Title:

Convolutional Neural Network

Research Question:

To what extent is AlexNet better at classifying dog and cat images in terms of training accuracy and validation accuracy than LeNet-5?

Subject:

Computer Science

Word Count:

3996

Table of Contents

Introduction

Image classification has broad applications across many fields, from medical diagnostics to retail in e-commerce. As the areas that can utilize machine learning image classification models continue to grow, it is important to evaluate which models are most effective and why. Convolutional Neural Networks (CNNs) are the most developed models commonly used for image classification, and the architecture of these models determines the quality of their predictions (geeksforgeeks, “Convolutional Neural Network (CNN) in Machine Learning”). Therefore, this paper will evaluate which CNN architecture provides better classification results.

I have chosen to examine LeNet-5 and AlexNet. LeNet-5, one of the earliest convolutional neural networks, was developed in 1998 and is credited with laying the foundation for modern CNNs (geeksforgeeks, “What Is LeNet?”). Due to its age, LeNet-5 features a simpler architecture compared to more recent models. Its simplicity makes it effective in scenarios involving small datasets, which are prone to overfitting, and in situations with limited computational resources. Tasks such as digit recognition using the MNIST dataset, which contains digits from 0 to 9 and was employed during the development of LeNet-5, and optical character recognition (OCR), used to identify characters and numbers in scanned documents, are well-suited for LeNet-5 due to its efficient handling of datasets with simple structures and low memory requirements. On the other hand, AlexNet, introduced in 2012, represents a significant advancement in deep learning for image classification (Tang et al.). It gained prominence for its performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), demonstrating the power of deeper and more complex CNN architectures (Tang et al.). Its more intricate structure allows it to detect finer details in images, such as texture, color, and resolution. This makes AlexNet suitable for larger and more complex datasets, such as the X-ray medical images.

By comparing LeNet-5 with AlexNet, this investigation will explore the trade-offs between simpler and more complex CNN architectures. Specifically, I will examine their ability to classify cat and dog images by analyzing training and validation accuracy. Although neither model was originally designed for this task, evaluating their performance on this dataset will offer insights into their adaptability to different inputs. The LeNet-5 and AlexNet will be developed, and using 25,000 cat and dog images, the models' training and validation accuracy will be recorded and compared to address the research question: "To what extent is AlexNet better at classifying cat and dog images than LeNet-5 in terms of training and validation accuracy?"

Background Research

Machine Learning

Machine Learning (ML) is a type of artificial intelligence (AI) where computers learn from data and improve their performance over time, similar to how humans learn from experience (IBM, “What Is Machine Learning?”).

Neural Network

A neural network (NN) is a type of machine learning program that allows models to make decisions similar to human biological neurons. It consists of nodes arranged in layers including, the input layer receiving the initial data; the hidden layer processing the given data; and the output layer producing the final result. Each node is connected to others and has its own weight, bias, and threshold (IBM, “What Are Neural Networks?”).

Weights and biases are randomly assigned to nodes during the training. Then, a linear combination of the inputs from the previous layer is computed by multiplying the inputs by their corresponding weights and adding a bias term. After this, the activation function is used to introduce non-linearity. The non-linearity allows the network to capture more complex patterns in the data. The threshold determines if a node activates and sends data to the next layer; if the node’s weight surpasses the threshold, it activates; otherwise, no data is passed along. Higher weights indicate more important information, so this process can only pass the important information (Lark Editorial Team).

This continues for every hidden layer, and the output layers produce the result. The activation function is also used in the output layer to calculate the predicted output, and the type of activation function depends on the classification that the model has to do. Finally, the loss is calculated by calculating the difference between the predicted output and the true target value. These processes are called forward propagation as the neural network passes the

input data through the network to produce an output (geeksforgeeks, “What Is Forward Propagation in Neural Networks?”).

After the forward propagation, backpropagation happens to reduce the error of the prediction. The gradient of the loss is calculated with respect to each weight using the chain rule. The calculated value indicates how much weight needs to change in order to reduce the loss. The partial derivatives of the loss with respect to the weights and biases are calculated, allowing the network to measure how each weight contributes to the error (geeksforgeeks, “Backpropagation in Neural Network”).

With these values, the weights are adjusted in a way that can reduce the error with the optimizers. The optimizers control how the model updates weights after backpropagation by considering the speed of the learning. They modify the weights so they can produce the most accurate predictions by reducing the loss efficiency (EITCA Academy).

The learning rate is a parameter in neural networks that determines the size of the steps taken toward minimizing the loss in the direction of the gradient. A learning rate that is too small can slow down the learning process, while one that is too high can cause the model to overshoot the optimal weights, failing to converge. It is used with the optimizer to ensure the network gradually improves its predictions (Raitoharju).

Deep Learning

Deep learning (DL) is a branch of machine learning that simulates the complex decision-making abilities of the human brain through multi-layered neural networks, or deep neural networks. Most of the AI we encounter in our daily lives is driven by deep learning in some capacity (IBM, “What Is Deep Learning?”).

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of deep learning algorithm designed for image identification. It consists of several layers inspired by the human brain's visual processing, making it effective at recognizing spatial connections and hierarchical patterns in images (geeksforgeeks, "Convolutional Neural Network (CNN) in Machine Learning").

Convolutional Layer

The most important layer is the convolutional layer which is responsible for feature extraction from input images. These layers apply convolutional operations using filters, or kernels, to identify and highlight various patterns within the data (geeksforgeeks, "Convolutional Neural Network (CNN) in Machine Learning").

A kernel, a small matrix of weights, moves across the image in a process called convolution. As it slides over different regions of the image, it analyzes sections known as receptive fields. The filter's size defines these receptive fields, determining the area of the image being processed. At each position, the filter computes a dot product between its weights and the image pixels in the receptive field. This computation results in a single value, which is placed in the output array to create a feature map, highlighting the detected features across the image (IBM, "What Are Convolutional Neural Networks? | IBM"). Figure 1 represents the calculation process of the feature map using the input layer and kernel.

