Video Link : https://youtu.be/vpSNA6YuWRQ?si=NEwNqCoRjo3S09t2

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FINAL YEAR PROJECT 1

FUSION OF NETWORK AND HANDCRAFTED FEATURES IN DEEP LEARNING FOR CLASSIFYING KNEE OSTEOTHRITIS

PREPARED BY : LIM YONG WEI (B22EC0025)

SUPERVISOR: TS.DR.CHAN WENG HOWE

EXAMINER 1 :
DR. SHARIN HAZLIN BINTI HUSPI

EXAMINER 2 :
DR. NIES HUI WEN

PRESENTATION OUTLINE

1 Introduction

- Background of study
- Problem of statement
- Goal & Objective
- Scope of Research
- Importance of Research

2 Literature Review

- Features
- Machine Learning
- Deep Learning

3 Methodology

- Research Framework
- Convolutional Neutral Network
- Feature-level Fusion
- Description of Datasets
- Performance Measurement
- Architecture Diagram

4 Results

- Preliminary Result

5 Conclusion

- Achievement of Research
- Research Constraints
- Suggestion for Improvement and Future Work

Background of Study

What is KOA ?

knee osteoarthritis, is a common **musculoskeletal disease**

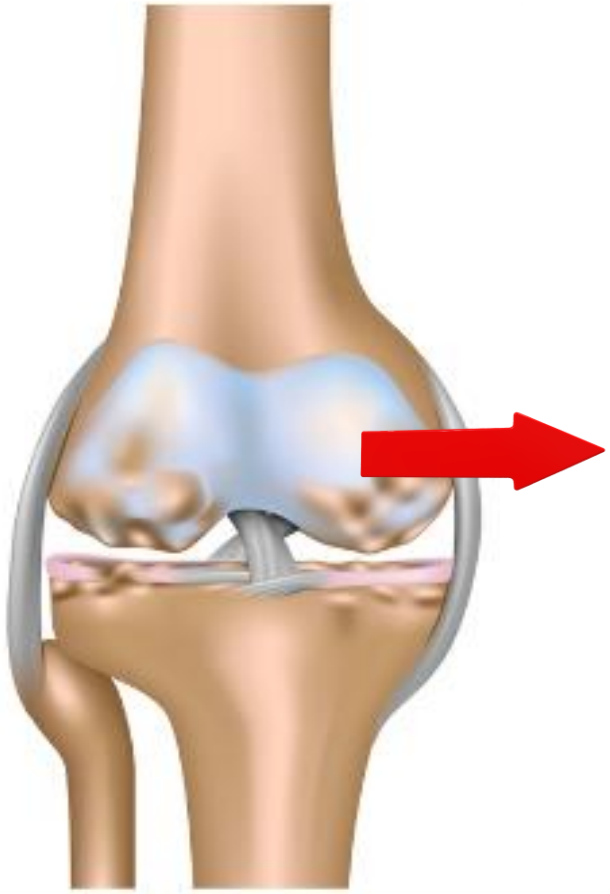
Why does KOA occur ?

due to the **degeneration of joint cartilage** in the knee, which leads to **joint pain** and disability

Who will interact with KOA?

primarily affects **elderly individuals**, but it can also impact those with risk factors such as **obesity, prior joint injuries, or a family history of osteoarthritis**

Problem Statement



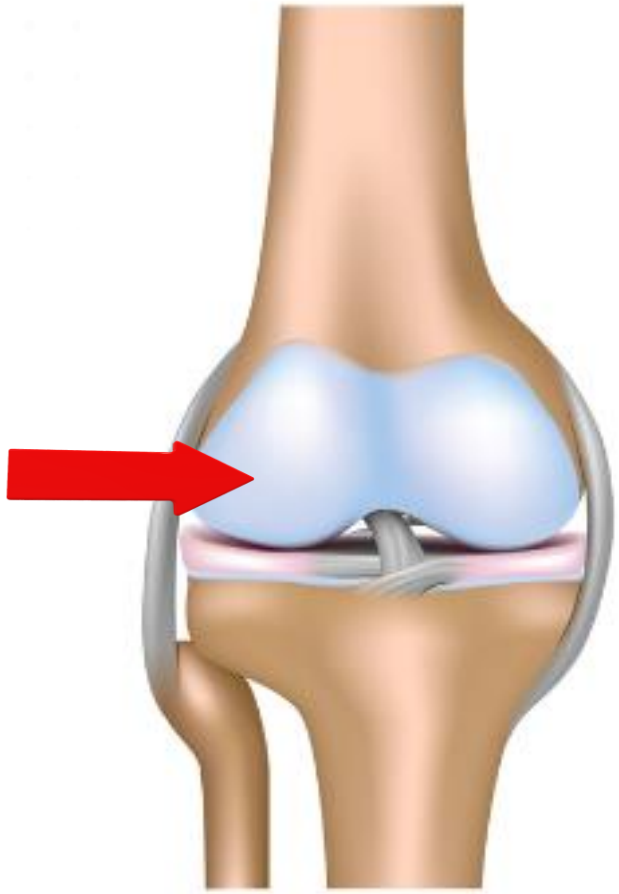
Osteoarthritis

1. Need for *Improved Diagnostic* Techniques

Predicting KOA is crucial because it's still caused by ***many factors, affects quality of life, and varies widely in how it develops***.

Source : An Evolutionary Machine Learning Approach for KOA

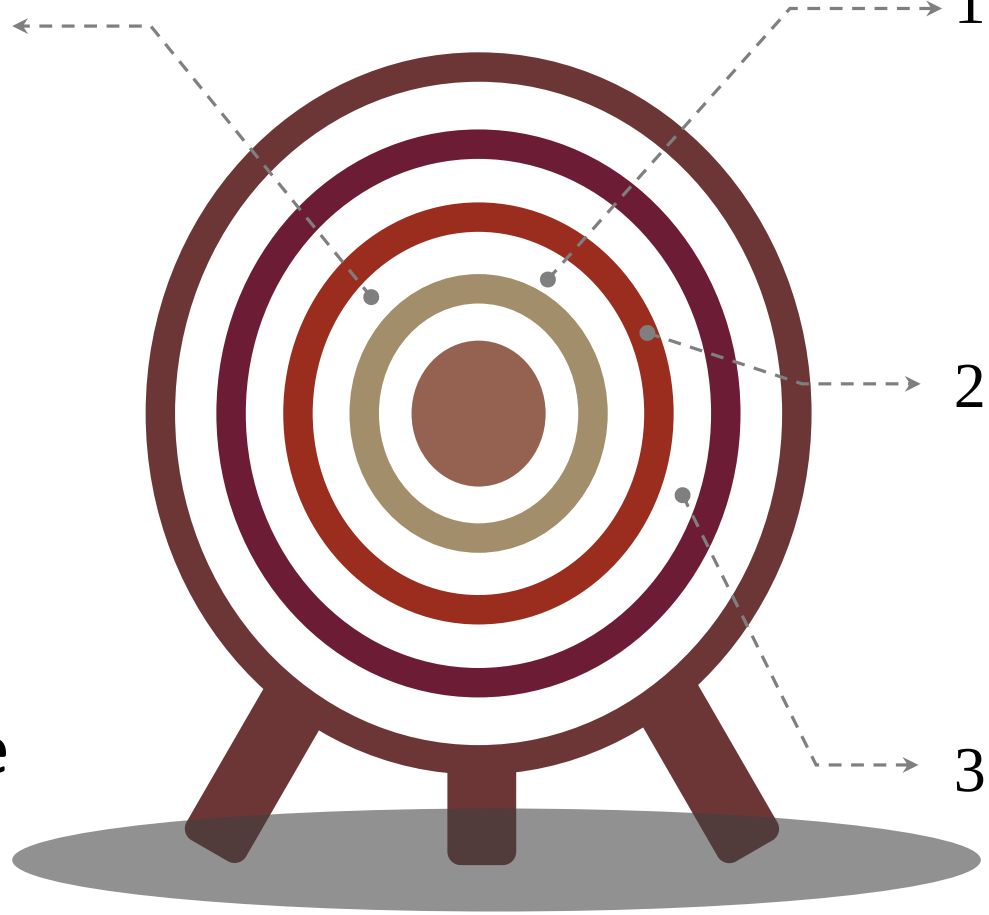
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8000487/



Healthy knee joint

Goal

1. To ***improve the accuracy and reliability of KOA grading***, thereby facilitating ***early detection and appropriate management strategies*** for knee osteoarthritis.



Objective

1. To investigate existing ***deep learning method and features used*** for classification KOA
2. To ***fuse handcrafted features and network features*** using features level fusion.
3. To ***implement Feed-Forward Neutral Network for classification*** of KOA using fused features.

Scope of Research

- Encompasses ***methodologies for KOA analysis and diagnosis***, including feature extraction, classification algorithms, and evaluation metrics.
- To ***compare and evaluate*** different methodologies to determine their effectiveness in accurately diagnosing KOA.
- Focuses on ***classifying the severity of KOA*** to improve diagnostic accuracy.

Importance of Research

- The research contributes to progress in ***medical image analysis and diagnosis***, particularly for knee osteoarthritis (KOA).
- The goal is to achieve better patient outcomes and ***more effective treatment planning*** through improved diagnostic techniques.

Studies for KOA using Features

Types	Reference	Approach	Methodology	Features
Network	Chen et al.,(2020)	Fully automatic knee osteoarthritis severity grading using deep neural networks with a novel ordinal loss.	This study detect knee joints using a customized one-stage YOLOv2 network then fine-tune the most popular CNN models to classify the detected knee joint images with a novel adjustable ordinal loss	DenseNet , InceptionV 3 , Resnet , VGG
Fusion	Prashantha and Prakash (2022)	Feature level fusion framework for brain MR image classification using supervised deep learning and handcrafted features	This study propose an efficient fusion framework for brain magnetic resonance (MR) image classification using deep learning and handcrafted feature extraction method	Feature Level Fusion
Hand-crafted	Khalid et al.,(2023)	Automatic Analysis of MRI Images for Early Prediction of Alzheimer's Disease Stages Based on Hybrid Features of CNN and Handcrafted Features	This study extract MRI image that by deep learning technique and combine with DWT , GLCM and LBP.	DWT , GLCM , LBP , CNN

Table 2.2 Studies KOA in DL

Studies for KOA using Machine Learning

Reference	Approach	Methodology	Accuracy / F1 Score
Heisinger et al.,(2020)	Predicting total knee replacement from symptomology and radiographic structural change using artificial neural networks—data from the osteoarthritis initiative (OAI)	This study uses longitudinal assessments of radiographic changes, knee pain, function, and quality of life, then classifies by artificial neural networks **(ANN)**	81.2%
Kokkotis et al.,(2020)	Identification of risk factors and machine learning-based prediction models for knee osteoarthritis patients.	This study uses a robust feature selection approach combining filter, wrapper, and embedded techniques, followed by classification with **SVM**	74.07%
Su et al.,(2023)	Improved Prediction of Knee Osteoarthritis by the Machine Learning Model XGBoost.	This study uses a dataset combining clinical parameters, imaging data, and patient history, then classifies by a **decision tree** algorithm	**0.553** (F1 Score)

Table 2.1 Studies KOA in ML

Studies for KOA using Deep Learning

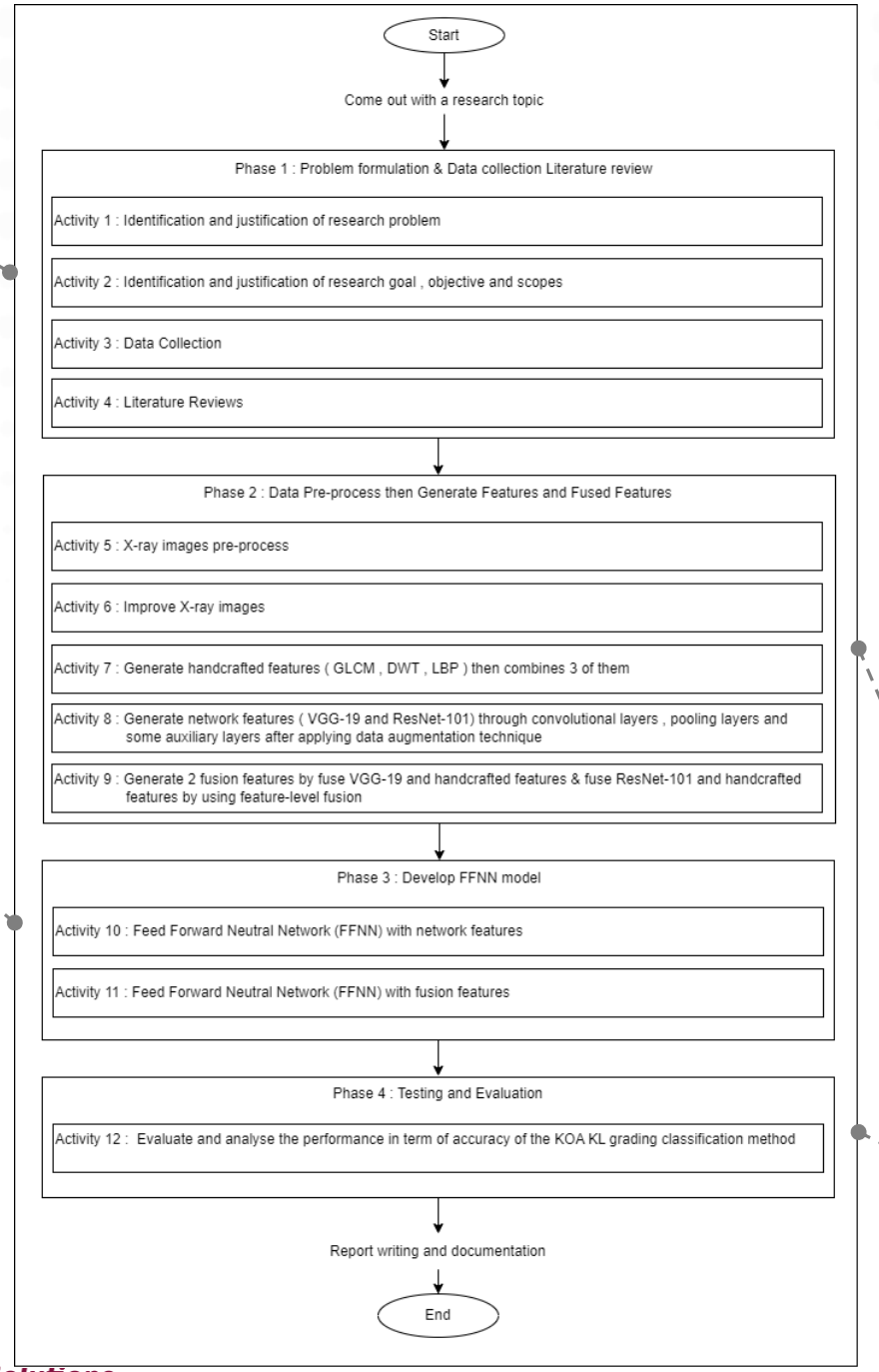
Reference	Approach	Methodology	Accuracy
Tiwari, Poduval and Bagaria (2021)	Using deep learning methods for orthopedic radiography, artificial intelligence models for osteoarthritis of the knee are evaluated.	This study uses **AI and DL** to develop a neural network-based assessment for classifying KOA severity from radiographic images annotated with KL grades	74%
Yunus et al.,(2022)	YOLOv2 is used to identify knee osteoarthritis (KOA), and convolutional neural networks are employed for classification.	This study uses 3D radiographic images with LBP features and deep features from Alex-Net and Dark-net-53, then classifies by **CN**N	90.6%
Moustakidis et al.,(2023)	Dense neural networks in knee osteoarthritis classification: a study on accuracy and fairness	This study uses self-reported clinical data and then classifies by **DNN**	79.6%

