PokeNet: Predicting Pokemon Card Features Through Deep Learning

Eddy Albarran and Piper Keyes {albarran, pckeyes}@stanford.edu

Abstract

Here we present a collection of Deep Learning networks (i.e. PokeNet) to predict various features of Pokemon cards. Specifically, our models predict type, hit points (HP), and card price. We implemented a 5 layer neural net (5LNN), a convolutional neural net (CNN), and DenseNet-121 to accomplish these tasks. We found that as the prediction task became more difficult (i.e. features to be predicted were less visible or absent from the input data), we required a more complex architecture to achieve good performance.

1 Introduction

Since its release in 1993, the Pokemon franchise has remained a prominent and beloved series in and beyond gamer culture. One facet of this franchise is the Pokemon Trading Card Game, which utilized sets of cards to simulate Pokemon battles based on the original game. Here, we present a set of Deep Learning networks, which we refer to as PokeNet, that predict features of these cards. Specifically, our networks take the image of the cards as input and predict the type and HP of the Pokemon depicted on the card, as well as the price of the card. This project touches upon various important topics in computer vision, such as extracting explicit (e.g. type) and subjective (e.g. price) information from images of objects.

2 Related Work

There is limited published work regarding the prediction of the price of objects. We derived some inspiration from prior work that used Deep Learning to predict stock market price [1][2], but these implementations did not utilize image data. We turned to architectures that would help with our computer vision task, deciding to use the DenseNet-121 model [3], a state-of-the-art model which has been shown to outperform other commonly used computer vision architectures, such as ResNet. We expected this model would perform better than a simple CNN given DenseNet's increased connectivity between distant layer of the network. We also leveraged previous techniques for class activation mapping [4][5], expecting that our type and HP prediction tasks would utilize aspects present in the card images, such as the color of the card and the text present on the card, whereas saliency maps derived from price prediction models would have more complex saliency as there is no explicit information on the card related to price.

3 Dataset and Features

We used Scrapy to scrape https://pkmncards.com/ to generate our dataset. From this webpage we gathered image urls of each currently available Pokemon card (9791 total) as our input data (Fig. 1, left) and information about each card, including the pokemon's type and HP, as well as the low, mid, and high estimates of the current market price of the card (*pkmn_spider.py*). The data were cleaned and stored in a .JSON file (*cards.json*) for preprocessing (*json_editor.py*).

Next, we extracted each card's values by iterating through the json object. For each card, we loaded the image using the url, rescaled the images to 224x224x3, and vectorized them into 1D arrays (*pkmn_create_data.py*). Card arrays were horizontally concatenated into a (224*224*3) x 9791 matrix and saved as a .TXT file (X.txt). Corresponding 1 x 9791 label arrays/lists (Y_type, Y_price, Y_HP) were each saved as separate .TXT files. We also saved the rescaled image as .JPEG files

(*pkmn_save_cards.py*). Finally, we created two load functions: *pkmn_load_data_vec.py*, which reads in the .TXT files and creates a numpy matrix corresponding to all vectorized images, as well as 1x9791 arrays/lists for the corresponding labels. *pkmn_load_data_img.py* uses the saved .JPEG files to create a 9791x224x224x3 numpy matrix (storing all card images) as well as the 1x9791 label structures. After loading our data, we normalized (divided values by 255), shuffled, and divided it into train/dev/test sets by an approximate 80%/10%/10% split (7832 cards/979 cards/980 cards, specifically).

4 Methods

4.1 5LNN (type-classification, HP-regression)

To examine the utility of a simple neural network on our prediction tasks, we implemented a 5LNN. This model was fed vectorized images of cards that propagated through five layers. The hidden units contained in each layer were 512, 512, 256, 128, 12. Each layer performed a linear computation followed by a ReLU activation function apart from the output layers. For classification tasks (i.e. type), we used a softmax activation with a cross-entropy cost function. For prediction of a linear value (i.e. HP), we used a linear output layer, with mean squared error (MSE) as the cost function.

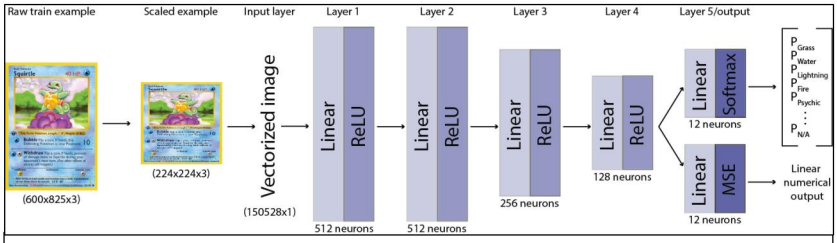

Fig 1. Example input image (left) and 5-layer neural network architecture.