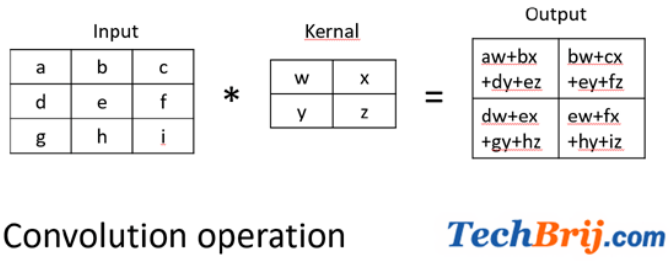

Fig. 1. Mohan, Brij. "TensorFlow 2: Convolutional Neural Networks (CNN) and Image Classification." *TechBrij*, 8 Jan. 2020.

The stride, or the number of pixels the filter shifts during each convolution step, affects the size of the feature map. A smaller stride produces a more detailed map, while a larger stride decreases the map size and computational effort (IBM, "What Are Convolutional Neural Networks? | IBM").

Padding adjusts image dimensions in CNNs. Zero-padding adds zeros around the image to fit the filter and preserve spatial dimensions such as edge information. Valid padding applies no extra pixels, potentially reducing output size. The same padding adds pixels to keep output dimensions the same as the input. Full padding increases output size by adding zeros around the image (IBM, "What Are Convolutional Neural Networks? | IBM").

After convolution, an activation function is applied to the feature map. This introduces non-linearity, allowing the network to capture complex patterns and relationships. For example, the most common activation function is ReLU (Rectified Linear Unit), which returns 0 for negative input values and the input value itself for positive inputs (Machine Learning in Plain English). Figure 2 is the visual representation of the ReLU function.

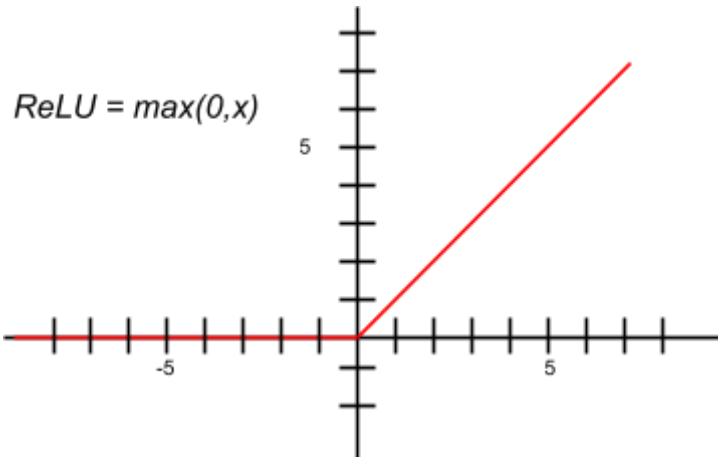

Fig. 2. Created by the candidate.

By converting negative values to 0, ReLU helps the network focus on important features, making it more efficient and reducing the risk of overfitting. This abstraction allows the network to generalize better across a broader range of images. Without this step, the network would only perform linear transformations, limiting its pattern recognition capabilities (Machine Learning in Plain English).

CNNs use multiple convolutional layers to detect features at different levels. Initial layers identify basic features like edges, mid-level layers combine these to form complex patterns, and deeper layers recognize high-level structures and objects. This hierarchy improves the network's ability to understand and classify intricate details (IBM, “What Are Convolutional Neural Networks? | IBM”).

Pooling Layer

The pooling layer, or downsampling layer, reduces dimension by minimizing the number of parameters in the input. Unlike convolutional layers, pooling layers use filters without weights; instead, they apply aggregation functions to values within each receptive field. Common types of pooling include max pooling, which selects the maximum value from each region, and average pooling, which calculates the average value. Although pooling

results in some loss of information, it simplifies the network, enhances efficiency, and helps prevent overfitting (IBM, “What Are Convolutional Neural Networks? | IBM”).

Fully-connected Layer

The fully connected layer connects every neuron in the layer to every neuron in the previous layer. This layer performs classification by applying matrix multiplication between its weights and the activations from the previous layer, adding biases to the result. Figure 3 is the representation of matrix multiplication with N elements in the input and T elements in the output.

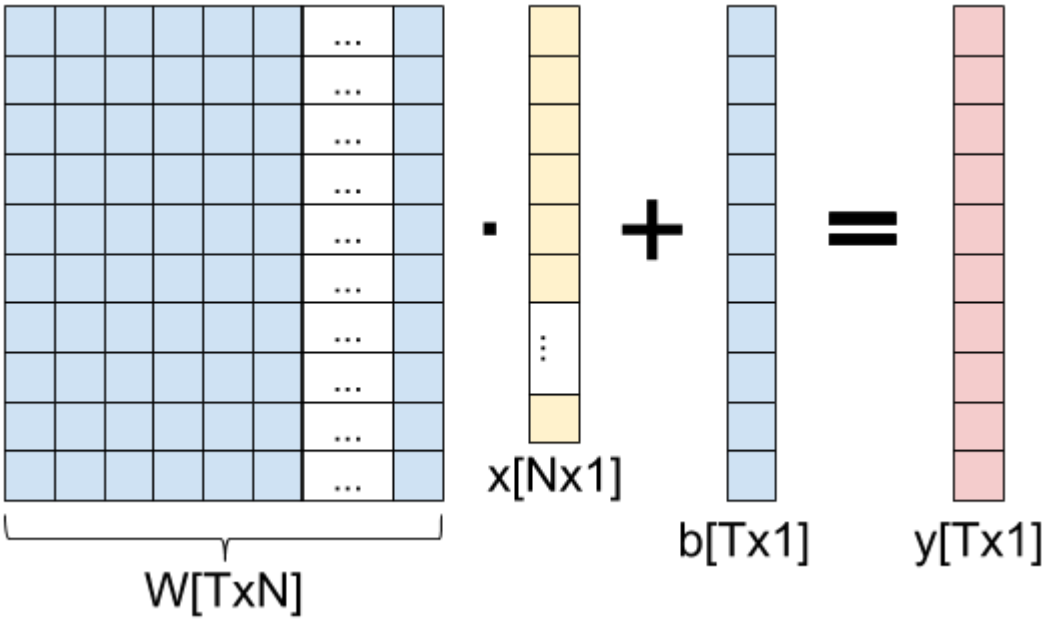

Fig. 3. Bendersky, Eli. “Backpropagation through a Fully-Connected Layer” *Thegreenplace.net*, 31 May 2018.

