FINAL YEAR PROJECT 2 : Demo Research Result

CLASSIFICATION OF KNEE OSTEOTHRITIS USING DEEP LEARNING MODELS FUSED WITH HANDCRAFTED FEATURES

PREPARED BY : LIM YONG WEI (B22EC0025)

SUPERVISOR: TS.DR.CHAN WENG HOWE

COORDINATOR: DR.ROZILAWATI BINTI DOLLAH @ MD.ZAIN

EXAMINER 1: DR. AZURAH BT. A.SAMAH

EXAMINER 2: DR. SHAMINI A/P RAJA KUMARAN

Demo Result OUTLINE

1 DataAutoSelection.py

- Sharpness Measurement to select X-ray images

2 DataPreparation.py

- Handcrafted Preprocess
- CNN Preprocess
- Split Dataset
- Augment Training Set

3 FeatureExtraction.py

- VGG19 & ResNet101
- DWT , GLCM and LBP

4 Fusion and FFNN Train&Evaluate.py

- FFNN Model Architecture
- Feature-level Fusion
- Performance Evaluate of Feed-Forward Neutral Network on 3 Models

5 Result Analysis & Comparative

- Confusion Matric on FFNN classify
- Comparison of FFNN Performance between 3 Models
- Discussion

Architecture Diagram

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

1 **Data AutoSelection** will be made before the "Preprocess" images

