Deeper Dive into Finding Similar Faces with Spotify's Annoy, Tensorflow, and Pytorch

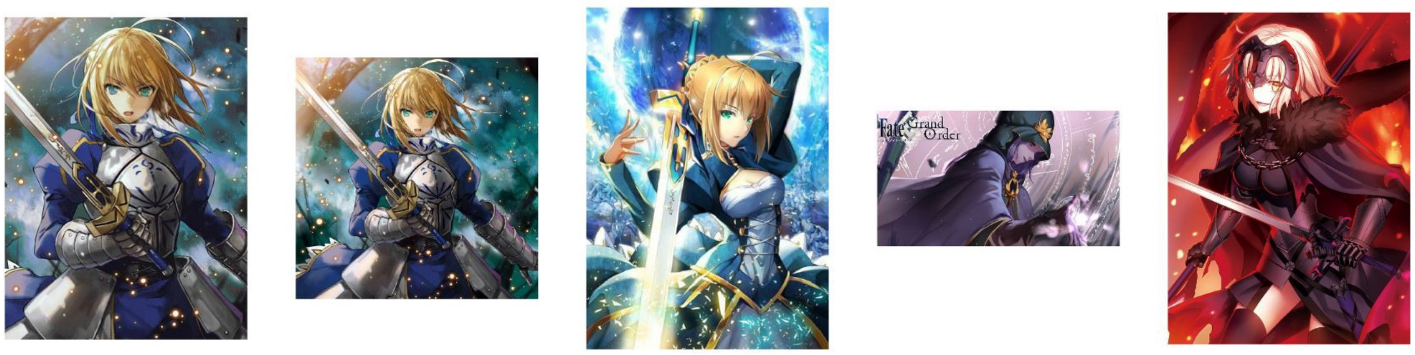
Michael Sugimura Follow
Feb 19 · 8 min read ★

In my previous post I teased that I had jumped down a rabbit hole to try and improve my Fate Grand Order facial similarity pipeline where I was making use of Tensorflow object detectors, Pytorch feature extractors, and Spotify's approximate nearest neighbor library (annoy). The general idea that I was running with was that I wanted to encode information around both character facial features (to find similar characters) and also background information (to return images with similar colors/feel).

The way that I figured I could go about that was to just concatenate the extracted feature vectors from the pretrained Resnet101 for both the base image and the headshot. However this approach turned out to be quite disappointing.

As a quick recap, here is an example image and similar pairings using the version 1 model which used the whole image.

Version 1 model output

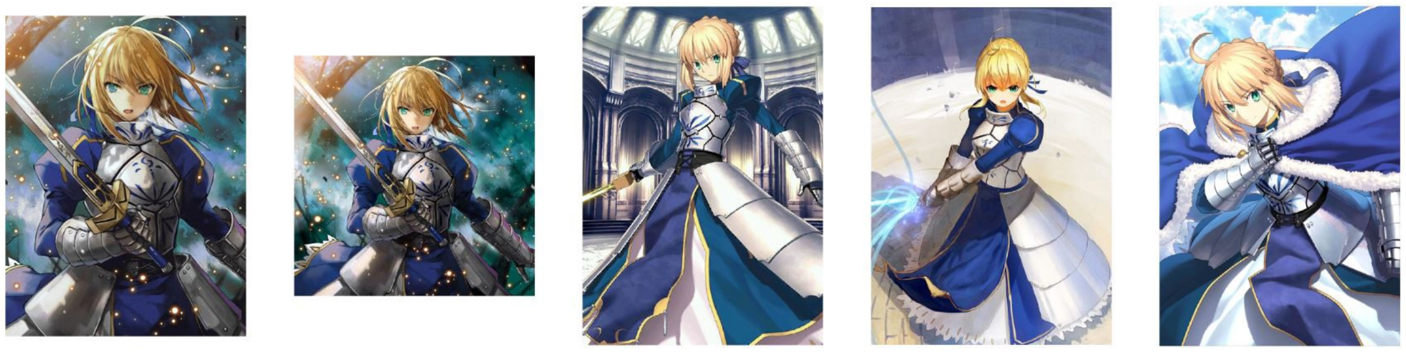


V1 full image sample output: Left image is the original and the next four are the four most similar

The output is ok, but 2 of the 4 similar images are of different characters.

Next a recap of the version 2 headshot based pipeline where I used a tailored Tensorflow object detector to crop out heads of characters in images and used those to build the annoy indexes and to find similar images.

