

For HW2 assignment, I worked on Problem 1.1 from HW1, specifically the implementation of Deep Learning algorithm for image classification. The CNN implementation for binary image classification is done for 4 different datasets prepared from the “Dogs vs Cats” dataset from Kaggle. Datasets for these 4 binary classifiers are consisted of 2 different sizes which are small size:1000 and large size:2000 images and 2 different resolutions which are low resolution:64x64 and high resolution:128x128.

The dataset can be found from this link: <https://www.kaggle.com/c/dogs-vs-cats> . The dataset included corrupted images. Therefore, I performed the data preprocessing in the first part of the source code instead of using built in data preprocessing tools in Keras. The data preprocessing included obtaining equal number of images from each class(cat and dog), removing the corrupted images, resizing the images, shuffling and preparing both the training and test sets. After the data preprocessing, for the CNN implementation I used combination of Tensorflow and Keras in python. I used Jupyter notebook in order to run the code and provided my codes in files with .py extensions.

The software frame worked is from <https://www.tensorflow.org> .

Moreover, I tried three different CNN implementations available online to compare their performances in addition to the performance comparison with respect to the 4 different datasets of different sizes and resolutions mentioned previously.

Input Analysis

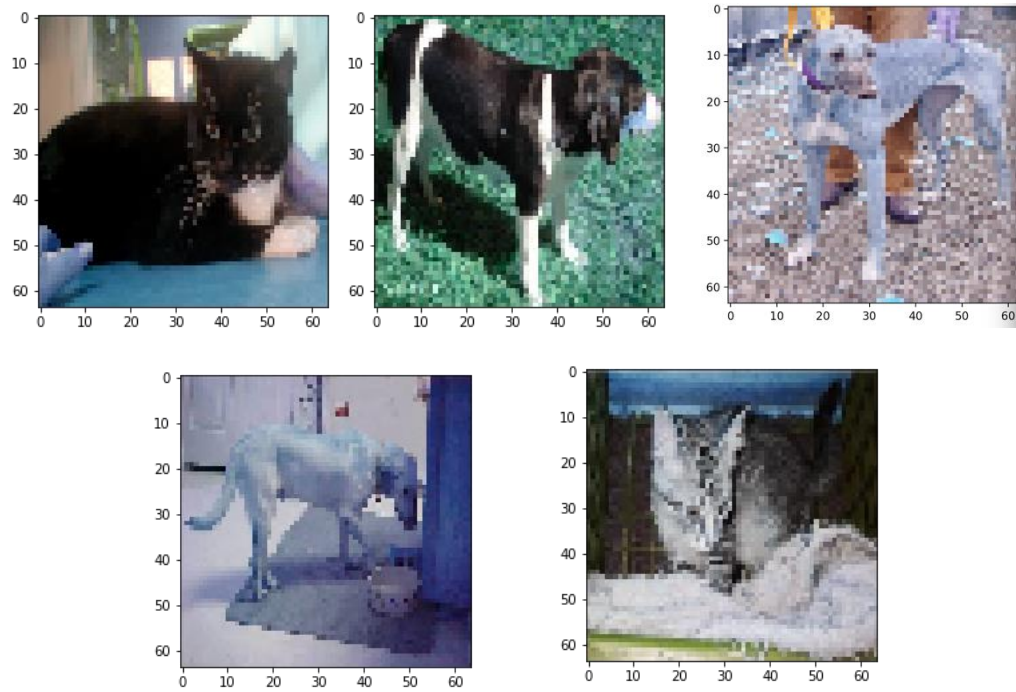
| Original Dataset | |
|--------------------------|--------------|
| Size | 25008 images |
| Resolution | 415x418 |
| Storage Size (per image) | 300.65 KB |
| Storage Size (dataset) | 917.3 MB |

Below, you can see the properties of the 4 different datasets after the preprocessing is done in order to use these datasets for generating 4 binary classifiers.

