1.facenet

Methodology of the Code:

1. Objective:

The goal of this code is to fine-tune a pre-trained FaceNet model for face recognition using a custom dataset, and classify faces into a set number of classes (10 in this case). The dataset is augmented using data augmentation techniques to improve the model's generalization.

---

2. Libraries Used:

- `tensorflow`, `keras`: For building and training the neural network.
- `keras_facenet`: Pre-trained FaceNet model for facial embeddings.
- `ImageDataGenerator`: For image preprocessing and augmentation.
- `Adam`: Optimizer for training the neural network.

---

3. Data Preprocessing and Augmentation:

The `ImageDataGenerator` class is used to perform data augmentation on the images before feeding them into the model.

- **Rescaling**: Pixel values are rescaled to the range [0, 1] by dividing by 255.
- **Rotation**: Images are randomly rotated by up to 30 degrees.
- **Shifting**: Images are randomly shifted horizontally and vertically by 20%.
- **Shearing and Zooming**: Shearing and zooming are applied to augment the image variety.
- **Horizontal Flip**: Images are randomly flipped horizontally.
- **Validation Split**: 20% of the data is used for validation.

The training and validation data are generated by calling the `.flow_from_directory()` method, which loads images from a directory and applies the augmentation.

---

4. Pre-trained FaceNet Model:

The code uses the `FaceNet` model, which is pre-trained on a large facial recognition dataset to generate embeddings for each face.

- **Input Shape**: FaceNet expects images of size `(160, 160, 3)`. Hence, the images are resized to this shape.

- The embeddings generated by FaceNet are used as input features for classification.

---

5. Custom Neural Network Layers:

On top of the FaceNet embeddings, the code adds custom layers for the classification task:

- **Flatten Layer**: Converts the 3D FaceNet embeddings into a 1D feature vector.
- **Dense Layer**: A fully connected layer with 128 units and ReLU activation. It introduces non-linearity to the network and helps in learning complex patterns.
- **Dropout Layer**: A dropout layer with a dropout rate of 50% is added to reduce overfitting by randomly dropping nodes during training.
- **Output Layer**: A dense layer with `NUM_CLASSES` units and a softmax activation function to classify the input image into one of the face classes.

---

6. Model Compilation:

The model is compiled using the Adam optimizer with a learning rate of 0.0001.

- **Loss Function**: Categorical cross-entropy, as this is a multi-class classification problem.
- **Metrics**: Accuracy is used to measure the performance of the model during training.

---

7. Model Training:

The model is trained using the `.fit()` method.

- **Epochs**: The model is trained for 20 epochs.
- **Batch Size**: 32 images per batch.
- **Steps per Epoch**: Defined by the number of samples in the training set divided by the batch size.
- **Validation Data**: The validation generator is used to evaluate the model after each epoch.
- **Verbose**: The training progress is printed to the console after each epoch.

Summary of the Methodology:

- The code leverages the pre-trained FaceNet model for extracting facial embeddings and builds a custom classification head to classify the faces into different classes. Data augmentation techniques are used to improve generalization and avoid overfitting. The model is compiled with the Adam optimizer and trained using augmented data for a fixed number of epochs.

2.VGGface2 using resnet

Methodology of the Code:

1. Objective:

The code aims to fine-tune a pre-trained **ResNet50** model for face recognition, utilizing a custom dataset of images. The model performs classification on faces into a set number of classes (10 in this case). The dataset is augmented using several image augmentation techniques to improve generalization and prevent overfitting.

---

2. Libraries Used:

- `tensorflow`, `keras`: For building and training the neural network.
- `ResNet50`: A pre-trained model used as a feature extractor.
- `ImageDataGenerator`: For image preprocessing and augmentation.
- `Adam`: Optimizer for model training.
- `l2`: Regularization technique to reduce overfitting by penalizing large weights.

