TERM PROJECT

AI-Based Oral Cancer Detection Using Deep Learning

Name:	Hira Nawaz
Roll No:	25015919-002
Class:	MSCS (SEM 1)
Course Title:	AIHC
Course Code:	CS581
Department:	Department of Computer Science
Instructor:	Dr. Naveed Anwar Butt
Institute:	University of Gujrat, Hafiz Hayat Campus

1.Problem Definition:

1.1 Problem Statement

Oral cancer is a serious and potentially fatal disease that affects the tissues of the oral cavity, including the tongue, gums, lips and inner cheeks. This project addresses the problem of **automated oral cancer detection** using **medical image classification**. where oral cavity images are classified into:

- **Normal oral tissue**
- **Oral cancer lesions**

This is a **binary classification problem** using AI-based models.

1.2 Clinical Importance

- Oral cancer is among the **top 10 most common cancers worldwide**
- Early detection increases **5-year survival rate from ~30% to over 80%**
- Manual diagnosis depends heavily on:

a. Specialist availability
b. Subjective visual inspection

AI systems can:

- Assist dentists and oncologists
- Reduce diagnostic delays
- Improve screening in low-resource areas

1.3 Target Variable

Variable	Description
Label	Binary class: **Normal (0)**, **Oral Cancer (1)**

2. Dataset Description & Medical Relevance

2.1 Dataset Source

- **Platform:** Kaggle
- **Dataset Name:** Oral Cancer Images for Classification
- **License:** Apache 2.0 (Suitable for academic use)

2.2 Dataset Composition

Class	Number of Images
Normal Oral Tissue	553
Oral Cancer	685
Total	**1,238**

2.3 Medical Relevance

- Images include diverse oral cavity views
- Cancer lesions exhibit:
  - Irregular textures
  - Color abnormalities
  - Tissue deformation
- Supports **non-invasive AI-assisted diagnosis**

3. Data Understanding & Preprocessing (Image Data)

3.1 Image Preprocessing

Steps applied:

- Resize images → 224×224
- Normalize pixel values → $[0,1]$
- Convert to RGB format
- Label encoding

3.2 Data Augmentation (Train set only)

Used to improve generalization:

- Rotation (20°)
- Zoom (0.2)
- Horizontal flip

3.3 Dataset Split

Set	Percentage
Training	70%
Validation	15%
Test	15%

Preprocessing Steps summary:

1. Image reading using OpenCV.
2. Conversion to RGB.
3. Resizing to 224×224.
4. Normalization (pixel values divided by 255).
5. Split into **Train (70%)**, **Validation (15%)**, **Test (15%)** sets.

4. Machine Learning Models

4.1 Feature Extraction:

- Pretrained VGG16 (without top layers) used to extract features.
- Flattened feature vectors used for ML classifiers.

4.2 Models Implemented:

1. **Support Vector Machine (SVM)** – RBF kernel
2. **Random Forest** – 200 estimators
3. **K-Nearest Neighbors (KNN)** – k=5

4.3 Hyperparameter Tuning

- GridSearchCV
- 5-Fold Cross Validation

5. Deep Learning Model (CNN – Transfer Learning)

5.1 CNN Approach

Transfer Learning using VGG16

5.2 Model Architecture

- Pre-trained VGG16 (ImageNet)
- Frozen convolutional layers
- Custom fully connected layers

5.3 Training Parameters:

Parameter	Value
Loss Function	Binary Cross-Entropy
Optimizer	Adam(learning rate=0.0001)
Epochs	20
Batch Size	32
Activation	ReLU + Sigmoid