4.2 CNN (type-classification, HP-regression, price-regression)

We employed a CNN approach for classification and regression tasks, wanting to compare the performance of the 5LNN and CNN implementations (Fig. 2). Input images were rescaled to 224x224 for input into our model consisting of two CONV2D (ReLU activation) + MAXPOOL blocks, followed by a FC layer with 12-unit softmax output (cross-entropy loss) for type classification or a single-unit linear output (MSE loss) for HP/price estimation. Adam optimization was used to update the CNN weights.

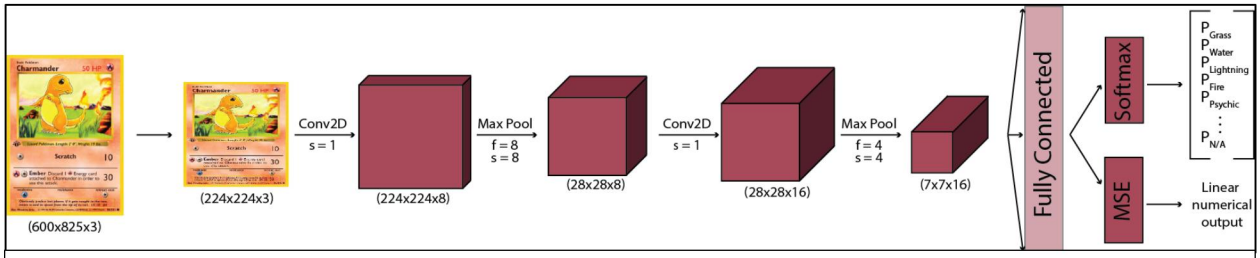

Fig 2. Convolutional neural network architecture.

4.3 DenseNet-121

Finally, we wanted to compare the price-prediction performance of our naively trained CNN model to that of a pre-trained model, re-trained on our task (i.e. transfer learning). We decided to retrain the DenseNet-121 model (pre-trained on ImageNet; Fig. 3). Input images rescaled to 224x224 were suitable for DenseNet-121. The first 420 layers were frozen and we added two additional dense layers (ReLU activation, dropout) with a single-unit linear output (MSE loss) for price estimation.

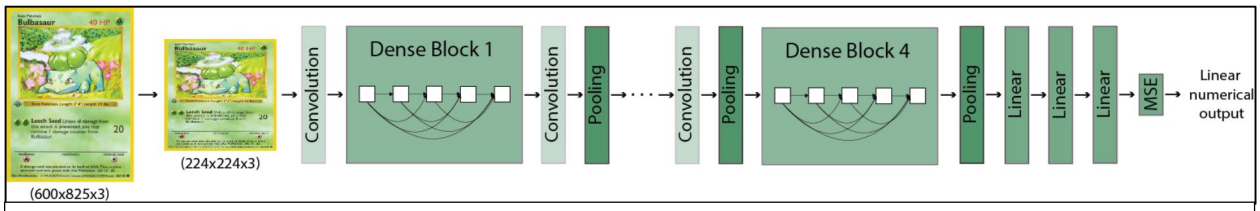

Fig 3. DenseNet-121 architecture. 420/426 layers were frozen during training while the remaining 6 layers and 4 additional custom layers

5 Experiments

5.1 Type classification

We first tested our 5LNN on type classification. We performed extensive hyperparameter tuning (Fig. 4). First, we tested various learning rates (5.0e-3, 5.0e-4, 5.0e-5) and found that both 5.0e-4 and 5.0e-5 performed well, with 5.0e-5 resulting in the best train and dev accuracy rates. We next sought to address bias in our model by training it for longer. We found that increased the epoch number improved our train accuracy with both the 5.0e-4 and 5.0e-5 learning rates, with 350 epochs performing the best in both cases. By improving the bias, we introduced variance into the model and sought to address this with L2 normalization. We found that L2 normalization slightly improved the variance of our model when used with a learning rate of 5.0e-4, 350 epochs, and a beta value of 1.0e-3. Our final train/dev/test accuracies were .99/.98/.98.

