Abstract

We use three types of neural networks (standard, convolutional, and recurrent) to classify bird calls from the surrounding Seattle area. We will create neural networks to complete three tasks: (1) perform binary classification between two bird call classes, (2) perform multiclass classification between twelve different bird call classes, (3) classification of three .mp3 sound files. Our models, in general, performed slightly better than random guessing, but later discussion will show some of the possible improvements to the models, namely decreasing the learning rate.

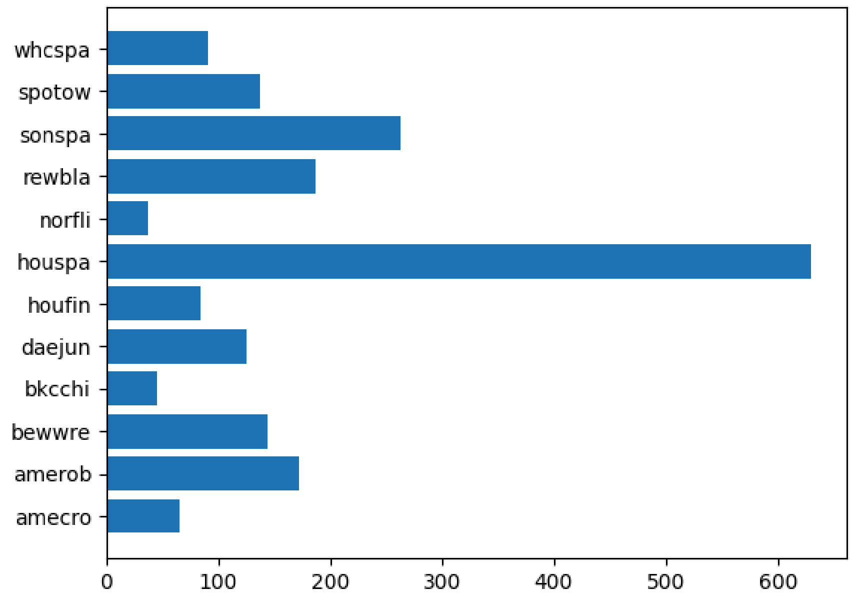
Introduction

The website, xeno-canto, is dedicated to extracting sounds from wildlife, many of which are stored as .mp3 files. We will be drawing data on bird calls from the Seattle area and use some of those samples to train a neural network for classification. What makes the data particularly difficult to work with is its unstructured nature. Namely, the sound clips of the birds are in an audio format, as opposed to a nice tabular structure that something like a .csv file would have.

We have 12 bird species in total.

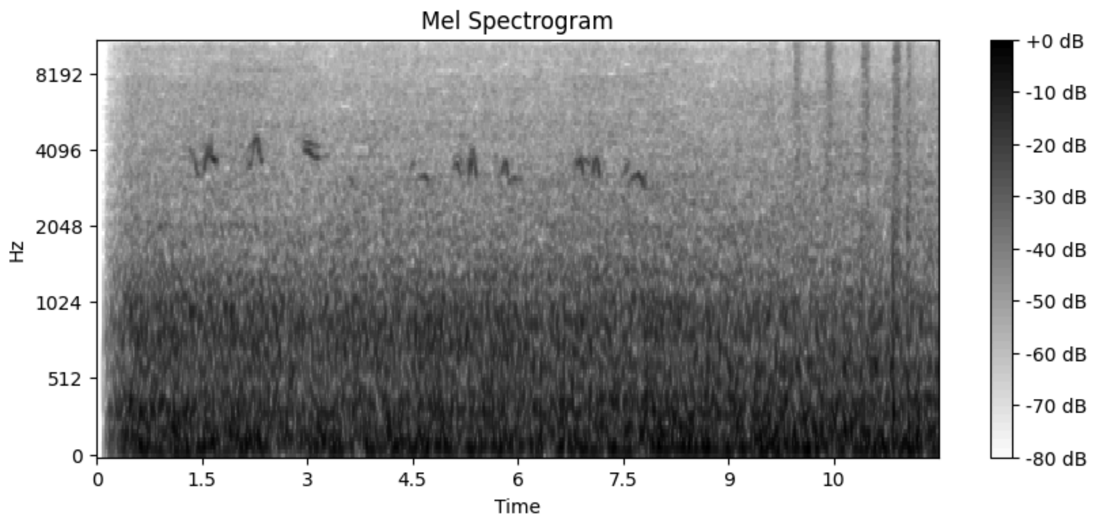
Variable Name	Bird Species
amecro	American Crow
amerob	American Robin
bewwre	Bewick’s Wren
bkcchi	Black-capped Chickadee
daejun	Dark-eyed Junco
houfun	House Finch
houspa	House Sparrow
norfli	Northern Flicker
rewbla	Red-winged Blackbird
sonspa	Song Sparrow
spotow	Spotted Towhee
whcspa	White-crowned Sparrow

