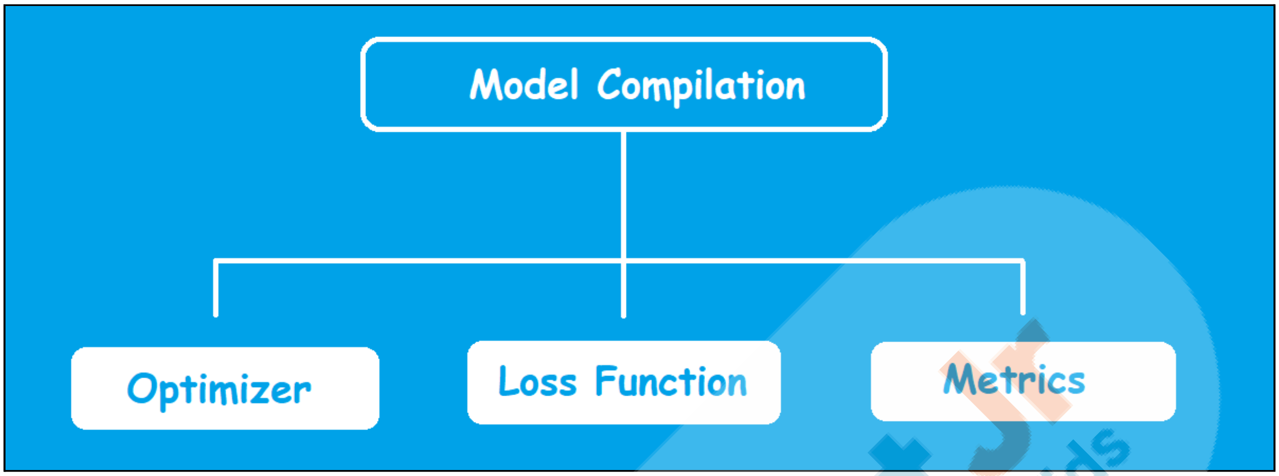
Topic	COMPILE, FIT & PREDICT	
Class Description	**The student will learn to compile and fit the Convolutional Neural Network(CNN) model for training. The student will learn to test the model which can predict the class of the image.**	
Class	**PRO C113**	
Class time	**45 mins**	
Goal	• Learn to compile and fit the Convolutional Neural Network(CNN) model for training. • Learn to test the CNN model for image classification predictions.	
Resources Required	• Teacher Resources: ○ Laptop with internet connectivity ○ Webcam ○ Earphones with mic ○ Notebook and pen ○ Smartphone • Student Resources: ○ Laptop with internet connectivity ○ Webcam ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-led Activity 1** **Student-led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit	Teachable Machines by Google. Tensorflow by Google Brain team. Keras by Google Engineer Francois Chollet.	
WARM-UP SESSION - 10 mins		

Teacher Starts Slideshow Slide 1 to 3 Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** ● Greet the student. ● Revision of previous class activities. ● Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide 4 to 13**	
Following are the session deliverables: ● Appreciate the student. ● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Teacher Action	**Student Action**
In today's we are going to put the whole model together and test it to make a prediction of the class, in which the unseen images belong, either "infected" or "uninfected" from Pneumothorax disease. Are you excited?	**ESR:** Yes.

Let's get started.	

Teacher Ends Slideshow
TEACHER-LED ACTIVITY - 10 mins
Teacher Initiates Screen Share
ACTIVITY • **Compile and Fit the CNN model.**

Teacher Action	Student Action
Open Teacher Activity 1 and run all the cells. **Model Compilation:** After defining our model with all the layers one after the other, we have to set up our model. We do this set up in the model compilation phase, which sets up the model for training. Now, before training the model, we need to compile it. We compile the model using the **compile()** method(Keras). The compile method takes many arguments, but we will pass the three arguments which must be specified. The arguments are: • Optimizers • Loss function • Metrics for prediction	



Need of Model Compilation:

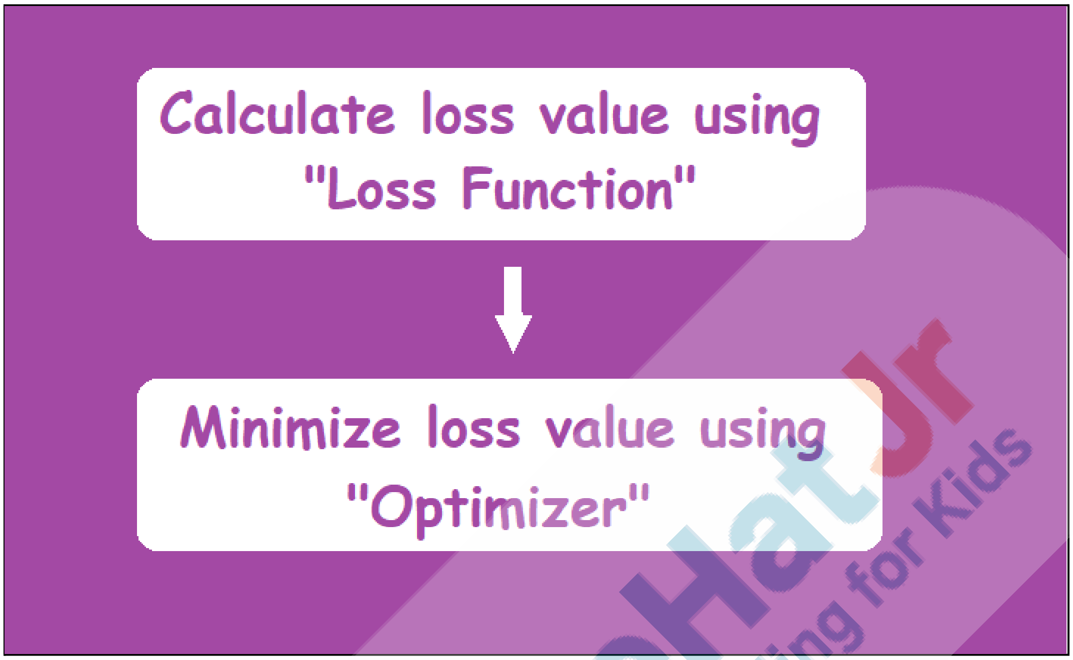
When the model is trained it can, almost never, be 100% efficient, that it cannot always predict the class of the image correctly.

This leads to the concept of loss during model training, which tells us how bad the model is performing.

Hence, we need to use the loss functions(these are mathematical computation functions) to get the value of the loss.

For example, if the result of the loss function gives us the value of 0.45, this means that 45 %(0.45X100) of the times, the model will predict wrong results, and only 55% times will predict the correct results.

That means, we should try to **minimize the loss function value**, because a lower loss value means our model is going to perform better. The process of minimizing (or maximizing) the value of a mathematical function/expression is called **optimization**.



Metrics are functions similar to loss functions which also tells how well the model is performing. The result of the metric function is needed at the time of **evaluation** of the model.

***Note**: We will not be covering the mathematical operations of the loss functions, optimizers and metrics.*

Now let's write the program for model compilation using the **compile()** method and specify the loss function, optimizer and metrics.

A few of the available **loss functions** are:

- binary_crossentropy
- categorical_crossentropy
- mean_squared_error
- mean_absolute _error

A few of the available **optimizers** are: ● Adam ● RMSprop ● SGD(Stochastic Gradient Descent) A few of the available metrics: ● accuracy ● binary_accuracy ● categorical_accuracy ***Note**: We will not be covering the mathematical operations of the loss functions, optimizers and metrics.* *The teacher sets the parameters of the **compile()** method with "**loss**" as string value, "**optimizer**" as string value and "**metrics**" as a list.*	

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model Training: We will use the **fit()** method to start the training. 1. Take a variable called **history**, and use **fit()** with parameter: a. **training_augmented_images**, b. **epochs**=20, this tells the number of times each CNN layer will be repeated for training. c. **validation_data** = validation_augmented_images.	

d. **verbose**=True, this helps to log the details while the training is going on. 2. Save the model as **.h5** file using **save()** method: a. model.save("Pneumothorax.h5")	

```
history = model.fit(training_augmented_images, epochs=20, validation_data =
validation_augmented_images, verbose=True)

model.save("Pneumothorax.h5")
```

The model training will start. Since we set the **verbose** to **True**, it will show the details of each **epoch**: **loss**: This represents the probability of a wrong prediction. **accuracy**: This represents the probability of a right prediction. **val_loss**: This represents validation of the probability of a wrong prediction. **val_accuracy**: This represents the probability of a right prediction. ***Note**: Training models take 5-10 mins depending on the system's configuration. You will have to wait until the model is trained(in this case until epoch 20 is complete) After this you can ask the student to do the Model compilation using **compile()** method and model training using the **fit()** method.* *If the training is taking way too much time, stop the cell execution and try with a smaller number of epochs.*	
Output(During ongoing training process)	

```
history = model.fit(training_augmented_images, batch_size=32, epochs=25, validation_data = validation_augmented_images, verbose=True)

model.save("Pneumothorax.h5")

Epoch 1/25
7/7 [==============================] - 31s 4s/step - loss: 0.7091 - accuracy: 0.5200 - val_loss: 0.6951 - val_accuracy: 0.5000
Epoch 2/25
7/7 [==============================] - 29s 4s/step - loss: 0.6964 - accuracy: 0.4700 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 3/25
7/7 [==============================] - 29s 4s/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 4/25
7/7 [==============================] - 29s 5s/step - loss: 0.6932 - accuracy: 0.4800 - val_loss: 0.6935 - val_accuracy: 0.5000
```

Meanwhile, the model is trained, let's understand the **accuracy metrics** parameter:

- Accuracy metric is a ratio of correctly predicted observations to the total observations. Since we are using images as our observations for training and testing, accuracy metric is:

  accuracy = Correctly Predicted Images / Total Images

- Accuracy is a model performance measure/metric that means this metric will tell how well the model is able to predict the class/category of the unseen image. The more images predicted with the correct class, the more accurate the model.

