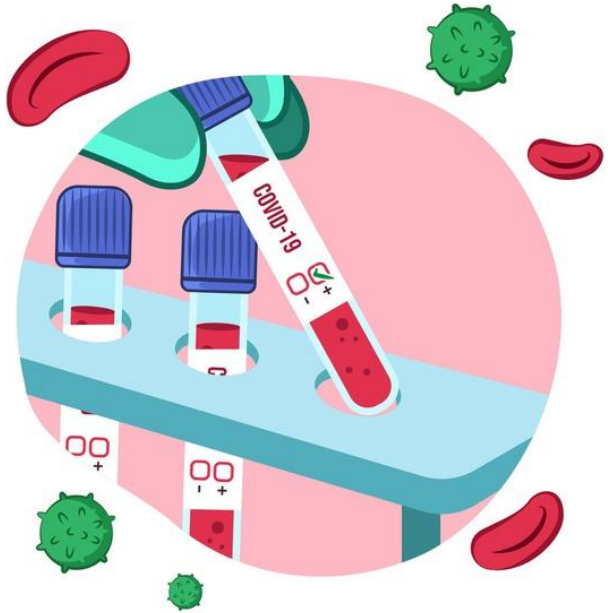
Deteksi Covid-19 pada Gambar CXR

Kelompok 5

Anggota Kelompok 5

- Naila Fadhilah F. 13216002
- Maurizfa 13216008
- M. Naufal Thariq 13216010
- Mikael Wahyu D. 13216014
- Robertsen Putra S. 13216024
- Gabrielle Shay A. 13216080
- Nicholas Nathanael 13216093
- Resfyanti N. A. 18317016



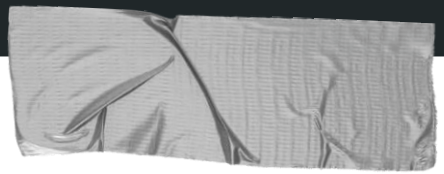
OUTLINE

01 LATAR BELAKANG

02 DATASET

03 METODE PEMBELAJARAN DEEP LEARNING

04 HASIL & ANALISIS



1. Latar Belakang

COVID-19 merupakan pandemi dengan waktu penyebaran yang sangat cepat dan gejala umum mirip dengan penyakit coronavirus lainnya. Kunci dari pandemi ini adalah **pendeteksian dini dan akurat** agar penyebarannya dapat ditekan. Salah satu metode pendeteksian COVID-19 selain PCR dan *rapid test* adalah Deteksi COVID-19 pada gambar CXR.

- ➔ **Adanya *false negative* pendeteksian dengan PCR pada awal infeksi**
  Gambar CXR (Chest X-ray) dapat menjadi komplementer untuk hasil PCR di awal sehingga dapat mengurangi *false negative* sebanyak-banyaknya.
- ➔ **Proses PCR lebih akurat, namun membutuhkan waktu yang lama dan belum banyak lab di Indonesia yang *capable*.**
  Gambar Radiologi cenderung cepat dan bisa dilakukan dimanapun.

Why X-Ray?

COVID-19 merupakan infeksi virus yang bisa dialami oleh siapapun. Rontgen X-ray memiliki keuntungan **harga yang murah dan cepat**; oleh karena itu, X-ray lebih mudah diakses oleh penyedia layanan kesehatan yang bekerja di daerah yang kecil dan/atau terpencil. **Model ini adalah sistem prototipe dan bukan untuk penggunaan medis dan belum dapat digunakan untuk diagnosis.**

Source: https://github.com/aildnont/covid-cxr

Tip

Setiap wawasan yang didapatkan dari pemodelan algoritma yang berhasil akan sangat berharga untuk mempelajari cara penanganan COVID-19.

