Animal Image Classification

Isaiah Martinez
CSUN
Computer Science Department
isaiah.martinez.891@my.csun.edu

Joycelyn Tuazon
CSUN
Computer Science Department
joycelyn.tuazon.251@my.csun.edu

12/02/2023

Contents

1 Introduction

1.1 Background

In recent years, AI has been an growing topic. Many advancements have been made in several areas within AI including Computer Vision, Natural Language Processing, and other Machine Learning (ML) and Deep Learning (DL) projects [1]. With the release of LLM's such as ChatGPT and Dall-E, the public can now use these tools. However, the general public views AI as a negative advancement in technology. The main belief is that it reduces privacy, but can assist in other areas of life such as finding accurate information online [2].

1.2 Project Description

Our project was designed to show AI can be helpful tool. Additionally, we are beginners in the realm of ML, so this project serves as a step towards real world problem solving. The problem we sought to solve was Image Classification. We employed several ML algorithms to perform Animal Classification for images. Our goal was to learn how to do Image Classification using different methodologies to accurately predict whether an image contained one of the following animals: a cat, dog, or panda.

1.3 Dataset Overview and Preparation

The dataset used contained 3000 RGB images separated into 3 folders, 1 folder per animal [3]. These images ranged varying sizes, ranging less than 100 pixels by 100 pixels to larger than 1000 pixels by 1000 pixels. Each folder contained 1000 images of the depicted animal. Not all images were representative of the subject described and were removed. After cleaning the dataset, we had 2,952 images remaining. Our criteria for removing images were as follows:

1. Multiple interested animal species in the same image. eg. a cat and a dog within the same image.
2. Too small of an image size: less than 100 pixels by 100 pixels.
3. Subject of the image was not easily visible: caused by obscurities, filters, low resolution, etc.
4. The image did not contain the subject animal.

After cleaning the images, we further prepared the dataset to be entered into a readable format for the various models we selected. These included:

- Resizing images to 500 pixels by 500 pixels or 224 pixels by 224 pixels
- Converting images to grayscale
- Converting images to arrays
- Creating new images by augmenting the given images using the following effects: rotation, zoom, horizontal flipping, vertical/horizonal shifting, shear transforming, and pixel filling.

With these changes in mind, the amount of data resulted in: 2,952 images and $(Length * Width * ColorMode) + 1$ Features. Thus for images in RGB sized 500 pixels by 500 pixels, we would have 750,001 features. The reason for implementing these changes to the dataset was to improve performance of the models, as well as speed up the creation of the models.

1.4 Outline

The outline of this paper is as follows: First, we will discuss the research that contributed to our understanding and gave insights into beginning the project. Second, we will describe the algorithms used as well as the techniques employed to achieve our goal of making image classifiers. Afterwards, we will evaluate and compare the performance of our models. Following the evaluations, we will summarize and draw conclusions from the results. Lastly, we will discuss some future works that can be done to explore further into this topic.

2 Related Works

This section outlines the research done to find several algorithms that can be used to construct image classifiers. Each of the following algorithms, except for CLIP, will be explained in more detail within the Methods section.

2.1 SVM

The first algorithm we looked at was Support Vector Machine (SVM) Classifier. Initial Research done showed that SVM outperformed Naive Bayes when perfoming image classification [4]. Through the use of Pipeline and Grid Search, we were able to find optimal performing parameters including: kernel, C, and gamma. Grid Search allowed us to perform K-Fold cross validation on the model so that we would obtain more stable results.

2.2 CNN

Convolutional Neural Networks (CNN) were the main types of algorithms used by others looking into Image Classification. This is because previous studies have shown the advantages to using such models to analyze and predict based on such complex datasets [5]. Additionally, there are various works using CNN models to perform varying types of Image Classification for several animals including primates, rodents, canines, felines, aves, and insecta [6–9]. When constructing our CNN models, we used the ReLu activation function for each hidden layer as well as the Adam optimizer since this seemed to be a common practice to improve performance.