| 4 Datasets Created | | | | |
|--------------------------|---------|---------|---------|---------|
| | Model 1 | Model 2 | Model 3 | Model 4 |
| Size (# of images) | 2000 | 1000 | 2000 | 1000 |
| Resolution | 64x64 | 64x64 | 128x128 | 128x128 |
| Storage Size (per image) | 12 KB | 12 KB | 49 KB | 49 KB |
| Storage Size (dataset) | 24.6 MB | 12.3 MB | 98.3 MB | 49.2 MB |

You can see the sample images per each class in each of the datasets as follows:

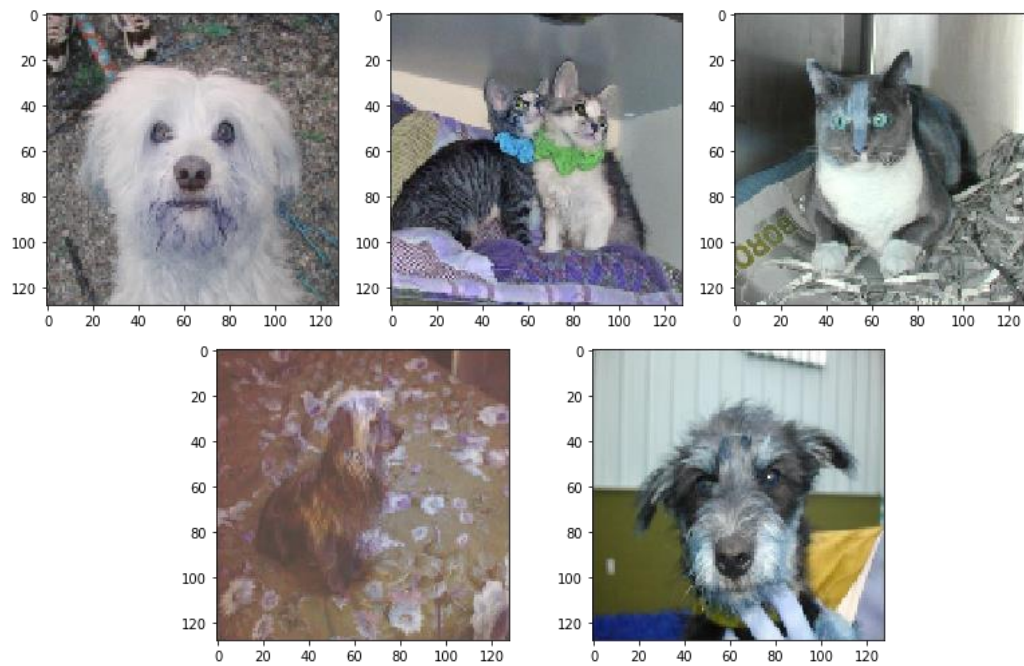
Model 1



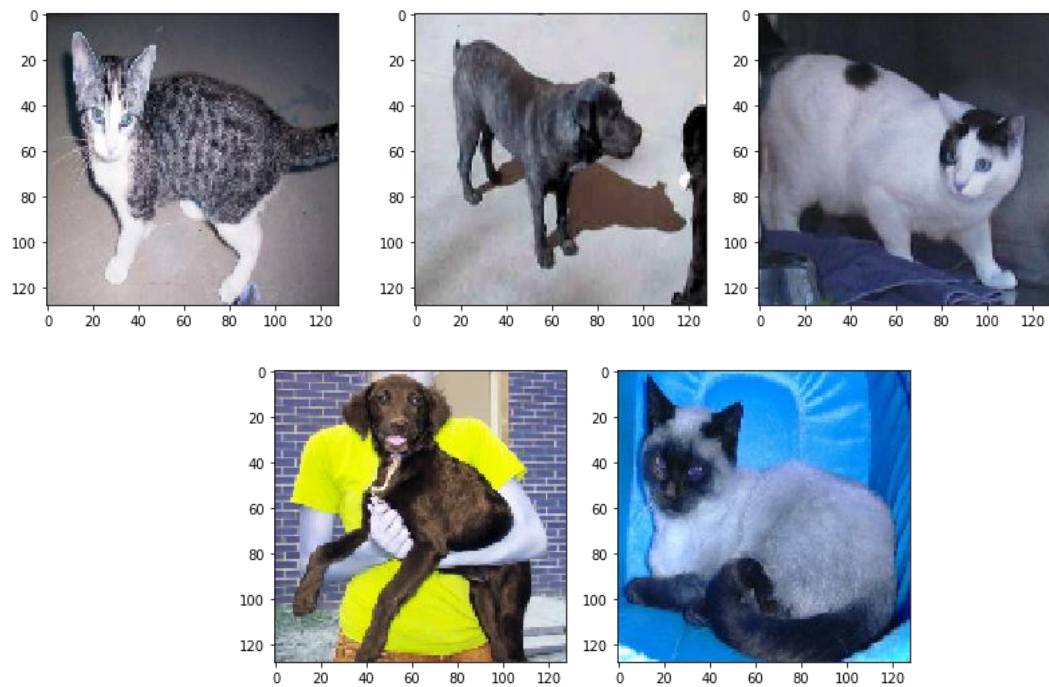
Model 2



Model 3



Model 4



The training and testing data split ratio is 8:2 . Therefore, the datasets which are 1000 in size have a training set of size 800 and testing set of size 200. Similarly, for the dataset with size 2000, the training set size is 1600 and the test set size is 400.

I tried both 2 Layer and 3 Layer CNN for the binary cat-dog classification.

The data is first preprocessed to 64x64x3 or 128x128x3 according to the dataset preference. The preprocessed data first goes through a convolutional layer in order to extract the features. After this step, it continues with a 2D max pooling layer in order to reduce both the size and the number of parameters of data. According to below 2 examples, you can see the implementation of this convolutional layer and 2D max pooling two times in 2 Layer CNN and 3 times in 3 Layer CNN. After these steps data is flattened before the output from previous steps are classified with label. It also goes through a dropout layer where the activations are randomly set to zero during the training so that the overfitting can be avoided. Last step includes two nodes for the classification of either dog or a cat. The output is the probability of each image belonging to a certain class.

For this HW you will see my main observations for the experimental comparison of the four classifiers as Remarks. I initially tried an implementation of 2 layer CNN as the one you see the code blow. I my code you can see that I tried some different models to see the effects on the classification task results. However, for the report I decided to do my analysis on the 3-layer CNN.

