Class note

RESnet50

1. Importing Libraries

```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
import os
```

- **TensorFlow & Keras**: The main libraries used for building and training the neural network.
- **ResNet50**: A pre-trained model on ImageNet, commonly used for image classification tasks.
- **Dense, Flatten, Dropout**: Keras layers used to construct the new model.
- **ImageDataGenerator**: A utility for augmenting image data in real-time during training.
- **Adam**: An optimizer used to update model weights.
- **l2**: A regularization technique to prevent overfitting.

2. Setting Parameters

```
BATCH_SIZE = 32
IMAGE_SIZE = (224, 224)  # VGG16 input size
TRAIN_DIR = '/content/drive/MyDrive/test_vggface'  # Training dataset directory
NUM_CLASSES = 10  # Number of classes (adjust this to your number of face classes)
```

- **BATCH_SIZE**: Number of samples processed before the model is updated.
- **IMAGE_SIZE**: The input size for the model, which must match the dimensions required by ResNet50 (224x224 pixels).
- **TRAIN_DIR**: Directory containing the training dataset.
- **NUM_CLASSES**: Number of distinct classes (faces) in the dataset.

3. Data Augmentation and Generators

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=45,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.4,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],  # Brightness variation between 80% and 120%
    fill_mode='nearest',
    validation_split=0.2  # Use 20% of the data for validation
)
```

- **ImageDataGenerator**: Configures real-time data augmentation:
  - **rescale**: Normalizes pixel values to the range [0, 1].
  - **rotation_range, width_shift_range, height_shift_range**: Random transformations to increase dataset variability.
  - **zoom_range, horizontal_flip, brightness_range**: Additional augmentations for better generalization.
  - **validation_split**: Splits the dataset into training and validation sets.

4. Creating Data Generators

```
train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'  # Use this for training set
)

val_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'  # Use this for validation set
)
```

- **flow_from_directory**: Loads images directly from the specified directory:
  - **target_size**: Resizes images to match the input size of the model.
  - **class_mode**: Set to 'categorical' for multi-class classification.
  - **subset**: Specifies whether the data generator is for training or validation.

5. Loading the Base Model

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

- Loads the ResNet50 model without the top layer (classification head) to use it as a feature extractor.

6. Freezing Layers in the Base Model

```
for layer in base_model.layers[-10:]:
    layer.trainable = True
```

- Freezes all layers except the last 10. This allows for fine-tuning of the top layers while retaining the pre-trained weights of the base model.

7. Constructing the New Model

```
x = Flatten()(base_model.output)  # Flatten the output layer to 1D
x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)  # L2
regularization
x = Dropout(0.6)(x)  # Dropout to reduce overfitting
output = Dense(NUM_CLASSES, activation='softmax')(x)  # Output layer for
classification
```

- **Flatten**: Converts the 2D feature maps into a 1D vector.
- **Dense(128)**: Fully connected layer with 128 neurons and ReLU activation. Uses L2 regularization to mitigate overfitting.
- **Dropout**: Randomly drops 60% of the neurons to further prevent overfitting.
- **Dense(NUM_CLASSES)**: Final output layer with softmax activation for multi-class classification.

8. Compiling the Model

```
model.compile(optimizer=Adam(learning_rate=0.00001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

- Compiles the model using the Adam optimizer, categorical cross-entropy loss function, and tracks accuracy during training.

9. Training the Model

```
EPOCHS = 20  # You can adjust this based on your hardware and needs

history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_data=val_generator,
    validation_steps=val_generator.samples // BATCH_SIZE,
```

```
    epochs=EPOCHS,
    verbose=1
)
```

- **fit**: Trains the model using the training generator for a specified number of epochs.
- **steps_per_epoch**: Number of batches to draw from the training set.
- **validation_steps**: Number of batches for validation.

10. Saving the Model

```
model.save('/content/drive/MyDrive/models_face_recognition/restnet50_face_recognition.h5')
```

- Saves the trained model to the specified path for later use.

11. Evaluating the Model

```
# Evaluate the model on training data
loss, accuracy = model.evaluate(train_generator)
print(f'Training Accuracy: {accuracy * 100:.2f}%')

# Evaluate the model on validation data
val_loss, val_accuracy = model.evaluate(val_generator)
print(f'Validation Accuracy: {val_accuracy * 100:.2f}%')
```

- **evaluate**: Tests the model on the training and validation datasets and prints the accuracy.

