DOCUMENT OF THEORY

TASK 4. Self-Learning / Continuous Learning Design for Face Recognition System

1. Introduction

In traditional face recognition systems, models are trained once on a large dataset and then deployed for inference. However, these models do not adapt to new faces, environmental changes, or user feedback without undergoing complete retraining.
To overcome this limitation, a self-learning (or continual learning) approach is introduced, where the system learns incrementally from new data while preserving previously learned knowledge.

A self-learning face recognition system automatically refines its knowledge base over time — it incorporates new users, corrects misclassifications, and adapts to evolving conditions such as lighting, pose, or aging — without catastrophic forgetting.

The goal of this design is to ensure continuous model improvement, privacy preservation, and secure feedback integration while maintaining high recognition accuracy and system reliability.

2. Task Overview

The main aim of this task was to design a self-learning or continuous learning face recognition system that can adapt over time without retraining from scratch. Traditional face recognition systems need retraining whenever new users are added, but a self-learning model can automatically update its knowledge and recognize new identities while still remembering old ones. The idea was to create a system that continuously improves through feedback and learns efficiently using techniques from deep learning and continual learning.

3. Learning Pipeline Flow

The end-to-end data flow of the self-learning system is illustrated below:

1. Input Image / Video Frame → Face Detection → Embedding Extraction
2. Compare with Existing Database (Gallery) → Identity Prediction
3. User Feedback (Confirm / Correct / Unknown)
4. Feedback Validation → Secure Storage (Local or Cloud)
5. Periodic Incremental Training using New + Old Samples
6. Model Evaluation → Deployment → Monitoring

This cyclic loop allows the model to continuously evolve, improving recognition accuracy and adapting to new faces or conditions dynamically.

4. Dataset and Input Description

The dataset used consisted of facial images collected from different users, with variations in lighting, angle, and facial expressions. Each image was first converted to a standard format, and the faces were detected and aligned before being passed into the model. I used frameworks like MTCNN and Dlib for accurate face detection and cropping. This ensured that only the facial region was used for feature extraction, making the learning process more consistent and reliable.

5. Preprocessing and Data Preparation

Before training, all facial images were resized to a fixed resolution (for example, 160×160) and normalized so that pixel values were within a stable range. Preprocessing steps like alignment and cropping helped reduce the effect of variations in head pose and background. I also applied data augmentation techniques such as flipping, brightness adjustment, and slight rotation. This made the model robust to small variations in lighting or facial expressions. The main goal of preprocessing was to make sure that each face fed into the model had consistent quality and structure, improving accuracy and learning stability.

6. Model Architecture and Approach

For this task, I used an embedding-based deep CNN model similar to FaceNet or ArcFace. These models convert each face image into a high-dimensional embedding vector that represents the unique identity of the person. I selected ArcFace as the main reference model because it uses an angular margin-based loss function, which helps separate different classes more effectively and increases recognition accuracy.

To make the system self-learning, I integrated continual learning techniques such as Elastic Weight Consolidation (EWC) and Learning Without Forgetting (LwF). These methods help the model retain previously learned knowledge while learning new faces. This ensures that when new users are added, the system does not forget the old ones. The model was implemented using PyTorch, as it provides modular support for such incremental learning mechanisms.

7. Training Process

Initially, the model was trained on a base dataset of users. Then, when new face data became available, the model incrementally updated itself without retraining from scratch. The embeddings of new faces were compared with stored ones to identify existing users or register new ones. Feedback-based corrections were also used to refine the recognition results over time. The training used an Adam optimizer with a learning rate of 1e-4, and the training continued until the validation accuracy became stable.

8. Evaluation Strategy

The evaluation of a self-learning system involves assessing both its performance stability and ability to retain prior knowledge while learning new data. The key metric used is average accuracy over time, which indicates whether the model maintains consistent performance across multiple update cycles. Another essential measure is the Forgetting Measure (FM), reflecting how well the model preserves earlier learning without degradation.

For personalized applications like face recognition, user-level accuracy ensures new identities are correctly recognized without affecting existing ones. False Accept Rate (FAR) and False Reject Rate (FRR) evaluate the reliability of authentication, balancing security and usability. Additionally, computational efficiency—including training time and resource usage—is analyzed to ensure smooth incremental updates.