Step 1: Preprocess Data

2

Step 2: Split Dataset

Step 3: Augment Training Set Only

Step 4: Extract CNN Features on Preprocessed CNN Images

3

Step 5: Extract Handcrafted Features on Preprocessed Hand Images

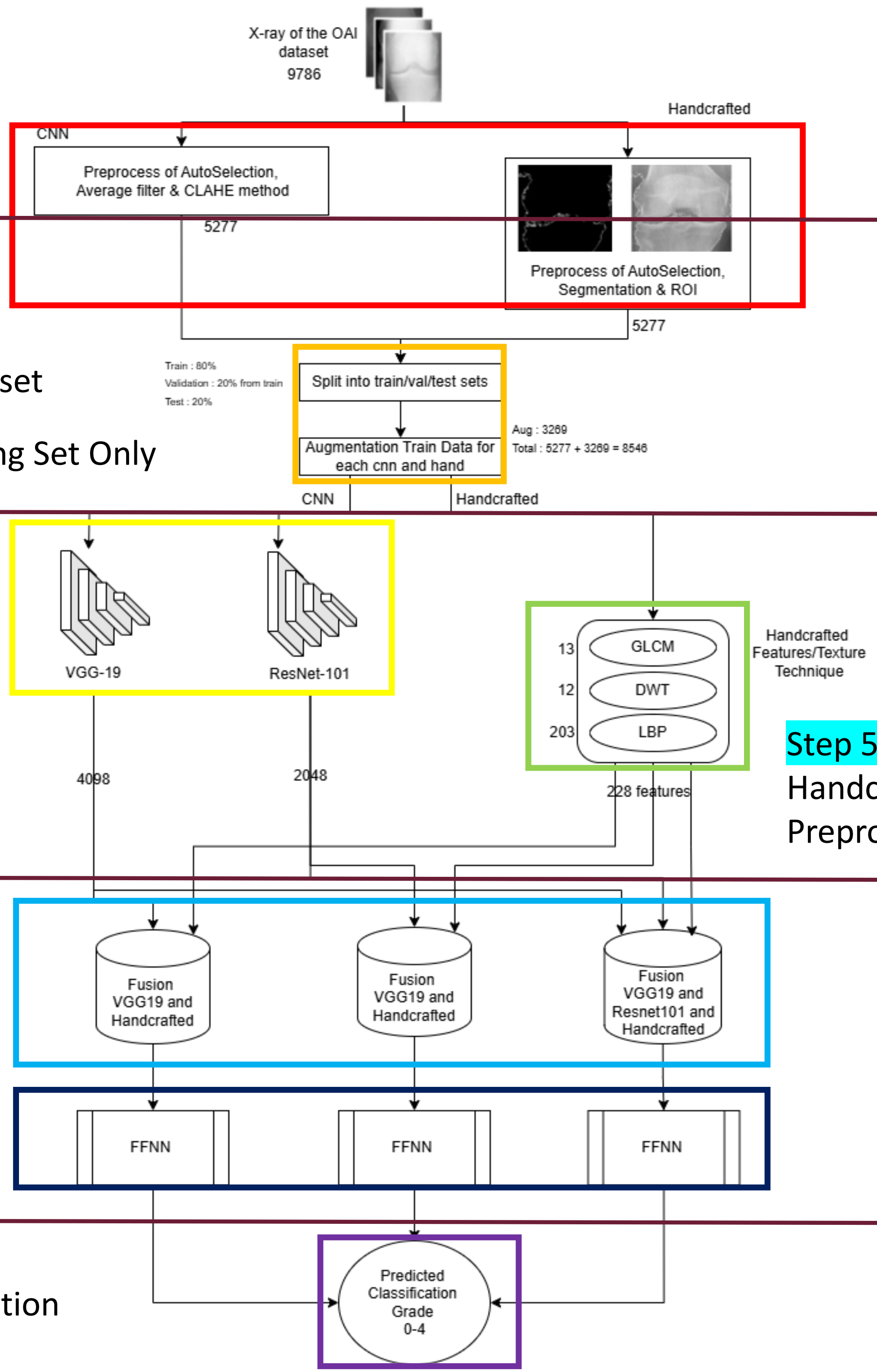
Step 6: Concatenate Features

4

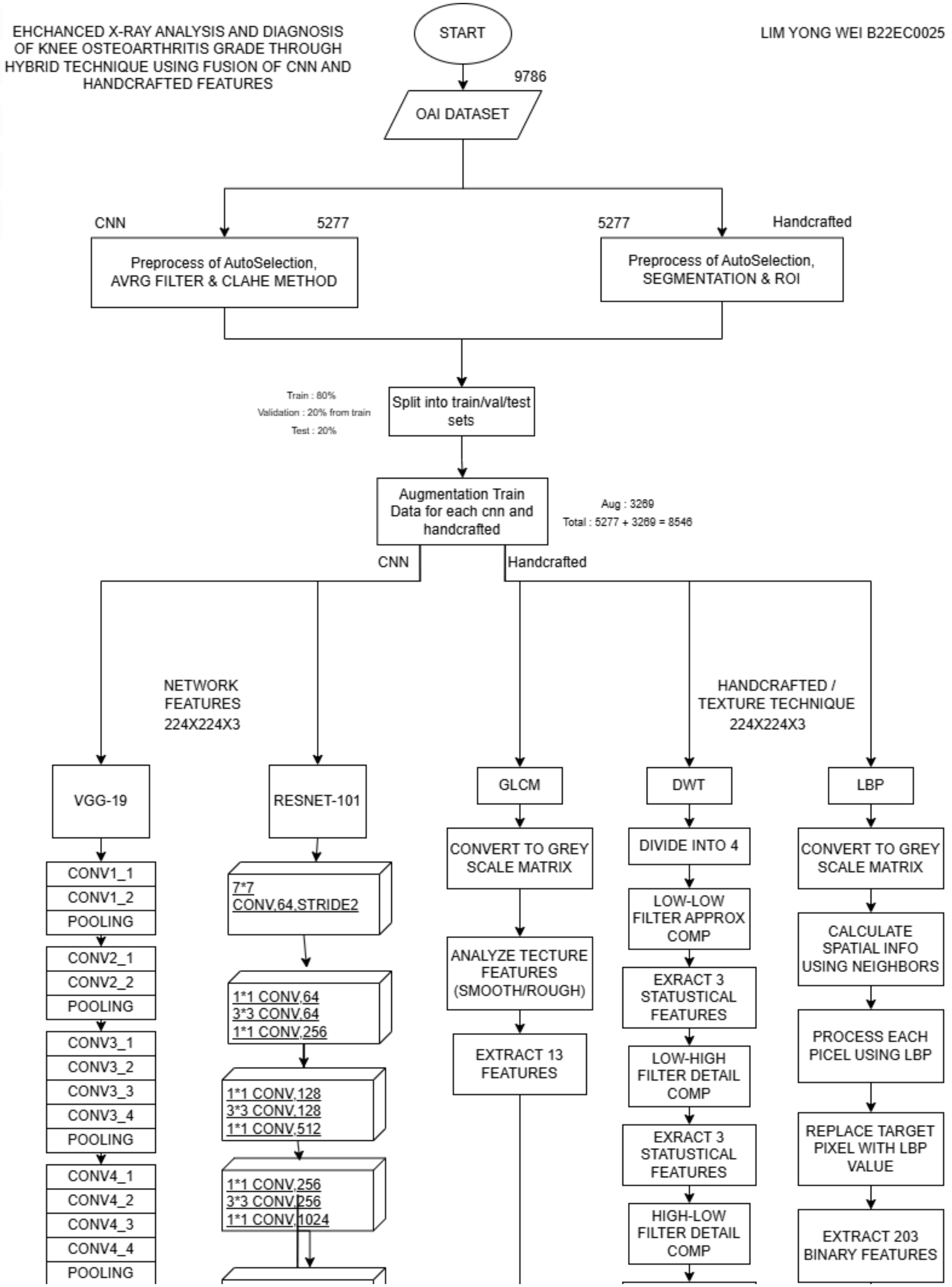
Step 7: FFNN Classification

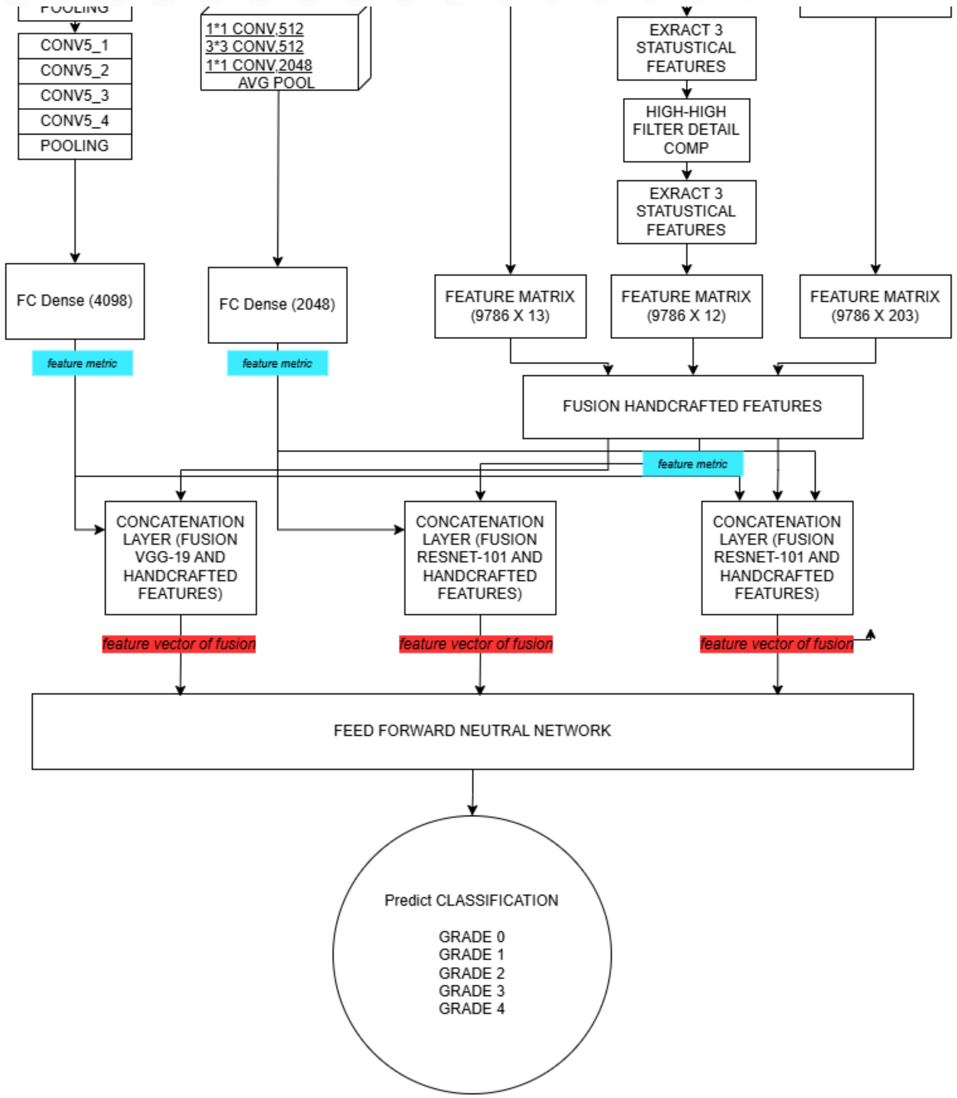
5

Step 8: Final Comparative Evaluation



Proposed Model Diagram

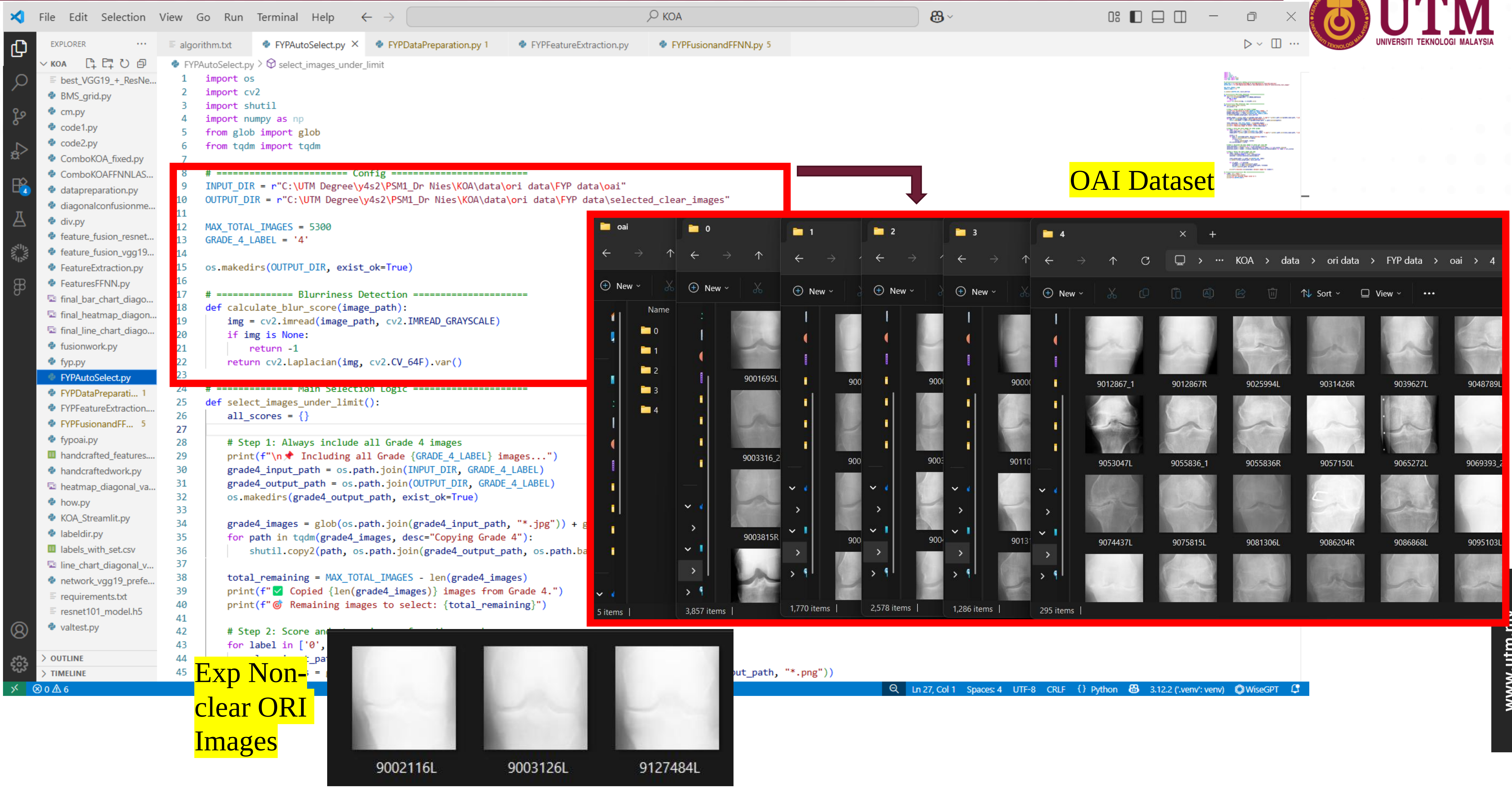




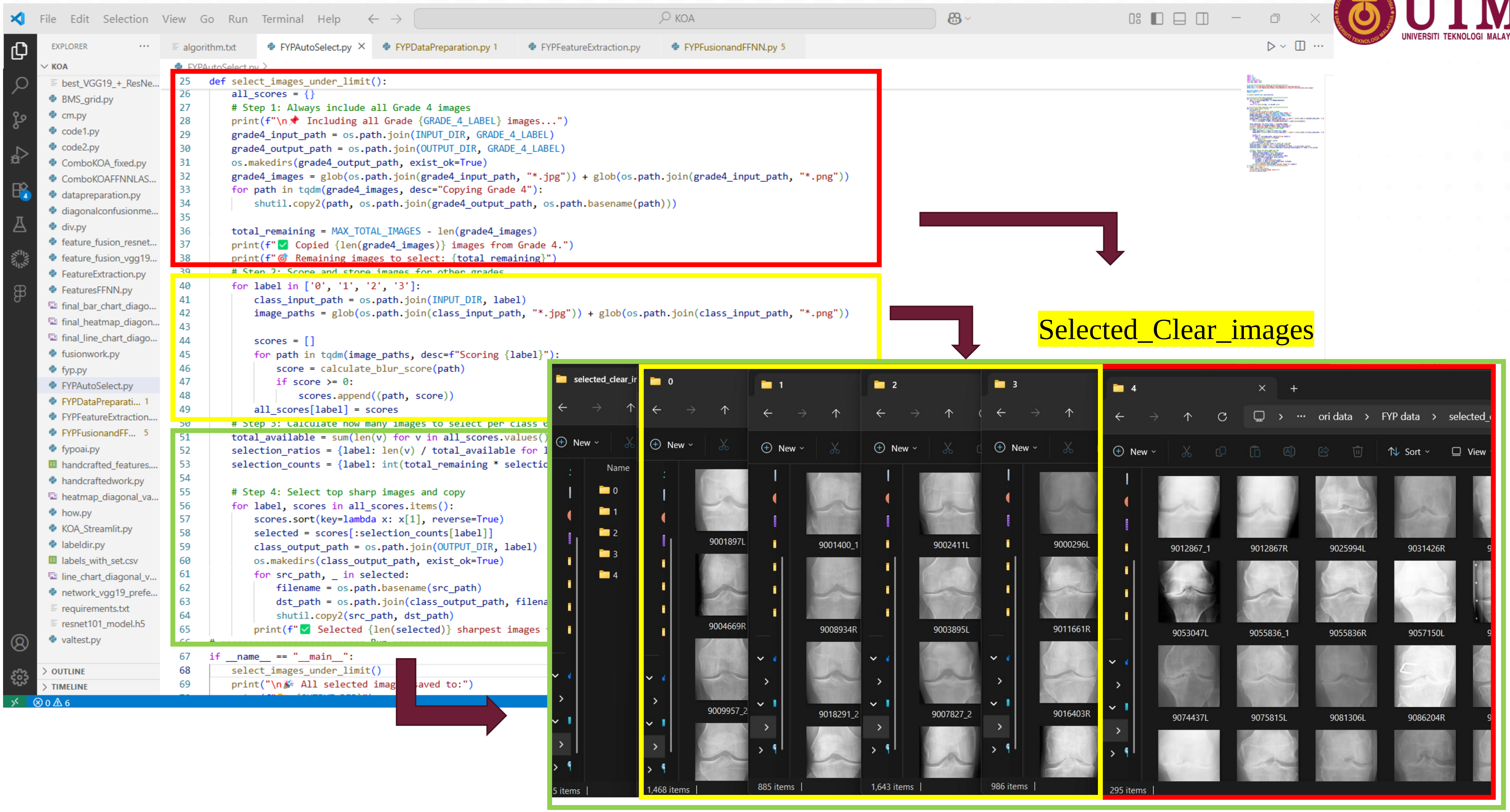
www.utm.my

1 **Data AutoSelection** will be made before the “Preprocess” images

Datasets AutoSelection – Sharpness Measurement to select X-ray Images



Datasets AutoSelection - Result

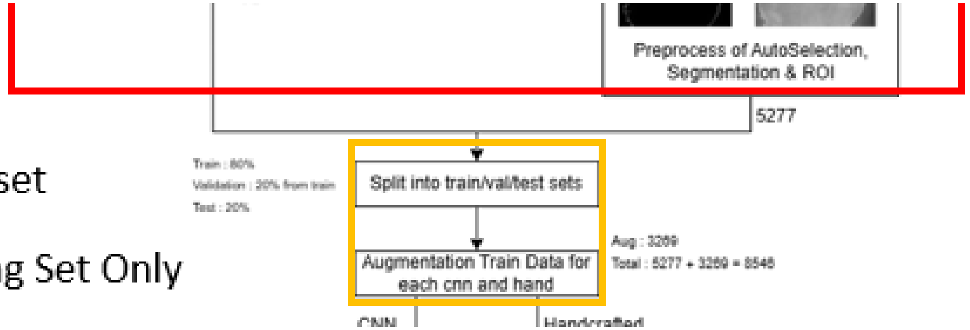


Step 1: Preprocess Data

2

Step 2: Split Dataset

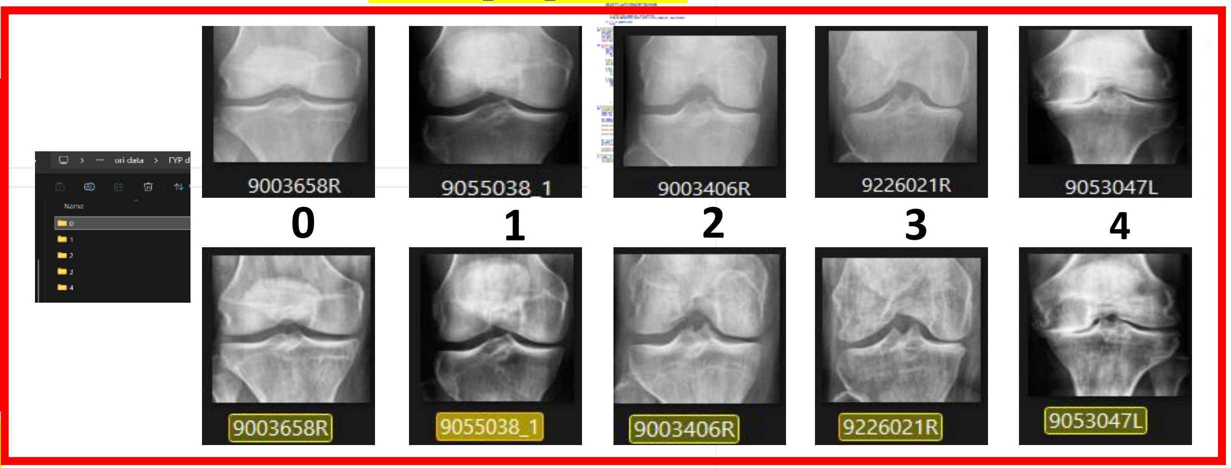
Step 3: Augment Training Set Only



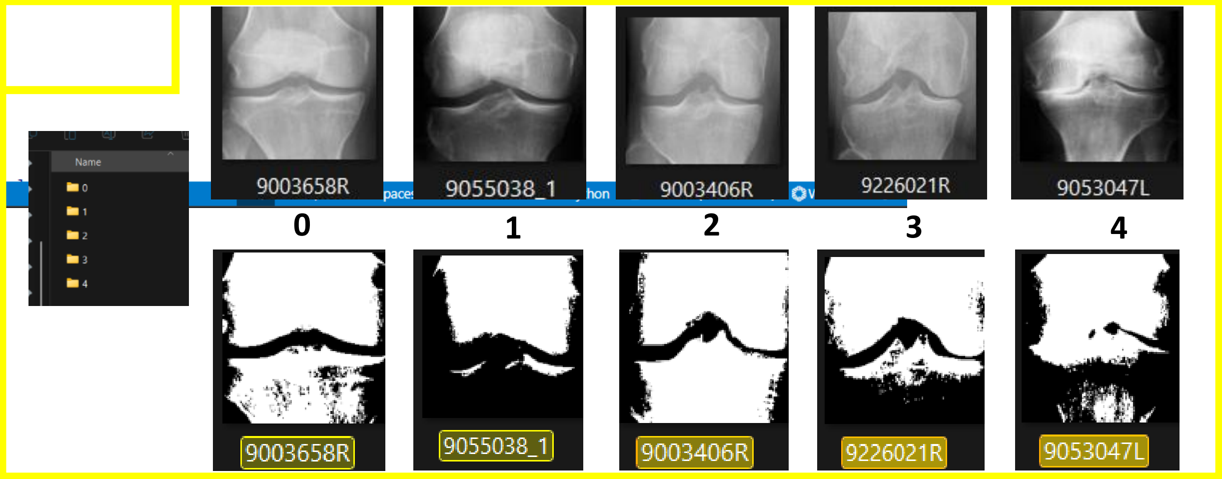
Data Preparation – Handcrafted and CNN Preprocess

