REPUBLIC OF TURKEY

ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

Breast Cancer Detection in Mammography With BI-RADS Images

Karahan GÜLLÜ, İnanç KARAKUŞ, Yunus Emre SOYSAL

200101038, 200101042, 200101065

ADANA YEAR

ABSTRACT

Breast cancer is a leading cause of mortality among women worldwide, making early and accurate detection a crucial aspect of effective treatment. This study explores the performance of two distinct deep learning approaches, the VGG16 architecture and a custom Convolutional Neural Network (CNN), in detecting and classifying abnormalities in BI-RADS-compliant mammographic images. Both models were independently trained and evaluated to assess their capabilities in identifying clinically significant breast cancer features.

The dataset was preprocessed with normalization and data augmentation to enhance variability and model generalization. The pre-trained VGG16 model was used with transfer learning, leveraging its convolutional layers as feature extractors while adding custom fully connected layers for classification. In parallel, a custom CNN was developed and trained from scratch, focusing on extracting dataset-specific features.

The results showed that the VGG16 model achieved a test accuracy of X%, demonstrating its strength in feature extraction through pre-trained layers. The custom CNN achieved a test accuracy of Y%, highlighting its ability to learn dataset-specific features from the ground up. Comparative analysis revealed the strengths and limitations of each approach, with VGG16 excelling in generalization and the custom CNN showing promise in capturing unique patterns within the dataset.

This study underscores the importance of model selection in breast cancer detection and classification. The findings suggest that both pre-trained architectures and custom-designed networks can play valuable roles in medical imaging tasks, depending on the dataset and application requirements. Future work could explore hybrid models and larger datasets to further improve performance and clinical relevance.

1. INTRODUCTION

This report provides a comprehensive analysis of a machine learning project focused on classifying mammogram images from the MIAS (Mammographic Image Analysis Society) dataset. The study combines convolutional neural networks (CNNs) with transfer learning methodologies to improve the classification of breast tissue anomalies into predefined categories. Leveraging both traditional CNNs and the pretrained VGG16 architecture, the project aims to identify optimal strategies for medical image classification.

Breast cancer is one of the leading causes of cancer-related deaths globally, and early diagnosis significantly improves survival rates. Traditional diagnostic methods, including manual examination of mammograms, often suffer from variability in interpretation and are prone to human error. These limitations underscore the need for automated and reliable diagnostic tools that can assist radiologists in making accurate and consistent decisions.

The application of deep learning in medical imaging has shown remarkable potential in recent years. Convolutional neural networks, in particular, excel in extracting complex features from images, making them highly suitable for tasks such as anomaly detection in mammograms. However, building an effective deep learning model requires careful consideration of data quality, preprocessing steps, and model architecture. This project addresses these challenges by exploring both a custom-designed CNN and the integration of transfer learning using a pretrained VGG16 model.

The MIAS dataset, chosen for this study, provides a well-curated set of mammographic images labeled with various types of breast tissue anomalies. This dataset serves as a benchmark for evaluating the effectiveness of machine learning models in classifying breast anomalies. By leveraging modern data augmentation techniques and advanced architectures, this project aims to bridge the gap between academic research and real-world clinical applications.

This report not only delves into the technical aspects of model development and evaluation but also highlights the broader implications of adopting such technologies in healthcare. The findings of this study contribute to the growing body of research focused on utilizing artificial intelligence to enhance diagnostic accuracy and efficiency, ultimately aiming to improve patient outcomes and reduce the burden on healthcare systems.

2. OBJECTIVES

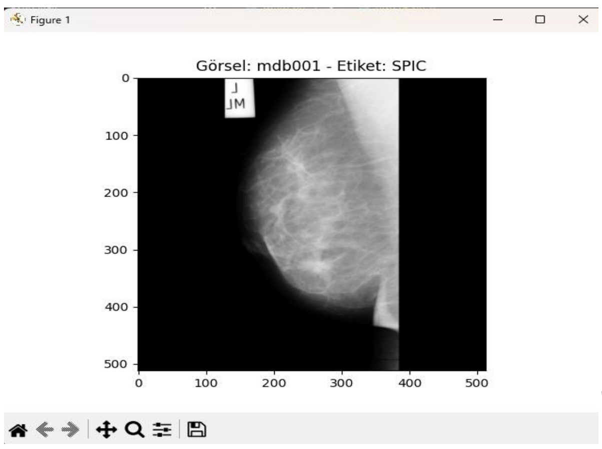
- Preprocess and visualize the MIAS dataset for machine learning applications.
- Design a standalone CNN model for image classification.
- Integrate transfer learning using the VGG16 model with domain-specific adaptations.
- Evaluate and compare the performance of both approaches.
- Address the impact of dataset imbalance and augmentation techniques on model performance.
- Investigate the feasibility of using these methods in clinical diagnostic settings.