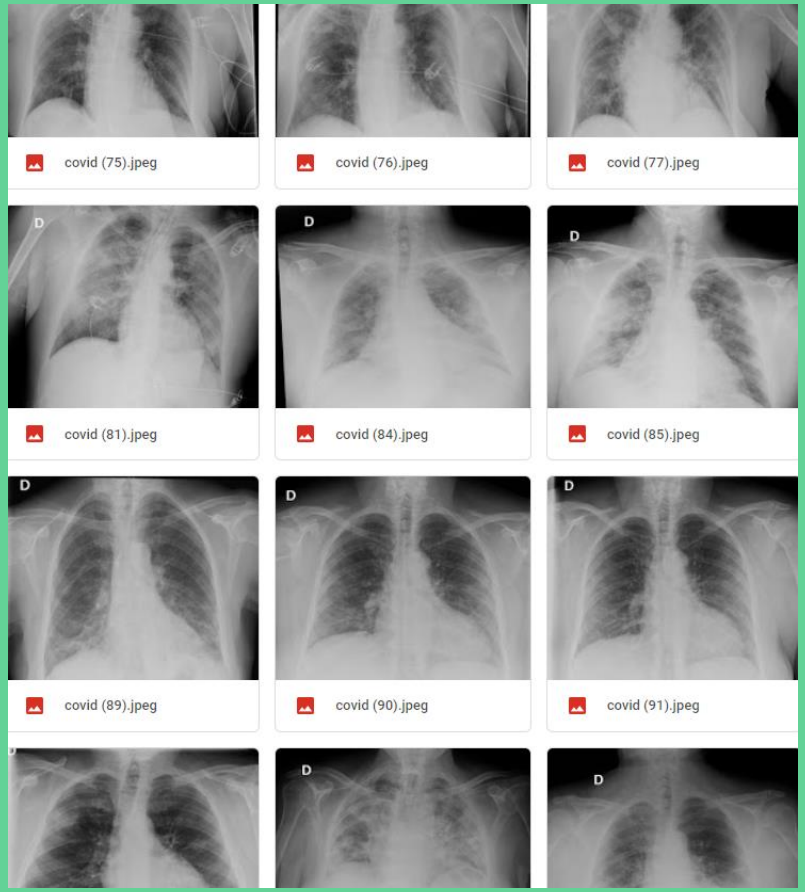
2. Dataset

Dataset yang digunakan pada tugas ini berjumlah sebanyak 266 buah gambar CXR, dengan keterangan:

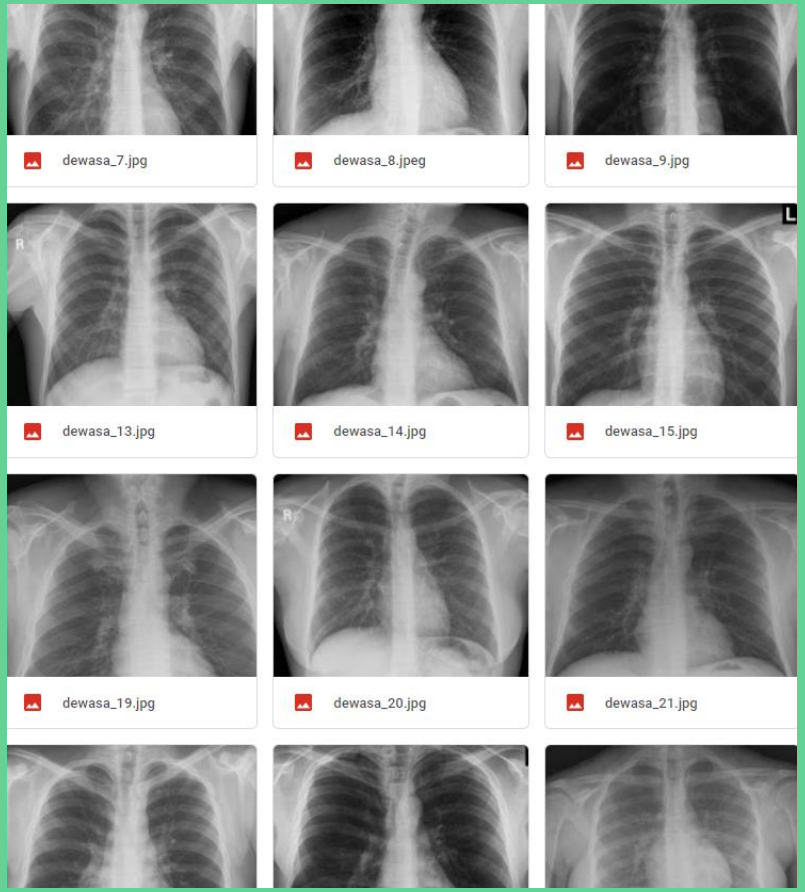
- ➔ **160 data thorax pasien COVID-19**
- ➔ **106 data thorax normal**

Gambar tersebut didapatkan dengan mengunduh dan memilah secara manual gambar thorax dengan penampang yang sama dari sumber-sumber data COVID yang telah disediakan (Dilampirkan pada daftar pustaka).

