Senior Design Project I

*Deep Learning Based Smart Otoscope Design with Optical Methods*

*Enes Dertli 180 709 012*

*Contents*

*1 Introduction*

*2 Literature Review*

*i) What is CNN?*

*ii) How CNN works?*

*iii) Advantage - Disadvantage*

*3 Subject & Methodology*

*4 Conclusion*

*5 Appendix*

*6 Reference*

*Deep Learning Based Smart Otoscope Design with Optical Methods*

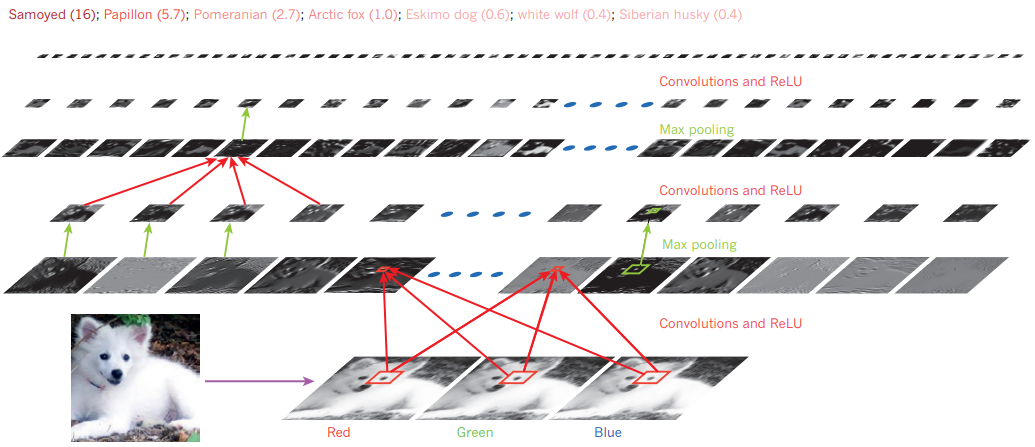
*Introduction*

Otitis media, a common disease characterized by the presence of fluid in the middle ear cavity, can cause various cognitive and affective problems in the individual. There are multiple types of otitis media, which occurs as a result of fluid accumulation in the middle ear and manifests itself with various symptoms such as severe pain felt in the ear, blood or yellow, green infectious discharges. It is reported that much worse cases are seen if the diagnosis is made and not treated appropriately. Similarly, it is stated that EOM is very common in children and is the most common cause of hearing loss (1). In order to diagnose otitis media, it is necessary to see an ENT specialist. It is normal to ask some questions for this. An otoscope, a lighted instrument for looking into the passages of the ear, nose and throat, will most likely be used. The accuracy of the diagnosis may vary from observer to observer, depending on the technical skills and experience of the physician as well as the subjective bias of the observer. This affects the correct administration of treatments, increases health costs and can lead to serious health complications. Most of the time, wrong diagnoses are made. As a result, a lot of unnecessary antibiotics are used, which leads to unnecessary expenditure. Here we are introducing a new approach to help diagnose otitis media that leverages advances in otoscopy and machine learning. With our project, doctors will be able to diagnose the inflammation behind the eardrum, which is difficult to see with the current system, in a highly accurate and timely manner. It will be processed in real time over the CNN network and will allow us to get real-time results. Thus, the project will eliminate the need for any practitioner to interpret the data and has the potential to reduce inappropriate antibiotic use, improve patient care outcomes and reduce healthcare costs associated with otitis media. This will enable real and simultaneous diagnosis.

*Literature Review*

*i) What is CNN?*

CNNs are a class of artificial neural networks that are designed to process data with a grid-like topology, such as an image. They are composed of multiple layers of interconnected nodes, which process and analyze the input data through a series of mathematical operations. The layers of a CNN typically include convolutional layers, which apply filters to the input image to detect specific patterns and features, and pooling layers, which reduce the spatial dimensions of the feature maps produced by the convolutional layers. The output of the CNN is a set of probabilities that represent the likelihood that the input image belongs to each of the categories that the model was trained to recognize. CNNs have been successful in a wide range of image recognition and processing tasks, including image classification, object detection, and face recognition. (2)



*ii) How CNN works?*

Convolutional neural networks (CNNs) are a type of artificial neural network designed for image recognition and processing. They are composed of multiple layers of interconnected nodes, which process and analyze the input data through a series of mathematical operations.(3)

Here is a brief overview of how CNNs work:

***Input***: The input layer of a CNN receives the raw image data, which is typically represented as a multi-dimensional array of pixel values.

***Convolutional layers***: These layers apply a series of filters to the input image, which scan the image and detect specific patterns and features. Each filter is a small matrix of weights that is applied to a region of the input image, and the output of the convolutional layer is a set of feature maps that represent the presence of different patterns and features in the input image.

***Pooling layers***: These layers reduce the spatial dimensions of the feature maps produced by the convolutional layers, which helps to reduce the number of parameters in the model and makes it more computationally efficient. Pooling is typically performed by taking the maximum or average value of a group of adjacent pixels in the feature map.

***Fully connected layers***: These layers perform classification by connecting all the nodes in the previous layers and applying a series of weights and biases to the input data. The output of the fully connected layer is a set of probabilities that represent the likelihood that the input image belongs to each of the categories that the model was trained to recognize.

***Output***: The output layer of the CNN produces the final classification of the input image, based on the probabilities produced by the fully connected layer.

During training, the weights and biases of the CNN are adjusted to minimize the error between the predicted and true labels of the input data. This process is typically done using an optimization algorithm, such as stochastic gradient descent, which adjusts the weights and biases in a direction that reduces the error.

*iii) Advantage - Disadvantage*

*a) Advantage*

* According to the review article “Convolutional Neural Networks” by Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2) CNNs are well-suited for image recognition and processing tasks due to their ability to automatically learn and extract features from images.
* As mentioned in the survey paper “A comprehensive survey on convolutional neural network in medical image analysis” by Xujing Yao, Xinyue Wang, Shui-Hua Wang and Yu-Dong Zhang (4)CNNs have the ability to process large amounts of data efficiently and learn from a large number of examples.
* According to the survey paper " A comprehensive survey on convolutional neural network in medical image analysis " by Xujing Yao, Xinyue Wang, Shui-Hua Wang and Yu-Dong Zhang (4) CNNs are able to handle noise and variations in the input data, making them robust to small changes in the input.
* CNNs have been successful in a wide range of image recognition and processing tasks, including image classification, object detection, and face recognition, among others.
* In the review article "Convolutional Neural Networks" by Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2), it is mentioned that CNNs are able to learn from a large number of examples, which allows them to learn more complex and abstract patterns in the data.

