PROJECT 1
FACE RECOGNITION

Indonesia AI - Computer Vision

PRESENTED BY

CV B - Oppenheimer

Man & Female Classification

Latar Belakang

Kebutuhan Identifikasi Gender

Identifikasi jenis kelamin merupakan aspek penting dalam banyak aplikasi, seperti keamanan, analisis demografi, dan pengalaman pengguna. Face recognition menjadi solusi utama untuk mengenali jenis kelamin seseorang secara otomatis

Meningkatnya Permintaan dari Industri Kecantikan

Industri kecantikan juga dapat memanfaatkan teknologi ini untuk memberikan rekomendasi produk yang lebih sesuai berdasarkan jenis kelamin pelanggan, meningkatkan pengalaman belanja

Timeline

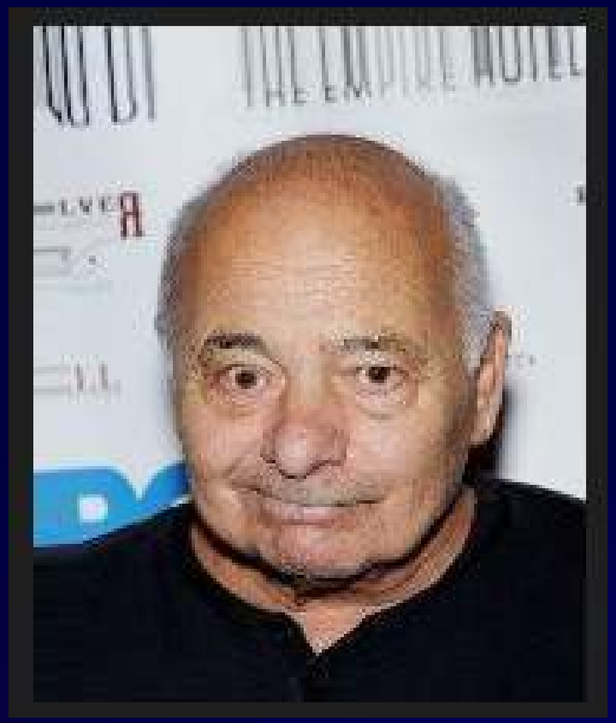
Task	Main Days	Requirements	28/10/2023	29/10/2023	30/10/2023	31/10/2023	01/11/2023	02/11/2023	03/11/2023	04/11/2023	05/11/2023	06/11/2023	07/11/2023	08/11/2023	09/11/2023	10/11/2023
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
Data Collection & Preparation	6															
Data Understanding	6															
Model Selection & Assignment	1	Data Collection & Preparation, Data Understanding														
Research Training, Evaluation, Optimization	4	Model Selection & Assignment														
Conclusion & Presentation Preparation	2	Training, Evaluation, Optimization														
Presentation	1	Presentation Preparation														

PIC Model	
VGG	Hendra, Fatturahman
GoogLeNet	Dani, Yogi
ResNet	Harrison, Fitrah

Dataset

- CelebA
  - 5000 face images
    - 2953 Female
    - 2047 Male
  - list_attribute.txt
    - `Male` column as target

Binary Classification



Model

- VGG
- GoogLeNet
- ResNet

VGG

- Preprocessing
- Arsitektur Model
- Hyperparameter
- Evaluasi

Preprocessing

Change the Feature Value -1 to 0

```
[ ]  df.replace(-1, 0, inplace=True)
```

```
<ipython-input-9-d84944b5799e>:1: S
A value is trying to be set on a co

See the caveats in the documentatio
  df.replace(-1, 0, inplace=True)
```

```
df.head()
```

	Male
000001.jpg	0
000002.jpg	0
000003.jpg	1
000004.jpg	0
000005.jpg	0

Preprocessing

Handle Imbalanced Data with Undersampling

```
[ ]  target_df.value_counts()
```

```
Male
0       2953
1       2047
dtype: int64
```

```
target_df = undersampling(target_df, 'Male')
```

```
1    2047
0    2047
Name: Male, dtype: int64
```

Preprocessing

Split the Dataset Into Train Val Test Split

```
train_df.head()
```

	Male
139406.jpg	1
172061.jpg	1
069813.jpg	0
142502.jpg	0
136281.jpg	0

```
train_df.value_counts()
```

```
Male
1       1433
0       1432
dtype: int64
```

```
val_df.head()
```

	Male
114107.jpg	0
094352.jpg	0
005334.jpg	0
016823.jpg	1
176695.jpg	0

```
val_df.value_counts()
```

```
Male
0       369
1       368
dtype: int64
```

```
test_df.head()
```

	Male
054207.jpg	0
016633.jpg	0
159609.jpg	1
198957.jpg	0
173754.jpg	0

```
test_df.value_counts()
```

```
Male
0       246
1       246
```

