1 PREDICTING DERIVATIVE NFT IMAGES USING CONVOLUTIONAL NEURAL NETWORK WITH THE DENSENET201 MODEL

[1]Rhama Andyka, [2]Yonathan Purbo Santosa
[1,2]Program Studi Teknik Informatika Fakultas Ilmu Komputer,
Universitas Katolik Soegijapranata
[2]yonathansantosa@unika.ac.id

ABSTRACT

Derivative NFTs are modified versions of the original NFTs that have been altered or obtained through additional processing. This modification process may include changes in the color, appearance, composition, or content of an existing digital asset. Penelitian ini bertujuan untuk mengembangkan algoritma prediksi untuk mengklasifikasikan derivatif NFT (Non-Fungible Token) menggunakan teknik deep learning. In this context, the developed algorithm uses the DenseNet-201 architecture and involves steps such as data comprehension, data preparation, image augmentation, and the use of callbacks to stop model training when it reaches the desired level of accuracy. This study uses NFT-derived datasets collected by the researchers themselves, because there is no source that provides a large number of NFT datasets. Through experiments conducted, it is known that the use of DenseNet-201 architecture with a target size of 50x50 or 150x150 can produce a good level of accuracy, reaching 86-99%. The experimental results show that the implemented DenseNet-201 model is capable of classifying NFT derivatives with a good level of accuracy. The use of data augmentation and adjustment of certain hyperparameters also affects the improvement of model accuracy. In addition, analysis and visualization of the results were carried out using a confusion matrix to evaluate the performance of the model in classifying each NFT derived class.

Keywords: Machine Learning, Convolution Neural Network, Transfer learning, DenseNet201 Architecture.

1.1 CHAPTER 1 INTRODUCTION

1.1.1 Background

With the advancement of technology, artworks are no longer in physical form but today there are also digital forms. Starting from 3D models, illustrations, character drawings, and others - others. Since the emergence of the Non-Fungible Token or commonly known as NFT in 2014[1], it has now been able to transform the views of art collectors and digital artists to develop it wider. Basically, NFT is a digital artwork that is printed on the blockchain network to give the identity or label of the encrypted work that the work is owned by someone and cannot be counterfeited. Some of the NFT images on the marketplace usually have their own avatars and art styles, but many also have some communities that mimic the avatar or are known as NFT derivatives so that they can spoil the collector as well as harm NFT itself. This is because people tend to prefer to imitate existing images so that their images are easier to highlight from the collector. In this project, the authors discuss how to detect NFT derivatives using Convolutional Neural Network (CNN).

In creating this NFT derivative detection program, algorithms and datasets are needed. Convolutional Neural Network with transfer learning DenseNet201 model is an algorithm that can solve this problem. CNN is a type of simulated neural network designed to process image data efficiently. CNN uses convolutionary filters to detect important features in the image and then use them to make predictions. The data set used by the author is 3000 NFT images that have been made by themselves. The 3000 images are divided into 3 types, each containing 1000 images that will be inserted into the system repeatedly to get accurate information and get the maximum prediction.

Thus, using Convolutional Neural Network (CNN) with transfer learning DenseNet201 model, the authors could predict NFT images with high accuracy. Using DenseNet201 model, the publisher can also find out which parameters are suitable to produce predictions with high accuracy. The success rate of the algorithm used will be described in more detail in chapter 3.

1.1.2 Problem Formulation

From this background, the author can formulate problems that will be answered in the research process and discuss in the final part, which questions will be discussed as follows:

1. How does DenseNet201 perform in predicting derivative NFT images?
2. What hyperparameter and learning algorithm is suitable for solving prediction derivative NFT image on this project?

1.1.3 Scope

There are also limitations of problems that are necessary in the creation of this program. So as not to go beyond the objectives of this project, among them are:

1. Uses the NFT image type with a total of 3000 images from the author with a size of 480x480 pixels as a dataset for the prediction process.
2. On this project focused on NFT derivative prediction using DenseNet201.

1.1.4 Objective

To classify an image, the human eye can immediately judge whether the image has something in common or not, imitating other images or not. The purpose of making this NFT derivative prediction program is in addition to knowing a NFT said derivative or not also to know how DenseNet201 model works in the process of predicting NFT images.

1.1.5 Chapter 2 Literature Study

In the preparation of this project there are also several journals that the author uses as references. Since no one has yet researched about the prediction of NFT images using CNN then with the presence of several of these journals, the author can explain and analyze how CNN works in the image predictions of projects already made by several sources so that this project to be made has the same scope and also to avoid plagiarism. Some of these journals will be described in narrative form, among them:

Norman Meuschke[2], this book discusses about how to introduce an image-based plagiarism detection approach that adapts itself to forms of image similarity found in academic work. The approach is adaptable because it includes techniques for analyzing heterogeneous image features, employs analysis techniques only when they are appropriate for the input image, employs a flexible method for identifying suspicious image similarities, and makes it simple to incorporate new analysis techniques in the future. The research intends to build an effective detection strategy capable of recognizing a wider subset of potentially suspicious image similarities and to derive requirements for the approach by looking at photographs in the VroniPlag collection. The AlexNet design is used in the deep convolutional neural network (CNN) used in this study. It has three completely linked layers, a softmax output layer, and five convolutional layers. The CNN is trained using the Caffe framework, an open-source deep learning system developed by the Berkeley Vision and Learning Center. The Berkeley Vision and Learning Center's open-source deep learning Caffe framework is used to train the CNN. This study uses the stochastic gradient descent (SGD) optimizer, a well-liked optimization technique for deep neural network training. The SGD optimizer modifies the network's weights in the direction of the loss function's negative gradient with respect to the weights. In this project, the learning rate—which determines the step size of the weight updates—is set to 0.01. A total of 50,000 iterations and a batch size of 128 are used to train the CNN. The training data is augmented by randomly cropping and flipping the images to increase the size of the training set and reduce overfitting. The CNN achieves an accuracy of 92% for photographs and 100% for bar charts, as manually checked by the authors. The training data in this project consists of images that are labeled according to their suitability for being analyzed using different analysis methods. The images are extracted from a set of 196 academic works containing alleged instances of plagiarism, which is the VroniPlag collection. The images are

classified into three categories: photographs, bar charts, and other image types. The CNN is trained using the Caffe framework, which is an open-source deep learning framework developed by the Berkeley Vision and Learning Center. The training data is augmented by randomly cropping and flipping the images to increase the size of the training set and reduce overfitting. The CNN is trained using a batch size of 128 and a total of 50,000 iterations. The learning rate, which controls the step size of the weight updates, is set to 0.01. The CNN achieves an accuracy of 92% for photographs and 100% for bar charts, as manually checked by the authors.

Praveen Krishnan[3], this book discusses about how to develop a system for matching handwritten document images. The system employs a convolutional neural network (CNN) named HWNet, which is trained using a multinomial logistic regression loss function on a dataset of handwritten words (iiit-hws). The HWNet design consists of two fully connected layers with 2048 neurons each, five convolutional layers with 64, 128, 256, 512, and 512 square filters, and a final fully connected layer with a dimension equal to the number of classes (10K in this example). After each weight layer until the last one, rectified linear units are used as the non-linear activation units, and max pooling is utilized. This is followed by the first, second, and fourth convolutional layers. The system also uses transfer learning from synthetic domain (iiit-hws) to real world setting using popular handwritten labeled corpora such as iam and gw. The training data is rendered using 100 randomly sampled fonts with varying kerning level, stroke width, and mean foreground and background pixel distributions. The system also performs Gaussian filtering to smooth the final rendered image and learns a case insensitive model for each word category by performing three types of rendering: all letters capitalized, all letters lower, and only the first letter in caps. The purpose of the project is to enhance and improve the performance of word spotting in the handwritten domain, which plays an important role in matching similar documents. The results of this project show that the proposed HW-DocSim system, which uses a combination of CNN-based feature extraction and a matching algorithm with locality constraints, outperforms existing state-of-the-art methods for detecting plagiarism in handwritten documents. The system achieves an nDCG score of 0.8569 and an AUC of 0.9465 on the HW-DocSim dataset, which contains 1000 handwritten pages from more than 100 students. The system also performs well on the HW-1K dataset, which contains nearly 1K handwritten pages from more than 100 students. The authors conclude that the proposed system can be used as an effective tool for detecting plagiarism in handwritten documents, which is an important task in many fields such as education, law, and journalism. They also suggest that future work can focus on improving the system's performance on documents with graphics and mathematical expressions.

Srikar Appalaraju[4], this book discusses about how to develop a content-based image similarity system using deep learning models. The project aims to explore effective and faster ways to train such models using curriculum learning. The dataset used in this project is from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012. The project uses a custom multi-scale CNN architecture, where three different CNNs are employed instead of using one CNN and sharing lower layers. The CNNs are trained using a curriculum learning methodology, which introduces easier training examples first and gradually increases the difficulty level of the

examples. The CNNs are designed using a contrastive loss function, and the models are evaluated in terms of accuracy. The results of this project show that the proposed content-based image similarity system using deep learning models with curriculum learning and a custom multi-scale CNN architecture outperforms the baseline model (pre-trained VGG16 CNN) in terms of accuracy. The joint embedding from all three CNNs has a 4096 embedding, and the effect of embedding dimension on the final performance was also studied. The project found that for shallower CNN architectures, 512 and 1024 embedding space were sufficient, while for deep CNN, 4096 embedding was used. The project concludes that curriculum learning is an effective and faster way to train deep learning models for content-based image similarity. The custom multi-scale CNN architecture and the use of a contrastive loss function also contribute to the improved performance of the proposed system. The project also highlights the importance of detecting and preventing model overfitting, testing with a clean test set, and debugging to better understand the mistakes the model is making.

Eman Al-Thwaib[5], this book discusses about how to develop an Arabic plagiarism detection system and to create a corpus dedicated to plagiarism detection that is authentic, big, versatile, and richly annotated. The JUPlag corpus was designed to compile academic texts for the purpose of training and testing the Arabic plagiarism detection system that is to be developed. The corpus was also intended to function as a test bed for the evaluation of plagiarism detection techniques. The dataset of this project is the JUPlag corpus, which is a collection of academic texts compiled from 2,312 dissertations defended by postgraduate students at the University of Jordan between 2001 and 2016. The corpus was designed to be used for training and testing an Arabic plagiarism detection system and as a test bed for evaluating plagiarism detection techniques. The JUPlag corpus was used to train and test a convolutional neural network (CNN) for Arabic plagiarism detection. The CNN was trained on a subset of the corpus and tested on a separate subset. The CNN architecture consisted of three convolutional layers, each followed by a max-pooling layer, and two fully connected layers. The input to the CNN was a sequence of word embeddings, which were learned during training. The CNN was trained using the Adam optimizer and cross-entropy loss function. The results showed that the CNN was able to achieve high accuracy in detecting plagiarism in Arabic texts. The best performing CNN model achieved an accuracy of 98.5% on the test set.

Eshwar[6], this project discusses about similar fashion apparel detection and classification using computer vision techniques. The dataset used in this project is collected from the e-commerce website Myntra, and it consists of 5093 images ranging over 5 classes. The CNN is used as an efficient technique for recognizing objects in images or videos. Transfer learning is employed to generate bottleneck values using the weights from the previous pre-trained layers and the images. The final layer of the CNN is trained to identify new classes. The architecture of the CNN used in this project is the GoogLeNet architecture, which is a 22 layer deep neural network initially trained on the ImageNet dataset. The final layer of the Inception v3 GoogLeNet model is removed and a new final layer is trained to classify the apparel dataset. The training data in this project is divided into a training set and a testing set containing 80% and 20% of the total number

of images respectively. The result of this project is a system that can accurately classify different types of apparel with high accuracy and suggest similar apparel for a query image belonging to the training set.

Azahro Choirunnisa[7], discusses the prediction of cat races using CNN with the EfficientNet-B0 architecture. The background of this project is that the number of native cat breeds is only 1% compared to mixed-race cats, so a cat prediction system is needed to identify cat types. With the various types of cats, there are also several images that have patterns and the similar posture. Therefore with this project the author wants to make a cat image classifier program based on race. The method used in this project is CNN using the EfficientNet-B0 architecture and comparing the Adam and RMSProp optimizers. The dataset obtained for this project comes from kaggle and google images as many as 2700 images containing 9 different cat breeds. 2160 training data, 540 validation data and 180 test data. The training data is pre-processed using a Gaussian blur filter and then enters the augmentation process to enrich the variations of the training data. Trials on this project were carried out in 3 scenarios, namely the first to compare if using a model with data pre-processing and without pre-processing, then to compare the optimizer Adam and RMSProp with a learning rate of 0.001 then the third with a learning rate of 0.0001. The results of this prediction produce the highest accuracy of 98% using the Adam and RMSProp optimizer with a learning rate of 0.001 but overfitting still occurs. the most optimal model gets 95% accuracy and 91% validation accuracy using RMSProp with a learning rate of 0.0001 and using data pre-processing.