We have a reasonable amount of data for each of these birds, ranging from 37 to 630.



We notice that house sparrows are rather common whereas the northern flicker is rather uncommon. This imbalance in the data may prove to be troublesome of our classification efforts and will be discussed later in the methodology section.

While xeno-canto records the sounds in the form of an .mp3 file, we will not be using those specific files. Instead, we will be making use of a spectrogram, namely a mel-spectrogram. A spectrogram is a visual representation of the frequency of the sound over time. We can therefore understand a spectrogram object as being analogous to a picture or a two dimensional graph, with time on the x-axis and frequency on the y-axis. The librosa package allows the following visualization.



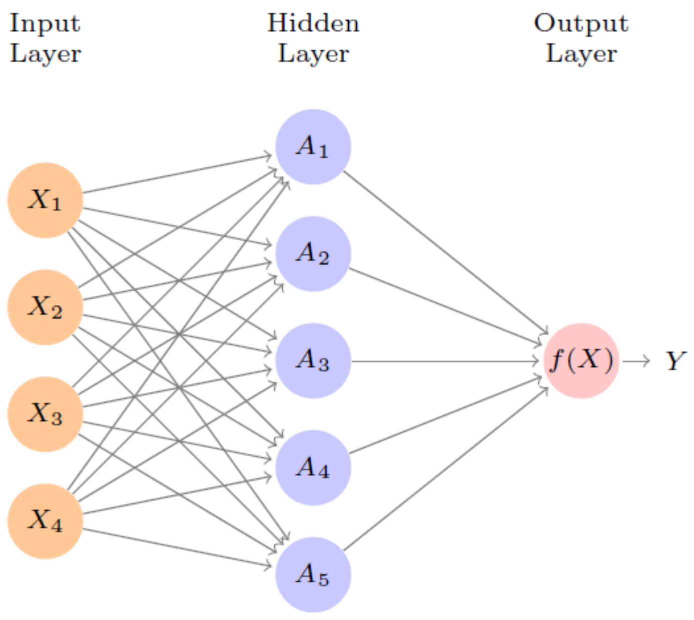
We see Hz being plotted along with the time. Then, given a time and frequency, the plot gives us the decibels, or intensity of the sound. However, do note that the data has already been scaled by the librosa package, so the time is just a 3-second snippet. The process of converting the sound file into a spectrogram involves the following:

1. The clip is split into 3-second increments.
2. The clip is then filtered to take frequencies from 0 Hz to 22050 Hz.
3. A spectrogram is made for each 3-second increment.

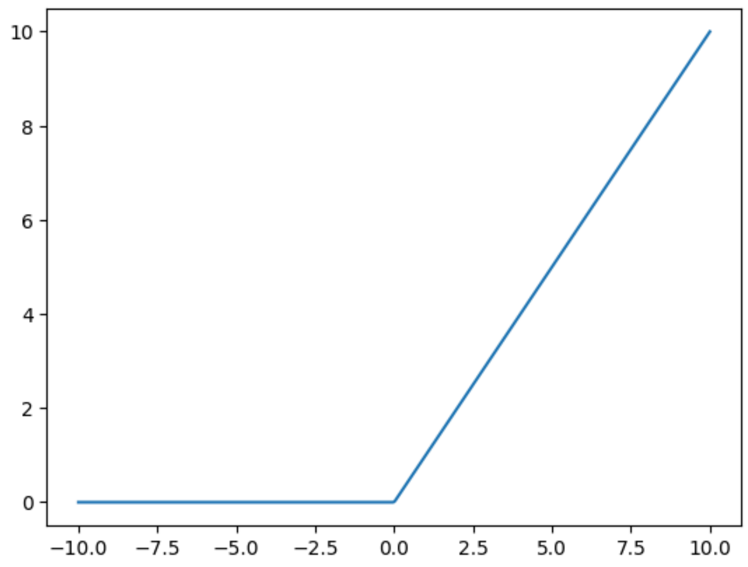
Of note is that because the neural network is being trained on 2-second increment spectrograms, the models will only take in 3-second increments. This will affect how our model is being run. Namely, anyone wanting to make a prediction of a clip longer than 3 seconds will need to break up the data into 3-second chunks.