The final output is then passed through an activation function, which converts the result into a probability distribution. This is different from the activation function used to introduce non-linearity during the training process. This process enables the network to make

classification decisions based on the features extracted by the preceding layers (IBM, “What Are Convolutional Neural Networks? | IBM”).

LeNet-5

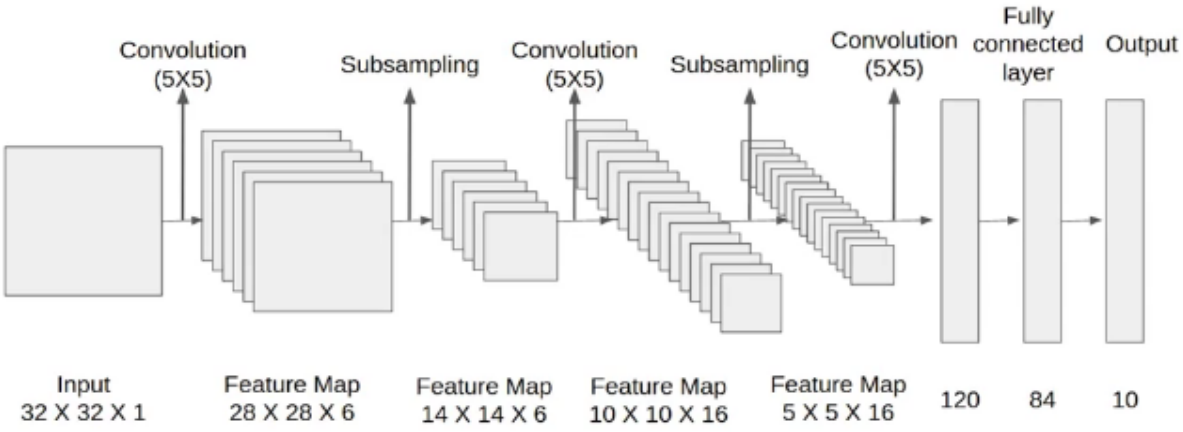

Fig. 4. Saxena, Shipra. “Lenet-5 | Lenet-5 Architecture | Introduction to Lenet-5.” *Analytics Vidhya*, 18 Mar. 2021.

LeNet-5 is one of the earliest CNN models, meaning that it has a simple architecture. It consists of 7 layers, excluding the input layer. The information on each layer can be found in Table 1.

Table 1 Structure of the LeNet-5

Layer	# of kernels	kernel size	stride	Size of feature map	Activation function
Input				32 x 32 x 1	
Conv 1	6	5 x 5	14	28 x 28 x 6	tanh
Avg Pool 1		2 x 2	2	14 x 14 x 6	
Conv 2	16	5 x 5	1	10 x 10 x 16	tanh
Avg Pool 2		2 x 2	2	5 x 5 x 16	
Conv 3	120	5 x 5	1	120	tanh
Fully Connected 1				84	tanh
Fully Connected 2				10	Sigmoid

Source: Saxena, Shipra. “Lenet-5 | Lenet-5 Architecture | Introduction to Lenet-5.” *Analytics Vidhya*, 18 Mar. 2021, www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/. Accessed 16 Sept. 2024.

AlexNet

AlexNet is CNN that won the Imagenet large-scale visual recognition challenge in 2012. Because of its increase in depth compared to LeNet-5, it is renowned for its contribution to the advancement of CNN architectures. The information on each layer can be found in Table 2.

Table 2 Structure of the AlexNet

Layer	# of kernels	kernel size	stride	padding	Size of feature map	Activation function
Input					227 x 227 x 3	
Conv 1	96	11 x 11	4		55 x 55 x 96	ReLU
Max Pool 1		3 x 3	2		27 x 27 x 96	
Conv 2	256	5 x 5	1	2	27 x 27 x 96	ReLU
Max Pool 2		3 x 3	2		13 x 13 x 256	
Conv 3	384	3 x 3	1	1	13 x 13 x 256	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 384	ReLU
Max Pool 3		3 x 3	2		6 x 6 x 256	
Dropout 1	Rate = 0.5				6 x 6 x 256	
Fully Connected 1					4096	ReLU
Dropout 2	Rate = 0.5				4096	
Fully Connected 2					4096	ReLU
Fully Connected 3					1000	Softmax

Source: Saxena, Shipra. “Alexnet Architecture | Introduction to Architecture of Alexnet.” *Analytics Vidhya*, 19 Mar. 2021, www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/. Accessed 16 Sept. 2024.

Differences between LeNet-5 and AlexNet

Input size preprocessing

The input layer of LeNet-5 is 32x32x1 which is smaller than the input layer of AlexNet's 227x227x3. The larger physical layer size allows to capture of more complex patterns and details in the images. The larger input size is appropriate for detailed image classification tasks. This allows AlexNet to be suitable for classifying a wide range of objects (Klingler). On the other hand, considering that LeNet-5 was developed for the MNIST dataset, which consists of handwritten digits ranging from 0 to 9, it might not be well-suited for a more complex dataset that requires capturing subtle details.

Convolutional Layers

Convolutional layers exhibit hierarchical feature learning, meaning that each layer extracts different types of features from the images. Earlier layers learn low-level features, such as edges and textures, while deeper layers learn more abstract features, such as shapes and objects. Consequently, having more convolutional layers allows the network to capture finer details and higher-level representations of the data, which are crucial for distinguishing subtle differences in complex datasets (Klingler). Since AlexNet has 5 convolutional layers, it is assumed to capture more features of the images than LeNet-5, which has 3 convolutional layers. This may result in better results for AlexNet.

Activation Functions

LeNet-5's activation function is tanh function, defined as, $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, which maps input values to a range between -1 and 1. $\tanh(x)$ is represented in Figure 5.

