Using Neural Network Classification for the Automated Scoring of Image Responses in TIMSS 2023

FINAL REPORT

Lillian Tyack, Dr. Lale Khorramdel, Dr. Matthias von Davier

Table of Contents

I. Introduction

This report is the fourth and final deliverable of the IEA-funded research project "Using Neural Network Classification for the Automated Scoring of Image Responses in TIMSS 2023". In this report, we describe the code optimization process, open source and public domain tools used in neural network modeling and identify TIMSS 2023 items that can be scored with artificial neural networks (Activity 3, from July 1st to October 31st, 2022). Moreover, a list of conference proposals and journal articles related to the work completed during this project is provided (Activity 4, from June 1st to October 31st 2022). Finally, an example workflow is provided in the appendix to illustrate the R code used for modeling.

The genesis of this research project dates back to a pilot study conducted in 2020 at the TIMSS & PIRLS International Study Center to see whether artificial intelligence could be used to validate human scoring of graphical responses. At the time, artificial neural networks (ANNs) were applied to one released item from the TIMSS 2019 Problem Solving and Inquiry (PSI) "Building" task. This trichotomous item asked students to draw the back wall and sides of a shed on a grid according to given specifications. It was found that the ANN validation technique achieved up to 97.53% agreement with the human raters and the AI models correctly classified 193 responses that were incorrectly classified by human raters, indicating that ANNs can help identify mistakes and inconsistencies in human scoring as well.

This IEA-funded research project aimed to extend the ANN validation technique to eight additional TIMSS 2019 graphical response items to see how the neural networks would perform across a variety of different tasks and possible response outcomes (Activity 2, from January 1st to June 30th 2022) and to implement the procedure operationally in TIMSS 2023 (Activity 3). It was found that the ANN technique was extremely successful when applied to the non-PSI items, with the most accurate models classifying over 99% of the image responses into the appropriate scoring category for dichotomous items, and almost 98% for one trichotomous item. Additionally, the models correctly classified a number of image responses that had been incorrectly scored by human raters, with most of the items having a higher number of incorrectly human-scored responses than responses misclassified by the ANNs.

These results indicate that automated scoring using ANNs is comparable to, and in many cases more accurate, than human raters, and that the validation technique could be used in TIMSS (and other large-scale assessments) as a low-cost and fast way to double-score all graphical response items. The method is also promising for identifying inconsistently scored responses that could be set aside for expert scoring from the mathematics coordinator. In sum, this validation technique will improve the accuracy and consistency of graphical response item scoring in future TIMSS cycles.

II. Code Optimization

During Activity 3, the ANN modeling scripts were revised to be legible to those unfamiliar with the project and to run smoother. The optimization process was guided by Robert C. Martin's "Clean Code: A Handbook of Agile Software Craftsmanship" (2008), with the following steps taken:

- Revision of variable names to be meaningful to unfamiliar readers (e.g., changing the name of a data frame from "df" to reflect what is in the data frame, such as "accuracy_results" or "training_data_information").
- Add comments so that readers can follow the structure of the script and understand what each chunk of code is enacting.
- Remove redundant or old functions that were no longer of use.
- Split larger functions into smaller functions that accomplish only one or two tasks.

Instead of one large script that covers the pre-processing, modeling, and results processes, each stage of modeling has a unique script:

- The pre-processing script converts image html files to PNG files, pre-processes them (e.g., convert to greyscale, increase saturation, etc.), assigns the responses to training and testing samples, and composes the image response arrays.
- The modeling script composes and trains ANNs based on a number of parameters provided by the user.
- The modeling performance script applies the trained models to the validation samples and saves the results, including model accuracy, loss, and F1 scores. It also can save image responses where there was a disagreement between the machine score and human rater score for further review.

The optimized scripts are ready to be used for TIMSS 2023 data collection and will likely require minimal adaptation only.

III. Open Source and Public Domain Tools Used

Multiple free R packages (R Core Team, 2022) formed the basis of the ANN modeling process and were integral to the success of the project. The packages contributed to multiple stages of the modeling process, including pre-processing, modeling, and producing performance statistics.

Image Pre-Processing Packages

webshot: This package was used to take screenshots of the html image response files. During the modeling of the Building PSI item performed prior to this grant project, html files were converted to PDFs manually in Adobe and then converted to PNG files. However, webshot was utilized for this project because it is a fast, automated way of performing batch conversions.

magick: This package was used for image modification. As explained in the second quarterly report, the images had to undergo manipulation so that their features could be better detected by the machine learning algorithms. The *magick* package has multiple functions for modification, including “image_crop()” and “image_chop()” to crop the image responses, “image_channel()” to convert the images to greyscale, “image_contrast()” to sharpen the images, “image_modulate()” to increase saturation of the blacks and whites, “image_convolve()” to add blurring to an image, and finally “image_scale()” to pixelate the image.

EBImage: This package was used to resize image responses prior to placement in the *keras* arrays. The “as_EBImage()” function from the *magick* package was used to convert magick objects to EBImage objects, then the “resize()” function was used to shrink the images to a given pixel height and width (typically 64x64 or 75x75 pixels).

ggplot2: This package was used to plot image responses from *keras* arrays to visualize approximately how they would appear to the CNN models during training.

ANN Modeling Packages

tensorflow: This package was used as the basis for all ANN modeling. It is an open-source package for machine learning, in particular deep learning.

keras: This package is the user interface to *tensorflow* (it operates on top of *tensorflow*), which provides all of the functions for composing and training the ANN models. The “array_reshape()” function reshapes a list of EBImage objects into an array that can be used for training or validating ANN models. The “keras_model_seqential()” function was used to build the CNN models, while the “fit()” function was used to train the models. The “evaluate()” function was used to produce model loss and accuracy on validation data.

