In [249…
```
%run Imports.ipynb
```

Introduction

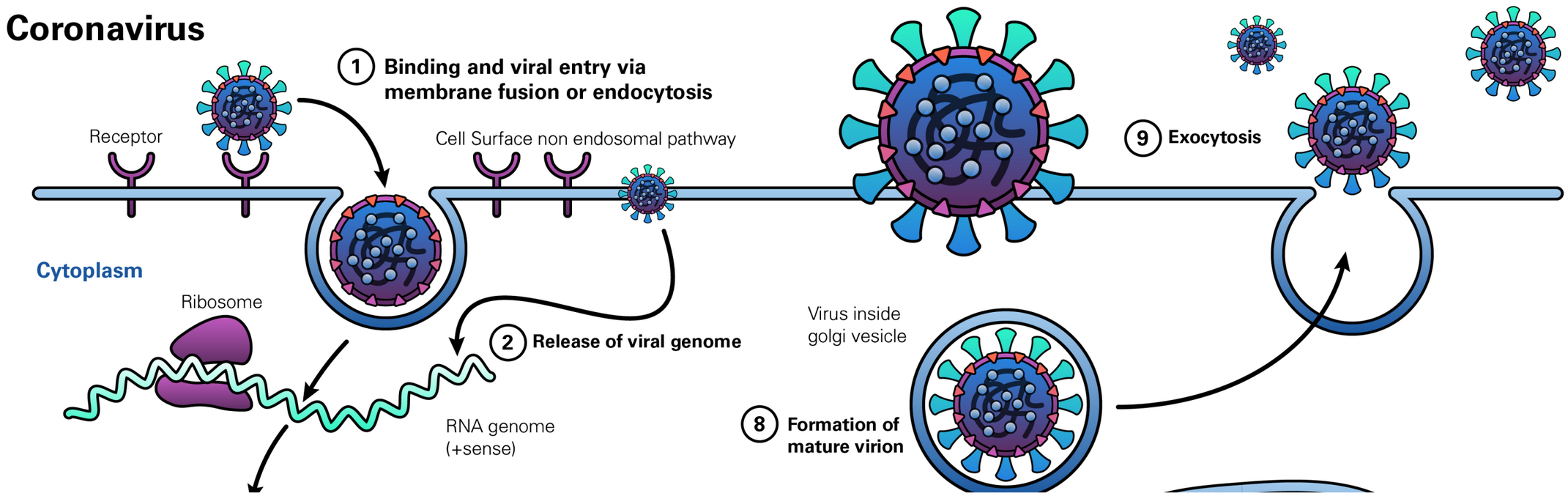
In this report, the biology of sars-cov-2 is discussed and the sars-cov-2 genome is analyzed. Deep learning will be used to determine whether a CT-scan has sars-cov-2 and to predict the following nucleotide in the genome sequence.

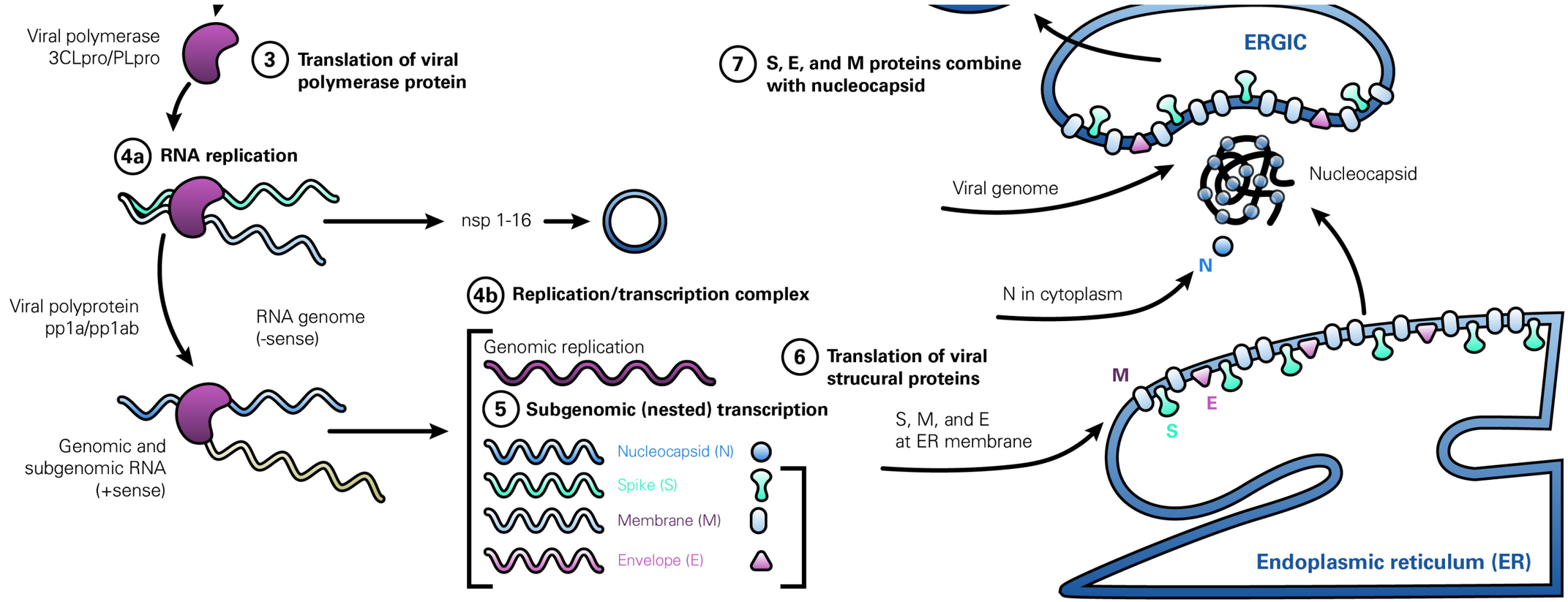
SARS-CoV-2 Biology

Coronaviruses are a sphere-like 20-sided polygon, icosahedron, shape and belong to the Nidovirales order. SARS-CoV-2 travels through the respitory tract and attaches itself to alveoli, air sacs, in the lungs. Coronaviruses are named after the dyed red spike glycoproteins (S) that cover them and attach these protiens by being activated by the host cell's proteases, ACE2 and TMPRSS2 ennzymes, at a sequential proteolytic cleavage S1 and S2 site to fuse with a host cell's enzymes to enter, endocytosis, the cell's plasma membrane to the cytosol. S1 is the end binding region of the S protein and S2 is the beam of the S protien. The spike glcoproteins can also attach to a cell's endosomes being activated by the host cell's CTSL enzyme. Inside of the icosahedron positive-sensed lipid bilayer envelope is a helical capsid that encloses an RNA genome (gRNA), which encodes the viral genetic information -- non-segmented-single-stranded RNA -- that is used during the lytic cycle to generate RNA polymerase, protease, and other proteins. The envelope is removed allowing genomic RNA to enter the cytoplasm where some of it is translated into pp1a and pp1ab proteins. Translation is when codons, groups of three nucleotides, move along a ribosome through tRNA that translates the codons to anticodons that generate an amino acid polypeptide chain of proteins. Both proteins are then cleaved by protease making 16 nonstructural proteins, NSPs. Some of the NSPs form a replication and transcription complex (RTC) in which gRNA is replicated and RNA-dependent RNA polymerase (RdRp) uses gRNA as a template to transcribe (-) subgenomic RNA (sgRNA) and then transcribe it back to (+) sgRNA, which is used as the genome of the new virus. sgRNA is translated into strutural proteins, wich are spike protein (S), envelope protein (E), membrane protein (M), and nucleocapsid protein (N). Nucleocapsid proteins assign the viral replication-transcription complexes (RTCs), where gRNA and sgRNAs are synthesized. Spike, envelope, and membrane proteins enter the endoplasmic reticulum, and the nucleocapsid protein encloses the sgRNA becoming a nucleoprotein complex. The proteins and nucleoprotein complex combine into a virus endoplasmic reticulum-Golgi apparatus, and then are excreted into a vesicle. The dyed yellow envelope proteins (E) and dyed orange membrane glycoproteins (M) assist in budding -- using the host cell's plasma membrane for envelope formation -- and then exiting, exocytosis, the host cell through lysis. The following is a diagram of the SARS-CoV-2 lifecycle,

In [4]:
```
Image(filename='Coronavirus_Life_Cycle.jpg')
```

Out[4]:

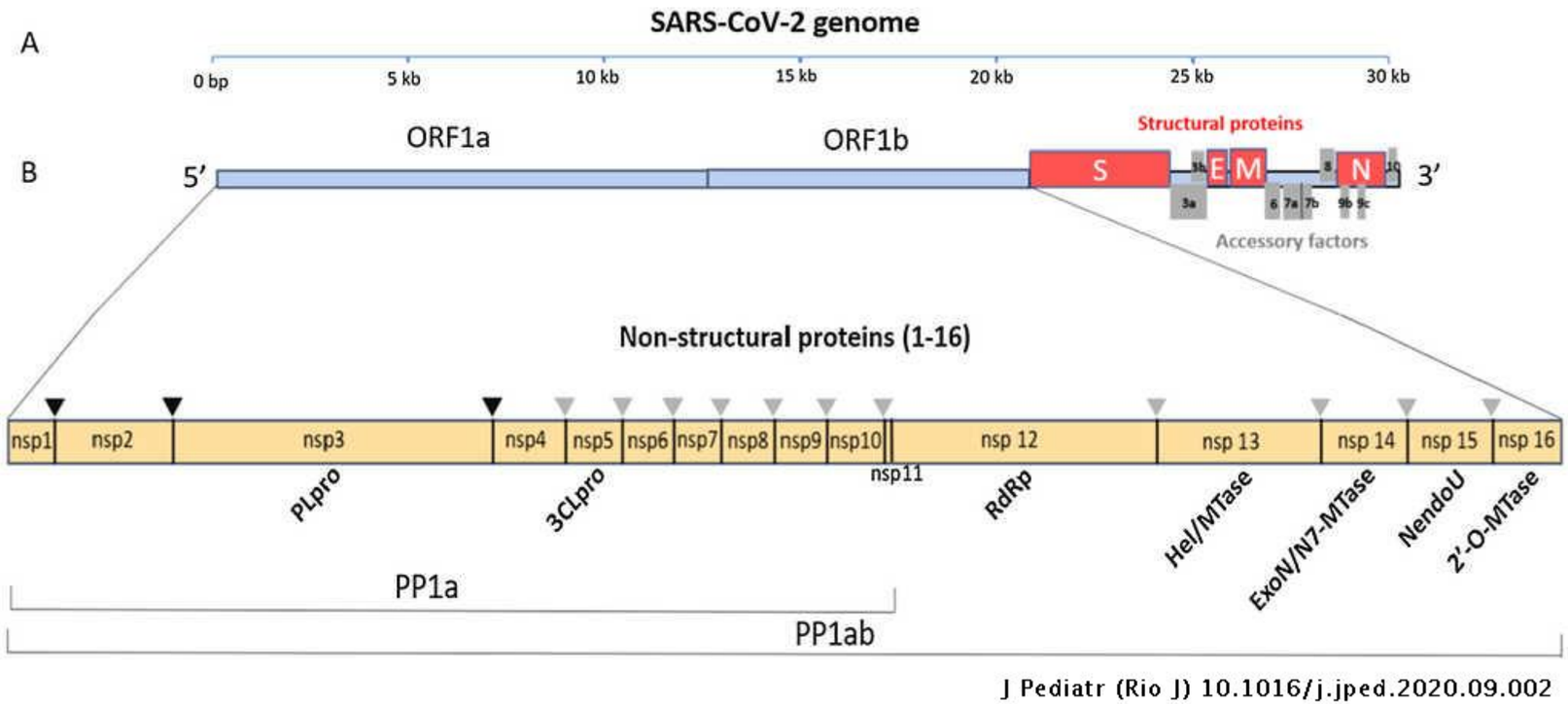




Sars-cov-2 has the largest RNA viral genome, which ranges from 26 to 32 kb. The genome of the virus starts with 5' carbon base, has an ORF1ab, S, E, M, N, and accesory region, and then end with 3' carbon base. ORF1ab are non-structural protiens. ORF1ab, open reading frames, comprises of 16 NSP regions. Nsp1 and nsp2 encode enzymes that bind to RNA and suppress gene expression. Nsp3 encodes enzymes that assist in the translation of viral mRNAs. Nsp4 encodes viroporin, which modify cellular membranes. Nsp5 encodes the organization of the viroplasm, where viral replication and assembly occur. Nsp6 generates autophagosomes, vesicles with cellular material that will be removed through autophagy. Nsp7 through 16 encode the generation of RNA synthesis and processing. Nsp7 through 11 encode the contribution to the nsp interactome, molecular interactions. Nsp12 encodes RNA polymerase. Nsp13 encodes helicase. Nsp14 through 16 encode mRNA capping, attaches the 5' cap to mRNA. The S region encodes the spike proteins, which bind to host cell receptors and fuse to the cell membrane. The M and E regions encode the membrane and envelope protiens, which assist in budding. The N region encodes the nucleocapsid protein, which assigns the viral replication-transcription complexes. The genome additionally contains an ORF coding for accessory proteins, which are not essential in virus replication but assist in pathogenesis, development of the virus.

In [13]:
```
Image(filename='SARS-COV-2_Genome.jpeg')
```

Out[13]:

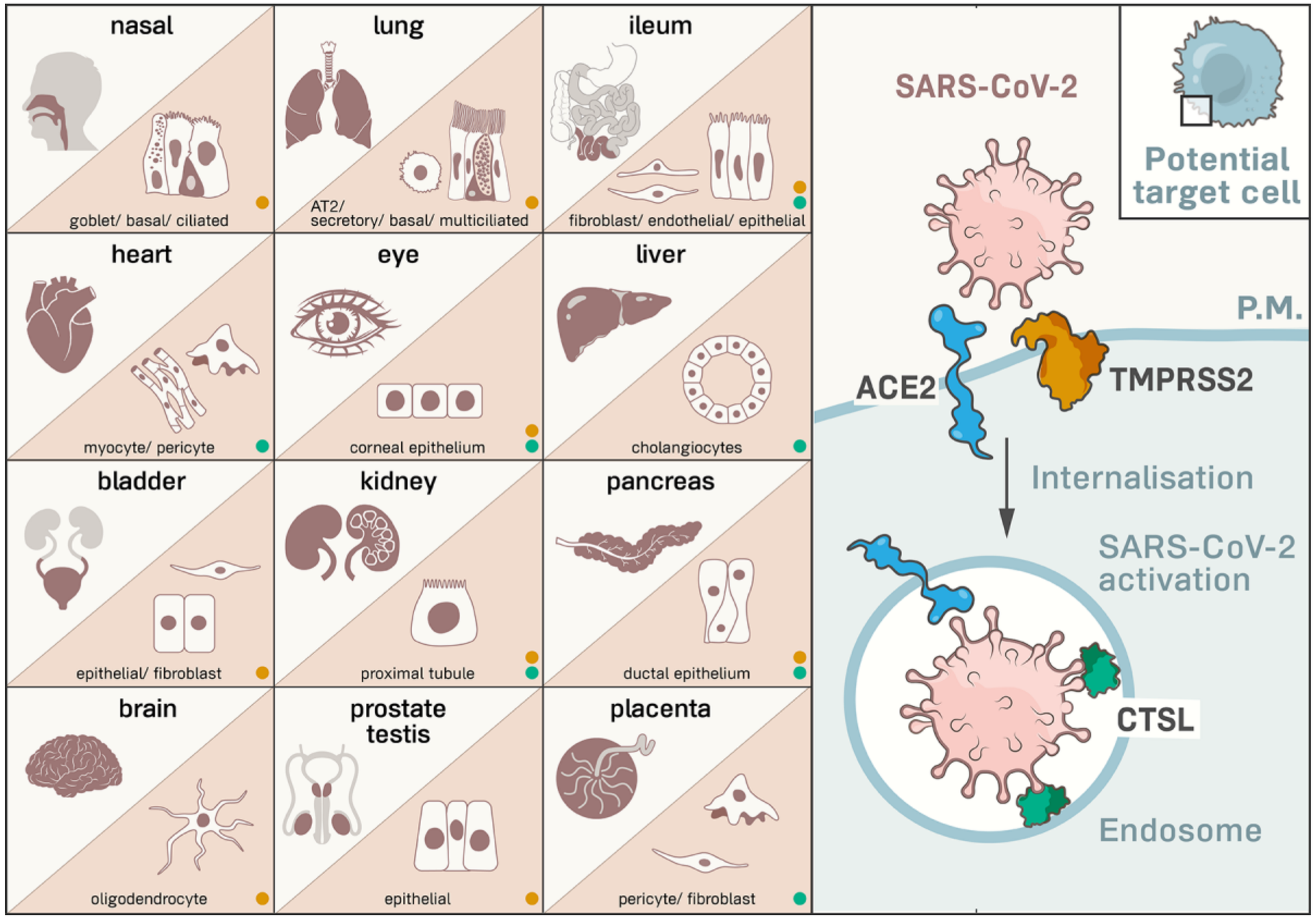


J Pediatr (Rio J) 10.1016/j.jped.2020.09.002

ACE2 modulates the activity of the angiotensin II protein, which increases blood pressure and inflammation. The SARS-CoV-2 virus binding to ACE2 inhibits ACE2 and can cause an increase in cell inflammation resulting in the death of cells in the alveoli units with hypoxic fluid flooding, an increase in blood preasure damaging blood vessels causing microvascular thrombosis, and the tranduction of the kidney podocytes resulting in acute kidney injury. The following is a diagram of the enzymes that get inhibited and the parts of the body that can get damaged.

In [5]:
```
Image(filename='ace2-thumb-l.png')
```

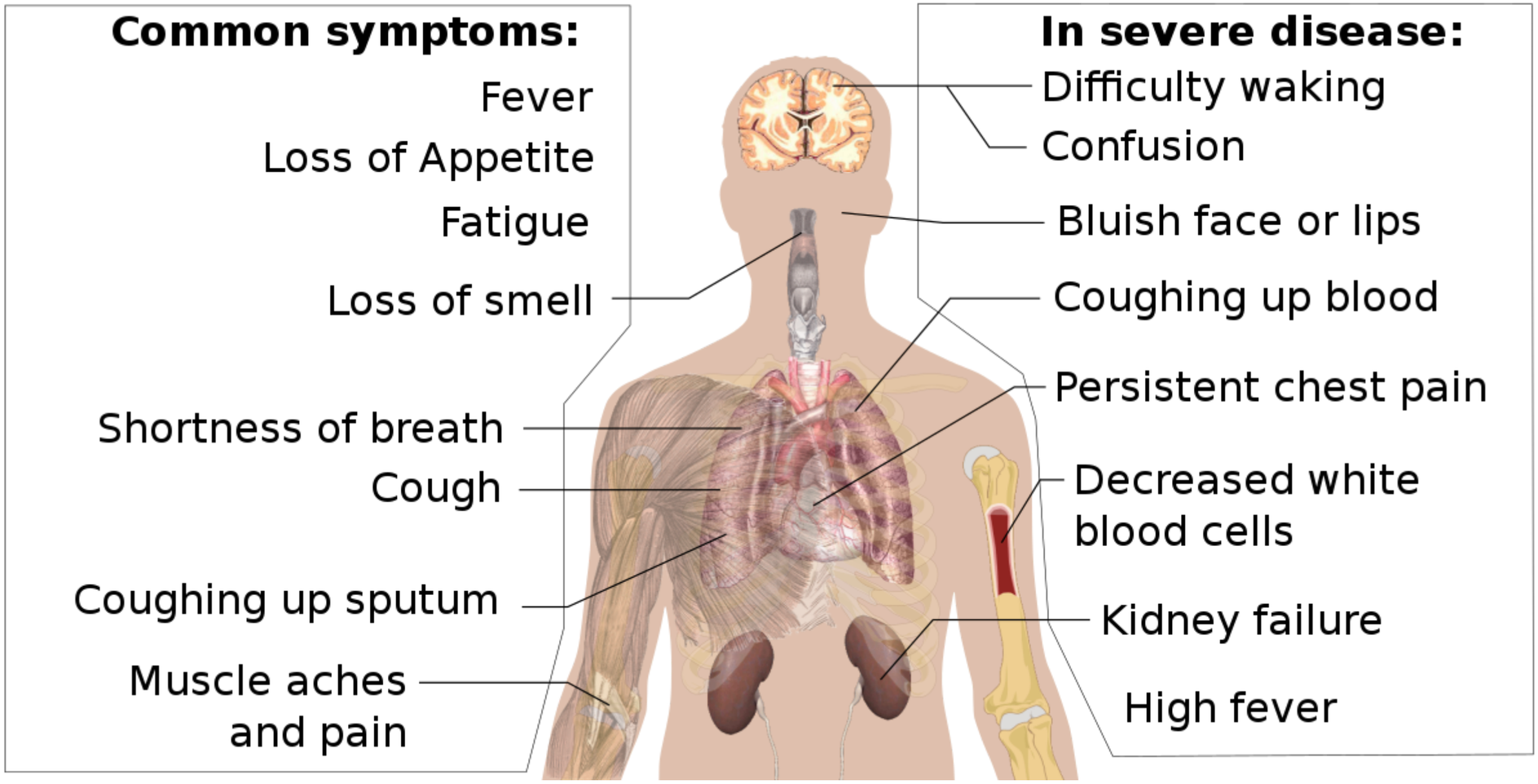
Out[5]:



SARS-CoV-2 is spread when an infected person coughs or sneezes droplets of saliva or mucus with the virus into the air. The respiratory droplets usually do not go further than a few feet, are airbourne for a few moments, and then land on a surface. SARS-CoV-2 has an incubation period of 2 to 14 days and the symptoms of the virus include cough, fever or chills, shortness of breath or difficulty breathing, muscle or body aches, sore throat, loss of taste or smell, diarrhea, headache, fatigue, nausea or vomiting, congestion or runny nose, and in rare cases the virus can lead to difficulty walking, confusion, bluish face or lips, coughing up blood, severe respiratory problems, kidney failure, high fever, or death. The following is a diagram of the symptoms.

In [4]:
```
Image(filename='Symptoms_of_coronavirus_disease_2019_4.0.svg.png')
```

Out[4]:

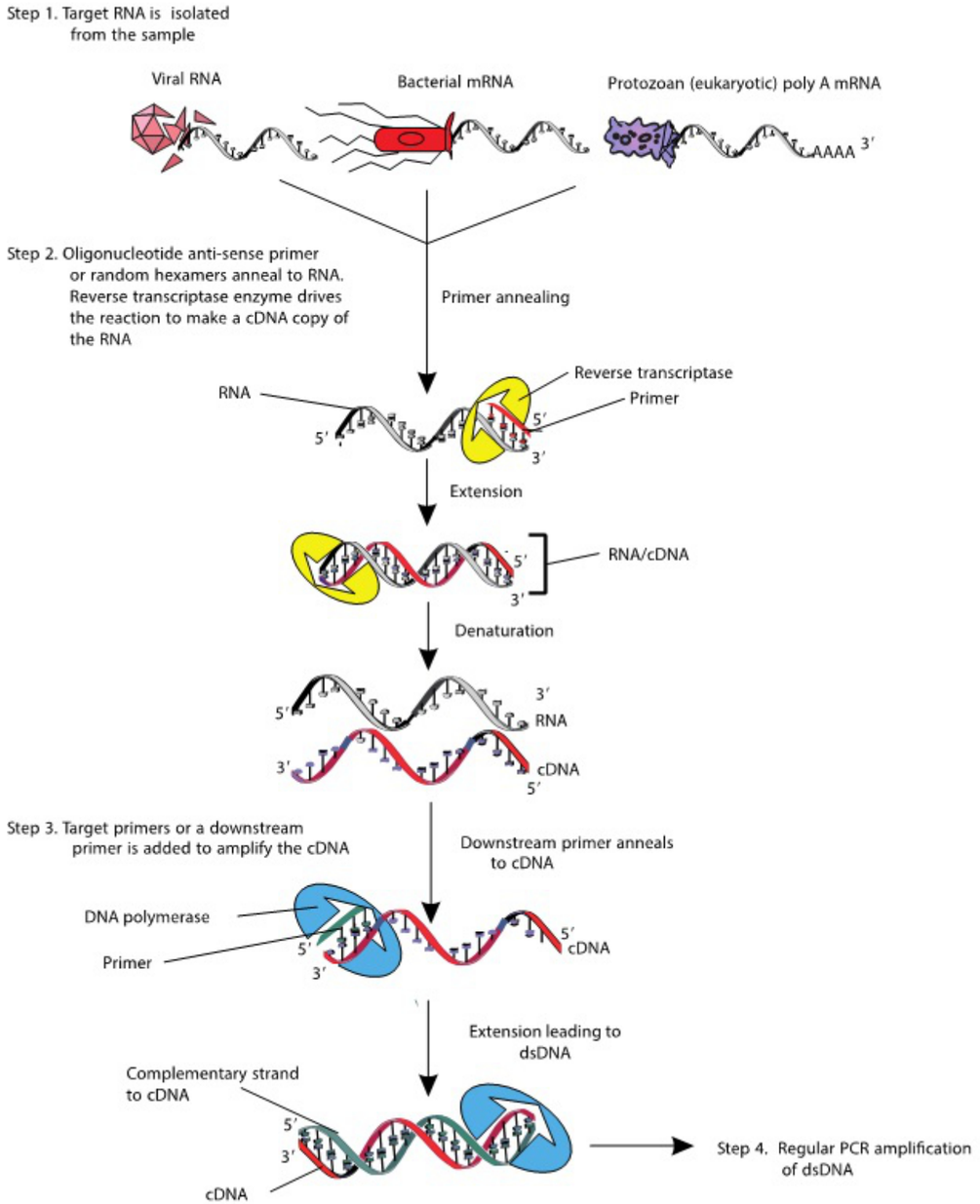


Testing for SARS-CoV-2

A patient can be tested for SARS-CoV-2 with the sequence-specific molecular nucleic acid assay, or the antigen-specific immunoassay. For a molecular-assay, a nasal or saliva sample is collected from upper respiratory fluid and then Real-time RT-PCR is conducted to quantify sequences within the RNA samples. Reverse transcriptase converts extracted RNA into cDNA that is used as a template for DNA polymerase to complete the strand of dsDNA and then RT-PCR amplifies the genetic regions and fluorescent probes bind to the regions for identification. For an immuno-assay, antibodies are put on a membrane and complexed with a potentialy virulent sample of which any antigen will be trapped that results in the membrane changing color. A SARS-CoV-2 recovered patient can be tested for whether the patient has developed anibodies against the virus by checking a blood sample for the antibodies. The following are diagrams of a molecular assay

In [6]:
```
Image(filename='10104fig1.jpg')
```

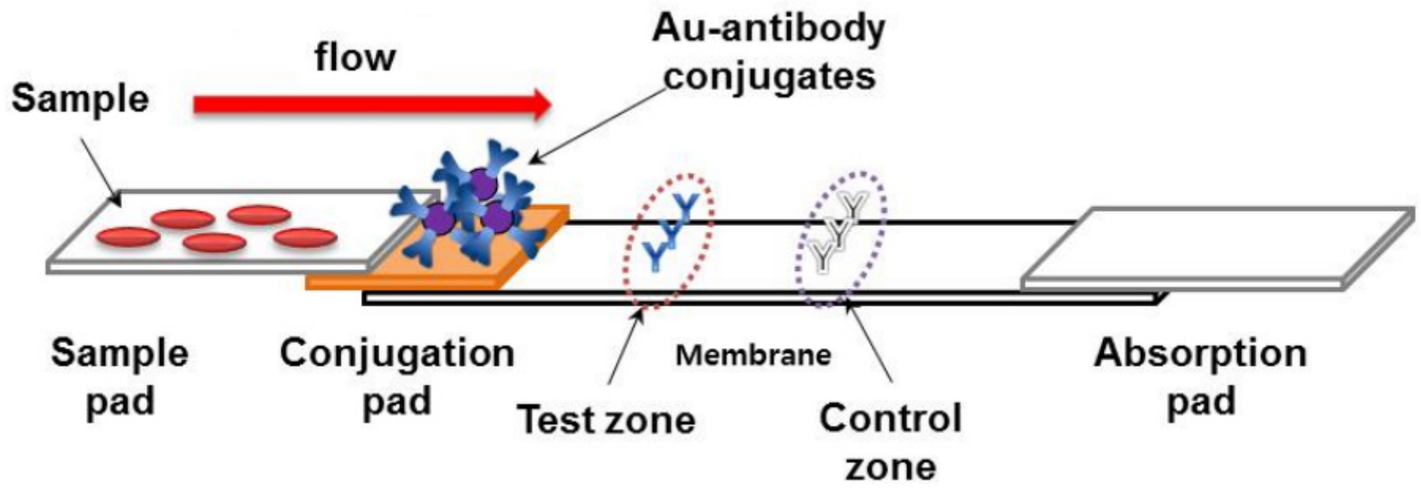
Out[6]:



and an immuno assay.