Table 2.2 Studies KOA in DL

Research Framework

Phase 1

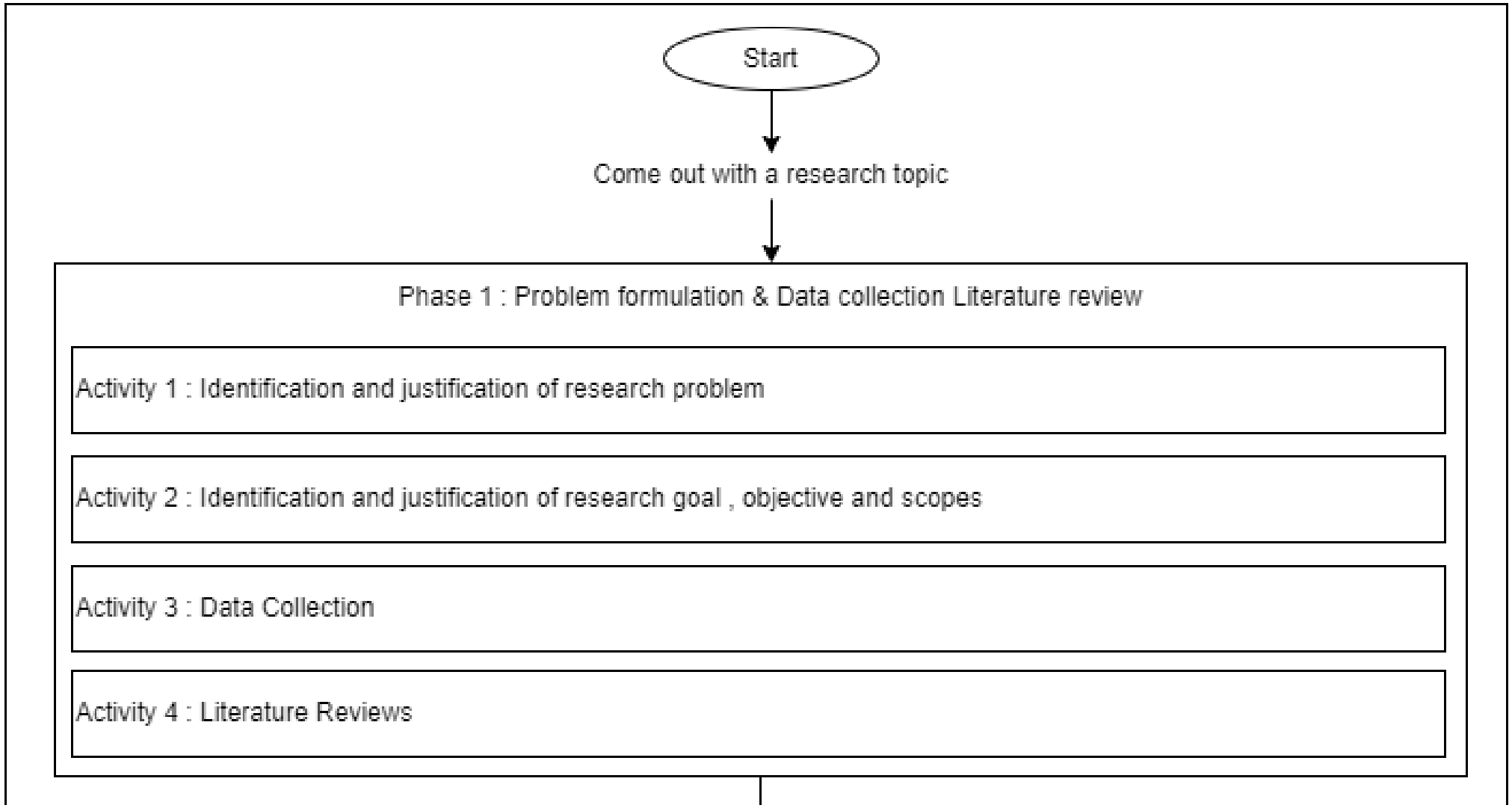
Phase 3



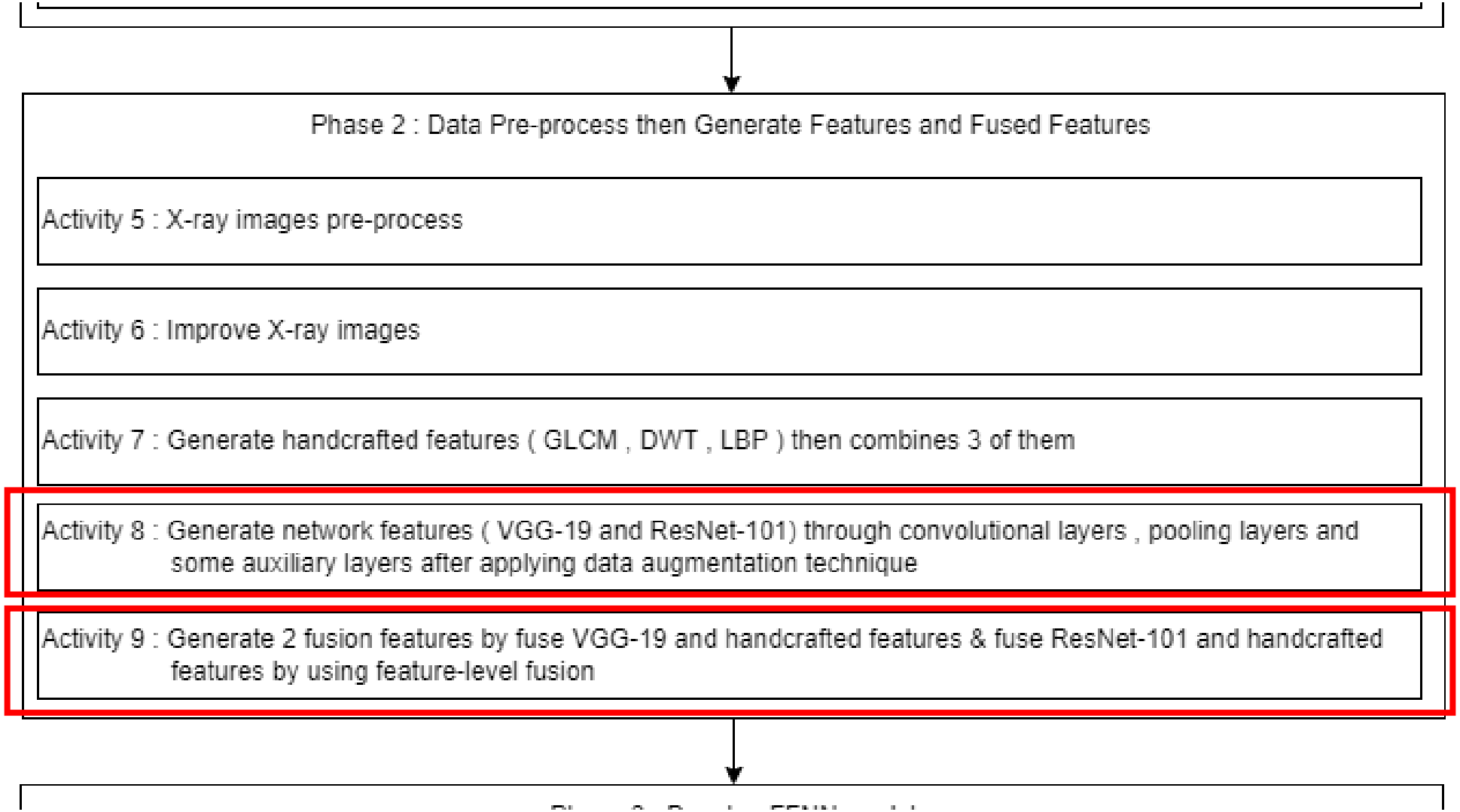
Phase 2

Phase 4

Research Framework – Phase 1



Research Framework – Phase 2

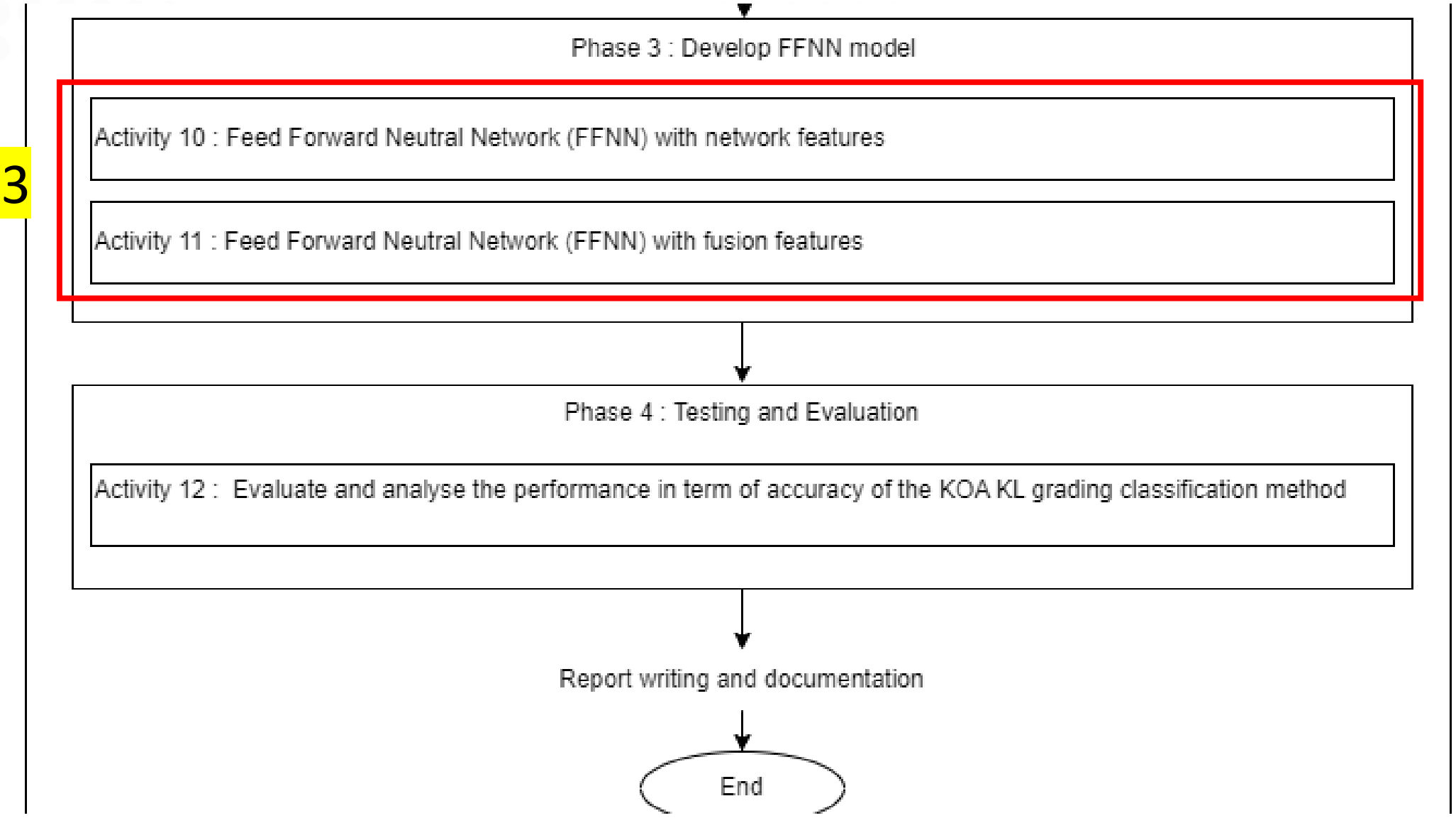


Obj 1

Obj 2

Research Framework – Phase 3&4

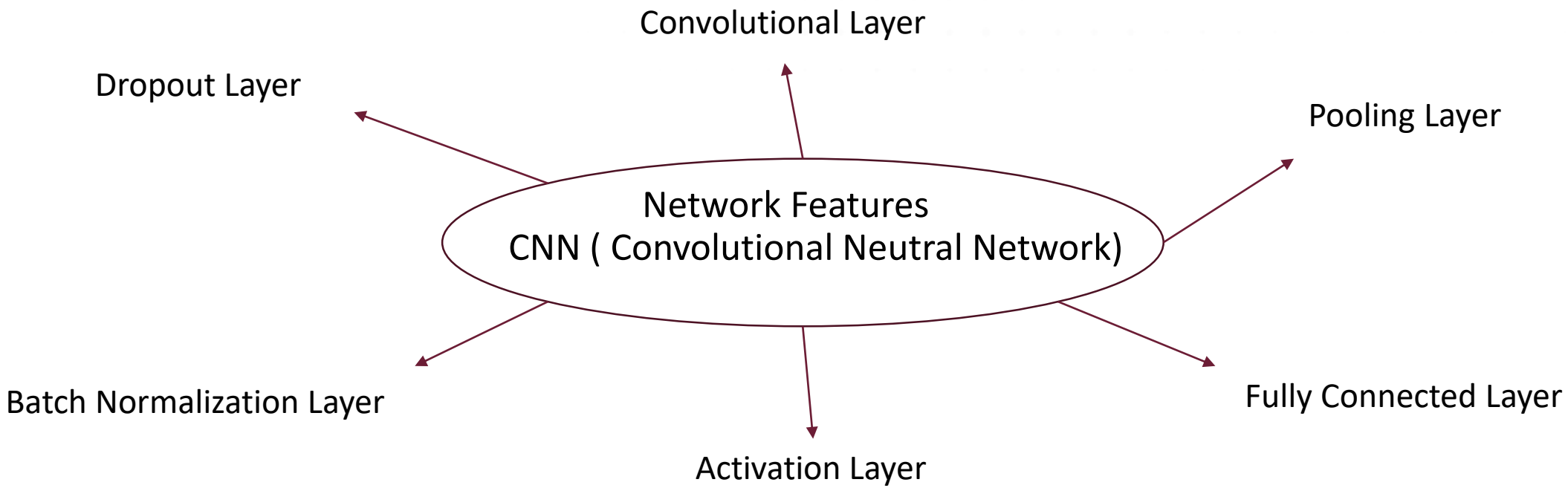
Obj 3



Convolutional Neutral Network

Obj 1

Activity 8 : Generate network features (VGG-19 and ResNet-101) through convolutional layers , pooling layers and some auxiliary layers after applying data augmentation technique