ANN Performance Packages

openxlsx: This package was used to create excel workbooks with multiple tabs using the “createWorkbook(),” “addWorksheet()”, “writeData(),” and “saveWorkbook()” functions.

yardstick: This package was used to compute F1 scores to evaluate model performance, using the “recall_vec()” function and the “precision_vec()” function to produce recall and precision estimates.

caret: This package was used to compute kappa and IOCC (improvement over chance criterion) statistics to evaluate model performance, using the “confusionMatrix()” function.

IV. List of TIMSS 2023 Items for ANN Modeling

Graphing Tool Items

In TIMSS 2023, items that use the "graphing tool" will be machine scored using rule-based scoring in python with response coordinate strings that are saved in the raw student data. These items include all of the secured trend items in this grant project, which were required to be human scored in TIMSS 2019 because the response coordinates were not saved for machine scoring. These items are eligible to be validated by ANNs using screenshots of student responses to both refine the python scripts and rescore any "borderline" responses that may appear. "Borderline" responses are those where students have the correct answer present, but additional lines (e.g., stray marks, guiding lines, etc.) make the students' understanding of the item unclear. We have found that the ANNs tend to give "borderline" responses credit while the rule-based machine scoring does not. Thus, there will be disagreements in classifications between the two methods, bringing these "borderline" responses to our teams' attention. We will then be able to manually reclassify them with input from the mathematics coordinator for especially tough cases. This approach aids in maintaining the trend scoring for the items by using the ANNs as an emulator of human scoring while being faster, cheaper, less labor-intensive and potentially more accurate.

The 10 "graphing tool" items, scored using python code and validated with ANNs, will be:

Grade 4

- ME61081A: Trend math item in the Geometry content area worth one point
- ME61081B: Trend math item in the Geometry content area worth one point
- ME61224: Trend math item in the Geometry content area worth one point
- ME71177: Trend math item in the Geometry content area worth one point
- ME71181: Trend math item in the Measurement content area worth one point
- ME71211: Trend math item in the Geometry content area worth one point
- ME81032: New math item in the Geometry content area worth one point
- ME81902: New math item in the Measurement content area worth one point

Grade 8

- ME72119: Trend math item in the Geometry and Measurement content area worth one point
- ME72181: Trend math item in the Geometry and Measurement content area worth one point

Drawing Tool Items

In TIMSS 2023, items that utilize the drawing tool are also eligible to be scored by the ANNs. Three new science items use the drawing tool function and can only be scored based upon image responses. The primary scoring method for these items will be human scoring, with the ANN approach used for validation. One item will have neural networks trained on all available image responses from the TIMSS 2023 field test. The two remaining items have been changed since the field test, and thus the field test responses would not be useful for training. A subset of responses from the data collection sample will be used for training instead. In total, one item is available at Grade 4 and two are available at Grade 8:

Grade 4

- SQ81R03: New science item in the Earth in the Solar System content area worth one point with 10,450 image responses available for training.

Grade 8

- SQ82T02B: New science item in the Earth in the Solar System and Universe content area worth one point.
- SQ82T04A: New science item in the Earth in the Solar System and Universe content area worth one point.

It should be noted that these two of these items were modeled using convolutional neural networks in the TIMSS 2023 field test. More information is provided in the following section.

V. CNN Modeling in the TIMSS 2023 Field Test

All drawing tool items were unable to be scored by human raters for the TIMSS 2023 field test due to technical difficulties involving the bulk export of image responses on the online player system platform. To produce some estimate of difficulty for the "Earth's Motions" PSI task at each grade, convolutional neural networks (CNNs) were used to score a random sample of 1,000 responses for four drawing tool items. The scoring procedure followed the same general steps that were used to model the TIMSS 2019 items: the images underwent some pre-processing to make their features more detectable to the neural networks, the models were trained on a subset of responses and applied to remaining responses, and a subset of response classifications were manually reviewed.

Because these items could not be scored by human raters hired by the countries, a random sample of 200 to 250 responses were human scored by a member of the research team at the TIMSS and PIRLS International study Center for the CNN training sample. Prior to placement in the training arrays, the responses were augmented to artificially increase the training sample size. Augmentation included copying, mirroring, and/or rotating the image responses 180 degrees. This process succeeded in doubling or even tripling the training sample size. Following augmentation, CNNs with two convolutional layers were constructed and underwent 100 epochs of training. Afterwards, the trained CNNs were applied to the remaining image responses and their scored classifications were saved.

For verification, some responses had to be reviewed by a human rater (again, a member of the research team at the TIMSS and PIRLS International Study Center). Any responses scored as incorrect by the CNN were reviewed if the probability of the item being incorrect was less than 90% for SQ82T05 and less than 100% for SQ81R03, SQ81R04B, and SQ81T02B.

The following table displays the percent correct and percent code 7 statistics (where applicable). Additionally, the table includes the model accuracy, which is estimated from the percent of responses in the validation sample where the CNN score did not have to be changed after the manual review (in other words, the percent of responses where the CNN score matched the human rater score for the subset of reviewed responses).

