T.C.

MANİSA CELAL BAYAR ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BREAST CANCER DETECTION

Tasarım Projesi / Lisans Bitirme Tezi

HAZIRLAYANLAR

180316016 EDA ÇELEBİ

DANIŞMAN

Dr. Öğr. Üyesi SEVCAN EMEK

MANİSA 2023

T.C.

MANİSA CELAL BAYAR ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Tasarım Projesi / Lisans Bitirme Tezi

KABUL VE ONAY BELGESİ

----------'ın

"----------"

isimli lisans projesi çalışması, aşağıda oluşturulan jüri tarafından değerlendirilmiş ve kabul edilmiştir.

Danışman : ……..

Üye : .....

Üye : .....

Projenin Savunulduğu Tarih : .....

Bilgisayar Mühendisliği Bölüm Başkanı

INGREDIENTS

1.PROBLEM DEFINITION....6
2.ANALYSIS LITERATURE....7
3.USING TECHNOLOGIES....8
3.1 CNN Architecture....8
3.1.1. Input Layer....9
3.1.2. Convolution layer....9
3.1.3. Activation (Relu) layer....10
3.1.4.Pooling Layer....11
3.1.5. Fully Connected Layer....11
3.1.6. Dropout Layer....12
3.1.7. Classification layer....13
3.1.8. Softmax Layer....13
3.1.9. Normalization Layer....13
3.2. Keras....13
3.3. TensorFlow....14
3.4. OpenCV....14
3.5. Deep Learning Algorithms....14
3.5.1. LeNet-5....14
3.5.2. AlexNet....14
3.5.3. VggNet....15
3.5.4. ResNet....15
3.5.5. GoogleNet....15
4.IMPLEMENTATION....16
4.1 Load Data....16
4.2. Converting String Label to categorical....17
4.3. Normalize:....17
4.4. Model Building , Compile and Visualization....17
4.4.1. Model LeNet5....19
4.4.2. Model AlexNet....19
4.4.3. Model VGG16....20
4.5. Train the Model....21
5.RESULTS....21
5.1.Result Lenet5....21
5.2.Result Alexnet....28
5.3.Result VGG16....33
KAYNAKÇA....35

LIST OF FIGURES

Figure 3.1.1. CNN Architecture.....9
Figure 3.1.2.1 Convolution operation.....10
Figure 3.1.3.1. Relu Activation function.....10
Figure 3.1.5.1 Full Connected Layer.....12
Figure 3.1.6.1. (a)Artifical Neural Network (b) Dropout applied neural network..12
Figure 4.2.1. Converting String Label to categorical.....16
Figure 4.4.1. Plot Accuracy And Loss.....17
Figure 4.4.2. Plot Confusion Matrix.....18
Figure 4.4.1.1 Model LeNet5.....18
Figure 4.4.2.1 Model AlexNet.....19
Figure 4.4.3.1 Model VGG16.....19
Figure 4.5.1. Train The Model.....20
Figure 5.1.1 Accuracy Lenet5.....21
Figure 5.1.2 Graph Lenet5.....22
Figure 5.1.3 Accuracy Lenet5(2) 22
Figure 5.1.4 Graph Lenet5.....23
Figure 5.1.5 Matrix Lenet5.....23
Figure 5.1.6 Accuracy Lenet5(3) 23
Figure 5.1.7 Graph Lenet5.....24
Figure 5.1.8 Matrix Lenet5.....24
Figure 5.1.9 Accuracy Lenet5(4) 25
Figure 5.1.10 Accuracy Lenet5(5) 25
Figure 5.1.11 Graph Lenet5.....26
Figure 5.1.12 Matrix Lenet5.....26
Figure 5.1.13 Accuracy Lenet5(6) 27
Figure 5.1.14 Graph Lenet5.....27
Figure 5.1.15 Matrix Lenet5.....28
Figure 5.2.1 Accuracy AlexNet.....28
Figure 5.2.2 Graph AlexNet.....29
Figure 5.2.3. Matrix AlexNet.....29
Figure 5.2.4. Accuracy AlexNet.....29
Figure 5.2.5. Graph AlexNet.....30

Figure 5.2.6. Matrix AlexNet.....30
Figure 5.2.7. Accuracy AlexNet.....31
Figure 5.2.8 Accuracy AlexNet.....31
Figure 5.2.9. Graph AlexNet.....32
Figure 5.2.10 Matrix AlexNet.....33
Figure 5.3.1. Accuracy VGG16.....34
Figure 5.3.2 Graph VGG16.....34
Figure 5.3.3 Matrix VGG16.....34

ABSTRACT

Breast cancer is one of the most common types of cancer and causes a large number of deaths in women. Breast cancer results from the breakdown of certain cells that proliferate and often form a mass called a tumor. Tumors can be benign (non-cancerous) or malignant (cancerous). Tests such as MRI (magnetic resonance imaging), mammography, ultrasound, and biopsy are commonly used to diagnose breast cancer. Since the data is three different labels, the estimation is divided into three categories (malignant, benign or normal). This is defined as a classification problem in machine learning. This study aims to classify whether breast ultrasound images are benign, malignant, or normal. The methodology used includes the classification model and the method of neural networks. Python modules are used to hide external data sets to recognize the data to comprehend the data well and to consider how the data will be handled in different ways. For this reason, neural network methods and classification methods were used to achieve high accuracy in the classification of breast cancer.

Keywords: Breast cancer, neural networks, deep learning, classification.

1.PROBLEM DEFINITION

It is known that breast cancer is the most common type of cancer in women. It can affect not only women but also men, even if they are very small. According to Professor Aghzadi Rajaa, one of the experts in this field, one in 10 women is at risk of breast cancer throughout his life. This is enough to make breast cancer a major public health problem. Although there are many effective treatment for breast cancer, the disease can be repeated in patients after treatment and these new treatment has led to the investigation of strategies (Ouardirhi, 2020).

Rapidly developing technology has contributed to the early diagnosis of vital importance for breast cancer, and projects in which classification algorithms and deep learning can achieve high accuracy have been developed.

In this study, we aim to classify breast ultrasound images into three categories (malignant, benign or normal).

ANALYSIS LITERATURE

Ref.	Dataset	Features	Method	Accuracy
[3]	Wisconsin Breast Cancer (original) dataset (WBCD)	11 attributes 699 instances	SVM, C4.5, NB, k-NN	Best accuracy for SVM 97.13%
[2]	Two datasets	First dataset: 11 attributes with 699 instances Second dataset: 117 attributes with 102294 instances	SVM with three functions and two features: bagging and boosting	96.85%, 95%
[5, 6]	Wisconsin Breast Cancer dataset	30 attributes with 699 instances	Random Forest, Naive Bayes, SVM, KNN	97.9%
[4]	Wisconsin Prognostic Breast Cancer dataset	32 attributes with 194 instances	Compare (K-means, EM, PAM, and fuzzy c-means) with SVM and C5.0	Better results for SVM with accuracy 97%
[1]	Wisconsin Diagnostic Breast Cancer (WDBC) dataset	30 attributes of 569 patients with 569 instances	Compare supervised learning (SL) with semi-supervised learning (SSL) for 9 algorithms	K-NN (SL = 98% & SSL = 97%) and logistics regression (SL = 97% & SSL = 98%)
[8]	Wisconsin Breast Cancer (original) dataset (WBCD)	11 attributes with 699 instances	Boosting Artificial Neural Network (BANN) with two SVMs	100%
[7]	Wisconsin Breast Cancer dataset (WBCD)	32 attributes with 569 instances	SVM with stochastic gradient descent optimization, simple logistic regression learning, and multilayer perceptron network	99.44%
[9]	DDSM, IN breast, and BCDR	DDSM: 5316 images, 641 cases of patients IN breast: 200 images for 50 cases BCDR: 600 images from 300 patients	CNN	97.35%, 95.50%, 96.67% for three datasets respectively
[10]	WBCD	9 attributes with 5699 instances	Deep neural network (DNN)	98.62%

[11]	The Datasets were collected at two medical institutions	990 images, 540 Malignant masses, and 450 benign lesions	CNN	87.68%
[12]	Data was collected from (2010-2016) at five imaging sites affiliated with New York University School of Medicine	1,001,093 images from 141,473 patients	CNN	The accuracy was calculated based on area under a curve (AUC)
[13]	Data was collected independently	12,000 cases, including 4000 samples proven cancers	CNN	The accuracy was calculated based on area under a curve (AUC)
[14]	Private dataset	67,520 mammographic images from 16,968 women	CNN	95%

3.USING TECHNOLOGIES

3.1 CNN Architecture

CNN is a deep learning algorithm, which is generally used in image processing and receiving images as inputs. This algorithm, which captures and classifies features of visuals with different operations, consists of different layers. Convolutional Layer, Pooling, and Fully Connected, which passes through these layers, are subjected to different processes and come to the consistency that will enter the deep learning model. When creating CNN models, we do not deal with the data pre -processing section compared to classical machine learning algorithms, since we deal with universal (irregular) data.

