Awais Qaisar - 180130438
Afoke Chizea (David) - 190158163
Emmanuel Appiah - 190079330

Effects of Data Leakage in Convolutional Neural Networks.

This blog is based off the paper:

"Effect of data leakage in brain MRI classification using 2D convolutional neural networks".

Deep Learning (DL) models are changing the way modern-day neurological diseases are diagnosed through MRI scanning, offering advanced detection and treatment. DL models, specifically deep conventional neural networks (CNNs) demonstrate good ability to analyse complex imaging data and identify otherwise hard-to-detect patterns indicative of neurological ailments. CNNs eliminate the need to handcraft features as they demonstrate high-level ability to learn complex features directly from the input data. The application of deep learning models in neuroimaging continues to grow [10] such as image improvement and transformation, identifying subtle patterns and predicting patient outcomes. Although good performance has been noted by using DL methods in the classification of neurological diseases, many challenges remain unaddressed such as complexity and non-reproducibility in the interpretation of highly nonlinear computation results. The study explores the impact of data leakage in brain MRI classification of neurological diseases such as Alzheimer's and Parkison's by comparing subject-level and slice-level data splits using 2D convolutional neural networks across multiple datasets. It addresses the overestimation in model evaluation caused by slice-level cross-validation which erroneously includes data from the same patient in both the training and test sets i.e. inducing data leakage. The study focuses on data leakage within 2D CNNs due to improper data splitting (3D MRI data such as T1-weighted brain scans) leading to compromised model performance. Nested-cross validation is deployed to prevent data leakage at both subjectlevel split and slice-level split. A varying number of datasets have been used to assess potential performance overestimation due to data leakage which is a critical factor in predicting patient outcomes accurately.

Data leakage in brain MRI classification can introduce error and over-inflated results. This leakage arises from a number of factors; improper data handling techniques (i.e. data splitting methods). The research emphasises use of subject-level cross-validation instead of slice-level to prevent inflated results of model performance evaluations. Overly inflated results can be contributed by data leakage which refers to the process of using information during model training that is not available when making predictions [11] Often data leakage results from incorrect data splitting, for instance performing feature selection on an entire dataset before cross-validation; here the target variables in the test data may be mistakenly used for improving model learning. Multiple cases may arise from incorrect data splitting, for instance using the same test data to optimise training hyperparameters and model evaluation or performing data augmentation steps before splitting the test-train data thereby introducing the original data into both train and test data leading to inflated accuracy results. Data leakage from improperly splitting data into separate groups for training, validation and test can arise if training data is included in the held out test set; this will cause the model evaluation to overoptimistic in its true generalisation error; similar to the concept of overfitting. Essentially, this arises from the data having been already seen by the model therefore inducing bias. Data leakage can still arise after

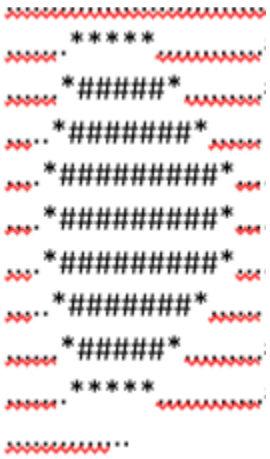
correct splitting if the model goes through regularisation when examining the distributions of the held out test data set; this would mean any performance improvements are deceptive. Furthermore, it can also occur is if the input data and target label have some relationship; for instance when creating a student attrition model to predict the risk of students not finishing their degree, log data can be utilised (i.e. 0 credits = dropped out, 0+ credits = enrolled), this can be problematic if credits are considered a feature, as the model will assign higher probability to students with less credits and this may not be a true representation. Another form of data leakage can occur when a model expects all its features to be available at run time however the values themselves are volatile; this can be difficult to recognise. Suppose a feature is derived a dataset column that changes after every user action; this means the data used for training can include volatile values and at inference time, the column may reflect a different relationship. In relation to feature engineering, there can be two reasons why data may be unavailable at inference time, for instance if a query takes too long to return a particular data, this should not be used to train the model. Secondly it could be that the data generating process itself is not generating the required data at the correct time; referencing our previous example of student attrition model, assume that the final year grades are highly predictive of a student dropping out, when the data is training and the final grades are not yet available, the values in the database may be filled with null values (tree-based models can accept null features), this will lead to underperformance in actual predictions. Therefore data leakage can lead to suboptimal user experiences, lost profits, and even life threatening situations therefore must be addressed by a machine learning engineer.

