 Introduction

Artificial intelligence (AI), a broad term given to the act of a computer completing a task based on a set of rules, is all around us (Garbade, 2018). In this era of groundbreaking developments across many fields, it's a given that new technology is being produced every day in order to be used as a tool to allow researchers to more easily facilitate the creation of knowledge. If you have ever played a video game with a bot, searched for anything on a web browser such as Google, or even found yourself locked out of an account needing to click on pictures of traffic lights as a form of security - a CAPTCHA - you have interacted with AI.

AI can be implemented using a machine learning algorithm. These machine learning algorithms are the main ways that people actually implement AI into pieces of technology and everyday life. The actual implementation is not so easy though, as there are many factors, or parameters, that these machine learning algorithms require, and prior research is necessary. Computer Vision is a field of AI that directly deals with taking in image data interpreting it, is a new field that has become extremely popular recently ("Advances in Computer Vision", 2020). This has many applications, and one of the most common today is image or feature recognition. Examples of this are seen more and more frequently in medical use (Qayyum et al., 2017). As seen more recently, researchers have developed a way to quantify COVID-19 and its severity using a machine learning algorithm known as a Convolutional Neural Network (CNN) (Chassagnon et al., 2020). CNNs are streamlined for use with Computer Vision and have become extremely popular as a result (An, 2017). To what extent are the accuracy and time taken by a CNN affected by the kernel size and activation function for image classification? There are many parameters that a CNN accepts, and this paper will be looking specifically at the number of kernels in a layer, and the activation function of each layer.

A matrix is a collection of elements that have a number of dimensions and a size (Org, n.d.). Kernels are matrices that, through the use of convolution, are used to take in image data to produce a feature map, another matrix that provides numerical information about important features that the convolutional layer was able to find (Thomas & Miap, 2019). This shrinks the data so that it may not take as long to train or test (Thomas & Miap, 2019). This is important because the more pictures that are fed in, the longer the CNN will take to process the images through training and testing (Thomas & Miap, 2019). An example of how kernels work is seen in Figure 1 below. Along with the size of the kernel matrix and its effects on accuracy and the time taken to train and test a CNN, this paper will also explore activation functions and their effects on the accuracy and time. Activation functions apply an equation to each neuron in the layer to determine the output at each one ("7 Types of Activation Functions in Neural Networks: How to Choose?", n.d.). This paper will address four activation functions: Sigmoid, TanH, ReLU, and Leaky ReLU. These are four of the most popular activation functions currently, and their graphs are shown in Figure 2 below ("7 Types of Activation Functions in Neural Networks: How to Choose?", n.d.). This paper will explore how the accuracy and time taken by a CNN are affected by the kernel size and the activation functions used because these are important hyperparameters, parameters that are externally set by the operator of the network ("Hyperparameters: Optimization Methods and Real World Model Management", n.d.).

This paper attempts to answer the question: "**To what extent are the accuracy and time taken by a Convolutional Neural Network (CNN) affected by the kernel size and activation function for image classification?**" This will be done by modifying the hyperparameters, kernel size and activation function, at each layer in a CNN, and running the same training and testing data through it to measure both accuracy and time taken. It is pertinent that the effects and implications of every hyperparameter choice is understood; this research can provide important information towards furthering different fields that require AI. Currently, in medicine, there is an ever-growing need for more AI. As an example, AI is needed in hospitals to be able to determine which patients need more care than others as these hospitals are running out of space due to the current pandemic. This is only one of many applications that deep learning and AI have in the present day. As the fields of AI and machine learning grow, become more accurate, and require less resources, society will be able to more effectively implement AI as a tool to help further the developments of the human race.

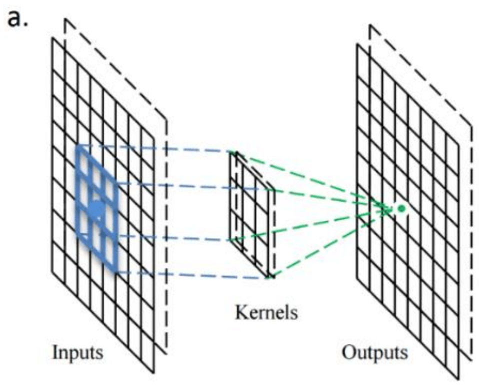

Figure 1: This image shows how the convolutional layer takes in an input and shrinks the image using a kernel matrix of (3x3), and turns that into a feature map or an output (Poletaev et al., 2016).

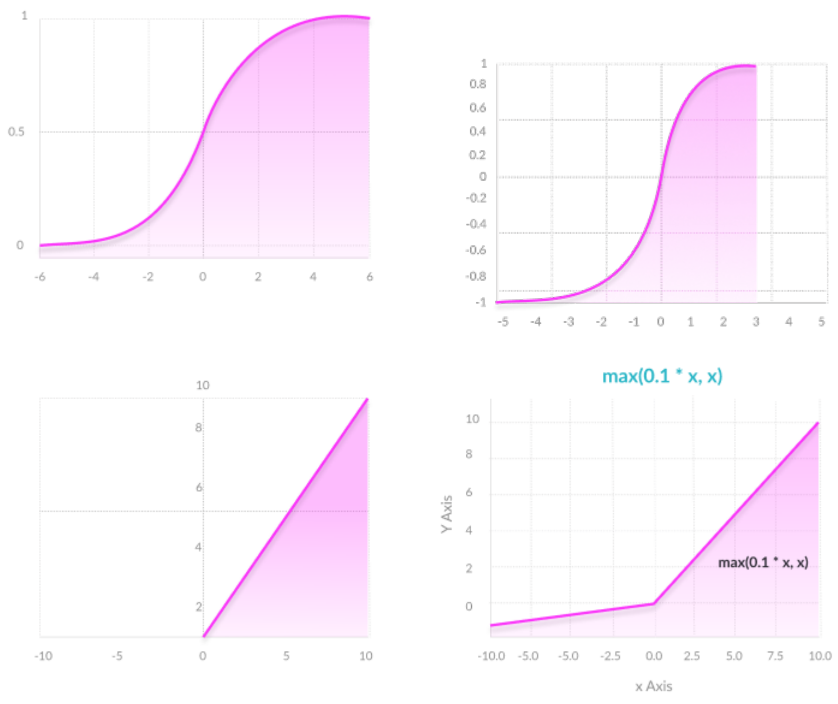

Figure 2: The images above show the different activation functions that will be used in this paper. From left to right on the top are the Sigmoid activation function, also known as the logistic function, and the TanH activation function. From left to right on the bottom are both ReLU and LeakyReLU respectively ("7 Types of Activation Functions in Neural Networks: How to Choose?", n.d.).