Overall, CNNs have proven to be a powerful tool for image recognition and processing tasks, and they have been widely used in a variety of applications. (5)

*b) Disadvantage*

* According to the review article "Convolutional Neural Networks" by Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2), CNNs require large amounts of data to train effectively, and can be computationally expensive to train and deploy.
* As stated in the survey paper “A comprehensive survey on convolutional neural network in medical image analysis” by Xujing Yao, Xinyue Wang, Shui-Hua Wang and Yu-Dong Zhang (4) CNNs may be sensitive to the quality and annotation of the training data, and can be difficult to interpret.
* In the review article "Convolutional Neural Networks" by Yann LeCun, Yoshua Bengio, and Geoffrey Hinton (2), it is mentioned that CNNs have the potential for overfitting when the training dataset is small or specialized.
* As summarized in the review article “ImageNet Classification with Deep Convolutional Neural Networks” by by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (5), CNNs can be computationally expensive to train and deploy, especially for large and complex models. This can be a challenge for tasks that require real-time processing or for systems with limited computational resources.
* According to the review article “ImageNet Classification with Deep Convolutional Neural Networks” by by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (5), CNNs can be prone to overfitting, especially when the training dataset is small or highly specialized. Overfitting occurs when the model becomes too closely tailored to the training data, which can reduce its ability to generalize to new data.

Overall, while CNNs have proven to be a powerful tool for image recognition and processing tasks, they are not always the best choice for every application and may have some limitations depending on the specific task and dataset. (6)

*Methodology & Subject*

**i)**

Otitis media, a common disease characterized by the presence of fluid in the middle ear cavity, can cause various cognitive and affective problems in the individual. There are more than one type of otitis media, which occurs as a result of fluid accumulation in the middle ear and manifests itself with various symptoms such as severe pain felt in the ear, blood or yellow, green infectious discharges. When the literature is examined; it is reported that approximately 90% of children have AOM at least once until the age of two, only 60% of them are diagnosed correctly and much worse pictures are seen as a result of not being treated appropriately. (1)*Similarly, EOM is reported to be very common in children and is the most common cause of hearing loss.* (7) *In order to diagnose otitis media, an ENT specialist must examine the patient. The doctor will first look for symptoms in the patient. It is normal to ask some questions for this. The doctor will most likely use an otoscope, a lighted instrument to look into the passages of the ear, nose and throat. A stethoscope is also used to listen to breathing. A pneumatic otoscope is the only special medical instrument a doctor may need to diagnose an ear infection. It is used to visualise the patient's ear and to see if there is fluid behind the eardrum.*

Apart from these medical education methods, important studies are being carried out in the scientific world to support the decision-making processes of experts in the diagnosis of otitis media with computer-based decision support systems. For this purpose; both the workload of the specialists is reduced and the correct diagnosis is facilitated at an early stage. When the studies were examined; in a study conducted with 66 children between the ages of 9 months and 16 years using telemedicine method, the middle ear images of children were evaluated and diagnosed by experts in different locations and expert recommendations were brought together. For this reason, an infrared light source will be added to the camera module of the device we will create instead of a pneumatic otoscope to diagnose otitis media. This is because, unlike visible light, Infrared light penetrates the eardrum more easily and is also heavily absorbed by water.

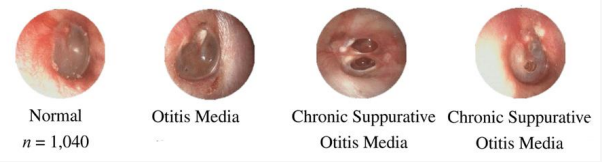
As a result, the Infrared light otoscope provides more reliable data from middle ear structures. Here we present a new approach to help diagnose otitis media that leverages advances in otoscopy and machine learning. With the dataset we have obtained with images taken from the middle ear, we will be able to distinguish between otitis media and healthy middle ear using Convolutional Neural Networks.

The aim is to automatically detect the vital eardrum region in the middle ear images obtained with the otoscope device we will create and classify normal and abnormal eardrum images with artificial intelligence methods. For this purpose; first of all, the eardrum region will be detected with the highest accuracy rate by using deep learning based object detection algorithms that have achieved successful results in biomedical images in object detection for detecting the eardrum region. In order to complete the model development studies in the shortest time in accordance with the desired performance criteria, a convolutional neural network-based transfer learning method, which minimizes high computational power and time, will be used.

The learning obtained with the pre-trained convolutional neural network algorithms will be transferred to the network model planned to be implemented for otitis media detection and classification, and transfer learning will be performed by retraining the last layers of the network model with the data we have obtained. (8) In deep learning applications, there are various methods to increase the data set volume when the data set is insufficient to perform classification operations. Considering the studies conducted in the literature, it was determined that data augmentation techniques such as playing with the brightness of the image, rotation, scaling, horizontal and vertical mirroring, adding noise, image segmentation, segmenting regions of interest, changing the contrast and intensity of the image were used.

For unbalanced data, the data augmentation method known as Gan Data Augmentation will be used. In case the desired success criteria cannot be achieved with the transfer learning method, Bayesian Optimization Method and Big Bang Big Crunch Optimization methods will be used and tried to adjust the hyper parameters and try to achieve the success criteria.

**ii**) In this section; data set preparation and image preprocessing will be carried out. The first work package of the project is the Data Preparation and Preprocessing step. In this step, the dataset consisting of color in-ear images required for training and validation of the deep learning model will be obtained and made ready for use through image processing techniques. Ready data   
In addition to the data sets, work will be done to increase the data sets through Muğla Sıtkı Koçman University Research Hospital. Sample color in-ear images and disease stages are shown in Figure 1:



In deep learning applications, there are various methods to increase the data set volume when the data sets are insufficient to perform classification operations. Considering the studies in the literature, it is determined that data augmentation techniques such as playing with the brightness of the image, rotation, scaling, horizontal and vertical mirroring, adding noise, image segmentation, segmenting regions of interest, changing the contrast and intensity of the image are used. For unbalanced data, the data augmentation method known as Gan Data Augmentation will be used. In case the desired success criteria cannot be met with the transfer learning method, Bayesian Optimization Method and Big Bang Big Crunch Optimization methods will be used and the hyper parameters will be adjusted. and success criteria will be ensured.

**iii)** First, we will need to introduce the data to our software. This is actually the most important part. Because there are more than one type of otitis media. Deep learning technique will be used at this stage. Deep learning uses algorithms known as artificial neural networks, which are inspired by the way biological nervous systems process information. This allows computers to identify what each piece of data represents and learn models. TensorFlow, one of the most popular tools in deep learning, was developed by researchers and engineers from the Google Brain Team to conduct machine learning and deep neural network research. An open source artificial intelligence and machine learning library used in sensing, discovery, classification, understanding and prediction applications, TensorFlow uses data flow graphs to create models and allows developers to build multi-layered and large-scale artificial neural networks. For these reasons, TensorFlow deep learning is planned to be used in the project. Along with TensorFlow, Convolutional Neural Networks (CNN) will be used as a deep learning network.