covid



normal



3. Deskripsi Metode Pembelajaran

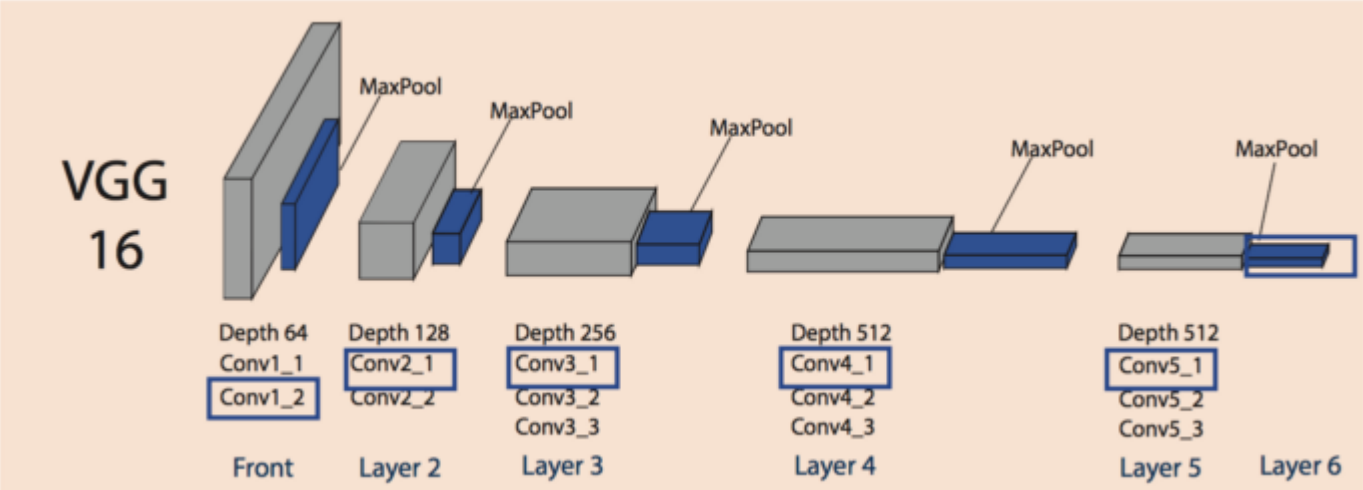
Digunakan metode **Deep Learning** untuk melakukan klasifikasi Pneumonia (Covid-19) pada gambar radiologi dengan mencari tanda kemunculan infiltrat/opasiti/konsolidasi.

➔ **Arsitektur**

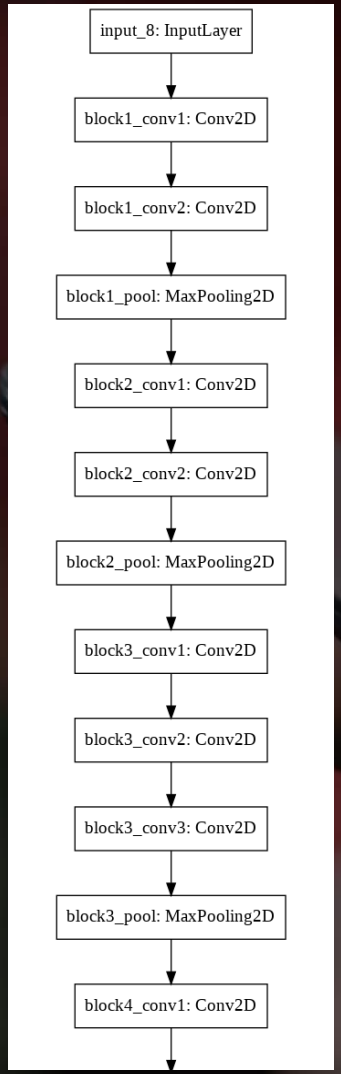
Digunakan metode transfer learning menggunakan pretrained model VGG16 yang sudah ditraining menggunakan dataset Imagenet. VGG-16 merupakan suatu arsitektur CNN, digunakan untuk ekstraksi fitur pada gambar.

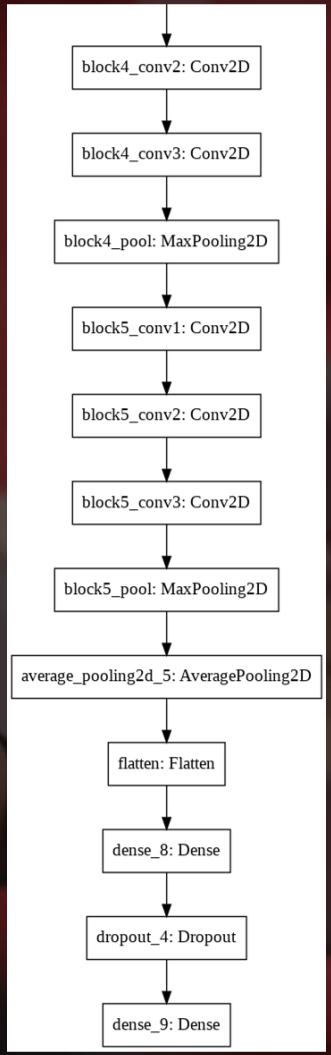
Arsitektur

Pretrained VGG-16 sebagai base model, lalu ditambahkan model baru (*trainable*) pada layer akhir VGG-16.



Source: https://miro.medium.com/max/700/1*8g0VV5VKbYwawo2nfu5_qQ.png





Keterangan :

Plot Model covid19 yang dipakai pada project kali ini. Terdapat perubahan jumlah neuron pada layer dense_8 dari 64 menjadi 128

Hyperparameter

- **Learning Rate (LR)**

  Merupakan *hyperparameter* yang berfungsi mengatur seberapa banyak perubahan *weight* dari *network* yang kita latih.

- **Batch Size (BS)**

  Jumlah training example dalam 1 kali iterasi.

- **Epoch**

  1 siklus training dari seluruh dataset.

