The Effects of Dataset Conditions on The Predictive Accuracy of ResNet152 for Classifying *Yu-Gi-Oh! Trading Card Game* Cards & Artwork

Michael Mays
mfmays@wisc.edu

Daniel Halberg
dhhalberg@wisc.edu

Abstract

The impact of dataset conditions on the classification accuracy of convolutional neural networks is well-studied. This report deploys a pre-trained deep neural network to classify Yu-Gi-Oh! Trading Card Game cards. We augment existing literature by examining how dataset effects manifest in a ResNet152 network trained on four combinations of Yu-Gi-Oh! card image type and dataset composition. Specifically, we find that dataset size and classification task (jointly) are the primary determinant of ResNet152's predictive accuracy, while our results for image type are inconclusive. We also present a compelling example of the extent to which information relevance mediates potential dataset size effects.

1. Introduction

The *Yu-Gi-Oh! Trading Card Game* (hereafter, *YGO* or "the *TCG*") is a competitive collectible card game launched in North America during 2002. *YGO* games (called "duels") involve two players who take turns drawing and playing cards according to the game's rules and card-specific conditions. Since its launch, over 10,000 distinct trading cards have been created for the *TCG*, each falling into one of three broad *card types*: monster cards, spell cards, and trap cards.

YGO cards also increasingly belong to one of 200+ *archetypes*, which are groups of related monster, spell, and/or trap cards that explicitly reference each other in their card effects. Archetypes tend to have similar artwork, a common naming structure, and card effects that strategically synergize well. Monsters within an archetype also tend to share one or more of these characteristics: race, primary type, secondary type, element, level/rank/link rating. For example, figure 6 (see additional figures in appendix) shows three cards (one each of monster, spell, and trap) from the "Dark Magician" archetype. These cards are part of the "Dark Magician" archetype because they explicitly reference "Dark Magician" in their card text (the box at the bottom of the card). They all share a naming structure encompassing Dark Magician, "Magician Girl" cards, and possessive "Magician's" cards, while their monsters share the Spellcaster race and (almost always) the DARK element.

Between these two extremes—grouping cards into 3 classes vs. 200+—there are many ways to divide *YGO* cards that yield a different number of classes. Of particular note, each card also has one or more *types* (distinct from *card* types) that are given at the top of the card's text box in order of decreasing 'priority.' The first type listed after the card's *race* (what kind of creature a monster card represents, such as Beast, Warrior, or Spellcaster) is the card's *primary type*. Primary type determines how a card functions (which rules govern how it can be played), and most primary types have distinct card border colors/hues. For example, link monsters have dark blue borders, ritual monsters' are light blue, fusion monsters' are violet, and trap cards' are purple.

The goal of this report is to assess and compare the ability of an existing convolutional neural network (CNN) to classify *YGO* cards using datasets that vary in size (that is, the number of examples) and information density (that is, the amount of classification-relevant data contained in the image). We aim to demonstrate the effect of various dataset conditions on classification accuracy using a dataset of *YGO* cards. Specifically, we explore the ability of ResNet152 to accurately classify four *YGO* card datasets: all full cards, all card artwork, full cards from only 'large' archetypes (see Section 4.1), and card artwork from only 'large' archetypes.

It is important to note that generalizability *per se* is not a primary goal of this report; as discussed in

the next Section, numerous articles exist demonstrating the effects of dataset and image information density on classification accuracy. Rather, we present a novel case study that augments existing literature by comparing how these effects manifest in CNNs trained on (1) images expressly designed to be dense with classification information (full *YGO* cards), and (2) images stripped of most—but not all—of this information. We further tease out any effects by comparing the two model–dataset combinations' classification accuracies in two tasks of varied difficulty. After briefly surveying related work in Section 2, we outline the architecture of ResNet152 (Section 3) before detailing the experiments we conducted (Section 4). We lay out our results in Section 5 and thereafter conclude the report (Section 6).

2. Related Work

Deep learning literature is rife with comparisons of different network architectures' accuracies on a given data set (see, for example, [2]; [14]; [7]). Likewise, there exist numerous studies showing the effects of various dataset conditions on a given network's ability to accuracy classify examples ([6]; [10]; [8]; and [15], to name a few). Work on these two topics is so ubiquitous and fundamental to deep learning writ large that general knowledge of the results—more data tends to improve predictive accuracy (up to a point), newer network architectures tend to perform better than older ones (up to a point), and so on—is assumed for the purposes of paper.

