Institute of Informatics Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science
Department of Graphics, Computer Vision and Digital Systems
Biologically Inspired Artificial Intelligence

Report

Cats and Dogs Image Classification

Monika Dudzińska

Short introduction presenting the project topic

The theme of the project is to classify images of cats and dogs using a neural network. Distinguishing images of dogs and cats is easy for humans, but it is particularly difficult to tell apart automatically. The aim of the project is to create a model that will classify the images as accurately as possible.

Analysis of the task

1. Methodology

The following types of neural networks were compared during the research:

- Artificial Neural Networks (ANN)

  ANN consists of at least 3 layers – one input layer, a hidden layer (the one that processes the inputs, there can be more than one), and one output layer. Inputs are processed only in the forward direction. Thanks to activation functions, ANN are capable of learning any nonlinear function. They can be used to solve problems related to: Tabular data, Image data, Text data. However, one of its main disadvantages is that it loses the spatial features (arrangement of pixels and the relationship between them in an image).

- Recurrent Neural Networks (RNN)

  The main difference between RNN and ANN is that RNN has a recurrent connection on the hidden state. This looping constraint ensures that the input data contains sequential information.
  RNN are used to solve the problems related to: Time Series data, Text data, and Audio data.

- Convolution Neural Networks (CNN)

  Although CNN also performs well on sequential inputs, it was introduced to solve problems related to image data. CNN automatically learns the filters, which help in extracting the relevant features from the input data by using the convolution operation. In contrast to ANN, CNN captures the spatial features from an image, which helps in identifying the object, the location of an object, and its relation with other objects in an image.
  The basic CNN structure is as follows: Convolution -> Pooling -> Convolution -> Pooling -> Fully Connected Layer -> Output

Since the project concerns image classification my choice was convolutional neural network.

2. Datasets

Google search results for the dataset, which contains images of cats and dogs, seem to be limited to almost only the Asirra dataset provided by Microsoft Research. Since it is commonly used and met the project requirements (the size and availability), I also decided to use it.

This dataset was supposed to contain 25,000 images, but at the time I was choosing it, I was not aware that 54 of them would turn out to be corrupted. In order to keep the dataset balanced I had to remove 6 images of cats. 40 images were taken to create a testing set._This way the training set contained 24,900 images, including 12,450 images of dogs and 12,450 images of cats.

3. The tools and libraries

While doing the research I acknowledged that ones of the most common Python Machine Learning IDEs are:

- Spyder

  This IDE is known for being simple, light-weight and easy to install. One of the prons is the Documentation Viewer that shows the documentation related to classes or functions in a project. Spyder is considered best for testing and development of scientific applications and scripts that use libraries such as SciPy, NumPy and Matplotlib.

- Geany

  This is also a light-weighted IDE but as capable as any other IDE present out there. Although it has features that can come in handy and make writing code more convenient, there are no special prons that would draw my attention towards this environment.

- Rodeo

  This Python IDE was built for the purpose of machine learning and data science. It uses the IPython kernel. It lets users explore, compare and interact with the data frames and plots, however, it does not consist of code analysis, PEP 8, etc.

- PyCharm

  PyCharm is most famous in the professional world for data science and conventional Python programming. It provides support for important libraries like Matplotlib, NumPy and Pandas. It also includes code completion, auto-indentation, runtime debugger, PEP-8 that enables writing neat codes. It has a documentation viewer and video tutorials.

- JuPyter

  It’s an open source platform that supports sharing live codes, and documents with equations and visualizations. It has Big Data integration within to help the data scientists.

The tool of my first choice was PyCharm since I was already familiar with it. Due to issues that occurred during implementation of a model I learnt that the hardware configuration of my machine was not proper for this IDE. Therefore, I switched to Google Colaboratory, in which it is not necessary to configure the environment and one can easily import libraries and datasets.

As we can read on google research page - “Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.”
Colab is especially well suited to machine learning, data analysis and education.

Deep learning libraries comparison:

	Keras	PyTorch	TensorFlow
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance
Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does It Have Trained Models?	Yes	Yes	Yes
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high-performance	Fast, high-performance
Written In	Python	Lua	C++, CUDA, Python

source: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

Keras can run on top of other open-source machine learning libraries such as TensorFlow. Considering all the above pros and cons I decided to use both TensorFlow and Keras.

Short summary of other used libraries:

THE LIBRARY	THE REASON FOR USE
numpy	array operations
opencv	image operations
os	joining paths of directories
random	shuffling the data
tqdm	progress bars
pickle	saving and loading data
time	tensorboard

The visualization tool used in this project is TensorBoard provided with TensorFlow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower dimensional space, and much more.

Specification of the software

The project is divided into three cells: data preparation, training the model, and ready program.

data preparation — loads a dataset from Google Drive and prepares it for being used by model.

Data structure	Type	Description
`CATEGORIES`	list	Holds categories of cats and dogs
`training_data`	list	Contains features and labels
`img_array`	array	Holds features (images converted into array)
`new_array`	array	Holds resized features
`X`	list	Contains features
`y`	list	Contains labels
`X`	np array	Contains features
`y`	np array	Contains labels

Function	Description
`create_training_data()`	Loads image files, resizes them, changes to grayscale and converts into an array Throws: `IncorrectFileException` – if the file is incorrect
`random.shuffle(training_data)`	Shuffle the sequence of `training_data` list in place.
`pack_training_data()`	Splits training_data list into features and labels lists. Converts them into an np array.
`pack_pickle()`	Saves features and labels arrays as pickle files.

model training – performs model training and saves it into a file. Crates TensorBoard files.

Data structure	Type	Description
`X`	np array	Contains features
`y`	np array	Contains labels
`dense_layers`	list	Contains number of Dense layers
`layer_nodes`	list	Contains number of nodes per layer
`conv_layers`	list	Contains number of Conv2D layers

Function	Description
`open_pickle()`	Opens pickle files and loads the data
`train_model()`	Trains various versions of models

ready program – loads model and classifies provided image files into Cat and Dog categories.

Data structure	Type	Description
`CATEGORIES`	list	Holds categories of cats and dogs
`img_array`	array	Holds features (images converted into array)
`new_array`	array	Holds resized features

Function	Description
`prepare(filepath)`	Resizes image file, changes to grayscale and converts into an array Parameters: path to the file Returns: array Throws: `IncorrectFileException` – if the file is incorrect
`proceed()`	Loads image files. Calls `prepare()` and `model.predict()`. Prints the images and the results.

Experiments

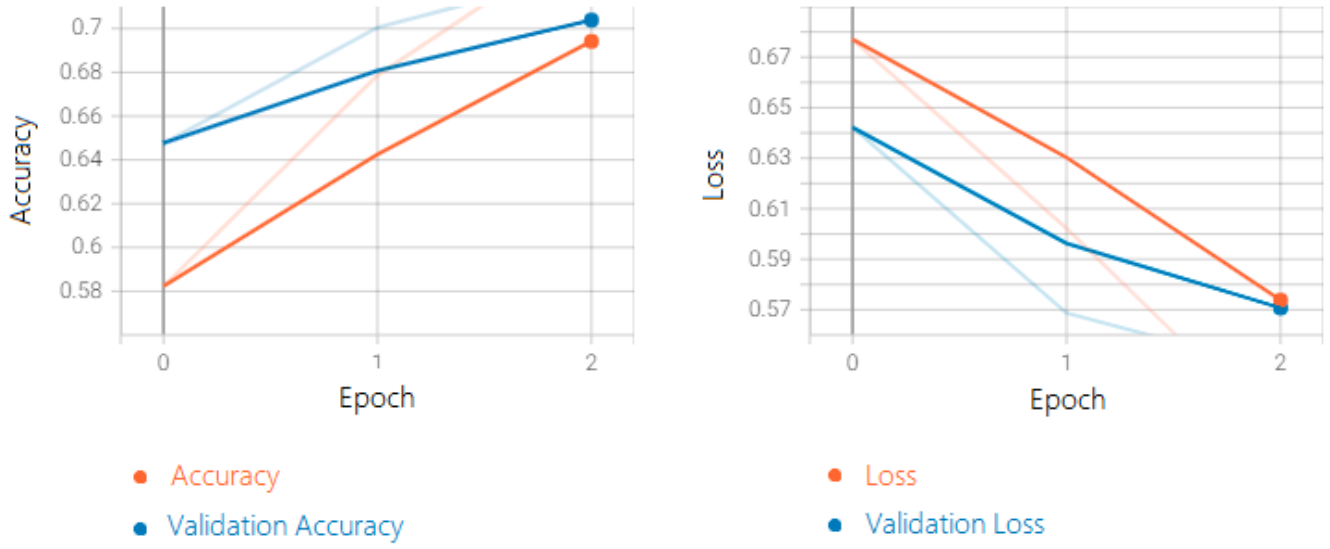
In order to determine a good starting point I did some brief research regarding the best and recommended structure for image classification, including; layers, activation functions, optimizers, number of nodes, Conv2D kernels and MaxPooling2D pool sizes.