Hendry Fonda[8], discusses the prediction of Riau batik using convolutional neural network. with the existence of various and even similar batik motifs, therefore a classification using CNN is needed in this project. Riau Batik is known since the 18th century and was used by royal nobles. Riau Batik is made using stamps mixed with dyes then printed on the fabric. The fabric used is usually silk. The background of this project is that, compared to Javanese batik, Riau batik is very slow to be accepted by the community. In this project, CNN will conduct training and testing on Riau batik so that a collection of batik models that have been classified based on the characteristics of Riau batik can be determined so that images can be determined which are Riau batik and which are not Riau batik. This project uses tensor flow with a training process totaling 168 images with 68 images in the form of Riau batik and 100 non-Riau batik images. Iteration using 30 epochs obtained an accuracy value from the data above is 65% with loss values of 2.5% and 2.1%. Predictions using CNN produce riau batik instead of riau batik with 65% accuracy. The accuracy of 65% is due to basically many of the same motifs between Riau batik and other batik with the difference lies in the color of the cerap on Riau batik.

Song A[9], This paper discusses a new fine-grained face recognition method for similar face recognition using the attention mechanism which combines the Internal Features and External Features. The authors also show how a largescale similar face dataset can be assembled by a combination of automation and human in the loop, and divide the dataset into five grades according to different degrees of similarity. The proposed method improves the true positive rate and

recognition accuracy rate for the LFW and CASIA-WebFace database, as well as the similar face dataset (SFD). The paper also introduces the IE-CNN model, which enhances the internal and external features of the face, and proposes a step-by-step training method to train the model. In this project, a face feature enhancement model called IE-CNN based on deep CNN was proposed. The IE-CNN model enhances the internal and external features of the face. The model uses a bottom-up top-down structure for the internal design of IE-CNN, which mimics the fast feedforward and feedback attention process. A five-way parallel structure is adopted for the five local feature maps, and each branch parameter is not shared. The model also uses a step-by-step training method to train the model. The experimental results show that the proposed method improves the true positive rate and recognition accuracy rate for the LFW and CASIA-WebFace database, as well as the similar face dataset. The proposed method in this project improves the true positive rate and recognition accuracy rate for the LFW and CASIA-WebFace database, as well as the similar face dataset (SFD). The experimental results show that the proposed IE-CNN model enhances the internal and external features of the face, and the step-by-step training method to train the model is effective. The fusion of complementary features in the IE-CNN model was successful, and the proposed method performed well with five grades of similarity. The recognition accuracy rate improved by 35.84% and the true positive rate improved 15.84% for grade I, and as the picture similarity in the dataset decreases from II to V, the recognition accuracy rate improved by 18.80 – 28.44%.

Xie T[10], this book discusses about visual robot relocalization based on multi-task and image-similarity strategy. The authors found that convnet representations trained on classification problems generalize well to other tasks. The propose is, a multi-task CNN for robot relocalization, which can simultaneously perform pose regression and scene recognition. Scene recognition determines whether the input image belongs to the current scene in which the robot is located, not only reducing the error of relocalization but also making understand with what confidence it can trust the prediction. Meanwhile, the authors found that when there is a large visual difference between testing images and training images, the pose precision becomes low. Based on this, the authors present the dual-level image-similarity strategy (DLISS), which consists of two levels: initial level and iteration-level. The initial level performs feature vector clustering in the training set and feature vector acquisition in testing images. The iteration level, namely, the PSO-based image-block selection algorithm, can select the testing images which are the most similar to training images based on the initial level, enabling us to gain higher pose accuracy in testing set. The method considers both the accuracy and the robustness of relocalization, and it can operate indoors and outdoors in real time, taking at most 27 ms per frame to compute. Finally, by used the Microsoft 7Scenes dataset and the Cambridge Landmarks dataset to evaluate our method. It can obtain approximately 0.33 m and 7.51◦ accuracy on 7Scenes dataset, and get approximately 1.44 m and 4.83◦ accuracy on the Cambridge Landmarks dataset. Compared with PoseNet, our CNN reduced the average positional error by 25% and the average angular error by 27.79% on 7Scenes dataset, and reduced the average positional error by 40% and the average angular error by 28.55% on the Cambridge Landmarks dataset. We show that our multi-task CNN can localize from high-

level features and is robust to images which are not in the current scene. Furthermore, we show that our multi-task CNN gets higher accuracy of relocalization by using testing images obtained by DLISS.

Kassim[11], this paper discusses an attribute-based query & retrieval system designed for fashion products. This system addresses the problem of carrying out fashion searches by the query image and attribute manipulation, e.g. replacing long sleeve attribute of a dress to sleeveless. The authors present the attributes in two groups: (1) general attributes (category, gender etc.) and (2) special attributes (sleeve length, collar etc.).. To facilitate more specific similarity learning, clothing items are represented by their structural subcomponents or ”parts”. The parts are estimated using an unsupervised segmentation method and used inside the proposed Convolutional Neural Network (CNN) as an attention mechanism. Meaning, different parts are connected to the special attributes, e.g. sleeve part is connected with sleeve length attribute. With this mechanism, part-based triplet ranking constraint is applied to learn similarity of each special attribute independently from one another in a single network. In the end, the well-defined features are used to conduct the fashion search. Additionally, an adaptive relevance feedback module is used to personalize the fashion search process with the feature descriptions. For the experiments, a new dataset is constructed containing 101,021 images which consist of pure clothing items. Besides achieving decent retrieval results in our dataset, the experiments show that proposed technique outperforms different baselines and is able to adapt towards user’s requests. CNN’s proved themselves to be very useful in both image recognition and retrieval problems. The authors adopt the well-known CNN model AlexNet. The proposed method achieves the best performance again with 53.1% top-20 retrieval accuracy. The second best performing method is AMNET [28] and gives 45.8% top20 retrieval accuracy. Removing the part extraction from the network decreases top-20 retrieval accuracy by around 7.5% which is a huge deal. AMNET and the method with part extraction perform quite similar to each other. If the authors were to remove both part extraction and the ranking loss would result in 40.1% top-20 retrieval accuracy which is better than directly using the classic method as it gives 33.2% top-20 retrieval accuracy.

1.2 CHAPTER 3 RESEARCH METHODOLOGY

This chapter describes in detail the steps taken on this project until later in the end find results that match what is done. This research stage discusses the workings of the system developed in this project. Here are some steps to take find the right and correct results.

1.2.1 Research Process

In conducting this research, the author needs knowledge first to make a project, especially knowledge about Convolutional Neural Network (CNN) and about Generative NFT. So by knowing these two things, researchers can continue research aimed at predicting whether the NFT image is an NFT derivative.

In this project, the processes carried out are:

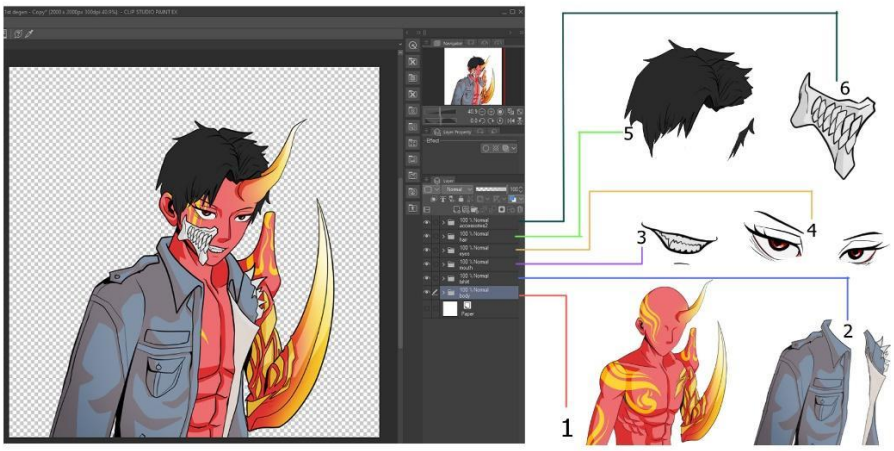
1. Formulate background, objectives, scope, and problem formulation.

2. Research articles or journals related to the project carried out.
3. Create datasets for training and testing, study the algorithms that have been used, and learn the best parameters to use.

Implementation and analysis of the results that have been carried out and then provide conclusions.

1.2.2 Collecting Datasets

To obtain the dataset, authors need thousands of NFT images for training and testing processes. From that, the author created an image generator program that was able to produce thousands of combinations of a series of layers of images created by the author. The writer creates 3 variants of avatars, each of which has a total of 1000 images and will be labeled or named for grouping. With this generator, the writer gets 3000 images as a dataset. Here are the steps – steps to generate thousands of images:



Drawing Characters

The first step is for the writer to draw a character with separate layers as in *Figure 3.1*. Like the body image drawn first, then the next layer is a picture of clothes with a position that has been determined by the author, then the next layer is the mouth, eyes, hair and accessories. In this step, the author uses illustration software, namely Clip Studio Paint with a canvas size of 480x480 pixels.

Counting the Number of Combinations

The image of one avatar consists of 6 layers including, the character's body, outfit, eyes, mouth, and accessories. Each part has several variants, namely the body consisting of 9 variants, outfit 6, mouth 7, eyes 7, hair 7, and accessories 4. With this variant, it will produce 74088 unique avatar combinations because one with another is different from the formula 9 x 6 x 7 x 7 x 7 x 4. The more variants of each category, the more unique the combination will be.

Layer Grouping

After creating a series of layers, each part must be stored in the images folder consisting of sub folders 0_body, 1_outfit, 2_eyes, 3_mouth, 4_hair, and 5_accessories folders in PNG format that have a transparent background. The function of saving images in PNG format is because the format has an alpha channel which means it can control the transparency or opacity of colors. The value can be represented as a real value, a percentage, or an integer.

Library dan Module

In building this generator there are also several libraries needed so that the program can run as expected. Some libraries include:

1. Pillow, PIL, or the Python Image Library, is the first library that gave Python the ability to work with images. PNG, TIFF, and BMP are just a few of the prominent file types that Pillow supports. Python and Pillow both offer additional decoder libraries if needed. masking, filtering, enhancing, adding text, pixel-by-pixel manipulation, and other types of modification. The first step in utilizing the pillow library is installing it using a virtual environment by entering pip install pillow. You can run this library by typing from PIL.
2. The Python programming language offers the OS module as a module to access operating system functions. Users can access files, folders, and other data kept on the operating system via this module. Additionally, this module enables users to retrieve details about the system's active processes and execute the shell from Python.
3. Random, or the random module in Python, is a module that offers methods and functions for generating random numbers. This module can be used for many different things, like generating random numbers for simulations or games and establishing strong passwords. Additionally, this module offers a function for selecting things at random from a list or order. So you can also produce random integers with a certain distribution with this module.

Script Python

In *Figure below* is the output of the generated image generate program. On making this program there are several classes made by the author so that the program can run.

This program is designed using python programming language using Visual Studio Code text editor. This script consists of 3 files, namely:

1. An executable script runs programs. This script will use the avatar_generator function of the AvatarGenerator class and pass it an argument specifying how many loops should be used to create the desired avatar.
2. The Layer class is used to access the layer subfolder's contents so that the AvatarGenerator class can later access it. This script has a random module that generates random numbers so that while the avatar generator is running, it creates avatars using random and distinct combinations (nothing is the same).
3. Class AvatarGenerator to organize the layer composition order, a looping function to combine avatars or characters into a row, a function to produce a background, and a function to store photos in a directory are all included in this class. This class comes with a cushion library that can be used to show and edit photos. The OS module is then used to modify the current operating system.

1.2.3 Data Augmentation

Image augmentation is the process of modifying an image to generate variants of the same subject in order to give the model a wider range of training examples. Due to the impossibility of precisely capturing every possible real-world scenario, augmentation is essential. By increasing the image collection, we can incorporate additional challenging-to-discover real-world scenarios and increase the training data sample size. By expanding the training data to generalize to many scenarios, the model can acquire knowledge from a broader range of events. In this study, the author randomly changes an input image's rotation, brightness, shear, horizontal flip, and scale.