In [7]:
```
Image(filename='tag-lateral-flow-immunoassay-2.jpg')
```

Out[7]:



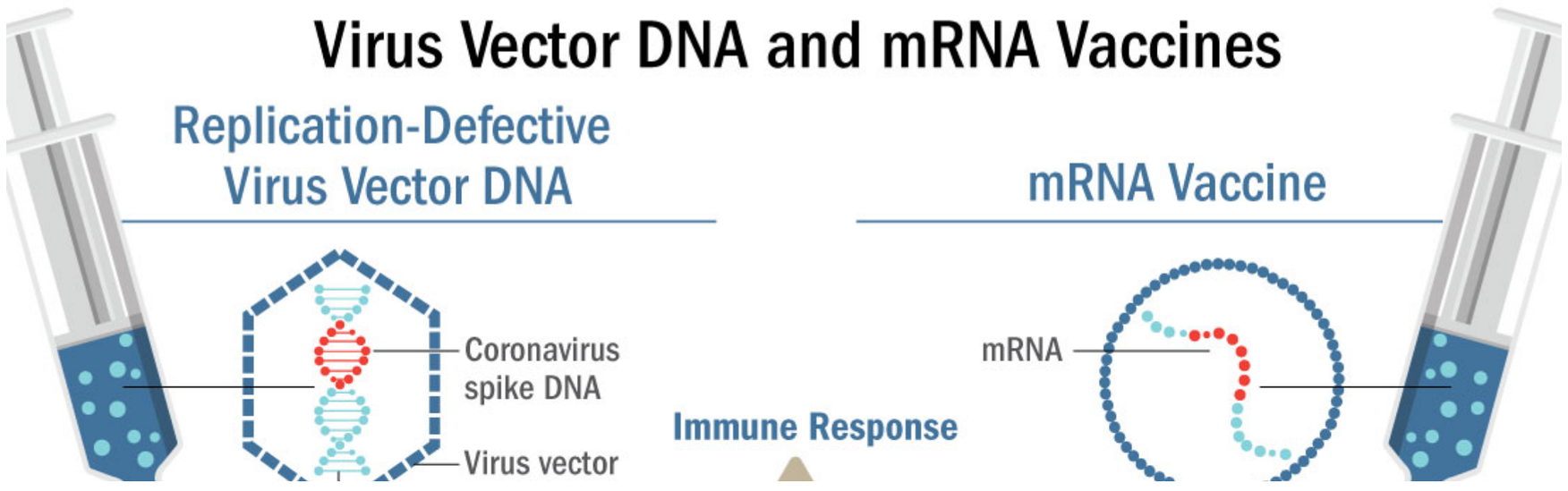
Vaccines for SARS-CoV-2

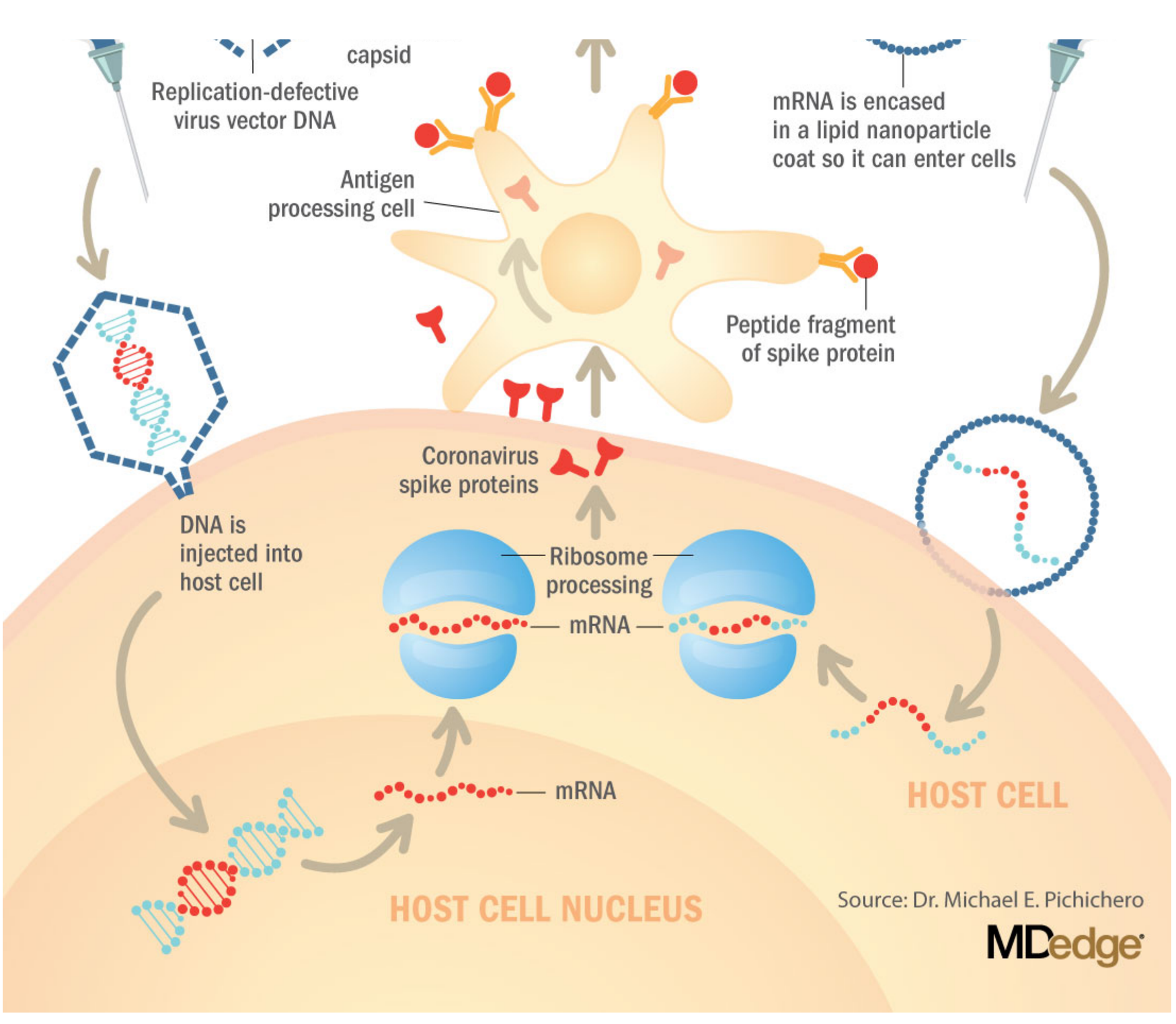
Vaccines have been developed to combat SARS-CoV-2. The Pfizer/BioNTech vaccine (BNT162b2) and The ModernaTX, Inc vaccine (mRNA-1273) are mRNA vaccines. mRNA vaccines deliver lab designed viral mRNA made by coding both the 5′-untranslated region (UTR) and the 3′-UTR into cells where it is translated into the encoded antigen to which an immune recognition by white blood cells, immunogenicity, results. The AstraZeneca in collaboration with the University of Oxford vaccine (AZD1222) is an adenovirus vaccine. Adenovirus vaccines are first gene sub-cloned into an intermediary vector, transfered to an adenovirus recombinant genome vector, transfected into packaging cells, amplified into a culture stock, and then titrated to determine the concentration of active adenoviruses in the stock. The adenoviruse is then delivered to cells where immunogenicity results. The following is a diagram of Virus vector DNA and mRNA vaccine immune response production.

The following is a diagram of Virus vector DNA and mRNA vaccine immune response production. Vaccines have been developed to combat SARS-CoV-2. The Pfizer/BioNTech vaccine (BNT162b2) and The ModernaTX, Inc vaccine (mRNA-1273) are mRNA vaccines. mRNA vaccines deliver lab designed viral mRNA made by coding both the 5′-untranslated region (UTR) and the 3′-UTR into cells where it is translated into the encoded antigen to which an immune recognition by white blood cells, immunogenicity, results. The AstraZeneca in collaboration with the University of Oxford vaccine (AZD1222) is an adenovirus vaccine. Adenovirus vaccines are first gene sub-cloned into an intermediary vector, transfered to an adenovirus recombinant genome vector, transfected into packaging cells, amplified into a culture stock, and then titrated to determine the concentration of active adenoviruses in the stock. The adenoviruse is then delivered to cells where immunogenicity results. The following is a diagram of Virus vector DNA and mRNA vaccine immune response production.

In [8]:
```
Image(filename='DNA_and_mrNA_Vaccines_web.jpeg')
```

Out[8]:



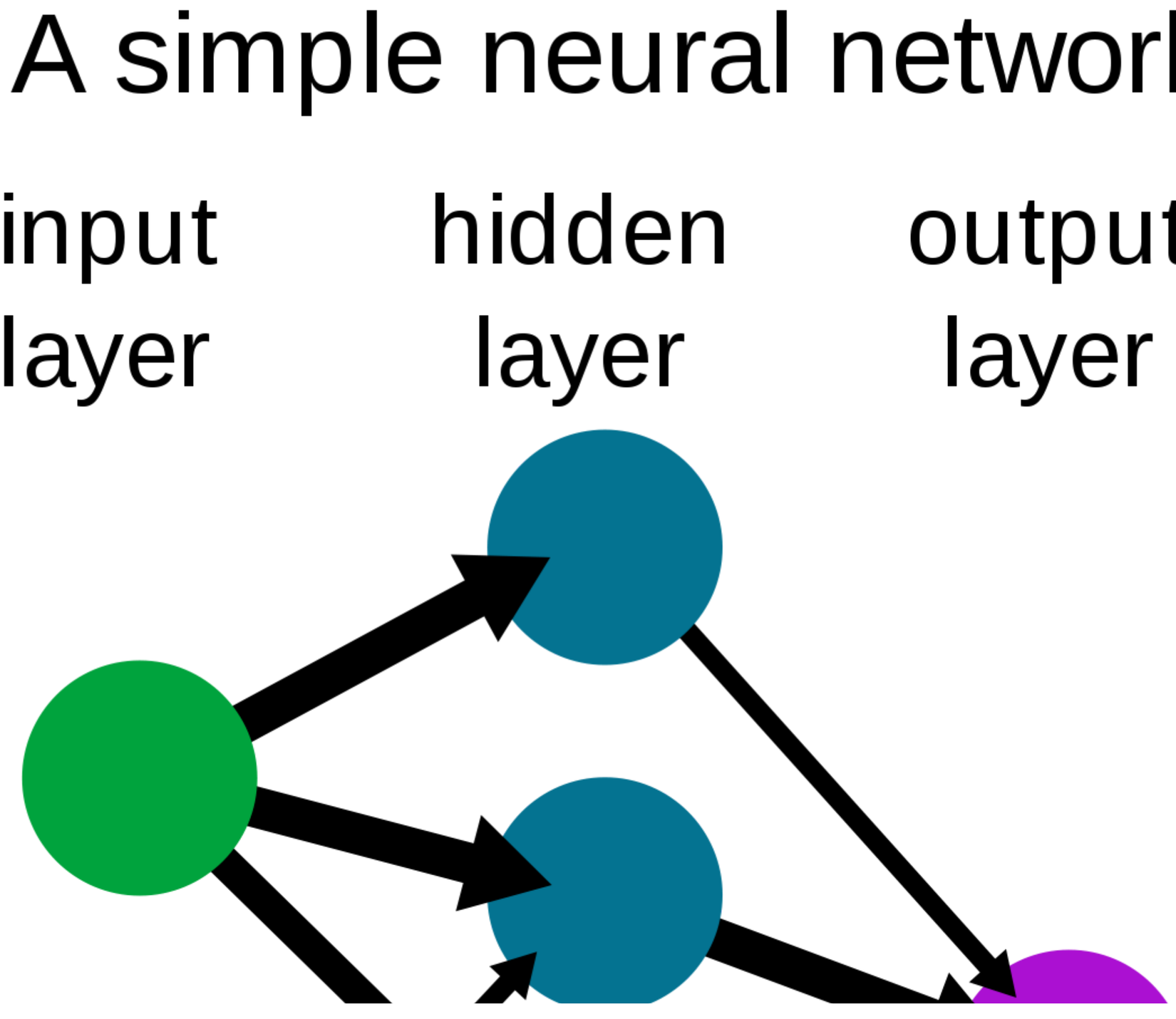


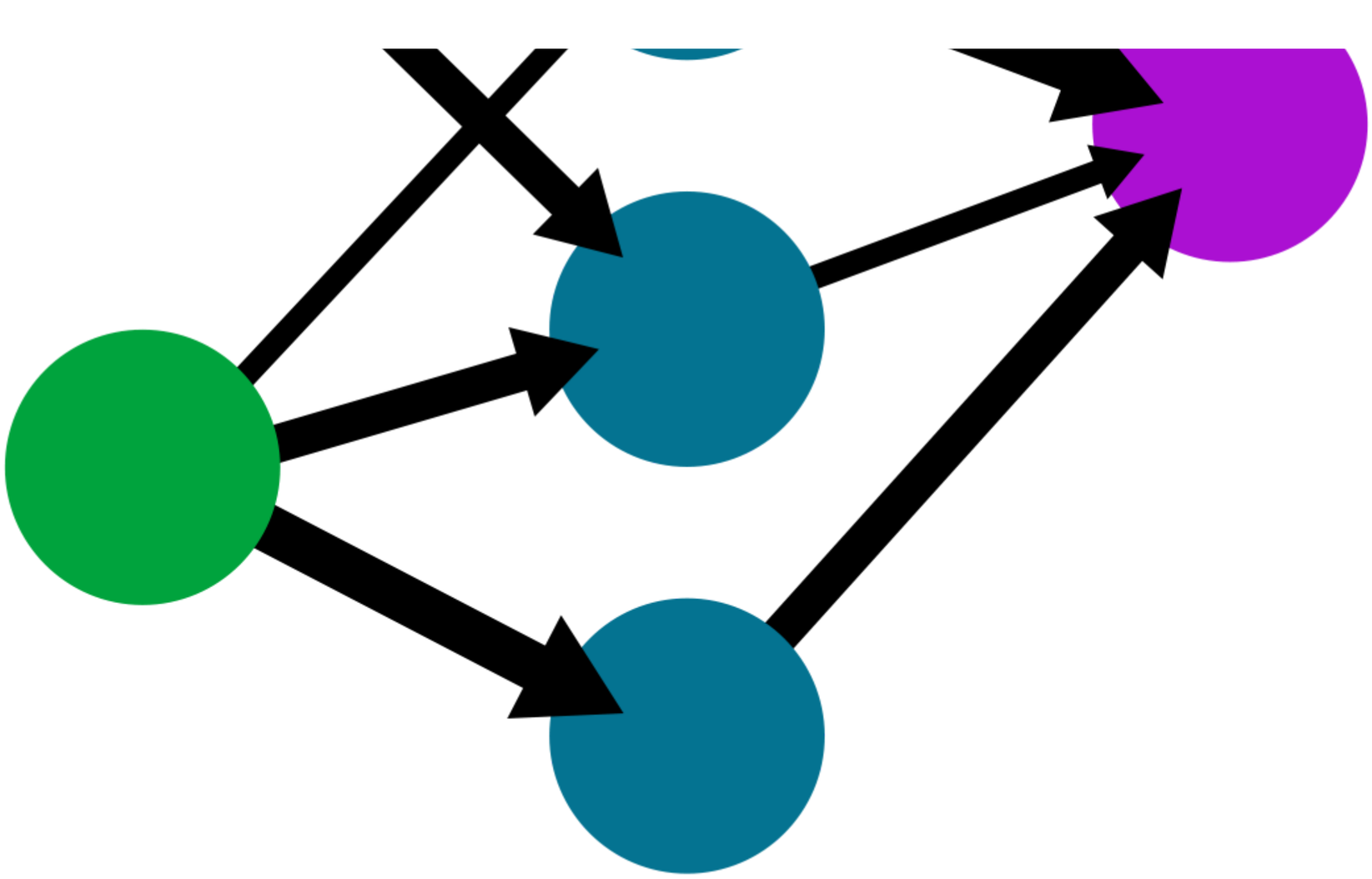
Neural Networks

A simple neural network has an input layer followed by a hidden layer and an output layer.

In [162…
```
Image(filename='ANN.png')
```

Out[162…

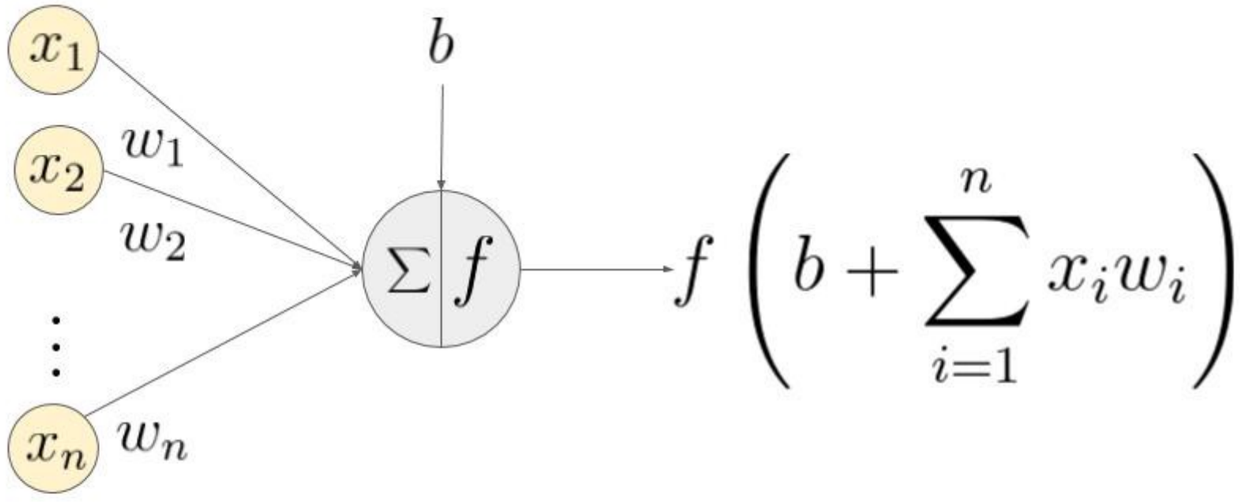




A neural network uses weights and bias through an activation function to determine which neurons will be passed to the next layer of the network.

In [163…
```
Image(filename='wb.jpeg')
```

Out[163…



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Load data of sars-cov-2 test CT-scans.

In [65]:
```
directory=r"/Users/owner/Downloads/archive 2"
```

Feature Engineering

In [68]:
```
# Pixels range between values of 1 to 255
datagen = ImageDataGenerator(rescale=1.0/255.0)


generator = datagen.flow_from_directory(
        # This is the target directory
        directory,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=2481,
        # Slabels are positive or negative for virus
        class_mode='binary')
```

```
Found 2481 images belonging to 2 classes.
```

In [69]:
```
images, labels = next(generator)
```

In [356…
```
X_train=images[:1501]
```

```
y_train=labels[:1501]
X_val=images[1501:1901]
y_val=labels[1501:1901]
X_test=images[1901:2482]
y_test=labels[1901:2482]
```

Data Mining

In [73]:
```
X_train.shape
```

Out[73]:
```
(1501, 150, 150, 3)
```

In [79]:
```
y_train.shape
```

Out[79]:
```
(1501,)
```

The train and val images are similarly distributed.

In [75]:
```
X_train.mean()
```

Out[75]:
```
0.64016646
```

In [76]:
```
y_train.mean()
```

Out[76]:
```
0.4903398
```

In [77]:
```
X_val.mean()
```

Out[77]:
```
0.6438065
```

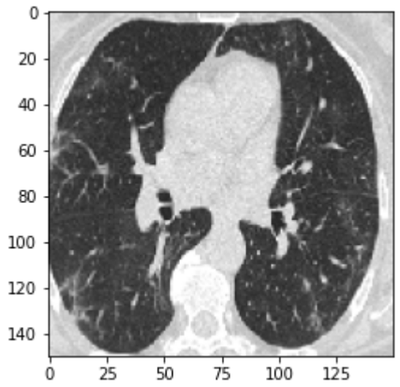
In [78]:
```
y_val.mean()
```

Out[78]:
```
0.5275
```

Data Exploration

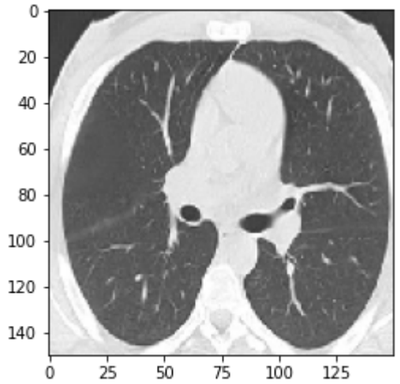
In [80]:
```
plt.imshow(X_train[0], cmap="gray")
print("class", y_train[0])
plt.show()
```

```
class 0.0
```



In [81]:
```
plt.imshow(X_train[1], cmap="gray")
print("class", y_train[1])
plt.show()
```

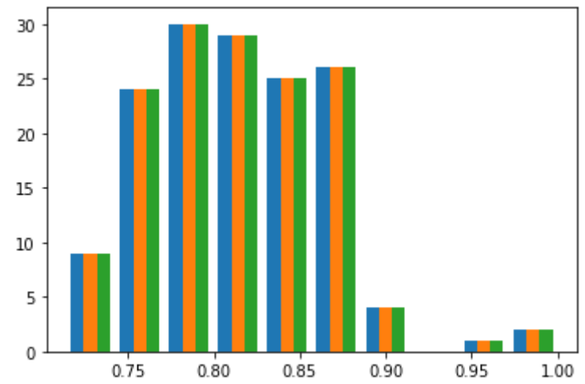
```
class 1.0
```



Distribution of image from class 0.

In [82]:
```
plt.hist(X_train[0][0])
```

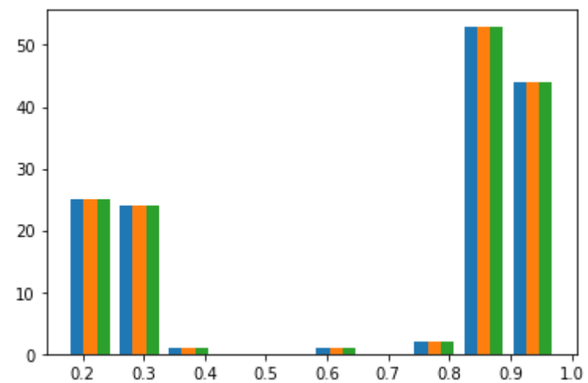
Out[82]:
```
(array([[ 9., 24., 30., 29., 25., 26.,  4.,  0.,  1.,  2.],
        [ 9., 24., 30., 29., 25., 26.,  4.,  0.,  1.,  2.],
        [ 9., 24., 30., 29., 25., 26.,  4.,  0.,  1.,  2.]]),
 array([0.7137255 , 0.74235296, 0.7709804 , 0.7996079 , 0.8282353 ,
        0.8568628 , 0.8854902 , 0.91411763, 0.9427451 , 0.97137254,
        1.        ], dtype=float32),
 <a list of 3 BarContainer objects>)
```



Distribution of image from class 1.

In [83]:
```
plt.hist(X_train[1][1])
```

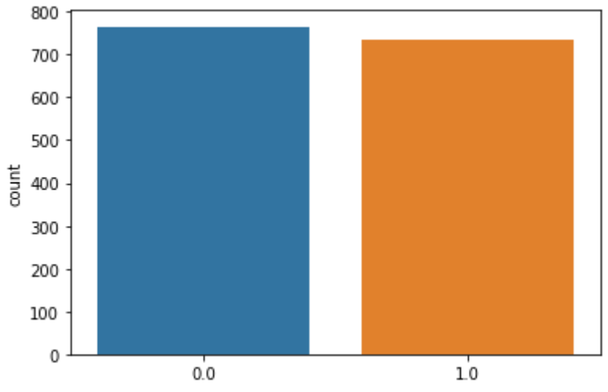
Out[83]:
```
(array([[25., 24.,  1.,  0.,  0.,  1.,  0.,  2., 53., 44.],
        [25., 24.,  1.,  0.,  0.,  1.,  0.,  2., 53., 44.],
        [25., 24.,  1.,  0.,  0.,  1.,  0.,  2., 53., 44.]]),
 array([0.17254902, 0.2529412 , 0.33333334, 0.41372553, 0.49411768,
        0.57450986, 0.654902  , 0.73529416, 0.81568635, 0.89607847,
        0.97647065], dtype=float32),
 <a list of 3 BarContainer objects>)
```



There is no class imbalance.

In [84]:
```
sns.countplot(y_train)
```

Out[84]:
```
<AxesSubplot:ylabel='count'>
```



ANOVA

An anova model uses an f-test to determine whether to accept or reject the null hypothesis. RSS is the residual sum of squares, ESS is the explained sum of squares, TSS is the total sum of squares, k is the number of groups, and n is the number of data points.

$$RSS = \sum (y - \hat{y})^2$$

$$ESS = \sum (\hat{y} - \bar{y})^2$$

$$TSS = RSS + ESS$$

$$MStreatment = \frac{TSS}{k-1}$$

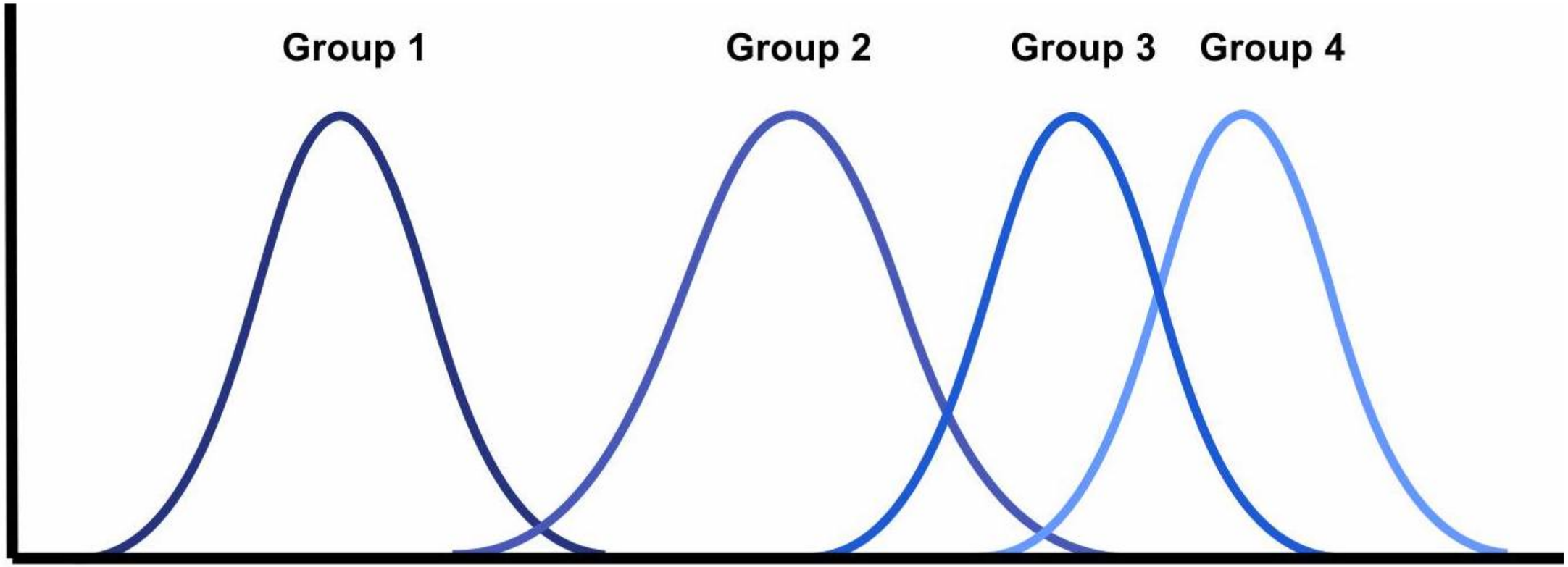
$$MSerror = \frac{RSS}{n-k}$$

$$F = \frac{MStreatment}{MSerror}$$

In [333…

```
Image(filename='anova.jpg')
```

Out[333…



$$H0 : \mu1 = \mu2 = \mu3 = \mu4$$

$$H1 : \mu1 \neq \mu2 \neq \mu3 \neq \mu4$$

In [174…
```
stat, p = f_oneway(X_train[0].flatten(),X_train[1].flatten(),X_train[2].flatten(),X_train[3].flatten())
print(f'stat={stat}, p={p}')#Null Hypothesis: no significant difference between the distributions
if p > 0.05:
    print('Don\'t reject null hypothesis of no significant difference between the distributions')
else:
    print('Reject null hypothesis of no significant difference between the distributions')
```

```
stat=10270.120803904983, p=0.0
Reject null hypothesis of no significant difference between the distributions
```

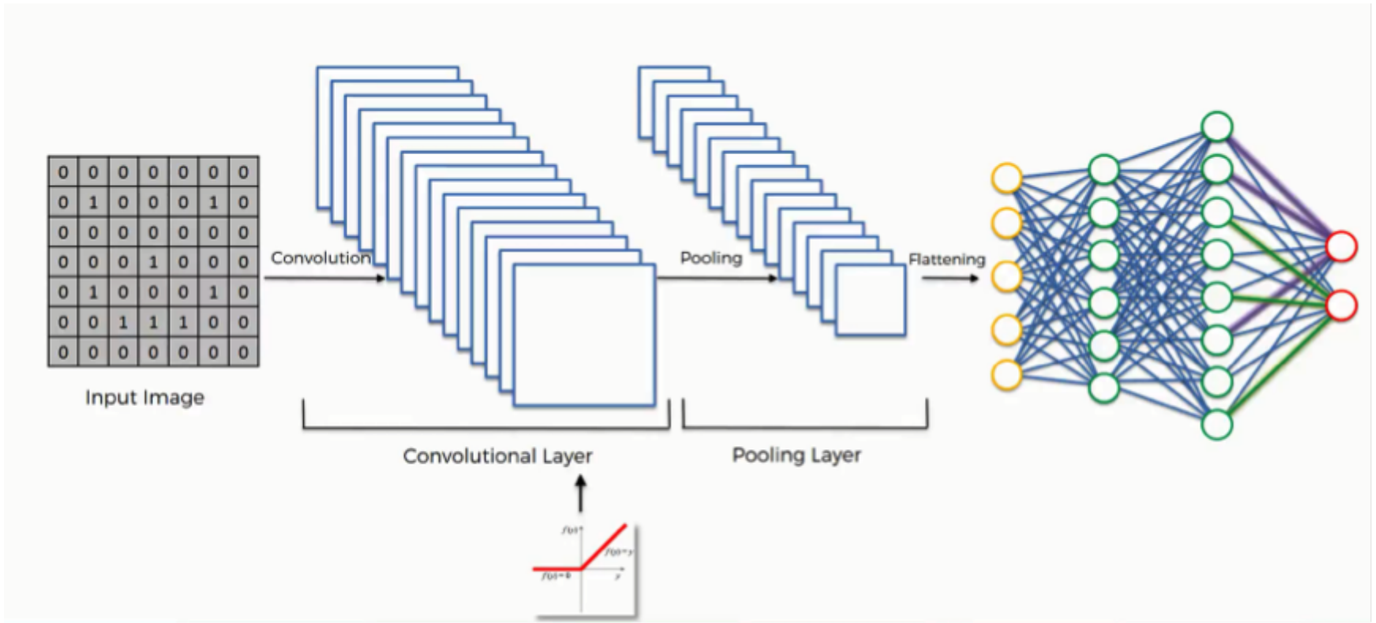
The images are distinguishable.

CNN Model

A convolutional neural network uses forward propagation to take an image represented by a matrix of pixel values, convolutes the image with a filter matrix to find patterns and pools the result to reduce dimensionality, and then flattens the result to be classified.

In [323…
```
Image(filename='cnn.png')
```

Out[323…



Forward propagation begins with the convolutional layer that takes the dot product of the image matrix, x, and the filter matrix, f.

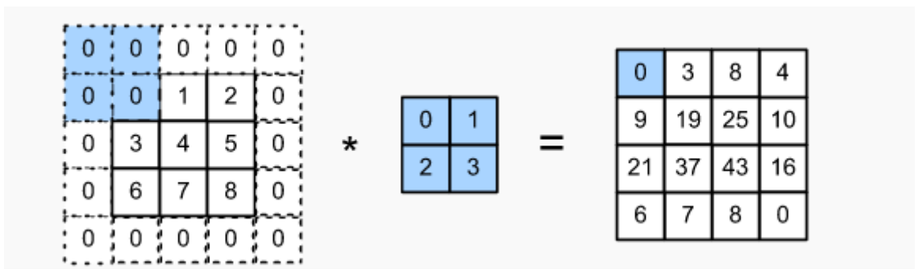
$$Z_1 = x \bullet f$$

A filter begins at the left of the image and then steps across the image. Convolutional layers will lose pixels on the perimeter of the image as the filter progresses. Padding adds extra pixels set to values of zero around the boundary of the image, and stride is the number of rows and columns traversed per step. h is the height, w is the width, p is the padding, and s is the stride.

$$[(xh - fh + ph + sh)/sh] \times [(xw - fw + pw + sw)/sw]$$

In [330…
```
Image(filename='padding.png')
```

Out[330…



The result is then passed through an activation function, which determines the neurons that get passed to the next layer of the network, a pooling layer that is used to reduce the image dimension size.