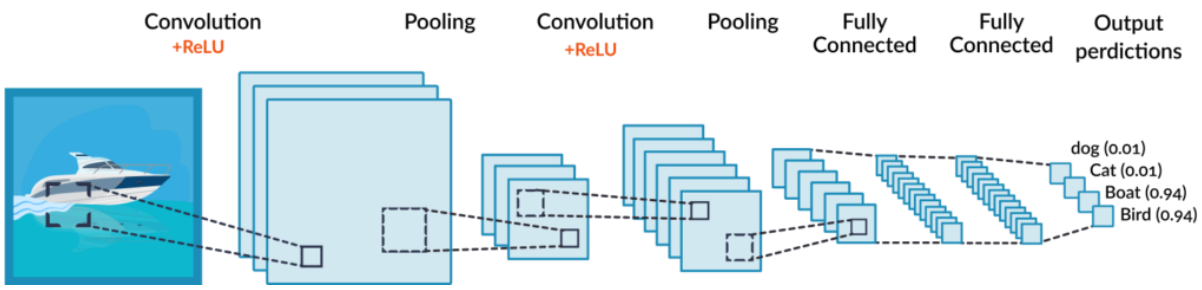

Figure 3.1.1. CNN Architecture

3.1.1. Input Layer

The Input Layer is known as the data input layer. The data set to be created in neural networks should be determined according to the architecture of the network. The data for each item comes one after the other to form a dataset. The size of this dataset increases the network speed, test time and memory requirement.

3.1.2. Convolution layer

It is the layer that forms the basis of convolutional neural networks. In this layer, it is aimed to reveal the distinctive features of the input by hovering a predetermined filter on the input data. As a result of filtering, it provides a smaller matrix than the input data.

Convolution is made by using filters of different sizes in deep learning algorithms. There are 11x11 matrices used in AlexNet. On the other hand, 7x7 filters are used in ZfNet. In GoogleNet, VggNet, ResNet deep learning architectures, filters such as 5x5, 3x3, 2x2, 1x1 were used. A matrix of size MxM is applied to a matrix of size NxN. Figure 11 shows the convolution operation. Convolution processing is an important part of deep convolution neural networks. With the filtering process performed in this layer, the properties of the item will be better reflected on the network. The filter to be chosen will directly affect the training process and success of the network.

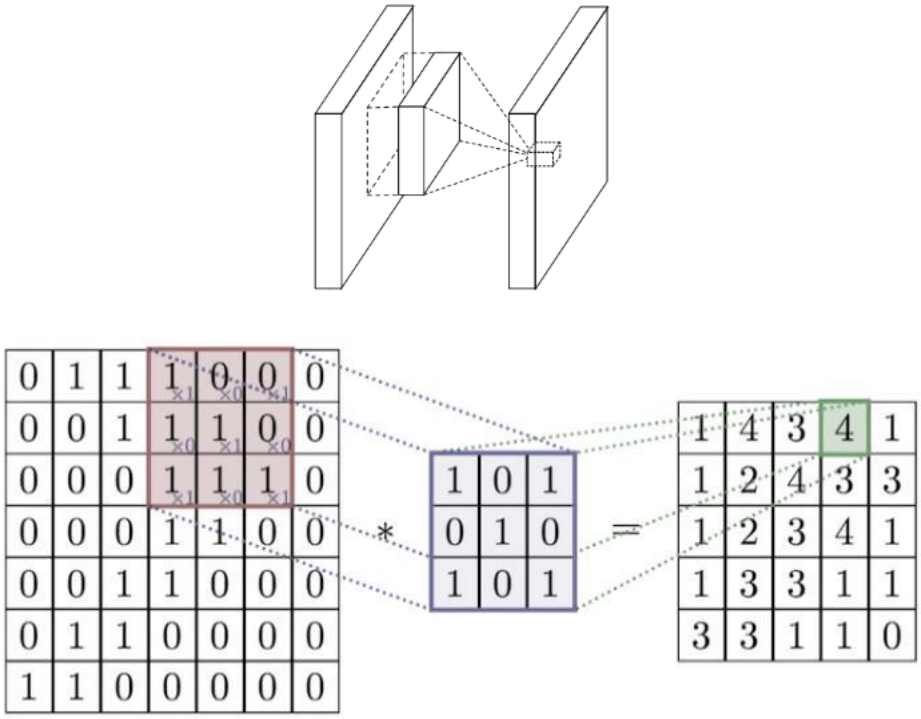

Figure 3.1.1.2 Convolution operation

3.1.3. Activation (Relu) layer

Usually, the activation layer comes after the convolution layer. Simoid, hyperbolic tangent, sine, step, threshold value functions are used as activation functions. Nonlinear transformations are used in multilayer artificial neural networks. Although there are many activation functions, Relu (Rectified Linear Unit) $f(x)$=max (0, x) function is used in deep learning architectures.

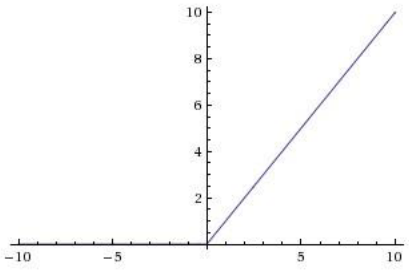

Figure 3.1.3.1. Relu Activation function

If the value obtained as a result of the activation process is negative, it takes the value 0, if it is positive, it takes the value 1.

3.1.4.Pooling Layer

The pooling layer, which is located after the relu layer, reduces the data and reduces the input size for the next layer in the network. With the pooling process, losses occur on the data. However, the decrease in the amount of data circulating on the network makes the network faster. The decrease in the amount of data in the network means that the amount of computation in the network and the amount of memory to be used decrease. In the pooling layer is processed using an appropriate filter. An NxN size filter to be used will be navigated through the data pool to find the appropriate value remaining in the matrix. The most used average (average pooling) and maximum value (max pooling) operations to obtain the value. In max pooling, the largest value remaining in the NxN size matrix will be the 1x1 size new value of the new data matrix to be created. In average pooling, it includes the 1x1 size value of the new data matrix to be obtained by averaging all the remaining values in the NxN matrix.

3.1.5. Fully Connected Layer

All neurons in this layer appear as a string. All neurons in the layer are fully connected to all the activations in the previous layer connected to this layer. The feature of the fully connected layer emerges depending on the previous layer. It is determined that the properties that will determine the object are related to which class. If a pattern is to be considered; The features that enable a person in the model to be identified will be located in the neuron with a high value in the activation maps. A fully connected layer looks at high-level properties that have a high degree of relevance to a class. By looking at the neurons with weights indicating these properties, it is revealed which class they belong to. Figure 14 shows an example of a fully connected layer structure.

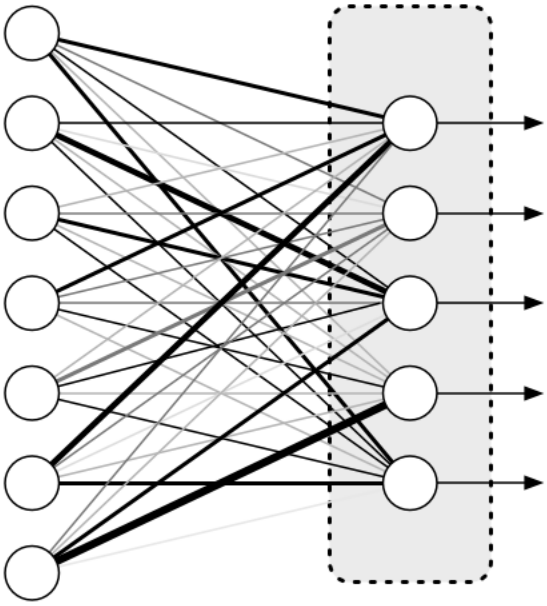

Figure 3.1.5.1 Full Connected Layer

3.1.6. Dropout Layer

In multi-layer artificial neural networks, while the neural network is being trained, the memorization of the network, called hyperlearning, takes place. This is undesirable. It is the elimination of some memorizing nodes in the network to prevent the network from being memorized. Thus, the memorization of the network is tried to be eliminated. The dropout layer has been proposed by Hinton et al as an editing layer for fully linked layers. It has been proven by test performances that dropout increases neural network modulation ability. Figure 15 shows the multilayer neural network and the dropout neural network structure.