This method forces the model to take into account how an image can appear in a range of scenarios, such as in the case of NFT image plagiarism.

1.2.4 Derivative NFT

NFT derivatives are NFT projects developed using intellectual property and artistic materials from already-existing projects. Derivative art NFTs frequently have titles that honor the original collections in addition to sharing a visual appearance with the original NFTs. The primary targets for derivatives are now well-known NFT collections like Bore Ape Yacht Club, CryptoPunks, and Creature World. Some derivatives are issued by other parties involved in derivatives initiative in addition to the officially issued NFT derivatives. Some of these derivatives initiatives can even combine two original NFTs. For instance, the Society of Derivative Apes (SODA) is a virtual NFT derivative that incorporates BAYC and Doodles features[12].

The rise of NFT derivatives is a result of NFT usage spreading internationally. Generally speaking, the NFT community has differing views on derivatives efforts. Others see them as a compliment to the original collections they are based on, while some view them as uninspired ripoffs of already-existing initiatives. However, the production and selling of NFT derivatives has generated considerable controversy because some contend that doing so constitutes plagiarism or a violation of intellectual property rights. Additionally, the lack of transparency and elucidation regarding the ownership and validity of NFT derivatives may worry purchasers and collectors. Investors should conduct their own research, perform due diligence, and be aware of any potential dangers before investing in any NFT collection, including derivatives[12].

NFT images are essentially derivative because they share many traits with the original image. The broad definition of a derivative is when someone replicates an existing collection (often blue-chip) and adds their own twist to it. A few examples of popular collections that provide excellent

targets for derivative collections include Bored Ape Yacht Club, CryptoPunks, and Creature World. Derivative works frequently have spin-off names that pay homage to the original collection in addition to having visually similar names.[12].

One could say that derivatives that are visible to the naked eye meet the requirements for an NFT. *Figure 3.3* illustrates the similarity between the picture characters of BAYC and BASC. Both adopt an ape-like appearance and dimensions. The BASC image is a variant of the BAYC image, except the BASC image uses a gradient background rather than a solid background, includes additional accessories that the BAYC image lacks, and solely modifies the skin tone. There is no doubt that BASC is a descendant of BAYC.

In the case of this project, the author makes 3 images with the same character so that they can be said as derivatives. The first is the "original" image where the image has a more complex color structure and what stands out the most is the shadow effect on the character. Then there is "derivative_1" where the image has a difference in terms of coloring of the character. The author made this character without any shadow effect on the character and changed the outline a bit to make it thinner, as well as changing the color of some of the accessories was also done by the author. Then in the 3rd image, namely "derivative_2", the author made the same image but added boldness to the outline and changed the color of some of the accessories on the character. If seen with the naked eye, someone will definitely judge that this image is plagiarism or derivative, therefore the author wants to prove whether a machine can distinguish this.

1.2.5 Convolutional Neural Network Algorithm

An image or image data processing-specific type of neural network architecture is called a convolutional neural network. Due to their capacity to distinguish local features in an image, such

as edges, angles, and certain shapes, which are then utilized to perform class predictions of the image, CNNs are particularly effective in pattern recognition tasks, such as image prediction.

Convolutional Neural Networks (CNNs) are the most common type of neural network architecture used in the field of image processing and pattern recognition. In *Figure above* feature learning and classification are two important aspects of CNN.

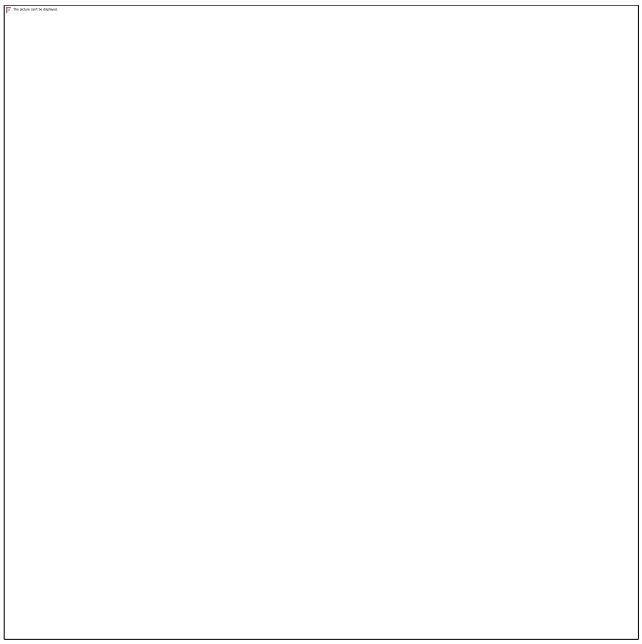
1. The process of extracting meaningful characteristics from input data, such as photography, is known as feature learning. CNN employs filters and features to extract key details from imagery as an input. Convolution, pooling, and activation functions continually apply these filters to the image, producing the features that are then processed by additional CNN layers.

2. Classification Layer, the neurons in this layer, which is made up of numerous layers, are completely interconnected with the layers above and below. This layer gets input from the feature learning section's output layer, which is then processed on a flattening layer structure with the addition of multiple hidden layers that are fully connected to create output in the form of classification accuracy for each class.

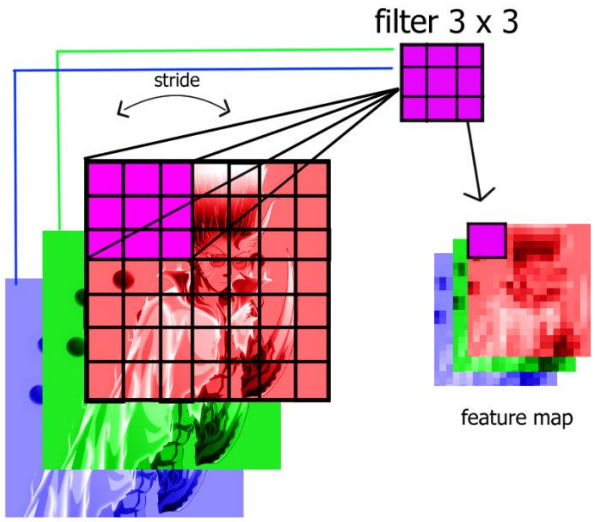
1.2.6 Convolution Layer

A unique kind of linear operation called a convolutional layer exists in a Convolutional Neural Network (CNN) at the feature learning stage. A neural network that uses convolution in place of a general matrix in at least one of its layers is known as a convolutional network. One of the mathematical processes used in image processing is convolution. In this process, feature maps of the input image are used to implement the output function. These inputs and outputs can be thought of as two valid arguments.

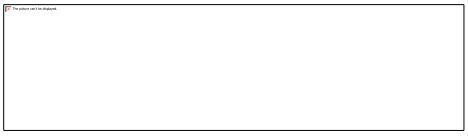
Convolutions are defined for each integral function defined above, and can be used for purposes other than taking weighted averages. The s(t) function gives a single output that is feature maps, the first argument is the input that is x and the second argument w is as the kernel or filter.



The image above is an RGB (Red, Green, Blue) image layer measuring 32x32 pixels which is actually a multidimensional array with a size of 32x32x3 (3 is the number of channels). For example, the first layer on the feature extraction layer is usually a conv. layer with a size of 5x5x3. 5 pixels long, 5 pixels high and thick or the number of 3 pieces according to the channel of the image. These three filters will be shifted to all parts of the image. Each shift will be carried out a "dot" operation between the input and value of the filter so as to produce an output or commonly referred to as an activation map or feature map.



To calculate the dimensions of the feature map there is, there are several parts that need to be known, namely stride, filter of length or height, input of length or height, zero padding. A formula can be used as below:



1.1.1.1.1.1.1.1.1 Formula to calculate The Dimension of The Feature Map

W = Input Length or Height

N = Filter Length or Height

P = Zero Padding

S = Stride

Here are some important concepts related to convolution layers on CNN:

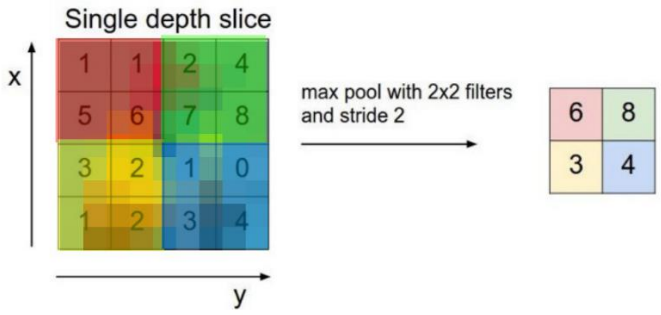
1. Filter or kernel: A matrix (often 3x3 or 5x5) used to extract features from an image is known as a filter or kernel. The image will be repeatedly subjected to the filter with the chosen strain.
2. Stride: The value of this parameter controls the amount by which the filter adjusts the input image during each convolution. A longer stride will provide less output and fewer distinguishable features.
3. Padding: Before convolution, padding is the process of adding pixels to the input image's edges. The objective is to preserve the size of the image after convolution and prevent information loss at the image's edges.

4. Activation function: This mathematical formula is used to decide whether or not a feature will be activated. One of the most often utilized activation functions in the convolution layer is the ReLU (Rectified Linear Unit) function. Through the elimination of negative values and the maintenance of positive values, this function stimulates neurons in the CNN layer.

1.2.7 Pooling Layer

A layer of functions known as a pooling layer uses feature maps as input and processes them using different statistical operations dependent on the closest pixel value. The pooling layer in the CNN model is often added on a regular basis following multiple convolutional layers. In the architecture stack of the CNN model, repeatedly inserting pooling layers between convolution layers can gradually reduce the output volume of feature maps, hence lowering the network's parameter and calculation requirements and controlling overfitting. When creating CNN models, it's crucial to select a variety of pooing layers because doing so can improve the model's performance.

The pooling layer works in each feature map stack and reduces its size. The shape of the pooling layer generally uses a filter with a size of 2x2 which is applied with a step of 2 stride and operates on each slice of the input. Here is an example image of a max-pooling operation:



In the image above, a max pooling surgical illustration, a group of boxes to be picked for their maximum are located on the left side, with the colored red, green, yellow, and blue. In order for the boxes to the right to display the procedure' outcomes. By using this method, it is guaranteed that the features will remain the same even when the image object is translated or shifted. In general, the more information that is lost and the wider the pooling region, the lower the resolution of the final image will be. However, layer pooling can hasten training time and strengthen the

model's resistance to variations in image size and rotation by reducing spatial dimensionality. In CNN, layer pooling plays an important role in strengthening the model's ability to understand important features in the image and improving the accuracy and generalizability of the model.

1.2.8 Fully Connected Layer

In the pre-classification stage of CNN, the flatten layer is a step where the output of the convolution layer and pooling layer is transformed into a one-dimensional vector, which then serves as the input for the fully connected layer. Each feature obtained from the convolution and pooling layers is displayed in this method as a multi-dimensional tensor. However, these multidimensional tensors must be transformed into one-dimensional vectors or arrays in order to be able to integrate these features as inputs to a fully connected layer. Every component of a multi-dimensional tensor that is flattened is effectively aligned or converted into a single component of a one-dimensional array. For example, if the output of a convolution layer is a tensor of size [batch_size, height, width, channels], then the flatten layer converts this tensor into a one-dimensional array of sizes [batch_size, height * width * channels]. After the flatten layer, the output vector will be forwarded to the fully connected layer for further processing in the classification or regression process. In the fully connected layer, each neuron is connected to each neuron in the previous layer, thus allowing the model to learn more complex relationships between features that have been found in previous convoluted layers.

The final layer of a convolutional neural network (CNN) design, the fully connected layer (FCL), is in charge of linking the output of the convolution and pooling layers to the output layer. Each neuron in FCL receives input from every cell in the preceding layer as it is made up of many neurons connecting to all of the neurons in the previous layer. Each neuron in the FCL receives the output from the preceding layer, which is the result of convolution and pooling, as its input. The output of each neuron in the FCL is generated using activation functions like ReLU, sigmoid,

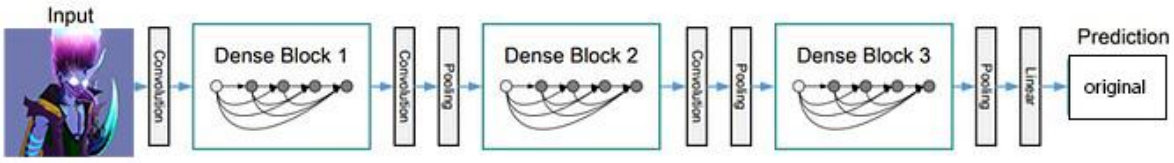
or tanh, and each neuron has a distinct weight and bias. The output of all neurons in the FCL is then combined and processed to produce the final output.

