Report: Image Content Classification Model with Streamlit Web Application

Github - **https://github.com/sufyn/image-content-filtration**

Link - https://image-filter.streamlit.app/

Objective

The objective of this project was to develop a machine learning model capable of classifying images into three categories: 'Violent', 'Adult Content', and 'Safe'. The model was to be integrated into a Streamlit web application to demonstrate its functionality in a live web interface.

Tools and Libraries Used

- **Python**: Main programming language used for the project.
- **TensorFlow/Keras**: Used for building and training the convolutional neural network (CNN).
- **NumPy**: For data manipulation and preprocessing.
- **Matplotlib**: For creating visualizations of the training process.
- **Streamlit**: For creating an interactive web app to showcase the model's capabilities.
- **Pillow**: For handling image processing.

Key Steps

1. **Environment Setup**
    - Installed necessary libraries: TensorFlow, NumPy, Matplotlib, Streamlit, and Pillow.
2. **Data Collection and Preprocessing**
    - Collected images from various sources, categorized them into 'Violent', 'Adult Content', and 'Safe'.
    - Used TensorFlow's `ImageDataGenerator` to resize, normalize, and augment images.
    - Split the data into training and validation sets with an 80-20 ratio.
3. **Model Development and Training**
    - Constructed a CNN using MobileNetV2 as the base model.
    - Added custom layers on top of MobileNetV2 for specific classification tasks.
    - Compiled the model with the Adam optimizer and categorical cross-entropy loss function.
    - Trained the model using early stopping and a learning rate scheduler to prevent overfitting.
    - Applied class weights to handle class imbalance.
4. **Model Evaluation**
    - Evaluated the model using validation data.

    - Visualized training and validation accuracy and loss over epochs to assess model performance.

5. **Streamlit Integration**
    - Developed a Streamlit web app allowing users to upload images and view classification results.
    - Displayed model accuracy, loss, and training history within the web app.

Model Performance

- **Training Results**:
    - Achieved a training accuracy of approximately 97% and a validation accuracy of approximately 71% after 20 epochs.
    - Validation loss and accuracy fluctuated, indicating possible overfitting, but were managed using data augmentation, dropout, and class weights.
- **Evaluation Metrics**:
    - Validation Loss: 1.6966
    - Validation Accuracy: 0.7132

Streamlit Web Application

The Streamlit web application provides an interactive interface for users to:

- Upload an image for classification.
- View the uploaded image and its predicted category.
- Display confidence scores for each category.
- Show model performance metrics such as validation loss and accuracy.
- Visualize training history for accuracy and loss if available.

Conclusion

The project successfully developed a machine learning model to classify images into 'Violent', 'Adult Content', and 'Safe' categories with a reasonable degree of accuracy. Integrating the model into a Streamlit web application provided an interactive platform to demonstrate the model's capabilities, allowing users to upload images and view real-time classification results.

Future work could focus on collecting more data, especially for underrepresented classes, and fine-tuning the model further to improve accuracy and reduce overfitting. Additionally, deploying the Streamlit app on a cloud platform would make it accessible to a broader audience.

CODE -

```
!unzip '/content/drive/MyDrive/data/newdata.zip'

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os

# Define image size and paths
IMG_SIZE = (128, 128)
BATCH_SIZE = 32
DATA_DIR = '/content/Classification'

# Create ImageDataGenerator for preprocessing and augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Create training and validation datasets
train_generator = datagen.flow_from_directory(
    DATA_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    DATA_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)

# Print class indices to check the balance
print("Class indices:", train_generator.class_indices)
print("Number of samples in each class (train):", {k: v for k, v in
zip(train_generator.class_indices.keys(),
np.bincount(train_generator.classes))})
```

```
print("Number of samples in each class (validation):", {k: v for k, v in
zip(validation_generator.class_indices.keys(),
np.bincount(validation_generator.classes))})
```

```
Found 239 images belonging to 3 classes.
Found 57 images belonging to 3 classes.
Class indices: {'Adult Content': 0, 'Safe': 1, 'Violent': 2}
Number of samples in each class (train): {'Adult Content': 80, 'Safe': 80,
'Violent': 79}
Number of samples in each class (validation): {'Adult Content': 19, 'Safe':
19, 'Violent': 19}
```

```
# Calculate class weights to handle imbalance
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_generator.classes),
    y=train_generator.classes
)
class_weights = {i: class_weights[i] for i in range(len(class_weights))}

print("Class weights:", class_weights)
```