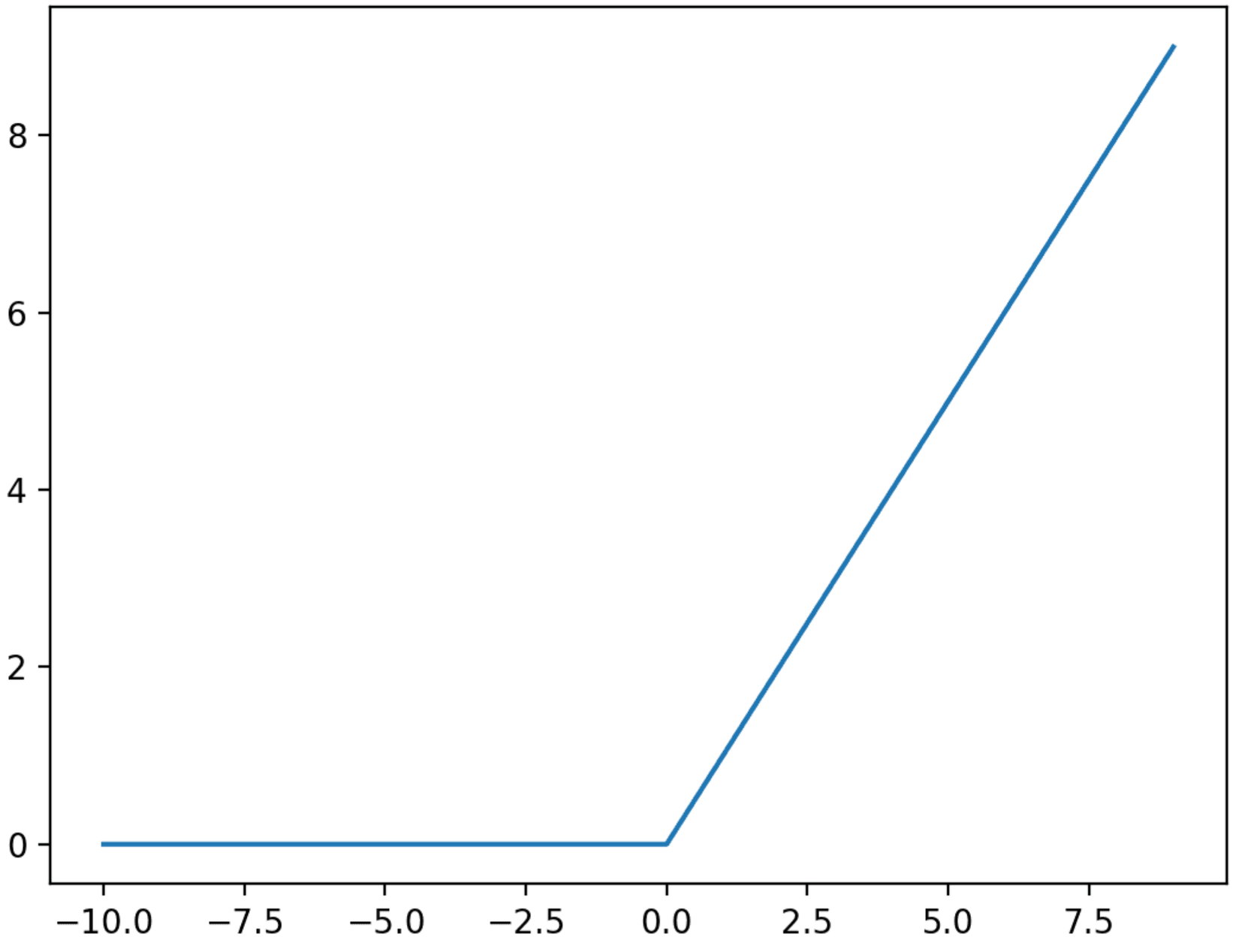
$$A = relu(Z_1)$$

ReLU ,rectified linear unit, is defined as:

$$y = max(0, x)$$

In [326…
```
Image(filename='relu.png')
```

Out[326…



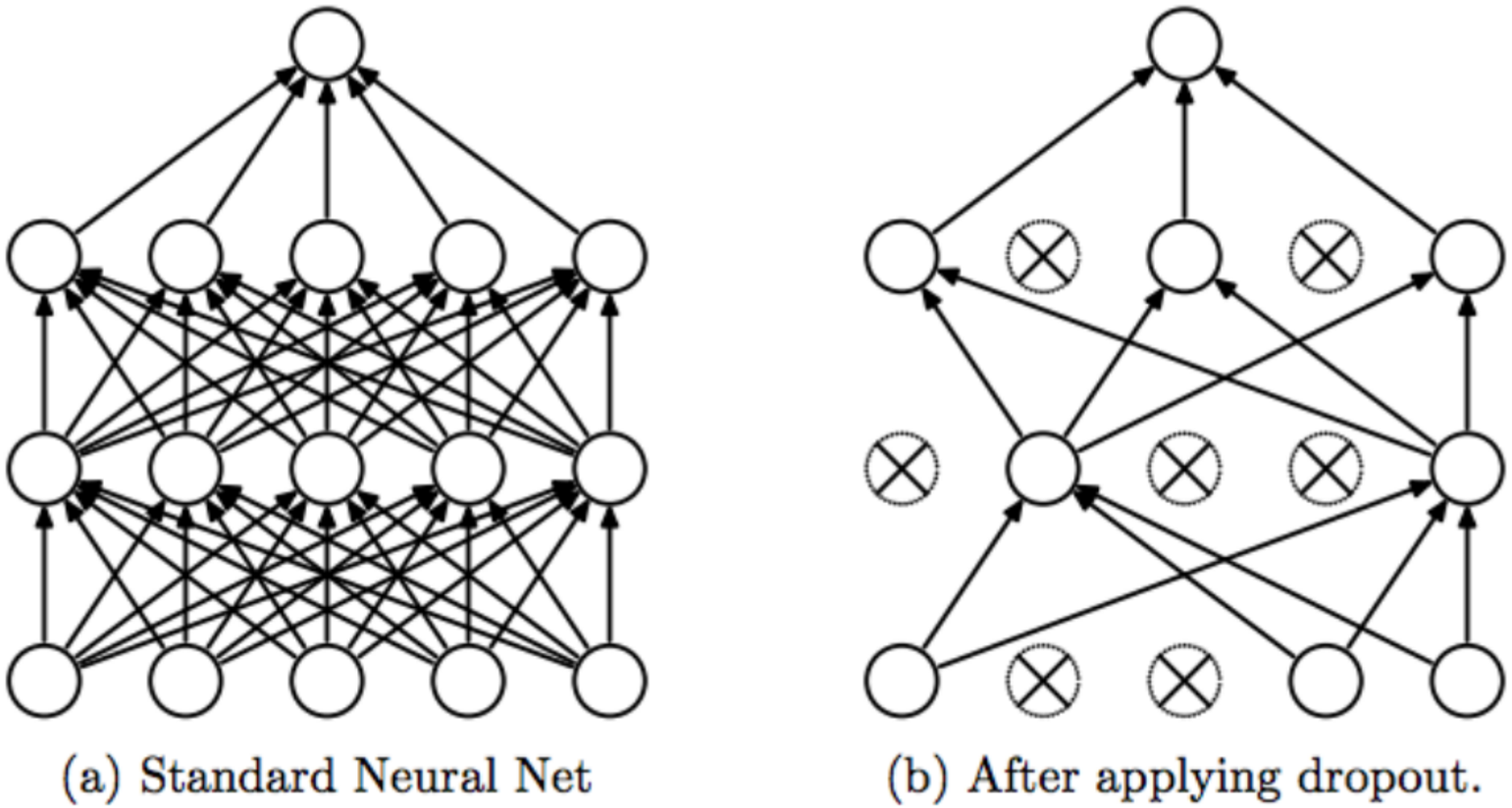
The result is then flattened and passed through a fully connected layer that applies a linear transformation to inputs that are weighted and summed with a bias in order to be classified.

$$Z_2 = w^T * A + b$$

Dropout is a regularization technique that randomly sets neurons to 0 at each epoch of training.

In [328…
```
Image(filename='dropout.png')
```

Out[328…



(a) Standard Neural Net

(b) After applying dropout.

A sigmoid function is then applied to get a binary output.

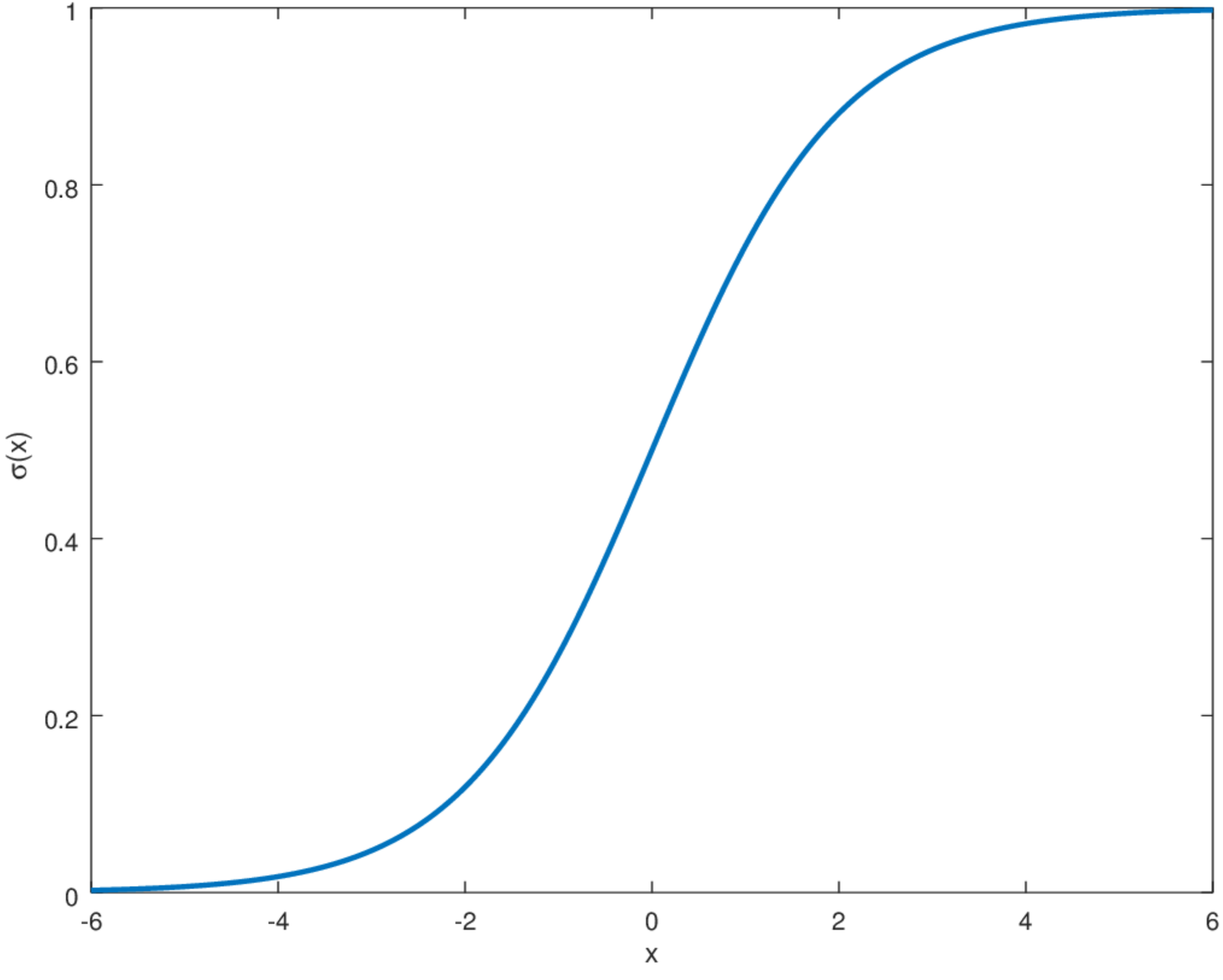
$$O = sigmoid(Z_2)$$

Sigmoid is defined as:

$$y = \frac{1}{1 + e^{-x}}$$

In [327…
```
Image(filename='sigmoid_function.png')
```

Out[327…



The convolutional neural network then uses backward propagation to calculate the error and update the parameters of the network by calculating the partial derivative of the error with respect to the weights using the chain rule.

$$argmin\frac{\partial e}{\partial w} = \frac{\partial e}{\partial O} \cdot \frac{\partial O}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial w}$$

Adaptive Moment Estimation, Adam, computes adaptive learning rates for each parameter by storing an exponentially decaying average of past squared gradients. Adam computes bias-corrected first, m(mean), and second, v(uncentered variance), moment estimates with decay rates, β, that are then used to update the parameters, θ, with an η learning rate.

$$\hat{m} = \frac{m_t}{1 - \beta_{t1}}$$

$$\hat{v} = \frac{v_t}{1 - \beta_{t2}}$$

$$x_j^{new} = x_j^{previous} - \frac{\eta * \hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon}$$

Binary Crossentropy is a loss function, and is the negative log of the log likelihood function. Optimize the function using the partial derivative of the loss function with repect to x, and then iterate toward optimal loss with an η stepsize.

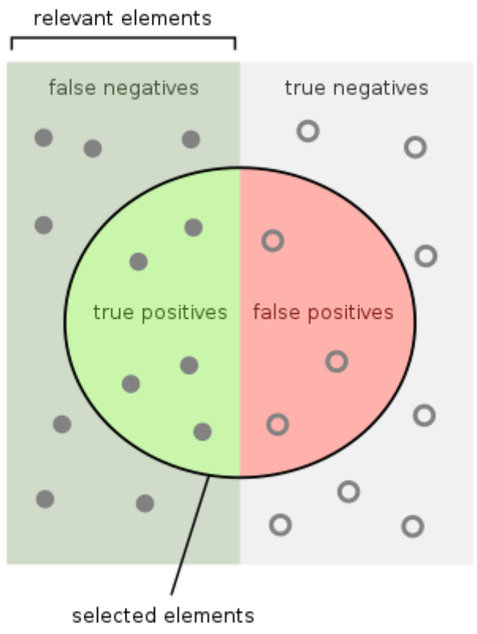
$$BCE = -\frac{1}{N}\sum_i^N y_i * log(x_i) + (1 - y_i) * log(1 - x_i)$$

$$x_j^{new} = x_j^{previous} + \eta * \nabla \frac{\partial BCE(x^{previous})}{\partial x^{previous}}$$

The following metrics will be used to evaluate the model:

In [331…
```
Image(filename='tpfptnfn.png')
```

Out[331…



$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f1 = \frac{2 * precision * recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

In [281…
```
cnn = Sequential()
cnn.add(Conv2D(50, 3, activation='relu', input_shape=[150, 150, 3])),
cnn.add(MaxPooling2D(pool_size=2)),
cnn.add(Flatten()),
cnn.add(Dense(50, activation='relu')),
cnn.add(Dropout(0.2)),
cnn.add(Dense(1, activation='sigmoid'))
```

In [282…
```
cnn.compile(loss="binary_crossentropy", optimizer='adam', metrics=["accuracy", precision, recall, specificity, f1])
```

In [283…
```
history = cnn.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
47/47 [==============================] - 20s 421ms/step - loss: 1.2340 - accuracy: 0.6089 - precision: 0.5271 - recall: 0.5461 - specificity: 0.6002 - f1: 0.4627 - val_loss: 0.6074 - va
l_accuracy: 0.5975 - val_precision: 0.9622 - val_recall: 0.2597 - val_specificity: 0.9840 - val_f1: 0.4004
Epoch 2/10
47/47 [==============================] - 18s 389ms/step - loss: 0.4327 - accuracy: 0.8248 - precision: 0.8187 - recall: 0.8303 - specificity: 0.8087 - f1: 0.8089 - val_loss: 0.4196 - va
l_accuracy: 0.8300 - val_precision: 0.8916 - val_recall: 0.7781 - val_specificity: 0.8854 - val_f1: 0.8274
Epoch 3/10
47/47 [==============================] - 18s 389ms/step - loss: 0.2863 - accuracy: 0.8981 - precision: 0.8899 - recall: 0.9049 - specificity: 0.8906 - f1: 0.8922 - val_loss: 0.3525 - va
l_accuracy: 0.8675 - val_precision: 0.8876 - val_recall: 0.8664 - val_specificity: 0.8663 - val_f1: 0.8748
Epoch 4/10
47/47 [==============================] - 18s 391ms/step - loss: 0.1797 - accuracy: 0.9554 - precision: 0.9434 - recall: 0.9719 - specificity: 0.9393 - f1: 0.9559 - val_loss: 0.3902 - va
l_accuracy: 0.8200 - val_precision: 0.9192 - val_recall: 0.7216 - val_specificity: 0.9286 - val_f1: 0.8049
Epoch 5/10
47/47 [==============================] - 19s 400ms/step - loss: 0.1209 - accuracy: 0.9727 - precision: 0.9685 - recall: 0.9779 - specificity: 0.9689 - f1: 0.9718 - val_loss: 0.3571 - va
l_accuracy: 0.8475 - val_precision: 0.9313 - val_recall: 0.7654 - val_specificity: 0.9326 - val_f1: 0.8382
Epoch 6/10
47/47 [==============================] - 19s 395ms/step - loss: 0.0688 - accuracy: 0.9913 - precision: 0.9900 - recall: 0.9901 - specificity: 0.9915 - f1: 0.9897 - val_loss: 0.3061 - va
l_accuracy: 0.8600 - val_precision: 0.8713 - val_recall: 0.8689 - val_specificity: 0.8422 - val_f1: 0.8669
Epoch 7/10
47/47 [==============================] - 19s 412ms/step - loss: 0.0432 - accuracy: 0.9980 - precision: 0.9964 - recall: 1.0000 - specificity: 0.9956 - f1: 0.9981 - val_loss: 0.3173 - va
l_accuracy: 0.8525 - val_precision: 0.8747 - val_recall: 0.8522 - val_specificity: 0.8462 - val_f1: 0.8595
Epoch 8/10
47/47 [==============================] - 19s 412ms/step - loss: 0.0263 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 1.0000 - f1: 1.0000 - val_loss: 0.3788 - va
l_accuracy: 0.8300 - val_precision: 0.9233 - val_recall: 0.7510 - val_specificity: 0.9207 - val_f1: 0.8242
Epoch 9/10
47/47 [==============================] - 19s 403ms/step - loss: 0.0183 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 1.0000 - f1: 1.0000 - val_loss: 0.3581 - va
l_accuracy: 0.8550 - val_precision: 0.9151 - val_recall: 0.8006 - val_specificity: 0.9124 - val_f1: 0.8515
Epoch 10/10
```

```
47/47 [==============================] - 19s 404ms/step - loss: 0.0117 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 1.0000 - f1: 1.0000 - val_loss: 0.3250 - va
l_accuracy: 0.8675 - val_precision: 0.8883 - val_recall: 0.8644 - val_specificity: 0.8661 - val_f1: 0.8730
```

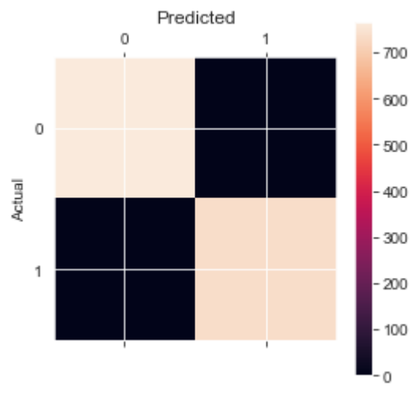
In [160…
```
cnn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 148, 148, 50)      1400
_________________________________________________________________
dropout_2 (Dropout)          (None, 148, 148, 50)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 74, 74, 50)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 273800)            0
_________________________________________________________________
dense_5 (Dense)              (None, 50)                13690050
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 51
=================================================================
Total params: 13,691,501
Trainable params: 13,691,501
Non-trainable params: 0
_________________________________________________________________
```

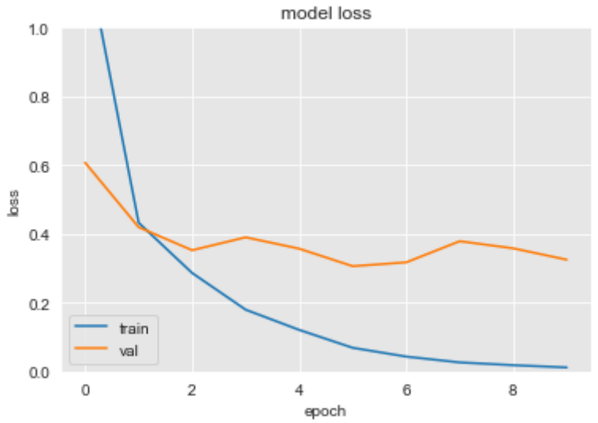
In [277…
```
preds = (cnn.predict(X_train) > 0.5).astype("int32")
```

In [279…
```
con_mat(y_train,preds)
```

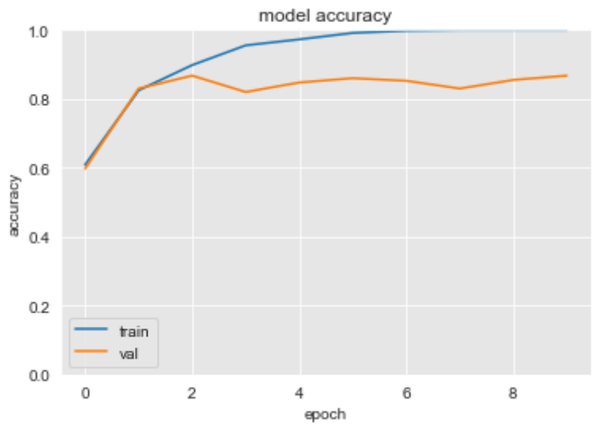
```
Confusion Matrix
----------------
      0    1
0   765    0
1     0  736
```



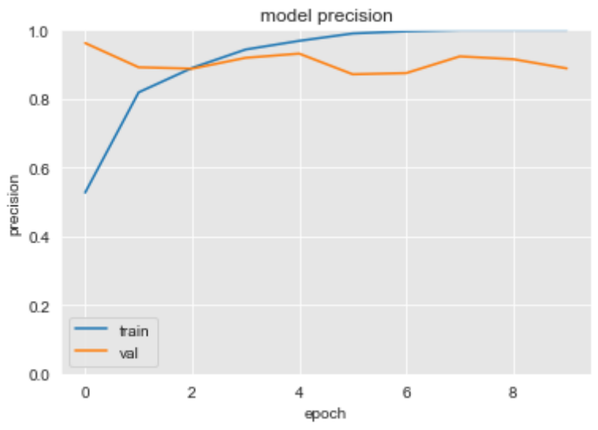
In [284…
```
neural_network_metrics(history.history['loss'],history.history['val_loss'],'model loss','loss')
```



In [285…
```
neural_network_metrics(history.history['accuracy'],history.history['val_accuracy'],'model accuracy','accuracy')
```

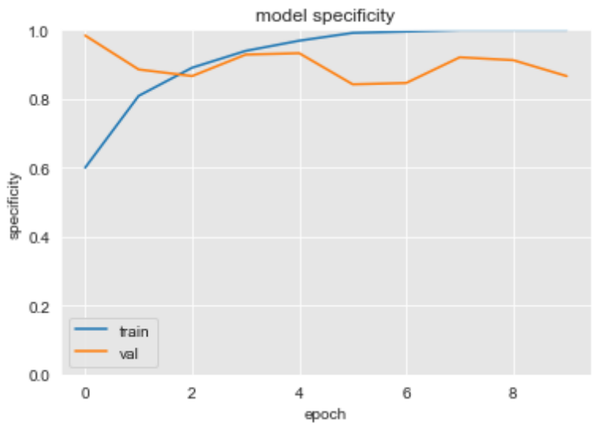


In [286…
```
neural_network_metrics(history.history['precision'],history.history['val_precision'],'model precision','precision')
```

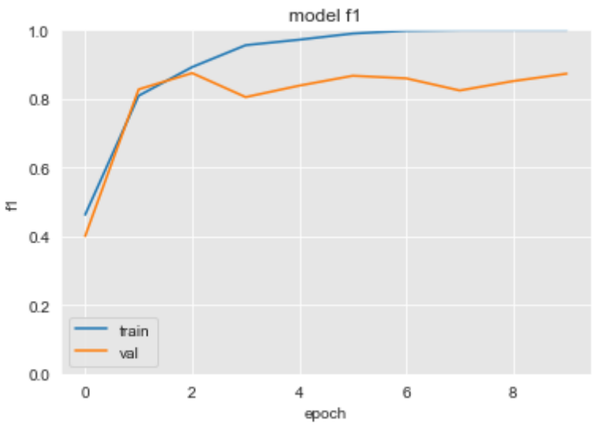


In [287…
```
neural_network_metrics(history.history['recall'],history.history['val_recall'],'model recall','recall')
```

In [288…
```
neural_network_metrics(history.history['specificity'],history.history['val_specificity'],'model specificity','specificity')
```



In [289…
```
neural_network_metrics(history.history['f1'],history.history['val_f1'],'model f1','f1')
```



Calculate the ROC AUC metric.

$$TruePositiveRate = \frac{TP}{TP + FN}$$

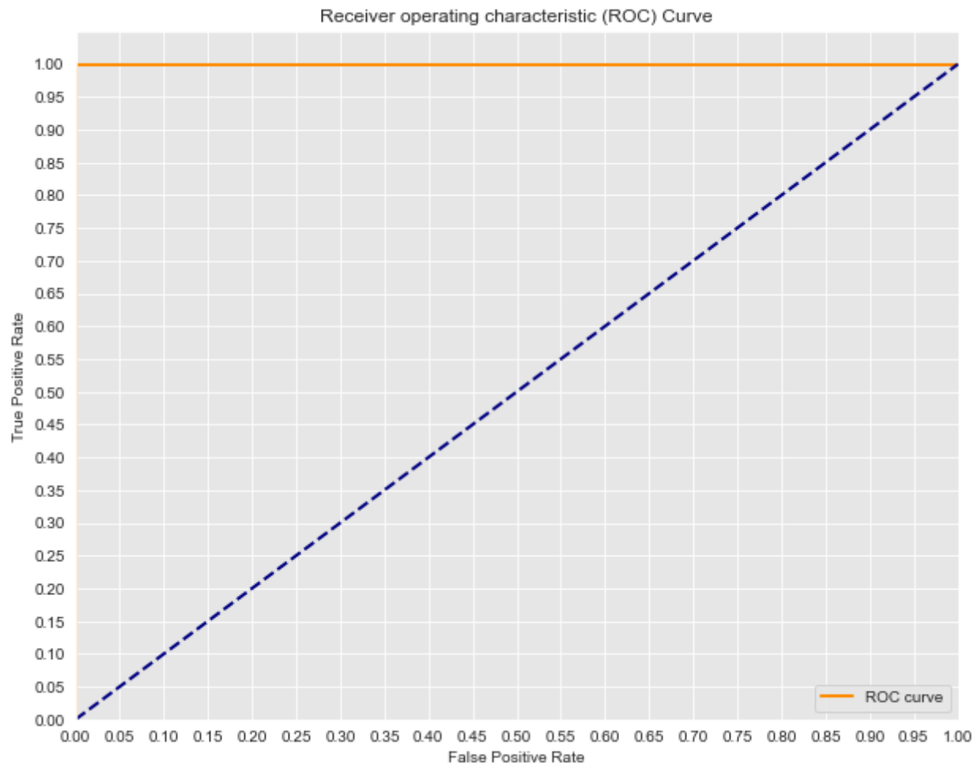
$$FalsePositiveRate = \frac{FP}{FP + TN}$$

$$ReceiverOperatingCharacteristic = \frac{TPR}{FPR}$$

$$AreaUnderCurve = \int_a^b TPR(FPR^{-1}(x))dx$$

In [290…
```
roc(y_train,preds)
```

```
AUC: 1.0
```



Cross validate model.

