Prabal Ghosh

Assignment - CNN

TP CNN

Diane LINGRAND

diane.lingrand@univ-cotedazur.fr (mailto:diane.lingrand@univ-cotedazur.fr)
Polytech - SI4 - 2021

Introduction

In [ ]:
```
from IPython.display import Image
import tensorflow as tf
print(tf.__version__)
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Activation
import matplotlib.pyplot as plt
```

```
2.15.0
```

The GPU

To enable GPU backend in Google colab for your notebook:

1. Runtime (top left corner) -> Change runtime type
2. Put GPU as "Hardware accelerator"
3. Save.

Or run the next cell:

In [ ]:
```
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

Convolutional Neural Networks (CNN)

Derived from the MLP, a convolutional neural network (CNN) is a type of artificial neural network that is specifically designed to process **pixel data**. The layers of a CNN consist of an **input layer**, an **output layer** and **hidden layers** that can include **convolutional layers**,

pooling layers, **fully connected layers** and **normalization layers**. It exists a lot of techniques to optimize CNN, like for example the dropout.

Loading the dataset

In this part, we will use photographies of animals from the kaggle dataset animals-10 (https://www.kaggle.com/alessiocorrado99/animals10). Please connect to their site before loading the dataset from this zip file (http://www.i3s.unice.fr/~lingrand/raw-img.zip). Decompress the zip file on your disk.

If you are using google colab, there is no need to download the dataset because I have a copy on my drive. You just need add to your drive this shared folder: https://drive.google.com/drive/folders/15cB1Ky-7OTUqfcQDZZyzc5HArt0GA6Sm?usp=sharing (https://drive.google.com/drive/folders/15cB1Ky-7OTUqfcQDZZyzc5HArt0GA6Sm?usp=sharing) You need to click on the link and click on "Add shortcut to Drive" and then select "My Drive".

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remou
nt, call drive.mount("/content/drive/", force_remount=True).
```

To feed the data to a CNN, we need to shape it as required by Keras. As input, a 2D convolutional layer needs a **4D tensor** with shape: **(batch, rows, cols, channels)**. Therefore, we need to precise the "channels" axis, which can be seen as the number of level of color of each input: 3 channels in our case. We will fix the dimension of images according to the VGG-16 network: (224, 224).

In [ ]:
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, MaxP
from sklearn.metrics import confusion_matrix, confusion_matrix, f1_scor
import tensorflow.keras
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
import glob
# when processing time is long, it's nice to see the progress bar
#!pip install tqdm
from tqdm import tqdm
```

In [ ]:
```
```

loading train data

Please read the code before running any of the cells!

In [ ]:

```
#datasetRoot='/home/lingrand/Ens/MachineLearning/animals/raw-img/'
#datasetRoot='/whereYouPutTheImages/'
datasetRoot='/content/drive/My Drive/raw-img/'
# I suggest to reduce the number of classes for a first trial.
# If you finish this notebook before the end of the course, you can ada
# classes = ['mucca', 'elefante', 'gatto', 'cavallo', 'scoiattolo', 'ra
classes = ['mucca','elefante','gatto']
nbClasses = len(classes)  # 4

#training data

rootTrain = datasetRoot+'train/'
# classLabel = 0
reducedSizePerClass = 200 #in order to reduce the number of images per
# reducedSizePerClass = 100
def readImages(path, classes, number):
    from sklearn.utils import shuffle
    dims = 224
    X = []
    y = []
    classLabel = 0
    for cl in classes:
        listImages = glob.glob(path+cl+'/*')
        for pathImg in tqdm(listImages[:number]):
            img = image.load_img(pathImg, target_size=(dims, dims))
            im = image.img_to_array(img)
            #im = preprocess_input(im)
            X += [im]
            y += [classLabel]
        classLabel += 1
    return shuffle(np.array(X), np.array(y).reshape(-1))


# xTrain,yTrain= readImages(rootTrain,classes,reducedSizePerClass)
xTrain,yTrain= readImages(rootTrain,classes,reducedSizePerClass)
```

```
100%|██████████| 200/200 [00:03<00:00, 51.91it/s]
100%|██████████| 200/200 [00:02<00:00, 94.73it/s]
100%|██████████| 200/200 [00:01<00:00, 123.60it/s]
```

In [ ]:

In [ ]:
```
asetRoot='/home/lingrand/Ens/MachineLearning/animals/raw-img/'
asetRoot='/whereYouPutTheImages/'
setRoot='/content/drive/My Drive/raw-img/'
suggest to reduce the number of classes for a first trial.
 you finish this notebook before the end of the course, you can add more
asses = ['mucca', 'elefante', 'gatto', 'cavallo', 'scoiattolo', 'ragno',
es = ['mucca','elefante','gatto', 'cavallo']
asses = len(classes)  # 4

ining data

Train = datasetRoot+'train/'
sLabel = 0
ducedSizePerClass = 200 #in order to reduce the number of images per cla
edSizePerClass = 100 #in order to reduce the number of images per class

Img = nbClasses * reducedSizePerClass
n = np.empty(shape=(totalImg,224,224,3))
n = []
 = True

l in classes:
istImages = glob.glob(rootTrain+cl+'/*')
Train += [classLabel]*reducedSizePerClass #len(listImages) # note that
or pathImg in tqdm(listImages[:reducedSizePerClass]): # and here, we ha
    img = image.load_img(pathImg, target_size=(224,224))
    im = image.img_to_array(img)
    im = np.expand_dims(im, axis=0)
    im = preprocess_input(im)
    xTrain[i,:,:,:] = im
    i += 1
lassLabel += 1
(len(yTrain))
(xTrain.shape)
n = tensorflow.keras.utils.to_categorical(yTrain, nbClasses)
```

In [ ]:
```
yTrain.shape
```

Out[7]: (600,)

Normalize

In [ ]:
```
# xTest = xTest/np.max(xTrain)
xTrain = xTrain/np.max(xTrain)
```

[TO DO - Students] What is the dimension of xTrain ? What do those dimensions represent ?

In [ ]:
```
#(2000, 224, 224, 3)------> there are batch =2000 image data with (heig
```

These are the dimensions of xTrain: (600, 224, 224, 3)

Above dimensions represent:

600 represent the number of images stored in xTrain, where we are working with 3 classes and from each class we have imported 200 images.

224,224 represents the dimensions each images have (rows X columns).

3 represet the channel (dimensions of image).As we have RGB(Red,Green,Blue) images, so we have defined the channel 3.

[TO DO - Students] Complete the following code to plot a few training images

In [ ]:
```
import matplotlib.pyplot as plt

square = 8
ix = 1
fig, axs = plt.subplots(square, square, figsize=(20, 20))
for i in range(square):
    for j in range(square):
        # specify subplot and turn of axis
        ax = axs[i,j]
        ax.set_xticks([])
        ax.set_yticks([])
        im = xTrain[ix]
        ax.imshow(im)
        ix += 1
plt.show()
```

In [ ]:

In [ ]:

In order to speed-up the time spent on this part of the lab, you may have noticed that we reduced the number of classes and the number of images per class. You can change these few lines of code if you want to work on the whole dataset.

loading test data

In [ ]:

```
# #you need to use the same classes for the test dataset than for the t
# rootTest = datasetRoot+'test/'
# classLabel = 0

# totalTestImg = 0
# for cl in classes:
#     totalTestImg += len(glob.glob(rootTest+cl+'/*'))

# print("There are ",totalTestImg, " images in test dataset.")
# xTest = np.empty(shape=(totalTestImg,224,224,3))
# yTest = []
# i = 0

# for cl in classes:
#     listImages = glob.glob(rootTest+cl+'/*')
#     yTest += [classLabel]*len(listImages)
#     for pathImg in listImages:
#         img = image.load_img(pathImg, target_size=(224, 224))
#         im = image.img_to_array(img)
#         im = np.expand_dims(im, axis=0)
#         im = preprocess_input(im)
#         xTest[i,:,:,:] = im
#     classLabel += 1
# print(len(yTest))
# print(xTest.shape)
# yTest = tensorflow.keras.utils.to_categorical(yTest, nbClasses)
```

In [ ]:
```
#you need to use the same classes for the test dataset than for the tra
rootTest = datasetRoot+'test/'
classLabel = 0

totalTestImg = 0
for cl in classes:
    totalTestImg += len(glob.glob(rootTest+cl+'/*'))

print("There are ",totalTestImg, " images in test dataset.")
xTest = np.empty(shape=(totalTestImg,224,224,3))
yTest = []
i = 0



def readImages(path, classes, number):
    from sklearn.utils import shuffle
    dims = 224
    X = []
    y = []
    classLabel = 0
    for cl in classes:
        listImages = glob.glob(path+cl+'/*')
        for pathImg in tqdm(listImages[:number]):
            img = image.load_img(pathImg, target_size=(dims, dims))
            im = image.img_to_array(img)
            #im = preprocess_input(im)
            X += [im]
            y += [classLabel]
        classLabel += 1
    return shuffle(np.array(X), np.array(y).reshape(-1))

xTest,yTest = readImages(rootTest, classes, 200)
print(len(yTest))
print(xTest.shape)
```

```
There are  991  images in test dataset.