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array

# ======================= Configuration =======================
INPUT_DIR = r"C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\selected_clear_images"
IMAGE_SIZE = (224, 224)
TRAIN_RATIO = 0.8
VAL_RATIO = 0.2
SEED = 42
# Output paths
handcrafted_output_base = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\hand\augSplitAfter'
cnn_output_base = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\cnn\augSplitAfter'

# =================== Preprocessing Functions ===================
# CNN preprocessing: CLAHE + Avg Filter
def preprocess_cnn(img_path):
    img = cv2.imread(img_path)
    if img is None:
        return None
    h, w = img.shape[:2]
    crop = img[int(h*0.1):int(h*0.9), int(w*0.1):int(w*0.9)]
    gray = cv2.cvtColor(crop, cv2.COLOR_BGR2GRAY)
    gray_resized = cv2.resize(gray, IMAGE_SIZE)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    clahe_img = clahe.apply(gray_resized)
    avg_filtered = cv2.blur(clahe_img, (3, 3))
    final_img = cv2.cvtColor(avg_filtered, cv2.COLOR_GRAY2BGR)
    return final_img

# Handcrafted preprocessing: ROI + basic threshold segmentation
def preprocess_handcrafted(img_path):
    img = cv2.imread(img_path)
    if img is None:
        return None
    h, w = img.shape[:2]
    crop = img[int(h*0.1):int(h*0.9), int(w*0.1):int(w*0.9)]
    gray = cv2.cvtColor(crop, cv2.COLOR_BGR2GRAY)
    gray_resized = cv2.resize(gray, IMAGE_SIZE)
    _, segmented = cv2.threshold(gray_resized, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    final_img = cv2.cvtColor(segmented, cv2.COLOR_GRAY2BGR)
    return final_img

# =================== Data Split Functions ===================
def load_images_by_class():
    image_info = []
    for label in ['0', '1', '2', '3', '4']:
```

CNN preprocess



Handcrafted preprocess



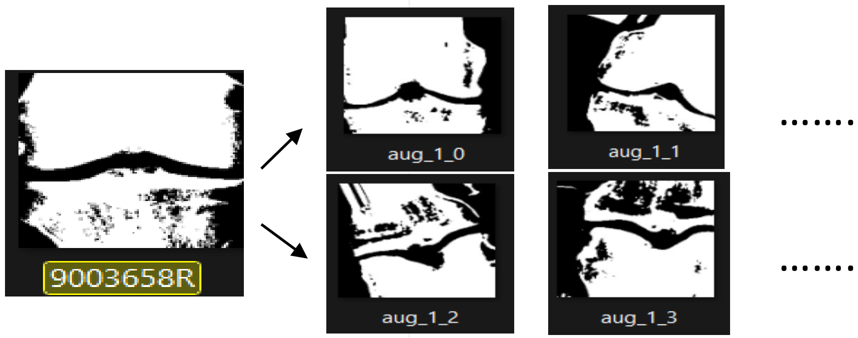
Data Preparation – Splitting and Augmentation

```python
IMAGE_SIZE = (224, 224)
TRAIN_RATIO = 0.8
VAL_RATIO = 0.2
SEED = 42
# Output paths
handcrafted_output_base = r'C:\UTM Degree\y4s2\PSM1_
cnn_output_base = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\
```

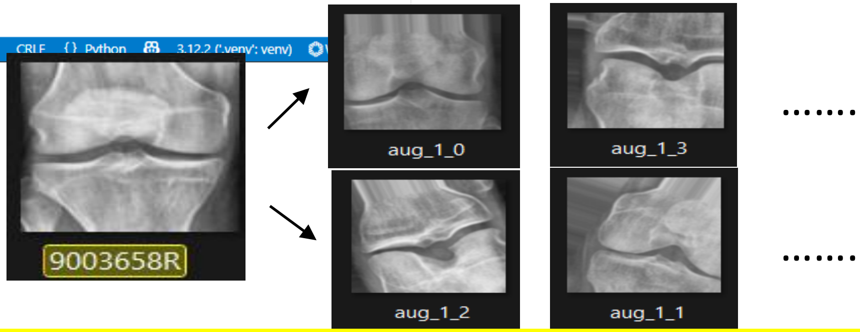
```python
def load_images_by_class():
    image_info = []
    for label in ['0', '1', '2', '3', '4']:
        paths = glob(os.path.join(INPUT_DIR, label, "*.jpg")) + glob(os.path.join(INPUT_DIR, label, "*.png"))
        for p in paths:
            image_info.append((p, label))
    return image_info

def split_data(image_info):
    train_val, test = train_test_split(image_info, test_size=(1 - TRAIN_RATIO),
                                       stratify=[lbl for _, lbl in image_info], random_state=SEED)
    train, val = train_test_split(train_val, test_size=VAL_RATIO,
                                  stratify=[lbl for _, lbl in train_val], random_state=SEED)
    return train, val, test

# =================== Augmentation ===================
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)

def augment_and_save(img_array, label, img_index, output_dir, csv_records, augment_count=2, all_images_dir=None):
    img_array = img_array.reshape((1,) + img_array.shape)
    for i, batch in enumerate(datagen.flow(img_array, batch_size=1)):
        aug_filename = f"aug_{label}_{img_index}_{i}.jpg"
        aug_path = os.path.join(output_dir, aug_filename)
        array_to_img(batch[0]).save(aug_path)
        csv_records.append((aug_filename, label, "train"))

        if all_images_dir:
            os.makedirs(all_images_dir, exist_ok=True)
            array_to_img(batch[0]).save(os.path.join(all_images_dir, aug_filename))

        if i + 1 >= augment_count:
            break

# =================== Image Saving ===================
def save_image(img, filename, base_output_dir, set_name, label):
```

Phase/Classes	Training 80% (Validation 20%)	Testing (20%)
Grade 0	940 (234)	294
Grade 1	567 (141)	177
Grade 2	1052 (263)	328
Grade 3	630 (159)	197
Grade 4	189 (47)	59
Total	**3378 (844)**	**1055**

Augmentation – Handcrafted Preprocessed Images

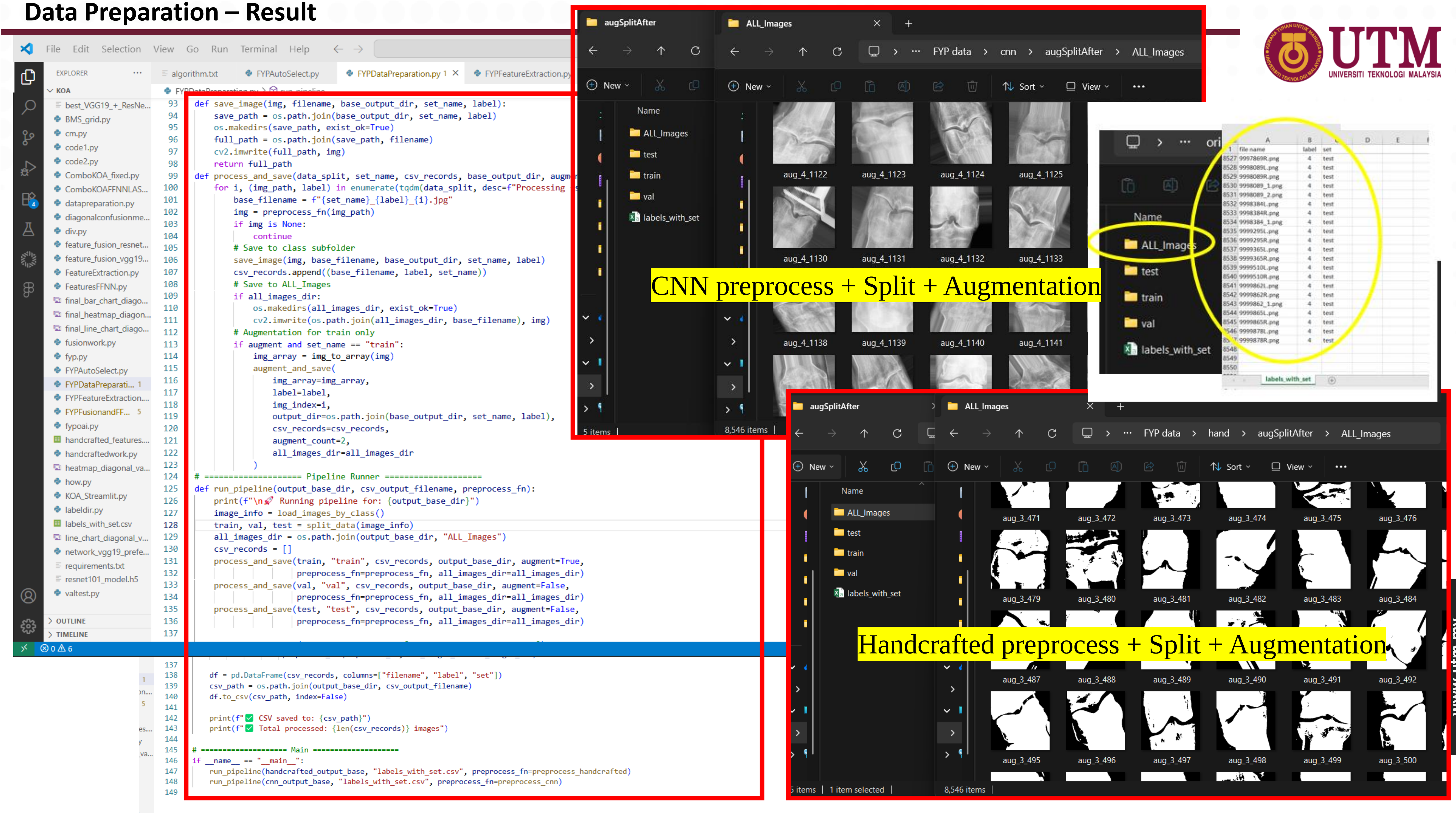


Augmentation – CNN Preprocessed Images



Data Preparation – Result

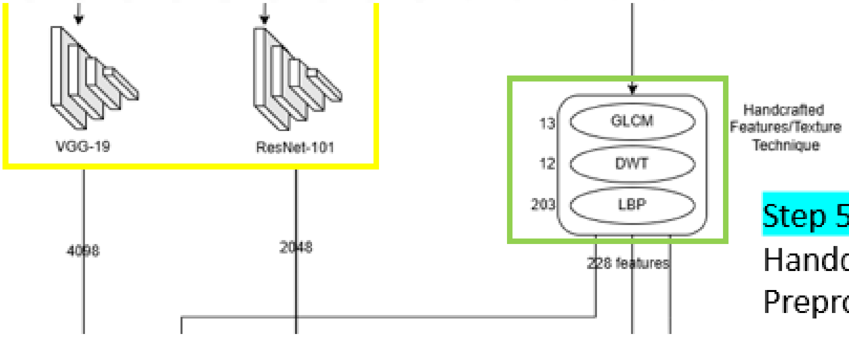
UNIVERSITI TEKNOLOGI MALAYSIA



www.utm.my

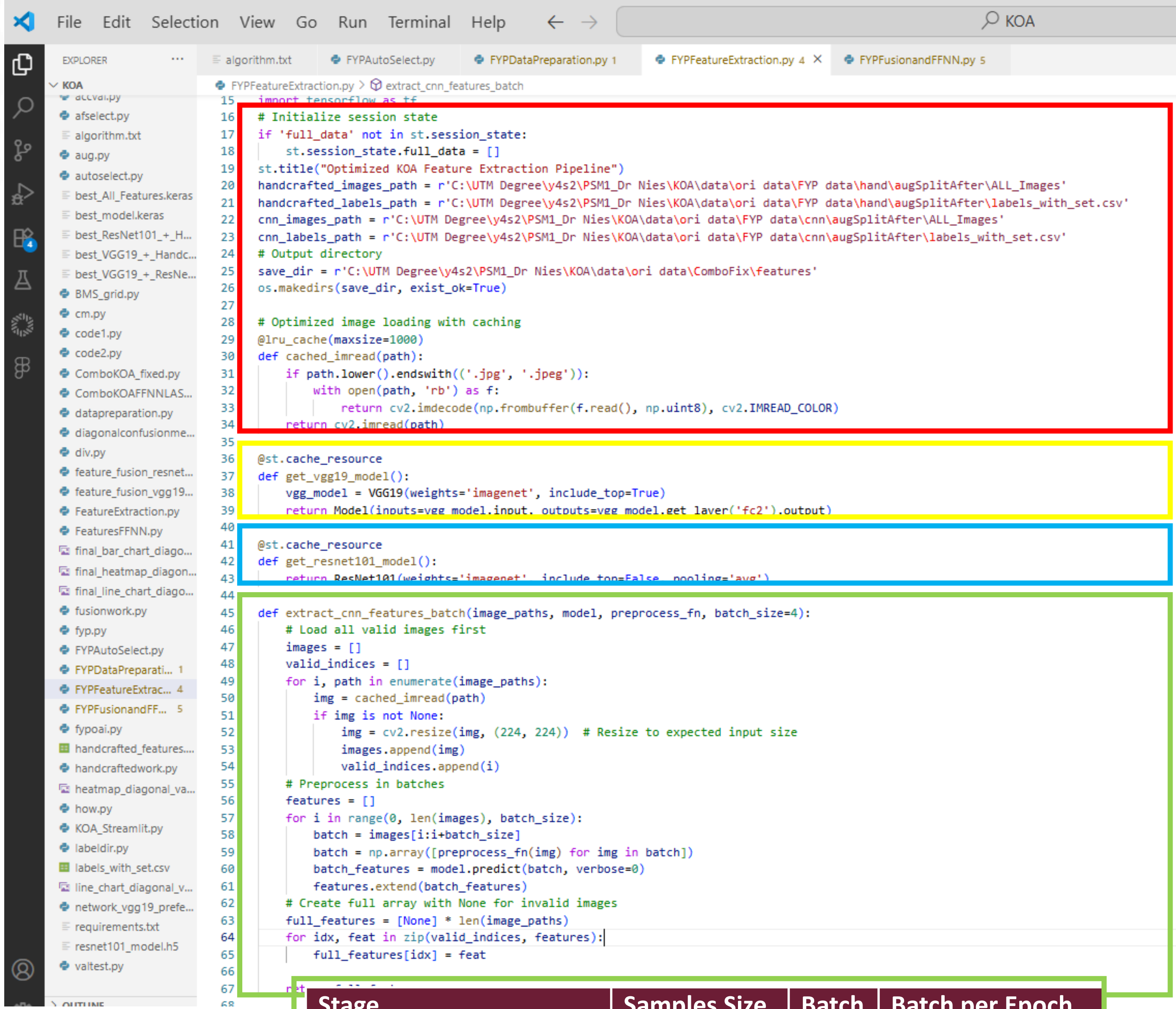
3

Step 4: Extract CNN Features on Preprocessed CNN Images



Step 5: Extract Handcrafted Features on Preprocessed Hand Images

Feature Extraction – VGG19 and Resnet101



