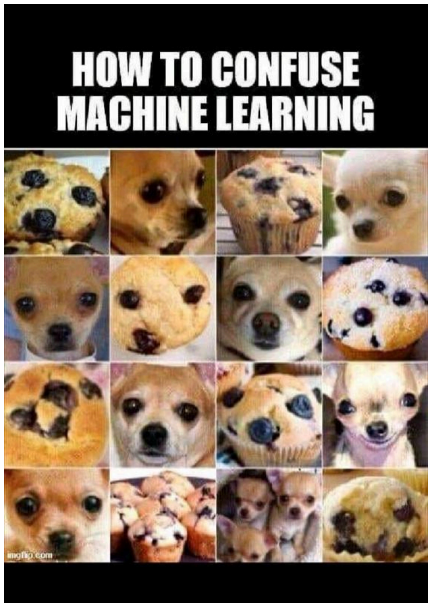
BYOD

Inspired by our professor's example we decided to choose the muffin vs chihuahua dataset. The popular meme shows a sample of very similar looking chihuahuas and muffins stating it would confuse machine learning. We decided to debunk this. Our classifiers will help people everywhere finally decide is that a muffin or chihuahua. We do not want people accidentally eating their pets for breakfast. Some challenges that will likely arise with this dataset is the broad range of inputs. Many of the images come in different aspect ratios and resolutions, so we will need to preprocess all images. Some of the *'chihuahuas'* look like those in the pictures below while others may be different colors, abstract drawings, cartoons or different breeds altogether.