Table 1. Item Review Statistics and Model Accuracy for Select T23 FT Earth's Motions Items

Item	Percentage Correct	Percentage Code 7	CNN Model Accuracy
SQ81R03	16.40%		95.25%
SQ81R04B*	5.30%	9.40%	90.13%
SQ82T02B	13.21%	15.62%	83.98%
SQ82T05*	9.82%		96.62%

**Note: Indicates item removed from the PSI task after item review*

VI. Journal Article

One journal article will be written related to the work completed during this research grant project:

<u>Using Convolutional Neural Networks to Automatically Score Eight TIMSS 2019 Graphical Response Items</u>

Abstract:

Large-scale assessments have used graphical response-based items to measure student ability for decades, but they have yet to implement automated scoring of these responses and instead rely on human scoring alone. To investigate how scores provided by machine learning algorithms compare to those provided by human raters, we applied convolutional neural networks (CNNs) to classify image-based responses from eight TIMSS 2019 items. Our results show that the most accurate CNN models classified over 99% of the image responses into the appropriate scoring category for dichotomous items, and almost 98% for one trichotomous item. Additionally, during the modeling process the CNNs correctly classified a number of image responses that had been incorrectly scored by human raters. For most items, the number of incorrectly human-scored responses was higher than the average number of responses misclassified by the most accurate models. These results suggest that automated scoring using CNNs is comparable to, and in many cases more accurate, than human raters. This paper argues that the machine learning procedure explored could be implemented in international large-scale assessments (ILSAs) as a verification method to improve the accuracy and consistency of graphical response item scores. In lieu of additional human raters, ILSAs could implement CNN-based automated scoring to provide a second set of scores, thus reducing the workload and costs associated with human scoring.

VII. Conference Proposals

Two conference proposals have been submitted related to the work conducted prior to this grant concerning the “Building” PSI item, as well as modeling on select TIMSS 2019 items conducted under this grant:

Conference submission 2: NCME 2023
Automated Scoring of TIMSS 2019 Graphical Responses using Convolutional Neural Networks

Abstract:

The transition of TIMSS to a digital format in 2019 allowed for the inclusion of more innovative item types, including items that ask students to respond using free drawings or graphing tools. Due to the complexity of student responses, these items require human scoring. While human scorers have been the standard for scoring constructed response items in international large-scale assessments (ILSAs), rater effects such as fatigue and leniency can lead to inconsistencies and errors in their scoring. This study measured the potential for machine scoring of graphical response items as a validation method using convolutional neural networks (CNNs) applied to eight TIMSS 2019 items. After image manipulation (e.g., increased contrast, cropping, etc.), CNNs were trained on a subset (20-30%) of student image responses using scores from the human raters, then applied to the remaining image responses unseen by the models. Their classifications were compared to the human ratings, and discrepancies in scores were assessed by an independent human rater. The models were extremely accurate, reaching over 99% accuracy for dichotomous items and 97% accuracy for one trichotomous item across five cross-validation samples. These results indicate that neural networks can be used to validate human scores in ILSAs for image-based responses. Additionally, during the process some responses were identified as being incorrectly scored by the human rater, further evidencing the benefit of using machine learning to improve measurement accuracy.

Conference submission 3: IEA International Research Conference 2023
Using Convolutional Neural Networks to Automatically Score TIMSS 2019 Graphical Response Items

Abstract:

TIMSS 2019 took advantage of its new digital format to include more interactive item types, including items that ask students to respond using graphing tools to produce image responses to assess skills in the Geometry content domain. While traditionally, these items have required human scoring in international large-scale assessments (ILSAs) due to the complex nature of student responses, scores can be influenced by rater effects like fatigue and leniency. Two consequences of rater effects are incorrect scoring and inconsistent scoring, which can impact the validity and reliability of the items. Supported by the IEA Research and Development fund, this study examined the possibility of scoring graphical response items using convolutional neural networks

(CNNs) as a validation method. To evaluate the efficacy of CNNs, eight TIMSS 2019 items were modeled and a second set of machine scores was produced for comparison against human scores. Prior to modeling, image responses were manipulated (increased contrast, conversion to greyscale, and pixelated), to improve feature detection by the neural networks. The CNNs were then trained on a subset (20-30%) of student image responses using scores from the human raters and applied to the remaining image responses unseen by the models.

The models produced were extremely accurate, reaching over 99% accuracy for dichotomous items and nearly 98% accuracy for one trichotomous item (having an additional score category of partial credit) across five cross-validation samples. Discrepancies between CNN classifications and human ratings were assessed by an independent human rater. It was found that several responses were incorrectly scored by the human raters, with more image responses incorrectly scored for the trichotomous item and items with more possible correct response options. After an independent rater review, the average number of misclassified responses for the final models ranged from 11 to 99 for the dichotomous items and 233 for the trichotomous item. By comparison, the average number of incorrectly human-scored responses that were identified ranged from 23 to 601 for the dichotomous items and 227 for the trichotomous item. These results demonstrate that CNNs can perform as well as, and sometimes even better than, human raters because most items had more incorrectly human-scored responses than incorrectly machine-scored responses. The method is cost-effective and time-efficient, requiring only a single independent rater to review a fraction of the image responses to produce a second set of scores. In sum, machine learning with convolutional neural networks is highly accurate. It can be used to validate human scores in ILSAs for image-based responses, thus improving measurement accuracy and reliability.

References

Allaire, J. J., & Chollet, F. (2021). keras: R interface to ‘‘Keras’’ (R Package Version 2.4.0). https://CRAN.R-project.org/package=keras

von Davier, M., Tyack, L., & Khorramdel, L. (2022). Scoring Graphical Responses in TIMSS 2019 Using Artificial Neural Networks. *Educational and Psychological Measurement*. https://doi.org/10.1177/00131644221098021

Kuhn, M. (2021). caret: Classification and Regression Training. (R Package Version 6.0-90). https://CRAN.R-project.org/package=caret

Kuhn, M., & Vaughan, D. (2021). yardstick: Tidy Characterizations of Model Performance. (R Package Version 0.0.9). https://CRAN.R-project.org/package=yardstick

Ole, A. (2022). EBImage: An R package for image processing with applications to cellular phenotypes (R Package Version 4.34.0). https://github.com/aoles/EBImage

Ooms, J. (2021). magick: Advanced graphics and image-processing in R (R Package Version 2.7.2). https://CRAN.R-project.org/package=magick

R Core Team (2022). R: A Language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Schauberger, P., & Walker, A. (2021). openxlsx: Read, Write and Edit xlsx Files. (R Package Version 4.2.4). https://CRAN.R-project.org/package=openxlsx

