DAFTAR PUSTAKA

Achmad, Y., Wihandika, R. C., & Dewi, C. (2019). Klasifikasi emosi berdasarkan ciri wajah wenggunakan convolutional neural network. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *3*(11), 10595–10604.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a Convolutional Neural Network. *International Conference on Engineering and Technology (ICET) (IEEE)*.

Armañanzas, R., Bielza, C., Chaudhuri, K. R., Martinez-Martin, P., & Larrañaga, P. (2013). Unveiling relevant non-motor Parkinson's disease severity symptoms using a machine learning approach. *Artificial Intelligence in Medicine*, *58*, 195–202. https://doi.org/10.1016/j.artmed.2013.04.002

Chakraborty, S., Aich, S., Jong-Seong-Sim, Han, E., Park, J., & Kim, H. C. (2020). Parkinson's Disease Detection from Spiral and Wave Drawings using Convolutional Neural Networks: A Multistage Classifier Approach. *International Conference on Advanced Communication Technology, ICACT*, 298–303. https://doi.org/10.23919/ICACT48636.2020.9061497

Dzulqarnain, M. F., Suprapto, S., & Makhrus, F. (2019). Improvement of Convolutional Neural Network Accuracy on Salak Classification Based Quality on Digital Image. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, *13*(2), 189–198. https://doi.org/10.22146/ijccs.42036

Gil-Martín, M., Montero, J. M., & San-Segundo, R. (2019). Parkinson's disease detection from drawing movements using convolutional neural networks. *Electronics (Switzerland)*, *8*, 1–10. https://doi.org/10.3390/electronics8080907

Goodfellow, I., Courville, A., & Bengio, Y. (2016). Deep Learning. Cambridge: MIT press.

Hanriko, R., & Anzani, B. P. (2018). Parkinson's Disease : Health Threat to the Agricultural Community. *J Agromedicine*, *5*(1), 508–512.

Imam. (2018). Memahami *Epoch Batch Size* dan *Iteration. Retrieved July* 27, 2020, from Imam Digmi: https://imam.digmi.id/post/memahami-epoch-batch-size-dan-iteration/

Jahan, N., Nesa, A., & Layek, M. A. (2021). Parkinson's Disease Detection Using ResNet50 with Transfer Learning. *International Journal of Computer Vision and Signal Processing*, *11*(1), 17–23. http://cennser.org/IJCVSP

Jumadi, J., Yupianti, & Sartika, D. (2021). Pengolahan Citra Digital Untuk Identifikasi Objek Menggunakan Metode Hierarchical Agglomerative Clustering. *JST (Jurnal Sains Dan Teknologi)*, *10*(2), 148–156. https://doi.org/10.23887/jstundiksha.v10i2.33636

Lina, Qolbiyatul. (2019). Apa itu Convolutional Neural Network?. https://medium.com/@16611110/apa-itu-convolutional-neural-network 836f70b193a4.

Louis, M. S., Azad, Z., Delshadtehrani, L., Gupta, S., Warden, P., Reddi, V. J., & Joshi, A. (2019). *Towards Deep Learning using TensorFlow Lite on RISC-V*. Proc. ACM CARRV, 1-6.

Mahmud, K. H., Adiwijaya, & Al Faraby, S. (2019). Klasifikasi Citra Multi-Kelas Menggunakan Convolutional Neural Network. *E-Proceeding of Engineering*, *6*(1), 2127–2136.

Pokhrel, Sabina. (2019). Beginners Guide to Convolutional Neural Networks. https://towardsdatascience.com/beginners-guide-to-understanding-convolutional-neural-networks-ae9ed58bb17d.

Ray Dorsey, E., Elbaz, A., Nichols, E., Abd-Allah, F., Abdelalim, A., Adsuar, J. C., Ansha, M. G., Brayne, C., Choi, J. Y. J., Collado-Mateo, D., Dahodwala, N., Do, H. P., Edessa, D., Endres, M., Fereshtehnejad, S. M., Foreman, K. J.,

Gankpe, F. G., Gupta, R., Hankey, G. J., … Murray, C. J. L. (2018). Global, regional, and national burden of Parkinson's disease, 1990–2016: a systematic analysis for the Global Burden of Disease Study 2016. *The Lancet Neurology*, *17*, 939–953. https://doi.org/10.1016/S1474-4422(18)30295-3