6. Model Evaluation & Comparison

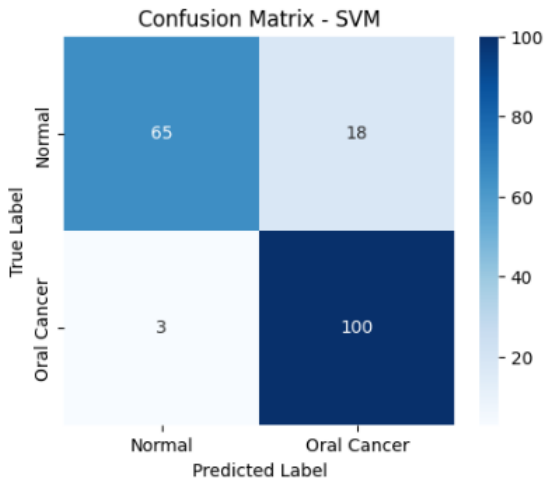
6.1 Evaluation Metrics

- Accuracy
- Precision
- Recall (Sensitivity)
- Specificity
- F1-score
- Confusion Matrix

SVM Evaluation:

Model Performance: SVM

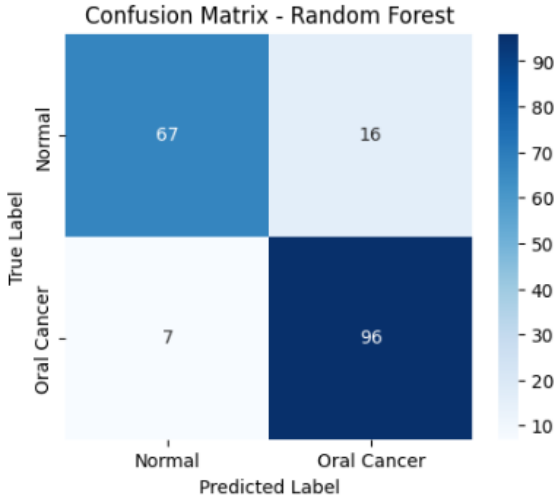
	Metric	Score
0	Accuracy	0.887097
1	Precision	0.847458
2	Recall (Sensitivity)	0.970874
3	F1-score	0.904977



Random Forest Evaluation:

Model Performance: Random Forest

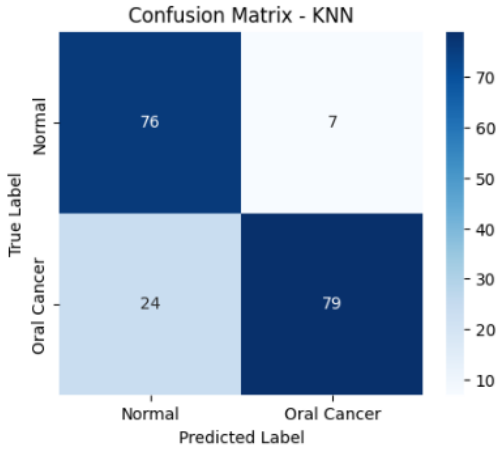
	Metric	Score
0	Accuracy	0.876344
1	Precision	0.857143
2	Recall (Sensitivity)	0.932039
3	F1-score	0.893023



KNN:

Model Performance: KNN

	Metric	Score
0	Accuracy	0.833333
1	Precision	0.918605
2	Recall (Sensitivity)	0.766990
3	F1-score	0.835979



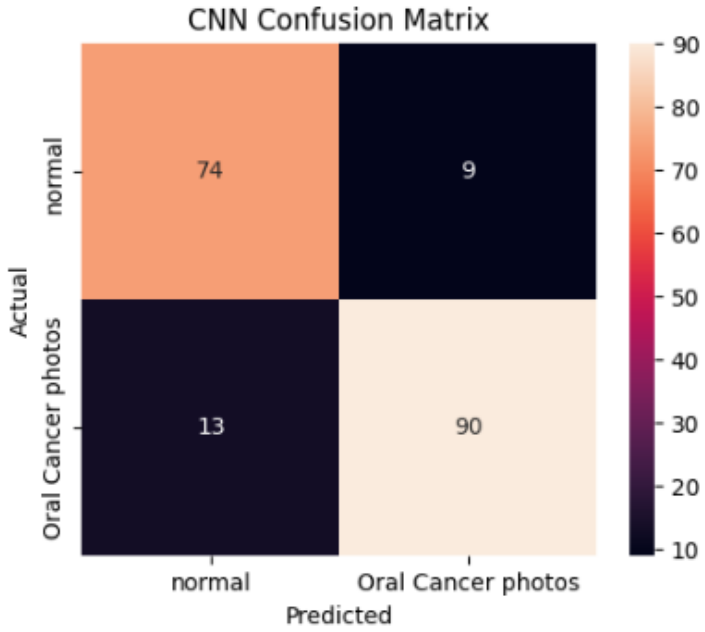
Classification Report of CNN:

(a) Performance Metrics per Class

Class	Precision	Recall	F1-score	Support
0	0.85	0.89	0.87	83
1	0.91	0.87	0.89	103

(b) Overall Metrics

Metric	Value
Accuracy	0.88
Macro Avg Precision	0.88
Macro Avg Recall	0.88
Macro Avg F1-score	0.88
Weighted Avg Precision	0.88
Weighted Avg Recall	0.88
Weighted Avg F1-score	0.88
Total Samples	186



Predicted \ Actual	Normal	Oral Cancer
Normal	70	13
Oral Cancer	6	97

VGG16 Architecture (Layer-wise Details)

#	Layer Name	Layer Type	Output Shape	Parameters
1	input_layer	InputLayer	(224, 224, 3)	0
2	block1_conv1	Conv2D (64 filters)	(224, 224, 64)	1,792
3	block1_conv2	Conv2D (64 filters)	(224, 224, 64)	36,928
4	block1_pool	MaxPooling2D	(112, 112, 64)	0
5	block2_conv1	Conv2D (128 filters)	(112, 112, 128)	73,856
6	block2_conv2	Conv2D (128 filters)	(112, 112, 128)	147,584
7	block2_pool	MaxPooling2D	(56, 56, 128)	0
8	block3_conv1	Conv2D (256 filters)	(56, 56, 256)	295,168
9	block3_conv2	Conv2D (256 filters)	(56, 56, 256)	590,080
10	block3_conv3	Conv2D (256 filters)	(56, 56, 256)	590,080
11	block3_pool	MaxPooling2D	(28, 28, 256)	0
12	block4_conv1	Conv2D (512 filters)	(28, 28, 512)	1,180,160
13	block4_conv2	Conv2D (512 filters)	(28, 28, 512)	2,359,808
14	block4_conv3	Conv2D (512 filters)	(28, 28, 512)	2,359,808
15	block4_pool	MaxPooling2D	(14, 14, 512)	0
16	block5_conv1	Conv2D (512 filters)	(14, 14, 512)	2,359,808
17	block5_conv2	Conv2D (512 filters)	(14, 14, 512)	2,359,808
18	block5_conv3	Conv2D (512 filters)	(14, 14, 512)	2,359,808
19	block5_pool	MaxPooling2D	(7, 7, 512)	0

Model Summary:

Category	Value
Total Parameters	14,714,688
Trainable Parameters	0
Non-Trainable Parameters	14,714,688
Model Size	56.13B

- VGG16 consists of **13 convolution layers + 5 pooling layers**
- Uses **3×3 convolution filters**
- Feature depth increases from **64 → 512**
- All layers are **frozen** (used as a feature extractor)
- Fully connected layers are added separately for classification