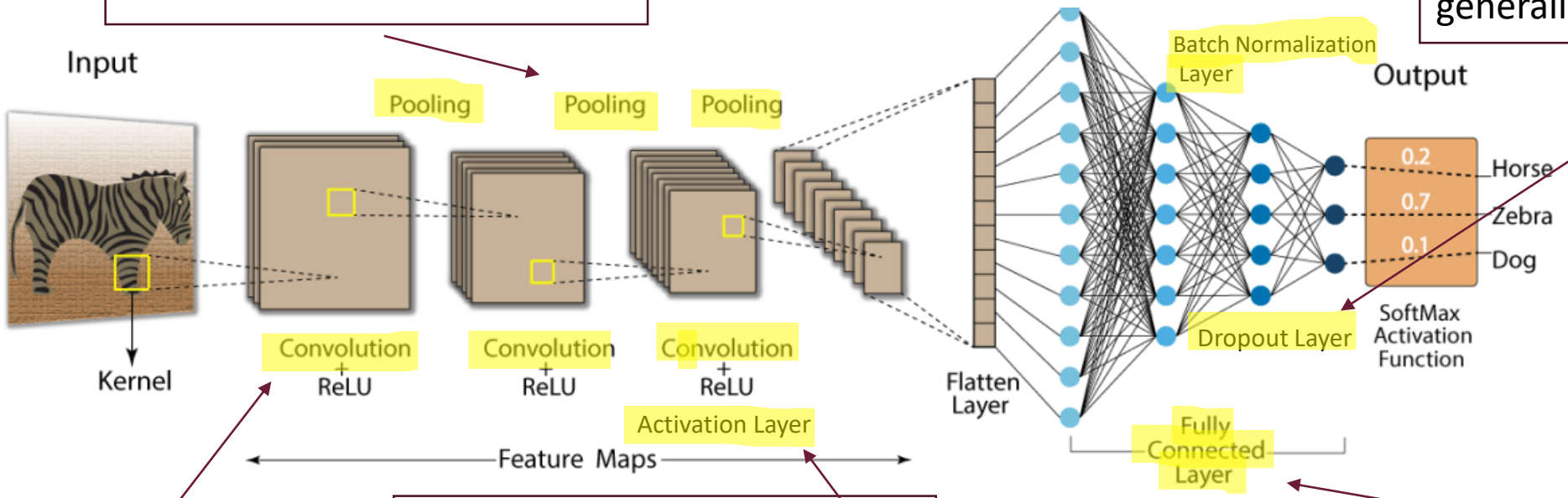


CNN – EXAMPLE

reduce the dimensionality of the feature maps while retaining the most important information

Normalizes the output of previous layers to improve training stability and performance

Randomly drops units during training to prevent overfitting and improve generalization



extracting color features, edges, contrast, and geometric features from X-rays.

Applies a ***non-linear activation function*** (e.g., ReLU) to introduce non-linearity into the model

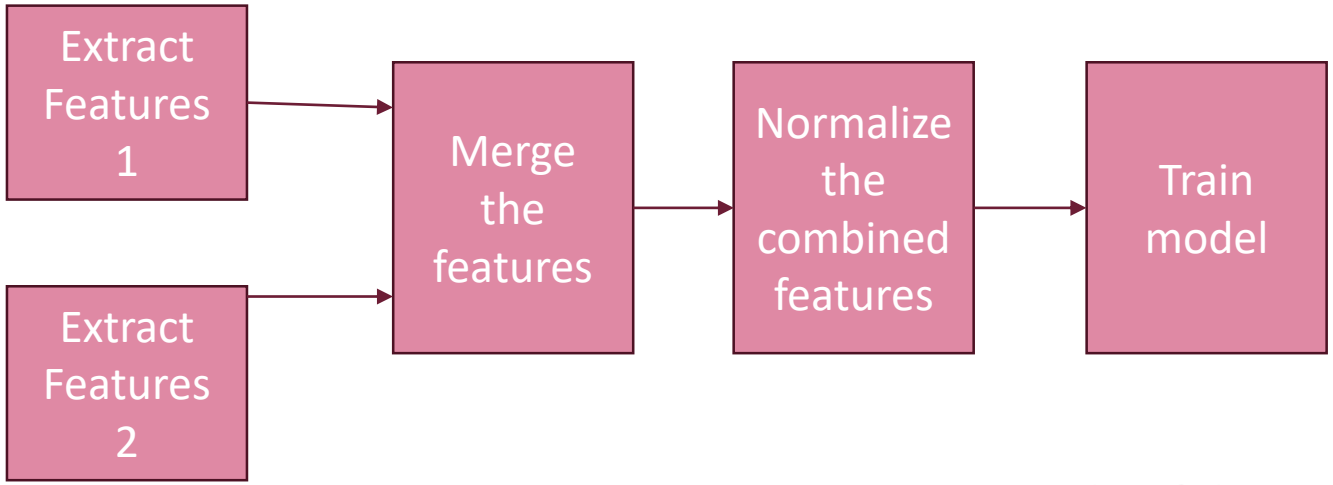
Integrates the extracted features to produce a final representation for classification

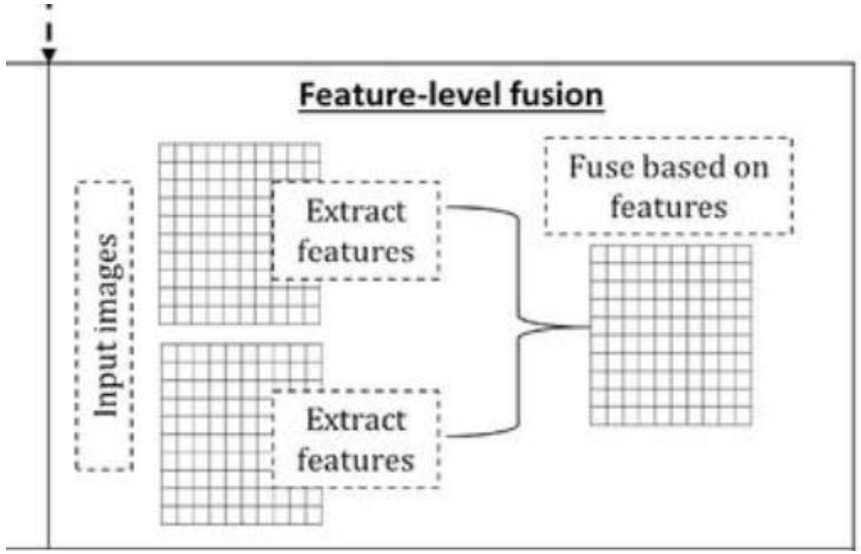
Feature Level Fusion

Obj 2	Activity 9 : Generate 2 fusion features by fuse VGG-19 and handcrafted features & fuse ResNet-101 and handcrafted features by using feature-level fusion

Fusion Features = Feature Level Fusion

is a technique where features extracted from multiple sources or models are combined into a single,





Feed-Forward Neutral Network

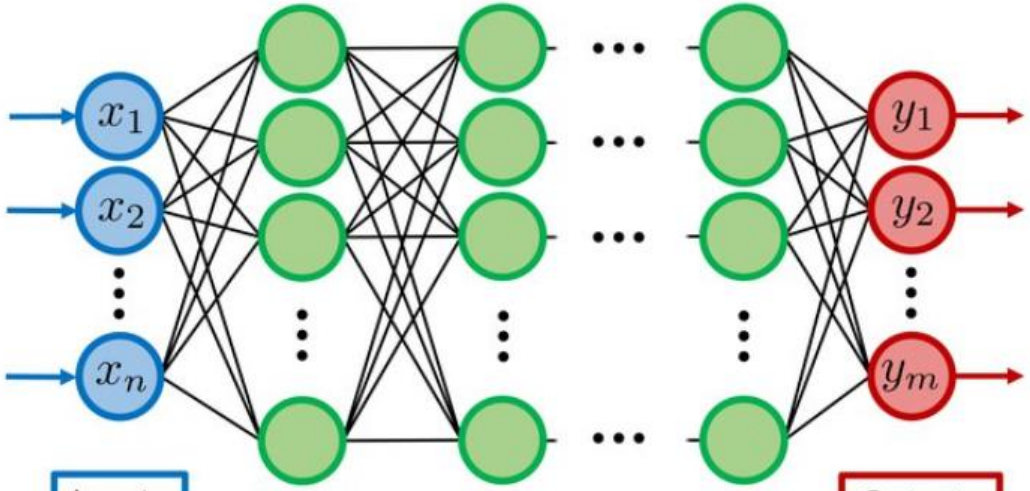
Obj 3

Activity 10 : Feed Forward Neutral Network (FFNN) with network features

Activity 11 : Feed Forward Neutral Network (FFNN) with fusion features

FFNN
(Feed- Forward Neutral Network)

type of ***artificial neural network*** where each neuron in one layer is connected to every neuron in the next layer.



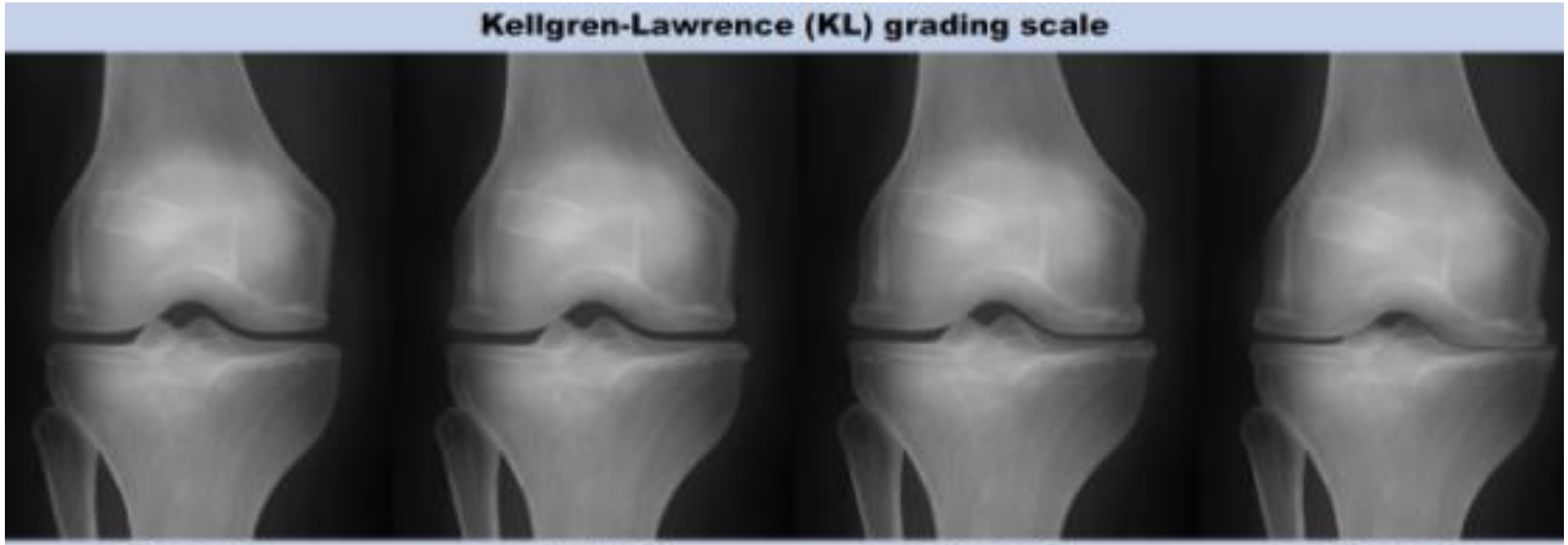
- Produces the final predictions
- For classification tasks, it might use a softmax activation function to output probabilities

set of features extracted from an image

- Consist of multiple layers of neurons
- Learn to capture complex patterns and relationships in the data

Description of Datasets – Classify

- Kellgren and Lawrence grades



CLASSIFICATION	Normal	Doubtful	Mild	Moderate	Severe
DESCRIPTION	No features of OA	Minute osteophyte: doubtful significance	Definite osteophyte: normal joint space	Moderate joint space reduction	Joint space greatly reduced: subchondral sclerosis

Description of Datasets - OAI

- Osteoarthritis Initiative (University of California, San Francisco's Osteoarthritis Initiative (OAI))

Types	Description of KL Grading	Data Sets
Grade 0	Healthy X-ray	3857
Grade 1	X-rays of doubtful narrowing of the joint with osteophytes tip over	1770
Grade 2	X-rays have minimal osteoarthritis containing joint space narrow with osteophytes	2578
Grade 3	X-rays have moderate osteoarthritis containing joint space stenosis, multiple osteophytes, and mild sclerosis	1286
Grade 4	X-rays have severe injuries containing large osteophytes and severe sclerosis with clear narrowing of the joints	295
Total		9786

Table 2.3 Description of data OAI

Performance Measurement

- Confusion Matrix is a table shown in below(Table 3) that **summarizes the performance** of a classification model, showing the counts of ***true positive(TP), true negative(TN), false positive(FP), and false negative(FN)*** predictions.