100%|██████████| 200/200 [00:02<00:00, 89.17it/s]
100%|██████████| 200/200 [00:01<00:00, 100.40it/s]
100%|██████████| 200/200 [00:01<00:00, 164.95it/s]

600
(600, 224, 224, 3)
```

In [ ]:
```
xTest = xTest/np.max(xTrain)
```

Build your own CNN network

[TO DO - Students] Start with the simplest CNN: 1 conv2D layer + 1 pooling + 1 dense layer. Fill the gaps and explain the parameters of the MaxPooling2D layer

In [ ]:
```
# model = Sequential()
# # model.add(Conv2D(32,(3,3),padding='same',activation='relu', input_s
# model.add(Conv2D(32,(3,3),padding='same',activation='relu', input_sha

# model.add(MaxPooling2D(pool_size=(4, 4), strides=4, padding='same'))
# model.add(Flatten())
# # model.add(Dense(..., activation=...))
# model.add(Dense(4, activation='softmax'))  # nbclasses= 4

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

# model.compile(optimizer='rmsprop', loss='categorical_crossentropy', m
```

In [ ]:
```
from keras import layers
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
```

In [ ]:
```
model = Sequential()
model.add(Input(shape=xTrain.shape[1:]))
model.add(Conv2D(16,(3,3),padding='valid', activation='relu'))  # filte
model.add(MaxPooling2D(pool_size=(2, 2),padding='valid'))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(nbClasses, activation='softmax'))  # nbclasses= 4
```

Let's look at the dimension of all inputs and outputs:

$\frac{224-3}{1}+1 = 222$

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 max_pooling2d (MaxPooling2  (None, 111, 111, 16)      0
 D)

 flatten (Flatten)           (None, 197136)            0

 dense (Dense)               (None, 1024)              201868288

 dense_1 (Dense)             (None, 3)                 3075

=================================================================
Total params: 201871811 (770.08 MB)
Trainable params: 201871811 (770.08 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[TO DO - Students] Train and test this network.

In [ ]:
```
## Your code here
# import keras
# optimizer = keras.optimizers.Adam(learning_rate=0.1)
```

In [ ]:
```

```

In [ ]:
```
# model.compile(optimizer=optimizer, loss='categorical_crossentropy',me
```

In [ ]:
```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
```

In [ ]:
```
# callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patie
# history=model.fit(xTrain, yTrain, epochs=40,validation_split=0.2,call
```

In [ ]:

```
# xTrain, yTrain = shuffle(xTrain, yTrain)
history = model.fit(xTrain, yTrain, epochs=100, validation_split=0.1,sh
                    callbacks=[EarlyStopping(monitor="val_loss", mode="
```

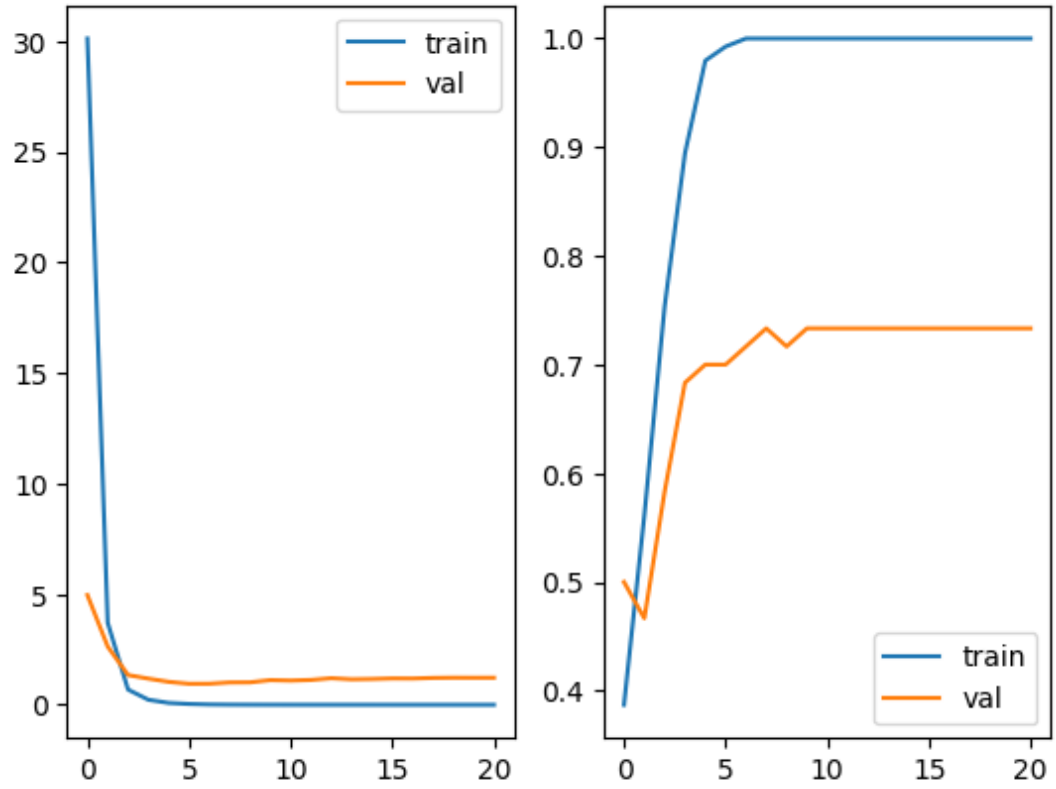
```
Epoch 1/100
17/17 [==============================] - 4s 113ms/step - loss: 30.1188
- accuracy: 0.3870 - val_loss: 4.9607 - val_accuracy: 0.5000
Epoch 2/100
17/17 [==============================] - 1s 76ms/step - loss: 3.7073 -
accuracy: 0.5611 - val_loss: 2.6331 - val_accuracy: 0.4667
Epoch 3/100
17/17 [==============================] - 1s 79ms/step - loss: 0.6888 -
accuracy: 0.7537 - val_loss: 1.3458 - val_accuracy: 0.5833
Epoch 4/100
17/17 [==============================] - 1s 83ms/step - loss: 0.2285 -
accuracy: 0.8944 - val_loss: 1.1846 - val_accuracy: 0.6833
Epoch 5/100
17/17 [==============================] - 1s 87ms/step - loss: 0.0847 -
accuracy: 0.9796 - val_loss: 1.0383 - val_accuracy: 0.7000
Epoch 6/100
17/17 [==============================] - 1s 82ms/step - loss: 0.0381 -
accuracy: 0.9926 - val_loss: 0.9519 - val_accuracy: 0.7000
Epoch 7/100
17/17 [==============================] - 1s 81ms/step - loss: 0.0156 -
accuracy: 1.0000 - val_loss: 0.9536 - val_accuracy: 0.7167
Epoch 8/100
17/17 [==============================] - 1s 77ms/step - loss: 0.0082 -
accuracy: 1.0000 - val_loss: 1.0111 - val_accuracy: 0.7333
Epoch 9/100
17/17 [==============================] - 1s 76ms/step - loss: 0.0050 -
accuracy: 1.0000 - val_loss: 1.0152 - val_accuracy: 0.7167
Epoch 10/100
17/17 [==============================] - 1s 76ms/step - loss: 0.0035 -
accuracy: 1.0000 - val_loss: 1.1164 - val_accuracy: 0.7333
Epoch 11/100
17/17 [==============================] - 1s 76ms/step - loss: 0.0026 -
accuracy: 1.0000 - val_loss: 1.0968 - val_accuracy: 0.7333
Epoch 12/100
17/17 [==============================] - 1s 76ms/step - loss: 0.0020 -
accuracy: 1.0000 - val_loss: 1.1241 - val_accuracy: 0.7333
Epoch 13/100
17/17 [==============================] - 1s 77ms/step - loss: 0.0017 -
accuracy: 1.0000 - val_loss: 1.2030 - val_accuracy: 0.7333
Epoch 14/100
17/17 [==============================] - 1s 77ms/step - loss: 0.0014 -
accuracy: 1.0000 - val_loss: 1.1537 - val_accuracy: 0.7333
Epoch 15/100
17/17 [==============================] - 1s 79ms/step - loss: 0.0011 -
accuracy: 1.0000 - val_loss: 1.1667 - val_accuracy: 0.7333
Epoch 16/100
17/17 [==============================] - 1s 81ms/step - loss: 9.7677e-
04 - accuracy: 1.0000 - val_loss: 1.1925 - val_accuracy: 0.7333
Epoch 17/100
17/17 [==============================] - 1s 84ms/step - loss: 8.4063e-
04 - accuracy: 1.0000 - val_loss: 1.1892 - val_accuracy: 0.7333
Epoch 18/100
17/17 [==============================] - 1s 84ms/step - loss: 7.4523e-
04 - accuracy: 1.0000 - val_loss: 1.2163 - val_accuracy: 0.7333
Epoch 19/100
17/17 [==============================] - 1s 81ms/step - loss: 6.5440e-
04 - accuracy: 1.0000 - val_loss: 1.2258 - val_accuracy: 0.7333
Epoch 20/100
17/17 [==============================] - 1s 76ms/step - loss: 5.7588e-
04 - accuracy: 1.0000 - val_loss: 1.2231 - val_accuracy: 0.7333
Epoch 21/100
```

