



---

# PROJECT 3: IMAGE CLASSIFICATION

---

CSC14003 - Artificial Intelligence



AUGUST 30, 2021

19CLC5  
HCMUS

## Contents

Student Information.....	2
Distinguish 3 types of training set, validation set, test set. Describe how to detect and prevent overfitting and underfitting problem. ....	2
Describe the components of the model you use and why you chose it. Point out the advantages and disadvantages of the model.....	3
Present the results you achieved, explain the reasons for that result. Suggest ideas for improvement. ....	3
1. Result:.....	3
2. Ideas for improvements: .....	5
Code explanation:.....	5
Reference .....	10

## Student Information

Name	Student ID
Phạm Trọng Vinh Khuê	19127038
Nguyễn Tất Trường	19127082
Ngô Văn Anh Kiệt	19127191

## Distinguish 3 types of training set, validation set, test set. Describe how to detect and prevent overfitting and underfitting problem.

- Training set: the data with the correct output, used for fitting the model (find the optimal weight). Contains labeled data, can be augmented to reduce overfitting.
- Validation set: used for evaluating the model's performance after each training. Contains labeled data, should not be augmented and should be as close to the actual real-life data as possible.
- Testing set: used for prediction after the model is trained. This is the actual non-labeled input of the problem and should not be augmented (used to test the error rate \).

### - **Overfitting: This is when the model's prediction ability is constrained to a set of inputs (the noise and the random fluctuation has been learnt by the model).**

#### + Detect:

- When the training accuracy is higher than the validation accuracy => The model is too fitted with the training set so it's ability to predict more generalized data is low.
- When the input is nonsense or invalid (such as noisy images or images that doesn't belong to any category), but the model still gives an answer.

#### + Prevent:

- Augment the training data so the the model can predict more generalized inputs.
- Reduce the learning rate while training so the training accuracy doesn't go too high than the validation accuracy.
- Use Pooling layer to gather only the relevant outputs after each filter.

### - **Underfitting: This is when the model's performance is bad (runs too slow or prediction is inaccurate), the model can neither model the training data nor generalize to new data**

#### + Detect:

- When the training accuracy is much lower than the validation accuracy => Could be due to the training set being more difficult than validation set.
- When the loss value is higher than the accuracy.
- When the loss increases while accuracy decreases.
- When training time/prediction time is too long.

#### + Prevent:

- Don't augment the training set too much.

- Reduce the complexity of the model architecture by making the model "wider" instead of "deeper", in other words, reduce the number of layers while increasing the size of the layers.
- Add more convolutional layers.
- Train the model more by increasing the training epoch and batch size.

Describe the components of the model you use and why you chose it.

Point out the advantages and disadvantages of the model.

The model architecture is based on VGG model, with 4 blocks of convolutional layers, each block comprises of a conv2D layer and a max pooling layer. All convolutional layers have kernel size of 3x3, and pooling layers have kernel size 2x2. The size of convolution layers in the 4 blocks are 32, 64, 128, 256 to filter more complex characteristics as the input goes through the network. The 5th block in the network which is the output block will have an average pooling layer to filter the most noticeable features in the picture, and 3 dense layers (fully-connected layer), one dense layer with size of 512 and one layer with 128 to gradually select the extracted features from the picture and the final dense layer is for calculating sigmoid probability. All layers will use Relu as activation function, except for the final dense layer which will use Sigmoid.

- Model's advantages:

- Overall, gives decent accuracy while only uses basic layers: Convolutional layers, Pooling layers, Fully-connected (Dense) layers.
- Training accuracy is close to validation accuracy => Not much overfitting.
- Loss is still going down while accuracy is still going up at epoch 20 => Still have lots of rooms for improvement.
- Can predict lightly transformed images

- Model's disadvantages:

- Unable to predict noisy images or images that contain both cat and dog or doesn't contain any cat or dog => Give arbitrary result that is either cat or dog.
- Training time is still too long (>900s for an epoch of batch size...).
- The model can not recognize images that does not have the face of cat/dog.