```python
import tensorflow as tf
# Initialize session state
if 'full_data' not in st.session_state:
    st.session_state.full_data = []
st.title("Optimized KOA Feature Extraction Pipeline")
handcrafted_images_path = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\hand\augSplitAfter\ALL_Images'
handcrafted_labels_path = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\hand\augSplitAfter\labels_with_set.csv'
cnn_images_path = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\cnn\augSplitAfter\ALL_Images'
cnn_labels_path = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\FYP data\cnn\augSplitAfter\labels_with_set.csv'
# Output directory
save_dir = r'C:\UTM Degree\y4s2\PSM1_Dr Nies\KOA\data\ori data\ComboFix\features'
os.makedirs(save_dir, exist_ok=True)

# Optimized image loading with caching
@lru_cache(maxsize=1000)
def cached_imread(path):
    if path.lower().endswith(('.jpg', '.jpeg')):
        with open(path, 'rb') as f:
            return cv2.imdecode(np.frombuffer(f.read(), np.uint8), cv2.IMREAD_COLOR)
    return cv2.imread(path)

@st.cache_resource
def get_vgg19_model():
    vgg_model = VGG19(weights='imagenet', include_top=True)
    return Model(inputs=vgg_model.input, outputs=vgg_model.get_layer('fc2').output)

@st.cache_resource
def get_resnet101_model():
    return ResNet101(weights='imagenet', include_top=False, pooling='avg')