2 Layer CNN Code

```
model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=feature.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

3 Layer CNN Code (the one used for the report)

Is the one used for the report and analysis. You can see the screenshot from my coding environment as below.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=feature.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Summary of my 3 CNN Model Used in Analysis

| | Filter_size | Filter number | Neuron size |
|---------------|-------------|---------------|-------------|
| Convolution 1 | 3x3 | 32 | |
| Pool 1 | | | 2x2 |
| Convolution 2 | 3x3 | 32 | |
| Pool 2 | | | 2x2 |
| Convolution 3 | 3x3 | 64 | |

Default Hyperparameters by Tensorflow

| | |
|-----------------|---------|
| Batch size | 32 |
| optimizer | adam |
| # of classes | 2 |
| Filter size | 3x3 |
| Learning rate | 0.0001 |
| Error threshold | < 0.005 |
| activation | ReLU |

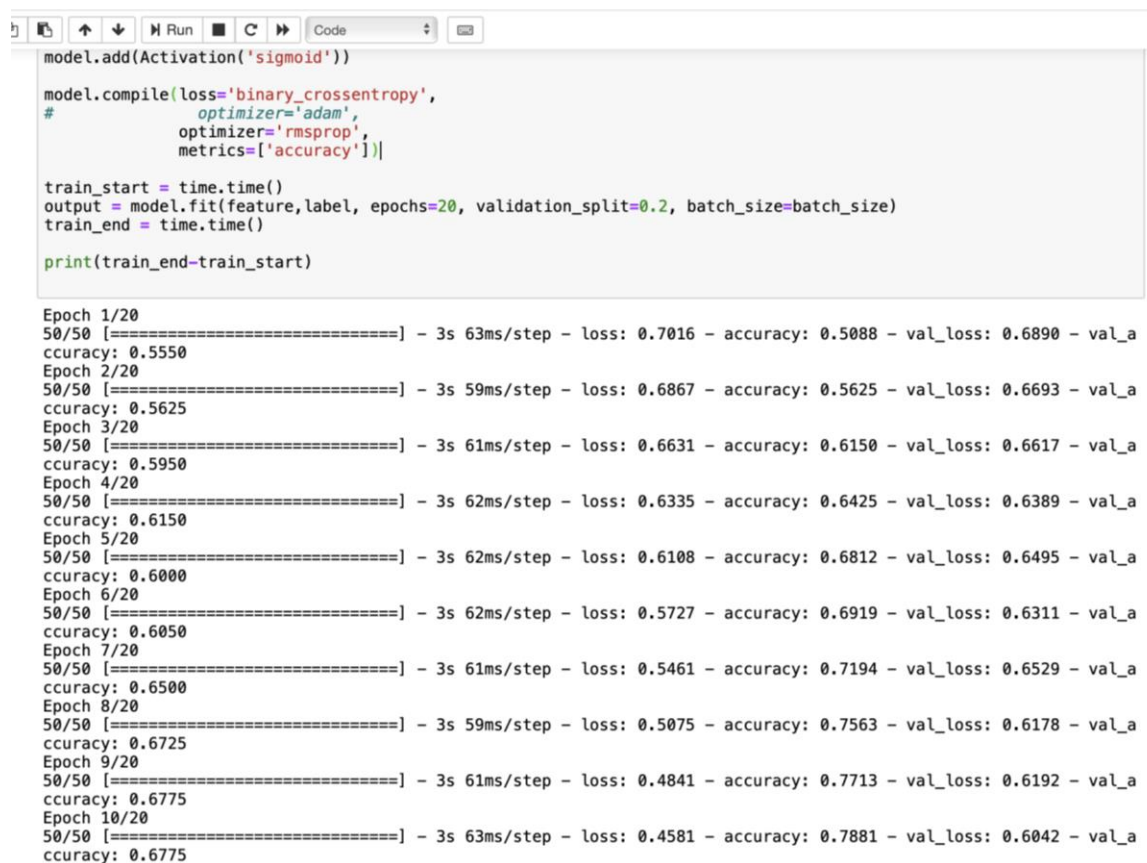
What I used differently for my models can be seen below and some parameters that are not specified below are the same as default above:

| | |
|------------------|-----------|
| Batch size | 32 |
| optimizer | rmsprop |
| Neuron size | 2x2 |
| Filter number | 32 and 64 |
| # of epochs | 20 |
| Validation_split | 20% |

Even though the default optimizer is adam, I am using “rmsprop” optimizer since the accuracy for my 3 CNN model was higher compared to using adam as an optimizer. In addition, I used a number of 20 epochs for all my experiments. Batch size is 32 for each experimentl.

Output Analysis

As can be seen from this screenshot below, when the code is executed we are able to see the metrics such as training loss, test loss, training accuracy, test accuracy and epochs.



```

model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              # optimizer='adam',
              optimizer='rmsprop',
              metrics=['accuracy'])

train_start = time.time()
output = model.fit(feature, label, epochs=20, validation_split=0.2, batch_size=batch_size)
train_end = time.time()