The very first model I created consisted of the following layers:

- Conv2D (64 filters, (3,3)kernel,dynamic input shape, relu activation function, (2,2)pool-size)
- Conv2D (64 filters, (3, 3) kernel, relu activation function, (2, 2) pool size)
- Flatten
- Dense (64 nodes, relu activation function)
- Dense (1 node, sigmoid activation function)

Flatten layer has been used to convert feature maps into 1D feature vectors. This need arose from the fact that the Convolutional layer is 2D while the Dense layer is 1D.

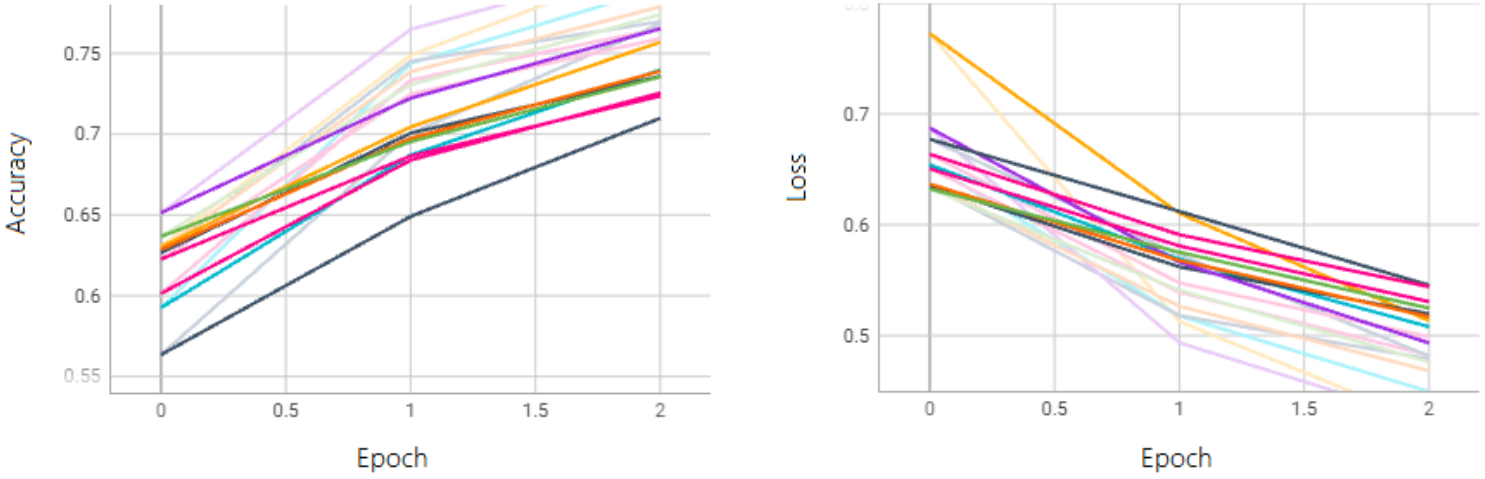
The first model training results for three epochs:



I came to the conclusion that the most basic things to modify are Conv2D layers, nodes per layer, and the number of Dense layers.

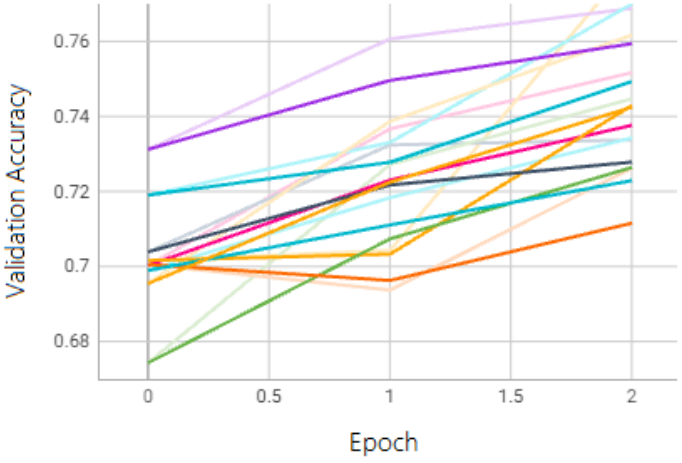
The graphs below present the results of training after particular modifications.

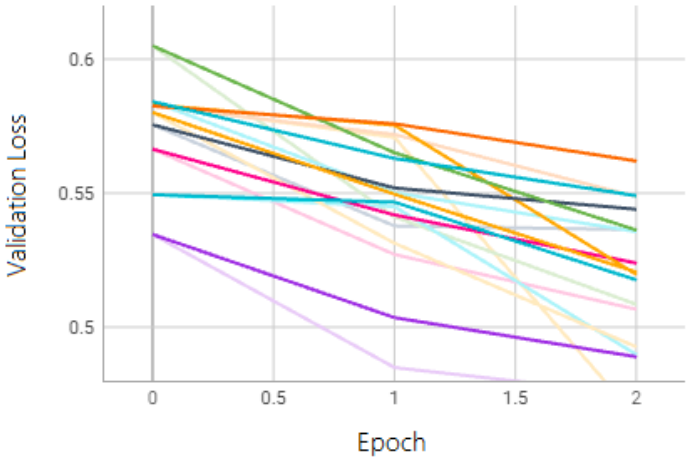
The 0 dense layer model training results
for various numbers of convolutional layers and nodes:



- 1 convolutional layer, 128 nodes per layer — Accuracy: 0.8073 Loss: 0.4205
- 1 convolutional layer, 32 nodes per layer — Accuracy: 0.7593 Loss: 0.4988
- 1 convolutional layer, 64 nodes per layer — Accuracy: 0.7741 Loss: 0.4232
- 2 convolutional layers, 128 nodes per layer — Accuracy: 0.7741 Loss: 0.4761
- 2 convolutional layers, 32 nodes per layer — Accuracy: 0.7697 Loss: 0.4788
- 2 convolutional layers, 64 nodes per layer — Accuracy: 0.7789 Loss: 0.4682
- 3 convolutional layers, 128 nodes per layer — Accuracy: 0.7684 Loss: 0.4814
- 3 convolutional layers, 32 nodes per layer — Accuracy: 0.7655 Loss: 0.4822
- 3 convolutional layers, 64 nodes per layer — Accuracy: 0.7904 Loss: 0.4494

The 0 dense layer model training validation results for various numbers of convolutional layers and nodes:



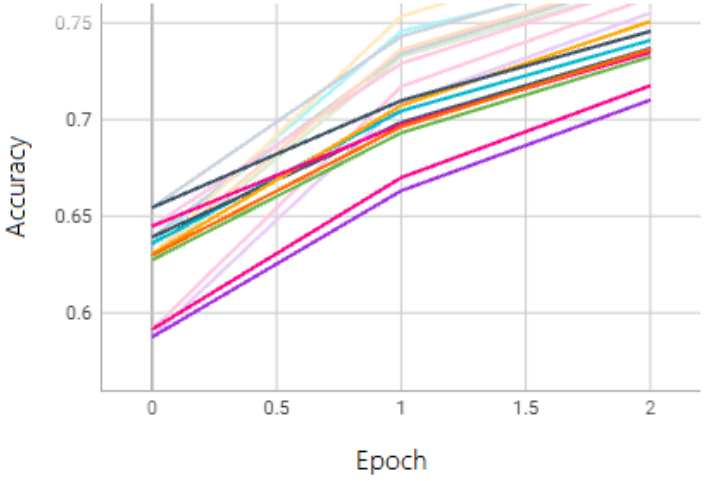


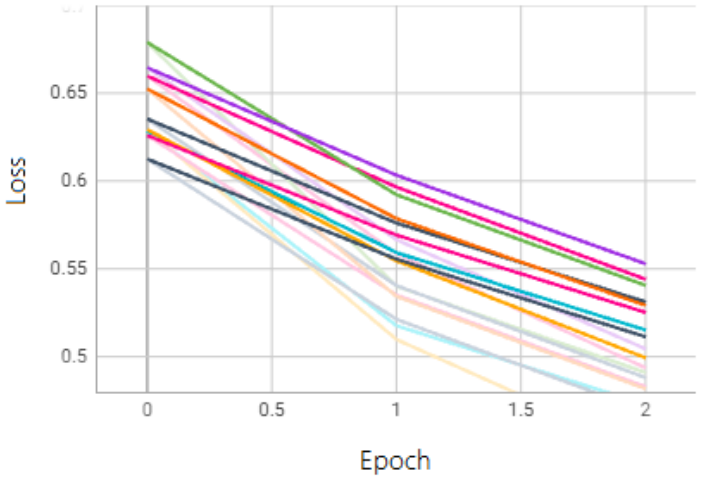
1 convolutional layer, 128 nodes per layer	Validation Accuracy: 0.7262	Validation Loss: 0.5487
1 convolutional layer, 32 nodes per layer	Validation Accuracy: 0.7343	Validation Loss: 0.5356
1 convolutional layer, 64 nodes per layer	Validation Accuracy: 0.7337	Validation Loss: 0.5363
2 convolutional layer, 128 nodes per layer	Validation Accuracy: 0.77	Validation Loss: 0.4897
2 convolutional layer, 32 nodes per layer	Validation Accuracy: 0.7617	Validation Loss: 0.4926
2 convolutional layer, 64 nodes per layer	Validation Accuracy: 0.7517	Validation Loss: 0.5066
3 convolutional layer, 128 nodes per layer	Validation Accuracy: 0.781	Validation Loss: 0.4661
3 convolutional layer, 32 nodes per layer	Validation Accuracy: 0.7447	Validation Loss: 0.5085
3 convolutional layer, 64 nodes per layer	Validation Accuracy: 0.7688	Validation Loss: 0.4748

Summary

The highest accuracy - 3 convolutional layers, 64 nodes [0.7925]
The lowest loss - 3 convolutional layers, 64 nodes [0.4462]
The highest validation accuracy - 3 convolutional layers, 128 nodes [0.781]
The lowest validation loss - 3 convolutional layers, 128 nodes [0.4661]

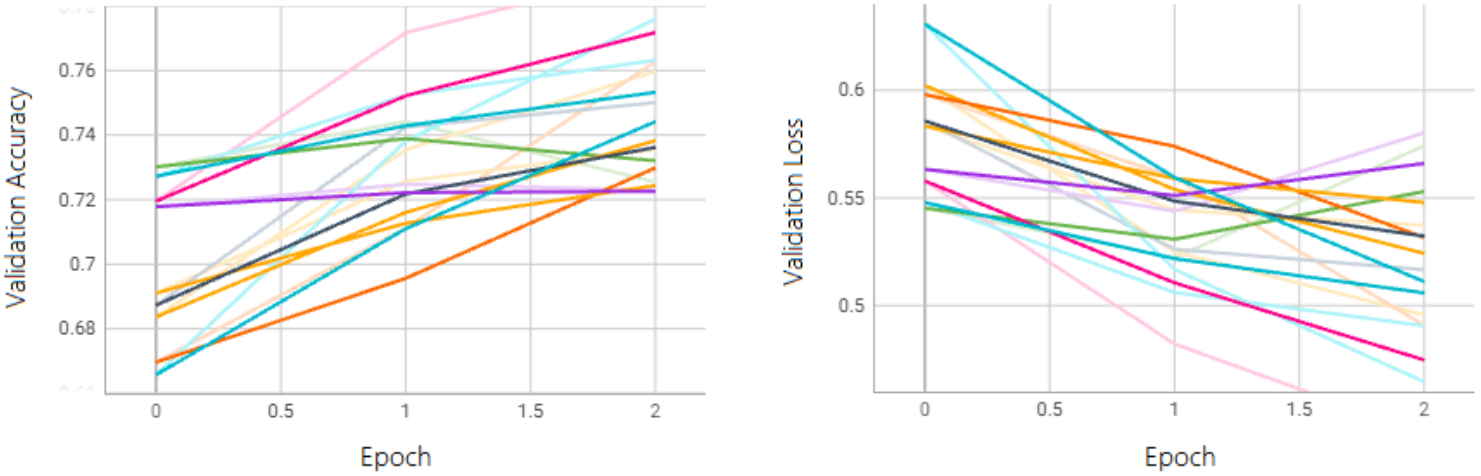
The 1 dense layer model training results
for various numbers of convolutional layers and nodes:





- 1 convolutional layer, 128 nodes per layer — Accuracy: 0.7702 Loss: 0.491
- 1 convolutional layer, 32 nodes per layer — Accuracy: 0.7736 Loss: 0.488
- 1 convolutional layer, 64 nodes per layer — Accuracy: 0.7748 Loss: 0.4816
- 2 convolutional layers, 128 nodes per layer — Accuracy: 0.7801 Loss: 0.4689
- 2 convolutional layers, 32 nodes per layer — Accuracy: 0.7706 Loss: 0.4829
- 2 convolutional layers, 64 nodes per layer — Accuracy: 0.7761 Loss: 0.4728
- 3 convolutional layers, 128 nodes per layer — Accuracy: 0.7633 Loss: 0.4936
- 3 convolutional layers, 32 nodes per layer — Accuracy: 0.7552 Loss: 0.5044
- 3 convolutional layers, 64 nodes per layer — Accuracy: 0.7925 Loss: 0.4462

The 1 dense layer model training validation results for various numbers of convolutional layers and nodes::