def extract_cnn_features_batch(image_paths, model, preprocess_fn, batch_size=4):
    # Load all valid images first
    images = []
    valid_indices = []
    for i, path in enumerate(image_paths):
        img = cached_imread(path)
        if img is not None:
            img = cv2.resize(img, (224, 224))  # Resize to expected input size
            images.append(img)
            valid_indices.append(i)
    # Preprocess in batches
    features = []
    for i in range(0, len(images), batch_size):
        batch = images[i:i+batch_size]
        batch = np.array([preprocess_fn(img) for img in batch])
        batch_features = model.predict(batch, verbose=0)
        features.extend(batch_features)
    # Create full array with None for invalid images
    full_features = [None] * len(image_paths)
    for idx, feat in zip(valid_indices, features):
        full_features[idx] = feat
```

Stage	Samples Size	Batch	Batch per Epoch
Network Features Extraction	6277	4	6277/4 = **1662**

Resnet101 Model Summary

Model Summary

Model: "functional_1"

Layer (type)	Output Shape	Param #
image (InputLayer)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 112, 112, 64)	9,472
max_pooling2d (MaxPooling2D)	(None, 56, 56, 64)	0
batch_normalization (BatchNormalization)	(None, 56, 56, 64)	256
conv2d_1 (Conv2D)	(None, 56, 56, 64)	4,160
conv2d_2 (Conv2D)	(None, 56, 56, 64)	36,928
conv2d_3 (Conv2D)	(None, 56, 56, 256)	16,640
batch_normalization_1 (BatchNormalization)	(None, 56, 56, 256)	1,024
conv2d_4 (Conv2D)	(None, 28, 28, 128)	32,896
conv2d_5 (Conv2D)	(None, 28, 28, 128)	147,584
conv2d_6 (Conv2D)	(None, 28, 28, 512)	66,048
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 512)	2,048
conv2d_7 (Conv2D)	(None, 14, 14, 256)	131,328
conv2d_8 (Conv2D)	(None, 14, 14, 256)	590,080
conv2d_9 (Conv2D)	(None, 14, 14, 1024)	263,168
batch_normalization_3 (BatchNormalization)	(None, 14, 14, 1024)	4,096
conv2d_10 (Conv2D)	(None, 7, 7, 512)	524,800
conv2d_11 (Conv2D)	(None, 7, 7, 512)	2,359,808
conv2d_12 (Conv2D)	(None, 7, 7, 2048)	1,050,624
batch_normalization_4 (BatchNormalization)	(None, 7, 7, 2048)	8,192
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0

Total params: 5,249,152 (20.02 MB)
Trainable params: 5,241,344 (19.99 MB)
Non-trainable params: 7,808 (30.50 KB)

Vgg19 Model Summary

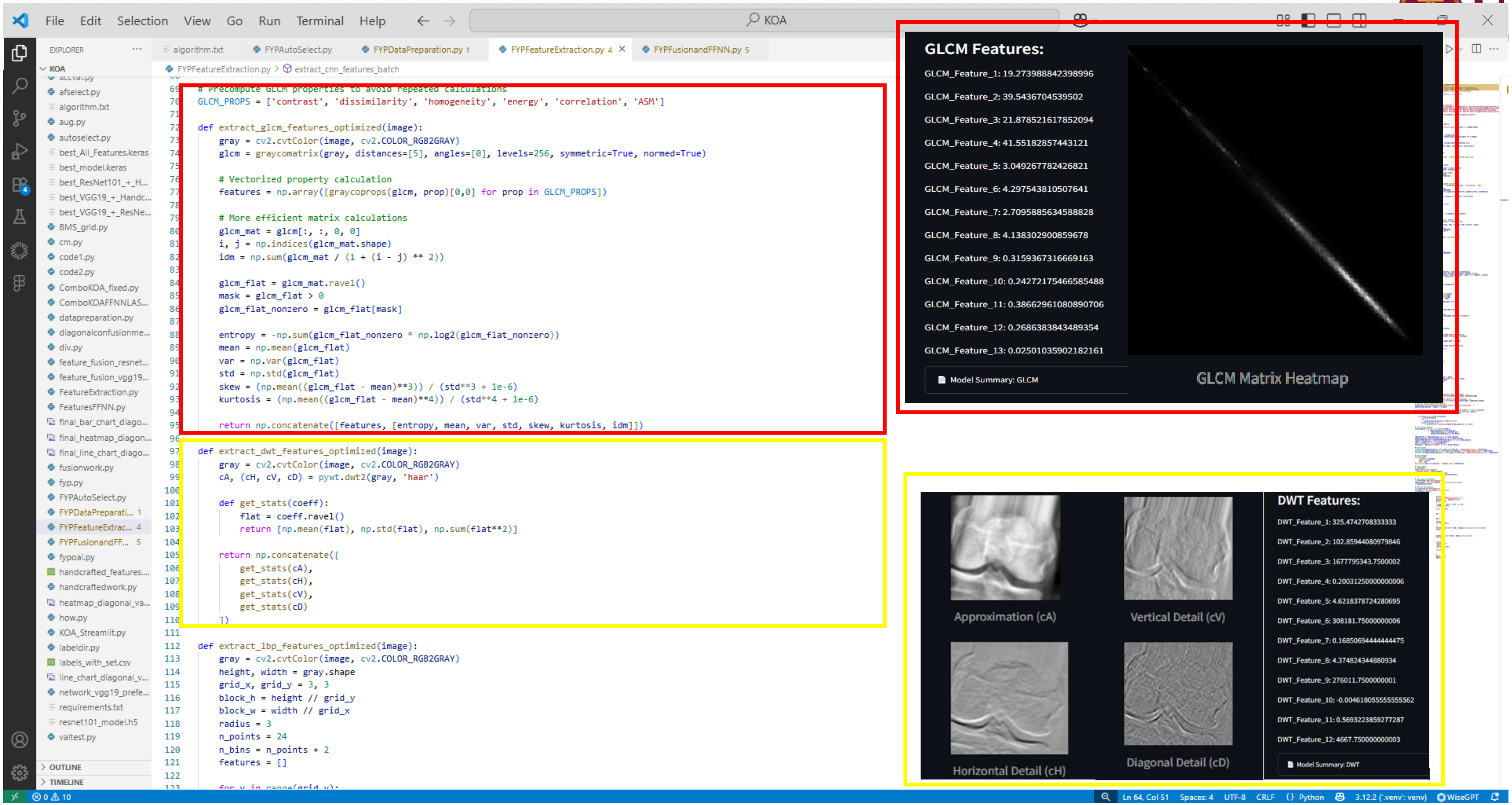
Model Summary

Model: "functional_1"

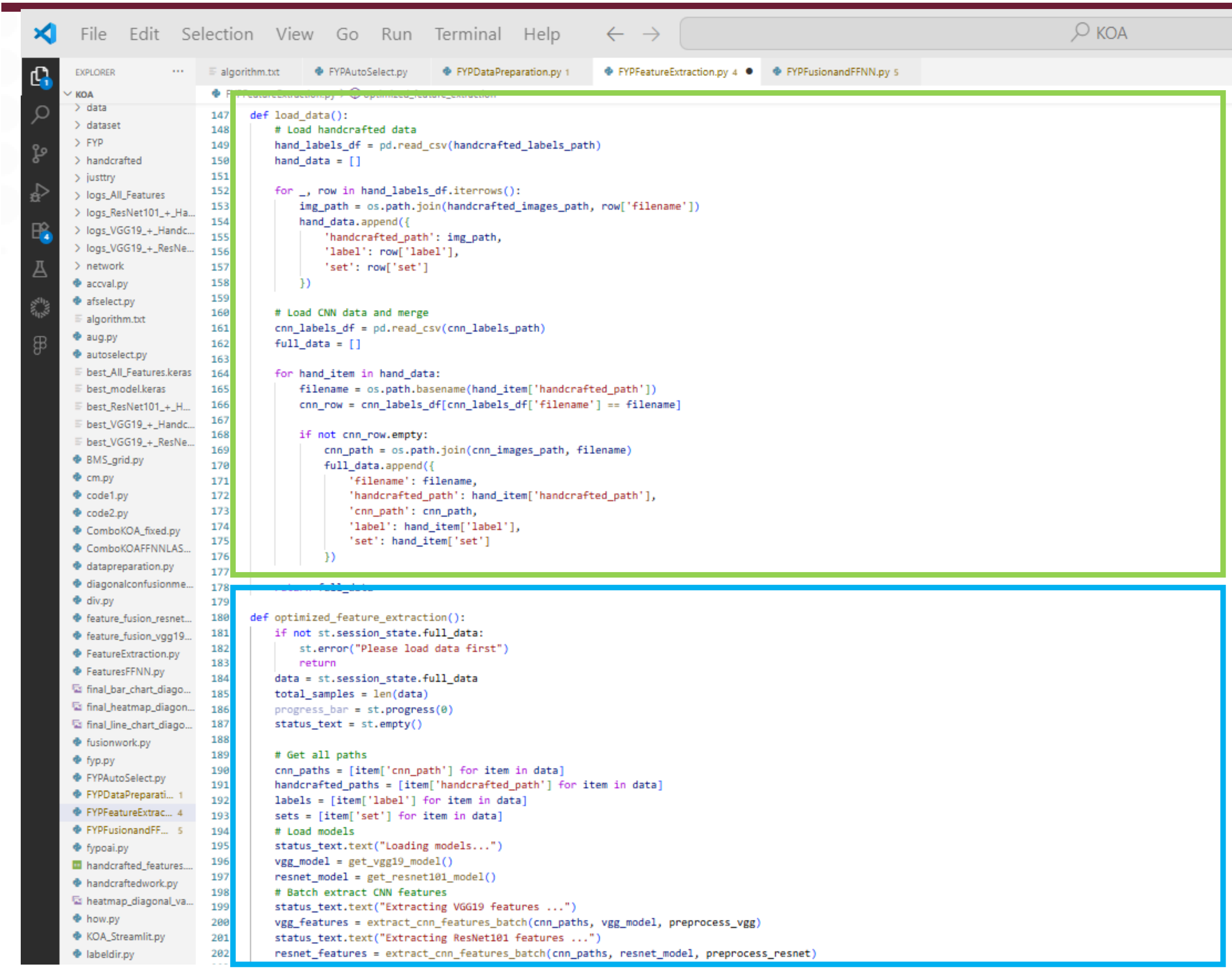
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 224, 224, 64)	1,792
re_lu (ReLU)	(None, 224, 224, 64)	0
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36,928
re_lu_1 (ReLU)	(None, 224, 224, 64)	0
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73,856
re_lu_2 (ReLU)	(None, 112, 112, 128)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147,584
re_lu_3 (ReLU)	(None, 112, 112, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295,168