Theoretical Background

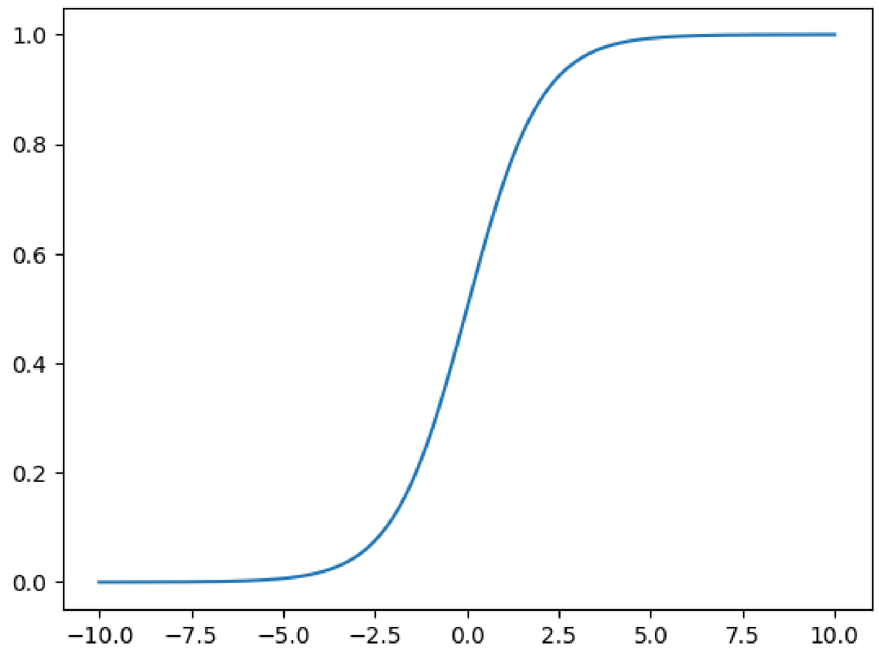
We will be making use of neural networks, which are a type of model that uses a combination of activation functions to model the data. As James et al. describe, a neural network takes several inputs, applies non-linear transformations to them, and then aggregates the inputs by taking a linear combination of them for our output. I use their illustration below in their 2020 text (405).



Let's consider the above neural network as an example. We see that our model takes in four inputs, each of the X terms. These could represent different samples we are feeding to the model. These inputs are then sent to hidden layers, containing activation functions. An activation function is just a function that adds some type of non-linearity to the data. This non-linearity is what allows a neural network to potentially perform better than a a linear model such as linear regression. Examples of such activation functions are the ReLu function and the sigmoid function. The ReLu function takes in a number and outputs either that number, if it is greater than 0, or 0, if that number is less than or equal to zero. This is good at introducing non-linearity to a function by potentially breaking it to many piecewise functions.



The sigmoid function, on the other hand, takes a number and sends it to a number that is extremely close to 1 or 0, essentially converting a numerical result into a categorical result.



A linear combination of these activation function helps create a complex and flexible model.

Classification and Encoding

It seems that a linear combination, however, would intuitively be numeric. While such a result might not be problematic for binary classification, since we can use the sigmoid function to help our network predict between our two classes with 1s and 0s, multiclass classification seems a little more difficult. Recall our neural network only takes in numerical values, namely vectors, and therefore does not have an immediate ability to distinguish between numerical and categorical data. Instead, we will make use of one-hot encoding, which allows us to convert an object like a string into a vector of numbers. This converts our output data, namely bird titles, into numerical code that we can later decode to acquire our answer.

Feed Forward Neural Networks:

A feed forward neural network, or what I will call a standard neural network, has already been described as above. However, there are several tools and parameters worth describing here.

