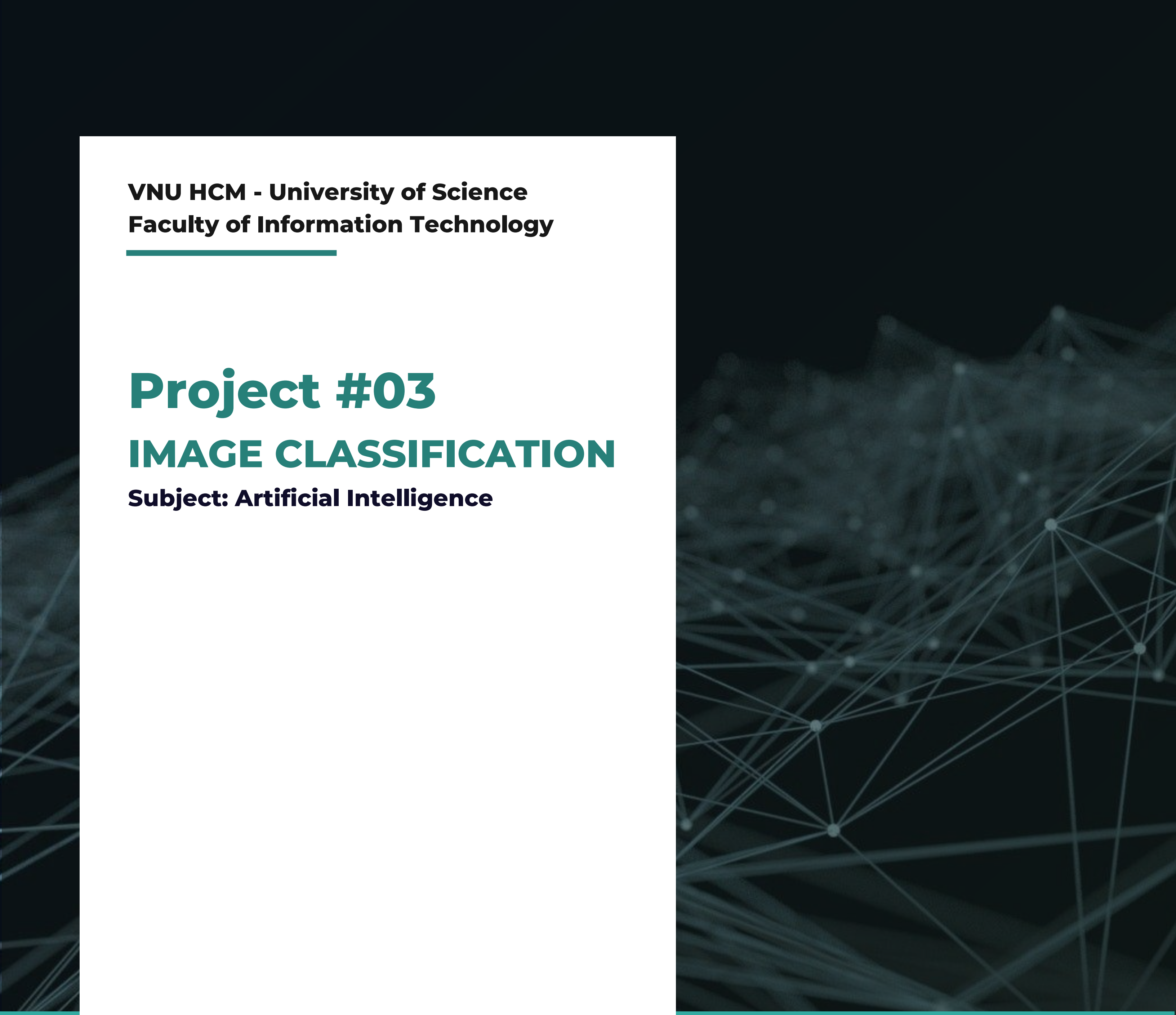


VNU HCM - University of Science
Faculty of Information Technology

Project #03

IMAGE CLASSIFICATION

Subject: Artificial Intelligence

SEPTEMBER 2

Authored by: Alexa
Lê Quang Tấn Long - 19127201
Trần Thanh Tùng - 19127311
Trần Quốc Tuấn - 19127650

Group members

Student ID	Full Name
19127201	Lê Quang Tấn Long
19127311	Trần Thanh Tùng
19127650	Trần Quốc Tuấn

Table of Content

I - DOG VS CAT CLASSIFICATION

1 - Training set, Validation set, Test set Distinction

A major part of machine learning is the construction of supervised algorithms that learn from data, as well as make predictions and classifications on said data through a mathematical model formed by the input data. In order to evaluate how well the model performs, these data are divided into three sets according to three phases: Training, Validation and Testing.

- ### Training set

Training set is the set of examples which is used to train and fit the parameters of the model through a considerable number of epochs, allowing the model to gradually learn more about the characteristics of the data set. This, which is called a learning process, depends on the current algorithm. The training set makes up for most of the data, commonly 70%.

- ### Validation set

During the learning process, the model is unable to determine whether itself is suitable for the data set or not. Therefore, the Validation set, which is separated from the Training set, is required to calculate the model's accuracy and adjust its parameters simultaneously with the training process repeatedly to train the final model. Additionally, another important usage of Validation set is to prevent Overfitting.

- ### Test set

After the process of training and validating the model, another separated set is used to estimate the model's performance. Accounting for 10-20% of the data, this set is called the Test set, whose major difference from the other two sets is that all of its data are unlabeled since it acts as a whole new data set to test the new fully-trained model.