It also requires a light source with a wavelength that covers the absorption spectrum of water and an Infrared Camera that can detect these wavelengths. Part of the photon emitted from the light source and passing through the water molecules will be absorbed by the water molecule and the amount of light absorbed will give the density of the liquid and the level of inflammation. Considering the absorption spectrum, a broadband light source with a wavelength of 800-2000 nm in the near infrared region with a wavelength greater than 760 nm can be used. The light reflected from the liquid in the ear will be taken with an Infrared Camera with a spectral range of 640-1100 nm.

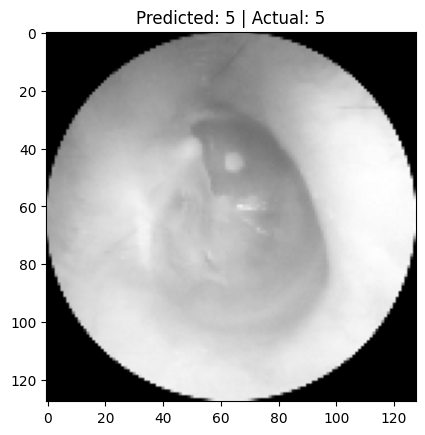
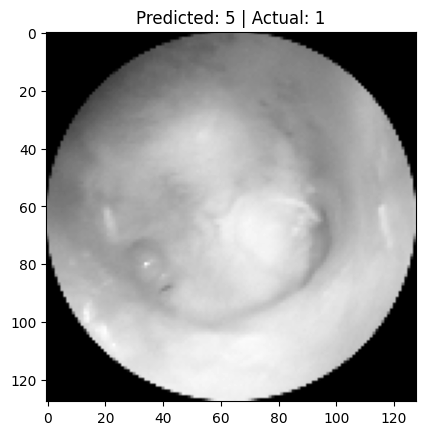
The first method considered in our project is the Bayesian Optimization method. In our project, we will train a network by processing the data we receive from the ear with the appropriate and more efficient method from CNN-RNN methods. We will embed this trained network into a Raspberry Pi 4 Model B developer board and then integrate the board into our Infrared Camera. In this way, the data we receive from our camera will be integrated with the card we embed the network we train and we will have the chance to convert the data we receive into results instantly.

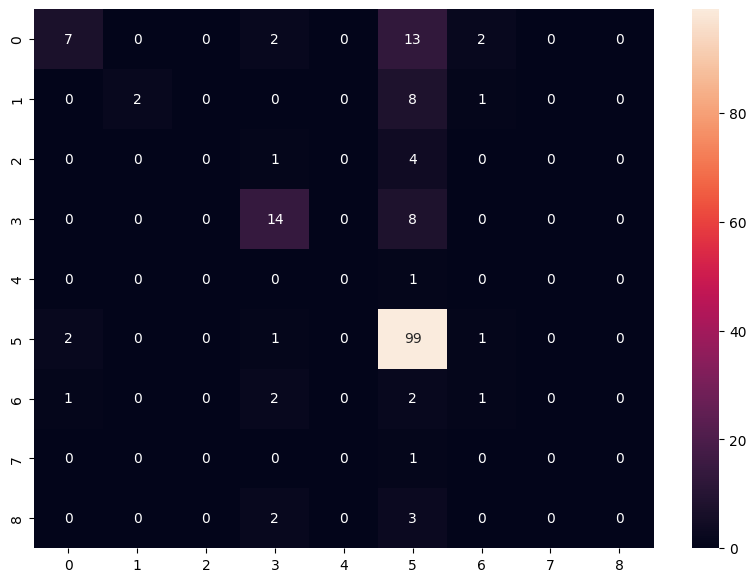
**iv)**

When the first model was trained using the available data, the accuracy was low. This is because too many Dense parameters were used compared to the number of data. The Dense parameter should be used considering the number of data. It was observed that the accuracy value increased to 0.6910 after reducing the number of Dense. This value is sufficient for now. In the test phase, if the classifications made by the model are not good enough, the photographs obtained with GAN technology will be added to the dataset.

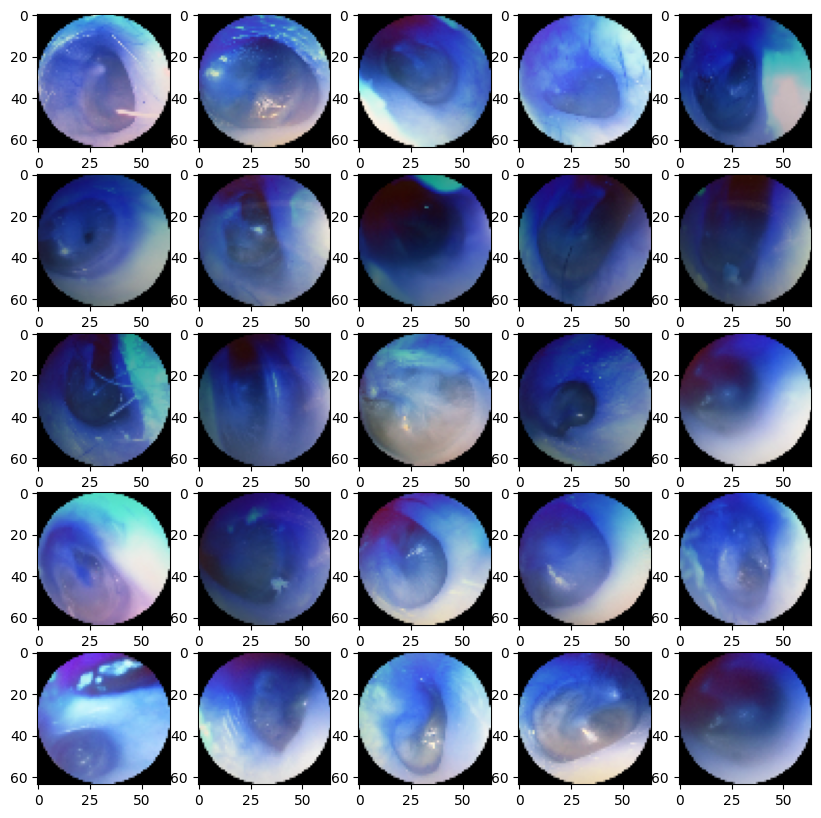
*Conclusion*

At the final stage of the project, the accuracy of the model was 0.69. The errors made by the model are usually on the 5th grade. This is because most of the available data belongs to class 5. Therefore, even if the accuracy is good enough, it is very likely that the classifications made by the model will be incorrectly specified as class 5. Sample classifications made by the model are as follows:

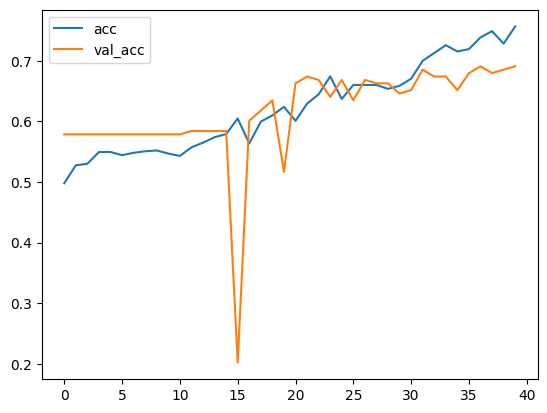




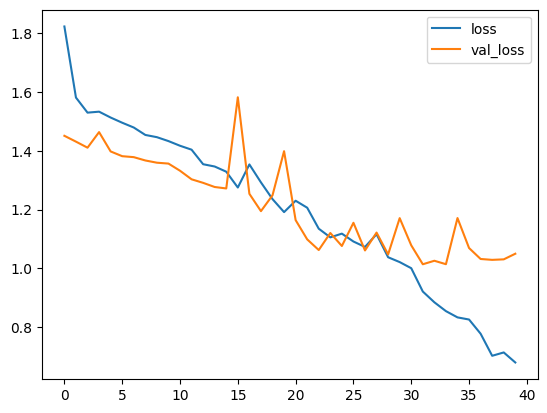
*Images for GAN*



*Accuracy Graph*



*Loss Graph*



*Appendix*

# CLASSIFICATION

|  |
| --- |
| import tensorflow as tf import numpy as np import matplotlib.pyplot as plt from sklearn.model\_selection import train\_test\_split import os import random import cv2 import pickle from tqdm import tqdm  images = [] number\_of\_errors = 0 number\_of\_processed = 0  dir = "S:\Downloads\data sets\eardrumDs\eardrumDs" for class\_name in os.listdir(dir):  path\_to\_class = dir + "\\" + class\_name  for image in os.listdir(path\_to\_class):      try:  path\_to\_image = f"{path\_to\_class}/{image}"   img = cv2.imread(path\_to\_image, 0)  img = cv2.resize(img, (128,128))  img = img/255  images.append([img, int(class\_name)])  number\_of\_processed += 1  except:  number\_of\_errors += 1  print(f"\rProcessed: {number\_of\_processed} | Erros: {number\_of\_errors}", end="")  Processed: 955 | Erros: 0  for \_ in range(27):  random.shuffle(images)  X = [] y = []  for image, idx in images:  X.append(image)  y.append(idx)  print(len(X)) print(len(y))  955 955 |

|  |
| --- |
| X\_train = [] y\_train = []  X\_val = [] y\_val = []  X\_test = [] y\_test = []  X\_train = X[:777] y\_train = y[:777]  X\_test = X[777:] y\_test = y[777:]  print(len(X\_train)) print(len(y\_train)) print(len(X\_test)) print(len(y\_test))  777 777 178 178  X\_train = np.array(X\_train) y\_train = np.array(y\_train)  X\_test = np.array(X\_test) y\_test = np.array(y\_test)  X\_train = X\_train.reshape(-1, 128, 128, 1)  X\_test = X\_test.reshape(-1, 128, 128, 1)  model = tf.keras.Sequential() model.add(tf.keras.layers.Conv2D(32, kernel\_size=(3,3), activation = 'relu', input\_shape=(128,128,1))) model.add(tf.keras.layers.MaxPooling2D(pool\_size=(2,2))) model.add(tf.keras.layers.Conv2D(64, kernel\_size=(3,3), activation = 'relu')) model.add(tf.keras.layers.MaxPooling2D(pool\_size=(2,2))) model.add(tf.keras.layers.Flatten()) model.add(tf.keras.layers.Dense(128, activation = 'relu'))  model.add(tf.keras.layers.Dropout(0.5)) model.add(tf.keras.layers.Dense(9, activation = 'softmax')) |

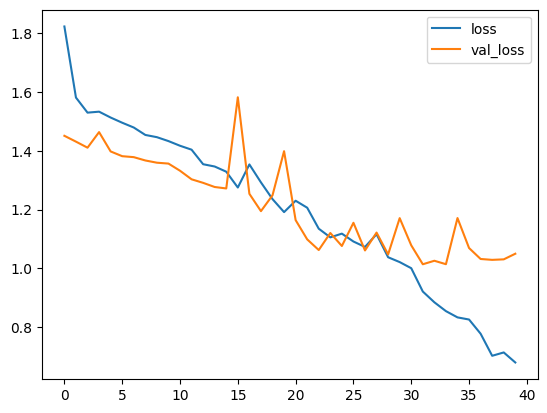
|  |
| --- |
| # Show the model summary model.summary()  Model: "sequential" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d (Conv2D) (None, 126, 126, 32) 320     max\_pooling2d (MaxPooling2D (None, 63, 63, 32) 0   )     conv2d\_1 (Conv2D) (None, 61, 61, 64) 18496     max\_pooling2d\_1 (MaxPooling (None, 30, 30, 64) 0   2D)     flatten (Flatten) (None, 57600) 0     dense (Dense) (None, 128) 7372928     dropout (Dropout) (None, 128) 0     dense\_1 (Dense) (None, 9) 1161    ================================================================= Total params: 7,392,905 Trainable params: 7,392,905 Non-trainable params: 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # Compile the model model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  history = model.fit(X\_train, y\_train, epochs=40, validation\_data=(X\_test, y\_test))  Epoch 1/40 25/25 [==============================] - 6s 240ms/step - loss: 1.8235 - accuracy: 0.4981 - val\_loss: 1.4512 - val\_accuracy: 0.5787  Epoch 2/40 25/25 [==============================] - 6s 229ms/step - loss: 1.5814 - accuracy: 0.5277 - val\_loss: 1.4313 - val\_accuracy: 0.5787 Epoch 3/40 25/25 [==============================] - 6s 226ms/step - loss: 1.5301 - accuracy: 0.5302 - val\_loss: 1.4107 - val\_accuracy: 0.5787 |