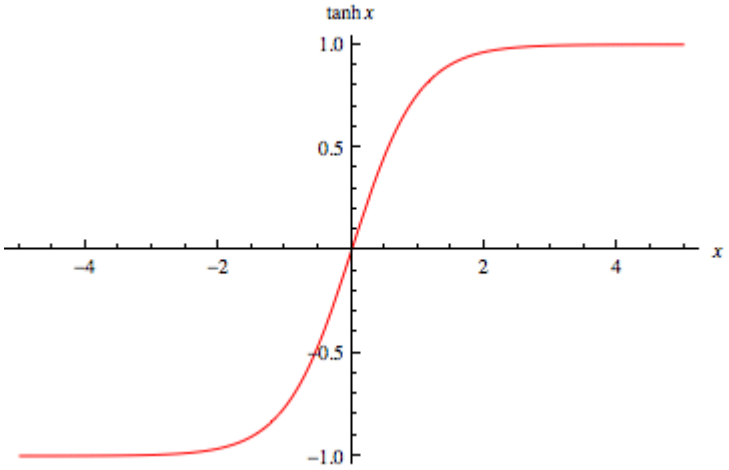

Fig. 5. Weisstein, Eric W. “Hyperbolic Tangent.” *Mathworld.wolfram.com*.

While the tanh(x) function helps center the data around zero, its gradient can become very small for large positive or negative inputs, leading to the vanishing gradient problem which happens due to a small gradient and slows the speed of learning. This issue can make it harder for the network to learn effectively, especially in deeper networks. However, in situations with a smaller number of inputs or simpler tasks, tanh(x) can still be beneficial by providing a normalized output, which can make training more stable and efficient. It is important to note that while tanh(x) can stabilize training in some contexts, it is generally slower and less effective in deep networks compared to activation functions like ReLU (EITCA Academy, “What Are the…”).

The activation function used in AlexNet is the ReLU (Rectified Linear Unit) function, defined as ReLU(x)=max(0,x) in Figure 2. ReLU outputs zero for negative inputs and a linear value for positive inputs. This helps address the vanishing gradient problem because the gradient is either zero or a constant positive value, avoiding the issue of very small gradients for positive inputs. As a result, ReLU allows models to converge faster during training by providing stronger gradients for positive inputs, which accelerates the optimization process.

Additionally, ReLU is computationally simple, involving only a comparison operation, which contributes to faster computation (EITCA Academy, “What Are the…”).

Pooling Layers

LeNet-5 utilizes average pooling which calculates the average value of each patch. This can dilute the influence of strong features since the pooling operation averages out the function, so it may lose critical information. On the other hand, AlexNext employs max pooling which selects the maximum value from each patch of the feature map. This helps preserve the most significant feature detected in that region. By retaining the maximum value, max pooling can better capture strong features, which are often crucial for distinguishing important patterns in the data (Zhao and Zhang).

Fully Connected Layers

LeNet-5, an earlier architecture, features a simpler fully connected layer setup with just a single dense layer of 84 neurons, followed by the output layer. This design is well-suited for simpler datasets like digit recognition, where computational resources and data complexity are limited (Saxena, “Lenet-5 | Lenet-5 Architecture | Introduction to Lenet-5”).

In contrast, AlexNet employs a more complex architecture with three fully connected layers, each containing 4096 neurons. To combat overfitting—where the model learns noise and outliers from the training data, impairing its generalization—AlexNet uses dropout layers. Dropout works by randomly deactivating neurons during training, preventing the network from relying too heavily on any single neuron and promoting more robust learning across the network. This design allows AlexNet to handle more complex and

high-dimensional datasets, such as ImageNet, by capturing intricate patterns and enhancing generalization (Saxena, “Alexnet Architecture | Introduction to Architecture of Alexnet”).

Testing Method

Training Accuracy and Validation Accuracy

Training accuracy accesses how well the model fits the training data. This indicates the proportion of correctly classified images out of the total number of images. During training, the model keeps making predictions image by image, and the correct number of predictions during the process is used to calculate the training accuracy. Too high training accuracy might mean overfitting, so it is important to evaluate the training accuracy with validation accuracy (Lehn).

Validation accuracy accesses the model's generalization ability on new data. During the training process, the validation accuracy is also calculated for each epoch with the validation data (Lehn).

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \times 100$$

The final training accuracy and final validation accuracy will be used to evaluate the performance of the models. However, since training accuracy and validation accuracy are calculated at each epoch, their trends can also be evaluated. Therefore, the trends will be assessed to understand their performance better.

Methodology

Environment

The experiments were conducted using Google Colab Pro+, which provides a cloud-based environment with advanced computational resources. The runtime environment utilized Python 3 and was equipped with a TPU v2 accelerator. The TPU v2, designed specifically for high-performance tensor computations, significantly enhanced the efficiency of both training and inference processes (Google Cloud). The Colab Pro+ platform, based on a Linux environment, facilitated the use of Google Drive for seamless data management and integration. TensorFlow framework, compatible with TPU v2, was employed to leverage the computational power of the hardware (Google Cloud). This setup provided a robust environment for executing complex models and ensured rapid processing capabilities essential for deep learning tasks.

Data Set

It is important to have balanced data, meaning that the number of images for Category 1 and Category 2 should be nearly equal to ensure the accuracy of the model. Imbalanced data can cause overfitting when a CNN is exposed to only certain categories, leading it to memorize features rather than learn general patterns. This results in a model that performs well on training data but poorly on new, unseen data because it has memorized the training data, including its noise and outliers, rather than learning the underlying patterns. Therefore, balancing the two categories is essential.

A dataset from Microsoft is used to train the model. It contains 25,000 images of dogs and cats. Randomly, 4,001 cat images and 4,006 dog images were selected and used as training data. A different set of 1,012 cat images and 1,013 dog images are randomly selected

and used as validation data. The dataset is structured to maintain a balance between the two data types: cats and dogs. The example images can be referred to Figure 6 and Figure 7.