We can see that the accuracy of the model is coming from 0.52(this number may vary for teachers and students) that is 52%(0.52x100).

This means this model can detect the images correctly only around 50% of the time, which is not that bad or good.

But this should be improved. As these X-Rays images disease detection will need to be more accurate.

***Note**: There are a lot of ways to improve accuracy of the model:* • *One of those is Data Augmentation, which we used while preprocessing the image dataset.* • *We can also use larger dataset. We are currently using only 100 images only, we can use around 2000-3000 images to train the machine to increase the accuracy. Note that training on the larger dataset takes a significant amount of the time and computer processing efficiency.* *There are other methods to improve accuracy of the model that will not be covered as a part of this lesson.*	

***Note**: Training models take 5-10 mins depending on the system's configuration. You will have to wait until the model is trained(in this case until epoch 20 is complete). While your model is training you can start with student activities starting with Model compilation using the **compile()** method and model training using the **fit()** method.*

Output(At the end of training process, here after 20 epochs)

```
Epoch 18/20
7/7 [==============================] - 27s 4s/step - loss: 0.6932 - accuracy: 0.5150 - val_loss: 0.6932 - val_accuracy: 0.5200
Epoch 19/20
7/7 [==============================] - 27s 4s/step - loss: 0.6928 - accuracy: 0.5000 - val_loss: 0.6932 - val_accuracy: 0.5050
Epoch 20/20
7/7 [==============================] - 27s 4s/step - loss: 0.6936 - accuracy: 0.5200 - val_loss: 0.6950 - val_accuracy: 0.5000
```

We learned to compile and fit the model, you will also compile and fit the model, after the model we will test the model class prediction on a few images. Are you excited?	**ESR:** Yes.

Teacher Stops Screen Share

Teacher Starts Slideshow
Slide 14 to 15

Refer to speaker notes and follow the instructions on each slide.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
Teacher Ends Slideshow	
STUDENT-LED ACTIVITY - 20 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Fullscreen.**	
ACTIVITY • **Compile and Fit the CNN model.** • **Make predictions using CNN model.**	
Teacher Action	**Student Action**
Guide the student to open the colab notebook and make a copy of the boilerplate code using Student Activity 1. Run all the cells to load the dataset and update the code cells output. Now we will use the **compile()** method for the model compilation and the **fit()** method for the model training. *Guide the student to set parameters of the:* • *The **compile()** method with "**loss**" as string value, "**optimizer**" as string value and "**metrics**" as a list.* • *The **fit()** method training_augmented_images, epochs, validation_data, verbose.*	

Compile Model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fit & Save Model

```
history = model.fit(training_augmented_images, epochs=20, validation_data = validation_augmented_images, verbose=True)

model.save("Pneumothorax.h5")
```

```
Epoch 1/20
7/7 [==============================] - 30s 4s/step - loss: 0.6927 - accuracy: 0.4550 - val_loss: 0.6933 - val_accuracy: 0.4950
Epoch 2/20
```

Proceed to upcoming steps only once all epochs(in this, 20 epochs) of the model training were completed.

We learned to compile and fit the model, now let's test the model.
We will need to import libraries/module:

This is a part of the boilerplate. Explain the use of these libraries for this activity.

os: to interact with system
numpy: for array manipulation
load_img(from keras **tensorflow**): to load images
img_to_array(from keras tensorflow): to convert images to as arrays.

```
import os
import numpy as np

import tensorflow
from tensorflow.keras.preprocessing.image import load_img

from tensorflow.keras.preprocessing.image import img_to_array
```

Let's take images from the "**testing_dataset/infected**" directory.

Even though we know these are infected images, these are unseen images for the model as we did not use them while training the model.

Now we will see if the model is also able to predict these as infected images.

1. Take a variable called **testing_image_directory**, assign the path of the "**infected**" subdirectory from the **testing_dataset** directory.