print(train_end-train_start)

```

Epoch 1/20
50/50 [=====] - 3s 63ms/step - loss: 0.7016 - accuracy: 0.5088 - val_loss: 0.6890 - val_a
ccuracy: 0.5550
Epoch 2/20
50/50 [=====] - 3s 59ms/step - loss: 0.6867 - accuracy: 0.5625 - val_loss: 0.6693 - val_a
ccuracy: 0.5625
Epoch 3/20
50/50 [=====] - 3s 61ms/step - loss: 0.6631 - accuracy: 0.6150 - val_loss: 0.6617 - val_a
ccuracy: 0.5950
Epoch 4/20
50/50 [=====] - 3s 62ms/step - loss: 0.6335 - accuracy: 0.6425 - val_loss: 0.6389 - val_a
ccuracy: 0.6150
Epoch 5/20
50/50 [=====] - 3s 62ms/step - loss: 0.6108 - accuracy: 0.6812 - val_loss: 0.6495 - val_a
ccuracy: 0.6000
Epoch 6/20
50/50 [=====] - 3s 62ms/step - loss: 0.5727 - accuracy: 0.6919 - val_loss: 0.6311 - val_a
ccuracy: 0.6050
Epoch 7/20
50/50 [=====] - 3s 61ms/step - loss: 0.5461 - accuracy: 0.7194 - val_loss: 0.6529 - val_a
ccuracy: 0.6500
Epoch 8/20
50/50 [=====] - 3s 59ms/step - loss: 0.5075 - accuracy: 0.7563 - val_loss: 0.6178 - val_a
ccuracy: 0.6725
Epoch 9/20
50/50 [=====] - 3s 61ms/step - loss: 0.4841 - accuracy: 0.7713 - val_loss: 0.6192 - val_a
ccuracy: 0.6775
Epoch 10/20
50/50 [=====] - 3s 63ms/step - loss: 0.4581 - accuracy: 0.7881 - val_loss: 0.6042 - val_a
ccuracy: 0.6775

After this, the code calculates the time for training and testing as well as printing the accuracy and saving the model.

Below, you can see the datasets used for training the models. These four datasets can be found in the “datasets” folder.

| | Size | Resolution |
|---------|------|------------|
| Model 1 | 2000 | 64x64 |
| Model 2 | 1000 | 64x64 |
| Model 3 | 2000 | 128x128 |
| Model 4 | 1000 | 128x128 |

Now, let’s observe the training times as well as accuracies. In order to obtain the results below, the training and the testing tasks of each model is done three times and below you can see the average results. I also included representative plots from these processes where we’ll be able to make some observations (remarks). Different models are saved and can be found in “models” folder.

| | Test Accuracy | Training Time | Testing Time | Storage Size | loss |
|---------|---------------|----------------|--------------|--------------|------|
| Model 1 | 0.91 | 63.62 seconds | 0.66 seconds | 1.6 MB | 0.24 |
| Model 2 | 0.90 | 31.04 seconds | 0.39 seconds | 1.6 MB | 0.26 |
| Model 3 | 0.82 | 225.5 seconds | 2.57 seconds | 6.9 MB | 0.66 |
| Model 4 | 0.92 | 110.15 seconds | 1.37 seconds | 6.9 MB | 0.31 |

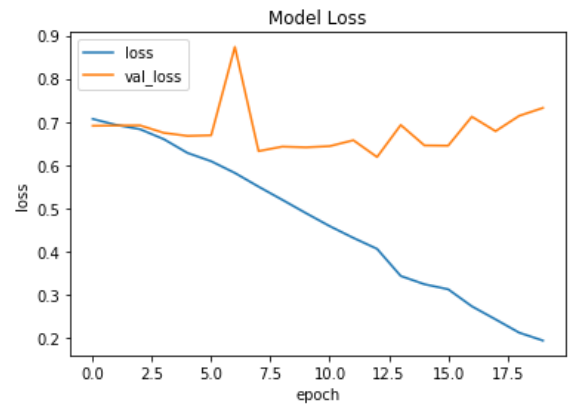
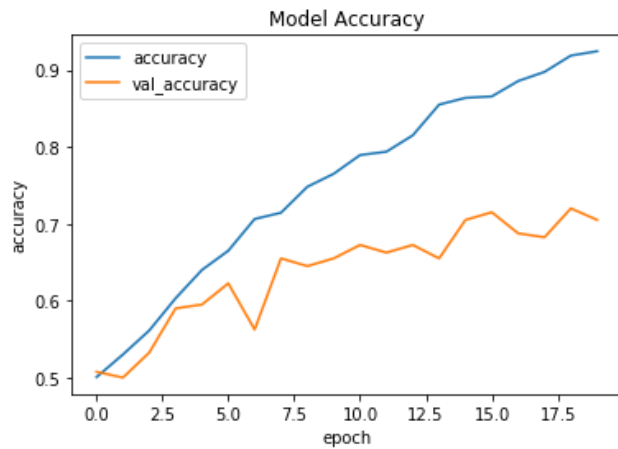
Remark 1