```
17/17 [==============================] - 1s 77ms/step - loss: 5.1261e-
04 - accuracy: 1.0000 - val_loss: 1.2243 - val_accuracy: 0.7333
```

In [ ]:

[TO DO - Students] Plot the training metrics (loss and accuracy). Test the model on the test data and compare the confusion matrix on the test data and train data

In [ ]:
```
# Plot history
f, (ax1, ax2) = plt.subplots(1,2)
ax1.plot(history.history['loss'], label='train')
ax1.plot(history.history['val_loss'], label='val')
ax1.legend()
ax2.plot(history.history['accuracy'], label='train')
ax2.plot(history.history['val_accuracy'], label='val')
ax2.legend()
plt.show()
```



In [ ]:
```
# for you !
from sklearn.metrics import f1_score

score = model.evaluate(xTest,yTest)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

ypred = np.argmax(model.predict(xTest), axis=1)
print("F1 score: ", f1_score(ypred, yTest,average='micro'))
```

```
19/19 [==============================] - 0s 22ms/step - loss: 317.2351
- accuracy: 0.6667
accuracy: 66.67%
19/19 [==============================] - 0s 14ms/step
F1 score:  0.6666666666666666
```

In [ ]:
```
yTest.shape, ypred.shape
```

Out[25]:
```
((600,), (600,))
```

Visualize the confusion matrix on the test dataset for this model

In [ ]:
```
yTest.dtype
```

Out[26]:
```
dtype('int64')
```

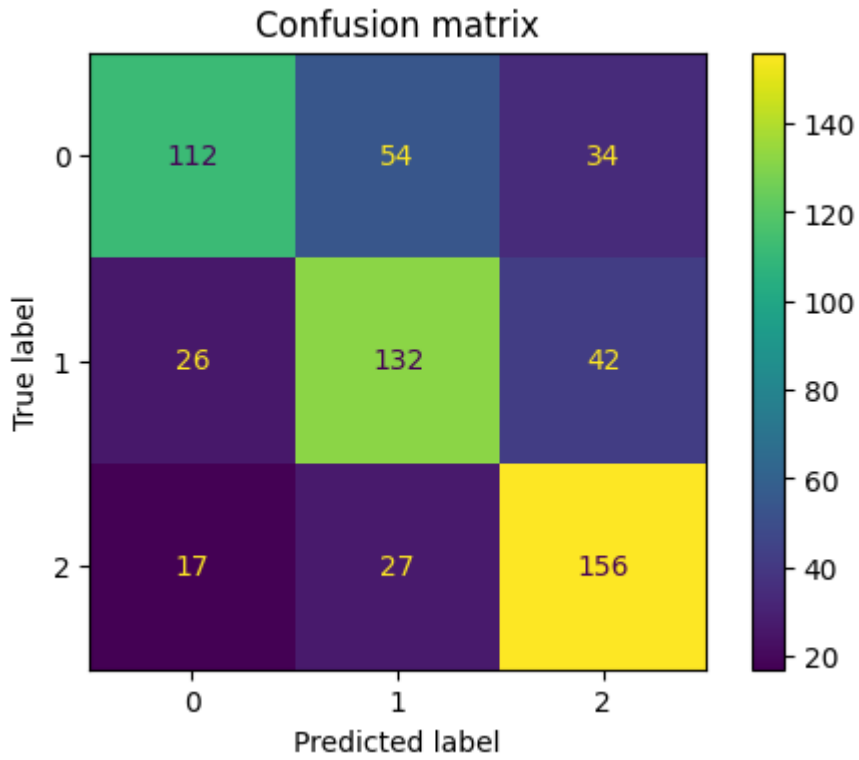
In [ ]:
```
y_pred = np.int64(ypred)
```

In [ ]:
```
# y_pred = model.predict(xTrain)
y_pred.shape
```

Out[28]:
```
(600,)
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix


#test performances


cm = confusion_matrix(yTest, y_pred)

fig, ax = plt.subplots(figsize=(6, 4))  # You can adjust the figsize as

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
# disp.plot()
disp.plot(ax=ax)

disp.ax_.set_title('Confusion matrix')
```

Out[29]: Text(0.5, 1.0, 'Confusion matrix')



In [ ]:
```
# from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
# from sklearn.preprocessing import LabelEncoder
# le = LabelEncoder()
# le.fit(classes)

# cm = confusion_matrix(le.inverse_transform(np.argmax(yTrain, axis=1)),
#                       le.inverse_transform(np.argmax(y_pred, axis=1)),
#                       labels=classes)
# disp = ConfusionMatrixDisplay(confusion_matrix=cm,
#                               display_labels=classes)
# disp.plot()
```

How is the accuracy or F1-measure on the test dataset? **Accuracy is 66.17% and F1 score: 0.6616**

Are you satisfied by the performances? **no after ploting confusion matrix I understood that many 'mucca', 'elefante', 'gatto' are wrongly predicted (False Positive)**

About Dropout

Study this part only if you have time for it. It concerns the previous network but prefer to study first part II and come back here after.

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By â€œignoringâ€□, I mean these units are not considered during a particular forward or backward pass.

Why use dropout ? A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to overfitting of training data.

Let's add dropout and activation functions to the network!

In [ ]:
```
xTrain.shape[1:]
```

Out[31]: (224, 224, 3)

In [ ]:
```

```

In [ ]:
```
from tensorflow.keras.layers import Dropout

model = Sequential()

model.add(Conv2D(256,(3,3),activation='relu',input_shape=(224,224,3)))
# model.add(Input(shape=xTrain.shape[1:]))
# model.add(Conv2D(16,(3,3),padding='valid', activation='relu'))

model.add(GlobalAveragePooling2D())
model.add(Dense(200,activation='relu'))
# adding dropout to the previous layer
model.add(Dropout(0.2))

model.add(Dense(nbClasses, activation='softmax'))