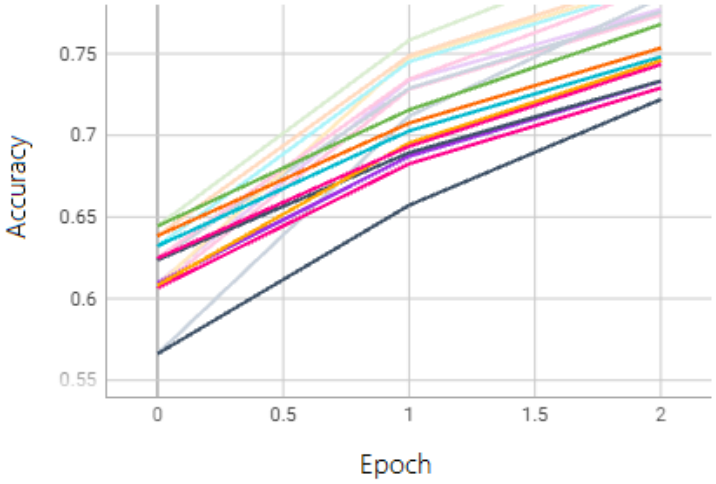


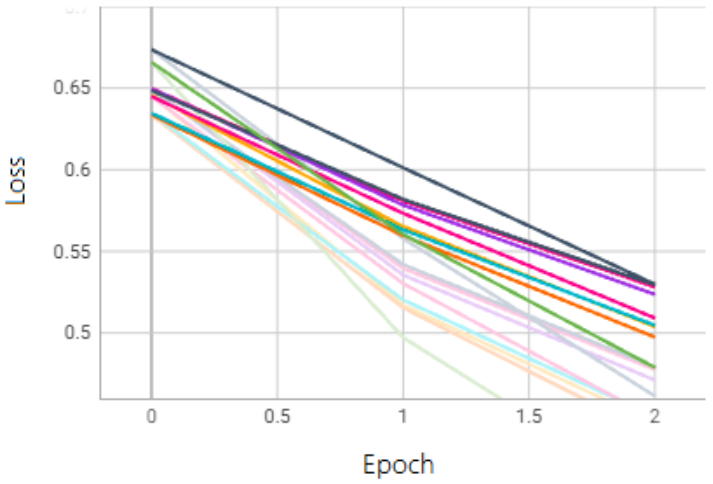
1 convolutional layer, 128 nodes per layer	Validation Accuracy: 0.723	Validation Loss: 0.5803
1 convolutional layer, 32 nodes per layer	Validation Accuracy: 0.7356	Validation Loss: 0.5372
1 convolutional layer, 64 nodes per layer	Validation Accuracy: 0.7254	Validation Loss: 0.5743
2 convolutional layers, 128 nodes per layer	Validation Accuracy: 0.7628	Validation Loss: 0.4908
2 convolutional layers, 32 nodes per layer	Validation Accuracy: 0.7632	Validation Loss: 0.4906
2 convolutional layers, 64 nodes per layer	Validation Accuracy: 0.7501	Validation Loss: 0.5166
3 convolutional layers, 128 nodes per layer	Validation Accuracy: 0.776	Validation Loss: 0.4645
3 convolutional layers, 32 nodes per layer	Validation Accuracy: 0.7597	Validation Loss: 0.4956
3 convolutional layers, 64 nodes per layer	Validation Accuracy: 0.7906	Validation Loss: 0.4402

Summary

The highest accuracy - 1 convolutional layer, 128 nodes [0.8073]
The lowest loss - 1 convolutional layer, 128 nodes [0.4205]
The highest validation accuracy - 3 convolutional layers, 64 nodes [0.7906]
The lowest validation loss - 3 convolutional layers, 64 nodes [0.4402]

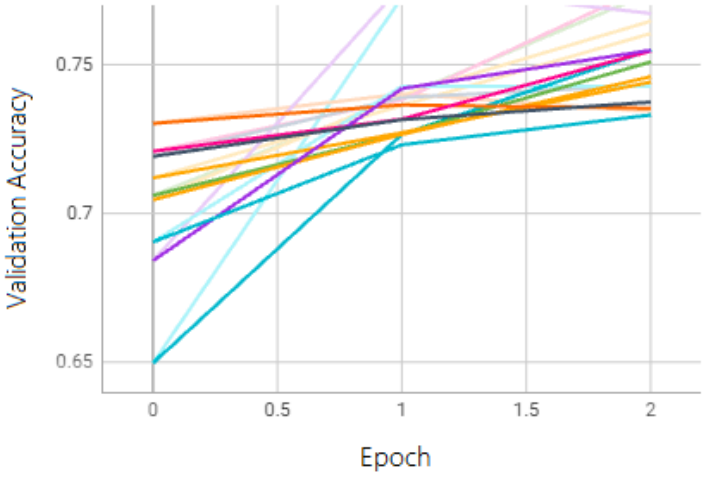
The 2 dense layer model training results
for various numbers of convolutional layers and nodes:

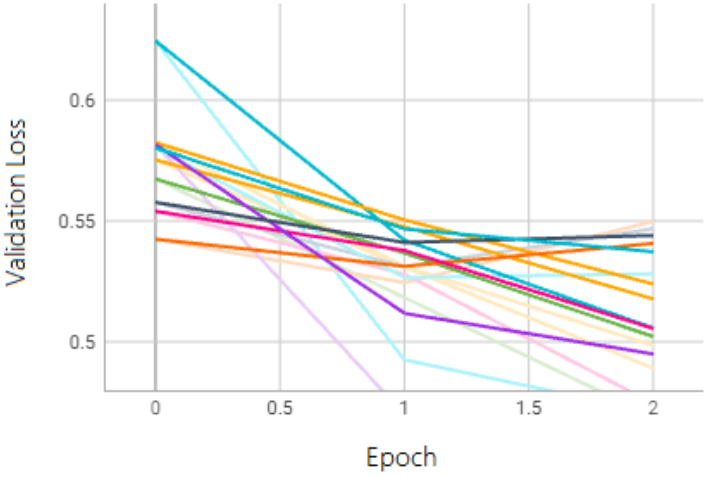




- 1 convolutional layer, 128 nodes per layer — Accuracy: 0.8183 Loss: 0.4004
- 1 convolutional layer, 32 nodes per layer — Accuracy: 0.7756 Loss: 0.4791
- 1 convolutional layer, 64 nodes per layer — Accuracy: 0.7978 Loss: 0.4376
- 2 convolutional layers, 128 nodes per layer — Accuracy: 0.7916 Loss: 0.4475
- 2 convolutional layers, 32 nodes per layer — Accuracy: 0.7737 Loss: 0.4775
- 2 convolutional layers, 64 nodes per layer — Accuracy: 0.7917 Loss: 0.4486
- 3 convolutional layers, 128 nodes per layer — Accuracy: 0.7843 Loss: 0.4613
- 3 convolutional layers, 32 nodes per layer — Accuracy: 0.7773 Loss: 0.4711
- 3 convolutional layers, 64 nodes per layer — Accuracy: 0.7943 Loss: 0.4448

The 2 dense layer model training validation results
for various numbers of convolutional layers and nodes:





- 1 convolutional layer, 128 nodes per layer — Validation Accuracy: 0.7343 Validation Loss: 0.5498
- 1 convolutional layer, 32 nodes per layer — Validation Accuracy: 0.7427 Validation Loss: 0.5281
- 1 convolutional layer, 64 nodes per layer — Validation Accuracy: 0.7431 Validation Loss: 0.5468
- 2 convolutional layers, 128 nodes per layer — Validation Accuracy: 0.7605 Validation Loss: 0.4984
- 2 convolutional layers, 32 nodes per layer — Validation Accuracy: 0.7647 Validation Loss: 0.4891
- 2 convolutional layers, 64 nodes per layer — Validation Accuracy: 0.7767 Validation Loss: 0.4745
- 3 convolutional layers, 128 nodes per layer — Validation Accuracy: 0.7822 Validation Loss: 0.4707
- 3 convolutional layers, 32 nodes per layer — Validation Accuracy: 0.7742 Validation Loss: 0.469
- 3 convolutional layers, 64 nodes per layer — Validation Accuracy: 0.7672 Validation Loss: 0.4788

Summary

The highest accuracy - 1 convolutional layer, 128 nodes [0.8183]
The lowest loss - 1 convolutional layer, 128 nodes [0.4004]
The highest validation accuracy - 3 convolutional layers, 128 nodes [0.7822]
The lowest validation loss - 3 convolutional layers, 32 nodes [0.469]