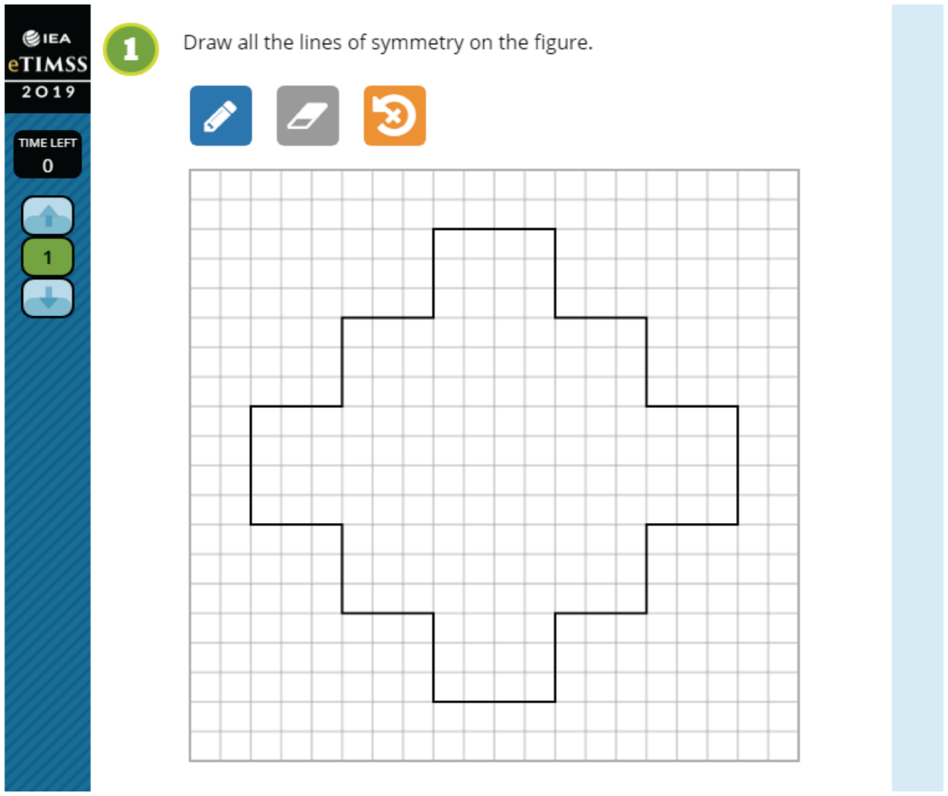
Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. (R Package Version 3.3.5). https://ggplot2.tidyverse.org

Appendix

Table of Contents

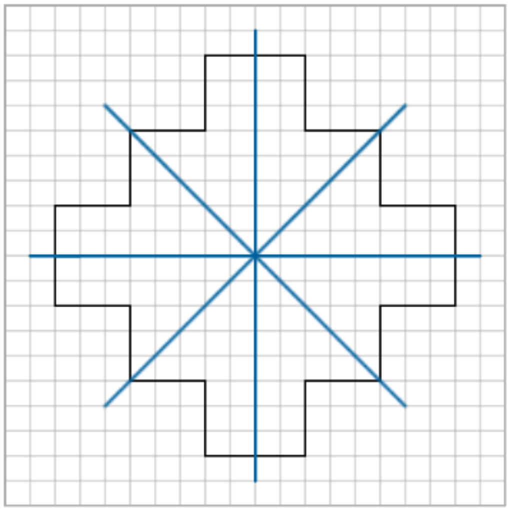
Introduction and Example Item Information

The following document describes the general workflow for modeling with a TIMSS 2019 image response-based item. The example item included is ME52048 which is a dichotomous released item that measures Geometry knowledge at Grade 8. This item asks students to draw four lines of symmetry on a given figure. Students receive credit if they draw four lines of symmetry on the diagram with no other lines drawn. Below is the storyboard of the item:



Note: To access, reuse, reproduce, or translate IEA materials you will need to complete a Permission Request Form. Please reach out to permission.requests@iea.nl for more information.

Below is an example of a correct response:



Stage 1: Pre-Processing

The following are the steps taken during the pre-processing stage:

1. Assign responses to training and validation samples based upon a set sample split. For example, we may select 30% of responses to be in the training sample and the remaining 70% of responses to be in the validation sample.
    a. An example function could be: make_assignment_file(), which would do the following:
        i. Read in data that have student IDs, the name of the images that correspond to their responses, and their human scores
        ii. Loop through each country, within each country and score category assign 30% of responses to the training sample (we call "train" in our code) and 70% to the validation sample (we call "test" in our code)
        iii. Save CSV files of the country assignments (if desired)
        iv. Save RDS file for all of the country data together

Script note: In our modeling script, we also have a function for creating additional sample assignments for cross-validation samples. Our function makes the process run faster, but any user can just repeat this example function additional times with separate modeling folders.

```
sample_split<-0.3 #Proportion of responses to be assigned to training sample
data<-read.xlsx(paste0(data_dir, "ME52048 example score file.xlsx"))
country_list<-unique(data$CNTRYCODE)
make_assignment_file<-function(){
  data$sample_assignment<-NA

  set.seed(1234)
  for (country in country_list){
    country_data<-data %>% filter (CNTRYCODE==country)
    for(score in c(0,1)){
        num_score<-length(country_data$score[country_data$score==score])
        assignment<-sample(c("train", "test"),
                           size = num_score,
                           replace = TRUE,
                           prob = c(sample_split, 1-sample_split))
        country_data$sample_assignment[country_data$score==score]<-assignment
    }
    file_name<-paste0(save_dir, "CSVs/", country, " sample assignment.csv")
    write.csv(country_data, file=file_name, row.names=F)

    data$sample_assignment[data$CNTRYCODE==country]<-
country_data$sample_assignment
  }
  saveRDS(data, paste0(save_dir, "List of image responses with
assignment.rds"))
}
make_assignment_file()
```