# model.compile(optimizer='rmsprop',loss='categorical_crossentropy', me
# model.summary()
```

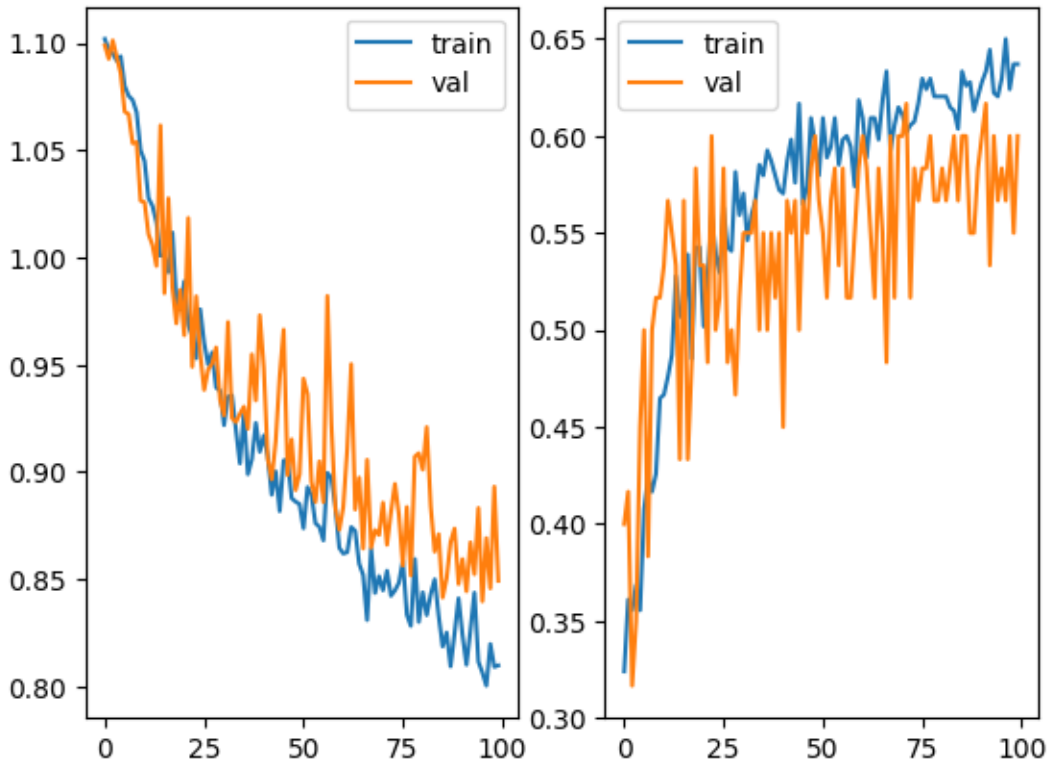
[TO DO - Students] Plot the training metrics (loss and accuracy). Test the model on the test data and compare the confusion matrix on the test data and train data

In [ ]:
```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
```

In [ ]:
```
# xTrain, yTrain = shuffle(xTrain, yTrain)
history = model.fit(xTrain, yTrain, epochs=100, validation_split=0.1,sh
                    callbacks=[EarlyStopping(monitor="val_loss", mode="
```

```
Epoch 1/100
17/17 [==============================] - 8s 243ms/step - loss: 1.1
019 - accuracy: 0.3241 - val_loss: 1.0990 - val_accuracy: 0.4000
Epoch 2/100
17/17 [==============================] - 2s 127ms/step - loss: 1.0
966 - accuracy: 0.3611 - val_loss: 1.0926 - val_accuracy: 0.4167
Epoch 3/100
17/17 [==============================] - 2s 124ms/step - loss: 1.0
951 - accuracy: 0.3556 - val_loss: 1.1012 - val_accuracy: 0.3167
Epoch 4/100
17/17 [==============================] - 2s 129ms/step - loss: 1.0
918 - accuracy: 0.3685 - val_loss: 1.0935 - val_accuracy: 0.3500
Epoch 5/100
17/17 [==============================] - 2s 124ms/step - loss: 1.0
938 - accuracy: 0.3556 - val_loss: 1.0858 - val_accuracy: 0.4500
Epoch 6/100
17/17 [==============================] - 2s 131ms/step - loss: 1.0
799 - accuracy: 0.4074 - val_loss: 1.0680 - val_accuracy: 0.5000
Epoch 7/100
```

In [ ]:
```
f, (ax1, ax2) = plt.subplots(1,2)
ax1.plot(history.history['loss'], label='train')
ax1.plot(history.history['val_loss'], label='val')
ax1.legend()
ax2.plot(history.history['accuracy'], label='train')
ax2.plot(history.history['val_accuracy'], label='val')
ax2.legend()
plt.show()
```

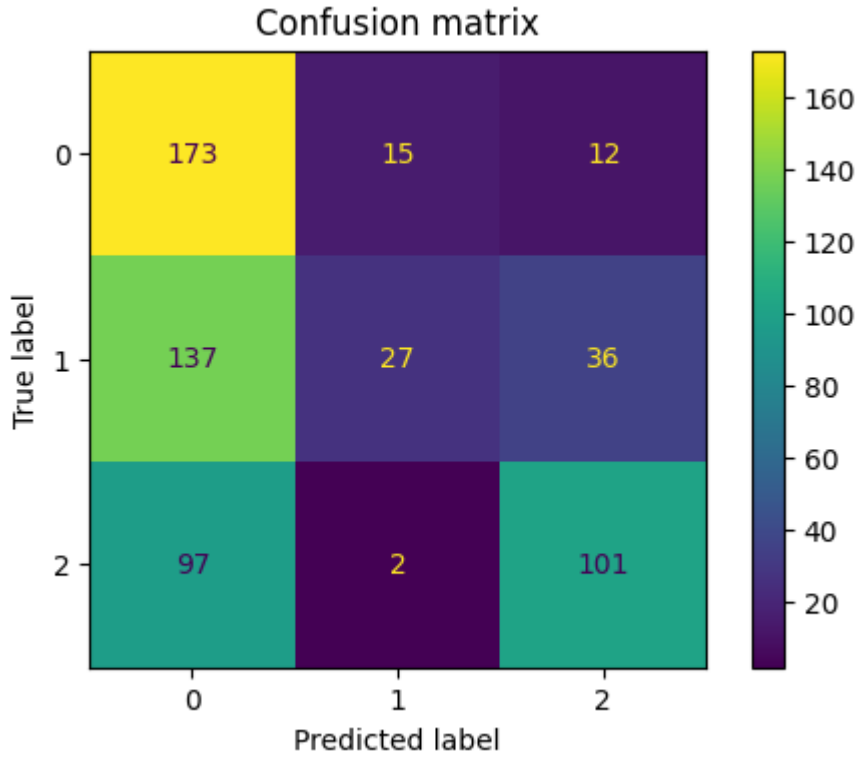


In [ ]:
```
# for you !
from sklearn.metrics import f1_score

score = model.evaluate(xTest,yTest)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

ypred = np.argmax(model.predict(xTest), axis=1)
print("F1 score: ", f1_score(ypred, yTest,average='micro'))
```

```
19/19 [==============================] - 1s 64ms/step - loss: 107.3790
- accuracy: 0.5017
accuracy: 50.17%
19/19 [==============================] - 1s 28ms/step
F1 score:  0.5016666666666667
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix


#test performances
y_pred = np.int64(ypred)


cm = confusion_matrix(yTest, y_pred)

fig, ax = plt.subplots(figsize=(6, 4))  # You can adjust the figsize as

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
# disp.plot()
disp.plot(ax=ax)

disp.ax_.set_title('Confusion matrix')
```

Out[37]: Text(0.5, 1.0, 'Confusion matrix')



In [ ]:

Using a pre-learned network

loading VGG-16 description part and adding layers to build our own classification network

In [ ]:
```
VGGmodel = VGG16(weights='imagenet', include_top=False)
#features = VGGmodel.predict(xTrain)
#print(features.shape)
VGGmodel.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[TO DO - Students] What is the goal of the include_top=false parameter and adapt the model to our classification model by filling the gaps of the following cell

The include_top=False parameter in the VGG16 model indicates that the top (fully connected) layers responsible for image classification are excluded.

This allows us to use the pre-trained convolutional base of VGG16 for feature extraction while customizing the classification part for our specific task.

In the provided code snippet, additional layers, including Global Average Pooling and fully connected layers, are added on top of the VGG16 convolutional base.

These added layers are designed to adapt the model for a specific classification task with a defined number of classes (nbClasses).

The goal is to leverage the learned features from VGG16 and fine-tune the model for a new classification problem.