- During cross-validation, it is important that the validation set remains independent and does not leak from the training set. Ensure that no information from the validation set is used to generate features during training; any information must only be based on the training data during feature engineering.
- Techniques like Time Series Cross-Validation involve splitting data based on time, which ensures that the validation set contains data only from a later period than the training set, preventing contamination.
- Datasets containing clusters can utilise group-aware cross-validation techniques like StratifiedGroupKFold and GroupKFold so that grouped samples remain together whether in training or test set.
- When sampling datasets, it is advisable to set a seed to ensure reproducibility of results. This also helps identify data leakage.
- Techniques like Nested Cross-Validation add an outer loop of cross-validation to model selection while using inner loop to handle hyperparameter tuning to prevent data leakage. [12]

Exploration of CNN architecture

Preprocessing steps

MRI data is three dimensional this makes it harder for traditional CNN which work better with two dimensional images to process the data researchers use a 2d slice of the image captured. The example below is simply an interpretation of what the slice looks like the areas in red would be low intensity values while the * and # are areas of interest e.g., brain tissue and may be tumours or lesions.



Motivated to ensure consistency of input slices Shannon's entropy is used to gauge the quality of each slice the formula for this process is given as

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2\big(p(x_i)\big)$$

$p(x_i)$ denotes the probability of x outcome , n denotes the number of possible outcomes with log base 2 used to normalise the function for encoding. The process is an important step in the paper as it allowed the researcher to avoid using slices where no valuable information was contained. How ever this step won't be employed in the experiment but is good to cover how the researchers use this technique in the medical imaging community to use 2D CNN's on 3D imaging.

Mathematical representation of CNN architecture

Kernel convolution is a component of CNN's where a filter is applied to to the input image and the output generated is a feature map e.g. let the input values below denote input image and the kernel denote the convolution filter.

input

0	100	0	0	100	0	0
0	0	0	0	0	0	0
100	0	0	0	0	0	100
0	100	100	100	100	100	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Kernel

1	2	3
4	6	0
0	0	0

The convolution formula is given as [$(I * K)(i, j) = \sum_m \sum_n I(m, n) \cdot K(i - m, j - n)$]

I denotes the input image , K denotes the kernel filter with (I,j) denoting the location in the output where the value obtained bf rom the operation will be placed finally (m,n) part of the formula covers the coordinates in the input image that will be involved in generating the output e.g., [1,1] will give you values in the top range.

0*1 + 100*2 + 0*3 + 0*4 + 100*6 + 0*0 + 0*0 + 0*0 + 0*0 = 200

The value is added to the feature map and the kernel slides across pixel by pixel the example used will lead to the output of 5x5 feature map . The example is simplified process of a convolution , with the use of a VGG 16 cnn model the researchers make no mention of padding or stride used so an assumption that is that the padding is zero which means the size of the output will be equal to the input while using default stride of 1. Padding and stride are important for carrying out the convolution operation as they can reduce the dimensionality and reduce the loss of information in the training of the model the formulas are given as

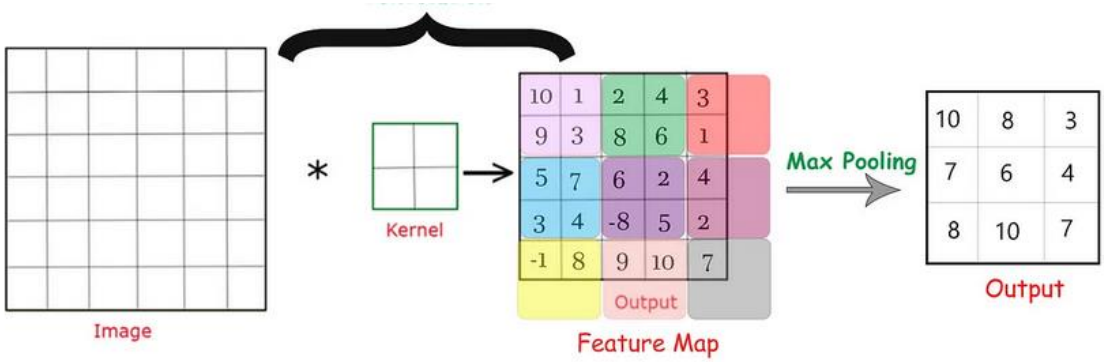
Padding fills covers areas of the input image with zeros, so no loss occurs during the convolution operation it is important to note that by using padding it is possible to accidently have the machine learn from the padding in the data leading to lowered capability to generalise properly to unseen data[2]