2. Next, we make the data arrays for modeling, which involves reading in the image responses, modifying them to enhance feature detection, and placing them in a *keras* array. We have found that it saves time to create one array that includes all data, and then reshape that array into separate training and validation arrays. This is especially useful when testing different training sample sizes and going through multiple iterations of modeling. Three example functions could be used:
    a. The modify_image_responses() function, which would do the following:
        i. Crop the image responses
        ii. Convert image responses to greyscale
        iii. Increase saturation and contrast of images
        iv. Pixelate the images

Script note: In our modeling script, we have to move the image responses from one location one the server to another location, so in the process we apply all modifications except for pixelation. Thus, images receive pixelation only when they are placed in the array.

```
chop_dims<-"11x47"
crop_dims<-"383x383"
pixel_width = 75
pixel_height = pixel_width

modify_image_responses<-function(image_list){
  #Crop images
  image_list<-lapply(image_list, image_chop, chop_dims)
  image_list<-lapply(image_list, image_crop, crop_dims)

  #Convert to greyscale
  image_list<-lapply(image_list, image_channel, channel = "C")

  #Increase contrast
  image_list<-lapply(image_list, image_modulate, saturation=200)
  image_list<-lapply(image_list, image_contrast, sharpen=1)

  #Add pixelation
  image_list<-lapply(image_list, image_scale, paste0("x", pixel_height))
  image_list<-lapply(image_list, image_scale, "x500")

  return(image_list)
}
```

b. The create_all_data_array(), which would do the following:
    i. Read in the data assignment file
    ii. Read in the image responses
    iii. Call the modify_image_responses() function to manipulate responses
    iv. Resize the image responses and arrange into an array
    v. Return the array of images and their corresponding labels (human rater scores)

Script note: In our modeling script, we actually loop the formation of the arrays. We've found that when dealing with thousands of image responses, it is faster to read in 500 to 1,000 at a time, reshape them and place them in small arrays. Once the small arrays are created they are then bound together into one larger array.

```
create_all_data_array<-function(){
  data<-readRDS(paste0(save_dir, "List of image responses with 
assignment.rds"))
  pics<-list()
  image_files<-paste0(image_dir, data$CNTRYCODE, "/", data$image_name, 
".png")
  #image_files<-image_files[1:25]
  for (i in 1:length(image_files)){
    pics[[i]]<-image_read(image_files[i])
  }

  pics<-modify_image_responses(pics)

  #Convert to EBImage object, resize, and convert to keras array
  pics<-lapply(pics, as_EBImage)
  pics<-lapply(pics, resize, w=pixel_width, h=pixel_height)
  pics<-lapply(pics, array_reshape, c(pixel_width, pixel_height, 1))
  images<-rlist::list.rbind(pics)
  images<-array_reshape(images, c(nrow(images), pixel_width, pixel_height, 
1))
  return(list(images=images, labels=data$score))
}
arrays<-create_all_data_array()
images=arrays$images
```

```
labels=arrays$labels
save(images, labels, file=paste0(save_dir, "all data arrays.RData"))
```

c. The create_sample_array() function, which could take the argument “sample_type”, which would indicate which sample array would be created, either the training or the validation sample. This function would do the following:
   i. Load the arrays for all of the data
   ii. Randomize the data and reduce it to just the sample type
   iii. Go through each response in the data, taking the image response object from the full data array and placing it into a new sample array

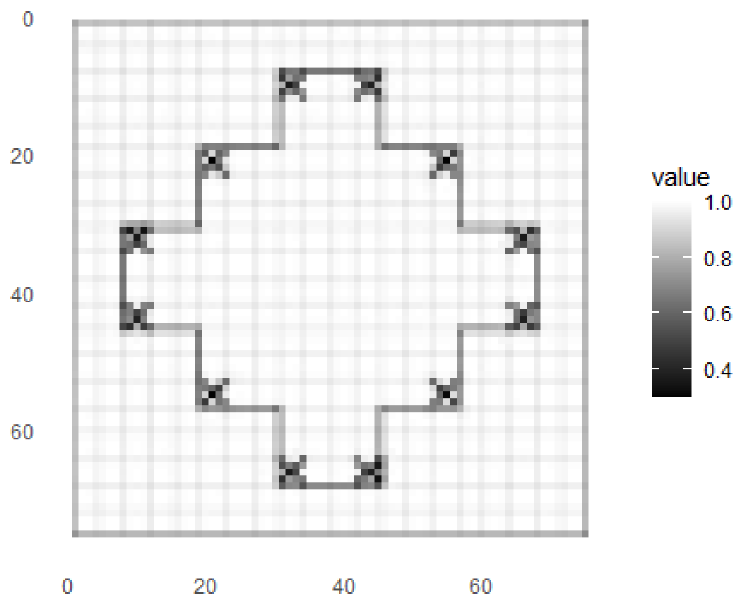
Script note: In our modeling script, we again create smaller arrays to merge into a larger array to quicken the process. Additionally, there is an option to create the cross-validation sample arrays automatically, but a user could just run this example function multiple times instead.

```
data<-readRDS(paste0(save_dir, "List of image responses with 
assignment.rds"))
data$id_num<-row.names(data)

make_sample_arrays<-function(sample_type){
  sample_data<-data%>%filter(data$sample_assignment==sample_type)
  set.seed(1234)
  sample_data<-sample_data[sample(nrow(sample_data), nrow(sample_data)),]
  all_data_array_nums<-sample_data$id_num
  all_data_array_nums<-as.numeric(all_data_array_nums)

  for (i in 1:length(all_data_array_nums)){
      if (i==1){
        ims<-images[all_data_array_nums[i], 1:pixel_height, 1:pixel_width, 1]
      } else if (i==2) {
        ims<-abind::abind(ims, images[all_data_array_nums[i], 1:pixel_height, 
1:pixel_width, 1], along=0)
      } else {
        ims<-abind::abind(ims, images[all_data_array_nums[i], 1:pixel_height, 
1:pixel_width, 1], along=1)
      }
  }
  sample_images<-array_reshape(ims, c(nrow(ims), pixel_width, pixel_height, 
1))
  return(list(images=sample_images, labels=sample_data$score))
}

#Create training sample arrays
arrays<-make_sample_arrays(sample_type="train")
train_images<-arrays$images
trainLabels<-arrays$labels
save(train_images, trainLabels,
     file=paste0(save_dir, "train data arrays.RData"))

#Create validation sample arrays
```

```
arrays<-make_sample_arrays(sample_type="test")
test_images<-arrays$images
testLabels<-arrays$labels
save(test_images, testLabels,
     file=paste0(save_dir, "test data arrays.RData"))
```