|  |
| --- |
| Epoch 5/40 25/25 [==============================] - 6s 228ms/step - loss: 1.5134 - accuracy: 0.5495 - val\_loss: 1.3979 - val\_accuracy: 0.5787 Epoch 6/40 25/25 [==============================] - 6s 226ms/step - loss: 1.4957 - accuracy: 0.5444 - val\_loss: 1.3818 - val\_accuracy: 0.5787 Epoch 7/40 25/25 [==============================] - 6s 227ms/step - loss: 1.4794 - accuracy: 0.5483 - val\_loss: 1.3785 - val\_accuracy: 0.5787 Epoch 8/40 25/25 [==============================] - 6s 228ms/step - loss: 1.4541 - accuracy: 0.5508 - val\_loss: 1.3672 - val\_accuracy: 0.5787 Epoch 9/40 25/25 [==============================] - 6s 225ms/step - loss: 1.4465 - accuracy: 0.5521 - val\_loss: 1.3596 - val\_accuracy: 0.5787 Epoch 10/40 25/25 [==============================] - 6s 227ms/step - loss: 1.4332 - accuracy: 0.5470 - val\_loss: 1.3566 - val\_accuracy: 0.5787 Epoch 11/40 25/25 [==============================] - 6s 227ms/step - loss: 1.4173 - accuracy: 0.5431 - val\_loss: 1.3323 - val\_accuracy: 0.5787 Epoch 12/40 25/25 [==============================] - 6s 226ms/step - loss: 1.4040 - accuracy: 0.5573 - val\_loss: 1.3031 - val\_accuracy: 0.5843 Epoch 13/40 25/25 [==============================] - 6s 227ms/step - loss: 1.3544 - accuracy: 0.5650 - val\_loss: 1.2910 - val\_accuracy: 0.5843 Epoch 14/40 25/25 [==============================] - 6s 235ms/step - loss: 1.3466 - accuracy: 0.5740 - val\_loss: 1.2768 - val\_accuracy: 0.5843 Epoch 15/40 25/25 [==============================] - 6s 226ms/step - loss: 1.3290 - accuracy: 0.5792 - val\_loss: 1.2719 - val\_accuracy: 0.5843 Epoch 16/40 25/25 [==============================] - 6s 234ms/step - loss: 1.2751 - accuracy: 0.6049 - val\_loss: 1.5823 - val\_accuracy: 0.2022 Epoch 17/40 25/25 [==============================] - 6s 232ms/step - loss: 1.3537 - accuracy: 0.5637 - val\_loss: 1.2537 - val\_accuracy: 0.6011 Epoch 18/40 25/25 [==============================] - 7s 270ms/step - loss: 1.2926 - accuracy: 0.5997 - val\_loss: 1.1947 - val\_accuracy: 0.6180 Epoch 19/40 25/25 [==============================] - 7s 269ms/step - loss: 1.2357 - accuracy: 0.6100 - val\_loss: 1.2486 - val\_accuracy: 0.6348 Epoch 20/40 25/25 [==============================] - 6s 244ms/step - loss: 1.1914 - accuracy: 0.6242 - val\_loss: 1.3989 - val\_accuracy: 0.5169 Epoch 21/40 |

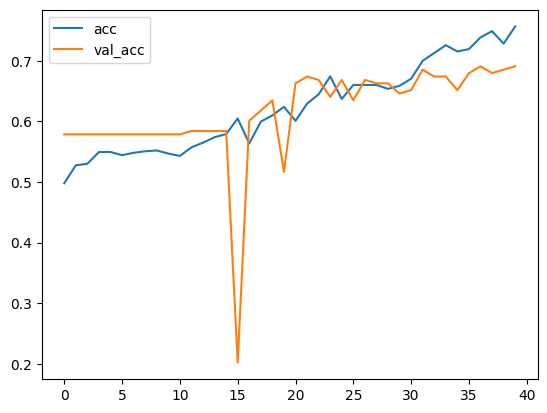
|  |
| --- |
| 25/25 [==============================] - 6s 253ms/step - loss: 1.2301 - accuracy: 0.6010 - val\_loss: 1.1641 - val\_accuracy: 0.6629 Epoch 22/40 25/25 [==============================] - 6s 258ms/step - loss: 1.2063 - accuracy: 0.6293 - val\_loss: 1.0986 - val\_accuracy: 0.6742 Epoch 23/40 25/25 [==============================] - 6s 250ms/step - loss: 1.1349 - accuracy: 0.6448 - val\_loss: 1.0622 - val\_accuracy: 0.6685 Epoch 24/40 25/25 [==============================] - 6s 227ms/step - loss: 1.1054 - accuracy: 0.6744 - val\_loss: 1.1203 - val\_accuracy: 0.6404 Epoch 25/40 25/25 [==============================] - 7s 282ms/step - loss: 1.1181 - accuracy: 0.6371 - val\_loss: 1.0760 - val\_accuracy: 0.6685 Epoch 26/40 25/25 [==============================] - 6s 227ms/step - loss: 1.0913 - accuracy: 0.6602 - val\_loss: 1.1550 - val\_accuracy: 0.6348 Epoch 27/40 25/25 [==============================] - 5s 218ms/step - loss: 1.0728 - accuracy: 0.6602 - val\_loss: 1.0608 - val\_accuracy: 0.6685 Epoch 28/40 25/25 [==============================] - 6s 236ms/step - loss: 1.1161 - accuracy: 0.6602 - val\_loss: 1.1217 - val\_accuracy: 0.6629 Epoch 29/40 25/25 [==============================] - 5s 214ms/step - loss: 1.0376 - accuracy: 0.6538 - val\_loss: 1.0473 - val\_accuracy: 0.6629 Epoch 30/40 25/25 [==============================] - 5s 217ms/step - loss: 1.0212 - accuracy: 0.6589 - val\_loss: 1.1708 - val\_accuracy: 0.6461 Epoch 31/40 25/25 [==============================] - 6s 227ms/step - loss: 1.0003 - accuracy: 0.6705 - val\_loss: 1.0783 - val\_accuracy: 0.6517 Epoch 32/40 25/25 [==============================] - 6s 223ms/step - loss: 0.9209 - accuracy: 0.7001 - val\_loss: 1.0138 - val\_accuracy: 0.6854 Epoch 33/40 25/25 [==============================] - 5s 218ms/step - loss: 0.8843 - accuracy: 0.7130 - val\_loss: 1.0257 - val\_accuracy: 0.6742 Epoch 34/40 25/25 [==============================] - 5s 214ms/step - loss: 0.8542 - accuracy: 0.7259 - val\_loss: 1.0140 - val\_accuracy: 0.6742 Epoch 35/40 25/25 [==============================] - 5s 214ms/step - loss: 0.8329 - accuracy: 0.7156 - val\_loss: 1.1711 - val\_accuracy: 0.6517 Epoch 36/40 25/25 [==============================] - 5s 213ms/step - loss: 0.8256 - accuracy: 0.7194 - val\_loss: 1.0691 - val\_accuracy: 0.6798 Epoch 37/40 25/25 [==============================] - 5s 216ms/step - loss: 0.7772 - accuracy: 0.7387 - val\_loss: 1.0317 - val\_accuracy: 0.6910 |

|  |
| --- |
| Epoch 38/40 25/25 [==============================] - 6s 225ms/step - loss: 0.7021 - accuracy: 0.7490 - val\_loss: 1.0288 - val\_accuracy: 0.6798 Epoch 39/40 25/25 [==============================] - 6s 243ms/step - loss: 0.7135 - accuracy: 0.7284 - val\_loss: 1.0305 - val\_accuracy: 0.6854 Epoch 40/40 25/25 [==============================] - 6s 221ms/step - loss: 0.6793 - accuracy: 0.7568 - val\_loss: 1.0496 - val\_accuracy: 0.6910 |