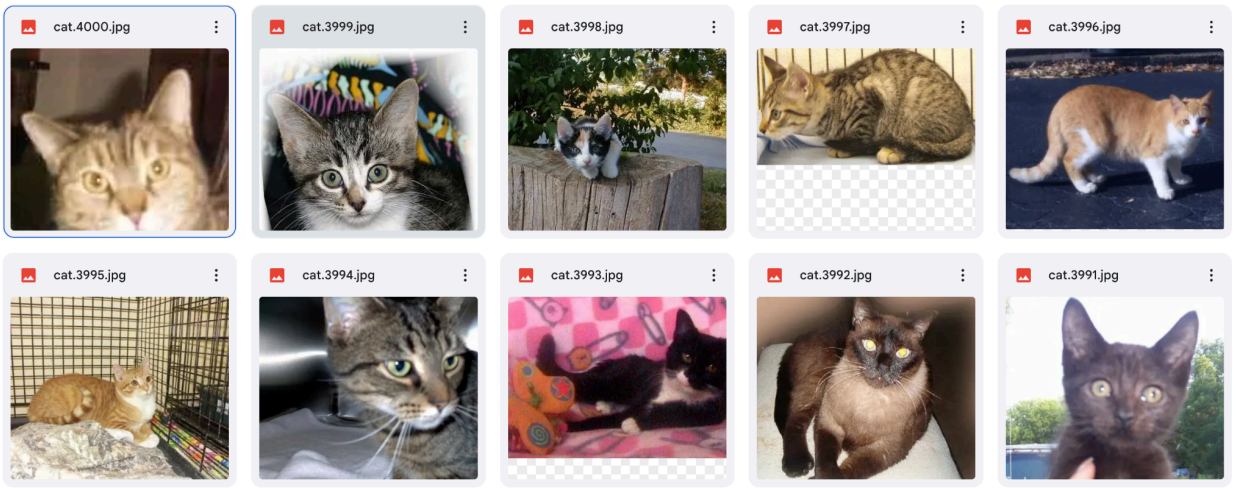

Fig. 6. Microsoft. "Kaggle Cats and Dogs Dataset." *Microsoft Download Center*, 5 Sept. 2022.

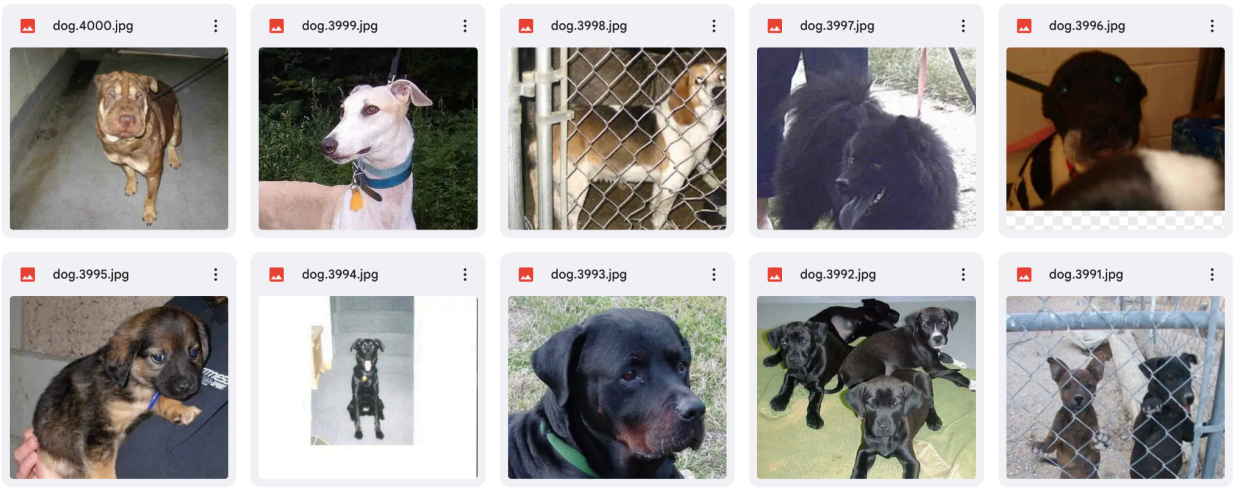

Fig. 7. Microsoft. "Kaggle Cats and Dogs Dataset." *Microsoft Download Center*, 5 Sept. 2022.

Models

Keras API in TensorFlow is used to build CNN models. To build the model, images are preprocessed, and the images are used to train the model.

While preprocessing the images to train the model, the ImageDataGenerator class from Keras is used to generate batches of image data. This approach helps manage data size by loading and processing only a small subset of the data at a time. Additionally, it facilitates

data augmentation, which enhances the diversity and robustness of the dataset by applying transformations such as rotation and shifting.

The rescale parameter normalizes pixel values to the range [0,1], which helps achieve numerical stability and faster convergence during training. Parameters such as rotation_range, width_shift_range, shear_range, zoom_range, and horizontal_flip are used to diversify the dataset by applying transformations to each image. This allows the model to learn more generalized patterns and helps prevent overfitting.

For the test data, only normalization is applied as the augmentation is not typically performed on test data to ensure consistent evaluation.

Then, flow_from_directory method loads images from a training folder and applies the specified data augmentation. The image size is adjusted to 32x32, which matches the input layer size of the LeNet-5 model. A batch size of 32 is used because it balances memory usage and provides stable model updates. This batch size is often selected to optimize training speed and make effective use of hardware acceleration. Since the input layer will be 32x32x1, the color_mode is set to 'grayscale' to ensure it has only one channel. Given that the model needs to classify two categories – cats and dogs – the class mode is set to binary.

The model is built using Keras's library. Convolutional layers, average pooling layers, flatten layers, and dense layers of Keras models are used when building the LeNet-5 model.

LeNet-5 is developed, mimicking the general architecture described in Figure 5. When convolutional layers are needed, Conv2D is used to process the images. For average pooling layers, AveragePooling2D is utilized. Since flatten layer is necessary before the fully connected layer, the flatten function is applied. For the fully connected layer, the dense layer is used, as it represents the fully connected layer in Keras. For the last layer, the output layer is set to 1 unit, which is different from the original LeNet-5 architecture. This design choice is made to use the Sigmoid activation function, which requires the output layer to be set as 1.

The model.compile method is used at the end to set up the model for training. The optimizer with learning rate, loss function, and metrics must be chosen. For the optimizer, the Stochastic Gradient Descent (SGD) optimizer is used in the original LeNet-5 function. However, this LeNet-5 model utilizes the Adam optimizer due to time constraints. The Adam optimizer is known for faster training during the early stages compared to the SGD optimizer. Since AlexNet also utilizes SGD, Adam optimizer is used for AlexNet as well to keep it consistent. For the loss function, binary cross-entropy is used because the model will be used in binary classification. The matrix is set as accuracy because our testing method is to compare the model's accuracy.

