Towards Generalizable Alzheimer's Disease Diagnosis Algorithms

Isabella Chittumuri, Oliver Shetler, and Blake Vente

The use of machine learning (ML)in the health sciences has exploded in recent years. Unfortunately, ML products have not produced consistently reproducible results (McDermott et al. 2021). One major reason for reproducibility issues in health ML is that data are often incomplete. We looked at issues in the performance of a deep convolutional neural network on an Alzheimer's Disease brain imaging data-set. We investigated if regression improves a deep CNN architecture's accuracy when there are hidden categories. Surprisingly, we found that regression drastically harmed our model's generalizability. Additionally, we explored the loss-landscape associated with various parameterizations of a base CNN architecture. We find that models with a higher number of parameters are more prone to overfitting because their loss landscapes are more hilly. **(Note: We provide an editable link to the Drive folder containing our data and Colab Notebooks in the Resources section.)**

Keywords: deep learning, neuroscience, Alzheimer's disease, deproducability, regression, loss landscape

Introduction

The use of machine learning (ML) for medical diagnosis has become a hot research topic in recent years. Big names in technology such as IBM and Google have attempted to leverage cloud computing to build ML platforms intended to aid or even replace manual diagnostic procedures. Unfortunately, many ML diagnostic techniques, especially deep learning techniques, have not been reliably reproducible (McDermott et al. 2021). So much so, in fact, that IBM recently

dropped its Watson for Health Sciences efforts, amid stagnating progress in the field—despite seemingly strong profits ("IBM May Sell Watson Health Business Generating $1 Billion Annual Revenue, Says Report"). It is therefore important to explore novel methods for improving the generalizability of deep learning models in the medical field.

Many generalizability issues in deep learning come from over fitting, rather than bias. Moreover, many deep learning architectures are constructed in ways that do not leverage all the structure associated with the data being analyzed. Therefore, it seemed prudent to explore a method for constraining a model in a way that introduces unused data structure into the model. One kind of structure that is often neglected is the ordering of categorical response variables. We decided to compare the conventional Softmax classification approach to a regression-like Mean Squared Errors plus Rounding (MSE+R) approach to classifying an ordinal variable. In deep learning, the Categorical Softmax approach is analogous to a multinomial regression—which assumes no ordering among categories—while the MSE+R approach is equivalent to doing a regression, and then rounding to the nearest coded variable. We hypothesized that the extra constraint introduced by the MSE+R approach would improve our model's ability to generalize beyond its training data set. We tested this by comparing our Categorical model to the MSE+R model with categories missing. We supposed that the pre-supposition of order in the response variable would result in superior performance for the MSE+R model under conditions of missing training data, especially when the missing category sits between two known categories.

In order to check the plausibility of our view that bias was not a particularly large risk, we also did a study of the loss-landscape of various CNN architectures, all of which were derived from our original architecture. The goal of this study was to verify that less constrained versions ("larger" in various senses) of our model are at greater risk of over-fitting than more constrained versions of our model.

We wanted our findings to have implications for real-world applications of deep-learning in medicine. As a consequence, we picked the task of diagnosing the severity of Alzheimer's Disease (AD) from brain scans. AD is not easy to diagnose from brain images alone. In general, the severity of AD is medically diagnosed using symptom check-lists. This kind of task contrasts with diagnostic procedures that can be done "by eye" such as the identification of brain tumors. We chose this challenging ML task because it represents the most important