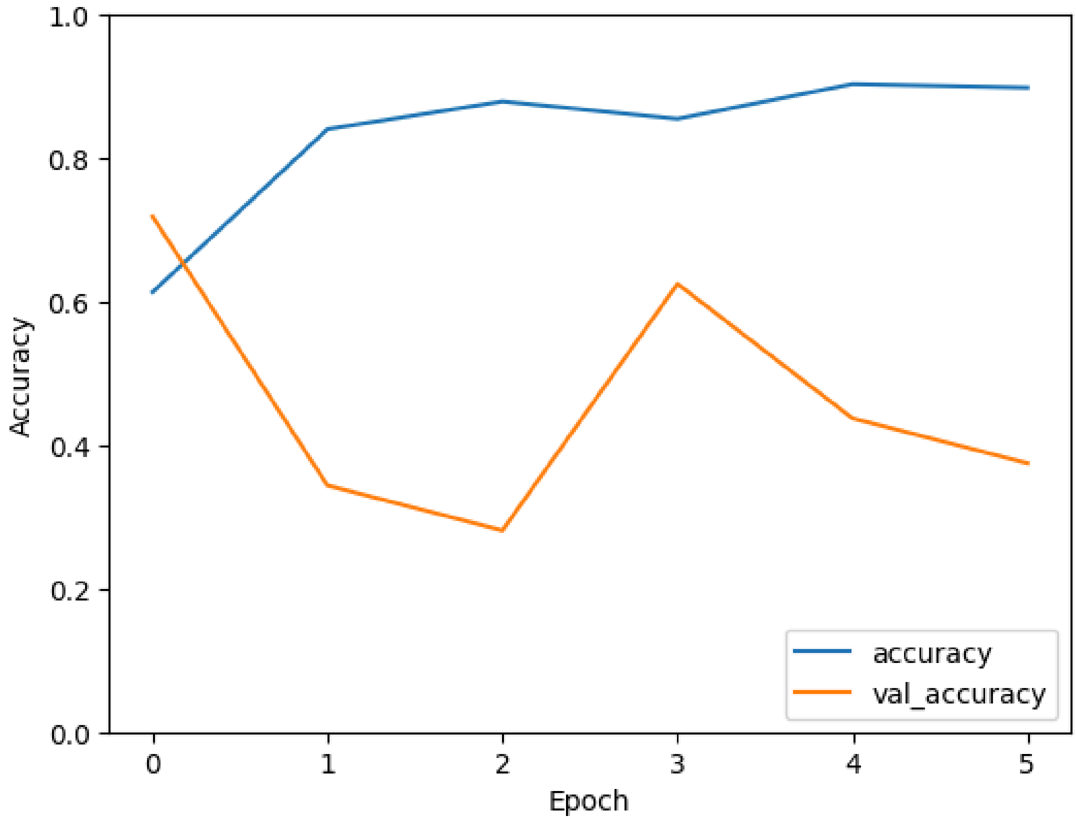
```
Class weights: {0: 0.9958333333333333, 1: 0.9958333333333333, 2:
1.0084388185654007}
```

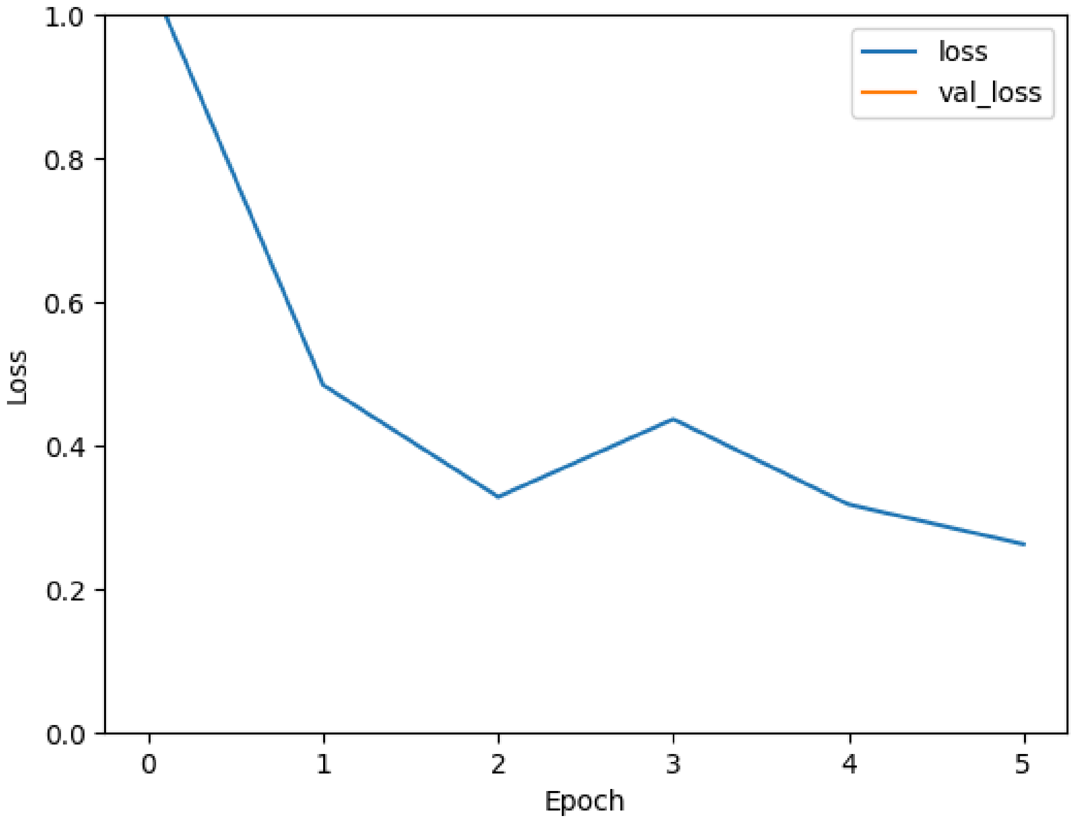
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Load the base model
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)  # Increasing dropout to reduce overfitting
x = Dense(128, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