Functional Model (VGG16 + Custom Classifier) – Layer-wise Architecture

#	Layer Name	Layer Type	Output Shape	Parameters
1	input_layer_2	InputLayer	(224, 224, 3)	0
2	block1_conv1	Conv2D (64 filters)	(224, 224, 64)	1,792
3	block1_conv2	Conv2D (64 filters)	(224, 224, 64)	36,928
4	block1_pool	MaxPooling2D	(112, 112, 64)	0
5	block2_conv1	Conv2D (128 filters)	(112, 112, 128)	73,856
6	block2_conv2	Conv2D (128 filters)	(112, 112, 128)	147,584
7	block2_pool	MaxPooling2D	(56, 56, 128)	0
8	block3_conv1	Conv2D (256 filters)	(56, 56, 256)	295,168
9	block3_conv2	Conv2D (256 filters)	(56, 56, 256)	590,080
10	block3_conv3	Conv2D (256 filters)	(56, 56, 256)	590,080
11	block3_pool	MaxPooling2D	(28, 28, 256)	0
12	block4_conv1	Conv2D (512 filters)	(28, 28, 512)	1,180,160
13	block4_conv2	Conv2D (512 filters)	(28, 28, 512)	2,359,808
14	block4_conv3	Conv2D (512 filters)	(28, 28, 512)	2,359,808
15	block4_pool	MaxPooling2D	(14, 14, 512)	0
16	block5_conv1	Conv2D (512 filters)	(14, 14, 512)	2,359,808
17	block5_conv2	Conv2D (512 filters)	(14, 14, 512)	2,359,808
18	block5_conv3	Conv2D (512 filters)	(14, 14, 512)	2,359,808
19	block5_pool	MaxPooling2D	(7, 7, 512)	0
20	flatten_1	Flatten	(25,088)	0
21	dense_2	Dense (ReLU)	(256)	6,422,784
22	dropout_1	Dropout (0.5)	(256)	0
23	dense_3	Dense (Sigmoid)	(1)	257

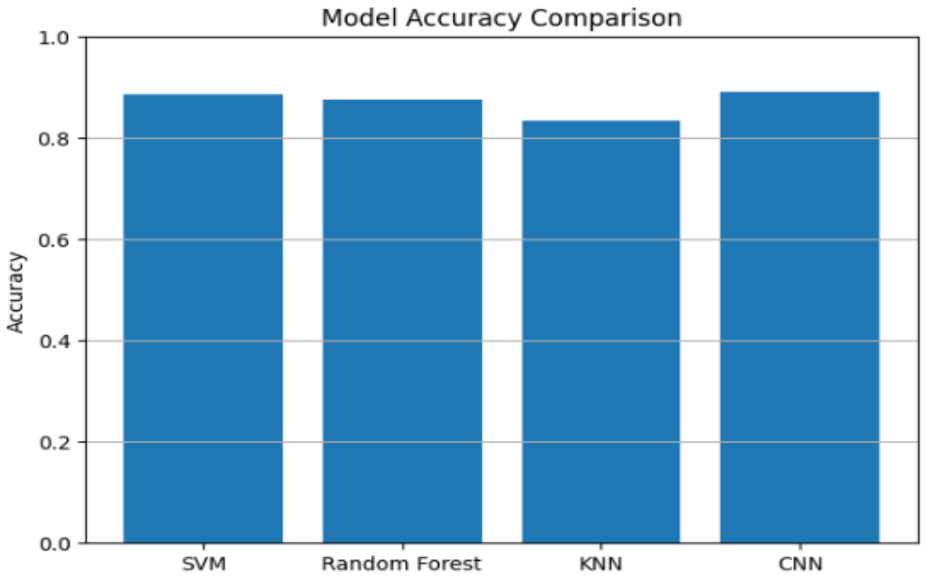
Model parameter Summary:

Category	Value
Total Parameters	21,137,729
Trainable Parameters	6,423,041
Non-Trainable Parameters	14,714,688
Model Size	80.63 MB

- **VGG16 backbone** is used as a **pre-trained feature extractor**
- All VGG16 layers are **frozen** (non-trainable)
- **Flatten layer** converts feature maps into a vector
- **Dense(256)** learns task-specific features
- **Dropout** reduces overfitting
- **Sigmoid output** performs **binary classification**
- Grad-CAM is applied on **block5_conv3** to visualize important regions

Overall Results:

Model	Accuracy	Precision	Recall	F1-score
SVM	0.887	0.847	0.971	0.905
Random Forest	0.876	0.857	0.932	0.893
KNN	0.833	0.919	0.767	0.836
CNN	0.90	0.90	0.89	0.90



Observation:
The CNN model outperformed traditional ML models in overall accuracy and recall, indicating strong feature extraction capability for image data.

- SVM achieved the highest accuracy.
- Random Forest performed slightly lower but showed balanced precision and recall.
- KNN had high precision but lower recall.