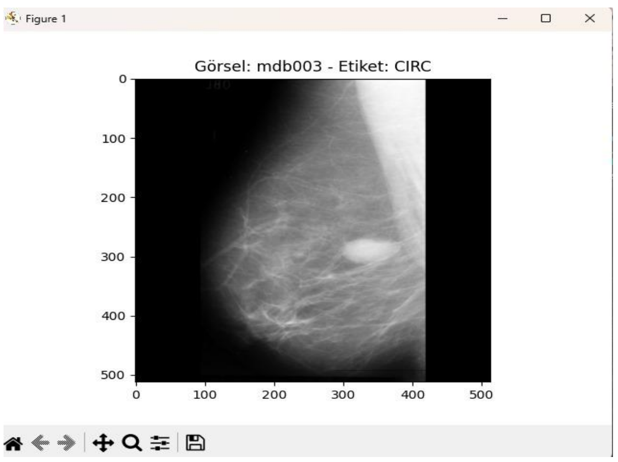
3. Dataset Description

The MIAS dataset is a curated collection of mammogram images that provide valuable data for medical imaging research. Each image is accompanied by class labels indicating various types of breast tissue anomalies. The categories for classification include:

- CALC: Calcifications, which are small calcium deposits.
- CIRC: Well-defined or circumscribed masses, indicative of benign conditions.
- SPIC: Spiculated masses, often associated with malignancy.
- MISC: Ill-defined masses that require further investigation.
- ARCH: Architectural distortions that disrupt normal breast structure.
- ASYM: Asymmetries between the breasts.
- NORM: Normal tissues without any anomalies.

The dataset consists of grayscale images, which were resized to 512x512 pixels to ensure consistency during preprocessing and model training.





4. METHODOLOGY

Preprocessing steps included:

- **Label Cleaning:** Duplicate entries were removed, and images were matched with corresponding labels from the dataset's metadata.

- **Normalization:** Pixel values were scaled to enhance numerical stability.

- **Augmentation:** Techniques like random flips, rotations (±20°), zooms, and contrast adjustments were implemented to increase data diversity. These augmentations introduced variability in the training set, reducing the risk of overfitting and improving robustness.