FCL is often used on image classification tasks, such as object recognition, where the output of the CNN must be categorized into a specific class. FCL can learn more complex patterns from previous output layers, thus improving the accuracy and performance of CNN models.

1.2.9 Transfer Learning of Dense Net 201 Architecture

Learning transfer is a method for helping current algorithms perform better with less data and in less time. Although this method has several advantages, there are some circumstances that need be taken into account in terms of learning transfer. Transfer learning can only be effective if the starting and target problems are sufficiently comparable for the first training to be applicable. In these circumstances, it is assumed that the source data and the target data are significantly dissimilar to one another and thus the negative transfer problem arises. The model might actually perform worse than if it hadn't been trained at all if the first round of training is too far off. Right now, there are still no clear standards on what types of training are sufficiently related, or how this should be measured.

Transfer learning is the ability to retain knowledge gained from addressing one problem and apply it to a different one later on. With transfer learning, models are created utilizing prior knowledge that demonstrate greater effectiveness and learn more quickly with less training data. The best thing about transfer learning is that only a portion of the trained model needs to be learned in order to use it. Transfer learning allows us to do so while saving time.



1.2.10 Hyperparameter Setting

A total of 3000 photos will be divide into 80% training and 20% validation portions, with all images resized to a size of 50 by 50. To train the model using NFT images, we first collect a dataset of NFT images from the above open-access resources along with their associated metadata, including file size, format, and structure. Next, the images are pre-processed by resizing them to a fixed size of 50 by 50 and standardizing their pixel values. For the model architecture, the author used Dense Net architecture. The models are trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 64 over 30 epochs, along with other hyperparameters mentioned in *Table below*.

Hyperparameter	DenseNet201 Architecture

In Activation	ReLU
Batch Size	64
Learning Rate	0.001
Training Data	80%
Validation Data	20%
Loss Function	Categorical Cross Entropy
Optimizer	Adam
Image Size	(50, 50)
Epoch	30

The hyperparameters must be set to optimal and equivalent values. In this study, we set all the models being compared to the same hyperparameters, as shown in *Table above*. ReLU activation was chosen because it is less computationally expensive and rectifies the vanishing gradient problem, which is better than other activation functions such as tanh and sigmoid. Furthermore, the default learning rate value of 0.001 was used in most Keras optimizers because it is recommended for beginners. Based on the insights from Face Net embeddings, the author selected a image size of (50, 50). This model was initially used for face clustering, verification, and identification, and provides greater precision with only 128 bytes per face. The batch size of 64 was chosen because it is appropriate for the amount of data used in the study, and using a mini-batch size that is a power of 2 is recommended.

The author chose to use the Adam optimizer, as it is a well-known deep-learning training technique that uses exponentially weighted moving averages to manage the gradient's momentum and the second moment, also known as leaky averaging. This optimizer tracks the relative prediction error of the loss function through a weighted average, making it more effective than the standard stochastic gradient descent (SGD) technique, which ignores the effects of outliers. Evaluation of the classification results of the intended architecture is performed in terms of

precision, recall, the F1-score, and classification accuracy. Among those parameters, precision is the proportion of samples with optimistic predictions concerning the total number of correct positive samples. The recall ratio of correctly predicted samples to the whole samples and the F1-score are the precision and recall weight. Finally, classification accuracy is the total correct predictions to the total number of samples.

1.3 IMPLEMENTATION AND RESULTS

1.3.1 Experiment Setup

In this project, the author uses a computer with Intel i5 Gen12 specifications, 16GB RAM, Nvidia RTX 3060 GPU, and uses the Python3 programming language on Google Colab. With these specifications can support the author in working on this project.

1.3.2 Implementation

In this chapter explains the implementation and testing of projects development about Predicting Derivative NFT Using Convolutional Neural Network Algorithm. Below is the code of the Convolutional Neural Network algorithms used to obtain results from the project developed.

The author's Colab notebook must first be mounted to the author's Google Drive account. The author must run the following code in order to accomplish this. The author will then be prompted to grant Colab access to their Google Drive account. When requested, input the permission code after adhering to the instructions. Once the Google Drive has been mounted successfully, the author can view their files from within their Colab notebook..

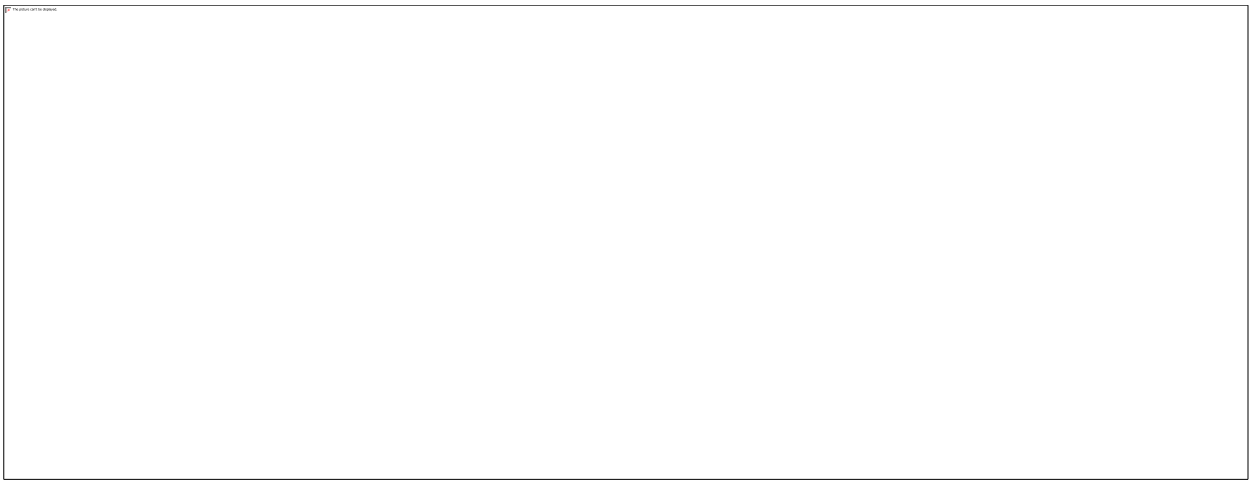
```
base_dir = '/content/drive/MyDrive/skripsi/skripsi cnn/finaldataset/bahan/'
```

```
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content"
```

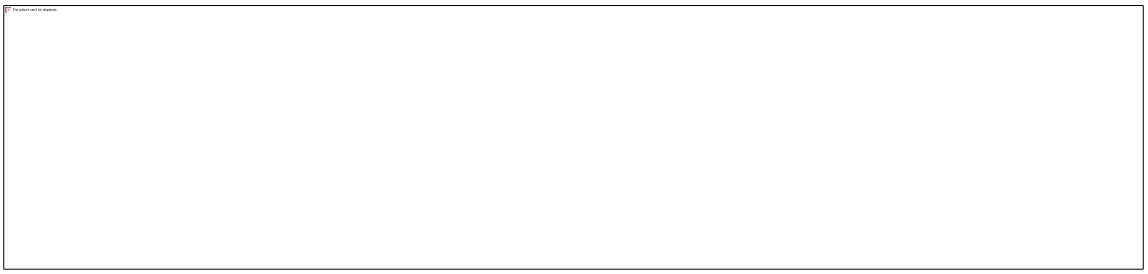
```
print(os.listdir(base_dir))
```

```
['original', 'derivative_1', 'derivative_2']
```

To understand an image dataset, the author first define the dataset directory into a variable. Then each available class or label can be known by calling *os.listdir*.

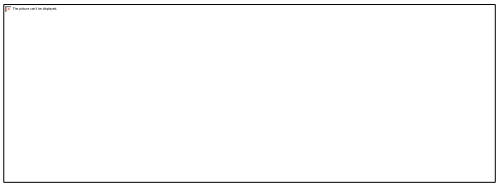


To find out the number of files present, the author use the len() function in each label directory and sum them. To facilitate reading the data can also be visualized using the help of the matplotlib library.



Import the required library according to the program created by the author. Matplotlib, numpy, and tensorflow are the main libraries and packages used in the following programs are what they do:

1. NumPy is a library for the Python programming language that adds support for sizable, multidimensional arrays and matrices as well as a substantial number of high-level mathematical functions to work with these arrays.
2. Matplotlib, a charting package for Python and its extension for numerical mathematics.
3. Tensorflow enables the creation of massively scalable neural networks. Tensorflow has aided researchers with their tasks.



Setting the seed, image size, and batch size. The number of samples that pass through the neural network before the model parameters are updated is represented by the batch size. Each set of samples undergoes a complete forward propagation and a complete backward propagation. Resize the image to have less pixels so that processing would be simpler because there will be less data to process. Random functions that generate random or random values need a seed to be initialized.

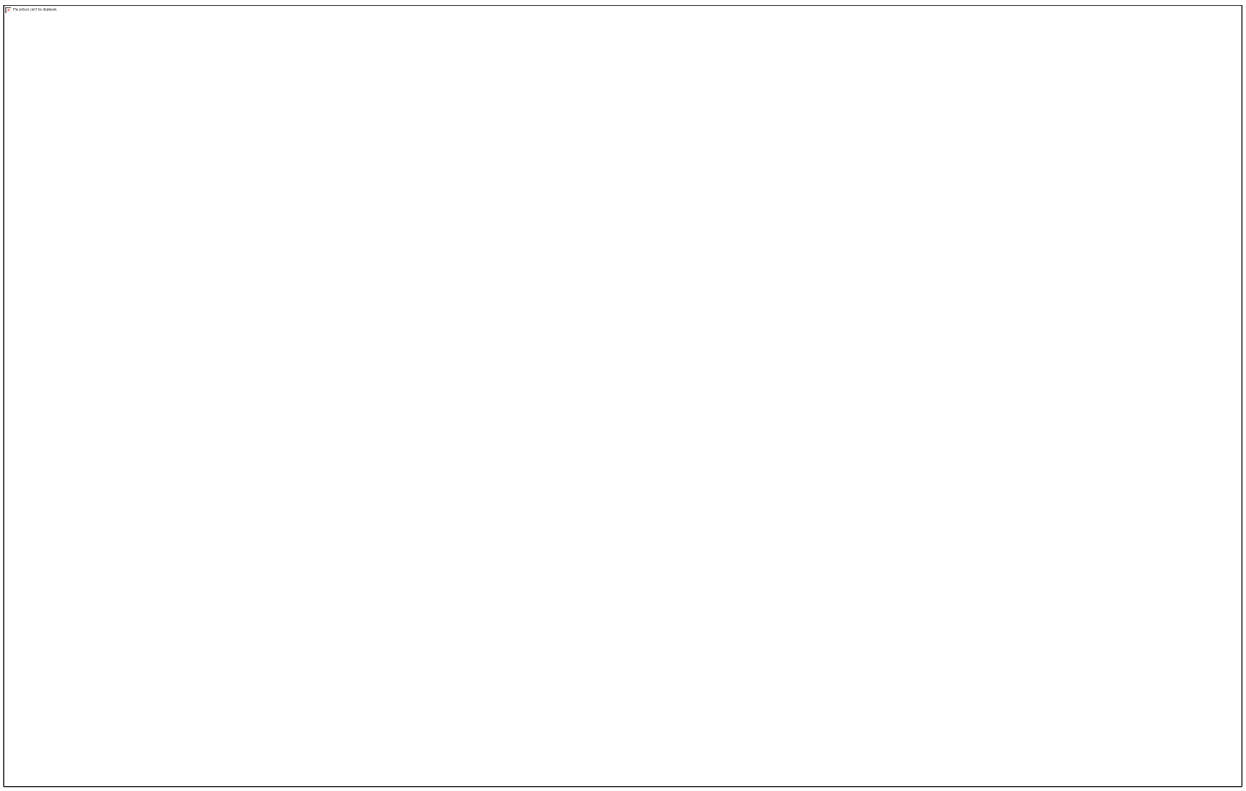
```
# Menggunakan ImageDataGenerator untuk preprocessing
import tensorflow as tf

datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    validation_split=0.2
)
```

```
# Menyiapkan data train dan data validation
train_data = datagen.flow_from_directory(
    base_dir,
    class_mode='categorical',
    subset='training',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    seed=SEED
)

valid_data = datagen.flow_from_directory(
    base_dir,
    class_mode='categorical',
    subset='validation',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    seed=SEED
)
```

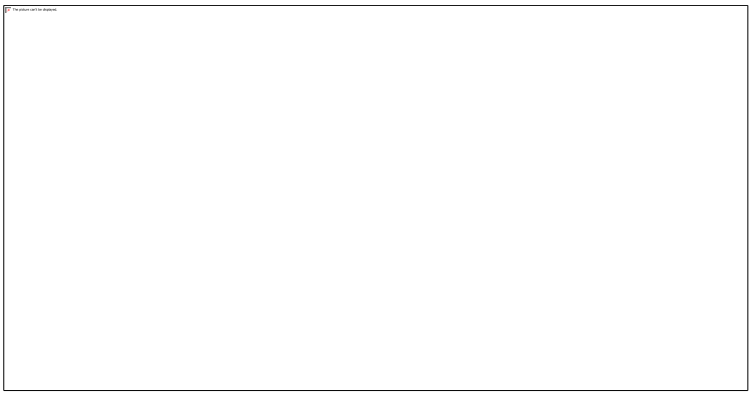
Next is to prepare the data before later going into modeling. This setup includes splitting data into training and validation data. This data sharing is needed before it is later used to train the model created and calculate the accuracy of the model.