Version 2 headshot model



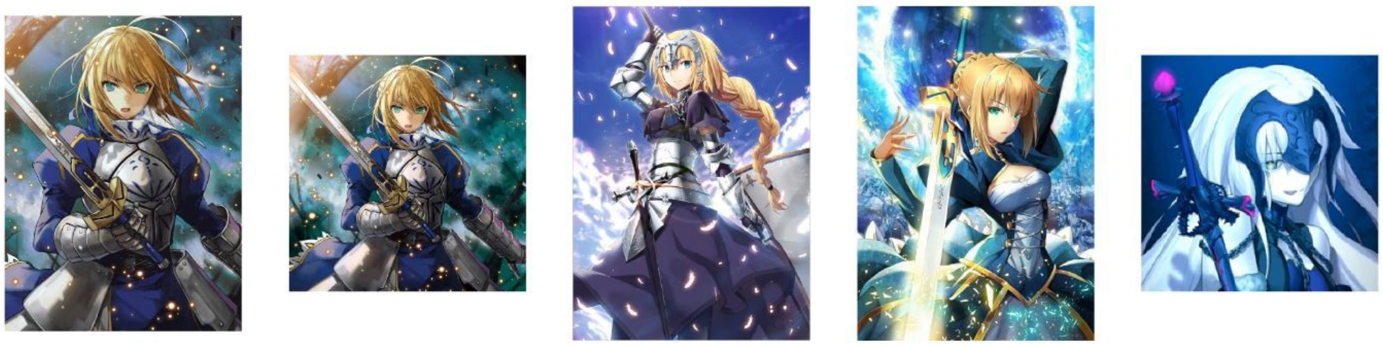
V2 headshot model output

This pipeline seemed fairly successful and returns image of the same character in all the images, but I wanted to see if I could get the "feel" to be closer to the original. Which is why I tried to combine the Resnet 101 output vectors and feed that into annoy to make a new model version 3.

Version 3: combine the first two

My next test was to take the feature vectors from the headshots and the base images and combine them together and feed that new length 2000 array into annoy. The hope was that the first part of the vector could help annoy find similar faced characters and the second part would encode similar information about the image as a whole.

Version 3 combined headshot and base image

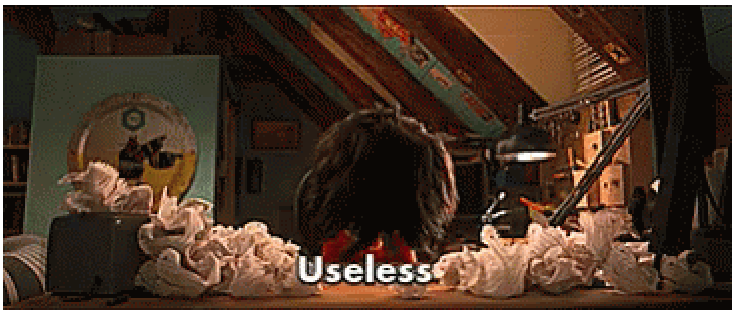


V3 combined feature array sample output

So this 3rd version of the pipeline is not quite what I was hoping for… while these characters do have similar faces 2 of the 4 outputs are of different characters than the one in the base image.

This was interesting for me because my first thought was that it was an issue with the quality of the feature representations I was building off of the images. I even tested it using a larger Resnet 152 but the results were basically the same.

Next was to examine if I thought I was doing something wrong in the rest of the pipeline… like was I doing something wrong by building the 1D representation of the images from their original sparse 3D pixel representation. However this is very tried and true…



My realization this time was re-reading the annoy git repo page and seeing this note.

> *Works better if you don't have too many dimensions (like <100) but seems to perform surprisingly well even up to 1,000 dimensions*

For both the base image and headshot versions of this pipeline the feature vector was the standard 1000 features out of the Resnet 101. Which is at that upper bound and concatenating the two vectors together pushes it to a length 2000 vector which is well outside of the recommended number of dimensions.

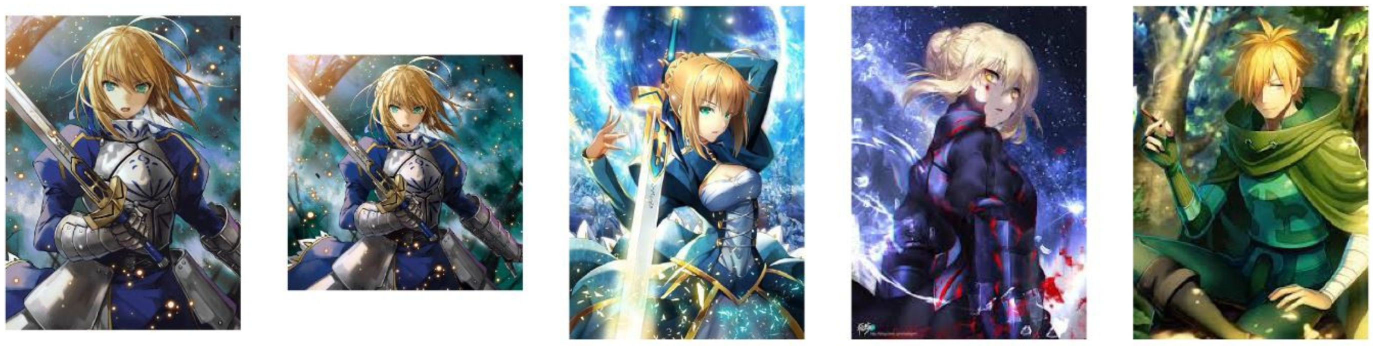
Version 4: Dumb it Down?

While it isn't very common to opt into using a less powerful network for deep learning, everything has a time and place.

With that note in mind I actually downgraded the feature extractor Resnet from a 101 to a Resnet 34 for the version 4 of this pipeline. This is because the layers before the final 1000 fully connected layer in the Resnet 101 is of size 2048 while the Resnet 34 is only 512. My thought was that by making use of this smaller layer I could concatenate two 512 layers to get a final vector of length 1024 rather than 2000. The hope being that this would decrease the number of dimensions enough to let annoy function well.

This change is one that I think improved the quality of my outputs.

Version 4 Resnet 34

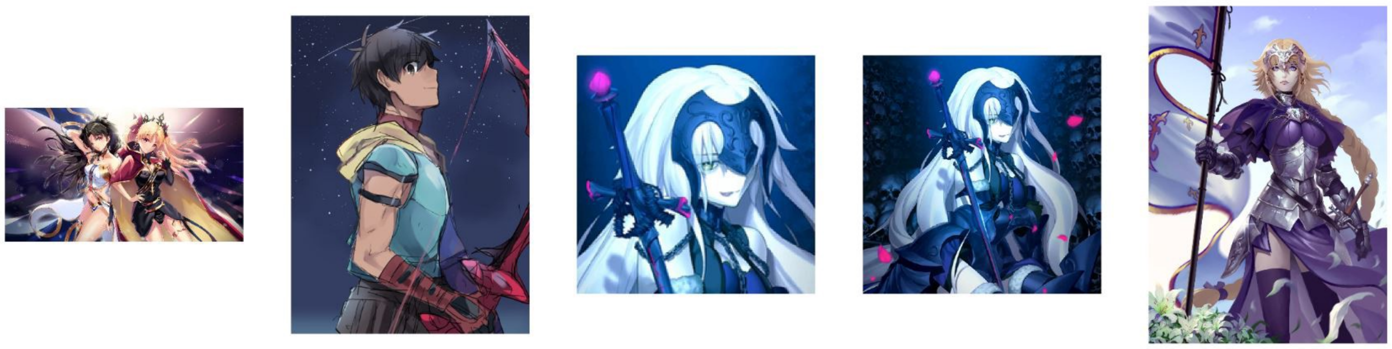


V4 Resnet34 headshot and base image model output

So running the same example images I was showing earlier, this Resnet 34 model seems to look better than the initial headshot + base image model using the larger Resnet 101. Two of the four images are a match of the character. The third image is just an evil version of the original character so I think it counts. The final image however seems very unrelated… so there is progress to be made.

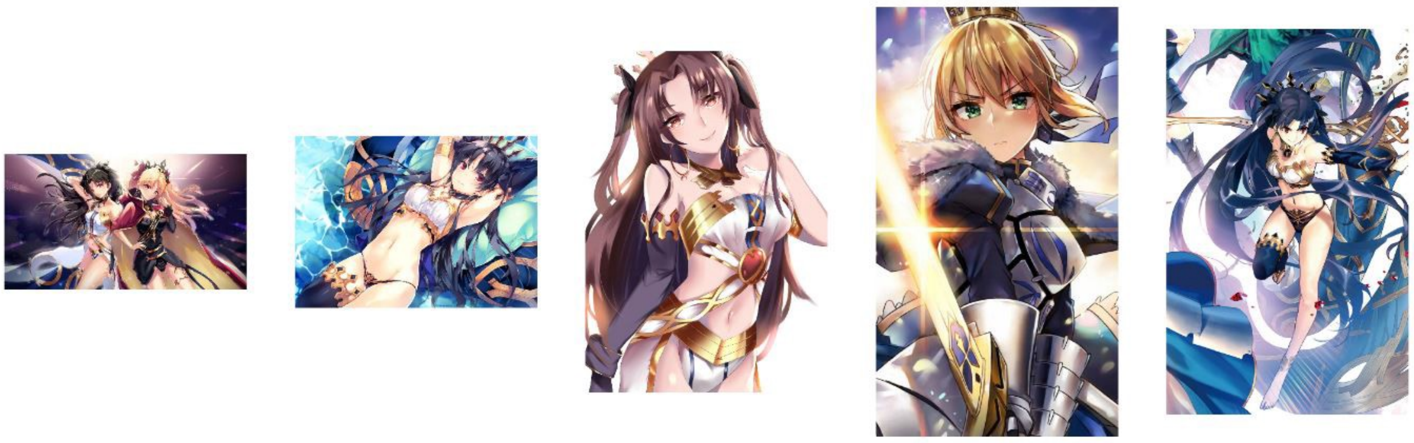
Here is another example showing this Resnet 34 model output versus the previous three versions. I comment on the outputs in the image captions.

Version 1 base image



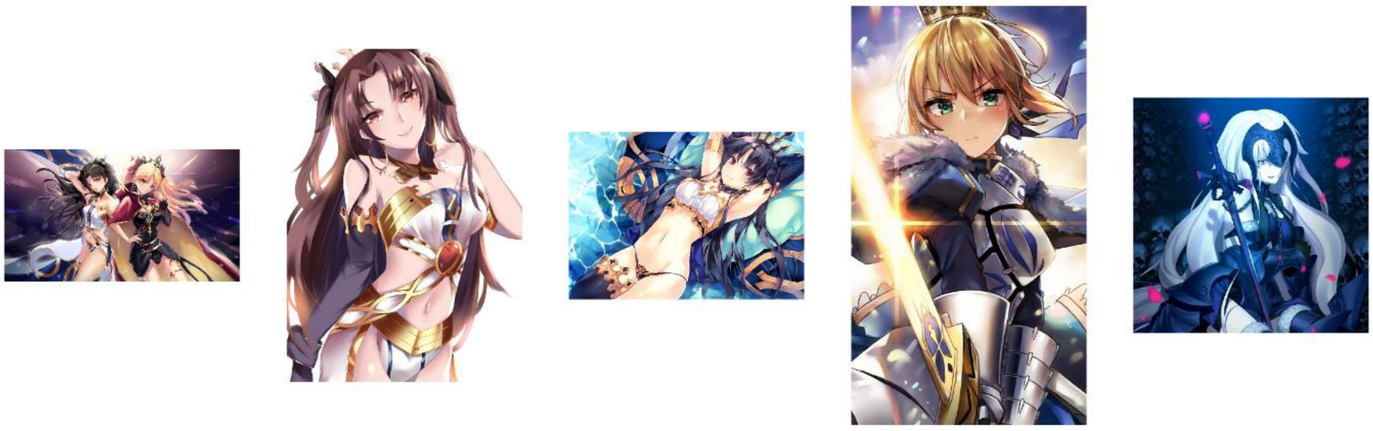
V1 Original full image model. no correct characters, similarly colored (sort of)

Version 2 headshots



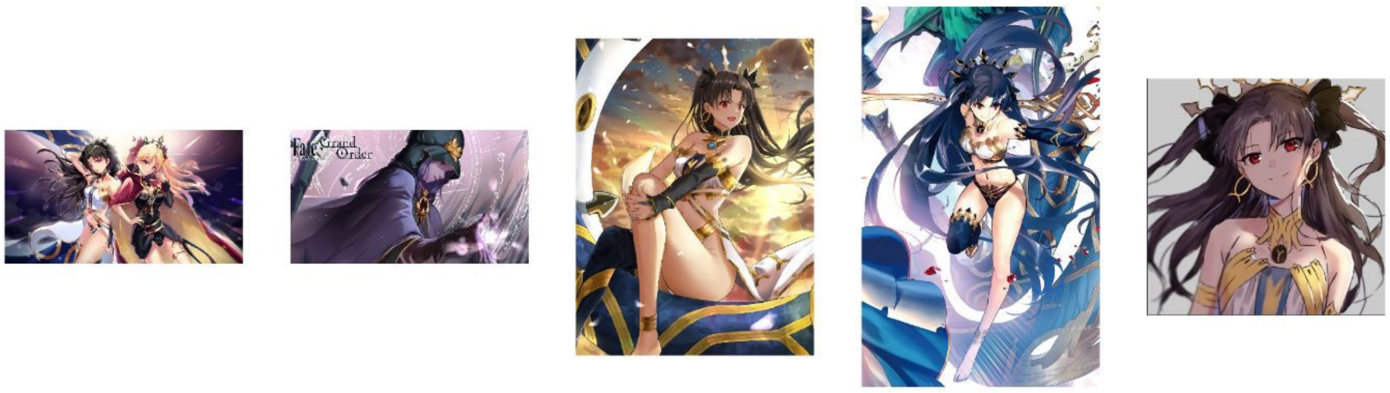
V2 model based on just the headshot. 3 similar characters but images arn't that similar to the base image.

Version 3 combined headshots and base image



v3 headshot + base image Resnet 101 length 2000 vector. This version of the pipeline seems to just combine the first two versions

Version 4 Resnet 34



V4 Resnet 34 length 1024 vectors. It produces decent looking output, but in looking through the outputs I usually find odd images included like the second image in this list

So while the Resnet 34 is an improvement over the Resnet 101 version of this pipeline it is still not performing well… I still believe I can improve it, the question is just, how?

Version 5: 1024 dimensions down to 52

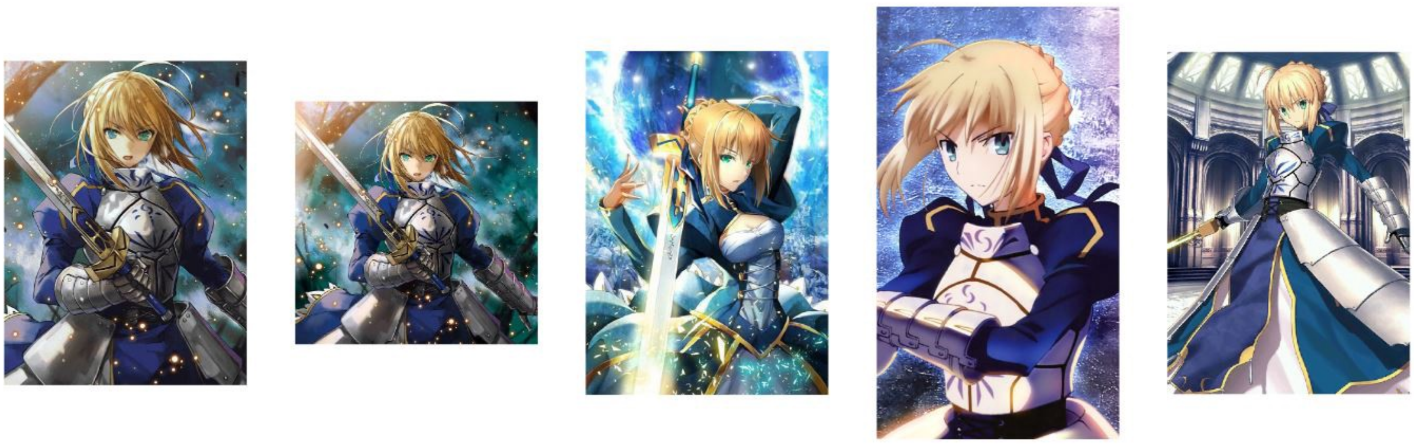
After thinking through how to get an even smaller final feature representation than the Resnet 34's 512 output and drawing a blank for awhile… I felt like an idiot because for another post I have been drafting I have been training a Pytorch based multi-task model on this same dataset to perform multi-label classification across 5 categories with 26 different target variables.

This network is a Pytorch Resnet 50 where I modified the final layers to handle 5 tasks that I try to optimize them simultaneously. I have that post in draft mode but have been too lazy to finish it… but in this case that network is very handy because it gave me access to a fine tuned network that has become skilled at differentiating Fate Grand Order images. The benefit here is that rather than having to use 512 or 1000 output nodes this particular network is skilled in this problem space and only has 26 output nodes which means I can represent the faces and base image with a vector of 52 dimensions rather than 1024 or 2000. This denser representation of these images should allow annoy to better differentiate between the images.

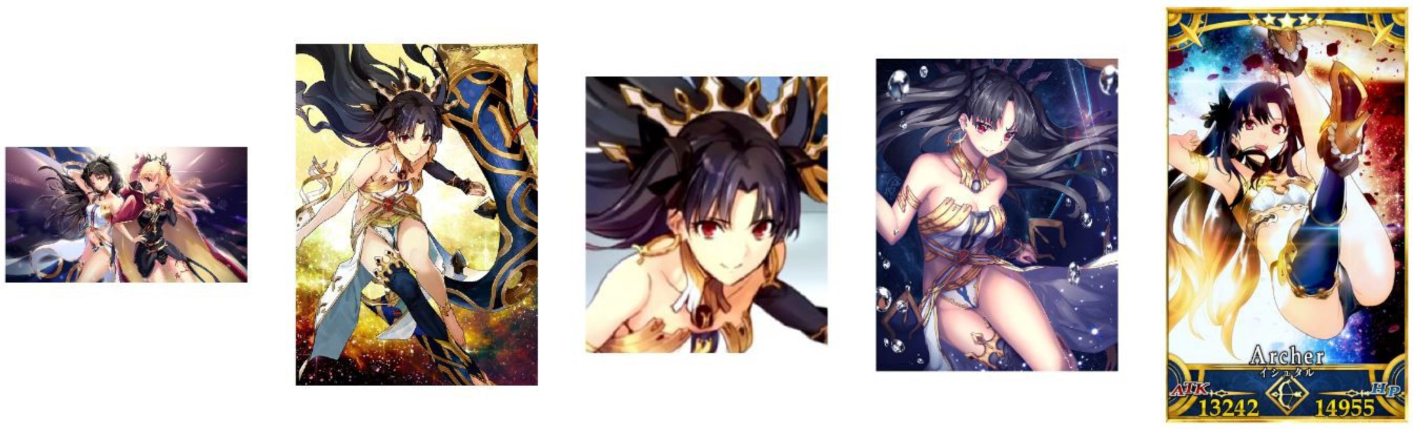
So after swapping out the backend feature extractors and testing the output this V5 model using my multi-task model for feature extraction appears to perform quite well.

See below for the two examples I highlighted in this post so far. The other models performed alright in areas but were lacking in others. I think that this version 5 model does get that best of both worlds I mentioned in my previous post.

Version 5 fine-tuned multi-task model



All of the output is of the correct character, and the images style/background seem more similar than just the v2 headshot model



Once again all the output is of the correct character

Overall this change increased the quality of the pipeline output significantly. I guess it highlights the importance of getting good feature representations and that making them denser is usually better.

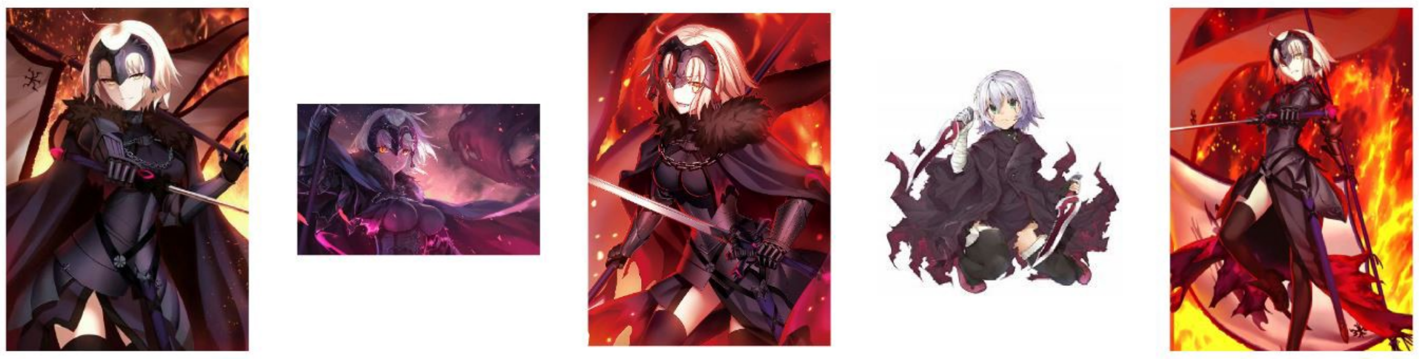
V5 sample output, same character in all images

Closing Thoughts

These two posts were a good reminder on the importance of reducing the dimensionality of your data in order to get better results.

Each of the images start at 150,528 pixels (224x224x3 width, height, rgb color) then with a standard pretrained imagenet network can be condensed down to only 1000 features. I think that is pretty cool even on its own.

However for this specific problem when I added a second image my final feature vector was 2000 features. This was once again too sparse in information for annoy to produce good results so I had to go through and find a way to even further reduce the dimensionality of my data.



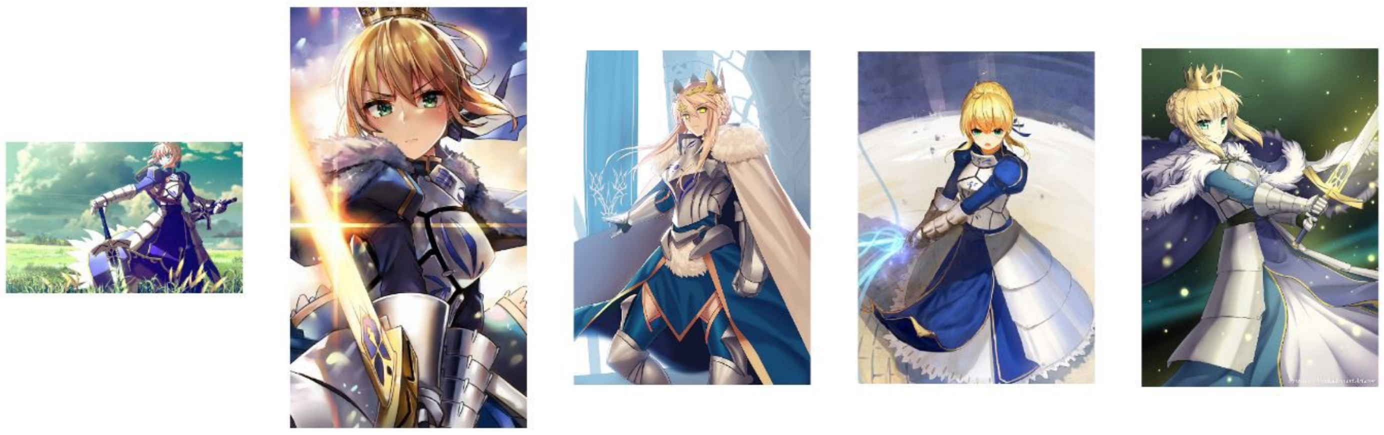
V5 output, 3 out of 4 images have the correct character and decent style matches. hair color in the odd image is a close match

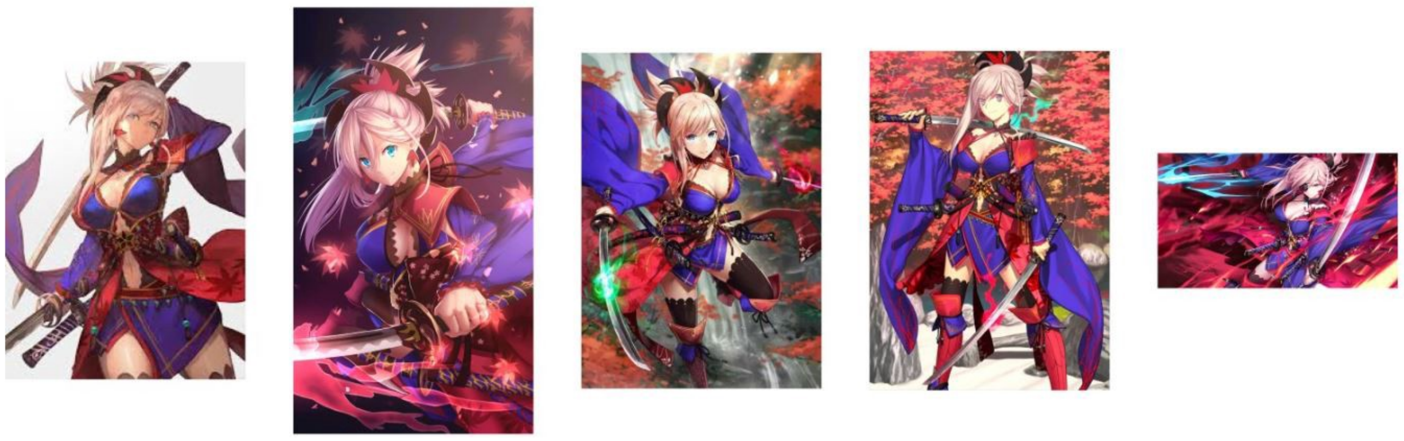
To further reduce the dimensionality of my representations of the images I used a smaller Resnet 34 and saw some improvements but the model would output odd similar images. Since that wasn't working I figured I could try and make an even denser representation of the data and finally I was able to use a network I had fine tuned in a similar problem space to get that original 150,528 pixels raw representation down to 26.

This is a similar process to feature selection or PCA where the goal is to reduce some raw level of input down to a set of meaningful features.This is also a useful note for myself for when I add in more additional context using object detectors or other inputs to try and further focus the details that a specific network learns.

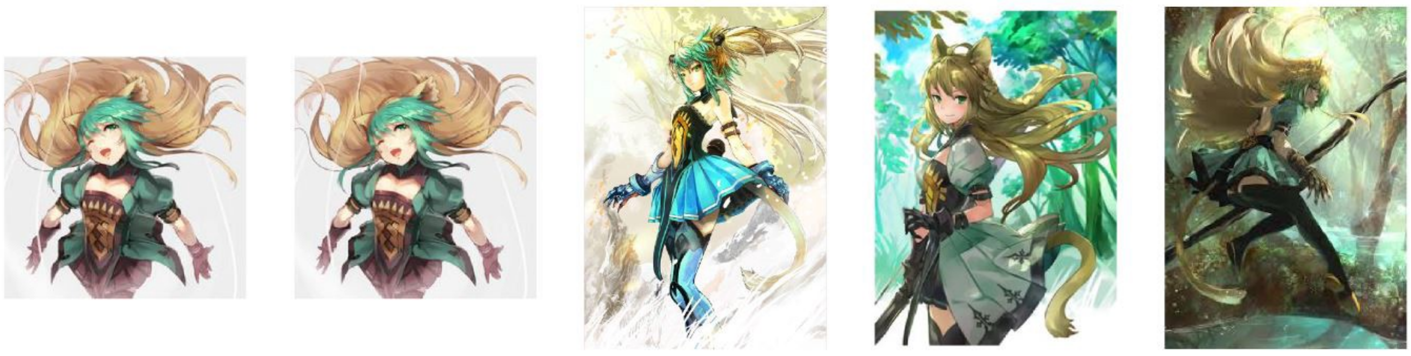
> *Feel free to check out the github repo for samples of the code.*

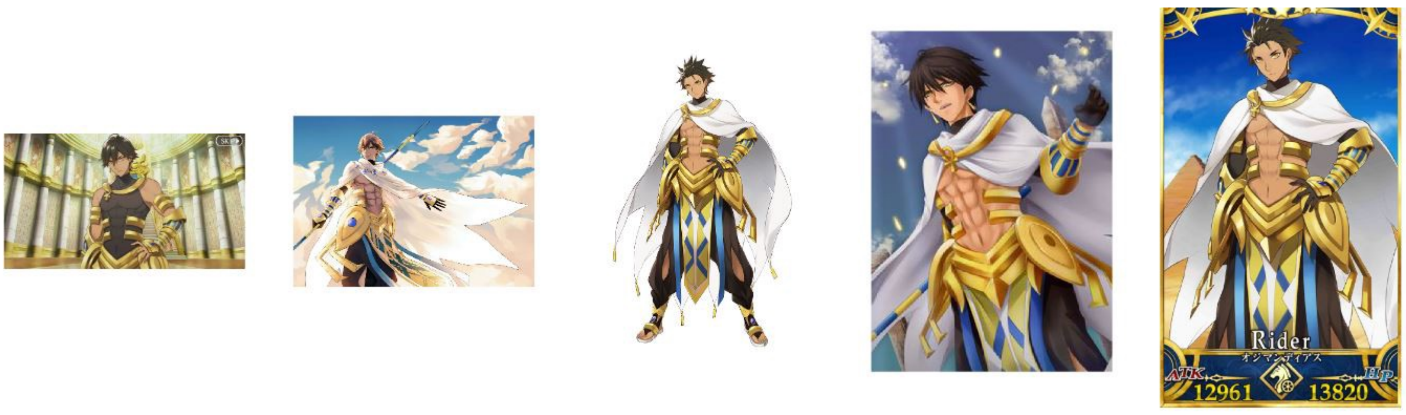
Since I am excited that this seems to be working well here are a bunch more image examples below.





沖田総司
Fate Grand Order




Rider
オジマンディアス
ATK
12961
HP
13820