2.3 VGG16

The VGG16 algorithm is available for use as a pre-trained model. It was pre-trained to classify grayscale images of dogs and cats [10]. Upon discovery of how the model was trained, we decided to convert our images to grayscale. VGG16 was pre-trained on a much larger data set than ours, and data augmentation methods were applied to our dataset as an approach to improving performance. This seemed to be a straightforward attempt to increase the amount of data that the VGG16 model would see.

2.4 CLIP

As a last experimental algorithm, we also used a recently developed algorithm called Contrastive Language-Image Pre-training (CLIP) from OpenAI, the creators of prominent AI development technologies [11]. This algorithm is a pre-trained model on images that are encoded, as well as associated text that has also been encoded to match the image (See: Figure 1 below). The advantage of using CLIP was to experiment by using a novel technique called Zero-Shot Prediction (ZSP). ZSP is the notion of encoding an image as well as a list of text that is the target feature, and having the pre-trained model perform a prediction based on no training from the desired dataset. Because of the inability to change parameters and adjust the model, it will be excluded from the comparison to the other techniques.

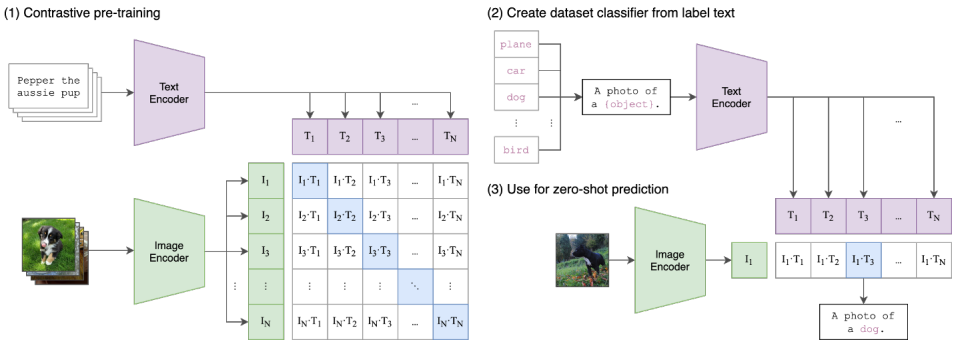


Figure 1: CLIP model summary

2.5 Hypothesis

Between the three animals, we expected to see pandas being the most accurately predicted class since they are always black and white, and have high contrast compared to the surroundings. This is especially true when compared to dogs and cats, which can come in a variety of colored fur. Furthermore, we expected the Neural Network Algorithms as well as the pre-trained algorithms to perform faster and more accurately than SVM.

3 Methods

We implemented the SVM, CNN, ResNet50, and VGG16 classifiers in our project as the algorithms of interest. Below, we will describe the algorithms used in detail.

3.1 SVM

SVM operates by defining a division line between classes using a kernel function. The chosen kernel function, uses parameters C and γ to perform the calculation to separate the dataset into their respective classes [12, 13]. In our case, we converted the images to arrays to allow SVM to process our dataset. Before feeding the dataset, it is imperative to have the dataset standardized. We decided to standardize it using MinMaxScaler, as StandardScaler would change the data to be within the range $[0, 1]$ and took extra time to process, while ultimately yielding no significant improvement in performance of the constructed model. By implementing a Pipeline, we were able to consolidate the code and concisely read the process being done by the program. Also, by using GridSearch, we were able to let the SVM classifier run with a K-Fold comparison added to find the best performing model.

```
pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('classifier', svm.SVC(cache_size = 16000, max_iter = 5000, decision_function_shape = 'ovr', verbose = True))
])

params = {
    'classifier__C' : [5, 1]
}

gridSearch = GridSearchCV(pipeline, params, cv = 5, n_jobs = 1, verbose = 4)

gridSearch.fit(xTrain, yTrain)
```