Salmanpour, M. R., Shamsaei, M., Saberi, A., Setayeshi, S., Klyuzhin, I. S., Sossi, V., & Rahmim, A. (2019). Optimized machine learning methods for prediction of cognitive outcome in Parkinson's disease. *Computers in Biology and Medicine*, *111*, 1–8. https://doi.org/10.1016/j.compbiomed.2019.103347

Santoso, A., & Ariyanto, G. (2018). Implementasi Deep Learning berbasis Keras untuk Pengenalan Wajah. *Emitor: Jurnal Teknik Elektro*, *18*(1), 15–21. https://doi.org/10.23917/emitor.v18i01.6235

Suartika E. P, I Wayan, Wijaya Arya Yudhi, R. S. (2016). Klasifikasi Citra Menggunakan Convolutional Neural Network (Cnn) Pada Caltech 101. *Jurnal Teknik ITS*, *5*(1), 1–5. http://repository.its.ac.id/48842/

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2016). Inception-v4, inception-ResNet and the impact of residual connections on learning. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*, 1–12. https://doi.org/10.1609/aaai.v31i1.11231

Tiwari, A. K. (2016). Machine Learning Based Approaches for Prediction of Parkinson's Disease. *Machine Learning and Applications: An International Journal*, *3*, 33–39. https://doi.org/10.5121/mlaij.2016.3203

Wiranda, N., Purba, H. S., & Sukmawati, R. A. (2020). Survei Penggunaan Tensorflow pada Machine Learning untuk Identifikasi Ikan Kawasan Lahan Basah. *IJEIS (Indonesian Journal of Electronics and Instrumentation Systems)*, *10*(2), 179–188. https://doi.org/10.22146/ijeis.58315

LAMPIRAN

Source Code

Berikut *Source Code* yang digunakan pada implementasi arsitektur *Inception ResNet-V2* dalam mendiagnosis penyakit parkinson.

In [1]:

```
import numpy as np
import tensorflow as tf
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns

from tensorflow.keras.layers import Dense, Flatten, Input, GlobalAveragePooling2D,
Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2
from tensorflow.keras.applications.inception_resnet_v2 import preprocess_input

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from sklearn.metrics import confusion_matrix, classification_report,
precision_recall_fscore_support, accuracy_score, roc_curve, auc
```

In [2]:

```
from tflite_support.metadata_writers import image_classifier
from tflite_support.metadata_writers import writer_utils
from tflite_support import metadata
```

In [3]:

```
print(tf.__version__)
print(tf.config.list_physical_devices('GPU'))
```

Out [3]:

```
2.6.0
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

In [4]:

```
seed = 101
np.random.seed(seed)
tf.random.set_seed(seed)
random.seed(seed)
os.environ['PYTHONHASHSEED'] = str(seed)
```

In [5]:

```
orig_path = 'D:\parkinson'

paths = []
labels = []
shapes = []
split = []

for root, dirs, files in os.walk(orig_path):
    path = root.split(os.sep)
    for file in files:
        current_path = '/'.join(path)+'/'+file
        paths += [current_path]
        labels += [current_path.split('/')[-2]]
        split += [current_path.split('/')[-3]]
        shapes += [current_path.split('/')[-4]]
```

In [5]:

```
list_combined = map(list, zip(*[paths, shapes, labels, split]))
```

In [6]:

```
df = pd.DataFrame(list_combined,columns=['path', 'shape', 'label', 'split'])
df
```

Out [6]:

	path	shape	label	split
0	parkinson/spiral/testing/healthy/V01HE01.png	spiral	healthy	testing
1	parkinson/spiral/testing/healthy/V02HE01.png	spiral	healthy	testing
2	parkinson/spiral/testing/healthy/V03HE1.png	spiral	healthy	testing
3	parkinson/spiral/testing/healthy/V04HE01.png	spiral	healthy	testing
4	parkinson/spiral/testing/healthy/V05HE01.png	spiral	healthy	testing
...	...	...	...	...
199	parkinson/wave/training/parkinson/V13PO03.png	wave	parkinson	training
200	parkinson/wave/training/parkinson/V14PO02.png	wave	parkinson	training
201	parkinson/wave/training/parkinson/V15PO01.png	wave	parkinson	training
202	parkinson/wave/training/parkinson/V15PO02.png	wave	parkinson	training
203	parkinson/wave/training/parkinson/V15PO03.png	wave	parkinson	training

204 rows × 4 columns

In [7]:

```
df_train = df[df['split']=='training'].drop(columns='split').sample(frac=1,
random_state=seed).reset_index(drop=True)
df_test = df[df['split']=='testing'].drop(columns='split').sample(frac=1,
random_state=seed).reset_index(drop=True)
```

In [8]:

```
df_train.head()
```

Out [8]:

	path	shape	label
0	parkinson/spiral/training/parkinson/V12PE03.png	spiral	parkinson
1	parkinson/wave/training/parkinson/V14PO02.png	wave	parkinson
2	parkinson/spiral/training/healthy/V09HE02.png	spiral	healthy
3	parkinson/spiral/training/parkinson/V13PE02.png	spiral	parkinson
4	parkinson/wave/training/parkinson/V03PO03.png	wave	parkinson

In [9]:

```
df_test.head()
```

Out [9]:

	path	shape	label
0	parkinson/wave/testing/parkinson/V01PO01.png	wave	parkinson
1	parkinson/wave/testing/parkinson/V11PO01.png	wave	parkinson
2	parkinson/wave/testing/parkinson/V08PO01.png	wave	parkinson
3	parkinson/spiral/testing/parkinson/V02PE01.png	spiral	parkinson
4	parkinson/spiral/testing/parkinson/V10PE03.png	spiral	parkinson

In [10]:

```
IMG_SIZE = 299
gen = ImageDataGenerator(
    #rescale=1.0 / 255,
    #rotation_range=15,
    #width_shift_range=0.15,
    #height_shift_range=0.15,
    #shear_range=0.5,
    zoom_range=0.2,
    #vertical_flip=True,
    #horizontal_flip=True,
    preprocessing_function=preprocess_input,
    fill_mode='nearest',
    validation_split=0.20
)
```

```
train_gen = gen.flow_from_dataframe(
    df_train, x_col='path', y_col='label', subset="training", shuffle=False,
target_size=(IMG_SIZE, IMG_SIZE),
    class_mode="categorical", seed=seed
)
gen = ImageDataGenerator(
    #rescale=1.0 / 255,
    validation_split=0.20,
    fill_mode='nearest',
    preprocessing_function=preprocess_input
)
val_gen = gen.flow_from_dataframe(
    df_train, x_col='path', y_col='label', subset="validation", shuffle=False,
target_size=(IMG_SIZE, IMG_SIZE),
    class_mode="categorical", seed=seed
)
gen = ImageDataGenerator(
    #rescale=1.0 / 255,
    preprocessing_function=preprocess_input
)
test_gen = gen.flow_from_dataframe(
    df_test, x_col='path', y_col='label', shuffle=False, target_size=(IMG_SIZE,
IMG_SIZE),
    class_mode="categorical", seed=seed
)
```

Out [10]:

```
Found 116 validated image filenames belonging to 2 classes.
Found 28 validated image filenames belonging to 2 classes.
Found 60 validated image filenames belonging to 2 classes.
```

In [11]:

```
classes = list(train_gen.class_indices.keys())
_, train_counts = np.unique(train_gen.classes, return_counts=True)
_, val_counts = np.unique(val_gen.classes, return_counts=True)
print('number of classes in each set:')
counts = pd.DataFrame([train_counts, val_counts], columns=classes, index=['train
count', 'val count']).T
counts['total'] = counts['train count'] + counts['val count']
print('total training data:', np.sum(counts['train count']))
print('total validation data:', np.sum(counts['val count']))
counts
```

Out [11]:

```
number of classes in each set:
total training data: 116
total validation data: 28
        train count     val count  total
healthy    60          12         72
parkinson  56          16         72
```

In [12]:

```
model = InceptionResNetV2(weights='imagenet', include_top=False,
input_shape=(IMG_SIZE, IMG_SIZE, 3))
model.trainable = False
#model.summary()
```

In [13]:

```
x = model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(2, activation='softmax')(x)
model = Model(inputs=model.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
#model.summary()
```

In [14]:

```
EPOCHS = 50
BATCH_SIZE = 8
```

In [15]:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=15)
checkpoint = ModelCheckpoint('checkpoint/check', monitor='val_loss', verbose=1,
save_best_only=True, save_weights_only=True, save_freq='epoch')
```

In [16]:

```
history = model.fit(train_gen, validation_data=val_gen, epochs=EPOCHS,
callbacks=[early_stopping, checkpoint])
```