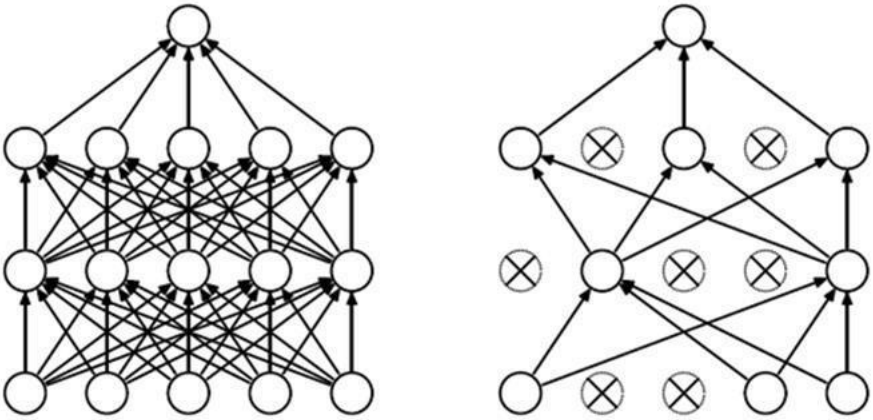

Figure 3.1.6.1. (a)Artifical Neural Network (b) Dropout applied neural network

3.1.7. Classification layer

The classification layer after the fully connected layer produces as many results as the number of items to be classified. Each of these results represents a class. Although different types of classifiers are used for the classifier layer, known as the last layer, the softmax classifier is generally used.

3.1.8. Softmax Layer

The softmax layer, which is the classification layer, takes the input data from the fully connected layer before it and uses it to classify. Indicates that a probabilistic input data belongs to a particular class. Generates value for which class it is closer to. It extracts the probability value for each class by performing probabilistic computation generated within the deep learning network. For these operations, cross-ethnropy is used.

3.1.9. Normalization Layer

Training deep convolutional neural networks requires a serious computational process.
One way to reduce the training time is to normalize the activation of neurons. The normalization layer is very effective in stabilizing the hidden layers in feedback networks. Normalization is usually performed after the Relu layer. Normalization affects the performance of the network. Therefore, the data coming from the layers should be in a certain order. Input data may be too large or too small. It is important for training and process to use these values in a certain range by normalizing them. Input data should be normalized and represented within a certain range.

3.2. Keras

Keras is a high -level deep learning library written in Python programming language. The most important feature is that it can also work on the tensorflow library.

3.3. TensorFlow

TensorFlow is an open source deep learning library supported by Google. It is generally preferred for numerical calculations for image processing.

3.4. OpenCV

OpenCV library contains more than 2500 algorithms for image processing and machine learning. With these algorithms, procedures such as face recognition, distinguishing objects, detecting human movements, object classification, plate recognition, three-dimensional image operation, image comparison, and optical character identification OCR (Optical Character Recognition) can be done easily.

3.5. Deep Learning Algorithms

3.5.1. LeNet-5

It is known as the first convolutional neural network prepared by LeCun. LeCun named this network they developed to identify numbers on bank checks, LeNet. A smoothing layer with 10 classes is used because it classifies numbers 0-9. Average pooling is used in this network.Within the LeNet architecture, it consists of a fully connected layer, relu (activation) layer, pooling layer, and softmax layers.

3.5.2. AlexNet

AlexNet deep learning architecture came first in the 2012 ILSVRC ImageNet Large-scale image recognition competition held in 2012. It reduced the error rate in pattern recognition from 26% to 15%. This success has made AlexNet one of the most well-known deep learning architectures. The activation function (Relu) was used for nonlinear functions. Since this activation function is faster than the classical tanh function, it has been used to shorten the training time. A dropout layer was used to prevent over-learning and stuttering during the training process. Gradient descent model was used for weight delays and momentum values. There are 5 basic convolution layers in the AlexNet deep learning network, which consists of 25 layers. Generally, an

activation layer, the relu layer, is used after each convolution layer. In addition, there are input layer, normalization layer, pooling layer, dropout layer, fully-connected layer, soft-connected (SoftMax) layer and output layer.

3.5.3. VggNet

It is a deep learning model developed in 2014. It showed a very successful performance with an error rate of 7.3% in the ImageNet 2014 competition. This architecture, designed by Simonyan and Zisserman at Oxford University, revealed 6 different architectures. It consists of 11, 13, 16, 19 convolution layers in 6 different models. Unlike the convolutions in previous deep learning architectures, 2x2 and 3x3 filters are applied here. In this architecture, there are 3 cascading fully connected (FC-FullConnected) layers. There are 1000 neurons in the last fully connected layer, and the classification layer to be produced for output has a soft-connected (SoftMax) layer. Among these models, Vgg-16 and Vgg-19 models are more common in the literature.

3.5.4. ResNet

Microsoft Resnet was the winner of the ILSVRC ImageNet competition held in 2015. It achieved an error rate of 3.6% in this competition. While humans classify images with an average error rate of 5-10%, an error rate of 3.6% has shown that they perform better visual recognition than humans. The number of layers of this architecture, which has a depth above the previous architectures, is higher than the number of layers in other deep learning architectures. In Microsoft ResNet architecture, it is created from a block (Residual Block) that is fed once a residual value (Residual Value) between two RELUs and linear layers. With this structure, it was thought that learning would take place faster.

3.5.5. GoogleNet

It was the winner of the ILSVRC competition held in 2014. GoogleNet (Inception) has a complex structure. It achieved a high performance with a low error rate of 5.7%. It has a depth of 22 layers and GoogleNet has a structure consisting of 144 layers. By filtering in different dimensions with the Inception module, it has revealed a different formation from the previous deep learning architectures. With the Inception module, different sizes of filters were

made. These filters are used to reduce size. It contains 12 times less parameters than AlexNet. The number of layers used may differ according to the individual building blocks. The filter elements in the Inception module are 1x1, 3x3, 5x5. A deep structure has been created by differentiating from the successive layered structure in other deep learning architectures. A modular filtering logic has been introduced that creates this depth.

4.IMPLEMENTATION

4.1Load Data

We load the images from our dataset according to the image size we have determined. Our goal is to classify the images we upload into 3 categories.

```
categories = ['benign', 'malignant', 'normal']
x = []
y = []
image_size = 32

for i in categories:
    folderPath = os.path.join('Dataset/', i)
    for j in tqdm(os.listdir(folderPath)):
        image = cv2.imread(os.path.join(folderPath, j),0)
        image = cv2.resize(image, (image_size, image_size))
        x.append(image)
        y.append(i)


x = np.array(x)
y = np.array(y)
```

We define the training and test sets.

test_size: With this parameter we can set the test/training size radio. We use 0.20 for all models.

random_state: The random state is simply the lot number of the set generated randomly in any operation. We can specify this lot number whenever we want the same set again.

```
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.20, random_state=42)
```

Figure 4.1.1. Train and Test Split

4.2. Converting String Label to categorical

['benign', 'malignant', 'normal']→ [0,1,2]

0 → 1.0.0.
1 → 0.1.0.
2 → 0.0.1.

```
y_train_new = []
y_test_new = []

for i in Y_train:
    y_train_new.append(categories.index(i))
Y_train = to_categorical(y_train_new)

for i in Y_test:
    y_test_new.append(categories.index(i))
Y_test = to_categorical(y_test_new)
```

Figure 4.2.1. Converting String Label to categorical

4.3. Normalize:

```
X_train_scaled /= 255
X_test_scaled /= 255
```

4.4. Model Building , Compile and Visualization

To build a neural network, it is necessary to set the layers of the model and then compile the model. The basic block of a neural network is the layer. Layers extract representations from the data.

Before starting the training of the model, it is necessary to compile the model with the compile() method. While compiling the model, the loss function and optimizer are determined.

The first graph shows us the learning accuracy of the test set and the train set. The second graph shows us the cost function.

```
def plot_accuracy_loss(self, history):
    fig = plt.figure(figsize=(10, 5))

    plt.subplot(221)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title("model accuracy")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend(['train', 'test'], loc='upper left')

    plt.subplot(222)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title("model loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

Figure 4.4.1. Plot Accuracy And Loss(

The function takes one argument. This matrix shows how much data for which class is estimated for which class with the heatmapping method. We have 3 different types so we use 3X3 matrix.