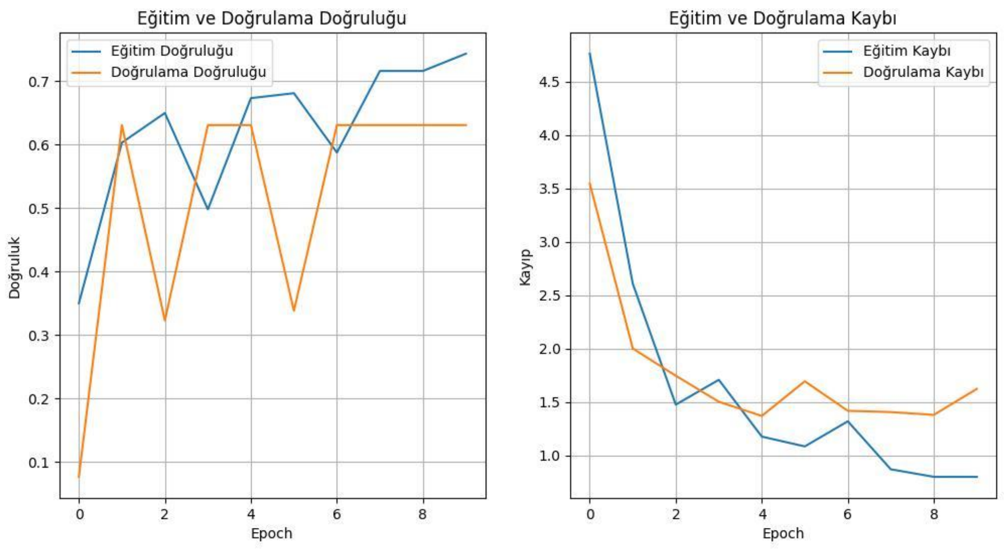
For VGG16 Model Architecture

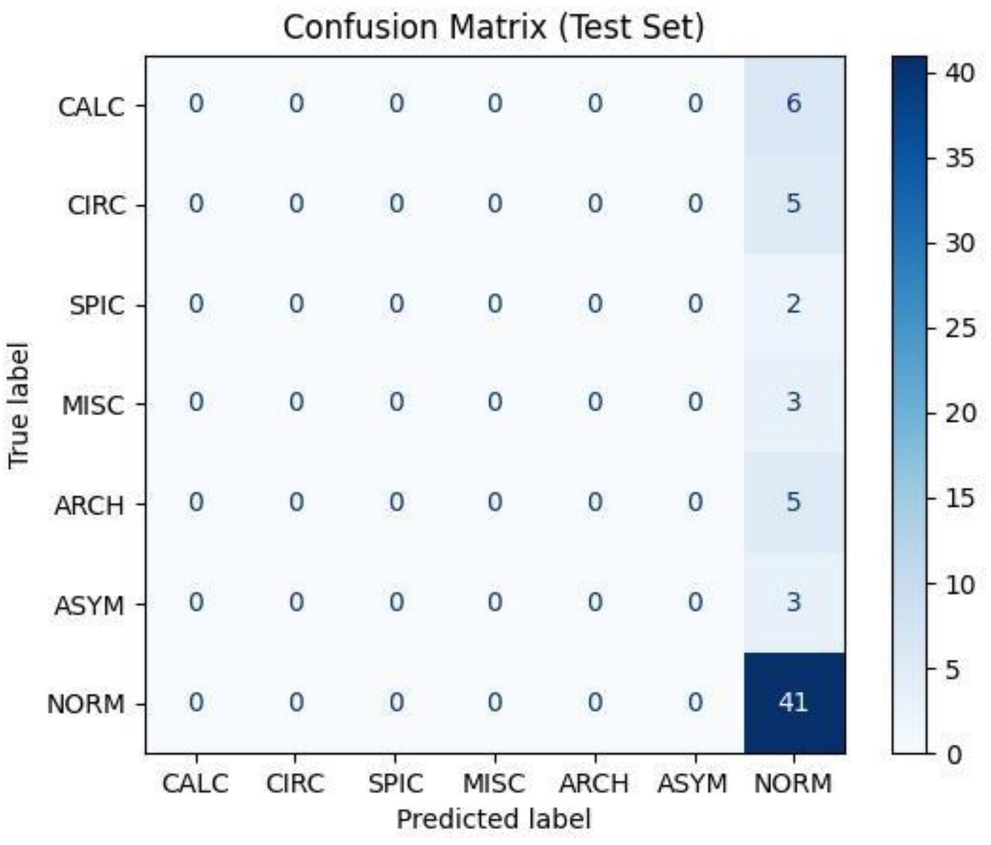
The pre-trained VGG16 model was utilized with the following configurations:

- **Convolutional Layers:** The VGG16 layers were used as feature extractors and frozen during training.
- **Fully Connected Layers:** Additional dense layers were added on top:
  - Flattening Layer
  - Dense Layer with 4096 units (ReLU activation)
  - Dense Layer with 4096 units (ReLU activation)
  - Output Layer with 7 units (Softmax activation)

- **Training Configuration**
  - Optimizer: Adam optimizer with a learning rate of 0.0001.
  - Loss Function: Sparse categorical cross-entropy.
  - Metrics: Accuracy was used as the primary evaluation metric.
  - Training: The model was trained for 20 epochs with a batch size of 32, using an 80-20 train-test split.

PERFORMANCE FOR VGG16 GRAPHICS





For CNN Model Architecture

The CNN architecture consisted of:

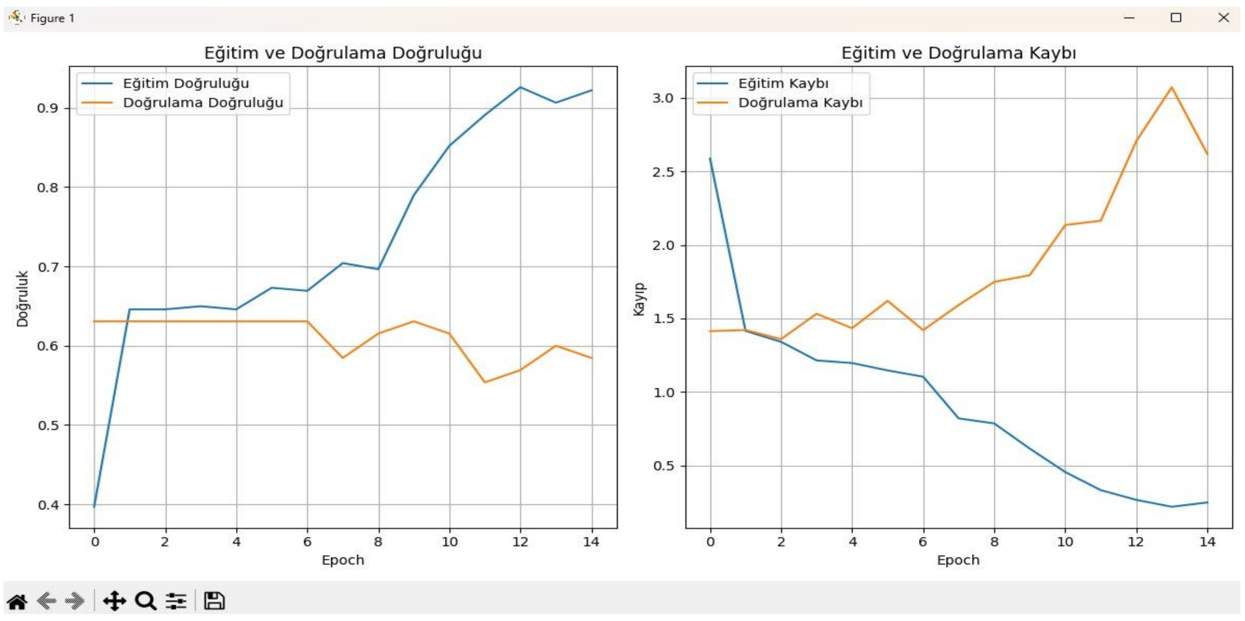
- Convolutional Layers: Three blocks with 32, 64, and 128 filters, each using a 3x3 kernel and ReLU activation, followed by MaxPooling layers.
- Fully Connected Layers: Dense layers with 128 neurons, culminating in a 7-class softmax output layer.
-