2. Take images files names as a list in a variable called, **img_files**, using **listdir()** method from **os** module.

```
# Testing image directory
testing_image_directory = '/content/PRO-M3-Pneumothorax-Image-Dataset/testing_dataset/infected'

# All image files in the directory
img_files = os.listdir(testing_image_directory)
```

3. Loop through any 9 image files(sub-steps a to m):

Note 1: *We will be using one variation(from multiple variations which we will not be discussing) of* ***subplot()*** *method later which can plot only 9 subplots at a time, hence 9 images only.*

Note 2: *Since we have 100 images in the* ***testing_dataset/infected****" directory, we can loop through any 9 images by setting* ***img_files[start_num : end_num]*** *such that start_number and end_number is between* ***0*** *and* ***100***.

`` # Loop through an 9 image files for file in img_files[51:60]: ```	
a. Take a variable called, **img_files_path,** to make a complete path of the image file using **join()** method from **os.path** module. b. Take a variable called **img_1**, and load the images from the file path using **load_img(img_files_path, target_size)** method from keras library. We need to set the size of the image as **target_size =(180, 180)** which is the size of the image on which the model is trained. Remember, it is must to keep the image size as the size of the image used during the training process of the model. c. Take a variable called **img_2**, and convert the image to an array to **img_to_array()** method from keras library. d. Take a variable called **img_3**, and expand the dimension using the **expand_dims()** method along the axis **0**. ***Note**: Help the student recollect the use of this method.*	

```
# Loop through an 9 image files
for file in img_files[91:100]:

    # full path of the image
    img_files_path = os.path.join(testing_image_directory, file)

    # load image
    img_1 = load_img(img_files_path,target_size=(180, 180))

    # convert image to an array
    img_2 = img_to_array(img_1)

    # increase the dimension
    img_3 = np.expand_dims(img_2, axis=0)
```

e. Take a variable called **prediction** and use **predict()** method to predict the probability of each class.

f. Print the value of the **prediction** variable.

```
# Loop through an 9 image files
for file in img_files[51:60]:

  # full path of the image
  img_files_path = os.path.join(testing_image_directory, file)

  # load image
  img_1 = load_img(img_files_path,target_size=(180, 180))

  # convert image to an array
  img_2 = img_to_array(img_1)

  # increase the dimension
  img_3 = np.expand_dims(img_2, axis=0)

  # predict the class of an unseen image
  prediction = model.predict(img_3)
  print(prediction)
```

Output: All the values in the below image are in this format: [[Probability infected, Probability uninfected]]

***Note**: The output value may not be exactly the same. It may vary based on the prediction of the model.*

```
[[0.00868821 0.9997891 ]]
[[0.8429589  0.17911252]]
[[0.8068269 0.567085  ]]
[[0.02241591 0.9857274 ]]
[[0.835863  0.2073184]]
[[0.97911847 0.03325471]]
[[0.9658046  0.06376567]]
[[0.70426434 0.34265167]]
[[0.8958045  0.14640713]]
```

Analyzing Output:

Let's look at the 2nd prediction value in the output.

***Reminder**: The output value may not be exactly the same. It may vary based on the prediction of the model. Choose any one output value to discuss the explanation below the explanation.*

[[0.8429589 , 0.17911252]]

This is a 2D array with outer array and inner array.

The inner array has 2 values:

- The first value represents how much(probability of) the given X-ray image belongs to 1st class(infected X-Ray images)
- The 2nd value represents how much(probability of) the given X-ray image belongs to the 2nd class(uninfected X-Ray Images).

Similarly, we had 10 classes, we would get the prediction output as probability value of belongingness to each class: [[value of class1, value of class2,……, value of class10]] ***Note**: Help the student recollect how the prediction output represents probabilities of belongingness in each class, like we did in image classification with face with and without mask.* Can you tell me which one out of the two values in the inner array,[[0.8429589, 0.17911252]], is maximum?	**ESR**: The second value, 0.8429589.
Yes. Correct! This means this maximum value in prediction value represents that X-Ray image belongs more to 1st class(infected), right?	**ESR**: Yes.
Great! We have successfully predicted the probability of each class for each image. Before we move on, do you remember the label given to the “**infected**” class for **Training** and **Validation** images by the **ImageDataGenerator** class ? `{'infected': 0, 'uninfected': 1}`	**ESR:** Yes. It is 0.
Now let’s understand how we can get labels of the class from these prediction output arrays for **Testing** images. So, can you tell me what is the index of this maximum value in the inner array, [[0.8429589, 0.17911252]], of the 2nd output prediction value?	**ESR:** It is 0.

Similarly, if we take the index of the maximum value of each prediction output, we will get either 0 or 1, because the image will either belong to the 1st class(represented by the 1st value) or the 2nd class(represented by the 2nd value), right?

ESR: Yes.