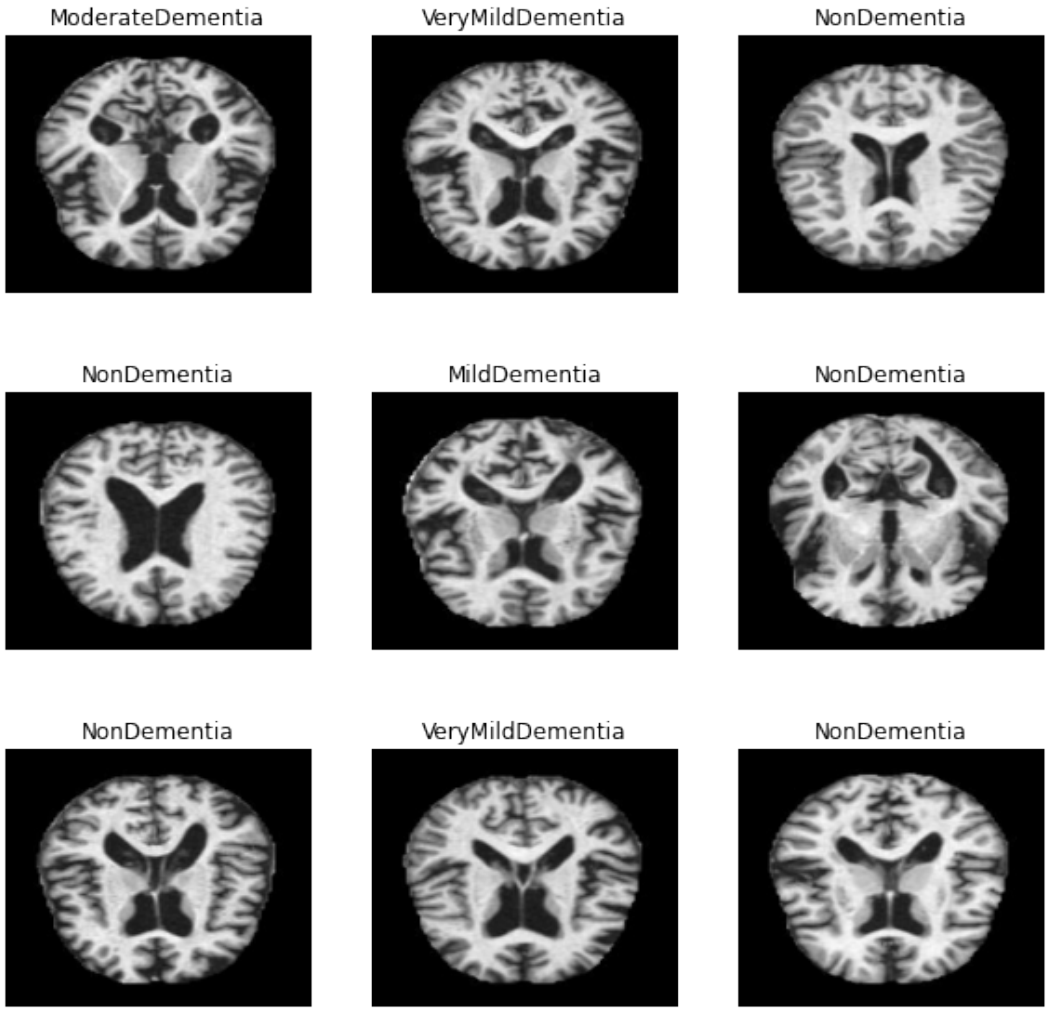
class of ML diagnostic tasks—those that do something humans cannot easily do. These sorts of tasks are the main reason why companies are investing money into this area.

Data

This project used an "Alzheimer's Dataset" from Kaggle. The data was already split into a training and testing set, each containing MRI images from four folders named after the diagnostic categories: Non Demented, Very Mild Demented, Mild Demented, and Moderate Demented. For our purpose, we split the training set where 20 percent of the data was in validation and 80 percent of the data was in training. The data established accuracy and validity because it was hand collected from various websites with each and every label verified.

The dataset was available in .jpg format and it was consistent in the classification of the four classes. However, the size of each class greatly varied resulting in an imbalanced data set. In total, Non Demented had 3200 observations, Very Mild Demented had 2240 observations, Mild Demented had 896 observations, and Moderate Demented had 64 observations.

Figure 1. Plots of Brain Scans



Note. A random sample of images was taken using the ‘choose()‘ function.

Categorical model

In creating a categorical model, we referenced Amy Jang’s model architecture corresponding to Kaggle website where we received the data (See the ”Resouces” secton). Similarly to Jang’s architecture, we defined a convolutional block which had two separable convolutional layers, a batch normalization layer, and a max pooling layer. Next, we defined a dense block which has

a dense layer, a batch normalization layer, and a dropout layer. We also one-hot encoded our labels because we were working with categorical data.