2 Background Information

Machine learning, that is the actual learning of a piece of technology through the use of data, is used to try to make predictions about data (Garbade, 2018). The CNN does use machine learning, but it uses a very specific type of machine learning called deep learning. In deep learning algorithms, it tries to model the way in which a brain actually functions, where tensors, or pieces of data, could be compared to a neuron. Imagine a scenario in which a machine with no idea of what a fruit is, is provided with thousands of different descriptions of fruit and each group of descriptions was given a hidden name of which fruit it was describing. The machine would then take in all of that data and learn what descriptions were used for each fruit. Then, the machine was given a testing dataset in which it had to take the descriptions without the hidden fruit name and map it to a fruit. In this example, the fruit names would be the training dataset, where the machine learning algorithm takes in the data and learns from it, and the testing dataset would be the one in which there was descriptions without the fruit names and it tried to accurately guess, based on calculations done, which fruit it was. This is a basic example of how machine learning works. In this paper, this was done with pictures of different landscapes in which the machine learning algorithm was required to state which one the image represented based on prior learning.

2.1 Convolutional Neural Networks

This paper will focus specifically on a newer machine learning algorithm that specializes in Computer Vision: the CNN. A CNN is a great tool in image processing due to its ability to take in and manipulate image data. This paper will deal with image classification. To use an example stated above, it would be like classifying which fruit is being described, but instead it is a mathematical representation of pixels that is manipulated to find specific features that are contained within those images to then finally classify it.

2.1.1 Images in CNNs

In a CNN, images cannot be put through as matrix of pixels. Those pixels have to undergo a transformation to be able to be put through. In this case, CNNs understand tensors, or multiple matrices side by side. Using tensors allows the machine learning algorithm to pass this data in because it is a numerical representation of an image. For this research paper, all images were of size 150x150 pixels and had three color channels: red, green, and blue. The tensor for this would then look like a 150x150 image, but three pixels deep. Figure 3 shows what an image would look like as a tensor if it was a 4x4 image with only two color channels: for example red and blue (Chm, 2018).

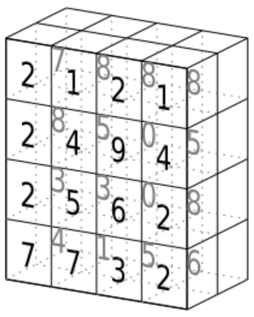

Figure 3: This is an adapted image from mc.ai that displays how a 4x4 image with two color channels would look like if it was represented as a tensor (Chm, 2018).

2.1.2 Convolutional Layers

CNNs are made up of layers: convolutional, pooling, and a fully connected layer for output. Traditionally, there is an organization between convolutional and pooling, going back and forth until the image is successfully downsampled to the programmer's desires. This downsampling is done through the convolutional and pooling layers. The convolutional layer accepts a 3D tensor of width, height, and depth, and four hyperparameters: the number of filters, or the number of kernels, the kernel size, the stride, or the number of pixels that the kernel will convolve over, and the amount of padding. The padding is what makes sure that one does not convolve over empty space, it simply places a pixel of 0 in place of where the kernel would have convolved. An example of a convolutional layer and the way that it works is seen in Figure 4 ("Convolutional neural networks for visual recognition", n.d.).