After dividing the data, the image used as the dataset will also be applied image augmentation. This is applied based on the image data that has been displayed before. Image augmentation is done here using Random Flip, Random Rotation, Random Zoom and Rescaling layers on the image.

Using the code above, a special callback class called myCallback is defined. The Keras API for TensorFlow uses this callback to track the status of training at each epoch. In particular, it evaluates the model's accuracy after each epoch and halts training if the accuracy approaches or exceeds 99%. At the conclusion of each epoch, a built-in callback method named on_epoch_end is invoked. The training metrics (logs) and the current epoch number are supplied as inputs inside of this procedure. 99% accuracy or more was attained by the model if the accuracy value in the logs dictionary (logs.get('accuracy')) is greater than 0.99%. In that case, the message "Akurasi mencapai 99%" is printed, and the attribute **stop_training** of the model is set to True. This will stop the training process, preventing any further epochs from being executed.

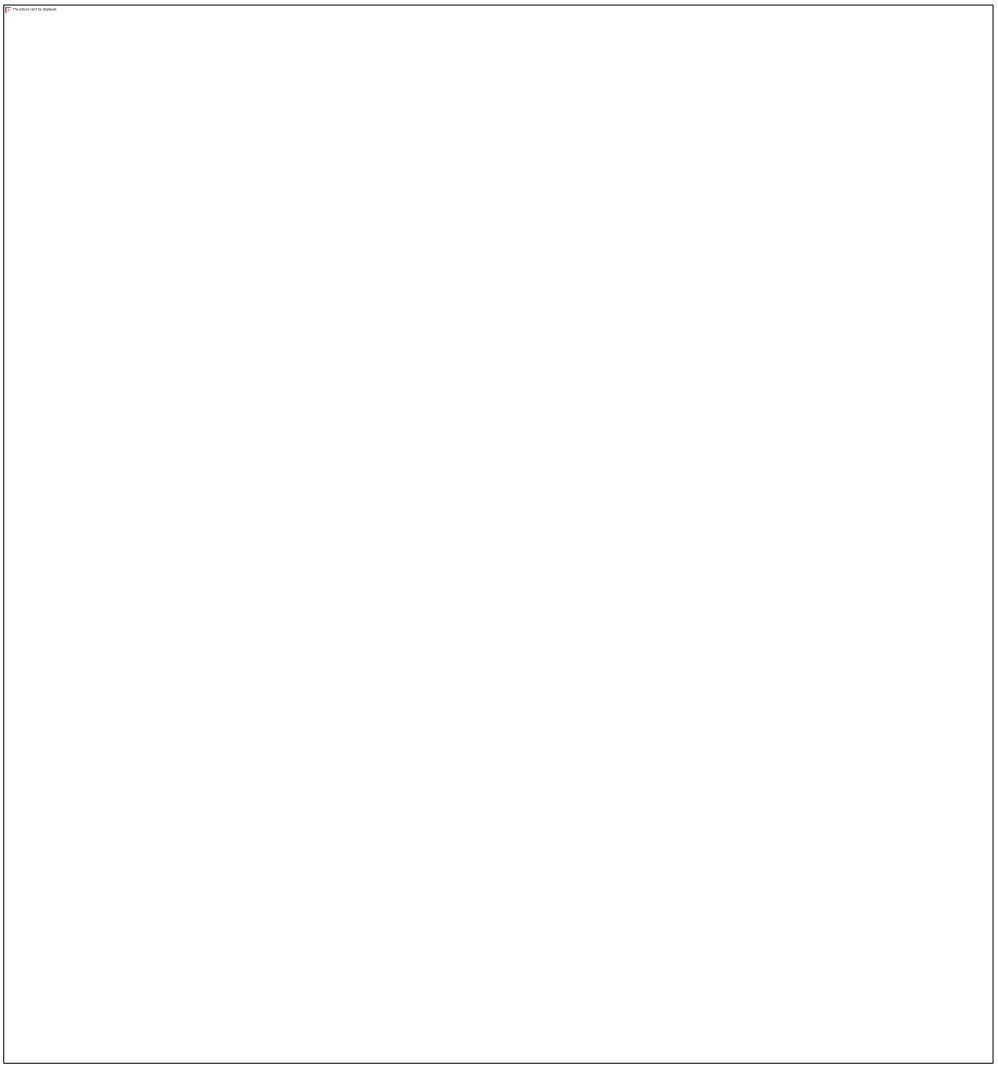


To facilitate easy learning transfer, Keras offers the Dense Net course. The DenseNet-201 class with ImageNet weights was employed by the author. The author developed his base model by adding the intentionally manufactured data, and the author rescaled the data set in accordance with the Dense Net model that was used in feature extraction. The inventor of this model stopped the weights in non-trainable layers from changing by setting the trainable property of this model to False. Otherwise, the model's learnt information would be lost. The fully-connected layer at the top of the network was left out because the author only used this model for feature extraction; instead, the author provided the input shape and pooling. The author also added his own pooling and dense layers.



The code above trains a DenseNet201 model using the **fit** method from TensorFlow's Keras API. It specifies the training data, the number of steps per epoch, the number of epochs to train

for, the validation data, and a custom callback. The **train_data** is used to train the model in 30 epochs, with 16 steps per epoch. After each epoch, the model's performance is evaluated on the **valid_data**. Additionally, a custom callback named **myCallback** is included, which stops the training if the accuracy reaches or exceeds 99%. The training history, including the loss and accuracy values for each epoch, is stored in the **densenet_hist** variable..



After everything is done with model training, the results obtained from model training are then evaluated using matplotlib where all training and validation results will be made graphs or flows that will show the accuracy and loss of functions.

The code above creates a confusion matrix to evaluate the performance of the trained DenseNet201 model. It first retrieves the true labels of a set of 100 data points by using the **test_data** generator, which is created with the **ImageDataGenerator** and points to the same directory as the training and validation data. The model's predictions are then obtained by applying the model to the **test_data** generator. The **argmax** function is used to convert the predicted probabilities into class labels. The confusion matrix is calculated using the true labels and predicted labels, and it represents the number of samples that were classified correctly and incorrectly for each class. Finally, the confusion matrix is visualized as a heatmap using the **seaborn** library, with the class labels shown on the x and y axes.

The last step of making this program is to test the model by uploading one of the images taken from the dataset and then the model that has been trained will predict whether the uploaded image is correct or not according to its respective class. From there the author can obtain prediction results whether appropriate or not with correct accuracy.

1.3.3 Results

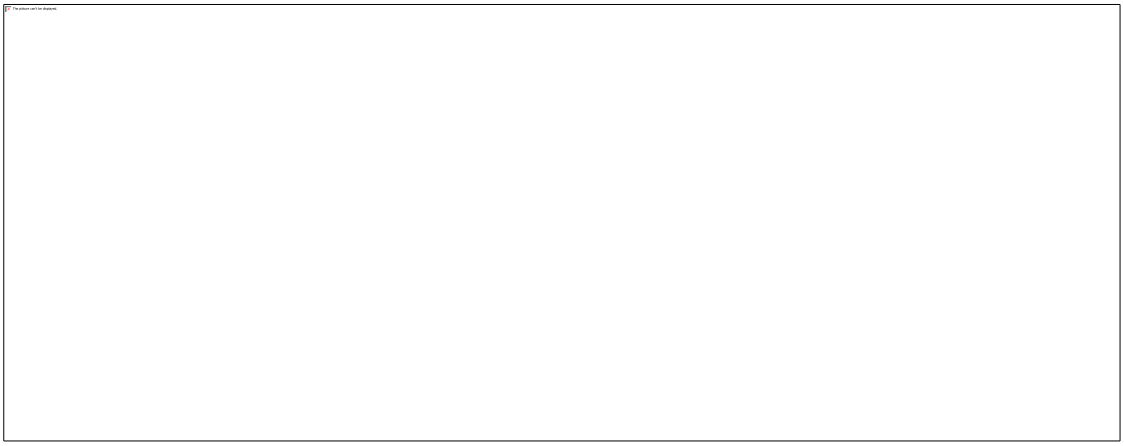
In this chapter, the author will explain the results obtained from running the program. By using transfer learning with DensNet201 Model, the author can find out the performance of the model through derivative NFT image classification.

