Source Code Files

cnn_gradcam.py

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import os

# --- Load your trained custom CNN model
model = load_model("models/simple_cnn_model.h5")

# === Generate for BOTH CLASSES ===

# Class: WITH MASK
with_mask_path = "dataset/with_mask/emknczty.jpg"
# Class: WITHOUT MASK
without_mask_path = "dataset/without_mask/agbxqqxi.jpg"

def generate_gradcam(img_path, save_path):
    # Load and preprocess image
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Since model was trained with rescale

    # Get last Conv layer
    last_conv_layer_name = None
    for layer in reversed(model.layers):
        if isinstance(layer, tf.keras.layers.Conv2D):
            last_conv_layer_name = layer.name
            break
    if last_conv_layer_name is None:
        raise ValueError("No Conv2D layer found.")

    # Build Grad-CAM
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output,
model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        pred_index = tf.argmax(predictions[0])
        class_channel = predictions[:, pred_index]

    grads = tape.gradient(class_channel, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    heatmap = heatmap.numpy()

    # Resize + overlay
    heatmap = cv2.resize(heatmap, (img.size[0], img.size[1]))
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    superimposed = heatmap * 0.4 + cv2.cvtColor(np.array(img),
cv2.COLOR_RGB2BGR)

    cv2.imwrite(save_path, cv2.cvtColor(superimposed.astype('uint8'),
cv2.COLOR_RGB2BGR))
    print(f"✅ Saved Grad-CAM to {save_path}")

# Generate both
generate_gradcam(with_mask_path, "cnn_gradcam_with_mask.jpg")
generate_gradcam(without_mask_path, "cnn_gradcam_without_mask.jpg")
```

confusion_matrix.py

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf

# Parameters
IMG_SIZE = 224
BATCH_SIZE = 32
DATA_DIR = "dataset"

# Load fine-tuned model
model = load_model("face_mask_finetuned.h5")

# Set up validation data generator
val_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    validation_split=0.2
)

val_generator = val_datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='validation',
    shuffle=False  # IMPORTANT: keep order for confusion matrix
)

# Get predictions
Y_pred = model.predict(val_generator)
y_pred = (Y_pred > 0.5).astype(int).reshape(-1)
```

```
y_true = val_generator.classes

# Labels
class_names = list(val_generator.class_indices.keys())

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot it
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.savefig("confusion_matrix.png")
plt.show()

# Optional: Print classification report
print(classification_report(y_true, y_pred, target_names=class_names))
```

gradcam.py

```
import tensorflow as tf
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import os

def make_gradcam_heatmap(img_array, model, last_conv_layer_name="Conv_1"):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output,
model.output]
    )
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        pred_index = tf.argmax(predictions[0])
        class_channel = predictions[:, pred_index]
    grads = tape.gradient(class_channel, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()

def process_image(img_path, model, save_as):
    # Load and preprocess image
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.mobilenet_v2.preprocess_input(img_array)
    # Generate heatmap
    heatmap = make_gradcam_heatmap(img_array, model)

    # Resize and overlay
    heatmap = cv2.resize(heatmap, (img.size[0], img.size[1]))
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    superimposed = heatmap * 0.4 + cv2.cvtColor(np.array(img),
cv2.COLOR_RGB2BGR)


    # Save and show
    output_img = cv2.cvtColor(superimposed.astype('uint8'), cv2.COLOR_RGB2BGR)
    cv2.imwrite(save_as, output_img)

    plt.imshow(cv2.cvtColor(output_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f"Grad-CAM: {os.path.basename(save_as)}")
    plt.show()


# Load your trained model
model = load_model("face_mask_finetuned.h5")

with_mask_img = "dataset/with_mask/emknczty.jpg"
without_mask_img = "dataset/without_mask/agbxqqxi.jpg"

# Generate both heatmaps
process_image(with_mask_img, model, "gradcam_with_mask.jpg")
process_image(without_mask_img, model, "gradcam_without_mask.jpg")
```

model 0 CNN.py

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
import matplotlib.pyplot as plt

# Params
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 10
DATA_DIR = "dataset"

# Data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='validation'
)

# Class weights
labels = train_generator.classes
class_weights = compute_class_weight(class_weight='balanced',
classes=np.unique(labels), y=labels)
class_weights_dict = dict(zip(np.unique(labels), class_weights))

# CNN Model (3 Conv blocks)
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()

# Train
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=EPOCHS,
    class_weight=class_weights_dict
)

model.save("simple_cnn_model.h5")

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Simple CNN Accuracy")
plt.legend()
plt.grid(True)
plt.savefig("simple_cnn_accuracy.png")
plt.show()
```

model.py

```
import os
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.optimizers import Adam

# Params
IMG_SIZE = 224
BATCH_SIZE = 32
EPOCHS = 10
DATA_DIR = "dataset"

# Data generators
train_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    validation_split=0.2,
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    DATA_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='validation'
)

# Import and compute class weights
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

labels = train_generator.classes
class_weights = compute_class_weight(class_weight='balanced',
classes=np.unique(labels), y=labels)
class_weights_dict = dict(zip(np.unique(labels), class_weights))
print("✅ Class weights:", class_weights_dict)