```
def plot_cofusion_matrix(self, conf):
    info = [
        'benign',     # 0
        'malignant',  # 1
        'normal',     # 2

    ]
    plt.figure(figsize=(10, 10))
    ax = sns.heatmap(conf, annot=True, square=True, xticklabels=info, yticklabels=info)
    ax.set_ylabel('Actual', fontsize=40)
    ax.set_xlabel('Predicted', fontsize=40)

    plt.show()
```

Figure 4.4.2. Plot Confusion Matrix

4.4.1. Model LeNet5

```
def lenet5(self):
    model = Sequential()
    model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh',
                     input_shape=(32, 32, 1), padding='valid'))
    model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
    model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
    model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
    model.add(Flatten())
    model.add(Dense(120, activation='tanh'))
    model.add(Dense(84, activation='tanh'))
    model.add(Dense(3, activation='softmax'))

    opt = SGD(learning_rate=0.001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

Figure 4.4.1.1 Model LeNet5

4.4.2. Model AlexNet

```
def alexNet(self):
    model =Sequential([
        Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4), activation='relu',
                           input_shape=(227, 227, 3)),
        BatchNormalization(),
        MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
        Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding="same"),
        BatchNormalization(),
        MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
        Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
        BatchNormalization(),
        Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
        BatchNormalization(),
        Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
        BatchNormalization(),
        MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
        Flatten(),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(3, activation='softmax')
    ])
```

Figure 4.4.2.1 Model AlexNet

4.4.3. Model VGG16

```
def VGG16(self):
    model = Sequential()
    model.add(Conv2D(64, (3, 3), input_shape=(32, 32, 3), padding='same',
              activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same' ))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(3, activation='softmax'))

opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
return model
```

Figure 4.4.3.1 Model VGG16

4.5. Train the Model

Training sets are used to train the model. The model is evaluated with validation data.

```
history = model.fit(x=X_train_scaled, y=Y_train,
                    batch_size=32,
                    epochs=20,
                    validation_split = 0.20,
                    verbose=1)
test_loss,test_acc = model.evaluate(X_test_scaled, Y_test)
print("ACCURACY:", test_acc)
```

Figure 4.5.1. Train The Model

5. RESULTS

5.1. Result Lenet5

İmage_size = 32,32,1
Activation function: sigmoid
Epoch: 1000
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: AveagePooling2D
Padding:valid

```
ACCURACY: 0.46305418014526367
13/13 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.46      1.00      0.63       188
           1       0.00      0.00      0.00       112
           2       0.00      0.00      0.00       106

    accuracy                           0.46       406
   macro avg       0.15      0.33      0.21       406
weighted avg       0.21      0.46      0.29       406
```

Figure 5.1.1 Accuracy Lenet5

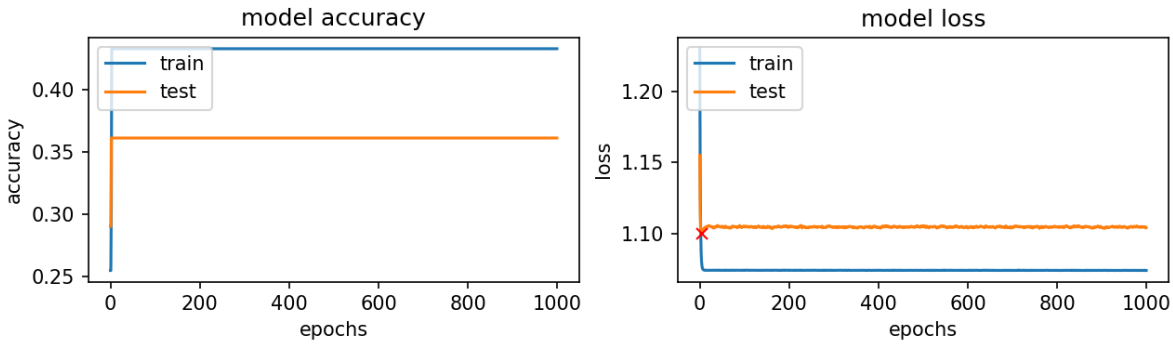

Figure 5.1.2 Graph Lenet5

İmage_size = 32,32,1
Activation function : tanh
Epoch : 1000
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: AveagePooling2D
Padding:valid

```
ACCURACY: 0.6403940916061401
13/13 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.67      0.76      0.71       188
           1       0.57      0.58      0.57       112
           2       0.66      0.50      0.57       106

    accuracy                           0.64       406
   macro avg       0.63      0.61      0.62       406
weighted avg       0.64      0.64      0.64       406
```

Figure 5.1.3 Accuracy Lenet5(2)

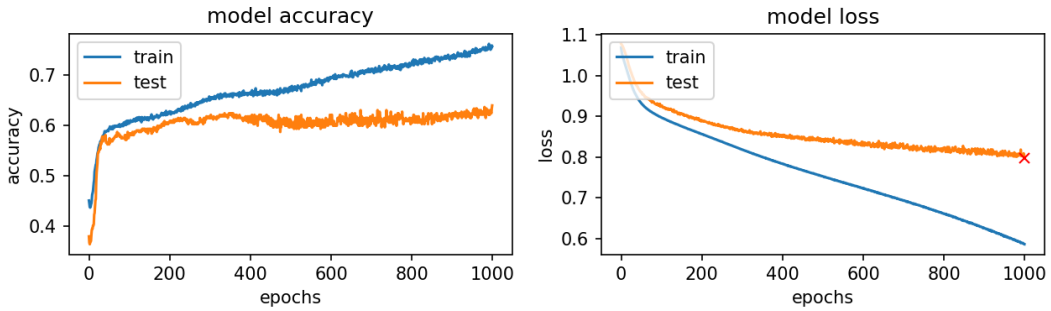

Figure 5.1.4 Graph Lenet5

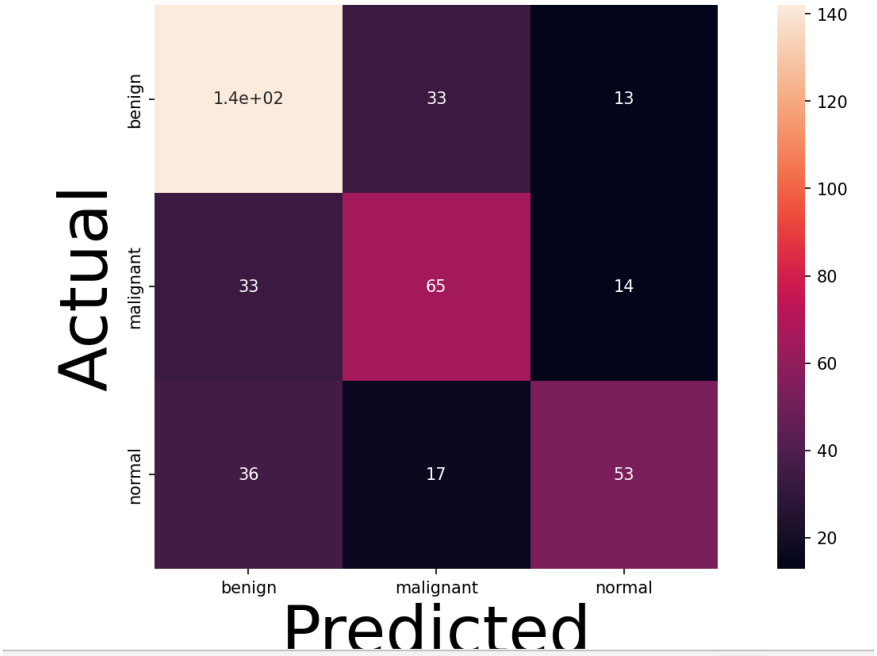

Figure 5.1.5 Matrix Lenet5

İmage_size = 32,32,1
Activation function : relu
Epoch : 1000
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: AveagePooling2D
Padding:valid

```
ACCURACY: 0.6157635450363159
13/13 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.67      0.64      0.65       188
           1       0.49      0.74      0.59       112
           2       0.81      0.44      0.57       106

    accuracy                           0.62       406
   macro avg       0.66      0.61      0.61       406
weighted avg       0.66      0.62      0.62       406
```

Figure 5.1.6 Accuracy Lenet5(3)