	Predict Positive	Predict Negative	Sum
Actual Positive	**TN**	**FP**	**TN + FP**
Actual Negative	**FN**	**TP**	**FN + TP**

Table 3 Confusion Matrix

Performance Measurement

Then , the systems' performance as determined by the evaluation scales listed in

$$\text{AUC} = \frac{\text{TPRate}}{\text{FPRate}} \times 100\%$$

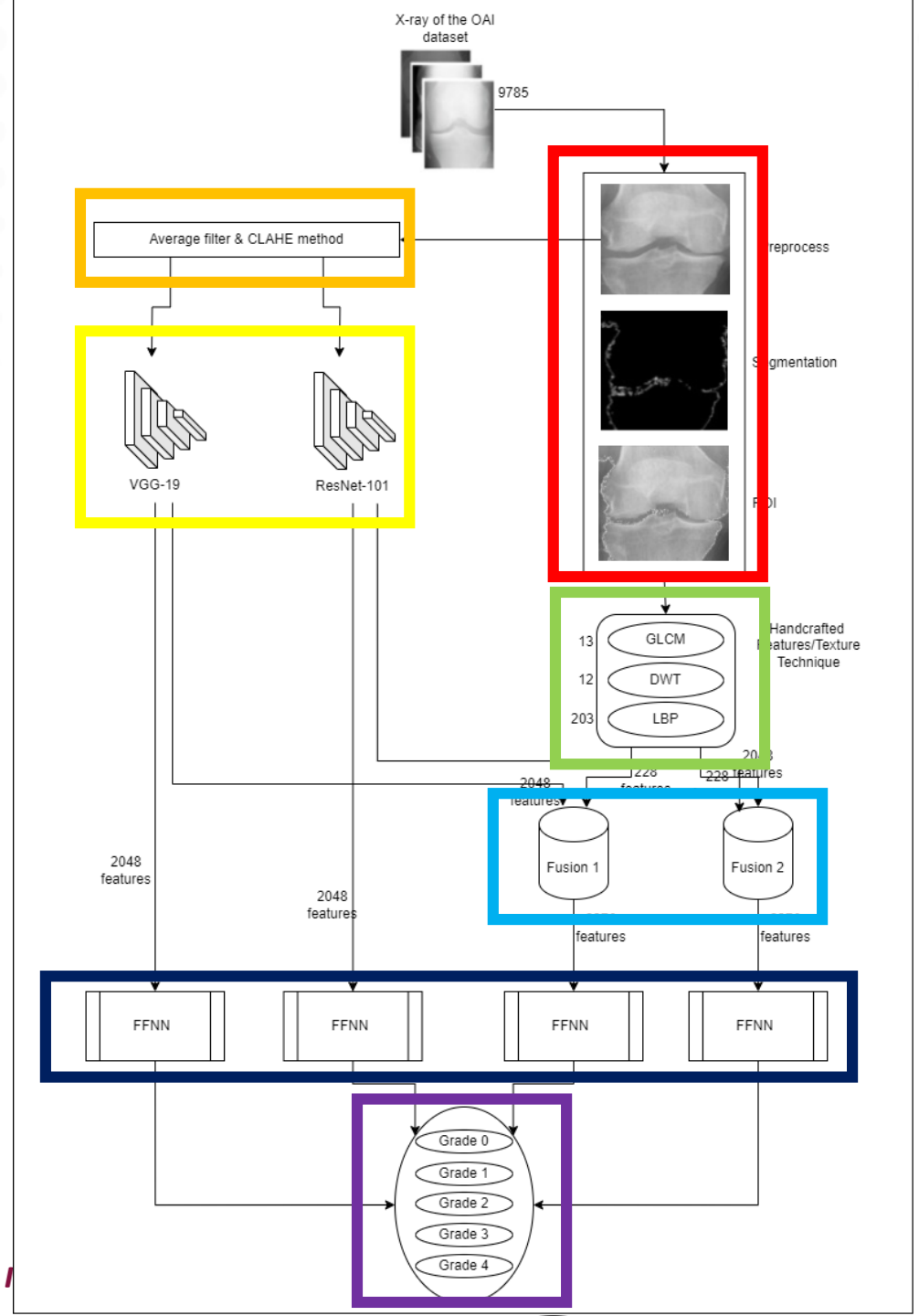
$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{FP}} \times 100\%$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\%$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \times 100$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \times 100\%$$

Architecture Diagram



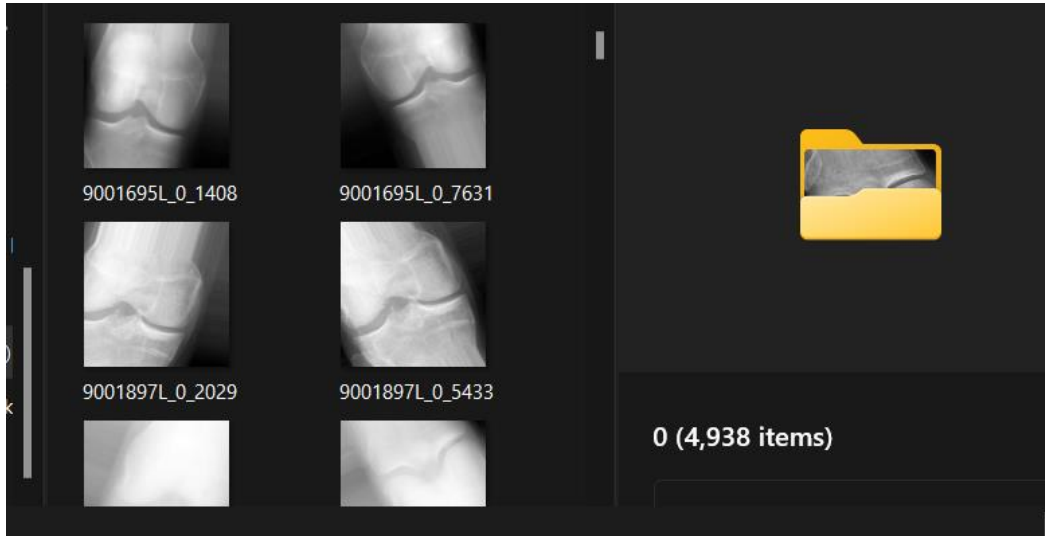
Balancing with Augmentation Data

Parameters	Value/description
Rotation	30 degree
Horizontal Flip	Probability: 0.5
Vertical Flip	Probability: 0.5
Width Shift	Range: ±20% of the total width
Height Shift	Range: ±20% of the total height
Zoom	Range: ±20% of the original size
Shear	Range: ±20% of the original size

Table 4.1 Balancing with Data Augmentation

aug.py > ...

```python
import os
import numpy as np
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_arra

# Define input and output directories for each grade
grades = {
    '0': {'input_dir': 'data/train/0', 'output_dir': 'data/aug/0'},
    '1': {'input_dir': 'data/train/1', 'output_dir': 'data/aug/1'},
    '2': {'input_dir': 'data/train/2', 'output_dir': 'data/aug/2'},
    '3': {'input_dir': 'data/train/3', 'output_dir': 'data/aug/3'},
    '4': {'input_dir': 'data/train/4', 'output_dir': 'data/aug/4'},
}

# Initialize ImageDataGenerator with augmentation options
datagen = ImageDataGenerator(
    rotation_range=30,
    horizontal_flip=True,
    vertical_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2
)
```



Before and After Augmentation Data

Classes	Grade 0	Grade 1	Grade 2	Grade 3	Grade 4	Total
Bef-aug	3857	1770	2578	1286	295	9786
Aft-aug	4938	4532	4950	4938	4914	24272

Table 4.2 Before and After Data Augmentation

Data Division and Arrangement

The OAI dataset will be partitioned into ***80:20 (train test) and validation sets will be obtained from 20% of training data*** segments because to its substantial size. Accordingly, training will occupy **80% of the data**, with the **remaining 20% going toward testing and validation** for each.

Grade 0 (Total: 4938)

- Training (80%): (4938 * 0.8 = 3950.4 = 3950)

- Testing (20%): (4938 * 0.2 = 987.6 = 987)

Grade 0(Training: 3950)

- Training (80%): (3950*0.8 = 3160)

- Validation (20%): (3950*0.2 = 790)

Grade 1(Total: 4532)

- Training (80%): (4532 * 0.8 = 3625.6 = 3626)

- Testing (20%): (4532 * 0.2 = 906.4 = 906)

Grade 1(Training: 3626)

- Training (80%): (3626 * 0.8 = 2900.8 =2901)

- Validation (20%): (3626 * 0.2 = 725.2=725)

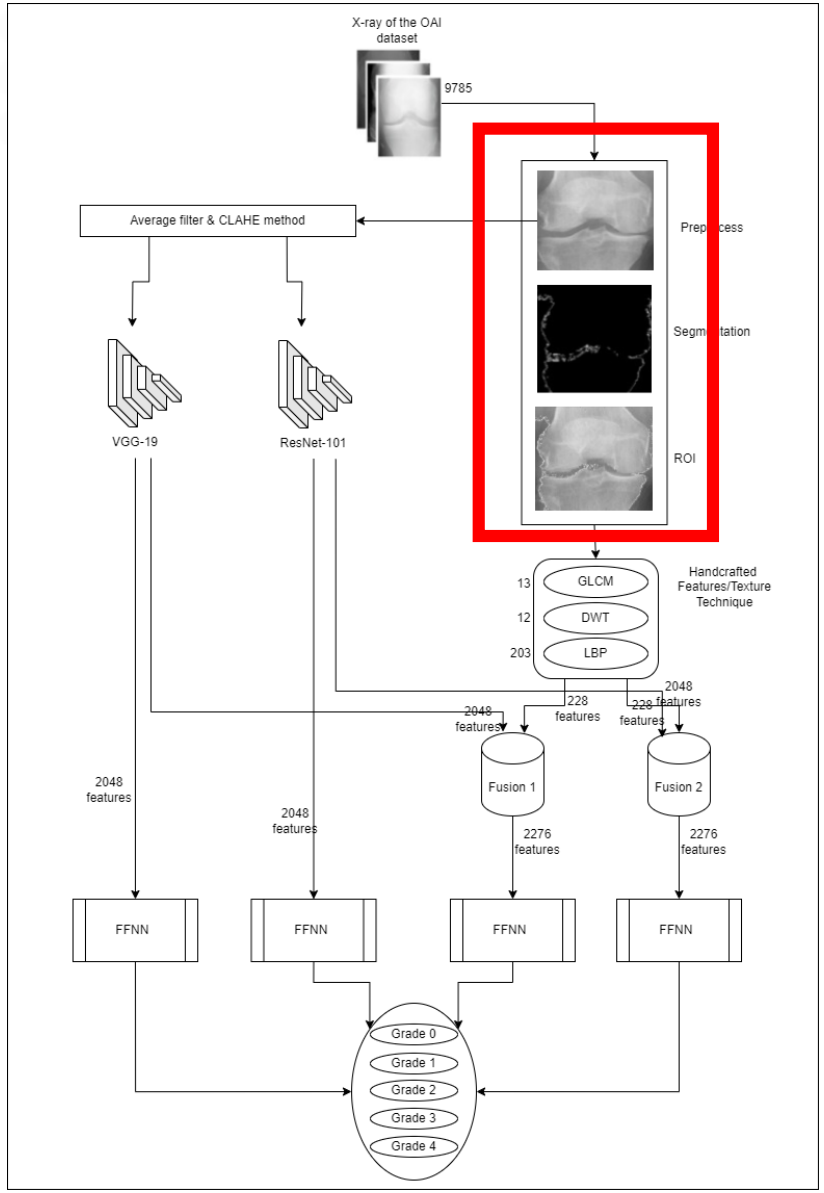
Grade 2 (Total:4950)

…

Phase/Classes	Training 80% (Validation 20%)	Testing 20%
Grade 0	3160 (790)	987
Grade 1	2901 (725)	906
Grade 2	3168 (792)	990
Grade 3	3160 (790)	988
Grade 4	3145 (786)	983

Table 4.3 Data Division and Calculation

Architecture Diagram

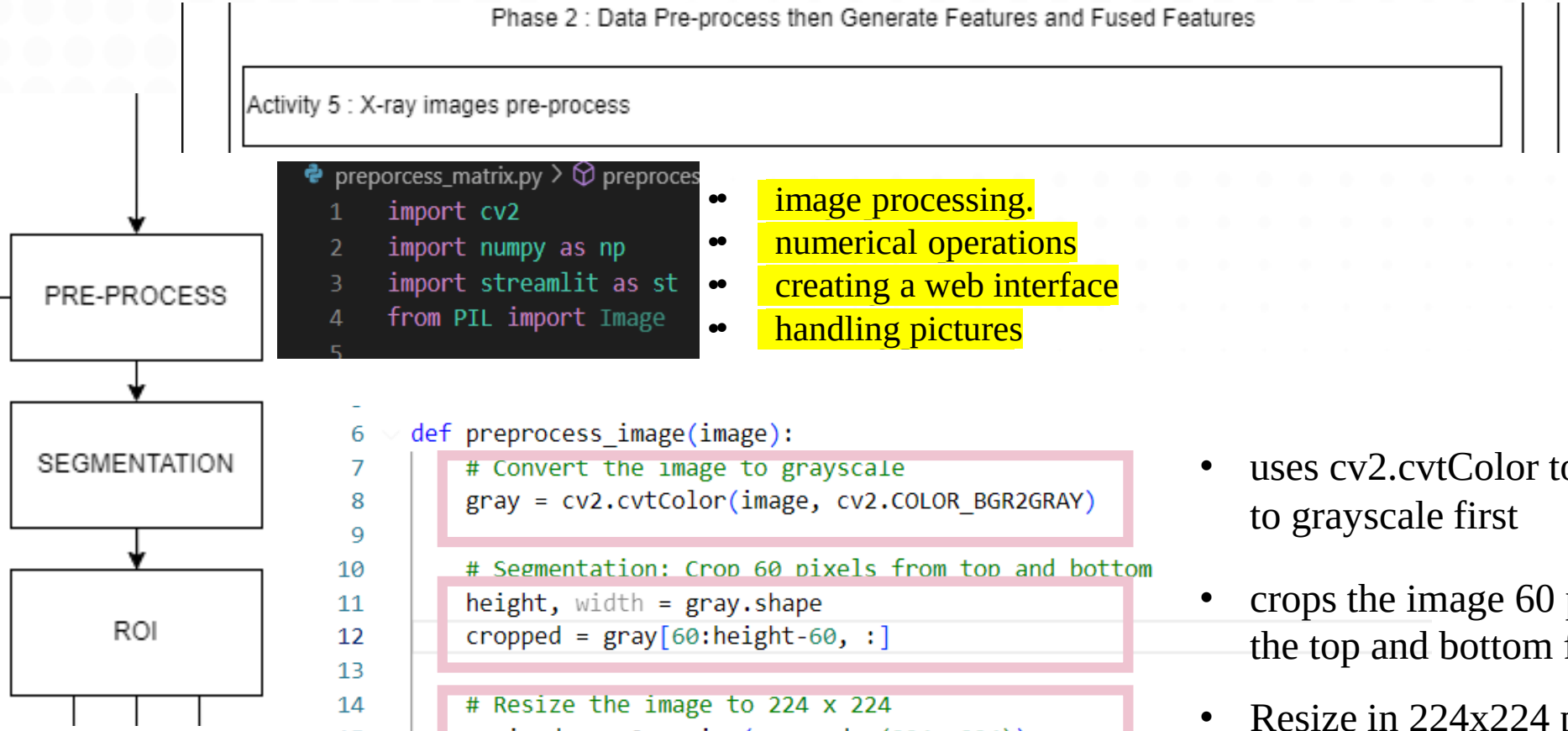


simplify the representation of an image into something that is more easier to analyze