Now to build the model, we used the keras sequential model with two convolutional layers both with 16 filters, a ReLU activation function and a max pooling layer. After, we called four convolutional blocks, where the filter size increases exponentially. Of those convolutional blocks, the last two included a dropout layer of rate 0.2. Then we added three dense blocks, consecutively decreasing in unit size and dropout rate. Lastly, we included a dense layers with an output of four, corresponding to the number of classes, and a softmax activation function.

Figure 2. categorical model Architecture

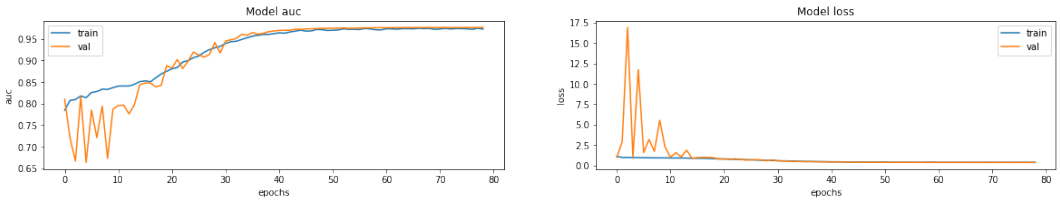
```
def conv_block(filters):
    block = tf.keras.Sequential([
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
        tf.keras.layers.SeparableConv2D(filters, 3, activation='relu', padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D()
    ]
    )

    return block

def dense_block(units, dropout_rate):
    block = tf.keras.Sequential([
        tf.keras.layers.Dense(units, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(dropout_rate)
    ])

    return block

def build_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(*IMAGE_SIZE, 3)),

        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.MaxPool2D(),

        conv_block(32),
        conv_block(64),

        conv_block(128),
        tf.keras.layers.Dropout(0.2),

        conv_block(256),
        tf.keras.layers.Dropout(0.2),

        tf.keras.layers.Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),

        tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
    ])
```

When compiling the model, we did not used accuracy because it cannot be used for imbalanced data set. Instead, we used Area Under the Curve (AUC) as a metric, an Adam optimizer, Categorical Cross-entropy loss function. An AUC closer to 1 suggests that our model

can distinguish between different classes, whereas the opposite applies with an AUC closer to 0. In training the model, we used a learning rate and exponential decay of 0.01 to help with minimizing the loss function. We also included early stopping with a patience of 10. Though we set our model to train for 100 epochs, it converged after 14 epochs with a training AUC of 0.84 and loss of 0.92 and a validation AUC of 0.71 and loss of 1.21.

Figure 3 shows what our categorical model's training AUC and loss would look like if it trained on 100 epochs. As the number of epochs increases, our training AUC steadily increases and converges towards 1. In contrast, our validation AUC is unstable during the first few epochs but starts to steadily increase and converge after 20 epochs. Similar results hold for our model's loss, where the training loss steadily declines while the validation loss is sporadic the first few epochs and then steadily declines and converges towards 0.

Figure 3. categorical model's AUC and Loss



When evaluating the model on the testing set, we got an AUC of 0.81 and a loss of 0.96. During both training and evaluating, we got a high AUC but our loss is also quite high which suggests that our model may not be preforming the way we wanted it to. To further evaluate our model, we used our model's predictions to get an accuracy score for the testing set which resulted in 0.48.

Mean Squared Error plus Rounding (Regression)

For the regression-like model, we used the ordinal codes:
0: Non-Dementia
1: Very Mild Dementia
2: Mild Dementia
3: Dementia

We built an MSE+R model using the same architecture as the categorical model with the only difference being the last dense layer had an output of 1, with a RelU output (all the target variables were positive; so this ensured no negative estimates). This single dense output layer forces the model to impose order since it has to differentiate on one layer.

