Class note

SEnet

1. Imports and Dataset Setup:

python
Copy code
```
import os
import cv2
import numpy as np
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

- **`os`**: A Python module that provides functions to interact with the operating system, such as listing directories and file paths.
- **`cv2`**: OpenCV library for image processing (used to read and resize images).
- **`numpy` (`np`)**: A library for handling numerical operations, especially arrays.
- **`to_categorical`**: A function from `tensorflow.keras.utils` that converts integer labels into one-hot encoded vectors for classification tasks.
- **`train_test_split`**: A function from `sklearn.model_selection` that splits your dataset into training and testing sets.

2. Dataset Parameters and Initialization:

python
Copy code
```
dataset_path = '/content/drive/MyDrive/test_vggface'
classes = os.listdir(dataset_path)  # Assuming class folders are
names of each person
```

- **`dataset_path`**: Path to the dataset folder containing subfolders for each class (person).
- **`os.listdir(dataset_path)`**: Lists all the folders (classes) in the dataset. Each folder is assumed to represent a different person.

3. Image and Label Parameters:

python
Copy code
```
img_size = 224
num_classes = len(classes)
```

- **`img_size = 224`**: All images will be resized to 224x224 pixels to match the input size required by the model.
- **`num_classes`**: The number of different classes (people) is determined by counting the number of subfolders in the dataset directory.

4. Data Loading and Preprocessing:

python
Copy code

```python
data = []
labels = []

for class_idx, class_name in enumerate(classes):
    class_path = os.path.join(dataset_path, class_name)
    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)
        img = cv2.imread(img_path)
        img = cv2.resize(img, (img_size, img_size))
        data.append(img)
        labels.append(class_idx)
```

- **`data = []` and `labels = []`**: Empty lists to store the image data and their corresponding labels.
- The loop goes over each **class** (person), loads each image using **`cv2.imread()`**, resizes it to 224x224 using **`cv2.resize()`**, and stores it in the **`data`** list.
- **`labels.append(class_idx)`**: The class index (numeric label corresponding to each person) is added to the `labels` list.

5. Normalization and One-Hot Encoding:

python
Copy code

```python
data = np.array(data, dtype='float32') / 255.0
labels = np.array(labels)
labels = to_categorical(labels, num_classes)
```

- **Normalization**: The image pixel values are divided by 255 to normalize them into the range [0, 1]. This is important for training neural networks because smaller values speed up convergence.
- **`labels = np.array(labels)`**: Converts the list of labels into a NumPy array.
- **One-Hot Encoding**: **`to_categorical()`** converts the integer labels into one-hot encoded vectors, where each class is represented by a binary vector. For example, if

there are 5 classes, label 0 becomes `[1, 0, 0, 0, 0]`, label 1 becomes `[0, 1, 0, 0, 0]`, etc.

6. Train-Test Split:

python
Copy code
```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

- **`train_test_split()`** splits the data and labels into a training set (80%) and a test set (20%).
- **`test_size=0.2`**: Specifies that 20% of the dataset should be used for testing.
- **`random_state=42`**: Ensures reproducibility, meaning that the data will be split the same way each time you run the code.

7. Squeeze-and-Excitation (SE) Block:

python
Copy code
```
def squeeze_excite_block(input_tensor, ratio=16):
    channel_axis = -1
    filters = input_tensor.shape[channel_axis]

    se = GlobalAveragePooling2D()(input_tensor)
    se = Reshape((1, 1, filters))(se)

    se = Dense(filters // ratio, activation='relu')(se)
    se = Dense(filters, activation='sigmoid')(se)

    se = Multiply()([input_tensor, se])

    return se
```

- **Squeeze-and-Excitation (SE)** block is a mechanism to recalibrate channel-wise feature responses.
    - **`GlobalAveragePooling2D()`**: Squeezes the feature map by calculating the global average of each channel, resulting in a vector.
    - **`Reshape()`**: Reshapes the squeezed vector into a form that can be processed by the Dense layers.
    - **`Dense()`**: Two fully connected (Dense) layers:
        - First, reduces the number of channels by a factor of `ratio`.
        - Second, restores the original number of channels.

- `Multiply()`: Scales the original input feature map by the recalibrated vector, emphasizing important features.

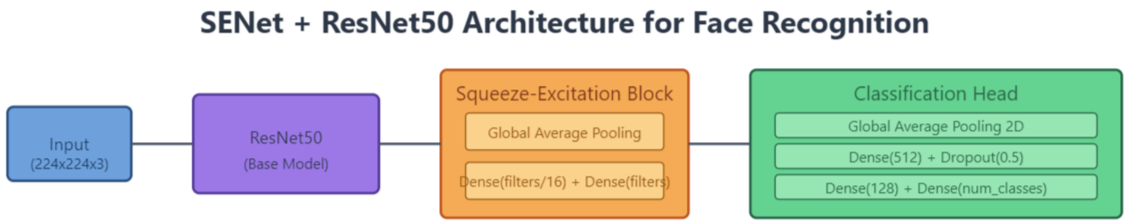
8. Model Architecture:

python
Copy code

```python
base_model = ResNet50(include_top=False, input_shape=(224, 224, 3))

x = base_model.output
x = squeeze_excite_block(x)

x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dense(num_classes, activation='softmax')(x)
```