Out [16]:

```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/50
4/4 [==============================] - ETA: 0s - loss: 0.9529 - accuracy: 0.4655
Epoch 1: val_loss improved from inf to 0.68898, saving model to checkpoint\check
4/4 [==============================] - 114s 22s/step - loss: 0.9529 - accuracy: 0.4655 - val_loss: 0.6890 - val_accuracy: 0.5714
Epoch 2/50
4/4 [==============================] - ETA: 0s - loss: 0.7171 - accuracy: 0.5431
Epoch 2: val_loss improved from 0.68898 to 0.57865, saving model to checkpoint\check
4/4 [==============================] - 69s 18s/step - loss: 0.7171 - accuracy: 0.5431 - val_loss: 0.5787 - val_accuracy: 0.6429
Epoch 3/50
4/4 [==============================] - ETA: 0s - loss: 0.5941 - accuracy: 0.6724
Epoch 3: val_loss improved from 0.57865 to 0.54977, saving model to checkpoint\check
4/4 [==============================] - 71s 20s/step - loss: 0.5941 - accuracy: 0.6724 - val_loss: 0.5498 - val_accuracy: 0.7500
Epoch 4/50
4/4 [==============================] - ETA: 0s - loss: 0.5247 - accuracy: 0.7500
Epoch 4: val_loss improved from 0.54977 to 0.49322, saving model to checkpoint\check
4/4 [==============================] - 71s 20s/step - loss: 0.5247 - accuracy: 0.7500 - val_loss: 0.4932 - val_accuracy: 0.8571
Epoch 5/50
4/4 [==============================] - ETA: 0s - loss: 0.5487 - accuracy: 0.7241
Epoch 5: val_loss improved from 0.49322 to 0.46605, saving model to checkpoint\check
4/4 [==============================] - 74s 19s/step - loss: 0.5487 - accuracy: 0.7241 - val_loss: 0.4661 - val_accuracy: 0.7857
Epoch 6/50
4/4 [==============================] - ETA: 0s - loss: 0.5868 - accuracy: 0.6724
Epoch 6: val_loss improved from 0.46605 to 0.45027, saving model to checkpoint\check
4/4 [==============================] - 65s 18s/step - loss: 0.5868 - accuracy: 0.6724 - val_loss: 0.4503 - val_accuracy: 0.7857
Epoch 7/50
...
Epoch 50/50
4/4 [==============================] - ETA: 0s - loss: 0.3293 - accuracy: 0.8707
Epoch 50: val_loss did not improve from 0.20714
4/4 [==============================] - 75s 19s/step - loss: 0.3293 - accuracy: 0.8707 - val_loss: 0.2184 - val_accuracy: 0.9286
```

In [17]:

```
model.load_weights('checkpoint/check')
```

Out [17]:

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x21eda69e3b0>
```

In [18]:

```
epoch_result = pd.DataFrame([history.history['accuracy'],
history.history['val_accuracy'], history.history['loss'], history.history['val_loss'], ],
columns=np.arange(1,EPOCHS+1,step=1), index=['accuracy', 'val_accuracy', 'loss',
'val_loss']).T
epoch_result.index.name='epoch'
for col in epoch_result:
    if col == 'loss' or col == 'val_loss':
        continue
    epoch_result[col] = epoch_result[col] * 100
epoch_result = epoch_result.round(2)
epoch_result
```

Out [18]:

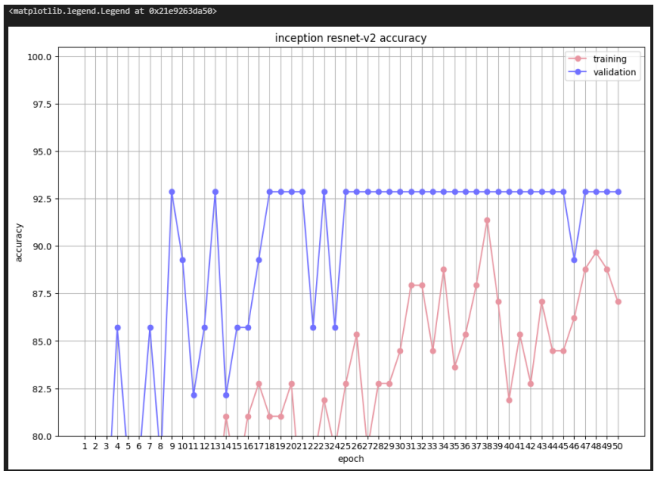
epoch	accuracy	val_accuracy	loss	val_loss
1	46.55	57.14	0.95	0.69
2	54.31	64.29	0.72	0.58
3	67.24	75.00	0.59	0.55
4	75.00	85.71	0.52	0.49
5	72.41	78.57	0.55	0.47
6	67.24	78.57	0.59	0.45
7	73.28	85.71	0.50	0.45
8	69.83	78.57	0.51	0.42
9	71.55	92.86	0.53	0.38
10	79.31	89.29	0.44	0.37
11	77.59	82.14	0.45	0.36
12	75.86	85.71	0.46	0.34
13	75.86	92.86	0.46	0.34
14	81.03	82.14	0.41	0.35
15	77.59	85.71	0.45	0.32
16	81.03	85.71	0.38	0.33
17	82.76	89.29	0.39	0.30
18	81.03	92.86	0.38	0.30
19	81.03	92.86	0.37	0.29
20	82.76	92.86	0.36	0.28
21	75.86	92.86	0.42	0.28
22	78.45	85.71	0.45	0.28
23	81.90	92.86	0.37	0.27
24	79.31	85.71	0.38	0.30
25	82.76	92.86	0.37	0.27
26	85.34	92.86	0.37	0.27
27	79.31	92.86	0.42	0.26
28	82.76	92.86	0.41	0.26
29	82.76	92.86	0.36	0.26
30	84.48	92.86	0.34	0.26
31	87.93	92.86	0.32	0.24
32	87.93	92.86	0.32	0.23
33	84.48	92.86	0.33	0.25
34	88.79	92.86	0.30	0.23
35	83.62	92.86	0.35	0.23

epoch	accuracy	val_accuracy	loss	val_loss
36	85.34	92.86	0.32	0.22
37	87.93	92.86	0.34	0.23
38	91.38	92.86	0.30	0.24
39	87.07	92.86	0.29	0.22
40	81.90	92.86	0.32	0.21
41	85.34	92.86	0.29	0.21
42	82.76	92.86	0.32	0.22
43	87.07	92.86	0.32	0.23
44	84.48	92.86	0.33	0.22
45	84.48	92.86	0.32	0.24
46	86.21	89.29	0.34	0.22
47	88.79	92.86	0.29	0.23
48	89.66	92.86	0.29	0.21
49	88.79	92.86	0.29	0.21
50	87.07	92.86	0.33	0.22

In [19]:

```
plt.figure(figsize=(12,8))
plt.plot(epoch_result['accuracy'],'-o', label='training', color='#e895a1')
plt.plot(epoch_result['val_accuracy'], '-o', label='validation', color='#7070ff')
plt.title('inception resnet-v2 accuracy')
plt.ylim(80,100.5)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.xticks(np.arange(1, EPOCHS+1, step=1))
plt.grid()
plt.legend()
```

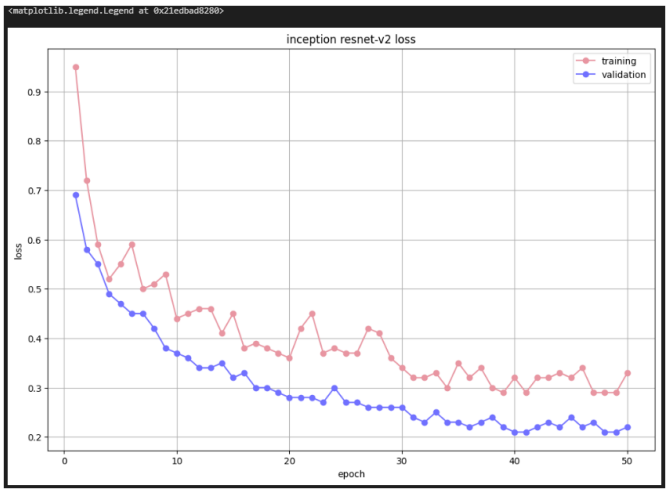
Out [19]:



In [20]:

```
plt.figure(figsize=(12,8))
plt.plot(epoch_result['loss'], '-o', label='training', color='#e895a1')
plt.plot(epoch_result['val_loss'], '-o', label='validation', color='#7070ff')
plt.title('inception resnet-v2 loss')
plt.xlabel('epoch')
plt.ylabel('loss')
#plt.xticks(np.arange(1, EPOCHS+1, step=1))
plt.grid()
plt.legend()
```

Out [20]:



In [21]:

```
y_pred_test = np.argmax(model.predict(test_gen),axis=1)
```

In [22]:

```
accuracy = accuracy_score(test_gen.classes, y_pred_test)
test_result = pd.DataFrame(precision_recall_fscore_support(test_gen.classes,
y_pred_test), columns=classes, index=['precision', 'recall', 'f1 score',
'support']).T.drop('support', axis=1)
#test_scores = [accuracy, np.average(test_result['precision']),
np.average(test_result['recall']), np.average(test_result['f1 score'])]
test_result
```

Out [22]:

	precision	recall	f1 score
healthy	0.812500	0.866667	0.838710
parkinson	0.857143	0.800000	0.827586

In [23]:

```
multiindex_pred = pd.MultiIndex.from_tuples(list(zip(*[['prediction']*26, classes])))
multiindex_actual = pd.MultiIndex.from_tuples(list(zip(*[['actual']*26, classes])))
```

In [24]:

```
test_cm = pd.DataFrame(confusion_matrix(test_gen.classes, y_pred_test))
test_cm.columns = multiindex_pred
test_cm.index = multiindex_actual
test_cm.index.name = 'actual'
test_cm
```

Out [24]:

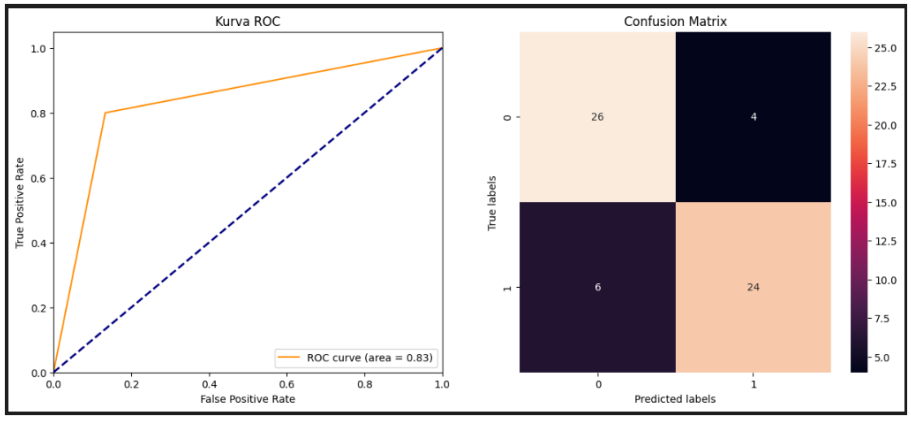
		prediction	
		healthy	parkinson
actual	healthy	26	4
	parkinson	6	24

In [25]:

```
y_true = test_gen.classes
y_pred = y_pred_test
cm = confusion_matrix(y_true, y_pred)
title = 'Kurva ROC'
lw = 2
fpr, tpr, thresholds = roc_curve(y_true, y_pred)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))
ax1.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % auc(fpr,
tpr))
ax1.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
ax1.set_xlim([0.0, 1.0]);ax1.set_ylim([0.0, 1.05]);
ax1.set_xlabel('False Positive Rate');ax1.set_ylabel('True Positive Rate');
ax1.set_title(title)
ax1.legend(loc="lower right")

sns.heatmap(cm, annot=True, ax = ax2, fmt='d');
ax2.set_xlabel('Predicted labels'); ax2.set_ylabel('True labels');
ax2.set_title('Confusion Matrix');
ax2.xaxis.set_ticklabels(['0', '1']); ax2.yaxis.set_ticklabels(['0', '1']);
```

Out [25]:



In [26]:

```
def autolabel(rects):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., height+0.015, f'{height:.3f}',
                ha='center', va='bottom')
```

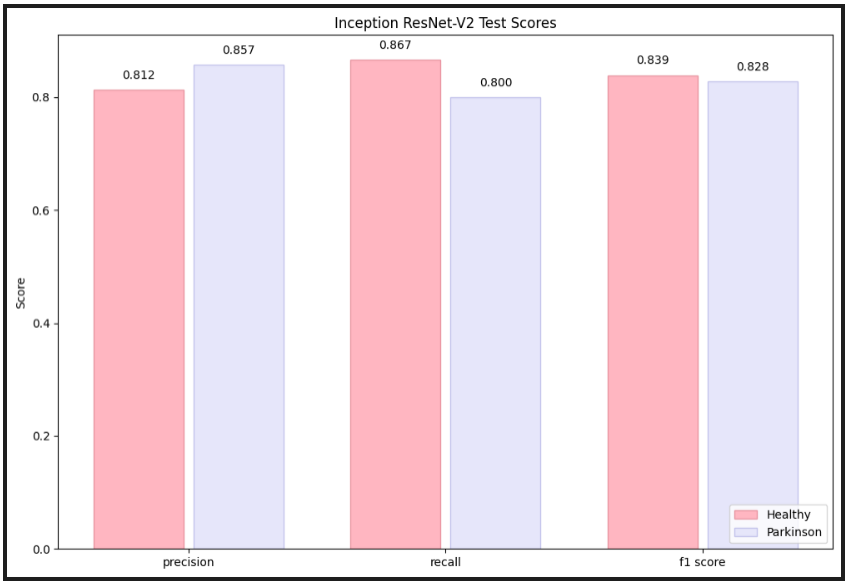
In [27]:

```
labels = ['precision', 'recall', 'f1 score']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars
fig, ax = plt.subplots(figsize=(12,8))
rects1 = ax.bar(x - 0.02 - width/2, test_result.loc['healthy'].tolist(), width,
label='Healthy', color='#FFB6C1', edgecolor='#e895a1')
rects2 = ax.bar(x + 0.02 + width/2, test_result.loc['parkinson'].tolist(), width,
label='Parkinson', color='#E6E6FA', edgecolor='#c3c3eb')

ax.set_ylabel('Score')
ax.set_title(f'Inception ResNet-V2 Test Scores')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc='lower right')

autolabel(rects1)
autolabel(rects2)
```

Out [27]:



In [28]:

```
def f(seq):
    seen = set()
    seen_add = seen.add
    return [x for x in seq if not (x in seen or seen_add(x))]
```

In [29]:

```
class_indices = dict((y,x) for x,y in train_gen.class_indices.items())
classes1 =
pd.DataFrame(f(train_gen.classes),columns=['a']).replace({'a':class_indices})
np.savetxt(r'labels.txt', classes1.values, fmt='%s')
```

In [30]:

```
saved_model_dir = 'D:/SKRIPSI/model.h5'
model.save(saved_model_dir)
```

In [31]:

```
model = tf.keras.models.load_model(saved_model_dir)
```

In [32]:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

# Save the model.
with open(saved_model_dir + '/alwa_new_quant.tflite', 'wb') as f:
  f.write(tflite_model)
```

In [33]:

```
ImageClassifierWriter = image_classifier.MetadataWriter
_MODEL_PATH = "D:/SKRIPSI/alwa_new_quant.tflite"
# Task Library expects label files that are in the same format as the one below.
_LABEL_FILE = "labels.txt"
_SAVE_TO_PATH = "D:/SKRIPSI/alwa_new_quant_metadata.tflite"
# Normalization parameters is required when reprocessing the image. It is
# optional if the image pixel values are in range of [0, 255] and the input
# tensor is quantized to uint8. See the introduction for normalization and
# quantization parameters below for more details.
#
https://www.tensorflow.org/lite/convert/metadata#normalization_and_quantization_p
arameters)
_INPUT_NORM_MEAN = 127.5
_INPUT_NORM_STD = 127.5
```

```
# Create the metadata writer.
writer = ImageClassifierWriter.create_for_inference(
    writer_utils.load_file(_MODEL_PATH), [_INPUT_NORM_MEAN],
[_INPUT_NORM_STD],
    [_LABEL_FILE])

# Verify the metadata generated by metadata writer.
print(writer.get_metadata_json())

# Populate the metadata into the model.
writer_utils.save_file(writer.populate(), _SAVE_TO_PATH)
```

Out[33]:

Output exceeds the size limit. Open the full output data in a text editor
{
"name": "ImageClassifier",
"description": "Identify the most prominent object in the image from a known set of categories.",
"subgraph_metadata": [
{
"input_tensor_metadata": [
{
"name": "image",
"description": "Input image to be classified.",
"content": {
"content_properties_type": "ImageProperties",
"content_properties": {
"color_space": "RGB"
}
},
"process_units": [
{
"options_type": "NormalizationOptions",
"options": {
"mean": [
127.5
],
"std": [
127.5
]
...
}
]
}

Aplikasi android

https://github.com/alwa889/Skripsi_H071181007