re_lu_4 (ReLU)	(None, 56, 56, 256)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590,080
re_lu_5 (ReLU)	(None, 56, 56, 256)	0
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590,080
re_lu_6 (ReLU)	(None, 56, 56, 256)	0
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590,080
re_lu_7 (ReLU)	(None, 56, 56, 256)	0
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1,180,160
re_lu_8 (ReLU)	(None, 28, 28, 512)	0
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2,359,808
re_lu_9 (ReLU)	(None, 28, 28, 512)	0
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2,359,808
re_lu_10 (ReLU)	(None, 28, 28, 512)	0
conv2d_11 (Conv2D)	(None, 28, 28, 512)	2,359,808
re_lu_11 (ReLU)	(None, 28, 28, 512)	0
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2,359,808
re_lu_12 (ReLU)	(None, 14, 14, 512)	0
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2,359,808
re_lu_13 (ReLU)	(None, 14, 14, 512)	0
conv2d_14 (Conv2D)	(None, 14, 14, 512)	2,359,808
re_lu_14 (ReLU)	(None, 14, 14, 512)	0
conv2d_15 (Conv2D)	(None, 14, 14, 512)	2,359,808
re_lu_15 (ReLU)	(None, 14, 14, 512)	0
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102,764,544
re_lu_16 (ReLU)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16,781,312
re_lu_17 (ReLU)	(None, 4096)	0
feature_output (Dense)	(None, 2048)	8,390,656

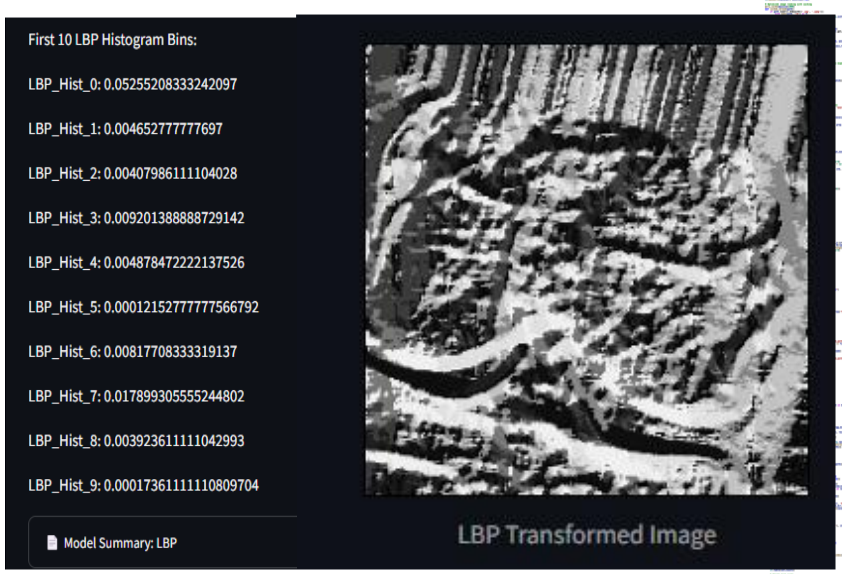
Total params: 147,960,896 (564.43 MB)
Trainable params: 147,960,896 (564.43 MB)
Non-trainable params: 0 (0.00 B)

Feature Extraction – Handcrafted Features (GLCM and DWT)



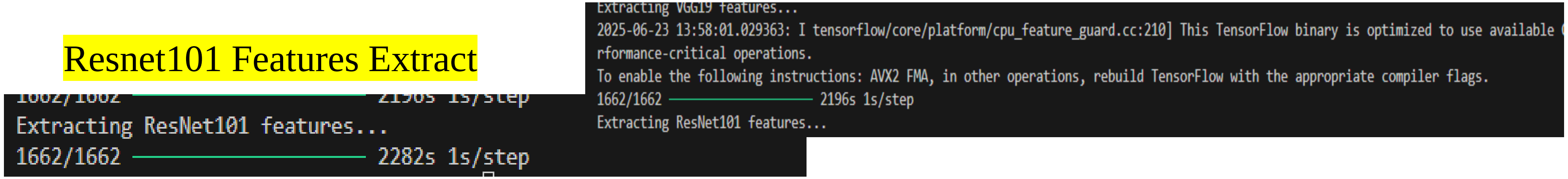
Feature Extraction – Handcrafted Features (LBP)



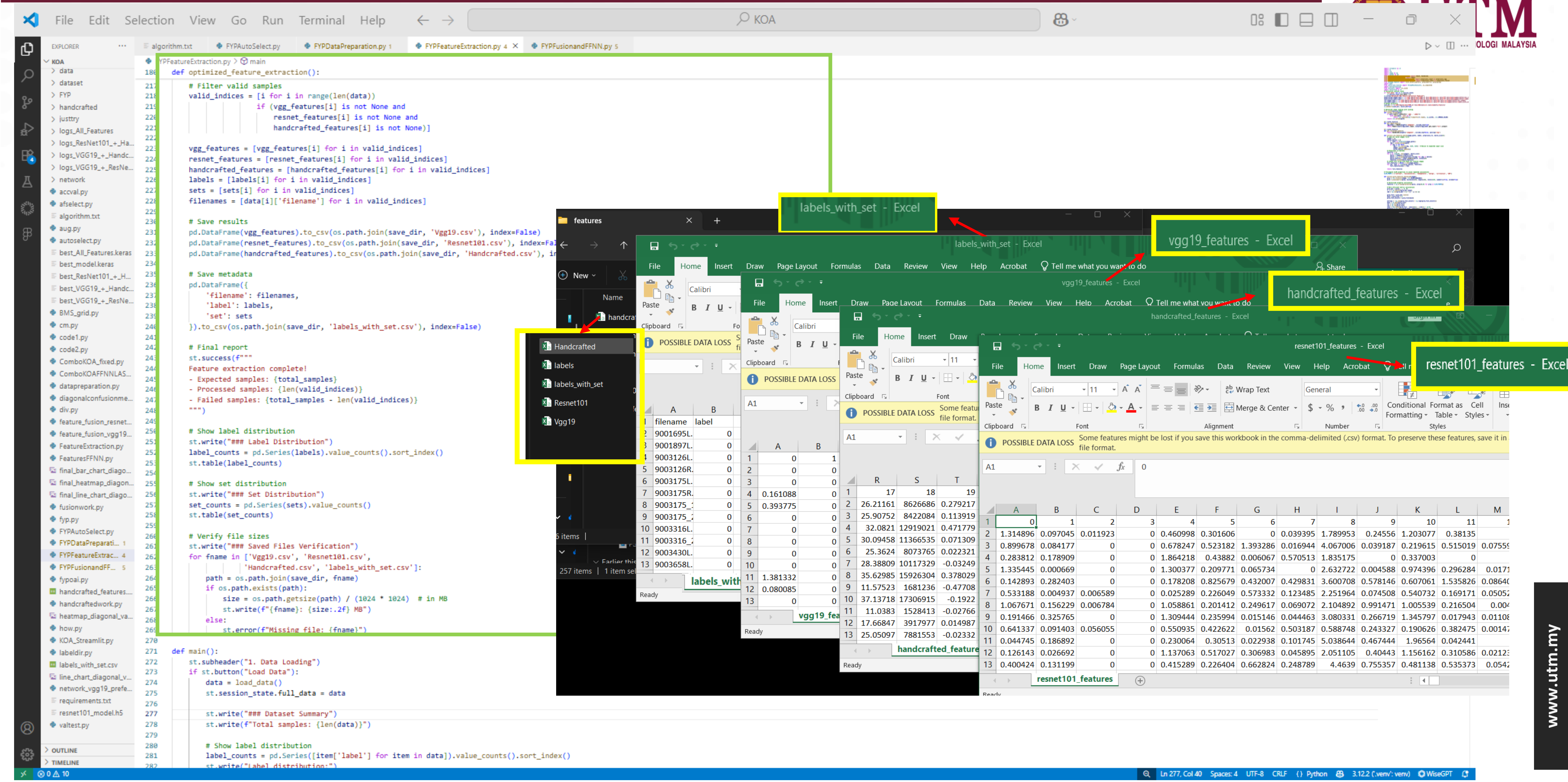


Vgg19 Features Extract

Resnet101 Features Extract



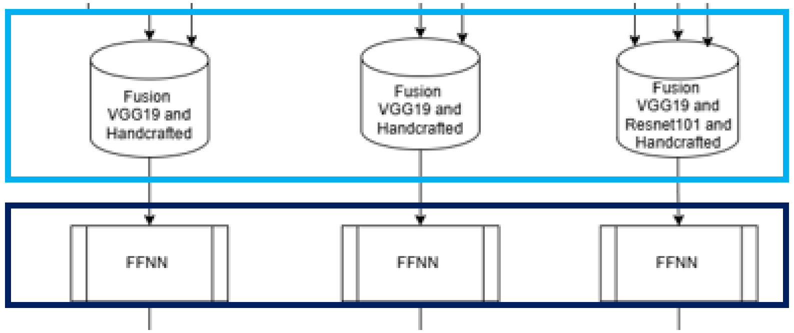
Feature Extraction – Result



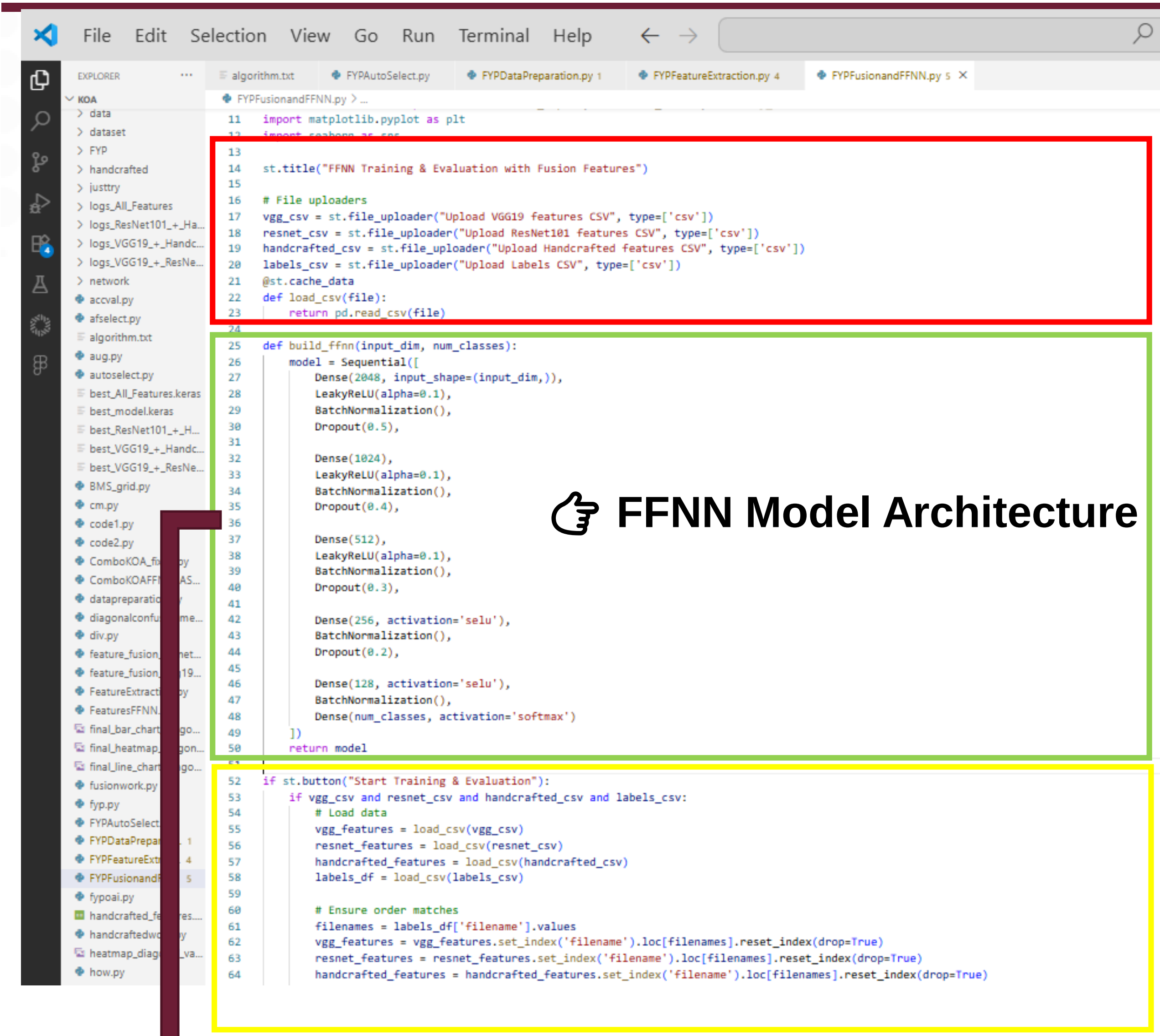
4

Step 6: Concatenate Features

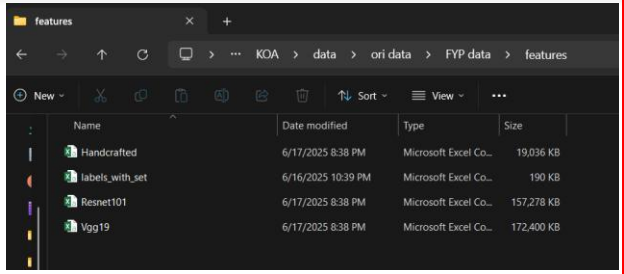
Step 7: FFNN Classification

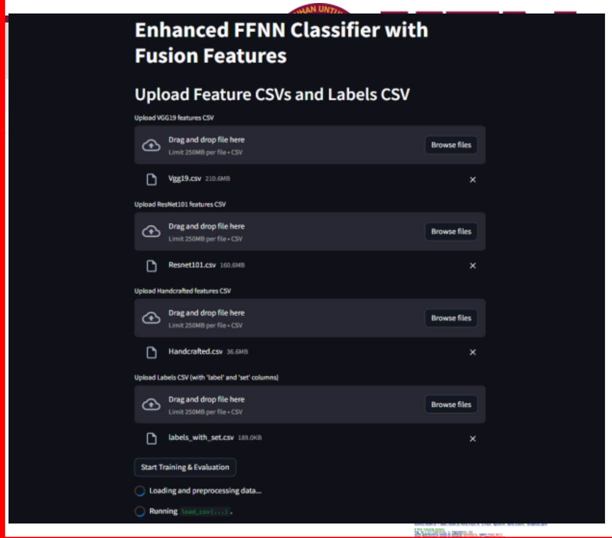


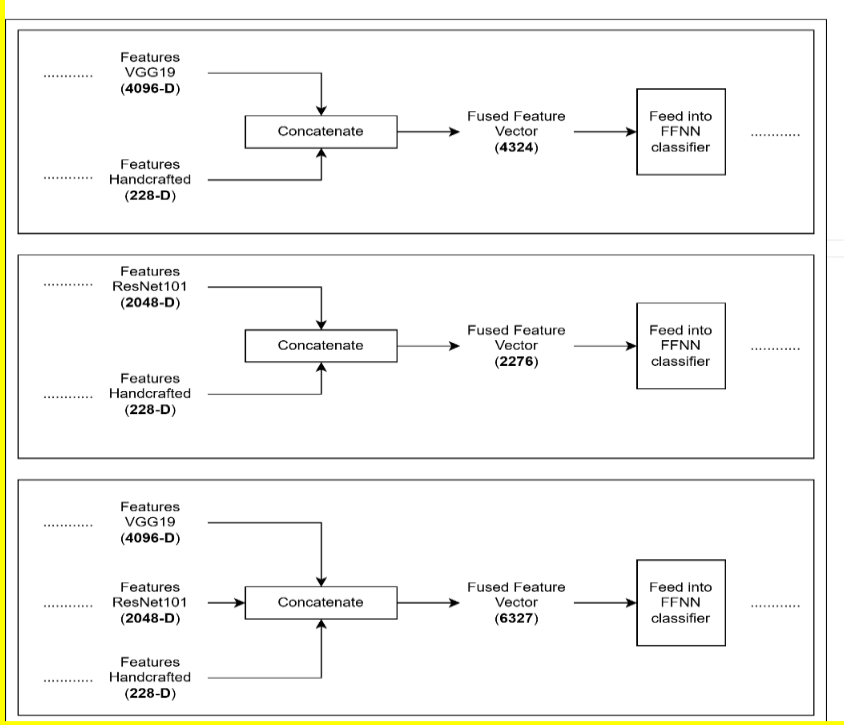
Fusion and FFNN – FFNN Model Architecture



FFNN Model Architecture

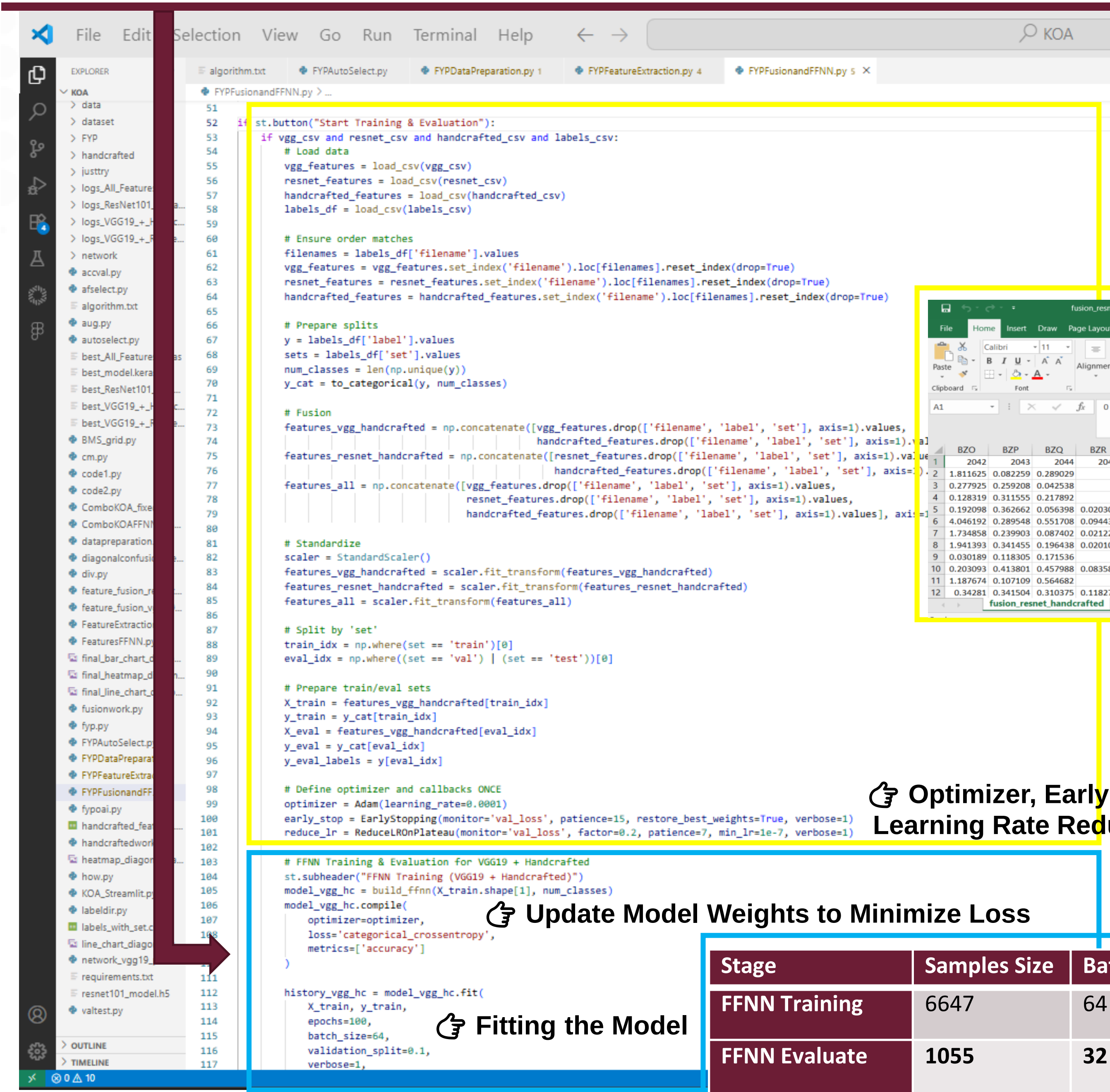


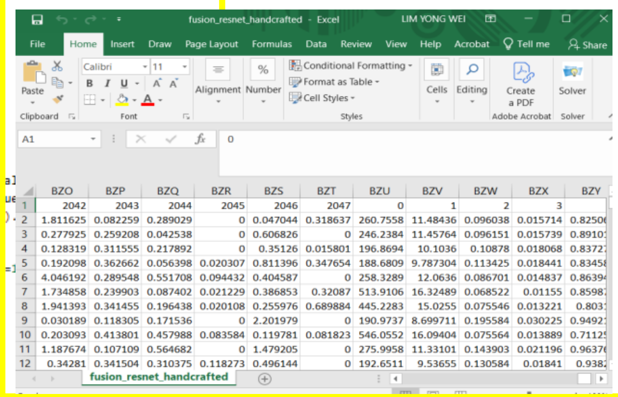




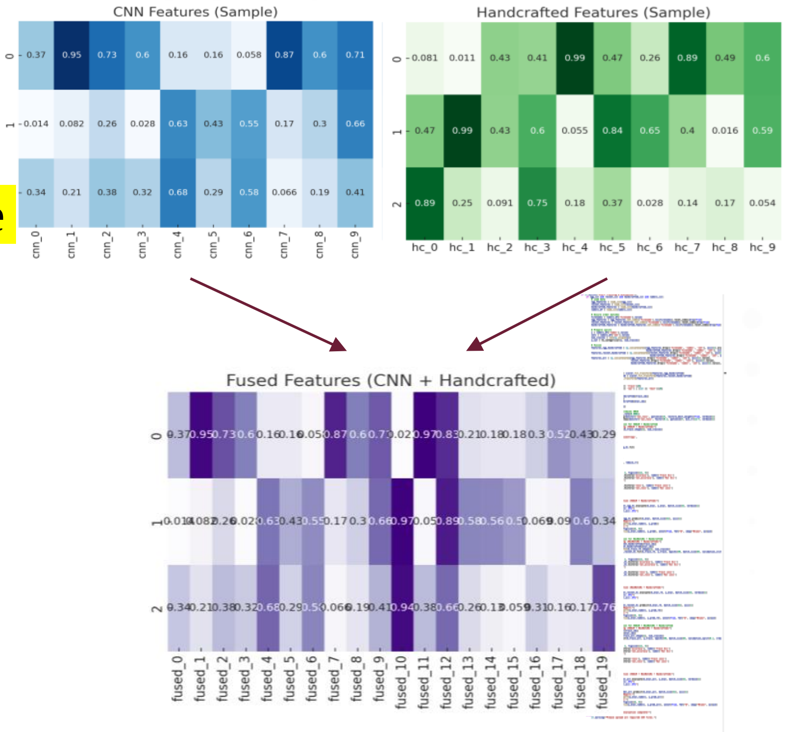
www.utm.my

Fusion and FFNN – Feature Level Fusion





Exp How concatenate work



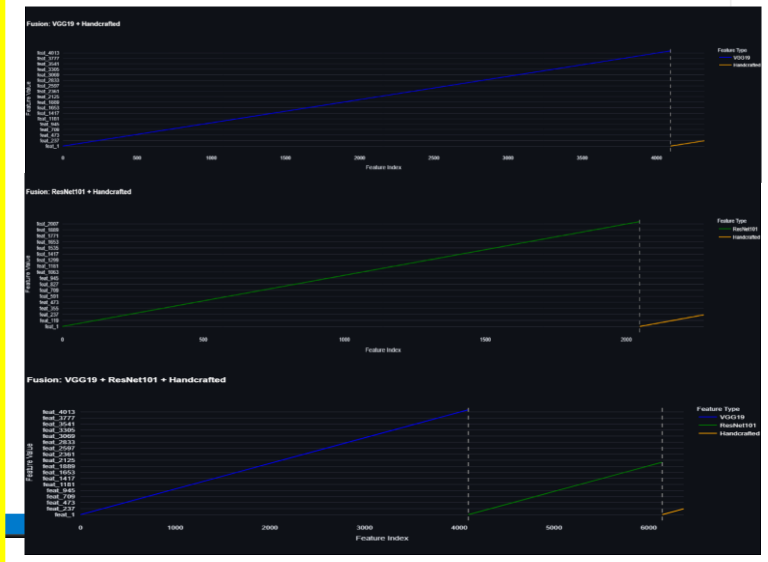
☞ Optimizer, Early Stopping, and Learning Rate Reduction

☞ Update Model Weights to Minimize Loss

☞ Fitting the Model

Stage	Samples Size	Batch	Batch per Epoch
FFNN Training	6647	64	6647/64 = 104
FFNN Evaluate	1055	32	1055/32= 33

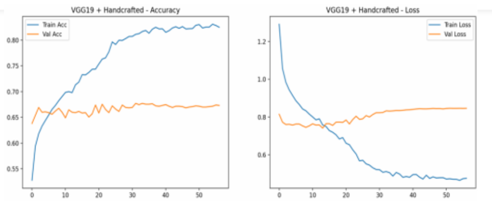
Feature Index Map (Single Sample)



Fusion and FFNN – Performance Evaluate of FFNN on 3 Models

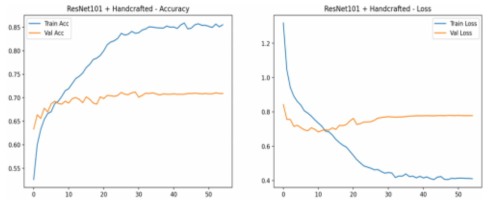
```python
    callbacks=[early_stop, reduce_lr]
)