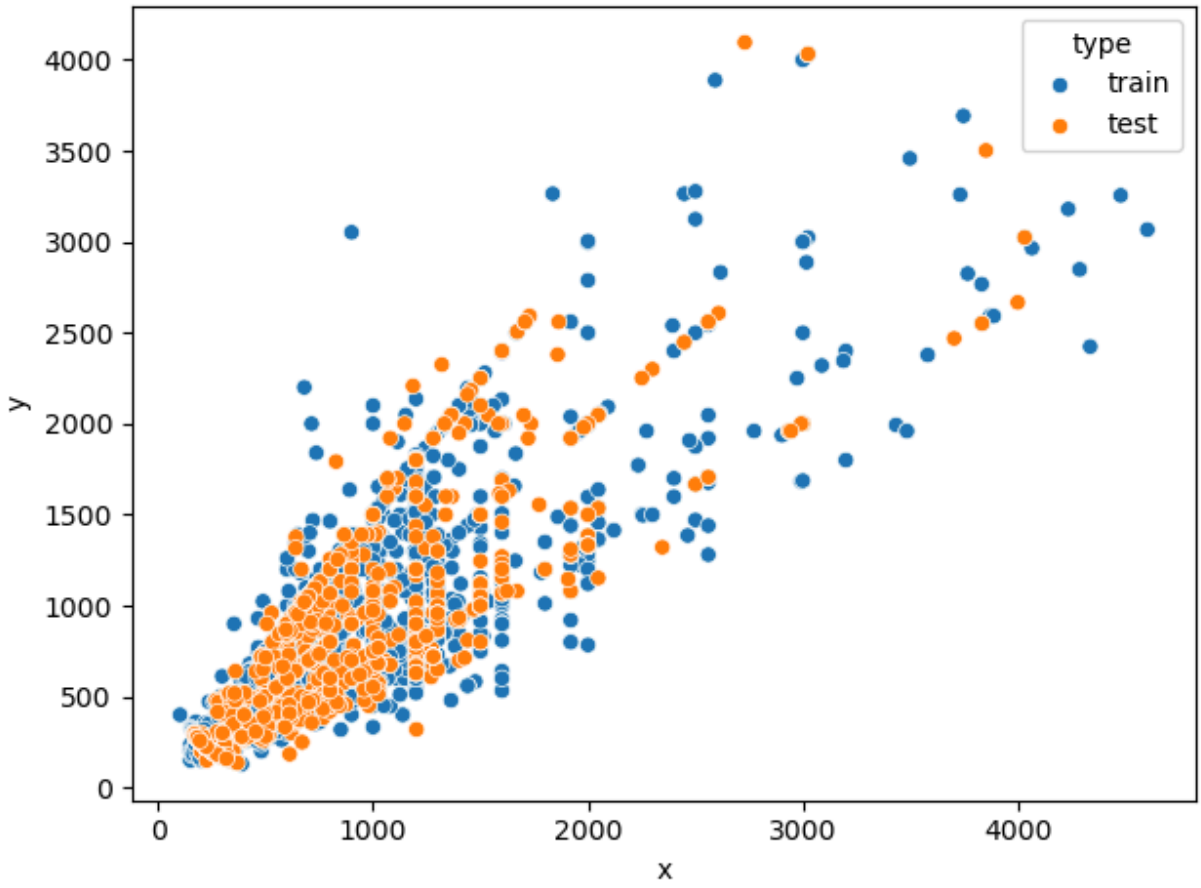


EDA

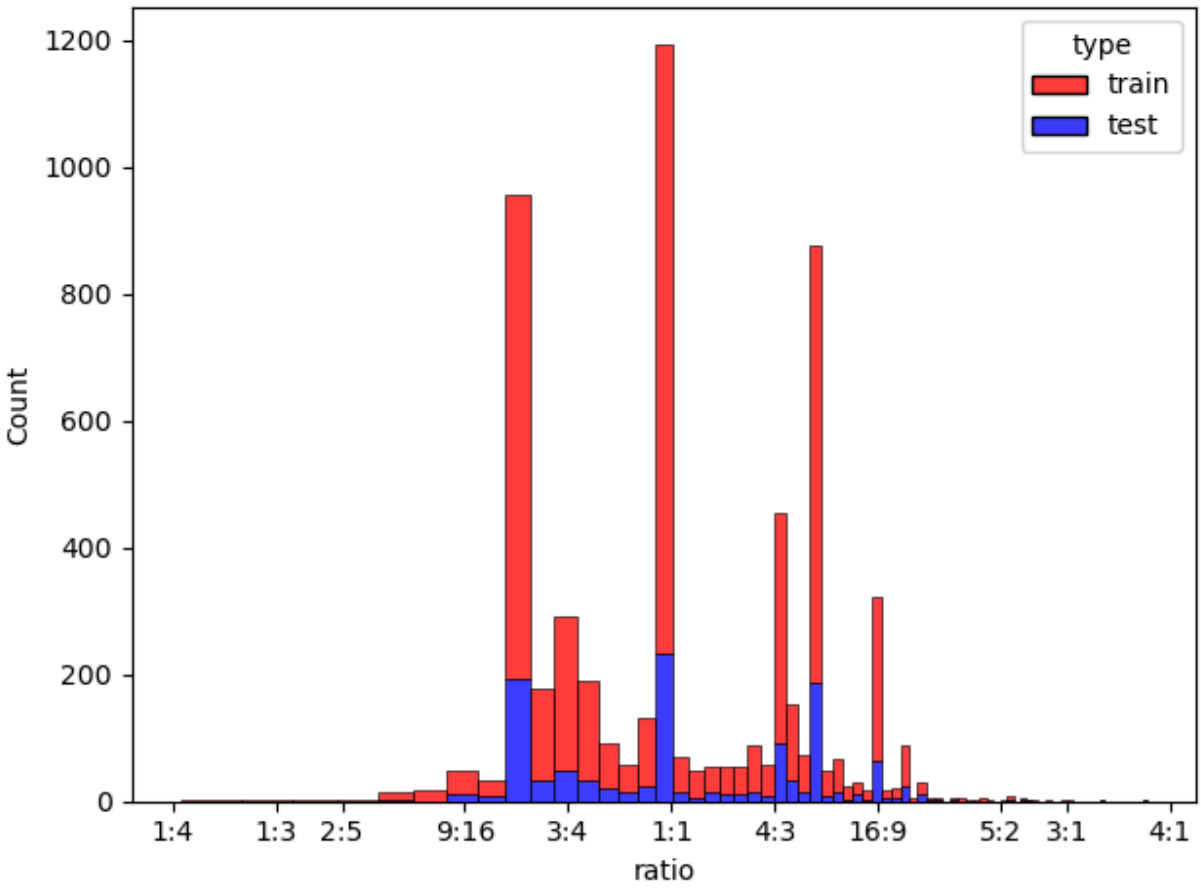
This dataset already had images split into train and test sets. We kept this separation. There are 4733 train images and 1184 test images. This means that 79.990% of images are train images. There are 3199 images labeled Chihuahua and 2718 labeled Muffin, for a 54.065% split of Chihuahua. Here is a table that breaks down the image counts by type and label.

Type	Label	Count	Percentage
Train	Chihuahua	2559	43.248%
Train	Muffin	2174	36.742%
Test	Chihuahua	640	10.816%
Test	Muffin	544	9.194%

Here is a graph of the sizes of images. Many are on the smaller end but some have upwards of 4000 pixels across.



Here is a graph of the images' aspect ratio. The x scale is logarithmic and many common aspect ratios are displayed below. Many of the images are square. Some seem to have the aspect ratios that phone cameras take pictures in. Some have ratios that other cameras like DSLRs take pictures in. Many have weird aspect ratios. The left side has images that are very tall while the right side has the very wide ones.



Problem Definition

We are going to solve the age old question of is that my dog or my breakfast. Our classifiers will take in an image and decide whether it is a chihuahua or muffin. Our EDA has helped us find the most common aspect ratios and image sizes. We decided to use this information to normalize our dataset in preprocessing each image.

Preprocessing

The script first downloads the zip file with all of the images from kaggle and unzips them to the local directory.

Many of the images are very big with weird aspect ratios and so the first step we did is normalize them to 256 by 256 pixels with rgb channels. For non square images, this was done by cropping the largest possible square box from the middle of the image. This box was then scaled to 256 by 256 pixels. And was then converted, if needed, to rgb. These files were saved separately.
For the decision tree, we wanted grayscale images. So we first converted our normalized images to grayscale, flattened them, and then converted them to equal width bins of 4 for each pixel. This left us with an array of length 65536 of the set {0,1,2,3}.
For the other classifiers, we used the full color normalized images, which each pixel channel normalized to the range of 0 to 1.
For models that wanted a validation set, we cut the test set in half, with one section being used solely for final model evaluation, and the other half for validation.