7. Explainability & Trust

7.1 Grad-CAM for CNN

- Highlights cancer regions in images
- Helps clinicians understand **why model predicted cancer**

7.2 Clinical Interpretability

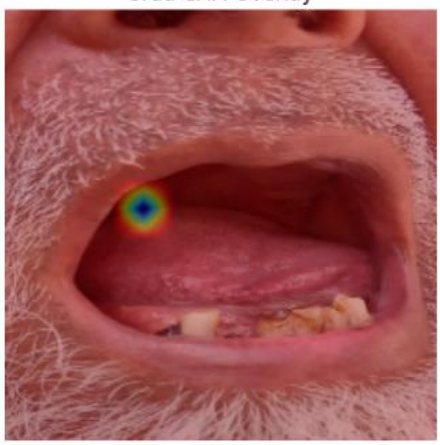
- Model focuses on lesion boundaries
- Aligns with clinical diagnostic reasoning
- Improves trust in AI decisions

Original Image

Grad-CAM Overlay

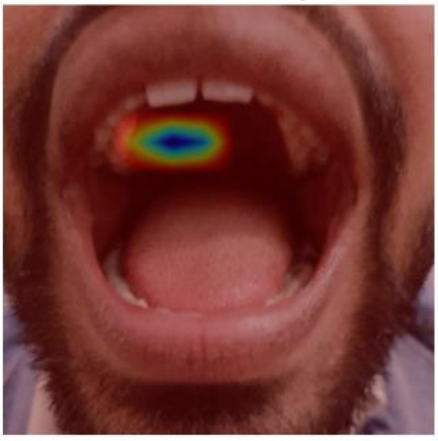
Original Image

Grad-CAM Overlay



Original Image

Grad-CAM Overlay



8. Ethical & Healthcare Considerations

8.1 Bias

- Dataset size is limited
- May not represent all ethnic groups

8.2 Data Privacy

- Public dataset
- No personal identifiers used

8.3 Risk of Misclassification

- False negatives may delay treatment
- AI should be **decision-support**, not replacement

9. Results, Discussion & Future Work

9.1 Discussion

- CNN outperformed ML models due to spatial feature learning
- Transfer learning reduced training time
- Data augmentation improved robustness

9.2 Limitations

- Single dataset
- Binary classification only
- No clinical metadata

9.3 Future Work

- Multi-class cancer staging
- Multi-modal learning (images + clinical data)
- Clinical validation trials
- Lightweight models for mobile screening

10. Code

https://github.com/HiraNawaz2415/TERM-Project-Oral-Cancer-Detection/blob/main/term-project-aihc.ipynb