# Create the full model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```python
# Learning rate scheduler
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=1e-6)
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
# Train the model with class weights and learning rate scheduler
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    epochs=11,
    callbacks=[early_stopping, reduce_lr],
    class_weight=class_weights
)

import matplotlib.pyplot as plt

# Evaluate the model on validation data
val_loss, val_accuracy = model.evaluate(validation_generator)

print(f"Validation Loss: {val_loss}")
print(f"Validation Accuracy: {val_accuracy}")

# Plot accuracy and loss over epochs
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.legend(loc='upper right')
plt.show()

# Save the model
model.save('ml2.h5')
```

```
Epoch 1/11
1/7 [===>..........................] - ETA: 2:09 - loss: 1.4170 - accuracy:
0.4375
```

```
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:996: UserWarning:
Palette images with Transparency expressed in bytes should be converted to
RGBA images
  warnings.warn(

7/7 [==============================] - 35s 2s/step - loss: 1.0566 - accuracy:
0.6135 - val_loss: 1.0067 - val_accuracy: 0.7188 - lr: 0.0010
Epoch 2/11
7/7 [==============================] - 15s 2s/step - loss: 0.4838 - accuracy:
0.8406 - val_loss: 3.4027 - val_accuracy: 0.3438 - lr: 0.0010
Epoch 3/11
7/7 [==============================] - 16s 2s/step - loss: 0.3280 - accuracy:
0.8792 - val_loss: 5.8738 - val_accuracy: 0.2812 - lr: 0.0010
Epoch 4/11
7/7 [==============================] - 15s 2s/step - loss: 0.4363 - accuracy:
0.8551 - val_loss: 2.9826 - val_accuracy: 0.6250 - lr: 0.0010
Epoch 5/11
7/7 [==============================] - 16s 2s/step - loss: 0.3175 - accuracy:
0.9034 - val_loss: 3.7836 - val_accuracy: 0.4375 - lr: 2.0000e-04
Epoch 6/11
7/7 [==============================] - 16s 2s/step - loss: 0.2621 - accuracy:
0.8986 - val_loss: 3.5318 - val_accuracy: 0.3750 - lr: 2.0000e-04
2/2 [==============================] - 1s 248ms/step - loss: 0.8011 -
accuracy: 0.7193
Validation Loss: 0.8010649681091309
Validation Accuracy: 0.719298243522644
```


1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0
1
2
3
4
5
Epoch
accuracy
val_accuracy



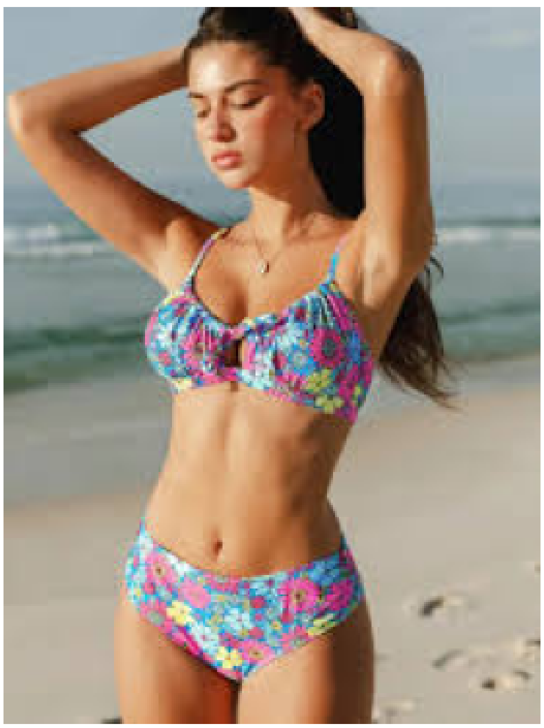

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

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

# Load the trained model
model = load_model('ml2.h5')

# Define the function to predict the class of a new image
def predict(image_path):
    img = image.load_img(image_path, target_size=(128, 128))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    prediction = model.predict(img_array)
    print(prediction)
    return np.argmax(prediction, axis=1)[0], prediction
```

```python
# Example usage with a new test image
test_image_path = '/content/adult.jpg'
label, prediction = predict(test_image_path)
categories = ['Adult Content','Safe','Violent']
predicted_category = categories[label]

# Display the test image and prediction
test_image = image.load_img(test_image_path)
plt.imshow(test_image)
plt.title(f'Predicted: {predicted_category} (Confidence:
{prediction[0][label]:.2f})')
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 1s 1s/step
[[9.9623835e-01 5.8698712e-04 3.1746132e-03]]
```

