Who's That Pokemon

"Go Pokemon" Team Members:

- Xiaonan Liu; Email: `xiaonan6@seas.upenn.edu`
- Shuhan Zhang; Email: `shuhanz@seas.upenn.edu`
- Simeng Shen; Email: `shensm@seas.upenn.edu`

Abstract

The field of image classification has shown great success in the past few years because of neural networks, especially CNN. However, apart from the achievements, there is limited attention on artistic classifications like cartoons or anime. This project focuses on classifying one kind of the fictional characters: Pokemon from images to build a "Pokedex". Exploratory data analysis was performed on the images first. Because the data is imbalanced between classes, we experimented with three approaches to achieve equal distribution. We used Convolutional Neural Net as our main model. We also used transfer learning with ResNet50 and VGG19 to serve as comparison. In the end, our CNN model achieved 59% accuracy.

1 Motivation

The ability of human beings to identify and recognize objects from images is undoubtedly perfect. However, for fictional characters, especially in large number of classes, is hard for humans to memorize and identify. Computers, on the other hand, has became more and more accurate in this decade. The appearance of deep learning algorithms in 2012 started a major revolution in many fields, especially computer vision including image classification.

Pokemon, the anime creatures from the world-renowned video game Pokemon Go, have hundreds of different kinds of them. Due to their large number of species and peculiar appearance, it is often hard to classify them. Since all drawings and images of Pokemon are created by artists, they are identified such way. However, this project aims to use deep learning to find an easy way of classifying them. The intuition behind this project is that although illustrations have very different properties considering edges, textures, and strokes from natural images, at a higher level, the essential parts that consist an image are the same. In fact, anime images usually contain less information in the background, which is easy for machine to learn the most important thing: the characters.

The applications of this work could be used to filter Pokemon images scrapped from the web. In addition, the structure of our Convolutional Neural Network can be modified and retrained to classify many other cartoon characters.

2 Related Work

Image classification is a popular field of deep learning. AlexNet [1] is the first deep learning model that outperformed classical computer vision models. With its large improvement of over 10%, neural networks became much more popular. VGG [2], GoogLeNet [3], and most recently ResNet [4], further improved AlexNet with more and more layers introduced along with other techniques like inception modules and "skip connections". These were some of the models we tried using transfer learning.

As for human-created images classification, there re some previous studies about classification on art works [5][6], which their success using CNN classifiers prompts us to go one step beyond by trying deep learning on cartoon characters. There are still few studies on applying nerual networks to comics. Manuel

Lagunas and Elena Garces in their work [7] talked about using transfer learning for illustration classification. They started with VGG19, and created two models based on that. The adjustments from the baseline VGG19 improved performance by 30%-60%, which is a significant number. Similar to the above work, Crowley and Zisserman [8] started the transfer learning problem with a pretrained model based on dataset of paintings, which is closer to comics than real life images. However, there is still a large gap between paintings based on real life and anime drawings. Despite these minor problems, we still want to use transfer learning to serve as comparison because of the amazing results achieved by them.

As for Pokemon images, there are already some different methods proposed in blog posts and Kaggle. For example, from an online publication of Yish Lim [9], she defined several functions to classify the pokemon image, including function returning the original and best-matching image, function returning the average pixel distance between two images and function returning a list of distances between the input image and images of every Pokemon and function nesting the three previous functions to output the result. The idea of her method is to use clustering to classify the Pokemon.

Two blog posts[1][2] also helped us during data engineering. Because of the imbalanced data, we need ways to oversample or undersample. Shubrashankh Chatterjee proposed technique for data augmentation, which involves using ImageFilter from PIL to blur or unsharp images and put in the training data to oversample. Similarly, Aditya Bhattacharya proposed using SMOTE to oversample the minority class and achieve class balance.

3 Dataset

We are using Pokemon Generation One data from Kaggle[3]. The dataset contains 10842 images from 150 different categories, which sums to a size of 2.46Gb. The large volume of this dataset challenges our computers' RAM while running models, which forced us to shrink images size to save computation power.

Because there are so many types of Pokemon, some of the most popular types can have extremely high number of appearances while the less known types have little images. Thus, the data is imbalanced. Most classes have around 60 images with a few outliers. The most frequent class has 307 images, whereas the least has 41. The first and third quantile is 59 and 67. Below is the lineplot of counts of images in each category.