Epoch:

An epoch can be described as a cycle of training. What a neural network does, essentially, is perform training cycles multiple times, attempting to decrease the loss function(accuracy in our case). After each epoch, the weights and biases of our model are adjusted to better minimize the loss function[1]. Intuitively, one might think of additional epochs as being strictly better. However, there are some drawbacks to additional epochs. First is the computational cost. It takes either time or computational resources to train for longer periods of time. Second, however, is the risk of overfitting. While a neural network does make use of a type of cross validation when training its model, if the data are sufficiently small enough, the model might end up fitting for that noise even if it hasn't seen it in the training set. However, we can control overfitting by lowering the learning rate.

Learning Rate:

As mentioned with epochs, our model is constantly updating our weights and biases in each layer. However, one potential issue is that our model might overcompensate, altering our weights and biases too much and not decreasing the loss. However, we can prevent this. In the training phase, we can set a smaller learning rate. The learning rate could be understood as the step size that the model takes when it updates its parameters (the weights and biases). By lowering the step size, we can prevent our model from making changes that are too drastic, therefore preventing our model values from oscillating. We will see lots of this oscillation in the results.

Layer number:

The image given above describes a neural network with only one hidden layer. However, given that the advantage of a neural network is that they have access to lots of complexity and non-linearity, it would be reasonable to try to increase the number of layers. The addition of more non-linear layers would also add more parameters (in the form of additional weights and

[1] The weights and biases mentioned here could be understood as the slope and y-intercept, respectively, of our linear combination when we take the activation function. The activation function is non-linear, but the linear transformation we apply to our inputs between each layer involves addition and multiplication by scalars and vectors of numbers.

biases) for our model to use. However, as with epochs, we have an issue with limited computational resources as more parameters would use up more of our memory in training.

Dropout Layers

One layer that may be useful for our models will be a dropout layer. Recall that neural networks, in comparison to linear models, are flexible. With this flexibility comes the risk of overfitting, where our data fits to the noise but not the more general structure of the data. On data that is sufficiently rich and large enough, we won't have to worry too much. However, there are still risks to overfitting. We can make use of an additional layer to help us, namely a dropout layer. A dropout layer, as the name implies, "drops out" some data on each iteration, or epoch.

Convolutional Neural Networks:

Let's start by describing one specialized neural network, convolutional neural networks. This is simply a neural network that involves changing, or *convolving*, our input objects. By a convolution, we are describing a filter, or layer, applied to our model that slightly alters or shifts the image. What this is meant to do is to, as James et al. describe, to highlight local features of different objects (414). This is because objects sent through a convolutional layer will still keep their local features. For example, a picture of a tiger sent through many features will still retain many important features, such as its teeth, nose, etc. Afterwards, our neural network condenses our object by making use of a pooling layer, which shrinks our object. This will likely eliminate noise that doesn't contribute a feature to the image. Convolutional neural networks, therefore, help us average out lots of noise by emphasizing local features.

Let's describe how convolutional neural networks can help with our purposes. The example given above was of an image of a tiger. It seems that convolutional neural networks can be useful when classifying images. However, what we have are spectrograms, which aren't exactly analogous to pictures. However, a graph could be understood as a picture and can serve as an image for the purposes of our classification task because bird calls would likely have some level of structure. Therefore, a convolutional neural network would be able to plausibly capture some of these local features.

Recursive Neural Networks

A recursive neural network is another specialization of a neural network that helps us particularly with data arranged chronologically. This neural network makes use of previous features and accommodates for the temporal nature of the data. By arranging the features timewise, our neural network can be "aware", in a sense, that the objects are sequential, in a similar way to how the words in a sentence are sequenced.

One particular benefit, though potentially a drawback, for a recurrent neural network, is weight sharing. Weight sharing is when the same object, but across different time slices, will have the same weights and biases as an object from earlier time slices. This is beneficial insofar as it lets the model train while keeping in mind that time should not drastically change the classification of an object. For example, a bird call at the start of a sound clip and a bird call at the end of a sound clip should plausibly be categorized as the same bird call[2].

Our bird calls are in a sequence (time), so a recurrent neural network seems like a promising candidate for our model.

Methodology

Data Imbalancing

Recall earlier that our data was particularly unbalanced, because of the large number of house sparrows. We should keep in mind that, if we did use all of our data for training, that a neural network would likely have a potentially 50% accuracy rate classifying house sparrows, just because about half of the data was house sparrows. This is known as data imbalancing.

