IMPORTING DATASETS FROM KAGGLE

```
from google.colab import files
```

```
!pip install -q kaggle
```

```
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell t

```
Saving kaggle.json to kaggle.json
```

```
!mkdir -p ~/.kaggle
```

```
!cp kaggle.json ~/.kaggle/
```

```
!ls ~/.kaggle
```

```
kaggle.json
```

```
!chmod 600 /root/.kaggle/kaggle.json
```

```
!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
```

```
Downloading chest-xray-pneumonia.zip to /content
 99% 2.27G/2.29G [00:44<00:00, 36.2MB/s]
100% 2.29G/2.29G [00:44<00:00, 55.8MB/s]
```

```
!kaggle datasets download -d bachrr/covid-chest-xray
```

```
Downloading covid-chest-xray.zip to /content
 97% 233M/241M [00:05<00:00, 44.2MB/s]
100% 241M/241M [00:05<00:00, 47.4MB/s]
```

```
!unzip chest-xray-pneumonia.zip
```

```
!unzip covid-chest-xray.zip
```

BUILDING DATASET

```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random
import shutil
import cv2
import os
```

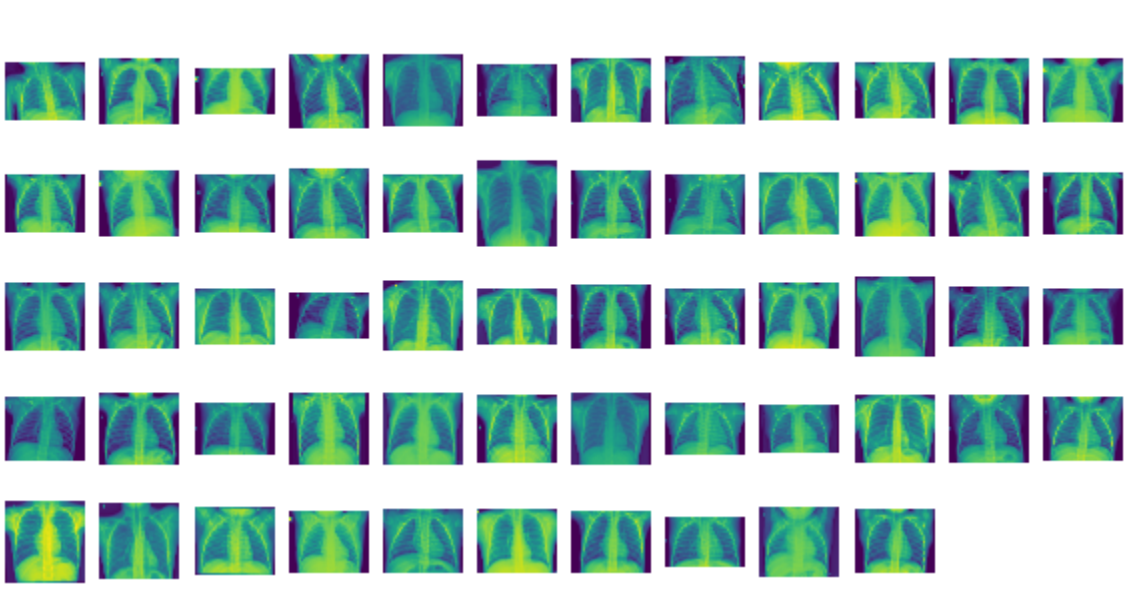
```
dataset_path = './dataset'
```

```
!rm -rf dataset
!mkdir -p dataset/covid
!mkdir -p dataset/normal
```

```
samples = 25
```

```
covid_dataset_path = '/content'


# construct the path to the metadata CSV file and load it
csvPath = os.path.sep.join([covid_dataset_path, "metadata.csv"])
df = pd.read_csv(csvPath)
i=1

# loop over the rows of the COVID-19 data frame
for (i, row) in df.iterrows():
    # if (1) the current case is not COVID-19 or (2) this is not
    # a 'PA' view, then ignore the row
    if row["finding"] != "COVID-19" or row["view"] != "PA":
        continue

    # build the path to the input image file
    imagePath = os.path.sep.join([covid_dataset_path, "images", row["filename"]])

    # if the input image file does not exist (there are some errors in
    # the COVID-19 metadeta file), ignore the row
    if not os.path.exists(imagePath):
        continue

    # extract the filename from the image path and then construct the
    # path to the copied image file
    filename = row["filename"].split(os.path.sep)[-1]
    outputPath = os.path.sep.join([f"{dataset_path}/covid", filename])