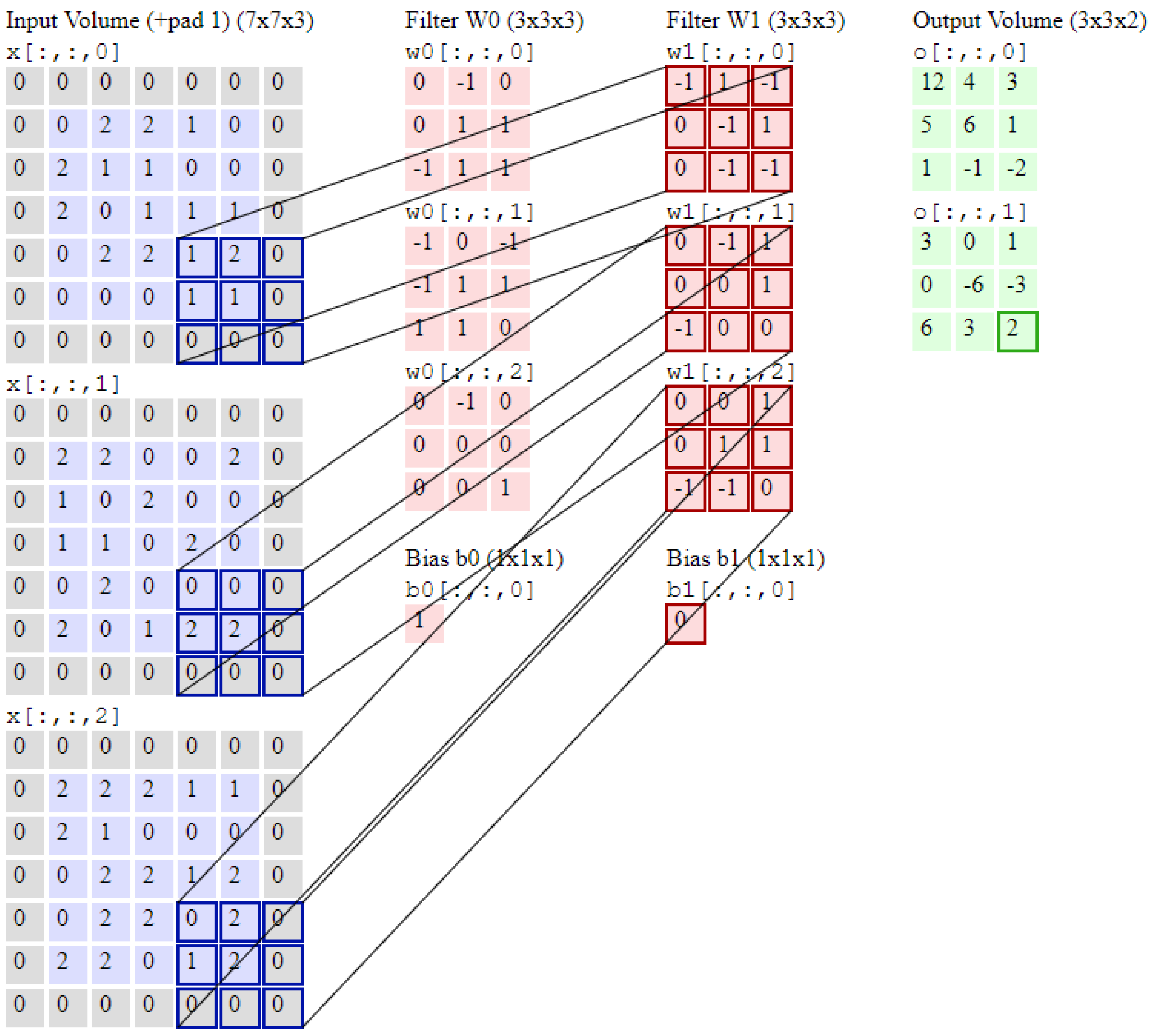

Figure 4: An adapted image from a GIF from cs231n.github.io of a convolutional layer in work on a 5x5 image with three color channels, two filters, 3x3 kernel size, two strides, and one layer of padding ("Convolutional neural networks for visual recognition", n.d.).

2.1.3 Pooling Layers

The pooling layer does most of the work in downsampling. It takes in a filter size and a stride number, and applies a function to determine which values will be chosen for the downsampling. Downsampling is a way for the machine learning algorithm to simplify data so that there are not so many trainable parameters. The more parameters there are, the more computational stress is put on one's machine and the more time it will take to train. An example of a pooling operation is called max pooling in which it takes the maximum value from the values that the filter passes over and it pushes that into the output to downsample. This is extremely useful in cases of large data such as a 150x150 image. An example of downsampling is seen in Figure 5 and an example of a max pool function being applied is seen in Figure 6 ("Convolutional neural networks for visual recognition", n.d.).

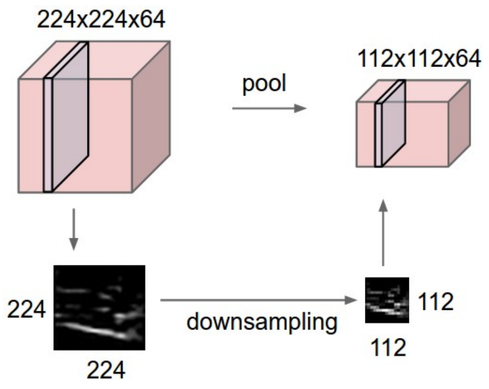

Figure 5: This is an image of how pooling downsamples tensors, outputting less data ("Convolutional neural networks for visual recognition", n.d.).

2.1.4 Fully Connected Layers

For the fully connected layers, in the case of the CNN created for this paper, dense connected layers were used. A dense layer is a fully connected layer that takes in an input volume and returns a vector whose dimensionality is equal to the number of classes that the machine learning algorithm is able to choose from. For example, if a machine learning algorithm was training on data of cats and dogs and was learning to classify images as either a cat or a dog, the dense layer would take in an input volume equal to the volume of the output of the last layer, and would return a prediction as to which of the two animals it thinks is in the test image provided. This would output a 2D vector because there are two classes, cat and dog, and determine based on the input of the previous layer which animal it most likely is (Deshpande, n.d.).

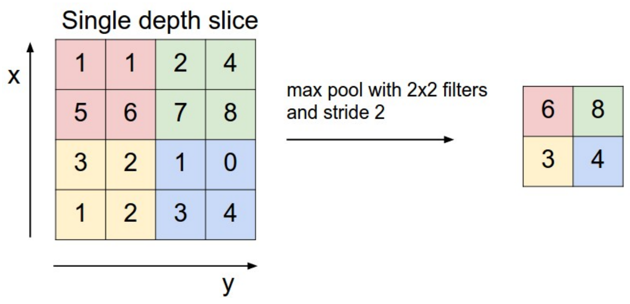

Figure 6: This is an image showing the actual process of a max pooling layer and its output from an input tensor ("Convolutional neural networks for visual recognition", n.d.).

2.1.5 Backpropagation

The way that computer vision algorithms ”learn” is through a process called backpropagation. The algorithm has no way of knowing what filter values, or weights, to apply. These weights are what determines what features the algorithm is looking for when trying to classify an image. Backpropagation is made up of four sections: the forward pass, loss function, backward pass, and weight update. The forward pass takes an input tensor of an image and passes it through the network, and the output vector, assuming the cat and dog classification example above, would look something like [.5 .5], where each value represents an equal probability of being a cat or a dog. Then, the loss function, a function that determines the error, uses the actual and predicted values to find a single number between 0 and 1 to determine the error, or loss. To find the actual value, the machine learning algorithm simply takes the label that was assigned to the image beforehand and gets the value. Using the cat and dog classification example, the label would look something like [1 0] if it was a cat, or [0 1] if it was a dog. A common loss function used is called MSE, or mean squared error, and the equation is shown in Figure 9 (Deshpande, n.d.). The MSE function gives a number closer to 1 for a predicted value, or output, the further it is from the actual value, or the target. This function allows one to minimize the error using the Calculus concept of optimization. The backward pass is then done to find which weights most contributed to the loss, and the weight update is done following the backward pass. The weights are adjusted according to which ones most contributed to the loss: the loss may then decrease. Adjusting these weights is done opposite to the gradient, or slope, of the loss function and is adjusted according to the learning rate, a rate determined by the programmer that defines how much the weights are updated with each pass through the network. There is a pictorial representation of both the weight adjustment equation and an example of how a high learning rate may affect a network in Figure 8 and in Figure 9 respectively (Deshpande, n.d.). Backpropagation is done iteratively until the number of batches, number of iterations, chosen by the programmer has been completed by the algorithm. This is especially important because when done correctly, the loss may be minimized and the algorithm can become more and more efficient, leading to higher accuracy (Deshpande, n.d.).