1.3.4 Model Custom Dense Net Architecture

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 50, 50, 3)         0

 densenet201 (Functional)    (None, 1920)              18321984

 dropout (Dropout)           (None, 1920)              0

 flatten (Flatten)           (None, 1920)              0

 dense (Dense)               (None, 64)                122944

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 3)                 195

=================================================================
Total params: 18,449,283
Trainable params: 127,299
Non-trainable params: 18,321,984
_________________________________________________________________
```

The DenseNet201-based transfer learning model's architecture and parameters are described in general in the model summary. A pre-trained DenseNet201 base model, an image data augmentation layer, a dropout layer, a flatten layer, and three dense layers are among the layers that make up the model. The model's input shape is set to (50, 50, 3), which represents RGB images of 50x50 pixels. A 1920-dimensional output is generated by the base DenseNet201 model, which has a total of 18,321,984 non-trainable parameters. By randomly changing a portion of input units to 0 during training, the dropout layer aids in preventing overfitting. The output of the base model is flattened by the flatten layer into a 1D tensor. Each of the subsequent dense layers' 64 units employs the ReLU activation function, while the final dense layer's 3 units classify data using softmax activation. The model has a total of 18,449,283 parameters, 127,299 of which can be learned, while the rest parameters come from the DenseNet201 base model that has already been trained.

1.3.5 Results of Model Accuracy



1.1.1.1.1.1.1.1.1 Accuracy of DenseNet201 Model with 50x50 Target Size without Data Augmentation

Next is to train the data of the images on the dataset into the model with the fit model. In performing a fit model used epoch = 30. Epoch means the number of times the network will see the entire data set. It can be seen from the experiment conducted by the author that the model used produces a fairly high accuracy value of 86.33%. By using this hyperparameter it took 29 second to reach 1 epoch.

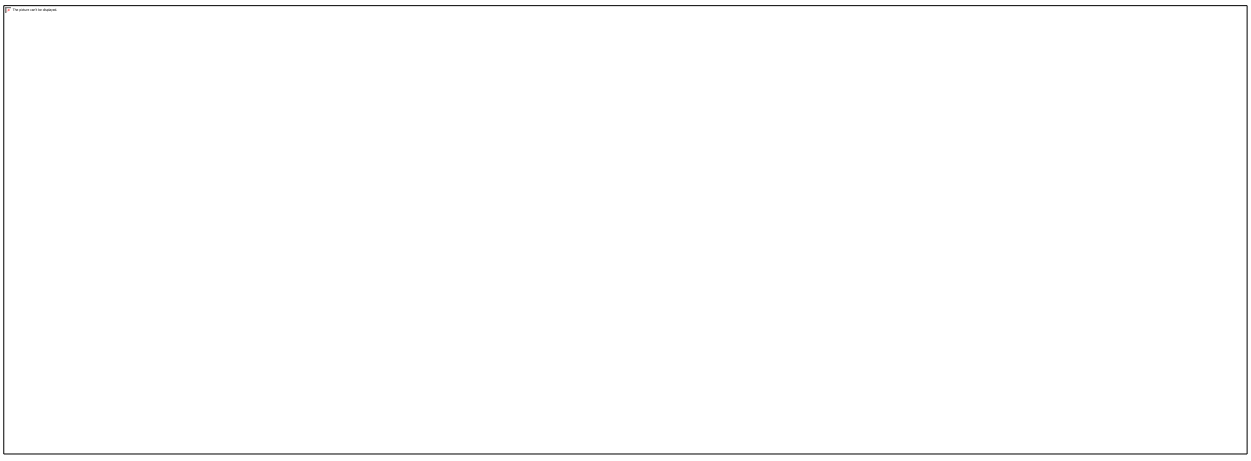
```
Epoch 20/30
16/16 [==============================] - 41s 3s/step - loss: 0.2626 - accuracy: 0.8965 - val_loss: 0.3012 - val_accuracy: 0.8917
Epoch 21/30
16/16 [==============================] - 43s 3s/step - loss: 0.2939 - accuracy: 0.8848 - val_loss: 0.2526 - val_accuracy: 0.9183
Epoch 22/30
16/16 [==============================] - 32s 2s/step - loss: 0.2494 - accuracy: 0.9083 - val_loss: 0.2616 - val_accuracy: 0.9017
Epoch 23/30
16/16 [==============================] - 42s 3s/step - loss: 0.2658 - accuracy: 0.8945 - val_loss: 0.2425 - val_accuracy: 0.9183
Epoch 24/30
16/16 [==============================] - 32s 2s/step - loss: 0.2363 - accuracy: 0.9014 - val_loss: 0.2430 - val_accuracy: 0.9217
Epoch 25/30
16/16 [==============================] - 45s 3s/step - loss: 0.2403 - accuracy: 0.9103 - val_loss: 0.2459 - val_accuracy: 0.9067
Epoch 26/30
16/16 [==============================] - 46s 3s/step - loss: 0.2468 - accuracy: 0.9073 - val_loss: 0.2864 - val_accuracy: 0.8917
Epoch 27/30
16/16 [==============================] - 42s 3s/step - loss: 0.2748 - accuracy: 0.8871 - val_loss: 0.2336 - val_accuracy: 0.9083
Epoch 28/30
16/16 [==============================] - 41s 3s/step - loss: 0.2704 - accuracy: 0.8942 - val_loss: 0.2238 - val_accuracy: 0.9217
Epoch 29/30
16/16 [==============================] - 43s 3s/step - loss: 0.2381 - accuracy: 0.9102 - val_loss: 0.2356 - val_accuracy: 0.9117
Epoch 30/30
16/16 [==============================] - 47s 3s/step - loss: 0.2066 - accuracy: 0.9180 - val_loss: 0.2964 - val_accuracy: 0.8850
```

1.1.1.1.1.1.1.1.2 Accuracy of DenseNet201 Model with 50x50 Target Size using Data Augmentation

In the second experiment the authors used the same hyperparameter but added augmentation data to increase the variance. The accuracy of the model reaches 91% and the validation reaches 88%. This is quite good because the model is not overfitting. Getting to 1 epoch takes about 43 seconds on this model.

1.1.1.1.1.1.1.3 Accuracy of DenseNet Model with 150x150 Target Size without Data Augmentation

In the third experiment the authors used the same hyperparameter but changed the target size to 150x150 which was originally only 50x50, and without using data augmentation. In this experiment, very good accuracy was obtained until it touched the limit or Callback, namely 99% during training and 98 during validation. However, in this model it takes a relatively longer time to reach 1 epoch, which is around 202 seconds. In this experiment only 16 epochs are needed to get the highest accuracy.



1.1.1.1.1.1.1.1.4 Accuracy of DenseNet201 with 50x50 Targer Size With Data Augmentation and using Sparse Categorical Cross Entropy Loss Function

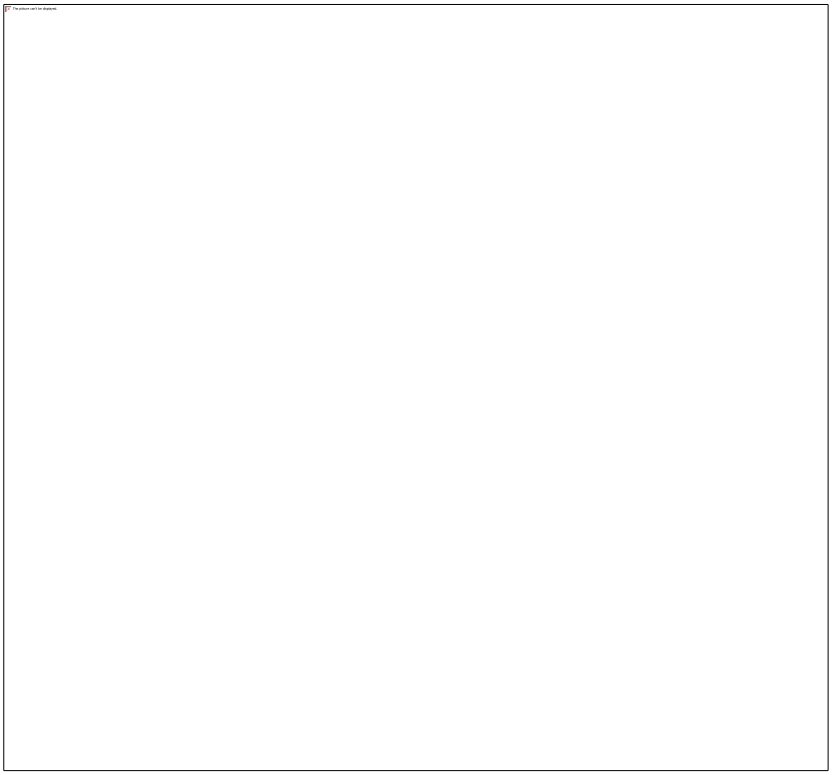
In the fourth experiment the authors used the same hyperparameter but changed the target size to 50x50, and using data augmentation. This experiment using different loss function, by using sparse categorical cross entropy there is got a good accuracy. Getting to 1 epoch takes about 23 seconds on this model.