Still, the above-referenced papers present only broad results; the body of work directly related to deep neural networks for classifying *YGO* or other card game images is more limited. Konami Holdings Corporation, the company that owns *YGO*, has released the *Yu-Gi-Oh! Neuron* phone application, which includes augmented reality card recognition [1]. Following *Neuron*'s release, Lowhur sought to imitate its functionality with an application that implements a deep neural network for one-shot learning [4]. While such an application could theoretically be used to categorize cards by primary type and/or archetype (by identifying the card and then simply looking up the card's primary type or archetype), Lowhur does not do so. Finally, GitHub user `chronoreaper` utilized a deep learning model to train an artificial intelligence that builds decks for and plays a *YGO* video game, which incorporates some elements of card recognition [3].

3. Proposed Method

All experiments make use of a pre-trained ResNet152 model implemented in `pytorch`. This is a convolutional neural network whose name is derived from its structure: it makes use of a *network* of *residual* blocks. The architecture of these blocks is shown in figure 1. Let $\mathbf{x}$ be the input and denote by $f(\mathbf{x})$ the mapping function that the model aims to learn. In a traditional CNN (the left diagram in figure 1), convolutional 'blocks' map $\mathbf{x} \mapsto f(\mathbf{x})$; thus, $\mathbf{x}$ is the sole determinant of $f(\mathbf{x})$, which is the value then fed to the activation function $\sigma()$. In a ResNet residual block (the right diagram in figure 1), the 'block' instead maps the input onto its residual $f(\mathbf{x}) - \mathbf{x}$ while the right-most line—called a *residual connection*—carries the input $\mathbf{x}$ (unmodified) to the addition operator. In ResNet, this is done via a 1×1 convolution. It is added to the block's output $f(\mathbf{x}) - \mathbf{x}$; the result, $f(\mathbf{x})$ is then fed to the activation function. ResNet's residual block design enables inputs to forward propagate more quickly across layers via residual connections.

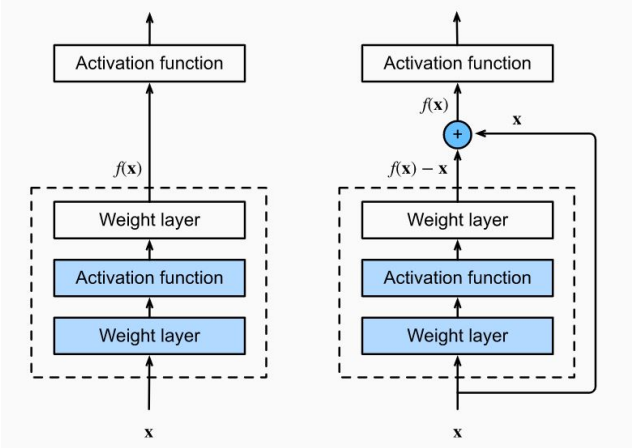

Figure 1. Diagram of residual block architecture. Source: [13]

An (abbreviated) diagram of ResNet152's architecture is shown in figure 2. The initial input is a 7×7 convolutional layer with 3 input channels, 64 output channels, and a stride of 2. This is followed by batch normalization, a ReLU activation function, and then a 3×3 maximum pooling layer with stride 2. Thereafter, each set of blocks (or *module*) continues as follows: the first residual block for each module doubles the number of channels (relative to the previous module), subsequent blocks in that module have the same number of input and output channels, and the height and width are halved. This continues until there are

152 total layers in the network. These convolutional layers culminate in an average pooling layer, then a size-1000 fully-connected layer, and finally a softmax activation function. The output of the network is then used to calculate cross-entropy loss, and the errors are used to update the network's weights.