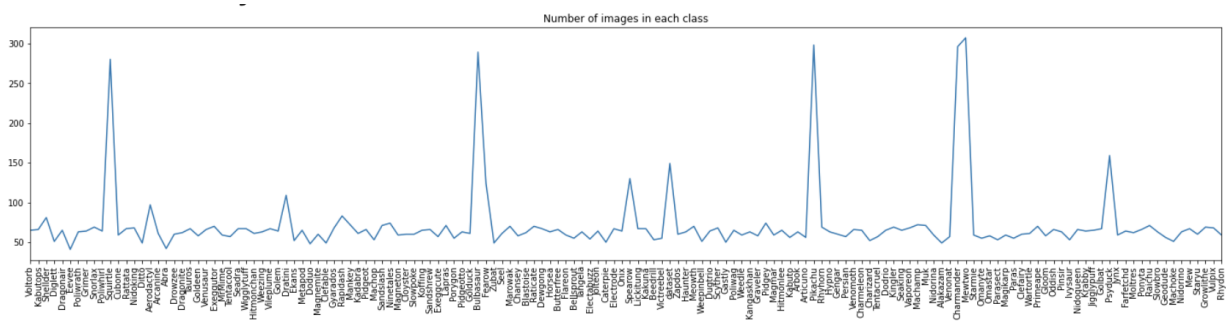


Figure 1: Line plot of Number of Images in Each Class

To handle the imbalanced problem, we decide to compare between four approaches. For the first approach, we generated more images based on the existing ones by applying transformations including blurring, unsharpening, and rotations, until all classes have at least 235 images.

[1]https://medium.com/swlh/how-to-use-smote-for-dealing-with-imbalanced-image-dataset-for-solving-classification-problems-3aba7d2b9cad

[2]https://towardsdatascience.com/deep-learning-unbalanced-training-data-solve-it-like-this-6c528e9efea6

[3]https://www.kaggle.com/thedagger/pokemon-generation-one

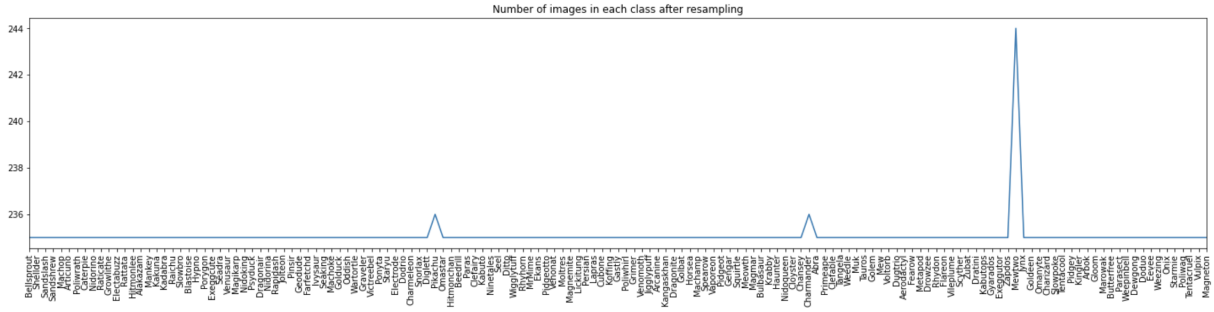


Figure 2: Line plot of Number of Images in the training set in Each Class after oversampling

Secondly, we undersampled the larger classes by selecting only 50 images at random, until all classes have at most 50 images. We create a more balanced set by discarding lots of examples from the most frequent class.

After undersampling, the number of counts for each pokemon category can be seen from the line plot below. The maximum number of images is 50; the minimum number of images of certain pokemon is 32; and the standard deviation is 3.76. We can see that our training set is more balanced than the original dataset.