With the function model.fit in the Keras, the LeNet-5 is trained. The images that are processed in train_generator are used to train the model, and the test_generator is used the validate the data. The model is trained with 20 epochs, which is sufficient training.

Similar methods are used for the AlexNet – the images are preprocessed, and the images are used to train the model.

The image is preprocessed according to the input size of AlexNet. The attributes of the ImageDataGenerator are the same as those used for LeNet-5 to maintain consistency. However, for both train_generator and test_generator, the target_size is changed to (227,227) to match the original AlexNet. categorical_mode is used for both train_generator and test_generator. Even though AlexNet is also used to classify two categories, to use the softmax activation function, it has to be a categorical_mode which is used for the multi-classification.

In the build_alexnet function, AlexNet is developed with the same general architecture as the original model. When the max pooling layer has to be used, the MaxPooling2D function is used. Then, the dropout function is used for the dropout.

The same optimizer and metrics as LeNet-5 are used to keep it consistent. For the loss function, categorical_crossentropy is chosen because of the usage of the softmax as the activation function, which requires categorical_crossentropy.

When the model is built and compiled, input_shape is set to (227,227,3), as defined by the general architecture of AlexNet. Since it incorporates the softmax function, num_classes is set to 2 for binary classification. To control both models, the number of epochs is also kept at 20.

Experiment Result

Table 3 Results of the experiment, including the final training accuracy and final validation accuracy for both LeNet-5 and AlexNet

	LeNet-5	AlexNet
Final Training Accuracy	0.6488	0.8602
Final Validation Accuracy	0.6451	0.8443