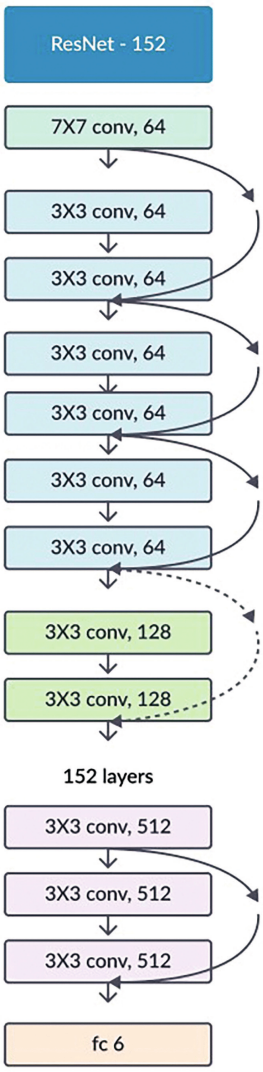

Figure 2. ResNet152 architecture diagram. Source: [5]

The same hyperparameter values (e.g., batch size, epoch count, etc.), optimizer, and learning rate scheduler were used for all four experiments described in the next section. The one exception was learning rate, which was tuned for each experiment separately; our goal is to compare the 'best' version of ResNet152 under the experimental conditions, so this is necessary for fair comparison. For the sake of space, these details are given in table 2.

4. Experiments

The four sets of experimental conditions are shown in table 1. These correspond to every combination of two, two-level experimental factors: image (full card or artwork) and data subset (all cards or 'large' archetypes only). The two different sets of targets—primary type and archetype—were also varied, but we did not fit every image–subset–target combination for two reasons. First, the all-cards–archetype subset–target combination is unworkable, since most cards do not belong to one of the 'large' archetypes. Second, the number of target classes is only meant to adjust the network's classification difficulty under each of the four experimental conditions, it is not intended to be a full experimental factor in itself. In other words, we primarily aim only to demonstrate how varying dataset size and image information density impacts classification accuracy; the two sets of targets merely act as illustrative examples on opposite ends of the classification difficulty spectrum.

		Target (Data subset)	
Variable	Image	Primary Type (All cards)	Archetype ('Large' arch.)
No. classes		Less	More
Dataset size		More	Less
Image info.	Full card	More	
	Artwork	Less	

Table 1. Experimental parameters. Rows represent the two different information densities (full vs. artwork). Columns represent the two different dataset sizes (all vs. 'large' archetypes).

Specifically, we address the following research questions:

1. **Dataset size & classification task**: For a given image type (either full cards or artwork only), how do the size of the *YGO* dataset (all cards vs. 'large' archetype cards) and classification task (either primary type or 'large' archetypes) jointly impact ResNet152's classification accuracy?

2. **Image type**: For a given size of *YGO* dataset (either all cards or 'large' archetype cards) and classification task (either primary type or 'large' archetypes), how does image type (full cards vs. artwork only) impact ResNet152's classification accuracy?

Hyperparameter/ Setting	Value(s)	Comment
Random seed	453	
Batch size	32	
Epochs	40	
Optimizer	Adam	
Learning rate	Full cards, primary type: 0.0005 Full cards, archetype: 0.0003 Artwork, primary type: 0.0001 Artwork, archetype: 0.0001	
Scheduler	Reduce learning rate on plateau Factor: 0.2	Reduces learning rate by `Factor` when minibatch loss stops improving for 10 epochs. New LR = LR $*$ Factor

Table 2. Model hyperparameter values.

Transform	Value(s)/Range	Comment
Resize	Full card: 312x211	HxW in px, half-sized
Random resized crop	Artwork: 272x322	HxW in px, each is the smaller image format's value for that dimension. This is akin to randomly shifting pendulum cards horizontally and non-pendulum cards vertically before cropping.
Random color jitter	Brightness: (0.75, 1.5) Contrast: (0.75, 1.5) Saturation: (0.75, 1.5) Hue: (0.9, 1.1)	Multiplier uniformly chosen from range (min, max)
Random flip	Horizontal: $p = 0.5$ Vertical: $p = 0.5$	
Random affine transformation	Interpolation: Bilinear	
	Translate: (0.2, 0.2)	Max. (H, W) shift (prop. of size)
	Shear: (0, 10)	Degrees, both x and y (separately)
Normalize	Mean: (0.485, 0.456, 0.406) Std. dev.: (0.229, 0.224, 0.225)	(R, G, B), see `https://pytorch.org/vision/stable/models.html`

Table 3. Data augmentation settings.