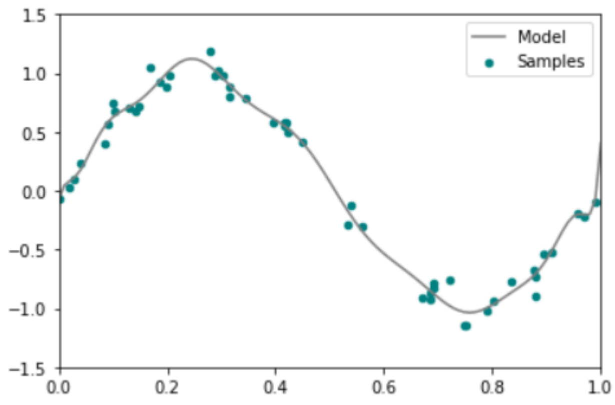
2 - Overfitting & Underfitting Detection & Prevention

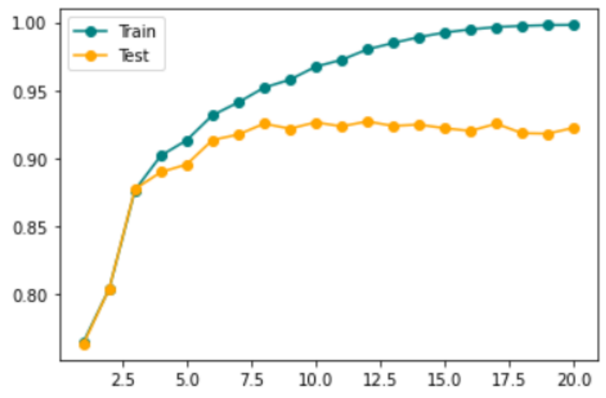
- ### Overfitting

Overfitting is the situation in which the model performs greatly on the training set but poorly on completely new sets, which means the model is only trained to memorize the training data instead of its relations and characteristics that have to be applied to the new data for better performance and accuracy.

- **Detection**
  In order to detect overfitting, firstly, data have to be split into different sets mentioned above. The training set will be used to train the model, while the validation set calculates errors and accuracy in order to compare with the training set's accuracy, and then determine if the model is overfitting or not.
- **Prevention**
  These are some common ways to prevent overfitting.
  - **Cross-validation:** From the initial data, we generate k subsets and iteratively train the model with k-1 subsets while the remaining acts as subsets. This allows training on k different data sets and construction of an accurate model that works well with unseen data.
  - **Train with more data:** Although not suitable for all models, this method may prove useful by using more clean and relevant data, increasing accuracy significantly.
  - **Early stopping:** As we can estimate the performance of the model during each iteration, we can identify which iteration is when overfitting occurs, and stop the process before it passes that point.

For example, the scatter plot on the left side contains a set of data and the accordingly overfitted model, while the one on the right, utilizing decision tree, evaluates the model's accuracy on both sets of Train and Test through increasing tree depths (horizontal axis).
We can easily see that the Test set's accuracy gets further away from the Train set's as the tree depth gets larger.





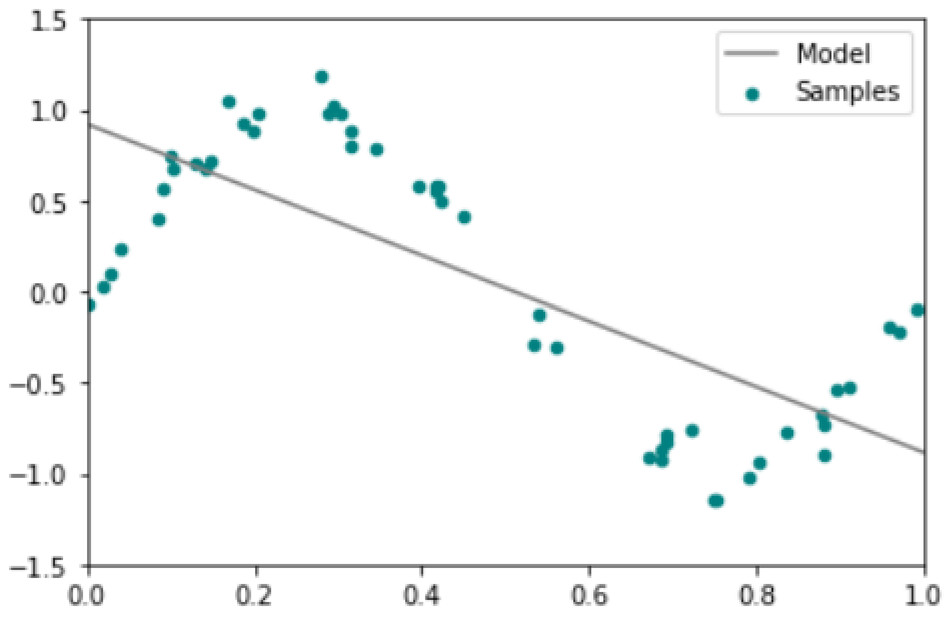
• Underfitting

In contrast to overfitting, underfitting occurs when the model fails to capture the trend and fit the data, leading to inaccuracy and poor performance.

- **Detection**
  During the phase of training and validation, we evaluate the errors of both training set and validation set. If both of them are high, then underfitting occurs.