In [322…
```
# K-fold Cross Validation model evaluation
kfold = KFold(n_splits=5, shuffle=True, random_state=7)
fold_no = 1
cvscores=[]
for train, val in kfold.split(X_train,y_train):

    # create model
    cnn = Sequential()
```

```
    cnn.add(Conv2D(50, 3, activation='relu', input_shape=[150, 150, 3])),
    cnn.add(Dropout(0.2)),
    cnn.add(MaxPooling2D(pool_size=2)),
    cnn.add(Flatten()),
    cnn.add(Dense(50, activation='relu')),
    cnn.add(Dense(1, activation='sigmoid'))
    cnn.compile(loss="binary_crossentropy", optimizer='adam', metrics=["accuracy", precision, recall, specificity, f1])



    # Generate a print
    print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')

    # Fit the model
    cnn.fit(X_train[train], y_train[train], epochs=10, batch_size=20, verbose=1)
    # evaluate the model
    scores = cnn.evaluate(X_train[val], y_train[val], verbose=1)
    cvscores.append(scores[1] * 100)
    fold_no+=1
print(f'Mean Accuracy CV Score:{np.mean(cvscores)}')
```

```
------------------------------------------------------------------------
Training for fold 1 ...
Epoch 1/10
60/60 [==============================] - 15s 250ms/step - loss: 1.6398 - accuracy: 0.5292 - precision: 0.3813 - recall: 0.6063 - specificity: 0.4931 - f1: 0.4355
Epoch 2/10
60/60 [==============================] - 15s 246ms/step - loss: 0.6164 - accuracy: 0.6792 - precision: 0.6809 - recall: 0.7534 - specificity: 0.6205 - f1: 0.6639
Epoch 3/10
60/60 [==============================] - 15s 247ms/step - loss: 0.4663 - accuracy: 0.7950 - precision: 0.7945 - recall: 0.8127 - specificity: 0.7722 - f1: 0.7880
Epoch 4/10
60/60 [==============================] - 15s 250ms/step - loss: 0.3541 - accuracy: 0.8717 - precision: 0.8649 - recall: 0.8816 - specificity: 0.8662 - f1: 0.8630
Epoch 5/10
60/60 [==============================] - 15s 253ms/step - loss: 0.2960 - accuracy: 0.8967 - precision: 0.9038 - recall: 0.9047 - specificity: 0.8990 - f1: 0.8903
Epoch 6/10
60/60 [==============================] - 15s 253ms/step - loss: 0.2039 - accuracy: 0.9375 - precision: 0.9307 - recall: 0.9455 - specificity: 0.9293 - f1: 0.9333
Epoch 7/10
60/60 [==============================] - 15s 249ms/step - loss: 0.1530 - accuracy: 0.9617 - precision: 0.9543 - recall: 0.9729 - specificity: 0.9553 - f1: 0.9611
Epoch 8/10
60/60 [==============================] - 15s 250ms/step - loss: 0.1155 - accuracy: 0.9750 - precision: 0.9673 - recall: 0.9804 - specificity: 0.9684 - f1: 0.9720
Epoch 9/10
60/60 [==============================] - 16s 266ms/step - loss: 0.0819 - accuracy: 0.9875 - precision: 0.9839 - recall: 0.9893 - specificity: 0.9826 - f1: 0.9856
Epoch 10/10
60/60 [==============================] - 16s 269ms/step - loss: 0.0725 - accuracy: 0.9917 - precision: 0.9873 - recall: 0.9964 - specificity: 0.9867 - f1: 0.9914
10/10 [==============================] - 10s 972ms/step - loss: 0.3522 - accuracy: 0.8372 - precision: 0.8270 - recall: 0.8385 - specificity: 0.8435 - f1: 0.8281
------------------------------------------------------------------------
Training for fold 2 ...
Epoch 1/10
61/61 [==============================] - 16s 258ms/step - loss: 1.5615 - accuracy: 0.6295 - precision: 0.4925 - recall: 0.5750 - specificity: 0.6312 - f1: 0.4895
Epoch 2/10
61/61 [==============================] - 21s 347ms/step - loss: 0.4239 - accuracy: 0.8135 - precision: 0.8080 - recall: 0.8143 - specificity: 0.7990 - f1: 0.7924
Epoch 3/10
61/61 [==============================] - 19s 313ms/step - loss: 0.2992 - accuracy: 0.8843 - precision: 0.8803 - recall: 0.8934 - specificity: 0.8647 - f1: 0.8721
Epoch 4/10
61/61 [==============================] - 17s 285ms/step - loss: 0.1476 - accuracy: 0.9559 - precision: 0.9331 - recall: 0.9430 - specificity: 0.9346 - f1: 0.9320
Epoch 5/10
61/61 [==============================] - 17s 272ms/step - loss: 0.1717 - accuracy: 0.9484 - precision: 0.9385 - recall: 0.9449 - specificity: 0.9343 - f1: 0.9342
Epoch 6/10
61/61 [==============================] - 17s 284ms/step - loss: 0.0613 - accuracy: 0.9917 - precision: 0.9910 - recall: 0.9923 - specificity: 0.9730 - f1: 0.9911
Epoch 7/10
61/61 [==============================] - 17s 286ms/step - loss: 0.0322 - accuracy: 0.9983 - precision: 0.9968 - recall: 1.0000 - specificity: 0.9804 - f1: 0.9983
Epoch 8/10
61/61 [==============================] - 18s 295ms/step - loss: 0.0221 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 0.9836 - f1: 1.0000
Epoch 9/10
```

```
61/61 [==============================] - 17s 281ms/step - loss: 0.0138 - accuracy: 1.0000 - precision: 0.9836 - recall: 0.9836 - specificity: 1.0000 - f1: 0.9836
Epoch 10/10
61/61 [==============================] - 12s 193ms/step - loss: 0.0082 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 0.9836 - f1: 1.0000
10/10 [==============================] - 1s 65ms/step - loss: 0.4356 - accuracy: 0.8567 - precision: 0.8657 - recall: 0.8544 - specificity: 0.8575 - f1: 0.8544
------------------------------------------------------------------------
Training for fold 3 ...
Epoch 1/10
61/61 [==============================] - 11s 188ms/step - loss: 1.6803 - accuracy: 0.5945 - precision: 0.4558 - recall: 0.4486 - specificity: 0.7040 - f1: 0.3921
Epoch 2/10
61/61 [==============================] - 11s 186ms/step - loss: 0.4622 - accuracy: 0.7943 - precision: 0.7825 - recall: 0.7652 - specificity: 0.8113 - f1: 0.7482
Epoch 3/10
61/61 [==============================] - 12s 204ms/step - loss: 0.2272 - accuracy: 0.9126 - precision: 0.8949 - recall: 0.9253 - specificity: 0.8774 - f1: 0.9025
Epoch 4/10
61/61 [==============================] - 12s 204ms/step - loss: 0.0798 - accuracy: 0.9833 - precision: 0.9804 - recall: 0.9881 - specificity: 0.9629 - f1: 0.9832
Epoch 5/10
61/61 [==============================] - 12s 190ms/step - loss: 0.0260 - accuracy: 0.9983 - precision: 0.9787 - recall: 0.9836 - specificity: 0.9975 - f1: 0.9809
Epoch 6/10
61/61 [==============================] - 12s 197ms/step - loss: 0.0091 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 0.9836 - f1: 1.0000
Epoch 7/10
61/61 [==============================] - 12s 202ms/step - loss: 0.0045 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 0.9836 - f1: 1.0000
Epoch 8/10
61/61 [==============================] - 13s 211ms/step - loss: 0.0025 - accuracy: 1.0000 - precision: 0.9836 - recall: 0.9836 - specificity: 1.0000 - f1: 0.9836
Epoch 9/10
61/61 [==============================] - 12s 197ms/step - loss: 0.0014 - accuracy: 1.0000 - precision: 0.9836 - recall: 0.9836 - specificity: 1.0000 - f1: 0.9836
Epoch 10/10
61/61 [==============================] - 12s 193ms/step - loss: 0.0013 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - specificity: 0.9836 - f1: 1.0000
10/10 [==============================] - 1s 66ms/step - loss: 0.5520 - accuracy: 0.8400 - precision: 0.8923 - recall: 0.8068 - specificity: 0.9022 - f1: 0.8436
------------------------------------------------------------------------
Training for fold 4 ...
Epoch 1/10
61/61 [==============================] - 11s 183ms/step - loss: 2.2806 - accuracy: 0.5587 - precision: 0.3891 - recall: 0.5415 - specificity: 0.5721 - f1: 0.4133
Epoch 2/10
61/61 [==============================] - 12s 190ms/step - loss: 0.5821 - accuracy: 0.7227 - precision: 0.6978 - recall: 0.6607 - specificity: 0.7760 - f1: 0.6380
Epoch 3/10
61/61 [==============================] - 12s 193ms/step - loss: 0.4730 - accuracy: 0.8027 - precision: 0.7739 - recall: 0.8501 - specificity: 0.7311 - f1: 0.7971
Epoch 4/10
61/61 [==============================] - 12s 200ms/step - loss: 0.3111 - accuracy: 0.8776 - precision: 0.8580 - recall: 0.8738 - specificity: 0.8589 - f1: 0.8568
Epoch 5/10
61/61 [==============================] - 11s 181ms/step - loss: 0.2168 - accuracy: 0.9317 - precision: 0.9320 - recall: 0.9268 - specificity: 0.9146 - f1: 0.9229
Epoch 6/10
61/61 [==============================] - 12s 203ms/step - loss: 0.1712 - accuracy: 0.9450 - precision: 0.9521 - recall: 0.9484 - specificity: 0.9264 - f1: 0.9447
Epoch 7/10
61/61 [==============================] - 12s 199ms/step - loss: 0.0887 - accuracy: 0.9842 - precision: 0.9789 - recall: 0.9902 - specificity: 0.9608 - f1: 0.9837
Epoch 8/10
61/61 [==============================] - 13s 211ms/step - loss: 0.0503 - accuracy: 0.9967 - precision: 0.9952 - recall: 0.9967 - specificity: 0.9789 - f1: 0.9956
Epoch 9/10
61/61 [==============================] - 13s 212ms/step - loss: 0.0319 - accuracy: 0.9983 - precision: 0.9801 - recall: 0.9836 - specificity: 0.9971 - f1: 0.9818
Epoch 10/10
61/61 [==============================] - 13s 206ms/step - loss: 0.0350 - accuracy: 0.9992 - precision: 0.9836 - recall: 0.9820 - specificity: 1.0000 - f1: 0.9827
10/10 [==============================] - 1s 62ms/step - loss: 0.3279 - accuracy: 0.8500 - precision: 0.8946 - recall: 0.8116 - specificity: 0.9140 - f1: 0.8466
------------------------------------------------------------------------
Training for fold 5 ...
Epoch 1/10
61/61 [==============================] - 12s 194ms/step - loss: 2.7381 - accuracy: 0.5720 - precision: 0.4152 - recall: 0.5902 - specificity: 0.5727 - f1: 0.4557
Epoch 2/10
61/61 [==============================] - 12s 198ms/step - loss: 0.5722 - accuracy: 0.7244 - precision: 0.6924 - recall: 0.6689 - specificity: 0.7517 - f1: 0.6343
Epoch 3/10
61/61 [==============================] - 12s 196ms/step - loss: 0.4560 - accuracy: 0.7993 - precision: 0.8209 - recall: 0.8228 - specificity: 0.7583 - f1: 0.7886
Epoch 4/10
61/61 [==============================] - 11s 187ms/step - loss: 0.3210 - accuracy: 0.8859 - precision: 0.8662 - recall: 0.8813 - specificity: 0.8815 - f1: 0.8624
Epoch 5/10
61/61 [==============================] - 11s 185ms/step - loss: 0.2186 - accuracy: 0.9292 - precision: 0.9206 - recall: 0.9332 - specificity: 0.9091 - f1: 0.9212
Epoch 6/10
61/61 [==============================] - 13s 207ms/step - loss: 0.1431 - accuracy: 0.9642 - precision: 0.9614 - recall: 0.9698 - specificity: 0.9400 - f1: 0.9630
Epoch 7/10
```

```
61/61 [==============================] - 12s 194ms/step - loss: 0.0934 - accuracy: 0.9784 - precision: 0.9594 - recall: 0.9665 - specificity: 0.9769 - f1: 0.9611
Epoch 8/10
61/61 [==============================] - 12s 194ms/step - loss: 0.0580 - accuracy: 0.9917 - precision: 0.9671 - recall: 0.9822 - specificity: 0.9867 - f1: 0.9740
Epoch 9/10
61/61 [==============================] - 12s 190ms/step - loss: 0.0371 - accuracy: 0.9967 - precision: 0.9806 - recall: 0.9788 - specificity: 0.9967 - f1: 0.9794
Epoch 10/10
61/61 [==============================] - 12s 189ms/step - loss: 0.0212 - accuracy: 1.0000 - precision: 0.9836 - recall: 0.9836 - specificity: 1.0000 - f1: 0.9836
10/10 [==============================] - 1s 63ms/step - loss: 0.3995 - accuracy: 0.8200 - precision: 0.8820 - recall: 0.7270 - specificity: 0.9160 - f1: 0.7950
Mean Accuracy CV Score:84.0775191783905
```

Test model.

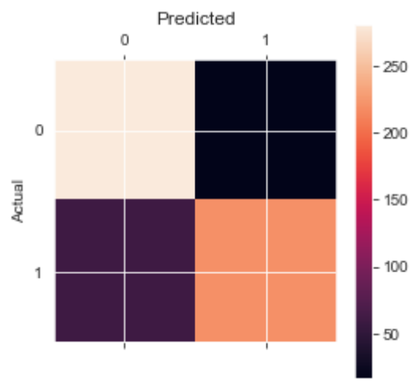
In [124…
```
scores = cnn.evaluate(X_test, y_test, batch_size=20, verbose=1)
```

```
29/29 [==============================] - 1s 35ms/step - loss: 0.3587 - accuracy: 0.8655 - precision: 0.9211 - recall: 0.7754 - specificity: 0.9466 - f1: 0.8363
```

In [358…
```
preds = (cnn.predict(X_test) > 0.5).astype("int32")
```

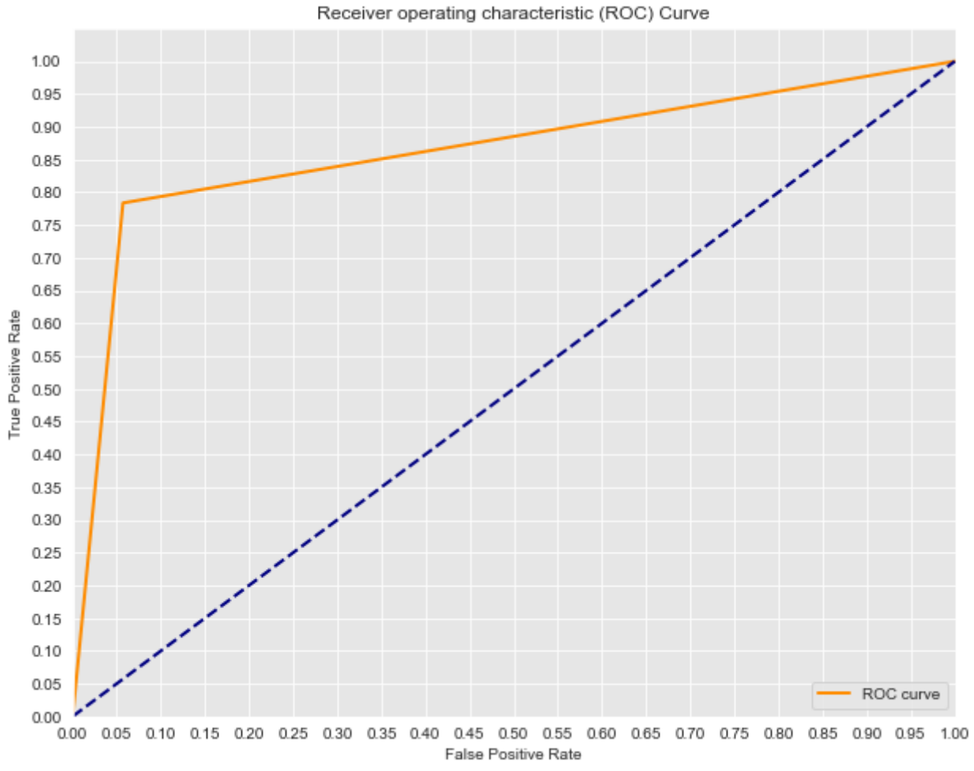
In [359…
```
con_mat(y_test,preds)
```

```
Confusion Matrix
----------------
     0    1
0  281   17
1   61  221
```



In [360…
```
roc(y_test,preds)
```

```
AUC: 0.8633204816983198
```



Import data of sars-cov-2 genome.

In [136…
```
df=Json('sars-cov-2_genome')
df.file("genome.fna")
genome=json_storage['sars-cov-2_genome']
genome
```

Out[136…
```
'\nACUUUGUGUCUACUCCUCUCAACUAAACGAAAUUUUUCUAGUGCUGUCAUUUGUUAUGGCAGUCCUAGUGUAAUUGAAAUUUCGUCAAGUUUGUAAACUGGUUAGGCAAGUGUUGUAUUUUCUGUGUUUAAGCACUGGUGGUUCUGUCCACUAGUGCACACAUUGAUACUUAAGUGGUGUUC
UGUCACUGCUUAUUGUGGAAGCAACGUUCUGUCGUUGUGGAAACCAAUAACUGCUAACCAUGUUUUACAAUCAAGUGACACUUGCUGUUGCAAGUGAUUCGGAAAUUUCAGGUUUUGGUUUUGCCAUUCCUUCUGUAGCCGUUCGCGCUUAUAGCGAAGCCGCUGCACAAGGUUUUCAGGCAUGC
CGCUUUGUUGCUUUUGGCUUACAGGAUUGUGUAACCGGUAUUAAUGAUGACGAUUAUGUCAUUGCAUUGACUGGUACUAAUCAGCUUUGUGCCAAAAUUUUACUUUUUUCUGAUAGACCUCUUAAUUUGCGAGGUUGGCUCAUUUUUUCUAACAGCAAUUAUGUUCUUCAGGACUUUGAUGUUGU
UUUUGGCCAUGGUGCAGGAAGUGUGGUUUUUGUGGAUAAGUAUAUGUGUGGUUUUGAUGGUAAACCUGUGUUACCUAAAAACAUGUGGGAAUUUAGAGAUUACUUUAAUGAUAAUACUGAUAGUAUUGUUAUUGAUGGUGUCACUUAUCAAUUAGCAUGGGAUGUUAUACGUAAAGACCUUUCUU
AUGAACAGCAAAAUGUUUUAGCUAUUGAGAGCAUUCAUUAUCUUGGUACUACAGGUCAUACUUUGAAGUCUGGUUGCAAACUCAUUAAUGCCAAGCCGCCUAAAUAUUCUUCUAAGGUUGUUUUGAGUGGUGAAUGGAAUGCUGUGUAUAAGGCGUUUGGUUCACCAUUUAUUACAAAUGGUAUA
UCAUUGCUAGAUAUAAUUGUUAAACCAGUUUUCUUUAAUGCUUUUGUUAAAUGCAAUUGUGGUUCUGAGAAUUGGAGUGUUGGUGCAUGGGAUGGUUAUCUAUCUUCUUGUUGUGGCACACCUGCUAAGAAACUUUGUGUUGUUCCUGGUAAUGUUGUUCCUGGUGAUGUGAUCAUCACCUCAAC
UGAUGCUGGUUGUGGUGUUAAAUACUAUGCUGGCUUAGUUGUUAAACAUAUUACUAACAUUACUGGUGUGUCUUUAUGGCGUGUUACAGCUGUUCAUUCUGAUGGAAUGUUUGUGGCAACAUCUUCUUAUGAUGCACUCUUGCAUAGAAAUUCAUUAGACCCUUUUUGCUUUGAUGUUAACACUU
UACUUUCUAAUCAAUUACGUCUAGCUUUUCUUGGUGCUUCUGUUACAGAAGAUGUUAAAUUUGCUGCUAGCACUGGUGUUAUUGACAUUAGUGCUGGUAUGUUUGGUCUUUACGAUGACAUAUUGACAAACAAUAAACCUUGGUUUGUACGCAAAGCUUCUGGGCUUUUUGAUGCAAUCUGGGAU
GCUUUUGUUGCCGCUAUUAAGCUUGUGCCAACUACUACUGGUGGUUUGGUUAGGUUUGUUAAGUCUAUCGCUUCAACUGUUUUAACUGUUUCUAAUGGUGUUAUUAUUAUGUGUGCAGAUGUUCCAGAUGCUUUUCAACCAGUUUACCGUACAUUUACACAAGCCAUUUGUGCUGCAUUUGAUUU
UUCUUUAGAUGUAUUUAAAAUUGGUGAUGUUAAAUUUAAACGACUUGGUGAUUAUGUUCUUACUGAAAAUGCUCUUGUUCGUUUGACUACUGAAGUUGUUCGUGGUGUUCGUGAUGCUCGCAUAAAGAAAGCCAUGUUUACUAAAGUAGUUGUAGGUCCUACAACUGAAGUUAAGUUUUCUGUUA
UUGAACUUGCCACUGUUAAUUUGCGUCUUGUUGAUUGUGCACCUGUAGUUUGCCCUAAAGGUAAAAUUGUUGUUAUUGCUGGACAAGCUUUUUUCUAUAGUGAUGGUUUUUAUCGUUUUAUGGUUGAUUCUACAACUGUAUUAAAUGACCCUGUUUUUACUGGUGAUUUAUUUUACACUAUUAAG
UUUAGUGGUUUUAAGCUUGAUGGUUUUAACCAUCAGUUUGUUAAUGCUAGUUCUGCUACAGAUGCCAUUAUUGCUGUUGAGCUGUUGUUUAUCGGAUUUUAAAACUGCAGUUUUUGUGUACACAUGUGUGGUUGAUGGUUGUAGUGUCAUUGUUAGACGUGAUGCUACAUUCGCCACACAUGUGUG
UUUUAAGGACUGUUAUAAUAUUUGGGAGCAAUUCUGCAUUGAUAAUUGUGGUGAGCCAUGGUUUUUGACUGAUUAUAAUGCUAUCUUGCAGAGUAAUAACCCUCAAUGUGCUAUUGUUCAAGCAUCGGAGUCUAAAGUUUUGCUUGAGAGGUUUUUACCUAAGUGUCCUGAAAUACUGUUGAGUA
UUGAUGAUGGCCAUUUAUGGAAUCUUUUUGUUGAAAAGUUUAAUUUUGUUACAGAUUGGUUAAAAACUCUUAAGCUUACACUUACUUCUAAUGGUCUUUUAGGUAAUUGUGCCAAACGUUUUAGACGUGUUUUGGUAAAAUUGCUUGAUGUCUAUAAUGGUUUUCUUGAAACUGUCUGUAGUGUC
GUACACACUGCUGGUGUUUGCAUUAAAUAUUAUGCUGUUAAUGUUCCAUAUGUAGUUAUUAGUGGUUUUGUAAGUCGUGUAAUUCGUAGAGAAAGGUGUGACGUGACUUUUCCUUGUGUUAGUUGUGUCACUUUUUUCUAUGAAUUUUUAGACACGUGUUUUAGUGUUAGUAAACCUAAUGCCAU
UGAUGUUGAACAUUUAGAGCUUAAAGAAACCGUUUUUGUUGAACCUAAGGAUGGUGGUCAAUUUUUUGUUUCUGAUGAUUAUCUUUGGUAUGUUGUAGAUGACAUUUAUUAUCCAGCUUCAUGUAAUGGUGUAUUGCCAGUUGCUUUUACAAAAUUGGCAGGUGGUAAAAUAUCUUUUUCUGAUG
AUGUUAUAGUCCAUGAUGUUGAACCUACCCAUAAAGUCAAGCUCAUAUUUGAGUUUGAAGAUGAUGUUGUUACCAGUCUUUGUAAGAAGAGUUUUGGUAAGUCUAUUAUUUAUACAGGUGAUUGGGAAGGUUUACAUGAAGUUCUUACAUCUGCAAUGAAUGUCAUUGGGCAACAUAUUAAGUUG
CCACAAUUUUAUAUUUAUGAUGAAGAGGGUGGUUAUGAUGUUUCUAAACCAGUUAUGAUUUCACAAUGGCCUAUUAGUGAUGAUAGUGGUGAUGUUGUUGAAGCGAGCACUGAUUUUCAUCAAUUAGAAUUUGUUAGAGAAGAGGUUGAUAUAAUUGAACAACCUUUUGGGGAAGUUGAACA
UUCGCUCCCAAUUAAACAACCUUUUUCUUUUUCUUUUAGAGAUGAAUUGGGUGUUCGUGUUUUAGAUCAAUCUGAUAAUAAUUGUUGGAUUAGUACCACACUUAUACAGUUGCAACUUACAAAGCUUUUGGAUGAUUCUAUUGAGAUGCAAUUGUUUAAAGUUGGUAAAGUUGAUUCAAUUGUUC
AAAAGUGUUAUGAGUUGUCUCAUUUAAUUAGUGGUUCACUUGGUGAUAGUGGUAAACUUCUUAGUGAACUUCUUAAAGAUAAAUAUACAUGUUCUAUAACUUUUGAGAUGUCUUGUGAUUGUGGUAAAAAGUUUGAUGAGCAAGUUGGUUGUUUGUUUUGGAUUAUGCCUUACACAAAACUUUUU
CAAAAAGGUGAGUGUUGUAUUUGUCAUAAAAUGCAGACUUAUAAGCUUGUUAGUAUGAAAGGUACUGGUGUGUUUGUACAGGAUCCAGCACCUAUUGACAUUGAUGCUUUCCCCUGUGAGACCUAUAUGUUCAUCUGUAUAUUUAGGUGUUAAGGGUUCUGGUCAUUAUCAAACAAAUUUAUACAG
UUUUGACAAAGCUAUUGAUGGUUUUGGUGUCUUUGACAUUAAAAAUAGUAGUGUUAAUACUGUCUGUUUUGUUGAUGUUGAUUUUCAUAGUGUAGAAAUAGAAGCUGGUGAAGUUAAACCUUUUGCUGUAUAUAAAAAUGUUAAAUUUUAUUUAGGUGAUAUUUCACACCUUGUAAACUGUGUUU
CUUUUGACUUUGUUGUCAAUGCUGCUAAUGAAAAUCUCAUGCAUGGAGGCGGUGUUGCACGUGCUAUUGAUAUUUUGACUGAAGGUCAACUUCAGUCAUUAUCUAAAGAUUACAUUAGUAGUAAUGGUCCACUUAAGGUUGGAGCAGGUGUUAUGUUGGAGUGUGAAAAAUUCAAUGUAUUUAAU
GUUGUUGGUCCGCGAACUGGUAAACAUGAGCAUUCAUUACUUGUUGAAGCUUAUAAUUCUAUUUUAUUUGAAAAUGGUAUUCCACUUAUGCCUCUUCUUAGUUGUGGUAUUUUUGGUGUAAGGAUUGAAAAUUCUCUUAAAGCUUUGUUUAGUUGUGACAUUAAUAAACCGUUGCAAGUUUUUGU
```