Segmentation & ROI

selecting a specific part of an image that is particular interest

Datasets Pre-Processing – Segmentation & ROI



preporcess_matrix.py > preproces

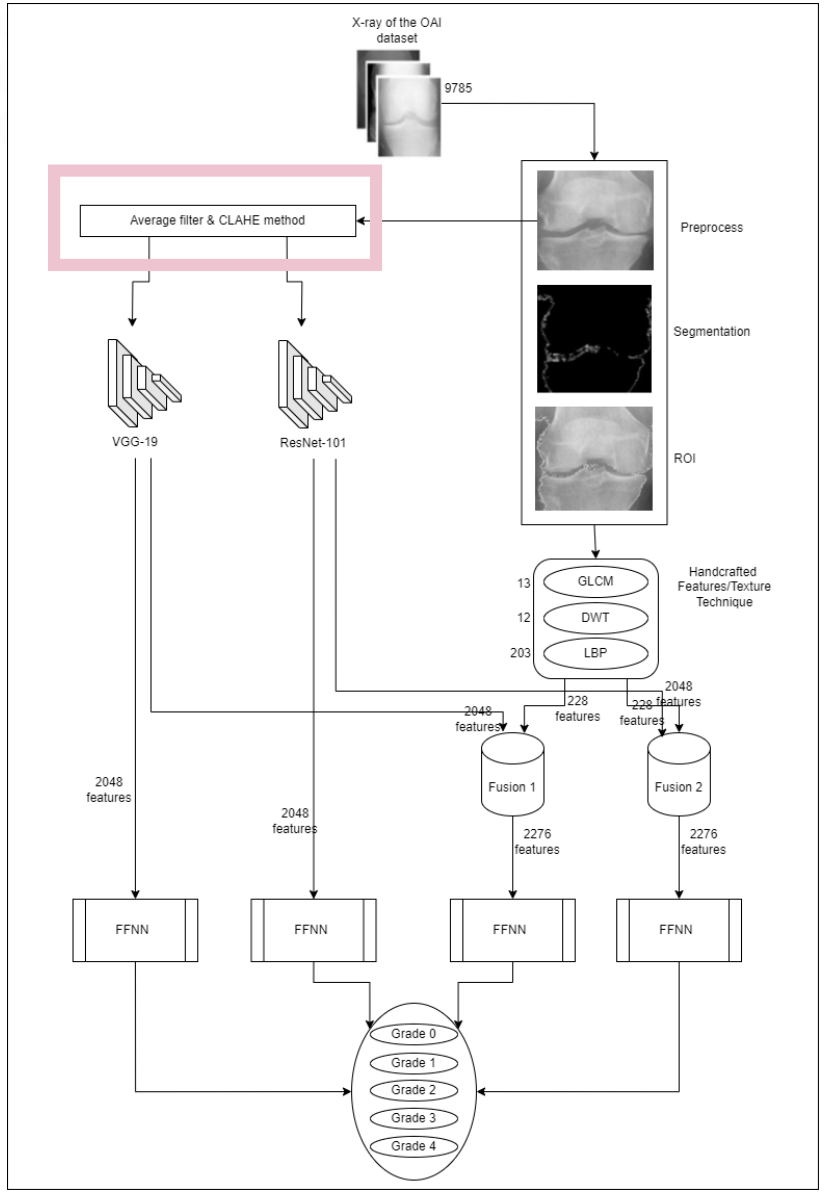
```python
import cv2
import numpy as np
import streamlit as st
from PIL import Image
```

- image processing.
- numerical operations
- creating a web interface
- handling pictures

```python
def preprocess_image(image):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Segmentation: Crop 60 pixels from top and bottom
    height, width = gray.shape
    cropped = gray[60:height-60, :]

    # Resize the image to 224 x 224
    resized = cv2.resize(cropped, (224, 224))

    return resized

def display_matrix(matrix):
    # Convert matrix to string format for better display
    matrix_str = "\n".join(["\t".join(map(str, row)) for row in matrix])
    return matrix_str
```

- uses cv2.cvtColor to convert the image to grayscale first
- crops the image 60 pixels on both the top and bottom for segmentation
- Resize in 224x224 pixel
- Transform into matrix

Architecture Diagram

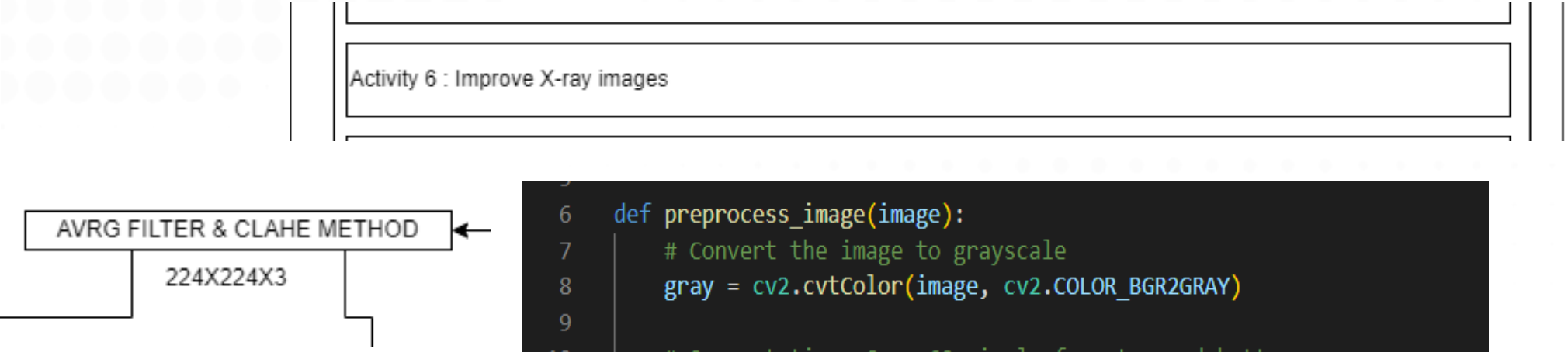


Use to reduce the noise , works by replacing each pixel's value with the average value of its neighboring pixels

Average Filter & CLAHE

improves the contrast of an image by applying adaptive histogram equalization

Datasets Pre-Processing – Average Filter & CLAHE



preprocess_filterclahe.py > prepr

```python
import cv2
import numpy as np
import streamlit as st
from PIL import Image
```

- image processing.
- numerical operations
- creating a web interface
- handling pictures

```python
def preprocess_image(image):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Segmentation: Crop 60 pixels from top and bottom
    height, width = gray.shape
    cropped = gray[60:height-60, :]

    # Apply average filtering
    avg_filtered = cv2.blur(cropped, (5, 5))

    # Apply CLAHE (Contrast Limited Adaptive Histogram Equalization)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    clahe_applied = clahe.apply(avg_filtered)

    # Resize the image to 224 x 224
    resized = cv2.resize(clahe_applied, (224, 224))

    return resized
```

- applies average filtering (`cv2.blur`) to reduce noise
- enhances contrast using CLAHE (`cv2.createCLAHE`)

Datasets After Pre-Processing

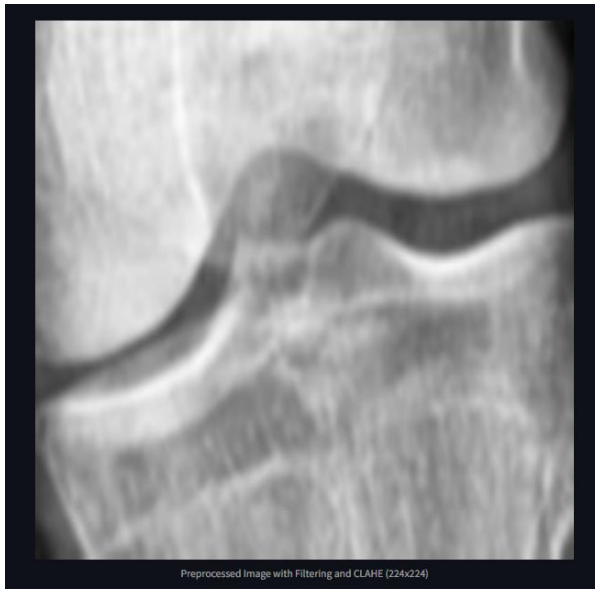
Before

After

- Pre-process segmentation and ROI

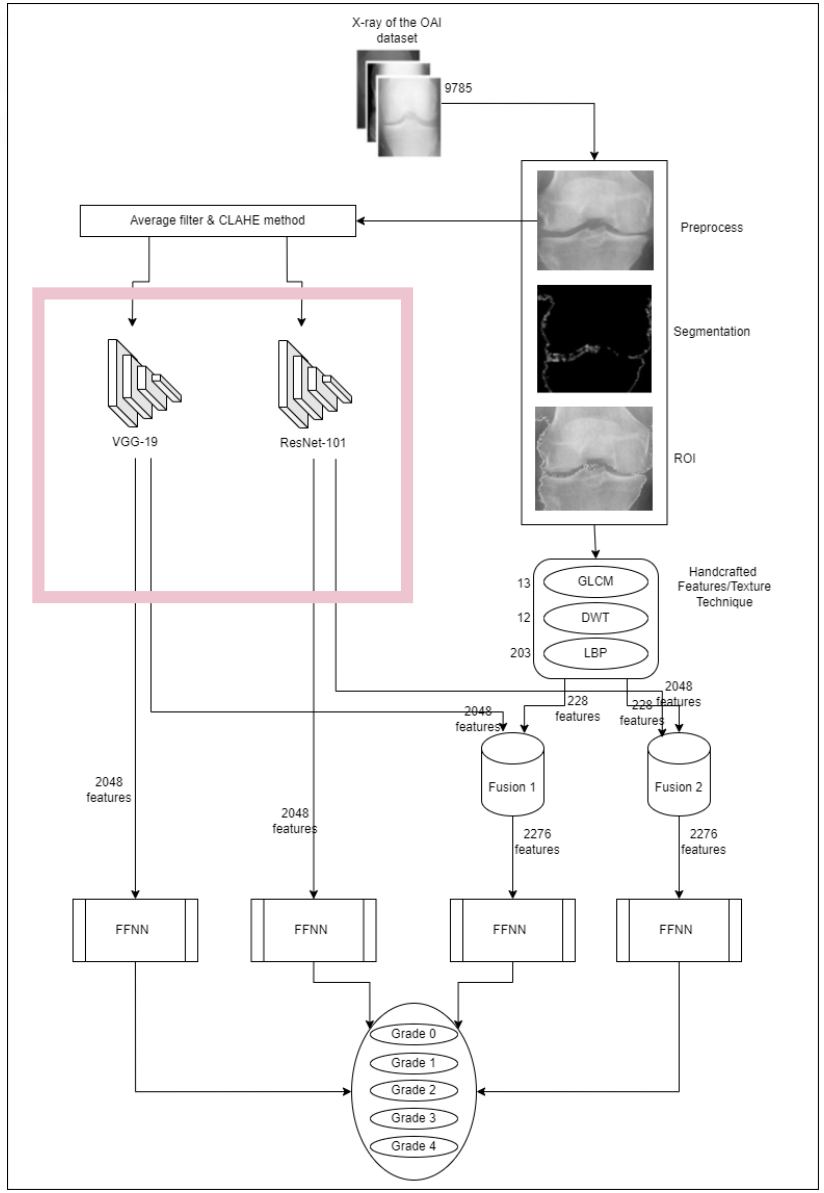
- Pre-process Avg filter and CLAHE

Before



After

Architecture Diagram



Network Features

CNN model

VGG-19

ResNet-101

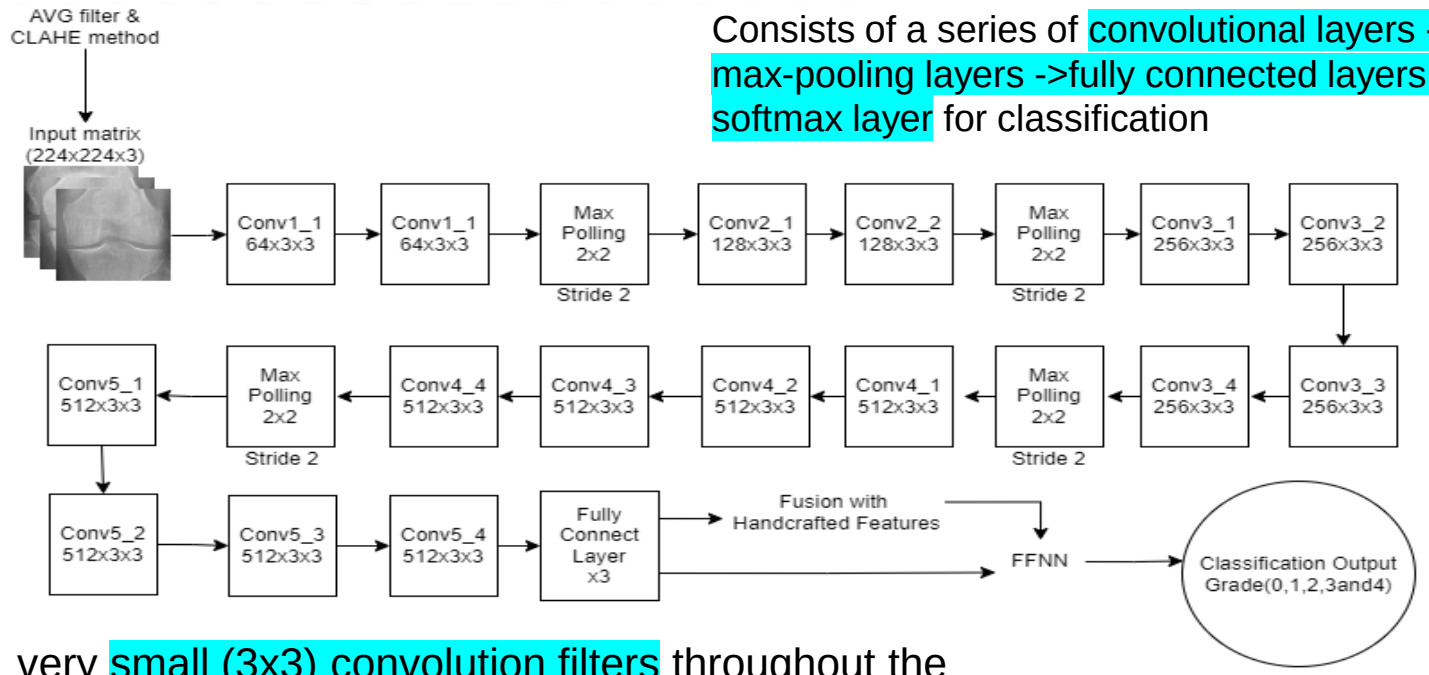
CNN (VGG-19)

Visual Geometry Group 19-layer network

It's a deep convolutional neural network with 19 layers.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	1,792
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36,928
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73,856
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147,584
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295,168
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590,080
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590,080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590,080
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1,180,160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2,359,808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2,359,808
conv2d_11 (Conv2D)	(None, 28, 28, 512)	2,359,808
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2,359,808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2,359,808
conv2d_14 (Conv2D)	(None, 14, 14, 512)	2,359,808
conv2d_15 (Conv2D)	(None, 14, 14, 512)	2,359,808
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102,764,544
dense_1 (Dense)	(None, 4096)	16,781,312
dense_2 (Dense)	(None, 1000)	4,097,000