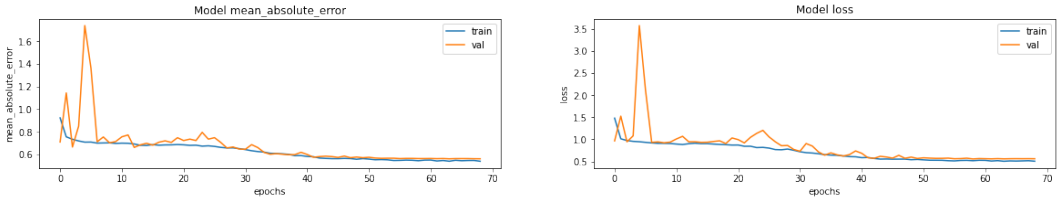
Figure 4. Regression Architecture

```
def build_regression_model():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(*IMAGE_SIZE, 3)),

        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.Conv2D(16, 3, activation='relu', padding='same'),
        tf.keras.layers.MaxPool2D(),

        conv_block(32),
        conv_block(64),

        conv_block(128),
        tf.keras.layers.Dropout(0.2),

        conv_block(256),
        tf.keras.layers.Dropout(0.2),

        tf.keras.layers.Flatten(),
        dense_block(512, 0.7),
        dense_block(128, 0.5),
        dense_block(64, 0.3),

        tf.keras.layers.Dense(1)
    ])
```

When compiling the model, we used the mean absolute error (MAE) as a metric, an Adam optimizer, a mean square error loss functions. The model converges at 69 epochs with a training MAE of .55 and loss of .52 and a validation MAE and loss of 0.56. When evaluating the model on the testing set, we got a MAE of 0.66, a loss of 0.84 and an accuracy of .40. We can see that our regression model did only slightly worse than our categorical model. However, looking at Figure 5, our validation MAE and loss tends to converge faster than that of our

categorical model.

Figure 5. Regression MAE and Loss



Xception

To verify that our categorical model was performing at a state-of-the-art level, we decided to see how the Xception model would preform with transfer learning on our dataset. We used Xception for this task, because we found that it was repeatedly used, with high performance, in medical image classification and specifically used for brain image classification (Hussain et al.).[1] When importing Xception, we used the image-net weights and did not include the final dense layer. We added a global average pooling and a final dense layer corresponding to the number of classes. When compiling the model, we kept the optimizer, loss function, and metrics the same as our categorical model.

[1]We chose not to use Xception as our base model because then we could not have easily varied the model architecture to study its loss landscape.

Figure 6. Xception Architecture

```
base_model = keras.applications.xception.Xception(weights="imagenet",
                                                  include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(NUM_CLASSES, activation="softmax")(avg)
model = keras.models.Model(inputs=base_model.input, outputs=output)
```

```
# for index, layer in enumerate(base_model.layers):
    # print(index, layer.name)
```

```
## Freeze the weights of the layers from the pretrained model
for layer in base_model.layers:
    layer.trainable = False

METRICS = [tf.keras.metrics.AUC(name='auc')]

model.compile(
        optimizer='adam',
        loss=tf.losses.CategoricalCrossentropy(),
        metrics=METRICS
    )
```

Using the one-hot encoded features, we first trained the model by freezing the weights of Xception. It converged after 15 epochs with a training AUC of 0.90 and loss of 0.90 and a validation AUC of 0.8 and loss of 2. When evaluating this model on the testing set, we got an AUC of 0.76, a loss of 6.33, and an accuracy of 0.38. Next we trained the model by unfreezing the weights. It converged after 25 epochs with a training AUC of 1 and loss of 5.9524e-05 and a validation AUC of 0.99 and loss of 0.13. When evaluating this model on the testing set, we got a training AUC of 0.84, a loss of 2.06 and an accuracy of 0.42. These transfer learning results are very similar to our custom model's results and therefore validate its architecture.

Figure 7. Frozen Xception AUC and Loss

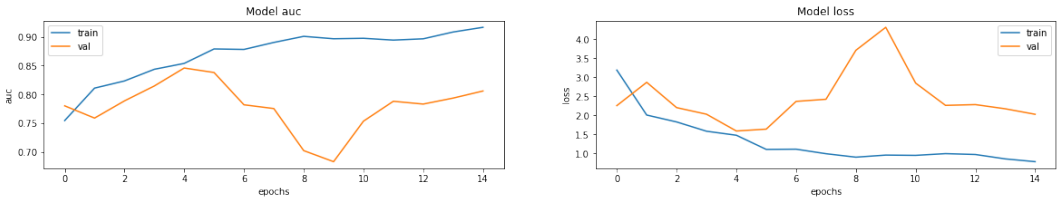
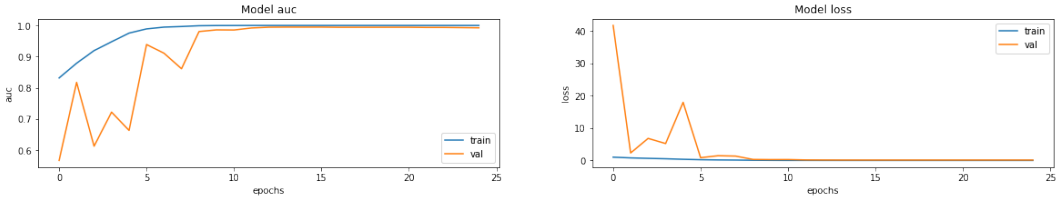