As can be seen from the results above, the training and testing time is directly proportional to the dataset size and resolution. For example, the 128x128 resolution 2000 size dataset has the highest training time while 64x64 resolution 1000 size dataset has the lowest training time.

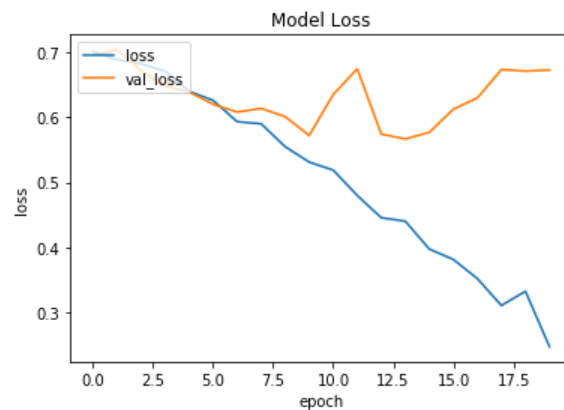
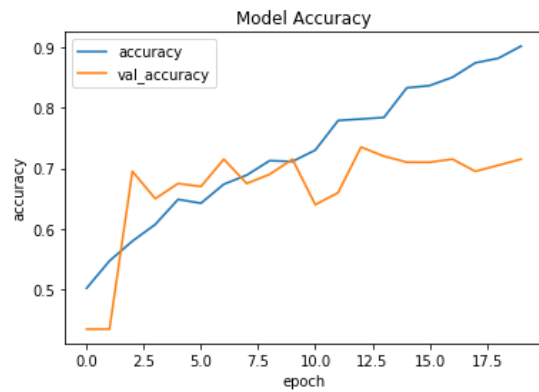
Remark 2

Model storage size increases as the image resolutions of the dataset increases. As can be seen above, model sizes are the highest for the high-resolution data which is 6.9 MB for 128x128 resolution. On the other hand, models trained on 64x64 resolution datasets are 1.6 MB in storage size.

Model 1

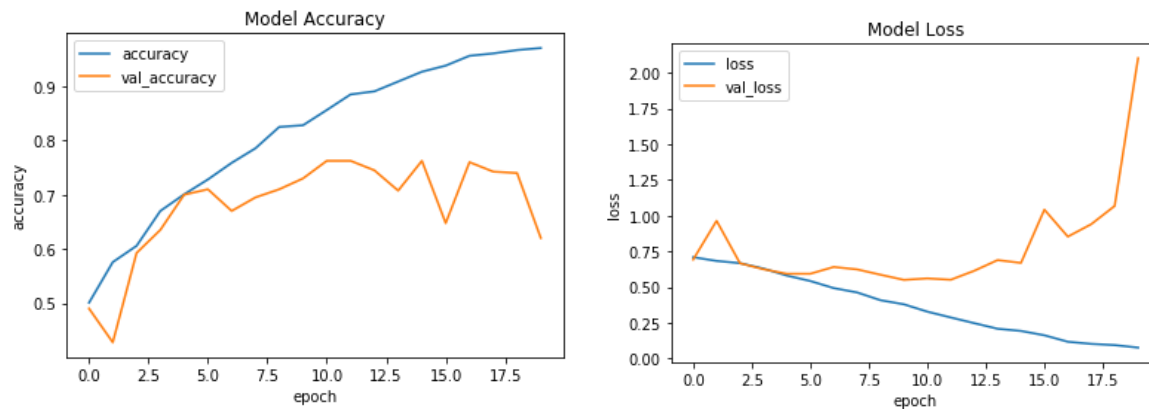


Model 2



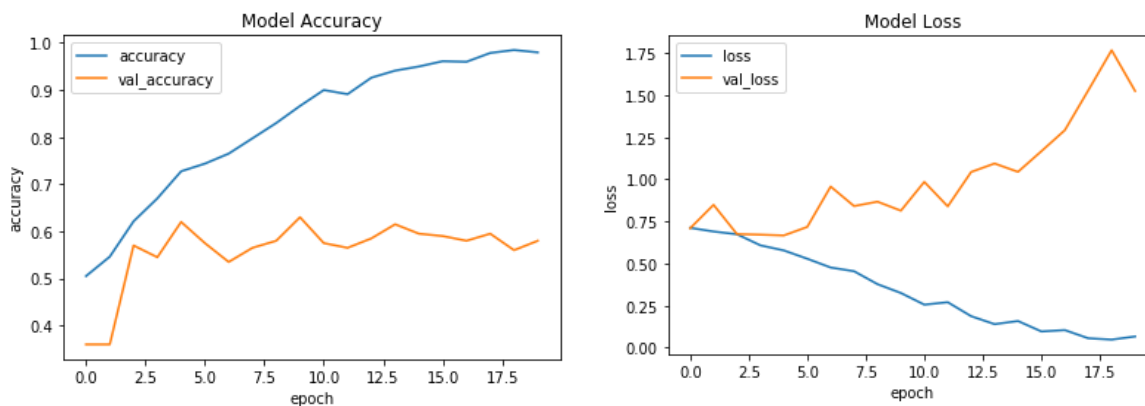
When model 1 and 2 plots are compare, we can say that the model trained on bigger dataset size have a higher increase rate for the validation accuracy as the epoch increases. This can be seen above as the model 1 have a more increasing trend for validation accuracy in the “Model Accuracy” plot compared to the Model 2.

Model 3



Model 3 which was trained on the highest resolution and largest training dataset among our datasets, performs well by means of validation/test accuracy as well as validation/test loss following a very similar rate and pattern in the beginning up to 5 epochs. After that with more epochs we see that the validation performance starts to diverge from the training metrics seen above.

Model 4



Remark 3

The performance of the model will be better when it learns more features via the larger size of training datasets and it can get more information in higher resolution cases as well. For all off the cases the accuracy was above 82%. Since the model is accomplishing a fairly simple classification task which is binary classification, it is expected for the neural net model to perform well as it does in this result.

Remark 4

From the plots model is likely to overfit. For example, we see that the training accuracy is higher than the validation/test set accuracy in most of the cases above.

Remark 5

From trial to trial, the learning process can differ since the weights in the network are randomly initialized or the input image data is being shuffled.

Outlier Test

For the outlier test, 5 images of birds are used. The images are provided in the “outlier” folder. You can see the images below. In order to understand the analysis below: Image number increases from left to right and top to bottom. For the analysis each model is run 3 times on the outlier dataset.



Total Number of Cat and Dog Predictions for Each Outlier Image

| | Bird 1 | | Bird 2 | | Bird 3 | | Bird 4 | | Bird 5 | |
|---------|--------|-----|--------|-----|--------|-----|--------|-----|--------|-----|