- **Prevention**
  These are some common ways to prevent underfitting.
  - **Complext model:** The model may be too simple for data to correctly fit. Therefore, another model with more complexity is required.
  - **Increase training time:** Underfitting may occurs since the model isn't given enough time to train, so a simple increase in training time may help.
  - **Decrease regularization:** Regularization is used to reduce the complexity of the given data. However, reducing too much leads to uniform data, making the model incapable of identifying dominant trend and underfitting.

An example of underfitting is shown below, where the model is a simple straight line, in clear contrast to the fact that the curve formed by the data set resembles a parabola. Therefore, the accuracy is obviously low.



3 - Model Components

- *Convolutional Layer*

In the terminology of convolutional neural networks, CNN detects the wanted features from the image data using corresponding filters and extracting the significant features for prediction.

By definition, Convolution is a mathematical operation on two objects to produce an outcome that expresses how the shape of one is modified by the other. To be specific, it is the multiplication performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

With this computation, we utilize Convolutional layers as a stack of filters to detect particular features from the input image and get the result having information about those features. Repeated application of the same filter to input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input image.

In observance of the model, we can conclude that if we use a bigger size of filters, the height and the width will be bigger. Furthermore, the number of filters will determine the depth of the outcome.

We use 2D Convolutional Layer as the first layer in a model. The problem is that images can have different image formats and sizes, hence we need to convert the image to the same format and fixed size. Therefore, we assign the keyword argument input_shape = (150, 150, 3) for 150x150 RGB color pictures. Moreover, the activation parameter "relu" is used to remove any negative pixel values in the feature map.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape = (150, 150, 3), activation = 'relu'))
```

- ***Pooling Layer***

Pooling is the process of merging and its purpose is to reduce the size of the data. By removing some noise in the data and extracting only the significant one, we can reduce overfitting and speed up the computation.

We utilize max pooling which only takes the maximum value inside the box on the left case by sliding a window. Repeating the operation of max pooling on each feature map results in a max pooling layer. Specifically, we use the MaxPooling2D() function to add the pooling layer with a 2x2 filter.

```
model.add(MaxPooling2D((2, 2)))
```

To improve accuracy and reduce overfitting, we add another two convolutional layers, each of which is accompanied with a max pooling layer.

```
model.add(Conv2D(32, (3, 3), input_shape = (150, 150, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), input_shape = (150, 150, 3)))
model.add(MaxPooling2D((2, 2)))
```

- ***Flattening and Fully-connected Layers***

As we are making a classification model, the processed data must be good input to the model. It needs to be in the form of a 1-dimensional linear vector.

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector.

```
model.add(Flatten())
```

The reason for not directly flattening the input image to a single vector is that we want to preserve the spatial structure, not the pixel values of the image, to avoid losing the spatial connection of each pixel with a pixel around it.

This vector that is considered as the input class is connected to the final classification model, which is called a Fully-Connected Layer. In other words, we put all the pixel data in one line and make connections with the final layer. In our model, the final layer is the classification of "cats and dogs".

We build a classifier using this vector as the input class by creating a hidden layer containing the number of nodes in the hidden layer and the activation functions "relu". Besides, we use Dropout to remove inputs to a layer to make nodes in the network generally more robust to the inputs. There will be fully connected layers heading to the layer for sigmoid (a binary case) function.

```
model.add(Dense(64, activation = 'relu'))

model.add(Dropout(0.5))

model.add(Dense(1, activation = 'sigmoid'))
```

4 - Model Compiling

We compile the CNN by choosing the SGD algorithm, with the loss function and the performance metrics. Moreover, binary_crossentropy and categorical_crossentropy are used for binary classification and many classification problems respectively.

```
model.compile(loss = 'binary_crossentropy', optimizer = keras.optimizers.Adam(lr=.0001,
      metrics = ['accuracy'])
```

The training images are divides into batches, and each batch will apply a random image transformation on a random selection of images, to produce more diverse images. Furthermore, we will use the flow_from_directory (directory) method.

```
train_datagen = ImageDataGenerator(rescale = 1. / 255, shear_range = 0.2, zoom_range =
      0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1. / 255)

train_set = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
      target_size = (150, 150), batch_size = 20, class_mode = 'binary')
test_set = train_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
      target_size = (150, 150), batch_size = 20, class_mode = 'binary')
```

5 - Execution

- ***Programming Language:*** Python.
- ***Libraries & Modules:***
  - *numpy:* Arrays & linear algebra handling.
  - *tensorflow.keras.models:* Correct model provision.
  - *tensorflow.keras.layers:* Layer provision.
  - *os path, shutil:* Directory & file handling.
  - *matplotlib:* Graph & image display.
  - *tensorflow.keras.preprocessing:* Image handling & conversion to array.
  - *random:* Random number generator.

- ***Input folders:*** Download the "Dogs vs Cats" dataset available on Kaggle, extract the downloaded zipped folder named "train.zip", then edit the path to the dataset "train" stored in variable "base_dir" in the source code.
    - */train:* A massive set of dog and cat images used to either train or validate the model.
    - */test:* An unseen set of dog and cat images used to test the model's accuracy.
- ***Source code:***
    - *Classification.ipynb:* Jupyter Notebook file for Classification.
- ***Execution:***
    - *Step 1:* Open Jupyter Notebook from Anaconda Navigator.
    - *Step 2:* Press "Upload" button in the upper right corner and locate the source code file.
    - *Step 3:* Scroll down to find and press the uploaded file.
    - *Step 4:* Since Jupyter Notebook file is made up of cells of code, for every cell from the start, press "Run" button or Shift + Enter to execute (Note: You should wait for the current cell to finish executing before running the next cell).
    - *Module installation:* If an error occurs due to an uninstalled library or module, add the line of code below in any cell, available or new, and change <Module> with the name of the missing module. (Note: Some module's exact name may not be correct, so further research is advised).