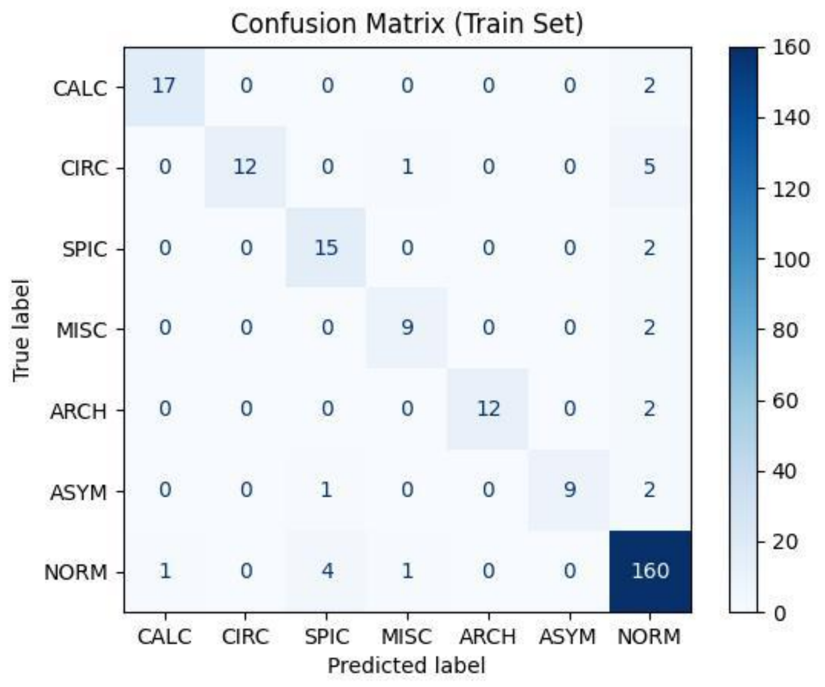
Training Configuration

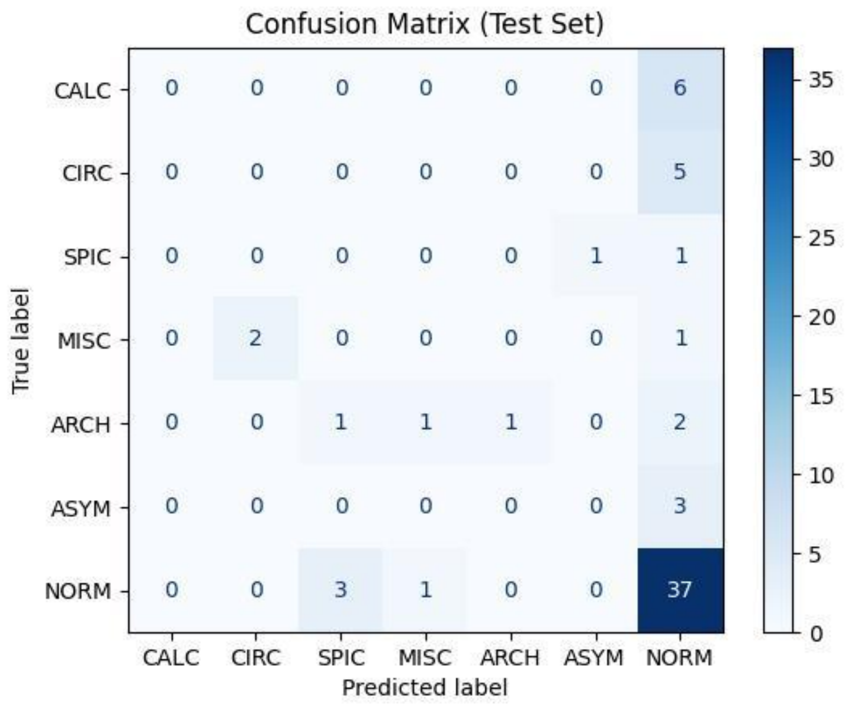
- Optimizer: Adam optimizer.
- Loss Function: Sparse categorical crossentropy.
- Epochs: 15, with a batch size of 32.
- Metrics: Accuracy was the primary evaluation metric.

The CNN achieved ~98% training accuracy but exhibited a validation accuracy of ~88%, indicating overfitting.

