Judging a Book by its Cover: A Modern Approach

Yien Xu
yien@cs.wisc.edu

Boyang Wei
bwei9@wisc.edu

Jiongyi Cao
jcao56@wisc.edu

Abstract

In this work, we employ three machine learning techniques to study the relationship between book covers and their popularity. We first train a classification model using a Convolutional Neural Network (CNN) to predict the popularity level given a resized and randomly cropped 120 $\times$ 120 image input of the book cover. Second we employ an interpretation algorithm called LIME to extract patches in the image to explain the prediction result. Finally, we explore GAN to generate the most popular cover by machine itself. We find that our classification model suffers from over fitting, and thus, we in general fail to conclude a clear relationship between the cover and popularity. However, given a specific instance, LIME extract reasonable features to explain the result. And finally, our GAN is able to provide some general tips of what popular book covers have.

1. Introduction

As a saying goes, "do not judge a book by its cover." This might be true, but what we are interested in is its literal meaning - despite the content, will a book attract more readers simply by a well-designed cover? Such information can be especially useful for book publishers: given a specific genre or topic, which book cover should be used to attract more readers? For example, for the book "Robinson Crusoe", should we use a portrait of Robinson, or should we use a photo of the land surrounded by the sea?

In this project, we try to answer the above question by applying several deep learning architectures to learn the relationship between a book cover and its popularity. There are three major goals in this project: first we want to understand if a book cover can be a truly informative factor to predict the popularity. Particularly, we train a deep learning model to classify the level of popularity given an image input of book covers. Second, if it is truly the case, we aim to further interpret our result, that is which part of the cover would explain the prediction? In other words, which patches in the image contribute or negatively affect the popularity the most? Finally, given all the information above, we want to see if we can generate the most popular book covers purely based on machine itself so that understand what a popular book should look like.

To answer the first question, we train a CORAL-CNN, which is well-known for classifying ordered rank-based images. To answer the second question, we apply LIME to interpret the weights obtained from the CORAL-CNN, given individual images from the dataset. Lastly, to answer the third question, we train a DCGAN with popular book covers to generate new ones from random noise.

2. Related Work

Visual design always has a purpose, whether to attract more people or to introduce the general idea of the designer. For example, a movie poster will serve as a purpose to draw people's attention and attract them to see the movie, while an artist will use his or her paintings to share his or her ideas about the world. Although there has been a long history with visual designs, the machine learning technique for exploring this field is relatively new.

Gatys et al.[2] introduced an artificial system that creates artistic images of high perceptual quality based on artistic style of paintings. Similarly, Karayev et al.[6] described an approach to predict style of images and preform a thorough evaluation of different image features.

In the field of popularity prediction, there have been a few attempts lately. Lu et al.[9] used CNNs to assess image aesthetics, relying on human devised features. In addition, Pham et al.[13] evaluated classification (Multilayer Perceptrons) and regression algorithms to predict popularity of songs and determined the types of features that hold the most predictive power.

Compared to recent works, the popularity of independent variable is measured as both categorical variable and continuous variable, while this project mainly focuses on the categorical responses due to the implementation of the CORAL-CNN method. In addition, some particular image types such as paintings have been largely related to social implications and religious traditions, which will have an impact of the prediction as noises. The book covers we used, however, does not have a strong social inference and are easily to explore the connection between the book cover and the popularity.

3. Proposed Method

We propose three different machine learning techniques throughout the paper:

- CORAL framework to address the problem of classifier inconsistency
- LIME to further understand which particular part of the book cover contribute to the book's popularity
- DCGAN to study what a popular book looks like

3.1. CORAL-CNN

We use Convolutional Neural Networks (CNN) as the architecture to make predictions of classifications. Specifically, we use ResNet-34 [4], which is a modern CNN architecture with relatively good performance on a variety of classification tasks, as our model. In addition, considering the ordinal property of our response variable (ranging from level 0 to level 4), we implement the framework called Consistent Rank Logits (CORAL) [1] to reflect the ordinal information, meanwhile keeping it rank-monotonic.

Some researchers proposed a general reduction framework to convert an ordinal regression problem into multiple binary classification problems [8]. However, since the cost-related weighting of each binary task is specific for each training example, this approach was described as unfeasible in practice due to its high training complexity [12]. The CORAL framework, on the other hand, does not require explicit weighting on each training example to generate a monotonic order information and to produce consistent predictions for each binary tasks.

In our project, we implement CORAL in order for the binary tasks to produce consistently ranked predictions. Define S = $\{\mathbf{x}_i, y_i\}$ as our training dataset consisting of N examples. Here, x_i denotes the i^{th} image and y_i denotes the i^{th} rank, where $y_i = \{r_1, r_2, ..., r_K\}$ with ordered rank $r_K > r_K - 1. > ... > r_1$. We convert the rank label y_i into $K-1$ binary labels $y_i^{(1)}, y_i^{(2)}, ...y_i^{(k-1)}$ in which $y_i^{(x)} \subset \{0, 1\}$ indicates whether y_i exceeds rank r_x or not. Then, with the extended binary labels as our model input, we train our ResNet-34 model with $K-1$ binary classifiers in the output layer.

In this case, although the $K-1$ binary tasks share the same weight parameter, they have independent bias units, which solves the inconsistency problem among the predicted binary responses and reduce the model complexity. The predicted rank for an input x_i is generated via

$$K = 1 + \sum_{k=1}^{K-1} f_k(\mathbf{x}_i) \tag{1}$$

where $f_k(\mathbf{x}_i)$ is the prediction of the k^{th} binary classifier in the output layer.

As for the loss function, we define $\mathbf{W}$ as the weight parameter besides the bias unit in the last layer. In the penultimate layer, the output is denoted as $h(\mathbf{x}_i, \mathbf{W})$, and in the final output layer, by adding the $K-1$ independent bias units, the input to the binary classifier becomes $\{h(\mathbf{x}_i, \mathbf{W}) + b_K\}$. In addition, we used the logistic sigmoid function as our activation function

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{2}$$

Therefore, our predicted probability would become

$$\hat{P}(y_i^{(k)} = 1) = \sigma(h(\mathbf{x}_i, \mathbf{W}) + b_K) \tag{3}$$

For model training, we minimize the loss function

$$L(\mathbf{W}, b) = -\sum_{i=1}^{N} \sum_{k=1}^{K-1} \lambda^{(k)} [log(\sigma(h(\mathbf{x}_i, \mathbf{W}) + b_K)) y_i^{(k)} + log(1 - \sigma(h(\mathbf{x}_i, \mathbf{W}) + b_K))(1 - y_i^{(k)})], \tag{4}$$

which is the weighted cross-entropy of $K-1$ binary classifiers. Here, $\lambda^{(k)}$ denotes the weight of the loss associated with the k^{th} classifier. Then, for rank predictions from Equation 1, the binary labels are generated via

$$f_k(\mathbf{x}_i) = 1\{\hat{P}(y_i^{(k)} = 1) > 0.5\} \tag{5}$$

3.2. LIME

To achieve our second objective, that is to interpret the predictive behavior of our CNN model, and to understand which particular part of the book cover would contribute (or negatively affect) the book's popularity, we employ Local Interpretive Model-Agnostic Explanations (LIME)[15] to our CNN model.

LIME is an algorithm for explaining the result of classification or regression by local approximation. Given an observation say a text, an image or a tabular data, LIME extracts features that are most relevant to the prediction output, and thus, interpret the result. Specifically, for image classification, LIME can present patches in an image to provide quantitative understating of the relationship between the instance and its prediction label.

In this project specifically, we apply LIME to answer given a book cover which parts of the image will explain its output, i.e., the popularity level. Particularly, a successful LIME output will extract patches in the book cover that are most deterministic to its popularity.

3.3. DCGAN

In addition to predicting the popularity of book covers, we would also like to study how the cover of a popular book should look like. Hence, we implement a generative model

here to observe how a machine itself can learn from the pattern of popular book covers to generate new book covers that are fancy enough for the general public.

GAN is a generative model that competes against an adversary, which is a discriminative model that learns by true or fake labels of images to determine whether a sample created by the generative model is from the distribution of the images [3]. We choose to use GAN specifically here because traditional generative methods suffer from the difficulty of approximating many interactable probabilities that arise from strategies like maximum likelihood estimation [3].

In this project, we implement a Deep Convolutional Generative Adversarial Network (DCGAN), which is a variant of the original GAN architecture [14]. Compared to GAN, DCGAN has done the changes listed as the following:

- Replace deterministic spatial pooling functions (e.g., max pooling) with all convolutional layers with strides.
- Eliminate fully-connected hidden layers in favor of a deeper network architecture
- Utilize batch normalization [5] in both the generator (after transposed convolutional layers) and the discriminator (after convolutional layers)
- Apply ReLU [11] as activation functions in the generator except for the output layer, which uses a TanH function
- Apply LeakyReLU [10] as activation functions in the discriminator

The architecture of our DCGAN is shown in Table 1. We denote G as the generator and D as the discriminator. G learns to generate a distribution p_g over data $\mathbf{x}$ with input noise variables $p_{\mathbf{z}}(\mathbf{z})$ given. $D(x)$ represents the probability of the input data coming from $\mathbf{x}$ rather than p_g. We train D to discriminate between the true data and fake data. In other words, we want to maximize the probability of categorizing the images from both the true dataset and the data generated by G correctly. Meanwhile, we train G to minimize the accuracy that D can get [3]. Thus, the training objective of DCGAN is:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(x)}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right] \tag{6}$$

where $V(D,G)$ is the value function of the minimax game.

We initialize the weights in convolutional and transposed convolutional layers to $N(\mu = 0, \sigma = 0.02)$, weights in batch normalization layers to $N(\mu = 1, \sigma = 0.02)$, and bias terms in batch normalization layers to 0.

Layer	Filters	Kernel	Stride	Padding
ConvTranspose2d	512	4×4	1	0
BatchNorm2d				
ReLU				
ConvTranspose2d	256	4×4	2	1
BatchNorm2d				
ReLU				
ConvTranspose2d	128	4×4	2	1
BatchNorm2d				
ReLU				
ConvTranspose2d	64	4×4	2	1
BatchNorm2d				
ReLU				
ConvTranspose2d	3	4×4	2	1
TanH				

Layer	Param	Kernel	Stride	Padding
Conv2d	64	4×4	2	1
LeakyReLU	$s = 0.2$			
Conv2d	128	4×4	2	1
BatchNorm2d				
LeakyReLU	$s = 0.2$			
Conv2d	256	4×4	2	1
BatchNorm2d				
LeakyReLU	$s = 0.2$			
Conv2d	512	4×4	2	1
BatchNorm2d				
LeakyReLU	$s = 0.2$			
Conv2d	1	4×4	1	0
Sigmoid				

Table 1: Architecture of DCGAN. The architecture of the generator is shown in the upper table, and the architecture of the discriminator is shown in the lower table.

We use the Adam optimizer [7] to accelerate training, for both the discriminator and the generator. Default parameters as recommended in the Adam paper are used, and the learning rate is set to 0.0005.

4. Experiments

In this section, we describe the dataset that we use to analyze book covers, and further operations that we perform in each of the three methods to obtain the results.

4.1. Dataset

We obtain our dataset from Kaggle's Goodreads' Best Books Ever[1]. There are 53,618 images in the dataset, 5 of which are corrupted. Hence, we have 53,613 images in total to work with. The book cover images are of various sizes

[1] https://www.kaggle.com/meetnaren/goodreads-best-books

and most of them are in RGB format. The list of books is sorted in the order of popularity by a score[2] computed by Goodreads based on multiple factors, including vote counts and vote scores. Thus, we can tell whether a book is popular by looking at its position on the list.

4.2. Hardware and Software

All neural network models are implemented in PyTorch 1.0 with Python 3.7. LIME interpretation is implemented in Python 3.7. We train our networks on a PC with NVIDIA GeForce GTX 1080 Graphics Card.

4.3. CORAL-CNN

Our dataset has 53,613 images in total, and we divid them into 5 levels as our ordinal responses(level 0 to level 4). Since 53,613 is not divisible by 5, we discard the 3 least popular images in order to separate the data evenly, which results in 53,610 images. Then, we split it randomly (with stratified sampling) into training set, validation set and test set with the proportion of 8:1:1. In order to keep the dimension of our image constant, we resize images to 128×128 pixels. Then we randomly crop of 120×120 pixels from each image as our input for our CNN model. Finally, we perform a center crop of 120×120 pixels for testing.

As for hyper-parameters, we train CNN model with 200 epochs with mini-batches via Adam [7] with exponential decay rates $\beta_0 = 0.9$ and $\beta_2 = 0.99$, learning rate $\alpha = 0.0005$ and batch size equals to 256.

In order to evaluate our model, we computed mean absolute error (MAE) and root mean squared error (RMSE)

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - g(\mathbf{x}_i)| \tag{7}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - g(\mathbf{x}_i))^2}, \tag{8}$$

which calculated the distance between the test label and the predicted label as Equations 7, 8 suggest. Here, y_i denotes the true rank of the i^{th} test example while $g(x)$ is the predicted rank of the test sample. Both MAE and RMSE are computed on the test set after the lasting training epoch after the last training epoch.

4.4. LIME

To ensure our inputs as well as the model can be applied to LIME correctly, we do extensive amount of work to make sure that the dimensions are matched with LIME's internal functions. Specifically, LIME asks for two external arguments to generate the explanation: the image instance that we want to explain, and our trained model for prediction. One essential function of LIME is `segmentation()`. The format of image instance and prediction output must match with LIME segmentation function so that LIME can proceed its internal algorithm.

First of all, LIME requires image input as Numpy arrays. Thus, we first transfer our tensor image into a Numpy array. Secondly, LIME works on image with dimension height $\times$ width $\times$ color. However our original input is in the form of color $\times$ height $\times$ width. Thus we further apply transpose function to the Numpy array so that LIME's image segmentation function can be proceeded.

4.5. DCGAN

Since our goal is to train a model that is able to generate new book covers with possible high popularity scores, we use a different collection of images to fit our DCGAN. More specifically, we still divide our dataset into 5 categories, each representing its level of popularity. We only choose the top 3 categories, which is a subset of the original dataset. We thereby obtain a set of book covers whose popularity score are relatively higher in the list. Therefore, we feed a collection of 32,166 images to the DCGAN for training.

Each book cover is resized such that its smaller edge contains 80 pixels to ensure that it is not distorted. Then, it is random cropped to a dimension of 64×64, with 3 color channels. Finally, we normalize it so that the pixel values range from -1 to 1.

After data preprocessing, we feed the data, a batch of 128 images, to the discriminator with real labels (1). We then ask the generator to generate fake data from a vector of 100 uniformly random numbers, ranging from 0 to 1. After that, we feed the fake data to the discriminator with fake labels (0). And finally, we update the two optimizers respectively. The loss function we pick here is the Binary Cross Entropy loss. The DCGAN is trained with 200 epochs, and we save all batches of real and fake images.

5. Results

In this section, we demonstrate in detail the results we get from each of the three models, CORAL-CNN, LIME, and DCGAN. Furthermore, we discuss the limitation of our models and some possible future work that can be done to imporve them.

5.1. CORAL-CNN

When computing our CORAL-CNN model, each epoch takes about 1.25 minutes, and the total training time is around 4 hours. Table 2 is the MAE and RMSE values we calculated after the training was done. As shown in the table, both MAE and RMSE in the test set are relatively larger

[2] `https://www.goodreads.com/list/show/1.Best_Books_Ever`

	MAE	RMSE
Training set	0.14	0.40
Test set	1.38	1.72

Table 2: MAE and RMSE of the model

compared to that of the training set, which suggests overfitting of our model. However, the good news is that our CNN model is able to tell whether a book cover is popular or not to some degree.

5.2. LIME

We employ LIME on the book covers of *Harry Potter* series.

The first experiment we did was to compare the differences between two covers from the second series: *Harry Potter and the Chamber of Secrets*. One of the cover includes main title: *Harry Potter*, while the other's was cropped when used for training. Figure 1 below shows the original two images we want to explain.

(a) Cover with main title (b) Cover without main title

Figure 1: Two book cover instances for explanation

The results of LIME explanation are shown below. Notice that both of these two covers are classified as label 0, which is the highest popularity label. Thus the explanation shows us the patches that contribute to book's popularity the most.

Figure 2: LIME explanation for cover with main title

Figure 3: LIME explanation for cover without main title

From Figure 2 we can clearly observe the patches with word "Harry". Here, LIME seems to pick main title for explaining its popularity. Interestingly, however, without main title included, LIME indicates that subtitle patches "Chamber of Secret" is the most decisive part for popularity. In all, these results seem to conclude that title of the book contributes significantly to the popularity of *Harry Potter*.

Secondly, we also experiment on two different covers of *Harry Potter and the Deathly Hallows*. From Figure 4 and Figure 5, we notice that both cover depict three protagonists. Thus, we want to understand if there is any general pattern that will explain such books popularity.

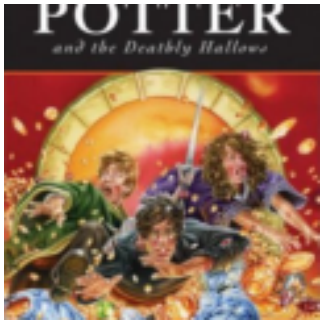

Figure 4: One version of 7^{th} series

Figure 5: Another version of 7^{th} series

LIME explanation on two instances are shown in Figure

6 and Figure 7. An interesting observation is that in both cases, LIME crop the patches of Harry Potter but includes Ron Weasley instead.

However, it is hard to conclude just from two images that if LIME truly recognize the difference between these two male characters, or it is just the figure on the left hand side that tend to contributes more on the popularity.

Figure 6: LIME explanation for the first version

Figure 7: LIME explanation for the second version

One shortcoming in this section of analysis is that all *Harry Potter* series are so popular that all the instances we used are in the same top label. Thus, in general lacking instances for comparison, we are unable to explain given two different covers from the same series why one cover is more popular than the other.

5.3. DCGAN

Since our goal for DCGAN is to create new book covers with high popularity scores, we evaluate the DCGAN by looking at the fake images created by the generator. As shown in Figure 8, a batch of 128 images are created in one of the epochs of our DCGAN.

The new book covers generated mainly consist of three trends - a general color theme, a fancy book title, and an impressive image of some object(s).

We can see that a general color theme is needed for a book cover to become popular. A cover works well as long as its color theme is consistent. Theme colors generated by the DCGAN are mostly either light or dark, and the colors are normally from the similar ranges. For example, some covers have white cover with red titles/images, which represents light-themed covers.

Figure 8: A batch of 128 fake images generated by DCGAN. Three trends that makes up a popular book cover are: general color themes, fancy book titles, and impressive images of some object(s)

Apart from a consistent color theme, a fancy book title is also needed. Most fake book covers generated have a obvious book titles that can be easily discovered. The location

of the book title does not matter, as you can see. It can be either placed at the top, in the middle, or at the bottom. However, there is one failure (row 4, column 3). This book cover contains pure red, without any book title included. Moreover, all titles in the generated images are illegible. We believe a more sophisticated method is needed here in order to generate book titles that are easy to read.

Finally, impressive images of object(s) are also needed by popular book covers. We discover that a meaningful cover image is crucial and our DCGAN is powerful enough to generate these. For example, the book cover in row 4, column 4 looks like a palace, and the book cover in row 9, column 8 looks like two people sitting together on a rock, facing the sea.

5.4. Discussion

We demonstrate that the CORAL-CNN is able to tell whether a book cover image is popular or not to some degree, LIME is able to interpret the CNN model given some input data, and DCGAN is able to generate new fancy book cover images.

However, limitations also apply to our models. CORAL-CNN trains extremely well on the training set, while performs relatively bad on the test set. One potential solution is to add dropout, which randomly sets activation to zero during the training process to avoid ovevfitting. In addition, it is also possible that the dataset is not well-chosen enough. The ranking of the book covers is based on a combination of vote counts and vote scores. We believe the CNN would perform better if popularity is measured through click counts, i.e., the number of clicks a book receives in an online retail store like Amazon.

We have also shown that LIME is able to interpret book covers of the *Harry Potter* Series. However, LIME sometimes fails to interpret images from the test set. We believe that this is due to the low accuracy resulting from CORAL-CNN. If the CNN itself is not able to perform well on the test set, LIME is not capable of interpreting given test images either. In the future, we wish to explore more about the book covers if we had a better-trained CNN.

Our DCGAN generates templates of what popular book covers should be. However, publishers cannot simply take the book covers generated by us because the book title is illegible and the resolution of book cover images needs improvement. We are glad that we observe some general design ideas from the results we have, but future work is needed to create book covers with high-resolution that can truly deceive human eyes.

6. Conclusion

In this paper, we implement the CORAL-CNN model to predict the popularity of the book simply based on its book covers. After we train our model, we use the output parameters to employ LIME to get a better understanding of which particular part of the book contribute the most (whether positively or negatively) to the book's popularity. Finally, we use DCGAN to train a model that is able to generate new book covers with high popularity scores.

As for the result, our CORAL-CNN model needs further improvement as it experiences overfitting issues. Potential methods include collecting more data and implementing other techniques such as a cleaner dataset and adding dropout layers to avoid overfitting. In addition, as for the result from LIME, we witness some interesting observations through comparison between book samples of Harry Potter Series. However, we only experiment on book covers which are in the same top label, and further comparison will be needed for book covers that are in different popularity levels. Lastly, we generate 128 book covers from DCGAN. Some of the images are surprisingly realistic and could even be described through human eyes, which would be a huge improvement if images generated have a higher resolution.

In conclusion, it is still not clear if we can judge a book by its cover. However, we do know that there are certain features in the book cover which relates to its popularity level. Moreover, it is surprisingly that machine could generate a few images with relatively surprising quality.

7. Contribution

All of the three members contributed a great amount of time to the project. Boyang is mainly contributes to CORAL-CNN, Jiongyi mainly contributes to LIME, and Yien mainly contributes to DCGAN. Each person writes up their own sections in the report, and the rest is shared among them. Group members talked to each other frequently for experiments and debugging.

References

[1] W. Cao, V. Mirjalili, and S. Raschka. Consistent rank logits for ordinal regression with convolutional neural networks. *CoRR*, abs/1901.07884, 2019.

[2] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] S. Karayev, A. Hertzmann, H. Winnemoeller, A. Agarwala, and T. Darrell. Recognizing image style. *CoRR*, abs/1311.3715, 2013.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] L. Li and H. Lin. Ordinal regression by extended binary classification. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 865–872, 2006.

[9] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang. RAPID: rating pictorial aesthetics using deep learning. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, pages 457–466, 2014.

[10] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[11] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[12] Z. Niu, M. Zhou, L. Wang, X. Gao, and G. Hua. Ordinal regression with multiple output CNN for age estimation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4920–4928, 2016.

[13] J. Pham, E. Kyauk, and E. P. and. Predicting song popularity. 2015.

[14] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.