---

3. Data Preprocessing and Augmentation:

The code uses the `ImageDataGenerator` class to perform data augmentation:

- **Rescaling**: Pixel values are normalized by scaling them from [0, 255] to [0, 1].
- **Rotation**: Images are randomly rotated up to 45 degrees.
- **Shifting**: Images are randomly shifted horizontally and vertically by 20%.
- **Shearing and Zooming**: Shear and zoom ranges are applied to make the model more robust to variations.
- **Brightness Variation**: Brightness is randomly adjusted between 80% and 120%.
- **Horizontal Flip**: Random horizontal flipping of the images is applied.
- **Validation Split**: 20% of the dataset is set aside for validation.

The `flow_from_directory()` method is used to load images from the directory and apply augmentation. The training and validation datasets are generated separately by specifying a subset (`training` or `validation`).

---

4. Pre-trained ResNet50 Model:

The ResNet50 model, pre-trained on the ImageNet dataset, is used for feature extraction. The model includes the convolutional layers (`include_top=False`), but excludes the final fully connected layers so that a custom classification head can be added.

- **Input Shape**: ResNet50 expects input images of size `(224, 224, 3)` which is why the target size is set to `(224, 224)`.

Freezing Layers: The last 10 layers of ResNet50 are set to be trainable, while the earlier layers are frozen. This allows fine-tuning on the dataset without losing the benefits of the pre-trained layers.

---

5. Custom Neural Network Layers:

On top of the ResNet50 model, custom fully connected layers are added for classification:

- **Flatten Layer**: Converts the multi-dimensional output from ResNet50 to a 1D feature vector.
- **Dense Layer**: A fully connected layer with 128 units and ReLU activation is added. L2 regularization is applied to prevent overfitting by penalizing large weights.
- **Dropout Layer**: Dropout with a probability of 0.6 is applied to randomly drop nodes during training to reduce overfitting.
- **Output Layer**: A fully connected layer with `NUM_CLASSES` units and softmax activation for multi-class classification.

---

6. Model Compilation:

The model is compiled with the following settings:

- **Optimizer**: Adam optimizer is used with a small learning rate of `0.00001` to allow fine-tuning the model without drastic weight updates.
- **Loss Function**: Categorical cross-entropy is used as the loss function, appropriate for multi-class classification tasks.
- **Metrics**: Accuracy is used to monitor the performance of the model during training.

---

7. Model Training:

The model is trained using the `.fit()` method:

- **Epochs**: The model is trained for 20 epochs.
- **Batch Size**: 32 images per batch.
- **Steps per Epoch**: The number of steps is calculated as the total number of samples in the training set divided by the batch size.
- **Validation Data**: Validation data is provided to evaluate the model's performance after each epoch.
- **Verbose**: The training process outputs details for each epoch to the console.

---

8. Model Saving:

After training, the trained model is saved to a specified directory (`/content/drive/MyDrive/models_face_recognition/restnet50_face_recognition.h5`) in HDF5 format for later use.

Summary of the Methodology:

This code uses a pre-trained **ResNet50** model as a feature extractor for facial recognition and fine-tunes the last 10 layers of ResNet50 for better accuracy on the new dataset. Custom fully connected layers are added to perform classification on 10 different classes, and the model is trained using augmented data with data augmentation techniques to improve generalization. L2 regularization and dropout are applied to reduce overfitting. Finally, the trained model is saved for future use.

3.ARCface

The project consists of several distinct stages, including the implementation of an **ArcFace loss function**, a **data generator**, a **model architecture using ResNet50**, a **training process**, and a **method to visualize predictions**. Here's a detailed breakdown of each component:

1. ArcFace Loss Function

- **Purpose**: The ArcFace loss function is used to optimise the deep learning model for face recognition. It ensures that the embeddings of face images from the same person are closer, while the embeddings of images from different people are pushed apart.
- **Parameters**:
  - `y_true`: True labels for the input data.
  - `y_pred`: The predicted embeddings (features) by the model.
  - `margin`: A margin added to the angle between embeddings to make the decision boundary more robust.
  - `scale`: A scaling factor applied to increase the influence of the margin on the final output.
- **Operation**:
  - Clips the predicted values to ensure they stay within the valid range of cosine values.
  - Computes the angular distance (theta) between embeddings using the inverse cosine function (`tf.acos`).
  - Adjusts the target logits by adding the margin, making correct class predictions more challenging.