Figure 8. Unfrozen Xception AUC and Loss



Comparing Categorical and MSE+R Methods

Since we verified that our custom models were working correctly, we used it in testing for hidden categories. We kept the model architecture the same and only changed the data it was trained on. First we dropped the non-demented cases from the full training set, but kept it as an empty folder. We trained this new training set for both the categorical model and regression model. We then tested both of these models with the full testing set to see if either model preforms better in predicting non-demented cases. This procedure was repeated for every class. Table 1 shows the accuracy evaluation of each these models, including the two previous model that trained on the full data. Though accuracy is not a good measure of absolute model performance, we used it to compare the relative performance of all the models.

Exploring the Loss Landscape

We know that neural network architectures are opaque to inspection, but that the internal internal representations of these models are significant. The objective with this course of experimentation is to expose the internal workings of our convolutional the neural network

Table 1: Accuracy Scores of..

Model Type	FD	DND	DVMi	DMi	DMo
Categorical	.48	.48	.39	.44	.05
Regression	.40	.12	.09	.50	.06

Note. Each model was run

FD	Full Data
DND	Drop No Dementia Cases
DVMi	Drop Very Mild Dementia Cases
DMi	Drop Mild Dementia Cases
DMo	Drop Moderate Dementia Cases

architecture by inspecting the loss landscapes, and thus performance of trained models. Since the model has a convolutional architecture and the depth axis of the model is not uniform, we chose to vary the width of the network. Then, we reason about how the loss landscape and resulting model performance inform us of the internal representations stored in each instance of the trained network.

We varied model architecture by adding an "internal dimensions" parameter to our model function while keeping the relative distribution of parameters among layers constant. In so doing, we isolate the effect of model width, and deduce an understanding of its ability to generalize over other inputs. To operationally define generalization, we say that a model generalizes well when it has a similar output for similar input. That is, a model with an internal loss space that projects inputs onto a loss landscape many inflection points does not generalize.

Figure 9. Parametric Model with a learning rate of .001. The observed performance distribution is characteristic of all learning rates, as seen in 10

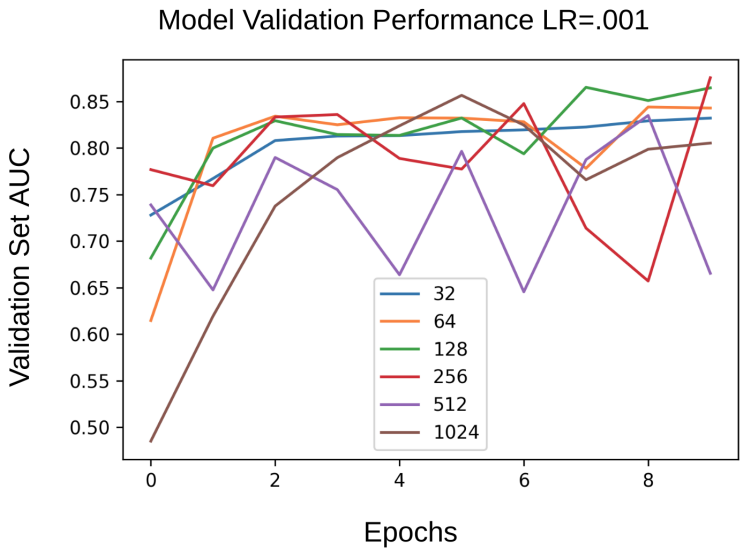
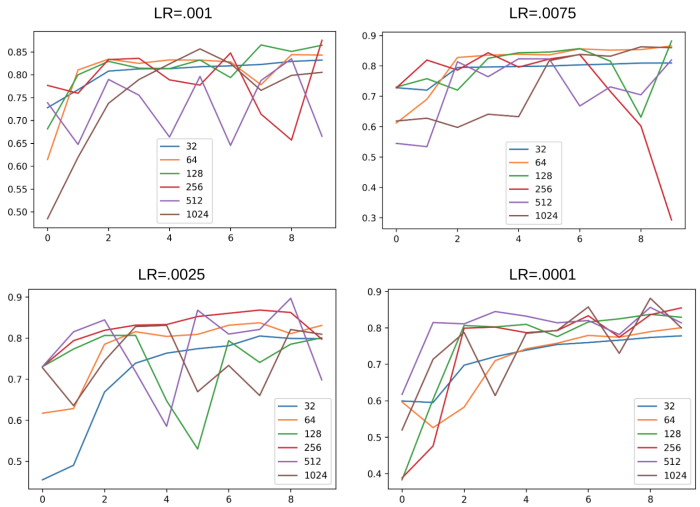


