# Week8_CNN_TensorFlow

April 13, 2024

## 1 Ví dụ 1.

Sử dụng CNN trong thư viện Keras TensorFlow để phân loại ảnh chó mèo.
(Phân loại nhị phân)

### 1.1 Chuẩn bị hệ thống

[ ]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2024-04-11 22:43:04.693133: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-04-11 22:43:04.693301: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-04-11 22:43:04.807421: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-04-11 22:43:05.010534: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-04-11 22:43:08.555349: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

```
2.15.0
```

[ ]:
```
import jupyter
print(jupyter.__version__)
```

```
1.0.0
```

[ ]:
```
import scipy
print(scipy.__version__)
```

```
1.9.3
```

### 1.2 Chuẩn bị dữ liệu

Hiển thị ra cấu trúc file, folder lưu trữ dữ liệu

[ ]:
```
import os

base_dir = "data/cat_dog_panda"
# Change the base_dir to where you put dataset
print("Contents of base directory:")
print(os.listdir(base_dir))

print("\nContents of train directory:")
print(os.listdir(f"{base_dir}/train"))

print("\nContents of validation directory:")
print(os.listdir(f"{base_dir}/validation"))
```

```
Contents of base directory:
['train', 'validation']

Contents of train directory:
['cats', 'dogs', 'panda']

Contents of validation directory:
['cats', 'dogs', 'panda']
```

Tham chiếu phân lớp vào tập train và validation (tên thư mục là tên phân lớp)

[ ]:
```
train_dir = os.path.join(base_dir, "train")
validation_dir = os.path.join(base_dir, "validation")

# Directory with training cat/dog pictures
train_cats_dir = os.path.join(train_dir, "cats")
train_dogs_dir = os.path.join(train_dir, "dogs")

# Directory with validation cat/dog pictures
validation_cats_dir = os.path.join(validation_dir, "cats")
validation_dogs_dir = os.path.join(validation_dir, "dogs")

print("\nContents of train directory:")
print(os.listdir(f"{base_dir}/train"))

print("\nContents of validation directory:")
print(os.listdir(f"{base_dir}/validation"))
```

```
Contents of train directory:
['cats', 'dogs', 'panda']

Contents of validation directory:
['cats', 'dogs', 'panda']
```

Lấy danh sách tên file từng loại ảnh train/validation và thống kê số data có cho mỗi phân lớp

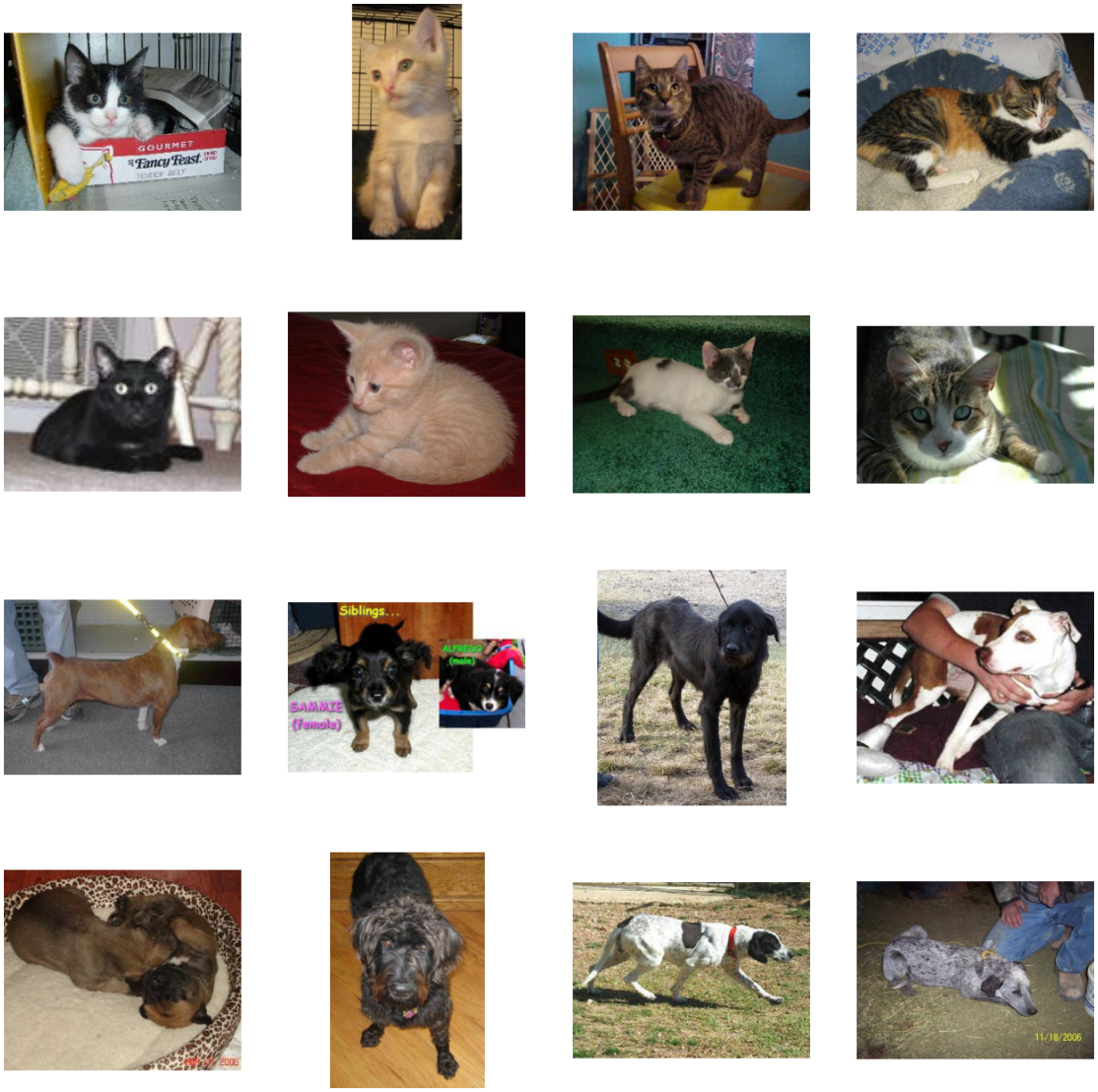
[ ]:
```
train_cat_fnames = os.listdir(train_cats_dir)
train_dog_fnames = os.listdir(train_dogs_dir)

print(train_cat_fnames[:10])
print(train_dog_fnames[:10])

print("total training cat images :", len(os.listdir(train_cats_dir)))
print("total training dog images :", len(os.listdir(train_dogs_dir)))

print("total validation cat images :", len(os.listdir(validation_cats_dir)))
print("total validation dog images :", len(os.listdir(validation_dogs_dir)))
```

```
['cats_00306.jpg', 'cats_00612.jpg', 'cats_00001.jpg', 'cats_00002.jpg',
'cats_00003.jpg', 'cats_00004.jpg', 'cats_00005.jpg', 'cats_00006.jpg',
'cats_00007.jpg', 'cats_00008.jpg']
['dogs_00306.jpg', 'dogs_00612.jpg', 'dogs_00001.jpg', 'dogs_00002.jpg',
'dogs_00003.jpg', 'dogs_00004.jpg', 'dogs_00005.jpg', 'dogs_00006.jpg',
'dogs_00007.jpg', 'dogs_00008.jpg']
total training cat images : 1000
total training dog images : 1000
total validation cat images : 1000
total validation dog images : 1000
```

In 1 số ảnh mỗi loại để kiểm tra

[ ]:
```
%matplotlib inline

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

pic_index = 0 # Index for iterating over images
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)

pic_index += 8
```

```
next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[ pic_index-8:pic_index]
               ]

next_dog_pix = [os.path.join(train_dogs_dir, fname)
                for fname in train_dog_fnames[ pic_index-8:pic_index]
               ]

for i, img_path in enumerate(next_cat_pix+next_dog_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```




### 1.3 Khởi tạo CNN

Kiến trúc mạng CNN sẽ xây dựng:
- 3 tầng tích chập (convolution layers) kết hợp với Pooling (MaxPooling layers), kích thước mặt nạ tích chập là 3x3, kích thước pooling là 2x2.
- Sau mỗi tầng tích chập, hàm kích hoạt được sử dụng là ReLU.
- Tiếp theo là một tầng Flatten và tầng tiếp theo là Full Connection (duỗi các kết quả đầu ra thành vector).
- Cuối cùng là một tầng phân loại ở đầu ra, sử dụng hàm Sigmoid (logistic – vì đây chỉ có 02 lớp).

[ ]:
```
import tensorflow as tf

model = tf.keras.models.Sequential(
```

```
    [
        # Note the input shape is the desired size of the image 150x150 with 3␣
 ↪bytes color
        tf.keras.layers.Conv2D(
            16, (3, 3), activation="relu", input_shape=(150, 150, 3)
        ),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(2, 2),
        # Flatten the results to feed into a DNN
        tf.keras.layers.Flatten(),
        # 512 neuron hidden layer
        tf.keras.layers.Dense(512, activation="relu"),
        # Only 1 output neuron.
        # It will contain a value from 0-1
        # where 0 for 1 class ('cats')
        # and 1 for the other ('dogs')
        tf.keras.layers.Dense(1, activation="sigmoid"),
    ]
)
```

[ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 16)      448

 max_pooling2d (MaxPooling2  (None, 74, 74, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 64)        0
 g2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 512)               9470464
```

```
 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 9494561 (36.22 MB)
Trainable params: 9494561 (36.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

### 1.4 Huấn luyện CNN

Thiết lập phương pháp giải bài toán tối ưu. Ở đây ta dùng RMSprop thay cho SGD (do có thể tự động chọn tham số học)

[ ]:
```
from tensorflow.keras.optimizers import RMSprop

model.compile(
    optimizer=RMSprop(learning_rate=0.001),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

Tiếp theo chúng ta chuẩn hóa dữ liệu bằng các đưa cường độ pixel về khoảng [0, 1], chỉnh kích thước ảnh về 150x150 và điều hướng các thư mục chứa dữ liệu training

[ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator(rescale=1.0 / 255.0)
test_datagen = ImageDataGenerator(rescale=1.0 / 255.0)

# Define the list of classes you want to use (excluding the folder you want to␣
 ↪exclude)
classes = [class_name for class_name in os.listdir(train_dir) if class_name !=␣
 ↪'panda']

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_directory(
    train_dir, batch_size=20, class_mode="binary", target_size=(150, 150),␣
 ↪classes=classes
)
# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
validation_generator = test_datagen.flow_from_directory(
    validation_dir, batch_size=20, class_mode="binary", target_size=(150, 150),␣
 ↪classes=classes
```

```
)
```

```
Found 2000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

Lệnh huấn luyện mô hình

[ ]:
```
history = model.fit(
    train_generator,
    steps_per_epoch=100,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=2,
)
```

```
Epoch 1/15
100/100 - 75s - loss: 0.7548 - accuracy: 0.5050 - val_loss: 0.6803 -
val_accuracy: 0.5000 - 75s/epoch - 752ms/step
Epoch 2/15
100/100 - 66s - loss: 0.6834 - accuracy: 0.5905 - val_loss: 0.6188 -
val_accuracy: 0.6710 - 66s/epoch - 660ms/step
Epoch 3/15
100/100 - 78s - loss: 0.6045 - accuracy: 0.6860 - val_loss: 0.5169 -
val_accuracy: 0.7330 - 78s/epoch - 782ms/step
Epoch 4/15
100/100 - 78s - loss: 0.5488 - accuracy: 0.7105 - val_loss: 0.4764 -
val_accuracy: 0.8000 - 78s/epoch - 784ms/step
Epoch 5/15
100/100 - 73s - loss: 0.5004 - accuracy: 0.7465 - val_loss: 0.4285 -
val_accuracy: 0.8000 - 73s/epoch - 726ms/step
Epoch 6/15
100/100 - 68s - loss: 0.4361 - accuracy: 0.7975 - val_loss: 0.3229 -
val_accuracy: 0.8590 - 68s/epoch - 684ms/step
Epoch 7/15
100/100 - 77s - loss: 0.3592 - accuracy: 0.8500 - val_loss: 0.2439 -
val_accuracy: 0.9240 - 77s/epoch - 766ms/step
Epoch 8/15
100/100 - 75s - loss: 0.2741 - accuracy: 0.8880 - val_loss: 0.1419 -
val_accuracy: 0.9550 - 75s/epoch - 751ms/step
Epoch 9/15
100/100 - 70s - loss: 0.1754 - accuracy: 0.9420 - val_loss: 0.1212 -
val_accuracy: 0.9600 - 70s/epoch - 704ms/step
Epoch 10/15
100/100 - 88s - loss: 0.1155 - accuracy: 0.9605 - val_loss: 0.2472 -
val_accuracy: 0.8980 - 88s/epoch - 883ms/step
Epoch 11/15
100/100 - 76s - loss: 0.0671 - accuracy: 0.9775 - val_loss: 0.1479 -
val_accuracy: 0.9330 - 76s/epoch - 756ms/step
```

```
Epoch 12/15
100/100 - 63s - loss: 0.0545 - accuracy: 0.9820 - val_loss: 0.0369 -
val_accuracy: 0.9880 - 63s/epoch - 629ms/step
Epoch 13/15
100/100 - 64s - loss: 0.0723 - accuracy: 0.9830 - val_loss: 0.0067 -
val_accuracy: 0.9990 - 64s/epoch - 636ms/step
Epoch 14/15
100/100 - 67s - loss: 0.0501 - accuracy: 0.9860 - val_loss: 0.0357 -
val_accuracy: 0.9900 - 67s/epoch - 666ms/step
Epoch 15/15
100/100 - 68s - loss: 0.0095 - accuracy: 0.9970 - val_loss: 0.0015 -
val_accuracy: 0.9990 - 68s/epoch - 683ms/step
```

[ ]:
```
model.save("best_model_cat_dog_no_dropout.keras")
```

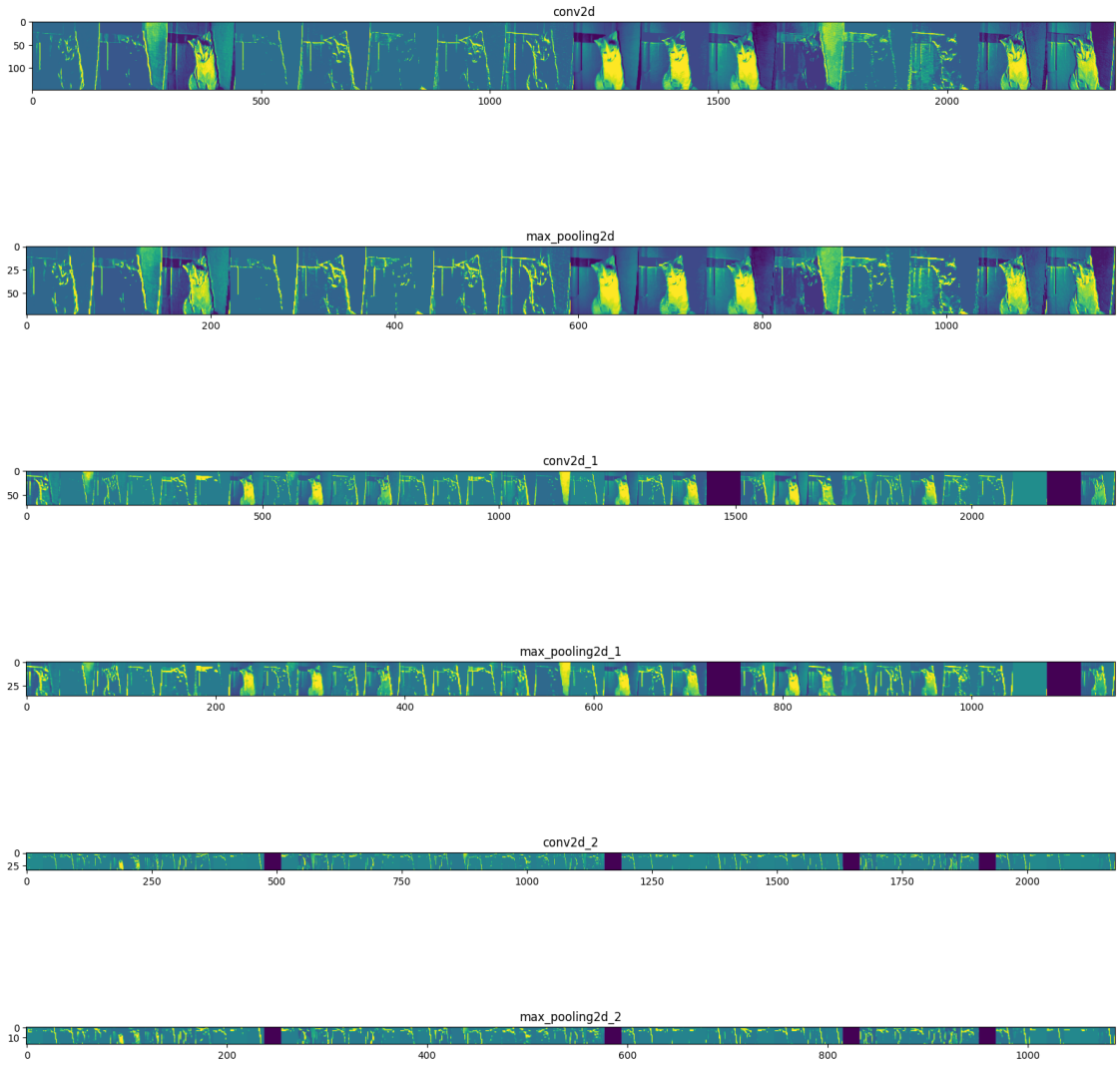
Xem xét 1 số kết quả, chúng ta sẽ in ra kết quả sau mỗi tầng ("ảnh" đầu ra sau mỗi tầng convolution và maxpool) để hiểu thêm cách hoạt động của CNN để cho ra dãy so sánh được ở tầng flatten.

[ ]:
```
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# Define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model
successive_outputs = [layer.output for layer in model.layers]
visualization_model = tf.keras.models.Model(
    inputs=model.input, outputs=successive_outputs
)

# Prepare a random input image from the training set.
cat_img_files = [os.path.join(train_cats_dir, f) for f in train_cat_fnames]
dog_img_files = [os.path.join(train_dogs_dir, f) for f in train_dog_fnames]
img_path = random.choice(cat_img_files + dog_img_files)
img = load_img(img_path, target_size=(150, 150))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Scale by 1/255
x /= 255.0

# Run the image through the network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so we can have them as part of our plot
layer_names = [layer.name for layer in model.layers]

# Display the representations
```

```
for layer_name, feature_map in zip(layer_names, successive_feature_maps):

    if len(feature_map.shape) == 4:

        # ----------------------------------------
        # Just do this for the conv / maxpool layers, not the fully-connected␣
 ↪layers
        # ----------------------------------------
        n_features = feature_map.shape[-1]  # number of features in the feature␣
 ↪map
        size = feature_map.shape[1]  # feature map shape (1, size, size,␣
 ↪n_features)

        # Tile the images in this matrix
        display_grid = np.zeros((size, size * n_features))

        # -------------------------------------------------
        # Postprocess the feature to be visually palatable
        # -------------------------------------------------
        for i in range(n_features):
            x = feature_map[0, :, :, i]
            x -= x.mean()
            x /= x.std()
            x *= 64
            x += 128
            x = np.clip(x, 0, 255).astype("uint8")
            display_grid[:, i * size : (i + 1) * size] = (
                x  # Tile each filter into a horizontal grid
            )

        # -----------------
        # Display the grid
        # -----------------
        scale = 20.0 / n_features
        plt.figure(figsize=(scale * n_features, scale))
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect="auto", cmap="viridis")
```

```
1/1 [==============================] - 0s 204ms/step
```

```
/tmp/ipykernel_22463/1428360066.py:50: RuntimeWarning: invalid value encountered
in divide
  x /= x.std()
/tmp/ipykernel_22463/1428360066.py:53: RuntimeWarning: invalid value encountered
in cast
  x = np.clip(x, 0, 255).astype("uint8")
```




## 1.5 Thử nghiệm trên tập validation

Thử test với 1 ảnh bất kỳ

[ ]:
```
import numpy as np

# from google.colab import files
from keras.preprocessing import image

# uploaded=files.upload()

# for fn in uploaded.keys():
fn = "cats_00001.jpg"  # change it to your image file
# predicting images
```

```
path = "/mnt/DataK/Univer/UniSubject/_3th_year/_2nd_term/3ii_ML/Lec_Ass/Week8/
  ↪data/cat_dog_panda/validation/cats/cats_00001.jpg"  # change it to your image
img = image.load_img(path, target_size=(150, 150))

x = image.img_to_array(img)
x /= 255
x = np.expand_dims(x, axis=0)
images = np.vstack([x])

classes = model.predict(images, batch_size=10)

print(classes[0])

if classes[0] > 0.5:
    print(fn + " is a dog")
else:
    print(fn + " is a cat")
```

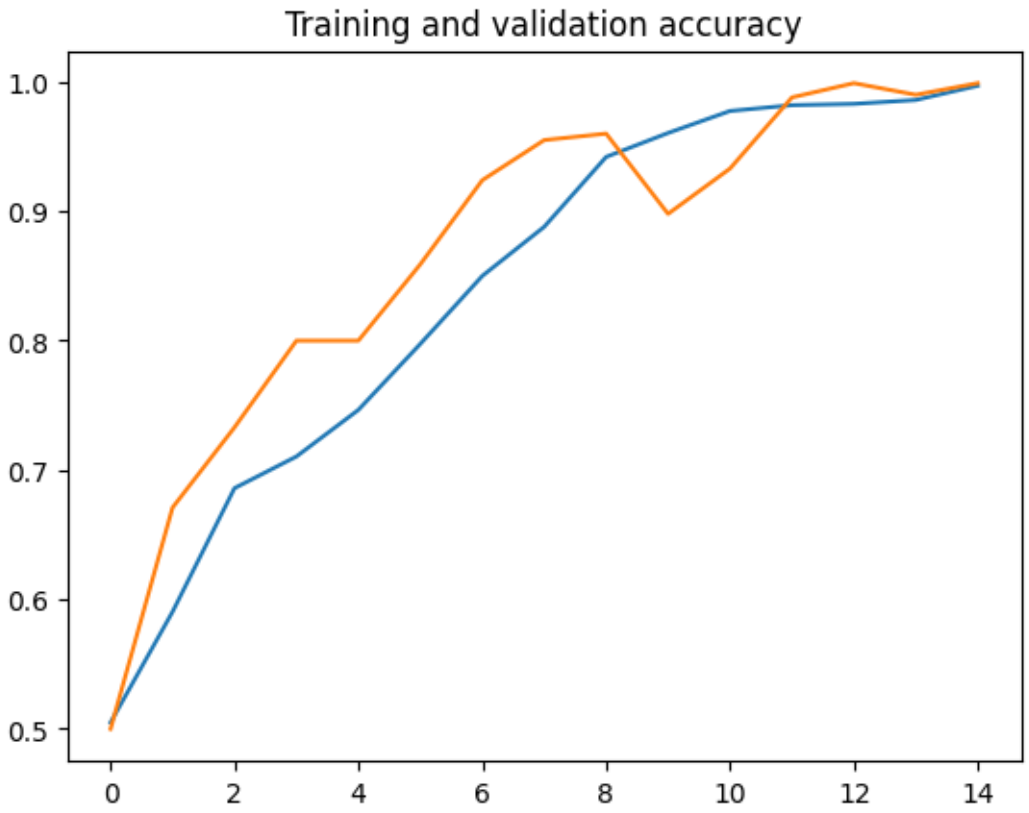
```
1/1 [==============================] - 0s 237ms/step
[7.743724e-05]
cats_00001.jpg is a cat
```

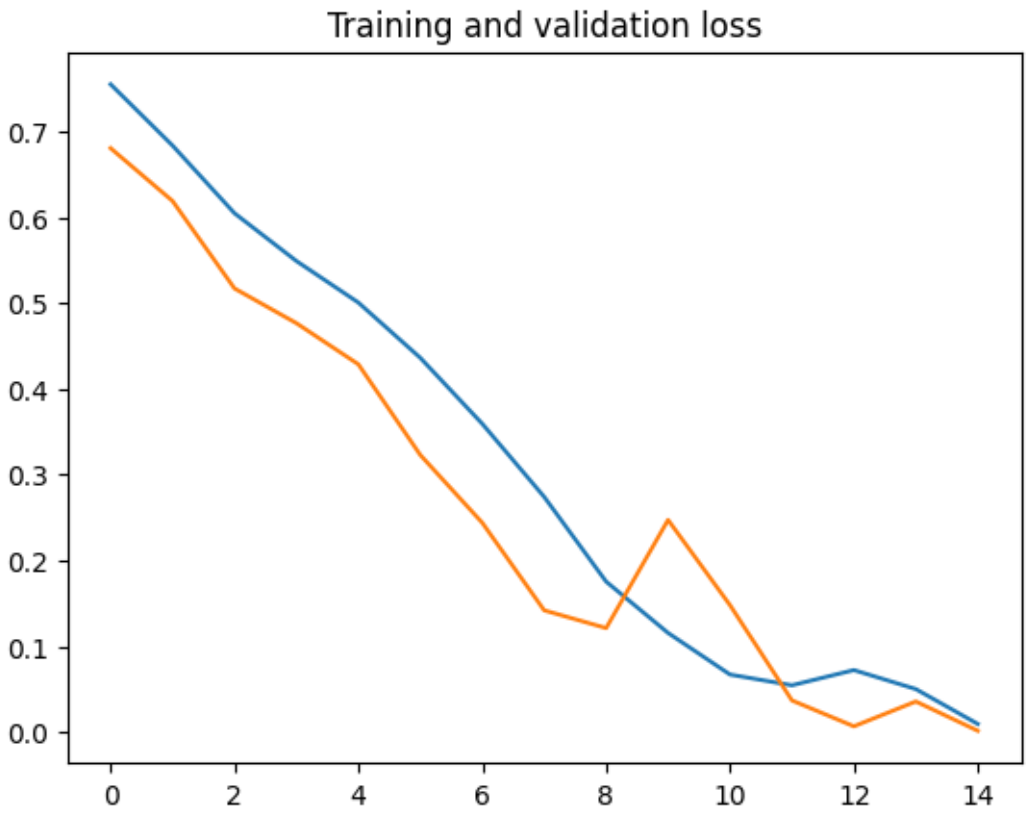
Tính toán độ chính xác, hàm tổn thất khi test trên tập validation

[ ]:
```
# ----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
# ----------------------------------------------------------
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

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

# ------------------------------------------------
# Plot training and validation accuracy per epoch
# ------------------------------------------------
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title("Training and validation accuracy")
plt.figure()

# ------------------------------------------------
# Plot training and validation loss per epoch
# ------------------------------------------------
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
```

```
plt.title("Training and validation loss")
```

[ ]: Text(0.5, 1.0, 'Training and validation loss')






## 1.6 Phần tự thực hành

### 1.6.1 Quan sát Loss Function và Accuracy trên tập Training và trên tập Validation với Model CNN trên - chưa có layer dropout. Giải thích những kết quả quan sát được.

Dựa theo file code của thầy thì quá trình train sẽ bị overfitting, cụ thể nguyên văn:

'As you can see, the model is **overfitting**. The training accuracy (in blue) gets close to 100% while the validation accuracy (in orange) stalls as 70%. The validation loss reaches its minimum after only five epochs.

Since we have a relatively small number of training examples (2000), overfitting should be the number one concern. Overfitting happens when a model exposed to too few examples learns patterns that do not generalize to new data.'

Tuy nhiên dựa trên biểu đồ sự thay đổi về Accuracy và Loss Function ở trên thì Accuracy và Loss Function ở cả train và validation đều tăng ở Accuracy và đều giảm trên Loss Function giống nhau. Tức là không bị trường hợp Overfitting.

Sự khác biệt với bài của thầy chỉ là thay vì dùng 2000 ảnh (1000 ảnh cho mỗi lớp trong 2 lớp) tải từ mledu-datasets thì em dùng luôn 2000 ảnh từ dữ liệu bài 2 (có 3 lớp - đã loại bỏ dữ liệu lớp Panda đi). Còn lại kiến trúc Model giữ nguyên. **Như vậy sự khác biệt để không bị overfitting như của thầy là do sự khác biệt về dữ liệu**

### 1.6.2 Hãy bổ sung các tầng Dropout này vào giữa các tầng Convolution, rate_drop khoảng từ 0.3 đến 0.5 và quan sát kết quả sau khi thay đổi của Model CNN đã có thêm layer dropout.

Trong thư viện tensorflow.keras.layer có một loại layer là Dropout(rate = ), trong đó giá trị của rate_drop thuộc [0, 1). Layer Dropout cho phép bỏ bớt dữ liệu train với tỉ lệ là rate_drop, nhằm mục đích tránh overfit.

Về việc đặt các lớp Dropout với tỷ lệ sao cho hợp lý:
Việc **thiết lập tỷ lệ Dropout lớn hơn ở các tầng sâu hơn** có thể có một số lợi ích trong việc đối phó với overfitting và cải thiện tính tổng quát hóa của mô hình. Dưới đây là một số lý do:

1. **Giảm kích thước đầu vào**: Khi chúng ta đi sâu vào mạng nơ-ron, kích thước đầu vào cho các tầng tiếp theo thường nhỏ hơn do sự giảm kích thước của các tensor khi đi qua các tầng trước đó. Do đó, việc loại bỏ một phần lớn dữ liệu tại các tầng sau có thể giúp ngăn chặn hiện tượng overfitting.
2. **Tăng cường tính đa dạng**: Càng sâu vào mạng, các đặc trưng trừu tượng và phức tạp hơn thường được học. Việc loại bỏ một phần lớn kết nối ở các tầng sau có thể buộc mô hình học cách sử dụng các đặc trưng này một cách chính xác và đa dạng hơn, thay vì dựa quá nhiều vào các đặc trưng cụ thể.
3. **Giảm thiểu overfitting**: Overfitting thường xảy ra khi mô hình học các quan hệ không tổng quát từ dữ liệu huấn luyện. Việc loại bỏ một phần lớn các kết nối thông qua Dropout ở các tầng sâu có thể giảm thiểu khả năng mô hình học các mối quan hệ cụ thể trong dữ liệu huấn luyện mà không phản ánh các mẫu tổng quát.
4. **Tăng tính linh hoạt và dễ tái sử dụng**: Việc sử dụng tỷ lệ Dropout lớn ở các tầng sâu có thể làm cho mô hình trở nên linh hoạt hơn và dễ dàng tái sử dụng. Điều này có thể giúp mô hình hoạt động tốt trên nhiều tập dữ liệu khác nhau và tránh overfitting trên dữ liệu mới.

Tuy nhiên, việc thiết lập tỷ lệ Dropout lớn hơn ở các tầng sâu cũng cần được cân nhắc cẩn thận, vì một tỷ lệ quá lớn có thể dẫn đến việc mất mát thông tin quá nhiều và làm giảm hiệu suất của mô hình trên dữ liệu kiểm tra. Do đó, việc điều chỉnh tỷ lệ Dropout cần phải được thực hiện dựa trên thử nghiệm và đánh giá kết quả trên dữ liệu thực tế.

[ ]:
```
import tensorflow as tf

model_with_dropout = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3␣
 ↪bytes color
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150,␣
 ↪3)),
    tf.keras.layers.Dropout(0.3),  # Adding Dropout layer after the first␣
 ↪convolutional layer
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.Dropout(0.4),  # Adding Dropout layer after the second␣
 ↪convolutional layer
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
```

```
    tf.keras.layers.Dropout(0.5),  # Adding Dropout layer after the third␣
 ↪convolutional layer
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding Dropout layer before the output␣
 ↪layer
    # Only 1 output neuron.
    # It will contain a value from 0-1 where 0 for 1 class ('cats')
    # and 1 for the other ('dogs')
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

[ ]:
```
from tensorflow.keras.optimizers import RMSprop

model_with_dropout.compile(
    optimizer=RMSprop(learning_rate=0.001),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

[ ]:
```
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint

# Define a callback to save the best model during training
checkpoint = ModelCheckpoint(
    "best_model_cat_dog_with_dropout.keras",
    monitor="val_accuracy",
    verbose=1,
    save_best_only=True,
    mode="max",
)

# Train the model
history = model_with_dropout.fit(
    train_generator,
    steps_per_epoch=100,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=2,
    callbacks=[checkpoint],
)
```

```
Epoch 1/15
```

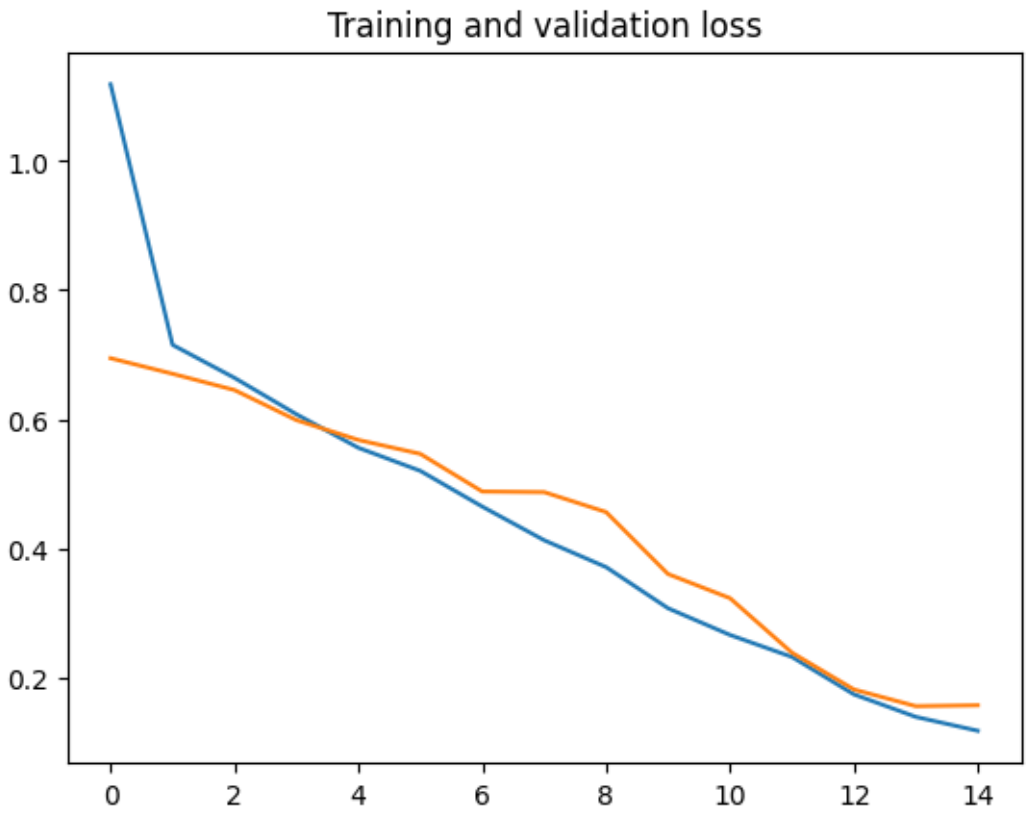
```
Epoch 1: val_accuracy improved from -inf to 0.48600, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 99s - loss: 1.1189 - accuracy: 0.5265 - val_loss: 0.6946 -
val_accuracy: 0.4860 - 99s/epoch - 995ms/step
Epoch 2/15

Epoch 2: val_accuracy improved from 0.48600 to 0.63900, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 90s - loss: 0.7153 - accuracy: 0.5685 - val_loss: 0.6704 -
val_accuracy: 0.6390 - 90s/epoch - 897ms/step
Epoch 3/15

Epoch 3: val_accuracy improved from 0.63900 to 0.66800, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 84s - loss: 0.6641 - accuracy: 0.6305 - val_loss: 0.6453 -
val_accuracy: 0.6680 - 84s/epoch - 842ms/step
Epoch 4/15

Epoch 4: val_accuracy improved from 0.66800 to 0.80300, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 91s - loss: 0.6078 - accuracy: 0.6770 - val_loss: 0.5987 -
val_accuracy: 0.8030 - 91s/epoch - 912ms/step
Epoch 5/15

Epoch 5: val_accuracy did not improve from 0.80300
100/100 - 84s - loss: 0.5558 - accuracy: 0.7235 - val_loss: 0.5679 -
val_accuracy: 0.6970 - 84s/epoch - 845ms/step
Epoch 6/15

Epoch 6: val_accuracy did not improve from 0.80300
100/100 - 80s - loss: 0.5201 - accuracy: 0.7550 - val_loss: 0.5464 -
val_accuracy: 0.7800 - 80s/epoch - 801ms/step
Epoch 7/15

Epoch 7: val_accuracy improved from 0.80300 to 0.80800, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 90s - loss: 0.4647 - accuracy: 0.7850 - val_loss: 0.4881 -
val_accuracy: 0.8080 - 90s/epoch - 901ms/step
Epoch 8/15

Epoch 8: val_accuracy did not improve from 0.80800
100/100 - 100s - loss: 0.4126 - accuracy: 0.8080 - val_loss: 0.4871 -
val_accuracy: 0.8070 - 100s/epoch - 1s/step
Epoch 9/15

Epoch 9: val_accuracy improved from 0.80800 to 0.82000, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 74s - loss: 0.3710 - accuracy: 0.8330 - val_loss: 0.4562 -
```

```
val_accuracy: 0.8200 - 74s/epoch - 740ms/step
Epoch 10/15

Epoch 10: val_accuracy improved from 0.82000 to 0.93000, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 99s - loss: 0.3072 - accuracy: 0.8670 - val_loss: 0.3602 -
val_accuracy: 0.9300 - 99s/epoch - 992ms/step
Epoch 11/15

Epoch 11: val_accuracy improved from 0.93000 to 0.95700, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 84s - loss: 0.2659 - accuracy: 0.8950 - val_loss: 0.3229 -
val_accuracy: 0.9570 - 84s/epoch - 837ms/step
Epoch 12/15

Epoch 12: val_accuracy improved from 0.95700 to 0.97700, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 82s - loss: 0.2318 - accuracy: 0.9035 - val_loss: 0.2384 -
val_accuracy: 0.9770 - 82s/epoch - 820ms/step
Epoch 13/15

Epoch 13: val_accuracy improved from 0.97700 to 0.97900, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 78s - loss: 0.1739 - accuracy: 0.9300 - val_loss: 0.1814 -
val_accuracy: 0.9790 - 78s/epoch - 781ms/step
Epoch 14/15

Epoch 14: val_accuracy improved from 0.97900 to 0.98700, saving model to
best_model_cat_dog_with_dropout.keras
100/100 - 89s - loss: 0.1390 - accuracy: 0.9475 - val_loss: 0.1556 -
val_accuracy: 0.9870 - 89s/epoch - 888ms/step
Epoch 15/15

Epoch 15: val_accuracy did not improve from 0.98700
100/100 - 91s - loss: 0.1175 - accuracy: 0.9565 - val_loss: 0.1570 -
val_accuracy: 0.9830 - 91s/epoch - 911ms/step
```

[ ]:
```python
# ----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
# ----------------------------------------------------------
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

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

```
# ------------------------------------------------
# Plot training and validation accuracy per epoch
# ------------------------------------------------
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title("Training and validation accuracy")
plt.figure()

# ------------------------------------------------
# Plot training and validation loss per epoch
# ------------------------------------------------
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title("Training and validation loss")
```

[ ]: Text(0.5, 1.0, 'Training and validation loss')




[ ]:
```
# Load the best saved model
# best_model_cat_dog_with_dropout = tf.keras.models.
  ↪load_model('best_model_cat_dog_with_dropout.keras')

# Evaluate the best model
# test_loss, test_acc = model_with_dropout.evaluate(validation_generator)
# print("Test Accuracy:", test_acc)
# print("Test Loss:", test_loss)
```

### 1.6.3 Sử dụng mô hình Hồi quy Logistic

**Đưa dữ liệu về size 150x150 & Chuyển ảnh về vector**

[ ]:
```
import numpy as np

# Function to load and preprocess images
def load_and_preprocess_images(directory, target_size=(150, 150)):
    image_data = []
    labels = []

    for class_name in os.listdir(directory):
        if class_name == 'panda': continue
```

```
        class_dir = os.path.join(directory, class_name)
        for image_name in os.listdir(class_dir):
            image_path = os.path.join(class_dir, image_name)
            img = image.load_img(image_path, target_size=target_size)
            img_array = image.img_to_array(img)
            img_vector = img_array.flatten() / 255.0  # Rescale to [0,1] and␣
 ↪flatten
            image_data.append(img_vector)
            labels.append(class_name)

    return np.array(image_data), np.array(labels)

logistic_base_dir = "data/cat_dog_panda"
logistic_train_dir = logistic_base_dir + "/train"
logistic_validation_dir = logistic_base_dir + "/validation"
# Load and preprocess images
logistic_train_data, logistic_train_labels =␣
 ↪load_and_preprocess_images(logistic_train_dir)
logistic_test_data, logistic_test_labels =␣
 ↪load_and_preprocess_images(logistic_validation_dir)
```

**Sử dụng hồi quy Logistic**

[ ]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression

# Encode labels
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(logistic_train_labels)
test_labels_encoded = label_encoder.transform(logistic_test_labels)

# Train a logistic regression model (Softmax)
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(logistic_train_data, train_labels_encoded)
```

```
/home/harito/venv/py/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:
```
LogisticRegression(max_iter=1000)
```

[ ]:
```
from joblib import dump

# Lưu mô hình Logistic Regression
dump(logistic_model, "best_logistic_model_cat_dog.joblib")
```

[ ]:
```
['best_logistic_model_cat_dog.joblib']
```

**Đánh giá các độ đo**

[ ]:
```
from sklearn.metrics import accuracy_score

# Evaluate logistic regression model
y_pred_softmax = logistic_model.predict(logistic_test_data)
```

[ ]:
```
accuracy_softmax = accuracy_score(test_labels_encoded, y_pred_softmax)
print("Softmax logistic regression accuracy:", accuracy_softmax)
```

```
Softmax logistic regression accuracy: 1.0
```

### 1.6.4 Sử dụng mạng ANN

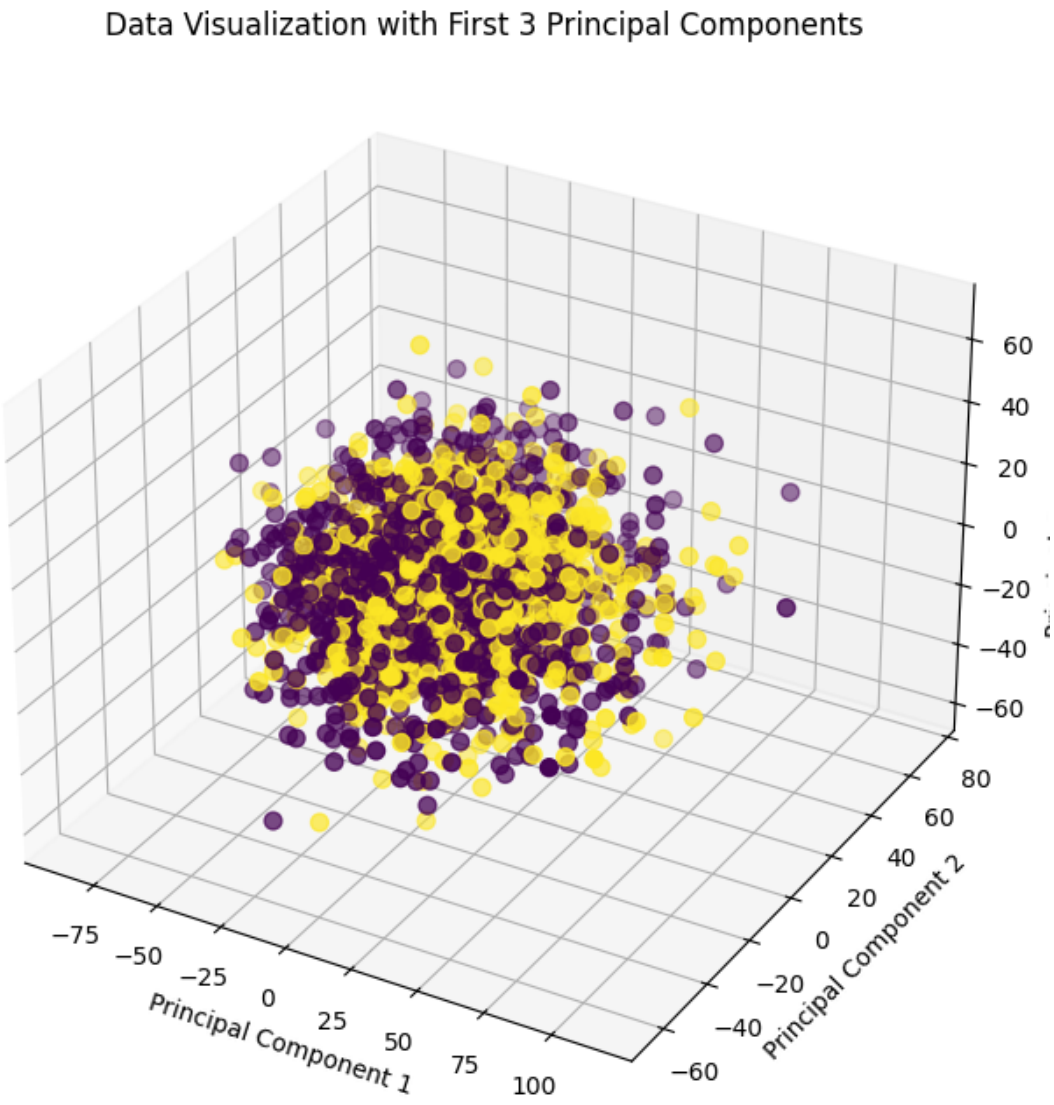
**Dùng PCA giảm số chiều về 225**

[ ]:
```
from sklearn.decomposition import PCA

# Perform PCA
pca = PCA(n_components=225)
X_train_pca = pca.fit_transform(logistic_train_data)
X_val_pca = pca.transform(logistic_test_data)
```

[ ]:
```
print(logistic_train_data.shape, X_train_pca.shape, X_val_pca.shape)
```

```
(2000, 67500) (2000, 225) (2000, 225)
```

[ ]:
```
from mpl_toolkits.mplot3d import Axes3D

# Lấy ba chiều đầu tiên
X_train_pca_3d = X_train_pca[:, :3]

# Tạo scatter plot 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_train_pca_3d[:, 0], X_train_pca_3d[:, 1], X_train_pca_3d[:, 2],␣
 ↪c=train_labels_encoded, cmap='viridis', s=50)
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('Data Visualization with First 3 Principal Components')
plt.show()
```

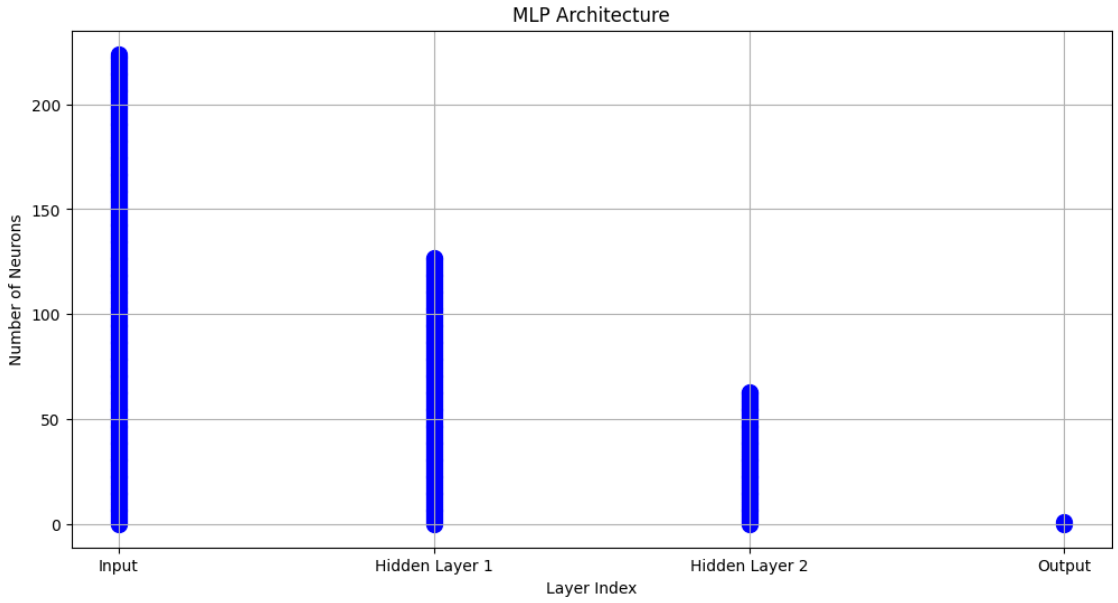



**Train ANN**

[ ]:
```
from sklearn.neural_network import MLPClassifier

# Train an ANN model
ann_model = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=1000,␣
 ↪random_state=42) # lớp ẩn 1 128 neutron, lớp ẩn 2 64 neutron
```

[ ]:
```
def plot_mlp_architecture(model, input_shape):
    layer_sizes = [input_shape[1]] + list(model.hidden_layer_sizes) + [len(np.
 ↪unique(train_labels_encoded))]
    plt.figure(figsize=(12, 6))
    plt.title('MLP Architecture')
```

```
    plt.xlabel('Layer Index')
    plt.ylabel('Number of Neurons')

    for i in range(len(layer_sizes)):
        # Vẽ neurons
        for j in range(layer_sizes[i]):
            plt.scatter(i, j, color='blue', s=100)
            # plt.text(i, j, f'Neuron {j}', ha='center', va='center',␣
 ↪fontsize=8)
        # # Vẽ kết nối
        # if i != len(layer_sizes) - 1:
        #     for j in range(layer_sizes[i]):
        #         for k in range(layer_sizes[i+1]):
        #             plt.plot([i, i+1], [j, k], color='black', alpha=0.5)

    plt.xticks(range(len(layer_sizes)), ['Input'] + [f'Hidden Layer {i}' for i␣
 ↪in range(1, len(layer_sizes)-1)] + ['Output'])
    plt.grid(True)
    plt.show()

# Gọi hàm trực quan hóa kiến trúc mạng
plot_mlp_architecture(ann_model, X_train_pca.shape)
```




[ ]:
```
ann_model.fit(X_train_pca, train_labels_encoded)
```

[ ]:
```
MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=1000, random_state=42)
```

[ ]:
```
from joblib import dump

# Lưu mô hình Logistic Regression
dump(ann_model, "best_ann_model_cat_dog.joblib")
```

[ ]:
```
['best_ann_model_cat_dog.joblib']
```

**Đánh giá các độ đo**

[ ]:
```
from sklearn.metrics import accuracy_score

# Evaluate ANN model
y_pred_ann = ann_model.predict(X_val_pca)
accuracy_ann = accuracy_score(test_labels_encoded, y_pred_ann)
print("ANN accuracy with PCA:", accuracy_ann)
```

```
ANN accuracy with PCA: 1.0
```

### 1.6.5 Nhận xét kết quả các độ đo của 3 phương pháp: Dùng ANN, Dùng Logistic, Dùng CNN

Ta có nhận xét như sau:
- ANN: 1.0
- Logistic: 1.0
- CNN: 1.0 (không dropout) và gần 1.0 (có dropout)

Như vậy thấy rằng các phương pháp đều học rất tốt. Hoặc có thể bộ dữ liệu đã được chuẩn hóa khá tốt.

### 1.6.6 Sưu tầm dữ liệu mới để test cho cả 3 phương pháp

**Sưu tầm >= 10 ảnh cho mỗi nhãn (có 2 nhãn)**

[ ]:
```
test_base_dir = "data/custom_data"
# Load and preprocess images
test_custom_data, test_custom_labels = load_and_preprocess_images(test_base_dir)
```

[ ]:
```
test_custom_data
```

[ ]:
```
array([[1.        , 1.        , 1.        , …, 1.        , 1.        ,
        1.        ],
       [0.15686275, 0.16470589, 0.16078432, …, 0.49803922, 0.4745098 ,
        0.38039216],
       [0.67058825, 0.58431375, 0.42352942, …, 0.43529412, 0.36078432,
        0.30588236],
       …,
       [0.21568628, 0.23137255, 0.03137255, …, 0.90588236, 0.78431374,
        0.6745098 ],
       [0.24313726, 0.34901962, 0.14901961, …, 0.20392157, 0.18039216,
        0.1882353 ],
```

```
       [0.54901963, 0.44313726, 0.3372549 , …, 0.47058824, 0.22745098,
        0.08235294]], dtype=float32)
```

[ ]:
```
test_custom_labels
```

[ ]:
```
array(['cats', 'cats', 'cats', 'cats', 'cats', 'cats', 'cats', 'cats',
       'cats', 'cats', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs', 'dogs',
       'dogs', 'dogs', 'dogs', 'dogs'], dtype='<U4')
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder

# Encode labels
label_encoder = LabelEncoder()
test_custom_labels_encoded = label_encoder.fit_transform(test_custom_labels)
```

[ ]:
```
test_custom_labels_encoded
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

**Thực hiện test dùng CNN**

[ ]:
```
# Load vào model đã train trước đó
from keras.models import load_model

# Đường dẫn đến file .keras
cnn_model_path = "best_model_cat_dog_no_dropout.keras"

# Load mô hình từ file
cnn_model = load_model(cnn_model_path)
```

[ ]:
```
import os
from keras.preprocessing import image

# Thư mục chứa dữ liệu kiểm tra
test_dir = "data/custom_data"

# Lưu số lượng dự đoán đúng
correct_predictions = 0
total_images = 0

# Duyệt qua các thư mục con trong thư mục dữ liệu kiểm tra
for class_name in os.listdir(test_dir):
    class_dir = os.path.join(test_dir, class_name)

    # Duyệt qua từng hình ảnh trong thư mục lớp hiện tại
    for fn in os.listdir(class_dir):
        path = os.path.join(class_dir, fn)
        img = image.load_img(path, target_size=(150, 150))
```

```
        x = image.img_to_array(img)
        x /= 255
        x = np.expand_dims(x, axis=0)
        images = np.vstack([x])

        # Dự đoán lớp của hình ảnh
        classes = cnn_model.predict(images, batch_size=10)
        predicted_class = "dogs" if classes[0] > 0.5 else "cats"
        print(f'Path = {path} \nProb = {classes}, Pred = {predicted_class},␣
 ↪Real = {class_name}')


        # Kiểm tra dự đoán
        if predicted_class == class_name:
            correct_predictions += 1

        total_images += 1

# Tính toán tỷ lệ dự đoán chính xác
accuracy = correct_predictions / total_images
print(f"Accuracy: {accuracy:.2f}")
```

```
1/1 [==============================] - 0s 46ms/step
Path = data/custom_data/cats/cat_0.jpg
Prob = [[0.99999684]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 37ms/step
Path = data/custom_data/cats/cat_1.jpg
Prob = [[0.00045746]], Pred = cats, Real = cats
1/1 [==============================] - 0s 39ms/step
Path = data/custom_data/cats/cat_2.jpg
Prob = [[0.21891448]], Pred = cats, Real = cats
1/1 [==============================] - 0s 40ms/step
Path = data/custom_data/cats/cat_3.jpg
Prob = [[0.8456861]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 52ms/step
Path = data/custom_data/cats/cat_4.jpg
Prob = [[0.0266785]], Pred = cats, Real = cats
1/1 [==============================] - 0s 96ms/step
Path = data/custom_data/cats/cat_5.jpg
Prob = [[0.99992466]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 106ms/step
Path = data/custom_data/cats/cat_6.jpg
Prob = [[0.00747295]], Pred = cats, Real = cats
1/1 [==============================] - 0s 100ms/step
Path = data/custom_data/cats/cat_7.jpg
Prob = [[3.3461314e-05]], Pred = cats, Real = cats
1/1 [==============================] - 0s 152ms/step
```

```
Path = data/custom_data/cats/cat_8.jpg
Prob = [[0.9999863]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 88ms/step
Path = data/custom_data/cats/cat_9.jpg
Prob = [[2.8652326e-07]], Pred = cats, Real = cats
1/1 [==============================] - 0s 100ms/step
Path = data/custom_data/dogs/dog_0.jpg
Prob = [[0.9997817]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 46ms/step
Path = data/custom_data/dogs/dog_1.jpg
Prob = [[0.73111343]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 51ms/step
Path = data/custom_data/dogs/dog_2.jpg
Prob = [[0.9999616]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 46ms/step
Path = data/custom_data/dogs/dog_3.jpg
Prob = [[0.3084595]], Pred = cats, Real = dogs
1/1 [==============================] - 0s 54ms/step
Path = data/custom_data/dogs/dog_4.jpg
Prob = [[0.09072394]], Pred = cats, Real = dogs
1/1 [==============================] - 0s 82ms/step
Path = data/custom_data/dogs/dog_5.jpg
Prob = [[0.00516271]], Pred = cats, Real = dogs
1/1 [==============================] - 0s 57ms/step
Path = data/custom_data/dogs/dog_6.jpg
Prob = [[0.99963766]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 67ms/step
Path = data/custom_data/dogs/dog_7.jpg
Prob = [[1.]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 49ms/step
Path = data/custom_data/dogs/dog_8.jpg
Prob = [[0.9912551]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 38ms/step
Path = data/custom_data/dogs/dog_9.jpg
Prob = [[0.9926731]], Pred = dogs, Real = dogs
Accuracy: 0.65
```

Có vẻ dữ liệu tự kiếm có nhiều nhiễu và cần chuẩn hóa trước, hoặc là do các dữ liệu trong validation khá tương đồng với train set.

**Thực hiện test dùng Logistic**

[ ]:
```
test_custom_y_pred_softmax = logistic_model.predict(test_custom_data)
```

[ ]:
```
from sklearn.metrics import accuracy_score

accuracy_softmax = accuracy_score(test_custom_labels_encoded,␣
 ↪test_custom_y_pred_softmax)
print("Softmax logistic regression accuracy:", accuracy_softmax)
```

```
Softmax logistic regression accuracy: 0.5
```

[ ]:
```
test_custom_y_pred_softmax
```

[ ]:
```
array([1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1])
```

[ ]:
```
test_custom_labels_encoded
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

**Thực hiện test dùng ANN**

[ ]:
```
from sklearn.decomposition import PCA

# Perform PCA
pca = PCA(n_components=225)
temp = pca.fit_transform(np.vstack((test_custom_data, logistic_train_data)))
test_custom_X_train_pca = temp[:20]
```

[ ]:
```
test_custom_X_train_pca.shape
```

[ ]:
```
(20, 225)
```

[ ]:
```
from sklearn.metrics import accuracy_score

test_custom_y_pred_ann = ann_model.predict(test_custom_X_train_pca)
accuracy_ann = accuracy_score(test_custom_labels_encoded,␣
 ↪test_custom_y_pred_ann)
print("ANN accuracy with PCA:", accuracy_ann)
```

```
ANN accuracy with PCA: 0.5
```

[ ]:
```
test_custom_y_pred_ann
```

[ ]:
```
array([1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1])
```

[ ]:
```
test_custom_labels_encoded
```

[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

**Nhận xét kết quả thu được**

- Dùng CNN cho accu = 0.65
- Dùng Softmax Logistic Regression cho accu = 0.5
- Dùng ANN (MLPClassifier) cho accu = 0.5

# 2 Ví dụ 2.

Dữ liệu vẫn là bên trên nhưng bổ sung thêm Panda
(Phân loại nhiều lớp > 2 class)

## 2.1 Chuẩn bị hệ thống

[ ]:
```
# %pip install tensorflow_datasets
```

[ ]:
```
import tensorflow as tf # Thư viện tensor.flow
from tensorflow import keras # Lớp keras và các công cụ liên quan
import tensorflow_datasets as tfds # Gọi và xử lý dữ liệu (với label là tên␣
 ↪folder chứa dữ liệu)
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

## 2.2 Thiết lập kiến trúc mạng CNN

Đoạn code tiếp theo thực hiện việc khởi dựng mô hình CNN với kiến trúc gồm:
- 3 tầng Convolution & MaxPooling nối tiếp: Tầng đầu gồm 16 filters; Tầng 2 gồm 32 filters và tầng ba gồm 64 filters, tất cả có cùng cỡ 3x3;
- Tiếp sau tầng Convolution thứ ba là tầng Flatten;
- Tiếp sau đó ta có tầng FullyConnected (FullConnection) FC với đầu ra là vector đặc trưng 512 phần tử. Tất cả các tầng Conv và FC đều sử dụng activation là ReLU;
- Cuối cùng, ta có tầng đầu ra với activation là SoftMax chia 03 loại

[ ]:
```
# Trains a model to classify images of 3 classes: cat, dog, and panda
def gen_model():
    # Defines & compiles the model
    model = tf.keras.models.Sequential(
        [
            # The first convolution
            tf.keras.layers.Conv2D(
                16, (3, 3), activation="relu", input_shape=(150, 150, 3)
            ),
            tf.keras.layers.MaxPooling2D(2, 2),
            keras.layers.Dropout(
                rate=0.15
            ),  # adding dropout regularization throughout the model to deal␣
 ↪with overfitting
            # The second convolution
            tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
            tf.keras.layers.MaxPooling2D(2, 2),
            keras.layers.Dropout(rate=0.1),
            # The third convolution
            tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),
            tf.keras.layers.MaxPooling2D(2, 2),
```

```
            keras.layers.Dropout(rate=0.10),
            # Flatten the results to feed into a DNN
            tf.keras.layers.Flatten(),
            # 512 neuron hidden layer (Fully connected layer)
            tf.keras.layers.Dense(512, activation="relu"),
            # 3 output neuron for the 3 classes of Animal Images
            tf.keras.layers.Dense(3, activation="softmax"),
        ]
    )

    # from tensorflow.keras.optimizers import RMSprop

    model.compile(loss="categorical_crossentropy", optimizer="adam",␣
↪metrics=["acc"])
    return model
```

### 2.3 Đọc vào dữ liệu

[ ]:
```
def train_test_animals():
    # Creates an instance of an ImageDataGenerator called train_datagen, and a␣
↪train_generator, train_datagen.flow_from_directory

    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    # splits data into training and testing(validation) sets
    train_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.25)

    import matplotlib.pyplot as plt

    data_dir = "data/cat_dog_panda/"
    # training data
    train_generator = train_datagen.flow_from_directory(
        data_dir + "train",
        # Source directory
        target_size=(150, 150),  # Resizes images
        batch_size=15,
        class_mode="categorical",
        subset="training",
    )

    epochs = 2
    # Testing data
    validation_generator = train_datagen.flow_from_directory(
        data_dir + "validation",
        target_size=(150, 150),
        batch_size=15,
        class_mode="categorical",
```

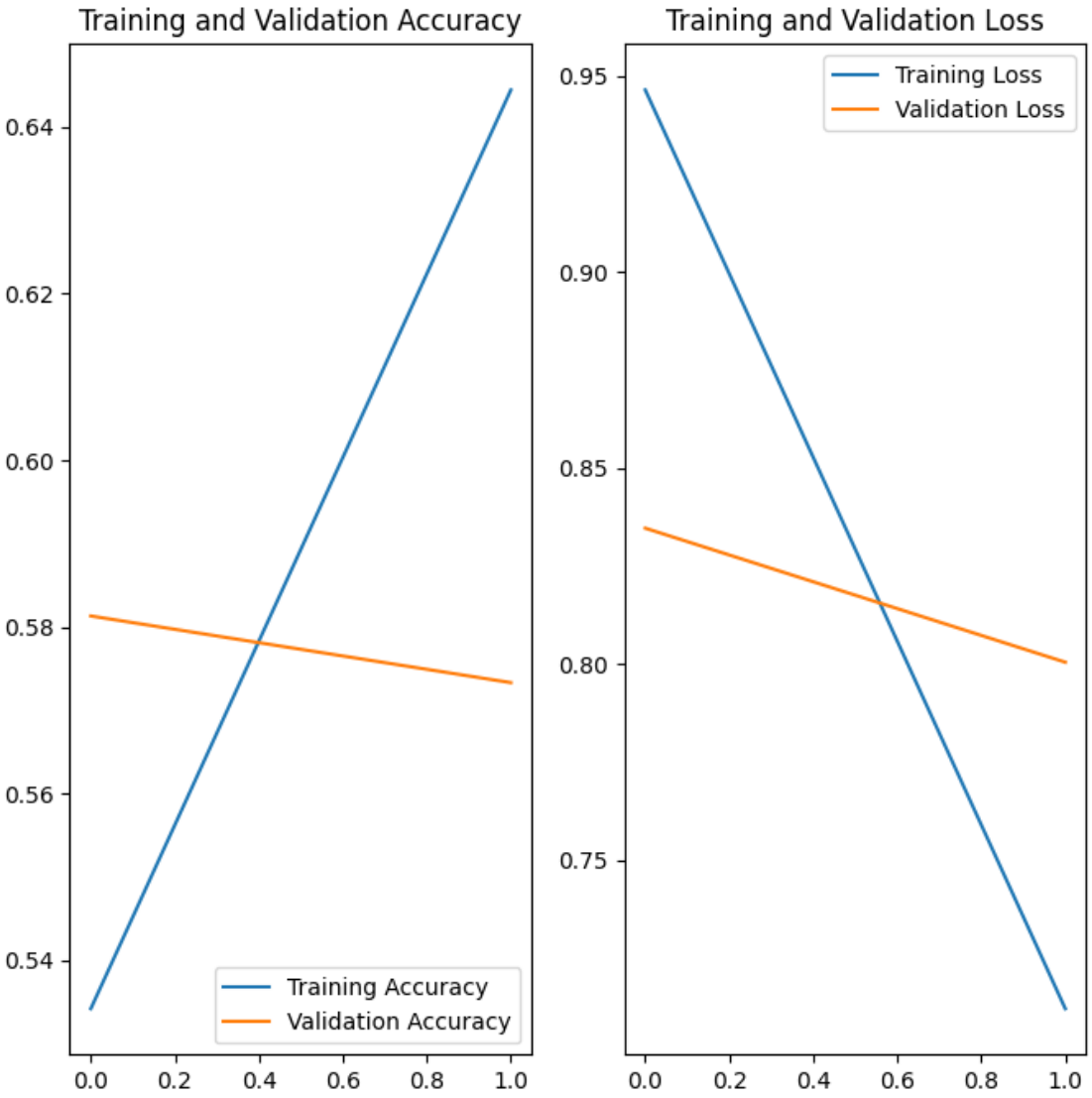
```
    subset="validation",
)  # set as validation data

model = gen_model()

# Model fitting for a number of epochs
history = model.fit(
    train_generator,
    steps_per_epoch=150,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=1,
)

# Save the model
model_filename = "best_model_cat_dog_panda_with_dropout.keras"
model.save(model_filename)
print("Model saved successfully.")

# Show training result
acc = history.history["acc"]
val_acc = history.history["val_acc"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]

# This code is used to plot the training and validation accuracy
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label="Training Accuracy")
plt.plot(epochs_range, val_acc, label="Validation Accuracy")
plt.legend(loc="lower right")
plt.title("Training and Validation Accuracy")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")
plt.legend(loc="upper right")
plt.title("Training and Validation Loss")
plt.show()

# returns accuracy of training
print("Training Accuracy:"), print(history.history["acc"][-1])
print("Testing Accuracy:"), print(history.history["val_acc"][-1])
```

```
    return model
```

### 2.4 Train CNN và visualize kết quả accuracy trên train set và validation set

[ ]:
```
cnn_model_cat_dog_panda = train_test_animals()
# run 3m2.4s
```

```
Found 2250 images belonging to 3 classes.
Found 750 images belonging to 3 classes.
Epoch 1/2
150/150 [==============================] - 93s 595ms/step - loss: 0.9465 - acc:
0.5342 - val_loss: 0.8347 - val_acc: 0.5813
Epoch 2/2
150/150 [==============================] - 85s 569ms/step - loss: 0.7121 - acc:
0.6444 - val_loss: 0.8005 - val_acc: 0.5733
Model saved successfully.
```




```
Training Accuracy:
0.644444465637207
Testing Accuracy:
0.5733333230018616
```

## 2.5 Phần tự thực hành

### 2.5.1 So sánh với Hồi quy SOFTMAX & ANN

**Sử dụng hồi quy SOFTMAX**

**Đưa dữ liệu về size 150x150**

[ ]:
```
import numpy as np


# Function to load and preprocess images
def load_and_preprocess_images(directory, target_size=(150, 150)):
    image_data = []
    labels = []
    import os
    from keras.preprocessing import image

    for class_name in os.listdir(directory):
        # if class_name == 'panda': continue
        class_dir = os.path.join(directory, class_name)
        for image_name in os.listdir(class_dir):
            image_path = os.path.join(class_dir, image_name)
            img = image.load_img(image_path, target_size=target_size)
            img_array = image.img_to_array(img)
            img_vector = img_array.flatten() / 255.0  # Rescale to [0,1] and␣
 ↪flatten
            image_data.append(img_vector)
            labels.append(class_name)

    return np.array(image_data), np.array(labels)


logistic_base_dir = "data/cat_dog_panda"
logistic_train_dir = logistic_base_dir + "/train"
logistic_validation_dir = logistic_base_dir + "/validation"
# Load and preprocess images
logistic_train_data, logistic_train_labels = load_and_preprocess_images(
    logistic_train_dir
)
logistic_test_data, logistic_test_labels = load_and_preprocess_images(
    logistic_validation_dir
)
# run 36.6s
```

**Thực hiện hồi quy SOFTMAX**

[ ]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression

# Encode labels
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(logistic_train_labels)
test_labels_encoded = label_encoder.transform(logistic_test_labels)

# Train a logistic regression model (Softmax)
```

```
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(logistic_train_data, train_labels_encoded)
# run 37m54.6s
```

```
/home/harito/venv/py/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:
```
LogisticRegression(max_iter=1000)
```

[ ]:
```
from joblib import dump

# Lưu mô hình Logistic Regression
dump(logistic_model, "best_logistic_model_cat_dog_panda.joblib")
```

[ ]:
```
['best_logistic_model_cat_dog_panda.joblib']
```

[ ]:
```
train_labels_encoded
```

[ ]:
```
array([0, 0, 0, …, 2, 2, 2])
```

[ ]:
```
logistic_train_data.shape
```

[ ]:
```
(3000, 67500)
```

[ ]:
```
from sklearn.metrics import accuracy_score

# Evaluate logistic regression model
y_pred_softmax = logistic_model.predict(logistic_test_data)
```

[ ]:
```
accuracy_softmax = accuracy_score(test_labels_encoded, y_pred_softmax)
print("Softmax logistic regression accuracy:", accuracy_softmax)
```

```
Softmax logistic regression accuracy: 1.0
```

### Sử dụng ANN

### Giảm số chiều về 225

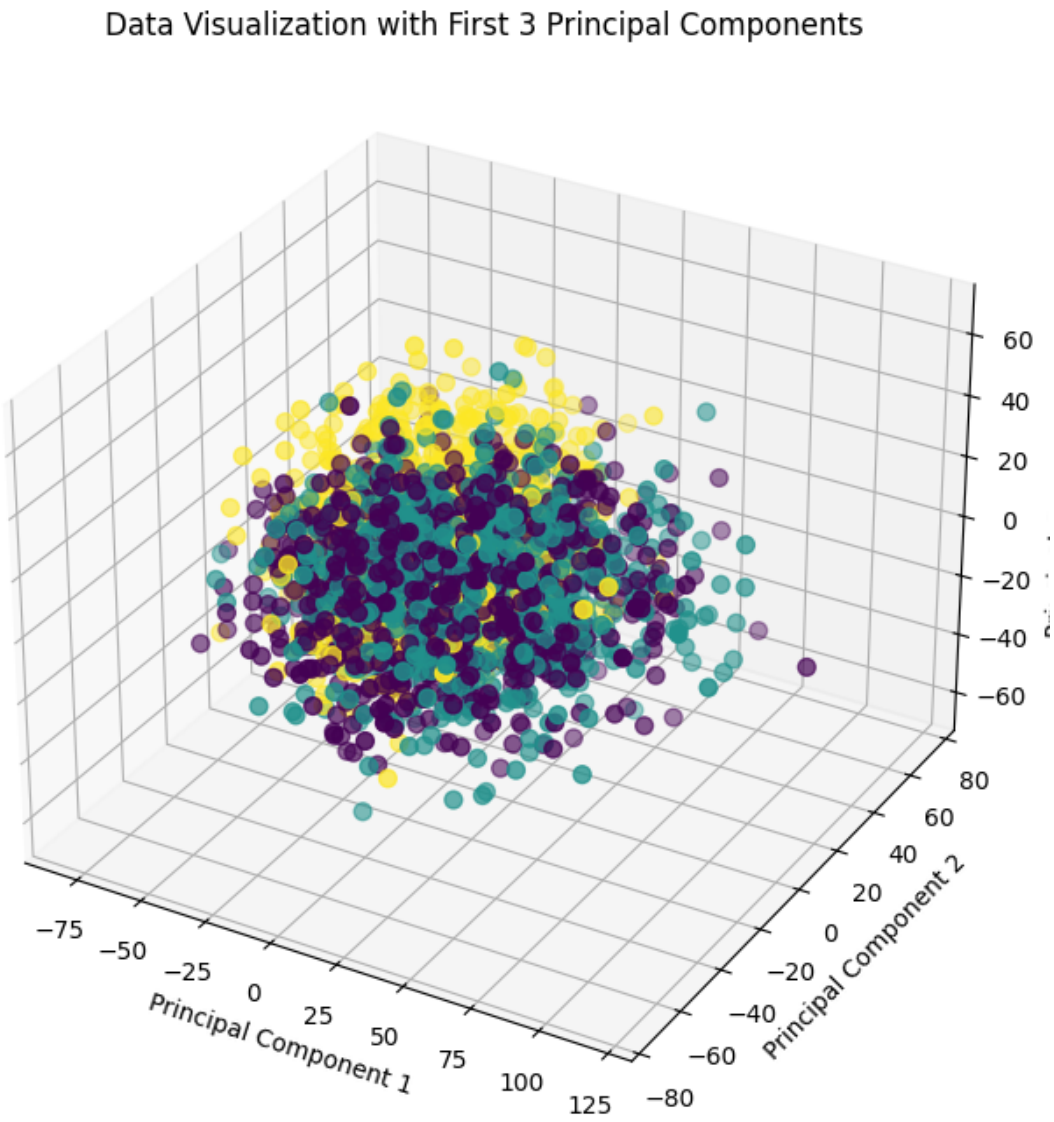
[ ]:
```
from sklearn.decomposition import PCA
```

```
# Perform PCA
pca = PCA(n_components=225)
X_train_pca = pca.fit_transform(logistic_train_data)
X_val_pca = pca.transform(logistic_test_data)
# run 1m25.7s
```

[ ]:
```
print(logistic_train_data.shape, X_train_pca.shape, X_val_pca.shape)
```

```
(3000, 67500) (3000, 225) (3000, 225)
```

[ ]:
```
from mpl_toolkits.mplot3d import Axes3D

# Lấy ba chiều đầu tiên
X_train_pca_3d = X_train_pca[:, :3]

# Tạo scatter plot 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_train_pca_3d[:, 0], X_train_pca_3d[:, 1], X_train_pca_3d[:, 2],␣
 ↪c=train_labels_encoded, cmap='viridis', s=50)
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('Data Visualization with First 3 Principal Components')
plt.show()
```

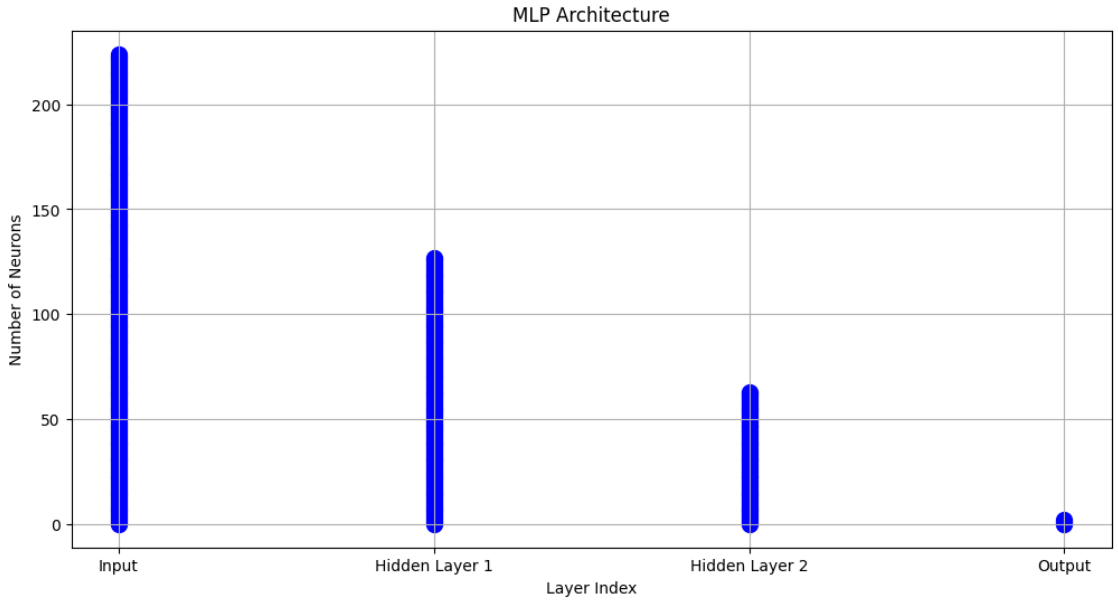



**Thực hiện phân loại với ANN**

[ ]:
```
from sklearn.neural_network import MLPClassifier

# Train an ANN model
ann_model = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=1000,␣
↪random_state=42) # lớp ẩn 1 128 neutron, lớp ẩn 2 64 neutron
```

[ ]:
```
def plot_mlp_architecture(model, input_shape):
    layer_sizes = [input_shape[1]] + list(model.hidden_layer_sizes) + [len(np.
↪unique(train_labels_encoded))]
    plt.figure(figsize=(12, 6))
```

```
    plt.title('MLP Architecture')
    plt.xlabel('Layer Index')
    plt.ylabel('Number of Neurons')

    for i in range(len(layer_sizes)):
        # Vẽ neurons
        for j in range(layer_sizes[i]):
            plt.scatter(i, j, color='blue', s=100)
            # plt.text(i, j, f'Neuron {j}', ha='center', va='center',␣
 ↪fontsize=8)
        # # Vẽ kết nối
        # if i != len(layer_sizes) - 1:
        #     for j in range(layer_sizes[i]):
        #         for k in range(layer_sizes[i+1]):
        #             plt.plot([i, i+1], [j, k], color='black', alpha=0.5)

    plt.xticks(range(len(layer_sizes)), ['Input'] + [f'Hidden Layer {i}' for i␣
 ↪in range(1, len(layer_sizes)-1)] + ['Output'])
    plt.grid(True)
    plt.show()

# Gọi hàm trực quan hóa kiến trúc mạng
plot_mlp_architecture(ann_model, X_train_pca.shape)
```




[ ]:
```
ann_model.fit(X_train_pca, train_labels_encoded)
```

[ ]:
```
MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=1000, random_state=42)
```

[ ]:
```
from joblib import dump

# Lưu mô hình Logistic Regression
dump(ann_model, "best_ann_model_cat_dog_panda.joblib")
```

[ ]:
```
['best_ann_model_cat_dog_panda.joblib']
```

[ ]:
```
from sklearn.metrics import accuracy_score

# Evaluate ANN model
y_pred_ann = ann_model.predict(X_val_pca)
accuracy_ann = accuracy_score(test_labels_encoded, y_pred_ann)
print("ANN accuracy with PCA:", accuracy_ann)
```

```
ANN accuracy with PCA: 1.0
```

**So sánh kết quả Accuracy của SOFTMAX, ANN, CNN trên train và valid set** Ta có nhận xét như sau trên tập test:
- ANN: 1.0
- Logistic: 1.0
- CNN: 0.57 (thậm chí epoch 2 còn có valid accuracy giảm đi)
Cụ thể:
Training Accuracy:
0.644444465637207
Testing Accuracy:
0.5733333230018616

Như vậy thấy rằng các phương pháp đều học rất tốt trừ CNN đang thiết lập kiến trúc chưa đủ tốt.

### 2.5.2 Một số tùy chỉnh với CNN để quan sát sự thay đổi của độ chính xác

**Thử nghiệm riêng lẻ với 1 vài dữ liệu test (tự kiếm)**

[ ]:
```
from sklearn.preprocessing import LabelEncoder

# Encode labels
label_encoder = LabelEncoder()
test_custom_labels_encoded = label_encoder.fit_transform(test_custom_labels)
```

[ ]:
```
import os
from keras.preprocessing import image

# Thư mục chứa dữ liệu kiểm tra
test_dir = "data/custom_data"

# Lưu số lượng dự đoán đúng
correct_predictions = 0
total_images = 0
```

```
cat_dog_panda_y_pred = []
cat_dog_panda_y_true = []
# Duyệt qua các thư mục con trong thư mục dữ liệu kiểm tra
for class_name in os.listdir(test_dir):
    class_dir = os.path.join(test_dir, class_name)

    # Duyệt qua từng hình ảnh trong thư mục lớp hiện tại
    for fn in os.listdir(class_dir):
        path = os.path.join(class_dir, fn)
        img = image.load_img(path, target_size=(150, 150))

        x = image.img_to_array(img)
        x /= 255
        x = np.expand_dims(x, axis=0)
        images = np.vstack([x])

        # Dự đoán lớp của hình ảnh
        classes = cnn_model_cat_dog_panda.predict(images, batch_size=10)
        predicted_class = np.argmax(
            classes
        )  # Lớp có xác suất cao nhất là lớp được dự đoán
        class_names = ["cats", "dogs", "pandas"]
        predicted_class_name = class_names[predicted_class]
        cat_dog_panda_y_pred.append(predicted_class_name)
        cat_dog_panda_y_true.append(class_name)
        print(
            f"Path = {path} \nProb = {classes}, Pred = {predicted_class_name},␣
 ↪Real = {class_name}"
        )

        # Kiểm tra dự đoán
        if predicted_class_name == class_name:
            correct_predictions += 1

        total_images += 1

# Tính toán tỷ lệ dự đoán chính xác
accuracy = correct_predictions / total_images
print(f"Accuracy: {accuracy:.2f}")
```

```
1/1 [==============================] - 0s 48ms/step
Path = data/custom_data/cats/cat_0.jpg
Prob = [[0.29309192 0.6410811  0.06582698]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 50ms/step
Path = data/custom_data/cats/cat_1.jpg
Prob = [[0.7146454  0.19932452 0.08603012]], Pred = cats, Real = cats
1/1 [==============================] - 0s 45ms/step
Path = data/custom_data/cats/cat_2.jpg
```

```
Prob = [[0.48388666 0.49088418 0.0252291 ]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 55ms/step
Path = data/custom_data/cats/cat_3.jpg
Prob = [[0.33516142 0.6101712  0.05466736]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 51ms/step
Path = data/custom_data/cats/cat_4.jpg
Prob = [[0.2505313  0.5593408  0.19012797]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 84ms/step
Path = data/custom_data/cats/cat_5.jpg
Prob = [[0.47646177 0.40005141 0.12348679]], Pred = cats, Real = cats
1/1 [==============================] - 0s 57ms/step
Path = data/custom_data/cats/cat_6.jpg
Prob = [[0.47129542 0.43934122 0.08936339]], Pred = cats, Real = cats
1/1 [==============================] - 0s 101ms/step
Path = data/custom_data/cats/cat_7.jpg
Prob = [[0.53641695 0.42680132 0.03678171]], Pred = cats, Real = cats
1/1 [==============================] - 0s 112ms/step
Path = data/custom_data/cats/cat_8.jpg
Prob = [[0.44601917 0.5420349  0.01194592]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 84ms/step
Path = data/custom_data/cats/cat_9.jpg
Prob = [[0.7993771  0.17576641 0.02485659]], Pred = cats, Real = cats
1/1 [==============================] - 0s 56ms/step
Path = data/custom_data/dogs/dog_0.jpg
Prob = [[0.4715515  0.49421963 0.0342289 ]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 61ms/step
Path = data/custom_data/dogs/dog_1.jpg
Prob = [[0.32397294 0.31961134 0.35641572]], Pred = pandas, Real = dogs
1/1 [==============================] - 0s 48ms/step
Path = data/custom_data/dogs/dog_2.jpg
Prob = [[0.317259   0.50342935 0.17931168]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 68ms/step
Path = data/custom_data/dogs/dog_3.jpg
Prob = [[0.327663   0.49271566 0.17962138]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 90ms/step
Path = data/custom_data/dogs/dog_4.jpg
Prob = [[0.23763354 0.45111242 0.31125408]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 222ms/step
Path = data/custom_data/dogs/dog_5.jpg
Prob = [[0.40022758 0.11584583 0.4839267 ]], Pred = pandas, Real = dogs
1/1 [==============================] - 0s 145ms/step
Path = data/custom_data/dogs/dog_6.jpg
Prob = [[0.50632596 0.46572623 0.02794778]], Pred = cats, Real = dogs
1/1 [==============================] - 0s 62ms/step
Path = data/custom_data/dogs/dog_7.jpg
Prob = [[0.42689085 0.54833066 0.02477846]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 93ms/step
Path = data/custom_data/dogs/dog_8.jpg
```

```
Prob = [[0.3997705  0.53999263 0.06023687]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 54ms/step
Path = data/custom_data/dogs/dog_9.jpg
Prob = [[0.49051535 0.48984075 0.0196439 ]], Pred = cats, Real = dogs
Accuracy: 0.55
```

[ ]:
```
cat_dog_panda_y_pred
```

[ ]:
```
['dogs',
 'cats',
 'dogs',
 'dogs',
 'dogs',
 'cats',
 'cats',
 'cats',
 'dogs',
 'cats',
 'dogs',
 'pandas',
 'dogs',
 'dogs',
 'dogs',
 'pandas',
 'cats',
 'dogs',
 'dogs',
 'cats']
```

[ ]:
```
cat_dog_panda_y_true
```

[ ]:
```
['cats',
 'cats',
 'cats',
 'cats',
 'cats',
 'cats',
 'cats',
 'cats',
 'cats',
 'cats',
 'dogs',
 'dogs',
 'dogs',
 'dogs',
 'dogs',
 'dogs',
 'dogs',
```

```
'dogs',
'dogs',
'dogs']
```

**Đưa về phân loại 2 lớp (chỉ chó và mèo) nhưng giữ nguyên hàm kích hoạt tầng đầu ra là SOFTMAX**

**Chỉ lấy 300 cho training và 100 cho validation** Đã thực hiện với data và cấu trúc mô hình như này ở ví dụ 1

**Train model và các chỉ số** Đã thực hiện với data và cấu trúc mô hình như này ở ví dụ 1

**Bổ sung các tầng Dropout vào giữa các tầng Convolution của mạng CNN ban đầu (bài toán phân loại 3 class với data 1000 ảnh ở mỗi class ở mỗi tập train và valid), rate_drop khoảng từ 0.3 đến 0.5 và quan sát kết quả sau khi thay đổi.** Bổ sung Dropout vào kiến trúc mạng CNN

[ ]:
```python
# Trains a model to classify images of 3 classes: cat, dog, and panda
def gen_model():
    # Defines & compiles the model
    model = tf.keras.models.Sequential(
        [
            # The first convolution
            tf.keras.layers.Conv2D(
                16, (3, 3), activation="relu", input_shape=(150, 150, 3)
            ),
            tf.keras.layers.MaxPooling2D(2, 2),
            keras.layers.Dropout(
                rate=0.3
            ),  # adding dropout regularization throughout the model to deal␣
 ↪with overfitting
            # The second convolution
            tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
            tf.keras.layers.MaxPooling2D(2, 2),
            keras.layers.Dropout(rate=0.4),
            # The third convolution
            tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),
            tf.keras.layers.MaxPooling2D(2, 2),
            keras.layers.Dropout(rate=0.5),
            # Flatten the results to feed into a DNN
            tf.keras.layers.Flatten(),
            # 512 neuron hidden layer (Fully connected layer)
            tf.keras.layers.Dense(512, activation="relu"),
            # 3 output neuron for the 3 classes of Animal Images
            tf.keras.layers.Dense(3, activation="softmax"),
        ]
    )
```

```
    # from tensorflow.keras.optimizers import RMSprop

    model.compile(loss="categorical_crossentropy", optimizer="adam",␣
↪metrics=["acc"])
    return model
```

Train CNN

[ ]:
```
def train_test_animals():
    # Creates an instance of an ImageDataGenerator called train_datagen, and a␣
↪train_generator, train_datagen.flow_from_directory

    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    # splits data into training and testing(validation) sets
    train_datagen = ImageDataGenerator(rescale=1.0 / 255, validation_split=0.25)

    import matplotlib.pyplot as plt

    data_dir = "data/cat_dog_panda/"
    # training data
    train_generator = train_datagen.flow_from_directory(
        data_dir + "train",
        # Source directory
        target_size=(150, 150),  # Resizes images
        batch_size=15,
        class_mode="categorical",
        subset="training",
    )

    epochs = 2
    # Testing data
    validation_generator = train_datagen.flow_from_directory(
        data_dir + "validation",
        target_size=(150, 150),
        batch_size=15,
        class_mode="categorical",
        subset="validation",
    )  # set as validation data

    model = gen_model()

    # Model fitting for a number of epochs
    history = model.fit(
        train_generator,
        steps_per_epoch=150,
```

```
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=50,
        verbose=1,
    )

    # Save the model
    model_filename = "best_model_cat_dog_panda_with_higher_dropout.keras"
    model.save(model_filename)
    print("Model saved successfully.")

    # Show training result
    acc = history.history["acc"]
    val_acc = history.history["val_acc"]

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]

    # This code is used to plot the training and validation accuracy
    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label="Training Accuracy")
    plt.plot(epochs_range, val_acc, label="Validation Accuracy")
    plt.legend(loc="lower right")
    plt.title("Training and Validation Accuracy")

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label="Training Loss")
    plt.plot(epochs_range, val_loss, label="Validation Loss")
    plt.legend(loc="upper right")
    plt.title("Training and Validation Loss")
    plt.show()

    # returns accuracy of training
    print("Training Accuracy:"), print(history.history["acc"][-1])
    print("Testing Accuracy:"), print(history.history["val_acc"][-1])

    return model
```

[ ]:
```
cnn_model_cat_dog_panda_higher_dropout = train_test_animals()
```

```
Found 2250 images belonging to 3 classes.
Found 750 images belonging to 3 classes.
Epoch 1/2
150/150 [==============================] - 149s 978ms/step - loss: 1.0046 - acc:
0.5093 - val_loss: 0.8391 - val_acc: 0.5747
```

```
Epoch 2/2
150/150 [==============================] - 77s 514ms/step - loss: 0.7331 - acc:
0.6204 - val_loss: 0.8186 - val_acc: 0.6080
Model saved successfully.
```


```
Training Accuracy:
0.6204444169998169
Testing Accuracy:
0.6079999804496765
```

Quan sát các chỉ số đánh giá

Mô hình đã tránh được overfitting so với việc không có mức dropout cao hơn.
Validation tăng, loss function giảm.

Thực hiện test thử nghiệm với dữ liệu tự kiếm

[ ]:
```
import os
from keras.preprocessing import image

# Thư mục chứa dữ liệu kiểm tra
test_dir = "data/custom_data_add_panda"

# Lưu số lượng dự đoán đúng
correct_predictions = 0
total_images = 0

cat_dog_panda_y_pred = []
cat_dog_panda_y_true = []
# Duyệt qua các thư mục con trong thư mục dữ liệu kiểm tra
for class_name in os.listdir(test_dir):
    class_dir = os.path.join(test_dir, class_name)

    # Duyệt qua từng hình ảnh trong thư mục lớp hiện tại
    for fn in os.listdir(class_dir):
        path = os.path.join(class_dir, fn)
        img = image.load_img(path, target_size=(150, 150))

        x = image.img_to_array(img)
        x /= 255
        x = np.expand_dims(x, axis=0)
        images = np.vstack([x])

        # Dự đoán lớp của hình ảnh
        classes = cnn_model_cat_dog_panda.predict(images, batch_size=10)
        predicted_class = np.argmax(
            classes
        )  # Lớp có xác suất cao nhất là lớp được dự đoán
        class_names = ["cats", "dogs", "pandas"]
        predicted_class_name = class_names[predicted_class]
        cat_dog_panda_y_pred.append(predicted_class_name)
        cat_dog_panda_y_true.append(class_name)
        print(
            f"Path = {path} \nProb = {classes}, Pred = {predicted_class_name},␣
↪Real = {class_name}"
        )

        # Kiểm tra dự đoán
        if predicted_class_name == class_name:
            correct_predictions += 1

        total_images += 1
```

```python
# Tính toán tỷ lệ dự đoán chính xác
accuracy = correct_predictions / total_images
print(f"Accuracy: {accuracy:.2f}")
```

```
1/1 [==============================] - 0s 46ms/step
Path = data/custom_data_add_panda/cats/cat_0.jpg
Prob = [[0.29309192 0.6410811  0.06582698]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 42ms/step
Path = data/custom_data_add_panda/cats/cat_1.jpg
Prob = [[0.7146454  0.19932452 0.08603012]], Pred = cats, Real = cats
1/1 [==============================] - 0s 51ms/step
Path = data/custom_data_add_panda/cats/cat_2.jpg
Prob = [[0.48388666 0.49088418 0.0252291 ]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 57ms/step
Path = data/custom_data_add_panda/cats/cat_3.jpg
Prob = [[0.33516142 0.6101712  0.05466736]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 157ms/step
Path = data/custom_data_add_panda/cats/cat_4.jpg
Prob = [[0.2505313  0.5593408  0.19012797]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 155ms/step
Path = data/custom_data_add_panda/cats/cat_5.jpg
Prob = [[0.47646177 0.40005141 0.12348679]], Pred = cats, Real = cats
1/1 [==============================] - 0s 75ms/step
Path = data/custom_data_add_panda/cats/cat_6.jpg
Prob = [[0.47129542 0.43934122 0.08936339]], Pred = cats, Real = cats
1/1 [==============================] - 0s 72ms/step
Path = data/custom_data_add_panda/cats/cat_7.jpg
Prob = [[0.53641695 0.42680132 0.03678171]], Pred = cats, Real = cats
1/1 [==============================] - 0s 94ms/step
Path = data/custom_data_add_panda/cats/cat_8.jpg
Prob = [[0.44601917 0.5420349  0.01194592]], Pred = dogs, Real = cats
1/1 [==============================] - 0s 69ms/step
Path = data/custom_data_add_panda/cats/cat_9.jpg
Prob = [[0.7993771  0.17576641 0.02485659]], Pred = cats, Real = cats
1/1 [==============================] - 0s 47ms/step
Path = data/custom_data_add_panda/dogs/dog_0.jpg
Prob = [[0.4715515  0.49421963 0.0342289 ]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 59ms/step
Path = data/custom_data_add_panda/dogs/dog_1.jpg
Prob = [[0.32397294 0.31961134 0.35641572]], Pred = pandas, Real = dogs
1/1 [==============================] - 0s 69ms/step
Path = data/custom_data_add_panda/dogs/dog_2.jpg
Prob = [[0.317259   0.50342935 0.17931168]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 127ms/step
Path = data/custom_data_add_panda/dogs/dog_3.jpg
Prob = [[0.327663   0.49271566 0.17962138]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 106ms/step
```

```
Path = data/custom_data_add_panda/dogs/dog_4.jpg
Prob = [[0.23763354 0.45111242 0.31125408]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 84ms/step
Path = data/custom_data_add_panda/dogs/dog_5.jpg
Prob = [[0.40022758 0.11584583 0.4839267 ]], Pred = pandas, Real = dogs
1/1 [==============================] - 0s 81ms/step
Path = data/custom_data_add_panda/dogs/dog_6.jpg
Prob = [[0.50632596 0.46572623 0.02794778]], Pred = cats, Real = dogs
1/1 [==============================] - 0s 80ms/step
Path = data/custom_data_add_panda/dogs/dog_7.jpg
Prob = [[0.42689085 0.54833066 0.02477846]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 76ms/step
Path = data/custom_data_add_panda/dogs/dog_8.jpg
Prob = [[0.3997705  0.53999263 0.06023687]], Pred = dogs, Real = dogs
1/1 [==============================] - 0s 129ms/step
Path = data/custom_data_add_panda/dogs/dog_9.jpg
Prob = [[0.49051535 0.48984075 0.0196439 ]], Pred = cats, Real = dogs
1/1 [==============================] - 0s 168ms/step
Path = data/custom_data_add_panda/pandas/panda0.jpg
Prob = [[0.43970224 0.38785374 0.17244396]], Pred = cats, Real = pandas
1/1 [==============================] - 0s 107ms/step
Path = data/custom_data_add_panda/pandas/panda1.jpg
Prob = [[0.29011223 0.3634618  0.34642598]], Pred = dogs, Real = pandas
1/1 [==============================] - 0s 113ms/step
Path = data/custom_data_add_panda/pandas/panda2.jpg
Prob = [[0.07200173 0.24186489 0.6861334 ]], Pred = pandas, Real = pandas
1/1 [==============================] - 0s 82ms/step
Path = data/custom_data_add_panda/pandas/panda3.jpg
Prob = [[0.2902974  0.32706642 0.3826362 ]], Pred = pandas, Real = pandas
1/1 [==============================] - 0s 92ms/step
Path = data/custom_data_add_panda/pandas/panda4.jpg
Prob = [[0.2832533  0.18978444 0.5269623 ]], Pred = pandas, Real = pandas
1/1 [==============================] - 0s 60ms/step
Path = data/custom_data_add_panda/pandas/panda5.jpg
Prob = [[0.33316872 0.04507088 0.6217604 ]], Pred = pandas, Real = pandas
1/1 [==============================] - 0s 48ms/step
Path = data/custom_data_add_panda/pandas/panda6.jpg
Prob = [[0.03404907 0.0335965  0.9323544 ]], Pred = pandas, Real = pandas
1/1 [==============================] - 0s 50ms/step
Path = data/custom_data_add_panda/pandas/panda7.jpg
Prob = [[0.41442993 0.46088406 0.12468608]], Pred = dogs, Real = pandas
1/1 [==============================] - 0s 100ms/step
Path = data/custom_data_add_panda/pandas/panda8.jpg
Prob = [[0.35779092 0.16880944 0.47339967]], Pred = pandas, Real = pandas
1/1 [==============================] - 0s 45ms/step
Path = data/custom_data_add_panda/pandas/panda9.jpg
Prob = [[0.3832214  0.17388484 0.4428938 ]], Pred = pandas, Real = pandas
Accuracy: 0.60
```

Nhận xét: Độ chính xác khá ổn định khi so sánh với tập train và validation

# 3 Ví dụ 3. (Bài tập tự thực hành)

Sử dụng CNN phân loại 11 lớp cảm xúc từ hình ảnh khuôn mặt.
(Phân loại nhiều nhãn (>= 3))
So sánh với dùng ANN và Logistic.

[ ]:
```
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import layers
```

```
2024-04-13 13:07:56.270742: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-04-13 13:07:56.270865: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-04-13 13:07:56.274241: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-04-13 13:07:56.292563: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-04-13 13:07:59.639914: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

## 3.1 Đọc vào dữ liệu

[ ]:
```
def read_data():
    # Load data
    path = "../Week7/data/face_data/"
    ids = range(1, 16)
    states = ["centerlight", "glasses", "happy", "leftlight", "noglasses",␣
 ↪"normal", "rightlight", "sad", "sleepy", "surprised", "wink"]
    prefix = "subject"
    suffix = ".png"

    images = []
    labels = []
```

```
    for person_id in ids:
        for state in states:
            filename = prefix + str(person_id).zfill(2) + "." + state + suffix
            image = cv2.imread(path + filename, cv2.IMREAD_GRAYSCALE)
            images.append(image)
            labels.append(states.index(state))

    return np.array(images), np.array(labels)
```

[ ]:
```
vd3_images, vd3_labels = read_data()
```

### 3.2 Phân chia train:test tỷ lệ 8:2

[ ]:
```
from sklearn.model_selection import train_test_split

# Phân chia dữ liệu thành train và test
vd3_X_train, vd3_X_test, vd3_Y_train, vd3_Y_test = train_test_split(vd3_images,␣
 ↪vd3_labels, test_size=0.2, random_state=42)

# In kích thước của tập train và test
print("Kích thước tập train:", vd3_X_train.shape)
print("Kích thước tập test:", vd3_X_test.shape)
```

```
Kích thước tập train: (132, 243, 320)
Kích thước tập test: (33, 243, 320)
```

[ ]:
```
# Xử lý dữ liệu để cho vào lớp input
# Prepare data for CNN
import cv2

# Resize ảnh về kích thước mới (128x128)
vd3_resized_X_train = np.array(
    [cv2.resize(image, (224, 224)) for image in vd3_X_train]
)
vd3_resized_X_test = np.array(
    [cv2.resize(image, (224, 224)) for image in vd3_X_test]
)

# Reshape dữ liệu
vd3_cnn_X_train = vd3_resized_X_train.reshape(
    -1, 224, 224, 1
)  # Add channel dimension for grayscale images
vd3_cnn_X_train = vd3_cnn_X_train.astype('float32') / 255.0  # Normalize pixel␣
 ↪values to [0, 1]   # ảnh vốn đã là ảnh xám

vd3_cnn_X_test = vd3_resized_X_test.reshape(-1, 224, 224, 1)
```

```
vd3_cnn_X_test = vd3_cnn_X_test.astype('float32') / 255.0

vd3_cnn_Y_train = tf.keras.utils.to_categorical(vd3_Y_train, num_classes=11)
vd3_cnn_Y_test = tf.keras.utils.to_categorical(vd3_Y_test, num_classes=11)
```

[ ]:
```
print(vd3_cnn_X_train[0])
```

```
[[[0.5294118 ]
  [0.5294118 ]
  [0.5294118 ]
  …
  [1.        ]
  [1.        ]
  [1.        ]]

 [[1.        ]
  [1.        ]
  [1.        ]
  …
  [1.        ]
  [1.        ]
  [1.        ]]

 [[1.        ]
  [1.        ]
  [0.99215686]
  …
  [1.        ]
  [1.        ]
  [1.        ]]

 …

 [[1.        ]
  [1.        ]
  [1.        ]
  …
  [0.93333334]
  [0.9254902 ]
  [0.88235295]]

 [[1.        ]
  [1.        ]
  [1.        ]
  …
  [0.9372549 ]
  [0.94509804]
  [0.8235294 ]]
```

```
[[0.29803923]
 [0.29803923]
 [0.29803923]
 …
 [0.29411766]
 [0.29411766]
 [0.2901961 ]]]
```

## 3.3 Huấn luyện CNN (kiến trúc như của Ví dụ 2)

### 3.3.1 Thiết lập kiến trúc CNN

[ ]:
```
# Trains a model to classify images of 3 classes: cat, dog, and panda
def gen_model():
    # Defines & compiles the model
    model = tf.keras.models.Sequential(
        [
            # layer 1
            tf.keras.layers.Conv2D(
                2**4,
                (5, 5),
                activation="relu",
                input_shape=(224, 224, 1),
                padding="same",
            ),
            tf.keras.layers.MaxPooling2D(5, 5),
            tf.keras.layers.Dropout(0.05),
            # layer 2
            tf.keras.layers.Conv2D(2**5, (3, 3), activation="relu",␣
 ↪padding="same"),
            # tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.Dropout(0.1),
            tf.keras.layers.Conv2D(2**5, (3, 3), activation="relu",␣
 ↪padding="same"),
            # tf.keras.layers.MaxPooling2D(2, 2),
            tf.keras.layers.AveragePooling2D(3, 3),
            tf.keras.layers.Dropout(0.15),
            # layer 3
            tf.keras.layers.Conv2D(2**6, (3, 3), activation="relu",␣
 ↪padding="same"),
            tf.keras.layers.Conv2D(2**6, (3, 3), activation="relu",␣
 ↪padding="same"),
            tf.keras.layers.AveragePooling2D(3, 3),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dropout(0.2),
            # layer 4
```

```
            tf.keras.layers.Dense(2**7, activation="relu"),
            tf.keras.layers.Dropout(0.25),
            tf.keras.layers.Dense(2**7, activation="relu"),
            tf.keras.layers.Dropout(0.3),
            tf.keras.layers.Dense(11, activation="softmax"),
        ]
    )

    # from tensorflow.keras.optimizers import RMSprop

    model.compile(loss="categorical_crossentropy", optimizer="adam",␣
↪metrics=["acc"])
    return model
```

### 3.3.2 Huấn luyện CNN

[ ]:
```
vd3_cnn_model = gen_model()
# Train the model
vd3_cnn_model.fit(vd3_cnn_X_train, vd3_cnn_Y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
5/5 [==============================] - 9s 742ms/step - loss: 2.4015 - acc:
0.0833
Epoch 2/10
5/5 [==============================] - 3s 629ms/step - loss: 2.3997 - acc:
0.0758
Epoch 3/10
5/5 [==============================] - 3s 604ms/step - loss: 2.3979 - acc:
0.1364
Epoch 4/10
5/5 [==============================] - 3s 637ms/step - loss: 2.3973 - acc:
0.0758
Epoch 5/10
5/5 [==============================] - 5s 959ms/step - loss: 2.3905 - acc:
0.1061
Epoch 6/10
5/5 [==============================] - 6s 1s/step - loss: 2.3931 - acc: 0.1364
Epoch 7/10
5/5 [==============================] - 6s 1s/step - loss: 2.3935 - acc: 0.1061
Epoch 8/10
5/5 [==============================] - 7s 1s/step - loss: 2.3837 - acc: 0.1591
Epoch 9/10
5/5 [==============================] - 4s 708ms/step - loss: 2.3896 - acc:
0.1136
Epoch 10/10
5/5 [==============================] - 5s 822ms/step - loss: 2.3940 - acc:
0.1364
```

[ ]:
```
<keras.src.callbacks.History at 0x7a85f2b9ba10>
```

Test accuracy dùng hàm evaluate của model

[ ]:
```
# Save the model weights
vd3_cnn_model.save('vd3_cnn_model.keras')

# Evaluate the model on test set
test_loss, test_acc = vd3_cnn_model.evaluate(vd3_cnn_X_test, vd3_cnn_Y_test)
print("Test accuracy:", test_acc)
```

```
2/2 [==============================] - 1s 14ms/step - loss: 2.4565 - acc: 0.1212
Test accuracy: 0.12121212482452393
```

Nhận xét thấy thay đổi kiến trúc mạng đã giúp accu tăng 3% là 9%. Bỏ padding = same đi thì giảm 3% về lại 6%. Thêm lại padding và tăng size ảnh input từ 150 lên 224 thì độ chính xác giảm từ 9% xuống 3%. Vẫn giữ nguyên size ảnh như trên nhưng bổ sung thêm các dropout sau các lớp convolution thì accu từ 3% lên 6%. Thực hiện điều chỉnh màu của ảnh từ [0, 255] về [0, 1] thì đã tăng lên 12%.

Test accuracy dùng hàm predict và hiện kết quả xem + tính accuracy

[ ]:
```
vd3_y_pred_cnn = vd3_cnn_model.predict(vd3_cnn_X_test)
```

```
2/2 [==============================] - 1s 11ms/step
2/2 [==============================] - 1s 11ms/step
```

[ ]:
```
import numpy as np
# Chuyển đổi dự đoán sang dạng one-hot encoding
vd3_y_pred_cnn_labels = np.argmax(vd3_y_pred_cnn, axis=1)

# Dữ liệu thực tế dạng one-hot
vd3_cnn_Y_test_labels = vd3_cnn_Y_test

# Chuyển đổi dữ liệu thực tế sang dạng nhãn
vd3_cnn_Y_test_labels = np.argmax(vd3_cnn_Y_test_labels, axis=1)

# Tính toán độ chính xác
accuracy = np.mean(vd3_y_pred_cnn_labels == vd3_cnn_Y_test_labels)

print("Accuracy:", accuracy)
```

```
Accuracy: 0.12121212121212122
```

[ ]:
```
vd3_cnn_Y_test_labels
```

[ ]:
```
array([ 3,  5, 10,  0,  7,  7,  4,  7,  2,  2,  8,  8,  4,  0,  2,  8,  0,
        6, 10,  5,  9,  7,  1,  9,  9,  2, 10,  1,  2,  7,  7,  1,  5])
```

[ ]:
```
vd3_y_pred_cnn_labels
```

[ ]:
```
array([3, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
       8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8])
```

### 3.3.3 Giải thích kết quả thu được

Kết quả thu được khá tệ - lý do hợp lý nhất hiện tại có thể kiến trúc thiết kế mạng CNN ở ví dụ 2 là chưa đủ tốt.

### 3.3.4 So sánh với cách dùng Logistic (đã resize lại ảnh) và ANN (đã thực hiện giảm số chiều)

Kết quả so sánh là: Dùng ANN thư viện > Dùng ANN tự custom > Dùng Logistic > Dùng CNN tự custom

**Sử dụng ANN**

[ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# path to the database - change it if needed
path = "../Week7/data/face_data/"

ids = range(1, 16)  # 15 people
states = [
    "centerlight",
    "glasses",
    "happy",
    "leftlight",
    "noglasses",
    "normal",
    "rightlight",
    "sad",
    "sleepy",
    "surprised",
    "wink",
]
prefix = "subject"
surfix = ".png"  # file extension is png

# open one picture to get the image's size
fn = prefix + "01." + states[0] + surfix
im = cv2.imread(path + fn, 0)

h = im.shape[0]  # hight
w = im.shape[1]  # width

D = h * w
N = len(states) * 15
```

```
print(N, D, h, w)

X = np.zeros((D, N))
Y = np.zeros(N)

# collect all data
cnt = 0

# there are 15 people
for person_id in range(1, 16):
    for state in states:

        # get name of each image file
        fn = path + prefix + str(person_id).zfill(2) + "." + state + surfix

        # open the file and read as grey image
        tmp = cv2.imread(fn, cv2.IMREAD_GRAYSCALE)

        # then add image to dataset X
        X[:, cnt] = tmp.reshape(D)
        Y[cnt] = states.index(state)

        cnt += 1
Y = Y.astype(int)
```

```
165 77760 243 320
```

[ ]:
```
# Doing PCA, note that each row is a datapoint
from sklearn.decomposition import PCA

# remain dim. k = 125 - change it!
pca = PCA(n_components=125)

# then apply to data X
pca.fit(X.T)
```

[ ]:
```
PCA(n_components=125)
```

[ ]:
```
# then build projection matrix
U = pca.components_.T
U.shape
```

[ ]:
```
(77760, 125)
```

[ ]:
```
Xhat = np.zeros((D, N))
x_mean = X.mean(1)
for i in range(N):
    Xhat[:,i] = X[:,i] - x_mean[:]
```

```
# Reduced dim. data Z (project of Xhat onto sub-space by Uk - bases)
Z = U.T.dot(Xhat)
Z.shape
```

[ ]: (125, 165)

[ ]:
```
import math


# one-hot coding
from scipy import sparse


def convert_labels(y, C):
    Y = sparse.coo_matrix(
        (np.ones_like(y), (y, np.arange(len(y)))), shape=(C, len(y))
    ).toarray()
    return Y


# softmax for multi-class
def softmax(V):
    e_V = np.exp(V - np.max(V, axis=0, keepdims=True))
    # print(np.max(V, axis = 0, keepdims = True))
    Z = e_V / e_V.sum(axis=0)
    return Z


# definition of ReLU, or you can use maximum directly
def ReLU(V):
    return np.maximum(V, 0)


# cost or loss function
def cost(Y, Yhat):
    return -np.sum(Y * np.log(Yhat)) / Y.shape[1]
```

[ ]:
```
# Gradient Descent Loop
# To use this code easily, we put it to a method


def ANN_3layer_SolveClassification(X_train, Y_train, eta, max_count, num):
    X, Y = X_train, Y_train
    d, N = X_train.shape
    C = Y_train.shape[0]
    d1, d2 = num, C
```

```
    # make random data
    W1 = 0.01 * np.random.randn(d, d1)
    b1 = np.zeros((d1, 1))
    W2 = 0.01 * np.random.randn(d1, d2)
    b2 = np.zeros((d2, 1))

    for i in range(max_count + 1):
        ## Feedforward
        Z1 = np.dot(W1.T, X) + b1
        A1 = ReLU(Z1)
        Z2 = np.dot(W2.T, A1) + b2
        Yhat = softmax(Z2)

        # print loss after each 10 iterations
        if i % 100 == 0:
            loss = cost(Y, Yhat)
            print("iter %d, loss: %f" % (i, loss))
            # print(i)
        # backpropagation
        E2 = (Yhat - Y) / N
        dW2 = np.dot(A1, E2.T)
        db2 = np.sum(E2, axis=1, keepdims=True)
        E1 = np.dot(W2, E2)
        E1[Z1 <= 0] = 0  # gradient of ReLU
        dW1 = np.dot(X, E1.T)
        db1 = np.sum(E1, axis=1, keepdims=True)

        # Gradient Descent update
        W1 += -eta * dW1
        b1 += -eta * db1
        W2 += -eta * dW2
        b2 += -eta * db2

    return W1, W2, b1, b2


def predict(W1, W2, b1, b2, images):
    Z1 = np.dot(W1.T, images) + b1
    A1 = ReLU(Z1)
    Z2 = np.dot(W2.T, A1) + b2
    a = softmax(Z2)
    # return  np.argmax(Z2, axis=0) #
    return np.argmax(a, axis=0)
```

[ ]:
```
# 80% for training set - You can change this rate by yourselves
M = (int)(Z.shape[1] * 0.95)
X_train = Z[:, :M]
```

```
Y_train = convert_labels(Y[:M], 11)

print(X_train.shape)
print(Y_train.shape)

W1, W2, b1, b2 = ANN_3layer_SolveClassification(X_train, Y_train, 1e-6, 4000,␣
 ↪num=400)
```

```
(125, 156)
(11, 156)
iter 0, loss: 42.121428
iter 100, loss: 14.928752
iter 200, loss: 8.801731
iter 300, loss: 6.031575
iter 400, loss: 4.301640
iter 500, loss: 3.060789
iter 600, loss: 2.200507
iter 700, loss: 1.605015
iter 800, loss: 1.186314
iter 900, loss: 0.893570
iter 1000, loss: 0.672202
iter 1100, loss: 0.529199
iter 1200, loss: 0.444232
iter 1300, loss: 0.391143
iter 1400, loss: 0.351993
iter 1500, loss: 0.320271
iter 1600, loss: 0.293608
iter 1700, loss: 0.271251
iter 1800, loss: 0.252182
iter 1900, loss: 0.235812
iter 2000, loss: 0.222000
iter 2100, loss: 0.210390
iter 2200, loss: 0.200647
iter 2300, loss: 0.192542
iter 2400, loss: 0.185539
iter 2500, loss: 0.179534
iter 2600, loss: 0.174329
iter 2700, loss: 0.169793
iter 2800, loss: 0.165838
iter 2900, loss: 0.162336
iter 3000, loss: 0.159223
iter 3100, loss: 0.156425
iter 3200, loss: 0.153901
iter 3300, loss: 0.151610
iter 3400, loss: 0.149526
iter 3500, loss: 0.147612
iter 3600, loss: 0.145883
iter 3700, loss: 0.144284
```

```
iter 3800, loss: 0.142798
iter 3900, loss: 0.141414
iter 4000, loss: 0.140125
```

[ ]:
```python
from sklearn.metrics import accuracy_score  # for evaluating results

Y_pred_train = predict(W1, W2, b1, b2, X_train)
print("accuracy training data: ", accuracy_score(Y[:M], Y_pred_train))

X_val = Z[:, M:]
Y_val = convert_labels(Y[M:], 11)


Y_pred_val = predict(W1, W2, b1, b2, X_val)
print(
    X_val.shape,
    Y_pred_val.shape,
    "accuracy validation data: ",
    accuracy_score(Y[M:], Y_pred_val),
)
print(Y[M:])
print(Y_pred_val)
```

```
accuracy training data:  0.9294871794871795
(125, 9) (9,) accuracy validation data:  0.3333333333333333
[ 2  3  4  5  6  7  8  9 10]
[1 3 8 7 6 1 8 2 1]
```

**Sử dụng Logistic**

[ ]:
```python
from sklearn.naive_bayes import GaussianNB

NB_model = GaussianNB()
NB_model.fit(X_train.T, Y[:M].T)

Y_predict_nb = NB_model.predict(X_train.T)
accuracy_nb = accuracy_score(Y[:M].T, Y_predict_nb)
print("Train Accuracy score using Naive Bayes: ", accuracy_nb)

Y_predict_val_nb = NB_model.predict(X_val.T)
accuracy_val_nb = accuracy_score(Y[M:].T, Y_predict_val_nb)
print("Validation Accuracy score using Naive Bayes: ", accuracy_val_nb)
```

```
Train Accuracy score using Naive Bayes:  0.8782051282051282
Validation Accuracy score using Naive Bayes:  0.2222222222222222
```

[ ]:
```python
from sklearn.linear_model import LogisticRegression
```

```
lorg=LogisticRegression(multi_class='multinomial',solver='sag', max_iter=5000,␣
 ↪penalty='none')
lorg.fit(X_train.T,Y[:M].T)

Y_pred_softmax =lorg.predict(X_train.T)
accuracy_LSM = accuracy_score(Y[:M].T, Y_pred_softmax)
print("Train Accuracy score using MultiNomial Logistic Reg.: ", accuracy_LSM)

Y_pred_val_softmax =lorg.predict(X_val.T)
accuracy_val_LSM = accuracy_score(Y[M:].T, Y_pred_val_softmax)
print("Train Accuracy score using MultiNomial Logistic Reg.: ",␣
 ↪accuracy_val_LSM)
```

```
Train Accuracy score using MultiNomial Logistic Reg.:  0.9423076923076923
Train Accuracy score using MultiNomial Logistic Reg.:  0.2222222222222222
```

**Sử dụng ANN MLPClassifier từ thư viện**

[ ]:
```
# Import Neural Network
from sklearn.neural_network import MLPClassifier
ann = MLPClassifier()

ann.fit(Z.T, Y)
```

[ ]:
```
MLPClassifier()
```

[ ]:
```
y_pred = ann.predict(Z.T)
y_pred
```

[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  5,  5,
        6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  4,  6,  7,  8,  9, 10,  0,
        1,  2,  3,  4,  5,  6,  5,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,
        7,  8,  9, 10,  0,  1,  2,  3,  5,  5,  6,  7,  8,  9, 10,  0,  1,
        2,  3,  5,  5,  6,  7,  8,  9, 10,  0,  5,  2,  3,  4,  5,  6,  7,
        8,  9, 10,  0,  1,  2,  3,  4,  4,  6,  7,  8,  9, 10,  0,  1,  2,
        3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,
        9, 10,  0,  1,  2,  3,  5,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,
        4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  5,  5,  6,  7,  8,  9,
       10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[ ]:
```
Y
```

[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,
        6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,
        1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,
        7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,
        2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,
        8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,
        3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,
```

```
 9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,
 4,  5,  6,  7,  8,  9, 10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
10,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[ ]:
```
from sklearn.metrics import accuracy_score # for evaluating results
print(accuracy_score(Y, y_pred))
```

```
0.9454545454545454
```