4.1. Dataset

Card data for 11,149 *YGO* cards (every *TCG* card as of February 21, 2021) was downloaded from the `YGOPRODeck` database [12]. Then, images for each card were downloaded from the database's API in Python using `requests`. Each card was then cropped down to its artwork using `PIL`. Note that there are two card formats: a "regular" card format, which is used for most cards; and a "pendulum" card format, which is used for pendulum cards. The artwork for these formats have different shapes and sizes. An example of a pre- and post-crop card for both formats is shown in figure 7 (see additional figures in appendix). This resulted in 11,149 RGB full-card images (614H x 422W), plus 11,149 RGB artwork images: 277 pendulum cards (272x367) and 10,872 regular cards (320x322). We will refer to the complete set of full-card images and cropped images as the 'full type' and 'artwork type' datasets, respectively. These datasets had 18 classes (one for each primary type) and reflect the left column of table 1. The distribution of these classes is shown in figure 3.

For archetype-related experiments, a subset of 3,451 cards was used containing all cards in the 107 archetypes with 20 or more member cards. This 20-card cutoff was chosen for two reasons. First, it removes non-archetype cards and those in archetypes too small for the network to feasibly classify, and thereby increases the examples-per-archetype ratio; preliminary attempts to classify the full dataset by archetype yielded accuracies so low as to be pure noise. Second, larger archetypes tend to be more distinct (that is, the 'gimmick' that relates cards in larger archetypes is typically more well-defined), so the full-card images for these archetypes contain more information that the network could theoretically use to classify examples. We will refer to this dataset generally as the 'large archetypes' dataset, and to its full-card and artwork variants as the 'full archetype' and 'artwork archetype' datasets, respectively. These datasets had 107 classes (one for each 'large' archetype) and reflect the right column of table 1. The distribution of archetype class sizes is shown in figure 3.

After all cards were downloaded and cropped, a train/validation/test split was created for each dataset–target (experimental) combination. Both 'type' datasets were split (stratified by class) with a train/validation/test ratio of 80/10/10, respectively. For the 'archetype' datasets, the split (stratified by class) was 60/20/20, respectively. These split sizes differ to account for the lower example-to-class ratio in the 'archetype' datasets. Finally, data augmentation was implemented to mitigate overfitting. The same data augmentation settings were used for every experiment

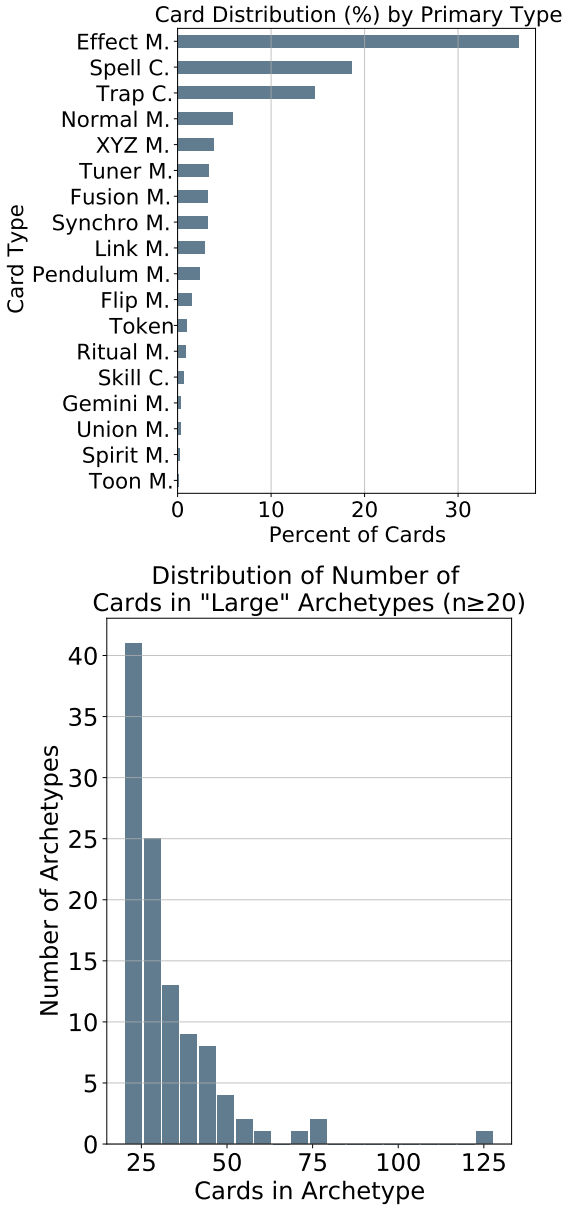

Figure 3. Top: Distribution of primary types. "M." is short for "Monster" and "C." is short for "Card".; Bottom: Distribution of archetype size among 'large' archetypes ($n \geq 20$).

to prevent confounding; these are given in table 3.