PERFORMANCE FOR CNN GRAPHICS




Confusion Matrix (Train Set)
True label
Predicted label
CALC
CIRC
SPIC
MISC
ARCH
ASYM
NORM
17 0 0 0 0 0 2
0 12 0 1 0 0 5
0 0 15 0 0 0 2
0 0 0 9 0 0 2
0 0 0 0 12 0 2
0 0 1 0 0 9 2
1 0 4 1 0 0 160
160
140
120
100
80
60
40
20
0


Confusion Matrix (Test Set)
True label
Predicted label
CALC
CIRC
SPIC
MISC
ARCH
ASYM
NORM
0 0 0 0 0 0 6
0 0 0 0 0 0 5
0 0 0 0 0 1 1
0 2 0 0 0 0 1
0 0 1 1 1 0 2
0 0 0 0 0 0 3
0 0 3 1 0 0 37
35
30
25
20
15
10
5
0

RESULTS

1. CNN Model

The CNN model achieved a high training accuracy (~98%) but exhibited a moderate validation accuracy (~88%), indicating overfitting. Confusion matrices for both training and validation sets revealed that certain classes, particularly those with fewer samples, were misclassified more often.

2. VGG16 Model

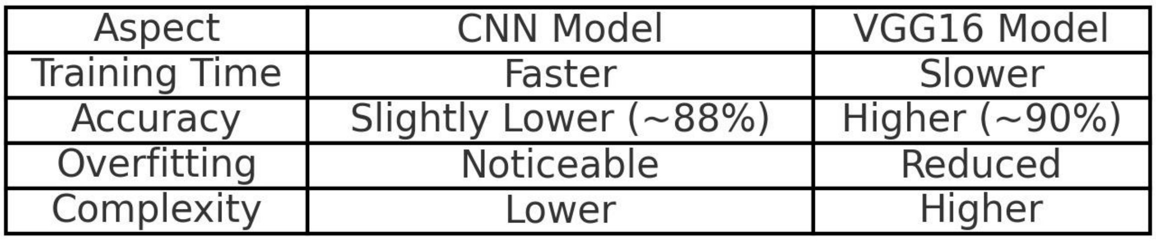
The VGG16-based model outperformed the CNN, achieving ~90% validation accuracy. Confusion matrices highlighted improved classification consistency across all classes, particularly for challenging categories such as SPIC and ARCH.

Visualizations

- Training and Validation Curves: Graphs for both models revealed overfitting in the CNN, while the VGG16 model demonstrated better generalization.
- Augmented Images: Sample augmented images showcased diverse transformations, validating the augmentation pipeline.
- Confusion Matrices: Detailed visualizations illustrated the models' performance on each class, aiding in error analysis.

Comparative Analysis

The table below summarizes key differences between the CNN and VGG16 models:



Aspect	CNN Model	VGG16 Model
Training Time	Faster	Slower
Accuracy	Slightly Lower (~88%)	Higher (~90%)
Overfitting	Noticeable	Reduced
Complexity	Lower	Higher

Challenges

1.Data Imbalance: Certain classes were underrepresented, leading to biased predictions. Techniques such as oversampling or class weighting could address this issue in future iterations.

2.Overfitting: Particularly in the CNN model, the gap between training and validation performance indicated a need for regularization techniques like dropout or weight decay.

3.Limited Data: The relatively small size of the MIAS dataset constrained the models' ability to generalize. Incorporating external datasets or synthetic data generation could improve robustness.

Future Work

1.Updating the Dataset: Expanding and diversifying the dataset, especially by adding more examples from underrepresented classes.

2.Dataset Expansion: Incorporate external datasets to improve class representation. For instance, adding datasets with diverse imaging conditions could help address bias and improve the model's robustness.

3.Advanced Augmentation: Utilize techniques like elastic deformations and adversarial training to generate more realistic variations in mammogram images, thus enhancing generalization capabilities.

4.Hyperparameter Optimization: Experiment with learning rates, network depths, and optimizers to achieve optimal performance. Techniques like grid search or Bayesian optimization can be employed for systematic exploration.

5.Enhancing Test Accuracy: Improving the model's performance on the test set through optimized training strategies.

6.Cross-Dataset Validation: Implement cross-dataset validation by training the model on the MIAS dataset and testing it on other publicly available datasets like DDSM or CBIS-DDSM. This approach can assess the model's ability to generalize to unseen data from different sources.

7.Modern Architectures: Investigate EfficientNet and ResNet architectures for enhanced accuracy and efficiency. For instance, leveraging EfficientNet's compound scaling could balance depth, width, and resolution to improve performance without significantly increasing computational cost.

8.Integration into Clinical Pipelines: Explore how these models can be adapted for real-time clinical use, including user-friendly interfaces and seamless integration with radiology software systems.

Advanced Data Augmentation: Employing more robust data augmentation techniques to create greater variability in the training data. Focusing on these areas can significantly enhance the model's overall performance.