```
Epoch 91/100
16/16 [==============================] - 19s 1s/step - loss: 0.1521 - accuracy: 0.9492 - val_loss: 0.1887 - val_accuracy: 0.9267
Epoch 92/100
16/16 [==============================] - 22s 1s/step - loss: 0.1687 - accuracy: 0.9375 - val_loss: 0.2454 - val_accuracy: 0.9017
Epoch 93/100
16/16 [==============================] - 21s 1s/step - loss: 0.1481 - accuracy: 0.9424 - val_loss: 0.1441 - val_accuracy: 0.9433
Epoch 94/100
16/16 [==============================] - 22s 1s/step - loss: 0.1411 - accuracy: 0.9446 - val_loss: 0.1094 - val_accuracy: 0.9583
Epoch 95/100
16/16 [==============================] - 22s 1s/step - loss: 0.1425 - accuracy: 0.9492 - val_loss: 0.1607 - val_accuracy: 0.9283
Epoch 96/100
16/16 [==============================] - 19s 1s/step - loss: 0.1233 - accuracy: 0.9486 - val_loss: 0.1598 - val_accuracy: 0.9400
Epoch 97/100
16/16 [==============================] - 22s 1s/step - loss: 0.1383 - accuracy: 0.9434 - val_loss: 0.1840 - val_accuracy: 0.9200
Epoch 98/100
16/16 [==============================] - 21s 1s/step - loss: 0.1368 - accuracy: 0.9476 - val_loss: 0.2533 - val_accuracy: 0.9000
Epoch 99/100
16/16 [==============================] - 22s 1s/step - loss: 0.1485 - accuracy: 0.9434 - val_loss: 0.2329 - val_accuracy: 0.9033
Epoch 100/100
16/16 [==============================] - 18s 1s/step - loss: 0.1450 - accuracy: 0.9443 - val_loss: 0.1720 - val_accuracy: 0.9367
```

1.1.1.1.1.1.1.1.5 Accuracy of DenseNet201 with 50x50 Targer Size With Data Augmentation and using 100 epoch

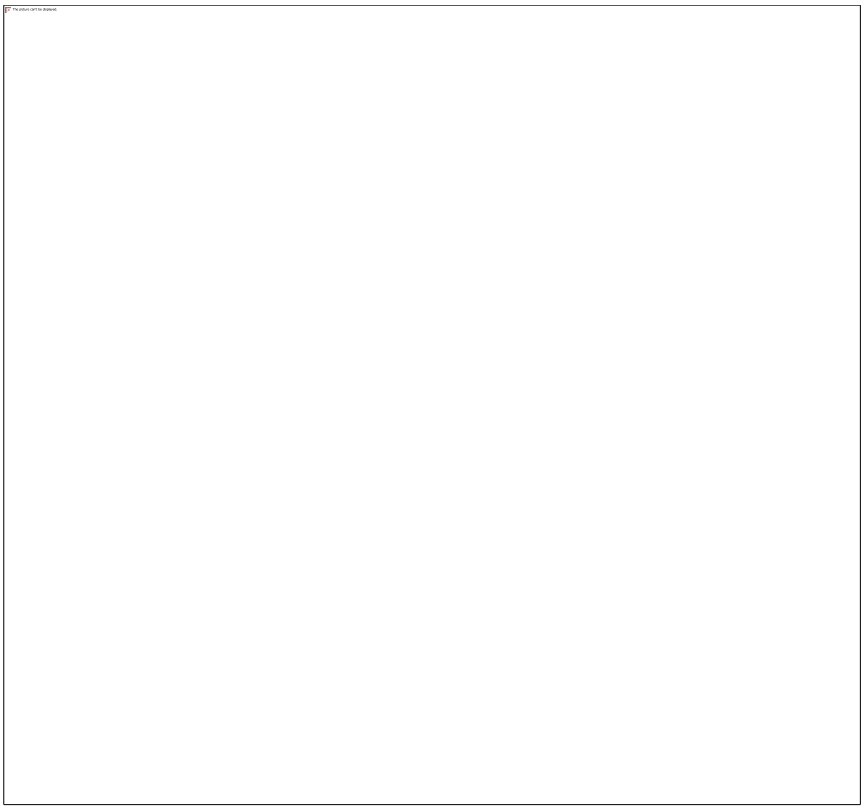
In the fifth experiment, the authors conducted 100 epoch training to prove whether the model was overfitting or not if it was not given a callback function. It can be seen that the accuracy of validation obtained is 93% and training is 94% with 22 seconds to complete 1 epoch.

1.3.6 Loss Function and Accuracy Charts



1.1.1.1.1.1.1.1.6 Accuracy and Loss using 50x50 Target Size without Data Augmentation

After generating the expected accuracy in the first experiment, the model is next assessed by showing graphs of the accuracy and loss functions using matplotlib. As can be seen in *Figure above*, the accuracy graph keeps growing and produces a high enough level of accuracy, while the loss function graph displays a decreasing flow, which indicates that prediction errors are also going down. As a result, the accuracy will keep growing and producing a high level of accuracy. Overall, despite the model's outstanding performance on the training set of data, the somewhat worse performance on the validation set of data and the widening gap between the training and validation accuracy raise concerns about the model's potential overfitting.

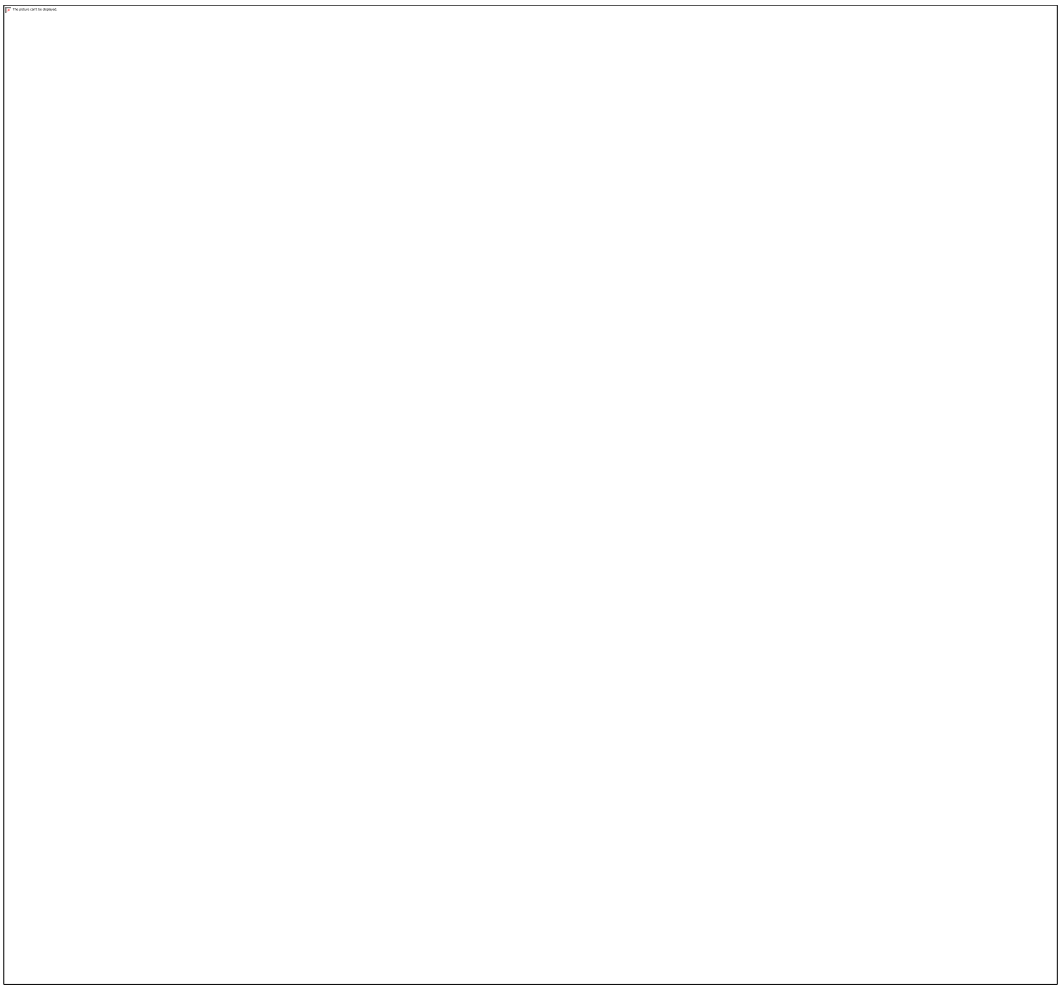


1.1.1.1.1.1.1.7 Accuracy and Loss using 50x50 Target Size using Data Augmentation

In the second experiment, the authors used the same hyperparameter but added data augmentation to the dataset. It can be seen in *Figure above* that the graph has experienced a significant decrease in loss and does not have a large gap between training and validation accuracy.

1.1.1.1.1.1.1.1.8 Accuracy and Loss using 150x150 Target Size without Data Augmentation

The author tries to utilize the same hyperparameter on the third experiment from *Figure above*, but without data augmentation, and also adjusts the target_size to 150x150. This experiment produced findings that were satisfactory. The model maintained a high level of accuracy throughout the training procedure, according to the reported training results. Over the epochs, the accuracy grew progressively until it reached 99% accuracy at epoch 16. This shows that the model has become highly accurate in classifying the photos in the dataset. The common consensus is that a model is operating well and producing correct predictions on both the training and validation datasets when it achieves an accuracy of 99%, which is regarded as being very good. It is crucial to remember that the performance of the model should not be judged exclusively on its accuracy. Other measures and considerations, such as the particular task, the make-up of the dataset, and the required level of performance, are significant.



1.1.1.1.1.1.1.1.9 Accuracy and Loss using 50x50 Target Size with Data Augmentation and using Sparse Categorical Cross Entropy

In the fourth experiment, the authors used the same hyperparameter and added data augmentation to the dataset. Using sparse categorical cross entropy it can be seen in *Figure above* that the graph has experienced a significant decrease in loss but have a small gap than second experiment.

1.1.1.1.1.1.1.1.10 Accuracy and Loss using 50x50 Target Size with Data Augmentation with 100 epoch

In the fifth experiment, the authors tried to add epoch to find out if the model was overfitting if it was not given a callback function. It can be seen that accuracy increases and loss decreases despite having some gaps between the train and test charts.

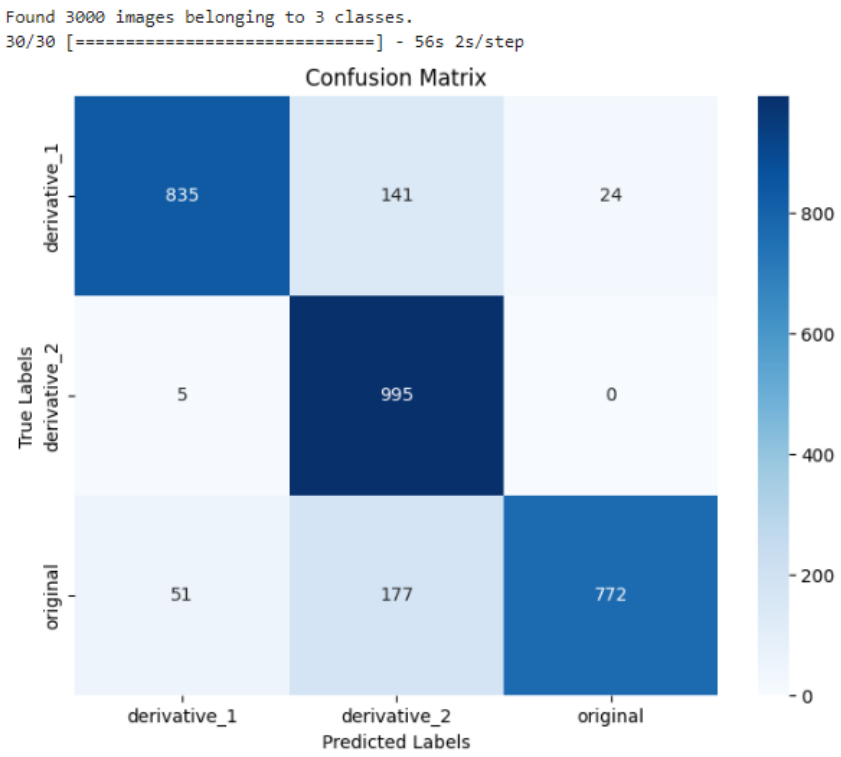
1.4 CONFUSION MATRIX

The confusion matrix provides insights into the performance of the model across different classes. It helps identify which classes are being misclassified and provides an overview of the model's overall accuracy and errors.

1.1.1.1.1.1.1.1.11 Plot Confusion Matrix Result on First Experiment

The code above computes and visualizes the confusion matrix for the test dataset using a trained DenseNet model. The test dataset consists of 3000 images belonging to 3 classes. The confusion matrix is a square matrix that shows the counts of true positive, true negative, false positive, and false negative predictions for each class. In the first experiment *Figure 4.25*, the confusion matrix has a size of 3x3 because there are 3 classes. From the confusion matrix:

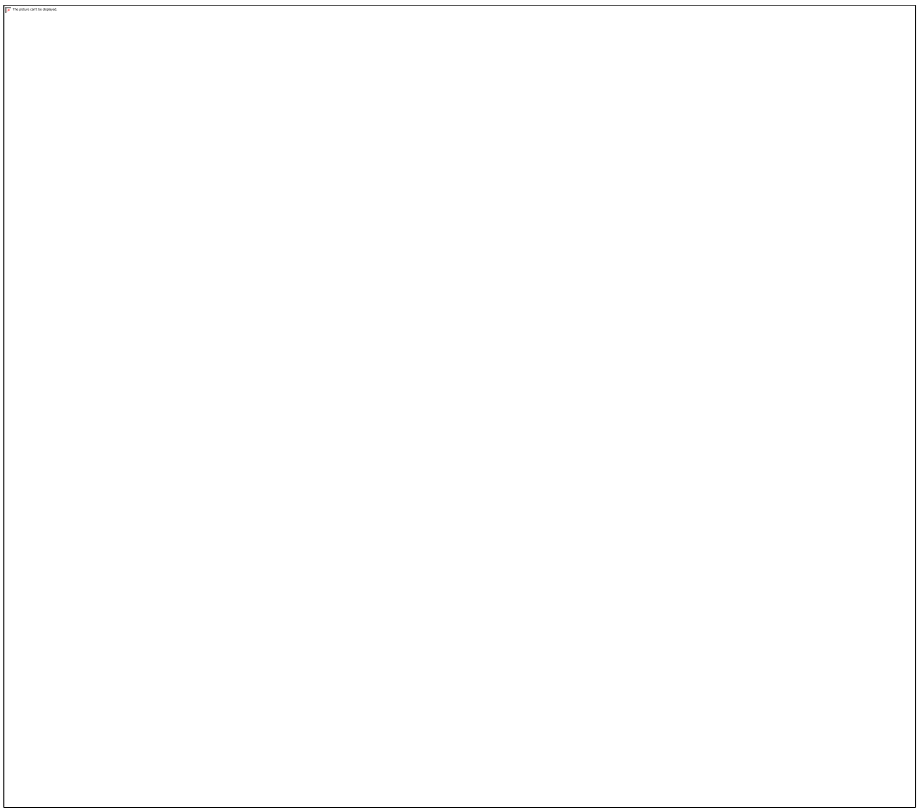
1. derivative_1: There are 775 photos that have been successfully identified as derivative_1 (true positives), 189 images that have been incorrectly identified as derivative_2 (false positives), and 36 images that have been incorrectly identified as original (false positives).
2. derivative_2: There were no misclassifications among the 996 photos of derivative_2 and they are all accurately identified as derivative_2 (true positives).
3. original: 769 photos have been accurately identified as original (true positives), while 211 and 20 original images, respectively, have been misclassified as derivative_1 and derivative_2 (false positives).



1.1.1.1.1.1.1.1.12 Plot Confusion Matrix Result on Second Experiment

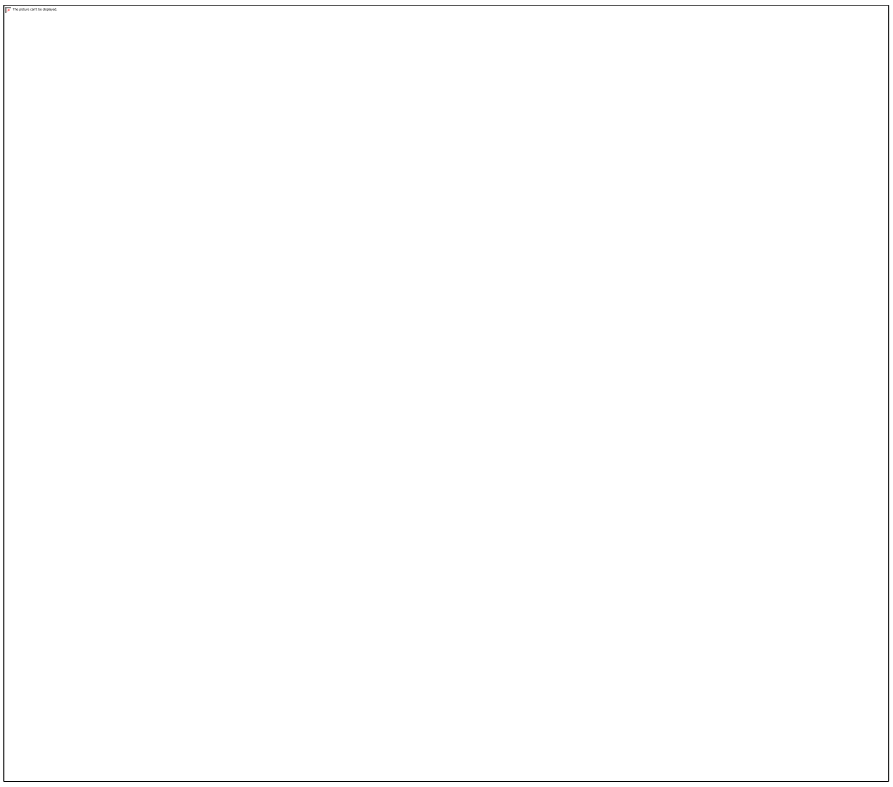
In the second experiment *Figure above*, the confusion matrix also has a size of 3x3 because there are 3 classes. From the confusion matrix:

1. derivative_1: There are 835 images correctly classified as derivative_1 (true positives), while 141 images of derivative_1 are misclassified as derivative_2 (false positives) and 24 images of derivative_1 are misclassified as original (false positives).
2. derivative_2: All 995 images of derivative_2 are correctly classified as derivative_2 (true positives), while 5 images of derivative_2 are misclassified as derivative_1 (false positives) and with no misclassifications.
3. original: There are 772 images correctly classified as original (true positives), while 51 images of original are misclassified as derivative_1 (false positives) and 177 images of original are misclassified as derivative_2 (false positives).



1.1.1.1.1.1.1.1.13 Plot Confusion Matrix Result on Third Experiment

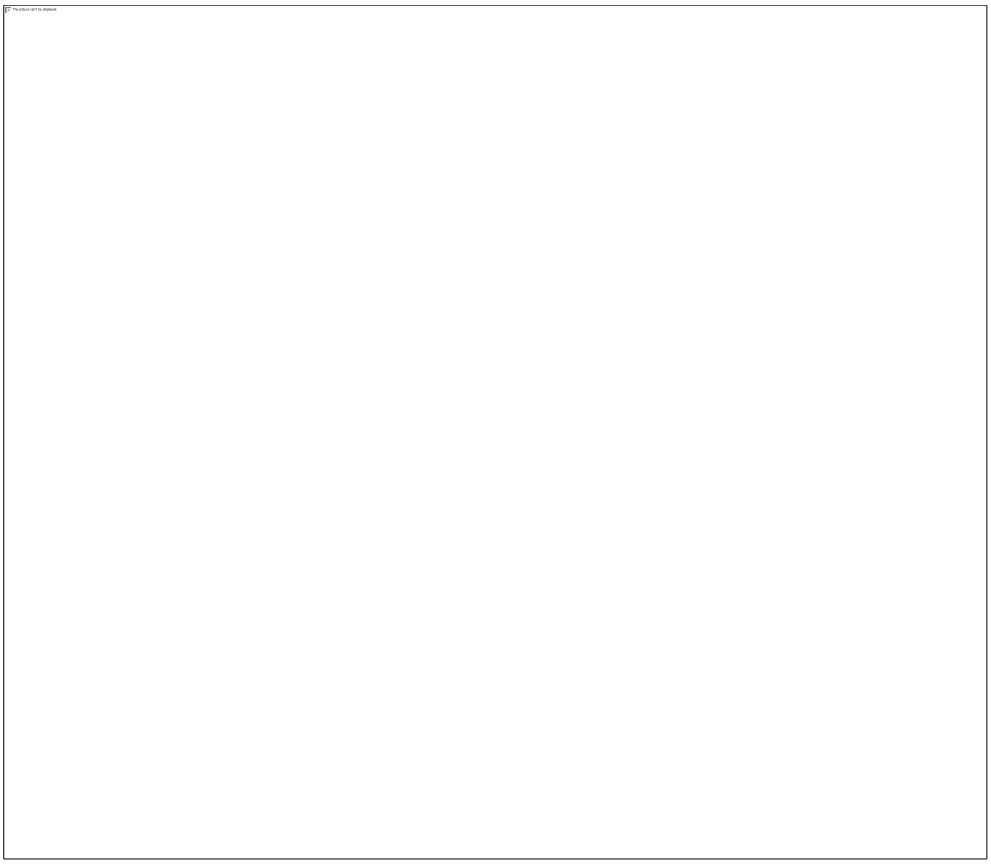
The given confusion matrix for the third experiment *Figure above* shows that the classification model performed flawlessly. Three classes are represented by the 3x3 matrix. The genuine labels are in each row, while the anticipated labels are in each column. In this instance, the matrix demonstrates that there were 1000 examples for each class, and they were all correctly categorized. No misclassifications were found in any of the classes. The final row also reveals that, while all other predictions were accurate, four examples from class 2 were incorrectly assigned to class 0. The fact that the vast majority of the predictions were correct shows that the model had an overall accuracy rate of 99.6%. The model performs well and is very accurate, making it a trustworthy classifier for the provided dataset.