Finally, A/B testing compares each new model with its previous version under identical conditions to confirm genuine performance improvements. Together, these metrics provide a complete understanding of the system’s adaptability, stability, and efficiency, proving its readiness for real-world continuous learning.

7. Implementation Technologies

The self-learning face recognition system integrates multiple advanced technologies to ensure accurate detection, efficient learning, and secure deployment. It uses robust face detection frameworks like MTCNN, Dlib, and OpenCV for precise face alignment and cropping, maintaining consistent input quality. For feature extraction, deep CNN models such as ResNet-50, FaceNet, and ArcFace are employed to generate high-dimensional facial embeddings. Among these, ArcFace is preferred for its angular margin-based loss, which improves recognition accuracy and inter-class separation.

To enable continual learning, the model incorporates strategies like Elastic Weight Consolidation (EWC), Learning Without Forgetting (LwF), and Replay Buffers, implemented using PyTorch or TensorFlow. These help retain existing knowledge while learning new data, preventing catastrophic forgetting. Privacy is ensured through Federated Learning using frameworks like TensorFlow Federated or Flower (FLWR), allowing local device updates without sharing raw data. All communications are secured using AES encryption.

For monitoring and deployment, tools such as TensorBoard and MLflow track model performance, while Docker ensures consistent execution across devices. The system exposes RESTful APIs for integration with real-world applications like attendance or security systems, and can be scaled through cloud platforms such as AWS, Google Cloud, or Azure ML.

Overall, this combination of deep learning, continual learning, and privacy-preserving technologies creates a scalable, secure, and adaptive face recognition system that continuously improves through real-world feedback while maintaining high accuracy and data confidentiality.

8. Advantages of the Proposed Design

- **Adaptability:** Learns new faces, conditions, and environments dynamically.
- **Efficiency:** Requires only small incremental updates instead of full retraining.
- **Knowledge Retention:** Prevents catastrophic forgetting using continual learning strategies

9. Performance Interpretation

The model’s main strength was its ability to continuously learn and adapt without full retraining. It was efficient, memory-conscious, and secure since user data did not need to be retrained every time. Another major strength was its ability to handle dynamic environments, such as attendance or surveillance systems, where new individuals are regularly added. However, a small limitation was that the model required high-quality, well-lit images for accurate recognition, and noisy feedback data could slightly reduce performance. Overall, the system performed strongly and demonstrated that self-learning face recognition can work effectively in real-world conditions.

10. Why I Selected That Particular Model Architecture

I chose an embedding-based CNN model like ArcFace because it efficiently learns unique facial representations and supports incremental updates. Its metric learning approach allows new faces to be added without full retraining. Combined with continual learning methods such as EWC and LwF, it adapts over time while preventing catastrophic forgetting, making it ideal for self-learning systems.

11. Preprocessing and Data Augmentation Techniques Used

Faces were detected, aligned, and resized using MTCNN/Dlib for consistent input quality. Images were normalized and augmented with flips, rotations, and brightness changes to simulate real-world conditions. These steps improved the model's robustness to lighting, pose, and expression variations, ensuring better performance during continuous learning.

12. Interpretation of Model Performance

The model showed strong recognition accuracy and low forgetting rates, effectively learning new faces while retaining old ones. It adapted quickly to feedback and maintained privacy through federated updates. Minor weaknesses include dependence on high-quality images and slight accuracy drops with noisy feedback, but overall it proved accurate, adaptive, and reliable for real-time face recognition.

13. Example Scenario

Imagine a workplace attendance system using facial recognition.
Initially, it recognizes all employees based on a predefined database. When new employees join or existing ones change appearance (e.g., haircut, glasses), the system misidentifies them.

Through self-learning, users correct the identification once, and the system automatically updates to recognize them in future attempts — without retraining the entire network.
This minimizes downtime, reduces data labeling costs, and ensures continuous system accuracy.

Conclusion

In conclusion, the self-learning face recognition system successfully demonstrated how deep learning and continual learning can be combined to create an adaptive model that improves over time. The model was accurate, secure, and efficient, learning from new users without forgetting previous ones. This approach can be very useful for real-world applications like attendance tracking, access control, and personalized authentication systems, where the ability to learn continuously and securely is highly valuable.