    - Computes the softmax cross-entropy loss on modified logits to train the model to minimise intra-class variance and maximise inter-class variance.

2. Data Generator

- **Purpose**: The data generator is used to efficiently load and preprocess image data for training.
- **Operation**:
    - The `ImageDataGenerator` applies preprocessing (in this case, the preprocessing function for ResNet50) to images, ensuring they are correctly normalised.
    - `flow_from_directory` is used to load image data from a directory, resizing images to the specified target size (224x224), and generating batches for training.
- **Parameters**:
    - `data_path`: Directory path where the training images are stored.
    - `img_height`, `img_width`: Target dimensions to resize images.
    - `batch_size`: Number of samples per batch.

3. Model Architecture

- **Purpose**: Build a deep learning model based on the **ResNet50** architecture for face recognition.
- **Architecture**:
    - **ResNet50**: A pre-trained model on ImageNet, used here without its final classification layers (`include_top=False`).
    - The output from ResNet50 is passed through a **Global Average Pooling** layer, followed by a dense layer of size 512.
    - **ArcFace-Specific Layer**: The output is normalised with `tf.nn.l2_normalize`, which ensures the embeddings lie on a unit hypersphere, as required by ArcFace loss.
    - Finally, a fully connected softmax layer is added for classification based on the number of classes in the dataset.
- **Parameters**:
    - `input_shape`: Shape of the input images (224, 224, 3).
    - `num_classes`: Number of classes in the classification task.

4. Training Process

- **Purpose**: Trains the model using the ArcFace loss function.
- **Steps**:
    - The model is compiled using the Adam optimizer, ArcFace loss function, and accuracy as a metric.
    - The model is trained using the training data generator (`train_gen`), for the specified number of epochs (10 in this case).
    - Each epoch processes the entire training dataset with the number of steps being the total number of images divided by the batch size.

- **Parameters**:
    - `epochs`: Number of training iterations.

5. Model Evaluation and Accuracy

- **Purpose**: Evaluate the trained model on the dataset and print the accuracy.
- **Steps**:
    - The `model.evaluate` method is used to calculate the loss and accuracy on the data generator.
    - Prints the accuracy after evaluation.

6. Prediction and Visualization

- **Purpose**: Randomly selects and visualises predictions made by the model, alongside the actual class labels.
- **Steps**:
    - A random set of images is selected from the data generator.
    - The model predicts class probabilities for the selected images.
    - The images are displayed in a grid (3x3), with each image showing:
        - The actual class label.
        - The predicted class label.
        - The model's confidence in its prediction (as a percentage).
- **Parameters**:
    - `generator`: The data generator used for input images.
    - `model`: The trained model used for making predictions.
    - `num_images`: Number of random images to display (default is 9).

Summary of the Approach

1. **ArcFace Loss** optimises the model to produce discriminative embeddings for face recognition.
2. **Data Generation** efficiently loads and preprocesses the image data.
3. **Model Architecture** uses ResNet50 as a feature extractor, customised with an ArcFace-specific layer to normalise embeddings.
4. **Training** is performed using the ArcFace loss function to enhance classification performance.
5. **Evaluation** gives accuracy, while **visualisation** helps in analysing the model's predictions visually.

4.SEnet

1. Data Loading and Preprocessing

- **Objective**: Load face image data from a dataset directory, preprocess the images, and prepare the labels for training and evaluation.

- **Steps**:
  - **Load Images**:
    - The dataset path is specified, and the folders within represent different face classes.
    - Images are loaded using `cv2.imread` and resized to the target size of 224x224 pixels (as required by ResNet50).
  - **Store Data**:
    - Images are stored in a `data` list and the corresponding class labels (indices) in the `labels` list.
  - **Normalisation**:
    - Convert the list of images into a NumPy array and normalise the pixel values to [0, 1] by dividing by 255.0.
  - **One-hot Encoding**:
    - The `labels` are one-hot encoded using `to_categorical`, making it suitable for classification tasks.
  - **Train-Test Split**:
    - Data is split into training and testing sets (80% for training and 20% for testing) using `train_test_split` from Scikit-learn.

