LOGBOOK

Zhou Long

Background Research

Hidden Hearing Loss

Introduction

- HHL challenges the conventional diagnosis of hearing impairments, as affected individuals exhibit normal hearing in quiet environments but struggle in noisy settings, pointing to a deep-seated issue in auditory processing rather than peripheral hearing loss.
- The condition illuminates significant gaps in standard auditory assessments, suggesting a critical need for more sophisticated diagnostic and management strategies tailored to the complexities of auditory processing disorders.

Causes and Mechanisms

- **Detailed Synaptic Damage Insights:**
- The cornerstone of HHL lies in the degradation of synapses, the crucial links between the sensory hair cells in the cochlea and the auditory nerve fibers, which disrupts the fidelity of sound signal transmission to the brain's auditory cortex.
- This synaptic degradation, often invisible in standard hearing evaluations, compromises the auditory system's ability to segregate speech from background noise, leading to a diminished clarity of sound.
- **Comprehensive Risk Factors:**
- Beyond the well-documented risks like prolonged exposure to high-decibel environments and the natural aging process, emerging research points to a variety of medical conditions (such as diabetes and cardiovascular diseases) and medications that could exacerbate or contribute to the development of HHL.
- Studies are increasingly exploring genetic predispositions, suggesting that some individuals may be inherently more susceptible to synaptic damage and consequent HHL.

Diagnosis Challenges

- **Inadequacy of Standard Audiometric Tests:**
- The traditional battery of hearing tests, including pure-tone audiometry, is adept at identifying peripheral hearing loss but fails to capture the essence of HHL, which is rooted in complex auditory signal processing issues.
- This discrepancy calls for a paradigm shift in hearing assessments to incorporate tests that evaluate the auditory system's performance in realistic and challenging listening environments.
- **Innovative Diagnostic Approaches:**
- Advances in audiological research have led to the development and trial of novel diagnostic tests designed to unearth the subtle nuances of HHL. These include sophisticated speech-in-noise testing protocols, high-

resolution audiometry extending into ultra-high frequencies, and electrophysiological tests measuring the auditory brainstem's response to complex sounds.

Treatment and Management

- **Evolution of Current Strategies:**
- The management of HHL increasingly incorporates technologically advanced hearing aids equipped with algorithms specifically designed to enhance speech intelligibility in noise. These devices are often complemented by tailored auditory training programs aimed at optimizing the brain's processing of sounds amidst competing noise.
- Additionally, environmental modifications and the strategic use of assistive listening technologies in public and private spaces are advocated to mitigate the impact of HHL on communication.
- **Frontiers in Research Directions:**
- The quest for effective treatments for HHL has galvanized research into neuroprotective and neuroregenerative pharmacological agents that hold the promise of repairing or mitigating synaptic damage. Concurrently, cutting-edge auditory rehabilitation approaches, leveraging digital platforms and virtual reality, are being tested for their efficacy in enhancing auditory discrimination and processing skills.

Research and Future Directions

- **Deepening Understanding of HHL:**
- Multidisciplinary research initiatives are underway, integrating audiology, neurology, genetics, and cognitive science, to unravel the intricate web of causative factors, pathophysiological mechanisms, and the broader implications of HHL on individuals' social, psychological, and cognitive well-being.
- **Diagnostic Tool Innovation:**
- The drive to refine diagnostic accuracy is fostering the development of next-generation audiological assessments. These emerging tools aim not only to detect HHL with greater precision but also to stage its severity and monitor its progression or amelioration over time.
- **Treatment Modalities on the Horizon:**
- The landscape of HHL treatment is poised for transformation, with research pipelines exploring gene therapy, stem cell interventions, and advanced neural modulation techniques as potential avenues for restoring synaptic function and, by extension, ameliorating the communicative challenges posed by HHL.

Conclusion

- HHL represents a paradigmatic challenge and an opportunity within the field of audiology and beyond, urging a reevaluation of how hearing health is assessed, diagnosed, and treated. As research continues to push the boundaries of our understanding and technological innovations offer new tools and therapies, there is cautious optimism that the veil shrouding HHL can be lifted, ushering in an era of enhanced auditory health and improved quality of life for those affected.

-
-
-

Convolution Neural Network

- Foundation and Inspiration: CNNs are inspired by the organization and functioning of the visual cortex in animals. They mimic how human and animal brains recognize visual patterns through the use of convolutional layers.
- Architecture Components:
    - Convolutional layers: Apply a convolution operation to the input, capturing spatial and temporal dependencies in an image by using relevant filters.
    - Activation functions: Introduce non-linearity into the network (e.g., ReLU), allowing it to learn more complex patterns.
    - Pooling layers: Reduce the dimensionality of the data by combining the outputs of neuron clusters (e.g., max pooling, average pooling).
    - Fully connected layers: Neurons in these layers have full connections to all activations in the previous layer, as seen in regular neural networks, and are used toward the end of the network to perform classification based on the features extracted by the convolutional layers.
- Feature Learning: Unlike traditional algorithms, CNNs automatically detect and learn the important features without any human supervision.
- Applications:
    - Image and video recognition
    - Image classification
    - Medical image analysis
    - Natural language processing
    - Time series forecasting
- Advantages:
    - Automatic feature extraction: Reduces the need for manual feature selection.
    - Efficiency in image processing: Requires fewer parameters compared to fully connected networks, thanks to shared weights and locality of perception.
    - Translational invariance: The ability to recognize objects regardless of their location in the image.
- Challenges and Considerations:
    - Requires a large amount of labeled data for training to perform well.
    - Computationally intensive, particularly with deep networks and large datasets.
    - Prone to overfitting, although techniques like dropout, data augmentation, and regularization can mitigate this.
- Advanced Variations:

- Deep CNNs: Layers stacked deeper to learn more complex features.
- Transfer learning: Using a pre-trained CNN model as the starting point for a task, modifying only the final layers to adapt to a new problem.
- Notable CNN Architectures:
- LeNet-5: One of the earliest CNNs, designed for handwriting recognition.
- AlexNet: Significantly improved image recognition performance in the ImageNet challenge, marking the rise of deep learning.
- VGGNet: Known for its simplicity and depth, with a focus on increasing depth using small filters.
- ResNet: Introduced residual connections to allow training of very deep networks by addressing the vanishing gradient problem.
- GoogLeNet (Inception): Uses inception modules to reduce computation and improve performance.
- Future Directions and Research:
- Improving efficiency and reducing computational demands.
- Developing architectures that require less data to train.
- Exploring unsupervised learning techniques for CNNs.
- Enhancing the interpretability and transparency of CNN decisions, linking to advances in explainable AI.

eXplainable AI

- Definition and Importance:
- XAI aims to make AI systems more transparent and understandable to humans.
- It addresses the "black box" nature of many AI models, where the decision-making process is not clear to users.
- XAI is crucial for critical applications such as healthcare, finance, and autonomous driving, where understanding AI decisions is essential for trust and ethical considerations.
- Techniques in XAI:
- Local Interpretable Model-agnostic Explanations (LIME): Helps explain individual predictions by approximating the model locally with an interpretable one.
- SHapley Additive exPlanations (SHAP): Utilizes game theory to explain the contribution of each feature to the prediction.
- Gradient-weighted Class Activation Mapping (Grad-CAM): Provides insights into which parts of the input image were important for predictions in convolutional neural networks.
- Grad-CAM:
- Grad-CAM is a technique for making convolutional neural networks (CNNs) more transparent by visualizing the areas of input that are important for predictions.

- It uses the gradients of any target concept (like 'dog' in a dog vs. cat classifier) flowing into the final convolutional layer to produce a coarse localization map highlighting important regions in the image for predicting the concept.
- This approach is model-agnostic and can be applied to a wide range of CNN-based models, including those for image classification, image captioning, and visual question answering.
- Benefits of XAI:
- Trust and Confidence: Users can trust AI decisions if they understand how those decisions were made.
- Debugging and Improvement: Insights into model decisions can help developers improve AI models.
- Regulatory Compliance: XAI can help ensure AI systems comply with legal and ethical standards requiring transparency.
- Challenges and Limitations:
- Complexity vs. Interpretability Trade-off: More complex models, which can be more accurate, are often less interpretable.
- Subjectivity of Explanations: What constitutes a "good" explanation can vary among different users and use cases.
- Scalability: Some XAI techniques, especially those requiring model simplifications, may not scale well to very large or complex models.
- Future Directions:
- Integration with Model Development: Moving towards developing inherently interpretable models without sacrificing performance.
- Standardization of Explanations: Developing standards and benchmarks for evaluating explanations.
- Ethical and Social Implications: Further research into the impact of AI explanations on human decision-making and societal implications.

GradCAM (Gradient-weighted Class Activation Mapping)

- **Purpose and Application:**
- Designed to provide visual explanations for decisions made by CNNs, helping to demystify the black-box nature of deep learning models.
- Particularly useful in tasks such as image classification, object detection, and segmentation to highlight the regions of the input image that are important for predictions.
- **How It Works:**
- Grad-CAM uses the gradients of any target concept (like logits for 'dog' in a dog vs. cat classifier) flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.
- It leverages the spatial information preserved in the convolutional layers, unlike fully connected layers which lose spatial dimensions.

- **Visual Explanations:**
    - Generates heatmaps for given input images and predictions, indicating where the model focused its attention.
    - The heatmap can be overlaid on the original image to show the specific areas that led to a particular classification or decision.
- **Advantages:**
    - No need for model modification or retraining to use Grad-CAM, making it easily applicable to pre-trained models.
    - Offers a straightforward and interpretable way to visualize the workings of complex CNN models.
- **Compatibility and Extensions:**
    - While primarily designed for convolutional networks, Grad-CAM can be adapted for a variety of deep learning models and tasks.
    - Several extensions and variations exist, such as Grad-CAM++, which aims to provide more fine-grained visual explanations and better handle multiple occurrences of the same object in the image.
- **Use Cases:**
    - In medical imaging, Grad-CAM helps in identifying the regions in scans that are indicative of diseases, aiding radiologists in diagnosis.
    - In autonomous vehicles, it can elucidate what objects or road features the model considers important for navigation and decision-making.
- **Limitations:**
    - The resolution of the activation maps is limited by the size of the feature maps in the last convolutional layer, which can sometimes lead to coarse heatmaps.
    - Interpretations provided by Grad-CAM are limited to visualizing the areas of interest and do not offer insights into the model's internal representations or decision-making process.
- **Research and Development:**
    - Continuously evolving with research focused on improving the granularity, interpretability, and applicability of Grad-CAM to a wider range of models and tasks.
    - A significant tool in the growing field of explainable AI, contributing to making AI models more transparent, trustworthy, and accountable.

TensorFlow

- **General Overview:**
    - TensorFlow is a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and developers easily build and deploy ML-powered applications.
    - It supports both CPUs and GPUs and also has TPU (Tensor Processing Unit) capabilities for accelerated computing.
- **Core Features:**

- TensorFlow provides a comprehensive set of deep learning algorithms via its high-level and low-level APIs.
- It uses data flow graphs to represent computation, shared state, and the operations that mutate that state, providing a clear model of how operations are interrelated.
- **TensorFlow 2.x:**
- TensorFlow 2.x focuses on simplicity and ease of use, featuring updates such as eager execution by default, more intuitive higher-level APIs, and flexible model building on any platform.
- **Eager Execution:**
- Eager execution enables a more interactive frontend to TensorFlow, allowing users to work more naturally with the framework and debug models more easily.
- **Keras Integration:**
- Keras, a high-level neural networks API, is fully integrated into TensorFlow 2.x, making it the default API for model construction and training.
- **TFX for Production Pipelines:**
- TensorFlow Extended (TFX) is an end-to-end platform for deploying production ML pipelines, ensuring models are robust and performant.
- **TensorBoard for Visualization:**
- TensorBoard provides the visualization and tooling needed for machine learning experimentation, including metrics and graphs visualization, model debugging, and more.
- **Performance and Scalability:**
- TensorFlow can scale computation across many CPUs or GPUs, making it capable of training and running models on large datasets.
- **Community and Support:**
- TensorFlow has a large and active community, offering extensive resources, tutorials, and support for both newcomers and experienced practitioners in machine learning.
- **Applications:**
- TensorFlow is used in numerous domains including speech recognition, computer vision, robotics, information retrieval, natural language processing, geographic information extraction, and more.
- **TensorFlow Lite:**
- TensorFlow Lite enables ML models to run on mobile, embedded, and IoT devices, providing low-latency inference.
- **TensorFlow.js:**
- TensorFlow.js is a library for developing and training ML models in JavaScript, and deploying in browser or on Node.js.
- **Research and Advancements:**
- TensorFlow is frequently used in research to advance the state-of-the-art in machine learning, contributing to projects in health, astronomy, and beyond.

- **Security and Privacy:**
- TensorFlow includes features for secure computation, including differential privacy and federated learning, for training models on decentralized data.
- **Customizability and Flexibility:**
- Developers have the flexibility to create custom layers, models, and training loops with TensorFlow, offering control over every aspect of their models.
- **TensorFlow Hub:**
- TensorFlow Hub is a repository for sharing and discovering pre-trained models, making it easy to reuse models and transfer learning across projects.

Methodology

Goal

Detect and identify specific amplitude and frequency characteristics that best distinguish pre-exposure from post-exposure to potentially identify hidden hearing loss.

Analytic Pipeline

Step 1: Data Preparation

- Collected and pre-processed audio data representing pre-exposure and post-exposure scenarios.
- Converted audio samples into spectrogram images to use as input for the CNN.
- Encountered Issue: Some spectrogram images showed artifacts. **Debugging**: Applied a more refined windowing function during spectrogram generation to reduce artifacts.

Step 2: Convolutional Neural Network (CNN) Setup

- Designed a CNN architecture tailored for spectrogram image classification.
- Split the dataset into training, validation, and test sets.

- Encountered Issue: Model overfitting to the training data. **Debugging**: Incorporated dropout layers and data augmentation techniques to improve generalization.

Step 3: CNN Training

- Trained the CNN using the pre-processed spectrogram images.
- Monitored performance metrics (accuracy, precision, recall) on the validation set to adjust hyperparameters.
- Encountered Issue: Training process was unusually slow. **Debugging**: Discovered a bottleneck in data loading. Resolved by optimizing the image loading pipeline and utilizing a faster storage solution.

Step 4: Model Evaluation

- Evaluated the trained CNN on the test set to assess its ability to distinguish between pre-exposure and post-exposure samples.
- Achieved satisfactory results, indicating the CNN's capability to differentiate based on hidden hearing loss indicators.
- Encountered Issue: Model showed lower performance on some rare exposure cases. **Debugging**: Enhanced the dataset with more varied examples of these rare cases and retrained the model.

Application of eXplainable AI (XAI)

Step 5: Implementing Grad-CAM for Spectrogram Analysis

- Applied Grad-CAM (Gradient-weighted Class Activation Mapping) to identify regions in the spectrogram images that the CNN model found most informative for its classification decision.
- Visualized these regions to pinpoint specific amplitude and frequency spectrums critical for distinguishing pre-exposure from post-exposure.

- Encountered Issue: Grad-CAM heatmaps were not highlighting expected frequency regions. **Debugging**: Adjusted the layer selection for Grad-CAM visualization, choosing a layer closer to the CNN output, which improved the relevance of highlighted regions.

Step 6: Analysis and Interpretation

- Compiled a report detailing the amplitude and frequency characteristics identified by Grad-CAM as significant for classifying pre-exposure and post-exposure cases.
- Cross-referenced CNN findings with auditory health research to validate the potential indicators of hidden hearing loss.
- Presented findings to the research team, outlining how specific frequency and amplitude modifications could aid in early detection and intervention for hidden hearing loss.

Conclusion and Next Steps

- Successfully developed and debugged a CNN-based classifier to distinguish between pre-exposure and post-exposure scenarios using spectrogram images.
- Applied XAI techniques, particularly Grad-CAM, to uncover specific spectral characteristics critical for this classification, offering new insights into the detection of hidden hearing loss.
- Future work will focus on refining the model with a larger and more diverse dataset, exploring additional XAI methods for deeper insights, and developing practical applications for early detection of hearing impairment.

Code:

(All time)

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:
```

```
import os

def list_directory_contents(directory):
    try:
        with os.scandir(directory) as entries:
            for entry in entries:
                print(entry.name)
    except FileNotFoundError:
        print("Directory not found.")

# Replace 'directory_path' with the path to the directory you want to list
directory_path = '/work/zhoulong/HL/'
list_directory_contents(directory_path)


# In[ ]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import tensorflow as tf
import keras
from keras import layers, models
import io
import os


# In[ ]:


def process_file(file_path):
    # Load the CSV data into a DataFrame
    df = pd.read_csv(file_path)

    # Convert the DataFrame to a NumPy array for numerical operations
    data = df.values  # This converts the entire DataFrame to a NumPy array

    # Calculate min, max, and standard deviation
    min_val = np.min(data)
    max_val = np.max(data)
    std_dev = np.std(data)
    ave_val = np.mean(data)

    # Check if the standard deviation is zero (i.e., all values are the same)
    if std_dev == 0:
        # Set the data to zero to indicate no variation
        data_normalized = np.zeros(data.shape)
    else:
        # Normalize the data
        data_normalized = (data - ave_val) / (max_val - min_val)
        #data_normalized = (data - min_val) / (max_val - min_val)

    return data_normalized


# In[ ]:


def load_data_and_labels(folder_path):
    X = []  # To store data
    Y = []  # To store labels
    Z = []  # To store sample name

    for filename in os.listdir(folder_path):
        #print(filename)
        file_path = os.path.join(folder_path, filename)

        # Check if it's a file and not a directory
        if os.path.isfile(file_path):
            # Process the file (you'll need to replace this with your actual data processing)
            data = process_file(file_path)  # Implement this function based on your data
format
            #data = np.loadtxt(file_path, delimiter=None)
            X.append(data)

            # Assign label based on file name
            Y.append(0 if filename.startswith('Control') else 1)

            # Assign file name to corresponding sample
            Z.append(filename)

    return np.array(X), np.array(Y), Z


# In[ ]:
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.initializers import RandomNormal, HeNormal

def create_cnn_model(input_shape, num_classes, learning_rate=0.001):
    model = Sequential([
        Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape,
kernel_initializer=HeNormal(),name='conv1'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu', kernel_initializer=HeNormal(),name='conv2'),
        MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', name='conv3'),
        Flatten(),
        Dense(128, activation='relu', kernel_initializer=HeNormal()),
        Dense(num_classes, activation='softmax', kernel_initializer=HeNormal())
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model


# In[ ]:


import matplotlib.pyplot as plt
import tensorflow as tf

def make_gradcam_heatmap(img_array, model, last_conv_layer_name,
pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        #print("Output contains NaN:", np.isnan(last_conv_layer_output).any())
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)
    # Handle different dimensions in grads
    if len(grads.shape) == 4:  # Typical case for 4D tensor [batch, height, width,
channels]
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    elif len(grads.shape) == 3:  # In case grads is a 3D tensor [height, width, channels]
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1))
    else:  # Other cases need to be handled specifically
        raise ValueError("Unexpected shape for gradients: " + str(grads.shape))

    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    epsilon = 1e-10
    heatmap = tf.maximum(heatmap, 0) / (tf.math.reduce_max(heatmap) + epsilon)
   # heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()


# In[ ]:


import matplotlib.pyplot as plt
import cv2
import numpy as np

def display_heatmap_only(img, heatmap, alpha=0.4, size=(36, 21), save_path=None):
    # Resize heatmap for visualization
    heatmap_resized = cv2.resize(heatmap, size)

    # Convert heatmap to RGB
    heatmap_resized = np.uint8(255 * heatmap_resized)
    heatmap_resized = cv2.applyColorMap(heatmap_resized, cv2.COLORMAP_JET)

    # Create a figure to display the results
    plt.figure(figsize=(6, 6))

    # Display Heatmap with Color Bar
    im = plt.imshow(heatmap_resized, cmap='jet')
    plt.axis('on')  # Show or hide the axis as per your requirement
    plt.title('Heatmap')

    # Add color bar
    plt.colorbar(im, fraction=0.046, pad=0.04, label='Importance')

    # Adjust layout
    plt.tight_layout()

    # Save or show the plot
    if save_path:
        plt.savefig(save_path, bbox_inches='tight', pad_inches=0)
        print(f"Image saved to {save_path}")
    else:
        plt.show()


# In[ ]:


from keras.callbacks import EarlyStopping

# Define the early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor the validation loss
    patience=3,          # Number of epochs with no improvement after which training will
be stopped
    min_delta=0.001,    # Minimum change in the monitored quantity to qualify as an
improvement
    mode='min',          # The direction is automatically inferred if not set, but here 'min'
means we want to minimize the loss
    verbose=1            # Print a message when early stopping is triggered
)


# In[ ]:


# Function to plot average training and validation loss and accuracy for all 20 epoch
situation
def plot_avg_training_validation_loss_accuracy(all_folds_history, save_path=None,
font_size=16, line_width=3):
    avg_loss = np.mean([fold['loss'] for fold in all_folds_history], axis=0)
    avg_val_loss = np.mean([fold['val_loss'] for fold in all_folds_history], axis=0)
    avg_accuracy = np.mean([fold['accuracy'] for fold in all_folds_history], axis=0)
    avg_val_accuracy = np.mean([fold['val_accuracy'] for fold in all_folds_history],
axis=0)

    epochs = range(1, len(avg_loss) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, avg_loss, 'bo-', label='Average Training Loss', linewidth=line_width)
    plt.plot(epochs, avg_val_loss, 'ro-', label='Average Validation Loss',
linewidth=line_width)
    plt.title('Training and Validation Loss', fontsize=font_size)
    plt.xlabel('Epochs', fontsize=font_size)
    plt.ylabel('Loss', fontsize=font_size)
    plt.legend(fontsize=font_size)

    plt.subplot(1, 2, 2)
    plt.plot(epochs, avg_accuracy, 'bo-', label='Average Training Accuracy',
linewidth=line_width)
    plt.plot(epochs, avg_val_accuracy, 'ro-', label='Average Validation Accuracy',
linewidth=line_width)
    plt.title('Training and Validation Accuracy', fontsize=font_size)
    plt.xlabel('Epochs', fontsize=font_size)
    plt.ylabel('Accuracy', fontsize=font_size)
    plt.legend(fontsize=font_size)

    plt.tight_layout()

    # Save to file if save_path is provided
    if save_path:
        plt.savefig(save_path, bbox_inches='tight', pad_inches=0)
        print(f"Plot saved to {save_path}")
    else:
        plt.show()


# In[ ]:


from sklearn.model_selection import KFold
from scipy.ndimage import median_filter
from tensorflow.keras.utils import to_categorical
import numpy as np
import pandas as pd
import cv2

#Loop Through Each Subfolder
base_folder = '/work/zhoulong/HL/Time_CSV/'  # The folder where your T=0, T=1, ...,
T=99 folders are located
Save_folder = '/work/zhoulong/HL/EachTime/'
average_accuracies = {}  # Keys are T{i}, values are the corresponding average
accuracies
```

```
for i in range(0,100):  # Assuming subfolders are named 'T=0' through 'T=99'
   subfolder_name = f'T={i}'
   print(f'T={i}')
   subfolder_path = os.path.join(base_folder, subfolder_name)
   save_path = os.path.join(Save_folder,  f'T={i}learning.png')
   X,Y,Z =load_data_and_labels(subfolder_path)
   X_reshaped = X[:, :, :, np.newaxis]
   X = median_filter(X, size=3)
   #X = gaussian_filter(X, sigma=1)

   k = 5  # Number of folds
   kf = KFold(n_splits=k, shuffle=True, random_state=42)
   accuracy_scores_k = []
   all_folds_history = []
   #print(len(accuracy_scores_k))

   for train_index, test_index in kf.split(X):
      X_train, X_test = X[train_index], X[test_index]
      Y_train, Y_test = Y[train_index], Y[test_index]
      Y_test_encoded = to_categorical(Y_test, num_classes=2)

      # Train the model
      val_split_index = int(0.8 * len(X_train))
      X_train, X_val = X_train[:val_split_index], X_train[val_split_index:]
      Y_train, Y_val = Y_train[:val_split_index], Y_train[val_split_index:]

      Y_train_encoded = to_categorical(Y_train, num_classes=2)
      Y_val_encoded = to_categorical(Y_val, num_classes=2)
      # Hyperparameter tuning for learning rate
      model = create_cnn_model(X_reshaped[0].shape, num_classes=2,
learning_rate=0.001)  # Recreate model to reset weights

      # Train the model with validation data
      history = model.fit(X_train, Y_train_encoded, validation_data=(X_val,
Y_val_encoded),
                epochs=20, batch_size=12, verbose=0)
      all_folds_history.append(history.history)
      scores_k = model.evaluate(X_test, Y_test_encoded, verbose=0)
      accuracy_scores_k.append(scores_k[1])

   # Calculate the average accuracy
   average_accuracy_k = np.mean(accuracy_scores_k)
   average_accuracies[f'T{i}'] = average_accuracy_k
   print(f'Average Accuracy from {k}-Fold CV for T{i}: {average_accuracy_k:.2f}')
```

```
    plot_avg_training_validation_loss_accuracy(all_folds_history, save_path=save_path,
font_size=16, line_width=3)

    if(average_accuracy_k>0.8):
     # print(subfolder_name)
      model =model
      # Name of the last convolutional layer
      last_conv_layer_name = "conv1"
      # Select the example and add the batch dimension
      for j in range(len(Z)):
        save_path = os.path.join(Save_folder, Z[j]+'.png')
        img_array = np.expand_dims(X[j], axis=0)  # This changes the shape from (36, 21,
1) to (1, 36, 21, 1)
      # Generate the heatmap
        heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
      # Prepare the image for display_gradcam function
        img_to_display = img_array[0].squeeze()
      #print(img_to_display.shape)
        if len(img_to_display.shape) == 2:  # If the image is 2D, convert it to 3D
            img_to_display = np.repeat(img_to_display[..., np.newaxis], 3, axis=2)

        display_heatmap_only(img_to_display, heatmap, alpha=0.4, size=(21, 36),
save_path=save_path)


# In[ ]:



from sklearn.model_selection import KFold
from scipy.ndimage import median_filter
from tensorflow.keras.utils import to_categorical
import numpy as np
import pandas as pd
import cv2

#Loop Through Each Subfolder
base_folder = '/work/zhoulong/HL/Time_CSV/'  # The folder where your T=0, T=1, ...,
T=99 folders are located
Save_folder = '/work/zhoulong/HL/EachTime/'
average_accuracies = {}  # Keys are T{i}, values are the corresponding average
accuracies

for i in range(0,100):  # Assuming subfolders are named 'T=0' through 'T=99'
    subfolder_name = f'T={i}'
    print(f'T={i}')
    subfolder_path = os.path.join(base_folder, subfolder_name)
    X,Y,Z =load_data_and_labels(subfolder_path)
    X_reshaped = X[:, :, :, np.newaxis]
    X = median_filter(X, size=3)
    #X = gaussian_filter(X, sigma=1)

    k = 5  # Number of folds
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    accuracy_scores_k = []
    #print(len(accuracy_scores_k))

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        Y_test_encoded = to_categorical(Y_test, num_classes=2)

        Y_train_encoded = to_categorical(Y_train, num_classes=2)
        # Hyperparameter tuning for learning rate
        model = create_cnn_model(X_reshaped[0].shape, num_classes=2, 
learning_rate=0.001)  # Recreate model to reset weights
        # Train the model with validation data
        model.fit(X_train, Y_train_encoded, epochs=20, batch_size=12, verbose=0)
        scores_k = model.evaluate(X_test, Y_test_encoded, verbose=0)
        accuracy_scores_k.append(scores_k[1])

    # Calculate the average accuracy
    average_accuracy_k = np.mean(accuracy_scores_k)
    average_accuracies[f'T{i}'] = average_accuracy_k
    print(f'Average Accuracy from {k}-Fold CV for T{i}: {average_accuracy_k:.2f}')



# In[ ]:


import numpy as np
import pandas as pd

def save_heatmap_to_csv(heatmap, file_path='heatmap.csv'):
    """
    Save the heatmap data to a CSV file.
    Parameters:
    - heatmap: Numpy array containing the heatmap data.
    - file_path: The local path where the CSV file will be saved.
    """
    # Convert the heatmap numpy array to a DataFrame
    df = pd.DataFrame(heatmap)
    # Save the DataFrame to a CSV file
    df.to_csv(file_path, index=False)
    print(f"Heatmap saved to {file_path}")


# In[ ]:


print(len(average_accuracies))


# In[ ]:


import matplotlib.pyplot as plt

labels = list(average_accuracies.keys())
accuracies = list(average_accuracies.values())

# Create a new list for sparse x-axis labels
sparse_labels = [label if i % 5 == 0 else '' for i, label in enumerate(labels)]

plt.figure(figsize=(14, 6))  # Adjusted figure size for better readability
plt.bar(labels, accuracies, color='skyblue')
plt.xlabel('File Identifier')
plt.ylabel('Average Accuracy')
plt.title('Average Accuracy for Each Time Point')
plt.xticks(labels, sparse_labels, rotation=0)  # Use sparse_labels here
plt.ylim(0,1)
plt.grid(True)
plt.tight_layout()  # Use tight layout to ensure everything fits without overlapping
plt.show()


# In[ ]:
```

(Each Time)

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:
```

```
import os

def list_directory_contents(directory):
    try:
        with os.scandir(directory) as entries:
            for entry in entries:
                print(entry.name)
    except FileNotFoundError:
        print("Directory not found.")

# Replace 'directory_path' with the path to the directory you want to list
directory_path = '/work/zhoulong/HL/'
list_directory_contents(directory_path)


# In[ ]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import tensorflow as tf
import keras
from keras import layers, models
import io
import os


# In[ ]:


def process_file(file_path):
    # Load the CSV data into a DataFrame
    df = pd.read_csv(file_path)

    # Convert the DataFrame to a NumPy array for numerical operations
    data = df.values  # This converts the entire DataFrame to a NumPy array

    # Calculate min, max, and standard deviation
    min_val = np.min(data)
    max_val = np.max(data)
    std_dev = np.std(data)
    ave_val = np.mean(data)

    # Check if the standard deviation is zero (i.e., all values are the same)
    if std_dev == 0:
        # Set the data to zero to indicate no variation
        data_normalized = np.zeros(data.shape)
    else:
        # Normalize the data
        data_normalized = (data - ave_val) / (max_val - min_val)
        #data_normalized = (data - min_val) / (max_val - min_val)

    return data_normalized


# In[ ]:


def load_data_and_labels(folder_path):
    X = []  # To store data
    Y = []  # To store labels
    Z = []  # To store sample name

    for filename in os.listdir(folder_path):
        #print(filename)
        file_path = os.path.join(folder_path, filename)

        # Check if it's a file and not a directory
        if os.path.isfile(file_path):
            # Process the file (you'll need to replace this with your actual data processing)
            data = process_file(file_path)  # Implement this function based on your data
format
            #data = np.loadtxt(file_path, delimiter=None)
            X.append(data)

            # Assign label based on file name
            Y.append(0 if filename.startswith('Control') else 1)

            # Assign file name to corresponding sample
            Z.append(filename)

    return np.array(X), np.array(Y), Z


# In[ ]:
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.initializers import RandomNormal, HeNormal

def create_cnn_model(input_shape, num_classes, learning_rate=0.001):
    model = Sequential([
        Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape,
kernel_initializer=HeNormal(),name='conv1'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu', kernel_initializer=HeNormal(),name='conv2'),
        MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', name='conv3'),
        Flatten(),
        Dense(128, activation='relu', kernel_initializer=HeNormal()),
        Dense(num_classes, activation='softmax', kernel_initializer=HeNormal())
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model


# In[ ]:


import matplotlib.pyplot as plt
import tensorflow as tf

def make_gradcam_heatmap(img_array, model, last_conv_layer_name,
pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        #print("Output contains NaN:", np.isnan(last_conv_layer_output).any())
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)

    # Handle different dimensions in grads
    if len(grads.shape) == 4:  # Typical case for 4D tensor [batch, height, width,
channels]
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    elif len(grads.shape) == 3:  # In case grads is a 3D tensor [height, width, channels]
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1))
    else:  # Other cases need to be handled specifically
        raise ValueError("Unexpected shape for gradients: " + str(grads.shape))

    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    epsilon = 1e-10
    heatmap = tf.maximum(heatmap, 0) / (tf.math.reduce_max(heatmap) + epsilon)
   # heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()


# In[ ]:


import matplotlib.pyplot as plt
import cv2
import numpy as np

def display_heatmap_only(img, heatmap, alpha=0.4, size=(36, 21), save_path=None):
    # Resize heatmap for visualization
    heatmap_resized = cv2.resize(heatmap, size)

    # Convert heatmap to RGB
    heatmap_resized = np.uint8(255 * heatmap_resized)
    heatmap_resized = cv2.applyColorMap(heatmap_resized, cv2.COLORMAP_JET)

    # Create a figure to display the results
    plt.figure(figsize=(6, 6))

    # Display Heatmap with Color Bar
    im = plt.imshow(heatmap_resized, cmap='jet')
    plt.axis('on')  # Show or hide the axis as per your requirement
    plt.title('Heatmap')

    # Add color bar
    plt.colorbar(im, fraction=0.046, pad=0.04, label='Importance')

    # Adjust layout
    plt.tight_layout()
    # Save or show the plot
    if save_path:
        plt.savefig(save_path, bbox_inches='tight', pad_inches=0)
        print(f"Image saved to {save_path}")
    else:
        plt.show()


# In[ ]:


from keras.callbacks import EarlyStopping

# Define the early stopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Monitor the validation loss
    patience=3,         # Number of epochs with no improvement after which training will
be stopped
    min_delta=0.001,    # Minimum change in the monitored quantity to qualify as an
improvement
    mode='min',         # The direction is automatically inferred if not set, but here 'min'
means we want to minimize the loss
    verbose=1           # Print a message when early stopping is triggered
)


# In[ ]:


# Function to plot average training and validation loss and accuracy for all 20 epoch
situation
def plot_avg_training_validation_loss_accuracy(all_folds_history, save_path=None,
font_size=16, line_width=3):
    avg_loss = np.mean([fold['loss'] for fold in all_folds_history], axis=0)
    avg_val_loss = np.mean([fold['val_loss'] for fold in all_folds_history], axis=0)
    avg_accuracy = np.mean([fold['accuracy'] for fold in all_folds_history], axis=0)
    avg_val_accuracy = np.mean([fold['val_accuracy'] for fold in all_folds_history],
axis=0)

    epochs = range(1, len(avg_loss) + 1)

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, avg_loss, 'bo-', label='Average Training Loss', linewidth=line_width)
    plt.plot(epochs, avg_val_loss, 'ro-', label='Average Validation Loss', 
linewidth=line_width)
    plt.title('Training and Validation Loss', fontsize=font_size)
    plt.xlabel('Epochs', fontsize=font_size)
    plt.ylabel('Loss', fontsize=font_size)
    plt.legend(fontsize=font_size)

    plt.subplot(1, 2, 2)
    plt.plot(epochs, avg_accuracy, 'bo-', label='Average Training Accuracy', 
linewidth=line_width)
    plt.plot(epochs, avg_val_accuracy, 'ro-', label='Average Validation Accuracy', 
linewidth=line_width)
    plt.title('Training and Validation Accuracy', fontsize=font_size)
    plt.xlabel('Epochs', fontsize=font_size)
    plt.ylabel('Accuracy', fontsize=font_size)
    plt.legend(fontsize=font_size)

    plt.tight_layout()

    # Save to file if save_path is provided
    if save_path:
        plt.savefig(save_path, bbox_inches='tight', pad_inches=0)
        print(f"Plot saved to {save_path}")
    else:
        plt.show()


# In[ ]:


from sklearn.model_selection import KFold
from scipy.ndimage import median_filter
from tensorflow.keras.utils import to_categorical
import numpy as np
import pandas as pd
import cv2

#Loop Through Each Subfolder
base_folder = '/work/zhoulong/HL/Time_CSV/'  # The folder where your T=0, T=1, ..., 
T=99 folders are located
Save_folder = '/work/zhoulong/HL/EachTime/'
average_accuracies = {}  # Keys are T{i}, values are the corresponding average 
accuracies
```

```
for i in range(0,100):  # Assuming subfolders are named 'T=0' through 'T=99'
    subfolder_name = f'T={i}'
    print(f'T={i}')
    subfolder_path = os.path.join(base_folder, subfolder_name)
    save_path = os.path.join(Save_folder,  f'T={i}learning.png')
    X,Y,Z =load_data_and_labels(subfolder_path)
    X_reshaped = X[:, :, :, np.newaxis]
    X = median_filter(X, size=3)
    #X = gaussian_filter(X, sigma=1)

    k = 5  # Number of folds
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    accuracy_scores_k = []
    all_folds_history = []
    #print(len(accuracy_scores_k))

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        Y_test_encoded = to_categorical(Y_test, num_classes=2)

        # Train the model
        val_split_index = int(0.8 * len(X_train))
        X_train, X_val = X_train[:val_split_index], X_train[val_split_index:]
        Y_train, Y_val = Y_train[:val_split_index], Y_train[val_split_index:]

        Y_train_encoded = to_categorical(Y_train, num_classes=2)
        Y_val_encoded = to_categorical(Y_val, num_classes=2)
        # Hyperparameter tuning for learning rate
        model = create_cnn_model(X_reshaped[0].shape, num_classes=2,
learning_rate=0.001)  # Recreate model to reset weights

        # Train the model with validation data
        history = model.fit(X_train, Y_train_encoded, validation_data=(X_val,
Y_val_encoded),
                      epochs=20, batch_size=12, verbose=0)
        all_folds_history.append(history.history)
        scores_k = model.evaluate(X_test, Y_test_encoded, verbose=0)
        accuracy_scores_k.append(scores_k[1])

    # Calculate the average accuracy
    average_accuracy_k = np.mean(accuracy_scores_k)
    average_accuracies[f'T{i}'] = average_accuracy_k
    print(f'Average Accuracy from {k}-Fold CV for T{i}: {average_accuracy_k:.2f}')
    plot_avg_training_validation_loss_accuracy(all_folds_history, save_path=save_path,
font_size=16, line_width=3)

    if(average_accuracy_k>0.8):
     # print(subfolder_name)
      model =model
      # Name of the last convolutional layer
      last_conv_layer_name = "conv1"
      # Select the example and add the batch dimension
      for j in range(len(Z)):
        save_path = os.path.join(Save_folder, Z[j]+'.png')
        img_array = np.expand_dims(X[j], axis=0)  # This changes the shape from (36, 21,
1) to (1, 36, 21, 1)
      # Generate the heatmap
        heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
      # Prepare the image for display_gradcam function
        img_to_display = img_array[0].squeeze()
      #print(img_to_display.shape)
        if len(img_to_display.shape) == 2:  # If the image is 2D, convert it to 3D
            img_to_display = np.repeat(img_to_display[..., np.newaxis], 3, axis=2)

        display_heatmap_only(img_to_display, heatmap, alpha=0.4, size=(21, 36),
save_path=save_path)


# In[ ]:


from sklearn.model_selection import KFold
from scipy.ndimage import median_filter
from tensorflow.keras.utils import to_categorical
import numpy as np
import pandas as pd
import cv2

#Loop Through Each Subfolder
base_folder = '/work/zhoulong/HL/Time_CSV/'  # The folder where your T=0, T=1, ...,
T=99 folders are located
Save_folder = '/work/zhoulong/HL/EachTime/'
average_accuracies = {}  # Keys are T{i}, values are the corresponding average
accuracies

for i in range(0,100):  # Assuming subfolders are named 'T=0' through 'T=99'
    subfolder_name = f'T={i}'
    print(f'T={i}')
    subfolder_path = os.path.join(base_folder, subfolder_name)
    X,Y,Z =load_data_and_labels(subfolder_path)
    X_reshaped = X[:, :, :, np.newaxis]
    X = median_filter(X, size=3)
    #X = gaussian_filter(X, sigma=1)

    k = 5  # Number of folds
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    accuracy_scores_k = []
    #print(len(accuracy_scores_k))

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        Y_test_encoded = to_categorical(Y_test, num_classes=2)

        Y_train_encoded = to_categorical(Y_train, num_classes=2)
        # Hyperparameter tuning for learning rate
        model = create_cnn_model(X_reshaped[0].shape, num_classes=2,
learning_rate=0.001)  # Recreate model to reset weights
        # Train the model with validation data
        model.fit(X_train, Y_train_encoded, epochs=20, batch_size=12, verbose=0)
        scores_k = model.evaluate(X_test, Y_test_encoded, verbose=0)
        accuracy_scores_k.append(scores_k[1])

    # Calculate the average accuracy
    average_accuracy_k = np.mean(accuracy_scores_k)
    average_accuracies[f'T{i}'] = average_accuracy_k
    print(f'Average Accuracy from {k}-Fold CV for T{i}: {average_accuracy_k:.2f}')



# In[ ]:



import numpy as np
import pandas as pd

def save_heatmap_to_csv(heatmap, file_path='heatmap.csv'):
    """
    Save the heatmap data to a CSV file.
    Parameters:
    - heatmap: Numpy array containing the heatmap data.
    - file_path: The local path where the CSV file will be saved.
    """
    # Convert the heatmap numpy array to a DataFrame
    df = pd.DataFrame(heatmap)
    # Save the DataFrame to a CSV file
    df.to_csv(file_path, index=False)
    print(f"Heatmap saved to {file_path}")


# In[ ]:


print(len(average_accuracies))


# In[ ]:


import matplotlib.pyplot as plt

labels = list(average_accuracies.keys())
accuracies = list(average_accuracies.values())

# Create a new list for sparse x-axis labels
sparse_labels = [label if i % 5 == 0 else '' for i, label in enumerate(labels)]

plt.figure(figsize=(14, 6))  # Adjusted figure size for better readability
plt.bar(labels, accuracies, color='skyblue')
plt.xlabel('File Identifier')
plt.ylabel('Average Accuracy')
plt.title('Average Accuracy for Each Time Point')
plt.xticks(labels, sparse_labels, rotation=0)  # Use sparse_labels here
plt.ylim(0,1)
plt.grid(True)
plt.tight_layout()  # Use tight layout to ensure everything fits without overlapping
plt.show()


# In[ ]:
```