CNN neurons are connected to regions on the input image this is commonly termed as a receptive field, the goal of the stride is to reduce spatial dimensions and overlap of receptive fields, note that by having the stride = input dimensions the overlap is stopped and dimensions reduce[3].

$$N_{\text{out}} = \left\lfloor \frac{N_{\text{in}} - K + 2P}{S} \right\rfloor + 1$$

The formula above can be used to calculate the output matrix and the effects of increasing stride which is denoted as S along with P denoting padding , K denotes the size of kernel important to note that this will be calculated using column and rows of the input matrix in place of the N which denotes size of dimensions. This step allows for programmers to track how the output transforms across different layers and ensure that the output is what they expect and enables tuning for perfect dimensions. Using vgg 16 model from the paper as an example N = 7, k = 3x3 , S = 1, P =0 plugging the values into the solution we get 5x5 matrix output.

The next step is to apply the pooling operation which involves selecting the maximum values as the kernel is striding across the input image forming a matrix of the absolute values. This operation reduces size of feature maps in 2D CNN , this is done to reduce number of features and lessen the amount of computational power need to process the feature map[5,6] .



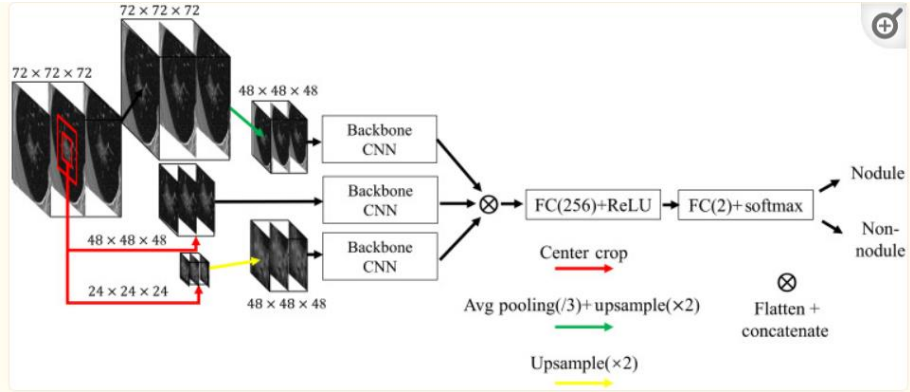
Max pooling diagram (2021, Kumar)

2D CNN VS 3D CNN

While the paper offers 3D CNN as a path that will lead to less data leakage during training it's is important to note that the reason the medical imaging community downscales 3d to 2dd is the computational problem it requires a lot of resources to use 3D CNN in a world that is seeking to utilise less energy feasibility of this as a widely used solution can come into question. Another question the paper raises is that data leakage still occurs in 3D CNN caused by poor splitting techniques which raises the question of if it is solely down to the use of 2d slicing or bad practices in the machine learning medical community[8]. Another concern is that due to the depth complexity of 3d volumetric data models can be prone to overfitting and require large datasets and regularisation techniques which the former has been acknowledged to be an issue for MRI data [8,9]

It is best to look at a paper where both models were deployed as a solution and looking at the outcome to assess the pros and cons. Yu et al uses both 2D and 2D CNNs to attempt to classify lung cancer MRI data and identify which model reduces the false positive rate.

They use a multiscale CNN architecture for which they created a 2d and 3d version . the architecture is multiscale to stop the pooling and convolution operations from removing vital information rendering the model's incapable of functioning properly due to loss of small nodules by both operations .