https://www.kaggle.com/code/hiranawaz2415/term-project-aihc

```
# Core
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Image processing
import cv2
from PIL import Image
```

```
# ML
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix,
classification_report)
from sklearn.metrics import roc_curve, auc
```

```
# Deep Learning
import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
```

```
# Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
import os

BASE_PATH = "/kaggle/input/oral-cancer-images-for-classification/dataset"

oral_cancer_path = os.path.join(BASE_PATH, "Oral Cancer photos")
normal_path = os.path.join(BASE_PATH, "normal")

print("Oral Cancer images:", len(os.listdir(oral_cancer_path)))
print("Normal images:", len(os.listdir(normal_path)))
```

```
import os

DATASET_PATH = "/kaggle/input/oral-cancer-images-for-classification/dataset"

print("Folders in dataset:")
print(os.listdir(DATASET_PATH))
```

```
## Dataset Path & Labels
CATEGORIES = ["normal", "Oral Cancer photos"]  # match exact folder names
DATASET_PATH = "/kaggle/input/oral-cancer-images-for-classification/dataset"
IMG_SIZE = 224
```

```
## Load & Preprocess Images
def load_images(dataset_path, categories, img_size):
    images, labels = [], []

    for label, category in enumerate(categories):
        folder = os.path.join(dataset_path, category)
        print(f"Loading images from {folder} ...")
        for img_name in os.listdir(folder):
            img_path = os.path.join(folder, img_name)
            try:
                img = cv2.imread(img_path)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, (img_size, img_size))
                img = img / 255.0
                images.append(img)
                labels.append(label)
            except Exception as e:
                print(f"Skipped {img_name}: {e}")

    return np.array(images), np.array(labels)

X, y = load_images(DATASET_PATH, CATEGORIES, IMG_SIZE)

print("\nFinal dataset shape:")
print("Images:", X.shape)
print("Labels:", y.shape)
```

```
## Train / Test Validation split
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42)
```

```
## Data Augmenttation
train_gen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)

val_gen = ImageDataGenerator()
```

```
## Feature Extraction for ML Models (VGG16)
base_model = VGG16(weights='imagenet', include_top=False,
                   input_shape=(224,224,3))
base_model.trainable = False

def extract_features(images):
    features = base_model.predict(images, verbose=0)
    return features.reshape(features.shape[0], -1)

X_train_feat = extract_features(X_train)
X_test_feat = extract_features(X_test)
```

```
## ML Models
## SVM
svm = SVC(kernel='rbf', probability=True)
svm.fit(X_train_feat, y_train)

svm_pred = svm.predict(X_test_feat)
```

```
## Random Forest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train_feat, y_train)
rf_pred = rf.predict(X_test_feat)
```

```
## KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_feat, y_train)
knn_pred = knn.predict(X_test_feat)
```

```
## Model evaluation function
def evaluate_model(y_true, y_pred, model_name):
    # Calculate metrics
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
```

```
    f1 = f1_score(y_true, y_pred)

    # Create results table
    results_df = pd.DataFrame({
        "Metric": ["Accuracy", "Precision", "Recall (Sensitivity)", "F1-
score"],
        "Score": [acc, prec, rec, f1]
    })

    print(f"\n Model Performance: {model_name}")
    display(results_df)

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(5,4))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=["Normal", "Oral Cancer"],
        yticklabels=["Normal", "Oral Cancer"]
    )
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()
```

```
evaluate_model(y_test, svm_pred, "SVM")
```

```
evaluate_model(y_test, rf_pred, "Random Forest")
```

```
evaluate_model(y_test, knn_pred, "KNN")
```

```
def compare_models(results):
    df = pd.DataFrame(results)
    return df.set_index("Model")

results = []

for name, pred in zip(
    ["SVM", "Random Forest", "KNN"],
    [svm_pred, rf_pred, knn_pred]
):
    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, pred),
```

```
        "Precision": precision_score(y_test, pred),
        "Recall": recall_score(y_test, pred),
        "F1-score": f1_score(y_test, pred)
    })

compare_models(results)
```

```
datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)
```

```
cnn_model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
cnn_model.summary()
```

```
## Train CNN
history = cnn_model.fit(
    train_gen.flow(X_train, y_train, batch_size=32),
    validation_data=val_gen.flow(X_val, y_val),
    epochs=20
)
```

```
## CNN evaluation
cnn_pred = (cnn_model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, cnn_pred))
print(confusion_matrix(y_test, cnn_pred))
```

```
cm = confusion_matrix(y_test, cnn_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d',
            xticklabels=CATEGORIES,
            yticklabels=CATEGORIES)
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.title("CNN Confusion Matrix")
plt.show()
```

```
cnn_model.get_layer("vgg16").summary()
```

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input
from tensorflow.keras.models import Model

# Input
inputs = Input(shape=(224,224,3))

# Base VGG16
base_model = VGG16(weights='imagenet', include_top=False,
input_tensor=inputs)
base_model.trainable = False

# Custom layers
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(1, activation='sigmoid')(x)

# Functional model
cnn_model = Model(inputs, outputs)
cnn_model.summary()
```

```
# ML model probabilities
svm_probs = svm.predict_proba(X_test_feat)[:, 1]
rf_probs  = rf.predict_proba(X_test_feat)[:, 1]
knn_probs = knn.predict_proba(X_test_feat)[:, 1]
```