Present the results you achieved, explain the reasons for that result.

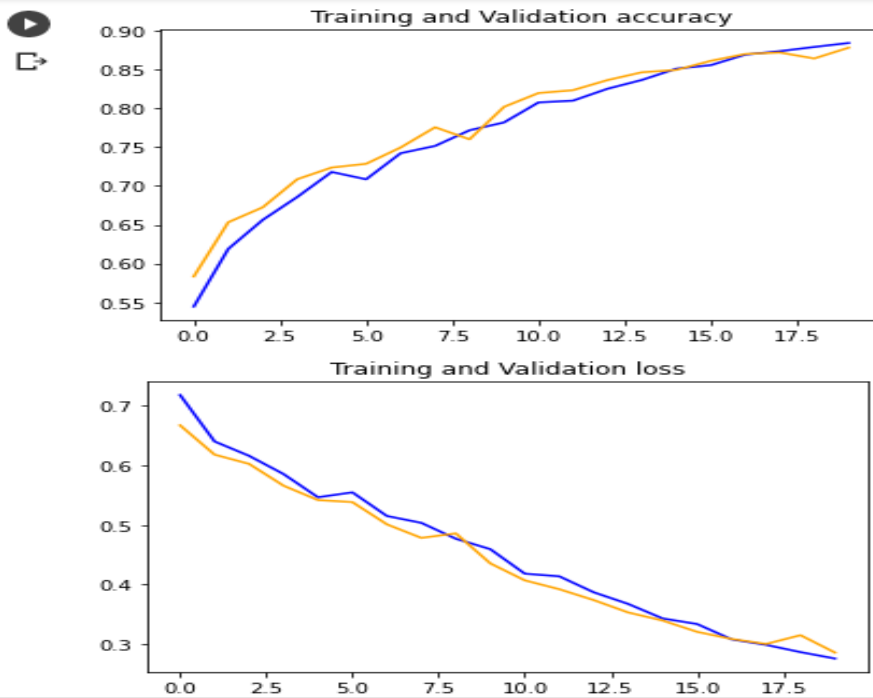
Suggest ideas for improvement.

#### 1. Result:

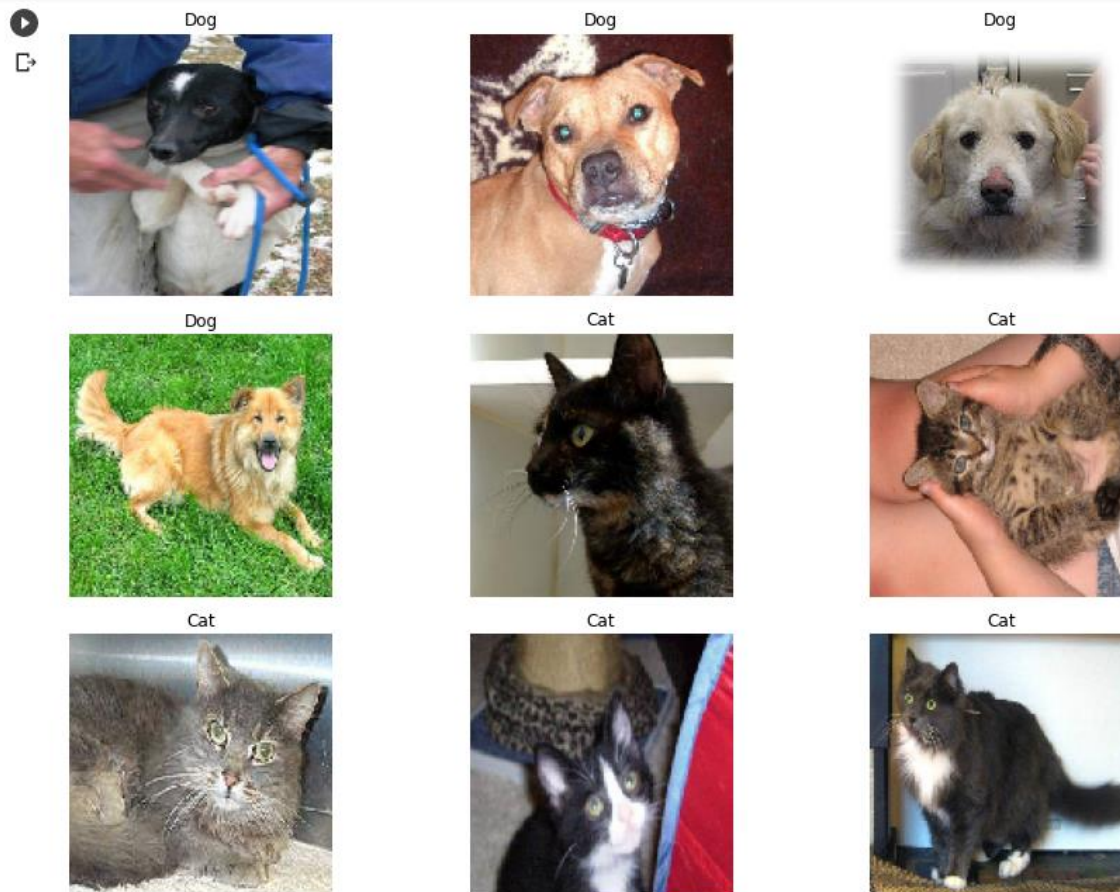
Diagram show the evaluate after training phase:

- Blue is for the training


- Orange is for the validation



Classify 9 images from the test set




## Classify an image from the Internet

 Image file name: dog.jpg  
Found 1 validated image filenames.

Dog



Failure when classify an image that does not contain the face of the object:

 Image file name: cat1.jpg  
Found 1 validated image filenames.

Dog



### 2. Ideas for improvements:

- + Add another set of data containing grainy images, noisy images, images without both cat and dog to the training set. Label them "invalid" and train the model with a 3rd category so it can recognize invalid images.
- + Add more convolutional layers to the model to improve the accuracy.
- + Can use some more types of layers such as: Batch normalization layer, Dropout layer, ...
- + Train the model for more epoch. We can use a callback function to save the training progress and resume later in case the training time is too long.

### Code explanation:

#### Part 1: Get data and extract

Cell 1: all of library that we used => should run this first.

```
# Libraries
from google.colab import drive # Comment this line if not running the notebook on Google Colab
import zipfile
import os
import sys
from PIL import Image
from tqdm import tqdm
import cv2
import shutil
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model, Sequential, Model # Use Sequential model for CNN
# Conv2D = Convolutional Layer, ...Pooling2D = Pooling layer, Dense = Fully-connected layer
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense
from keras.callbacks import ReduceLROnPlateau
```

Cell 2: used to get data from google drive => should comment this cell if you don't run this project in google colab.

```
[ ] drive.mount('/content/drive', force_remount=True) # DON'T RUN THIS CELL IF NOT ON GOOGLE COLAB
# Change drive/MyDrive/Datasets with the path to Datasets on your Drive after mounting
!cp -avr drive/MyDrive/Datasets Datasets
!cp -avr drive/MyDrive/Saving/weights.h5 weights.h5
!cp -avr drive/MyDrive/Saving/model.h5 model.h5

Mounted at /content/drive
'drive/MyDrive/Datasets' -> 'Datasets/Datasets'
'drive/MyDrive/Datasets/dogs_cats' -> 'Datasets/Datasets/dogs_cats'
'drive/MyDrive/Datasets/dogs_cats/test1.zip' -> 'Datasets/Datasets/dogs_cats/test1.zip'
'drive/MyDrive/Datasets/dogs_cats/train.zip' -> 'Datasets/Datasets/dogs_cats/train.zip'
'drive/MyDrive/Saving/weights.h5' -> 'weights.h5'
'drive/MyDrive/Saving/model.h5' -> 'model.h5'
```

Cell 3: unmount the data after extracting

```
[ ] # Unmount drive after extracting. DON'T RUN THIS CELL IF NOT ON GOOGLE COLAB
drive.flush_and_unmount()
```