Tuning Hyperparameter dengan Metode Grid Search

Coarse Tuning LR, BS, dan Epoch

LR = [0.001, 0.02, 0.1]

BS = [5,10,15]

Epoch = [10]

Best: 0.8501747012138366, using {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}

0.8350104808807373 (0.13326081145148522) with: {'INIT_LR': 0.001, 'batch_size': 5, 'epochs': 10, 'opti': 'Adam'}
0.8278826117515564 (0.15507166911153816) with: {'INIT_LR': 0.001, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.8123689770698548 (0.1353460319020693) with: {'INIT_LR': 0.001, 'batch_size': 15, 'epochs': 10, 'opti': 'Adam'}
0.8009084701538086 (0.13608229407892677) with: {'INIT_LR': 0.02, 'batch_size': 5, 'epochs': 10, 'opti': 'Adam'}
0.8501747012138366 (0.1484054389753976) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.6726065784692764 (0.34143801395765117) with: {'INIT_LR': 0.02, 'batch_size': 15, 'epochs': 10, 'opti': 'Adam'}
0.19622641801834106 (0.39245283603668213) with: {'INIT_LR': 0.1, 'batch_size': 5, 'epochs': 10, 'opti': 'Adam'}
0.13207546770572662 (0.2548217732567237) with: {'INIT_LR': 0.1, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.4037735849618912 (0.4868655787836572) with: {'INIT_LR': 0.1, 'batch_size': 15, 'epochs': 10, 'opti': 'Adam'}

Optimizer

- **SGD (Stochastic Gradient Descent)**

  Metode serupa dengan gradient descent, tetapi hanya mengambil berapa sampel random untuk memperbarui parameter pada setiap iterasi.

- **RMSprop (Root Mean Square Propagation)**

  Bertujuan untuk meredam osilasi (yang sangat mungkin terjadi pada SGD)

- **ADAM (Adaptive Moment Estimation)**

  Menyesuaikan learning rate pada penghitungan setiap gradien sampel.

Tuning Optimizer

Opti = [‘Adam’, ‘SGD’, ‘RMSprop’]

Best: 0.8876310348510742, using {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}

0.8876310348510742 (0.08904895017484625) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.6879105508327484 (0.30722008332494627) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'SGD'}
0.6943396210670472 (0.3856448695442646) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'RMSprop'}

Fine Learning Rate Tuning

LR = [0.008, 0.02, 0.04]
BS = [10]
Epoch = [10]

Best: 0.9170510292053222, using {'INIT_LR': 0.008, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}

0.9170510292053222 (0.05291516342185715) with: {'INIT_LR': 0.008, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.905870008468628 (0.0707105273633731) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.5547169804573059 (0.4604082881546126) with: {'INIT_LR': 0.04, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}

Fine Batch Size Tuning

LR = [0.02]
BS = [5,7,10,12]
Epoch = [10]

Best: 0.932145357131958, using {'INIT_LR': 0.02, 'batch_size': 7, 'epochs': 10, 'opti': 'Adam'}

0.6990915417671204 (0.3649947039121264) with: {'INIT_LR': 0.02, 'batch_size': 5, 'epochs': 10, 'opti': 'Adam'}
0.932145357131958 (0.0660628278129808) with: {'INIT_LR': 0.02, 'batch_size': 7, 'epochs': 10, 'opti': 'Adam'}
0.8719077587127686 (0.09517542117335681) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.8830887675285339 (0.10572534381912121) with: {'INIT_LR': 0.02, 'batch_size': 12, 'epochs': 10, 'opti': 'Adam'}

Fine Epochs Tuning

LR = [0.02]
BS = [10]
Epoch = [5, 7, 10, 12]

Best: 0.884136974811554, using {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 12, 'opti': 'Adam'}

0.8275331974029541 (0.10701311814271167) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 5, 'opti': 'Adam'}
0.7489168405532837 (0.1718184879826354) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 7, 'opti': 'Adam'}
0.835080373287201 (0.10759800492567591) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 10, 'opti': 'Adam'}
0.884136974811554 (0.11478860290959854) with: {'INIT_LR': 0.02, 'batch_size': 10, 'epochs': 12, 'opti': 'Adam'}

Hasil Tuning

- Learning Rate (LR) = 0.008
- Batch Size (BS) = 7
- Epoch = 12
- Optimizer = Adam