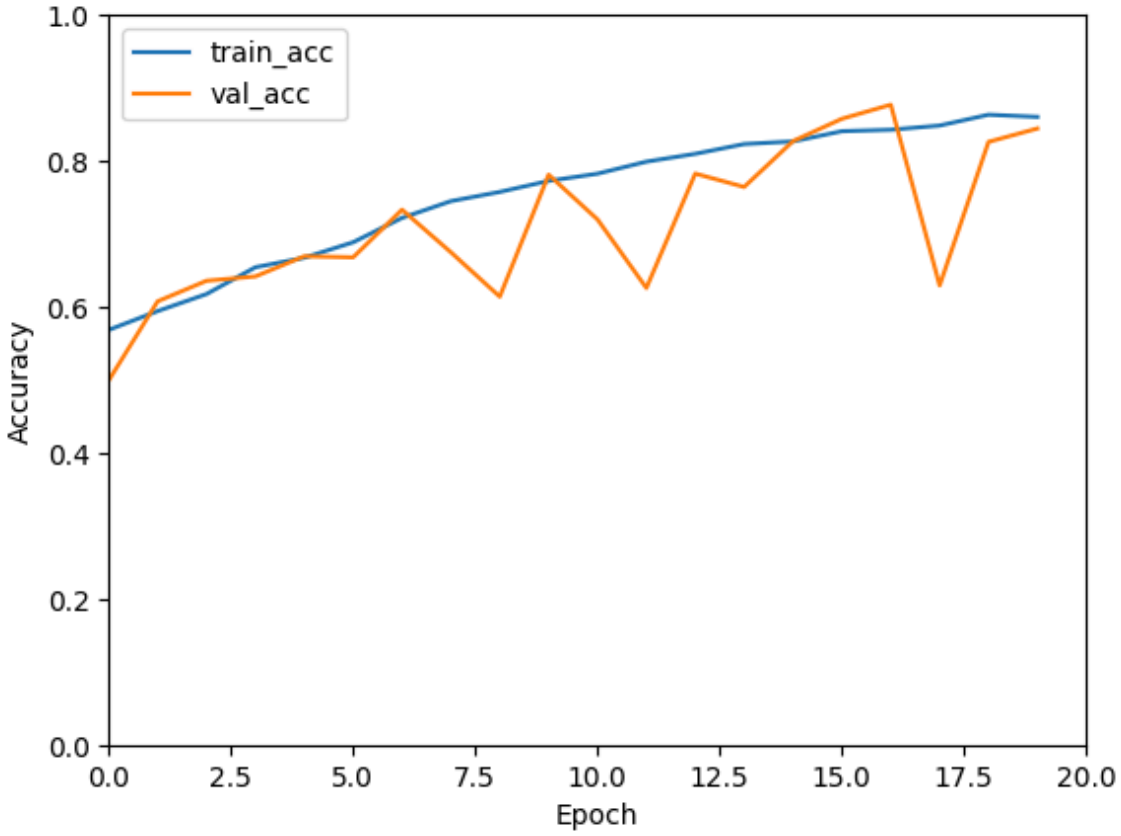

Fig. 7. Created by the candidate

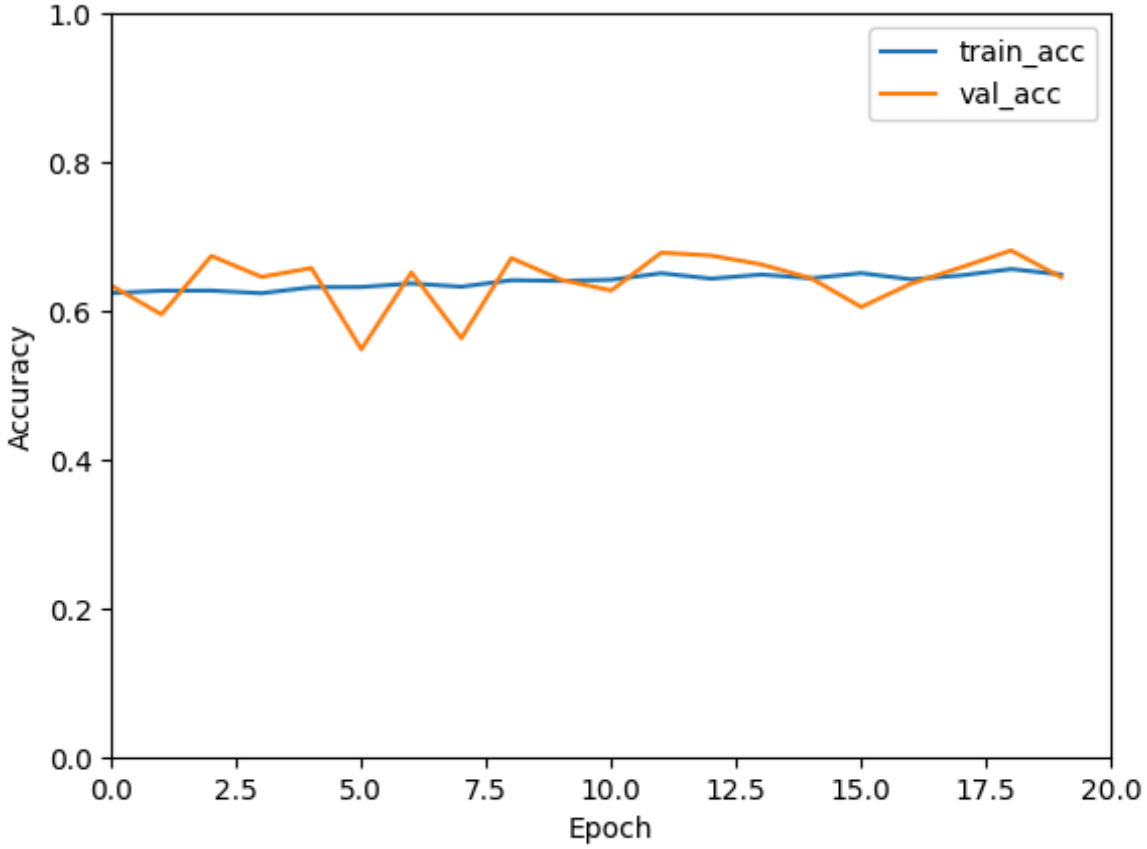

Fig. 8. Created by the candidate

Result Discussion

LeNet-5's lower training and validation accuracy compared to AlexNet highlights its limitations in classifying cat and dog images. AlexNet's accuracy improves over time, indicating effective learning as the epochs progress, whereas LeNet-5 shows little improvement, suggesting its architecture is too simple for this task. With accuracy hovering around 0.5, LeNet-5 struggles to classify the images effectively.

Input Size

AlexNet processes larger RGB images (227x227), allowing it to capture more intricate features, while LeNet-5 handles smaller grayscale images (32x32). LeNet-5 was originally designed for simpler datasets, such as digit classification, and its smaller input size is insufficient to handle the complexity of images like cats and dogs.

Layer Structure

AlexNet has a deeper architecture with more convolutional layers, enabling it to extract complex and abstract features. Its use of max pooling helps retain important details, while LeNet-5’s average pooling tends to lose crucial high-level information. Additionally, AlexNet’s two fully connected layers, with a higher number of nodes, allow for better generalization. In contrast, LeNet-5’s single fully connected layer limits its ability to learn detailed features.

Activation Function

AlexNet's use of ReLU prevents the vanishing gradient problem, speeding up learning. Meanwhile, LeNet-5 relies on the Tanh function, which can slow training due to smaller gradients, further hindering its performance.

Conclusion

After building LeNet-5 and AlexNet and classifying cat and dog images through both models, it is clear that AlexNet is better at classifying the images which is supported by its high training and validation accuracy. This is because AlexNet utilizes a larger input size to introduce various characteristics, deeper and better layers to capture more complex features, and better activation functions which enhances the rate of learning. Through this process, utilizing recent models allows for more accurate results. By enhancing the model, it seems to be possible to classify more complicated data such as chest X-rays to determine the presence of diseases.

Evaluation

This experiment has several limitations. Due to time constraints, I could not replicate the exact LeNet-5 and AlexNet architectures, particularly regarding the optimizer. I used the Adam optimizer for faster training, while the original models used stochastic gradient descent (SGD), which might have provided more accurate results.

The dataset size, though large at 25,000 images, is smaller than what modern CNNs typically require for optimal performance. A larger dataset would likely have improved both training and validation accuracy. Additionally, I only trained the models for 20 epochs, while CNNs usually benefit from training over 100 epochs to fully converge. Longer training could have produced more generalizable results.

Lastly, relying solely on training and validation accuracy gives a limited view of performance. Metrics like precision, recall, and F1 score would offer deeper insights, particularly in distinguishing between similar classes like cats and dogs. Including these metrics would have provided a more comprehensive evaluation of the models.

To achieve more accurate results, I would address these limitations by increasing the dataset size, training for more epochs, and incorporating metrics like precision and F1 score.

Works Cited

Bendersky, Eli. “Backpropagation through a Fully-Connected Layer - Eli Bendersky’s Website.” *Thegreenplace.net*, 31 May 2018, eli.thegreenplace.net/2018/backpropagation-through-a-fully-connected-layer/. Accessed 16 Sept. 2024.

EITCA Academy. “What Are the Key Differences between Activation Functions such as Sigmoid, Tanh, and ReLU, and How Do They Impact the Performance and Training of Neural Networks?” *EITCA Academy*, 21 May 2024, eitca.org/artificial-intelligence/eitc-ai-adl-advanced-deep-learning/neural-networks/neural-networks-foundations/examination-review-neural-networks-foundations/what-are-the-key-differences-between-activation-functions-such-as-sigmoid-tanh-and-relu-and-how-do-they-impact-the-performance-and-training-of-neural-networks/. Accessed 16 Sept. 2024.

---. “What Is the Role of the Optimizer in Training a Neural Network Model? - EITCA Academy.” *EITCA Academy*, 13 Aug. 2023, eitca.org/artificial-intelligence/eitc-ai-dlpp-deep-learning-with-python-and-pytorch/neural-network/training-model-neural-network/examination-review-training-model-neural-network/what-is-the-role-of-the-optimizer-in-training-a-neural-network-model/. Accessed 16 Sept. 2024.

geeksforgeeks. “Backpropagation in Neural Network.” *GeeksforGeeks*, 4 Mar. 2024, www.geeksforgeeks.org/backpropagation-in-neural-network/. Accessed 16 Sept. 2024.

---. “Convolutional Neural Network (CNN) in Machine Learning.” *GeeksforGeeks*, 25 Dec. 2020,

www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/. Accessed 16 Sept. 2024.