4.2. Hardware & Software

Work was done in Python via Google Colab in order to (1) enable easier collaboration between the authors, and (2) make use of Google GPU resources. This was accessed via our personal laptops. We made use of `PyTorch` and the pre-trained ResNet152 model implemented in `torchvision` for all experiments; we also used the latter for data augmentation (for details, see

table 3 in additional figures appendix).

Since both datasets are heavily class-imbalanced, all data loaders included the `ImbalancedDatasetSampler` found in `torchsampler` [11]. This over- and under-samples from under- and over-represented classes, respectively, in order to create a uniform distribution of classes in the train, validate, and test datasets. Data wrangling was done with `Pandas` and `NumPy`, and splitting was done with `train_test_split` in `sklearn`. Card data objects were serialized for speed and reproducibility purposes using `pickle`. Plots were created using `matplotlib` based on modified code from Dr. Sebastian Raschka's `helper_plotting.py` [9]. Finally, to ease implementing the experimental conditions, reduce possible sources of human error, and improve reproducibility, 'wrapper' functions and classes were created; under the hood, these call functions from Dr. Sebastian Raschka in `helper_dataset.py`, `helper_evaluation.py`, and `helper_train.py` [9].

5. Results and Discussion

This section first lays out the results model-by-model before addressing each research question; for reasons explained below, we begin with the second research question. We begin with the full-card primary type combination because it will serve as a baseline for comparisons. The minibatch loss and running average loss for this network are shown in blue in figure 4. Clearly, this combination minimized the cross-entropy loss function both the fastest and most completely among all four image–dataset combinations. This is reflected in its training accuracy (again in blue, figure 5, left), which starts highest among all combinations and approaches 100% well before the full card/archetype combination does (in epoch terms). The validation accuracy (in blue, figure 5, right) likewise starts the highest by a wide margin (∼80% vs. the next-highest at ∼25%) and remains untouched, ending with a validation accuracy near ∼95%.

The full card/archetype combination (shown in brown) is the only other image–dataset combination that approached the full card/primary type combination in terms of both loss minimization and training accuracy. In both cases, however, the archetype network takes longer to begin approaching these values. This is perhaps an expected result given that the archetype network has a 'harder' task (classifying fewer examples into more classes). The high training accuracy is not matched by the validation accuracy, which rises from the single-digits into the ∼30% region. The fact that the network is able to do so, however, seems largely a figment of the classifier-rich nature of the images rather than genuine on-target learning. One notable result to this effect is that other archetype-targeting combination, artwork/archetype (gray), has a validation accuracy trajectory that closely matches its full-card counterpart despite being less effective at both minimizing loss (ending with a loss near 0.5) and classifying training examples (<90% training accuracy). We return to this topic after discussing the final image–dataset combination.

Finally, the artwork/primary type combination (green) struggled most. The network ended with roughly the same loss as its archetype counterpart, and marginally lower training accuracy at ∼80%. Despite its relatively 'big' dataset (all cards' artwork) and training for many more iterations (due to being fit on a larger dataset of all cards' artworks), the network seemingly failed to 'learn' much of anything; its validation accuracy fluctuated around 25% throughout training, starting at a distant second-best but ending in last place.

This is a sensible place to initiate a broader discussion of our results because it represents a worst case scenario where the training data cannot meaningfully 'teach' a network because the images simply do not contain information about the target. Primary type is only reflected in each card's text box and (usually) card border color; it is not reflected in the card's artwork. For example, nearly all Synchro Monsters could be remade as Effect Monsters or Fusion Monsters without touching their artwork. As a result, this network failed to learn at all, performing worse than its archetype counterpart despite training on a 'bigger' dataset for a (theoretically) simpler classification task. Comparisons to the network's full card counterpart are thus trivial; the full card network learned better from the get-go, and this training translated into the validation dataset. In other words, when the ResNet152 is trained on images containing the information necessary for accurate classification, it can accurately classify new images. This is an expected (and, frankly, trivial) result for our second research question.