In [ ]:

```
# we will add layers to this feature extraction part of VGG network
m = VGGmodel.output
# we start with a global average pooling
m = GlobalAveragePooling2D()(m)
# and add a fully-connected layer
m = Dense(1024, activation='relu')(m)
# finally, the softmax layer for predictions (we have nbClasses classes
# predictions = Dense(..., activation=...)(m)

predictions = Dense(nbClasses, activation='softmax')(m)

# global network
model = Model(inputs=VGGmodel.input, outputs=predictions)
```

Can you display the architecture of this entire network?

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

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

 global_average_pooling2d_1  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1024)              525312

 dense_5 (Dense)             (None, 3)                 3075

=================================================================
Total params: 15243075 (58.15 MB)
Trainable params: 15243075 (58.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[TO DO - Students] What would happen if we ran model.fit now ? Make it so that the training will only train the new layers and train the model.

In [ ]:
```
## Your code here
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
```

```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_accuracy',min_delta=0.0001, verbose=0, patience=30,
mode='auto', baseline=None, restore_best_weights=True)

for layer in model.layers[:19]:
  layer.trainable=False

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

history=model.fit(xTrain, yTrain, epochs=50, batch_size=32, validation_split=0.2,callbacks=
[early_stopping],verbose=1)
```

In [ ]:

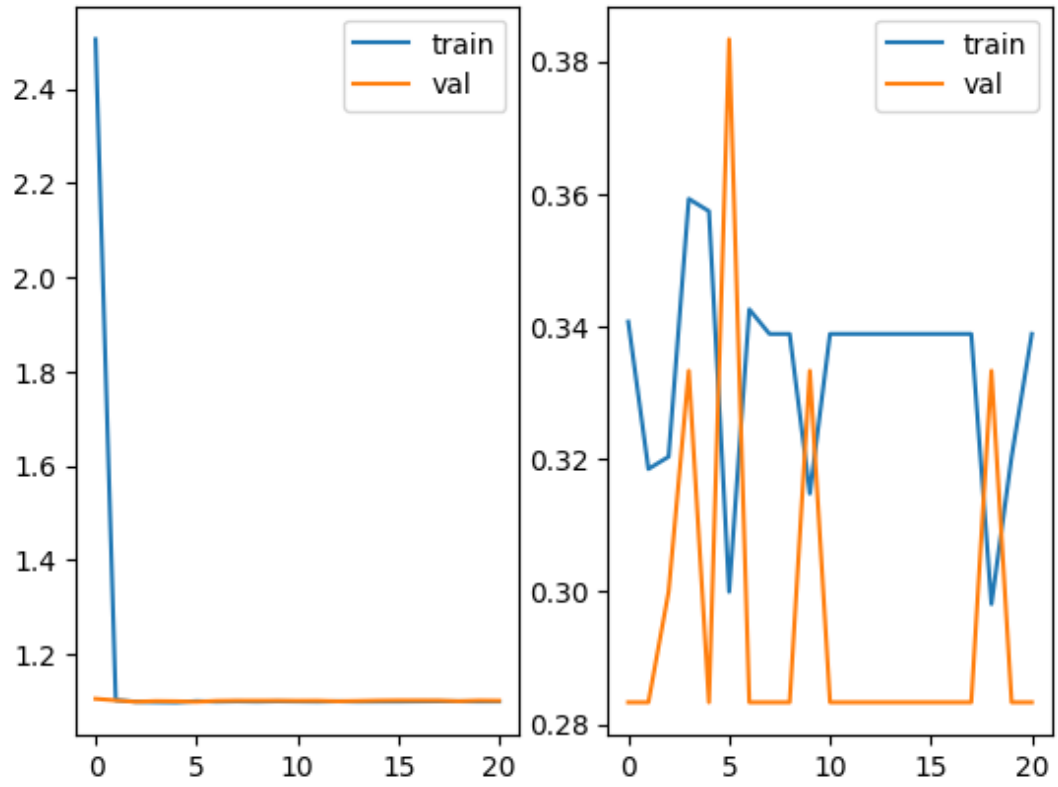
```
# xTrain, yTrain = shuffle(xTrain, yTrain)
history = model.fit(xTrain, yTrain, epochs=100, validation_split=0.1,sh
                    callbacks=[EarlyStopping(monitor="val_loss", mode="
```

```
Epoch 1/100
17/17 [==============================] - 48s 1s/step - loss: 2.5053 -
accuracy: 0.3407 - val_loss: 1.1047 - val_accuracy: 0.2833
Epoch 2/100
17/17 [==============================] - 7s 398ms/step - loss: 1.1020
- accuracy: 0.3185 - val_loss: 1.1014 - val_accuracy: 0.2833
Epoch 3/100
17/17 [==============================] - 7s 395ms/step - loss: 1.0985
- accuracy: 0.3204 - val_loss: 1.0983 - val_accuracy: 0.3000
Epoch 4/100
17/17 [==============================] - 7s 399ms/step - loss: 1.0977
- accuracy: 0.3593 - val_loss: 1.0998 - val_accuracy: 0.3333
Epoch 5/100
17/17 [==============================] - 7s 389ms/step - loss: 1.0972
- accuracy: 0.3574 - val_loss: 1.0993 - val_accuracy: 0.2833
Epoch 6/100
17/17 [==============================] - 7s 394ms/step - loss: 1.0997
- accuracy: 0.3000 - val_loss: 1.0982 - val_accuracy: 0.3833
Epoch 7/100
17/17 [==============================] - 7s 384ms/step - loss: 1.0986
- accuracy: 0.3426 - val_loss: 1.1003 - val_accuracy: 0.2833
Epoch 8/100
17/17 [==============================] - 7s 388ms/step - loss: 1.0992
- accuracy: 0.3389 - val_loss: 1.1008 - val_accuracy: 0.2833
Epoch 9/100
17/17 [==============================] - 7s 390ms/step - loss: 1.0987
- accuracy: 0.3389 - val_loss: 1.1008 - val_accuracy: 0.2833
Epoch 10/100
17/17 [==============================] - 7s 390ms/step - loss: 1.0994
- accuracy: 0.3148 - val_loss: 1.1008 - val_accuracy: 0.3333
Epoch 11/100
17/17 [==============================] - 7s 388ms/step - loss: 1.0990
- accuracy: 0.3389 - val_loss: 1.1005 - val_accuracy: 0.2833
Epoch 12/100
17/17 [==============================] - 7s 386ms/step - loss: 1.0987
- accuracy: 0.3389 - val_loss: 1.1006 - val_accuracy: 0.2833
Epoch 13/100
17/17 [==============================] - 7s 393ms/step - loss: 1.0993
- accuracy: 0.3389 - val_loss: 1.0994 - val_accuracy: 0.2833
Epoch 14/100
17/17 [==============================] - 7s 387ms/step - loss: 1.0987
- accuracy: 0.3389 - val_loss: 1.1001 - val_accuracy: 0.2833
Epoch 15/100
17/17 [==============================] - 7s 388ms/step - loss: 1.0989
- accuracy: 0.3389 - val_loss: 1.1009 - val_accuracy: 0.2833
Epoch 16/100
17/17 [==============================] - 7s 390ms/step - loss: 1.0989
- accuracy: 0.3389 - val_loss: 1.1013 - val_accuracy: 0.2833
Epoch 17/100
17/17 [==============================] - 7s 388ms/step - loss: 1.0992
- accuracy: 0.3389 - val_loss: 1.1012 - val_accuracy: 0.2833
Epoch 18/100
17/17 [==============================] - 7s 390ms/step - loss: 1.0994
- accuracy: 0.3389 - val_loss: 1.1010 - val_accuracy: 0.2833
Epoch 19/100
17/17 [==============================] - 7s 392ms/step - loss: 1.0993
- accuracy: 0.2981 - val_loss: 1.0997 - val_accuracy: 0.3333
Epoch 20/100
17/17 [==============================] - 7s 391ms/step - loss: 1.0990
- accuracy: 0.3204 - val_loss: 1.1011 - val_accuracy: 0.2833
Epoch 21/100
```

```
17/17 [==============================] - 7s 389ms/step - loss: 1.0991
- accuracy: 0.3389 - val_loss: 1.1007 - val_accuracy: 0.2833
```

