Stress_Image_Classificatioin

January 22, 2025

[89]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow import keras
```

[ ]:
```python
import kagglehub

# Download the dataset
path = kagglehub.dataset_download("preritbhagat/stress-non-stress-images")

print("Path to dataset files:", path)
```

[ ]:
```python
import os
current_path = os.getcwd()
print("Current path:", current_path)

# Importing the dataset
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    './KDEF/KDEF/Train',
    shuffle=True,
    batch_size=32,
)

print(len(dataset))
```

[ ]:
```python
# Two types of images in the dataset
class_titles = dataset.class_names
class_titles
```

[ ]:
```python
# Displaying some images
plt.figure(figsize=(10, 10))

for image, label in dataset.take(1): # take(1) takes the first batch of 32␣
↪images
    labels = label.numpy()
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(image[i].numpy().astype("uint8"))
        plt.title(class_titles[labels[i]], fontsize=10)
        plt.axis("off")
```

[ ]:
```
# Build up the CNN model
model = tf.keras.Sequential([
    # Preprocessing layer
    tf.keras.layers.Resizing(256, 256),
    tf.keras.layers.Rescaling(1./255),

    # Convolutional layers and pooling layers
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=(32, 256, 256, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    # Dense layers
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5), # Dropout layer to prevent overfitting
    tf.keras.layers.Dense(2, activation='softmax')
])

model.build((32, 256, 256, 3))

model.summary()
```

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

# Train the model
history = model.fit(
    dataset,
    batch_size = 32,
    verbose = 1,
    epochs = 10
)
```

[ ]:
```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.legend()
plt.show()
```

[ ]:
```python
# import the test dataset
dataset_test = tf.keras.preprocessing.image_dataset_from_directory(
    './KDEF/KDEF/Test',
    shuffle=True,
    batch_size=32,
)

# Evaluate the model
test_loss, test_acc = model.evaluate(dataset_test)
print(f'The test accuracy is: {test_acc} and the test loss is: {test_loss}')
```

[ ]:
```python
# Visualize the predicted types vs actual types
plt.figure(figsize=(10, 10))

prediction_dataset = dataset_test.skip(5).take(1)

for image, label in prediction_dataset:
    labels = label.numpy()
    predictions = model.predict(image)

    batch_size = image.shape[0]
    for i in range(min(batch_size, 9)):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image[i].numpy().astype("uint8"))
        predicted_label = np.argmax(predictions[i])
        actual_label = labels[i]
        plt.title(f'Actual: {class_titles[actual_label]}, Predicted:␣
 ↪{class_titles[predicted_label]}', fontsize=10)
        plt.axis("off")

plt.tight_layout()
plt.show()
```

[98]:
```python
# Try to combine the transformer model with the CNN model
# Build the transformer model
# Transformer Encoder Block
class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = tf.keras.layers.MultiHeadAttention(num_heads=num_heads,␣
 ↪key_dim=embed_dim)
        self.ffn = tf.keras.Sequential([
            tf.keras.layers.Dense(ff_dim, activation="relu"),
            tf.keras.layers.Dense(embed_dim),
        ])
        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
```

```
        self.dropout1 = tf.keras.layers.Dropout(rate)
        self.dropout2 = tf.keras.layers.Dropout(rate)

    def call(self, inputs, training=None):  # Ensure `training` is passed
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

[ ]:

```
# Combine CNN and Transformer
def build_combined_model(input_shape=(256, 256, 3), num_classes=2):
    inputs = tf.keras.Input(shape=input_shape)

    # Step 1: CNN Feature Extraction
    x = tf.keras.layers.Resizing(256, 256)(inputs)
    x = tf.keras.layers.Rescaling(1./255)(x)
    x = tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)
    x = tf.keras.layers.Flatten()(x)

    # Step 2: Transformer Encoder
    x = tf.keras.layers.Dense(128, activation='relu')(x)  # Embedding
    x = tf.keras.layers.Reshape((1, 128))(x)  # Ensure shape is [batch_size,␣
 ↪seq_len, embed_dim]
    transformer_block = TransformerBlock(embed_dim=128, num_heads=4, ff_dim=512)
    x = transformer_block(x)

    # Step 3: Classification
    x = tf.keras.layers.GlobalAveragePooling1D()(x)  # Reduce sequence dimension
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(num_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model


# Build and compile the model
combined_model = build_combined_model()
combined_model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
```

```
combined_model.summary()
```

[ ]:
```
history = combined_model.fit(
    dataset,
    epochs=10,
    batch_size=32,
    verbose=1
)
```

[ ]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.legend()
plt.show()
```

[ ]:
```
 # import the test dataset
dataset_test = tf.keras.preprocessing.image_dataset_from_directory(
    './KDEF/KDEF/Test',
    shuffle=True,
    batch_size=32,
)

# Evaluate the model
test_loss, test_acc = model.evaluate(dataset_test)
print(f'The test accuracy is: {test_acc} and the test loss is: {test_loss}')
```

[ ]:
```
# Visualize the predicted types vs actual types
plt.figure(figsize=(10, 10))

prediction_dataset = dataset_test.skip(5).take(1)

for image, label in prediction_dataset:
    labels = label.numpy()
    predictions = model.predict(image)

    batch_size = image.shape[0]
    for i in range(min(batch_size, 9)):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(image[i].numpy().astype("uint8"))
        predicted_label = np.argmax(predictions[i])
        actual_label = labels[i]
        plt.title(f'Actual: {class_titles[actual_label]}, Predicted:␣
 ↪{class_titles[predicted_label]}', fontsize=10)
        plt.axis("off")

plt.tight_layout()
```

```
plt.show()
```