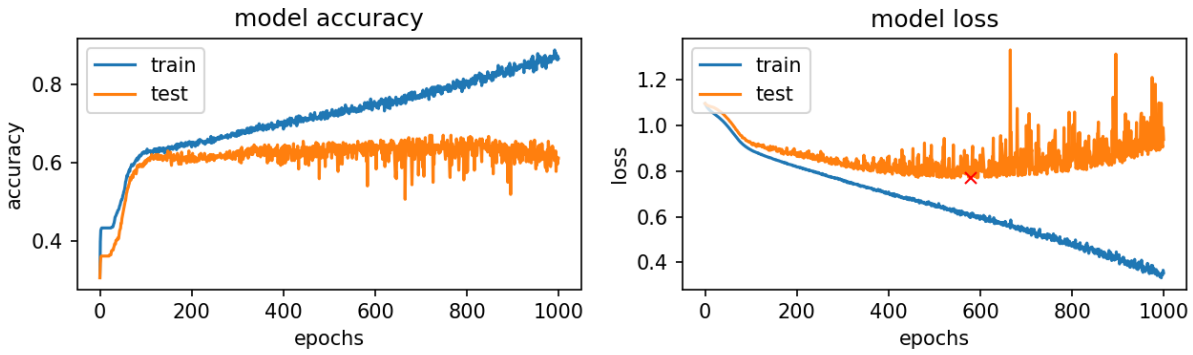

Figure 5.1.7 Graph Lenet5

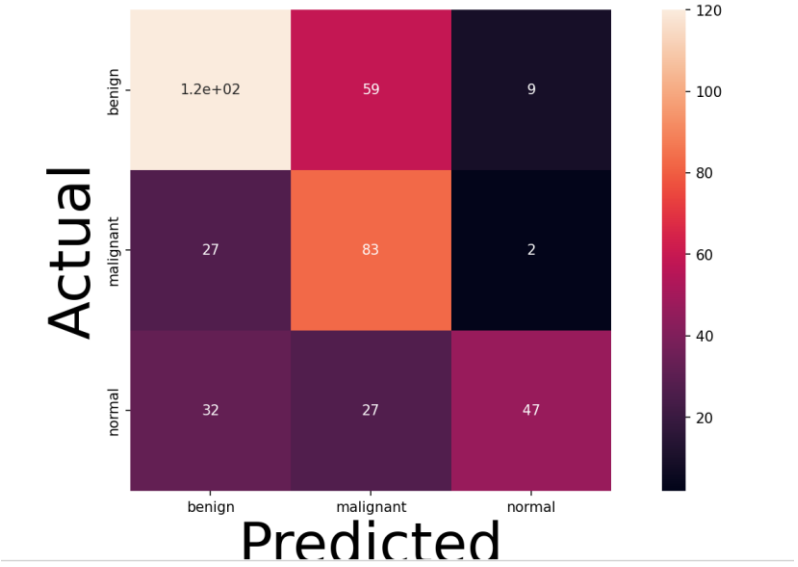

Figure 5.1.8 Matrix Lenet5

İmage_size = 32,32,1
Activation function : tanh
Epoch : 1000
Test_size and validation_split:0.20
Optimizer: Adam
Pooling: AveagePooling2D
Padding:valid

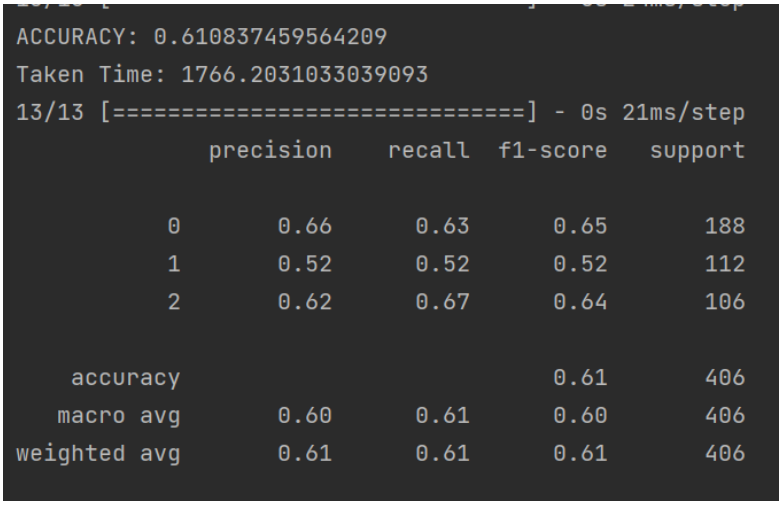

Figure 5.1.9 Accuracy Lenet5(4)

Activation function : tanh
İmage_size = 32,32,1
Epoch : 1000
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: MaxPooling2D
Padding:valid

```
ACCURACY: 0.7192118167877197
Taken Time: 3003.864510536194
13/13 [==============================] - 1s 19ms/step
              precision    recall  f1-score   support

           0       0.76      0.75      0.75       188
           1       0.68      0.64      0.66       112
           2       0.69      0.75      0.72       106

    accuracy                           0.72       406
   macro avg       0.71      0.71      0.71       406
weighted avg       0.72      0.72      0.72       406
```

Figure 5.1.10 Accuracy Lenet5(5)

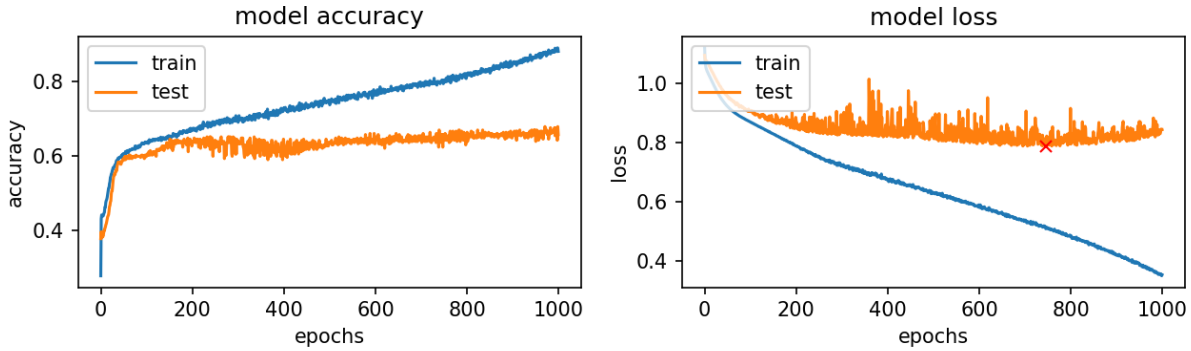

Figure 5.1.11 Graph Lenet5

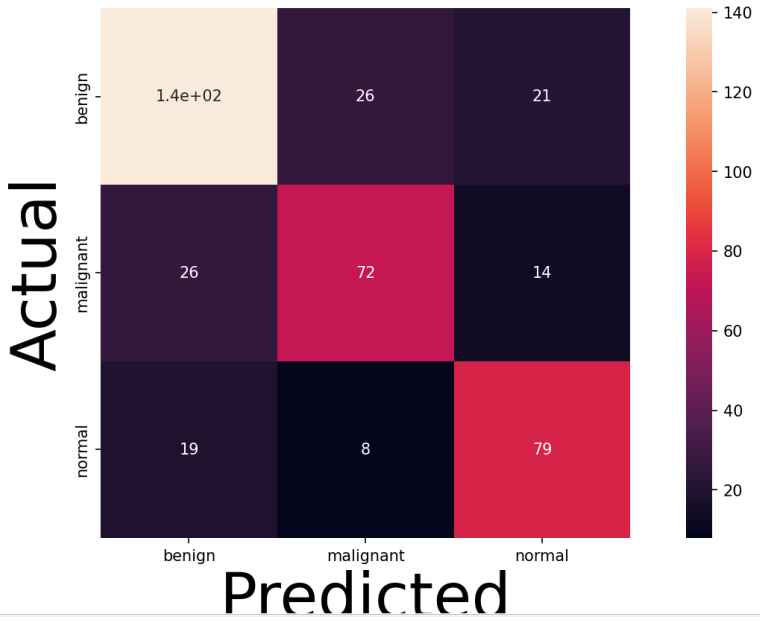

Figure 5.1.12 Matrix Lenet5

İmage_size = 32,32,1
Activation function: tanh
Epoch: 1000
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: MaxPooling2D
Padding:same

```
ACCURACY: 0.7142857313156128
Taken Time: 2963.3499085903168
13/13 [==============================] - 1s 3ms/step
              precision    recall  f1-score   support

           0       0.78      0.73      0.76       188
           1       0.63      0.69      0.66       112
           2       0.69      0.71      0.70       106

    accuracy                           0.71       406
   macro avg       0.70      0.71      0.71       406
weighted avg       0.72      0.71      0.72       406
```

Figure 5.1.13 Accuracy Lenet5(6)