UUAUUCUUCAAAUGAAGAACAAGCUGUUCUUAAGUUUUUAGAUGGUUUAGAUUUAACACCAGUCAUUGAUGAUGUUGAUGUUGUUAAACCUUUUAGAGUUGAAGGUAAUUUUUCAUUCUUUGAUUGUGGUGUCAAUGCCUUGGAUGGUGAUAUUUACUUAUUAUUUACUAACUCUAUUUUAAUGU
UGGAUAAACAAGGACAAUUAUUGGACACAAAACUUAAUGGUAUUUUGCAACAGGCAGCUCUUGAUUAUCUUGCUACAGUUAAAACUGUACCAGCUGGUAAUUUGGUUAAACUUGUUGUGAGAGUUGUACCAUUUAUGUGUGUUGUACCAUCGAUAAAUGAUCUUCUUUGAUAAAAUCUU
GGUCGUUGUGUGCGUAAACUUAAUAGAUUGAAAACUUGUGUUAUUGCCAAUGUUCCUGCUAUUGAUGUUUUGAAAAAGCUUCUUUCAAGUUUGACUUUAACUGUUAAAUUUGUUGUAGAGAGUAAUGUUAUGGAUGUUAACGACUGUUUUAAGAAUGAUAAUGUAGUUUUGAAAAUCACUGAAGA
UGGUAUUAAUGUUAAAGAUGUUGUUGUUGAGUCUUCUAAGUCACUUGGUAAACAAUUGGGUGUUGUGAGUGAUGGUGUUGACUCUUUUGAAGGUGUUUUACCUAUUAAUACUGAUACUGUCUUAUCUGUAGCUCCAGAAGUUGACUGGGUUGCUUUUUACGGUUUUGAAAAGGCAGCACUUUUUG
CUUCUUUGGAUGUAAAGCCAUAUGGUUACCCUAAUGAUUUUGUUGGUGGUUUUAGAGUUCUUGGGACCACCGACAAUAAUUGUUGGGUUAAUGCAACUUGUAUAAUUUUACAGUAUCUUAAGCCUACUUUUAAAUCUAAGGGUUUAAAUGUUCUUUGGAAUAAAUUUGUUACAGGUGAUGUUGGA
CCUUUUGUUAGUUUUAUUUAUUUUAUAACUAUGUCUUCAAAGGGUCAAAAGGGUGAUGCUGAAGAGGCAUUAUCUAAAUUGUCAGAGUAUUUGAUUUAGUGAUUCUAUGUUACUCUUGAACAAUAUUCAACUUGUGACAUUUGUUGUAAAGUACUGUAGUGAAGUUAAAAGUGCUAUUGUCUGUGC
UAGUGUGCUUAAAGAUGGUUGUGAUGUUGGUUUUUGUCCACACAGACAUAAAUUGCGUUCACGUGUUAAGUUUGUUAAUGGACGUGUUGUUAUUACCAAUGUUGGUGAACCUAUAAUUUCACAACCUUCUAAGUUGCUUAAUGGUAUUGCUUAUACAACAUUUUCAGGUUCUUUUGAUAACGGUC
ACUAUGUAGUUUAUGAUGCUGCUAAUAAUGCUGUCUAUGAUGGUGCUCGUUUAUUUGCUUCAGAUUUGUCUACUUUAGCUGUUACAGCUAUUGUUGUAGUAGGUGGUUGUGUAACAUCUAAUGUUCCAUCAAUUGUUAGUGAGAAAAUUUCUGUUAUGGAUAAACUUGAUACUGGUGCACAAAAA
UUUUUCCAAUUUGGUGAUUUUGUUAUGAAUAACAUUGUUCUGUUUUUAACUUGGUUGCUUAGUAUGUUUAGUCUUUUACGUACUUCUAUUAUGAAGCAUGAUAUUAAAGUUAUUGCCAAGGCUCCUAAACGUACAGGUGUUAUUUUGACACGUAGUUUUAAGUAUAACAUUAGAUCUGCUUUGUU
UGUUAUAAAGCAGAAGUGGUGUGUUAUUGUUACUUUGUUUAAGUUCUUAUUGUUAUUAUAUGCUAUUUAUGCACUUGUUUUUAUGAUUGUGCAAUUUAGUCCUUUUAAUAGUCUUUUAUGUGGUGACAUUGUAAGUGGUUAUGAAAAAUCCACUUUUAAUAAGGAUAUUUAUUGUGGUAAUUCUA
UGGUUUGUAAGAUGUGUUUGUUCAGUUAUCAAGAGUUUAAUGAUUUGGAUCAUACUAGUCUUGUUUGGAAGCACAUUCGUGAUCCUAUAUUAAUCAGUUUACAACCAUUUGUUAUACUUGUUAAUUUGUUAAUUUUGGUAAUAUGUAUUUGCGUUUUGGACUUUUAUAUUUGUUGCACAAUUU
AUUAGUACUUUUGGUUCUUUCUUAGGCUUUCAUCAGAAACAGUGGUUUUUACAUUUUGUGCCGUUUGAUGUUUUAUGUAAUGAGUUUUUAGCUACAUUUAUUGUCUGCAAAAUUGUUUUAUUUGUUAGACAUAUUAUUGUUGGCUGUAAUAAUGCUGACUGUGUAGCUUGUUCUAAAAGUGCUAG
ACUUAAACGUGUACCACUUCAAACUAUUAUUAAUGGUAUGCAUAAAUCAUUCUAUGUUAAUGCUAAUGGUGGUACUUGUUUCUGUAAUAAACAUAACUUCUUUUGUGUUAAUUGUGAUUCUUUUGGGCCUGGUAACACUUUUAUUAAUGGUGAUAUUGCAAGAGAGCUUGGUAAUGUUGUUAAAA
CAGCUGUUCAACCCACAGCUCCUGCAUAUAUUAUUAUUGAUAAGGUAGAUUUUGUUAAUGGAUUUUAUCGUCUUUUAUAGUGGUGACACUUUUUGGCGGUAUGACUUUGACAUUACUGAAUCUAAGUAUAGUUGUAAAGAGGUUCUGAAGAAUUGUAAUGUUUUAGAAAAUUUUAUUGUUUACAAU
AAUAGUGGUAGUAACAUUACACAGAUUAAAAAUGCUUGUGUUUAUUUUUCUCAAUUGUUGUGUGAACCUAUAAAGUUGGUAAAUUCAGAGUUGUUGUCAACUUUAUCAGUUGAUUUUAAUGGUGUUUUGCAUAAGGCAUAUGUUGAUGUUGUAAUAGUUUUUUAAGGAGCUAACUGCUAA
CAUGUCCAUGGCUGAAUGUAAAGCUACACUUGGUUUGACUGUUUCUGAUGAUGAUUUUGUUUCAGCUGUUGCCAAUGCACAUAGGUAUGACGUUUUGCUUUCAGAUUUGUCAUUUAAUAAUUUUUUAUUUCUUAUGCUAAACCUGAAGAUAAGUUGUCCGUUUAUGAUAUUGCUUGUUGAUGC
GUGCCGGUUCUAAGGUUGUUAACCAUAAUGUUUUAAUUAAAGAGUCAAUACCUAUUGUUUGGGGUGUCAAGGACUUUAAUACUCUUUCUCAAGAAGGUAAGAAGUACCUUGUUAAAACAACUAAAGCAAAGGGUUUGACUUUUUUAUUAACUUUUAAUGAUAACCAAGCAAUUACACAAGUUCCU
GCUACUAGUAUAGUGCAAAACAGGGUGCUGGUUUUAAACGUACUUAUAAUUUUCUGUGGUAUGUAUGUUUAUUUGUUGUUGCAUUGUUUAUUGGUGUCUCAUUUAUUGAUUACACAACCACUGUAACUAGCUUUCAUGGUUAUGAUUUUAAGUACAUUGAGAAUGGUCAGUUGAAGGUGUUUGA
AGCACCUUUACACUGUGUUCGUAAUGUUUUGAUAAUUUUAAUCAAUGGCAUGAGGCUAAGUUUGGUGUUGUUACUACUAAUAGUGAUAAAUGUCCUAUAGUUGUUGGUGUUUCAGAGCGUAUUAAUGUUGUUCCUGGUGUUCCAACAAAUGUAUAUUUGGUAGGAAAGACUCUUGUUUUUACAU
UACAGGCUGCUUUUGGAAACACAGGUGUUUGUUAUGACUUUGAUGGUGUUACCACUAGUGAUAAGUGAUUUUUAAUUCUGCUUGUACUAGGUUGGAAGGUUUGGUGGUGACAAGUUUAUUGUUACAACACUGAUCUUAUUGAAGGUUCUAAACCUUAUAGUACUUUACAGCCCAAUGCUUAU
UAUAAGUAUGAUGUUAAAAAUUAUGUACGUUUUCCAGAAAUUUUAGCUAGAGGUUUUGGCUUACGUACUAUUAGAACUUUGGCUACACGUUAUUGUAGAGUUGGUGAAUGCCGUGACUCACAUAAAGGUGUUUGUUUUGGUUUUGAUAAAUGGUAUGUUAAUGAUGGACGUGUUGAUGAUGGUUA
CAUUUGUGGUGAUGGUCUUAUAGACCUUCUUGUUAAUGUACUCUCAAUCUUUAGUUCAUCUUUUAGCGUUGUGGCUAUGUCUGGACAUAUGUUGUUUAACUUUCUUUUUGCAGCAUUUAUUACAUUUUUGUGCUUUUUAGUUACUAAAUUUAAACGUGUUUUUGGUGAUCUUUCUUAUGGUGUUU
UUACUGUUGUUUGUGCAACUUUGAUUAAUAACAUUUCUUAUGUUGUUACUCAAAAUUUAUUUUUAUGUUGCUUUAUGCUAUUUUGUAUUUUGUUUUUACUAGGACAGUGCGUUAUGCUUGGAUUUGGCAUAUUGCAUACAUUGUUGCAUACUUCUUGUUAAUACCAUGGUGGCUUCUCACAUGG
UUUAGUUUUGCUGCAUUUUUAGAGCUUUUACCUAAUGUUUUUAAGUUAAAAAUCUCUACUCAAUUGUUUGAAGGUGAUAAGUUUAUAGGUACUUUUGAGAGUGCUGCUGCAGGUACAUUUGUUCUUGACAUGCGUUCUUAUGAAAGGCUGAUAAAUACUAUUUCACCUGAGAAACUUAAGAUUA
UGCUGCAAGUUAUAAUAAAUAUAAAUAUUAUAGUGGUAGUGCUAGUGAGGCUGAUUAUCGUUGUGCUUGUUAUGCUCAUUUAGCCAAGGCUAUGUUAGAUUACGCAAAAGAUCAUAAUGACAUGUUAUAUUCUCCACCUACCAUUAGCUACAAUUCCACCUUACAAUCUGGUCUUAAGAAGAUGG
CACAACCAUCUGGUUGUGUUGAGAGAUGUGUGGUUCGCGUCUGUUAUGGUAGUACUGUGCUUAAUGGAGUUUGGUUAGGUGACACUGUUACUUGUCCUAGACAUGUCAUAGCACCAUCAACCACUGUUCUUAUUGAUUAUGAUCAUGCAUAUAGUACUAUGCGUUUGCAUAAUUUUUCAGUGUCU
CAUAAUGGGUGUCUUCUUGGGAGUUGUUGGUGUUACAAUGCAUGGUUCUGUGUUGCGUAUUAAGGUUUCACAAUCUAAUGUACAUACACCUAAACAUGUUUUUAAAACGUUGAAACCUGGUGAUUCUUUUAAUAUUUUAGCAUGUUAUGAAGGUAUUGCAUCUGGUGUUUUGGUGUUAAUUUACG
UACAAACUUUACUAUUAAAGGUUCUUUUAUAAAUGGAGCUUGUGGUUCUCCUGGUUAUAAUGUUAGAAAUGAUGGUACUGUUGAGUUUUGUUAUUUACACCAAAUUGAGUUAGGUAGUGGUGCUCAUGUUGGUUCUGAUUUUACUGGUAGUGUUUAUGGUAAUUUUGAUGACCAACCUAGUUUGC
AAGUUGAGAGUGCCAACCUUAUGCUAUCAGAUAAUGUUGUGCCUUUUUGUAUGCUGCUUUGUUGAAUGGUUGUAGGUGGUGGUUGCGUUCAACUAGAGUUAAUGUUGAUGGUUUUAAUGAAUGGGCUAUGGCUAAUGGUUAUACAAGUGUUUCUAGUGUUGAGUGCUAUCUUAUUUUGGCAGCA
AAAACUGGUGUUAGUGUUGAACAAUUGUUAGCUUCCAUUCAACAUCUUCAUGAAGGUUUUGGUGGUAAAAACAUACUUGGUUAUUCUAGUUUAUGUGAUGAGUUCACACUAGCUGAAGUUGUGAAGCAGAUGUAUGGUGUUAACUUGCAAAGUGGUAAGGUUAUUUUGGUUUAAAAACAAUGUU
UUUAUUUAGCGUUUUCUUCACAAUGUUUUGGGCAGAACUCUUUAUUUAUACAAACACUAUAUGGAUAAACCCUGUGAUACUUACACCUAUAUUUUGUCUACUUUUGUUUUUGUCAUUAGUUUUAACUAUGUUUCUUAAACAUAAGUUUUUGUUUUUGCAAGUAUUUUAUUACCUACUGUUAUUG
CAACUGCUUUAUAUAAUUGUGUUUUGGAUUAUUACAUAGUAAAAUUUUUGGCUGACCAUUUUAACUAUAAUGUUUCAGUAUUACAAAUGGAUGUUCAGGGUUUAGUUAAUGUUUUGGUUUGUUUAUUUGUUGUAUUUUUACACACAUGGCGCUUUUCUAAAGAACGUUUACACAUUGGUUUACA
UAUGUGUGUUCUCUUAUAGCAGUAGCUUACACUUAUUUUUAUAGUGGUGACUUUUUGAGUUUGCUUGUUAUGUUUUUAUGUGCUAUAUCUAGUGAUUGGUACAUUGGUGCCAUUGUUUUUAGGUUGUCACGUUUGAUUGUAAUUUUUUCACCUGAAAGUGUAUUUAGUGUUUUUGGUGAUGUGAA
ACUUACUUUAGUUGUUUAUUUAAUUUGUGGUUAUUUAGUUUGUACUUAUUGGGGCAUUUUGUAUUGGUUCAAUAGGUUUUUUAAAUGUACUAUGGGUGUUUAUGAUUUUAAGGUGAGUGCUGCUGAAUUUAAAUACAUGGUUGCUAAUGGACUUCAUGCACCACAUGGACCUUUUGAUGCACUUU
GGUUAUCAUUCAAACUACUUGGUAUUGGUGGUGACCGUUGUAUAAAAAUUUCAACUGUCCAAUCCAAACUGACUGAUUUGAAGUGUACUAAUGUUGUGUUAUUGGGUUGUUUGUCUAGUAUGAACAUUGCAGCAAAUUCUAGUGAAUGGGCUUAUUGUGUUGAUUUACACAAUAAGAUUAAUCUU
UGUGAUGACCCUGAAAAAGCUCAAGGUAUGUUGUUAGCACUCCUUGCGUUCUUUCUAAGUAAACAUAGUGAUUUUGGUCUUGAUGGCCUUAUUGAUUCUUAUUUUGAUAAUAGUAGCACCCUUCAGAGUGUUGCUUCAUCAUUUGUUAGUAUGCCAUCAUAUAUUGCUUAUGAAAAUGCUAGACA
AGCUUAUGAAGAGAUGCUAUUGCUAAUGGAUCUUCUUCUCAACUUAUUAAACAAUUGAAGCGUGCCAUGAAUAUCGCAAAGUCUGAAUUUGAUCAUGAGAUAUCUGUUCAGAAGAAAAUUAAUAGAAUGGCUGAACAAGCUGCUACACAGAUGUAUAAAGAAGCACGCUCUGUUAAUAGAAAAUCUA
AAGUUAUUAGUGCUAUGCACUCUUUACUUUUUGGAAUGUUAAGACGUUUGGAUAUGUCUAGUGUUGAAACUGUUUUGAAUUUAGCACGUGAUGGUGUUGUGCCAUUGUCAGUAAUACUGCAACUUCAGCUUCCAAACUAACUAUUGUAGUCCAGAUCUUGAAUCUUAUUCUAAGAUUGUUUGU
GAUGGUUCUGUUCAUUAUGCUGGAGUUGUUUGGCACUUAAUGAUGUUAAAGACAAUGAUGGUAGACCUGUUCAUGUUAAAGAGAUUACAAAGGAAAAUGUUGAAACUUUGACAUGGCCUCUUAUCCUUAAUUGUGAACGUGUUGUUAAACUUCAAAAUAAUGAAAUUAUGCCUGGUAAACUUAA
GCAAAAACCUAUGAAAGCUGAGGGUGAUGGUGGGUGUUUAGGUGAUGGUAAUGCCUUGUAUAAUACUGAGGGUGGUAAAACUUUUAUGUACGCUUAUAUUUCUAAUAAAGCUGACCUUAAAUUUGUUAAGUGGGAGUAUGAGGGUGGUUGCAACACAAUCGAGUUAGACUCUCCUUGUCGAUUUA
UGGUCGAAACACCUAAUGGUCCUCAAGUGAAGUAUUUGUAUUUUGUUAAAAAUUUAAACACCUUACGUAGAGGUGCCGUUCUUGGUUUUAUAGGUGCCACAAUUCGUCUACAAGCUGGUAAACAAACUGAAUUGGCUGUUAAUUCUGGACUUUUAACUGCUUGUGCUUUUUCUGUUGAUCCAGCA
ACUACUUACUUGGAAGCUGUUAAACAUGGUGCAAAACCUGUAAGUAAUUGUAUUAAGAUGUUAUCUAAUGGUGCUGGUAAUGGUCAAGCUAUAACAACUAGUGUAGAUGCUAACACCAAUCAAGAUUCUUAUGGUGGAGCGUCUAUUUGUUUGUAUUGUCGAGCUCACGUUCCUCACCCUAGUAU
GGAUGGUUACUGUAAGUUUAAGGGUAAAUGUGUUCAGGUUCCUAUUGGUUGUUUGGAUCCUAUUAGGUUUUGUUUAGAAAAUAAUGUGUGUAAUGUUUGUGGUUGCUGGUUGGGACAUGGAUGUGCUUGUGACCGUACAACUAUUCAAAGUGUUGACAUUUCUUAUUUAAACGAGCAAGGGGUUC
UAGUGCAGCUCGACUAGAACCCUGUAAUGGCACGGACAUCGAUAAGUGUGUUCGUGCUUUUGACAUUUAUAAUAAAAAUGUUUCAUUCUUGGGUAAGUGUUUGAAGAUGAACUGUGUUCGUUUUAAAAAUGCUGAUCUUAAGGAUGGUUAUUUUGUUAUAAAGAGGUGUACUAAGUCGGUUAUGG
AACACGAGCAAUCCAUGUAUAACCUACUUAACUUUUCUGGUGCUUUGGCUGAGCAUGACUUCUUUACUUGGAAAGAUGGCAGAGUCAUUUAUGGUAAUGUUAGUAGACAUAAUCUUACUAAAUAUACUAUGAUGGACUUGGUCUAUGCUAUGCGUAACUUUGAUGAACAAAAUUGUGAUGUUCUA
AAAGAAGUAUUAGUUUUAACUGGUUGUUGUGACAAUUCUUAUUUUGAUAGUAAGGGUUGGUAUGACCCAGUUGAAAAUGAAGAUAUACAUAGAGUUUAUGCAUCUCUUGGCAAAAUUGUAGCUAGAGCUAUGCUUAAAUGCGUUGCUCUAUGUGAUGCGAUGGUUGCUAAAGGUGUUGUUGGUGU
UUUAACAUUAGAUAACCAAGAUCUUAAUGGUAACUUUUAUGAUUUUGGUGAUUUUGUUGUAGCUUACCUAAUAUGGGUGUUCCCUGUGUGUACAUCAUAUUAUUCUUAUAUGAUGCCUAUUAUGGGUUUAACUAAUGUUUUAGCUAGUGAGUGUUUUGUCAAGAGUGAUAUUUUUGGUAGUGAUU
UUAAAACUUUUGAUUUGCUUAAGUAUGAUUUCACUGAACAUAAAGAAAAUUUAUUCAAUAAGUACUUUAAGCAUUGGAGUUUUGAUUAUCAUCCUAAUUGUAGUGACUGUUAUGAUGAUAUGUGUGUUAUACAUUGUGCUAAUUUUAAUACACUAUUUGCCACAACUAUACCAGGUACUGCUUUU
GGUCCACUAUGUCGUAAAGUUUUUAUAGAUGGUGUUCCACUUGUUACAACUGCUGGUUAUCAUUUUAAGCAAUUAGGUUUGGUUUGGAAUAAAGAUGUUAACACACACUCAGUUAGGUUGACAAUUACUGAACUUUUGCAAUUUGUCACCGACCCUUCCUUGAUAAUAGCUUCUUCCCCAGCACU
CGUUGAUCAACGCACUAUUUGUUUUUCUGUUGCAGCAUUGAGUACUGGUUUGACAAAUCAAGUGUUAAGCCAGGUCAUUUUAAUGAAGAGUUUUAUAACUUUCUUCGUUUAAGAGGUUUCUUUGAUGAAGGUUCUGAACUUACAUUAAAACAUUUCUUCUUCGCACAGAAUGGUGAUGCUGCUG
UUAAAGAUUUUGACUUUUACCGUUAUAAUAAGCCUACCAUUUUAGAUAUUUGUCAAGCUAGAGUCACAUAUAAGAUAGUCUCUCGUUAUUUUGACAUUUAUGAAGGUGGCUGUAUUAAGGCAUGUGAAGUUGUUGUAACAAAUCUUAAUAAGAGUGCUGGUUGGCCAUUAAAUAAGUUUGGUAAA
GCUAGUUUGUAUUAUGAAUCUAUAUCUUAUGAAGAACAGGAUGCUUUGUUUGCUUUGACAAAGCGUAAUGUCUUCCCUACUAUGACACAGCUGAAUCUUAAGUAUGCUAUUAGUGGUAAAGAACGUGCUAGAACUGUUGGUGGUGUUUCUCUGUUGUCCACAAUGACCACAGACAAUACCAUCA
AAAACAUCUUAAAUCCAUUGUUAAUACACGCAAUGCCACUGUUGUUAUUGGUACUACCAAAUUUUAUGGUGGUUGGAAUAAUAUGUUGCGUACUUUAAUUGAUGGUGUUGAAAACCCUAUGCUCAUGGGUUGGGAUUAUCCCAAAUGUGAUAGAGCUUUGCCUAACAUGAUACGUAUGAUUUCAG
CCAUGGUGUUGGGCUCUAAGCAUGUUAAUUGUUGUACUGCAACAGAUAGGUUUUAUAGGCUUGGUAAUGAGUUGGCACAAGUUUUAACAGAAGUUGUUUAUUCUAAUGGUGGUUUUUAUUUUAAGCCAGGUGGUACGACUUCUGGUGACGCUAGUACAGCUUAUGCUAAUUCUAUUUUUAACAUU
UUUCAAGCCGUAAGCUCUAACAUUAACAGGUUGCUUAGUGUCCCAUCAGAUUCAUGUAAUAAUGUUAAUGUUAGGGAUCUACAACGACGUCUGUAUGAUAAUUGCUAUAGGUUAACUAGUGUUGAAGAGUCAUUCAUUGAUGAUUAUUAUGGUUAUCUUAGGAAACAUUUUUCAAUGAUGAUUCU
CUCUGAUGAUGGUGUUGCUGUUAUAACAAGGAUUAUGCUGAGUUAGGUUAUAUAGCAGACAUUAUGUGCUUUUAAAGCCACUUUGUAUUACCAGAAUAAUGUCUUUAUGAGUACUUCUAAAUGUUGGUUGAAGAAGAUUUAACUAAGGGACCACAUGAGUUUUGUUCCCAGCAUACUAUGCAAA
UAGUUGACAAAGAUGGUACCUAUUAUUUGCCUUACCCAGAUCCUAGUAGGAUCUUGUCAGCUGGUGUUUUUGUUGAUGUUGUUAAGACAGAGCUGUUGUUUGCUUUAGCUAUUGAUGCAUACCCUCUUUCAAAACACCCUAAUUCUGAAUAUCGUAAGGUUUUU
UACGUAUUACUUGAUUGGGUUAAGCAUCUUAACAAAAAUUUGAAUGAGGGUGUUCUUGAAUCUUUUUCUGUUACACUUCUUGAUAAUCAAGAAGAUAAGUUUUGGUGUGAAGAUUUUUAUGCUAGUAUGUAUGAAAAUUCUACGAUAUUGCAAGCUGCUGGUUUAUGUGUUGUUUGGUUCACA
AACUGUACUUCGUUGUGGUGAUUGUCUGCGUAAGCCUAUGUUGUGCACUAAAUGCGCAUAUGAUCAUGUAUUUGGUACCGACCACAAGUUUAUUUUGGCUAUAACACCGUAUGUAUGUAAUGCAUCAGGUUGUGGUGUUAGUGAUGUCAAAAAAUUGUAUCUUGGUGGUUUGAAUUACUAUUGUA
CAAAUCAUAAACCACAGUUGUCUUUUCCAUUAUGUUCAGCUGGUAAUAUAUUUGGUUUAUAUAAAAAUUCAGCAACUGGUUCCUUAGAUGUUGAAGUUUUUAAUAGGCUUGCAACGUCUGAUUGGACUGAUGUUAGGGACUAUAAACUUGCUAAUGAUGUUAAAGAUACACUUAGACUCUUUGCG
GCUGAAACUAUUAAAGCUAAAGAAGAGAGUGUUAAGUCUUCUUAUGCUUUUGCAACUCUUAAAGAGGUUGUUGGCCUAAAGAAUUGCUUCUUAGUUGGGAAAGUGGUAAAGUUAAACCACCUUUGAAUCGUAAUUCUGUUUUCACCUGUUUUCAAAUAAGUAAGGACUCAAAUUCCAAAUAGG
UGAGUUCAUCUUUGAGAGGUUGAAUAUGGUUCUGAUACUGUUACGUAUAAGUCUACUGUAACUACUAAGUUAGUUCCUGGUAUGAUUUUUGUCUUAACAUCUCACAAUGUCCAACCUUUACGUGCACCAACUAUUGCUAACCAAGAGAAGUAUUCUAGCAUUUAUAAAUGCACCCUGCUUUUA
AUGUCAGUGAUGCAUAUGCUAAUUUGGUUCCAUAUUACCAACUUAUUGGUAAACAAAAGAUAACUACAAUACAGGGUCCUCCUGGUAGUGGUAAGUCACAUUGUUCUAUUGGACUUGGAUUGUACUACCCAGGUGCGCGUAUUGUUUUUGUUGCUUGUGCCCAUGCUGCUGUUGAUUCCUUAUGU
GCAAAAGCUAUGACUGUUUAUAGCAUUGAUAAGUGUACUAGGAUUAUACCUGCAAGAGCUCGGGUUGAGUGUUAUAGUGGCUUUAAACCAAAUAACACUAGUGCACAAUACAUAUUUAGCACUGUUAACGCAUUACCUGAGUGUAAUGCUGAUAUCGUUGUUGUAGAUGAAGUUUCAAUGUGUAC
AAAUUAUGACCUUUCUGUUAUUAAUCAGCGUUUAUCAUAUAAACAUAUUGUUUAUGUUGGUGAUCCACAACAACUUCCUGCACCUAGAGUAAUGAUUACUAAAGGUGUUAUGGAGCCUGUUGAUUAUAACGUUGUUACUCAACGUAUGUGUGCUAUAGGCCCUGAUGUUUUUCUUCAUAAAUGUU