```
!pip install <Module>
# Example: !pip install keras
```

6 - Summary

The input to Conv2D layer is of fixed size 150 x 150 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with a 2×2 receptive. Spatial pooling is carried out by max-pooling layers, which follow the convolutional layers.
Fully-Connected Layers is connected with a single vector generated by flattening the output of the convolutional layers to create a single long feature vector. The final layer is the sigmoid layer.

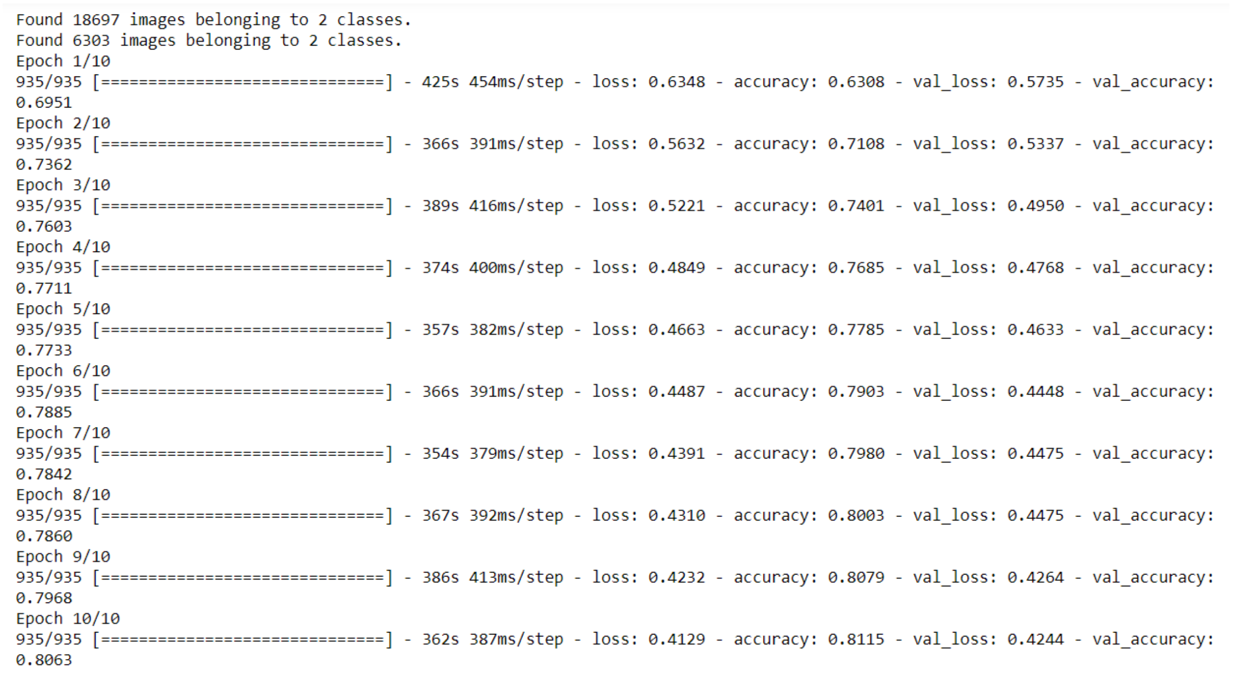
The program follows these steps to build an image classifier from scratch to distinguish cat photos from dog photos:

1. Import the libraries.
2. Define image properties.
3. Prepare dataset for training model.
4. Create the neural net model.
5. Analyzing model.
6. Define callbacks and learning rate.
7. Manage data.
8. Training and validation data generator.
9. Model Training.
10. Test data preparation.
11. Make prediction.

- *Advantages*
  - The model is based on Three-Block VGG Model, which works well in most of cases. Hence, the model's application in practice is appreciated.
  - The programmers know the model to the most basics and can modify it in case there are new requirements.
  - The model is efficient in terms of size and training time as there may be a smaller number of layers as compared to any pretrained model trained for more than one purpose.
  - The model is appropriate for beginners in Computer Vision as it does not require mastery in Deep Learning to use pretrained models.
- *Disadvantages*
  - It will take a large amount of resource, including time and computation power to train big models from scratch.
  - Many pre-trained models are trained for less or more different purposes, hence the model incompatible with them in some cases.
  - The accuracy of the model is above 80%, which is relatively lower than several models.

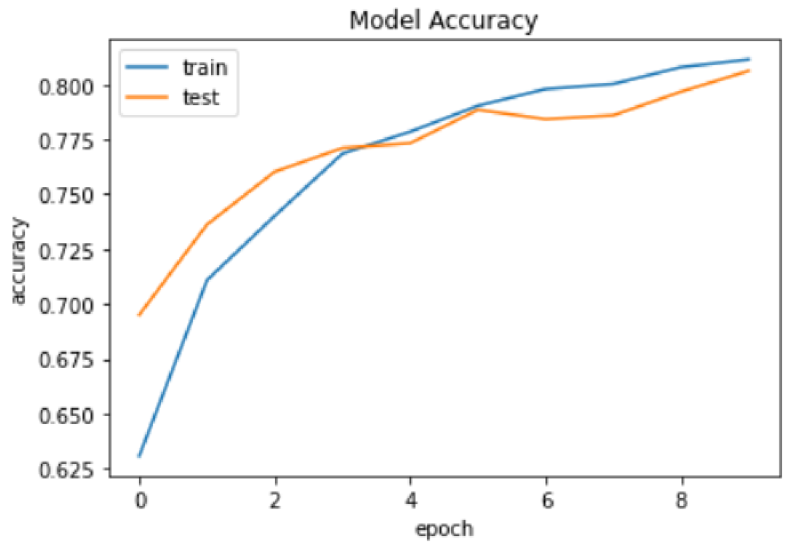
7 - Result & Improvement

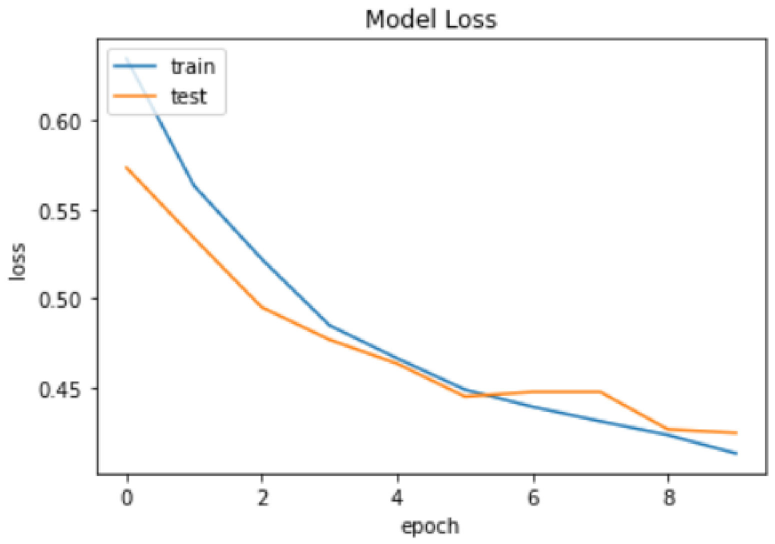
The following picture illustrates the results of the model captured on the code implementation.