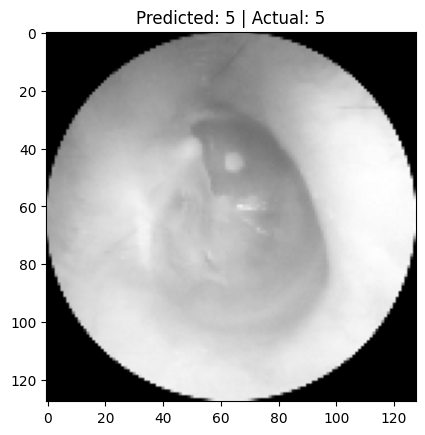
|  |
| --- |
| test\_loss, test\_acc = model.evaluate(X\_test, y\_test)  6/6 [==============================] - 0s 46ms/step - loss: 1.0496 - accuracy: 0.6910  # Figure of the loss plt.plot(history.history['loss'], label='loss') plt.plot(history.history['val\_loss'], label='val\_loss') plt.legend()  <matplotlib.legend.Legend at 0x1e2146461a0> |



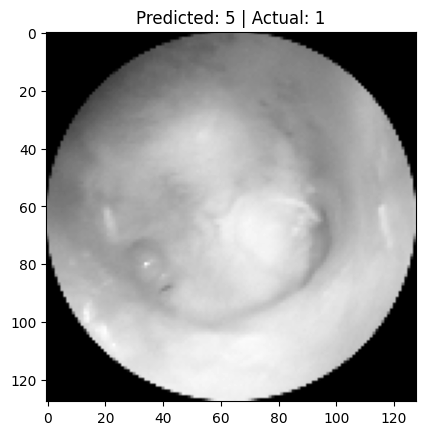
|  |
| --- |
| # Figure of the accuracy plt.plot(history.history['accuracy'], label='acc') plt.plot(history.history['val\_accuracy'], label='val\_acc') plt.legend()  <matplotlib.legend.Legend at 0x1e2145f6230> |



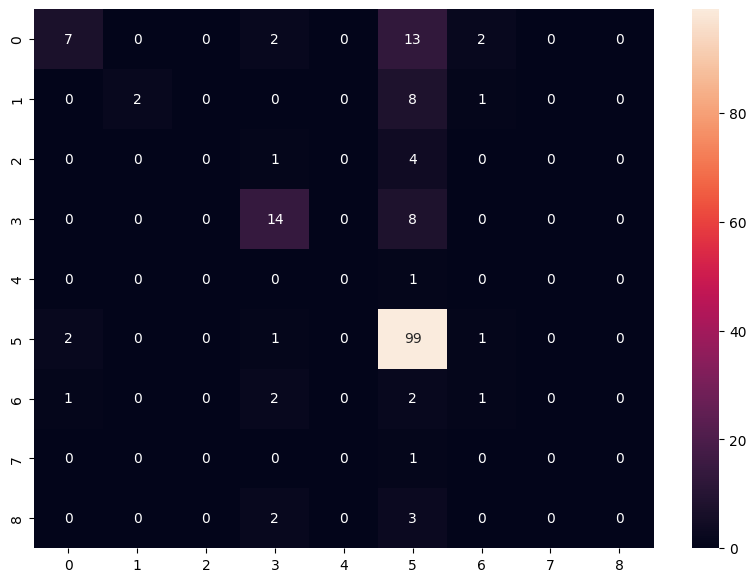
|  |
| --- |
| # Predict the model predictions = model.predict(X\_test).argmax(axis=1)  6/6 [==============================] - 0s 45ms/step  Predicts  i = random.randint(0, len(X\_test)) plt.imshow(X\_test[i].reshape(128,128), cmap='gray') plt.title(f"Predicted: {predictions[i]} | Actual: {y\_test[i]}")  Text(0.5, 1.0, 'Predicted: 5 | Actual: 5') |



|  |
| --- |
| # Wrong Predict  missclassified = np.where(predictions != y\_test)[0] i = random.choice(missclassified) plt.imshow(X\_test[i].reshape(128,128), cmap='gray') plt.title(f"Predicted: {predictions[i]} | Actual: {y\_test[i]}")  Text(0.5, 1.0, 'Predicted: 5 | Actual: 1') |



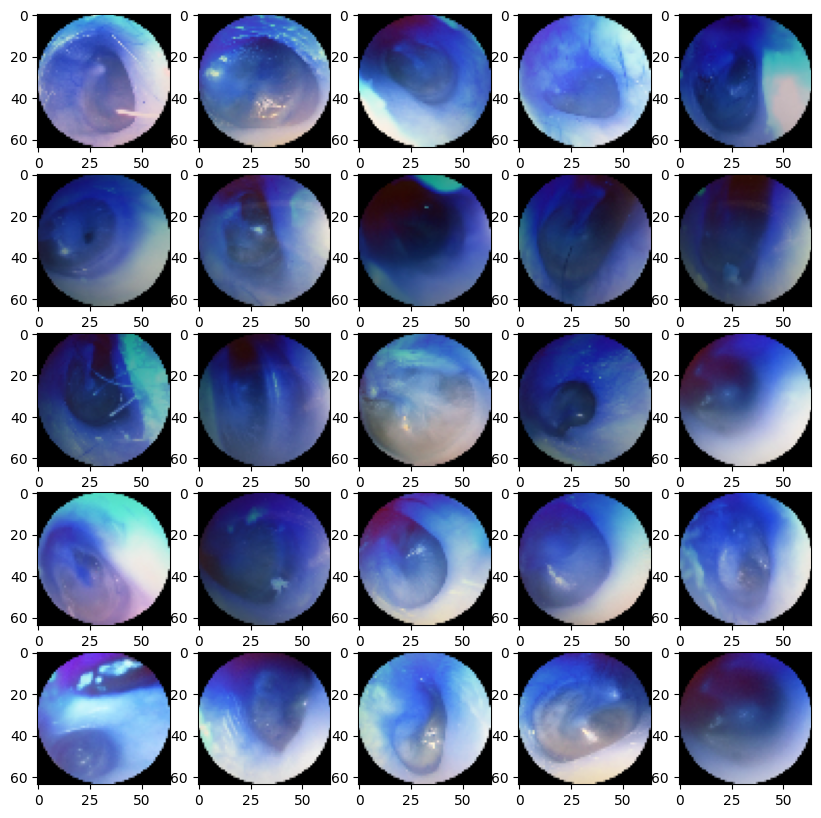
|  |
| --- |
| # Confusion matrix from sklearn.metrics import confusion\_matrix import seaborn as sns import pandas as pd  cm = confusion\_matrix(y\_test, predictions) df\_cm = pd.DataFrame(cm, range(9), range(9)) plt.figure(figsize=(10,7)) sns.heatmap(df\_cm, annot=True, fmt='g')  <AxesSubplot:> |