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Figure 7: This is an image showing the MSE, mean squared error, equation. The loss is equal to half of the actual minus predicted values squared. This will provide a number closer to 0 for a value that most closely matches its actual value, and a number closer to 1 for a value far from the actual value, or target (Deshpande, n.d.).

$$w = w_i - \eta \frac{dL}{dW}$$

Figure 8: This is an image showing how the weights are adjusted after each iteration of back-propagation occurs. The weight is adjusted by the learning rate multiplied by the gradient, or slope, of the loss function (Deshpande, n.d.).

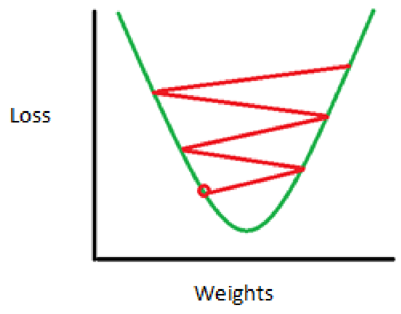

Figure 9: This image shows that a very high learning rate leads to an unoptimized algorithm that cannot adjust its weights effectively. The loss may not be minimized effectively; the loss will be higher and the accuracy will be lower than if the learning rate was chosen more effectively by the programmer (Deshpande, n.d.).

2.1.6 Image Classification

It's a bird! It's a plane! It's a cat. It's a mountain... Image classification is just that: looking at an image and being able to recognize specific features and determine what the image is. In the case of humans, it is common to pull prior information from memory to classify or to determine what something is. On the other hand, in the case of machine learning, image classification is learned through training data and feature extraction in the hidden layers, layers like the convolutional layers that aren't input or output. A machine learning algorithm can attempt to classify an image based on the feature extraction done in order to produce the most accurate classification given the specific hyperparameters (Bonner, 2019).

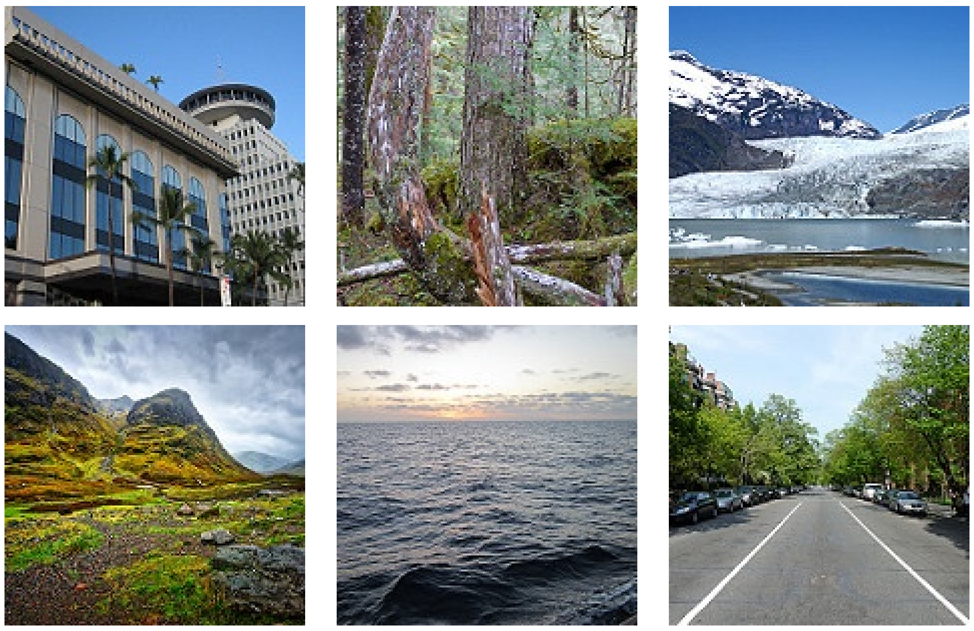

Figure 10: These are images from the training dataset used for the CNN in this paper. From left to right and then top to bottom the different classes of images are "buildings," "forest," "glacier," "mountain," "sea," and "street" (Bansal, 2019).

2.2 Hyperparameters

Hyperparameters and parameters are two words that many people think are interchangeable when it comes to machine learning and artificial intelligence, but in reality they are not ("Hyperparameters: Optimization Methods and Real World Model Management", n.d.). Hyperparameters are values that the programmer manually sets within the code, whereas parameters are not set by the programmer, ”they are estimated or learned automatically from training samples” (“Hyperparameters: Optimization Methods and Real World Model Management”, n.d.). This paper will look at the effects of two hyperparameters on the time and accuracy of a CNN: kernel size and activation functions.

2.2.1 Kernel Size

When analyzing pixel data from an image, the CNN must be able to ”see” the image. The way that this is done is similar to human eyes, and a CNN uses something called a filter, also known as a kernel. This filter can have many different sizes, and is chosen by the programmer. The CNN finds features by using the kernel and places them on a feature map as seen in Figure 11 (Sahoo, 2018). A small kernel size is analogous to looking at an object through a peephole in a door, whereas a large kernel size is analogous to looking at an object without any hindrances. Having a small kernel size may not be the best option for a CNN. Without being able to analyze neighboring pixels, the CNN will not be able to easily find useful features. On the other hand, having a large kernel size may also not be the best option. For example, facial recognition may look at local features and need a smaller kernel size, and a medical scan may look at larger features and need a larger kernel size. According to Sabyasachi Sahoo, ”most of the useful features in an image are usually local,” meaning that having too large of a kernel size may also hinder the CNN’s ability to find useful features in that it may scan over some of the useful features (Sahoo, 2018).