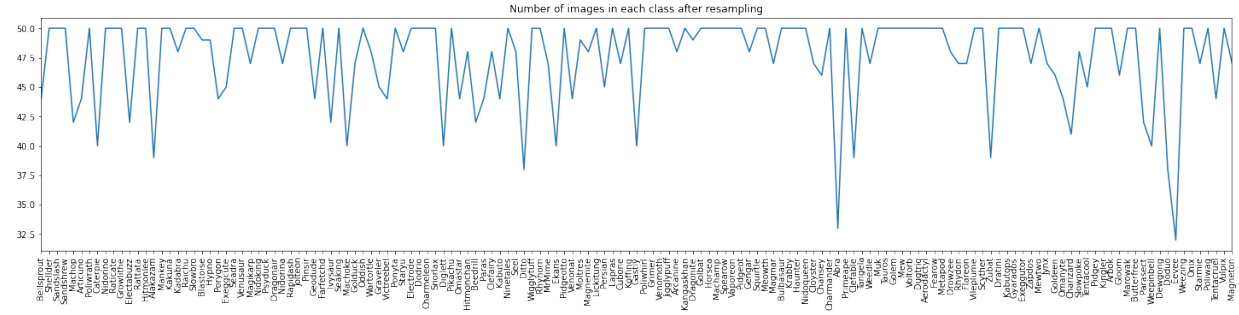
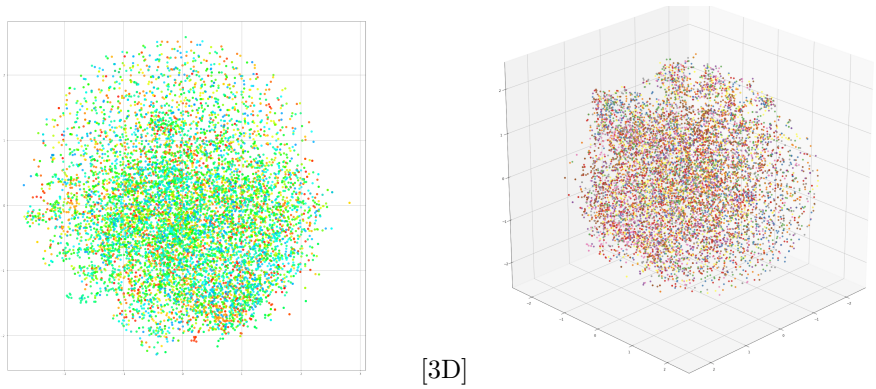


Figure 3: Line plot of Number of Images in the training set in Each Class after undersampling

Lastly, we apply no change to the current images, and run our models as comparison.

To visualize our data, we used t-SNE method to reduce our dimensions. Below is the result graph by reducing all images in all labels to 2 and 3 dimensions.



[2D] [3D]

Figure 4: t-SNE in 2D and 3D Space with All Data

As we can see, there are no clear clustering or separations between groups. It may be due to the reason

that there are too many classes to visualize, which not only makes the color scheme extremely similar between labels, but also interfere with our interpretations. It is the same scenario with 3 dimensions visualization.

Thus, we decided to only plot the first 9 classes to see if there are any signs of separations.

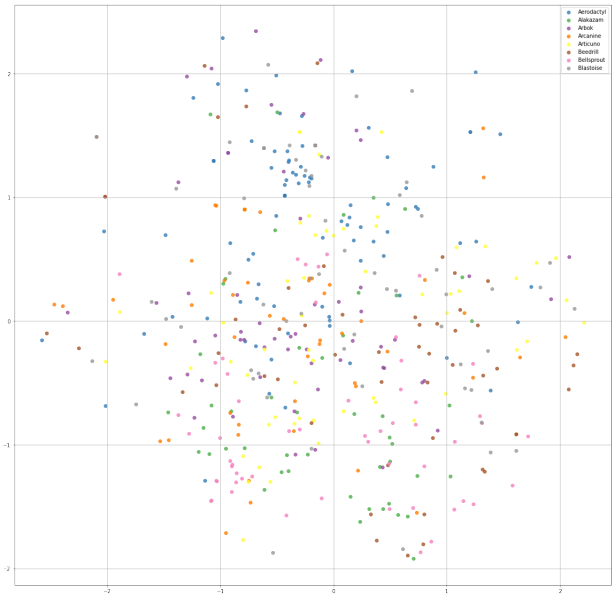


Figure 5: t-SNE in 2D Space with Images in the First 9 Labels

It is as expected to not able to find any relationships or separations, since according to PCA analysis, we need as large as 300 components to achieve 90% explained variance. So it doesn't help much to visualize our data using dimensionality reduction.