| | Cat | Dog | Cat | Dog | Cat | Dog | Cat | Dog | Cat | Dog |
| Model 1 | | 3 | | 3 | | 3 | | 3 | | 3 |
| Model 2 | | 3 | | 3 | | 3 | | 3 | 3 | |
| Model 3 | | 3 | 3 | | 3 | | 3 | | | 3 |
| Model 4 | | 3 | | 3 | 3 | | | 3 | 3 | |

Above, I highlighted the classes that each image is classified as the most in total. For example the first bird image is solely classified as dog for each type of model. All of the models mostly classified the bird images as dogs. Only third and fifth bird images are showing a pattern of equal amounts of dog and cat classification. This might be because the third and fourth bird images have more pointy features. For example the tail and the pointy part on the head of the third bird image is closer to a cat by means of shape since the cat also have pointy ears and a thinner and more obvious tail than a dog. Moreover, the fifth bird image also have pointy features. The other images which are classified heavily as dogs, have one thing in common: bird's round-like and less pointy body is the most common feature, so the NN might have identified the oval like shapes tracing the bird's body shape which is less irregular than the dominant parts of the bird's body in the other pictures. On the other hand, images 3 and 5 have more line like, triangular or sharp-shaped features. I think cats have more pointy features than dogs. Therefore, I am thinking that these might be the reasons for the above classification output.

Test Accuracy

| | Trial 1 | Trial 2 | Trial 3 | Average |
|---------|---------|---------|---------|---------|
| Model 1 | 0 | 0 | 0 | 0 |
| Model 2 | 0.2 | 0.2 | 0.2 | 0.2 |
| Model 3 | 0.6 | 0.6 | 0.6 | 0.6 |
| Model 4 | 0.4 | 0.4 | 0.4 | 0.4 |

Regardless of the input being outlier, the model accuracy to predict this input is higher when the model is trained on a bigger size and higher resolution dataset. This tells me that the model trained on a bigger and higher resolution dataset is able to identify more features than other models since this model learns from more input. However, this still doesn't guarantee the misclassification when an outlier is input such as labeling the bird as a dog or cat. This is because the model can only classify the images as the same type of classes of data it is trained on.

Test Time (seconds)

| | Trial 1 | Trial 2 | Trial 3 | Average |
|---------|---------|---------|---------|---------|
| Model 1 | 0.20 | 0.16 | 0.16 | 0.17 |
| Model 2 | 0.16 | 0.17 | 0.17 | 0.17 |
| Model 3 | 0.16 | 0.2 | 0.19 | 0.18 |
| Model 4 | 0.2 | 0.16 | 0.21 | 0.19 |