# Plot training history
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
ax[0].plot(history_vgg_hc.history['accuracy'], label='Train Acc')
ax[0].plot(history_vgg_hc.history['val_accuracy'], label='Val Acc')
ax[0].set_title('Accuracy')
ax[0].legend()
ax[1].plot(history_vgg_hc.history['loss'], label='Train Loss')
ax[1].plot(history_vgg_hc.history['val_loss'], label='Val Loss')
ax[1].set_title('Loss')
ax[1].legend()
st.pyplot(fig)

st.subheader("FFNN Evaluation (VGG19 + Handcrafted)")
# FFNN EVALUATION
eval_loss, eval_acc = model_vgg_hc.evaluate(X_eval, y_eval, batch_size=32, verbose=1)
st.write(f"Loss: {eval_loss:.4f}")
st.write(f"Accuracy: {eval_acc:.4f}")

# Predictions and metrics
y_pred = np.argmax(model_vgg_hc.predict(X_eval, batch_size=32), axis=1)
st.write("Classification Report:")
st.text(classification_report(y_eval_labels, y_pred))
st.write("Confusion Matrix:")
fig2, ax2 = plt.subplots(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_eval_labels, y_pred), annot=True, fmt="d", cmap="Blues", ax=ax2)
st.pyplot(fig2)

# FFNN Training & Evaluation for ResNet101 + Handcrafted
st.subheader("FFNN Training (ResNet101 + Handcrafted)")
X_train_rh = features_resnet_handcrafted[train_idx]
X_eval_rh = features_resnet_handcrafted[eval_idx]
model_resnet_hc = build_ffnn(X_train_rh.shape[1], num_classes)
history_resnet_hc = model_resnet_hc.fit(X_train_rh, y_train, epochs=50, batch_size=64, validation_split=0.1, verbose=1)

# Plot training history
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
ax[0].plot(history_resnet_hc.history['accuracy'], label='Train Acc')
ax[0].plot(history_resnet_hc.history['val_accuracy'], label='Val Acc')
ax[0].set_title('Accuracy')
ax[0].legend()
ax[1].plot(history_resnet_hc.history['loss'], label='Train Loss')
ax[1].plot(history_resnet_hc.history['val_loss'], label='Val Loss')
ax[1].set_title('Loss')
ax[1].legend()
st.pyplot(fig)

st.subheader("FFNN Evaluation (ResNet101 + Handcrafted)")
# FFNN EVALUATION
eval_loss, eval_acc = model_resnet_hc.evaluate(X_eval_rh, y_eval, batch_size=32, verbose=1)
st.write(f"Loss: {eval_loss:.4f}")
st.write(f"Accuracy: {eval_acc:.4f}")

# Predictions and metrics
y_pred_rh = np.argmax(model_resnet_hc.predict(X_eval_rh, batch_size=32), axis=1)
st.write("Classification Report:")
st.text(classification_report(y_eval_labels, y_pred_rh))
st.write("Confusion Matrix:")
fig2, ax2 = plt.subplots(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_eval_labels, y_pred_rh), annot=True, fmt="d", cmap="Blues", ax=ax2)
st.pyplot(fig2)

# FFNN Training & Evaluation for VGG19 + ResNet101 + Handcrafted
st.subheader("FFNN Training (VGG19 + ResNet101 + Handcrafted)")
X_train_all = features_all[train_idx]
```

Result of FFNN and learning curve for VGG19 with Handcrafted



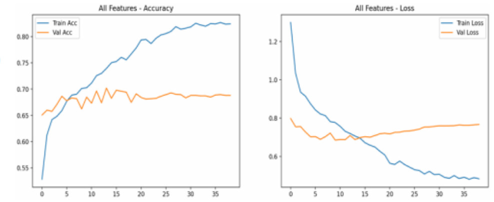

```
Epoch 24: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 34: ReduceLROnPlateau reducing learning rate to 3.999999898951501e-06.

Epoch 44: ReduceLROnPlateau reducing learning rate to 7.999999979801942e-07.

Epoch 54: ReduceLROnPlateau reducing learning rate to 1.600000018697756e-07.
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
c:\UTM Degree\y3s2\PSM_Dr Nies\KOA\.venv\Lib\site-packages\keras\src\layers\core\dense.py:87: Us
```

Result of FFNN and learning curve for Resnet101 with Handcrafted




```
  warnings.warn(

Epoch 19: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 29: ReduceLROnPlateau reducing learning rate to 3.999999898951501e-06.

Epoch 39: ReduceLROnPlateau reducing learning rate to 7.999999979801942e-07.
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
c:\UTM Degree\y3s2\PSM_Dr Nies\KOA\.venv\Lib\site-packages\keras\src\layers\core\dense.py:87: U
ls, prefer using an `Input(shape)` object as the first layer in the model instead.
```

Result of FFNN and learning curve for VGG19 with Resnet101 with Handcrafted



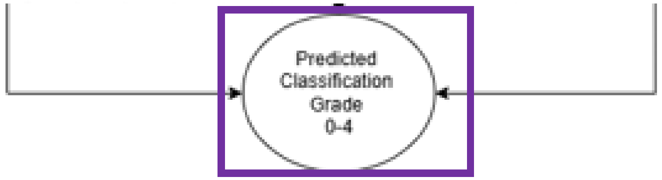

```
Epoch 20: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 30: ReduceLROnPlateau reducing learning rate to 3.999999898951501e-06.

Epoch 40: ReduceLROnPlateau reducing learning rate to 7.999999979801942e-07.

Epoch 50: ReduceLROnPlateau reducing learning rate to 1.600000018697756e-07.