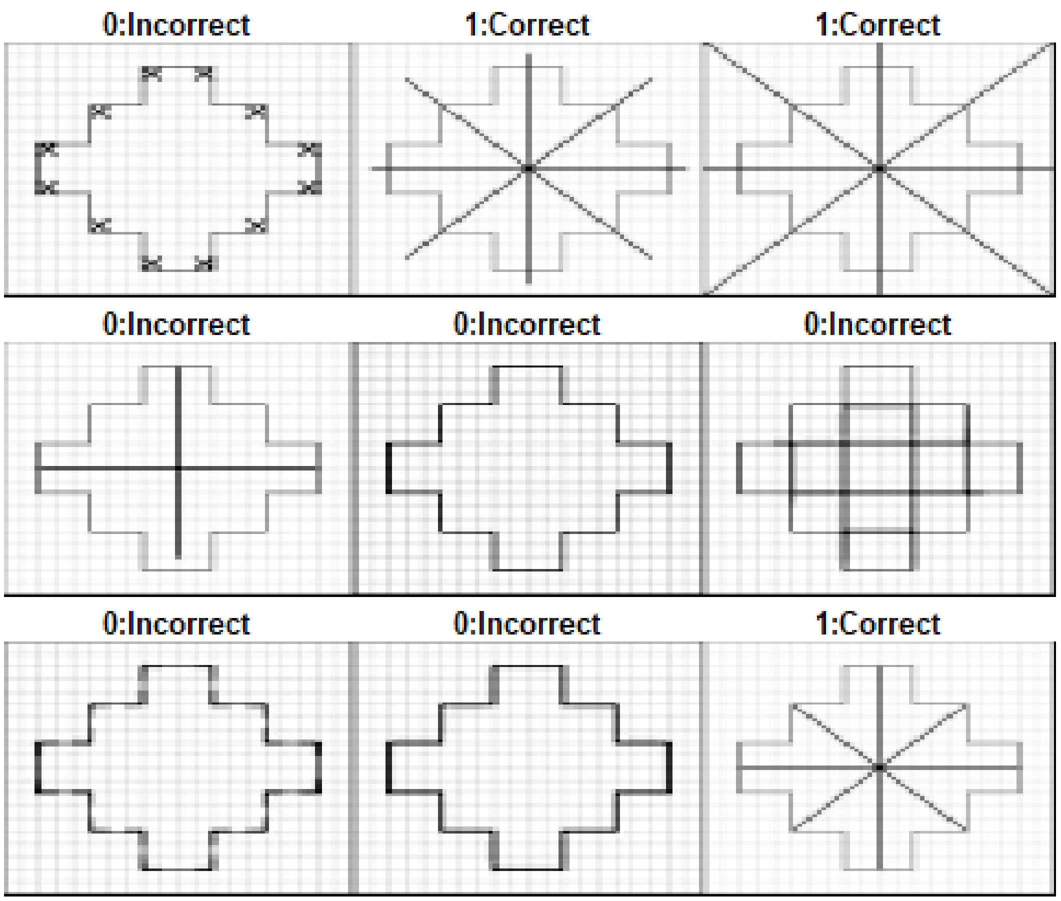
3. Finally, we can plot the image responses in the arrays to get a rough estimate of how the neural networks will "view" them:
   a. We can plot just one image

```
image <- as.data.frame(images[1, , , ])
colnames(image) <- seq_len(ncol(image))
image$y <- seq_len(nrow(image))
image <- gather(image, "x", "value", -y)
image$x <- as.integer(image$x)

ggplot(image, aes(x = x, y = y, fill = value)) +
    geom_tile() +
    scale_fill_gradient(high = "white", low = "black", na.value = NA) +
    scale_y_reverse() +
    theme_minimal() +
    theme(panel.grid = element_blank())   +
    theme(aspect.ratio = 1) +
    xlab("") +
    ylab("")
```



b. Or we can plot multiple images at a time

```
par(mfcol=c(3,3))
par(mar=c(0, 0, 1.5, 0), xaxs='i', yaxs='i')
for (i in 1:9) {
    img <- images[i, , , ]
    img <- t(apply(img, 2, rev))
    image(1:pixel_width, 1:pixel_height, img, col = gray((0:255)/255), xaxt =
'n', yaxt = 'n',
          main = paste(class_names[grepl(labels[i], class_names)]))
}
```



Stage 2: Modeling

During the modeling process, the training samples are loaded and models are constructed and trained on the training sample array. Any user can create a model, with the most basic convolutional neural network having one convolutional layer and two dense layers. A basic model, for example, would be the following:

```
model<-keras_model_sequential()

model %>%
  layer_conv_2d(filters=32, kernel_size=c(3,3), activation='relu',
input_shape=
                  c(pixel_width, pixel_height,1)) %>%
  layer_max_pooling_2d(pool_size=c(2,2)) %>%
  layer_flatten()%>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units=3, activation='softmax')


model %>% compile(
  optimizer = 'nadam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
model %>% fit (train_images, trainLabels, epochs=25, verbose=FALSE,
view_metrics=TRUE)
```