Total params: 143,667,240 (548.05 MB)
Trainable params: 143,667,240 (548.05 MB)
Non-trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
PS C:\UTM Degree\y3s2\PSM_Dr Nies\KOA>



Consists of a series of convolutional layers -> max-pooling layers ->fully connected layers -> softmax layer for classification

very small (3x3) convolution filters throughout the network, which allows for deeper architectures

```
network > VGG19_model.py > create_vgg19_model
1  import cv2
2  import numpy as np
3  import streamlit as st
4  from PIL import Image
5  import tensorflow as tf
6  from tensorflow.keras import layers, mod
7  import matplotlib.pyplot as plt
8
```

- image processing.
- numerical operations
- creating a web interface
- handling pictures
- building and running neural networks
- use to create static, animated, and interactive visualizations

```
def create_vgg19_model(input_shape=(224, 224, 3)):
    model = models.Sequential()

    # Block 1
    model.add(layers.Conv2D(64, (3, 3), padding='same', input_shape=input_shape))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(64, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(layers.Conv2D(128, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(128, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(layers.Conv2D(256, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(256, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(256, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(256, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Conv2D(512, (3, 3), padding='same'))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Fully Connected Layers
    model.add(layers.Flatten())
    model.add(layers.Dense(4096))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Dense(4096))
    model.add(layers.ReLU())  # ReLU activation
    model.add(layers.Dense(1000, activation='softmax'))  # Output layer

    return model
```

- Defines the model as a sequence of layers.
- Consists of convolutional layers followed by ReLU activations and max pooling layers.
- Flattens the output from the convolutional layers and passes it through two dense (fully connected) layers with ReLU activations, and then a final dense layer with softmax activation for classification

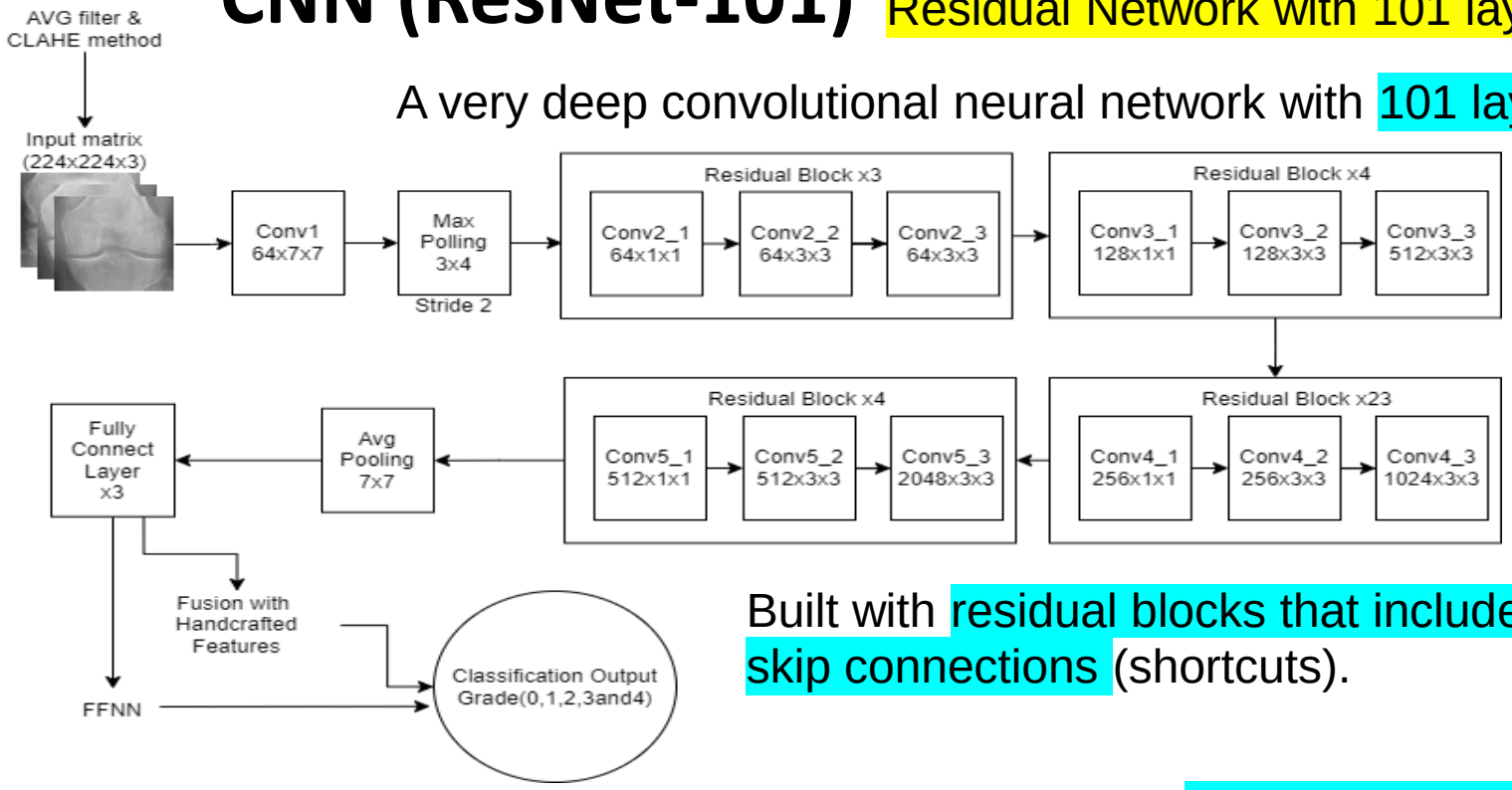
```
    # Expand the grayscale image to 3 channels
    preprocessed_image_3ch = np.stack((preprocessed_image,)*3, axis=-1)

    # Normalize the image
    preprocessed_image_3ch = preprocessed_image_3ch / 255.0

    # Add batch dimension
    preprocessed_image_3ch = np.expand_dims(preprocessed_image_3ch, axis=0)

    # Load and compile the VGG19 model
    model = create_vgg19_model(input_shape=(224, 224, 3))

    # Here you should load the pre-trained weights
    # model.load_weights('path_to_vgg19_weights.h5')

    # For the purpose of this example, we'll skip loading weights
    st.write("Predicting using VGG19 model...")

    # Perform the prediction
    predictions = model.predict(preprocessed_image_3ch)

    # Since we're not doing classification, just display the 2048 features
    st.write("Extracted features:")
    st.write(predictions[0])

    # Calculate total number of features extracted
    total_features = predictions.shape[1]
    st.write(f"Total features extracted: {total_features}")

if __name__ == "__main__":
    main()
```

- Converts the grayscale image to a 3-channel image
- Normalizes the pixel values to [0, 1]
- Adds a batch dimension, making it suitable for model input.
- Uses the model to extract features from the preprocessed image
- Shows the extracted features and the total number of features extracted

CNN (ResNet-101)

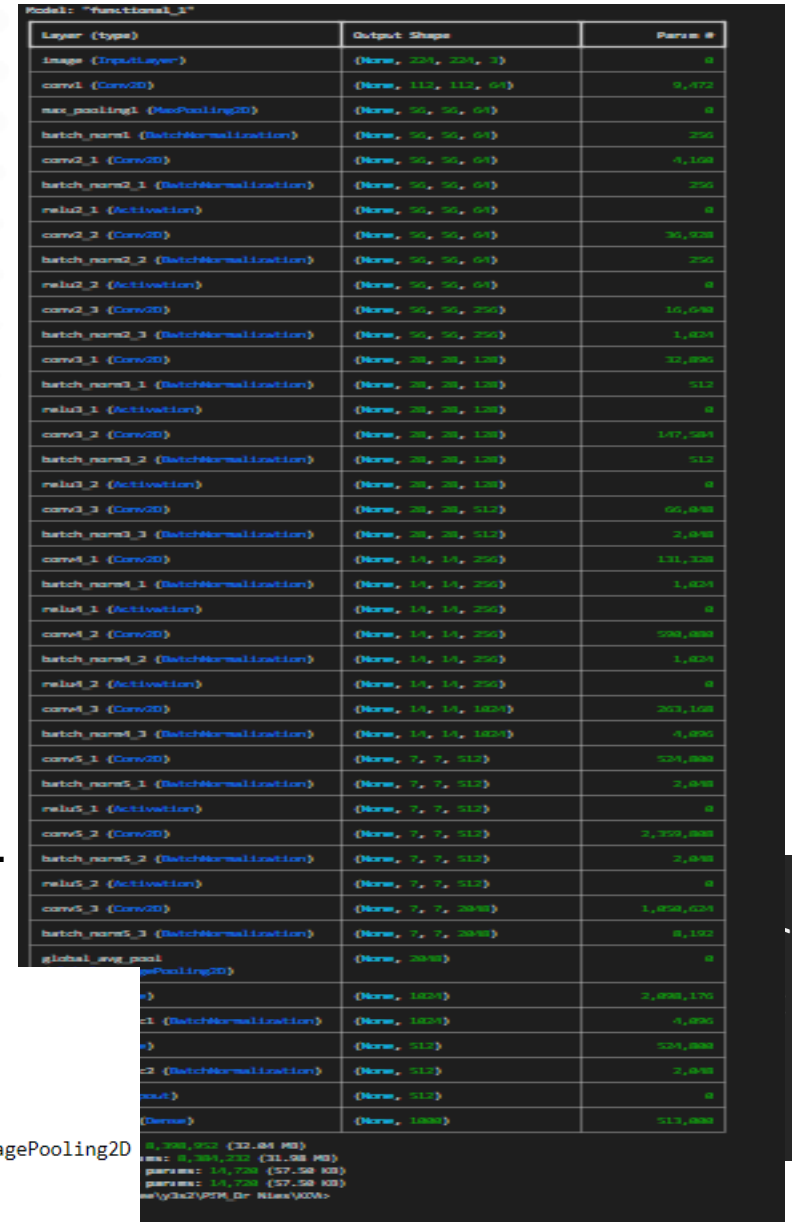
Residual Network with 101 layers.

A very deep convolutional neural network with 101 layers.



Built with residual blocks that include skip connections (shortcuts).

skip connections help to mitigate the vanishing gradient problem, allowing for very deep networks.



network > Resnet101_model.py > create_resnet101_model

```
1  import streamlit as st
2  import numpy as np
3  import cv2
4  from PIL import Image
5  import tensorflow as tf
6  from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, BatchNormal
7  from tensorflow.keras.models import Model
8  import matplotlib.pyplot as plt
```

- creating a web interface
- numerical operations
- image processing.
- handling pictures
- deep learning framework for building and training models.
- building and running neural networks
- use to create static, animated, and interactive visualizations

RESULT – Preliminary Result

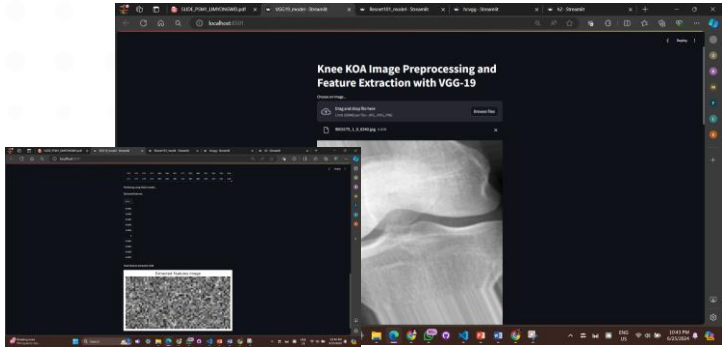
```python
def create_resnet101_model(input_shape=(224, 224, 3)):
    inputs = Input(shape=input_shape, name="image")

    # Initial Conv Layer
    x = Conv2D(64, (7, 7), strides=2, padding='same', name='conv1')(inputs)
    x = MaxPooling2D((3, 3), strides=2, padding='same', name='max_pooling1')(x)
    x = BatchNormalization(name='batch_norm1')(x)

    # Example Block 1
    x = Conv2D(64, (1, 1), strides=1, padding='same', name='conv2_1')(x)
    x = BatchNormalization(name='batch_norm2_1')(x)
    x = Activation('relu', name='relu2_1')(x)

    x = Conv2D(64, (3, 3), strides=1, padding='same', name='conv2_2')(x)
    x = BatchNormalization(name='batch_norm2_2')(x)
    x = Activation('relu', name='relu2_2')(x)

    x = Conv2D(256, (1, 1), strides=1, padding='same', name='conv2_3')(x)
    x = BatchNormalization(name='batch_norm2_3')(x)

    # Example Block 2
    x = Conv2D(128, (1, 1), strides=2, padding='same', name='conv3_1')(x)
    x = BatchNormalization(name='batch_norm3_1')(x)
    x = Activation('relu', name='relu3_1')(x)

    x = Conv2D(128, (3, 3), strides=1, padding='same', name='conv3_2')(x)
    x = BatchNormalization(name='batch_norm3_2')(x)
    x = Activation('relu', name='relu3_2')(x)

    x = Conv2D(512, (1, 1), strides=1, padding='same', name='conv3_3')(x)
    x = BatchNormalization(name='batch_norm3_3')(x)

    # Example Block 3
    x = Conv2D(256, (1, 1), strides=2, padding='same', name='conv4_1')(x)
    x = BatchNormalization(name='batch_norm4_1')(x)
    x = Activation('relu', name='relu4_1')(x)

    x = Conv2D(256, (3, 3), strides=1, padding='same', name='conv4_2')(x)
    x = BatchNormalization(name='batch_norm4_2')(x)
    x = Activation('relu', name='relu4_2')(x)

    x = Conv2D(1024, (1, 1), strides=1, padding='same', name='conv4_3')(x)
    x = BatchNormalization(name='batch_norm4_3')(x)

    # Example Block 4
    x = Conv2D(512, (1, 1), strides=2, padding='same', name='conv5_1')(x)
    x = BatchNormalization(name='batch_norm5_1')(x)
    x = Activation('relu', name='relu5_1')(x)

    x = Conv2D(512, (3, 3), strides=1, padding='same', name='conv5_2')(x)
    x = BatchNormalization(name='batch_norm5_2')(x)
    x = Activation('relu', name='relu5_2')(x)

    x = Conv2D(2048, (1, 1), strides=1, padding='same', name='conv5_3')(x)
    x = BatchNormalization(name='batch_norm5_3')(x)

    # Average Pooling
    x = AveragePooling2D(pool_size=(7, 7), name='average_pool')(x)
    x = Reshape((2048,))(x)  # Flatten the output to a 1D tensor

    # Fully Connected Layers
    x = Dense(1024, activation='relu', name='dense1')(x)
    x = BatchNormalization(name='batch_norm_fc1')(x)
    x = Dense(512, activation='relu', name='dense2')(x)
    x = BatchNormalization(name='batch_norm_fc2')(x)

    # Dropout Layer
    dropout_rate = 0.5
    x = Dropout(dropout_rate, name='dropout')(x)

    outputs = Dense(1000, activation='softmax', name='predictions')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model
```