## Part 2: Some constant that will be used

### ▸ DEFINE CONSTANTS AND FILE DIRECTORIES FOR MODEL TRAINING

```
ROOT_DIR = "/content/Datasets/dogs_cats" # Change this to the path of your Datasets
```

```
[ ] # Extract datasets to root directory
zipfile.ZipFile(os.path.join(ROOT_DIR, "train.zip"), "r").extractall(ROOT_DIR)
zipfile.ZipFile(os.path.join(ROOT_DIR, "test1.zip"), "r").extractall(ROOT_DIR)
```

```
[ ] # DEFINE SOME CONSTANTS FOR PROCESSING DATA
TEST_DIR = os.path.join(ROOT_DIR, "test1")
TRAIN_DIR = os.path.join(ROOT_DIR, "train")
Train_file = "/content/train_data.npy"
Test_file = "/content/test_data.npy"
IMG_SIZE = 128
RANDOM_STATE = 2021
VALIDATION_RATIO = 0.25
SAMPLE_SIZE = 25000
BATCH_SIZE = 250
EPOCH = 20
```

## Part 3: Create dataframes from training set and test set

## ▼ Read lists of image datasets into dataframes

```
[ ] # Get lists of images in 2 folders
train_list = os.listdir(TRAIN_DIR)[0:SAMPLE_SIZE] # SAMPLE SIZE = 25000 means we are using all available images
test_list = os.listdir(TEST_DIR)

[ ] # Return a pandas dataframe with labeled image file list
def create_labeled_df(img_list):
    label = []
    for img in img_list:
        label.append(img.split('.')[0])
    df = pd.DataFrame({
        "file": img_list,
        "type": label
    })
    return df
```

The test dataframe will be sorted after being created.

```
train_df = create_labeled_df(train_list)
# Split original training set into training and validation set
validation_df = train_df.sample(frac=VALIDATION_RATIO)
train_df = train_df.drop(validation_df.index)
# Make testing set without labels. THIS IS ONLY FOR WHEN WE WANT TO DO PREDICTION IN BATCHES
test_df = pd.DataFrame({
    "file": test_list
})
# Sort the testing set so we can output the prediction in the right order
test_df.sort_values(by="file", key=lambda col: [int(x.split('.')[0]) for x in col], inplace=True)
test_df.reset_index(drop=True, inplace=True)
```

## Part 4: Configure the data generators from dataframe

```
[ ] # Augment the training data to prevent overfitting
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 15,
    horizontal_flip = True,
    zoom_range = 0.2,
    shear_range = 0.1,
    fill_mode = 'reflect',
    width_shift_range = 0.1,
    height_shift_range = 0.1,
)

# Validation and testing set can use the same datagen
test_datagen = ImageDataGenerator(
    rescale = 1./255
)

train_sample = train_df.sample(1875)
validation_sample = validation_df.sample(625)
test_sample = test_df.sample(1250)
```

## Part 5: Build and compile the model, we use 3 layers to build this model

```
model = Sequential()
# Block 1
model.add(Conv2D(32, (3,3), padding="same", activation="relu", kernel_initializer="he_uniform", input_shape=(IMG_SIZE, IMG_SIZE, CHANNELS)))
model.add(MaxPooling2D((2,2)))
# Block 2
model.add(Conv2D(64, (3,3), padding="same", activation="relu", kernel_initializer="he_uniform"))
model.add(MaxPooling2D((2,2)))
# Block 3
model.add(Conv2D(128, (3,3), padding="same", activation="relu", kernel_initializer="he_uniform"))
model.add(MaxPooling2D((2,2)))
# Block 4
model.add(Conv2D(256, (3,3), padding="same", activation="relu", kernel_initializer="he_uniform"))
model.add(MaxPooling2D((2,2)))
# Output block
model.add(GlobalAveragePooling2D())
model.add(Dense(512, activation="relu", kernel_initializer="he_uniform"))
model.add(Dense(128, activation="relu", kernel_initializer="he_uniform"))
model.add(Dense(1, activation="sigmoid"))

# Summary the model architecture
model.summary()
```



```

# Compile the model
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])
# A callback function to automatically modify learning rate while training
LR_Reduction = ReduceLROnPlateau(monitor = 'val_accuracy',
                                patience=2,
                                factor=0.35,
                                min_lr = 0.00001,
                                verbose = 1)

callback_func = [LR_Reduction]