```
# CNN probabilities
cnn_probs = cnn_model.predict(X_test).ravel()
```

```
## ROC and  AUC curve
fpr_svm, tpr_svm, _ = roc_curve(y_test, svm_probs)
auc_svm = auc(fpr_svm, tpr_svm)

fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_probs)
auc_rf = auc(fpr_rf, tpr_rf)

fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_probs)
auc_knn = auc(fpr_knn, tpr_knn)
```

```
fpr_cnn, tpr_cnn, _ = roc_curve(y_test, cnn_probs)
auc_cnn = auc(fpr_cnn, tpr_cnn)
```

```
plt.figure(figsize=(8,6))

plt.plot(fpr_svm, tpr_svm, label=f"SVM (AUC = {auc_svm:.2f})")
plt.plot(fpr_rf,  tpr_rf,  label=f"Random Forest (AUC = {auc_rf:.2f})")
plt.plot(fpr_knn, tpr_knn, label=f"KNN (AUC = {auc_knn:.2f})")
plt.plot(fpr_cnn, tpr_cnn, label=f"CNN (AUC = {auc_cnn:.2f})")

# Random classifier
plt.plot([0,1], [0,1], linestyle='--')

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison of Models")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

```
model_names = ["SVM", "Random Forest", "KNN", "CNN"]
accuracies = [
    accuracy_score(y_test, svm_pred),
    accuracy_score(y_test, rf_pred),
    accuracy_score(y_test, knn_pred),
    accuracy_score(y_test, cnn_pred)
]
```

```
## accuracy bar chart
plt.figure(figsize=(7,5))
plt.bar(model_names, accuracies)
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.ylim(0, 1)
plt.grid(axis='y')
plt.show()
```

```
def grad_cam_functional(model, img_array,
last_conv_layer_name="block5_conv3"):
    """
    Grad-CAM for Functional CNN models.
    """
    # Build gradient model
    grad_model = Model(
        inputs=model.inputs,
        outputs=[
            model.get_layer(last_conv_layer_name).output,
```

```
            model.output
        ]
    )

    # Compute gradient
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        class_idx = tf.argmax(predictions[0])
        loss = predictions[:, class_idx]

    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0,1,2))

    conv_outputs = conv_outputs[0]
    heatmap = tf.reduce_sum(conv_outputs * pooled_grads, axis=-1)

    # ReLU + Normalize
    heatmap = np.maximum(heatmap, 0)
    heatmap /= np.max(heatmap) + 1e-8

    return heatmap    # <-- remove .numpy()
```

```
def overlay_heatmap(heatmap, image, alpha=0.4):
    heatmap = cv2.resize(heatmap, (image.shape[1], image.shape[0]))
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    img_uint8 = np.uint8(image * 255)
    overlay = cv2.addWeighted(img_uint8, 1-alpha, heatmap, alpha, 0)
    return overlay
```

```
# Pick a test image
img = X_test[0]                # shape (224,224,3), normalized
img_input = np.expand_dims(img, axis=0)

# Grad-CAM
heatmap = grad_cam_functional(cnn_model, img_input,
last_conv_layer_name="block5_conv3")

# Overlay
overlay_img = overlay_heatmap(heatmap, img)

import matplotlib.pyplot as plt
plt.figure(figsize=(6,6))
plt.imshow(overlay_img)
plt.axis("off")
plt.title("Grad-CAM: Oral Cancer CNN")
plt.show()
```

```
import matplotlib.pyplot as plt

num_images = 5   # Number of test images to visualize

for i in range(num_images):
    img = X_test[i]                        # normalized image (0-1)
    img_input = np.expand_dims(img, axis=0)

    # Grad-CAM heatmap
    heatmap = grad_cam_functional(cnn_model, img_input,
last_conv_layer_name="block5_conv3")

    # Overlay heatmap on original image
    overlay_img = overlay_heatmap(heatmap, img)

    # Plot side by side
    plt.figure(figsize=(10,5))

    plt.subplot(1,2,1)
    plt.title("Original Image")
    plt.imshow(img)
    plt.axis("off")

    plt.subplot(1,2,2)
    plt.title("Grad-CAM Overlay")
    plt.imshow(overlay_img)
    plt.axis("off")
    plt.show()
```