However, the two archetype combinations saw near-identical validation accuracy increases throughout training despite having different image types. This implies that dataset size and classification task (jointly) were the bottleneck in these experiments. In other words, there likely weren't enough images in the dataset to train ResNet152 for a 107-class classification task unless those images are *very highly* class-distinct, which neither image type was. The archetype networks' validation accuracies represent a kind of boundary condition for ResNet152 trained on this dataset–target combination. Our experiments are thus inconclusive as to any specific effect of image type on model

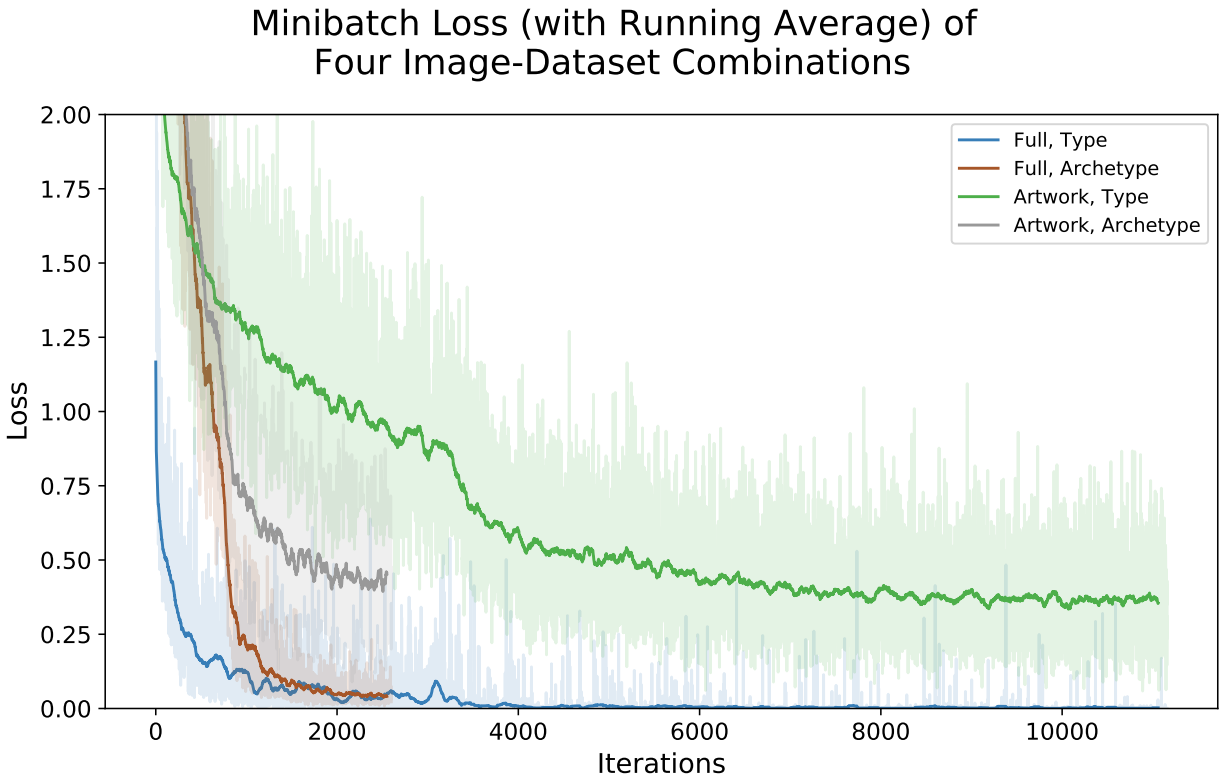

Figure 4. Minibatch loss curves with running average for ResNet152 under each set of image–dataset conditions. Note that the gray and brown curves (Artwork, Archetype and Full, Archetype, respectively) end 'prematurely' because these networks were trained on a smaller dataset for the same number of epochs as all other networks. An iteration is one epoch–batch, so for a fixed number of epochs, smaller datasets are trained for fewer total iterations.

accuracy for this dataset–target combination, and offer no supplement to the trivial result presented above for the second research question.

On the other hand, these results strongly demonstrate the primacy of dataset size and classification task (jointly) in determining how accurately a network can classify novel data. The 'large' archetype dataset and archetype classification task, together, represented a hard limit on ResNet152's ability to accurately classify examples regardless of what information was in the images it trained on. Only when this limitation was removed—as it was for the other dataset–target combination—was image type able to impact model accuracy. Only after surplus training examples were available could the *quality* of those examples become the limiting factor of model accuracy. Thus, our experiments suggest an expected but compelling answer to the first research question: when classifying *YGO* cards with ResNet152, dataset size and classification task (jointly) have a substantial and direct impact on the network's classification accuracy. Attempting to slice the data too 'thin' will yield poor performance.