A straightforward, though rather coarse, solution to imbalanced data is to sample only as much data as the minimum amount of data points across all classes. In other words, since our Northern Flicker has the least samples with 37, we will just take 37 samples per class to prevent imbalancing the data. This just has the small issue that our data will be of a slightly lower quality.

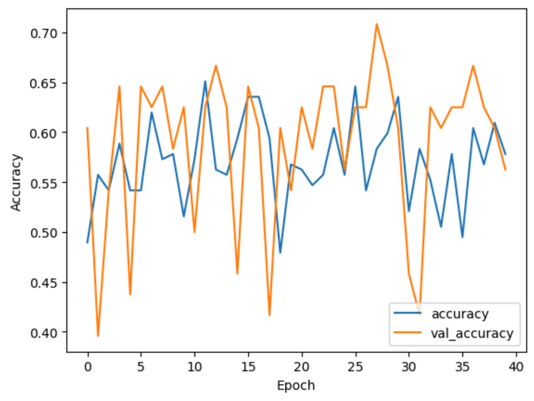
Data Re-shaping

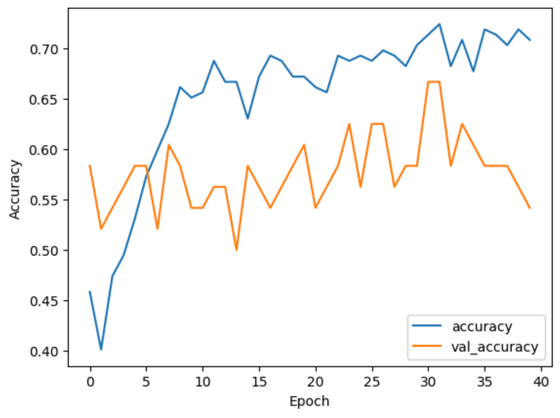
While we first described neural networks, it may have seemed that convolutional and recurrent neural networks were just additional layers. We should note that our data needs reshaped depending on which type of neural network we are using. Each of our objects is a two dimensional matrix, like a piece of paper where each of the spaces on the paper are values for the decibels. Let's discuss how this should be reshaped. In particular, a feed forward neural network should *flatten* our two dimensional matrix into a 1 dimensional vector. This might lead to some problems since it erases much of the structure of the matrix. For convolutional and recurrent neural networks, we should keep it roughly the same, but add an additional dimension to the matrix to create a third dimension. However, this third dimension does not drastically change the structure of our matrix but is just for the neural network to read the input matrix.

Learning rate fine tuning

[2] Shared weights might be an issue for objects that are significantly affected by time, namely stock market prices over decades. It's plausible that our weights would in fact be changed over time.

We mentioned above that a high learning rate can lead to a great amount of oscillation in the data. A prime example is the following graph of binary classification accuracy of one of our models. The left hand plot has a high learning rate while the right hand plot has a low learning rate. We see that while both plots have validation accuracies that drastically change, the model with a lower learning rate has experienced much smaller oscillations compared to the model with higher learning rate. We should try to fine tune our learning rate such that the variance in the accuracy decreases. It is true that a lower learning rate could lead to a potentially higher validation accuracy, it seems that our model would be very inconsistent and its parameters would not converge. We make use of Keras' RMS prop's learning rate variable to lower our learning rate. Combined with more epochs, our model could potentially converge.





Results

Let's discuss our results for each of the tasks. Let's first look at our binary classification. For our binary classification task, we decided to classify the American Robin and Red-winged Blackbird. This was because, for these two datasets, we did not have to significantly cut down the dataset. The data were in fact fairly equal in terms of size, having about 130 samples each. Let's compare the accuracies for our different models.

Neural Network	Accuracy
Feed forward	46.22%
Feed forward (lowered dropout rate)	53.78%
Convolutional	53.78%
Convolutional (increased epochs, lowered	59.66%

layers, removed dropout layer)	
Recurrent	45.38%
Recurrent (decreased learning rate)	58.98%

It seems that our specialized models, with hyperparameter finetuning, have done pretty well for the binary classification task. It appears that the decreased dropout rate has improved our accuracy simply due to the low amount of data that we have.

Let's now discuss the multiclass classification accuracy that we have from our different models.