1.1.1.1.1.1.1.1.14 Plot Confusion Matrix Result on Fourth Experiment

The presented confusion matrix relates to a three-class multi-class classification issue: derivative_1 (class 0), derivative_2 (class 1), and original (class 2). Each row represents the true class, while each column represents the predicted class, and the matrix indicates the performance of the model.

1. derivative_1: 884 were correctly identified as derivative_1, 99 were incorrectly categorized as derivative_2, and 17 examples were incorrectly categorized as original. Class Derivative_1 has an accuracy of 88.4% (884/1000).
2. derivative_2: Of the 1000 instances of derivative_2, 991 were correctly identified as derivative_2, six as derivative_1, and three as original. Class Derivative_2 has a 99.1% (991/1000) accuracy rate.
3. original: Of the 1000 original cases, 753 were correctly identified as original, 67 as derivative_1, and 180 as derivative_2. The class original's accuracy is 75.3% (753/1000).



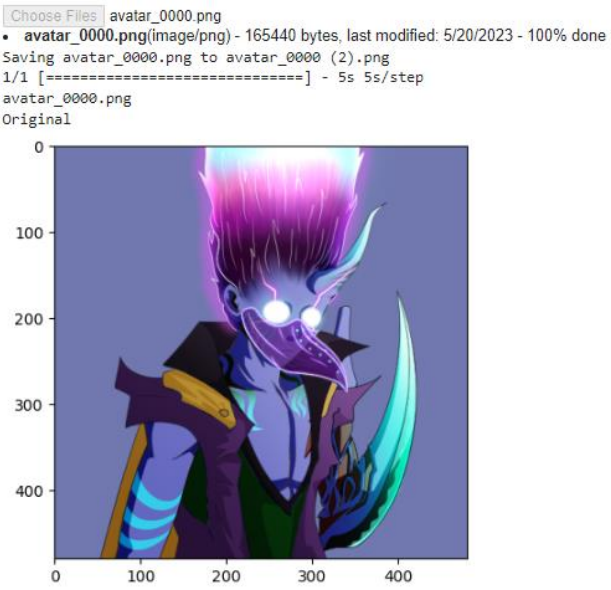
1.1.1.1.1.1.1.1.15 Plot Confusion Matrix Result on Fifth Experiment using 100 Epoch

From the results of 100 epochs in the fifth experiment, a classification as shown in *Figure above* was obtained. The following is an explanation of the Confusion Matrix plot above:

1. Class derivative_1 (class 0): The model successfully predicted 950 occurrences (true positives) out of 1000 instances that belonged to class derivative_1. However, it incorrectly labeled 21 instances of derivative_1 as original and 29 cases of derivative_1 as derivative_2 (false negatives).
2. Class derivative_2 (class 1): The model accurately categorized 992 of the 1000 cases that belonged to class derivative_2 (true positives). It incorrectly categorized 6 instances of derivative_2 as derivative_1 (false negatives) and 2 instances of derivative_2 as original (false negatives).
3. Original class (class 2): The model correctly predicted 873 of the 1000 occurrences that made up the original class (true positives). 51 original cases were incorrectly labeled as derivative_1 (false negatives), while 76 original instances were incorrectly labeled as derivative_2 (false negatives).

In the fifth experiment, it can be seen that the machine is able to classify quite well by using hyperparameters of 100 epochs and using categorical cross entropy. But it is still not good compared to using callbacks and using a larger target size as in the third experiment.

Test Drive Model



1.1.1.1.1.1.1.16 Test Drive Model

The last step is to test the model that has been created using the image data used into a dataset. To predict the results of the image, the author must upload one of the images in the dataset used and then the model will begin to predict the image according to the label created or not. In the picture above, it can be seen that with the resulting accuracy that the predicted image is in accordance with the label.

1.4.1 Discussion

From the results of these studies there are several analyzes conducted by the author. in collecting datasets the author must make it himself, because there is no site that provides NFT image datasets in large quantities, and it is not possible to download NFT images one by one from the marketplace. This iteration of the project uses 30 epochs so the process doesn't take long. In 30 epochs, the highest accuracy is 99% which stops at 16 epoch. By reducing the target size, it also affects the training process to make it faster. Data augmentation in the dataset is also very influential in this project to increase variations. This project focuses on classifying NFT derivative images using transfer learning with the DenseNet201 model. With this model it is proven that DensNet201 can classify NFT derivative images with fairly good accuracy.

1.5 CONCLUSION

In the last chapter, conclude the answer submitted in the problem formulation. From the experiments that have been carried out by the author, it can be concluded that the performance of the Dense Net-201 architecture is suitable for use in this topic, which is about predicting NFT images with 3 labels used, namely original, derivative 1, and derivative 2. In addition to the architecture used, hyperparameters also determine the level of accuracy produced such as batch size, learning rate, epoch, and many more. In this experiment, the loss function Categorical Cross Entropy is said to be very suitable than Sparse Categorical Cross Entropy, so that it affects the accuracy produced because it uses 3 labels where this type of cross entropy requires labels to be encoded as categories. From the results of experiments that have been carried out by the author, to speed up the training process with small target sizes, it is recommended to use augmentation data for increase the accuracy. However, to get even better accuracy, a larger target size is very influential in increasing accuracy without having to add augmentation data. but by using a large target size it will take a relatively long time to pass 1 epoch. And from the results of the fifth experiment, using 100 epochs, it was proven that the engine is good enough to classify NFT derivative images but still has some gaps between the train graph and the test on loss and accuracy calculations.

Next research if you want to develop a project with predicting algorithm ensure the architecture model and hyperparameters match the dataset used. Deep learning is all about experimentation. You can improve the performance of your model by using a different pre-processing method or by transfer learning with a completely different model. You can also make major changes to your model by tampering with the hyperparameter tuning.

REFERENCES

[1] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges," May 2021, [Online]. Available: http://arxiv.org/abs/2105.07447

[2] N. Meuschke, C. Gondek, D. Seebacher, C. Breitinger, D. Keim, and B. Gipp, "An Adaptive Image-based Plagiarism Detection Approach," in *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 131–140. doi: 10.1145/3197026.3197042.

[3] Praveen Krishnan, *Computer Vision – ECCV 2016*, vol. 9905. in Lecture Notes in Computer Science, vol. 9905. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-46448-0.

[4] S. Appalaraju and V. Chaoji, "Image similarity using Deep CNN and Curriculum Learning." Accessed: Jul. 07, 2023. [Online]. Available: https://arxiv.org/abs/1709.08761

[5] E. Al-Thwaib, B. H. Hammo, and S. Yagi, "An academic Arabic corpus for plagiarism detection: design, construction and experimentation," *International Journal of Educational Technology in Higher Education*, vol. 17, no. 1, Dec. 2020, doi: 10.1186/s41239-019-0174-x.

[6] D. K. Mishra, R. Sheikh, S. Jain, and Institute of Electrical and Electronics Engineers, *Apparel Classification Using Convolutional NeuralNetworksEshwar*. 2016. doi: 10.1109/ICTBIG.2016.7892641.

[7] N. Azahro Choirunisa, T. Karlita, and R. Asmara, "Deteksi Ras Kucing Menggunakan Compound Model Scaling Convolutional Neural Network," *Technomedia Journal*, vol. 6, no. 2, pp. 236–251, 2021, doi: 10.33050/tmj.v6i2.1704.

[8] H. Fonda, "Klasifikasi Batik Riau Dengan Menggunakan Convolutional Neural Networks (Cnn)," *Jurnal Ilmu Komputer*, vol. 9, no. 1, pp. 7–10, 2020, doi: 10.33060/jik/2020/vol9.iss1.144.

[9] A. P. Song, Q. Hu, X. H. Ding, X. Y. Di, and Z. H. Song, "Similar Face Recognition Using the IE-CNN Model," *IEEE Access*, vol. 8, pp. 45244–45253, 2020, doi: 10.1109/ACCESS.2020.2978938.

[10] T. Xie, K. Wang, R. Li, and X. Tang, "Visual robot relocalization based on multi-task CNN and image-similarity strategy," *Sensors (Switzerland)*, vol. 20, no. 23, pp. 1–20, Dec. 2020, doi: 10.3390/s20236943.

[11] K. E. Ak, J. H. Lim, J. Y. Tham, and A. A. Kassim, "Efficient multi-attribute similarity learning towards attribute-based fashion search," in *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, Institute of Electrical and Electronics Engineers Inc., May 2018, pp. 1671–1679. doi: 10.1109/WACV.2018.00186.

[12] A. T. Prihatno, N. Suryanto, S. Oh, T. T. H. Le, and H. Kim, "NFT Image Plagiarism Check Using EfficientNet-Based Deep Neural Network with Triplet Semi-Hard Loss," *Applied Sciences (Switzerland)*, vol. 13, no. 5, 2023, doi: 10.3390/app13053072.