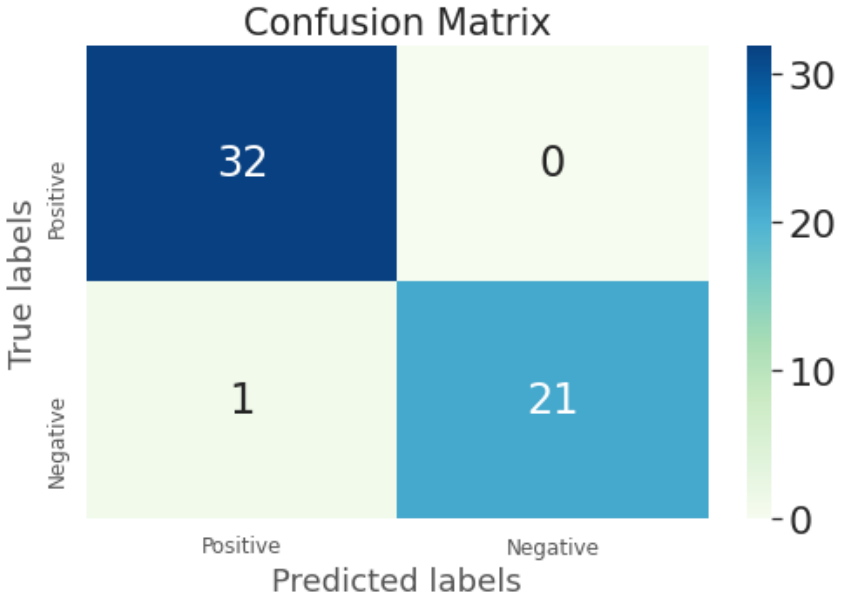
4. Hasil DL dengan Confusion Matrix

Accuracy = 0.9815

Precision = TP/(TP+FP) = 0.97

Sensitivity = TP/(TP+FN) = 1.000

Specificity = TN/(TN+FP) = 0.9545



5. Pembahasan Ringkas Hasil Eksperimen

Performa model yang telah di-*train* menggunakan dataset baru memiliki **peningkatan nilai akurasi** dibandingkan performa model awal, yaitu **dari 0.9000 menjadi 0.97.**

Beberapa metode yang dilakukan untuk meningkatkan performa model:

- ➔ **Augmentasi Data**
  Teknik untuk "memperbanyak" jumlah data training dengan memodifikasi gambar yang ada. Augmentasi membantu model mengenali gambar/data yang diberikan dari berbagai posisi/sudut pandang, terutama ketika jumlah data yang dimiliki terbatas dan variasinya sedikit. Tujuan dari augmentasi data adalah memberikan berbagai kemungkinan kondisi nyata dari sedikit data yang tersedia untuk melatih model. Karena data yang diambil untuk melakukan hal tersebut berasal dari data train, bagian data itulah yang dilakukan augmentasi.
  Pada eksperimen, digunakan augmentasi berupa shift, yaitu menggeser gambar secara vertikal ataupun horizontal.

➔ Optimizer

Optimizer diperoleh untuk mendapatkan hasil prediksi dengan akurasi terbaik.

Terdapat tiga opsi Optimizer yang dijadikan pertimbangan, yaitu Adam, SGD, dan RMSProp. Dari Dalam makalah hasil penelitian Kingma & Ba, *Adam: A Method for Stochastic Optimization* teruji bahwa training cost Adam lebih kecil daripada optimizer yang lainnya. Selain itu, berdasarkan hasil performa yang telah diperoleh, dipilih Adam sebagai Optimizer pada program kali ini.

Hasil performance dari masing-masing optimizer dicari menggunakan metode Grid Search

Evaluasi Model

- **Accuracy**
  menunjukkan berapa banyak klasifikasi yang tepat dilakukan oleh model dari seluruh hasil prediksi. Akurasi yang diperoleh sebesar 0.97. Hal ini berarti sebanyak 97% data test diklasifikasikan dengan tepat oleh model. Berdasarkan confusion matrix, terlihat bahwa terdapat 1/54 data tidak diklasifikasikan dengan tepat.

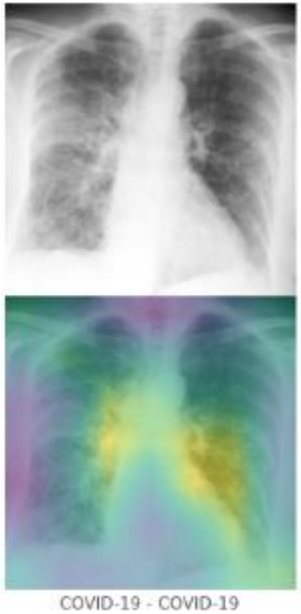
- **Sensitivity**
  menunjukkan berapa banyak data dalam kelas Positive diprediksi ke dalam kelas Positive. Semakin tinggi sensitivity, berarti semakin sedikit data yang termasuk dalam kategori false negative (data *covid* yang diprediksi sebagai normal). Diinginkan nilai sensitivity yang tinggi. Sensitivity yang diperoleh sebesar 1.000, berarti tidak ada data *covid* yang diprediksi sebagai normal.

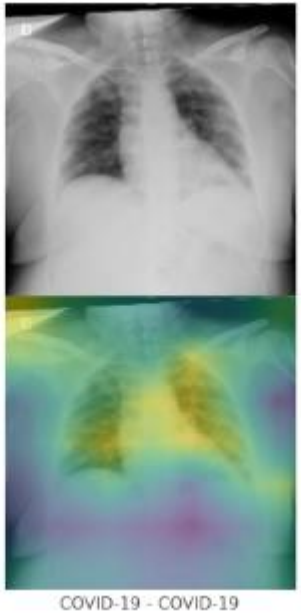
- **Specificity**
  menunjukkan berapa banyak data dalam kelas Negative diprediksi ke dalam kelas Negative. Semakin tinggi specificity, berarti semakin sedikit data yang termasuk dalam kategori false positive (data normal yang diprediksi sebagai *covid*). Specificity yang diperoleh sebesar 0.9545.