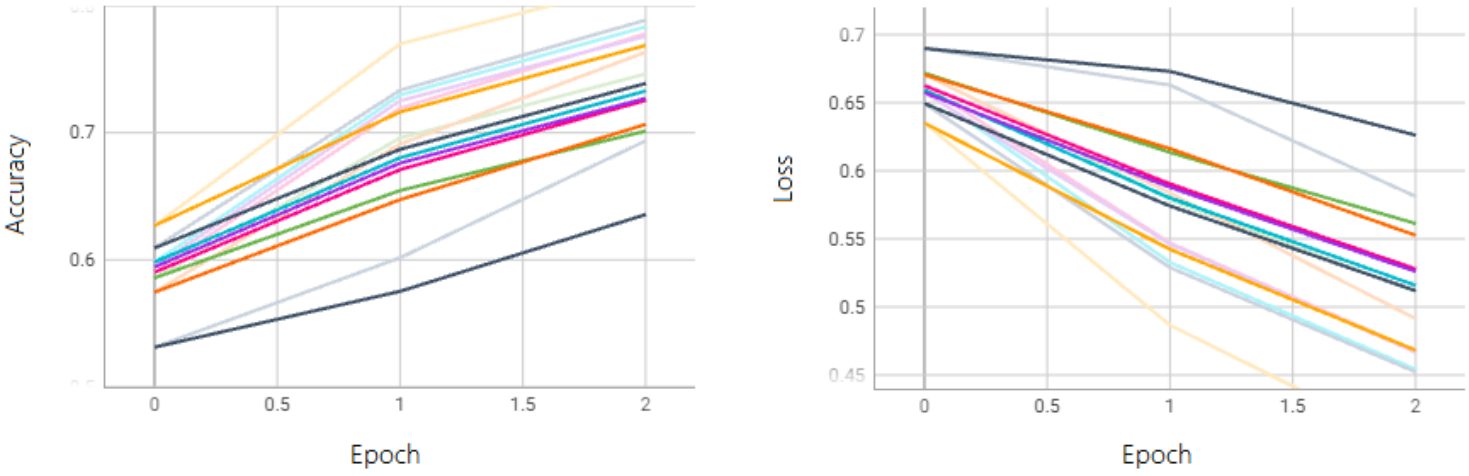
Since the training loss indicates how well the model is fitting the training data and the validation loss indicates how well the model fits new data, I decided to choose models with the lowest validation loss.

The top 3 models with the lowest validation loss:

1 dense layer, 3 convolutional layers, 64 nodes [0.4402]
1 dense layer, 3 convolutional layers, 128 nodes [0.4645]
0 dense layers, 3 convolutional layers, 128 nodes [0.4661]

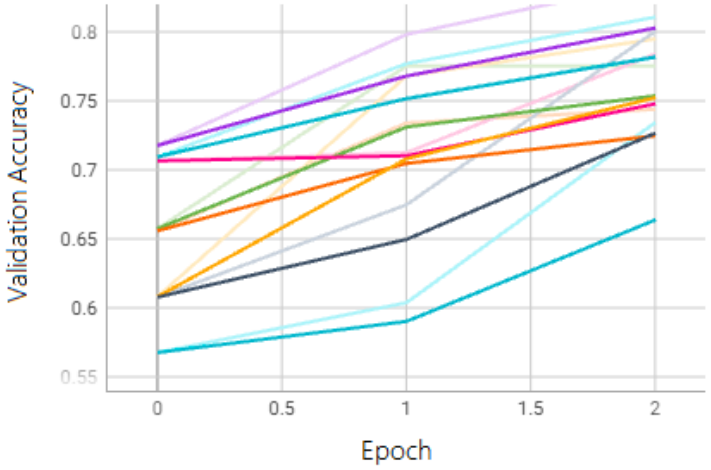
The results above led me to the decision of creating models with 0/1 dense layer, 3/4 convolutional layers, and 128/256 nodes.

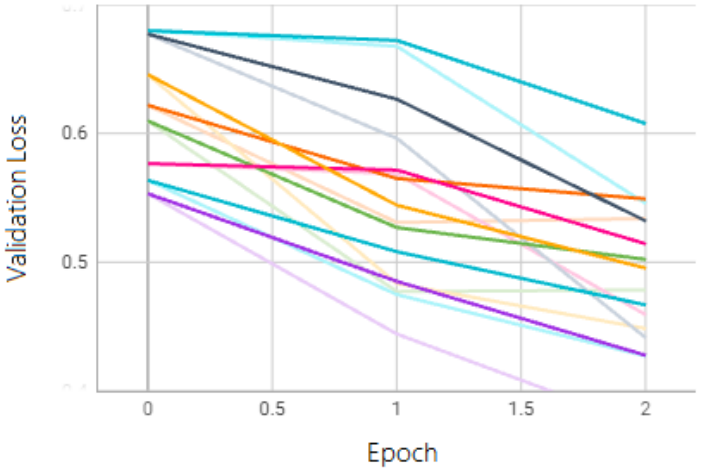
The 0/1 dense layer model training results for various numbers of convolutional layers and nodes:



3 convolutional layer, 128 nodes per layer, 0 dense layers — Accuracy: 0.7887	Loss: 0.4522
4 convolutional layer, 128 nodes per layer, 0 dense layers — Accuracy: 0.778	Loss: 0.4677
3 convolutional layer, 256 nodes per layer, 0 dense layers — Accuracy: 0.776	Loss: 0.4666
4 convolutional layers, 256 nodes per layer, 0 dense layers — Accuracy: 0.7635	Loss: 0.4913
3 convolutional layers, 128 nodes per layer, 1 dense layer — Accuracy: 0.7834	Loss: 0.4541
4 convolutional layers, 128 nodes per layer, 1 dense layer — Accuracy: 0.8191	Loss: 0.397
3 convolutional layers, 256 nodes per layer, 1 dense layer — Accuracy: 0.7463	Loss: 0.5111
4 convolutional layers, 256 nodes per layer, 1 dense layer — Accuracy: 0.6937	Loss: 0.5811

The 0/1 dense layer model training validation results for various numbers of convolutional layers and nodes:





3 convolutional layer, 128 nodes per layer, 0 dense layers	Validation Accuracy: 0.8107	Validation Loss: 0.4271
4 convolutional layer, 128 nodes per layer, 0 dense layers	Validation Accuracy: 0.7949	Validation Loss: 0.4484
3 convolutional layer, 256 nodes per layer, 0 dense layers	Validation Accuracy: 0.7554	Validation Loss: 0.4784
4 convolutional layers, 256 nodes per layer, 0 dense layers	Validation Accuracy: 0.8005	Validation Loss: 0.4412
3 convolutional layers, 128 nodes per layer, 1 dense layer	Validation Accuracy: 0.7841	Validation Loss: 0.4591
4 convolutional layers, 128 nodes per layer, 1 dense layer	Validation Accuracy: 0.8363	Validation Loss: 0.3725
3 convolutional layers, 256 nodes per layer, 1 dense layer	Validation Accuracy: 0.7439	Validation Loss: 0.5339
4 convolutional layers, 256 nodes per layer, 1 dense layer	Validation Accuracy: 0.7344	Validation Loss: 0.5455

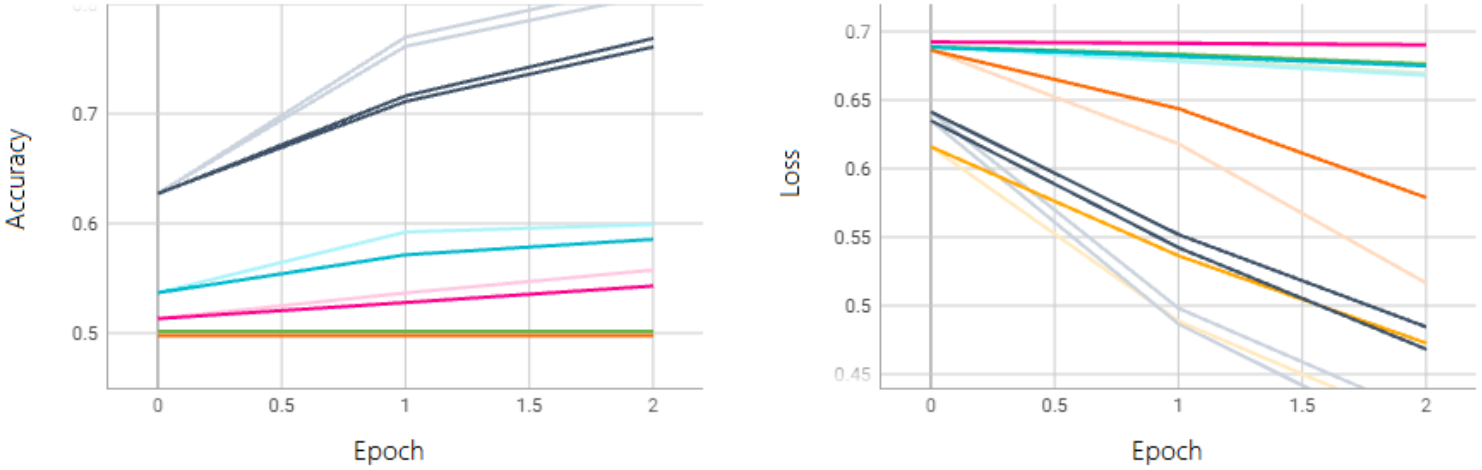
Summary

The highest accuracy - 1 dense layer, 4 convolutional layers, 128 nodes [0.8191]
The lowest loss - 1 dense layer, 4 convolutional layers, 128 nodes [0.397]
The highest validation accuracy - 1 dense layer, 4 convolutional layers, 128 nodes [0.8363]
The lowest validation loss - 1 dense layer, 4 convolutional layers, 128 nodes [0.3725]

According to my research, the relu activation function, sigmoid output function, and adam optimizer are the most recommended in their categories. I decided to modify the best model (1 dense layer, 4 convolutional layers, 128 nodes) and check if the tanh activation function, softmax output function, and SGD optimizer (which are also recommended) will conduct better or worse results.