	Itertation 1: identify optimal learn rate			Itertation 2: address bias by increasing epoch number				Itertation 3: adress variance with L2 norm			
Learning rate	5.0e-3	5.0e-4	5.0e-5	5.0e-4	5.0e-4	5.0e-5	5.0e-5	5.0e-4	5.0e-4	5.0e-5	5.0e-5
Epoch num	150	150	150	250	350	250	350	350	350	350	350
Beta value								1.0e-3	1.0e-4	1.0e-3	1.0e-4
Train accuracy	0.146	0.930	**0.959**	0.963	0.979	0.984	**0.996**	**0.997**	0.997	0.999	0.995
Dev accuracy	0.158	0.914	**0.953**	0.951	0.951	0.959	**0.971**	**0.978**	0.976	0.971	0.973
Final cost	2.334	0.370	0.225	0.178	0.104	0.101	0.050	0.618	0.114	0.780	0.136

Fig 4. Hyperparameter search for type classification with 5LNN. Search was broken into three iterations with a defined goal.

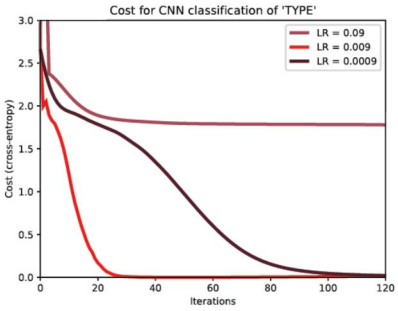

Fig 5. CNN costs across epochs by learning rate when predicting price.

Next, we tested our CNN on this classification task. We trained our CNN model using 3 distinct learning rates (0.09, 0.009, and 0.0009) for 250 epochs (Fig. 5) and observed that a learning rate = 0.09 achieved the best accuracy (.972/0.97/0.968, train/dev/test respectively). We decided to leverage the CNN architecture to visualize model classification by generating class activation maps (CAMs) for various test inputs (Fig. 6). CAMs for each class (type) were generated by summing over the product of (weights for a specific class) * (the input features of a specific input image), where weights act as relative importance of those features for a given classification. CAMs revealed that our CNN model gave large weight to color features when determining type classification of cards with a type. Because colors depicted on the background/border of cards typically correspond to Pokemon type, these areas unsurprisingly had high activation. However, for non-typed cards like Trainer cards, our CNN model paid particularly attention to the text on the card.

Both 5LNN and CNN approaches achieved good performance on type classification. Although the 5LNN approach yielded better accuracy, the CNN approach achieved comparable accuracy with only 60% the training time. The high accuracy is perhaps expected given the relative ease of this task for humans and the strong correlation between card colors and card type. There were however, a small subset of cards whose color did not correspond well with its type, which were problematic for these models (e.g. Fig. 6, right).

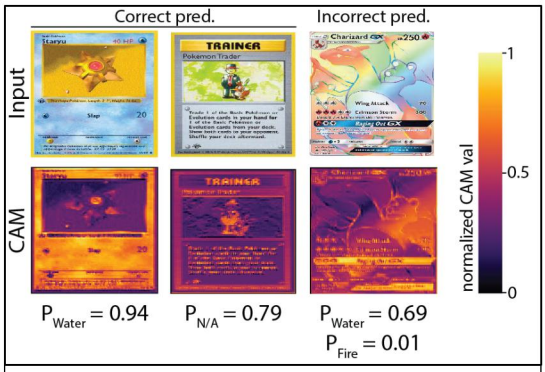

Fig 6. CAMs for two cards with correct type prediction and one with incorrect type prediction.

5.2 HP prediction

We tested our 5LNN on this prediction task with hyperparameter tuning. We found that using the same

learning rates used for the type classification task (5.0e-3, 5.0e-4, 5.0e-5) and 150 epochs of training that our model did not perform very well. Each pokemon card has an HP that is a multiple of 10, and the best train and dev RMSE we achieved on this task were 23.8 and 23.6, respectively. Thus, the 5LNN achieved about $\mp$ 2 HP levels. When increasing the epoch number to 250 and using the best learn rate (5.0e-4), we found that the train and dev RMSE improved slightly to 18.7 and 20.1, and our test MSE was also 20.1.