6. Conclusions

Our major finding is that dataset size and classification task, jointly, were the primary determinant of ResNet152's ability to classify *YGO* images in our experiments. This outcome aligns well with existing literature and serves as a clear example of how these dataset conditions—specifically, dataset size and classification task—can impact ResNet152's ability to 'learn' for the purposes of prediction. It is important to note that we cannot disentangle the individual main effects of dataset size and classification task due to the design of our experiments. An experiment that distinguishes the two—that is, a design where dataset size and classification task were crossed—offers a promising route for future deep learning research on this dataset.

Clearly, the first limitation of our analysis is the lack of a non-trivial result for the second research question. Our result—images containing classification-relevant

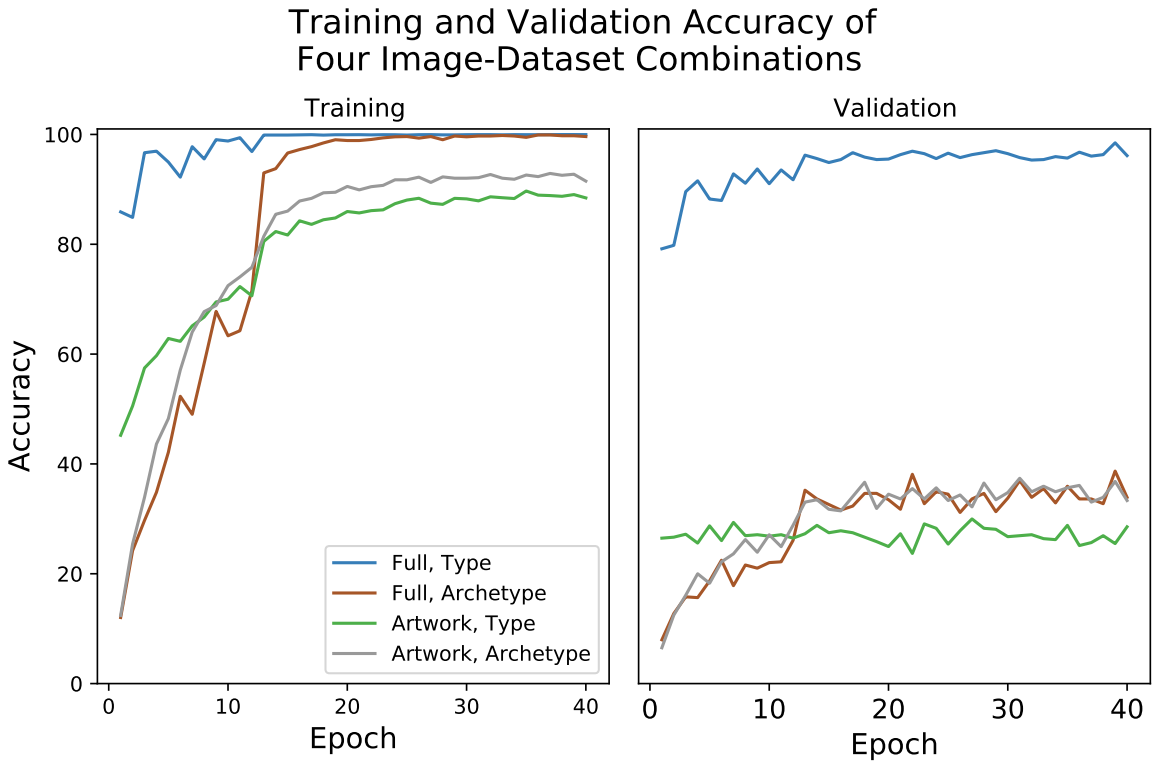

Figure 5. Training (left) and validation (right) accuracy curves for ResNet152 under each set of image–dataset conditions.

information are more useful for training a CNN than those that do not—is not particularly revelatory. Future work could improve our experiment by adding an additional dataset–target combination that lies between the two difficulty extremes examined herein. For example, a dataset composed only of monster cards would retain over 65% of the complete dataset; it is possible that a dataset this size would support a classification task as or more 'difficult' than the primary type task, but less 'difficult' than the archetype task.

