Multi-Label Image Classification: Rock, Paper, Scissors

Jessica Mele
Operations Research
North Carolina State University
jmele@ncsu.edu

Margaret Tobey
Operations Research
North Carolina State University
mgtobey@ncsu.edu

Johnna Brooks
Biomathematics
North Carolina State University
jbbrook3@ncsu.edu

I. Introduction

Multi-label image classification is a complex challenge. Images are complex data structures that require extensive computational time and memory in order to process and utilize. Currently, image classification techniques can be found in a wide range of disciplines such as medicine, game development, and satellite imaging. In recent years, deep learning algorithms have become an essential tool for image classification problems. In particular, deep Convolutional Neural Networks are a commonly used tool that have shown high performance for these tasks. [4]

Most image classification techniques rely on feature extraction to determine unique characteristics in images of specific classes. The more possible image classes that exist, the more complex the problem becomes. Neural networks provide a unique framework that allow for analysis of large amounts of data at one time that can provide predictions for multiple classes. To handle more complex images, we can add more feature extraction techniques by increasing the hidden layers and experimenting with different activation functions that are incorporated in the model. One potential pitfall of increasing the complexity of the model is the possibility of overfitting to your training set. Some tools to avoid overfitting that we explore in this project are data augmentation and the addition of dropout layers. [5] Some methods of data augmentation can include reflections, rotations and RBG alterations. Dropout can be altered by increasing the 'keep' probability and introducing this layer after some of our fully connected layers. [5] We will compare results of using different combinations of training techniques through experimentation and evaluation with validation and testing sets.

The game 'Rock, Paper, Scissors' requires two people to extend one of three hand gestures at the same time. There is a hierarchy of gestures that determines a winner unless the two individuals extend the same gesture, in which case a tie is determined. The owners of this data set took various pictures of these three gestures in order to develop a 'Rock, Paper, Scissors' computer game. [6] Although this problem is not as large scale as other image classification tasks that have been introduced in class, we felt that the tools used to build this model could be extended to more complex and important tasks. More specifically, we thought a multi-label image classification tool of human hand gestures might be particularly useful in the use Artificial Intelligence for American Sign Language interpretation or translations.

In this paper we explore supervised learning techniques to perform multi-label image classification of a set of images with three classes. [6]

A. Modules and Training Environment

All models were created using a Python, open-source environment. The final model is developed using Keras Sequential API [2] (Tensorflow backend [3]). The baseline methods, a multilayer perceptron and support vector machine classifier, were created using SciKit Learn API. [1] Evaluation and metrics were performed using SciKit Learn. [1] Image pre-processing and data augmentation was performed using Keras. [2]

The model was trained using Google Colab: Intel(R) Xeon(R) CPU @ 2.30GHz.

II. Data

The data consists of images of a variety of hand gestures pertaining to the game 'Rock, Paper, Scissors' that was downloaded from the Rock-Paper-Scissors data set taken from a Kaggle repository. [1]. The entire data set is relatively balanced and contains 2188 images with the following breakdown of images in each category:

	Image Count	Numeric Label
Paper	710	0
Rock	726	1
Scissors	752	2

TABLE I: Class breakdown in complete data set.

Each of the images has a width of 300 and a height of 200. As you can see from Figures 1a, 1b, and 1c the images are presented in RGB format with a green background. The entire data set is relatively consistent in terms of the background color and overall lighting and image sharpness. Each image has a label that corresponds to its image directory, i.e. rock, paper or scissor, which is transformed to one-hot encoding format when imported from the repository. When a

[1]https://www.kaggle.com/drgfreeman/rockpaperscissors

numeric label is used to indicate the class, the classes are in alphabetical order and have the numeric labels indicated in Table I.

III. Methods

A. Image Preprocessing

1) Converting to greyscale: Since all of the images have a neutral, green background we felt that it was appropriate to remove the color from the images. We also felt that this would help train the model on the feature extraction from the actual hand gestures without interference from the green background.

2) Dimensionality reduction: The original images were of size (200, 300). In order to reduce the complexity of our models we reduced the dimensionality of the images so that we had less pixels, but attempted to avoid introducing ambiguity by decreasing the resolution too much. In our final model, the images are reduced to size (100, 150).