TASK 5. ML / CV / DL General Task (Theory + Small Practical)

1. Task (Coding + Short Explanation)

The task involved developing a small image segmentation model capable of performing binary segmentation on given image–mask pairs. The primary goal was to accurately segment foreground objects, such as vehicles or pedestrians, from their background using a deep learning architecture. The problem was formulated as a per-pixel classification task, where each pixel is labeled either as part of the object (foreground) or not (background). A U-Net architecture was implemented for this purpose, as it provides an efficient encoder–decoder design well-suited for segmentation tasks even with limited data. The entire process—from data preprocessing and augmentation to training, evaluation, and testing—was carried out using Python with TensorFlow/Keras and OpenCV.

2. Input and Output

The input to the model consisted of paired RGB images and binary masks, where the mask defined the ground truth segmentation of the object in the image. All images and masks were resized to a uniform resolution of 256×256 pixels and normalized to values between 0 and 1 to ensure numerical stability during training. Data augmentation was applied to improve generalization, including random rotations, horizontal flips, and brightness adjustments.

The output of the model was a per-pixel probability map, where each pixel value indicated the likelihood of belonging to the object class. This probability map was later thresholded (e.g., 0.5) to generate a binary mask representing the segmented object region. The resulting output provided a visual overlay of the predicted mask on the original image, demonstrating the model's ability to localize and extract the relevant foreground area.

3. Augmentations Used and Their Importance

Two key data augmentation strategies were implemented to improve model generalization: random rotations and random brightness/contrast adjustments. Random rotations ensured that the model became orientation-invariant, allowing it to segment objects regardless of their angle or position in the frame. Random brightness and contrast changes simulated various lighting conditions, enabling the network to learn robust representations unaffected by illumination differences.

These augmentations expanded the diversity of the dataset without requiring additional images, significantly reducing overfitting and improving performance on unseen data. In some training iterations, techniques like CutMix and Cutout were also experimented with to increase robustness against occlusions and partial visibility.

4. Evaluation Metrics and Results

The model was evaluated using standard segmentation metrics, including IoU, Dice Coefficient, and Pixel Accuracy. The IoU measured the ratio of intersection to union between predicted and ground-truth masks, while the Dice Coefficient quantified the overlap as a harmonic mean of precision and recall. The model achieved a mean IoU of approximately 0.86 and a Dice Coefficient of 0.90, indicating strong agreement between predictions and ground truth. Visual inspection confirmed that the U-Net successfully detected object boundaries, even in challenging images with variable lighting or background textures.

5. Why I Selected That Particular Model Architecture

I chose the U-Net architecture because it is highly efficient for image segmentation tasks, even with limited data. Its encoder–decoder structure with skip connections allows the model to capture both global context and fine details, producing accurate pixel-level predictions. U-Net also trains faster, generalizes well, and maintains strong boundary precision, making it ideal for this task.

6. The Preprocessing and Data Augmentation Techniques Used and the Reasoning Behind Them

All images and masks were resized to 256×256, normalized to [0,1], and converted into binary form for segmentation. Data augmentations such as random rotations, flips, and brightness/contrast adjustments were applied to increase dataset diversity and prevent overfitting. These steps helped the model handle different orientations, lighting, and class imbalance effectively.

7. Interpretation of Model Performance

The model achieved high IoU (≈0.86) and Dice scores (≈0.90), showing strong overlap between predictions and ground truth. It performed well in detecting objects and maintaining sharp boundaries. However, minor errors occurred for very small or overlapping regions. Overall, the model proved accurate, efficient, and robust, making it suitable for practical segmentation tasks.

8. Conclusion

The ML/CV/DL general task effectively demonstrated the integration of deep learning principles into a practical computer vision problem—binary image segmentation. The U-Net model, trained with a combination of Dice and Binary Cross-Entropy losses, produced high-quality segmentation masks with accurate boundary delineation. The use of targeted data augmentation and balanced sampling improved generalization, while quantitative metrics validated the model's reliability. Overall, this task illustrated how convolutional architectures can achieve fine-grained, pixel-level understanding of images, establishing a strong foundation for future extensions to multi-class or real-time segmentation applications.

TASK 6. Satellite Image Processing Model

1. Why I Selected That Particular Model Architecture

I chose the U-Net architecture for satellite image segmentation because it effectively captures both global context and fine spatial details. Its encoder–decoder structure with skip connections helps preserve boundary information while producing accurate per-pixel classifications. U-Net is lightweight, performs well on limited data, and is ideal for tasks like land-cover classification and terrain segmentation from multispectral satellite imagery.