Figure 2: SVM using Pipeline and Grid Search within the Jupyter Notebook

3.2 CNN

Convolutional Neural Networks (CNN) are commonly used for image classification tasks. Typically, CNN models consist of convolutional layers that apply filters to extract features and patterns, pooling layers to retain the important features and patterns, and fully connected layers to make predictions based on the features learned in the convolutional and pooling layers [14]. Our CNN model implements a sequential structure, where the layers are added linearly. The input layer takes in the parameters of the image, in our case our images were 500 pixels by 500 pixels with a color channel of 1 (since they are greyscale images). After input, the first convolutional layer applies filters to capture the features in the image data by moving a matrix of learnable weights over the image. In our model, the filters had a size of 3x3 and the number of filters increased in each convolutional layer. The weights are randomized before training and begin updating their value throughout the training process. The extracted features are put into a feature map that the following pooling layer uses to further extract important features. Our model uses max-pooling, which looks for the maximum value extracted by each filter from the previous layer so that the initial feature map is downsampled. This pattern of convolutional layer followed by pooling layer iterates to continue extracting features and downsampling the feature map before applying a flatten layer. This reshapes the feature map into a one-dimensional vector that can be inputted into a fully connected dense layer. The convolutional layers and max-pooling layers iterate 3 times in our model before the feature map is flattened. We found that less iterations yielded poorer performance (underfitting) and more iterations resulted in longer training time (model was more complex and computational load was larger). The dense layer is considered a "fully connected" layer because each neuron in a dense layer is connected to every extracted feature and is assigned a weight value. The weight of each of the connections in the dense layer attempt to understand the relationship between the important features extracted from the previous layers as a representation of a prominent feature. The output layer is a dense layer with

neurons equal to the number of classes in the dataset; in our case there are 3 classes (cat, dog, and panda). The summary of the structure of our CNN model is shown in Figure 3 below.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 498, 498, 16)	160
max_pooling2d (MaxPooling2D)	(None, 249, 249, 16)	0
conv2d_1 (Conv2D)	(None, 247, 247, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 123, 123, 32)	0
conv2d_2 (Conv2D)	(None, 121, 121, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 60, 60, 64)	0
dropout (Dropout)	(None, 60, 60, 64)	0
flatten (Flatten)	(None, 230400)	0
dense (Dense)	(None, 128)	29491328
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

Figure 3: CNN model summary.

Our convoluted layers use the Rectified Linear Unit (ReLu) activation function to introduce non-linearity, this way our model can learn complex relationships between the features of our data. The ReLu activation function sets negative values found in the convolution matrix to 0, allowing the model to focus on the positive values that likely represent relevant features [15]. In between the dense layers, we implemented dropout regularization to prevent overfitting. Dropout works by temporarily setting the weights of a random set of neurons to 0 ("dropping out") per epoch [16]. This prevents overfitting by forcing the model to learn more robustly instead of relying too much on the recognition of certain features and "memorizing" the training data. The output dense layer uses the Softmax activation function to scale the numerical output of the model into probabilities distributed over the possible output classes represented by indexes. The class with the highest probability is considered the predicted class which is mapped to the string value of the class using the index value.

While fitting the model, we implemented the EarlyStopping callback to reduce the likelihood of overfitting by stopping training if the validation accuracy did not improve after 5 consecutive epochs. The EarlyStopping callback can also be useful in preventing underfitting if the number of epochs is set to a higher number than what is expected to be necessary.

3.3 ResNet50

Residual Network 50 (ResNet50) is a type of CNN algorithm that contains 50 hidden layers. This algorithm allows for the use of Convolution2D and MaxPooling2D layers just like the standard CNN algorithm. The 'Residual' portion refers to how the gradient is handled within the structure of the algorithm itself. The gradient is the result calculated from the partial derivatives of the loss function, with respect to the given data's parameters [17]. Instead of using standard back-propogation, it has a separate structure called a residual block which allows for the gradient to flow more easily through the entire structured network [18].