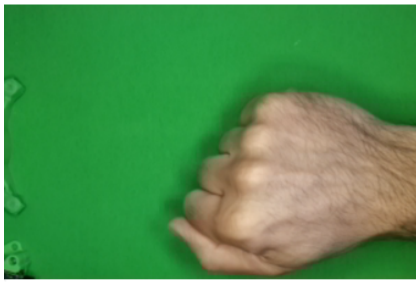
3) Data augmentation: In order to improve performance of the model we decided to introduce some more randomness to the training data set by using a Keras Sequential preprocessing random rotation with a factor of 0.2. A factor of 0.2 results in an image rotation by rotating by a random amount in the range $[-0.2\pi, 0.2\pi]$. Here a negative value indicates a clock-wise rotation whereas a positive value indicates a counter clock-wise rotation. An example of images after being rotated, with their respective labels can be found in Figure 2.

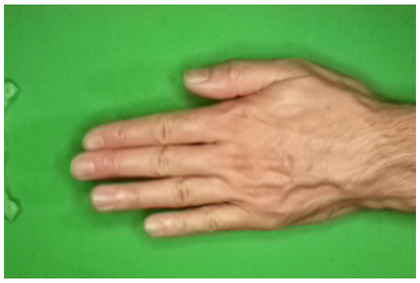
4) Data splitting: In order to train the data we split into (60/20/20) training, validation and testing sets. The augmentation was only performed on the training data.

B. Baseline Methods

1) Vanilla neural network: Our first baseline method is a vanilla neural network (VNN), or a simple multilayer perceptron or linear regression extension. This model flattens the input images into a vector of size (1, 100*150). The hidden layer is a fully connected layer with default weight and bias initializations and a softmax activation function. The default weight initializer is 'glorot_uniform' and the default bias initializer is 'zeros'. The output is a 1x3 vector and the prediction is made off of the softmax values. The Adam optimizer was used to train the model. The VNN was trained and evaluated using both the pre-augmented training data images and the augmented training data images. The VNN performed better when using the pre-augmented training data images and achieved a 0.71 out of sample test accuracy.

2) SVM: Our second baseline method is a support vector machine (SVM) classifier with default parameters. The regularization constant is 1.0, radial basis function (rbf) kernel, and a 'one v. rest' decision function shape. The SVM was also trained and evaluated using both the pre-augmented training data images and the augmented training data images. The SVM also performed better when


(a) An example of a rock image.


(b) An example of a paper image.

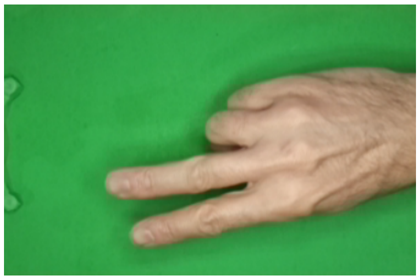

(c) An example of a scissors image.

Fig. 1: Images from data set.

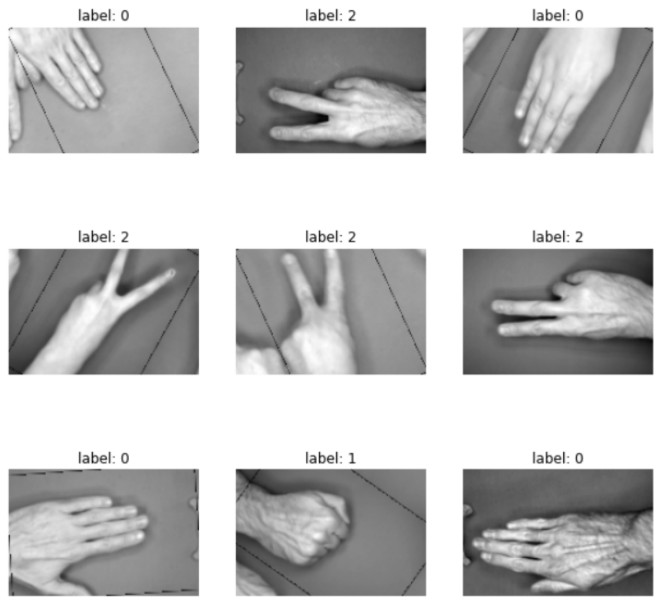


Fig. 2: Images and their respective labels after introducing random rotations by preprocessing.