- Defines a ResNet-101 model architecture using Keras Functional API

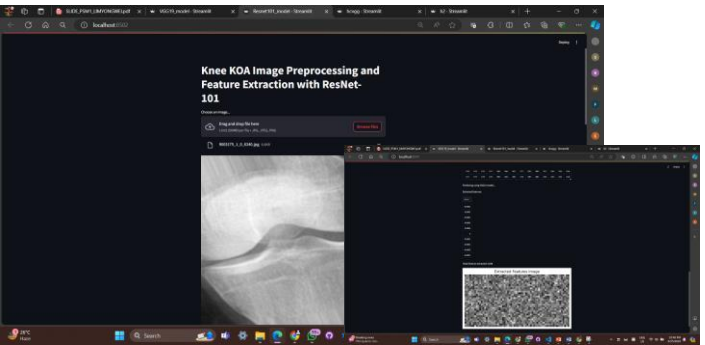
```python
# Convert the preprocessed image to a format that can be displayed in Streamlit
preprocessed_pil_image = Image.fromarray(preprocessed_image)

# Display the preprocessed image
st.image(preprocessed_pil_image, caption="Preprocessed Image with Filtering and CLAHE", use_column_width=True)

# Expand the grayscale image to 3 channels
preprocessed_image_3ch = np.stack((preprocessed_image,)*3, axis=-1)

# Normalize the image
preprocessed_image_3ch = preprocessed_image_3ch / 255.0

# Add batch dimension
preprocessed_image_3ch = np.expand_dims(preprocessed_image_3ch, axis=0)

# Load and compile the ResNet-101 model
model = create_resnet101_model(input_shape=(224, 224, 3))

# Here you should load the pre-trained weights
# model.load_weights('path_to_resnet101_weights.h5')

# For the purpose of this example, we'll skip loading weights
st.write("Extracting features using ResNet-101 model...")

# Extract features from the layer before the final dense layer
feature_extraction_model = Model(inputs=model.input, outputs=model.get_layer('global_avg_pool').output)

# Perform feature extraction
features = feature_extraction_model.predict(preprocessed_image_3ch)

# Display the features
st.write("Extracted Features:")
st.write(features)

# Calculate total number of features extracted
total_features = features.size
st.write(f"Total features extracted: {total_features}")
```

- Same function as in VGG

Proposed Model Result After Extract VGG-19 and ResNet-101

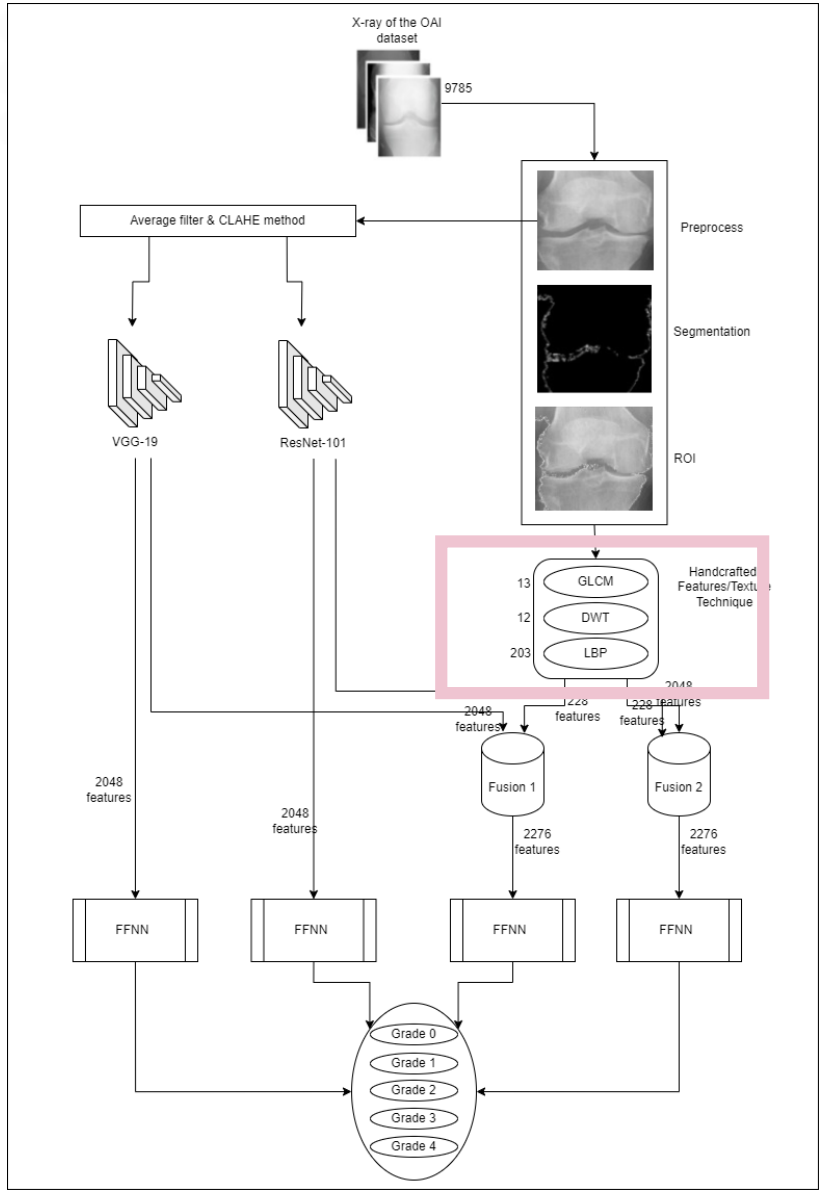


- VGG-19



- ResNet-101

Architecture Diagram



refer to manually designed features used in traditional machine learning and image processing tasks

Handcrafted Features

Discrete Wavelet Transform (DWT)
Gray-Level Co-occurrence Matrix (GLCM)
Local Binary Pattern (LBP)

Proposed Model – Handcrafted (DWT)

handcrafted > dwt.py > main

```python
import pywt
import numpy as np
import cv2
```

- performing discrete wavelet transforms (DWT)
- numerical computations
- **o**penCV library for computer vision tasks.

Discrete Wavelet Transform = decomposes a signal or an image into different frequency components

```python
def extract_dwt_features(image):
    # Convert the image to grayscale if it's not already
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Perform DWT on the image
    coeffs = pywt.dwt2(image, 'haar')  # Using Haar wavelet, you can choose others as needed
    cA, (cH, cV, cD) = coeffs  # cA: Approximation, cH: Horizontal detail, cV: Vertical detail, cD: Diagonal detail

    # Select a fixed number of features from each coefficient
    features = np.concatenate([
        cA.flatten()[:3],
        cH.flatten()[:3],
        cV.flatten()[:3],
        cD.flatten()[:3]
    ])

    return features
```

- performs a 2D discrete wavelet transform (DWT) using the Haar wavelet (‘haar’)
- extracts a fixed number of features (first 3 values) from each of the resulting coefficients (cA,cH,cV,cD)

Handcrafted Features (GLCM)

handcrafted > glcm.py > ...

```python
import cv2
import numpy as np
from skimage.feature import graycomatrix, graycoprops

def extract_glcm_features(image):
    # Convert the image to grayscale if it's not already
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calculate GLCM with specified parameters
    distances = [1]  # Using only one distance to simplify feature count
    angles = [0, np.pi/4, np.pi/2, 3*np.pi/4]  # Specify angles for GLCM

    glcm = graycomatrix(image, distances=distances, angles=angles, symmetric=True, normed=True)

    # Extracting some properties from GLCM
    contrast = graycoprops(glcm, 'contrast').ravel()
    dissimilarity = graycoprops(glcm, 'dissimilarity').ravel()
    homogeneity = graycoprops(glcm, 'homogeneity').ravel()
    energy = graycoprops(glcm, 'energy').ravel()
    correlation = graycoprops(glcm, 'correlation').ravel()

    # Concatenate all features into a single feature vector
    features = np.hstack([contrast, dissimilarity, homogeneity, energy, correlation])

    # Pad or truncate features to ensure size 13
    if features.shape[0] < 13:
        features = np.pad(features, (0, 13 - features.shape[0]), 'constant')
    else:
        features = features[:13]

    return features
```

- openCV library for computer vision tasks
- numerical computations
- calculating Gray-Level Co-occurrence Matrix (GLCM) and its properties.

Gray-Level Co-occurrence Matrix = captures the spatial relationship between pixels by calculating how often pairs of pixel intensities occur together

- extracts a fixed number of features (first 3 values) from each of the resulting coefficients
- Computes GLCM using graycomatrix function with specified distances
- Compute properties
- Concatenates all these properties into a single feature vector

Handcrafted Features (LBP)

```
handcrafted / lbp.py / main
1   import numpy as np
2   import cv2
3   from skimage.feature import local_binary_pattern
```

- numerical computations
- openCV library for computer vision tasks
- compute Local Binary Patterns (LBP)

Local Binary Pattern = describes the local texture pattern of an image by comparing each pixel with its neighboring pixels

```
def extract_lbp_features(image):
    # Convert the image to grayscale if it's not already
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Parameters for LBP
    radius = 3
    n_points = 8 * radius

    # Calculate LBP
    lbp = local_binary_pattern(image, n_points, radius, method='uniform')

    # Calculate histogram of LBP and normalize it
    hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, n_points + 3), range=(0, n_points + 2), density=True)

    # Pad or truncate histogram to ensure size 203
    if hist.shape[0] < 203:
        hist = np.pad(hist, (0, 203 - hist.shape[0]), 'constant')
    else:
        hist = hist[:203]

    return hist
```

- Checks if the input image is grayscale
- Sets parameters for LBP calculation
- Computes LBP
- Computes the histogram of LBP value
- Size less then 203

Proposed Model Result after Handcrafted Features(DWT , GLCM , LBP)

Total DWT features extracted: 12

Total GLCM features extracted: 13

Total LBP features extracted: 203

DWT Features:

value
397.5
410.5
413.5
0.5
0.5
0.5
-3.5
-1.5
-1.5
-0.5

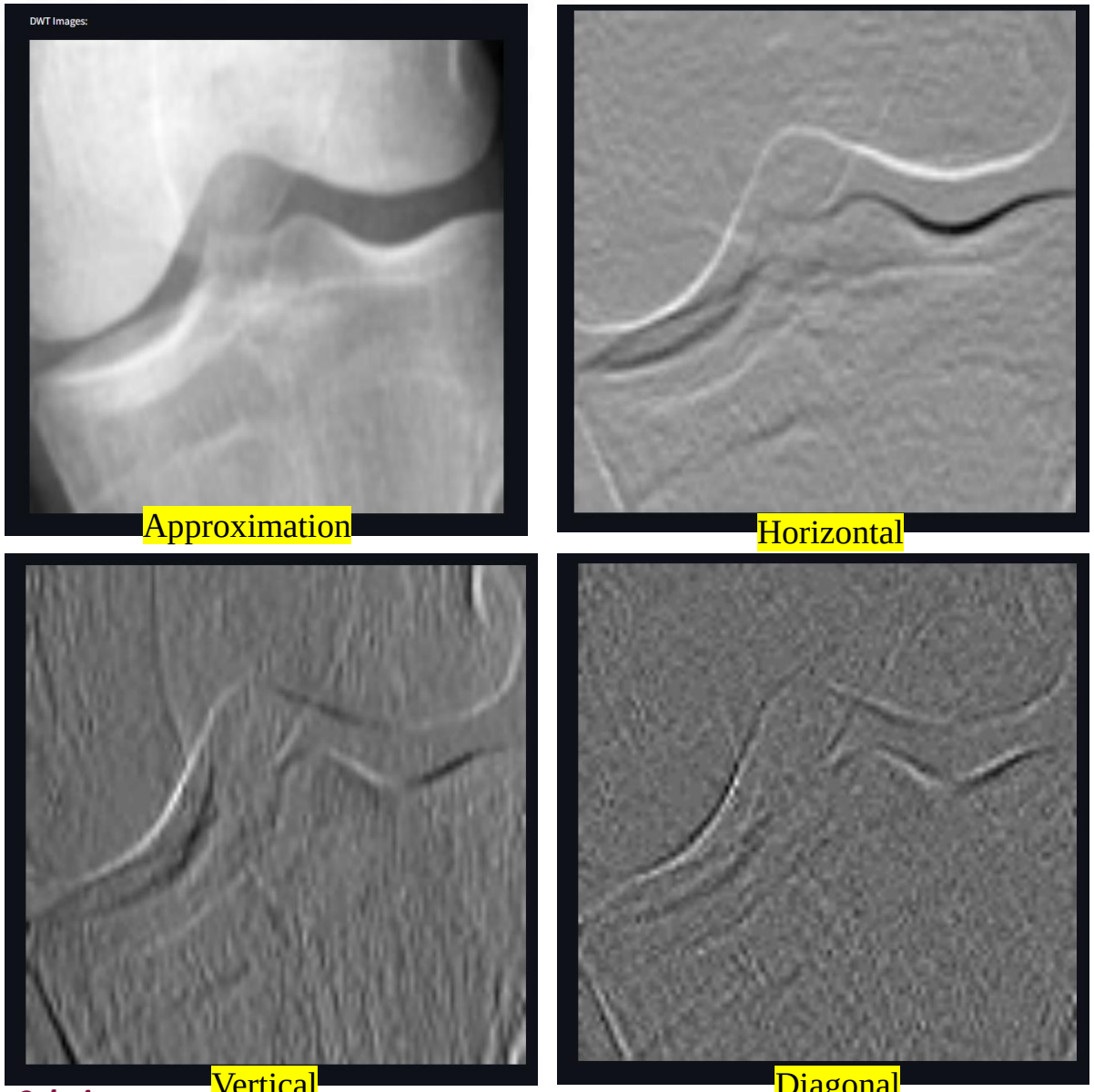
GLCM Features:

value
13.5843
28.3796
11.8677
20.8187
2.5844
3.3953
1.9781
3.1497
0.3403
0.3

LBP Features:

value
0.0382
0.0142
0.0202
0.0211
0.0217
0.0199
0.019
0.0211
0.0239
0.0276

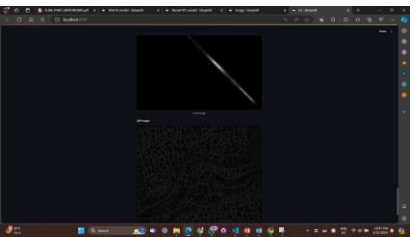
- DWT , GLCM , LBP



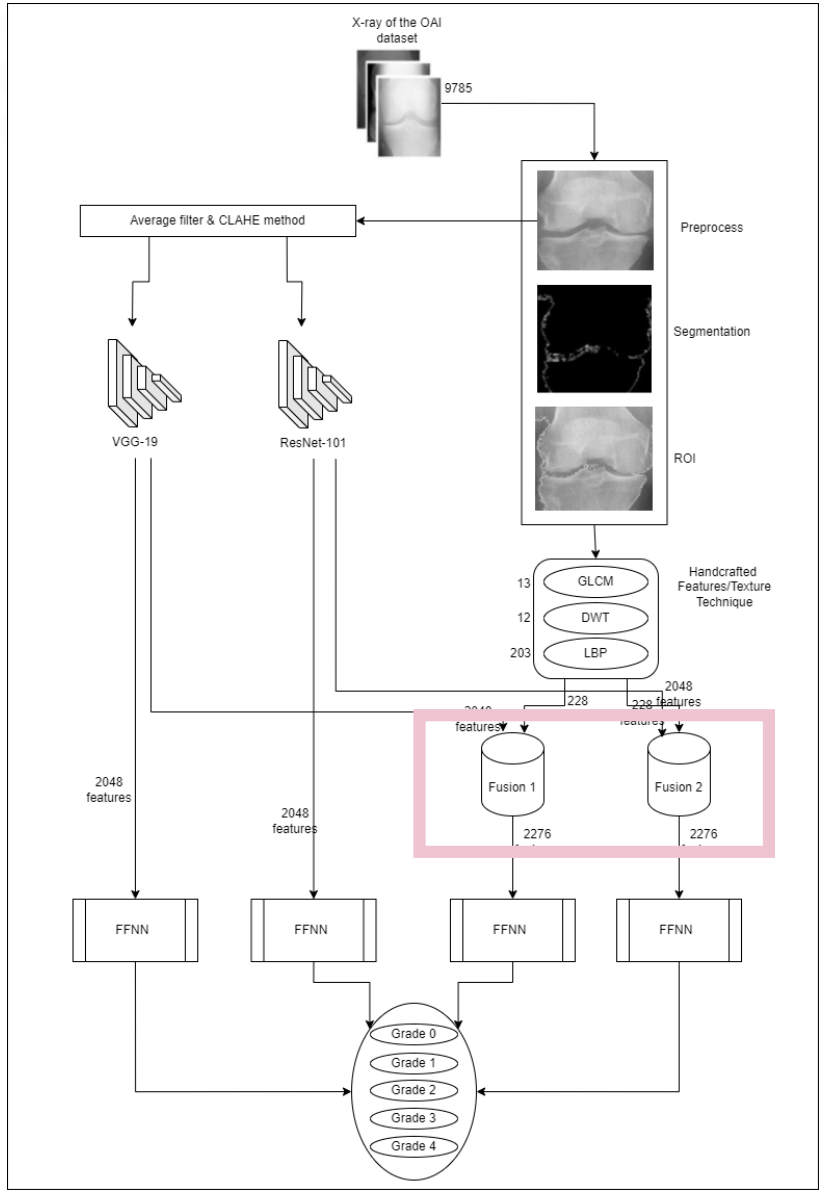
- DWT

Proposed Model Result after Handcrafted Features(DWT , GLCM , LBP)

- GLCM
- LBP



Architecture Diagram



refers to the process of combining multiple sets of features to create a more comprehensive

Fusion Features

Early Fusion (Feature-Level Fusion):

Features are combined at the initial stage before being fed into the model. This involves concatenating feature vectors or applying dimensionality reduction techniques like Principal Component Analysis (PCA)

Late Fusion (Decision-Level Fusion):

Models are trained separately on different feature sets, and their outputs are combined at the decision level.

Feature Level Fusion (VGG19 and Handcrafted)

handcrafted > hcvgg.py > ...

```python
import cv2
import numpy as np
import streamlit as st
from PIL import Image
import matplotlib.pyplot as plt
# Import functions directly from their respective modules
from h2 import extract_dwt_features, extract_glcm_features, extract_lbp_features
from vgg import extract_vgg19_features
```

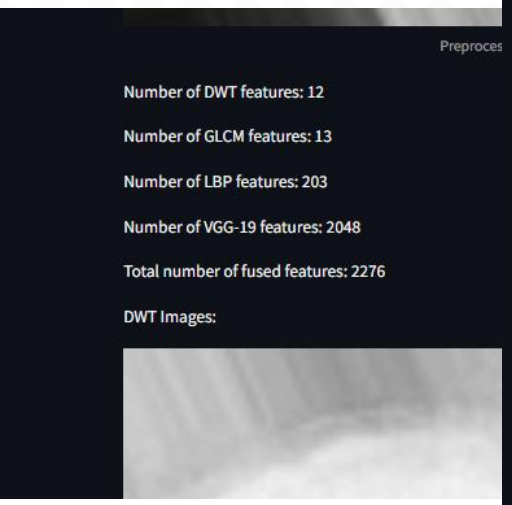
- OpenCV library for image processing tasks.
- Library for numerical computations in Python.
- Library for building interactive web apps.
- Python Imaging Library for image processing tasks.
- Custom functions imported from respective modules

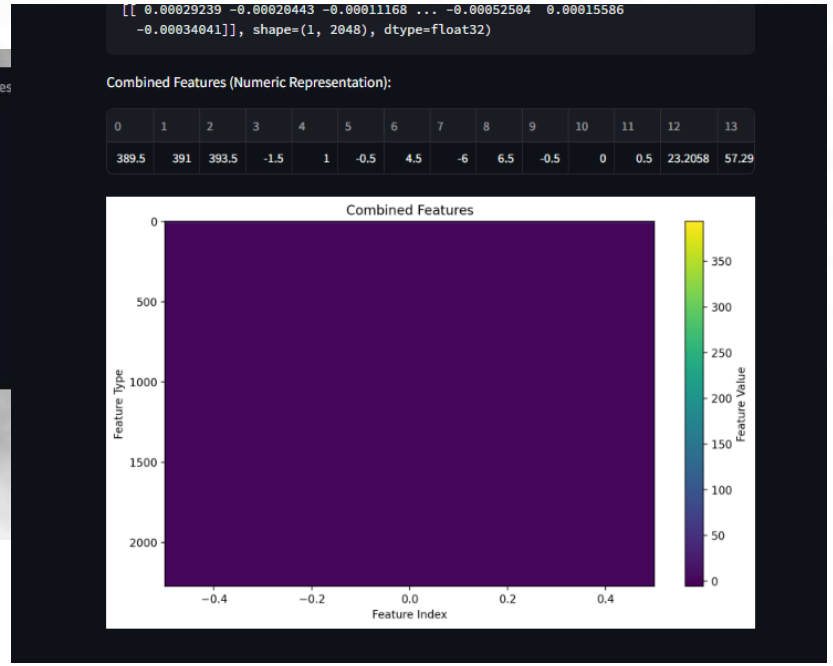
- Concatenates all feature arrays
- Displays the shape and contents

```python
# Combine features
combined_features = np.concatenate((dwt_features, glcm_features, lbp_features, vgg19_features), axis=1)

# Optionally, perform classification or further analysis with combined features
st.write("Combined Features Shape:", combined_features.shape)

# Display or use combined features as needed
st.write("Combined Features:")
st.write(combined_features)
```

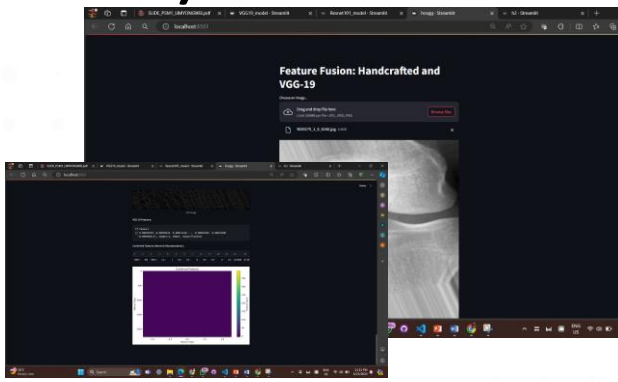
Same function use in Resnet101 fusion with handcrafted

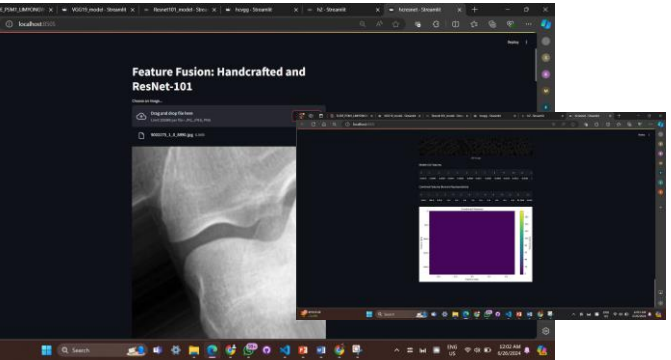
Proposed Model Result after Fusion Feature(Network and Handcrafted)



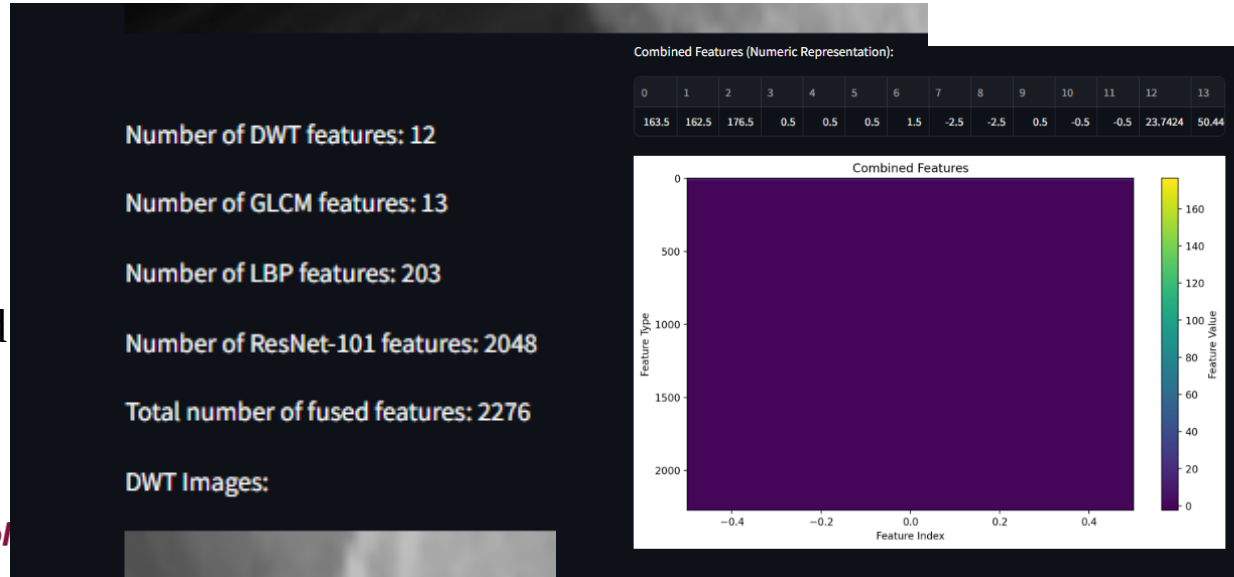


- VGG-19 and handcrafted





- ResNet-101 and handcrafted



Corresponding Hyperparameters definition and Ranges

Element index	Corresponding hyperparameters definition	Corresponding range
1.	Training loss function	Categorical Crossentropy
2.	Training batch size	[8 – 256]
3.	Model dropout ratio	[0.1, 0.5]
4.	Transfer learning freezing ratio	[0,1]
5.	Weights(parameters) optimizer	SGD, Adam, RMSprop
6.	Dimension scaling technique	Resizing, Normalization
7.	Utilize data augmentation techniques or not	[YES, NO]
8.	The value of rotation (If YES)	0 ◦ →30◦
9.	The value of width shift (If YES)	[0 – 0.2]
10.	The value of height shift (If YES)	[0 – 0.2]
11.	The value of shear (If YES)	[0 – 0.2]
12.	The value of zoom (If YES)	[0 – 0.2]
13.	The value of horizontal flipping flag (If YES)	[Yes, No]
14.	The value of vertical flipping flag (If YES)	[Yes, No]

Table 4.4 Hyperparameters

Configuration

Configuration	Specifications
Datasets	Table 3.2
Apply data shuffling?	Yes
Input images size	224x224x3
Hyperparameters optimizer	Convolutional Neutral Network
Train split ratio	80% to 20%(80% for training9and validation) and 20% for testing)
Activation function	SoftMax
Pre-trained models	VGG-19, ResNet-101
Pre-trained parameters initializers	Table 4.2
Hyperparameters	Table
Scripting language	Python
Python major packages	Streamlit, Tensorflow, Keras, NumPy, OpenCV, Matplotlib, Skimage and PIL
Working environment	Visual Studio

Table 4.5 Configuration

Achievement of Project Objective

- Two of the three goals of this study, which were to "Enhanced X-ray Analysis and Diagnosis of Knee Osteoarthritis Grade through Hybrid Technique Using Fusion of Network Features and Handcrafted Features," have been accomplished.

- The initial goal was to look into the characteristics and deep learning techniques already in use for KOA classification. A thorough analysis of the literature and the feature extraction process from the *VGG-19 and ResNet-101*models were used to achieve this.

- The second goal was to employ feature-level fusion to combine ***manually created features with network features***. Combining handmade methods like DWT, GLCM, and LBP with features taken from CNN models allowed for the successful completion of this task.

Research Constraints

- The study was hampered by a number of factors, including the **high computing demands of feature-level fusion** and ***deep learning model training***, which may be difficult for researchers with little funding.

Suggestion for Improvement and Future Work

- Future improvements will **focus on enhancing the fusion** of handcrafted and CNN features.

- The implementation and **optimization of FFNN** for KOA classification using these fused features will be addressed in future work.

What I had done ?

Activities	Done in PSM 1	Future Work
Phase 1 : Problem Formulation & Data Collection Literature Review	☑	
Activity 1 : Identification and justification of research problem	☑	
Activity 2 : Identification and justification of research goal, objective and scopes	☑	
Activity 3 : Data Collection	☑	
Activity 4 : Literature Review	☑	
Phase 2 : Data Pre-process then Generate Features and Fused Features		
Activity 5 : X-ray images pre-process	☑	
Activity 6 : Improve X-ray images (Network features)	☑	
Activity 7 : Generate handcrafted features (DWT , GLCM , LBP)	☑	
Activity 8 : Generate network features (VGG-19 and ResNet-101)	☑	
Activity 9 : Generate 2 fusion features (Network fuse with handcrafted features)	☑	
Phase 3 : Develop FFNN model		
Activity 10 : …		☑

THANK YOU

QnA

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

In the Name of God for Mankind

www.utm.my