To enable us to run multiple models quickly, we wrote a function named run_CNN_models(), which constructs and trains one or more CNN models according to user specifications, including:

- The number of dense (fully-connected) layers desired, can either be a number (e.g., “2”) or a vector (e.g., “c(2, 3)”), which would run multiple models, (one with two dense layers, and one with three dense layers). The default is 2.
- The unit of the first dense layer, default is 128.
- The number of units by which subsequent dense layers should increase, default is 64.
- The number of convolutional layers desired. Like with the argument for the dense layers, the user can specify either a single number or a vector.
- The number of filters of the first convolutional layer, the default is 32.
- The number of filters by which subsequent convolutional layers should increase, default is 32.
- Max pooling size, default is 2x2
- Padding used, can be either “zero”, “no”, or “both,” which produces models with zero padding and no padding, default is “both”.
- Whether dropout layers should be included, default is FALSE.
- The dropout rate of the dropout layers, default is 0.25.

- Whether the training plot should be shown, the default is FALSE.

Say we want to run four convolutional models total, two with one convolutional layer and two with two convolutional layers, both testing zero and no padding. We would set up the function as follows:

```
source(paste0(root_dir, "CNN Modeling Example_Modeling Functions.R"))
opt<-"nadam"
epochs<-25
CNNs<-run_CNN_models(dense_layers=2,
                       dense_base_unit=128,
                       dense_unit_increase=64,
                       convolutional_layers=c(1,2),
                       c_filters=32,
                       c_filters_increase=32,
                       pool_size=c(2,2),
                       padding="both",
                       with_dropout=FALSE,
                       dropout_rate=0.25,
                       view_plot=FALSE)
```

When the function is run, the models are returned as a list (along with names to save the models under and the time it took to run the models). The models can then be saved to a folder so they can be used again in the future:

```
models<-CNNs$models
model_names<-CNNs$model_names

for (i in 1:length(models)){
  filename<- paste0(save_dir, opt," ", model_names[i]," (", epochs, " 
epochs)", ".hdf5")
  save_model_hdf5(models[[i]], filename, overwrite = TRUE, include_optimizer 
= T)
}
```

Stage 3: Evaluating Model Performance

Modeling performance can be evaluated in a number of ways: overall accuracy, model loss (how well the model fits the data), precision, recall, F1-scores, improvement over chance criterion, etc. In our modeling scripts, we produce all of these statistics in addition to classification accuracy by country and classification accuracy for incorrectly human-scored responses. For simplicity, our example will just cover the basic model performance statistics.

1. First, we can produce the basic model performance metrics:

a. Begin with accuracy and loss using the *keras* evaluate() function:

```
model<-models[[1]]
classify_test_samp<-model %>% evaluate(test_images, testLabels)
loss<-classify_test_samp[1]
accuracy<-classify_test_samp[2]
```

b. Next, compute the precision, recall, and F1-Scores using the *yardstick* package:

```
compute_precision_recall_f1<-function(model){
  predicted_probs <- model %>% predict(test_images, verbose=0)
  class_pred <- as.array(k_argmax(predicted_probs, axis=-1))
  rater_scores<-as.factor(testLabels)
  classifications<-as.factor(class_pred)
  levels(classifications)<-c("0", "1")

  recall<-yardstick::recall_vec(truth=rater_scores,
estimate=classifications)
  precision<-yardstick::precision_vec(truth=rater_scores,
estimate=classifications)
  f1_score<-(2*precision*recall)/(precision+recall)

  return(list(recall=recall, precision=precision, f1=f1_score))
}

prf1<-compute_precision_recall_f1(model)
recall<-prf1$recall
precision<-prf1$precision
f1_score<-prf1$f1
```

c. Compute the improvement over chance criterion (IOCC— how much better a model classified the responses in the validation sample than if the classification had been done by chance) using the *caret* package:

```
compute_IOCC<-function(model){
  predicted_probs <- model %>% predict(test_images, verbose=0)
  class_pred <- as.array(k_argmax(predicted_probs, axis=-1))
  rater_scores<-as.factor(testLabels)
  classifications<-as.factor(class_pred)
  levels(classifications)<-c("0", "1")

  cm<-caret::confusionMatrix(data=classifications,
reference=rater_scores)
  Ho<-length(classifications[classifications==rater_scores])
  N<-length(testLabels)
  t<-table(testLabels)
  if (n_score_categories==3){
    He<-((1/3)*(t[1]))+((1/3)*(t[2]))+((1/3)*(t[3]))
  } else {
    He<-((1/2)*(t[1]))+((1/2)*(t[2]))
  }
  IOCC=((Ho/N)-(He/N))/(1-(He/N))
  return(IOCC)
}
IOCC<-compute_IOCC(model)
```

d. Finally, put all of the results together into a data frame:

```
num_incorrect=length(testLabels[testLabels!=as.array(k_argmax(model
%>% predict(test_images, verbose=0), axis=-1))])
results<-data.frame(loss, accuracy, precision, recall, f1_score,
IOCC, num_incorrect)
row.names(results)[1]<-"model"
results
```

```
##          loss    accuracy precision recall   f1_score   IOCC
num_incorrect
## model    0.0541  0.9857   0.9946    0.9786  0.9865    0.9713  10
```