```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
Epoch 1/10
935/935 [==============================] - 425s 454ms/step - loss: 0.6348 - accuracy: 0.6308 - val_loss: 0.5735 - val_accuracy:
0.6951
Epoch 2/10
935/935 [==============================] - 366s 391ms/step - loss: 0.5632 - accuracy: 0.7108 - val_loss: 0.5337 - val_accuracy:
0.7362
Epoch 3/10
935/935 [==============================] - 389s 416ms/step - loss: 0.5221 - accuracy: 0.7401 - val_loss: 0.4950 - val_accuracy:
0.7603
Epoch 4/10
935/935 [==============================] - 374s 400ms/step - loss: 0.4849 - accuracy: 0.7685 - val_loss: 0.4768 - val_accuracy:
0.7711
Epoch 5/10
935/935 [==============================] - 357s 382ms/step - loss: 0.4663 - accuracy: 0.7785 - val_loss: 0.4633 - val_accuracy:
0.7733
Epoch 6/10
935/935 [==============================] - 366s 391ms/step - loss: 0.4487 - accuracy: 0.7903 - val_loss: 0.4448 - val_accuracy:
0.7885
Epoch 7/10
935/935 [==============================] - 354s 379ms/step - loss: 0.4391 - accuracy: 0.7980 - val_loss: 0.4475 - val_accuracy:
0.7842
Epoch 8/10
935/935 [==============================] - 367s 392ms/step - loss: 0.4310 - accuracy: 0.8003 - val_loss: 0.4475 - val_accuracy:
0.7860
Epoch 9/10
935/935 [==============================] - 386s 413ms/step - loss: 0.4232 - accuracy: 0.8079 - val_loss: 0.4264 - val_accuracy:
0.7968
Epoch 10/10
935/935 [==============================] - 362s 387ms/step - loss: 0.4129 - accuracy: 0.8115 - val_loss: 0.4244 - val_accuracy:
0.8063
```

In this case, we can see that the model achieved good results with a classification accuracy of about 81% on the holdout test dataset.
To visualize the results, we create figures showing a line plot for the accuracy and another for the loss of the model on both the train and test datasets throughout 10 epochs.





It cannot be denied that the model has overfit the training dataset at around epoch 6 by reviewing the plot of the learning curves. Followings are reasons why there appear to be those results and the trend of overfitting:

- ***Objective causes:***
    - The model is likely to learn the detail and noise in the training dataset to the extent, which negatively impacts the performance of the model on a new dataset.
    - The size of the training dataset used may be not enough.
- ***Subjective causes:***
    - The results imply that further training epochs may result in further improvement of the model. Therefore, the number of epochs is one of siginificant factors that lead to those results of the model.
    - The number of pooling layers and max pooling layers are average, as opposed to that of complex model, being extremely high.

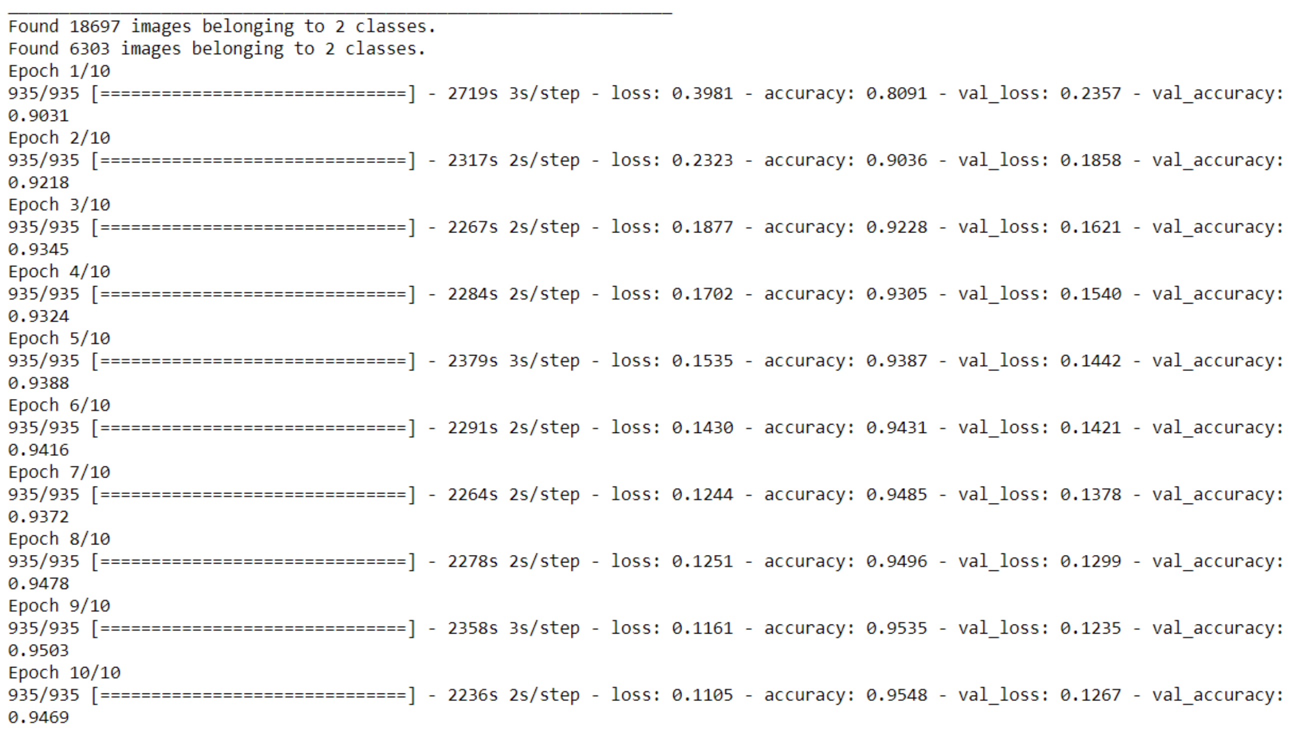
We can improve the results by furthering training epochs or incrementing the number of layers.
It is essential to suggest improved models expected to slow the rate of improvement during training and counter the overfitting of the training dataset. VGG-16 model is an improvement that shoud be mentioned as it is famous for achieving 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

Configurations of VGG-16 follow the generic design and differ only in the depth. The width of convolutional layers is rather small, starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer, until it reaches 512.

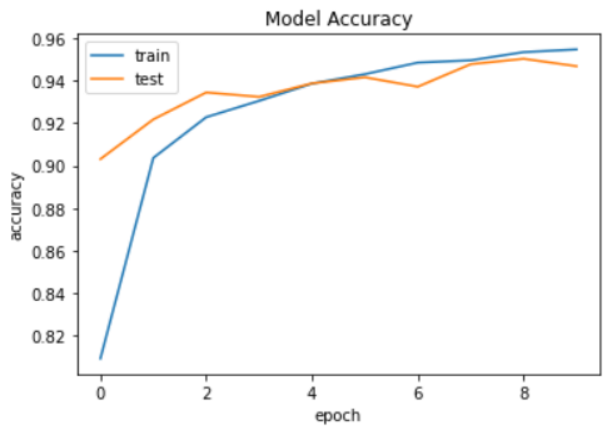
The main difference in VGG-16 model is that the classifier part of the model is made up of fully connected layers and the output layer. After removing the fully connected layers from the output-end of the model, we add the new fully connected layers to interpret the model output and make a prediction. Specifically, the weights of all the convolutional layers fixed during training are reserved, and only train new fully connected layers that will learn to interpret the features and make a binary classification. In other words, the loaded model ends at the last max pooling layer, after which we can manually add a Flatten layer and the new clasifier layers.

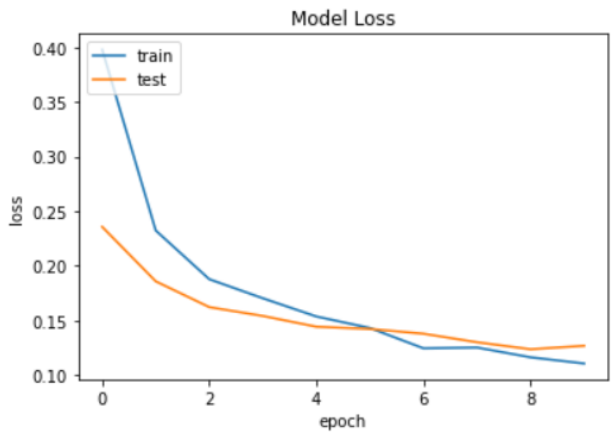
```
pre_trained_model = keras.applications.VGG16(input_shape=(150, 150, 3),
       include_top=False, weights="imagenet")