<u>We can use this idea of **taking an index of the maximum values in the prediction** output to get the labels of the class.</u>

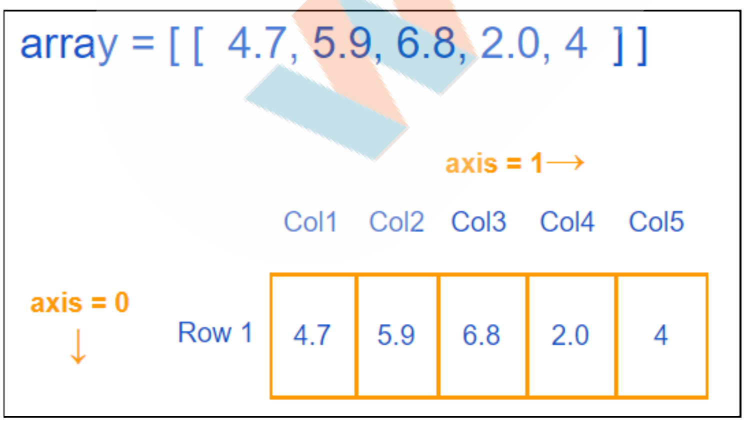
We can use the **argmax(array, axis)** method of the NumPy library to find the index of the maximum value in any array, for this we need to pass an array(of any dimension 1D, 2D, 3D and so on) and the axis(which represents the direction of the dimension).

Let’s take an example of a 2D array(which is similar to our prediction output array) to understand this:

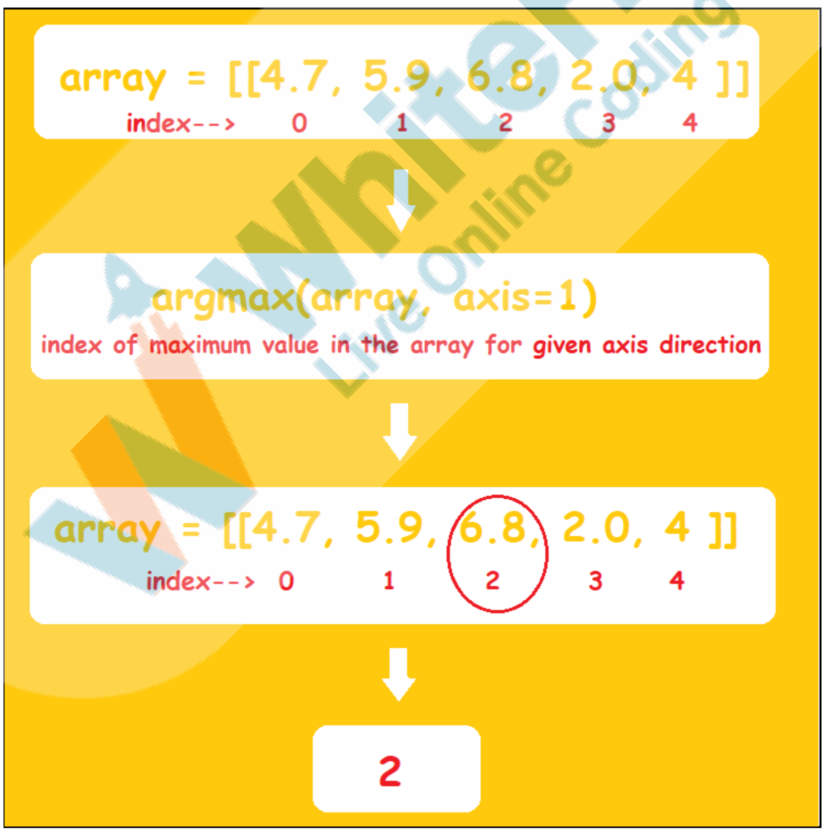
```
array = [ [  4.7, 5.9, 6.8, 2.0, 4  ] ]
```

Open [Teacher Activity 2](#) to show this image to the student while explaining.

This is a 2D array with 1 row and 5 columns.



Can you tell me what the maximum value, along the axis 1(since we want to find the maximum value of all the columns of the inner array) is? Yes. Correct! And what is the index of the maximum value(that is 6.8) along the axis 1? *Open [Teacher Activity 3] to show this image to the student while explaining.* When we pass this **array** and **axis** along which we want the maximum value, to the **argmax()** method, we will get 2 as shown below.	**ESR**: It is 6.8. **ESR**: It is 2.



g. Take a variable called **predict_class**, either 0 or 1 in this case, and use **argmax()** to get the index value of the maximum prediction value in the output by passing the prediction output array and axis 1.