|  |
| --- |
| klasor\_adresi = "S:\\Downloads\\data sets\\eardrumDs\\eardrumDs\\0"  veri = [] process\_success = 0 process\_fail = 0  for resim\_adi in tqdm(os.listdir(klasor\_adresi)):  resim\_adresi = os.path.join(klasor\_adresi, resim\_adi)  resim = cv2.imread(resim\_adresi, cv2.IMREAD\_COLOR)  if(resim is not None):  resim = cv2.resize(resim, (64, 64))  veri.append([resim])  process\_success += 1  else:  process\_fail += 1  print(f"\rProcessed: {process\_success} | Erros: {process\_fail}", end="")  44%|████▎ | 52/119 [00:00<00:00, 255.25it/s]  Processed: 52 | Erros: 0  96%|█████████▌| 114/119 [00:00<00:00, 286.75it/s]  Processed: 114 | Erros: 0  100%|██████████| 119/119 [00:00<00:00, 280.15it/s]  Processed: 119 | Erros: 0 |

# GAN

|  |
| --- |
| fig = plt.figure(figsize=(10, 10)) for i in range(25):  ax = fig.add\_subplot(5, 5, i+1)  ax.imshow(veri[i][0]) |



|  |
| --- |
| import numpy as np veri = np.array(veri).reshape(-1, 64, 64, 3).astype('float32') veri = (veri - 127.5) / 127.5  veri = tf.data.Dataset.from\_tensor\_slices(veri).batch(256)  initializer = tf.random\_normal\_initializer(0., 0.02)  gen = tf.keras.Sequential()  gen.add(tf.keras.layers.Dense(4\*4\*512, use\_bias=False, input\_shape=(100,))) gen.add(tf.keras.layers.BatchNormalization()) gen.add(tf.keras.layers.LeakyReLU()) gen.add(tf.keras.layers.Reshape((4, 4, 512)))  gen.add(tf.keras.layers.Conv2DTranspose(512, (3, 3), strides=(1, 1), padding='same',kernel\_initializer = initializer, use\_bias=False)) gen.add(tf.keras.layers.BatchNormalization()) gen.add(tf.keras.layers.LeakyReLU())  gen.add(tf.keras.layers.Conv2DTranspose(256, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False)) gen.add(tf.keras.layers.BatchNormalization()) gen.add(tf.keras.layers.LeakyReLU())  gen.add(tf.keras.layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False)) gen.add(tf.keras.layers.BatchNormalization()) gen.add(tf.keras.layers.LeakyReLU())  gen.add(tf.keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False)) gen.add(tf.keras.layers.BatchNormalization()) gen.add(tf.keras.layers.LeakyReLU())  gen.add(tf.keras.layers.Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False, activation='tanh')) |

|  |
| --- |
| noise = tf.random.normal([1, 100]) print(noise)  test\_resmi = gen(noise, training=False) plt.imshow((test\_resmi[0, :, :, :] + 1) / 2)  tf.Tensor( [[-1.4108709e+00 -7.9783612e-01 1.6564533e+00 -4.7264487e-01  -6.4100347e-02 1.3462068e+00 3.1129608e-01 -1.7392050e-01  -7.8415209e-01 -4.5682019e-01 -1.3382607e+00 -4.5034388e-01  -3.9992660e-01 -6.2788850e-01 -4.2628849e-01 -1.3032112e+00  -1.2588472e+00 7.1649319e-01 2.5717940e+00 3.3481117e-02  -1.1470229e-01 -6.7221262e-02 1.6650473e+00 -7.7259046e-01  -7.5087124e-01 1.8739203e+00 6.2864864e-01 -5.5647619e-02  -3.0312818e-01 4.4558784e-01 -1.2404317e+00 -5.4492116e-01  4.3289065e-01 -1.7379138e+00 -2.3392439e+00 -1.2989806e+00  -1.4058831e+00 -3.9592564e-01 -1.5095948e-01 1.0343949e+00  -2.4152705e-01 9.6464944e-01 -4.0884650e-01 -1.0388081e+00  8.8523984e-02 6.3123793e-04 5.9804034e-01 -9.0243226e-01  -7.7645618e-01 -3.3977219e-01 1.2248029e+00 -3.9508969e-01  1.0731276e+00 1.0665120e+00 1.6004660e+00 -2.8129581e-01  9.9942409e-02 -1.7574042e+00 -1.2310634e+00 2.1046028e-02  -1.4306396e+00 4.6216202e-01 1.0946506e+00 5.9033609e-03  -2.9508862e-01 -1.1040881e+00 -5.8021575e-01 3.6340800e-01  -1.6659357e-01 -4.6644804e-01 2.2010070e-01 -8.4141254e-01  -1.6794944e+00 -5.7726461e-01 8.7109190e-01 -2.7260438e-01  -2.2384930e+00 -1.0811609e+00 3.3946521e+00 5.3547299e-01  -7.6083642e-01 -1.3143849e+00 -1.0653905e-01 -7.0718479e-01  5.5133468e-01 -8.0286181e-01 -1.9345399e+00 -8.0483741e-01  2.1211745e-01 -1.0608994e+00 -6.9850379e-01 3.7806508e-01  1.3738961e+00 -1.4638900e+00 -1.8609408e-02 1.0125256e+00  -1.6687752e+00 2.1138054e-01 -3.8048160e-01 1.7856196e+00]], shape=(1, 100), dtype=float32)  <matplotlib.image.AxesImage at 0x1abd320e8c0> |