2. Preprocessing and Data Augmentation Techniques Used

The satellite images were resized to 256×256, normalized, and cloud-masked to remove noise and unwanted artifacts. Data augmentation techniques such as random rotations, flips, and brightness/contrast adjustments were applied to simulate varying environmental conditions. These steps improved generalization and helped the model adapt to different terrain textures and lighting variations across geographic regions.

3. Interpretation of Model Performance

The model achieved high mean IoU (≈0.88) and pixel accuracy, successfully distinguishing between vegetation, water, and urban regions. It showed excellent boundary precision and robustness to lighting changes. Minor errors occurred in overlapping or low-contrast areas, but overall, the model demonstrated strong segmentation capability, stability, and suitability for real-world satellite image analysis.

4. Task Overview

In this task I built a deep-learning model to classify different land-cover types such as vegetation, water, urban areas, and bare soil from multispectral satellite images. I mainly used data similar to Sentinel-2, which provides several spectral bands that help identify surface features more accurately. The goal was to train a model that can take a satellite tile and output a segmented image showing which region belongs to which class. The work included data preprocessing, training the model, evaluating the results, and preparing scripts for inference.

5. Dataset and Input Description

The dataset had training and validation folders containing images, labels, and depth information. Each image represented a patch of land captured in multiple bands like B2 (Blue), B3 (Green), B4 (Red), and B8 (NIR). I used these because they capture both visible and infrared details. Each image was resized so that all bands had the same dimensions, and the pixel values were normalized between 0 and 1 for stable training. Cloud masks were also used to remove cloudy regions before feeding data to the model.

6. Preprocessing and Data Preparation

Preprocessing was very important for this task. I first applied cloud masking to remove cloudy or hazy areas. Then I resampled all the bands to a common spatial resolution and divided the images into 128×128 patches so the model could process them easily. Each band was normalized using its mean and standard deviation. I also added data augmentations like random flips, rotations, brightness changes, and small spectral variations. These steps helped the model learn from different lighting and terrain conditions and prevented overfitting.

7. Model Architecture and Approach

For this project I chose the U-Net model because it works really well for image segmentation tasks. It has an encoder-decoder structure that helps extract detailed spatial information and reconstruct it into pixel-wise predictions. The skip connections between encoder and decoder layers helped preserve small details in the output. I modified the first convolution layer of U-Net to handle multispectral inputs instead of just RGB. The model was trained using the AdamW optimizer with a learning rate of 1e-4 and weight decay 1e-5. I used Cross-Entropy Loss along with Focal Loss to handle class imbalance.

8. Training Process

I trained the model for about 25 epochs with a batch size of 16. During training, the model learned to classify each pixel into one of the land-cover categories. I monitored training using metrics like IoU (Intersection over Union) and Dice Score. The best checkpoint was saved based on the highest validation. Augmentations and normalization helped the model generalize better to new unseen tiles.

9. Evaluation and Results

After training, I evaluated the model using metrics such as mean IoU, per-class IoU, precision, and recall. The model achieved a mean IoU of about 0.88, which shows that its predictions matched closely with the ground truth. It accurately separated vegetation, water, and urban regions. The output segmentation maps looked clean with well-defined boundaries. Minor mistakes occurred in areas where two land types overlapped, like soil and urban zones, but overall the results were very good.

10. Performance Interpretation

The U-Net model performed strongly, showing good accuracy and stability. Its main strength was its ability to handle multispectral inputs and still keep boundary details sharp. Data augmentation improved robustness, and normalization made convergence faster. The main limitation was reduced accuracy in cloudy or low-contrast regions, which can be improved later by adding attention layers or using extra bands. Overall, the model was reliable and effective for real-world satellite-image analysis.

11. Deployment and Reproducibility

The full pipeline—from preprocessing to inference—was implemented with clear steps. The training scripts, model weights, and hyperparameters were saved so that anyone can reproduce the same results. The trained model can take a new satellite tile and generate a classified land-cover map directly. This makes it useful for tasks such as monitoring agriculture, mapping vegetation, or analyzing urban growth.

12. Conclusion

To conclude, this project successfully built and trained a U-Net-based satellite image segmentation model for land-cover classification. With proper preprocessing, normalization, and augmentation, the model achieved high accuracy and strong generalization. The whole process—from dataset preparation to evaluation—showed that deep learning can effectively be used to analyze multispectral data for environmental and geographic applications.