In [ ]:

```
f, (ax1, ax2) = plt.subplots(1,2)
ax1.plot(history.history['loss'], label='train')
ax1.plot(history.history['val_loss'], label='val')
ax1.legend()
ax2.plot(history.history['accuracy'], label='train')
ax2.plot(history.history['val_accuracy'], label='val')
ax2.legend()
plt.show()
```



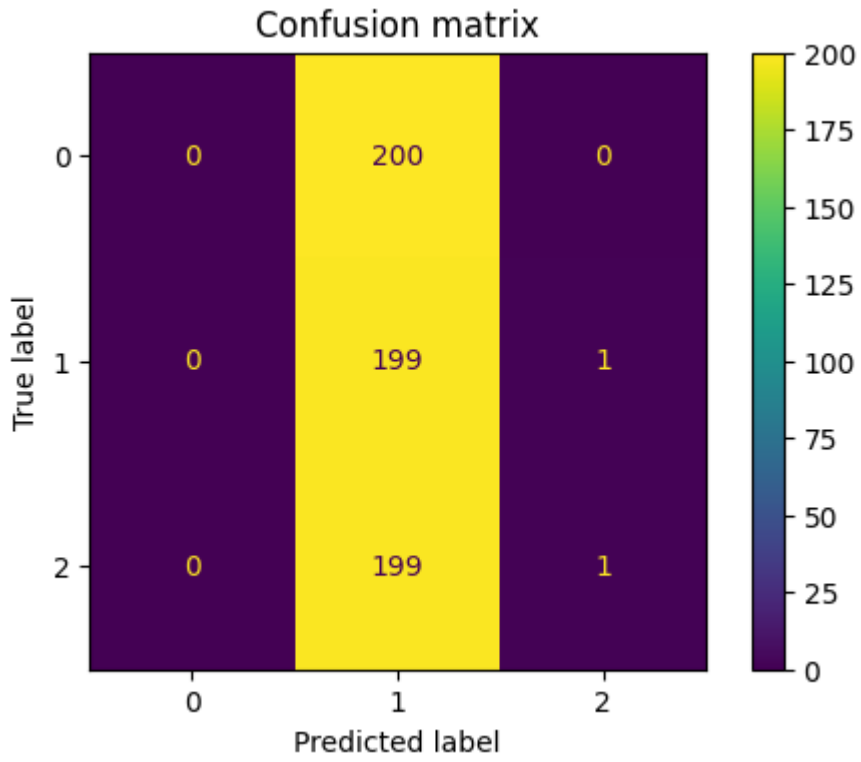
In [ ]:

```
# for you !
from sklearn.metrics import f1_score

score = model.evaluate(xTest,yTest)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

ypred = np.argmax(model.predict(xTest), axis=1)
print("F1 score: ", f1_score(ypred, yTest,average='micro'))
```

```
19/19 [==============================] - 7s 382ms/step - loss: 1.0987
- accuracy: 0.3333
accuracy: 33.33%
19/19 [==============================] - 2s 116ms/step
F1 score:  0.3333333333333333
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix


#test performances
y_pred = np.int64(ypred)


cm = confusion_matrix(yTest, y_pred)

fig, ax = plt.subplots(figsize=(6, 4))  # You can adjust the figsize as

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
# disp.plot()
disp.plot(ax=ax)

disp.ax_.set_title('Confusion matrix')
```

Out[45]: Text(0.5, 1.0, 'Confusion matrix')



In [ ]:

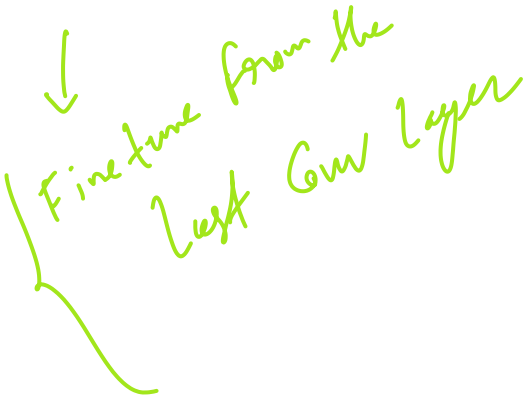
In [ ]:

In [ ]:

Some classes are not predicted because we did not shuffle the data and every samples of some datasets are part of the validation set.

fine-tune the network

Fine-tune the entire network if you have enough computing ressouces, otherwise, carefully choose the layers you want to fine-tune.



In [ ]:

```
for i, layer in enumerate(VGGmodel.layers):
   print(i, layer.name)
model.summary()
```

```
0 input_2
1 block1_conv1
2 block1_conv2
3 block1_pool
4 block2_conv1
5 block2_conv2
6 block2_pool
7 block3_conv1
8 block3_conv2
9 block3_conv3
10 block3_pool
11 block4_conv1
12 block4_conv2
13 block4_conv3
14 block4_pool
15 block5_conv1
16 block5_conv2
17 block5_conv3
18 block5_pool
Model: "model"
```

Last Conv Layer

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, None, None, 3)]	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

```
 global_average_pooling2d_1  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1024)              525312

 dense_5 (Dense)             (None, 3)                 3075

=================================================================
Total params: 15243075 (58.15 MB)
Trainable params: 15243075 (58.15 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In this example, we will fine-tune the last convolution block starting at layer number 15 (block5_conv).

In [ ]:

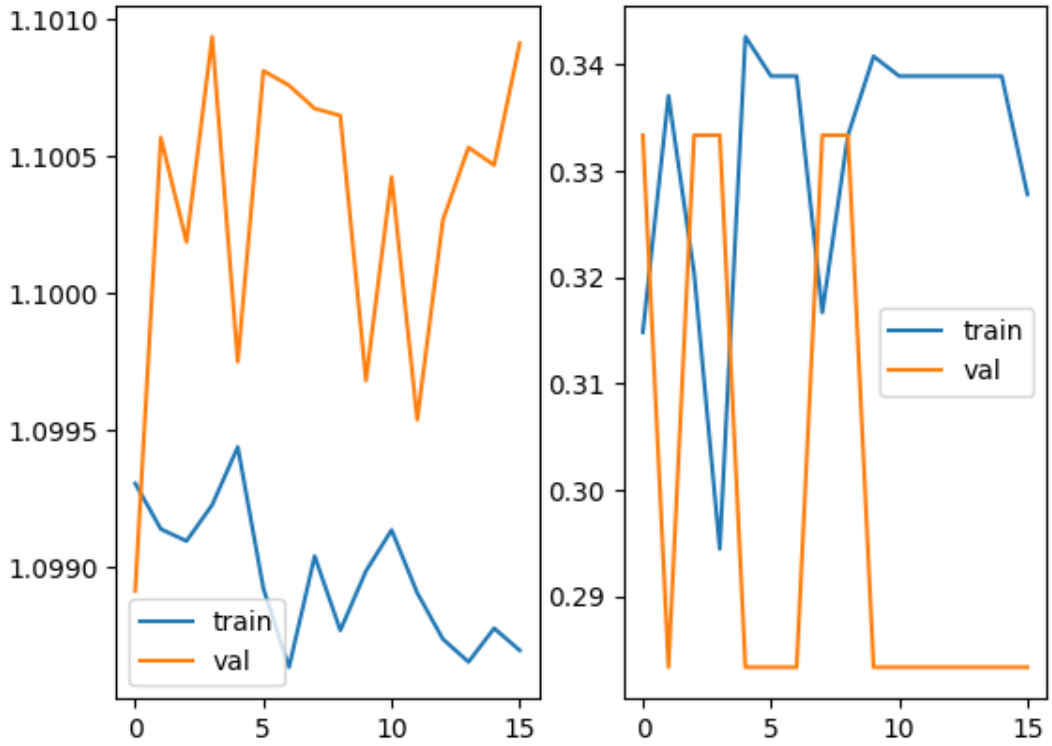
In [ ]:

```
# from tensorflow.keras.optimizers import RMSprop
for layer in model.layers[:14]:
   layer.trainable = False
for layer in model.layers[14:]:
   layer.trainable = True
#need to recompile the network
## Your code here
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
# model.compile(optimizer=RMSprop(learning_rate=0.0001), loss='categori
#and train again ...
# history=model.fit(xTrain, yTrain, epochs=20, batch_size=128, validati
# xTrain, yTrain = shuffle(xTrain, yTrain)
history = model.fit(xTrain, yTrain, epochs=100, validation_split=0.1,sh
                    callbacks=[EarlyStopping(monitor="val_loss", mode="
```

