

Identifying Pokémon Types

Mihai Matei

Practical Machine Learning Course

Data Science Master, 2019

University of Bucharest, Faculty of Mathematics and Computer Science

Abstract

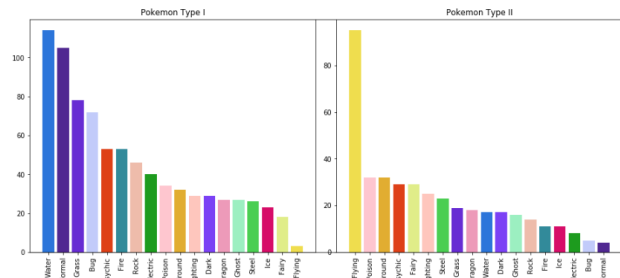
This paper tries to apply machine learning and statistic methods to identify the type of a Pokémon from a given image, without any prior knowledge of that particular character. Thus this is not an image recognition challenge, rather to figure out the type of a Pokémon given a totally new character. Any correlation in the design of Pokémon of a particular class by the human artists should be identified by machine learning methods. I will discuss the challenges involved, what I could or could not overcome, as well as present my findings.

1. Introduction

Pokémon are very well known fictional creatures made popular around the world through movies and popular computer game series, like the whole madness of 2016. The Wikipedia page [1] describes the "890 fictional species of collectible monsters, each having unique designs and skills. Apart from their fantasy appearance each Pokémon has one or two “types”, such as Fire, Water, or Grass. In battle, certain types are strong against other types. For example, a fire-type attack will do more damage to a grass-type Pokémon than a water-type attack, in a rock-paper-scissors-style relationship. In order to test if the specific Pokémon type can be inferred from a collection of previous types I chose a database consisting of all types and applied two different machine learning models: a popular Convolutional Neural Network model created using transfer learning from a state-of-the-art model as well as the ever popular Support Vector Machine with some statistical feature selection. In order to support different machine learning library used for the models I implemented a helper python package, `matmih`, to have a common layer between the frameworks. Even though the classification results are not ideal they perform better than the human intuition for this task.

2. Pokémon Dataset

The data set [2] was chosen to contain all pokémon character images one time only and was taken from the popular Kaggle repository. It consists of a list of 809 characters, with their specific two types of attack, as well as 809 120x120x3 RGB images, one for each. All pokémon contain type1 information, but only 405 have the additional type2 provided. There are 18 target classes of pokémon types to use in the multi label classification (Bug, Dark, Dragon, Electric, Fairy, Fighting, Fire, Flying, Ghost, Grass, Ground, Ice, Normal, Poison, Psychic, Rock, Steel, Water).



You can notice that the target class distribution in the above histogram is quite unbalanced, being dominated by the Water and Normal class for type1 and for Flying for Type2.

2.1. Data set split and preprocessing

In order to apply machine learning techniques and test their performance we do a standard train, validation, test **random** split with the weights 0.9, 0.05, 0.05 due to the fact that the sample size is quite limited. Since the class distribution is highly unbalanced, **before** doing the splits, I first augment the data by also adding the type2 information as a new type1 entry. As such the total resulting data will be 1214 samples. In order not to have multiple target labels with the same feature vector, the augmented data images will suffer certain modifications such as being rotated by 20 degrees. Then the test set is chosen randomly from the

Pokemon Type after merge (train)



















Pokemon Type	Count
Water	110
Fire	90
Grass	78
Electric	75
Ice	70
Fighting	65
Poison	58
Ground	55
Rock	48
Bug	45
Dragon	42
Steel	40
Normal	38
Psychic	35
Dark	32
Fairy	30
Ghost	28
Other	25

Pokemon Type after merge (validation)

Pokemon Type	Count
Water	7.0
Fire	5.0
Grass	5.0
Electric	5.0
Ice	4.0
Fighting	4.0
Poison	3.0
Ground	3.0
Rock	3.0
Bug	3.0
Dragon	3.0
Steel	3.0
Normal	3.0
Psychic	3.0
Dark	3.0
Fairy	3.0
Ghost	3.0
Other	2.0

2.2 Image generation for augmentation

Bug	Dark	Dragon	Electric	Fairy	Fighting	Fire	Flying	Ghost
								
Grass	Ground	Ice	Normal	Poison	Psychic	Rock	Steel	Water
								

type ? 	type ? 	type ? 	type ? 	type ? 	type ? 	type ? 	type ? 	type ? 
type ? 	type ? 	type ? 	type ? 	type ? 	type ? 	type ? 	type ? 	type ? 

Pokemon Type after augmentation (train)

Pokemon Type	Count
Dark	135
Electric	130
Normal	115
Fire	110
Water	105
Grass	100
Fighting	95
Ground	90
Psychic	85
Ice	80
Dragon	75
Fairy	70
Poison	65
Rock	60
Shadow	55
Steel	50
Ghost	45
Light	40

Pokemon Type before augmentation (train)

Pokemon Type	Count
Dark	135
Electric	100
Normal	80
Fire	75
Water	70
Grass	65
Fighting	60
Ground	55
Psychic	50
Ice	45
Dragon	40
Fairy	35
Poison	30
Rock	25
Shadow	20
Steel	15
Ghost	10
Light	5

To assess the overall performance of any algorithm we have to take into account that random choice model as well as the human accuracy level for the task. Since there are 18 target classes the expectation accuracy of random model is $1/18 = 0.0555\dots$. To confirm this a random classifier was implemented and the accuracy result for running the classifier 1000 times was (0.0555, 0.0547, 0.0564) on the train/validation/test set. To further analyze the random model we also tested the accuracy rate for the random model that predicts the target class according to the class distribution in the training set, yielding a slightly higher accuracy of (0.0666, 0.0676, 0.0617). Human intuition appears to be better than the random performance only

in a few classes that are associated with a more “trained” human perception, namely Bug, Dragon, Fire and Flying class. In the example only Bug, Poison, Rock and Water class have a decent human level intuition, but outliers are present even for those.

4. Transfer learning with CNN

Convolutional neural networks have been extremely popular in recent years for image recognition. Since convolutional neural network creation requires a lot of trial and error modeling most of the current work being is to address some of the limitations and behavior encountered in state of the art CNNs. Online literature that contain updated performance ratings and original papers can be consulted [3]. A transfer learning approach was chosen for the Pokémon type classification. Among the many CNNs available, such as VGG or ResNet, MobileNetV2 was chosen due to low memory footprint (more tractable hyper parameter search) and its trained availability in Keras framework as a loadable application. When doing transfer learning one important aspect to consider is the data set the model was trained on, it's input feature size, the normalization that was being done, the optimizer used. The original paper of the MobileNetV2 [4] and the online implementation [5] were consulted to get the above information. The imported Keras MobileNetV2 application was trained using the ImageNet data set consisting of over 14 million images and 1000 target classes. The MobileNetV2 CNN consists of 156 convolutional layers (Conv, Padding, Pull, Relu Activation, Batch Normalization) and one last Dense Layer. It has over 3.5 million parameters (1.3 million from its last Dense Layer) and yield a top1 and top5 accuracy of 0.713 and 0.901 with only 14MB [6]. In order to do transfer learning the last Dense layer was removed and a 128 dense layer with ReLu activation and batch normalization was added followed by a dropout and finally a 18 dense layer with Softmax activation.

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_128	(Model) (None, 1280)	2257984
dense_1(Dense)	(None, 128)	163968
batch_normalization	(Batch (None, 128)	512
activation_relu (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 18)	2322
activation_softmax (Activation)	(None, 18)	0

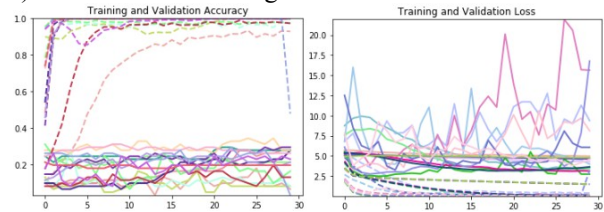
The training of the model is done in batches of 32 images. As per MobileNetV2 paper the image data

features corresponding to each RGB channel was uniformly normalized to $[-1, 1]$.

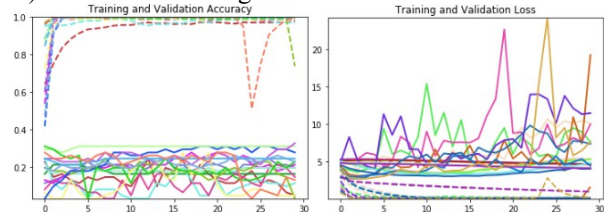
4.1 Hyper parameter selection

To further configure the model a grid search was done in the hyper parameter space consisting of the number of nodes in the first dense layer, the dense layer L2 normalization, the dropout rate, and a selection of different optimizers (Adam, SGD, RMSProp) and learning rates. The trained data was selected with or without image augmentation to test how this influences the model. The loss function was chosen to be the default sparse cross entropy loss and the model was tested with or without applying the class weights from the training distribution. Because the training and hyper parameter space search is exponential I limited the epoch size to 30 since I noticed that the validation accuracy starts to constantly fluctuate after epoch 20 suggesting that the model jumps around a local minima. The optimizers learning rate was chosen to be either their default Keras values or 10 times smaller. Due to the time complexity of the hyper parameter search no cross validation was done to assess the model. Below you can see the train and validation accuracy for different hyper parameter selections using the same color as well as the train and validation loss. The high values of accuracy and the low values of loss correspond to training metrics.

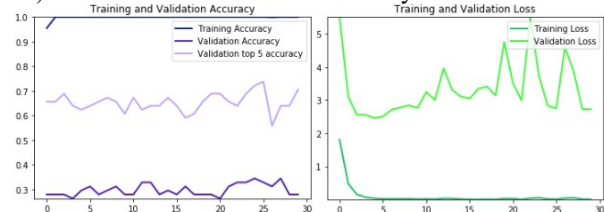
i) Train data without augmentation – all models



ii) Train data with augmentation – all models



iii) Best model – 0.3442623 accuracy



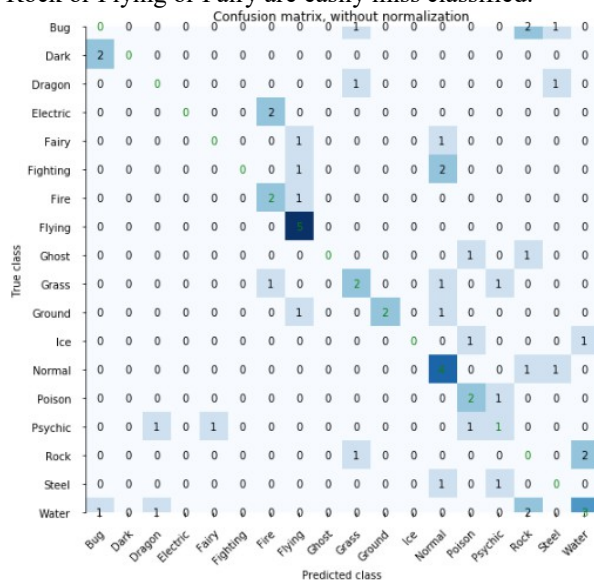
The best model was found using the augmented data set containing the additional generated images with a

dropout rate of 0.4 using the Adam optimizer with a learning rate of 0.001 and no class weights. Top 5 accuracy is over 0.7 plotted above in purple. The total run time for this grid search yielding 32 models with 30 epochs was over 1.5 hours running on a NVIDIA GPU 1070TI. From the plots in i) and ii) one can notice that there is a big difference between the performance of different parameters. Both type of training data, augmented with generated images or not, have kind of the same upper validation accuracy of a little over 0.3. In general it seems that using the RMSProp and Adam optimizer yields better results. The accuracy curves clearly shows over fitting but trying a more simpler model, such as reducing the Dense layers size, did not succeed in improving the overall performance.

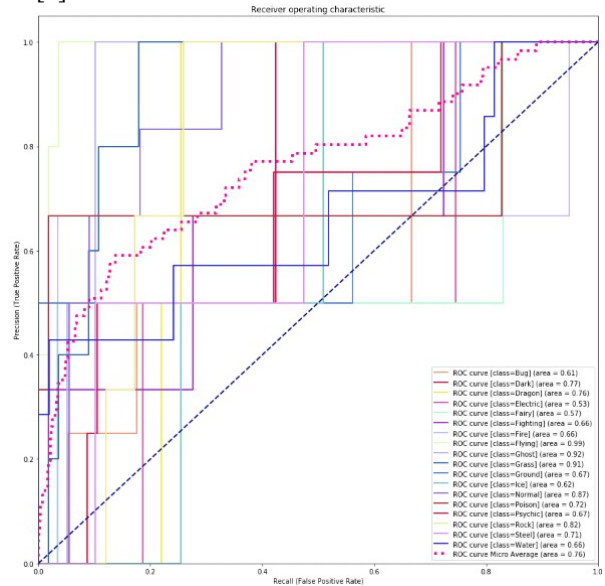
4.2 Best Model analysis

To get the maximum out of the best model we did not retrain the model on the best epoch but rather a did a model checkpoint of the highest validation accuracy found in training. This way there is no variance when retraining the model due to random weight initialization of the Tensorflow framework.

When doing model analysis there are several metrics to use for a classification algorithm [7]. Apart from the accuracy metric used in hyper parameter selection and model validation, to check the individual model performance for the multi class classification (18 classes) the confusion matrix and the receiver operating characteristic curve is a good source of information. This allow us to see how the model performs on the individual classes, thus one can identify if it over fits one class more than the other, or if the training data would be better with more samples from some specific classes. From the confusion matrix computed on the validation set one can notice that certain classes that have more correlated features such as Flying or Water are better identified. Unfortunately other classes that might have more features in common such as Bug and Rock or Flying or Fairy are easily miss classified.



To further analyze the per class performance the ROC curve was plotted for each class along with the micro-averaged red dotted curve that aggregates the contribution of each class to compute the average metric. How ROC curve is computed is best described in [8].

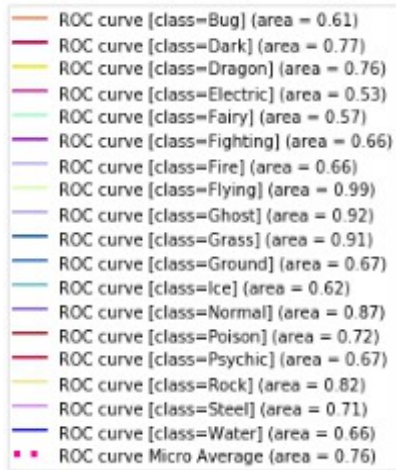


A ROC curve is a two-dimensional plot of Recall (Sensitivity, True Positive Rate) vs False Positive Rate. False Positive Rate is $1 - \text{True Positive Rate}$ (Specificity) which is inversely proportional to the True Positive Rate (Recall). This is why when we vary the classification threshold for the predicted scores for that class we get the above ROC probability curve. In general ROC is a probability curve of a model ability to distinguish a specific class based on the classification threshold. A probability of 0.5, blue dotted line, represents the random model so each ROC should be at least greater than that.

In a multi class classification environment the ROC curve for each class denotes the model's ability to differentiate that specific class from all other. Since the ROC curve is essentially a probability curve, one measure to be used is the Area Under the Curve which gives the overall metric on how the model performs for that class (for a probability distribution curve the AUC would be 1). A perfect model would have an AUC of 1, a random one an AUC of 0.5. An AUC of 0.8 is generally considered to be a good model. There are 3 classes where the AUC is over 0.9 namely Flying, Ghost and Grass. On the other hand the Fairy and Electric class is close to 0.5, meaning the model cannot correctly distinguish those. This translates to the model over fitting those first 3 classes. To improve on this one

might consider adding more training data of the classes that have an low AUC value.

The AUC values for the model are shown below:



The micro-averaged result is 0.76 which is not bad. To improve on this one should get more samples for all the classes with a AUC value under 0.7 or use a custom loss function that penalizes classes that are misclassified.

5. Support Vector Machines

SVM is one of the most popular linear model for binary classification. It extends the classic linear model by adding the concept of support vectors [9]. In the binary classification the SVM tries to find the maximum margin hyperplane that divides the 2 classes. The SVM can also be extended to support data that is not linear separable by introducing the soft-margin classifier and hinge loss. Hinge loss only penalizes predictions that are on the wrong side of the hyperplane and is 0 for the corrected predicted data, thus the classifier does not learn from data that it considers to be correctly classified. The soft margin parameter allows the classifier to ignore some data that is correctly miss predicted and acts as some sort of regularization. To extend the nonlinear classification ability, the SVM can be computed in the dual form that allows to apply different kernels to move the features in an larger multi dimensional simplex where the data is linear separable.

To be able to use all of the SVM extensions dual form implementation was chosen from libsvm implementation in scikit-learn framework. In a multi label classification environment SVM is used as a binary classifier applied for each class in a one-vs-one or one-vs-many approach yielding either $C*(C-1)/2$ or C classifiers and then applying **plurality voting**.

5.1 Data set split and preprocessing

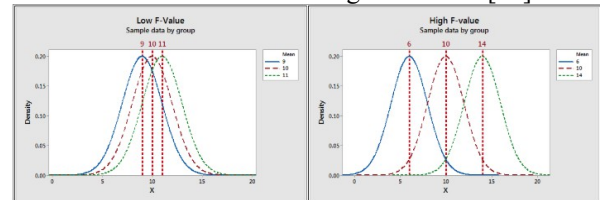
The same data set split was used as in the CNN model with the exception that no image generation augmentation was done on the training split. One limitation of using the dual form is the computational requirements for generating the Gram matrix. In our training case of 1096 samples and 49152 features the kernel is applied $1096*1096/2$ times (the Gram matrix is symmetric) on 2 (1, 49152) vectors. This does not scale well and as a solution filtering the feature space was chosen as a dimensional reduction technique.

Prior to applying a filtering technique the features are first flatten then each 49152 feature is normalized to mean 0 and standard deviation of 1. The transformation matrix learned during the normalization of the training set is also applied to the validation and test set.

The first filter that is applied is to reduce the feature count is to remove features that have a variance of less than 0.1. This will remove most of the common white pixels in the data set, including the ones that were added by increasing the image size to 128x128.

Then the ANOVA F-value [10] is computed for each feature in the training set and the best 1000 features are selected. The resulting transformation mask is then applied to both the validation and test set. In general an F-test is a statistical test that tries to compare that data distribution (F-distribution) under the null hypothesis (a plain model with 0 parameters). In our case the one-way ANOVA F statistic is:

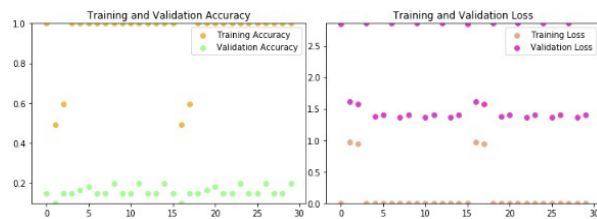
$$F = \text{variance of class means} / \text{variance of all samples}$$
The above F value is computed for all features. A low F value means that the variance between the means of the target class labels is less the overall variance for that pixel channel. This means that that pixel feature might not be a good fit to separate the model. On the other hand a high F value for a feature means that each class mean value varies a lot more with respect to the other classes then the overall variance of that feature. This means that that feature might contain different information for each class. Below one can see how some of the feature variance might look like [11]:



Thus I reduced the training set is now 1096x1000

5.2 Hyper parameter selection

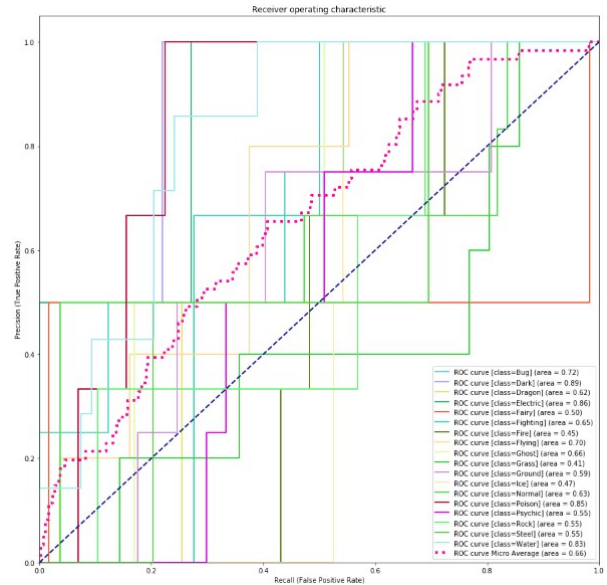
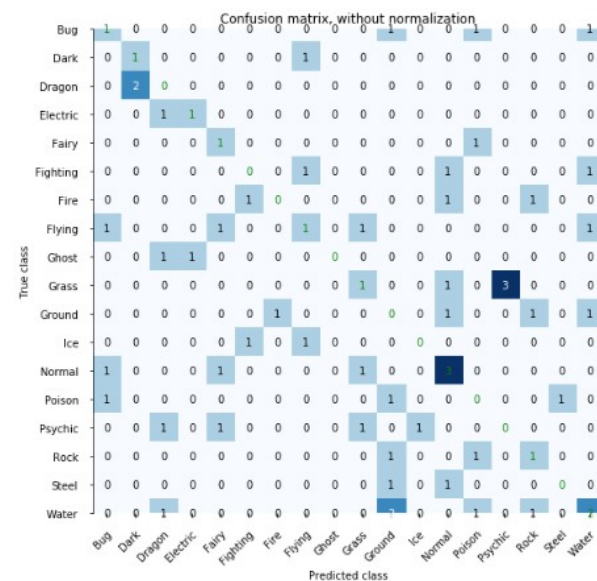
To configure the SVM model in dual form for our multi class classification the soft penalty, the kernel type and kernel parameters were chosen as the model's hyper parameters. A small value of soft penalty means the SVM will act as a hard margin SVM and a higher value will contribute to a more biased model. Each of the hyper parameter combinations will be tested with a one-vs-all, one-vs-one classifier. For the grid search a soft penalty of 0.1, 10, 100, 1000, 10000 was tried along with a linear, 3 degree polynomial and RBF kernel. A total of 30 different models were tried, their accuracy and loss performance shown below:



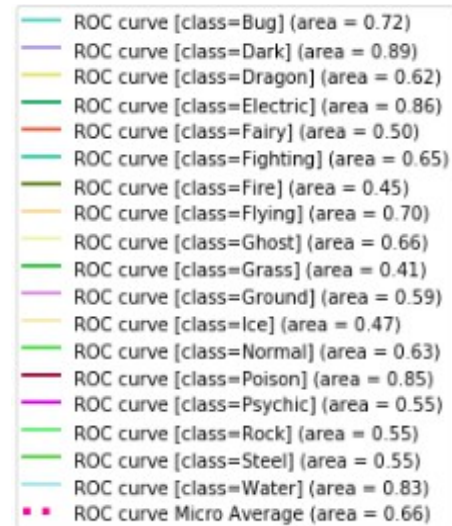
There was not a significant difference between the one vs all and one vs one models, the first being preferred due to the number of individual binary classifiers.

5.3 Best Model analysis

The best model was chosen as the highest accuracy model on the validation set, a SVM one vs all with a soft penalty of 100 and RBF kernel yielded 0.1967 accuracy on the validation set and 1.0 accuracy on the training set, also clearly indicating over fitting in this case as well. The confusion matrix does not show certain class over fitting as in the CNN case, rather the inability of the model to properly differentiate between the classes. A better metric is the ROC curves



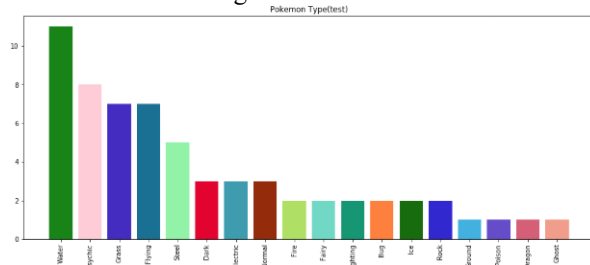
with the micro averaged AUC at 0.66, less than the 0.76 of the CNN model. From the AUC scores we can see that some class are better distinguished such as Bug, Dark, Electric, Water, Poison and other classes such as Ice, Grass, Fairly being left out of the model. This is possible due to the ANOVA feature selection that prefers the features that differentiate better a class from others and certain classes have many more such features.



To address this more ANOVA selected features should be added or using a prior convolution layers to reduce the feature space. Also for the classes that show random classification performance, such as Ice or Rock manually adding certain features (pixels) that denote a good separation boundary can be done along with the ANOVA selection.

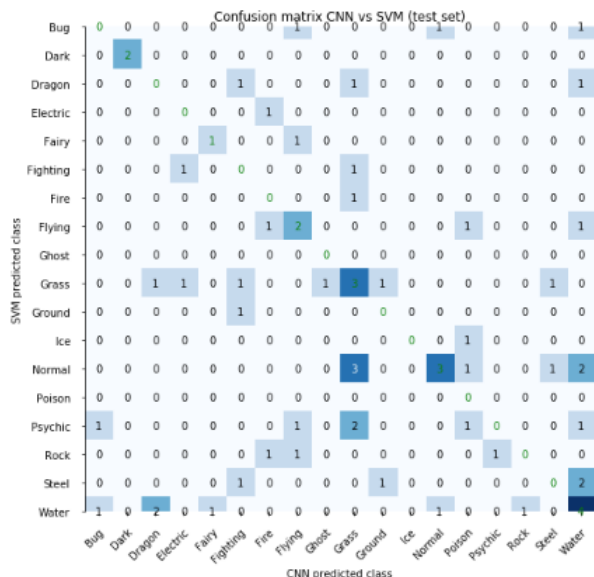
6. Conclusions on test results

To assess the models performance on a totally random data the 2 best models were used to predict the test set. The test set histogram distribution of classes can be seen below and was left to have the same distribution as the original set.



The CNN model yielded a 0.36508 accuracy and the SVM model 0.19048. This clearly shows that the CNN model is the better model that did not over fit the training and validation set as much as the SVM. Also training the CNN model on a more uniform distribution of classes did result in a better performance.

One interesting thing we can learn about the class distribution is creating the confusion matrix of the 2 models versus each other. I am not interesting on the true value of the prediction but rather which classes are better identified by both models. This way one can see which classes exhibit better defined features that both models are prone to over fit them.



From the confusion matrix above one can see that the Water, Normal, Grass, Fire and Dark classes are identified the same (though not necessary correct. This suggest that these classes have features that are more relevant for a convolution layer as well as for an

ANOVA selection. Both model have problems distinguishing the Grass and Normal class. One can use this information to further augment and modify the training set to give the poorly identified classes more specific features, such as using image modification techniques to enhance features that seem to have a more human intuition. For example for the Grass class one can increase the amount of the green channel in that image so that characteristic would be better enforced.

7. References

- [1]https://en.wikipedia.org/wiki/List_of_Pok%C3%A9mon
- [2]<https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types>
- [3]<https://paperswithcode.com/task/image-classification>
- [4]<https://arxiv.org/abs/1801.04381>
- [5]https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet/mobilenet_v2.py
- [6]<https://keras.io/applications/#models-for-image-classification-with-weights-trained-on-imagenet>
- [7]<https://www.innoarchitech.com/blog/machine-learning-an-in-depth-non-technical-guide-part-4>
- [8]<https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>
- [9]<https://practical-ml-fmi.github.io/ML/>
- [10]<https://en.wikipedia.org/wiki/F-test>
- [11]<https://blog.minitab.com/blog/adventures-in-statistics-2/understanding-analysis-of-variance-anova-and-the-f-test>