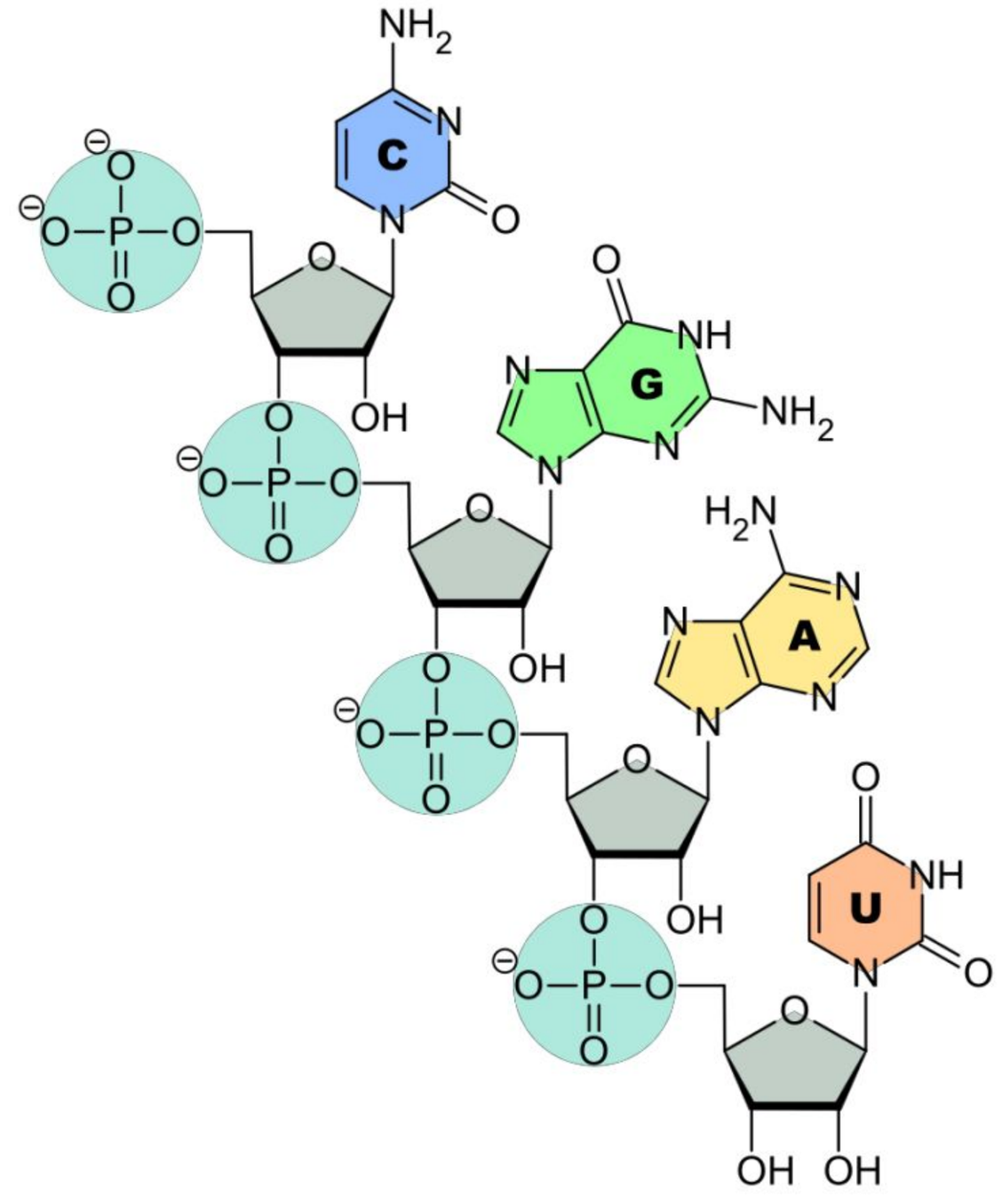
AUAGAUGUCCUGCUGAAAUAGUUAAUACAGUUUCUGAACUUGUUUAUGAGAACAAGUUUGUCCCUGUUAAACCUGCUAGUAAACAGUGUUUUAAAGUCUUUUUUAAGGGUAAUGUACAGGUUGACAAUGGUUCUAGUAUUAACAGAAAGCAGCUUGAAAUAGUUAAGCUGUUUUUAGUUAAAAAU
CCAAGUUGGAGUAAGGCUGUGUUUAUUUCUCCUUAUAAUAGUCAGAAUUAUGUUGCUAGUAGAUUUUUAGGACUUCAAAUUCAAACUGUUGAUUCUUCUCAAGGUAGUGAGUAUGAUUAUGUAAUCUAUGCACAAACUUCUGACACUGCACAUGCUUGCAAUGUAAACCGUUUUAAUGUUGCUAU
AACACGUGCUAAGAAGGGUAUAUUUUGUGUAAUGUGUGAUAAAACUUUGUUUGAUUCACUUAAGUUUUUUGAGAUUAAACAUGCAGAUUUACACUCUAGCCAGGUUUGUGGCUUGUUUAAAAAUUGUACACGCACUCCUCUUAAUUUACCACCAACUCAUGCACACACUUUCUUGUCGUUGUCAG
AUCAGUUUAAGACUACAGGUGAUUUAGCUGUUCAAAUAGGUUCAAAUAACGUUUGUACUUAUGAACAUGUUAUAUCAUUUAUGGGUUUUAGGUUUGAUAUUAGUAUUCCUGGUAGUCAUAGUUUGUUUUGUACACGUGACUUUGCUAUUCGCAAUGUGCGUGGUUGGUUGGGUAUGGAUGUUGAA
AGUGCUCAUGUUUGUGGCGAUAACAUAGGUACUAAUGUUCCUUUACAGGUUGGUUUUUCAAAUGGUGUUAAUUUUGUUGUGCAAACUGAAGGUUGUGUGUCUACCAAUUUUGGUGAUGUUAUUAAACCUGUUUGUGCAAAAUCUCCACCAGGUGAACAAUUUAGACACCUUGUUCCUCUUUUACG
UAAAGGACAACCUUGGUUAAUUGUUCGUAGACGCAUUGUGCAAAUGAUAUCUGAUUAUUUGUCCAAUUUGUCUGACAUUCUUGUCUUUGUUUUGUGGGCAGGUAGUUUGGAAUUAACUACAAUGCGUUACUUUGUAAAAAUAGGGCCAAUUAAAUAUUGUUAUUGUGGUAAUUCUGCCACUUGUU
AUAAUUCAGUUAGUAAUGAAUAUUGUUGUUUUAAACAUGCAUUGGGUUGUGAUUAUGUUUACAAUCCGUAUGCUUUUGAUAUACAACAGUGGGGUUAUGUUGGUUCCUUGAGCCAAAACCACCACACAUUCUGUAACAUUCAUAGAAACGAGCAUGAUGCCUCUGGUGAUGCUGUUAUGACACGU
UGUUUGGCAGUACAUGAUUGUUUUUGUCAAAAAUGUUGAUUGGACUGUAACGUACCCCUUUAUUGCAAAUGAGAAAUUUAUCAAUGGCUGUGGGCGUAAUGUCCAGGGACAUGUUGUUCGUGCAGCCUUGAAAUUGUAUAAACCUAGUGUUAUUCAUGACAUUGGUAAUCCUAAAGGUGUACGUUG
UGCUGUUACUGAUGCCAAAUGGUACUGUUAUGACAAGCAACCUGUUAAUAGUAAUGUCAAGUUGUUGGAUUAUGAUUAUGCAACUCAUGGUCAACUUGAUGGUCUUUGUCUAUUCUGGAAUUGUAAUGUUGAUAUGUAUCCAGAAUUUUCAAUUGUGUGUCGUUUUGACACACGUACUCGUCUG
UUUUUAAUUUAGAAGGUGUAAUGGUGGUUCUCUUUAUGUUAACAAACAUGCGUUUCAUACACCAGCAUAUGAUAAACGUGCUUUUGUUAAAUUAAAACCUAUGCCCUUUUUUUACUUUGAUGACAGUGAUUGUGAUGUUUGUGCAAGAACAAGUUAAUUAUGUACCCCUUCGCGCUAGUAGUUGU
GUUACCCGUUGUAAUAUAGGUGGUGCUGUUUGUUCAAAACAUGCAAAUUUGUAUCAAAAAUAUGUUGAGGCAUAUAAUACAUUUACACAGGCUGGUUUUUAACAUUUGGGUACCACAUAGUUUUGAUGUUUAUAAUUUGUGGCAAAUUUUUAUUGAAACUAAUUUACAAAGUCUUGAAAAUAUAGC
AUUUAAUGUUGUAAAAAAAGGGUGUUUUACUGGUGUUGAUGGUGAGUUACCUGUUGCAGUUGUUAACGACAAAGUUUUUGUUCGCUAUGGCGAUGUUGACAACUUAGUUUUUACAAAUAAAACAACAUUGCCUACUAAUGUUGCUUUUGAAUUGUUUGCAAAACGAAAAAUGGGUUUAACACCAC
CAUUGUCUAUUCUCAAAAAUCUCGGUGUUGUUGCUACAUAUAAAUUUGUUUUAUGGGAUUAUGAAGCUGAAAGACCUUUUACCUCAUAUACUAAGAGUGUAUGUAAAUACACUGAUUUUAAUGAGGAUGUUUGUGUUUGUUUUGACAAUAGUAUUCAGGGUUCGUAUGAGCGUUUUACGCUUACU
ACGAACGCUGUUUUAUUUUCUACUGUUGUCAUUAAAAAUUUAACACCUAUAAAGUUGAAUUUUGGUAUGUUGAAUGGUAUGCCAGUUUCUUCUAUUAAGGGUGAUAAAGGUGUUGAAAAAUUAGUUAAUUGGUACAUAUAUGUUCGUAAAAAUGGUCAAUUUCAAGAUCACUAUGAUGGUUUUUA
CACUCAAGGUAGGAAUUUAUCAGACUUUACACCAAGAAGUGAUAUGGAGUAUGAUUUUCUUAACAUGGAUAUGGGUGUUUUUAUUAAUAAAUAUGGUCUUGAGGAUUUUAAUUUUGAACAUGUUGUAUAUGGUGAUGUUUCAAAAACUACAUUAGGAGGUCUUCAUUUGUUGAUAUCACAGUUUA
GGCUUAGUAAAAUGGGUGUUUUGAAAGCUGAUGAUUUUGUCAUUGCUUCUGACACAACUUUGAGGUGCUGCACUGUUACUUAUCUUAAUGAACUUAGUUCAAAAGUUUGUUGUUGGACGACUUUGUUACUAUACUAAAGAGUUUAGAUCUUGGUGUAAUAUCUAAA
GUUCAUGAAGUUAUUAUAGAUAAUAAACCUUAUAGGUGGAUGUUGGUGUAAAGAUAACCACUUGUCCACUUUUAUCCACAGUUGCAGUCUGCUGAAUGGAAGUGUGGUUAUGCUAUGCCACAAAUUUAUAAGCUUCAACGUAUGUGUUUGGAACCUUGUAAUUUAUAUAAUUAAUGGUGCUGG
UAUUAAGUUGCCUAGUGGUAUAAUGUUAAAUGUUGUUAAAUACACUCAGCUUUGUCAAUACCUAAAUAGCACUACAAUGUGCGUACCUCAUAAUAUGCGUGUUUUGCACUAUGGUGCUGGUUCUGACAAAGGUGUGGCACCUGGUACAACUGUUUUAAAACGUUGGCUACCACCUGAUGCAAUAA
UCAUUGAUAAUGACAUCAAUGAUUAUGUUAGUGAUGCAGAUUUUAGCAUUACAGGUGAUUGUGCUACUGUUUACCUUGAAGAUAAGUUUGACUUACUUAUUUCUGAUAUGUAUGAUGGUAGAAUUAAAUUUUGUGAUGGUGAAAACGUCUCUAAAGAUGGUUUUUUUACUUAUCUUAAUGGUGUU
AUUAGAGAAAAAUUAGCUAUUGGUGGUAGUGUUGCCAUUAAGAUUACAGAAUAUAGUUGGAAUAAGUAUCUUUAUGAAUUAAUACAGAGAUUUGCUUUUUGGACUUUGUUUUGCACGUCUGUUAAUACAUCCUCUUCAGAAGCUUUUCUUAUUGGUAUUAAUUAUUUAGGUGACUUUAUUCAAGG
UCCUUUUAUAGCUGGUAACACUGUUCAUGCUAAUUAUAUAUUUUGGCGUAAUUCUACUAUUAUGUCUUUGUCAUACAAUUCAGUUUUAGAUCUAAGUAAGUUUGAAUGUAAACAUAAGGCCACUGUUGUUGUUACACUUAAAGAUAGUGAUGUAAAUGAUAUGGUUUUGAGUUUGAUUAAGAGUG
GUAGGUUGUUGUUACGUAAUAGUGGCCGUUUUGGUGGUUUUAGUAAUCAUUUAGUCUCAACUAAAUGAAACUUUUCUUGAUUUUGCUUUAUUUUGCCCCUGGUUUCUUGCUUUUCUACAUGUAACAGUAAUGCUAGUAUUUCUAUGUUACAAUUAGGUGUUCCUGAUAACUCUUCAACUAUUGUCA
CAGGUUUGUUGCCAGUCCAUUGGAUUUGUGCUAAUCAGAGCACAUCUAGUUACCCAGCCAAUGGCUUUUUCUAUAUUGAUGUCGGUAAACACCGUAGUGCUUUUGCACUCCAUAGUGGUUAUUAUGAAGUCGUAACCAGUAUUAUAUUUAUGAUCUCACUAAUAAAAUACAUUUAAAUGCUCCUGUCACU
CUGAAGAUUUGUAAGUUUGGAAACACUUCUUUUGAUUUUUUAAGUAAUGUUUCUACUUCUCAUCAUUGUAUAGUUAAUUCGUCAUUCACAGAACAGUUAGGUGUGCCUUUGGGCAUAACUAUAUCGGGUGAAACUGUACGUUUGCAUUUAUAUAAUGCAACUCGCACCUUUUAUGUGCCUGCCGC
UUAUAAACUUACUAAACUUAGUGUUAAAUGUUACUUUAGUGAAUCCUGUGUUUUUAGUGUUGUCAAUGCUACCGUUACUGUUAAUGUCACCACACUUAAUGGCCGUAUAGUUAAUUACACUGUUUGUGAUGAUUGUAAUGGUUACACUGAUAACAUAUUUUCUGUUCAACAGGAUGGCCGCAUUC
CUAAUGGUUUUUCUUUUAAUAAUUGGUUUUUGUUAACUAAUGGUUCUACACUAGUGGACGGUGUCUCUAGACUUUACCAACCACUUCGUUUAACUUGUUUAUGGCCUGUACCUGGUCUUAAAUCUUCGACUGGUUUUGUUUAUUUUAACGCCACUGGUUCUGAUGUUAAUUGUAAUGGUUAUCAA
CAUCAUUCUGUUGCUGAUGUUAUGCGUUACAAUCUUAACUUCAGUUCUAAUUCUGUGGAUAAUCUCAAGAGUGGUGUUAUAGUUUUUAAAACUCUACAGUACGAUGUUUUGUUUUACUGUAGUAAUUCUUCUUCGGGCGUUCUAGACACCACAAUACCUUUUGGCCCUUCCUCUCAACCUUAUUA
CUGUUUUAUAAAUAGUACCAUCAACACCACUCAUGUUAGCACUUUUGUGGGUGUCCUACCACCCAUUGUGCGUGAGAUUGUUGUUGCUAGAACUGGUCAGUUUUAUAUUAAUGGUUUUAAGUAUUUUGAUUUGGGUUUUAUCGAAGCUGUUAAUUUUAAUGUCACGACAGCUAGUGCUACAGAUU
UUUGGACGGUUGCAUUUGCUACUUUUGUUGAUGUUUUGGUUAAUGUUAGUGCAACUAACAUUCAAAACUUACUUUAUUGCGAUUCUCCAUUUGAAAGAUGGCAGUGUGAGCACUUGCAGUUUGGAUUGCAAGUUUGUUUAUUCUGCAAAUUUUCCUGAUGAUAAUGUUUUGCCUGAUACUUAU
GUUGCACUUCCUAUUUAUUAUCAACAUACUGAUAUAAAUUUUACUGCGACUGCAUCUUUUGGUGGUUCUUGUUAUGUUUGCAAACCACAUCAGGUUAAAUAUAUCUCUUAAUGGUAAUACUUCAGUGUGUGUUAGAACAUCUCAUUUUUCAAUUAGGUAUAUUUAUAACCGCGUUAAGAGUGGUCC
ACCAGGUGACUCUUCAUGGCACAUUUAUUUAAAGAGUGGCACUUGUCCAUUUUCUUUUUCUAAGUUAAAUAAUUUUCAAAAGUUUAAGACUAUUUGUUUCUCAACCGUCGUAGUGCCUGGUAGUUGUAAUUUUCCACUUGAAGCCACCUGGCAUUACACUUCUUAUACUAUUGUUGGUGCUUUGU
AUGUUACUUGGUCUGAAGGUAAUUCCAUUACUGGUGUACCUUAUCCUGUCUCUGGUAUUCGUGAGUUUAGUAAUUUAGUUUUAAAUAAUUGUACCAAAUAUAAUAUUUAUGAUUAUGUUGGUACUGGAAUUAUACGUUCUUCAAACCAGUCACUUGCUGGUGGUAUUACAUAUGUUUCUAACUCU
GGUAAUUUACUUGGUUUUAAAAAUGUUUCCACUGGUAACAUUUUUAUUGUGACACCAUGUAACCAACCAGACCAAGUAGCUGUUAUCAACAAAGCAUUAUUGGUGCCAUGACUGCUGUUAAUGAGUCUAGAUAUGGCUUGCAAAACUUACUACAGUUACCUAACUUUUAUUAUGUUAGUAAUGG
UGGUAACAAUUGCACUACGGCCGUUAUGACUUAUUCUAAUUUUGGUAUUUGUGCUGAUGGUUCUUUAAUUCCUGUUCGUCCGCGUAAUUCUAGUGAUAAUGGUAUUUCAGCCAUAGUCACUGCUAAUUUAUCCAUUCCCUCUAACUGGACUACUUCAGUUCAAGUUGAGUACCUCCAAAUUACUA
GUACUCCAAUAGUUGUUGAUUGUGCUACUUAUGUGUGUAAUGGUAACCCUCGCUGUAAGAAUCUACUUAAGCAGUAUACUUCUGCUUGUAAAACUAUUGAAGAUGCCUUACGACUUAGUGCUCAUUUGGAAACUAAUGAUGUUAGUAGUAUGCUAACUUUUGAUAGCAAUGCUUUUAGUUUGGCU
AAUGUUACUAGUUUUGGAGAUUAUAACCUUUCUAGUGUUUUACCUCAGAGAAACAUUCAUUCAAGCCGUAUAGCAGGACGUAGUGCUUUGGAAGAUUUGUUGUUUAGCAAAGUUGUUACAUCUGGUUUGGGUACUGUUGAUGUUGACUAUAAGUCUUGUACUAAAGGUCUUUCUAUUGCUGACCU
UGCUUGUGCUCAGUACUACAAUGGCAUAAUGGUUUUGCCAGGUGUUGCUGAUGCUGAACGUAUGGCCAUGUACACAGGUUCUCUUAUAGGUGGCAUGGUGCUCGGAGGUCUUACAUCAGCAGCCGCCAUACCUUUUUCUUUGGCACUGCAAGCACGACUUAACUAUGUUGCUUUACAAACUGAUG
UGCUUCAAGAAAAUCAGAAAAUUUUGGCUGCAUCAUUUAAUAAGGCUAUUAAUAACAUUGUUGCUUCUUUUAGUAGCGUUAAUGAUGCUAUUACACAAACUGCAGAGGCUAUACAUACUGUUACUAUUGCACUUAAUAAGAUUCAGGAUGUUGUUAAUCAACAGGGUAGUGCUCUUAACCAUCUC
ACUUCACAAUUGAGACAUAAUUUUCAGGCCAUUUCUAAUUCAAUUCAAGCUAUUUAUGACCGGCUUGAUUCAAUUCAAGCCGAUCAACAAGUUGACAGAUUAAUUACUGGACGGCUUGCAGCUUUGAAUGCAUUUGUUUCCCAAGUUUUGAAUAAAUAUACUGAAGUUCGUAGUUCCAGACGCUU
AGCACAGCAGAAGAUUAAUGAAUGUGUCAAAUCACAAUCUAAUAGAUAUGGUUUUUGUGGCAAUGGCACCCAUAUCUUUUCAAUCGUCAACUCAGCUCCAGAUGGUUUGCUUUUUCUUCAUACUGUUUUGCUGCCAACCGACUACAAGAAUGUAAAGGCGUGGUCGGGUAUCUGUGUUGAUGGCA
UUUACGGCUAUGUUCUGCGUCAACCUAACUUGGUUCUUUAUUCUGAUAACGGUGUCUUUCGUGUAACUUCCAGGGUCAUGUUUCAACCUCGUUUACCUGUUUUGUCUGAUUUUGUGCAAAUAUAUAAUUGUAAUGUUACUUUUGUUAACAUAUCUCGUGUCGAGUUACAUACUGUCAUACCUGAC
UACGUUGAUGUUAAUAAAACAUUACAAGAGUUUGCACAAAACUUACCAAAGUAUGUUAAGCCUAAUUUUGACUUGACUCCUUUUAAUUUAACAUAUCUUAAUUUGAGUUCUGAGUUGAAGCAACUCGAAGCUAAAACUGCUAGUCUUUUCCAAACUACUGUUGAAUUACAAGGUCUUAUUGAUCA
GAUUAACAGUACAUAUGUUGAUUUGAAGUUGCUUAAUAGGUUUGAAAAUUAUAUUAAAUGGCCUUGGUGGGUUUGGCUCAUUAUUUCUGUUGUUUUUGUUGUAUUGUUGAGUCUUCUUGUGUUUUUGUUGUCUUUCUACAGGUUGUUGUGGUUGUUGCAAUUGUUUAACUUCAUCAAUGCGAGGUU
GUUGUGAUUGUGGUUCAACUAAACUUCCUUAUUACGAAUUUGAAAAGGUCCACGUUCAAUAAUGCCUUUCGGUGGCCUAUUUCAACUUACUCUUGAAAGUACUAUUAAUAAGAGUGUGGCUAAUCUCAAAUUACCACCUCAUGAUGUUACUGUCUUGCGUGACAAUCCUCAAACCUGUUACUACAC
UUAGUACUAUCACUGCUUAUUUGUUAGUUAGUUUGUUUGUCACUUAUUUUGCUUUAUUCAAACCUCUUACUGCUAGAGGUCGUGUUGCUUGUUUUGUUUUAAAACUAUUGACACUAUUUGUCUAUGUGCCUUUAUUGGUUCUUUUUGGUAUGUAUCUUGACAGUUUUAUAAUUUUUUCUACGCUG
UUGUUUCGAUUCAUACAUGUUGGCUAUUAUGCCUAUCUCUAUAAAAAUUUUUCAUUUGUUUUGUUCAAUGUUACUAAACUAUGCUUCGUUUCAGGCAAGUGUUGGUAUCUUGAACAAUCAUUUUAUGAAAAUCGUUUUGCUGCUAUUUAUGGUGGUGACCACUAUGUCGUUUUAGGUGGUGAAAC
UAUUACUUUUGUUUCUUUUGAUGACCUUUAUGUUGCUAUUAGAGGUUCUUGUGAAAAGAACCUACAACUUAUGCGUAAGGUUGACUUGUAUAAUGGUGCUGUCAUUUACAUUUUUGCCGAAGAGCCUGUUGUUGGUAUAGUCUACUCCUCUCAACUAUACGAAGAUGUUCCUUCGAUUAAUUGAU
GACAAUGGCAUUGUCCUCAAUUCCAUUUUAUGGCUCCUUGUUAUGAUAUUUUUCUUUGUGUUGGCAAUGACUUUUAUUAAACUGAUUCAAUUGUGUUUUACUUGUCAUUAUUUUUUUAGUAGGACAUUAUAUCAACCAGUUUAUAAAAUUUUUCUUGCUUACCAAGAUUAUAUGCAAAUAGCACC
UGUUCCAGCUGAAGUACUAAAUGUCUAAACUAAACGAUGUCUAAUAGUAGUGUGCCUCUUUCAGAGGUUUAUGUCCAUUUACGUAACUGGAACUUUAGUUGGAAUUUAAUUCUAACGCUUUUUAUAGUUGUGUUGCAGUAUGGGCAUUAUAAGUAUAGCAGACUUCUUUAUGGUUUAAAGAUGUC
UGUUUUAUGGUGUUUAUGGCCACUUGUUCUAGCUUUGUCUAUUUUUGACUGUUUUGUCAAUUUUAAUGUGGACUGGGUCUUUUUUGGUUUUAGUAUUCUUAUGUCUAUUAUUACACUUUGUUUGUGGGUUAUGUAUUUUGUUAAUAGUUUCAGACUUUGGCGCCGUGUUAAAACUUUUUGGGCUU
UUAAUCCUGAAACUAAUGCAAUCAUCUCUCUCCAGGUUUACGGACAUAAAUUAUUACUUACCGGUGAUGGCUGCACCUACAGGUGUUACAUUAACACUUCUCAGUGGGUGUACUUCUUGUUGAUGGCCAUAAGAUUGCUACUCGUGUUCAAGUGGGUCAGUUGCCUAAAUAUGUAAUAGUUGCUACA
CCUAGUACCACAAUUGUUUGUGACCGUGUUGGUCGCUCUGUUAAUGAAACAAGCCAGACUGGUUGGGCAUUCUACGUCCGUGCUAAACAUGGUGAUUUUUCUGGUGUUGCCUCUCAGGAGGGUGUUUUGUCAGAAAGAGAGAAGUUGCUUCAUUUAAUCUAAACUAAACAAAAUGGCUAGUGUAA
AUUGGGCCGAUGACAGAGCUGCUAGGAAGAAAUUUCCUCCUCCUUCAUUUUACAUGCCUCUUUUGGUUAGUUCUGAUAAGGCACCAUAUAGGGUCAUCCCCAGGAAUCUUGUCCCUAUUGGUAAGGGUAAUAAAGAUGAGCAGAUUGGUUAUUGGAAUGUUCAAGAGCGUUGGCGUAUGCGCAGG
GGGCAACGUGUUGAUUUGCCUCCUAAAGUUCAUUUUUAUUACCUAGGUACUGGACCUCAUAAGGACCUUAAAUUCAGACAACGUUCUGAUGGUGUUGUUUGGGUUGCUAAGGAAGGUGCUAAAACUGUUAAUACCAGUCUUGGUAAUCGCAAACGUAAUCAGAAACCUUUGGAACCAAAGUUCUC
UAUUGCUUUGCCUCCAGAGCUCUCUGUUGUUGAGUUUGAGGAUCGCUCUAAUAACUCAUCUCGUGCUAGCAGUCGUUCUUCAACUCGUAACAACUCACGAGACUCUUCUCGUAGCACUUCAAGACAACAGUCUCGCACUCGUUCUGAUUCUAACCAGUCUUCUUCAGACCUUGUUGCUGCUGUUA
CUUUGGCUUUAAAGAACUUAGGUUUUGAUAACCAGUCGAAGUCACCUAGUUCUUCUGGUACUUCCACUCCUAAGAAACCUAAUAAGCCUCUUCUCAGUUGAAGAAACCUCGUUGGAAGCGUGUUCCUACCAGAGAGGAAAAUGUUAUUCAGUGCUUUUGU
CCUCGUGAUUUUAAUCACAAUAUGGGGGAUUCAGAUCUUGUUCAGAAUGGUUGUGAUGCCAAAGGUUUUCCACAGCUUGCUGAAUUGAUUCCCAAUCAGGCUGCGUUAUUCUUUGAUAGUGAGGUUAGCACUGAUGAAGUGGGUGAUAAUGUUCAGAUUACCUAUACCUACAAAAUGCUUGUAGC
UAAGGAUAAUAAGAACCUUCCUAAGUUCAUUGAGCAGAUUAGUGCUUUUACUAAACCCAGUUCUAUCAAAGAAAUGCAGUCACAAUCAUCUCAUGCUGUUCAGAACACAGUACUUAAUGCUUCUAUUCCAGAAUCUAAACCAUUGGCUGAUGAUGAUUCAGCCAUUAUAGAAAUUGCAACGAGG
UUUUGCAUUAAAUUGUUUUGUAAUUCCAGUUGAAUGUUUAUUAUUAUUAGUUGCAACCCCAUGCGUUUAGCGCAUGAUAAGGGUUUAGUCUUACACACAAUGGUAGGCCAGUGAUAGUAAAGUGUAAGUAAUUUGCUAUCAUAUUAACAUGUCUAGAGGAAAGUCAGAACUUUUUCUGUUUGUGU
UGUUGGAGUACUUAAAGAUCGCAUAGGCGCGCCAACAAUGGAAGAGCCAACAACAUAUCUAAAAAUGUUUUGUCUGGUACUUGUUAAUGAUA'

The RNA genome is made of phosphate groups, ribose sugars, and four nucliotides, which are cytosine, guanine, adonine, and uricil.

In [32]:
```
Image(filename='RNA_molecule.jpg')
```

Out[32]:

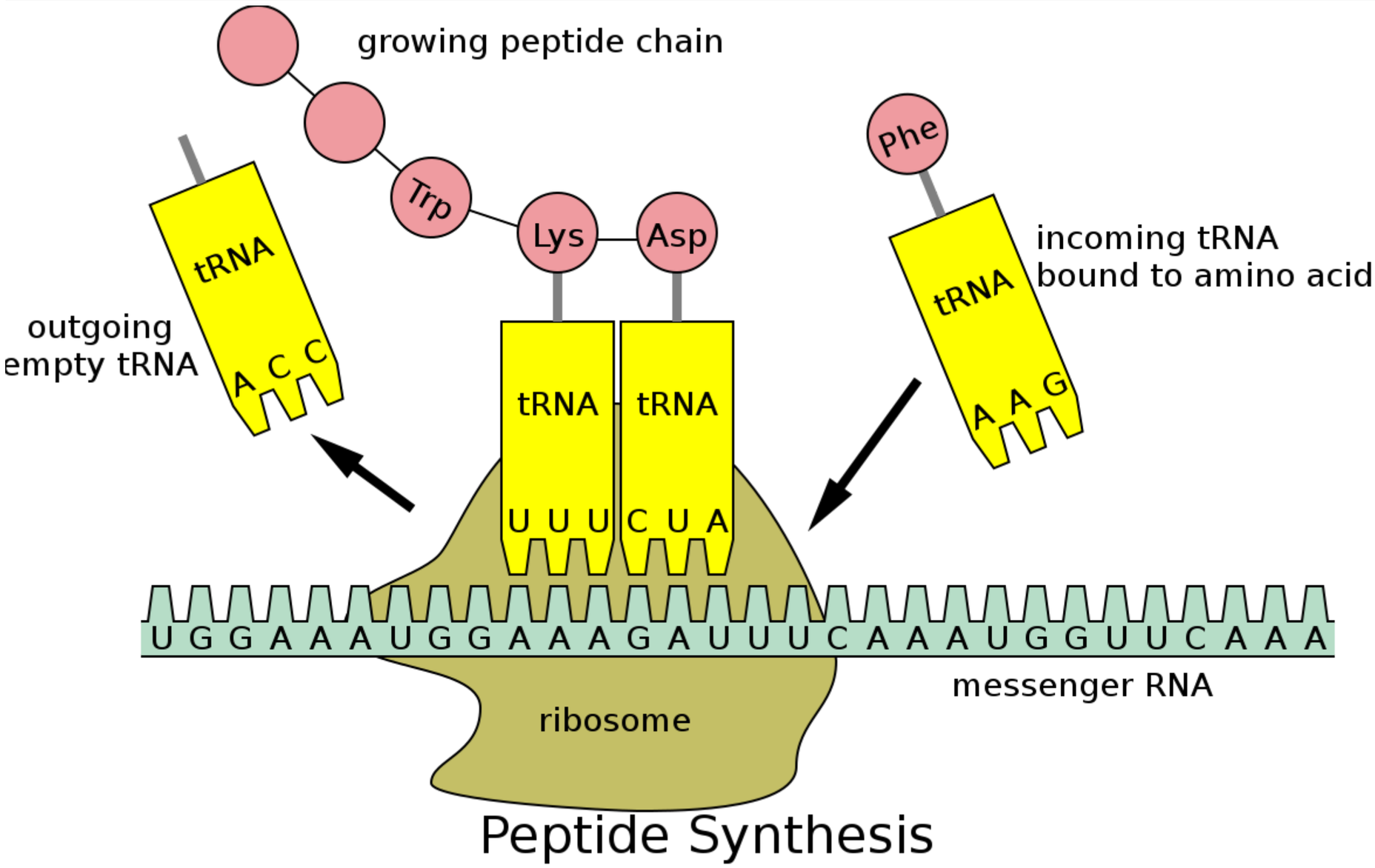
RNA



The RNA is translated to proteins in three steps: initiation, which is when a ribosome, mRNA (sgRNA is a type of mRNA), and tRNA connect; elongation, which is when tRNA transfers codons to anticodons, and then brings amino acids to the ribosome to be linked; and termination, which is when the polypeptide chain is complete.

In [225…
```
Image(filename='translation.png')
```

Out[225…



Data Mining

In [138…
```
len(genome)
```

Out[138…
```
27470
```

In the genome is an '\n' character that is python for new line that will be removed.

In [137…
```
set(genome)
```

Out[137…
```
{'\n', 'A', 'C', 'G', 'U'}
```

Data Cleaning

Remove '\n' from genome and protein strings.

In [139…
```
genome=genome.replace('\n','')
```

Makes a dataframe of sars-cov-2 mRNA nucleotides and mRNA antinucleotides.

In [140…
```
nucleotides_df=pd.DataFrame(columns=['Nucleotides'])
for i in genome:
    nucleotides_df.loc[len(nucleotides_df.index)] = [i]
nucleotides_df
```

Out[140…

	Nucleotides
0	A
1	C
2	U
3	U
4	U
…	…
27464	U
27465	G
27466	A
27467	U
27468	A

27469 rows × 1 columns

Adds a lagged RNA nucleotides column for nucleotide sequence prediction.

In [141…
```
nucleotides_df['Nucleotides_lagged']=nucleotides_df['Nucleotides'].shift(-1)
nucleotides_df
```

Out[141…

	Nucleotides	Nucleotides_lagged
0	A	C
1	C	U
2	U	U
3	U	U
4	U	G

	Nucleotides	Nucleotides_lagged
...	...	...
27464	U	G
27465	G	A
27466	A	U
27467	U	A
27468	A	NaN

27469 rows × 2 columns

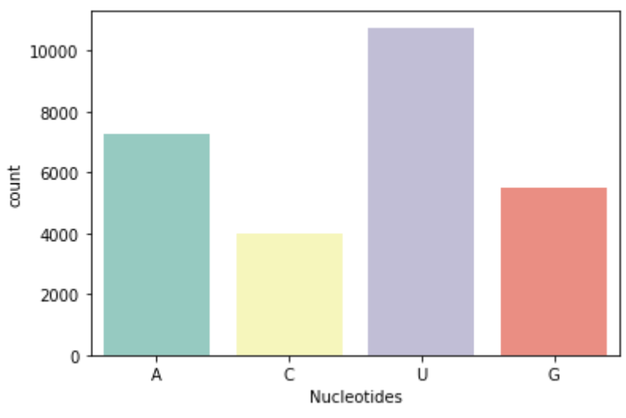
In [142…
```
nucleotides_df['Nucleotides_lagged']=nucleotides_df['Nucleotides_lagged'].fillna('A')
```

Data Exploration

Sars-cov-2 genome is made up of apoximatly 30% adenine, 18% guanine, 20% cytosine and 32% uracil.

In [894…
```
sns.countplot(nucleotides_df['Nucleotides'], palette='Set3')
```

Out[894… `<AxesSubplot:xlabel='Nucleotides', ylabel='count'>`



Percentage of uricil in genome data.

In [901…
```
len(nucleotides_df['Nucleotides'].loc[nucleotides_df['Nucleotides']=='U'])/len(nucleotides_df['Nucleotides'])*100
```

Out[901… `39.14594633950999`

Percentage of guanine in genome data.

In [902…
```
len(nucleotides_df['Nucleotides'].loc[nucleotides_df['Nucleotides']=='G'])/len(nucleotides_df['Nucleotides'])*100
```

Out[902… `20.01528996323128`

Percentage of adonine in genome data.

In [903…
```
len(nucleotides_df['Nucleotides'].loc[nucleotides_df['Nucleotides']=='A'])/len(nucleotides_df['Nucleotides'])*100
```

Out[903… 26.338781899595908

Percentage of cytosine in genome data.

In [904…
```
len(nucleotides_df['Nucleotides'].loc[nucleotides_df['Nucleotides']=='C'])/len(nucleotides_df['Nucleotides'])*100
```

Out[904… 14.49998179766282

Feature Engineering

Encodes nucleotides to numbers.

In [335…
```
nucleotides_df['Nucleotides']=nucleotides_df['Nucleotides'].astype('category')
nucleotides_df['Nucleotides']=nucleotides_df['Nucleotides'].cat.codes
nucleotides_df['Nucleotides_lagged']=nucleotides_df['Nucleotides_lagged'].astype('category')
nucleotides_df['Nucleotides_lagged']=nucleotides_df['Nucleotides_lagged'].cat.codes
```

Makes independent and dependent variable.

In [336…
```
X=nucleotides_df['Nucleotides']
y=nucleotides_df['Nucleotides_lagged']
```

Makes train, validation, and test set.

In [337…
```
X_train= X[:21000]
y_train= y[:21000]
X_val=X[21001:21601]
y_val=y[21001:21601]
X_test=X[21601:24601]
y_test=y[21601:24601]
```

Encodes numbers to lists of zeros and a one that represents the class.

In [338…
```
X_train = to_categorical(X_train)
y_train = to_categorical(y_train)
X_val = to_categorical(X_val)
y_val = to_categorical(y_val)
X_test = to_categorical(X_test)
y_test = to_categorical(y_test)
```

Splits data into threes to represent each codon.

In [339…
```
X_train = [X_train[x:x+3] for x in range(0, len(X_train),3)]
y_train=[y_train[x:x+3] for x in range(0, len(y_train),3)]
X_val = [X_val[x:x+3] for x in range(0, len(X_val),3)]
y_val=[y_val[x:x+3] for x in range(0, len(y_val),3)]
X_test = [X_test[x:x+3] for x in range(0, len(X_test),3)]
y_test=[y_test[x:x+3] for x in range(0, len(y_test),3)]
```

Makes data into arrays.

In [340…
```
X_train=np.array(X_train)
y_train=np.array(y_train)
X_val=np.array(X_val)
y_val=np.array(y_val)
X_test=np.array(X_test)
y_test=np.array(y_test)
```

The model will train on 7000 codons in order to predict the next nucleotide.

In [341…
```
X_train.shape
```

Out[341…
```
(7000, 3, 4)
```

In [342…
```
y_train.shape
```

Out[342…
```
(7000, 3, 4)
```

In [343…
```
X_train
```

Out[343…
```
array([[[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.]],

       [[0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.]],

       [[0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]],

       ...,

       [[0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.]],

       [[0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.]],

       [[0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.]]], dtype=float32)
```

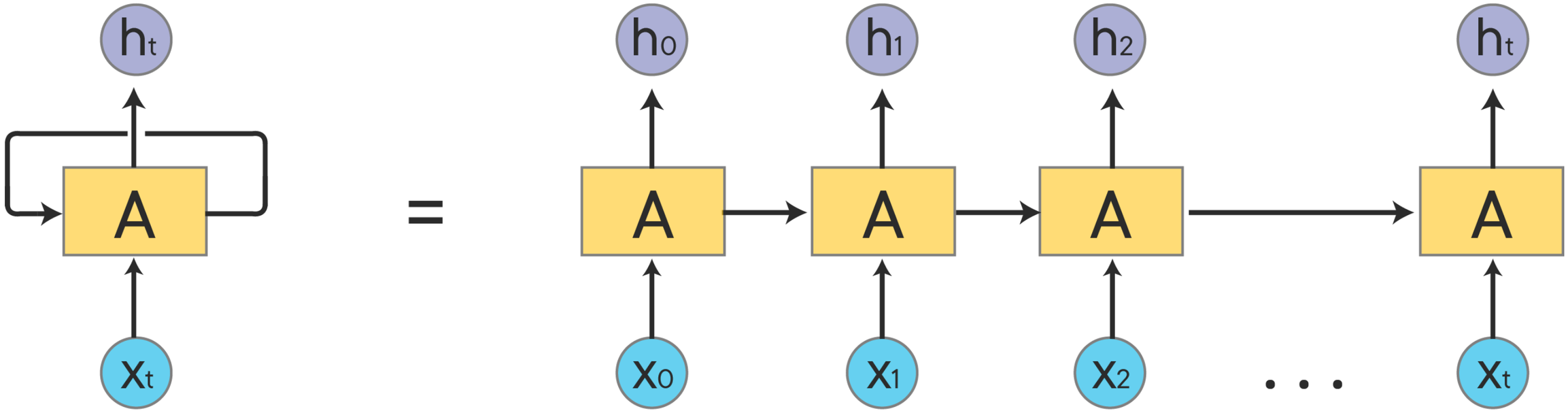
RNN Model

A Recurrent Neural Network is a neural network that passes its output, h, from a layer back into itself as input for the next layer. The model starts at the most recent output, and then works backward to calculate the loss and update the weights at each time step. The hidden state, ht, is a function of the current input, xt, the previous hidden state, ht-1, and the bias, bt.

$$h_t = \sigma(W * x_t + U * h_{t-1} + b_t)$$

In [33]:
```
Image(filename='RNN.png')
```

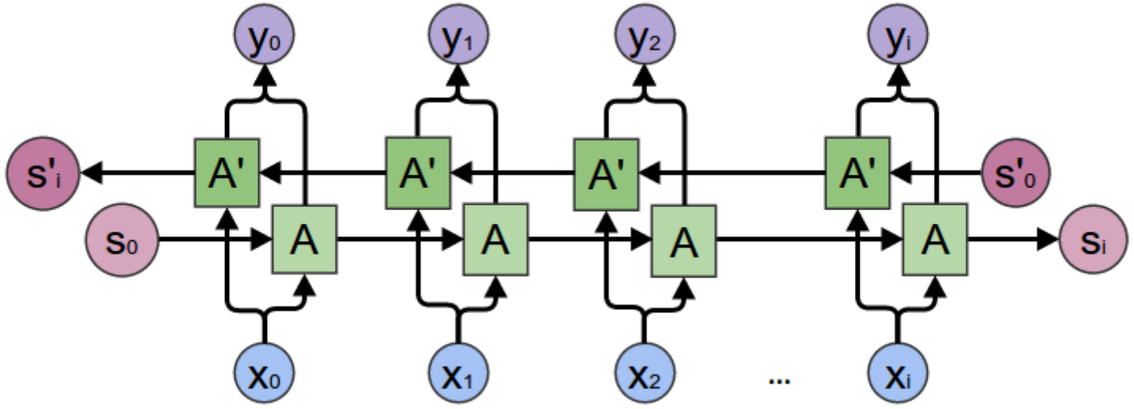
Out[33]:



In a bidirectional RNN, the input sequence is passed forward in normal time order through one network, and then in reverse time order through another network. The outputs of the two networks are then concatenated at each time step. This enables the resulting network to have backward and forward information about the sequence at every time step.