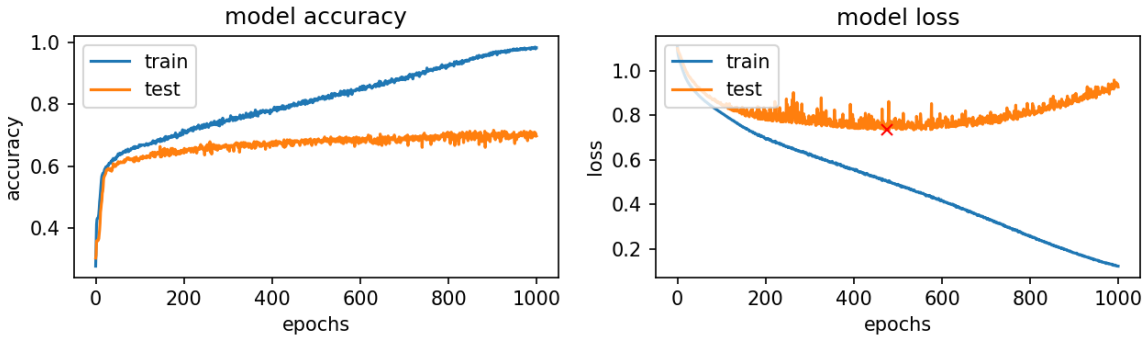

Figure 5.1.14 Graph Lenet5

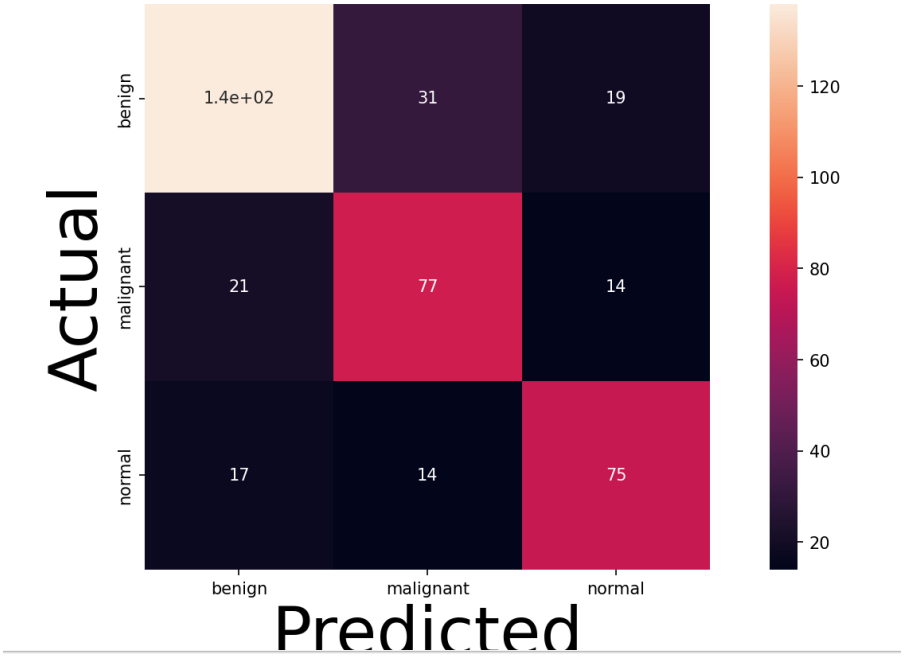

Figure 5.1.15 Matrix Lenet5

5.2. Result Alexnet

İmage_size = 128,128,1
Activation function: relu
Epoch: 20
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: MaxPooling2D
Padding:same

```
ACCURACY: 0.7364531755447388
Taken Time: 340.43457102775574
13/13 [==============================] - 2s 126ms/step
              precision    recall  f1-score   support

           0       0.80      0.76      0.78       188
           1       0.70      0.67      0.68       112
           2       0.68      0.76      0.72       106

    accuracy                           0.74       406
   macro avg       0.72      0.73      0.73       406
weighted avg       0.74      0.74      0.74       406
```

Figure 5.2.1 Accuracy AlexNet

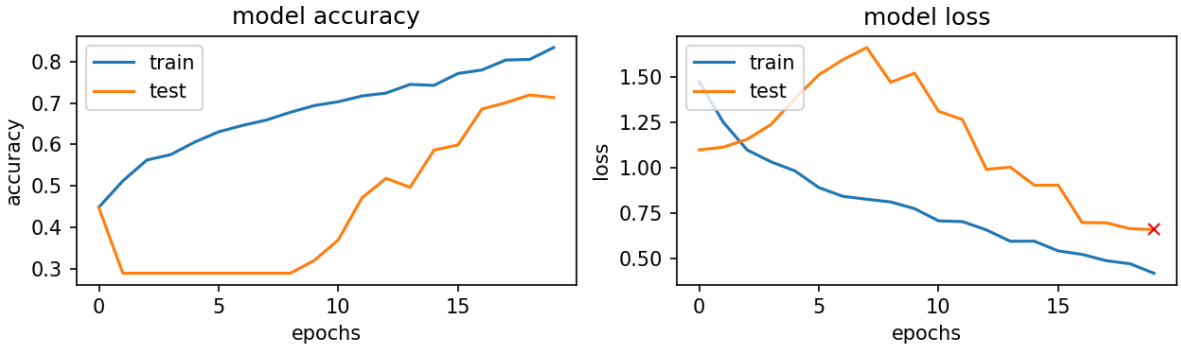

Figure 5.2.2 Graph AlexNet

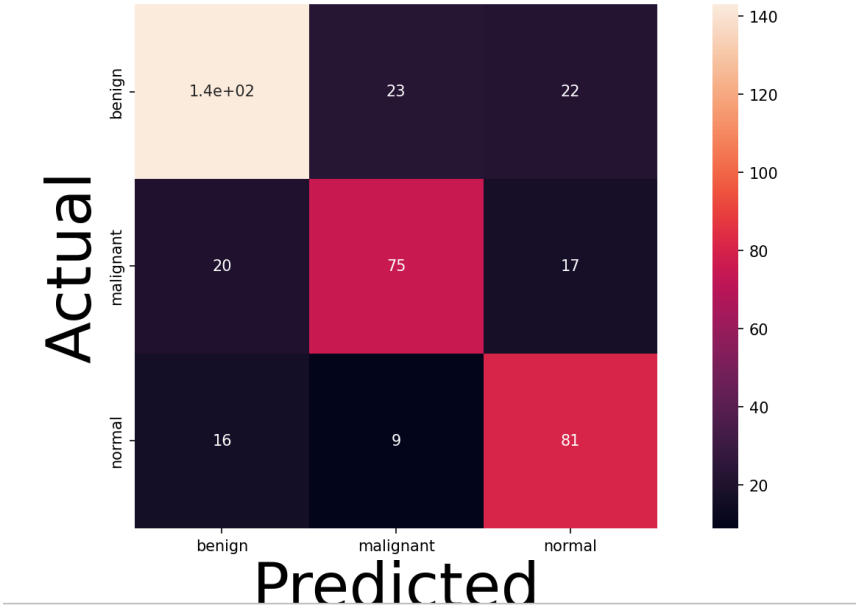

Figure 5.2.3. Matrix AlexNet

İmage_size = 227,227,1
Activation function: relu
Epoch: 20
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: MaxPooling2D
Padding:same

```
ACCURACY: 0.7266010046005249
Taken Time: 1261.841706752777
13/13 [==============================] - 7s 476ms/step
              precision    recall  f1-score   support

           0       0.84      0.68      0.75       188
           1       0.72      0.66      0.69       112
           2       0.62      0.88      0.73       106

    accuracy                           0.73       406
   macro avg       0.73      0.74      0.72       406
weighted avg       0.75      0.73      0.73       406
```