e. To speed up the process, we can put these capabilities together into a function named model_performance_statistics(), which could take the argument “model”, which is a trained *keras* model. After creating this model, we could run it for every model produced and save the results in an excel file:

```
model_performance_statistics<-function(model){
  classify_test_samp<-model %>% evaluate(test_images, testLabels,
verbose=0)
  loss<-classify_test_samp[1]
  accuracy<-classify_test_samp[2]


  prf1<-compute_precision_recall_f1(model)
  recall<-prf1$recall
  precision<-prf1$precision
  f1_score<-prf1$f1

  IOCC<-compute_IOCC(model)


responses_incorrect=length(testLabels[testLabels!=as.array(k_argmax(
model %>%       predict(test_images, verbose=0), axis=-1))])
  results<-data.frame(loss, accuracy, precision, recall, f1_score,
IOCC, responses_incorrect)
  return(results)
}
```

```r
for (i in 1:length(models)){
  model_results<-model_performance_statistics(model=models[[i]])
  row.names(model_results)[1]<-model_names[i]
  if (i==1){
    all_results<-model_results
  } else {
    all_results<-rbind(all_results, model_results)
  }
}
write.xlsx(all_results, file=paste0(save_dir, "Model performance
statistics.xlsx"), rowNames=T)
```

2. Another helpful document to have may be an excel workbook with the classifications by score category for each model produced. To create this file, we could use a function called score_category_classifications(), which could take the argument "model" (a trained *keras* model). After creating this model, we could run it for every CNN produced and save the results in an excel workbook with a separate page for each model:

```r
score_category_classifications<-function(model){
  score_class<-data.frame(matrix(ncol=3, nrow=3))
  names(score_class)<-c("Correctly_classified", "Misclassified", "Total")
  row.names(score_class)<-c("0: Incorrect", "1: Correct", "Total")

  predicted_probs <- model %>% predict(test_images, verbose=0)
  CNN_scores <- as.array(k_argmax(predicted_probs, axis=-1))
  rater_scores<-testLabels
  classifications<-data.frame(rater_scores, CNN_scores)
  classifications$match<-
classifications$rater_scores==classifications$CNN_scores

  score_class$Total[1]<-
length(classifications$rater_scores[classifications$rater_scores==0])
  score_class$Total[2]<-
length(classifications$rater_scores[classifications$rater_scores==1])
  score_class$Total[3]<-length(classifications$rater_scores)

  incorrect_match<-
length(classifications$rater_scores[classifications$rater_scores==0&classi
fications$match==TRUE])
  correct_match<-
length(classifications$rater_scores[classifications$rater_scores==1&classi
fications$match==TRUE])
  score_class$Correctly_classified[1]<-incorrect_match
  score_class$Correctly_classified[2]<-correct_match

  score_class$Misclassified[1]<-score_class$Total[1]-incorrect_match
  score_class$Misclassified[2]<-score_class$Total[2]-correct_match
  score_class$Correctly_classified[3]<-
length(classifications$match[classifications$match==TRUE])
  score_class$Misclassified[3]<-
length(classifications$match[classifications$match==FALSE])

  return(score_class)
}



score_class_workbook<-createWorkbook()
for (i in 1:length(models)){
  classification_results<-
score_category_classifications(model=models[[i]])
  addWorksheet(score_class_workbook, model_names[i])
  writeData(score_class_workbook, model_names[i], classification_results,
rowNames = T)
  setColWidths(score_class_workbook, model_names[i], cols =
1:ncol(classification_results), widths = "auto")
}
saveWorkbook(score_class_workbook, file=paste0(save_dir, "Classifications
by score category.xlsx"), overwrite=T)
```

3. A final document that we should produce is the assignment file with the machine scores added. This way, we can see which specific responses have misclassifications according to a given model.

```
#Read in the assignment file
data<-readRDS(paste0(save_dir, "List of image responses with
assignment.rds"))

#Narrow down to just responses in validation sample
data<-data%>%filter(sample_assignment=="test")

#Reorder rows to match the order in array
set.seed(1234)
data<-data[sample(nrow(data), nrow(data)),]
all.equal(data$score, testLabels) #Make sure labels and scores match

## [1] TRUE



data<-data[1:6]
names(data)[6]<-"human_score"
for (i in 1:length(models)){
  model<-models[[i]]
  predicted_probs <- model %>% predict(test_images, verbose=0)
  CNN_scores <- as.numeric(k_argmax(predicted_probs, axis=-1))
  data[,6+i]<-CNN_scores
  names(data)[6+i]<-paste0(model_names[i], "_score")
}
write.xlsx(data, file=paste0(save_dir, "Validation sample information with
machine scores.xlsx"), overwrite=T)
```

4. The last step would be to review any responses where there are disagreements between the human rater scores and the CNN scores. To review responses with mismatching scores, select a model that was run and compare the human scores and the machine scores by adding a new column, for example "match". For any responses where "match" is false, we can copy the image response file from the image directory to a subdirectory in the modeling folder.

```
selected_model<-models[[2]]
selected_model_name<-model_names[2]

data<-read.xlsx(paste0(save_dir, "Validation sample information with
machine scores.xlsx"))
data_sub<-data[1:6]
col_name<-paste0(selected_model_name, "_score")
col_name<-gsub(" ", ".", col_name)
data_sub<-cbind(data_sub, data[[col_name]])
names(data_sub)[7]<-paste0(selected_model_name, "_score")
data_sub$match<-data_sub[,6]==data_sub[,7]

mismatching_responses<-data_sub%>%filter(match==FALSE)

dir.create(paste0(save_dir, selected_model_name, " mismatching
responses/"))



for (row in 1:nrow(mismatching_responses)){
  file.copy(from=paste0(image_dir, mismatching_responses$CNTRYCODE[row],
"/", mismatching_responses$image_png[row]),
            to=paste0(save_dir, selected_model_name, " mismatching
responses/", mismatching_responses$CNTRYCODE[row], "_",
mismatching_responses$image_png[row]))
}
```