Model Selection, Training, and Optimization

The 5 models that we chose were a KNN, a Naive Bayes Classifier, a XGBoost decision tree, a CNN, and a DenseNet. We also made a DummyClassifier for a baseline. For the KNN, we tried 3NN, 5NN, 7NN, 9NN, and 11NN. 5NN did the best but not by much. For the CNN we took all the training images, copied them 4 times, and rotated them by variations of 90deg. We did this to enhance the dataset by generating more images. It did better, but not by much. The first Naive Bayes Classifier we used was Gaussian, though we later tried a Complement Naive Bayes Classifier which did considerably better.

Model Evaluation

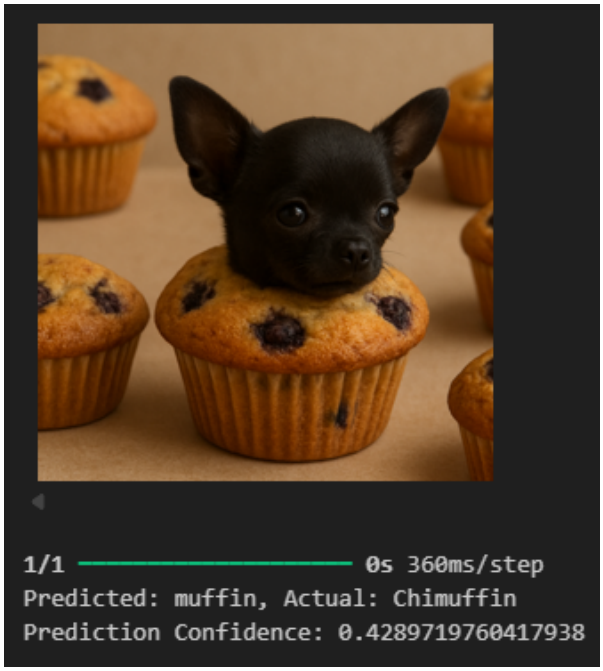
KNN	Accuracy	Precision	Recall	F1
3NN	57.69%	56.32%	96.72%	71.19%
5NN	**57.85%**	**56.31%**	**98.28%**	**71.60%**
7NN	57.43%	56.06%	98.28%	71.40%
9NN	56.33%	55.43%	98.12%	70.84%
11NN	56.25%	55.38%	98.12%	70.80%

5NN did the best but was not that far ahead of the Dummy Classifier. It seems that the images in our dataset are two different from one another for KNN to be useful.

Model	Accuracy	Precision	Recall	F1
DummyClassifier	54.05%	54.05%	100%	70.18%
5NN	57.85%	56.31%	98.28%	71.60%
Gaussian Naive Bayes	59.71%	67.91%	48.28%	56.44%
Complement Naive Bayes	69.68%	73.00%	69.69%	71.30%
XGBoost	80.91%	83.12%	80.76%	81.92%
CNN	92.40%	90.96%	95.27%	93.07%
CNN with Rotation	94.76%	93.33%	97.16%	95.21%
DenseNet	**99.49%**	**100.00%**	**99.05%**	**99.52%**

Conclusion

The meme is debunked! It is very possible for machine learning to classify chihuahuas vs muffins. The images for each model were preprocessed identically by normalizing the pixel values and resizing to 256x256. The formatted images are then passed through one of our models. We made eight different classifiers. The first is a dummy classifier, it always answers chihuahua. Next we used the K nearest neighbor method, we found that 5NN had the highest accuracy, but this is likely due to coincidence since it is not much better than the dummy classifier. The images were not similar enough to be useful. The complete naive bayes worked much better than the gaussian naive bayes. This is because the pixel intensities do not follow a gaussian distribution, so the classifier is not right to use for this data. While unorthodox for image data, the XGBoost tree method performed very well scoring the best outside of the machine learning methods. The first machine learning model is a basic CNN, it scores very well, but is a bit overfit to the training data. Next is the CNN trained with rotation, it scored slightly better, but is still prone to overfitting. Last is the DenseNet, we took tensorflow's pretrained DenseNet121 model, a chain of interconnected CNNs pretrained on a massive dataset for feature extraction. We took this model and froze all weights except the last block which is the last 16 layers of the model. This model is extremely accurate and does not suffer from overfitting. To check this I gave the CNN with rotation and the fine tuned DenseNet an image outside of the dataset, a chimuffin, an image that is half chihuahua and half muffin made by chat gpt. This image in theory when given to a perfect model should score ~50% on the chihuahua - muffin scale.

CNN with rotation	DenseNet
	

The CNN with rotation is very confident that this is an image of a chihuahua, likely due to overfitting instead of complete image understanding. The DenseNet is able to decide that it has both features of chihuahuas and muffins present, and then makes a decision based on the feature information.