4 Problem Formulation

We formulate our project as a supervised learning problem with classification task. By training models with images with labels, we want to predict Pokemon species using our model for the images in the test data set. We decide to look at accuracy as our primary measure when evaluating the performance of each model. Since there is no real harm in a type I or type II error, and the cost of both errors are equal, there is no need to add recall or precision to our evaluation metrics.

We will also evaluate our models based on processing speed. Since this is a large dataset, the oversampling method may take significantly longer time to train, which is a big disadvantage considering we have limited time and computing power.

5 Methods

5.1 Baseline

For baseline, we trained a simple neural net with one convolutional layer and one dense layer. This will perform just slightly better than a logistic regression achieved through Keras by using only one dense layer. Although we want to make limited improvements and applied this model directly to the imbalanced data without any sampling methods, to serve as comparison, we still ran the model using all three sampling methods. We choose this baseline because it is computationally friendly, which is really important in context. We should not spend hours training a baseline. In addition, it provides some insights about what structures should we consider in our CNN model, since it uses the same structure.

5.2 CNN

Convolutional Neural Network (CNN) works best for the data that can be represented in a spatial manner such as images in MxN pixels. Therefore, for our pokemon classification problem, CNN will be a good model to use. And we built our CNN achitecture layer-by-layer with Keras API.
In our CNN achitecture, we have utilized 4 kinds of hidden layers. The first kind of hidden layer is a standard convolutional layer (Conv2D). There are 32 filters in the first hidden layer. Every filter should be contributing to identifying one aspect of the image, such as shape, diagonal boundaries, vertical boundaries, colors, and so on. Then we add a layer to normalize the inputs (Batch normalization layer), by applying a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. Then we add the third kind of hidden layer: max-pooling layer (MaxPool2D) to reduce the computational complexity and provides translation invariance. Lastly, we add dropout layers to help prevent overfitting to improve the model performance. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time to help with overfitting prevention. Also, Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged. And we designed our achitecture by adding more above layers and finally to flatten and dense those hidden layers to make our predictions in terms of pokemon category. There are 9,889,813 total parameters and 9,887,445 trainable parameters.

conv2d (Conv2D)	(None, 96, 96, 32)	896
batch_normalization (BatchNo	(None, 96, 96, 32)	128
max_pooling2d (MaxPooling2D)	(None, 48, 48, 32)	0
dropout (Dropout)	(None, 48, 48, 32)	0
conv2d_1 (Conv2D)	(None, 48, 48, 64)	18496
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
conv2d_2 (Conv2D)	(None, 48, 48, 64)	36928
batch_normalization_2 (Batch	(None, 48, 48, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73856
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
conv2d_4 (Conv2D)	(None, 24, 24, 128)	147584
batch_normalization_4 (Batch	(None, 24, 24, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
flatten (Flatten)	(None, 18432)	0
dense (Dense)	(None, 512)	9437696
batch_normalization_5 (Batch	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
batch_normalization_6 (Batch	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 149)	38293

Total params: 9,889,813

Figure 6: Structure of CNN

5.3 VGG16

For transfer learning using a pre-trained VGG16 from Keras, we only changed the dense layers. However, VGG16 has a large number of parameters, making it not only hard to load in memory, but also extremely slow to train. We already chose VGG16 over VGG19, which has even more layers and parameters, but even VGG16 requires too much computing power. For the last few layers, we used one flatten layer and 2 dense layers. The structure of our VGG16 model is shown below.

Layer (type)	Output Shape	Param #
input_9 (InputLayer)	[(None, 96, 96, 3)]	0
block1_conv1 (Conv2D)	(None, 96, 96, 64)	1792
block1_conv2 (Conv2D)	(None, 96, 96, 64)	36928
block1_pool (MaxPooling2D)	(None, 48, 48, 64)	0
block2_conv1 (Conv2D)	(None, 48, 48, 128)	73856
block2_conv2 (Conv2D)	(None, 48, 48, 128)	147584
block2_pool (MaxPooling2D)	(None, 24, 24, 128)	0
block3_conv1 (Conv2D)	(None, 24, 24, 256)	295168
block3_conv2 (Conv2D)	(None, 24, 24, 256)	590080
block3_conv3 (Conv2D)	(None, 24, 24, 256)	590080
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0
block4_conv1 (Conv2D)	(None, 12, 12, 512)	1180160
block4_conv2 (Conv2D)	(None, 12, 12, 512)	2359808
block4_conv3 (Conv2D)	(None, 12, 12, 512)	2359808
block4_pool (MaxPooling2D)	(None, 6, 6, 512)	0
block5_conv1 (Conv2D)	(None, 6, 6, 512)	2359808
block5_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block5_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
block5_pool (MaxPooling2D)	(None, 3, 3, 512)	0
sequential_8 (Sequential)	(None, 149)	2436245