Neural Network	Test Accuracy
Feed forward	7.48%
Convolutional	9.52%
Recurrent	12.93%

It appears that our recurrent neural network performed the best, with a non-trivial margin above the other methods. However, this accuracy still isn't too good, because we are only doing slightly better than random guessing, and the convolutional neural network and feed forward neural network have performed almost as badly as random guessing. However, this is likely due to our data imbalance.

Classification Task for Random Test Files

For our data, we were also provided with three unlabeled sound clips that we would need to classify. For that, we decided to use our best performing model, the recurrent neural network. We then took the first three seconds and converted those three seconds into a spectrograph object that we then passed to our model. We decoded our results and got these answers.

Sound Clip	Prediction
test1.mp3	Black-capped Chickadee
test2.mp3	Northern Flicker
test3.mp3	Black-capped Chickadee

Of note is that we should have also broken each of the clips down into three second intervals to run through our models. It is very much likely that there could be multiple birds in the clip or that the bird most prominently heard in the clip is not that one that appeared in the first three seconds.

Discussion

Limitations:

It appears that, with the high level of oscillation in our loss functions, that our models could have served to be improved. However, there were a few limitations to our methods. The most important restriction here is our computational power. On several instances, our model, in particular the convolutional neural network, did not have enough memory to continue running. To compensate for that loss of memory, we often decreased the batch size and the number of layers to lower the amount of parameters. That computational gain likely leads to a loss in accuracy.

Consider the following trainable parameters for our models:

Model	Parameters
Binary CNN	23,626
Multiclass CNN	16,975,756

Noticeably, our multiclass convolutional neural network was larger than our binary convolutional neural network.

When considering time as a resource, due to the small number of samples in the dataset, no training run took more than 10 minutes, given that there were at most 170 samples in the dataset. As above, this likely led to some losses in our model.

Predictions Regarding Different Species:

While our models did not exactly compute the accuracy per bird species, we should discuss our choice of which classes to place together. For example, we chose binary classification among bird groups whose groups were about the same size, though even that gave us rather subpar results. Let’s discuss the particular sounds.

Recall that we began our preprocessing at taking frequencies between 0 and 8192 Hz. However, this seems very restrictive. After all, it's quite plausible that many birds could make sounds above 8000 Hz. In particular, a quick search reveals that the Bewick's Wren makes sound at a high frequency, at least according to the Cornell Lab of Ornithology. We should then consider allowing higher pitches to fully capture the range of some of these birds.

Considerations of Other Models:

While the current discussion primarily involves neural networks, we should consider other models as well, such as linear decision trees, linear regression by least squares, or support vector machines. The primary issue with applying these models will just be how we format the data. We notice that the models mentioned above take in data in nice tabular form. The only way to simplify the data might be to flatten each of the matrices, but this would still leave us with an unwieldy amount of data. The reason we are working with neural networks now is just because our data is too complicated.

An initial way we might consider using other models is to aggregate several slices of our data so that we take the average decibels per time chunk and per frequency chunk. However, this would greatly diminish the quality of our data. It is also unclear how such transformations would preserve any structure in our data. We would only be able to classify based on, for example, the average frequency the bird calls at.

Conclusion

We still have much work to complete when it comes to classifying birds. While our models only performed slightly better than random guessing, there remain many hyperparameters that we could fine-tune. In particular, we noticed above that tuning the learning rate afforded us much less oscillation in the data. Additionally, we should consider increasing the training size set to strengthen our models as well. With this data, we could potentially train models that, like many birdwatching apps, can be used to identify birds by their call, as opposed to some subjective features.

References

[1]Cornell Lab of Ornithology. (n.d.). Bewick's Wren sounds. All About Birds. Retrieved May 12, 2025, from https://www.allaboutbirds.org/guide/Bewicks_Wren/sounds

[2] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (1st ed.) [PDF]. Springer.

[3] skadaver (https://stats.stackexchange.com/users/254303/skadaver), Cross Entropy vs. Sparse Cross Entropy: When to use one over the other, URL (version: 2020-02-03): https://stats.stackexchange.com/q/420730

[4] Stack Abuse. (2023, July 19). Don’t Use Flatten: Global Pooling for CNNs with TensorFlow and Keras.
https://stackabuse.com/dont-use-flatten-global-pooling-for-cnns-with-tensorflow-and-keras/