    # copy the image
    shutil.copy2(imagePath, outputPath)


pneumonia_dataset_path ='/content/chest_xray/chest_xray'


basePath = os.path.sep.join([pneumonia_dataset_path, "train", "NORMAL"])
imagePaths = list(paths.list_images(basePath))
j=1
# randomly sample the image paths
random.seed(42)
random.shuffle(imagePaths)
imagePaths = imagePaths[:samples]

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
    # extract the filename from the image path and then construct the
    # path to the copied image file
    filename = imagePath.split(os.path.sep)[-1]
    outputPath = os.path.sep.join([f"{dataset_path}/normal", filename])

    # copy the image
    shutil.copy2(imagePath, outputPath)


basePath = os.path.sep.join([pneumonia_dataset_path, "test", "NORMAL"])
imagePaths = list(paths.list_images(basePath))
j=1
# randomly sample the image paths
random.seed(42)
random.shuffle(imagePaths)
imagePaths = imagePaths[:samples]

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
    # extract the filename from the image path and then construct the
    # path to the copied image file
    filename = imagePath.split(os.path.sep)[-1]
    outputPath = os.path.sep.join([f"{dataset_path}/normal", filename])

    # copy the image
    shutil.copy2(imagePath, outputPath)


basePath = os.path.sep.join([pneumonia_dataset_path, "val", "NORMAL"])
imagePaths = list(paths.list_images(basePath))
j=1
# randomly sample the image paths
random.seed(42)
random.shuffle(imagePaths)
imagePaths = imagePaths[:samples]

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
    # extract the filename from the image path and then construct the
    # path to the copied image file
    filename = imagePath.split(os.path.sep)[-1]
```

```
    filename = imagePath.split(os.path.sep)[-1]
    outputPath = os.path.sep.join([f"{dataset_path}/normal", filename])
    j=j+1
    # copy the image
    shutil.copy2(imagePath, outputPath)


def ceildiv(a, b):
    return -(-a // b)

def plots_from_files(imspaths, figsize=(10,5), rows=1, titles=None, maintitle=None):
    """Plot the images in a grid"""
    f = plt.figure(figsize=figsize)
    if maintitle is not None: plt.suptitle(maintitle, fontsize=10)
    for i in range(len(imspaths)):
        sp = f.add_subplot(rows, ceildiv(len(imspaths), rows), i+1)
        sp.axis('Off')
        if titles is not None: sp.set_title(titles[i], fontsize=16)
        img = plt.imread(imspaths[i])
        plt.imshow(img)


normal_images = list(paths.list_images(f"{dataset_path}/normal"))
covid_images = list(paths.list_images(f"{dataset_path}/covid"))


plots_from_files(normal_images, rows=5, maintitle="Normal X-ray images")
```




```
plots_from_files(covid_images, rows=5, maintitle="Covid-19 X-ray images")
```

DATA PREPROCESSING

```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-3
EPOCHS = 5
BS = 24


# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(dataset_path))
data = []
labels = []
# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]
```

```
    label = imagePath.split(os.path.sep)[-2]
    # load the image, swap color channels, and resize it to be a fixed
    # 224x224 pixels while ignoring aspect ratio
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224, 224))
    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)
# convert the data and labels to NumPy arrays while scaling the pixel
# intensities to the range [0, 1]
data = np.array(data) / 255.0
labels = np.array(labels)
```

```
[INFO] loading images...
```

```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.20, stratify=labels, random_state=42)
# initialize the training data augmentation object
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")
```

MODEL

```
# load the VGG16 network, ensuring the head FC layer sets are left
# off
baseModel = VGG16(weights="imagenet", include_top=False, input_tensor=Input(shape=(224, 224, 3)))
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(2, 2))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.15)(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