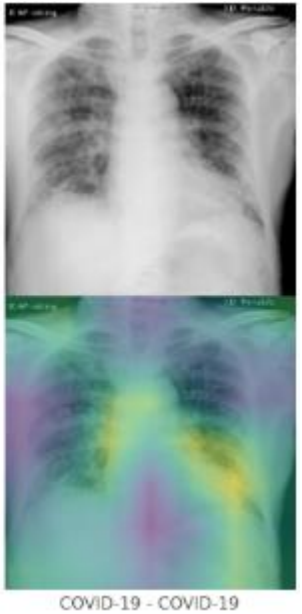
Analisis Kesesuaian Model berdasarkan CAM

Class Activation Map (CAM) dapat memvisualisasi confidence level untuk klasifikasi.

- **Sebagian besar CAM** telah menunjukkan daerah dengan confidence level klasifikasi tinggi ("daerah klasifikasi") yang sesuai dengan pedoman radiologi yang didapatkan dari referensi [11].
- COVID-19 yang memiliki gejala pneumonia dapat dikarakterisasikan dengan adanya **konsolidasi** (peningkatan densitas di gambar paru-paru) di daerah yang terinfeksi (dipenuhi cairan).

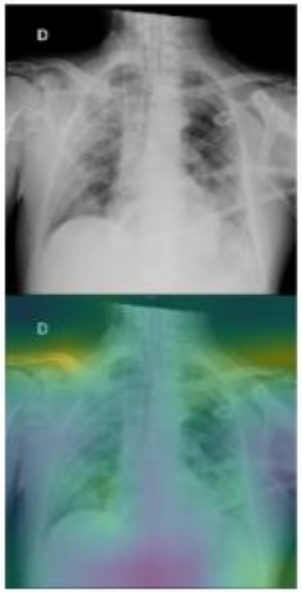

COVID-19 - COVID-19


COVID-19 - COVID-19

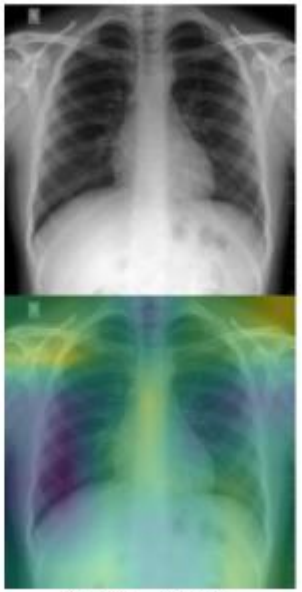

COVID-19 - COVID-19

Analisis Kesesuaian Model berdasarkan CAM

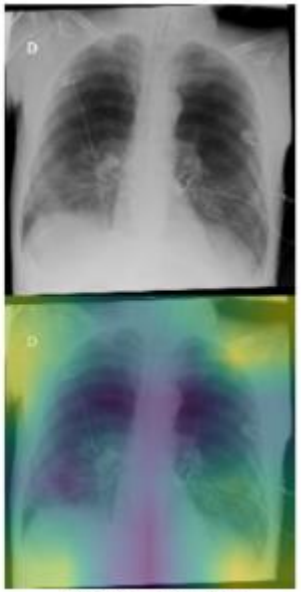
- Namun, **terdapat beberapa daerah klasifikasi yang tidak sesuai dengan pedoman radiologi**.
- Salah satu karakteristik dari kesalahan ini terdapat di gambar di samping, dengan daerah klasifikasi terletak di tulang selangka (klavikula).
- Model cenderung **mengikutsertakan bentuk klavikula** dalam klasifikasi COVID-19. Bentuk klavikula dipengaruhi oleh berbagai faktor, termasuk **sudut pengambilan gambar, postur pasien saat pengambilan gambar** dan **umur pasien**.
- Metadata menunjukkan bahwa sebagian kecil pasien memiliki umur di bawah 20 tahun. Ini dapat menjelaskan sebagian fenomena ini.



COVID-19 - COVID-19



NORMAL - NORMAL



COVID-19 - COVID-19

6. Source Code

Grid Search Function

```
def create_network(INIT_LR = 2e-2,EPOCHS = 10 ,opti = 'Adam'):
  baseModel = VGG16(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

  # construct the new head of the model
  headModel = baseModel.output
  headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
  headModel = Flatten(name="flatten")(headModel)
  headModel = Dense(64, activation="relu")(headModel)
  headModel = Dropout(0.5)(headModel)
  headModel = Dense(2, activation="softmax")(headModel)

  # place the new head model on top of the base model
  model = Model(inputs=baseModel.input, outputs=headModel)

  # freeze the base model
  for layer in baseModel.layers:
    layer.trainable = False
  if (opti == 'Adam') :
    opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
  elif(opti == 'SGD') :
    opt = SGD(lr=INIT_LR, decay=INIT_LR / EPOCHS)
  else :
    opt = RMSprop(lr=INIT_LR, decay=INIT_LR / EPOCHS)
  model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
  return model

# Create sklearn model
from keras.wrappers.scikit_learn import KerasClassifier
model = KerasClassifier(build_fn=create_network, verbose = 0)
```