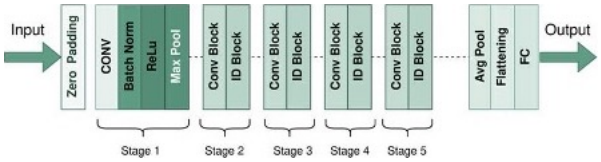
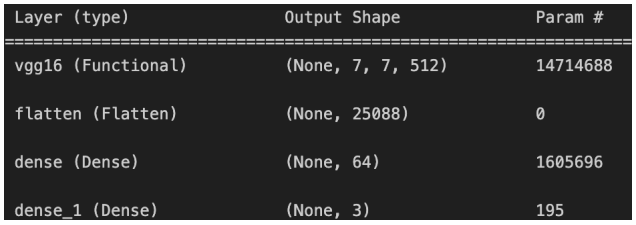


Figure 4: A sample structure of the Res Net 50 Algorithm

3.4 VGG16

VGG16 is a pre-trained CNN model available for use through Tensorflow. VGG16 was pre-trained for image classification on large datasets such as ImageNet where the input image size was 224 pixels by 224 pixels with 3 color channels (RGB). Its structure consists of 13 convoluted layers and 3 fully connected dense layers (16 layers total) [19]. We decided to use a pre-trained VGG16 model instead of building one from scratch to learn how efficiently it would execute our particular image classification task. To utilize the pre-trained VGG16 model, we first load the base model without the top layers as these layers are what we need to modify to have the model classify our cat, dog, and panda images. Initially, the custom layers we added included 3 fully connected dense layers. However, we found the training time to be especially long and after experimenting with removing a dense layer to make the model less complex, we found that the training time was shortened with very little change in performance. Ultimately, we chose to keep working with the shortened training time with two dense layers. The first dense layer utilizes ReLu activation as well as the He uniform kernel initializer to initialize the weights of the 64 neurons within a uniform range dependent on the number of input neurons. The He uniform combats the possibility of too many dead neurons by increasing the likelihood that neurons will activate and not be stuck in an inactive state throughout the entirety of training. This gives more neurons the chance to contribute to learning. The structure of the pre-trained VGG16 is shown in Figure 5 below. The last output dense layer acts the same as it did in our previous CNN model, utilizing Softmax to generate probabilities distributed over the three output classes mapped to their corresponding string values.



Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 64)	1605696
dense_1 (Dense)	(None, 3)	195

Figure 5: Pre-trained VGG16 model summary. The base model is loaded and the top dense layers are added manually.

4 Evaluation

4.1 Confusion Matrix

When evaluating the performance of our models, we analyzed the relationships between the statistical measures when analyzing the first class. These included:

- TP = true positive instances
- TN = true negative instances
- FP = false positive instances
- FN = false negative instances.

Together, they can be used to construct a confusion matrix. The confusion matrix allows for researchers to evaluate the models performance per class. In our case we had three columns and three rows, with the columns representing the predicted class while the rows represented the actual class of the image. Figure 6 demonstrates the value of the elements of a multi-class confusion matrix from the perspective of the first class. Using the aforementioned metrics and confusion matrix, we were able to obtain the following metrics which were used to compare the performance of the models created.

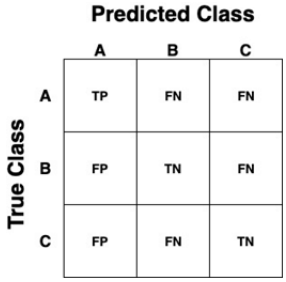


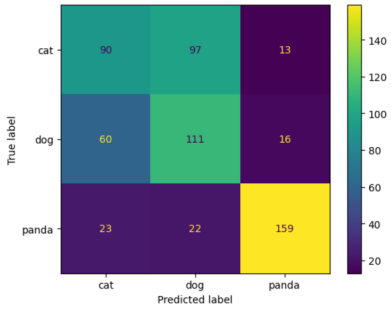
Figure 6: Multi-Class Confusion Matrix

4.2 Performance Metrics

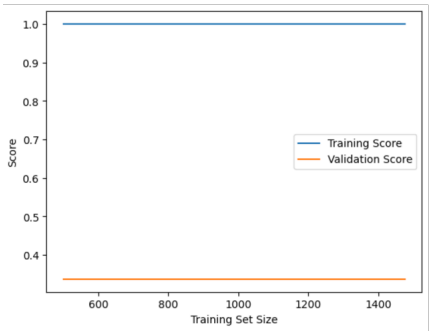
- Test accuracy
  - Measures how well the model makes correct predictions on data it has not seen before in training. $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision score
  - Measures how many positive predictions are true positives. $\frac{TP}{TP+FP}$
- Recall score
  - Measures how well the model predicts all positive instances. $\frac{TP}{TP+FN}$
- F1 score
  - Measure the mean between precision and recall. $\frac{2*Precision*Recall}{Precision+Recall}$
  - This is especially useful since our dataset has a different amount of instances of each class.