Figure 5.2.4. Accuracy AlexNet

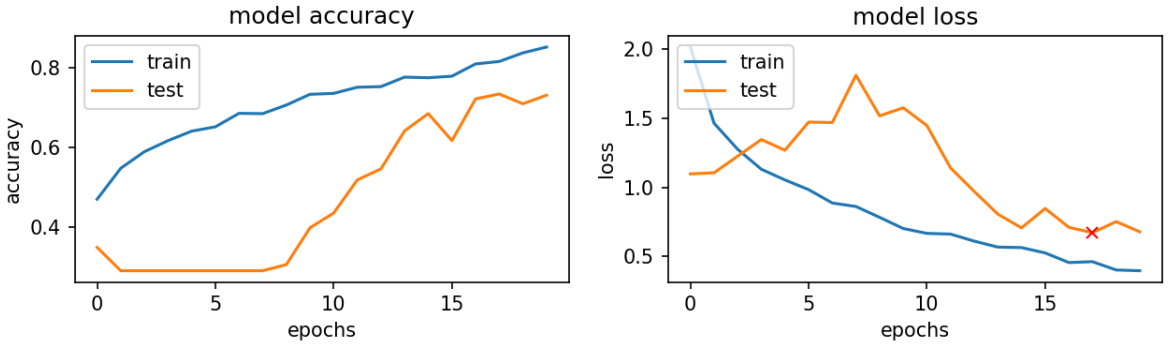

Figure 5.2.5. Graph AlexNet

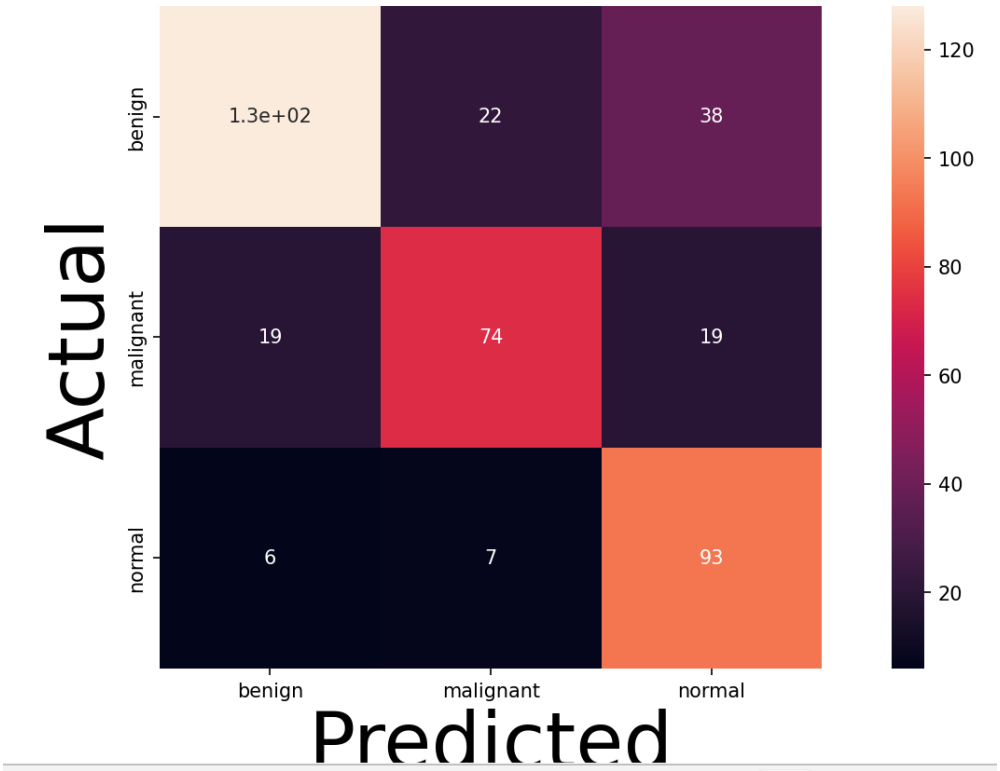

Figure 5.2.6. Matrix AlexNet

İmage_size = 128,128,1
Activation function: relu
Epoch: 20
Test_size and validation_split:0.20
Optimizer: Adam
Pooling: MaxPooling2D
Padding:same

```
ACCURACY: 0.5591133236885071
Taken Time: 450.0337393283844
13/13 [==============================] - 2s 150ms/step
              precision    recall  f1-score   support

           0       0.63      0.52      0.57       188
           1       0.44      0.79      0.56       112
           2       0.82      0.40      0.54       106

    accuracy                           0.56       406
   macro avg       0.63      0.57      0.55       406
weighted avg       0.63      0.56      0.56       406
```

Figure 5.2.7. Accuracy AlexNet

İmage_size = 128,128,1
Activation function: relu
Epoch: 50
Test_size and validation_split:0.20
Optimizer: sgd
Pooling: MaxPooling2D
Padding:same
Learning_rate = 0.0001

```
ACCURACY: 0.7339901328086853
Taken Time: 1040.891269683838
13/13 [==============================] - 2s 143ms/step
              precision    recall  f1-score   support

           0       0.74      0.81      0.77       188
           1       0.68      0.71      0.70       112
           2       0.80      0.61      0.70       106

    accuracy                           0.73       406
   macro avg       0.74      0.71      0.72       406
weighted avg       0.74      0.73      0.73       406
```

Figure 5.2.8 Accuracy AlexNet

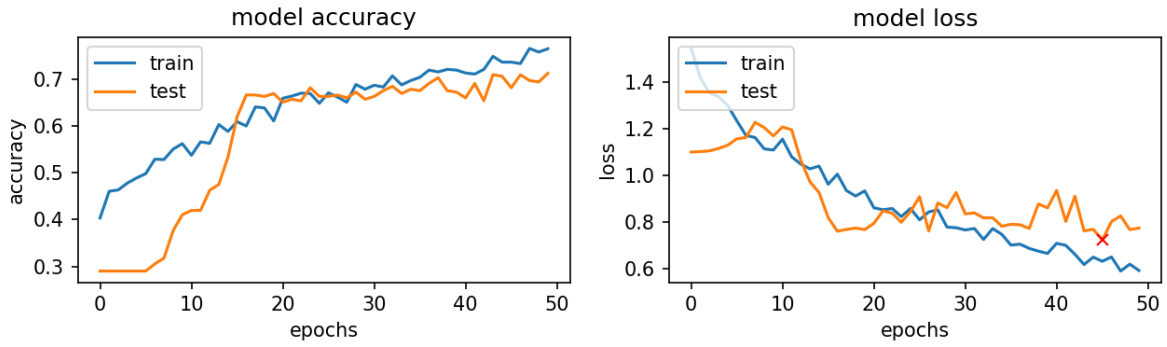

Figure 5.2.9. Graph AlexNet

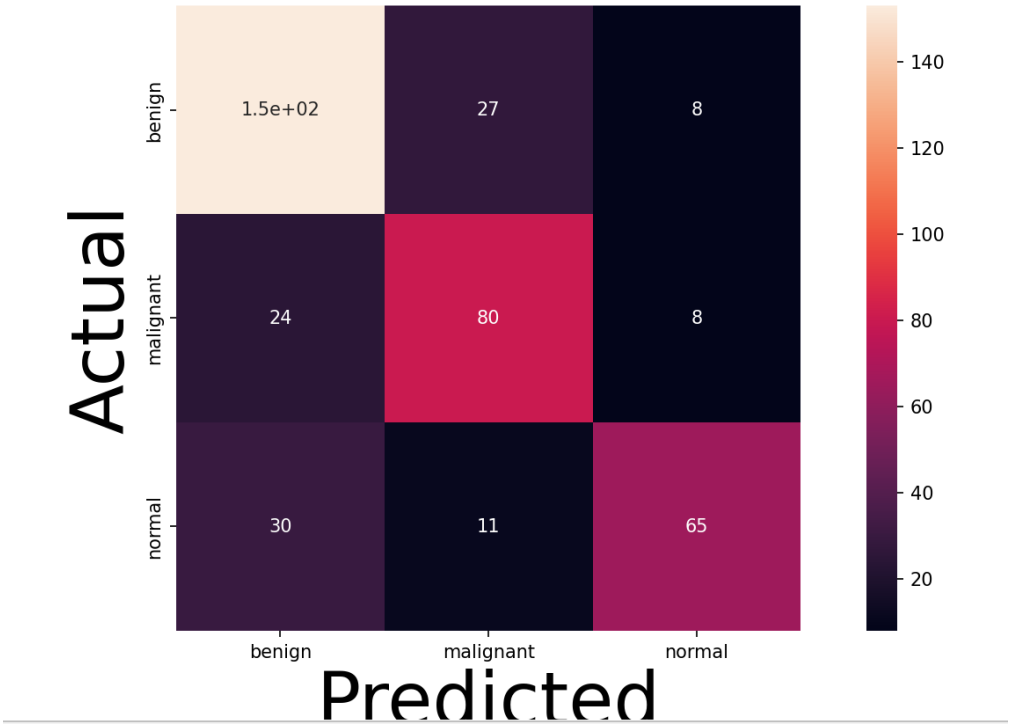

Figure 5.2.10 Matrix AlexNet

5.3. Result VGG16

İmage_size = 224,224,1
Activation function: relu
Epoch: 20
Test_size and validation_split:0.20
Optimizer: Adam
Pooling: MaxPooling2D
Padding:same

```
ACCURACY: 0.46305418014526367
Taken Time: 845.1812410354614
13/13 [==============================] - 2s 158ms/step
              precision    recall  f1-score   support

           0       0.46      1.00      0.63       188
           1       0.00      0.00      0.00       112
           2       0.00      0.00      0.00       106

    accuracy                           0.46       406
   macro avg       0.15      0.33      0.21       406
weighted avg       0.21      0.46      0.29       406
```