6. Source Code

Coarse Tuning

```
# Coarse Tuning

optimizers = ['Adam']
learningrate = [0.001, 0.02, 0.1]
batchsize = [5,10,15]                          # Diambil dari Coarse Tuning. Will be optimized again next tune
epochs = [10]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size=batchsize,epochs=epochs, INIT_LR=learningrate,opti = optimizers)

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_results = grid.fit(data, labels)

# Summarize the results in a readable format
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))

means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

6. Source Code

Tuning Optimizer

```
# Tuning Optimizers

optimizers = ['Adam', 'SGD', 'RMSprop']
learningrate = [0.02]
batchsize = [10]
epochs = [10]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size=batchsize,epochs=epochs, INIT_LR=learningrate,opti = optimizers)

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_results = grid.fit(data, labels)

# Summarize the results in a readable format
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))

means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

6. Source Code

Fine Tuning LR

```
# Fine Tuning LR
optimizers = ['Adam']
learningrate = [0.008, 0.02, 0.04]
batchsize = [10]                          # Diambil dari Coarse Tuning. Will be optimized again next tune
epochs = [10]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size=batchsize,epochs=epochs, INIT_LR=learningrate,opti = optimizers)

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_results = grid.fit(data, labels)

# Summarize the results in a readable format
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))

means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

6. Source Code

Fine Batch Size Tuning

```
# Fine Batch Size Tuning

optimizers = ['Adam']
learningrate = [0.02]
batchsize = [5,7,10,12]
epochs = [10]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size=batchsize,epochs=epochs, INIT_LR=learningrate,opti = optimizers)

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_results = grid.fit(data, labels)

# Summarize the results in a readable format
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))

means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

6. Source Code

Fine Epochs Tuning

```
# Fine EPOCHS Tuning

optimizers = ['Adam']
learningrate = [0.02]
batchsize = [10]
epochs = [5,7,10,12]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size=batchsize,epochs=epochs, INIT_LR=learningrate,opti = optimizers)

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_results = grid.fit(data, labels)

# Summarize the results in a readable format
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))

means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

6. Source Code

Importing Dependencies

```
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.applications import imagenet_utils
from tensorflow.keras.models import Model
import tensorflow as tf
import numpy as np
import argparse
import imutils
import cv2
```

6. Source Code

GradCAM

```
class GradCAM:
  def __init__(self, model, classIdx, layerName=None):
    # store the model, the class index used to measure the class
    # activation map, and the layer to be used when visualizing
    # the class activation map
    self.model = model
    self.classIdx = classIdx
    self.layerName = layerName

    # if the layer name is None, attempt to automatically find
    # the target output layer
    if self.layerName is None:
      self.layerName = self.find_target_layer()

  def find_target_layer(self):
    # attempt to find the final convolutional layer in the network
    # by looping over the layers of the network in reverse order
    for layer in reversed(self.model.layers):
      # check to see if the layer has a 4D output
      if len(layer.output_shape) == 4:
        return layer.name

    # otherwise, we could not find a 4D layer so the GradCAM
    # algorithm cannot be applied
    raise ValueError("Could not find 4D layer. Cannot apply GradCAM.")
```

6. Source Code

```
def compute_heatmap(self, image, eps=1e-8):
  # construct our gradient model by supplying (1) the inputs
  # to our pre-trained model, (2) the output of the (presumably)
  # final 4D layer in the network, and (3) the output of the
  # softmax activations from the model
  gradModel = Model(
    inputs=[self.model.inputs],
    outputs=[self.model.get_layer(self.layerName).output,
      self.model.output])

  # record operations for automatic differentiation
  with tf.GradientTape() as tape:
    # cast the image tensor to a float-32 data type, pass the
    # image through the gradient model, and grab the loss
    # associated with the specific class index
    inputs = tf.cast(image, tf.float32)
    (convOutputs, predictions) = gradModel(inputs)
    loss = predictions[:, self.classIdx]

  # use automatic differentiation to compute the gradients
  grads = tape.gradient(loss, convOutputs)
  grads = tf.math.truediv(
    tf.math.subtract(
      grads,
      tf.math.reduce_min(grads)
    ),
    tf.math.subtract(
      tf.math.reduce_max(grads),
      tf.math.reduce_min(grads)
    )
  )
```

6. Source Code

```
# compute the guided gradients
castConvOutputs = tf.cast(convOutputs > 0, "float32")
castGrads = tf.cast(grads > 0, "float32")
guidedGrads = castConvOutputs * castGrads * grads

# the convolution and guided gradients have a batch dimension
# (which we don't need) so let's grab the volume itself and
# discard the batch
convOutputs = convOutputs[0]
guidedGrads = guidedGrads[0]