---. “What Is Forward Propagation in Neural Networks?” *GeeksforGeeks*, GeeksforGeeks, May 2024, www.geeksforgeeks.org/what-is-forward-propagation-in-neural-networks/. Accessed 16 Sept. 2024.

---. “What Is LeNet?” *GeeksforGeeks*, 24 May 2024, www.geeksforgeeks.org/what-is-lenet/. Accessed 16 Sept. 2024.

Google Cloud. “Introduction to Cloud TPU.” *Google Cloud*, cloud.google.com/tpu/docs/intro-to-tpu. Accessed 16 Sept. 2024.

IBM. “What Are Convolutional Neural Networks? | IBM.” *Www.ibm.com*, IBM, www.ibm.com/topics/convolutional-neural-networks. Accessed 16 Sept. 2024.

---. “What Are Neural Networks?” *Www.ibm.com*, IBM, 2023, www.ibm.com/topics/neural-networks. Accessed 16 Sept. 2024.

---. “What Is Deep Learning?” *Www.ibm.com*, IBM, 2023, www.ibm.com/topics/deep-learning. Accessed 16 Sept. 2024.

---. “What Is Machine Learning?” *IBM*, 2023, www.ibm.com/topics/machine-learning. Accessed 16 Sept. 2024.

Klingler, Nico. “AlexNet: A Revolutionary Deep Learning Architecture.” *Viso.ai*, 29 Apr. 2024, viso.ai/deep-learning/alexnet/. Accessed 16 Sept. 2024.

Lark Editorial Team. “Forward Propagation.” *Larksuite.com*, 26 Dec. 2023, www.larksuite.com/en_us/topics/ai-glossary/forward-propagation. Accessed 16 Sept. 2024.

Lehn, Frederik. “Interpreting Training/Validation Accuracy and Loss.” *Medium*, 8 Nov. 2023, medium.com/@frederik.vl/interpreting-training-validation-accuracy-and-loss-cf16f0d5329f. Accessed 16 Sept. 2024.

Machine Learning in Plain English. “Convolutional Neural Network — Lesson 9: Activation Functions in CNNs.” *Medium*, 21 June 2023, medium.com/@nerdjock/convolutional-neural-network-lesson-9-activation-functions-in-cnns-57def9c6e759. Accessed 16 Sept. 2024.

Mohan, Brij. “TensorFlow 2: Convolutional Neural Networks (CNN) and Image Classification.” *TechBrij*, 8 Jan. 2020, techbrij.com/tensorflow-cnn-image-classification. Accessed 16 Sept. 2024.

Raitoharju, Jenni. *Chapter 3 - Convolutional Neural Networks*. Edited by Alexandros Iosifidis and Anastasios Tefas, Academic Press, 2022, pp. 35–69, https://doi.org/10.1016/B978-0-32-385787-1.00008-7. Accessed 16 Sept. 2024.

Saxena, Shipra. “Alexnet Architecture | Introduction to Architecture of Alexnet.” *Analytics Vidhya*, 19 Mar. 2021, www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/. Accessed 16 Sept. 2024.

---. “Lenet-5 | Lenet-5 Architecture | Introduction to Lenet-5.” *Analytics Vidhya*, 18 Mar. 2021, www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/. Accessed 16 Sept. 2024.

Tang, Wenhao, et al. “Review of AlexNet for Medical Image Classification.” *EAI Endorsed Trans. E Learn.*, vol. 9, 2023, api.semanticscholar.org/CorpusID:265213036. Accessed 16 Sept. 2024.

Weisstein, Eric W. “Hyperbolic Tangent.” *Mathworld.wolfram.com*, mathworld.wolfram.com/HyperbolicTangent.html.

Zhao, Lei, and Zhonglin Zhang. “A Improved Pooling Method for Convolutional Neural Networks.” *Scientific Reports*, vol. 14, no. 1, Jan. 2024, p. 1589, https://doi.org/10.1038/s41598-024-51258-6. Accessed 16 Sept. 2024.

Appendix

1. LeNet

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten,
Dense, Dropout, Input, BatchNormalization
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

test_dir = '/content/drive/My Drive/CD/test_set/'
train_dir =  '/content/drive/My Drive/CD/training_set/'

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(32, 32),
    batch_size=32,
    color_mode='grayscale',
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(32, 32),
    batch_size=32,
    color_mode='grayscale',
    class_mode='binary'
)
```

```
model = Sequential([
    Input(shape=(32,32,1)),


    Conv2D(6, (5, 5), activation='tanh', padding='same', strides=(1,
1)),
    AveragePooling2D((2, 2), strides=(2, 2)),
    BatchNormalization(),

    Conv2D(16, (5, 5), activation='tanh', strides=(1, 1)),
    AveragePooling2D((2, 2), strides=(2, 2)),
    BatchNormalization(),
```

```
    Conv2D(120, (5, 5), activation='tanh', strides=(1, 1)),

    Flatten(),

    Dense(84, activation='tanh'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=test_generator
)
```

2. AlexNet

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout, Input, BatchNormalization
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

test_dir = '/content/drive/My Drive/CD/test_set/'
train_dir =  '/content/drive/My Drive/CD/training_set/'

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(227, 227),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(227, 227),
    batch_size=32,
    class_mode='categorical'
)
```

```
model = Sequential([
    Input(shape=(227,227,3)),

    Conv2D(96, (11, 11), strides=(4, 4), activation='relu',
input_shape=(227, 227, 3)),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    BatchNormalization(),

    Conv2D(256, (5, 5), strides=(1, 1), activation='relu',
padding='same'),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    BatchNormalization(),

    Conv2D(384, (3, 3), strides=(1, 1), activation='relu',
padding='same'),

    Conv2D(384, (3, 3), strides=(1, 1), activation='relu',
padding='same'),

    Conv2D(256, (3, 3), strides=(1, 1), activation='relu',
padding='same'),
    MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    BatchNormalization(),

    Flatten(),

    Dense(4096, activation='relu'),
    Dropout(0.5),

    Dense(4096, activation='relu'),
    Dropout(0.5),

    Dense(2, activation='softmax')
])

model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=test_generator
)
```