h. Print the **predict_class** variable.

```
# Loop through an 9 image files
for file in img_files[51:60]:

  # full path of the image
  img_files_path = os.path.join(testing_image_directory, file)

  # load image
  img_1 = load_img(img_files_path,target_size=(180, 180))

  # convert image to an array
  img_2 = img_to_array(img_1)

  # increase the dimension
  img_3 = np.expand_dims(img_2, axis=0)

  # predict the class of an unseen image
  prediction = model.predict(img_3)
  print(prediction)

  predict_class = np.argmax(prediction, axis=1)
  print(predict_class)
```

Output : Since we have all the infected images in this folder, the index of prediction values for these images should be 0, that means the maximum value must be the 1st value in the output, but since the model is only accurate around 50% of the time, we might get 2nd maximum value as the maximum, hence the index 1.

```
[[0.00868821 0.9997891 ]]
[1]
[[0.8429589  0.17911252]]
[0]
[[0.8068269 0.567085 ]]
[0]
[[0.02241591 0.9857274 ]]
[1]
[[0.835863   0.2073184]]
[0]
[[0.97911847 0.03325471]]
[0]
[[0.9658046  0.06376567]]
[0]
[[0.70426434 0.34265167]]
[0]
[[0.8958045  0.14640713]]
[0]
```

i. Comment the **print()** statements before plotting the images using **matplotlib.pyplot**.

```
prediction = model.predict(img_3)
# print(prediction)

predict_class = np.argmax(prediction, axis=1)
# print(predict_class)
```

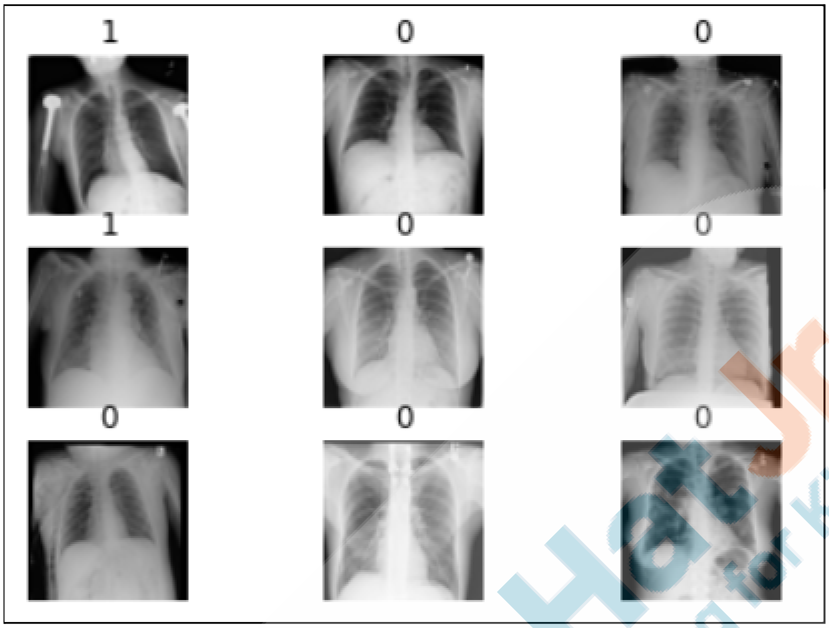
j. Plot subplot using **subplot(rows, cols, indexpos_of_the_subplot)** method from the **matplotlib.pyplot** library.

***Note**: The subplot(rows, col, index) method of this syntax can plot only 9 images. To plot more images together, we need to use different variations of this method, which we will not cover in this class.*

i. Take a variable **i**, outside the loop, to assign the **indexpos_of_the_subplot** value. ii. Increase the index position of the plot using **i = i +1**. k. Show the image using **imshow()** method from the **matplotlib.pyplot** library. Note that we need to send the 3D image array for the **imshow** method. Hence, we will use **img_2** which is before expanding the dimension to a 4D array in **img_3**. l. Add title to the plot using the **predict_class[0].** We can add a title to any plot using the **title()** method from the **matplotlib.pyplot** library. The value inside the **title()** method is of string or number type. m. Switch off the axis(x and y in the plot figures) display using the **axis('off')** method from the **matplotlib.pyplot** library. 4. Show the complete plot(with all the subplots figures) using the **show()** method(outside the loop) from the **matplotlib.pyplot** library.	

```python
i= 0

# Loop through an 9 image files
for file in img_files[51:60]:

  # full path of the image
  img_files_path = os.path.join(testing_image_directory, file)

  # load image
  img_1 = load_img(img_files_path,target_size=(180, 180))

  # convert image to an array
  img_2 = img_to_array(img_1)

  # increase the dimension
  img_3 = np.expand_dims(img_2, axis=0)

  # predict the class of an unseen image
  prediction = model.predict(img_3)
  # print(prediction)

  predict_class = np.argmax(prediction, axis=1)
  # print(predict_class)

  # plot the image using subplot
  pyplot.subplot(3, 3, i+1)
  pyplot.imshow(img_2.astype('uint8'))

  # Add title of the plot as predicted class value
  pyplot.title(predict_class[0])

  # Do not show x and y axis with the image
  pyplot.axis('off')

  i=i+1

pyplot.show()
```

Output : We can now see the images, each with their class labels



We could teach machines to identify X-Ray images with Pneumothorax disease infection!

You did amazing work today!

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

Teacher Starts Slideshow
Slide 16 to 21

Activity Details:

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ

WhiteHat Jr
Live online Coding for Kids

Click on In-Class Quiz

Continue WRAP-UP Session
Slide 22 to 27

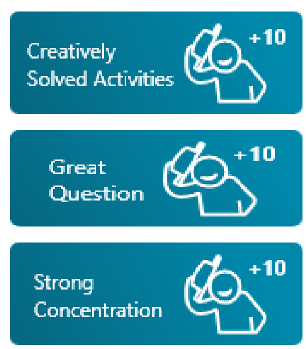
Activity Details:

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate the student for his/her efforts in the class.**
- **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**

Teacher Action	Student Action
You get Hats off for your excellent work!	*Make sure you have given at least 2 Hats Off during the class for:* 

PROJECT OVERVIEW DISCUSSION
Refer the document below in Activity Links Sections

Teacher Clicks	✖ End Class
ADDITIONAL ACTIVITIES	
Encourage the student to plot the accuracy of the trained model. **Visualize the accuracy of the model**: It is always good to understand the result visually. We will use **matplotlib.pyplot** library to plot the accuracy points of all epochs(20 epochs in this case). This will help us to see the accuracy of the model during all epochs visually: We started the model history into a variable called history. 1. Import **matplotlib.pyplot** library 2. Take 2 variables, **acc** and **val_acc** to get the values of the **accuracy** and **validation_accuracy** of the accuracy metric stored in the history variable. 3. Print **acc** or **val_acc** to check the values, which will be two 2 lists with **accuracy** data and **validation_accuray** data of the model.	

```
from matplotlib import pyplot

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

print(acc)
print(val_acc)
```

```
[0.5049999952316284, 0.5099999904632568, 0.5, 0.5,
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.44999998807907104,
```

4. Use the **plot(x, y, fmt)** method to plot the graph figure:

    a. **Plot Training Accuracy Data Points**:

        i. **x**: The x value to plot the graph. This will be equal to the length of the **acc**(or **val_acc**) list.
        Take a variable **epochs = range(len(acc))**

        ii. **y**: The y value to plot the graph. This will be value from **acc** list values.

        iii. **fmt**: formatted strings for color. We can use '**r**' for red color points.

    b. **Plot Validation Accuracy Data Points**:

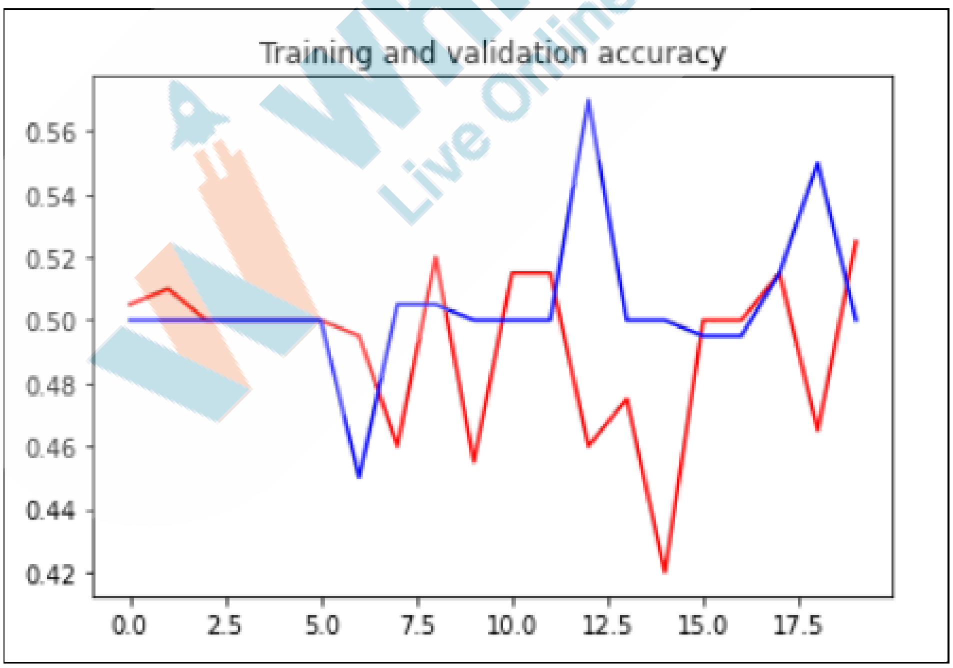
        i. **x**: The x value to plot the graph. This will be equal to the length of the **acc**(or **val_acc**) list.

ii. **y**: The y value to plot the graph. This will be a value from **val_acc** list values.

iii. **fmt**: formatted strings for color. We can use '**b**' for blue color points.