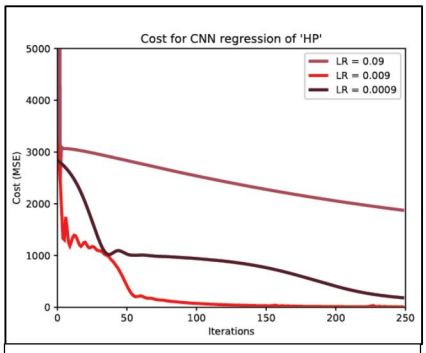

Fig 7. CNN costs across epochs by learning rate when predicting HP.

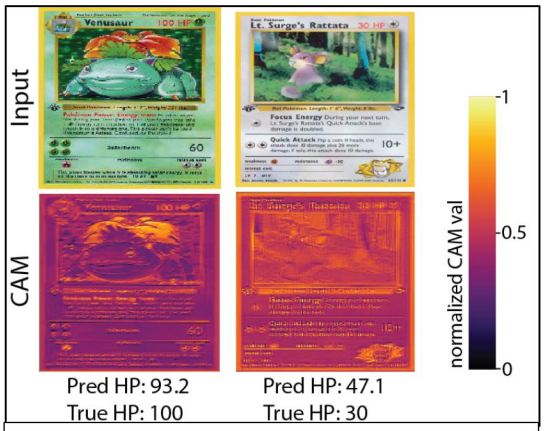

Fig 8. Saliency maps of CNN HP prediction.

For our CNN implementation we modified the output layer of the model used previously for type classification, this time utilizing a linear output and MSE as the model loss. We trained our model for 250 epochs using various learning rate values (Fig. 7) and once again found learning rate = 0.009 to be the most optimal, achieving RMSE of 8.6/9.8/8.5 for train/dev/test respectively, half the error of the 5LNN approach. Once again we generated CAMs for our predictions (in this case, with no classes, we refer to these as a saliency maps) and observed that input features corresponding to text were weighted heavily, including the text on the cards that note the actual HP of the card (Fig. 8).

The task of HP prediction was intuitively more difficult than type classification, thus we were not surprised that our 5LNN model performed poorly. Our CNN model achieved relatively good performance (HP is always a multiple of 10 and our avg error is <10). It is likely that a text-recognition based approach would be able to perform better on this task given that the information for HP is located on the card itself. Interestingly, computed saliency maps reflected the fact that our models did not solely focus on HP text in generating HP predictions (Fig. 8).

5.3 Price prediction

We first approached the price-prediction task by utilizing our CNN implementation used for HP regression. We trained our CNN model for 250 epochs across three different learning rates (0.09, 0.009, 0.0009) on all three price point labels. In order to account for the fact that card prices are not normally distributed (which introduces an accuracy bias towards lower prices) (Fig. 9), we employed a MSE cost function weighted by a function corresponding to the inverse of the price distribution. A learning rate of 0.009 gave the most optimal performance for 250 epochs (Fig. 10), with RMSE values of 0.09/0.11/0.10 (train/dev/test) on low-price, 0.19/0.31/0.25 for mid-price, and 0.52/0.64/0.58 for high-price predictions.

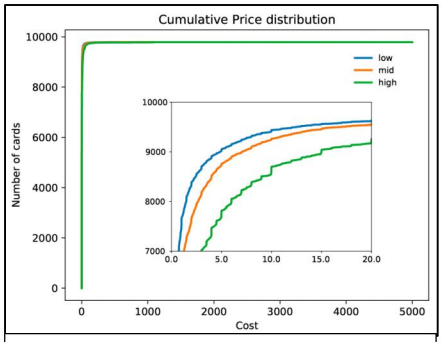

Fig 9. Cumulative distribution of card prices.

Lastly, we employed transfer learning on DenseNet-121. We utilized a learning rate of 0.009 (optimal across 0.09, 0.009, and 0.0009), momentum of 0.9, and minibatch size of 128, for at least 200 epochs (Fig. 11). Using MSE as the model loss, we arrived at final RMSE values of 0.07/0.09/0.098 for train/dev/test on low-price, 0.15/0.24/0.23 for mid-price, and 0.29/0.39/0.58 for high-price predictions. Together, our CNN implementation resulted in an average test error of ~$0.25 (mid-price) and the DenseNet-121 transfer learning approach achieved slightly better results with an average test error of ~$0.23. Distribution plots show that only 2.34% of total mid-price cards are below this cost. Finally, as depicted in a generated saliency map from our CNN (Fig. 12), our model utilized a complex weighting of color, text, and background features in generating price prediction.