Chart

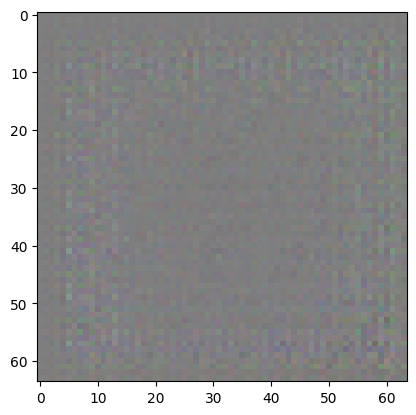
Description automatically generated

|  |
| --- |
| dis = tf.keras.Sequential()  dis.add(tf.keras.layers.Conv2D(64, (7, 7), strides=(2, 2), padding='same',kernel\_initializer =  initializer, input\_shape=[64, 64, 3], use\_bias=False)) dis.add(tf.keras.layers.BatchNormalization()) dis.add(tf.keras.layers.LeakyReLU()) dis.add(tf.keras.layers.Dropout(0.3))  dis.add(tf.keras.layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False)) dis.add(tf.keras.layers.BatchNormalization()) dis.add(tf.keras.layers.LeakyReLU()) dis.add(tf.keras.layers.Dropout(0.3))  dis.add(tf.keras.layers.Conv2D(256, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False)) dis.add(tf.keras.layers.BatchNormalization()) dis.add(tf.keras.layers.LeakyReLU()) dis.add(tf.keras.layers.Dropout(0.3))  dis.add(tf.keras.layers.Conv2D(512, (5, 5), strides=(2, 2), padding='same',kernel\_initializer = initializer, use\_bias=False)) dis.add(tf.keras.layers.BatchNormalization()) dis.add(tf.keras.layers.LeakyReLU()) dis.add(tf.keras.layers.Dropout(0.3))  dis.add(tf.keras.layers.Flatten()) |

|  |
| --- |
| #! dis.add(tf.keras.layers.Dense(1))  print(dis(test\_resmi))  tf.Tensor([[0.00168461]], shape=(1, 1), dtype=float32)  cross\_entropy = tf.keras.losses.BinaryCrossentropy(from\_logits=True)  def dis\_loss(real\_output, fake\_output):  real\_loss = cross\_entropy(tf.ones\_like(real\_output), real\_output)  fake\_loss = cross\_entropy(tf.zeros\_like(fake\_output), fake\_output)  total\_loss = real\_loss + fake\_loss  return total\_loss  def gen\_loss(fake\_output):  return cross\_entropy(tf.ones\_like(fake\_output), fake\_output)  gen\_opt = tf.keras.optimizers.Adam(1e-4) dis\_opt = tf.keras.optimizers.Adam(1e-4) |

|  |
| --- |
| @tf.function def train\_step(images):  noise = tf.random.normal([256, 100])  with tf.GradientTape() as gen\_tape, tf.GradientTape() as dis\_tape:  generated\_images = gen(noise, training = True)  real\_output = dis(images, training = True)  fake\_output = dis(generated\_images, training = True)  gen\_loss\_val = gen\_loss(fake\_output)  dis\_loss\_val = dis\_loss(real\_output, fake\_output)  grad\_gen = gen\_tape.gradient(gen\_loss\_val, gen.trainable\_variables)  grad\_dis = dis\_tape.gradient(dis\_loss\_val, dis.trainable\_variables)  gen\_opt.apply\_gradients(zip(grad\_gen, gen.trainable\_variables))  dis\_opt.apply\_gradients(zip(grad\_dis, dis.trainable\_variables))  from IPython import display EPOCHS = 5000 seed = tf.random.normal([25, 100]) for epoch in range(EPOCHS):  for image\_batch in veri:  train\_step(image\_batch)  predictions = gen(seed, training=False)  fig = plt.figure(figsize=(10,10))  for i in range(predictions.shape[0]):  plt.subplot(5, 5, i+1)  plt.imshow((predictions[i, :, :, :] + 1) / 2)  plt.axis('off')  plt.show()  veri.shuffle(10000)  display.clear\_output(wait=True)  noise = tf.random.normal([1, 100]) test\_resmi = gen(noise, training=False) plt.imshow((test\_resmi[0, :, :, :] + 1) / 2) print(dis(test\_resmi)) |

tf.Tensor([[-4.462775]], shape=(1, 1), dtype=float32)



*Raspberry Pi*

The following steps are followed to operate a camera connected to a Raspberry Pi:

|  |
| --- |
| sudo apt-get update  sudo apt-get install -y libjpeg-dev libv4l-dev imagemagick  sudo apt-get install -y fswebcam |

The camera can be tested using the "raspistill" or "fswebcam" commands. For example, the following command will take an image and save it to the file "image.jpg":

|  |
| --- |
| raspistill -o image.jpg |

The captured photo can be given to the trained model and the result can be obtained. This process is quite simple. The important thing is that the model used can classify well enough.

The captured photo is displayed on a screen connected to the Raspberry Pi:

|  |
| --- |
| sudo apt-get update  sudo apt-get install python3-pygame  pip3 install Pillow  from PIL import Image  image = Image.open(image.jpg')  import pygame  # Pygame window  pygame.init()  screen = pygame.display.set\_mode((800, 600))  screen.blit(image, (0, 0))  pygame.display.flip() |

*References*

1. Teele DW, Klein JO, Rosner B. Epidemiology of otitis media during the first seven years of life in children in greater Boston: a prospective, cohort study. J Infect Dis [Internet]. 1989 [cited 2022 Dec 30];160(1):83–94. Available from: https://pubmed.ncbi.nlm.nih.gov/2732519/

2. Lecun Y, Bengio Y, Hinton G. Deep learning. Vol. 521, Nature. Nature Publishing Group; 2015. p. 436–44.

3. CS231n Convolutional Neural Networks for Visual Recognition [Internet]. [cited 2022 Dec 30]. Available from: https://cs231n.github.io/convolutional-networks/

4. Yao X, Wang X, Wang SH, Zhang YD, Zhang yudongzhang YD, Xujing Yao ieeeorg. A comprehensive survey on convolutional neural network in medical image analysis. [cited 2023 Jan 5]; Available from: https://doi.org/10.1007/s11042-020-09634-7

5. Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. [cited 2022 Dec 30]; Available from: http://code.google.com/p/cuda-convnet/

6. Mielke SJ, Eisner J. Spell Once, Summon Anywhere: A Two-Level Open-Vocabulary Language Model. [cited 2022 Dec 30]; Available from: www.aaai.org

7. Sezgin Z. Efüzyonlu Otitis Media: Tanı ve Tedavi Yaklaşımlarına Genel Bakış Otitis Media with Effusion: Overview of Diagnosis and Treatment Approaches. Pediatric Practice and Research. 2016;4(1):1–11.

8. Kashani RG, Młyńczak MC, Zarabanda D, Solis-Pazmino P, Huland DM, Ahmad IN, et al. Shortwave infrared otoscopy for diagnosis of middle ear effusions: a machine-learning-based approach. Scientific Reports 2021 11:1 [Internet]. 2021 Jun 15 [cited 2022 Dec 30];11(1):1–10. Available from: https://www.nature.com/articles/s41598-021-91736-9