5. Use the **title()** method to add the title of the plot.
6. Use **show()** method to show the graph figure.

```
epochs = range(len(acc))

pyplot.plot(epochs, acc, 'r')
pyplot.plot(epochs, val_acc, 'b')

pyplot.title('Training and validation accuracy')
```



We can see how the accuracy of the model starts improving as epoch increases, but drops in between with some points.
When we look at this graph, it is difficult to see what the red line represents and what the blue line represents.

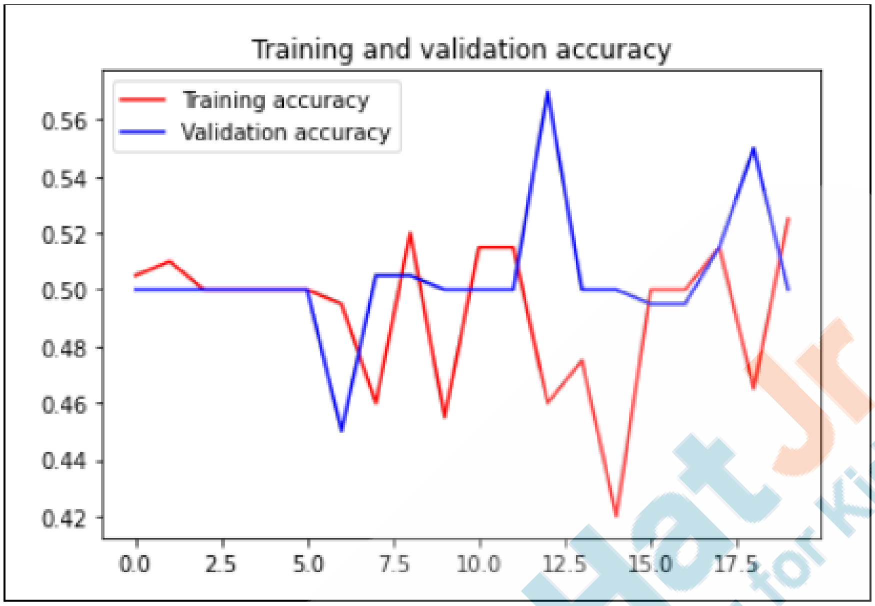
For this, we can add the information at top, about which line represents which information:

7. Use **legend()** method to add the legend in the graph figure in the top left corner.

   The legend is a side section of the chart that gives a small text description of each plot data series(here we have 2 data series one for acc, and other for val_acc series)

8. Update **plot()** method as **plot(x, y, fmt, label)**:
   label: text will be displayed in the legend.

```
from matplotlib import pyplot

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = range(len(acc))

pyplot.plot(epochs, acc, 'r', label='Training accuracy')
pyplot.plot(epochs, val_acc, 'b', label='Validation accuracy')

pyplot.title('Training and validation accuracy')

pyplot.legend()

pyplot.show()
```



ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Teacher Boilerplate Code	https://colab.research.google.com/drive/1-ETZ7OGVR3u1P4aJ58qA5_VzQ7yTO2O0?usp=sharing
Teacher Activity 2	Maximum Value Along Axis 1	https://s3-whjr-curriculum-uploads.whjr.online/bc0f4eda-1116-4064-b49f-d4666a73ffef.jpg
Teacher Activity 3	Index of Maximum Value Along Axis 1	https://s3-whjr-curriculum-uploads.whjr.online/5be15b97-cd98-414f-8880-1cd592560247.png
Teacher Activity 4	Reference Code	https://colab.research.google.com/drive/18pahhCylsm8QiRP--2FJ0av-CRrG7S_W?usp=sharing
Student Activity 1	Student Boilerplate Code	https://colab.research.google.com/drive/1KXUBnJc2S2qbP0uTd3fBMZanQpwL9rDG?usp=sharing
Teacher Reference 1	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/dc3a69ee-760a-439b-8370-886c412e32b4.pdf

Teacher Reference 2	Project Solution	https://colab.research.google.com/drive/1J0J7-WJhenhz5Pq9TtpeoYCqd4MQgL4V?authuser=7#scrollTo=-U38f_TwqzRZ
Teacher Reference 3	Visual Aid Link	https://s3-whjr-curriculum-uploads.whjr.online/d57a6456-38ac-4015-8816-cbf85bfbfbbb.html
Teacher Reference 4	In Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/e4cb5c78-8d77-4f9e-bb4c-9982c630b4dd.pdf