TRAINING

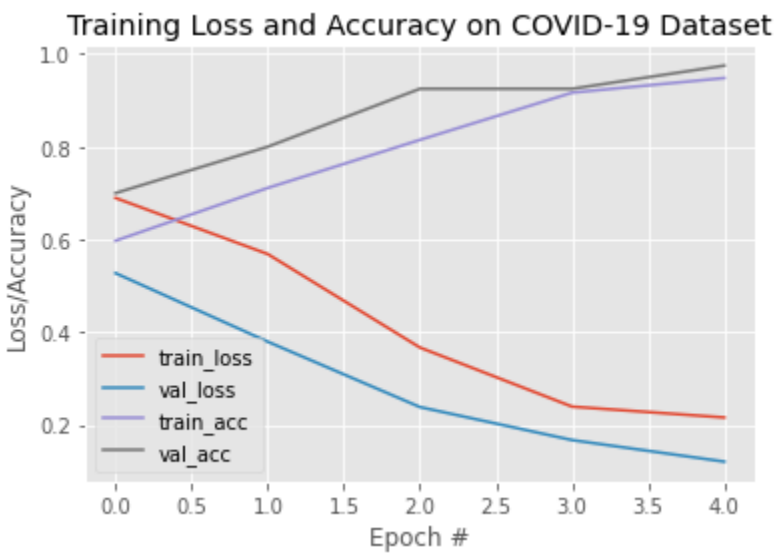
```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.fit_generator(
    trainAug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

```
[INFO] compiling model...
[INFO] training head...
Epoch 1/5
6/6 [==============================] - 2s 274ms/step - loss: 0.6895 - accuracy: 0.5972 - val_loss: 0.5273 - val_accuracy: 0.7000
Epoch 2/5
6/6 [==============================] - 1s 224ms/step - loss: 0.5685 - accuracy: 0.7111 - val_loss: 0.3794 - val_accuracy: 0.8000
Epoch 3/5
6/6 [==============================] - 1s 219ms/step - loss: 0.3668 - accuracy: 0.8148 - val_loss: 0.2384 - val_accuracy: 0.9250
Epoch 4/5
6/6 [==============================] - 1s 232ms/step - loss: 0.2393 - accuracy: 0.9167 - val_loss: 0.1672 - val_accuracy: 0.9250
Epoch 5/5
6/6 [==============================] - 1s 220ms/step - loss: 0.2159 - accuracy: 0.9481 - val_loss: 0.1207 - val_accuracy: 0.9750
```

```
# plot the training loss and accuracy
```

```
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
#plt.savefig("plot.png")
```

```
<matplotlib.legend.Legend at 0x7fa8447d0dd8>
```




```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs, target_names=lb.classes_))
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

       covid       0.97      1.00      0.98        28
      normal       1.00      0.92      0.96        12

    accuracy                           0.97        40
   macro avg       0.98      0.96      0.97        40
weighted avg       0.98      0.97      0.97        40
```

```
# compute the confusion matrix and and use it to derive the raw
# accuracy, sensitivity, and specificity
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[28  0]
 [ 1 11]]
acc: 0.9750
sensitivity: 1.0000
specificity: 0.9167
```

PREDICTION

1: NEGATIVE

0: POSITIVE

```
# serialize the model to disk
print("[INFO] saving COVID-19 detector model...")
model.save("covid19.model", save_format="h5")
```

```
[INFO] saving COVID-19 detector model...
```

```
from tensorflow import keras
```

```
model = keras.models.load_model('/content/covid19.model')


model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])


validate = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell t

```
Saving covid.jpg to covid.jpg


IMG_SIZ=224
img = cv2.imread('/content/covid.jpg')
img = cv2.resize(img,(IMG_SIZ,IMG_SIZ),3)
img = np.reshape(img,[1,224,224,3])

classes = model.predict(img)
prediction=np.argmax(classes,axis=1)

print(prediction)
```

```
[0]


validate = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell t

```
Saving covidpos.jpg to covidpos.jpg


IMG_SIZ=224
img = cv2.imread('/content/covidpos.jpg')
img = cv2.resize(img,(IMG_SIZ,IMG_SIZ),3)
img = np.reshape(img,[1,224,224,3])

classes = model.predict(img)
prediction=np.argmax(classes,axis=1)

print(prediction)
```

```
[0]


validate = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell t

```
Saving negative.jpeg to negative.jpeg


IMG_SIZ=224
img = cv2.imread('/content/negative.jpeg')
img = cv2.resize(img,(IMG_SIZ,IMG_SIZ),3)
img = np.reshape(img,[1,224,224,3])

classes = model.predict(img)
prediction=np.argmax(classes,axis=1)

print(prediction)
```

```
[1]


validate = files.upload()
```


Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell t

```
Saving normal.jpeg to normal.jpeg


IMG_SIZ=224
img = cv2.imread('/content/normal.jpeg')
img = cv2.resize(img,(IMG_SIZ,IMG_SIZ),3)
img = np.reshape(img,[1,224,224,3])

classes = model.predict(img)
prediction=np.argmax(classes,axis=1)

print(prediction)
```



```
[1]
```