In [40]:
```
Image(filename='bd.png')
```

Out[40]:

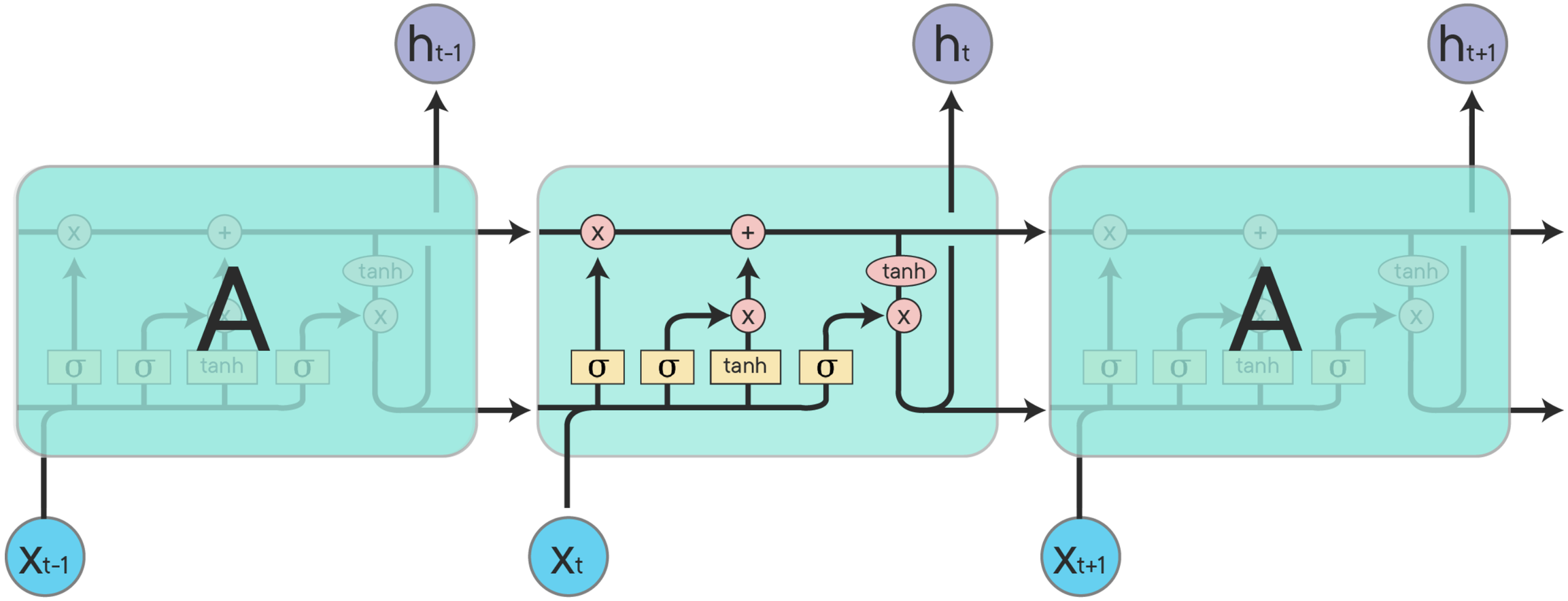


LSTMs are memory cells that use three gates:

- Forget Gate, which determines how much of the current state should be forgotten
- Input Gate, which determines how much should be kept of the cell that was passed
- Output Gate, which determines how much of the current state should be exposed to the next layer in the network

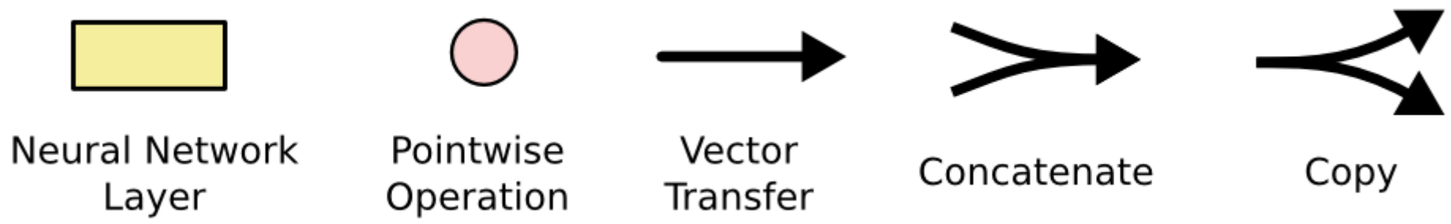
In [34]:
```
Image(filename='LSTM.png')
```

Out[34]:



In [24]:
```
Image(filename='LSTM-notation.png')
```

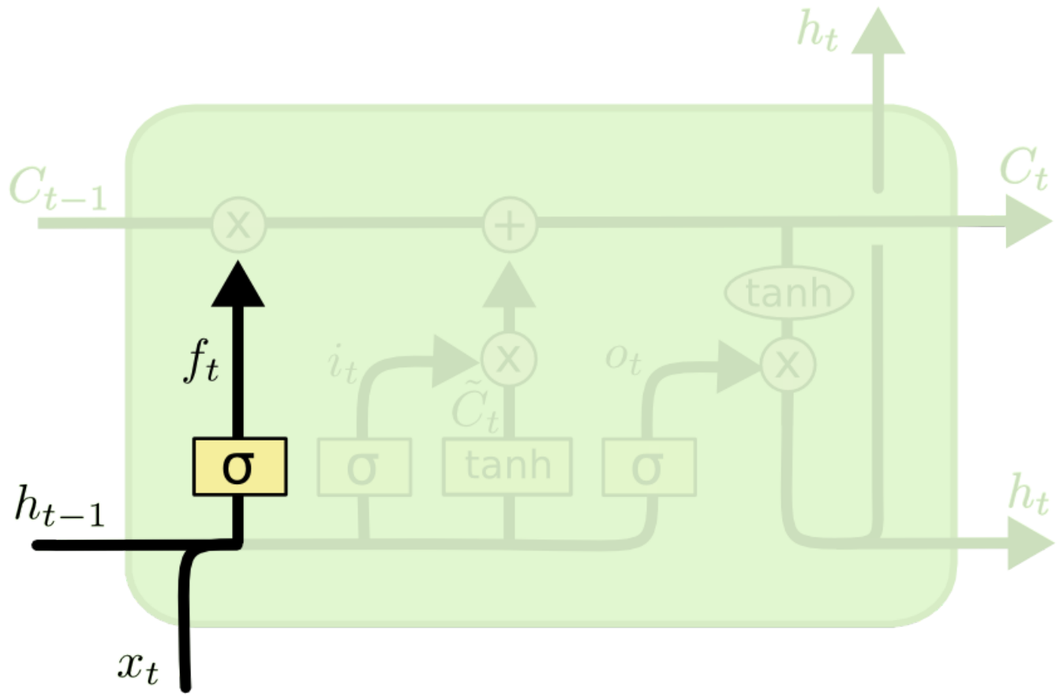
Out[24]:



forget gate activation vector determines how much of the current state should be forgotten with a sigmoid function.

In [27]:
```
Image(filename='LSTM-f.png')
```

Out[27]:

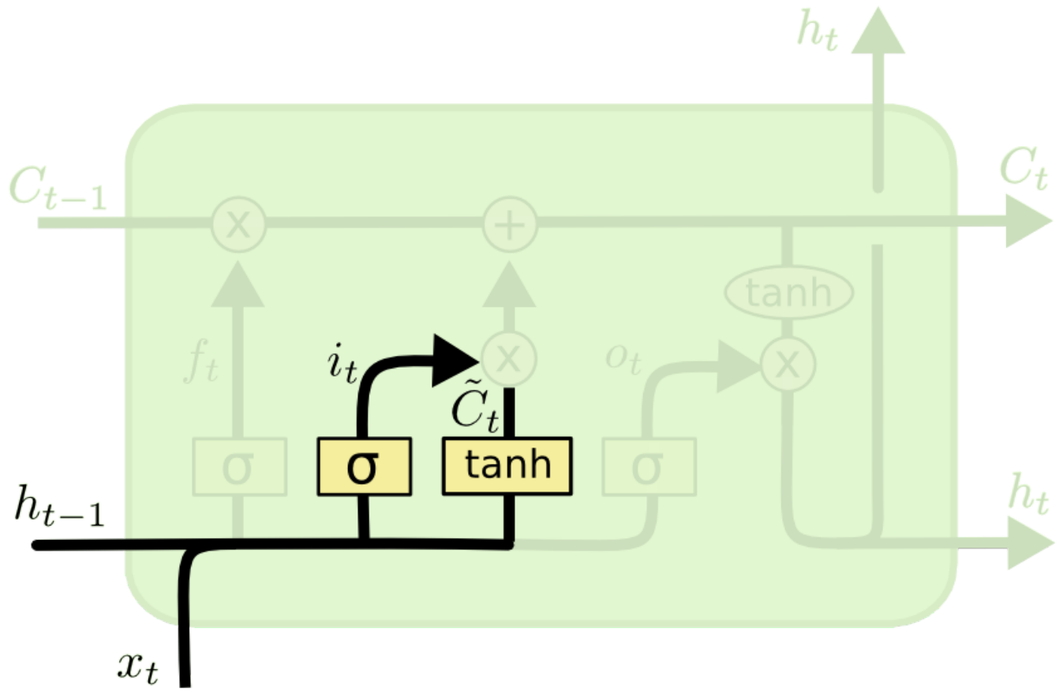


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

New information is stored in the cell state, Ct, in two steps. An input gate layer, sigmoid function, decides which values get updated, and then a tanh layer makes a vector of new candidate values, C̃t, that may be added to the state.

In [28]:
```
Image(filename='LSTM-i.png')
```

Out[28]:



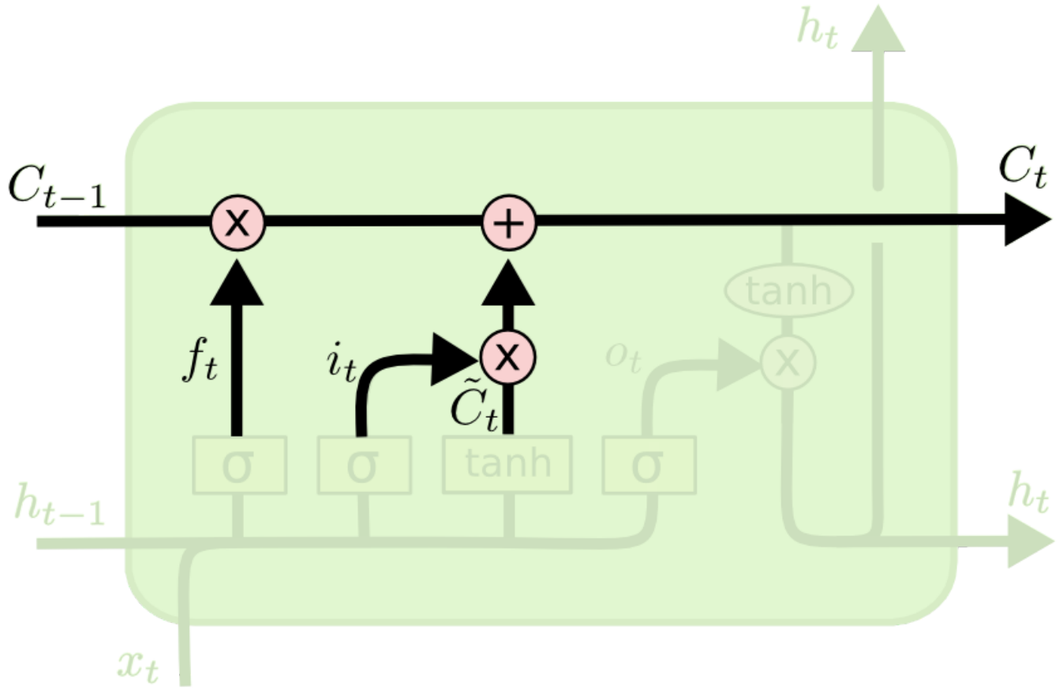
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Update the previous cell state, Ct−1, into the new cell state Ct by multiplying Ct−1 by ft without forgetton elements, and then adding it multiplied by C̃t.

In [29]:
```
Image(filename='LSTM-C.png')
```

Out[29]:

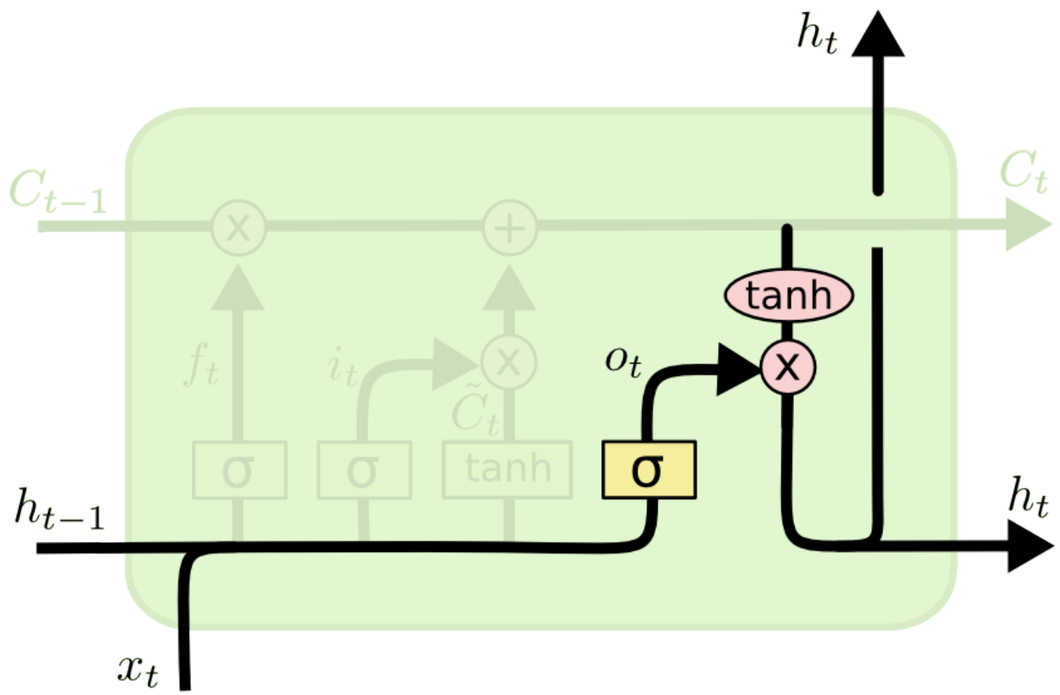


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Determine the output by making a sigmoid layer that decides wich parts of the Ct will be outputed, passing the Ct through tanh to make the values be between −1 and 1, and multiplying the result by the output gate result.

In [30]:
```
Image(filename='LSTM-o.png')
```

Out[30]:



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

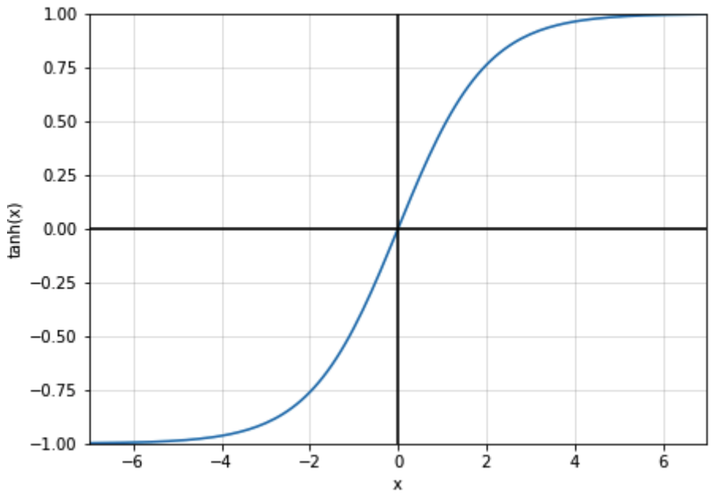
$$h_t = o_t * \tanh\left(C_t\right)$$

Tanh is the hyperbolic tangent function, and is used as an activation function to determine which neurons get passed to the next network layer.

$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

In [21]:
```
Image(filename='tanh.png')
```

Out[21]:

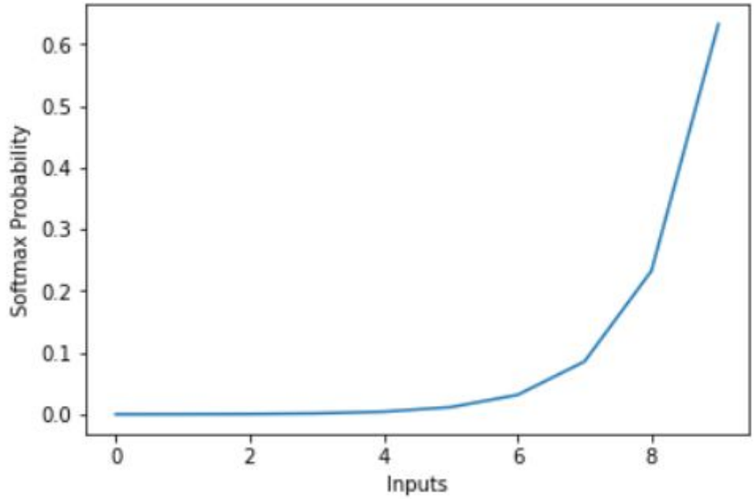


Softmax is an activation function that makes a vector of numbers into a vector of probabilities for each of the 4 classes.

$$softmax = \frac{e^{x_i}}{\sum_j^k e^{x_j}}$$

In [23]:
```
Image(filename='softmax.JPG')
```

Out[23]:



Categorical Crossentropy is a loss function, and is the negative log of the softmax function. Optimize the function using the partial derivative of the loss function with repect to x, and then iterate toward optimal loss with an η stepsize.

$$CCE = -log(\frac{e^{x_i}}{\sum_j^k e^{x_j}})$$

$$x_j^{new} = x_j^{previous} + \eta * \nabla \frac{\partial CCE(x^{previous})}{\partial x^{previous}}$$

In [300…
```
rnn = Sequential()
rnn.add(Bidirectional(LSTM(units=50, activation='tanh', return_sequences=True, input_shape=(3,4))))
rnn.add(Dropout(0.2))
rnn.add(Dense(units=4, activation='softmax'))
rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', precision, recall, specificity, f1])
```

In [301…
```
history=rnn.fit(X_train, y_train, epochs = 10, batch_size = 20, verbose=1, validation_data=(X_val, y_val))
```

```
Epoch 1/10
350/350 [==============================] - 1s 4ms/step - loss: 0.8064 - accuracy: 0.6717 - precision: 0.7267 - recall: 0.4583 - specificity: 0.9735 - f1: 0.5379 - val_loss: 0.4104 - val
_accuracy: 0.8733 - val_precision: 0.9554 - val_recall: 0.7500 - val_specificity: 0.9883 - val_f1: 0.8401
Epoch 2/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4148 - accuracy: 0.8164 - precision: 0.9392 - recall: 0.7271 - specificity: 0.9841 - f1: 0.8193 - val_loss: 0.3787 - val
_accuracy: 0.8767 - val_precision: 0.9521 - val_recall: 0.7633 - val_specificity: 0.9872 - val_f1: 0.8470
Epoch 3/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4078 - accuracy: 0.8154 - precision: 0.9393 - recall: 0.7318 - specificity: 0.9839 - f1: 0.8222 - val_loss: 0.3763 - val
_accuracy: 0.8767 - val_precision: 0.9537 - val_recall: 0.7583 - val_specificity: 0.9878 - val_f1: 0.8445
Epoch 4/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4059 - accuracy: 0.8179 - precision: 0.9409 - recall: 0.7335 - specificity: 0.9844 - f1: 0.8238 - val_loss: 0.3808 - val
_accuracy: 0.8750 - val_precision: 0.9553 - val_recall: 0.7483 - val_specificity: 0.9883 - val_f1: 0.8390
Epoch 5/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4045 - accuracy: 0.8172 - precision: 0.9409 - recall: 0.7320 - specificity: 0.9844 - f1: 0.8229 - val_loss: 0.3835 - val
_accuracy: 0.8633 - val_precision: 0.9536 - val_recall: 0.7567 - val_specificity: 0.9878 - val_f1: 0.8434
Epoch 6/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4042 - accuracy: 0.8179 - precision: 0.9406 - recall: 0.7340 - specificity: 0.9843 - f1: 0.8241 - val_loss: 0.3755 - val
_accuracy: 0.8733 - val_precision: 0.9518 - val_recall: 0.7600 - val_specificity: 0.9872 - val_f1: 0.8448
Epoch 7/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4045 - accuracy: 0.8174 - precision: 0.9413 - recall: 0.7312 - specificity: 0.9845 - f1: 0.8226 - val_loss: 0.3719 - val
_accuracy: 0.8750 - val_precision: 0.9444 - val_recall: 0.7683 - val_specificity: 0.9850 - val_f1: 0.8470
Epoch 8/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4041 - accuracy: 0.8176 - precision: 0.9390 - recall: 0.7367 - specificity: 0.9838 - f1: 0.8252 - val_loss: 0.3775 - val
_accuracy: 0.8733 - val_precision: 0.9518 - val_recall: 0.7600 - val_specificity: 0.9872 - val_f1: 0.8448
Epoch 9/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4028 - accuracy: 0.8175 - precision: 0.9426 - recall: 0.7331 - specificity: 0.9849 - f1: 0.8243 - val_loss: 0.3718 - val
_accuracy: 0.8733 - val_precision: 0.9496 - val_recall: 0.7767 - val_specificity: 0.9861 - val_f1: 0.8542
Epoch 10/10
350/350 [==============================] - 1s 2ms/step - loss: 0.4019 - accuracy: 0.8201 - precision: 0.9392 - recall: 0.7355 - specificity: 0.9839 - f1: 0.8245 - val_loss: 0.3786 - val
_accuracy: 0.8567 - val_precision: 0.9504 - val_recall: 0.7533 - val_specificity: 0.9867 - val_f1: 0.8401
```

In [302…
```
rnn.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_3 (Bidirection (20, 3, 100)              22000
_________________________________________________________________
dropout_6 (Dropout)          (20, 3, 100)              0
_________________________________________________________________
dense_11 (Dense)             (20, 3, 4)                404
=================================================================
Total params: 22,404
Trainable params: 22,404
Non-trainable params: 0
_________________________________________________________________
```

In [303…
```
predictions=rnn.predict(X_train)
```

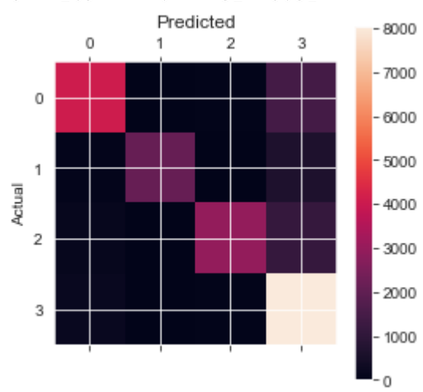
Make encoded actual values and predicted probabilities into numbers.

In [304…
```
actual=[]
for i in y_train:
    for l in i:
        k_max=0
        e_max=0
        for e,k in enumerate(l):
            if k>k_max:
                k_max=k
                e_max=e
        actual.append(e_max)
```

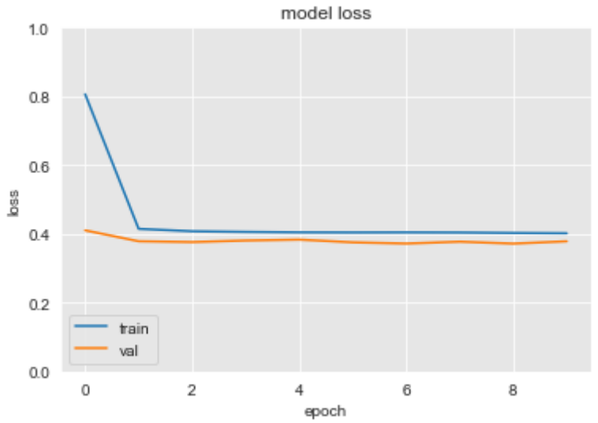
In [305…
```
preds=[]
for i in predictions:
    for l in i:
        k_max=0
        e_max=0
        for e,k in enumerate(l):
            if k>k_max:
                k_max=k
                e_max=e
        preds.append(e_max)
```

In [306…
```
con_mat(actual,preds)
```

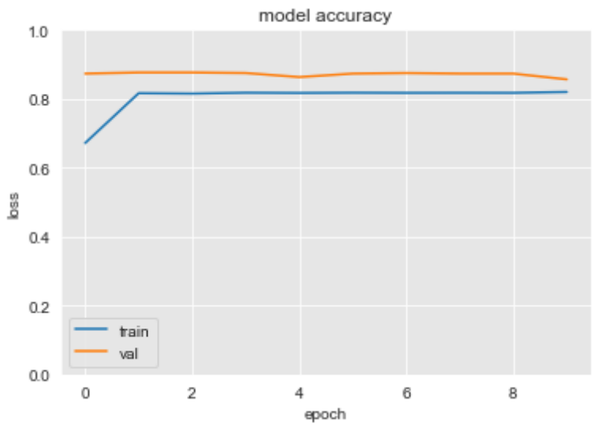
```
Confusion Matrix
----------------
      0     1     2     3
0  4085     0    37  1464
1    85  2132    12   628
2   136     0  2976  1153
3   209     0    52  8031
```



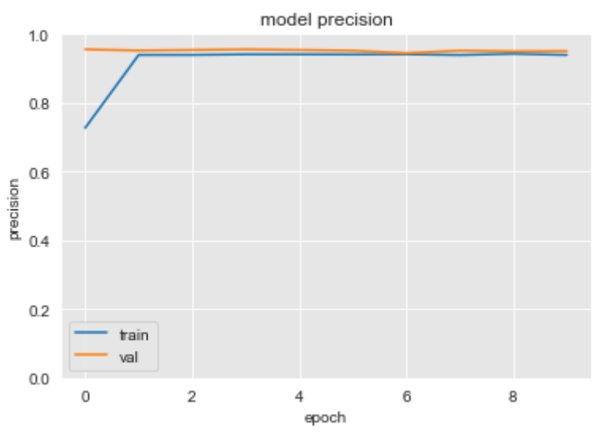
In [307…
```
neural_network_metrics(history.history['loss'],history.history['val_loss'],'model loss','loss')
```



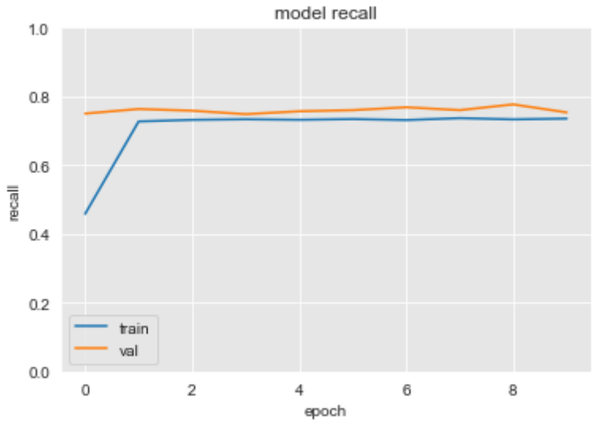
In [308…
```
neural_network_metrics(history.history['accuracy'],history.history['val_accuracy'],'model accuracy','loss')
```



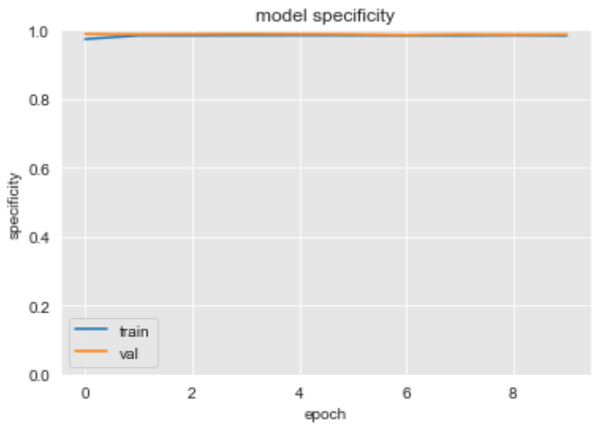
In [309…
```
neural_network_metrics(history.history['precision'],history.history['val_precision'],'model precision','precision')
```



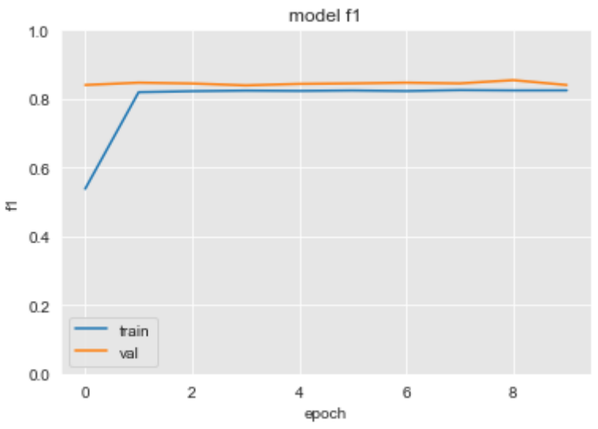
In [310…
```
neural_network_metrics(history.history['recall'],history.history['val_recall'],'model recall','recall')
```



In [311…
```
neural_network_metrics(history.history['specificity'],history.history['val_specificity'],'model specificity','specificity')
```



In [312…
```
neural_network_metrics(history.history['f1'],history.history['val_f1'],'model f1','f1')
```



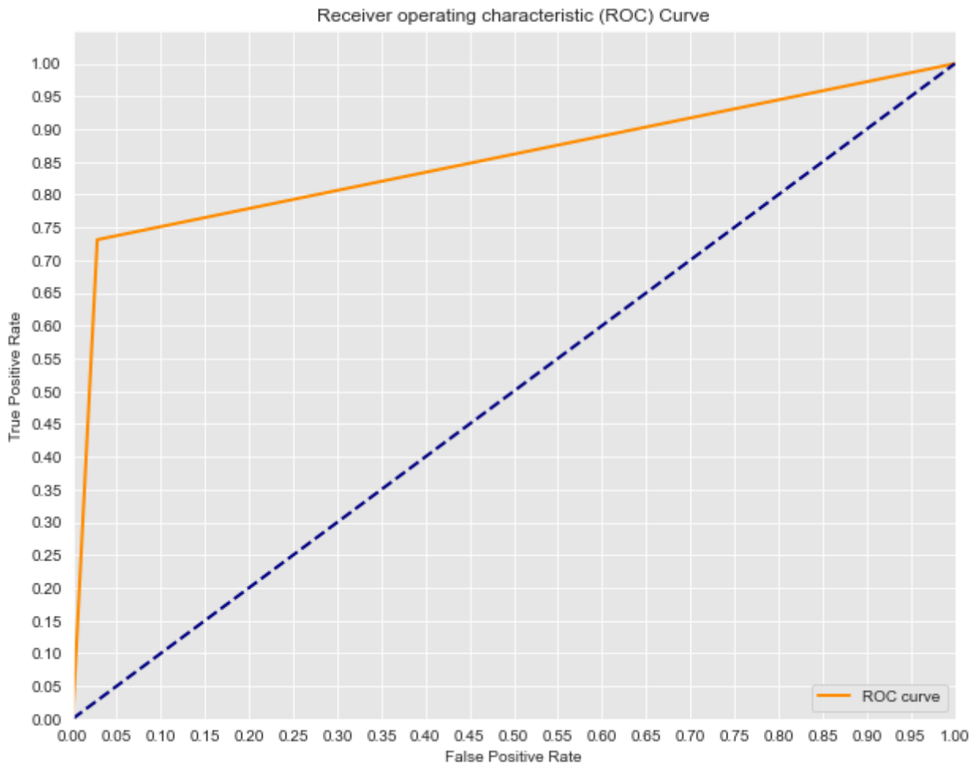
In [313…
```
actual_class0=np.array(actual)==0
actual_class1=np.array(actual)==1
```

```
actual_class2=np.array(actual)==2
actual_class3=np.array(actual)==3
predictions_class0=np.array(preds)==0
predictions_class1=np.array(preds)==1
predictions_class2=np.array(preds)==2
predictions_class3=np.array(preds)==3
a=[actual_class0,actual_class1,actual_class2,actual_class3]
p=[predictions_class0,predictions_class1,predictions_class2,predictions_class3]
```