	Class	Precision	Recall	f1	Test Accuracy
VNN	0	0.68	0.77	0.72	0.71
	1	0.75	0.72	0.73	
	2	0.71	0.65	0.68	
VNN with Augmentation	0	0.64	0.57	0.60	0.62
	1	0.53	0.79	0.63	
	2	0.78	0.50	0.61	
SVM	0	0.68	0.82	0.74	0.73
	1	0.78	0.74	0.76	
	2	0.76	0.65	0.70	
SVM with Augmentation	0	0.58	0.59	0.58	0.59
	1	0.60	0.52	0.56	
	2	0.60	0.67	0.63	

TABLE II: Baseline method results.

using the pre-augmented training data images and achieved a 0.73 out of sample test accuracy. We opted to keep the break-ties option of the classifier set to false in order to keep computational times low. This means that in the instance of a tie, the first class of the reported tied classes gets reported. This could have resulted in a decrease in performance.

Both baseline methods were trained and evaluated on the pre-augmented training images and the augmented training images. A comparison of the results for both methods with each set of training data is shown in Table II. Both methods performed better on the pre-augmented training images and the SVM model performed slightly better than the VNN model. In the results section, we also compare the best VNN and best SVM model to the final CNN model in Table VII.

C. *Convolutional Neural Network*

The baseline models performed well but we knew a convolutional neural network (CNN) would be better at recognizing features in the images and perform better. We set up an initial CNN with two sets of convolutional layers each followed by a max pooling layer, then flattened, and followed by two dense layers. The layers of the CNN are depicted in Figure 3. We chose some initial parameters based off of examples of similar CNN models but intended to tune these parameters to select the optimal parameters. This model with the initial parameters was trained for 20 epochs and evaluated on the data both before we added the data augmentation and after adding the data augmentation. When training the model, the model with the lowest validation loss was kept in each epoch. A comparison of these initial results can be seen in Table III. Before we tuned the parameters, we already saw that the data augmentation did improve the accuracy of the model's predictions from 0.91 to 0.93. Figure 4 shows three examples of images that were predicted incorrectly before the data augmentation was implemented on the training data but predicted correctly after the training data included more rotated images. These three images show a slight rotation from the center line in a way that was not commonly depicted in the training data set before the augmentation was performed. We believe that by including more rotated images in the training data set, the model was able to predict these examples correctly.

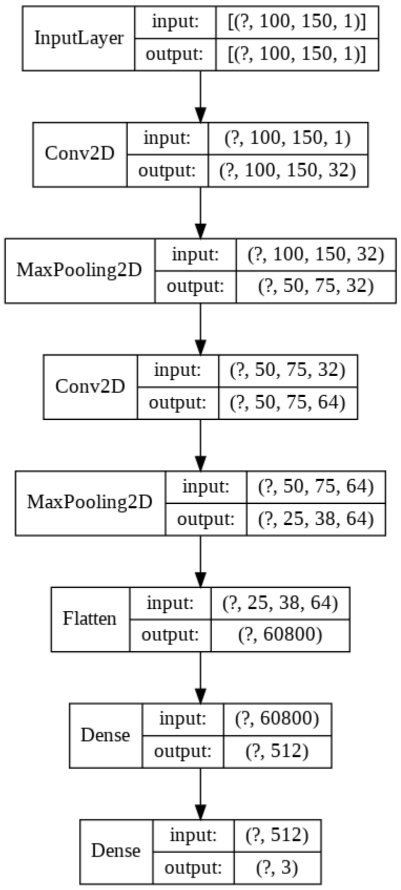


Fig. 3: Layers of the CNN model.

	Class	Precision	Recall	f1	Test Accuracy
No Augmentation	0	0.85	0.90	0.88	0.91
	1	0.93	0.95	0.94	
	2	0.94	0.88	0.91	
Initial Augmentation	0	0.92	0.88	0.90	0.93
	1	0.95	0.97	0.96	
	2	0.93	0.95	0.94	

TABLE III: Comparison of CNN models before and after image augmentation with initial parameters.

D. *Hyperparamater Tuning*

In order to further improve the accuracy of the CNN model, we conducted a hyperparameter tuning experiment. To tune the hyperparameters, we ran a series of trials with different values for each batch size, kernel size, and learning rate. For each experiment, the model was trained for 20 epochs and retained the model with the lowest validation loss in each epoch. After each model was trained, it was also evaluated on testing data. We report the test accuracy for each experiment in Table IV.