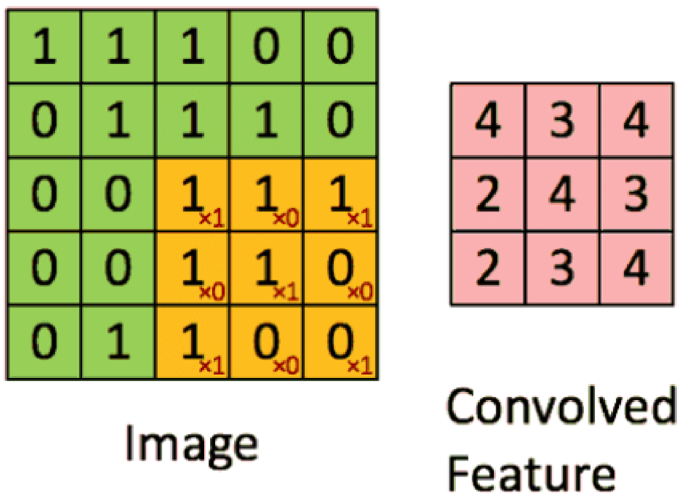

Figure 11: This is an adapted image taken from a gif from towardsdatascience.com and it shows how the pixel data from an image can be converted to a feature map through the use of a kernel. This image is using a 3x3 kernel size (Sahoo, 2018).

2.2.2 Activation Functions

According to missinglink.ai, ”activation functions are mathematical equations that determine the output of a neural network” (“7 Types of Activation Functions in Neural Networks: How to Choose?”, n.d.). To understand how these activation functions operate, a deeper understanding of the structure of a neural network is needed. As seen in Figure 12, there are three main layer types: input, hidden, and output (Raycad, 2017). Neurons are found at each layer and can be seen as a bubble in many diagrams. There are neurons found at each layer and each one has an activation function applied to it to determine what is output into the next layer. The activation function also takes in the output from the previous layer’s neuron and converts it into a form which can act as input for the next layer’s neuron. The activation function plays a key role in determining what the final output is. In the case of a multiclass image classifying neural network, will be what class, or type of image, the given image is. By using a different activation function for a CNN, each neuron has a different mathematical equation applied to it, and thus the output will very likely change. Finding the best activation function and kernel size for every machine learning algorithm is one very important step forwards towards the future of AI (“7 Types of Activation Functions in Neural Networks: How to Choose?”, n.d.).

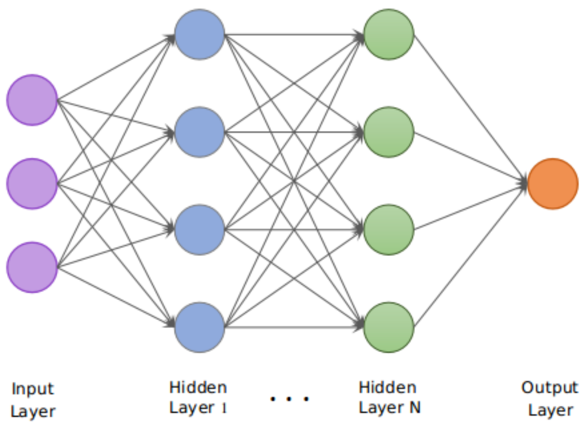

Figure 12: This image shows the different layers of a neural network, input, hidden, and output, and represents neurons with colored circles (Raycad, 2017).

3 Methodology

To analyze how accuracy and time are affected by differing kernel sizes and activation functions, four different kernel sizes and activation functions were used and the time taken to train and validate was measured along with the accuracy from the validation. The four kernel sizes used were (1, 1), (2, 2), (3, 3), and (4, 4), and the four activation functions used were TanH, Sigmoid, ReLU, and Leaky ReLU. The four kernel sizes and four activation functions create 16 different sets of hyperparameters with 10 epochs, or passes through the CNN, for each combination. To analyze both accuracy and time together, the formula (accuracy/time) was used. A larger number as an output from this function is better; accuracy should be maximized and time should be minimized for a more time efficient CNN.

3.1 Variables

The independent variables are the kernel size and the activation function in the convolutional layers. The dependent variables are time and accuracy. The control is the dataset used.

3.2 Dataset Used

A multiclass image scene classification dataset was used from kaggle.com and the title is ”Intel Image Classification” (Bansal, 2019). It contains 14,304 training images and 3,000 validation images in six different classes of images: ”buildings,” ”forest,” ”glacier,” ”mountain,” ”sea,” and ”street” (Bansal, 2019).

4 Results

Time (h:mm:ss)	**(1, 1)**	**(2, 2)**	**(3, 3)**	**(4, 4)**
TanH	00:02:21.850	00:02:46.030	00:03:06.610	00:03:37.890
Sigmoid	00:02:18.900	00:02:43.950	00:03:04.850	00:03:38.220
ReLU	00:02:12.370	00:02:33.920	00:02:47.300	00:03:35.760
Leaky ReLU	00:02:27.550	00:02:51.110	00:03:11.630	00:03:50.720

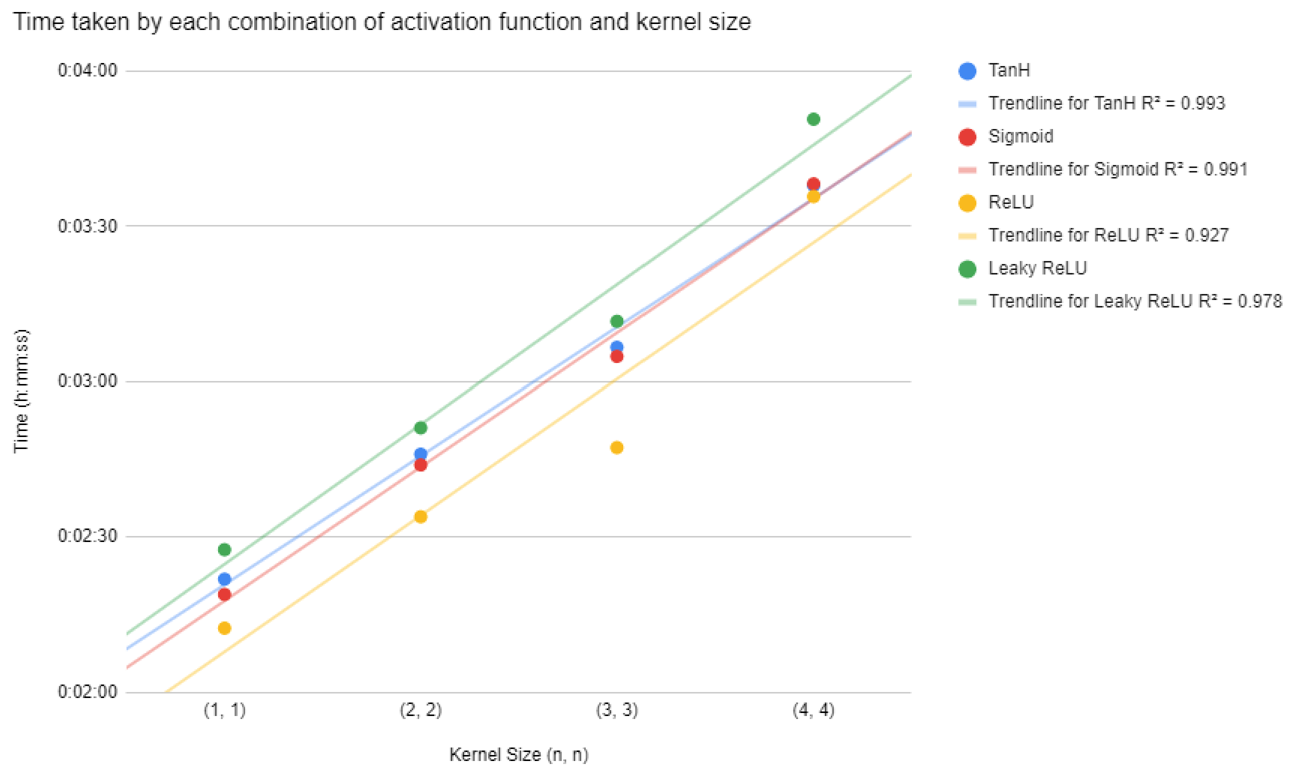

Figure 13: A visual representation of the time table above.

Accuracy	**(1, 1)**	**(2, 2)**	**(3, 3)**	**(4, 4)**
TanH	0.7076667	0.76166666	0.748	0.733
Sigmoid	0.175	0.175	0.175	0.6993333
ReLU	0.7316667	0.7776667	0.799	0.791
Leaky ReLU	0.73066664	0.7633333	0.8056667	0.785

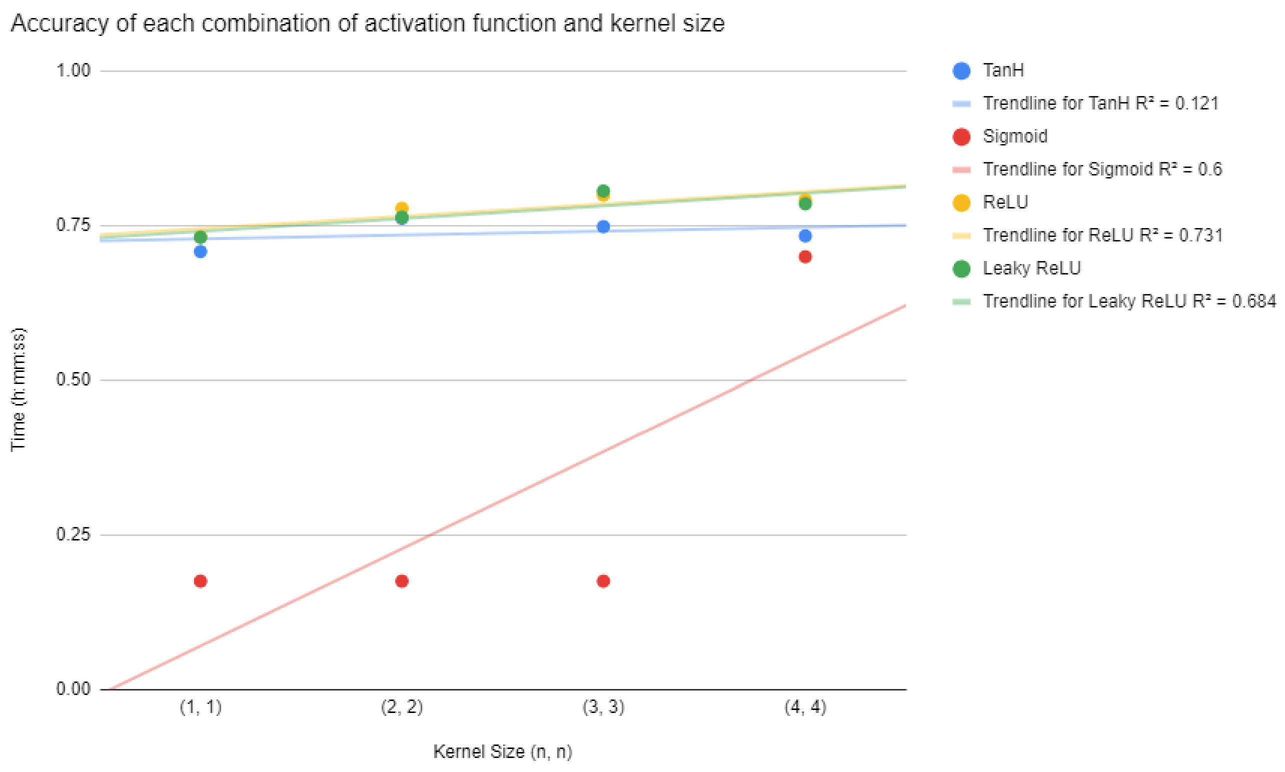

Figure 14: A visual representation of the accuracy table above.