Figure 10. Parametric Model with all learning rates.



Loss Landscape Analysis

Across all models seen in Figures 9 and 10, we report the following traits of the loss landscape.

1. Models with a higher number of parameters have a higher number of inflection points. That is, the loss landscapes of these models are more hilly. This conclusion provides intuition into why models with higher numbers of parameters are more prone to overfitting. Their loss landscapes are higher dimensional so the distribution of weights across hidden units in a layer creates a more ***hilly*** optimization space. Since the space has many regions that are non-convex, it is more likely that these models fall into local minima.

2. Models with the lowest number of parameters increased performance monotonically. This implies that the optimization space has fewer hilly regions and their hills are shorter. We did not observe a small model outperforming a substantially larger model, but our smaller models are unrivaled in terms of consistency. We attribute this to having more regular internal weights rather than the high disparity observed in higher parameter models.

3. We chose the Adam (Adaptive Momentum) optimizer because it makes this hilly optimization space visible. With the use of momentum along with gradient descent, we can understand that Adam is pulling the model state down and climbing up the hills out of local minima that the model is otherwise prone to settling in.

Conclusions

While our findings are preliminary, we may tentatively conclude that the MSE+R method can backfire and badly damage a model's generalizability to missing categories. This finding was extremely unexpected and we caution the reader that the code needs to be validated, and the results need to be reproduced with other models. One major concern regarding our data is that the Regression (MSE+R) method wildly deviated from its main trend for the Drop Mild Dementia condition, and the Categorical model did the same for the Drop Moderate Dementia condition. If we had more time, we would have repeated our trials several more times, and gone through our code line-by-line seeking an explanation for these anomalies. Future work in

this area would have to reproduce our results in several trials, after comprehensive debugging efforts.

Our results regarding the loss-landscape of the family of CNN architectures we investigated was much more in line with theoretical expectations. With the thrashing we observe in higher parameter models, we conclude that even though the test set performance is comparable, the high internal weights make these models undesirable.

Resources

Amy Jang's Kaggle Notebook: https://www.kaggle.com/amyjang/alzheimer-mri-model-tensorflow-2-3-data-loading

Our Data and Colab Notebooks: `https://drive.google.com/drive/folders/1CxDISMXk0mMjxudxPCuCz`

References

Hussain E., Hasan M., Hassan S. Z., Hassan Azmi T.,Rahman M. A. and Zavid Parvez M., "Deep Learning Based Binary Classification for Alzheimer's Disease Detection using Brain MRI Images," 2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA), Kristiansand, Norway, 2020, pp. 1115-1120, doi: 10.1109/ICIEA48937.2020.9248213.

"IBM May Sell Watson Health Business Generating $1 Billion Annual Revenue, Says Report." n.d. Business Insider. Accessed April 18, 2021. https://www.businessinsider.in/business/news/ibm-may-sell-watson-health-business-generating-1-billion-annual-revenue-says-report/articleshow/81121827.cms.

McDermott, Matthew B. A., Shirly Wang, Nikki Marinsek, Rajesh Ranganath, Luca Foschini, and Marzyeh Ghassemi. 2021. "Reproducibility in Machine Learning for Health Research: Still a Ways to Go." Science Translational Medicine 13 (586).

Monya Baker. 1,500 scientists lift the lid on reproducibility.Nature News, 533(7604):452, May 2016.