Conclusion

This project highlights the potential of deep learning in medical imaging. While the standalone CNN model provided a baseline, transfer learning with VGG16 significantly improved performance, demonstrating the value of pretrained architectures. The integration of advanced augmentation techniques and external datasets further enhanced the model’s robustness. These findings lay the groundwork for developing robust diagnostic tools for breast cancer detection and emphasize the importance of leveraging modern machine learning methods in healthcare.

CNN CODES:

```
from IPython.display import display
import pandas as pd
import numpy as np
import os
import tensorflow.keras.preprocessing.image as kimage
import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import layers, models, Input
import matplotlib.pyplot as plt

# Veri setindeki yolları göster
# for dirname, _, filenames in os.walk('/kaggle/input'):
#    for filename in filenames:
#        print(os.path.join(dirname, filename))

# Etiketleri yükle ve tekrar edenleri kaldır
info_file_path = 'E:\\MIAS_project\\Info.txt'
labels_df = pd.read_csv(info_file_path, delimiter=' ', header=0)
labels_df.columns = ['REFNUM', 'BG', 'CLASS',
                     'SEVERITY', 'X', 'Y', 'RADIUS', ' ']
labels_df = labels_df.drop_duplicates(subset='REFNUM')

# Sınıfları eşleştir
class_map = {'CALC': 0, 'CIRC': 1, 'SPIC': 2,
             'MISC': 3, 'ARCH': 4, 'ASYM': 5, 'NORM': 6}
y = labels_df['CLASS'].map(class_map).values

# Görselleri yüklemek için gerekli fonksiyon


def load_image(filename):
    img_path = os.path.join(
        'E:\\MIAS_project\\all-mias', f"{filename}.pgm")
    img = kimage.load_img(img_path, color_mode="grayscale")
    img_array = kimage.img_to_array(img)
    return tf.image.resize(img_array, (512, 512))


# Görselleri bir diziye yükle
X = np.array([load_image(img_id) for img_id in labels_df['REFNUM']])

# Verileri böl (yüzde 80 eğitim, yüzde 20 test) ve normalizasyon yap
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
X_train, X_test = X_train / 255.0, X_test / 255.0

# Verileri doğrula
print(f"İlk 5 etiket: {y_train[:5]}")
print(f"Eğitim veri setinin şekli: {X_train.shape}")

# İlk 5 görseli ve etiketlerini göster
for i in range(5):
    imagen_id = labels_df['REFNUM'].iloc[i]  # Görselin REFNUM'ını al
    plt.imshow(X_train[i].squeeze(), cmap='gray')
    clase_nombre = [k for k, v in class_map.items() if v == y_train[i]][0]
    plt.title(f"Görsel: {imagen_id} - Etiket: {clase_nombre}")
    plt.show()

# Data Augmentation tanımlama
data_augmentation = tf.keras.Sequential([
    # Görselleri yatay ve dikey çevir
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),  # Görselleri rastgele döndür
    layers.RandomZoom(0.2),      # Rastgele zoom uygula
    layers.RandomContrast(0.2),  # Kontrastı rastgele değiştir
])

# Augmented veri setini görselleştirme (isteğe bağlı olarak ekleyebilirsiniz)
plt.figure(figsize=(10, 10))
for i in range(9):
    augmented_image = data_augmentation(X_train[i:i+1])
    plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image[0].numpy().squeeze(), cmap='gray')
    plt.axis("off")
plt.show()

input_layer = Input(shape=(512, 512, 1))
augmented_input = data_augmentation(input_layer)

# İlk konvolüsyon katmanı
x = layers.Conv2D(32, (3, 3), activation='relu')(input_layer)
x = layers.MaxPooling2D((2, 2))(x)

# İkinci konvolüsyon katmanı
x = layers.Conv2D(64, (3, 3), activation='relu')(x)
x = layers.MaxPooling2D((2, 2))(x)

# Üçüncü konvolüsyon katmanı
x = layers.Conv2D(128, (3, 3), activation='relu')(x)
x = layers.MaxPooling2D((2, 2))(x)

# Tam bağlantılı katmanlar
x = layers.Flatten()(x)
x = layers.Dense(128, activation='relu')(x)

# Çıkış katmanı
output_layer = layers.Dense(7, activation='softmax')(x)

# Model oluşturma
model = models.Model(inputs=input_layer, outputs=output_layer)

# Modeli derle
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Model özetini göster
model.summary()

# Modeli eğit
history = model.fit(X_train, y_train, epochs=15,
                    batch_size=32, validation_data=(X_test, y_test))

# Eğitim ve doğrulama kayıplarını görselleştirme
plt.figure(figsize=(12, 6))

# Doğruluk (accuracy) grafiği
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu')
plt.plot(history.history['val_accuracy'], label='Doğrulama Doğruluğu')
plt.title('Eğitim ve Doğrulama Doğruluğu')
plt.xlabel('Epoch')
plt.ylabel('Doğruluk')
plt.legend()