# compute the average of the gradient values, and using them
# as weights, compute the ponderation of the filters with
# respect to the weights
weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
cam = tf.reduce_sum(tf.multiply(weights, convOutputs), axis=-1)

# grab the spatial dimensions of the input image and resize
# the output class activation map to match the input image
# dimensions
(w, h) = (image.shape[2], image.shape[1])
heatmap = cv2.resize(cam.numpy(), (w, h))

# normalize the heatmap such that all values lie in the range
# [0, 1], scale the resulting values to the range [0, 255],
# and then convert to an unsigned 8-bit integer
numer = heatmap - np.min(heatmap)
denom = (heatmap.max() - heatmap.min()) + eps
heatmap = numer / denom
heatmap = (heatmap * 255).astype("uint8")

# return the resulting heatmap to the calling function
return heatmap
```

6. Source Code

```
def overlay_heatmap(self, heatmap, image, alpha=0.5, colormap=cv2.COLORMAP_VIRIDIS):
  # apply the supplied color map to the heatmap and then
  # overlay the heatmap on the input image
  heatmap = cv2.applyColorMap(heatmap, colormap)
  output = cv2.addWeighted(image, alpha, heatmap, 1 - alpha, 0)

  # return a 2-tuple of the color mapped heatmap and the output,
  # overlaid image
  return (heatmap, output)
```

6. Source Code

```
plt.figure(figsize=(30,100))
for i in range(len(testX)):
  plt.subplot(11, 6, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)

  orig = testX[i]
  orig_image = 255 * orig
  orig_image = orig_image.astype(np.uint8)

  image = np.expand_dims(orig, axis=0)

  # use the network to make predictions on the input image and find
  # the class label index with the largest corresponding probability
  preds = model.predict(image)
  idx = np.argmax(preds[0])

  # initialize our gradient class activation map and build the heatmap
  cam = GradCAM(model, idx, "block5_pool")
  heatmap = cam.compute_heatmap(image)

  # resize the resulting heatmap to the original input image dimensions
  # and then overlay heatmap on top of the image
  heatmap = cv2.resize(heatmap, (orig_image.shape[1], orig_image.shape[0]))
  (heatmap, output) = cam.overlay_heatmap(heatmap, orig_image, alpha=0.5)
```

6. Source Code

```
  from google.colab.patches import cv2_imshow

  # display the original image and resulting heatmap and output image to our screen
  output = np.vstack([orig_image, output])
  output = imutils.resize(output, height=700)

  plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))

  if (testY[i][0] >= testY[i][1]):
    label = "COVID-19"
  else:
    label = "NORMAL"

  if (preds[0][0] >= preds[0][1]) :
    string = label + " - COVID-19"
    plt.xlabel(string)
  else :
    string = label + " - NORMAL"
    plt.xlabel(string)
plt.show()
```


Thank you for
your attention.

Daftar Pustaka

1. Joseph Paul Cohen and Paul Morrison and Lan Dao COVID-19 image data collection, arXiv:2003.11597, 2020 https://github.com/ieee8023/covid-chestxray-dataset
2. Matt Ross and Blake Vanberlo COVID-19 Chest X-Ray Model, 2020 https://github.com/aildnont/covid-cxr
3. DarwinAI Corp., Canada and Vision and Image Processing Research Group and Vision and Image Processing Research Group, University of Waterloo, Canada, 2020 https://github.com/agchung/Figure1-COVID-chestxray-dataset/blob/master/README.md
4. Linda Wang of DarwinAI Corp., Canada and Vision and Image Processing Research Group, 2020 https://github.com/lindawangg/COVID-Net
5. Wang X, Peng Y, Lu L, Lu Z, Bagheri M, Summers RM. ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. IEEE CVPR 2017, ChestX-ray8Hospital-ScaleChestCVPR2017_paper.pdf
6. NIH News release: NIH Clinical Center provides one of the largest publicly available chest x-ray datasets to scientific community https://www.kaggle.com/nih-chest-xrays/data
7. Original source files and documents: https://nihcc.app.box.com/v/ChestXray-NIHCC/folder/36938765345
8. Chest Imaging Twitter, COVID-19 CXR (all SARS-CoV-2 PCR+) from Spain hospital, 2020 https://twitter.com/ChestImaging/status/1243928581983670272
9. Adrian Rosebrock, Detecting COVID-19 in X-ray images with Keras, TensorFlow, and Deep Learning, 2020 https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/
10. Normal Chest X-Ray Data, 2020, http://www.chestx-ray.com/index.php/education/normal-cxr-module-train-your-eye#!129
11. Robin Smithuis, Chest X-Ray - Lung disease, https://radiologyassistant.nl/chest/chest-x-ray-lung-disease#consolidation
12. Jason Brownlee, Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
13. Kevin Markham, Data School:ROC Curves and Area Under the Curve (AUC) Explained, https://www.youtube.com/watch?v=OAl6eAyP-yo