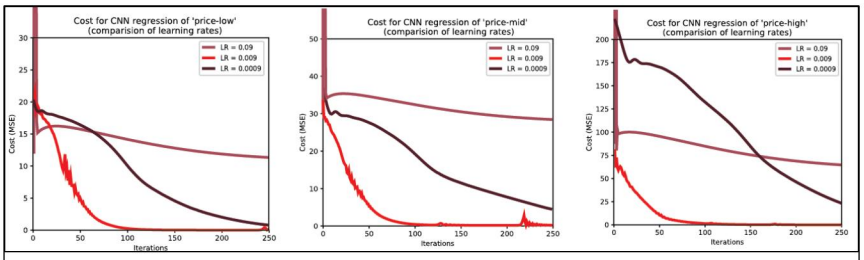

Fig 10. CNN costs across epochs by learning rate when predicting low, mid, and high price.

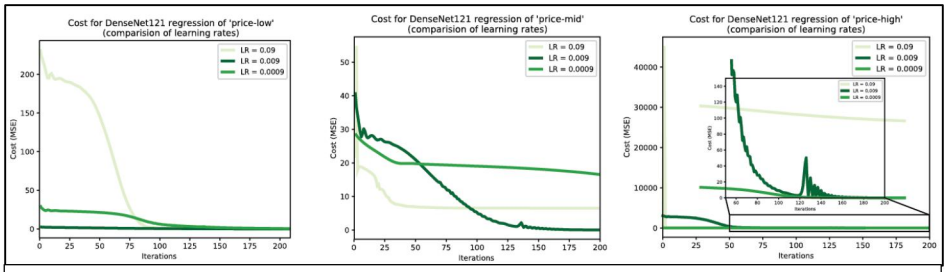

Fig 11. DenseNet costs with different learning-rates for predicting low, mid, and high price.

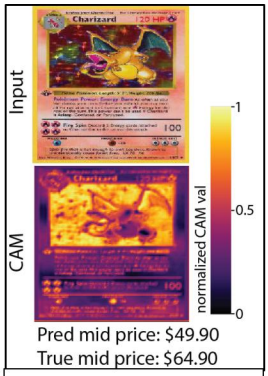

Fig 12. Saliency map

6 Conclusions/Future work

We found that predicting type could be accomplished with almost perfect accuracy by either the 5LNN or the CNN, and saliency mapping showed us that the CNN used either the dominant color of the card or text to make its prediction. For predicting HP, the CNN approach outperformed the 5LNN model and we expect if we trained the CNN for longer, we could achieve even better results. Furthermore, as HP is factors of 10, we could use a softmax activation output layer with cross-entropy cost function and compare its performance with our current regression implementation. Additionally, it may be worth trying to predict HP after cropping the text that explicitly states HP out of the card, to see how our networks utilize the other information present in the card to predict HP. Finally, both our naive CNN and pretrained DenseNet models performed relatively well on predicting price. Further tuning of how many layers we freeze during re-training of the DenseNet or changing details of the added layers might improve future performance. In summary, the work presented here forms the basis of our ultimate goal of creating a real-life Pokedex, an application that, when shown any picture of a specific Pokemon, can predict all of the features of that Pokemon and display it to the trainer. This application would fulfill a life-long dream of many people that once, and perhaps still, aspired to become Pokemon Masters.

7 Contributions

Both authors contributed to the design of the project, training, analysis/interpretation of results, debugging, and production of deliverables for the project. Piper wrote the web scraper and the 5LNNs. Eddy wrote the scripts to vectorize and load the pokemon cards as well as the CNNs, and adapted and customized the DenseNet-121 and CAM functionality from previous implementations. All code can be found at: https://github.com/pckeyes/PokeNet

References

[1] X. Ding, Y. Zhang, T. Liu, J. Duan. Deep Learning for Event-Driven Stock. *IJCAI,* 2015.

[2] R. Akita, A. Yoshihara, T. Matsubara. Deep learning for stock prediction using numerical and textual information. *IEEE ICIS,* June 2016.

[3] G. Huang, Z. Lie, L. van der Maaten, K. Q. Weinberger. Densely Connected Convolutional Networks. *ArXiv e-prints,* January 2018.

[4] K. Simonyan, A. Vedaldi, A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *ArXiv e-prints,* December 2013.

[5] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba. Learning Deep Features for Discriminative Localization. *Computer Vision and Pattern Recognition,* 2016.