for layer in pre_trained_model.layers[:15]:
    layer.trainable = False

for layer in pre_trained_model.layers[15:]:
    layer.trainable = True

last_layer = pre_trained_model.get_layer('block5_pool')
last_output = last_layer.output

# Flatten the output layer to 1 dimension
x = GlobalMaxPooling2D()(last_output)
# Add a fully connected layer with 512 hidden units and ReLU activation
x = Dense(512, activation='relu')(x)
# Add a dropout rate of 0.5
x = Dropout(0.5)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

model = Model(pre_trained_model.input, x)

model.compile(loss='binary_crossentropy',
       optimizer=keras.optimizers.SGD(learning_rate=1e-4, momentum=0.9),
       metrics=['accuracy'])

model.summary()

train_datagen = ImageDataGenerator(rescale = 1. / 255, shear_range = 0.2,
       zoom_range = 0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1. / 255)

train_set = train_datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
       target_size = (150, 150), batch_size = 20, class_mode = 'binary')
test_set = train_datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
       target_size = (150, 150), batch_size = 20, class_mode = 'binary')

history = model.fit(train_set, steps_per_epoch = len(train_set), epochs = 10,
       validation_data = test_set, validation_steps = len(test_set))

model.save('model.h5')
```

Following pictures illustrate the results of the improved model captured on the code implementation and the line plot for the accuracy and another for the loss of the model on both the train and test datasets.