((Accuracy)/(Time (s))) (rounded to 3 significant figures)	**(1, 1)**	**(2, 2)**	**(3, 3)**	**(4, 4)**
TanH	0.499	0.459	0.401	0.336
Sigmoid	0.126	0.107	0.0947	0.320
ReLU	0.553	0.505	0.478	0.367
Leaky ReLU	0.495	0.446	0.420	0.340

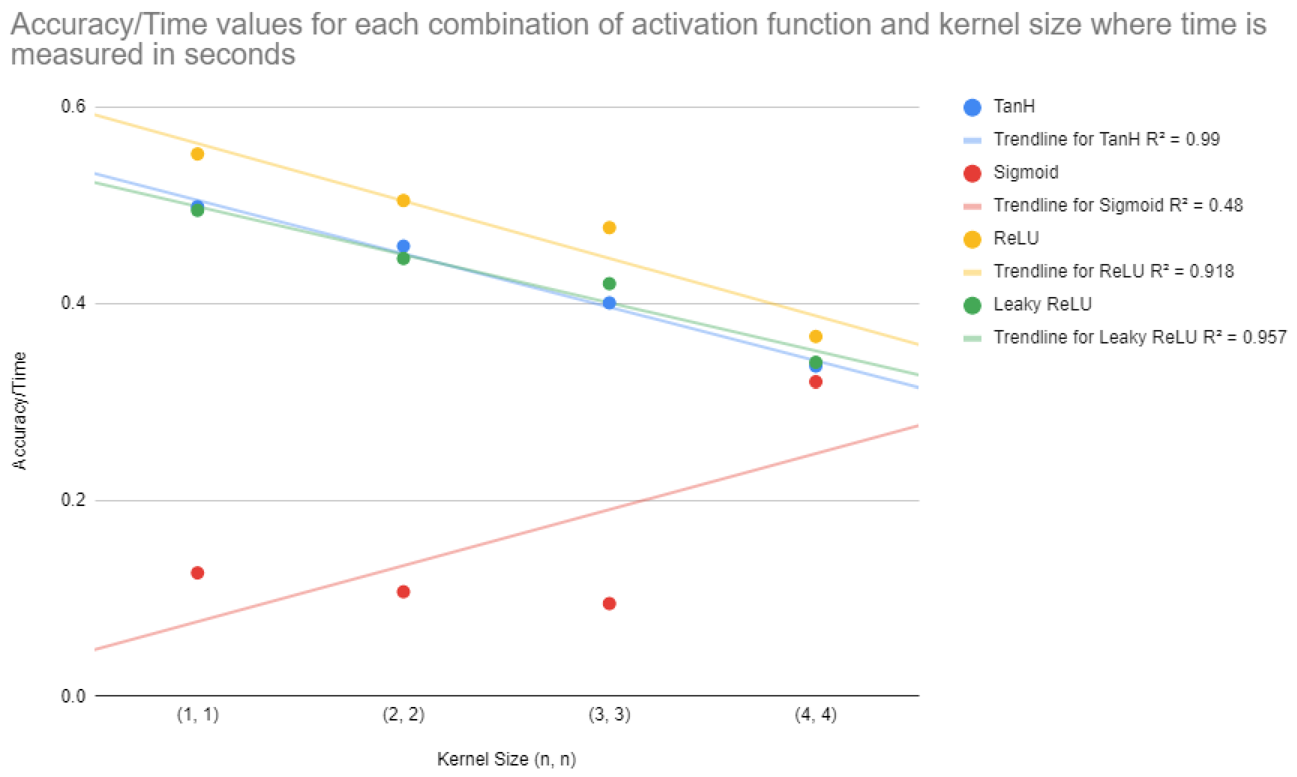

Figure 15: A visual representation of the accuracy/time table above.

4.1 Discussion of Results

The time taken by each combination was expected. The functions themselves don't carry a heavy load on the RAM, and the only thing that made a large difference was the kernel size. The accuracy however is very interesting. There is a discrepancy in the accuracy of the Sigmoid activation function at kernel sizes below (4, 4). As stated in missinglink.ai, the problem seems to lie in a ”vanishing gradient” where having a low value can lead to minimal changes in the prediction, causing the CNN to not learn further (“7 Types of Activation Functions in Neural Networks: How to Choose?”, n.d.). This is one of the reasons that many people dislike using the Sigmoid function. It is also interesting that at about a (3, 3) kernel size, TanH, ReLU, and

Leaky ReLU hit a peak and then their accuracies lowered with the larger kernel size. The best overall performance was seen by ReLU with Leaky ReLU, TanH, and Sigmoid trailing behind respectively.

5 Conclusion

AI has a long way to go before it can effectively and efficiently be integrated into our every day lives. With the introduction of voice assistants in various devices, the spread of misinformation about AI has skyrocketed over the past few years. Although the CNN used had a basic architecture, this paper was able to effectively explore and answer the question: "**To what extent are the accuracy and time taken by a Convolutional Neural Network (CNN) affected by the kernel size and activation function for image classification?**" With the optimization of hyperparameters and a more complex architecture, it is most certainly possible to approach 100% accuracy.

5.1 Implications

With the optimization of hyperparameters, AI is able to become more effective, efficient, and accurate. This means that as society approaches a more modern era in which surgeries are done completely with AI, X-rays and CAT scans are analyzed by AI to detect problems inside one's body, and cars are being driven entirely by AI, the small improvements made daily on any given machine learning algorithm have wide-reaching implications. As AI becomes more accurate, companies will be more inclined to include it in their systems, and society will benefit greatly as a result. As an example, according to the Tesla website, in the 2nd quarter of 2020, there was one accident for every 4.53 million miles while autopilot was enabled, and without autopilot, in the United States, there was an average of one accident for every 479,000 miles ("Tesla Vehicle Safety Report", 2020). That means that with Tesla's AI, it is 10 times less likely to have an accident ("Tesla Vehicle Safety Report", 2020).

5.2 Reflection and Further Research

If this were to be replicated, the results would most definitely be improved if each combination of kernel size and activation function were repeated multiple times using different seeds and taking an average. Another change that could be made is to use a larger number of kernel sizes and experiment with them over more activation functions. This would allow for a wider range of results and more conclusions could be drawn about the effectiveness of both the activation functions and kernel sizes. Finally, having a more complex architecture for the CNN and having more epochs would also improve the quality of the results and be beneficial in providing higher accuracies across all tests.

6 References

References

7 types of activation functions in neural networks: How to choose? (n.d.). https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

Advances in computer vision. (2020). *Advances in Intelligent Systems and Computing*. https://doi.org/10.1007/978-3-030-17795-9

An, S. (2017). Convolutional neural networks. https://www.cc.gatech.edu/~san37/post/dlhc-cnn/

Bansal, P. (2019). Intel image classification. https://www.kaggle.com/puneet6060/intel-image-classification

Bonner, A. (2019). The complete beginner's guide to deep learning: Convolutional neural networks. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

Chassagnon, G., Vakalopoulou, M., Battistella, E., Christodoulidis, S., Hoang-Thi, T.-N., Dangeard, S., Deutsch, E., Andre, F., Guillo, E., Halm, N., Hajj, S. E., Bompard, F., Neveu, S., Hani, C., Saab, I., Campredon, A., Koulakian, H., Bennani, S., Freche, G., … Paragios, N. (2020). Ai-driven ct-based quantification, staging and short-term outcome prediction of covid-19 pneumonia.