2. Squeeze-and-Excitation (SE) Block

- **Objective**: Implement a Squeeze-and-Excitation (SE) block to enhance the representational power of the convolutional neural network (CNN).
- **Explanation**:
  - **Squeeze**: A global average pooling layer is applied to aggregate the spatial information of each channel into a single value (creating a descriptor for each channel).
  - **Excitation**:
    - A fully connected network (two dense layers) is used to learn the relationships between channels.
    - The first dense layer reduces the number of channels by a factor of `ratio` (16 by default) and applies a ReLU activation.
    - The second dense layer restores the number of channels and applies a sigmoid activation to create a set of channel weights.
  - **Scale**: The input tensor is multiplied element-wise by the learned channel weights, allowing the model to focus on more informative channels.
  - **Purpose**: The SE block selectively weights channels to improve feature discrimination.

3. Model Architecture with ResNet50 and SE Block

- **Objective**: Build a face recognition model based on ResNet50 with the inclusion of the Squeeze-and-Excitation (SE) block.
- **Explanation**:
  - **Base Model**: ResNet50 (pre-trained on ImageNet) is loaded without its classification head (`include_top=False`), which allows the use of ResNet50 as a feature extractor.

    - **SE Block**: The SE block is applied to the output of ResNet50 to enhance the learned features.
    - **Classification Layers**:
        - The output from the SE block is passed through a **Global Average Pooling** layer, which reduces the spatial dimensions and retains only the most important features.
        - Dense layers with ReLU activation and dropout are added to prevent overfitting.
        - A final softmax layer is added for classification with the number of output neurons equal to the number of face classes (`num_classes`).

4. Model Compilation and Training

- **Objective**: Compile and train the face recognition model on the preprocessed dataset.
- **Steps**:
    - **Compilation**:
        - The model is compiled using the Adam optimizer and categorical cross-entropy as the loss function, which is suitable for multi-class classification.
        - Accuracy is used as the evaluation metric.
    - **Training**:
        - The model is trained for 10 epochs on the training data (`X_train` and `y_train`) with a batch size of 32.
        - Validation data (`X_test` and `y_test`) is used to monitor the model's performance during training.
    - The training history is stored in `history` for later use (e.g., plotting training accuracy and loss).

5. Model Evaluation

- **Objective**: Evaluate the performance of the trained model on the test set and print the test loss and accuracy.
- **Steps**:
    - **Evaluation**: The `model.evaluate` function is used to compute the test loss and accuracy.
    - The results are printed out for interpretation.

6. Prediction and Visualization

- **Objective**: Visualise the model's predictions on a set of randomly selected test images.
- **Steps**:
    - **Select Random Images**: Nine random indices are selected from the test set using `np.random.choice`.
    - **Make Predictions**: The model predicts class probabilities for these selected images.
    - **Display Results**:

- The actual and predicted class labels are displayed on a 3x3 grid using `matplotlib`.
- The images are displayed with titles showing the true class (actual label) and the predicted class from the model.
- This provides a visual inspection of how well the model performs on unseen data.

Summary of the Methodology

1. **Data Loading**: The dataset is loaded from disk, preprocessed, and split into training and test sets. The images are normalised, and labels are one-hot encoded.
2. **Squeeze-and-Excitation Block**: The SE block enhances the feature maps by selectively emphasising more informative channels.
3. **Model Building**: ResNet50 is used as the base model, followed by the SE block and additional dense layers for classification.
4. **Training**: The model is trained using the Adam optimizer and categorical cross-entropy loss.
5. **Evaluation**: The model is evaluated for performance on the test set in terms of accuracy and loss.
6. **Prediction Visualisation**: A subset of predictions is displayed along with actual labels, providing insight into the model's classification capabilities.