```
from tensorflow.keras.optimizers import RMSprop
for layer in model.layers[:14]:
   layer.trainable = False
for layer in model.layers[14:]:
   layer.trainable = True
#need to recompile the network
model.compile(optimizer=RMSprop(learning_rate=0.0001), loss='categorical_crossentropy',metrics=['accuracy'])
#and train again ...
history=model.fit(xTrain, yTrain, epochs=20, batch_size=128, validation_split=0.2, callbacks=
[early_stopping],verbose=1)
```

```
Epoch 1/100
17/17 [==============================] - 5s 175ms/step - loss: 1.0993
- accuracy: 0.3148 - val_loss: 1.0989 - val_accuracy: 0.3333
Epoch 2/100
17/17 [==============================] - 3s 154ms/step - loss: 1.0991
- accuracy: 0.3370 - val_loss: 1.1006 - val_accuracy: 0.2833
Epoch 3/100
17/17 [==============================] - 3s 162ms/step - loss: 1.0991
- accuracy: 0.3204 - val_loss: 1.1002 - val_accuracy: 0.3333
Epoch 4/100
17/17 [==============================] - 3s 158ms/step - loss: 1.0992
- accuracy: 0.2944 - val_loss: 1.1009 - val_accuracy: 0.3333
Epoch 5/100
17/17 [==============================] - 3s 157ms/step - loss: 1.0994
- accuracy: 0.3426 - val_loss: 1.0997 - val_accuracy: 0.2833
Epoch 6/100
17/17 [==============================] - 3s 158ms/step - loss: 1.0989
- accuracy: 0.3389 - val_loss: 1.1008 - val_accuracy: 0.2833
Epoch 7/100
17/17 [==============================] - 3s 153ms/step - loss: 1.0986
- accuracy: 0.3389 - val_loss: 1.1008 - val_accuracy: 0.2833
Epoch 8/100
17/17 [==============================] - 3s 158ms/step - loss: 1.0990
- accuracy: 0.3167 - val_loss: 1.1007 - val_accuracy: 0.3333
Epoch 9/100
17/17 [==============================] - 3s 165ms/step - loss: 1.0988
- accuracy: 0.3333 - val_loss: 1.1006 - val_accuracy: 0.3333
Epoch 10/100
17/17 [==============================] - 3s 154ms/step - loss: 1.0990
- accuracy: 0.3407 - val_loss: 1.0997 - val_accuracy: 0.2833
Epoch 11/100
17/17 [==============================] - 3s 153ms/step - loss: 1.0991
- accuracy: 0.3389 - val_loss: 1.1004 - val_accuracy: 0.2833
Epoch 12/100
17/17 [==============================] - 3s 154ms/step - loss: 1.0989
- accuracy: 0.3389 - val_loss: 1.0995 - val_accuracy: 0.2833
Epoch 13/100
17/17 [==============================] - 3s 158ms/step - loss: 1.0987
- accuracy: 0.3389 - val_loss: 1.1003 - val_accuracy: 0.2833
Epoch 14/100
17/17 [==============================] - 3s 159ms/step - loss: 1.0987
- accuracy: 0.3389 - val_loss: 1.1005 - val_accuracy: 0.2833
Epoch 15/100
17/17 [==============================] - 3s 158ms/step - loss: 1.0988
- accuracy: 0.3389 - val_loss: 1.1005 - val_accuracy: 0.2833
Epoch 16/100
17/17 [==============================] - 3s 159ms/step - loss: 1.0987
- accuracy: 0.3278 - val_loss: 1.1009 - val_accuracy: 0.2833
```

In [ ]:
```
f, (ax1, ax2) = plt.subplots(1,2)
ax1.plot(history.history['loss'], label='train')
ax1.plot(history.history['val_loss'], label='val')
ax1.legend()
ax2.plot(history.history['accuracy'], label='train')
ax2.plot(history.history['val_accuracy'], label='val')
ax2.legend()
plt.show()
```



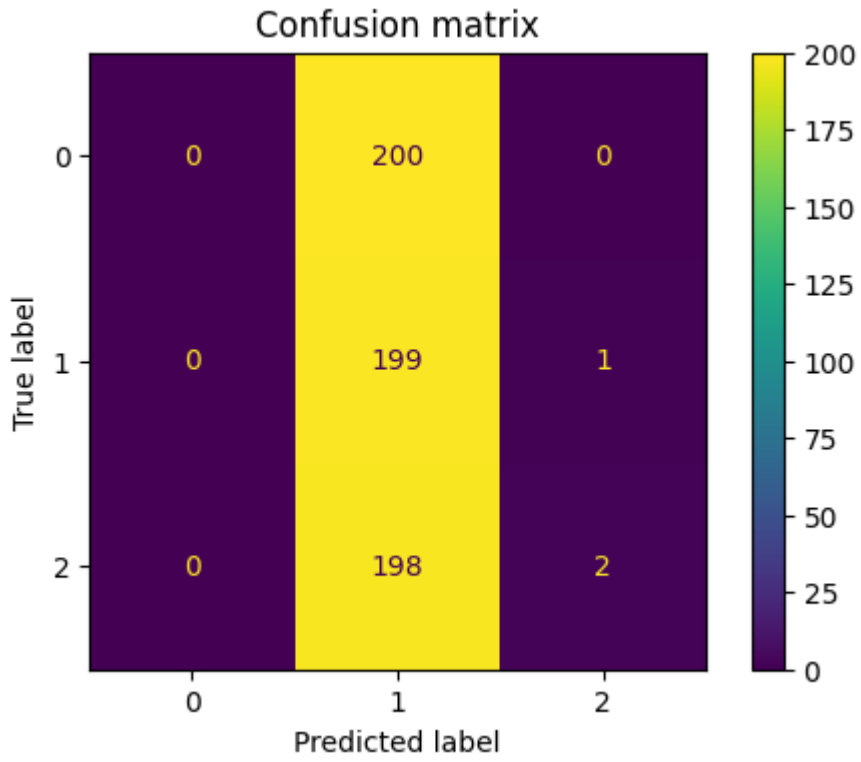
You already know how to evaluate the performances on the test dataset and display the confusion matrix. You can also modify the code that loads the test dataset in order to reduce it's size. Let's do it!

In [ ]:
```
# for you !
from sklearn.metrics import f1_score

score = model.evaluate(xTest,yTest)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

ypred = np.argmax(model.predict(xTest), axis=1)
print("F1 score: ", f1_score(ypred, yTest,average='micro'))
```

```
19/19 [==============================] - 2s 118ms/step - loss: 1.0987
- accuracy: 0.3350
accuracy: 33.50%
19/19 [==============================] - 2s 113ms/step
F1 score:  0.335
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix


#test performances
y_pred = np.int64(ypred)


cm = confusion_matrix(yTest, y_pred)

fig, ax = plt.subplots(figsize=(6, 4))  # You can adjust the figsize as

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
# disp.plot()
disp.plot(ax=ax)

disp.ax_.set_title('Confusion matrix')
```

Out[50]: Text(0.5, 1.0, 'Confusion matrix')



You are now free to experiments changes in the network:

- add a dense layer
- modify the number of neurons in dense layer(s)
- change the global average polling
- add classes and data
- experiment other optimizers (SGD, Adam, ...)

...

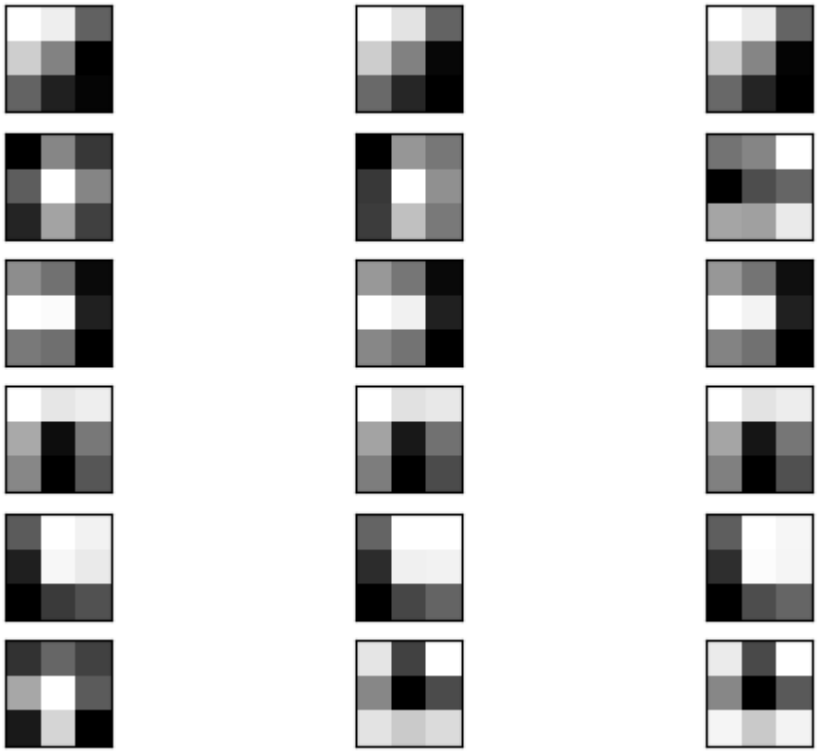
Visualizing the convolution filters

In this part, we'll visualize the convolution filters and their effect on the input for our previously trained model

[TO DO - Students] What is the following code plotting ?

In [ ]:

```
import matplotlib.pyplot as pyplot
layer_id = 1
# retrieve weights from the second hidden layer
filters, biases = model.layers[layer_id].get_weights()
# normalize filter values to 0-1 so we can visualize them
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)
# plot first few filters
n_filters, ix = 6, 1
for i in range(n_filters):
    # get the filter
    f = filters[:, :, :, i]
    # plot each channel separately
    for j in range(3):
        # specify subplot and turn of axis
        ax = pyplot.subplot(n_filters, 3, ix)
        ax.set_xticks([])
        ax.set_yticks([])
        # plot filter channel in grayscale
        pyplot.imshow(f[:, :, j], cmap='gray')
        ix += 1
# show the figure
pyplot.show()
```



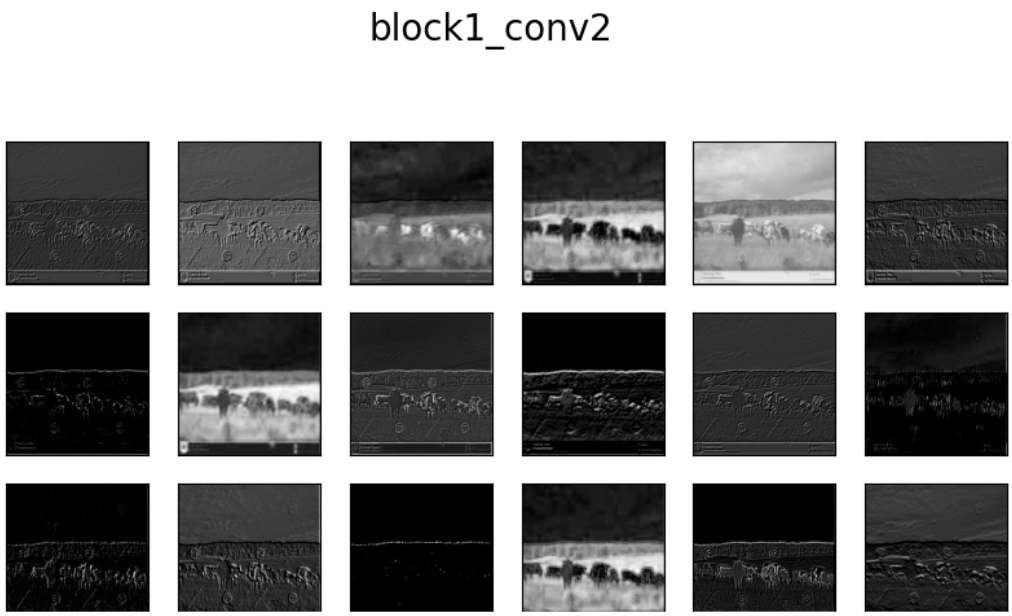
[TO DO - Students] Now, let's visualize the feature maps of various depths. Fill the

Type *Markdown* and LaTeX: α^2

In [ ]:

```
# redefine model to output right after the first hidden layer
# img = xTrain[...].reshape(...)
img = xTrain[12].reshape((1,) + xTrain[10].shape)
for layer in ['block1_conv2', 'block2_conv2', 'block3_conv1', 'block4_c
    # model_fm = Model(inputs=model.inputs, outputs=...)
    model_fm = Model(inputs=model.inputs, outputs=model.get_layer(layer

    feature_maps = model_fm.predict(img)
    # plot all 64 maps in an 8x8 squares
    square = 6
    ix = 1
    fig, axs = plt.subplots(square, square, figsize=(10, 10))
    for i in range(square):
      for j in range(square):
        # specify subplot and turn of axis
        ax = axs[i,j]
        ax.set_xticks([])
        ax.set_yticks([])
        # plot filter channel in grayscale
        # ax.imshow(feature_maps[0, :, :, ...], cmap='gray')
        ax.imshow(feature_maps[0, :, :, ix-1], cmap='gray')
        ix += 1
    # show the figure
    fig.suptitle(layer, fontsize=20)
    plt.show()
```

```
1/1 [==============================] - 0s 274ms/step
```



Activation maximization

Another solution to interpret the inner mecanisms of the network is to use Activation Maximization. This method computes the optimal output which gives the maximum value of a particular activation. Used on the classification layers, this can give us an idea of the patterns recognized to classify a particular class.

To do that we'll use the tf_keras_vis module.

In [ ]:
```
[i for i in range(10)]
```

Out[53]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

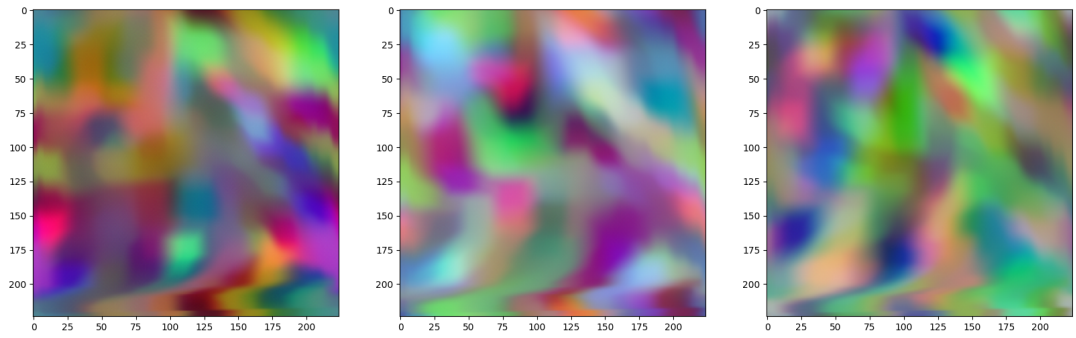
In [ ]:
```
# ! pip install tf_keras_vis
from tf_keras_vis.activation_maximization import ActivationMaximization
```

In [ ]:
```
def loss(output):
  return (output[0][0], output[1][1], output[2][2])

def model_modifier(m):
    m.layers[-1].activation = tensorflow.keras.activations.linear

visualize_activation = ActivationMaximization(model, model_modifier)
```

In [ ]:
```
seed_input = tensorflow.random.uniform((3, 224, 224, 3), 0, 255)
activations = visualize_activation(loss, seed_input=seed_input, steps=5
# images = [activation.astype(np.float32) for activation in activations
```

In [ ]:
```
tf.experimental.numpy.experimental_enable_numpy_behavior()
images = [activation.astype(np.float32) for activation in activations]
```

In [ ]:
```
fig, axs = plt.subplots(1, 3, figsize=(20, 20))
for i in range(0, len(images)):
  ax = axs[i]
  visualization = images[i].reshape(224,224,3)
  visualization = (visualization - visualization.min())/(visualization.
  visualization = visualization[:,:,[2,1,0]]
  ax.imshow(visualization)
```



In [ ]:
```

```