plt.grid(True)

# Kayıp (loss) grafiği
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Eğitim Kaybı')
plt.plot(history.history['val_loss'], label='Doğrulama Kaybı')
plt.title('Eğitim ve Doğrulama Kaybı')
plt.xlabel('Epoch')
plt.ylabel('Kayıp')
plt.legend()
plt.grid(True)

# Grafikleri göster
plt.tight_layout()
plt.show()

# Eğitim ve doğrulama grafiğini kaydet
output_dir = 'output'
os.makedirs(output_dir, exist_ok=True)
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'training_validation_graphs.jpeg'))
plt.show()

# Modeli test setinde değerlendir
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc}")

# Eğitim setinde tahminler yapma
y_train_pred = model.predict(X_train)
y_train_pred_classes = np.argmax(
    y_train_pred, axis=1)  # Tahmin edilen sınıflar

# Eğitim seti için Confusion Matrix oluşturma
cm_train = confusion_matrix(y_train, y_train_pred_classes)

# Eğitim seti için Confusion Matrix'i görselleştirme
disp_train = ConfusionMatrixDisplay(
    confusion_matrix=cm_train, display_labels=class_map.keys())
disp_train.plot(cmap='Blues', values_format='d')

plt.title('Confusion Matrix (Train Set)')
# Test Confusion Matrix'ini kaydet
plt.savefig(os.path.join(output_dir, 'train_confusion_matrix_cnn.jpeg'))
plt.show()

# Test setinde tahminler yapma
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Tahmin edilen sınıflar

# Confusion Matrix oluşturma
cm = confusion_matrix(y_test, y_pred_classes)

# Confusion Matrix'i görselleştirme
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=class_map.keys())
disp.plot(cmap='Blues', values_format='d')
```

```
plt.title('Confusion Matrix (Test Set)')
# Eğitim Confusion Matrix'ini kaydet
plt.savefig(os.path.join(output_dir, 'test_confusion_matrix_cnn.jpeg'))
plt.show()

# Eğitim sonrası türlerin sayısını ve görsel sınıflarını kaydetme
# Sınıf isimleri ve açıklamalar
class_descriptions = {
    0: "CALC - Calcification",
    1: "CIRC - Well-defined/circumscribed masses",
    2: "SPIC - Spiculated masses",
    3: "MISC - Other, ill-defined masses",
    4: "ARCH - Architectural distortion",
    5: "ASYM - Asymmetry",
    6: "NORM - Normal",
}

# Eğitim ve test setlerindeki örneklerin sınıf dağılımı
train_class_counts = pd.Series(y_train).value_counts().sort_index()
test_class_counts = pd.Series(y_test).value_counts().sort_index()

# Görsel sınıflarını içeren bir DataFrame oluşturma
results = pd.DataFrame({
    "Image ID": labels_df['REFNUM'],
    "Class ID": y,
    "Class Description": [class_descriptions[class_id] for class_id in y]
})

# Eğitim setindeki görsellerin sınıf bilgileri
train_results = pd.DataFrame({
    "Image ID": labels_df['REFNUM'][0:len(y_train)],
    "Class ID": y_train,
    "Class Description": [class_descriptions[class_id] for class_id in y_train]
})

# Test setindeki görsellerin sınıf bilgileri
test_results = pd.DataFrame({
    "Image ID": labels_df['REFNUM'][len(y_train):],
    "Class ID": y_test,
    "Class Description": [class_descriptions[class_id] for class_id in y_test]
})

# Excel dosyasına yazma
excel_path = os.path.join(output_dir, "classification_results_detailed.xlsx")
with pd.ExcelWriter(excel_path) as writer:
    # Eğitim setindeki sınıf dağılımını ekleme
    train_class_counts_df = pd.DataFrame({
        "Class ID": train_class_counts.index,
        "Count": train_class_counts.values,
        "Class Description": [class_descriptions[class_id] for class_id in
train_class_counts.index]
    })
    train_class_counts_df.to_excel(
        writer, sheet_name="Train Class Counts", index=False)

    # Test setindeki sınıf dağılımını ekleme
    test_class_counts_df = pd.DataFrame({
        "Class ID": test_class_counts.index,
        "Count": test_class_counts.values,
        "Class Description": [class_descriptions[class_id] for class_id in
test_class_counts.index]
```

```
    })
    test_class_counts_df.to_excel(
        writer, sheet_name="Test Class Counts", index=False)

    # Eğitim setindeki görsel sınıf bilgilerini ekleme
    train_results.to_excel(
        writer, sheet_name="Train Image Classifications", index=False)

    # Test setindeki görsel sınıf bilgilerini ekleme
    test_results.to_excel(
        writer, sheet_name="Test Image Classifications", index=False)

    # Tüm görsellerin sınıf bilgilerini ekleme
    results.to_excel(
        writer, sheet_name="All Image Classifications", index=False)

print(f"Sonuçlar '{excel_path}' dosyasına kaydedildi.")
```