Chm. (2018). Logo detection using pytorch. https://mc.ai/logo-detection-using-pytorch/

Convolutional neural network (cnn) tensorflow core. (n.d.). https://www.tensorflow.org/tutorials/images/cnn

Convolutional neural networks for visual recognition. (n.d.). https://cs231n.github.io/convolutional-networks/#norm

Deshpande, A. (n.d.). A beginner's guide to understanding convolutional neural networks. https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Garbade, M. J. (2018). Clearing the confusion: Ai vs machine learning vs deep learning differences. https://towardsdatascience.com/clearing-the-confusion-ai-vs-machine-learning-vs-deep-learning-differences-fce69b21d5eb

Hyperparameters: Optimization methods and real world model management. (n.d.). https://missinglink.ai/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/

Jeffheaton. (n.d.). Jeffheaton/t81_558_deep_learning. https://github.com/jeffheaton/t81_558_deep_learning

Org, K. A. I. D. (n.d.). https://chortle.ccsu.edu/VectorLessons/vmch13/vmch13_2.html

Poletaev, I., Pervunin, K., & Tokarev, M. (2016). Artificial neural network for bubbles pattern recognition on the images. *Journal of Physics Conference Series*, *754*, (072002)–13. https://doi.org/10.1088/1742-6596/754/7/072002

Qayyum, A., Anwar, S. M., Majid, M., Awais, M., & Alnowami, M. R. (2017). Medical image analysis using convolutional neural networks: A review. *CoRR*, *abs/1709.02250*arXiv 1709.02250. http://arxiv.org/abs/1709.02250

Raycad. (2017). Convolutional neural network (cnn). https://medium.com/@raycad.seedotech/convolutional-neural-network-cnn-8d1908c010ab

Sahoo, S. (2018). Deciding optimal filter size for cnns. https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

Tesla vehicle safety report. (2020). https://www.tesla.com/VehicleSafetyReport

Thomas, C., & Miap. (2019). An introduction to convolutional neural networks. https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7

Vincee. (2020). Intel image classification (cnn - keras). https://www.kaggle.com/vincee/intel-image-classification-cnn-keras

7 Appendices

7.1 Tables and Calculations

Time (s)	**(1, 1)**	**(2, 2)**	**(3, 3)**	**(4, 4)**
TanH	141.85	166.03	186.61	217.89
Sigmoid	138.9	163.95	184.85	218.22
ReLU	132.37	153.92	167.3	215.76
Leaky ReLU	147.55	171.11	191.63	230.72

((Accuracy)/(Time (s)))	**(1, 1)**	**(2, 2)**	**(3, 3)**	**(4, 4)**
TanH	0.4988838209	0.4587524303	0.4008359681	0.3364082794
Sigmoid	0.1259899208	0.1067398597	0.09467135515	0.32047168
ReLU	0.5527435975	0.5052408394	0.4775851763	0.3666110493
Leaky ReLU	0.4951993494	0.4461067734	0.4204282732	0.340239251

7.2 CNN Code

This code below was adapted from Jeff Heaton's machine learning tutorials on GitHub, the CNN tutorial on the TensorFlow documentation website, and from user vincee for the Intel Image Classification challenge on the Kaggle website ("Convolutional Neural Network (CNN) TensorFlow Core", n.d.; Jeffheaton, n.d.; Vincee, 2020).

[ ]:
```python
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import cv2
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tqdm import tqdm
import time
```

[ ]:
```python
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)
```

[ ]:
```python
class_names = ['mountain', 'street', 'glacier', 'buildings', 'sea', 'forest']
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}
number_of_classes = len(class_names)
img_size = (150, 150)
```

[ ]:
```python
def load_data():
    datasets = ['D:/Downloads/IIC/seg_train', 'D:/Downloads/IIC/seg_test']
    output = []

    for dataset in datasets:
        images = []
        labels = []
        print("Loading {}".format(dataset))
        for folder in os.listdir(dataset):
            label = class_names_label[folder]
            for file in tqdm(os.listdir(os.path.join(dataset, folder))):
                img_path = os.path.join(os.path.join(dataset, folder), file)
                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, img_size)
                images.append(image)
                labels.append(label)
        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')
        output.append((images, labels))
    return output
```

[ ]:
```python
(train_images, train_labels), (test_images, test_labels) = load_data()
```

[ ]:
```python
n_train = train_labels.shape[0]
n_test = test_labels.shape[0]
```

```
print ("Number of training examples: {}".format(n_train))
print ("Number of testing examples: {}".format(n_test))
```

[ ]:
```
train_images, train_labels = shuffle(train_images, train_labels,␣
 ↪random_state=242)
```

[ ]:
```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

[ ]:
```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

[ ]:
```
model = models.Sequential()
model.add(layers.Conv2D(64, (x, x), strides=4, activation='function',␣
 ↪input_shape=(150, 150, 3))) # the x's are replaced with the kernel size, and␣
 ↪the word 'function' is replaced with the activation function for each␣
 ↪combination
model.add(layers.MaxPooling2D((2, 2), strides=2))
model.add(layers.Conv2D(64, (x, x), activation='function')) # the x's are␣
 ↪replaced with the kernel size, and the word 'function' is replaced with the␣
 ↪activation function for each combination
model.add(layers.MaxPooling2D((2, 2), strides=2))
```

[ ]:
```
model.summary()
```

[ ]:
```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(6))
```

[ ]:
```
model.summary()
start_time = time.time()
```

[ ]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(train_images[:], train_labels[:], epochs=10,
```

2

```
                    validation_data=(test_images, test_labels))
```

[ ]:
```
end_time = time.time()
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

[ ]:
```
time_elapsed=end_time-start_time
print(test_acc)
print(f"Time elapsed: {hms_string(time_elapsed)}")
```

[ ]:
```

```

3