# MobileNetV2 model
base_model = MobileNetV2(include_top=False, weights="imagenet",
input_shape=(IMG_SIZE, IMG_SIZE, 3))
base_model.trainable = False  # freeze feature extractor

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
output = Dense(1, activation="sigmoid")(x)

model = Model(inputs=base_model.input, outputs=output)

# Compile
model.compile(optimizer=Adam(learning_rate=1e-4),
              loss="binary_crossentropy",
              metrics=["accuracy"])

# Train
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=EPOCHS,
    class_weight=class_weights_dict
)

# Save model
model.save("face_mask_finetuned.h5")

# 🔁 FINE-TUNING STARTS HERE
print("🔧 Starting fine-tuning...")

# Unfreeze MobileNetV2 top layers
base_model.trainable = True
fine_tune_at = 100  # Unfreeze from layer 100 onwards

for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

# Re-compile the model with a lower learning rate
model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Fine-tune the model
fine_tune_history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=5,  # fine-tune for a few more epochs
    class_weight=class_weights_dict
)

# Save fine-tuned model
model.save("face_mask_finetuned.h5")
print("✅ Fine-tuned model saved as face_mask_finetuned.h5")


# Plot accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig("accuracy_plot.png")
plt.show()
```

streamlit_app.py

```
import streamlit as st
from PIL import Image
import plotly.graph_objects as go

st.set_page_config(page_title="Face Mask Classification Dashboard",
layout="wide")
st.title("😷 Face Mask Detection - Interactive Dashboard")
st.markdown("This dashboard visualizes model training, performance, and
explainability for face mask detection using MobileNetV2 and a Custom CNN.")

# Sidebar navigation
section = st.sidebar.radio("Choose Section", [
    "Model Accuracy", "Confusion Matrix", "Grad-CAM Viewer", "Model
Comparison", "Classification Report"
])

# Section 1: Accuracy Plot
if section == "Model Accuracy":
    st.subheader("📈 Training & Validation Accuracy")
    col1, col2 = st.columns(2)

    with col1:
        st.image("accuracy_plot.png", caption="MobileNetV2 Accuracy",
use_container_width=True)
```

```
    with col2:
        st.image("simple_cnn_accuracy.png", caption="Custom CNN Accuracy",
use_container_width=True)

# Section 2: Confusion Matrix
elif section == "Confusion Matrix":
    st.subheader("📊 Confusion Matrix")
    st.image("confusion_matrix.png", caption="Confusion Matrix (MobileNetV2)",
use_container_width=True)

# Section 3: Grad-CAM Viewer
elif section == "Grad-CAM Viewer":
    st.subheader("🧠 Grad-CAM Comparison")
    model = st.radio("Select Model:", ["MobileNetV2", "Custom CNN"],
horizontal=True)

    if model == "MobileNetV2":
        with_mask_img = "gradcam_with_mask.jpg"
        without_mask_img = "gradcam_without_mask.jpg"
    else:
        with_mask_img = "cnn_gradcam_with_mask.jpg"
        without_mask_img = "cnn_gradcam_without_mask.jpg"

    col1, col2 = st.columns(2)

    with col1:
        st.subheader("With Mask")
        st.image(with_mask_img, use_container_width=True)

    with col2:
        st.subheader("Without Mask")
        st.image(without_mask_img, use_container_width=True)

# Section 4: Model Comparison Table
elif section == "Model Comparison":
    st.subheader("🤖 Model Performance Comparison")

    st.markdown("""
    | Metric                | MobileNetV2 | Custom CNN |
    |-----------------------|-------------|------------|
    | Validation Accuracy   | 95.01%      | 97.16%     |
    | Train Accuracy        | 97.51%      | 97.90%     |
    | Parameters            | ~2.2M       | ~11.1M     |
    | Training Time         | ~15 min     | ~25 min    |
    | Interpretability      | ✅ Good      | ✅ Good     |
    """)

    st.markdown(
        "> The custom CNN achieved higher validation accuracy but is heavier
and slower to train. MobileNetV2 remains a strong choice for real-time,
lightweight deployment."
    )
```

```
# Section 5: Classification Report
elif section == "Classification Report":
    st.subheader("📋 Classification Report – Interactive View")

    # Metrics
    labels = ["with_mask", "without_mask"]
    precision = [0.87, 1.00]
    recall = [1.00, 0.94]
    f1_score = [0.93, 0.97]
    support = [290, 732]

    # Plotly chart
    fig = go.Figure()
    fig.add_trace(go.Bar(name='Precision', x=labels, y=precision,
marker_color='mediumturquoise'))
    fig.add_trace(go.Bar(name='Recall', x=labels, y=recall,
marker_color='orange'))
    fig.add_trace(go.Bar(name='F1-Score', x=labels, y=f1_score,
marker_color='tomato'))

    fig.update_layout(
        title='MobileNetV2 Classification Report',
        yaxis=dict(title='Score'),
        barmode='group',
        xaxis_tickangle=-15
    )

    st.plotly_chart(fig, use_container_width=True)

    # Add support count separately
    st.markdown("**Support:**")
    for label, val in zip(labels, support):
        st.markdown(f"- `{label}`: {val} samples")

    st.markdown("**Accuracy: 0.96**")

st.markdown("---")
st.caption("Interactive dashboard by Ahmed Hujairi | Northumbria University |
Powered by Streamlit")
```