```
Found 18697 images belonging to 2 classes.
Found 6303 images belonging to 2 classes.
Epoch 1/10
935/935 [==============================] - 2719s 3s/step - loss: 0.3981 - accuracy: 0.8091 - val_loss: 0.2357 - val_accuracy:
0.9031
Epoch 2/10
935/935 [==============================] - 2317s 2s/step - loss: 0.2323 - accuracy: 0.9036 - val_loss: 0.1858 - val_accuracy:
0.9218
Epoch 3/10
935/935 [==============================] - 2267s 2s/step - loss: 0.1877 - accuracy: 0.9228 - val_loss: 0.1621 - val_accuracy:
0.9345
Epoch 4/10
935/935 [==============================] - 2284s 2s/step - loss: 0.1702 - accuracy: 0.9305 - val_loss: 0.1540 - val_accuracy:
0.9324
Epoch 5/10
935/935 [==============================] - 2379s 3s/step - loss: 0.1535 - accuracy: 0.9387 - val_loss: 0.1442 - val_accuracy:
0.9388
Epoch 6/10
935/935 [==============================] - 2291s 2s/step - loss: 0.1430 - accuracy: 0.9431 - val_loss: 0.1421 - val_accuracy:
0.9416
Epoch 7/10
935/935 [==============================] - 2264s 2s/step - loss: 0.1244 - accuracy: 0.9485 - val_loss: 0.1378 - val_accuracy:
0.9372
Epoch 8/10
935/935 [==============================] - 2278s 2s/step - loss: 0.1251 - accuracy: 0.9496 - val_loss: 0.1299 - val_accuracy:
0.9478
Epoch 9/10
935/935 [==============================] - 2358s 3s/step - loss: 0.1161 - accuracy: 0.9535 - val_loss: 0.1235 - val_accuracy:
0.9503
Epoch 10/10
935/935 [==============================] - 2236s 2s/step - loss: 0.1105 - accuracy: 0.9548 - val_loss: 0.1267 - val_accuracy:
0.9469
```





The model ends up with very impressive results with a classification accuracy of about 96% on the holdout test dataset and effectively counters the overfitting of the training dataset.

8 - References

Train, Test & Validation Sets Explained
https://deeplizard.com/learn/video/Zi-0rlM4RDs
Train, Test & Validation Sets
https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets
The Train, Validation, Test Split and Why You Need It
https://blog.roboflow.com/train-test-split/
ML | Underfitting and Overfitting
https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/
Overfitting and Underfitting With Algorithms in ML
https://www.knowledgehut.com/blog/data-science/overfitting-and-underfitting-in-machine-learning

How to Classify Photos of Dogs and Cats (with 97% accuracy)
https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
Beginner-friendly Project - Cat and Dog classification using CNN
https://www.analyticsvidhya.com/blog/2021/06/beginner-friendly-project-cat-and-dog-classification-using-cnn/
Dogs vs. Cats
https://www.kaggle.com/c/dogs-vs-cats/data
Image Recognition: Dogs vs. Cats! (92%)
https://thedatafrog.com/en/articles/dogs-vs-cats/
Classifying Cats vs Dogs with a Convolutional Neural Network on Kaggle