4.3 SVM

SVM took a long time to run when looking at all the data, resulting in several days being taken to compute a single model. By using Grid Search to find the best hyperparameter values, we were able to compute the following Confusion Matrix seen in Figure 7a. However, this model was severely overfitting as seen in Figure 7b. It was decided that instead of trying to work with SVM which would take a long time to compute the model, we would focus more on the various CNN algorithms. The best hyperparameters found to compute the following results were $C = 10$ and $\gamma = \frac{1}{250{,}001}$.



(a) Confusion Matrix of SVM



(b) Loss function vs Training Size(Epochs) for SVM

4.4 CNN

The first few training runs were done on a simpler CNN model with less convoluted and max-pooling layers. While run time was fairly quick, performance was low as the model was too simple. We experimented with adding more convoluted and max-pooling layers and found that adding too many

made training time too long considering that performance did not improve much. The final structure is what we found to be the most efficient training-time wise for our project while also considering performance. Most runs averaged between 50-60% accuracy before applying data augmentation and improved to 60-70% after applying data augmentation. Our final run reached about 76% test accuracy with a 76% precision score, 76% recall score, and a 75% f1 score. The confusion matrix for our final CNN run is shown in Figure 8 below. The model consistently had the most difficulty classifying dogs and the most success classifying pandas.

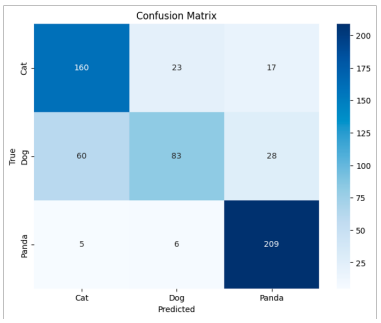
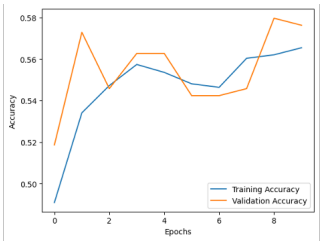


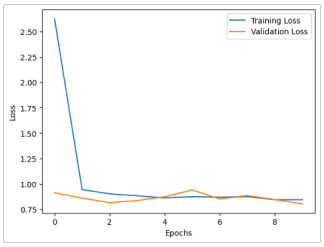
Figure 8: Confusion matrix for final CNN run

4.5 ResNet50

There were several models that were made in order to get better results. The structure of the networks used differing amounts of hidden layers, dropout, Conv2D, and MaxPooling2D layers in order to find the best performing model. Ultimately, it was reduced to modifying 2 structures and having a basic model that would be the baseline to compare to. We found that the best perfoming model was a simple structure with a single hidden layer with a dropout of 20%, i.e. the baseline. This model yielded an accuracy of 61%, and showed strong signs of learning without overfitting nor underfitting seen in Figures 9a and 9b.



(a) Res Net 50: Accuracy vs Epochs



(b) Res Net 50: Loss vs Epochs

4.6 VGG16

The first training run was done with 3 dense layers manually added on top of the pre-trained VGG16 model and took approximately 14 hours to run through 15 epochs (there were 25 total epochs but the EarlyStopping callback was implemented which stopped training after 15 epochs). This first run yielded about 50% accuracy and after experimenting with different hyperparameter values, it was apparent that the long run time could be shortened by removing a dense layer to decrease the complexity of the model without sacrificing performance too much. This is most likely because our dataset is small compared to what the VGG16 model was pre-trained on, and so an increase in complexity did not increase performance as the model struggled learn stably. Reducing the number of dense layers to 2 decreased training time to approximately 18 minutes per epoch (as opposed to 1 hour and 20 minutes per epoch before removing the third dense layer). Most runs stayed between 50-60% with and without data augmentation. Our final run scored about 55% on test accuracy, 58% on precision, 55% on recall, and 49% on f1. The confusion matrix for our final VGG16 run is shown in Figure10 below, where a significant number of false predictions can be observed.