E. *Final Model Selection*

To select the optimal hyperparameters, we compared the out of sample test accuracy for each model in the experiment and

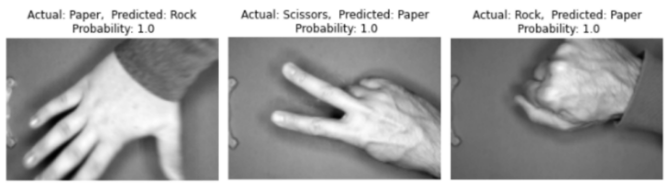

Actual: Paper, Predicted: Rock
Probability: 1.0
Actual: Scissors, Predicted: Paper
Probability: 1.0
Actual: Rock, Predicted: Paper
Probability: 1.0

Fig. 4: Images that were only predicted correctly after rotations performed on training data.

Batch Size	Learning Rate	Kernel Size	Test Accuracy
40	0.0001	3	0.8950
40	0.0001	4	0.9178
40	0.0001	5	0.9612
40	0.001	3	0.7352
40	0.001	4	0.7740
40	0.001	5	0.6027
50	0.0001	3	0.9338
50	0.0001	4	0.9224
50	0.0001	5	0.9452
50	0.001	3	0.7922
50	0.001	4	0.8014
50	0.001	5	0.8470
60	0.0001	3	0.9132
60	0.0001	4	0.8973
60	0.0001	5	0.9269
60	0.001	3	0.7215
60	0.001	4	0.7854
60	0.001	5	0.7511

TABLE IV: Experimental results for hyperparameter tuning.

selected the model with the highest test accuracy. The best model is highlighted in the table and uses a batch size of 40, a learning rate of 0.0001, and a kernel size of 5, resulting in an out of sample test accuracy of 96%. The best model with these parameters was achieved in the 19th epoch of the 20 epochs that the model was trained for. The training and validation loss and accuracy at each epoch is plotted in Figure 5. The best model had a training accuracy of 97.30%, a validation accuracy of 93.36%, and an out of sample test accuracy of 96.12%.

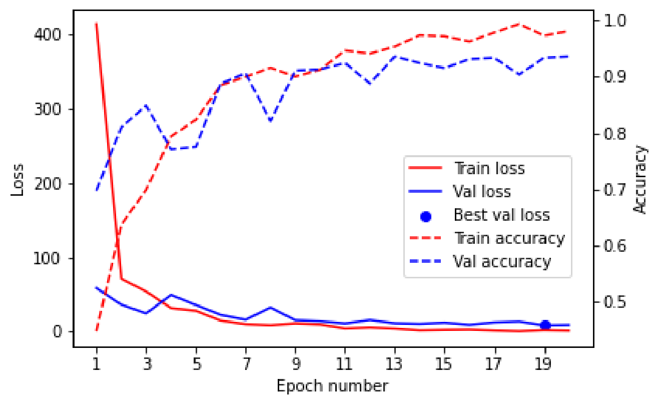


Fig. 5: Training and validation loss and accuracy by epoch.

		Predicted			
		0	1	2	Total
Actual	0	133	6	1	140
	1	6	138	1	145
	2	2	1	150	153
	Total	141	145	152	438

TABLE V: Confusion matrix for final CNN model.

Actual	Predicted	Probability
Paper	Rock	1.000
Paper	Rock	1.000
Paper	Rock	1.000
Paper	Rock	1.000
Paper	Rock	1.000
Paper	Rock	1.000
Paper	Scissors	1.000
Rock	Paper	1.000
Rock	Paper	1.000
Rock	Paper	0.998
Rock	Paper	1.000
Rock	Paper	1.000
Rock	Paper	1.000
Rock	Scissors	1.000
Scissors	Paper	1.000
Scissors	Paper	1.000
Scissors	Rock	1.000

TABLE VI: Breakdown of incorrect predictions.

IV. Results

A. Confusion Matrix

The confusion matrix for the final CNN model is depicted in Table V. Paper and rock were both incorrectly labeled 7 times and scissors were incorrectly labeled 3 times. Table VI lists the 17 misclassifications and what each image was incorrectly labeled as. It is noted that in all but 2 instances, the rocks were misclassified as paper and the papers were misclassified as rock. This indicates that the model particularly struggled with correctly labeling these two hand gestures. The third column of Table VI shows how confident the model was in predicting these incorrect labels. Except for 1 occurrence, every incorrect prediction in the model was 100% confident that it was predicting the correct label. In the only instance that the model was not 100% confident in its incorrect prediction, the model was still over 99% confident, meaning that the model was almost certain in its predictions.