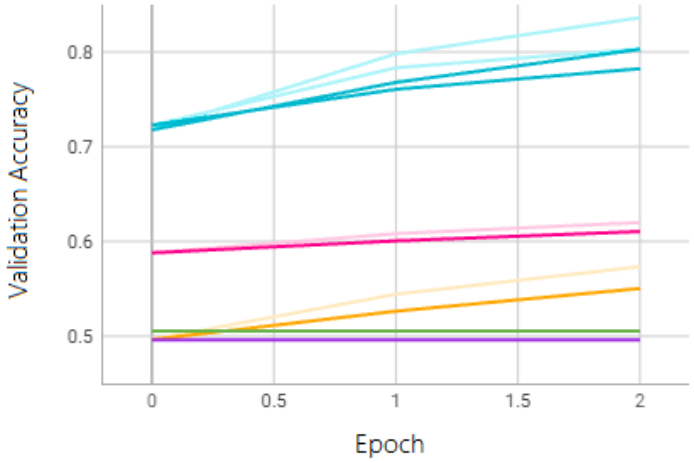
There are many other factors that could have also been changed and tested however that would be even more time-consuming.

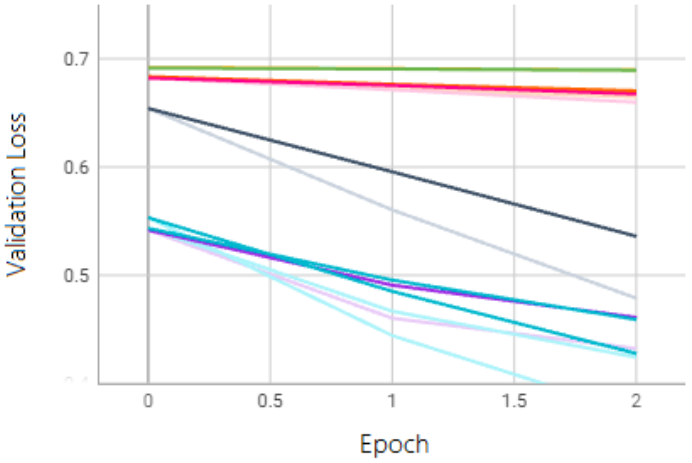
The 4-Conv2D 1-Dense 128-nodes model training results for various activation functions and optimizers:



relu activation function, sigmoid output function, SGD optimizer — Accuracy: 0.5575	Loss: 0.6893
relu activation function, sigmoid output function, adam optimizer — Accuracy: 0.8191	Loss: 0.397
relu activation function, softmax output function, SGD optimizer — Accuracy: 0.4977	Loss: 0.6894
relu activation function, softmax output function, adam optimizer — Accuracy: 0.4977	Loss: 0.5164
tanh activation function, sigmoid output function, SGD optimizer — Accuracy: 0.5991	Loss: 0.668
tanh activation function, sigmoid output function, adam optimizer — Accuracy: 0.8087	Loss: 0.4197
tanh activation function, softmax output function, SGD optimizer — Accuracy: 0.5017	Loss: 0.6695
tanh activation function, softmax output function, adam optimizer — Accuracy: 0.5017	Loss: 0.4114

The 4-Conv2D 1-Dense 128-nodes model training validation results for various activation functions and optimizers:





relu activation function, sigmoid output function, SGD optimizer — Validation Accuracy: 0.5734 Validation Loss: 0.6884

relu activation function, sigmoid output function, adam optimizer — Validation Accuracy: 0.8363 Validation Loss: 0.3725

relu activation function, softmax output function, SGD optimizer — Validation Accuracy: 0.5054 Validation Loss: 0.6886

relu activation function, softmax output function, adam optimizer — Validation Accuracy: 0.5054 Validation Loss: 0.4787

tanh activation function, sigmoid output function, SGD optimizer — Validation Accuracy: 0.6199 Validation Loss: 0.6598

tanh activation function, sigmoid output function, adam optimizer — Validation Accuracy: 0.8033 Validation Loss: 0.4241

tanh activation function, softmax output function, SGD optimizer — Validation Accuracy: 0.496 Validation Loss: 0.6646

tanh activation function, softmax output function, adam optimizer — Validation Accuracy: 0.496 Validation Loss: 0.4323