One final potential avenue for future research is an experiment that makes better use of the classifier-dense card images. Simply put, there are dozens of ways to crop *YGO* cards that yield different levels of information density, and these cropped images could, in turn, support a plethora of dataset sizes and classification tasks. One obvious example is cropping the cards down to the card effect box and/or artwork and using this to classify, say, archetype or primary type. This crop would remove the 'easy' classification information—name, element, level/rank, and card border, for a few examples—while retaining some of that information in more subtle ways. For instance, card border colors are reflected in the background color of the card's effect box, though the latter's color tends to be a muted and/or desaturated version of the former, and is therefore less distinct.

7. Acknowledgements

The authors wish to thank Dr. Raschka for the code on which our network training is based, as well as his assistance in fleshing out our experiment.

8. Contributions

Card data and images were scraped by Michael. The authors worked collaboratively on all major aspects of the project: model implementation, model evaluation, and writing. Since the authors communicate frequently, more fine-grained details about which author did which specific portions of the project are neither possible nor necessary. It is the opinion of both authors that all work was sufficiently collaborative in nature, and neither author feels that the other did less than their "fair share".

References

[1] Yu-gi-oh! neuron available worldwide from today for ios and android, Jul 2020.

[2] K. Bressem, L. Adams, C. Erxleben, B. Hamm, S. Niehues, and J. Vahldiek. Comparing different deep learning architectures for classification of chest radiographs. *Scientific Reports*, (1):10, 2020.

[3] chronoreaper. YugiohAi, 2020. https://github.com/chronoreaper/YugiohAi.

[4] A. Lowhur. I made an ai to recognize over 10,000 yugioh cards, Dec 2020.

[5] S. Ložnjak, T. Kramberger, I. Cesar, and R. Kovačević. Automobile classification using transfer learning on resnet neural network architecture. 01 2020.

[6] C. Luo, X. Li, L. Wang, J. He, D. Li, and J. Zhou. How does the data set affect cnn-based image classification performance? In *2018 5th International Conference on Systems and Informatics (ICSAI)*, pages 361–366, 2018.

[7] A. Mukhopadhyay, P. Biswas, A. Agarwal, and I. Mukherjee. Performance comparison of different cnn models for indian road dataset. In *Proceedings of the 2019 3rd International Conference on Graphics and Signal Processing*, ICGSP '19, page 29–33, New York, NY, USA, 2019. Association for Computing Machinery.

[8] H. N. Nguyen and C. Lee. Effects of hyper-parameters and dataset on cnn training. *Journal of The Institute of Korean Electrical and Electronics Engineers*, 22.1:14–20, 2018.

[9] S. Raschka. L13, Mar. 2021. https://github.com/rasbt/stat453-deep-learning-ss21/tree/main/L13/code.

[10] P. Roy, S. Ghosh, S. Bhattacharya, and U. Pal. Effects of degradations on deep neural network architectures, 2019.

[11] M. Yang. Imbalanced dataset sampler, 2021. https://github.com/ufoym/imbalanced-dataset-sampler.

[12] YGOPRODeck. Yu-Gi-Oh! API Guide - YGOPRODECK, Nov. 2020. https://db.ygoprodeck.com/api-guide/.

[13] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. 2020. `https://d2l.ai`.

[14] Y. Zhang. Evaluation of cnn models with fashion mnist data, 2019.

[15] W. Zhao. Research on the deep learning of the small sample data based on transfer learning. *AIP Conference Proceedings*, 1864(1):020018, 2017.

Appendix: Additional Figures

Figure 6. Monster, spell, and trap cards (left, center, and right, respectively) in the "Dark Magician" archetype cards. Note that all cards reference "Dark Magician" in their card text.
Left image source: `https://storage.googleapis.com/ygoprodeck.com/pics/97631303.jpg`
Middle image source: `https://storage.googleapis.com/ygoprodeck.com/pics/73616671.jpg`
Right image source: `https://storage.googleapis.com/ygoprodeck.com/pics/86509711.jpg`

Figure 7. Far Left: Regular card before cropping. Center-Left: Regular card artwork after cropping.
Center-right: Pendulum card before cropping. Far right: Pendulum card artwork after cropping.
Far left image source: `https://storage.googleapis.com/ygoprodeck.com/pics/10000.jpg`
Center-right image source: `https://storage.googleapis.com/ygoprodeck.com/pics/25629622.jpg`