Figure 5.3.1. Accuracy VGG16

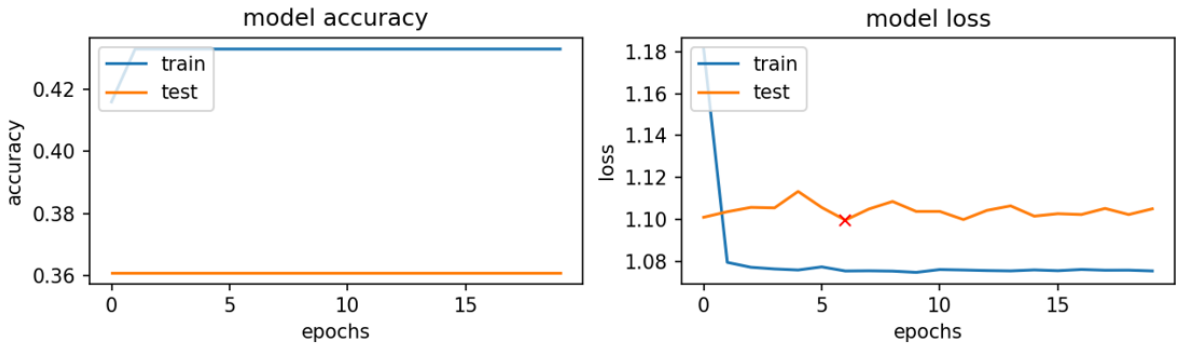

Figure 5.3.2 Graph VGG16

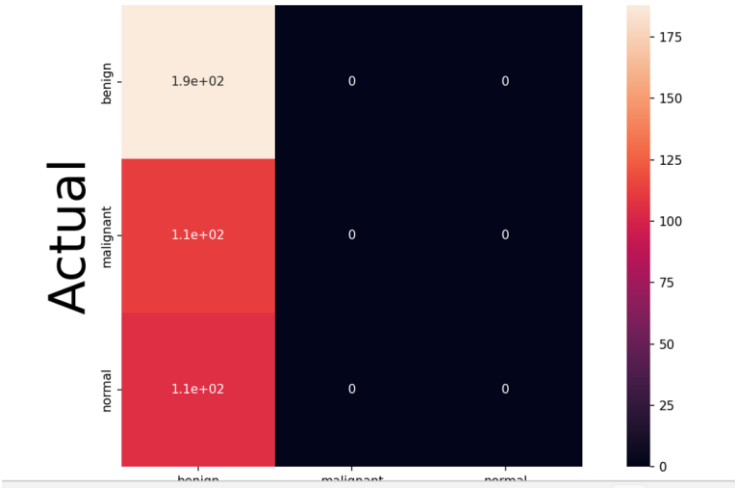

Figure 5.3.3 Matrix VGG16

KAYNAKÇA

[1] N. Al-Azzam and I. Shatnawi, "Comparing supervised and semi-supervised Machine Learning Models on Diagnosing Breast Cancer," Ann. Med. Surg., vol. 62, no. December 2020, pp. 53–64, 2021.

[2] M. W. Huang, C. W. Chen, W. C. Lin, S. W. Ke, and C. F. Tsai, "SVM and SVM ensembles in breast cancer prediction," PLoS One, vol. 12, no. 1, pp. 1–14, 2017.

[3] H. Asri, H. Mousannif, H. Al Moatassime, and T. Noel, "Using Machine Learning Algorithms for Breast Cancer Risk Prediction and Diagnosis," Procedia Comput. Sci., vol. 83, no. Fams, pp. 1064–1069, 2016.

[4] R. Rawal, "BREAST CANCER PREDICTION USING MACHINE LEARNING," J. Emerg. Technol. Innov. Res., vol. 7, no. 5, 2020.

[5] Y. K. and M. Bahaj, "Applying Best Machine Learning Algorithms for Breast Cancer Prediction and Classification," in International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS), 2018, pp. 1–5.

[6] Y. K. and M. Bahaj, "Feature Selection with Fast Correlation-Based Filter for Breast Cancer Prediction and Classification Using Machine Learning Algorithms," in 2018 International Symposium on Advanced Electrical and Communication Technologies (ISAECT), 2018, pp. 1–6.

[7] A. S. Assiri, S. Nazir, and S. A. Velastin, "Breast Tumor Classification Using an Ensemble Machine Learning Method," J. Imaging, vol. 6, no. 6, p. 39, May 2020.

[8] M. Abdar and V. Makarenkov, "CWV-BANN-SVM ensemble learning classifier for an accurate diagnosis of breast cancer," Meas. J. Int. Meas. Confed., vol. 146, no. May, pp. 557–570, 2019.

[9] A. O. Chougrad H, Zouaki H, "Deep convolutional neural networks for breast cancer screening," Comput Methods Prog Biomed, vol. 157, pp. 19–30, 2018.

[10] and P. V. S. S. R. C. M. S. Karthik, R. Srinivasa Perumal, "Breast cancer classification using deep neural networks," Knowl Comput Its Appl Knowl Manip Process Tech, vol. 1, pp. 227–241, 2018.

[11] H. Cai, Q. Huang, W. Rong, Y. Song, J. Li, J. Wang, J. Chen, and L. Li, "Breast Microcalcification Diagnosis Using Deep Convolutional Neural Network from Digital Mammograms," Comput. Math. Methods Med., vol. 2019, no. January 2020, 2019.

[12] N. W. et Al, "Deep Neural Networks Improve Radiologists' Performance in Breast Cancer Screening," IEEE Trans. Med. Imaging, vol. 39, no. 4, pp. 1184–1194, 2020.

[13] E. F. Conant, A. Y. Toledano, S. Periaswamy, S. V. Fotin, J. Go, J. E. Boatsman, and J. W. Hoffmeister, "Improving Accuracy and Efficiency with Concurrent Use of Artificial Intelligence for Digital Breast Tomosynthesis," Radiol. Artif. Intell., vol. 1, no. 4, p. e180096, 2019.

[14] G. V. Ionescu, M. Fergie, M. Berks, E. F. Harkness, J. Hulleman, A. R. Brentnall, J. Cuzick, D. G. Evans, and S. M. Astley, "Prediction of reader estimates of mammographic density using convolutional neural networks," J. Med. Imaging, vol. 6, no. 03, p. 1, 2019.

[15] Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In IJCAI Proceedings-International Joint Conference on Artificial Intelligence (Vol. 22, No. 1, p. 1237).

[16] İnik, Ö., & Ülker, E.,(2017) Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri. Gaziosmanpaşa Bilimsel Araştırma Dergisi, 6(3), 85-104.

[17] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks

with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

[18] İnik, Ö., & Ülker, E.,(2017) Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri. Gaziosmanpaşa Bilimsel Araştırma Dergisi, 6(3), 85-104.

[19] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456).

[20] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[21] LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10), 1995.

[22] Li, H., Lin, Z., Shen, X., Brandt, J., & Hua, G. (2015). A convolutional neural network cascade for face detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5325-5334).

[23] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

[24] Pang, Y., Sun, M., Jiang, X., & Li, X. (2017). Convolution in convolution for network in network. IEEE transactions on neural networks and learning systems.

[25] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Berg, A. C. (2015). Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3), 211-252.

[26] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[28] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. Cvpr.

[29] Tamura, S. I., & Tateishi, M. (1997). Capabilities of a four-layered feedforward neural network: four layers versus three. IEEE Transactions on Neural Networks, 8(2), 251-255.

[30] Tang, Y. (2013). Deep learning using linear support vector machines. arXiv preprint arXiv:1306.0239.

[31] You, Y., Zhang, Z., Hsieh, C. J., Demmel, J., & Keutzer, K. (2017). 100-epoch ImageNet training with alexnet in 24 minutes. ArXiv e-prints.

[32] Ouardirhi, A. (2020). Meme kanseri: bilgi, farkındalık ve önleme, Al Bayane.http://albayane.press.ma/cancerdu-sein-linformation-la-sensibilisation-et-la-preventiondabord.html erişim adresi

[33] https://teknoloji.org/cnn-convolutional-neural-networks-nedir/

RESUME

Canik Temel Lisesi

Manisa Celal Bayar University

Permo Elektrik-Elektronik ve Otomasyon Sistemleri -Summer Intorn(2021)