	Class	Precision	Recall	f1	Test Accuracy
VNN	0	0.68	0.77	0.72	0.71
	1	0.75	0.72	0.73	
	2	0.71	0.65	0.68	
SVM	0	0.68	0.82	0.74	0.73
	1	0.78	0.74	0.76	
	2	0.76	0.65	0.70	
Model	0	0.94	0.95	0.95	0.96
	1	0.95	0.95	0.95	
	2	0.99	0.98	0.98	

TABLE VII: Comparison of model with baseline methods.

B. Comparison to Baseline

The results for the final CNN model are depicted in Table VII and compared to the two baseline models. When comparing the metrics of the three models, the CNN model performed the best in classifying the rock, paper, scissor images. This confirms the initial speculation that the CNN model would be the most adequate in recognizing features in image data.

V. Discussion

Our baseline methods both performed worse than our final model, both with and without the introduction of data augmentation. The results from these initial runs can be viewed in Figure II. This gives us reasonable justification to continue our work with the development of a CNN, which are known tools with high performance for tasks of this nature.

Our model performed considerably better after introducing data augmentation. We believe that this was due to the fact that the images in the original data were not all completely aligned or, the images in the training set were more aligned than those in the testing set. Once we experimented with data augmentation and introduced random rotations to the set, we observed an increase in our accuracy for our testing set.

An interesting remark that we found in our observations of the final model deals with the instances in which the model predicted the class label incorrectly. As noted in the Results section above, we list the instances in which the model was wrong and the corresponding probability in which the model made the guess. Remarkably enough, even in instances where the label was misclassified, the model never predicted a class with less that 99%. This could be the result of overfitting to the training set. One approach to counter the effects of this would be to introduce dropout. During our experimental phase, we noticed that the introduction of dropout layers did not improve our accuracy so we opted to remove it from the model before beginning our experiments. Perhaps if we had fine-tuned this layer we could have seen different probabilities in the final predictions as we would have exposed our model to more variability before introducing the testing set.

It is worth noting that the images in the data set were taken in a seemingly controlled environment. That is, all of the images were taken with a green background, the hand was somewhat centrally place in the image and the lighting was consistent across all of the pictures. This lack of variance in the data set could indicate that our model might not perform as well on a random, real-life image data set.

VI. Future Work

We could explore other methods of image augmentation like translations or scaling. Figure 6 shows four images that are incorrectly predicted by the model but we suspect could be predicted better if more image augmentation was performed. The images in the first row are slightly off screen and not centered in the frame like most of the training images and may benefit from random translations on the training data. The images in the second row show different levels of scaling and may benefit from randomly rescaling the training images. It is also worth noting the slightly darker background in the bottom right image which may have contributed to the incorrect prediction.

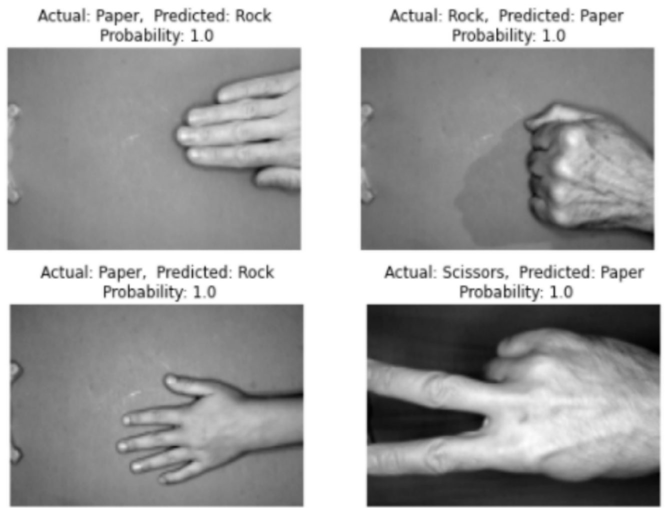

Actual: Paper, Predicted: Rock
Probability: 1.0
Actual: Rock, Predicted: Paper
Probability: 1.0
Actual: Paper, Predicted: Rock
Probability: 1.0
Actual: Scissors, Predicted: Paper
Probability: 1.0

Fig. 6: Images that could benefit from further data augmentation.

References

[1] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[2] François Chollet et al. *Keras*. https://keras.io. 2015.

[3] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[4] T. Guo et al. “Simple convolutional neural network on image classification”. In: *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. 2017, pp. 721–724. DOI: 10.1109/ICBDA.2017.8078730.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[6] Julien de la Bruère-Terreault. URL: https://www.kaggle.com/drgfreeman/rockpaperscissors.