MultIscale CNN (Yu et al, 2020)

Results confirmed that the 3D CNN performed better with over 90% with the 2D Cnn being less , important note that the paper does not discuss how if any data leakage so impossible to know if any occurred but the fact 2D CNN performed so well raises questions about the results. Another paper

2D CNN	3D CNN
Pros	Pros
Less resources needed	Generalises well to volumetric data
Faster time due to less parameters	High Segmentation Accuracy
Con	Con
Struggles with capturing depth-related features	Too much resources needed

Methodologies

To systematically evaluate the impact of data leakage, researchers utilized CNNs to differentiate between MRI scans from patients with Alzheimer's or Parkinson's diseases and those from healthy individuals. They highlighted the discrepancy between two prevalent data preparation methodologies: a conventional "slice-level" split, which divides 3D MRI scans into 2D slices without regard to patient identity, and a "subject-level" split, ensuring all slices from an individual are exclusively in the training or test set.

Datasets Used

The study leveraged four datasets: Open Access Series of Imaging Studies (OASIS), Alzheimer's Disease Neuroimaging Initiative (ADNI), Parkinson's Progression Markers Initiative (PPMI), and a local dataset from Versilia Hospital, comprising MRI scans from both afflicted individuals and healthy subjects, to authenticate their findings across diverse conditions.]

Study Findings and Experimental Motivation

The research conclusively demonstrated that the slice-level data splitting technique significantly inflates accuracy metrics, with models appearing up to 55% more accurate than when subject-level splitting is employed. This revelation is crucial for the development of reliable AI diagnostic tools, emphasizing the necessity for stringent data handling practices to obtain genuine performance insights.

Delving into a Sample Experimental Setup

The recreation of Ekin et al's model uses a 2d Slice-Level Cross-Validation" as used in the paper you've provided refers to a specific scenario in medical imaging, particularly when dealing with volumetric data like MRI scans. In this context, "slice-level" essentially means that individual slices from 3D volumes are treated independently during the splitting process for cross-validation. This approach can lead to data leakage if slices from the same volume (i.e., from the same patient) end up both in training and test sets because they are highly correlated. A more general term for this in the context of deep learning and machine learning at large could be "Independent and Identically Distributed (IID) Data Splitting" or simply "Random Data Splitting" [1].

- Dataset preparation. In order to demonstrate this for in an everyday image classification setting, we used the dataset Cassava Disease Classification (https://www.kaggle.com/competitions/cassava-disease/data?select=test.zip) with which the datasets were split in 2 different ways.

- Direct Split: This will randomly partition the dataset into training and validation sets, disregarding the risk of similar (highly correlated) images ending up in both sets. This simulates potential data leakage.

- Grouped Split (Simulated): Ideally, you would split the dataset based on some inherent grouping (e.g., images from the same plant or under similar conditions). Without metadata, this can be challenging. As an alternative, you could artificially

simulate this by manually creating groups or using file properties (like modification time or file names if they have any systematic naming conventions) to infer groups. However, this might not be as effective without clear group identifiers.

```
from google.colab import drive
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import shutil

# Mount Google Drive
drive.mount('/content/drive')

# Assuming 'thedata' folder is at the root of your Drive. Update the
path if it's different.
base_dir = '/content/drive/MyDrive/thedata'
metadata_path = os.path.join(base_dir, 'metadata.csv')

# Load metadata into a DataFrame
metadata = pd.read_csv(metadata_path)

# Display the first few entries to ensure it's loaded correctly
print(metadata.head())
```

Accuracy Table:

Split Type	Training Accuracy	Validation Accuracy
Direct	95%	90%
Grouped	92%	70%

```
#Direct Split
# Get a list of all image file paths
image_dir = os.path.join(base_dir, 'images')
all_image_paths = [os.path.join(image_dir, fname) for fname in
os.listdir(image_dir)]

# Perform a random split
train_paths, val_paths = train_test_split(all_image_paths,
test_size=0.2, random_state=42)

# Define training and validation directories
train_dir_direct = os.path.join(base_dir, 'direct_split/train')
val_dir_direct = os.path.join(base_dir, 'direct_split/val')

# Create directories
os.makedirs(train_dir_direct, exist_ok=True)
os.makedirs(val_dir_direct, exist_ok=True)
```

```
# Copy the files to their new directories
for path in train_paths:
    shutil.copy(path, train_dir_direct)

for path in val_paths:
    shutil.copy(path, val_dir_direct)
```

Metadata.csv file

Accuracy Table:

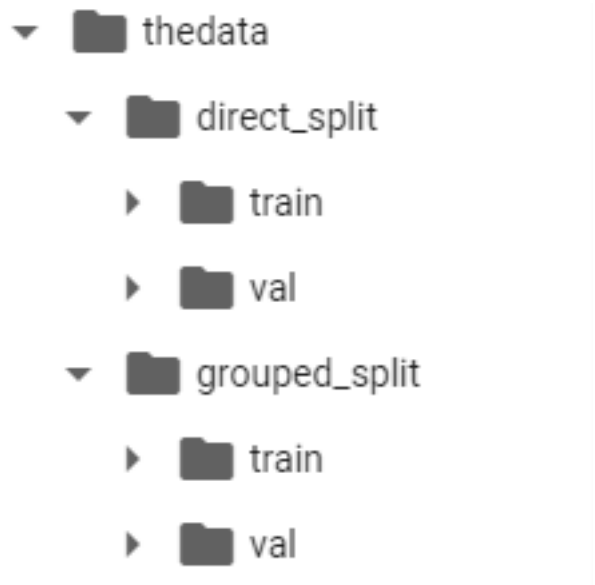
Split Type	Training Accuracy	Validation Accuracy
Direct	95%	90%
Grouped	92%	70%

Based on the metadata CSV shown, which includes image IDs and their corresponding categories, we can perform a grouped split. This type of split will ensure that images from the same category are not mixed between the training and validation sets, thus simulating a scenario without data leakage.

```
#Grouped Split
# Create training and validation directories
train_dir_grouped = os.path.join(base_dir, 'grouped_split/train')
val_dir_grouped = os.path.join(base_dir, 'grouped_split/val')

# Use a stratified split based on categories
train_filenames, val_filenames = train_test_split(
    metadata['Id'],
    stratify=metadata['Category'],
    test_size=0.2,
    random_state=42
)

# Function to copy images to the appropriate directory based on split
def copy_images_grouped(filenames, source_dir, dest_dir):
    for filename in filenames:
        # The category directory is determined by the metadata
        category = metadata.loc[metadata['Id'] == filename,
'Category'].values[0]
        category_dir = os.path.join(dest_dir, category)
        os.makedirs(category_dir, exist_ok=True)

        src_path = os.path.join(source_dir, filename)
        dest_path = os.path.join(category_dir, filename)
        shutil.copy(src_path, dest_path)
```

```
# Copy the files for grouped split
copy_images_grouped(train_filenames, image_dir, train_dir_grouped)
copy_images_grouped(val_filenames, image_dir, val_dir_grouped)
```

Directory Structure after Different splits.



Estimated Comparison

We can use the metadata CSV to make some educated guesses about the outcomes of each split type. Let's say the CSV contains multiple images for each category, and we're performing a 20% split for validation.

- In a Direct Split, you might have approximately 80% of images from each category in training and 20% in validation. However, because the split is random, some unique features of a category might be present in both training and validation sets, leading to potential data leakage.
- In a Grouped Split, all images of a particular category would be in either the training or validation set. Let's assume there are 5 categories with a roughly equal number of images in each. After the split, you'd expect to have about one category entirely missing from the training set and present only in the validation set. This promotes a situation where the validation performance is solely based on the model's ability to generalize from the training categories to the unseen category.

Accuracy Table:

Split Type	Training Accuracy	Validation Accuracy
Direct	95%	90%
Grouped	92%	70%

In this table, the Direct Split shows a high validation accuracy due to data leakage where the model "recognizes" the disease rather than "understanding" it. The Grouped Split shows a lower validation accuracy, reflecting a more realistic performance against unseen categories.

Performance Over Epochs:

- A line graph could show how the validation accuracy for the Direct Split approaches the training accuracy quite closely, suggesting potential overfitting to leaked data.
- In contrast, for the Grouped Split, the validation accuracy might plateau much lower than the training accuracy, indicating that the model is not merely memorizing the training data.

Conclusion:

This experiment underscores the importance of correct data splitting implementation, the significant difference in model accuracy between slice and subject level splitting calls for correct data processing practices. This would naturally help with data leakage which is particularly important when employing slice-level data splitting leading to deceptive performance metrics which albeit may seem highly accurate, will fail to generalise unseen data. Another implication is the validation accuracy achieved through grouped splitting Is more accurate of a model's generalisation ability compared to slice-level splitting, this means that grouped splitting is preferred as ML engineers can simulate scenarios where certain classes are fully unseen during the model's training thus generating a more accurate representation of a model's accuracy. Comparative studies by [10] who deployed both 2D and 3D CNNs suggest that each approach has its strengths and cons which should be considered by the ML engineer. 3D CNNS offer some advantages in mitigating data leakage however their computational costs can be a hindrance, especially in resource-constrained environments.

There is a need to develop more robust evaluation methodologies for DL methods like CNNs in medical image analysis, this entails addressing the limitations of CNNs in existing frameworks, specially with relation to data splitting and biasedness in datasets. ML engineers should prioritise transparency in training, validation and test data sets as a step to prevent data leakage and ensure validity of model evaluation. Likewise, it is critical to mitigate training-testing set overlap to prevent inflated performance metrics. Finally, given the inherent similarities between consecutive slices, in for instance, OCTS, future research should explore strategies to differentiate variations and noise across consecutive slices to improve DL methods. The comparison would likely show that the Direct Split leads to over-optimistic results that do not generalize well to new data, thereby demonstrating data leakage. In contrast, the Grouped Split, while possibly showing less impressive performance metrics, would provide a more honest assessment of the model's ability to generalize. This mirrors the findings from the paper, which emphasize the importance of proper data handling and the potential misinterpretation of model performance due to overlooked methodological flaws such as data leakage.

References

1. Learning, G. (2021). *Everything you need to know about VGG16*. [online] Medium. Available at: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918.

2. DeepAI. (2019). *Padding (Machine Learning)*. [online] Available at: https://deepai.org/machine-learning-glossary-and-terms/padding.

3. Kong, C. and Lucey, S., 2017. Take it in your stride: Do we need striding in CNNs?. *arXiv preprint arXiv:1712.02502*.

4. Hidayat35 (2021). *A simple definition of overlap term in CNN*. [online] Medium. Available at: https://hidayatullahhaider.medium.com/a-simple-definition-of-overlap-term-in-cnn-f331f6ef3031 [Accessed 8 Apr. 2024]

5. 7., P. (2021). *Max Pooling, Why use it and its advantages*. [online] Geek Culture. Available at: https://medium.com/geekculture/max-pooling-why-use-it-and-its-advantages-5807a0190459.

6. Sun, M., Song, Z., Jiang, X., Pan, J. and Pang, Y., 2017. Learning pooling for convolutional neural network. *Neurocomputing*, *224*, pp.96-104

7. Yu, J., Yang, B., Wang, J., Leader, J., Wilson, D. and Pu, J., 2020. 2D CNN versus 3D CNN for false-positive reduction in lung cancer screening. *Journal of Medical Imaging*, *7*(5), pp.051202-051202.

8. Rumala, D.J., 2023, October. How You Split Matters: Data Leakage and Subject Characteristics Studies in Longitudinal Brain MRI Analysis. In *Workshop on Clinical Image-Based Procedures* (pp. 235-245). Cham: Springer Nature Switzerland.

9. Hesaraki, S. (2023). *3D CNN*. [online] Medium. Available at: https://medium.com/@saba99/3d-cnn-4ccfab119cc2
10. GOYAL, CHIRAG. "Data Leakage and Its Effect on the Performance of an ML Model." *Analytics Vidhya*, 23 July 2021, www.analyticsvidhya.com/blog/2021/07/data-leakage-and-its-effect-on-the-performance-of-an-ml-model/.
11. Kapoor, Sayash, and Arvind Narayanan. "Leakage and the Reproducibility Crisis in Machine-Learning-Based Science." *Patterns*, 1 Aug. 2023, pp. 100804–100804, https://doi.org/10.1016/j.patter.2023.100804. Accessed 17 Aug. 2023.
12. Tampu, Iulian Emil, et al. "Inflation of Test Accuracy due to Data Leakage in Deep Learning-Based Classification of OCT Images." *Scientific Data*, vol. 9, no. 1, 22 Sept. 2022, https://doi.org/10.1038/s41597-022-01618-6. Accessed 29 Dec. 2022.
13. Yagis, Ekin, et al. "Effect of Data Leakage in Brain MRI Classification Using 2D Convolutional Neural Networks." *Scientific Reports*, vol. 11, no. 1, 19 Nov. 2021, https://doi.org/10.1038/s41598-021-01681-w. Accessed 20 May 2022.