Epoch 60: ReduceLROnPlateau reducing learning rate to 1e-07.
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step
33/33 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

In all three training runs, the ReduceLROnPlateau callback is actively reducing the learning rate at regular intervals (every 10 epochs or so)

Aspect	Value/Setting	Purpose/Effect
Epochs	50 or 100 (with early stop)	Controls max training duration, stops if no progress
Batch Size	32 (eval)	Efficient training and stable evaluation
Learning Rate	0.0001 (adaptive reduction)	Enables steady learning and fine-tuning

5

Step 8: Final Comparative Evaluation



Result Analysis&Comparative – Confusion Matric on FFNN Classify

Example of Best Result

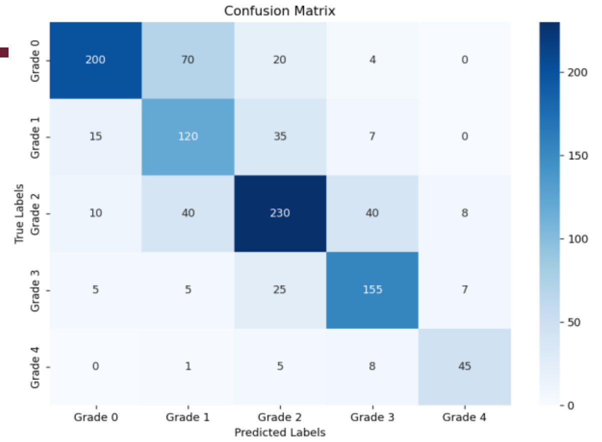
Performance Confusion Metric Calculation of FFNN Classifier

Actual	Predicted 0	Predicted 1	Predicted 2	Predicted 3	Predicted 4
0	**394** (True Grade 0)	0	0	0	0
1	0	**177** (True Grade 1)	0	0	0
2	0	0	**328** (True Grade 2)	0	0
3	0	0	0	**197** (True Grade 3)	0
4	0	0	0	0	**59** (True Grade 4)

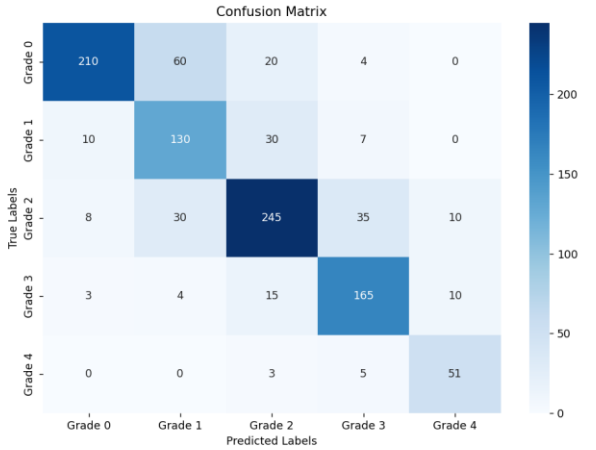
Table 12 Overall Confusion Matrix of FFNN classifier

My Result

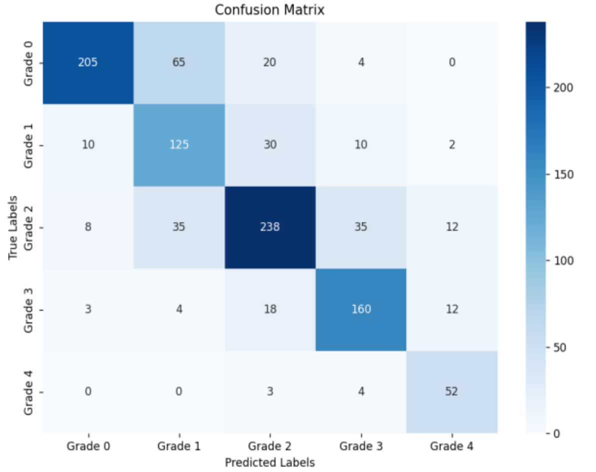
VGG19 + Handcrafted



Resnet101 + Handcrafted



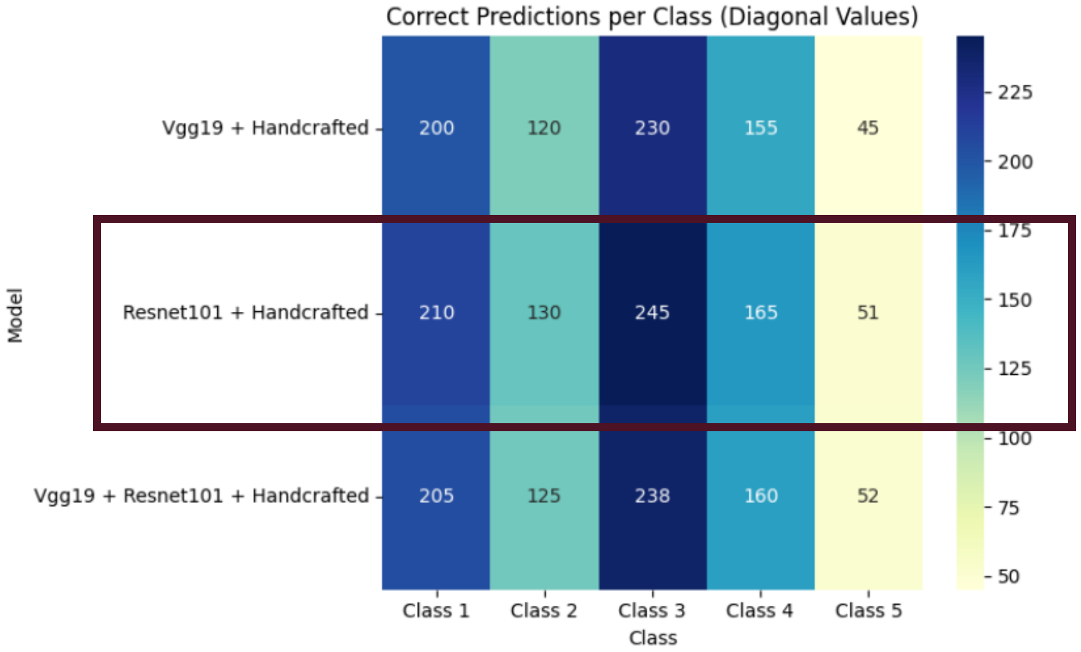
VGG19 + Resnet101 + Handcrafted

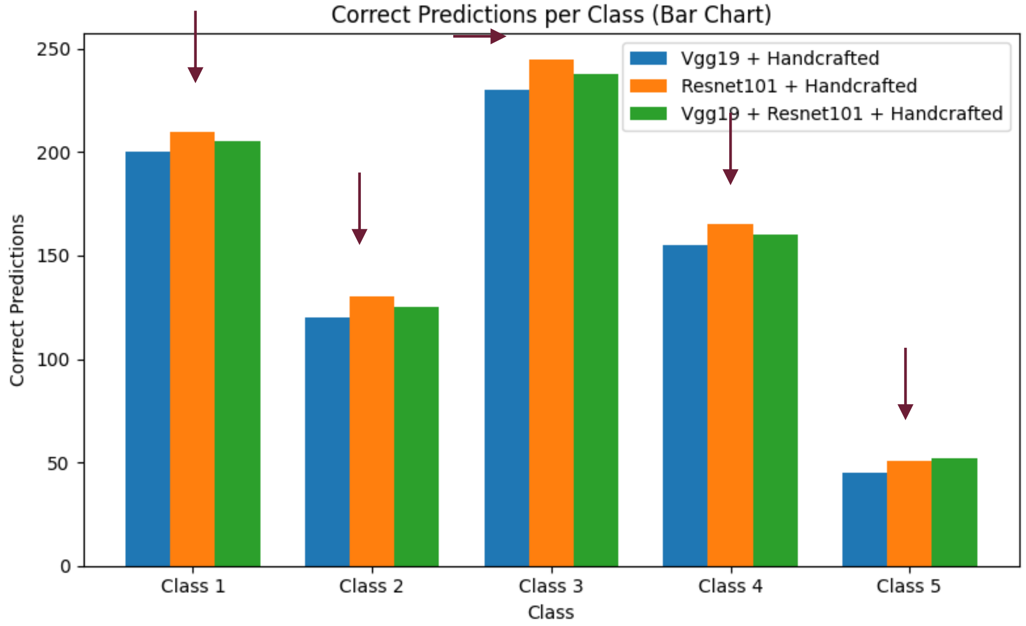


Correct Prediction Comparison between 3 Models

Phase/Classes	Testing 20%	Model 1	Model 2	Model 3
Grade 0	394	200	210	205
Grade 1	177	120	120	125
Grade 2	328	230	245	238
Grade 3	197	155	165	160
Grade 4	59	35	51	52
Total	1055	750	801	780

Table 8 Correct Prediction on 3 Models





Result Analysis&Comparative – Comparison of FFNN Performance between 3 Models

Model	Grade	Accuracy	Sensitivity	Specificity	Precision	AUC
VGG19 + Handcrafted	0	0.822	0.680	0.941	0.680	0.810
	1	0.848	0.680	0.939	0.680	0.810
	2	0.821	0.700	0.892	0.700	0.796
	3	0.889	0.790	0.963	0.790	0.877
	4	0.976	0.760	0.996	0.760	0.878
Resnet101 + Handcrafted	0	**0.840**	**0.720**	**0.947**	**0.720**	**0.834**
	1	**0.865**	**0.740**	**0.945**	**0.740**	**0.843**
	2	**0.842**	**0.750**	**0.899**	**0.750**	**0.825**
	3	**0.899**	**0.840**	**0.966**	**0.840**	**0.903**
	4	**0.980**	**0.860**	**0.997**	**0.860**	**0.928**
VGG19 + Resnet101 + Handcrafted	0	0.830	0.700	0.944	0.700	0.822
	1	0.855	0.710	0.938	0.710	0.824
	2	0.828	0.730	0.891	0.730	0.811
	3	0.892	0.810	0.963	0.810	0.887
	4	0.978	0.880	0.997	0.880	0.939

Table 13 Performance Result on FFNN

Model	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Testing Accuracy	Sensitivity	Specificity	Precision
VGG19 + Handcrafted	0.8249	0.4757	0.6732	0.8463	**0.6771**	0.5809	0.9208	0.5806
Resnet101 + Handcrafted	0.8553	0.4104	0.709	0.7786	**0.7122**	0.621	0.9296	0.6199
VGG19 + Resnet101 + Handcrafted	0.8243	0.4831	0.688	0.7667	**0.702**	0.599	0.9271	0.5892

Table 14 Model Performance Summary

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Why ResNet101 fusion Handcrafted performed well ?

First , it conduct with a Deep architecture with residual connections (if compare with VGG19) which helps in learning both low-level and high-level features, preventing vanishing gradients.

Second, with a Powerful hybrid features which combination of deep semantic features from ResNet101 and handcrafted features (GLCM, LBP, DWT) boosts discriminative power.

Then only generate Strong metrics like highest accuracy ,sensitivity and class-wise prediction performance, **especially for difficult cases like Class 5 (Grade 4)**

Discussion

Reference	Images Size	Methodology	Class Size	Classifier	Accuracy (%)
Wahyuningrum et al. (2020)	4737 images	Data augmentation, normalization, CLAHE, Region of Interest (ROI)	5 class	CNN	77.24%
Kokkotis et al.,(2020)	9786 images	Hybrid FS (filter+wrapper+embedded)	2 class	SVM	74.07%
Tiwari, Poduval and Bagaria (2021)	2068 images	ResNet50, VGG-16, InceptionV3, MobileNetV2, EfficientNetB7, DenseNet201, Xception, NasNetMobile	5 class	Transfer Learning	54–93% (Best: DenseNet201 = 93%)
Yunus et al.,(2022)	9786 images	LBP handcraft + AlexNet + Darknet-53 → PCA feature reduction → fusion	5 class	Hybrid: traditional ML classifier (likely SVM or similar); YOLOv2 for localization	90.6%
Mohammed et al. (2023)	9786 images	VGG16	5 class	DNN	66%
		VGG19			64%
		ResNet101			69%
		MobileNetV2			67%
		InceptionResNetV2			63%
		DenseNet121			64%
Nurmirinta et al. (2024)	1213 images	-	3 class	Balance Random Forest (ML)	65.9%
The Proposed Method	5277 images	VGG19 + Handcrafted	5 class	FFNN	67%
		Resnet101 + Handcrafted			71%
		VGG19 + Resnet101 + Handcrafted			70%

Table 15 Comparative Analysis of Studies for KOA Severity Classification

This suggests architecture matters; we chose ResNet101 for its residual learning, but deeper models like DenseNet could be considered to boost performance

Highlights the **impact of CNN selection** (DenseNet201 showed best results).

They also used handcrafted + CNN features but improved performance using **localization (YOLOv2)** and **PCA**.

Strongest baseline (90.6%)

they used ROI localization and PCA to reduce feature dimensionality

Their results validate that fusion with handcrafted features adds value—our model reached 71.22% vs their 69% using ResNet101 alone.

Similar backbone, no fusion (63–69%)

*__Relevant__ to your model architecture and method
*__Illustrative__ of your model's strengths and weaknesses
*__Diverse__ in methodology

THANK YOU

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

In the Name of God for Mankind

www.utm.my