12. Random Predictions

```
def predict_random_images(generator, model, num_images=9):
    # Get class indices and names
    class_indices = generator.class_indices
    class_names = list(class_indices.keys())  # Class names in the order of their indices
    total_batches = len(generator)  # Total number of batches
    num_batches = (num_images + generator.batch_size - 1) // generator.batch_size # Number of batches needed

    # Randomly select batch indices
    random_batches = random.sample(range(total_batches), num_batches)

    images = []
    actual_labels = []
    predicted_classes = []
    confidences = []
```

```
    for batch_idx in random_batches:
        # Get the batch of images and labels
        batch_images, batch_labels = generator[batch_idx]

        # Make predictions for the batch
        predictions = model.predict(batch_images)

        # Loop through each image in the batch
        for i in range(len(batch_images)):
            if len(images) >= num_images:
                break  # Stop if we have enough images
            images.append(batch_images[i])
            actual_labels.append(batch_labels[i])
            predicted_class = np.argmax(predictions[i])
            predicted_classes.append(predicted_class)
            confidences.append(np.max(predictions[i]) * 100)  # Get confidence
score

    # Plot the images with their actual and predicted labels
    plt.figure(figsize=(12, 12))
    for i in range(num_images):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])  # Display the image
        plt.title(f"Actual: {class_names[np.argmax(actual_labels[i])]} \n"  #
Getting the actual class name
                  f"Predicted: {class_names[predicted_classes[i]]}\n"
                  f"Confidence: {confidences[i]:.2f}%")
        plt.axis("off")
    plt.show()

# Call the function to predict 9 random images from the validation generator
predict_random_images(val_generator, model, num_images=9)
```

- **predict_random_images**: A function to randomly select images from the validation generator, make predictions, and visualize them alongside their actual labels and confidence scores.

Summary

The code effectively builds, trains, and evaluates a face recognition model based on the ResNet50 architecture, incorporating data augmentation, dropout for regularization, and L2 regularization to improve generalization. The training process involves real-time image processing for better model robustness, and the ability to visualize predictions helps in assessing model performance effectively.

ResNet50 is a specific variant of the ResNet architecture that contains 50 layers. It is widely used in various computer vision tasks, particularly image classification, due to its balance between depth and computational efficiency. Here's a detailed explanation of its structure, components, and applications:

Overview of ResNet50

1. **Architecture**:
    - **Input Layer**: Accepts input images typically sized at 224x224 pixels with three color channels (RGB).
    - **Convolutional Layers**: The architecture starts with a single convolutional layer followed by several residual blocks.
    - **Residual Blocks**: Each block contains convolutional layers and skip connections that facilitate the flow of gradients during backpropagation.
    - **Global Average Pooling**: Instead of traditional fully connected layers, ResNet50 uses global average pooling before the final classification layer.
    - **Output Layer**: A softmax activation layer provides probabilities for each class in multi-class classification tasks.

Key Components of ResNet50

1. **Convolutional Layers**:
    - The first layer is a 7x7 convolution followed by batch normalization and ReLU activation, which reduces the spatial dimensions of the input.
2. **Max Pooling**:
    - After the initial convolution, a 3x3 max pooling layer is applied to downsample the feature maps.
3. **Residual Blocks**:
    - ResNet50 consists of 16 residual blocks, each containing two 3x3 convolutional layers (Basic Block) with ReLU activation. The output of these layers is added to the input (the shortcut connection).
    - The shortcut connections help the model learn residual mappings rather than direct mappings, which alleviates the vanishing gradient problem in deep networks.
4. **Bottleneck Architecture**:
    - To optimize computational efficiency, ResNet50 uses a bottleneck design within its residual blocks, which includes:
        - A 1x1 convolution to reduce dimensionality.
        - A 3x3 convolution for feature extraction.
        - Another 1x1 convolution to restore the original dimensionality.
5. **Global Average Pooling Layer**:
    - Instead of fully connected layers, ResNet50 uses a global average pooling layer to reduce the spatial dimensions of the feature maps to a single value per feature, minimizing overfitting.
6. **Final Dense Layer**:
    - The output layer uses softmax activation to predict the class probabilities.

Layer Breakdown of ResNet50

- **Total Layers**: 50 layers, including convolutional, pooling, and fully connected layers.
- **Layer Configuration**:
    - 1 Conv layer (7x7)
    - 1 Max Pooling layer (3x3)
    - 16 Residual Blocks (composed of 2 or 3 convolutional layers)
    - 1 Global Average Pooling layer
    - 1 Dense layer (softmax for classification)

Applications

- **Image Classification**: Used extensively in large datasets like ImageNet.
- **Feature Extraction**: The architecture serves as a backbone for various computer vision tasks, including object detection and image segmentation.
- **Transfer Learning**: ResNet50 is often used as a pre-trained model on various datasets, allowing researchers and practitioners to fine-tune it for specific tasks.

Advantages of ResNet50

1. **Easier Training of Deep Networks**:
    - Residual connections enable the training of deep networks without suffering from the vanishing gradient problem.
2. **High Performance**:
    - The architecture achieves state-of-the-art results on various benchmarks due to its depth and effective use of residual connections.
3. **Reduced Overfitting**:
    - The use of global average pooling helps reduce the risk of overfitting compared to traditional fully connected layers.
4. **Flexibility**:
    - ResNet50 can be fine-tuned and adapted to a wide range of applications beyond image classification.

Conclusion

ResNet50 represents a significant advancement in deep learning architectures by enabling the effective training of very deep networks. Its design principles, particularly the use of residual connections, have influenced the development of subsequent architectures in the field. As a versatile model, ResNet50 continues to be widely adopted in both research and industry applications for image classification and other vision tasks.