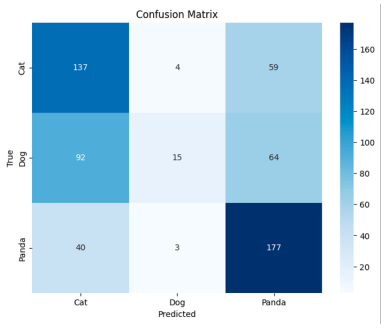


Figure 10: Confusion matrix for final VGG16 run.

5 Conclusion

We found that the best performing algorithm was CNN with a 76% accuracy and the least amount of under/overfitting. Although evaluation of the CNN loss functions suggests that the learning process is slightly unstable, there were still signs of convergence and the overall metrics indicated good performance. Throughout this project we were able to learn about the different approaches to image classification and can understand why CNN is a popular choice for the task. The architecture of CNN models allow for efficient handling of image classification aspects as the layers within the architecture capture and learn patterns in images. Since image classification involves the recognition of abstract and hierarchical features, CNN out performs SVM as SVM tends to perform better alongside intricate feature engineering and lacks the ability to learn patterns over a spatial hierarchy. SVM is also known to struggle with large datasets, performing inefficiently and experiencing futile and long training times. It is interesting to find that CNN seemed to have the opposite problem, finding our dataset to be on the smaller side and performing better when more variance through data augmentation was applied. The performance of our pre-trained VGG16 model also indicated potential performance improvement on a larger dataset where a lengthy training time would be more efficient than SVM.

6 Future Work

Some improvements that can be done on our project would be to:

- Increase dataset size and diversity of images
  - A larger dataset may allow CNN models to achieve learning stability, increasing overall performance
  - This would particularly be useful while applying the VGG16 model, as it is more computationally expansive and was pretrained on larger datasets
- For CNN models built from scratch, more layers, filters, and neurons may need to be added to increase complexity, ensuring that the intricate features and patterns of the images were captured
- Stricter set of criteria for clean images
- Additional augmentations of images
- For SVM, use a smaller subset of the data to perform training faster

References

[1] Stanford: What are the most important Advances in AI?

[2] PRC: Growing public concern about the role of artificial intelligence in daily life

[3] Kaggle: Animal Image Dataset(DOG, CAT and PANDA)

[4] *Comparison of Support Vector Machine Classifier and NaÃ¯ve Bayes Classifier on Road Surface Type Classification* by Marianingsih and Utaminingrum.

[5] *How Does the Data set Affect CNN-based Image Classification Performance?* by Luo, Li, Wang, et. al.

[6] *An Enhanced Animal Species Classification and Prediction Engine using CNN* by Priya, Kalyan, et. al.

[7] *Animal Species Image Classification* by Prudhivi, Krishna et. al.

[8] *Animal Breed Classification and Prediction Using Convolutional Neural Network Primates as a Case Study* by Kamepalli, Kolli, and Bandaru

[9] *CNN Architectures Performance Evaluation for Image Classification of Mosquito in Indonesia* by Amiruddin and Kadir

[10] *An Approach to Run Pre-Trained Deep Learning Models on Grayscale Images* by Ahmad and Shin

[11] CLIP algorithm Github Page

[12] SVM explained from IBM.

[13] Article explaining SVM

[14] *A Review of Convolutional Neural Networks, its Variants and Applications* by Shruti and Rekha

[15] ReLu Exaplined

[16] Dropout explained from Keras

[17] Gradient Explained

[18] Res Net 50 Explained

[19] VGG16 Explained