Predicted: Adult Content (Confidence: 1.00)



```python
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.15.0
```

Streamlit -

```
import streamlit as st

from PIL import Image

import tensorflow as tf

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing import image

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


# Load the trained model

model = load_model('ml2.h5')



# predict the class

def predict(image):

    image = image.resize((128, 128))

    image = np.array(image) / 255.0

    image = np.expand_dims(image, axis=0)

    prediction = model.predict(image)

    print(prediction)

    return np.argmax(prediction, axis=1)[0], prediction
```

```python
st.title('Image Content Classification')

st.write('Upload an image to classify it into Violent, Adult Content, or Safe.')


uploaded_file = st.file_uploader("Choose an image...", type="jpg")


if uploaded_file is not None:

    image = Image.open(uploaded_file)

    st.image(image, caption='Uploaded Image', use_column_width=True)

    st.write("")

    st.write("Classifying...")

    label, prediction = predict(image)

    categories = ['Adult Content','Safe','Violent']

    predicted_category = categories[label]

    st.write(f'This image is classified as: {categories[label]}')

    st.write(f'Predicted: {predicted_category} (Confidence: 
{prediction[0][label]:.2f})')


    # Access individual class probabilities from prediction

    class_probabilities = {category: prob for category, prob in zip(categories, 
prediction[0])}

    Validation_Loss= 0.8010649681091309

    Validation_Accuracy= 0.719298243522644


    st.write("Model Statistics:")

    st.write(f"- Validation Accuracy: {Validation_Accuracy:.2f}")
```

```
st.write(f"- Validation Loss: {Validation_Loss:.2f}")


with st.expander("Detailed Prediction Breakdown"):
    for i, category in enumerate(categories):
        class_probability = prediction[0][i]
        st.write(f"- {category}: {class_probability:.2f}")



# Display class probabilities as a table
st.subheader('Class Probabilities')
st.table(class_probabilities)


# Display prediction details
fig, ax = plt.subplots()
ax.bar(categories, prediction[0])
ax.set_ylabel('Confidence')
ax.set_title('Prediction Confidence for Each Category')
st.pyplot(fig)


# Display a pie chart of class probabilities
fig, ax = plt.subplots()
ax.pie(prediction[0], labels=categories, autopct='%1.1f%%')


ax.set_title('Class Probabilities')
```

```python
st.pyplot(fig)


# Display a bar chart of class probabilities

fig, ax = plt.subplots()

ax.barh(categories, prediction[0])

ax.set_xlabel('Confidence')

ax.set_title('Prediction Confidence for Each Category')

st.pyplot(fig)


# Display model accuracy and loss

history = np.load('model/ml2_history.npy', allow_pickle=True).item()


st.write("### Model Performance")

st.write(f"Validation Accuracy: {history['val_accuracy'][-1]:.2f}")

st.write(f"Validation Loss: {history['val_loss'][-1]:.2f}")


# Plot accuracy and loss over epochs

st.write("### Training History")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

ax1.plot(history['accuracy'], label='Training Accuracy')

ax1.plot(history['val_accuracy'], label='Validation Accuracy')

ax1.set_title('Accuracy')

ax1.set_xlabel('Epoch')

ax1.set_ylabel('Accuracy')
```

```
    ax1.legend()


    ax2.plot(history['loss'], label='Training Loss')

    ax2.plot(history['val_loss'], label='Validation Loss')

    ax2.set_title('Loss')

    ax2.set_xlabel('Epoch')

    ax2.set_ylabel('Loss')

    ax2.legend()


    st.pyplot(fig)
```