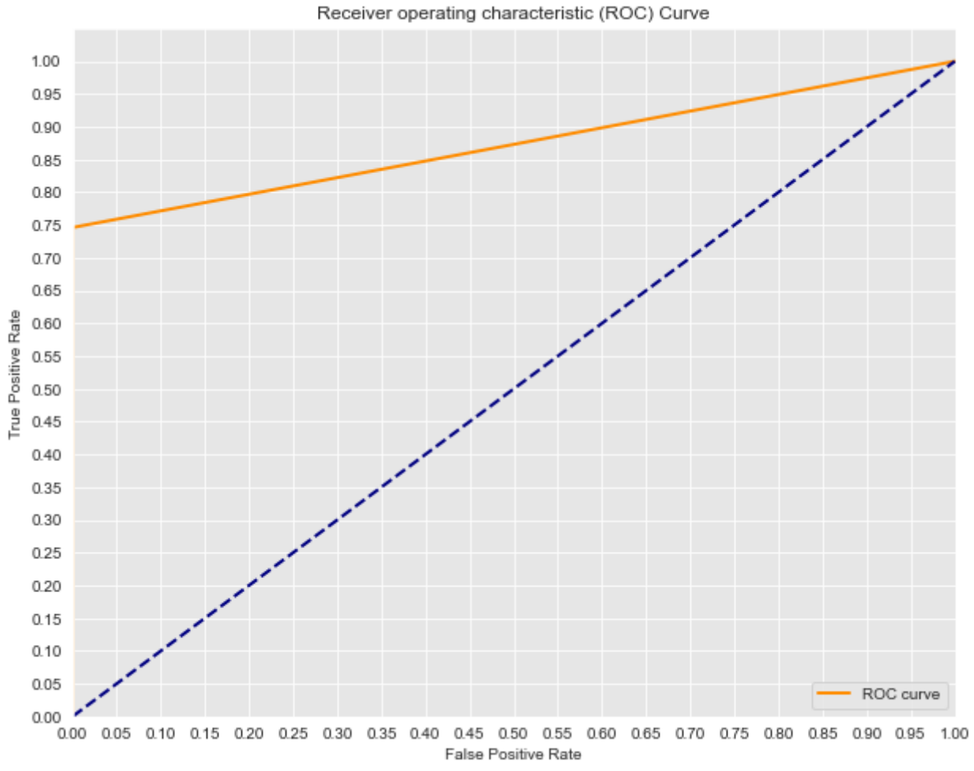
In [314…

```
c=0
for i,l in zip(a,p):
    print(f'Class: {c}')
    roc(i,l)
    c+=1
```

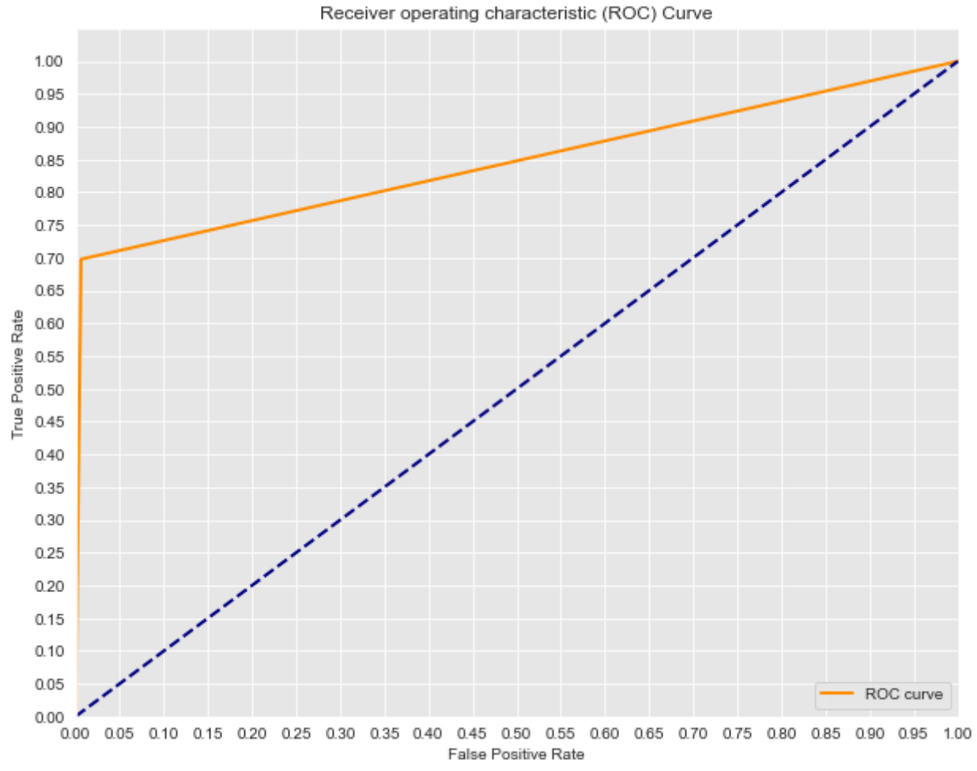
```
Class: 0
AUC: 0.8516978998683942
```



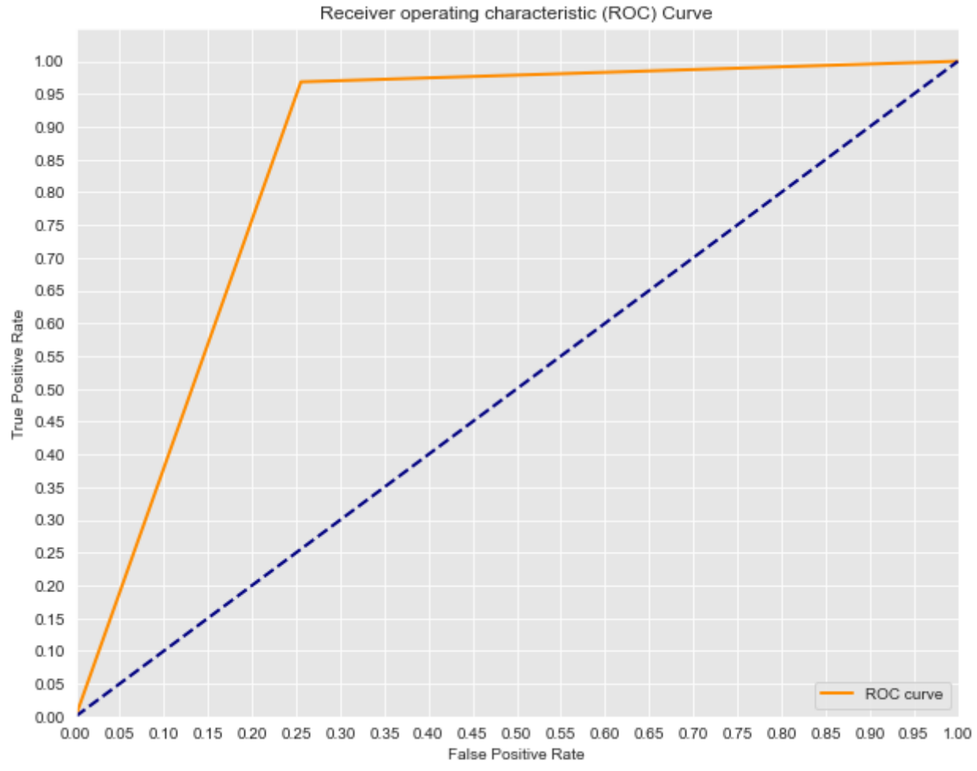

```
Class: 1
AUC: 0.8731186559327966
```




```
Class: 2
AUC: 0.84586865597825
```




```
Class: 3
AUC: 0.8565864592059335
```



Cross validate model.

In [319…

```
# K-fold Cross Validation model evaluation
kfold = KFold(n_splits=5, shuffle=True, random_state=7)
fold_no = 1
cvscores=[]
for train, val in kfold.split(X_train,y_train):

    # create model
    rnn = Sequential()
    rnn.add(Bidirectional(LSTM(units=50, activation='tanh', return_sequences=True, input_shape=(3,4))))
    rnn.add(Dropout(0.2))
    rnn.add(Dense(units=4, activation='softmax'))
    rnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy', precision, recall, specificity, f1])



    # Generate a print
    print('------------------------------------------------------------------------')
    print(f'Training for fold {fold_no} ...')

    # Fit the model
    rnn.fit(X_train[train], y_train[train], epochs=10, batch_size=20, verbose=1)
    # evaluate the model
    scores = rnn.evaluate(X_train[val], y_train[val], verbose=1)
    cvscores.append(scores[1] * 100)
    fold_no+=1
print(f'Mean Accuracy CV Score:{np.mean(cvscores)}')
```

```
------------------------------------------------------------------------
Training for fold 1 ...
Epoch 1/10
280/280 [==============================] - 1s 2ms/step - loss: 0.9063 - accuracy: 0.6340 - precision: 0.6356 - recall: 0.3812 - specificity: 0.9723 - f1: 0.4611
Epoch 2/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4238 - accuracy: 0.8173 - precision: 0.9421 - recall: 0.7189 - specificity: 0.9850 - f1: 0.8149
Epoch 3/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4099 - accuracy: 0.8162 - precision: 0.9444 - recall: 0.7242 - specificity: 0.9855 - f1: 0.8192
Epoch 4/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4085 - accuracy: 0.8174 - precision: 0.9429 - recall: 0.7270 - specificity: 0.9850 - f1: 0.8203
Epoch 5/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4064 - accuracy: 0.8179 - precision: 0.9433 - recall: 0.7280 - specificity: 0.9852 - f1: 0.8212
Epoch 6/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4068 - accuracy: 0.8182 - precision: 0.9413 - recall: 0.7277 - specificity: 0.9846 - f1: 0.8202
Epoch 7/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4068 - accuracy: 0.8164 - precision: 0.9417 - recall: 0.7274 - specificity: 0.9847 - f1: 0.8203
Epoch 8/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4045 - accuracy: 0.8170 - precision: 0.9451 - recall: 0.7266 - specificity: 0.9858 - f1: 0.8211
Epoch 9/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4043 - accuracy: 0.8168 - precision: 0.9442 - recall: 0.7287 - specificity: 0.9855 - f1: 0.8221
Epoch 10/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4047 - accuracy: 0.8191 - precision: 0.9417 - recall: 0.7276 - specificity: 0.9847 - f1: 0.8205
44/44 [==============================] - 0s 1ms/step - loss: 0.3975 - accuracy: 0.8238 - precision: 0.9353 - recall: 0.7440 - specificity: 0.9827 - f1: 0.8277
------------------------------------------------------------------------
Training for fold 2 ...
Epoch 1/10
280/280 [==============================] - 1s 3ms/step - loss: 0.9041 - accuracy: 0.6203 - precision: 0.6431 - recall: 0.3765 - specificity: 0.9746 - f1: 0.4537
Epoch 2/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4234 - accuracy: 0.8167 - precision: 0.9357 - recall: 0.7318 - specificity: 0.9830 - f1: 0.8208
Epoch 3/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4090 - accuracy: 0.8161 - precision: 0.9412 - recall: 0.7323 - specificity: 0.9845 - f1: 0.8232
Epoch 4/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4060 - accuracy: 0.8164 - precision: 0.9430 - recall: 0.7303 - specificity: 0.9849 - f1: 0.8225
Epoch 5/10
280/280 [==============================] - 1s 3ms/step - loss: 0.4046 - accuracy: 0.8202 - precision: 0.9426 - recall: 0.7318 - specificity: 0.9849 - f1: 0.8235
Epoch 6/10
280/280 [==============================] - 1s 3ms/step - loss: 0.4052 - accuracy: 0.8184 - precision: 0.9403 - recall: 0.7330 - specificity: 0.9842 - f1: 0.8233
Epoch 7/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4050 - accuracy: 0.8182 - precision: 0.9400 - recall: 0.7324 - specificity: 0.9842 - f1: 0.8228
Epoch 8/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4043 - accuracy: 0.8158 - precision: 0.9391 - recall: 0.7336 - specificity: 0.9838 - f1: 0.8230
Epoch 9/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4043 - accuracy: 0.8157 - precision: 0.9422 - recall: 0.7322 - specificity: 0.9848 - f1: 0.8236
Epoch 10/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4031 - accuracy: 0.8177 - precision: 0.9426 - recall: 0.7324 - specificity: 0.9849 - f1: 0.8239
44/44 [==============================] - 0s 1ms/step - loss: 0.4044 - accuracy: 0.8200 - precision: 0.9255 - recall: 0.7435 - specificity: 0.9800 - f1: 0.8239
------------------------------------------------------------------------
Training for fold 3 ...
Epoch 1/10
280/280 [==============================] - 1s 2ms/step - loss: 0.9023 - accuracy: 0.6368 - precision: 0.6414 - recall: 0.3885 - specificity: 0.9731 - f1: 0.4664
Epoch 2/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4220 - accuracy: 0.8170 - precision: 0.9483 - recall: 0.7222 - specificity: 0.9866 - f1: 0.8194
Epoch 3/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4099 - accuracy: 0.8140 - precision: 0.9442 - recall: 0.7251 - specificity: 0.9855 - f1: 0.8199
Epoch 4/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4068 - accuracy: 0.8186 - precision: 0.9420 - recall: 0.7292 - specificity: 0.9847 - f1: 0.8215
Epoch 5/10
280/280 [==============================] - 1s 3ms/step - loss: 0.4059 - accuracy: 0.8161 - precision: 0.9445 - recall: 0.7292 - specificity: 0.9855 - f1: 0.8225
Epoch 6/10
280/280 [==============================] - 1s 3ms/step - loss: 0.4053 - accuracy: 0.8180 - precision: 0.9404 - recall: 0.7292 - specificity: 0.9843 - f1: 0.8208
Epoch 7/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4051 - accuracy: 0.8171 - precision: 0.9438 - recall: 0.7289 - specificity: 0.9853 - f1: 0.8221
Epoch 8/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4041 - accuracy: 0.8174 - precision: 0.9424 - recall: 0.7324 - specificity: 0.9848 - f1: 0.8238
Epoch 9/10
```

```
280/280 [==============================] - 1s 2ms/step - loss: 0.4050 - accuracy: 0.8150 - precision: 0.9434 - recall: 0.7305 - specificity: 0.9852 - f1: 0.8230
Epoch 10/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4040 - accuracy: 0.8174 - precision: 0.9414 - recall: 0.7297 - specificity: 0.9847 - f1: 0.8218
44/44 [==============================] - 0s 1ms/step - loss: 0.3986 - accuracy: 0.8243 - precision: 0.9523 - recall: 0.7261 - specificity: 0.9876 - f1: 0.8230
------------------------------------------------------------------------
Training for fold 4 ...
Epoch 1/10
280/280 [==============================] - 1s 2ms/step - loss: 0.8990 - accuracy: 0.6374 - precision: 0.6479 - recall: 0.3877 - specificity: 0.9757 - f1: 0.4607
Epoch 2/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4263 - accuracy: 0.8179 - precision: 0.9328 - recall: 0.7314 - specificity: 0.9821 - f1: 0.8194
Epoch 3/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4089 - accuracy: 0.8174 - precision: 0.9408 - recall: 0.7342 - specificity: 0.9844 - f1: 0.8242
Epoch 4/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4073 - accuracy: 0.8185 - precision: 0.9371 - recall: 0.7349 - specificity: 0.9832 - f1: 0.8232
Epoch 5/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4047 - accuracy: 0.8188 - precision: 0.9389 - recall: 0.7338 - specificity: 0.9838 - f1: 0.8233
Epoch 6/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4044 - accuracy: 0.8199 - precision: 0.9397 - recall: 0.7320 - specificity: 0.9841 - f1: 0.8224
Epoch 7/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4038 - accuracy: 0.8184 - precision: 0.9403 - recall: 0.7358 - specificity: 0.9842 - f1: 0.8251
Epoch 8/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4040 - accuracy: 0.8180 - precision: 0.9403 - recall: 0.7346 - specificity: 0.9842 - f1: 0.8244
Epoch 9/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4021 - accuracy: 0.8195 - precision: 0.9431 - recall: 0.7340 - specificity: 0.9850 - f1: 0.8250
Epoch 10/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4023 - accuracy: 0.8205 - precision: 0.9399 - recall: 0.7346 - specificity: 0.9841 - f1: 0.8242
44/44 [==============================] - 0s 1ms/step - loss: 0.4021 - accuracy: 0.8138 - precision: 0.9476 - recall: 0.7175 - specificity: 0.9867 - f1: 0.8162
------------------------------------------------------------------------
Training for fold 5 ...
Epoch 1/10
280/280 [==============================] - 1s 2ms/step - loss: 0.9391 - accuracy: 0.6154 - precision: 0.6253 - recall: 0.3599 - specificity: 0.9701 - f1: 0.4314
Epoch 2/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4284 - accuracy: 0.8169 - precision: 0.9369 - recall: 0.7278 - specificity: 0.9834 - f1: 0.8187
Epoch 3/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4103 - accuracy: 0.8170 - precision: 0.9421 - recall: 0.7270 - specificity: 0.9848 - f1: 0.8202
Epoch 4/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4052 - accuracy: 0.8177 - precision: 0.9385 - recall: 0.7315 - specificity: 0.9838 - f1: 0.8217
Epoch 5/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4048 - accuracy: 0.8158 - precision: 0.9389 - recall: 0.7310 - specificity: 0.9839 - f1: 0.8215
Epoch 6/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4037 - accuracy: 0.8186 - precision: 0.9405 - recall: 0.7317 - specificity: 0.9842 - f1: 0.8225
Epoch 7/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4035 - accuracy: 0.8168 - precision: 0.9416 - recall: 0.7318 - specificity: 0.9846 - f1: 0.8231
Epoch 8/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4038 - accuracy: 0.8161 - precision: 0.9401 - recall: 0.7319 - specificity: 0.9842 - f1: 0.8226
Epoch 9/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4035 - accuracy: 0.8182 - precision: 0.9424 - recall: 0.7321 - specificity: 0.9849 - f1: 0.8236
Epoch 10/10
280/280 [==============================] - 1s 2ms/step - loss: 0.4019 - accuracy: 0.8191 - precision: 0.9408 - recall: 0.7343 - specificity: 0.9843 - f1: 0.8243
44/44 [==============================] - 0s 1ms/step - loss: 0.4042 - accuracy: 0.8190 - precision: 0.9504 - recall: 0.7285 - specificity: 0.9870 - f1: 0.8235
Mean Accuracy CV Score:82.01904654502869
```

Test model.

In [315…
```
scores = rnn.evaluate(X_test, y_test, batch_size=20, verbose=1)
```

```
50/50 [==============================] - 0s 1ms/step - loss: 0.3833 - accuracy: 0.8433 - precision: 0.9428 - recall: 0.7490 - specificity: 0.9847 - f1: 0.8341
```

In [349…
```
predictions=rnn.predict(X_test)
```

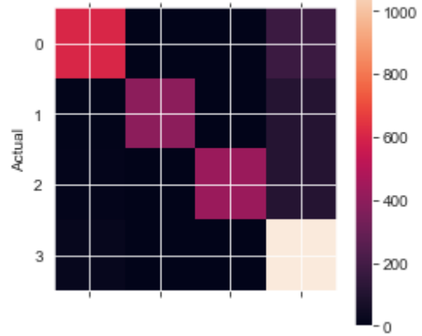
In [350…
```
actual=[]
for i in y_test:
    for l in i:
        k_max=0
```

```
        e_max=0
        for e,k in enumerate(l):
            if k>k_max:
                k_max=k
                e_max=e
        actual.append(e_max)
```

In [351…
```
preds=[]
for i in predictions:
    for l in i:
        k_max=0
        e_max=0
        for e,k in enumerate(l):
            if k>k_max:
                k_max=k
                e_max=e
        preds.append(e_max)
```

In [352…
```
con_mat(actual,preds)
```

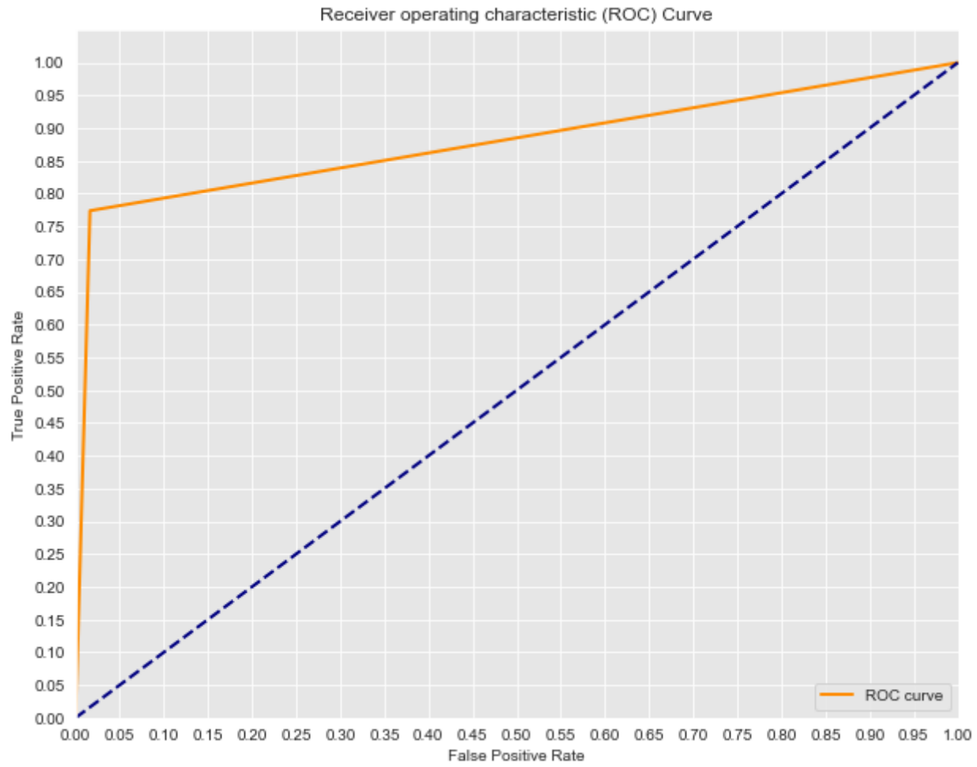
```
Confusion Matrix
----------------
      0    1    2     3
0   606    0    0   177
1     8  398    0   112
2    10    0  433   113
3    18    0    3  1122
```



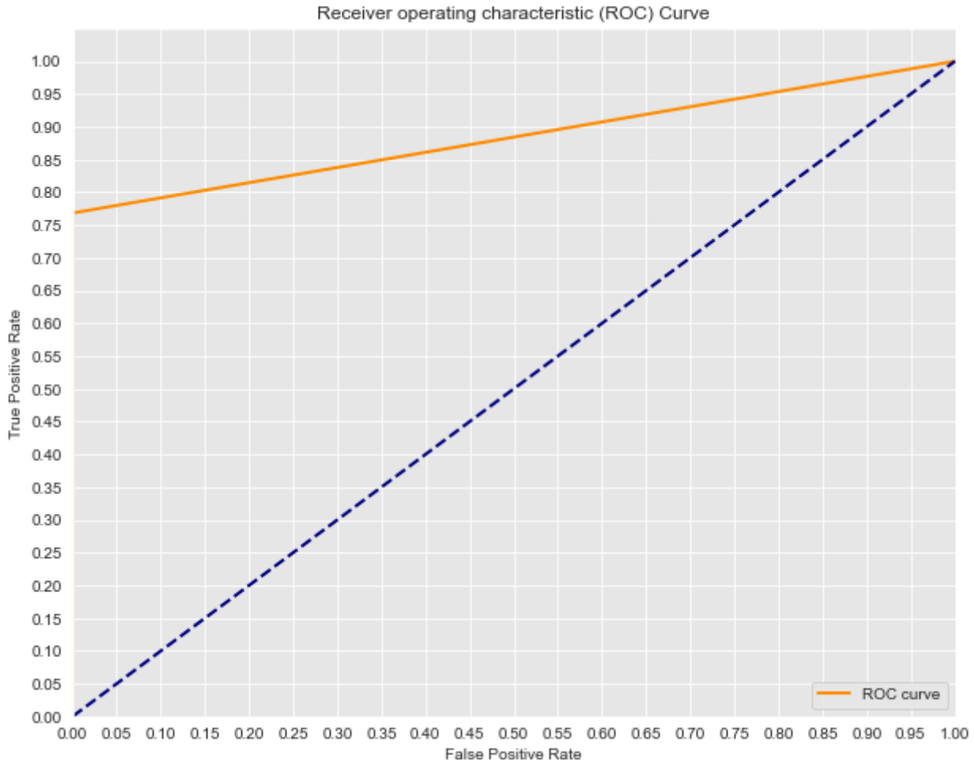
In [353…
```
actual_class0=np.array(actual)==0
actual_class1=np.array(actual)==1
actual_class2=np.array(actual)==2
actual_class3=np.array(actual)==3
predictions_class0=np.array(preds)==0
predictions_class1=np.array(preds)==1
predictions_class2=np.array(preds)==2
predictions_class3=np.array(preds)==3
a=[actual_class0,actual_class1,actual_class2,actual_class3]
p=[predictions_class0,predictions_class1,predictions_class2,predictions_class3]
```

In [354…
```
c=0
for i,l in zip(a,p):
    print(f'Class: {c}')
    roc(i,l)
    c+=1
```

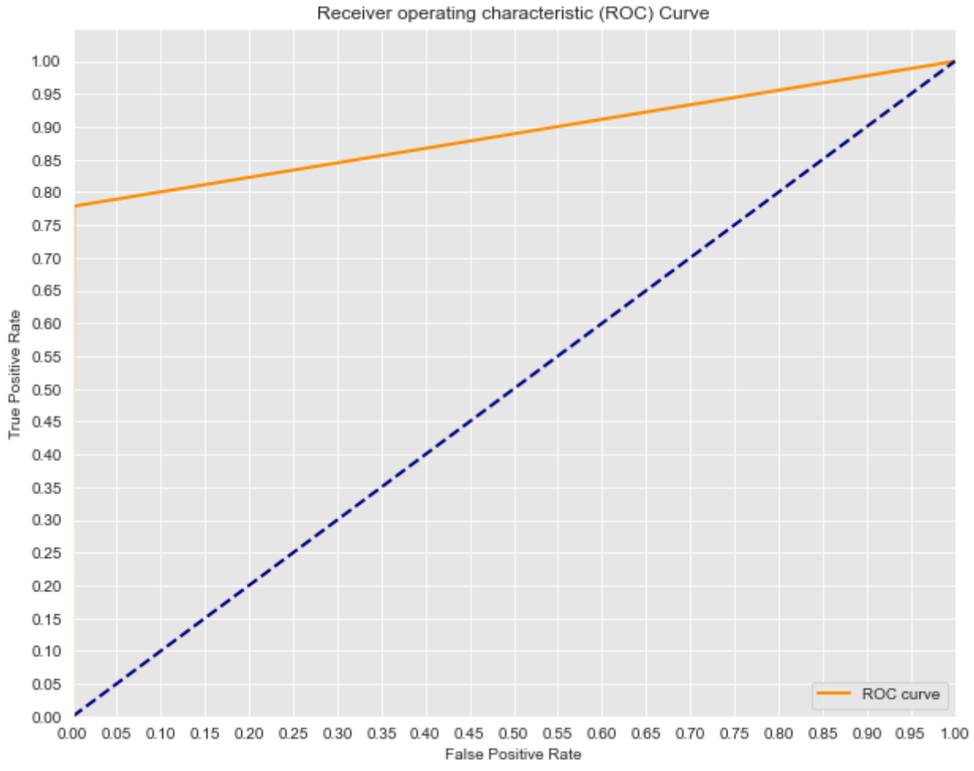
```
Class: 0
AUC: 0.8788541002390099
```



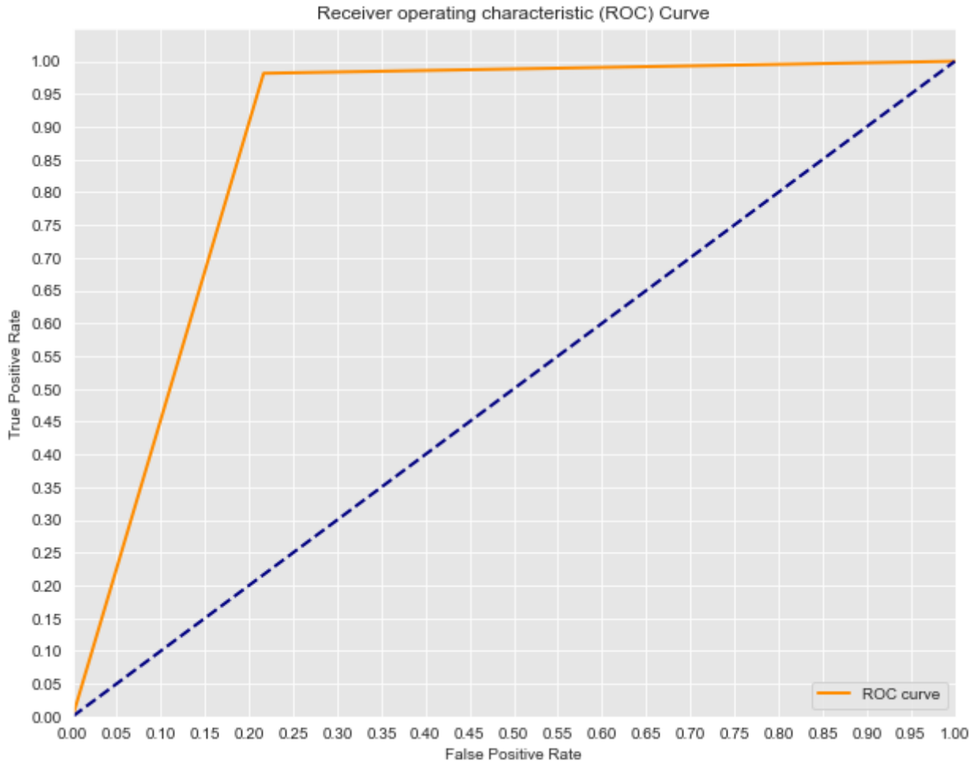

```
Class: 1
AUC: 0.8841698841698842
```




```
Class: 2
AUC: 0.8887747412544597
```




```
Class: 3
AUC: 0.8825745529789389
```



Conclusion

The cnn is able to determine whether a CT-scan has sars-cov-2 with 86.6% accuracy and the rnn is able to predict the next nucleotide of the sars-cov-2 genome with 84.3% accuracy.

In [ ]: