Final Project

Sarah Fieck & Sreya Vadlamudi

Introduction

Album art can convey unspoken information on what the album's contents hold. Our final project looks to incorporate machine learning practices in order to categorize musical album covers based on if the album art appears to be of a progressive rock album or not. Progressive rock, typically shortened to "prog rock" is a subgenre of rock music that was made popular in the 1960's and 1970's, known for its explorative approach. Classic bands such as Pink Floyd, Rush, and Yes are just a few of many bands found in this genre. Prog rock albums typically have influences of classical music, longer song compositions, and concept-driven lyrics. This experimental emphasis is also featured in the album's cover art. Prog rock album art can have abstract or fantastical imagery, and oftentimes incorporate aspects of science fiction or irony.

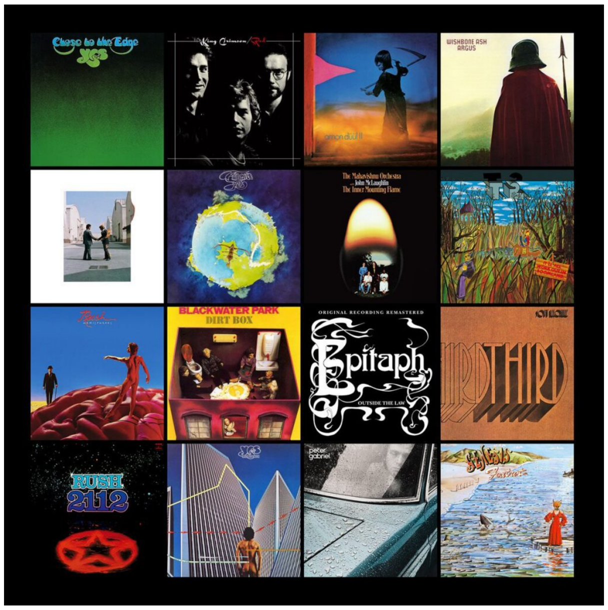


Figure 1: Progressive Rock Album Examples

As machine learning becomes more advanced, we are starting to see it have more hands in artistic industries, such as music and visual art. We believe a project like this is relevant because it aims to capture the synergy between human creativity and machine learning, as we have seen in this ever-changing era of technology.

Analysis

To complete this project, we used two datasets made available on Kaggle. We used one dataset full of uncategorized album cover images spanning across different genres. We also used another dataset full of progressive rock album cover images. The uncategorized dataset has over 80,000 512 x 512 images, and the progressive rock dataset had 2,316 260 x 260 images. All images involved are RGB images.

Our goal with our model is to have it be able to categorize the images as progressive rock versus any other genre, so we needed to do some data cleaning and joining to create these classes. First, we noticed that the uncategorized dataset had 512 x 512 sized images, which needed to be scaled down in order to work with the smaller progressive rock album images. We resized these images to be 260 x 260, just like the size of the progressive rock albums. Once we had all of the images scaled down, we created two directories, a training and a testing one. These two directories contained two subdirectories each, one labeled "is_prog" for progressive rock album covers, and then one labeled "not_prog" for uncategorized album covers. We used 80% of the uncategorized and prog data in the training set, and 20% of the data in the testing set.

Methods

As stated, the goal of our model was to categorize album covers as either progressive rock or non-progressive rock. To achieve this goal, we had to create a model that could use image classification via a Convolutional Neural Network model. If the CNN model is accurate, it would be able to recognize different patterns in the image and be able to determine which of the two genres the image would fall under. More importantly, we wanted to know what features of a progressive rock album were the most crucial to our model when determining it as a progressive rock album. Some of these decisions may be obvious to both us and the model, but some features may be only visible to the computer. To see what the model is looking at, we used a visualization process known as GradCAM, which we will discuss in later sections. We chose a CNN as our model because they are made to process spatial data, like pixels in an image. They basically operate with filters, or matrix of numbers that can be used as weights. This filter is used to slide over the image and be applied to the numeric values of the pixels. The neural network aspect comes into play when discerning what the images contents hold.

This is an incredibly simplified explanation of a CNN, as there are many parameters, and adjustments that can be included in the construction.

In our model's architecture, we included several convolutional 2D layers. These layers apply a set of filters to the input data to collect features that will help the model decide what genre the album is. We included seven convolutional layers: two with 32 filters, two with 64 filters, and three with 128 filters, which all contained 3 x 3 dimensions. In these layers, the other two details we provided were the activation functions and the input shape. The input shape for all the layers was the image width, 260, by the image height 260, and a 3 was included to indicate that this data contains RGB convolutional channels. We used the ReLU activation function in order to introduce non-linearity to the network. ReLU specifically works for our model because it tells whether a feature is present in an image or not.

In addition to these layers and their properties, this model contained seven maximum pooling layers (MaxPooling2D), all of 2x2 dimensions. Maximum pooling is a method of downsampling that basically summarizes our input by taking the maximum valued pixel per convolution. When all of these pixels are combined after the convolutions are complete, we have a downsampled output of the highest valued pixels. This is desirable for discerning an album cover's traits because maximum pooling assumes translational invariance. Meaning that if the contents in the image shift slightly, maximum pooling will still be able to locate highly valued contents due to its search for highly valued pixels. With maximum pooling, our model will always find the most important parts of the album cover, no matter where they are in the image.

Furthermore, a flattening layer was added in order to compress the output of the convolutional portion into a singular array, and this compressed output was fed into a feed-forward network. This portion had the first dense layer that contains 100 neurons, with ReLU activation. This means that there were 100 units of processing information obtained from the convolutional section of our model. Following this layer was an output dense layer with a singular neuron. This layer had a sigmoid activation which serves to squish values between 0 and 1. This is useful when categorizing data between two possible values, which we are doing.

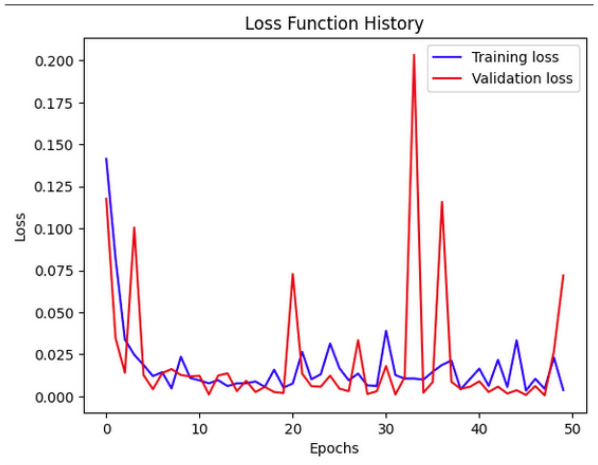
The last necessary step for this model was to include regularization steps. In order to combat overfitting and make sure that the model is as accurate as possible, we included a diverse usage of regularization methods. One method we introduced was called Dropout, which basically prevents our model from becoming too over-reliant on certain connections by randomly dropping nodes in the network. Another method of regularization we used was Batch Normalization, which normalizes data in the output before sending it to further steps of a model. In our model's architecture, we alternated between layers having Dropout regularization or Batch Normalization, so we could have optimal normalization.

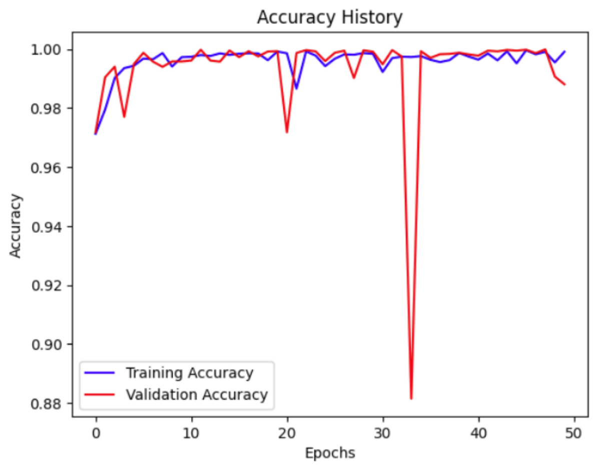
In order to measure the difference between our model's predicted values and the actual values, we used binary cross entropy, a loss function used for classification where two categories are involved. We used Adam as an optimizer because it can be used to help our model learn quicker. Adam can unbias the momentum and learning rate terms so that the terms are not biased towards zero. That way, our model won't have decaying gradients, which will cause our model to not train well.

Our first version model performed very well. However, when attempting to create the heatmap and GradCAM visualizations, it did not work out initially. This was due to some previous model architectures that we used before settling on using the model we previously explained. In order to fix the shortcomings of our first model, we decided to change and expand the model into the architecture we previously explained. After creating our new and improved convolutional neural network model, we were able to create the heatmap and the superimposed GradCAM visualization. I loaded in the model's data from the server where I ran it, and then we used code provided to us through class notes and online in order to create the visualizations by using the convolutional layers from the model. After this, we used the save and display GradCAM function in order to create the superimposed visual.

Results

Based on our metrics, we found that our model performed well. We used binary cross entropy as our loss function, applying this measurement to our model when it looked at both training and testing data. A loss function measures how different a model's predicted and actual values are. Our goal is to have smaller loss function metrics, and high accuracy. At the end of the 50 epochs, we found that our loss function values stayed around 0.02, and our accuracy around 99%.



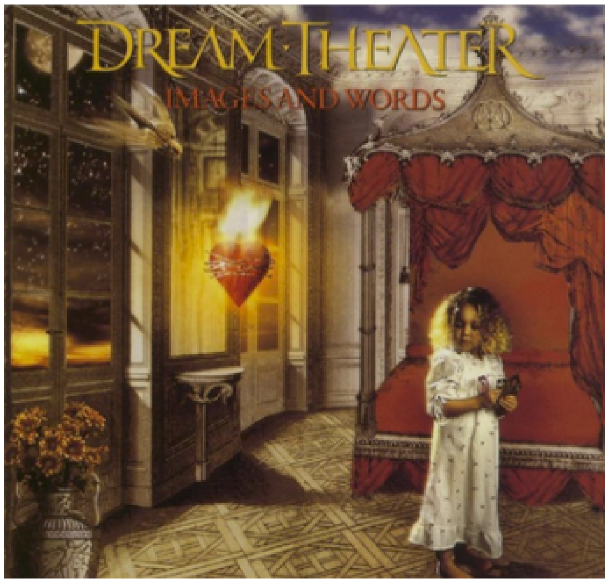


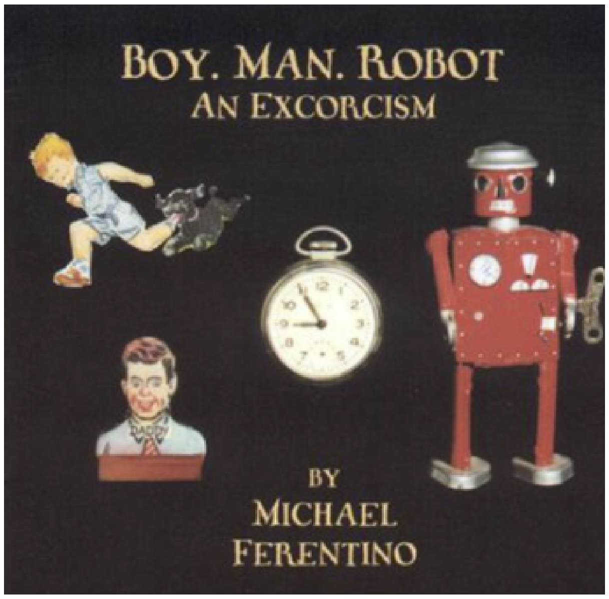
We graphed the history of both the loss function and the accuracy over the 50 epochs. The loss function, while not presenting smooth lines, does maintain low metrics throughout the training process. The testing data somewhat mimics a similar trend, but does exhibit some differences in it's sudden large spikes. However, these spikes only go up to 0.2, which really is not that high of a value for a loss function. Even though there is some divergence in the training and testing loss function, we believe that it is not enough to consider the model overfit, or too reliant on training data. The testing data exhibits similar loss function metrics, meaning that the model knows what to look at when looking at other data beyond the data it had been trained on prior.

Continuing our metric measurements, we also looked at the accuracy of our model. Accuracy measures how close a model is to predicting the true value of some data. Our model exhibited high accuracy in training, with values above 98%. Looking at the validation, or testing accuracy line, we can see that the model exhibits similar accuracy, with some sudden drops. However, once again, those drops do not appear to be that severe for accuracy, as the lowest it drops to is about 88%. It is still primarily accurate. Like the loss function, we believe the accuracy history proves our model is accurate and not overfit, as the metrics are consistently high through training and testing processes.

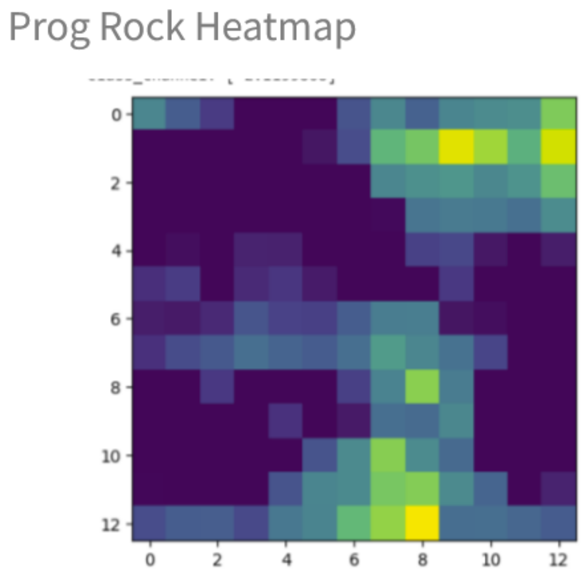
To further measure our model's ability to detect progressive rock albums versus non-progressive rock ones, we utilized some visualizations to see which aspects of the images the model looks to when categorizing. We utilized GradCAM, which is short for Class Activation Heatmaps. This process creates heatmaps of how intensely the input images activate the class categorization in our model. This method is incredibly useful when we want to see what made the model categorize the image in a specific class. Below, we have two images we want the model to look at. One of them being a progressive rock album, and the other one being an album from the genreless dataset. We put them through the model, and obtained heatmap visualizations of the two to see what parts of the image the model will notice.

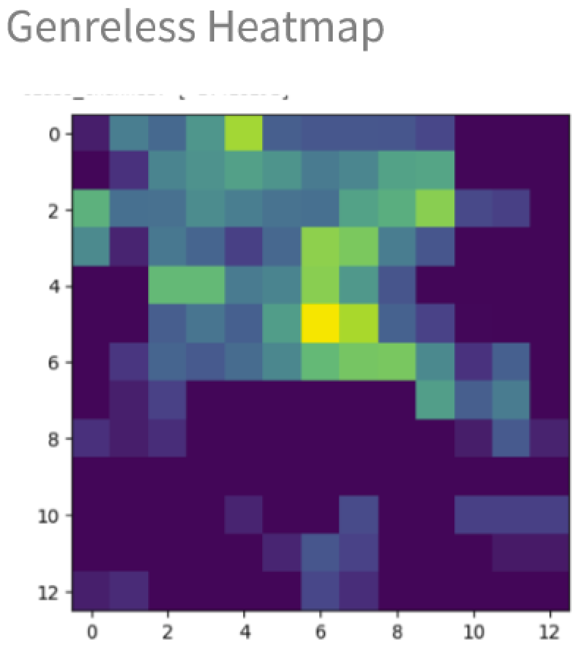
Image Examples





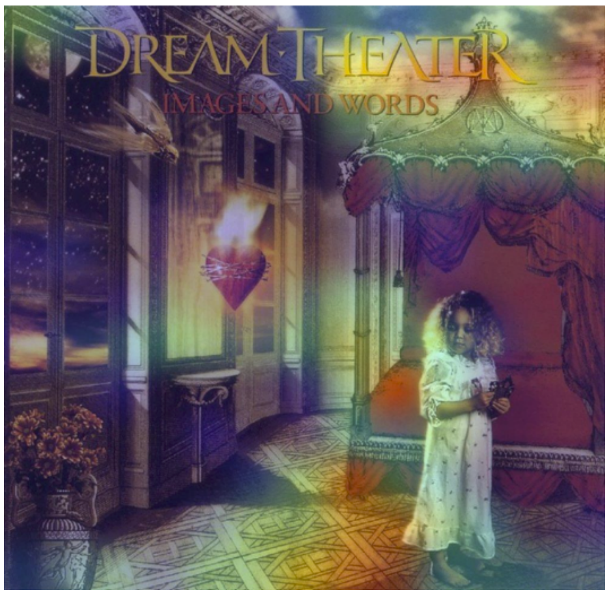
Heatmap Examples

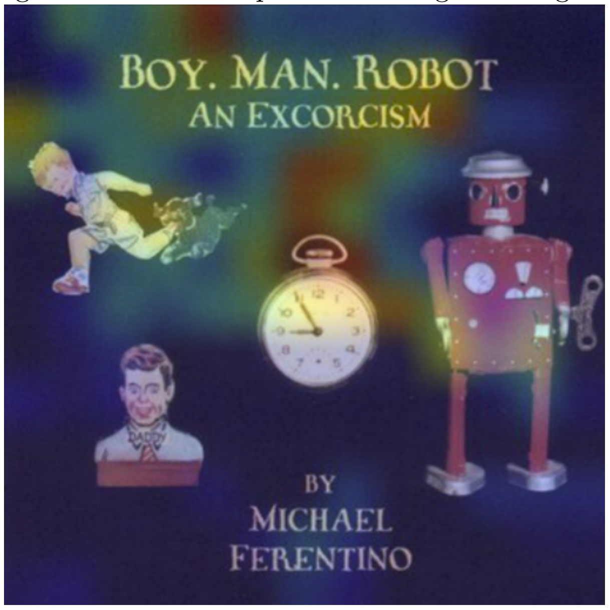




Superimposed GradCAM Examples

Going further, we created superimposed visualizations using GradCAM methods, so we can see the activated parts of the images highlighted as a heatmap over the original image.

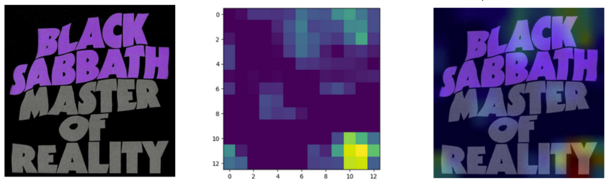




In the prog rock image, we can see the visual highlights the sky in the top right corner, the abstract tile pattern, and the floating red heart on fire. This means that the model notices these things when looking at this image, and classifying it as a progressive rock album. All of these traits seem pretty in line with what constitutes as a progressive rock album, as the model found abstract patterns and elements of fantasy. On the other hand, the non-categorized album had parts of the title highlighted, as well as some of the clock. Titles are incredibly common for album covers across the board, so it comes as no surprised that that would be noticed.

Extra Examples

We also wanted to look at some more progressive rock albums, and how the model identifies it as one. Here is how it reacted to a Black Sabbath album, which is more of a simple and minimal album design.



As we can see by the heatmap and the superimposed image, this album did not have much rhyme or reason as to what parts activated it. The model really looked to the Y in the album title, which is random. This goes to show that our model can sometimes make no sense in it's classification. We looked at another progressive rock album from the band Echolyn to see how the model would take to it.

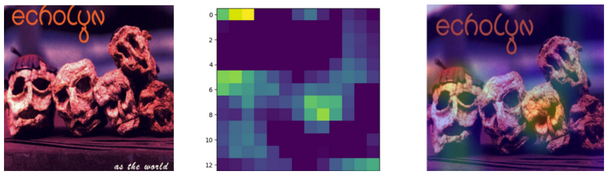


Figure 2: Echolyn's As the World

Unlike the previous album, this one has more of a pattern in the heatmap. The rock-face subjects are immediately highlighted, which makes sense because they are pretty fantastical. This visual shows that our model can sometimes make a lot of sense in its classification. Here are some more GradCAM superimposed visuals to show what our model looks at in these albums:

All in all, we believe that we have the beginnings of a great model on our hands. The metrics prove that it is accurate in its classifications, and the visuals show what parts of the images our model looks at to classify our albums. By creating this model and getting the outcomes we got, we believe that we succeeded in our goal.

Reflection

After working on this project, we learned a lot more about the effects of regularization and how useful it is on CNNs. We also found that this project was great practice on how to interpret a heatmap visualization. If we did this project again, we think that we would not assume that a simple model with little regularization layers would be good to start out with. Even though this first model did not perform poorly in metrics, it was not a good model to use for the other aspects of our project, like creating visualizations. It was easier to get desirable results with a deeper architecture and regularization.

Using complicated models helped us understand the importance of deep learning. Having deep architectures is necessary for the complexity of image classification, so our model can assess proper nuances in the data. In addition, another approach that we would have liked to take if we had more time is the usage of more genres categorized in our dataset. For example, it would have been interesting to look at album covers in the pop genre and compare this to progressive rock. It would be really interesting to see how deep architectures could further classify other genres.

One worry we have about our model is if it is actually classifying non-progressive rock albums correctly. Due to the fact that we had to scale down data in our cleaning phase, we wondered if the model is secretly looking at any side-effects of descaling an image instead of image contents like we hoped. Based on the visualizations, we do not quite see this to be the case, however, there are a lot of details computers look at that go unnoticed by the human eye.