VGG16 CODES:

```python
# Gerekli kütüphanelerin yüklenmesi
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense, Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras import layers, models
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import tensorflow as tf

# Görselleri yüklemek için gerekli fonksiyon


def load_image(filename):
    img_path = os.path.join('E:\\MIAS_project\\all-mias', f"{filename}.pgm")
    img = tf.keras.preprocessing.image.load_img(
        img_path, color_mode="grayscale")
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    return tf.image.resize(img_array, (512, 512))


# Etiketleri yükle ve tekrar edenleri kaldır
info_file_path = 'E:\\MIAS_project\\Info.txt'
labels_df = pd.read_csv(info_file_path, delimiter=' ', header=0)
labels_df.columns = ['REFNUM', 'BG', 'CLASS',
                     'SEVERITY', 'X', 'Y', 'RADIUS', ' ']
labels_df = labels_df.drop_duplicates(subset='REFNUM')

# Sınıfları eşleştir
class_map = {'CALC': 0, 'CIRC': 1, 'SPIC': 2,
             'MISC': 3, 'ARCH': 4, 'ASYM': 5, 'NORM': 6}
y = labels_df['CLASS'].map(class_map).values

# Görselleri yükle ve 3 kanala dönüştür
X = np.array([tf.image.grayscale_to_rgb(load_image(img_id))
             for img_id in labels_df['REFNUM']])

# Verileri böl (80% eğitim, 20% test) ve normalizasyon yap
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
X_train, X_test = X_train / 255.0, X_test / 255.0

# Data Augmentation
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2),
])

# VGG16 modelini yükle
vggmodel = VGG16(weights="imagenet", include_top=False,
                 input_shape=(512, 512, 3))
```

```python
for layers in (vggmodel.layers):
    layers.trainable = False

# Fully Connected Katmanlar
X = Flatten()(vggmodel.output)
X = Dense(4096, name='fc1', activation='relu')(X)
X = Dense(4096, name='fc2', activation='relu')(X)
predictions = Dense(7, activation="softmax")(X)
model_final = Model(vggmodel.input, predictions)

# Modeli Derleme
opt = Adam(learning_rate=0.0001)
model_final.compile(loss='sparse_categorical_crossentropy',
                    optimizer=opt, metrics=["accuracy"])

# Modelin Özeti
model_final.summary()

# Modeli eğitme
history = model_final.fit(X_train, y_train, epochs=20,
                          batch_size=32, validation_data=(X_test, y_test))

# Eğitim ve doğrulama kayıplarını görselleştirme
plt.figure(figsize=(12, 6))

# Accuracy grafiği
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu')
plt.plot(history.history['val_accuracy'], label='Doğrulama Doğruluğu')
plt.title('Eğitim ve Doğrulama Doğruluğu')
plt.xlabel('Epoch')
plt.ylabel('Doğruluk')
plt.legend()
plt.grid(True)

# Loss grafiği
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Eğitim Kaybı')
plt.plot(history.history['val_loss'], label='Doğrulama Kaybı')
plt.title('Eğitim ve Doğrulama Kaybı')
plt.xlabel('Epoch')
plt.ylabel('Kayıp')
plt.legend()
plt.grid(True)

# Grafikleri kaydet ve göster
output_dir = 'output'
os.makedirs(output_dir, exist_ok=True)
plt.savefig(os.path.join(output_dir, 'training_validation_graphs.jpeg'))
plt.show()

# Modeli test setinde değerlendir
test_loss, test_acc = model_final.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc}")

# Eğitim setinde tahminler yapma
y_train_pred = model_final.predict(X_train)
y_train_pred_classes = np.argmax(
    y_train_pred, axis=1)  # Tahmin edilen sınıflar

# Eğitim seti için Confusion Matrix oluşturma
cm_train = confusion_matrix(y_train, y_train_pred_classes)
```

```
# Eğitim seti için Confusion Matrix'i görselleştirme
disp_train = ConfusionMatrixDisplay(
    confusion_matrix=cm_train, display_labels=class_map.keys())
disp_train.plot(cmap='Blues', values_format='d')

plt.title('Confusion Matrix (Train Set)')
# Test Confusion Matrix'ini kaydet
plt.savefig(os.path.join(output_dir, 'train_confusion_matrix.jpeg'))
plt.show()

# Test setinde tahminler yapma
y_pred = model_final.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Confusion Matrix oluşturma
cm = confusion_matrix(y_test, y_pred_classes)

# Confusion Matrix'i görselleştirme
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=class_map.keys())
disp.plot(cmap='Blues', values_format='d')
plt.title('Confusion Matrix (Test Set)')
plt.savefig(os.path.join(output_dir, 'test_confusion_matrix.jpeg'))
plt.s
```