There is not really a direct correlation in this output about the test time and the input or the model. We can only infer from this output that when the data resolution of the dataset that the model is trained on is high, the model can take more time when it is used on validation set. However, this needs to be experimented further.

Remark 6

According to the above explanation, the model classifies the outlier image in accordance with its similarities to the features of the each class that it is trained previously on. Therefore, if an outlier image contains similar features (can be color or shape) of for example the A class from the training set instead of B class, then the model will label it as a member of A class. Model can only classify as what it learns from the training set.

Some Options for Stopping Criteria

Lastly, for stopping criteria Keras can stop the training early by the EarlyStopping callback. Different arguments can be used for this. For example, via “monitor” you can specify which performance measure to be monitored as the stopping criterion for the training. Such as using “val_loss” to monitor the loss measure on validation dataset. The objective for this monitoring can be chosen whether “min” as the metric to decrease or “max” as the metric to increase. By default it is set to “auto” which aims to minimize the loss while maximizing the accuracy. Basically the default training convergence condition or the stopping criteria is related to minimizing the loss when maximizing the accuracy. Other criterion such as “baseline” which refers to the threshold for the performance, can be specified as a stopping criteria. This can be applicable in cases where you are familiar learning curve or fine tuning the model and once a certain validation loss is achieved the training can be stopped.

Conclusion

In conclusion, 3-layer CNN is a good algorithm for binary classification task. It has high accuracy levels for predicting the class of the image. Model accuracy and model loss plots tells us about the models performance and gives us the opportunity to compare different CNN K-classifier models which in our case was the binary classification with 3-layer CNN performed on 4 different datasets. When outliers are tested on the model, we see that the algorithm can only predict the class of the outlier in a way that it finds the features of the outlier image similar to the features belong to a certain class such as cats or dogs.

References

<https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/>

<https://cs231n.github.io/convolutional-networks/>

<https://www.youtube.com/watch?v=WvoLTXIjBYU&t=364s>

https://www.tensorflow.org/guide/keras/train_and_evaluate