https://pythonprogramming.net/convolutional-neural-network-kats-vs-dogs-machine-learning-tutorial/
VGG16 - Convolutional Network for Classification and Detection
https://neurohive.io/en/popular-networks/vgg16/

II - DOG VS CAT DETECTION

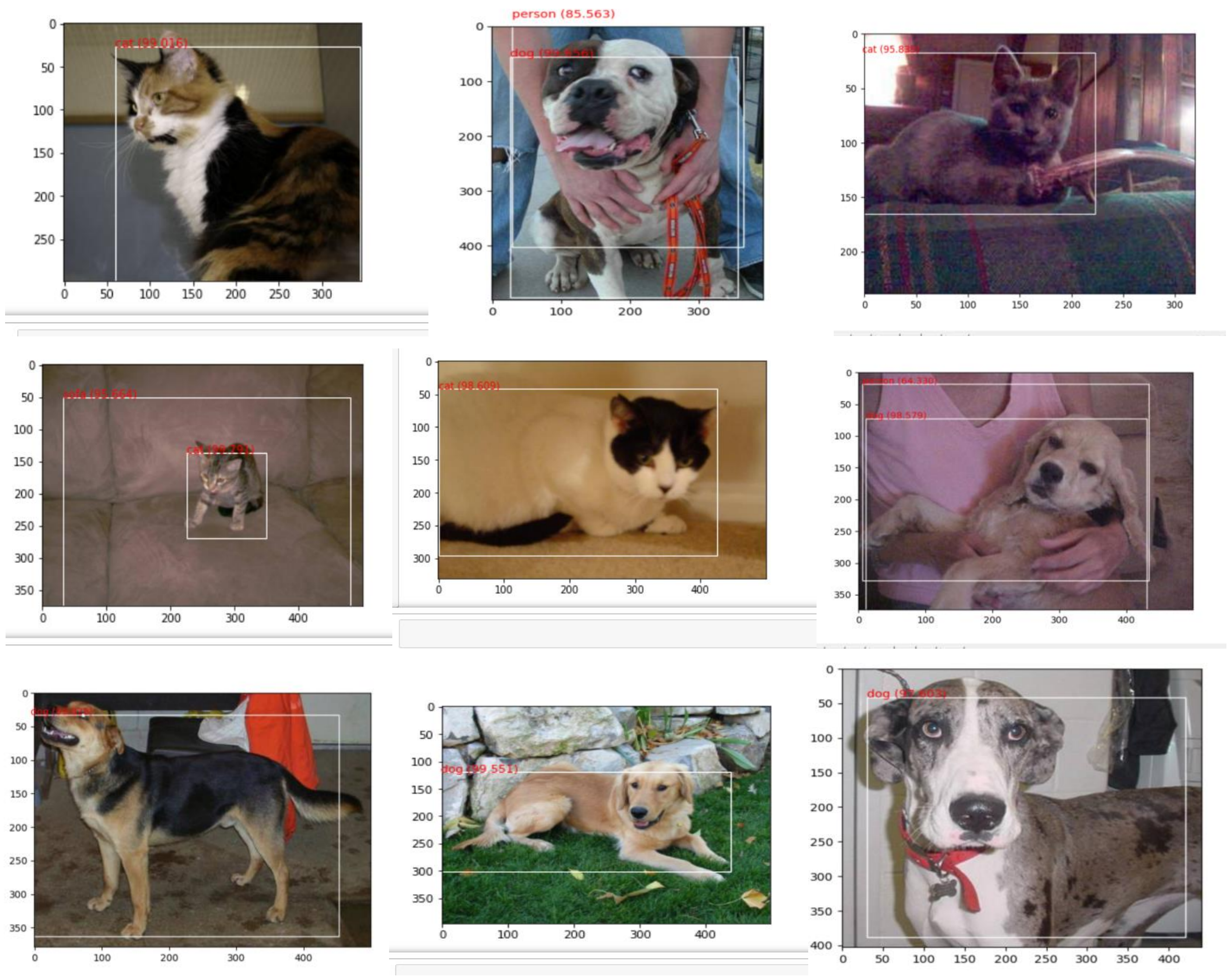
1 - Execution

- ***Programming Language:*** Python.
- ***Libraries & Modules:***
  - *numpy:* Arrays & linear algebra handling.
  - *matplotlib:* Graph display.
  - *tensorflow.keras.models:* Correct model provision.
  - *os:* Directory & file handling.
  - *cv2:* Image handling.
  - *random:* Random number generator.
- ***Prerequisites:***
  - *model.weights:* Download the file from https://bit.ly/3t7Rd2m and place it in the same directory with the source code.
  - *cat_dog.jpg:* Test image.
  - *All necessary files (Backup link):* https://bit.ly/3mPSPwv
- ***Source code:***
  - *Detection.ipynb:* Jupyter Notebook file for Detection.
  - *Detection.py: Python file for Detection.*
- ***Execution:***
  - *Detection.ipynb:* The same as in ***Part I - Dog vs Cat Detection***.
  - *Detection.py:* Follow one of these methods:
    - Command Prompt: Make sure you're in the directory containing the source code. Type "python Detection.py" ("python" can be removed).
    - Visual Studio Code: Press "Run and Debug" button. Make sure the python debugger/compiler extension is installed.

2 - Result & Bad points

Below are the test results on 8 images in nearly 50 of any tested in the dataset. In general, the program's accuracy results are over 70%. The drawn bounding box does a good job with images with sharp contrasts in color, especially bright colors.
When it should be too dark or the tones are quite close together, the drawn bounding box has less accuracy.
For small objects, the program's detection ability is much higher than for large objects.

The program will encounter an error and be difficult to recognize when there are too many objects in the image that appear close to each other. In some cases, the animal's eyes were closed, which greatly affected the recognition (Figure 7).



3 - Solution to improve the program

To further improve the accuracy of a model that requires pre-trained model weights with many close-toed figures, objects that do not clearly show eyes and nose should also be considered. Using inputs consisting of small objects gives the program more accuracy than large objects.

4 - References

Dogspotting: Using Machine Learning to Draw Bounding Boxes around Dogs in Pictures
https://www.rileynwong.com/blog/2019/3/5/dogspotting-using-machine-learning-to-draw-bounding-boxes-around-dogs-in-pictures
Object Detection with Deep Learning and OpenCV
https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/
Introduction to YOLO Algorithm for Object Detection
https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/