```

## Part 6: Training phase, evaluate the loss and accuracy visualization

```

history = model.fit_generator(
    train_generator,
    steps_per_epoch=75, # 18750 images = batch_size * steps
    epochs=EPOCH,
    validation_data=validation_generator,
    validation_steps=25, # 6250 images = batch_size * steps
    verbose=2,
    callbacks=callback_func)

```

```

# Retrieve a list of accuracy results on training and validation data sets for each training epoch
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Retrieve a list of list results on training and validation data sets for each training epoch
loss = history.history['loss']
val_loss = history.history['val_loss']

# Get number of epochs
epochs = range(len(acc))

# Plot training and validation accuracy per epoch
plt.plot(epochs, acc, color='blue', label='train')
plt.plot(epochs, val_acc, color='orange', label='validation')
plt.title('Training and Validation accuracy')
plt.legend()
plt.savefig('accuracy_plot.png')
plt.figure()

# Plot training and validation loss per epoch
plt.plot(epochs, loss, color='blue', label='train')
plt.plot(epochs, val_loss, color='orange', label='validation')
plt.title('Training and Validation loss')
plt.legend()
plt.savefig('loss_plot.png')
plt.figure()

plt.close()

```

## Part 7: Testing

Cell 1: test 9 images from test set.

```

sample_test_gen.reset()
results = model.predict(sample_test_gen)
sample_test_gen.reset()
plt.figure(figsize=(12, 12))
for i in range(9):
    ax = plt.subplot(4, 3, i+1)
    for X_batch in sample_test_gen:
        image = X_batch[0]
        plt.imshow(image)
        label = str
        if results[i] < 0.5: label = "Cat"
        else: label = "Dog"
        ax.set_title(label)
        ax.axis("off")
    break
plt.tight_layout()
plt.show()

```

Cell 2: test a random image from your computer

```

▶ # CONSTANTS FOR USING THE MODEL
IMG_SIZE = 128
# Get user input
model_path = input("Model path: ")
image_dir = input("Image directory: ")
try:
    isfile = os.path.isfile(model_path)
    isDir = os.path.isdir(image_dir)
    if isfile == False or isDir == False: raise FileNotFoundError
except FileNotFoundError:
    print('Error: Input file/directory is not exist')
else:
    image_name = input("Image file name: ")
    # Create dataframe from input path
    img = os.listdir(image_dir)
    img_df = pd.DataFrame({
        "file": img
    })
    # Load the model
    model = load_model(model_path)
    # Data generator for transforming input image
    test_datagen = ImageDataGenerator(
        rescale = 1./255
    )

```

```

image_test = img_df.loc[img_df["file"] == image_name]
image_gen = test_datagen.flow_from_dataframe(
    image_test,
    directory = image_dir,
    x_col = "file",
    y_col = None,
    class_mode = None,
    target_size = (IMG_SIZE, IMG_SIZE),
    batch_size = 1
)
# Predict the image
results = model.predict(image_gen)
# Print output
plt.figure(figsize=(12, 12))
for i in range(1):
    ax = plt.subplot(4, 3, i+1)
    for X_batch in image_gen:
        image = X_batch[0]
        plt.imshow(image)
        label = str
        if results[i] < 0.5: label = "Cat"
        else: label = "Dog"
        ax.set_title(label)
        ax.axis("off")
    break
plt.tight_layout()
plt.show()

```

## Reference

<https://www.analyticsvidhya.com/blog/2021/06/beginner-friendly-project-cat-and-dog-classification-using-cnn/>

<https://developers.google.com/machine-learning/practica/image-classification/exercise-1>

<https://arxiv.org/pdf/1409.1556.pdf>