- **Input Layer**
  - Accepts 224x224x3 RGB images
  - Images are normalized to [0,1] range
  - Batch processing of 32 images
- **ResNet50 Base Model**
  - Pre-trained on ImageNet
  - Excludes top classification layers
  - Acts as powerful feature extractor
- **Squeeze-Excitation Block**

python

Copy

```python
def squeeze_excite_block(input_tensor, ratio=16):
    filters = input_tensor.shape[-1]
    # Squeeze: Global information embedding
    se = GlobalAveragePooling2D()(input_tensor)
    se = Reshape((1, 1, filters))(se)
    # Excitation: Channel-wise calibration
```

```
    se = Dense(filters // ratio, activation='relu')(se)
    se = Dense(filters, activation='sigmoid')(se)
    # Scale: Feature recalibration

return Multiply()([input_tensor, se])
```

- Adds channel attention mechanism
- Highlights important features
- Reduction ratio of 16 for efficiency

4. **Classification Head**
    - Global Average Pooling to reduce spatial dimensions
    - Dense layers with decreasing dimensions:
        - 512 units with ReLU
        - Dropout(0.5) for regularization
        - 128 units with ReLU
        - Final layer with num_classes units and softmax
5. **Training Configuration**
    - Optimizer: Adam
    - Loss: Categorical Crossentropy
    - Metrics: Accuracy
    - Batch size: 32
    - Train-test split: 80-20

9. Model Compilation:

python
Copy code

```
model = Model(inputs=base_model.input, outputs=x)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

- **Model**: Combines the base ResNet50 model with the SE block and fully connected layers to form the complete model.
- **Compilation**:
    - Optimizer: Adam, a commonly used optimizer for training neural networks.
    - Loss: Categorical cross-entropy, appropriate for multi-class classification.
    - Metrics: Tracks accuracy during training and testing.

10. Training the Model:

python
Copy code

```
history = model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=10, batch_size=32)
```

- **`model.fit()`** trains the model for 10 epochs using a batch size of 32.
- The model is trained on `X_train` and `y_train`, and validated on the test data (`X_test`, `y_test`).

11. Saving the Model:

python
Copy code

```
model.save('/content/drive/MyDrive/models_face_recognition/SENet.ker
as')
```

- After training, the model is saved to the specified file path for future use.

12. Evaluating the Model:

python
Copy code

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')
```

- **`model.evaluate()`** evaluates the model on the test set (`X_test`, `y_test`) and prints the loss and accuracy.

13. Displaying Predictions:

python
Copy code

```
import numpy as np
import matplotlib.pyplot as plt

num_images = 9
random_indices = np.random.choice(X_test.shape[0], num_images,
replace=False)
predicted_labels = model.predict(X_test[random_indices])
predicted_classes = np.argmax(predicted_labels, axis=1)
actual_classes = np.argmax(y_test[random_indices], axis=1)
```

- Randomly selects 9 images from the test set (`X_test`) and makes predictions using `model.predict()`.
- The predicted and actual class labels are obtained using `np.argmax()`.

14. Displaying Images and Predictions:

python
Copy code
```
plt.figure(figsize=(10, 10))
for i, idx in enumerate(random_indices):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X_test[idx])
    plt.title(f"Actual: {actual_classes[i]}, Predicted:
{predicted_classes[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

- A 3x3 grid of the selected images is displayed, along with their actual and predicted labels.

How does SEnet works

SENet (Squeeze-and-Excitation Network) enhances the representational power of a convolutional neural network by explicitly modeling the interdependencies between channels. Here's how SENet works within the given code context and its significance in improving the model performance:

How SENet Works

1. **Squeeze Operation**:
    - In the **squeeze** phase, global average pooling is applied to the feature maps produced by the preceding convolutional layers.
    - This operation reduces each channel to a single value by averaging all spatial locations (H x W) in that channel.
    - The output is a 1D tensor that captures the global context for each channel, resulting in a vector of size CCC (number of channels).
2. **Excitation Operation**:
    - The squeezed output is passed through two fully connected (Dense) layers.
    - The first layer reduces the number of channels to C/rC/rC/r (where rrr is a reduction ratio, usually a small integer like 16), applying a ReLU activation.
    - The second layer restores the output to CCC channels and applies a sigmoid activation, producing a set of channel weights sss in the range [0,1][0, 1][0,1].

3. **Recalibration**:
    - Finally, the channel weights are multiplied with the original feature maps. This step emphasizes the channels deemed more important while suppressing the less significant ones.

Implementation in Your Code

In the provided code, the SENet block is integrated as follows:

python
Copy code
```
def squeeze_excite_block(input_tensor, ratio=16):
    channel_axis = -1
    filters = input_tensor.shape[channel_axis]

    # Squeeze: global average pooling
    se = GlobalAveragePooling2D()(input_tensor)
    se = Reshape((1, 1, filters))(se)

    # Excitation: Dense layers
    se = Dense(filters // ratio, activation='relu')(se)
    se = Dense(filters, activation='sigmoid')(se)

    # Scale: element-wise multiplication with the original input
    se = Multiply()([input_tensor, se])

    return se
```

Steps in the Implementation

1. **Global Average Pooling**: The `GlobalAveragePooling2D()` layer reduces the spatial dimensions of the input tensor, resulting in a 1×1×C1 \times 1 \times C1×1×C tensor.
2. **Reshape**: This tensor is reshaped to facilitate channel-wise operations.
3. **Dense Layers**: Two Dense layers apply the squeeze and excitation operations, using the specified ratio to control the number of channels.
4. **Multiplication**: The original input tensor is multiplied by the weights generated in the excitation step, producing the recalibrated feature map.

Significance of SENet

- **Enhanced Feature Representation**: By learning the importance of channels adaptively, SENet can improve the model's ability to focus on informative features while disregarding less relevant ones.
- **Improved Performance**: Integrating SENet in the model typically leads to better accuracy and robustness, especially in tasks like image classification, where distinguishing between classes is critical.
- **Versatility**: SENet can be applied to various architectures (like ResNet, Inception) without substantial modifications, making it a powerful addition to many CNNs.