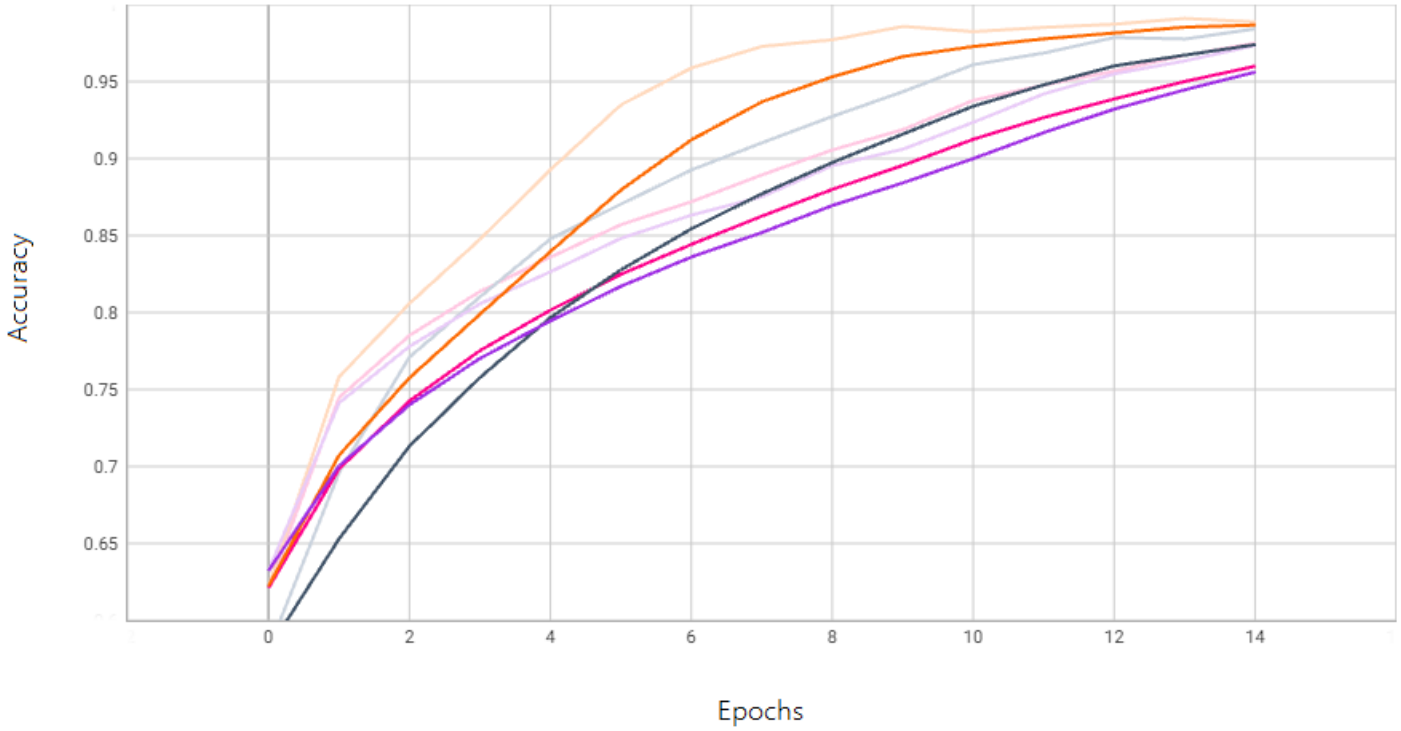
Summary

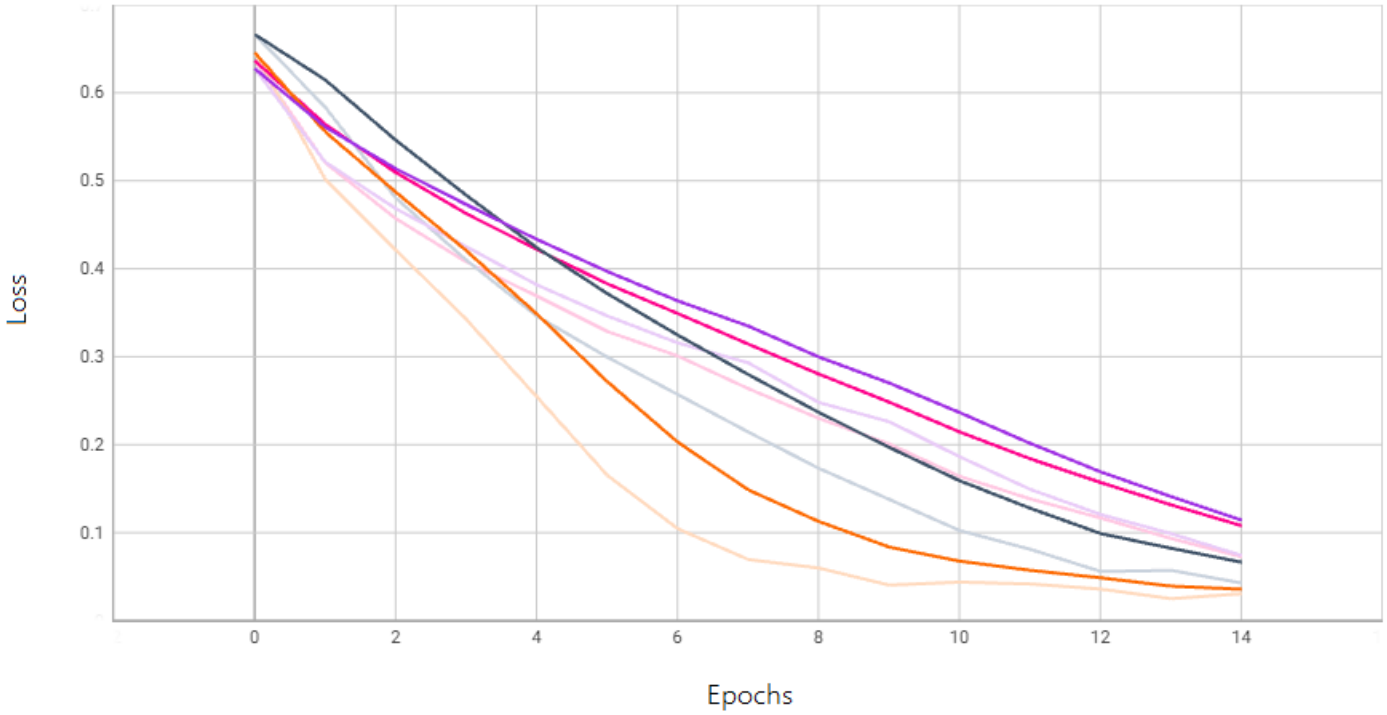
The highest accuracy - relu activation function, sigmoid output function, adam optimizer [0.8191]
The lowest loss - relu activation function, sigmoid output function, adam optimizer [0.397]
The highest validation accuracy - relu activation function, sigmoid output function, adam optimizer [0.8363]
The lowest validation loss - relu activation function, sigmoid output function, adam optimizer [0.3725]

The top 4 models with the lowest validation loss:

Relu sigmoid adam 4conv, 1dense, 128nodes [0.3725]
Tanh sigmoid adam 4conv, 1dense, 128nodes [0.4241]
Relu sigmoid adam 3conv, 0dense, 128nodes [0.4271]
Relu sigmoid adam 3conv, 1dense, 64nodes [0.4402]

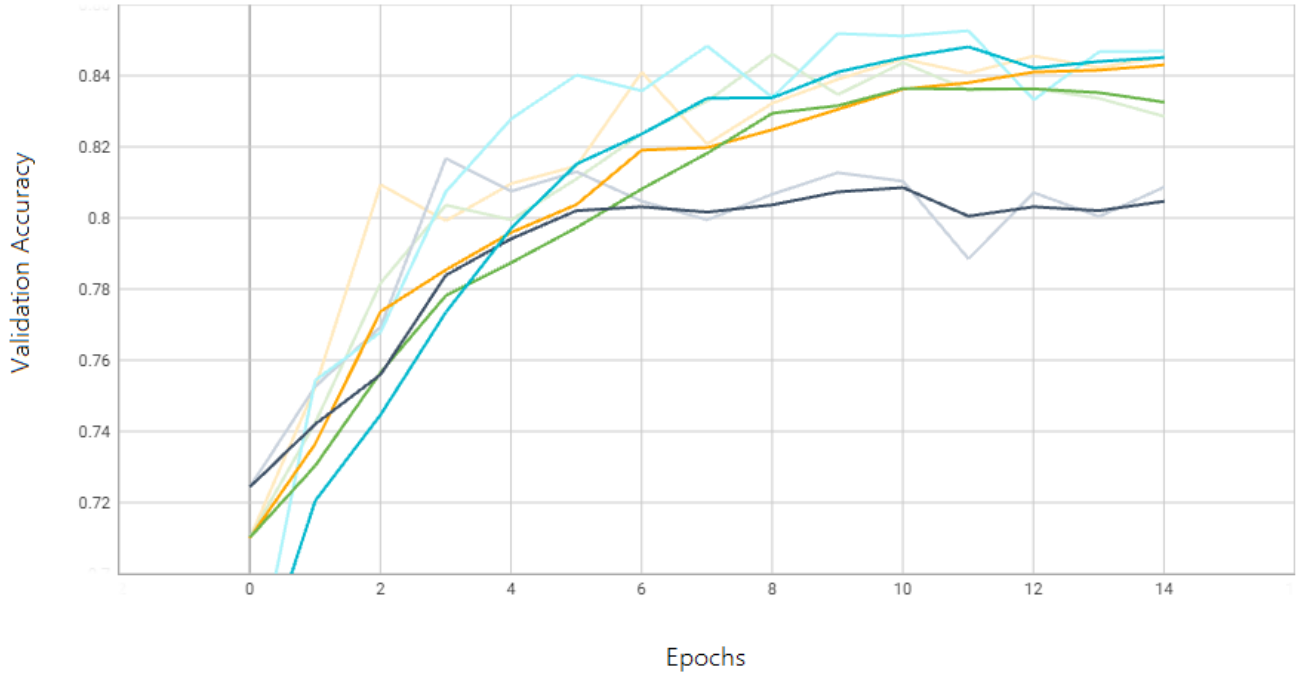
Comparison of the training results of the best four models in 15 epoches:

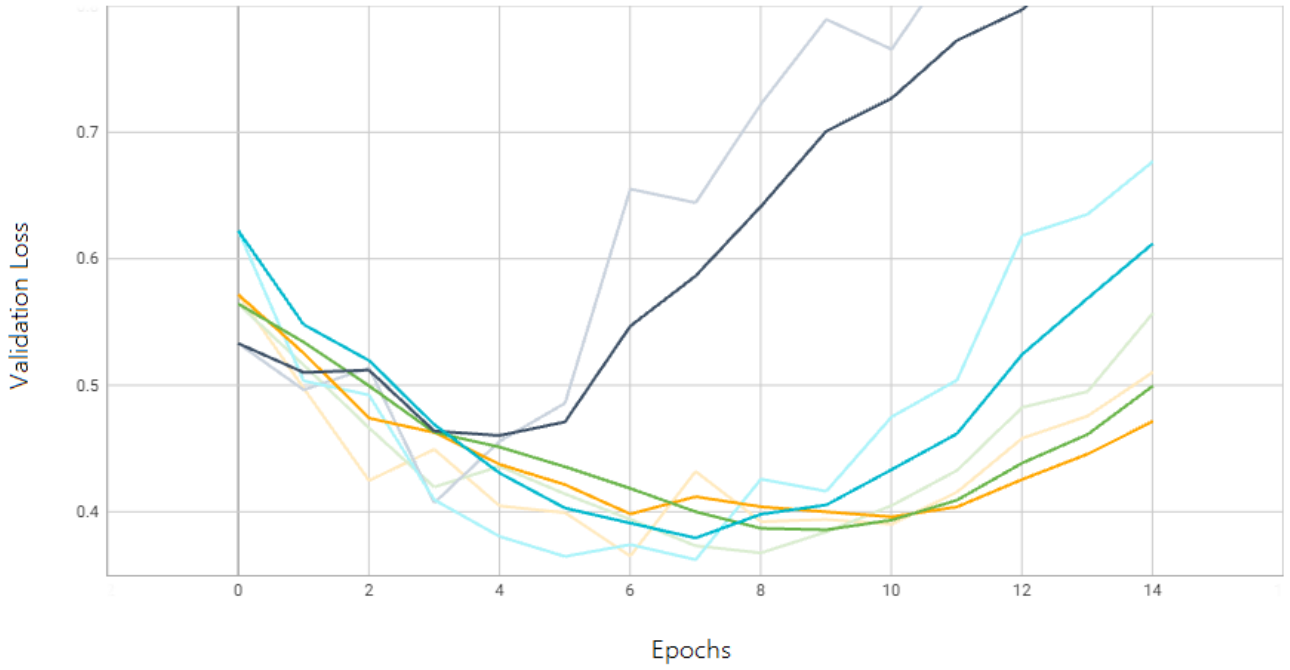




- relu, sigmoid, adam, 4 convolutional layers, 128 nodes per layer, 1 dense layer — Accuracy [0.9843] Loss [0.04274]
- relu, sigmoid, adam, 3 convolutional layers, 128 nodes per layer, 0 dense layers — Accuracy [0.9749] Loss [0.07243]
- relu, sigmoid, adam, 3 convolutional layers, 64 nodes per layer, 1 dense layer — Accuracy [0.9736] Loss [0.07404]
- tanh, sigmoid, adam, 4 convolutional layers, 128 nodes per layer, 1 dense layer — Accuracy [0.9888] Loss [0.03062]

Comparison of the training validation results of the best four models in 15 epoches:





- relu, sigmoid, adam, 4 convolutional layers, 128 nodes per layer, 1 dense layer — Validation Accuracy [0.8469] Validation Loss [0.6769]
- relu, sigmoid, adam, 3 convolutional layers, 128 nodes per layer, 0 dense layers — Validation Accuracy [0.8452] Validation Loss [0.5104]
- relu, sigmoid, adam, 3 convolutional layers, 64 nodes per layer, 1 dense layer — Validation Accuracy [0.8285] Validation Loss [0.557]
- tanh, sigmoid, adam, 4 convolutional layers, 128 nodes per layer, 1 dense layer — Validation Accuracy [0.8087] Validation Loss [0.886]

In total, I tested 40 models. The best out of them turned out to be: “relu + sigmoid + adam + 3 convolutional layers + 128 nodes per layer + 0 dense layers” and thus it became the one I decided to use in the project.

The final model structure:

- Conv2D (128 filters, (3, 3) kernel, dynamic input shape, relu activation function, (2,2) pool-size)
- Conv2D (128 filters, (3, 3) kernel, relu activation function, (2, 2) pool size)
- Conv2D (128 filters, (3, 3) kernel, relu activation function, (2, 2) pool size)
- Flatten
- Dense (1 node, sigmoid activation function)

The training results of the final model in 30 epoches:

Analyzing the results

- The training loss goes down over time, achieving low values
- The validation loss goes down until epoch 8, and there it starts going up again

The epoch 8 is the beginning of overfitting, therefore in the project I use the model trained in only 8 epochs — accuracy 0.8860, loss 0.2733, validation accuracy 0.8371, validation loss 0.3650

Testing the model

Cats and Dogs Image Classification

In order to run the porgram please paste the path to the image and run the code (Shift + Enter) in the cell below

```
CATEGORIES = ["Dog", "Cat"]
from google.colab.patches import cv2_imshow
def prepare(filepath):
    IMG_SIZE = 100
    img_array = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)
    try:
      new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
      return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
    except Exception as IncorrectFileException:
      print(" Bad file: " + filepath)


model = tf.keras.models.load_model("30-CNN.model")
for img in os.listdir('drive/My Drive/PetImages/Test'):
  img_shown = cv2.imread('drive/My Drive/PetImages/Test/' + img)
  new_img_shown = cv2.resize(img_shown, (140, 220))
  cv2_imshow(new_img_shown)
  prediction = model.predict([prepare('drive/My Drive/PetImages/Test/' + img)])
  print('\n' + img + " is a " + CATEGORIES[int(prediction[0][0])] + '\n')
```

```
Uszaty.jpg is a Dog
```

Results of the test performed on the testing set:

1.jpg — Cat ✔️
2.jpg — Cat ✔️
3.jpg — Dog ✔️
4.jpg — Dog ❌
5.jpg — Dog ❌
6.jpg — Dog ✔️
7.jpg — Dog ✔️
8.jpg — Cat ✔️
9.jpg — Dog ✔️
10.jpg — Cat ✔️
11.jpg — Cat ✔️
12.jpg — Cat ✔️
13.jpg — Cat ✔️
14.jpg — Cat ✔️
15.jpg — Cat ❌
16.jpg — Dog ✔️
17.jpg — Dog ❌
18.jpg — Cat ✔️
19.jpg — Dog ✔️
20.jpg — Dog ❌
21.jpg — Dog ✔️
22.jpg — Cat ✔️
23.jpg — Cat ✔️
24.jpg — Dog ✔️
25.jpg — Cat ✔️
26.jpg — Dog ✔️
27.jpg — Dog ✔️
28.jpg — Cat ✔️
29.jpg — Dog ✔️
30.jpg — Cat ✔️
31.jpg — Dog ✔️
32.jpg — Dog ❌
33.jpg — Cat ❌
34.jpg — Cat ✔️
35.jpg — Dog ✔️
36.jpg — Dog ✔️
37.jpg — Cat ❌
38.jpg — Dog ✔️
39.jpg — Cat ❌
40.jpg — Dog ✔️

77.5% correct

Conclusions

I noticed that there is some randomness in models. Two rounds of optimizations were different from each other — they were quite close, but not identical.
Since models are initialized with random weights I think the short number of epochs at the commencement of training could significantly impact them. It would be better to train all of the models using a greater number of epochs because such comparison would be more accurate. It would be however also more time-consuming.

References

https://www.kaggle.com/c/dogs-vs-cats
https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765
https://www.thecrazyprogrammer.com/2017/11/best-python-machine-learning-ides.html
https://en.wikipedia.org/wiki/Convolutional_neural_network
https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article
https://keras.io/api/layers/convolution_layers/convolution2d
https://towardsdatascience.com/exploring-activation-functions-for-neural-networks-73498da59b02
https://datascience.stackexchange.com/questions/38327/optimizer-for-convolutional-neural-network
https://www.tensorflow.org/tensorboard/get_started
https://www.baeldung.com/cs/learning-curve-ml
https://shop.primeeducation.com.au/draw-scientific-graphs-correctly-physics/