Total params: 17,150,933
Trainable params: 17,150,933
Non-trainable params: 0

Figure 7: Structure of VGG16

This model has over 17,150,933 parameters, which is almost twice the size of our CNN model. In addition, because this model's early appearance in history, it does not implement batch normalization and residual networks, which are not invented yet; thus, the vanishing gradient problem and its large training time caused a lot of problems. One epoch takes around 60 minutes to train, making the total training time 40 hours or more. Since training time is an important metrics by our definition, we decided to stop using this model and try other pre-trained models that is computationally friendly.

5.4 ResNet50

Another pre-trained model we used is ResNet50 from Keras. ResNet50 has 152 layers, which is significantly larger than VGG16, yet results in faster training speed. The main reason is the implementation of residual connections technique and Batch Normalization. For the last, we added a one dense layers. The structure of our ResNet model is shown below.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 8: Structure of ResNet50

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2048)              23587712
_________________________________________________________________
dense_3 (Dense)              (None, 149)               305301
=================================================================
Total params: 23,893,013
Trainable params: 23,839,893
Non-trainable params: 53,120
_________________________________________________________________
```

Figure 9: Structure of our ResNet model

The ResNet model has totally 23,893,013 trainable parameters, which has the largest number of paremeters among all four methods. However, it is still significantly faster than VGG16. On the other hand, its computational need exceeds our threshold while using oversampling method. Thus, we discarded using ResNet50 on oversampling method.

6 Experiments and Results

We evaluate our results based on the accuracy of classification for all the classes. During the process, the first thing we did was to deal with the class imbalance issue. So we first split the dataset into training set, validation set, and test set. 80% of the images of every category went into training set; 15% of the images of every category went into training set; 5% data went into test set. And then, we experimented on three methods(over-sampling, under-sampling, and control group–no modification on training set) to see whether class imbalance is an issue and also to deal with this potential issue to improve the model performance. We need to do oversampling after splitting the data into training, testing, and validation sets because if there are copies of similar images in both training and testing sets, the accuracy may not be as reliable as it should be. Also, in the process of training, we ran into several technique difficulties, such as google colab platform out of RAM, disconnecting because running out of TPU limit, and so on. We solve these issues by resizing

our images to be smaller so it will less likely to run out of RAM. And we save our model regularly in case the google colab platform disconnect and our model will have to be trained again for a long time.

6.1 CNN with original data

HyperParameters:

Batch Size:100
Optimizer:Adam
Loss Function:Softmax categorical cross entropy
Epochs: 50
Per Epoch Time on average: 542s

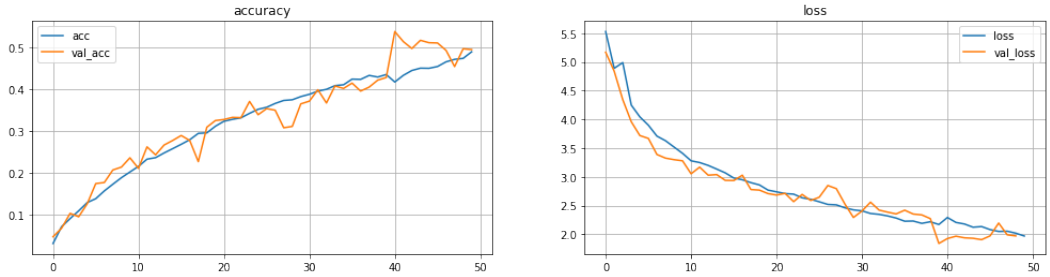


Figure 10: CNN learning Curve

6.2 CNN with undersampling

HyperParameters:

Batch Size:100
Optimizer:Adam
Loss Function:Softmax categorical cross entropy
Epochs: 40
Per Epoch Time on average: 440s

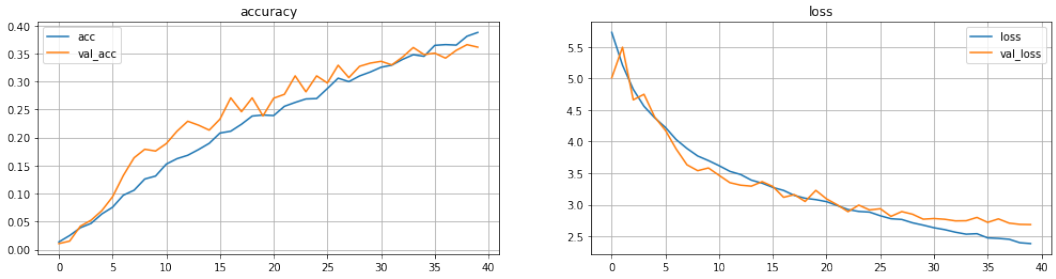


Figure 11: CNN learning Curve

6.3 CNN with oversampling

HyperParameters:

Batch Size:100
Optimizer:Adam

Loss Function:Softmax categorical cross entropy
Epochs: 20
Per Epoch Time on average: 2100s

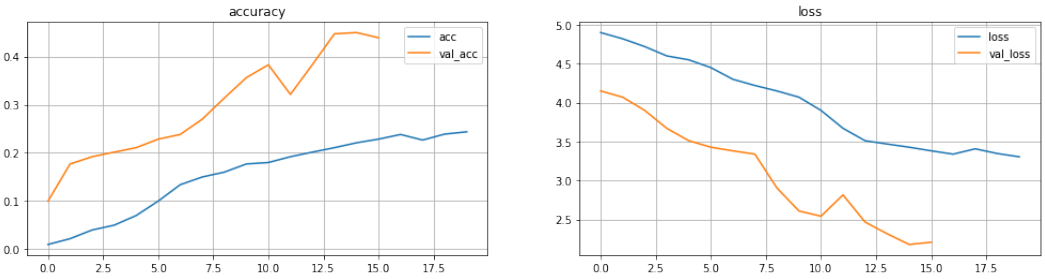


Figure 12: CNN learning Curve

6.4 ResNet50 with original data

Epochs:20
Per Epoch Time:1430s

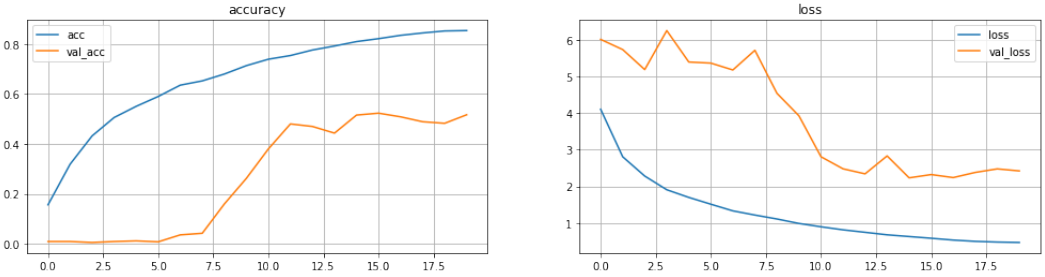


Figure 13: ResNet50 learning Curve with no sampling

6.5 ResNet50 with undersampling

Epochs:20
Per Epoch Time:1200s

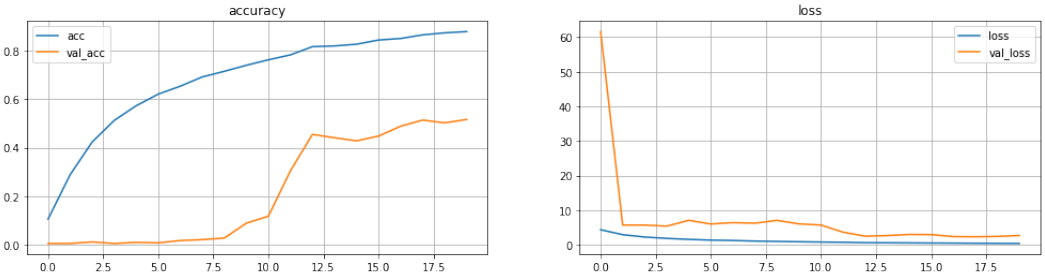


Figure 14: ResNet50 learning Curve with undersampling

6.6 ResNet50 with oversampling

Epochs:13
Per Epoch Time:2400s

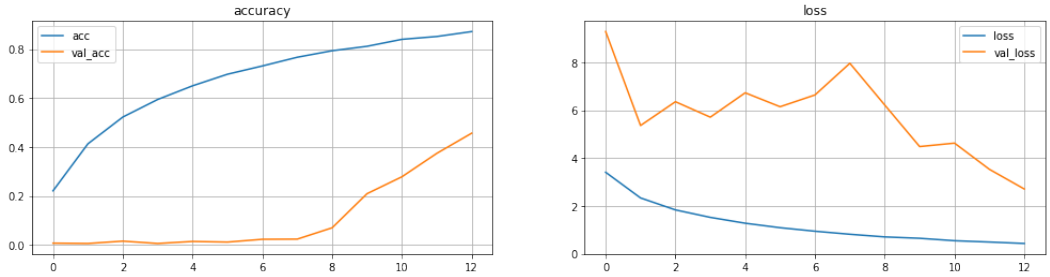


Figure 15: ResNet50 learning Curve with oversampling

Evaluation Metrics:

Model	Undersampling Accuracy	Oversampling Accuracy	No Sampling Accuracy
baseline	0.63 %	0.85 %	0.49 %
CNN	43 %	59 %	57 %
ResNet50	53.125 %	44%(13 epoch)	47.94 %

Table 1: Test Accuracy of Different Models for Classification

7 Conclusion and Discussion

This study proposed to use CNN and transfer learning to classify comic characters: Pokemon. Three approaches were implemented to handle the imbalanced data, including oversampling, undersampling, and control group. Four models were designed including baseline, while only three were actually trained. Based on the results from the previous section following conclusions can be drawn:

- CNN model we built yields better accuracy than ResNet50 and baseline. Training the full model instead of the last few layers tends to give better performance. In addition, The models in transfer learning are popular models pre-trained with significantly larger amount of data with a size of one million images, which makes them easy to overfit with such complexity, especially on training data that only contains 10000 images. Thus simpler models tend to work well for our study.
- CNN model also trains faster than both ResNet50 and VGG16. It is about three times faster than ResNet50, and ten times faster than VGG16. These pre-trained models have too much parameters, which slowed down training speed significantly.
- As for sampling methods, oversampling methods have higher accuracy than the other two. On the other hand, oversampling methods take much more time to train, because we included much more images in such method. Undersampling is still a valid strategy, improving accuracy from unsampled data for around 8%. It is also efficient in regard with training time. As for data with no sampling methods, not only are methods yielding lower accuracy, the imbalance would also cause accuracy as a metric less valid.

An important drawback of our study is the limitation of computing power. With the large amount of images, we are limited from training on larger batches, better quality images, and more epochs. The training time as long as an hour per epoch makes it hard to make adjustments. So, future work can certainly incorporate machines with better computing power, or use distributed computing systems. In addition, future work can also incorporate other methods to deal with the class imbalance issue. First, SMOTE can be used to obtain a more class balanced dataset. Second, a new loss function which gives a higher penalty for getting a rare class label wrong than for getting an item from a common class wrong can be used.

References

[1] Krizhevsky Alex, Sutskever Ilya and Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems, vol 25*, 2012.

[2] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: ICLR; 2015.

[3] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR); 2015.

[4] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR); 2016. p. 770â8.

[5] Tan WR, Chan CS, Aguirre HE, Tanaka K. Ceci nâest pas une pipe: a deep convolutional network for fine-art paintings classification. In: 2016 IEEE international conference on image processing (ICIP); 2016. p. 3703â7.

[6] Hicsonmez S, Samet N, Sener F, Duygulu P. Draw: deep networks for recognizing styles of artists who illustrate childrenâs books. In: Proceedings of the 2017 ACM on international conference on multimedia retrieval. ICMR â17; 2017. p. 338â46.

[7] Lagunas M and Garces E. Transfer Learning for Illustration Classification. *Spanish Computer Graphics Conference (CEIG)*, 2017.

[8] Crowley E. J., Zisserman A.. In search of art. *European Conference on Computer Vision (ECCV)*, 2014.

[9] Lim, Yish. "Who's That Pokemon?" Medium, Towards Data Science, 13 Jan. 2019, towardsdatascience.com/whos-that-pok%C3%A9mon-39d1150aedfe.