Preprocessing

Create Folder Structure

Create Folder Structure

```
[ ]  train_male_path = os.path.join(drive_dataset_path, 'gender/train/male')
     train_female_path = os.path.join(drive_dataset_path, 'gender/train/female')

     val_male_path = os.path.join(drive_dataset_path, 'gender/validation/male')
     val_female_path = os.path.join(drive_dataset_path, 'gender/validation/female')

     test_male_path = os.path.join(drive_dataset_path, 'gender/test/male')
     test_female_path = os.path.join(drive_dataset_path, 'gender/test/female')
```

Preprocessing

create three image data generator objects and normalize the image with rescale = 1/255 for data augmentation and image preprocessing

```
def get_load_data_generator(self, transform=False):
    self.__train_datagen = ImageDataGenerator(
        preprocessing_function=self.preprocess_func,
        rescale=1/255)
    self.__validation_datagen = ImageDataGenerator(
        preprocessing_function=self.preprocess_func,
        rescale=1/255
        )
    self.__test_datagen = ImageDataGenerator(
        preprocessing_function=self.preprocess_func,
        rescale=1/255
        )
```

Preprocessing

Transform Data Train Images

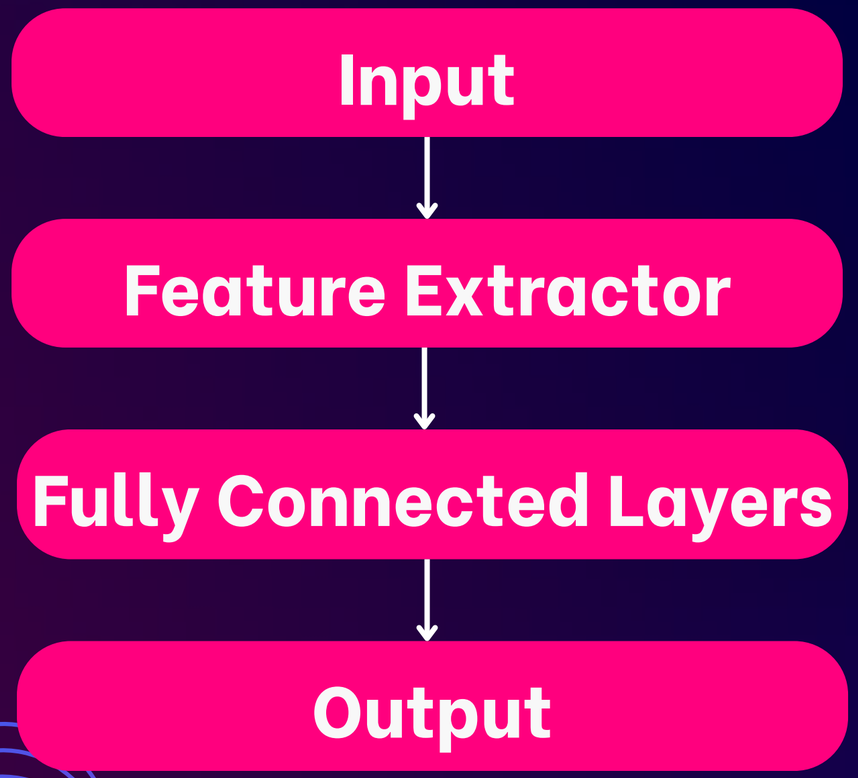
```
        )
    if transform:
        self.__train_datagen = ImageDataGenerator(
            preprocessing_function=self.preprocess_func,
            rescale=1./255,                 # Normalize pixel values
            rotation_range=30,              # Randomly rotate images (e.g., 10 degrees)
            width_shift_range=0.2,          # Randomly shift width
            height_shift_range=0.2,         # Randomly shift height
            brightness_range=[0.7, 1.3],    # Random brightness adjustment
            zoom_range=0.2,                 # Random zoom
            horizontal_flip=True,           # Randomly flip horizontally
            fill_mode='nearest'             # Fill in newly created pixels using the nearest existing pixels
        )
```

Preprocessing

Configure data generators for training, validation, and testing

```python
def preprocess(self, transform=False, train_batch_size=128, val_batch_size=32):
    # self.app.set_config('transform', transform)

    self.get_load_data_generator(transform)
    self.app.train_generator = self.__train_datagen.flow_from_directory(
            self.__train_dir,
            target_size=(self.app.img_size, self.app.img_size),
            batch_size=train_batch_size,
            keep_aspect_ratio=True,
            class_mode='binary')

    self.app.validation_generator = self.__validation_datagen.flow_from_directory(
            self.__val_dir,
            target_size=(self.app.img_size, self.app.img_size),
            batch_size=val_batch_size,
            keep_aspect_ratio=True,
            shuffle=False,
            class_mode='binary')

    self.app.test_generator = self.__test_datagen.flow_from_directory(
            self.__test_dir,
            target_size=(self.app.img_size, self.app.img_size),
            batch_size=val_batch_size,
            keep_aspect_ratio=True,
            shuffle=False,
            class_mode='binary')
```

Arsitektur Model



224 x 224 x 3

VGG16 & VGG19 with pretrained weights imagenet

Flatten & 1 Dense layer 512 neurons activation relu

Dense 1 neuron activation sigmoid
Binary cross entropy loss

VGG16

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 27560769 (105.14 MB)
Trainable params: 12846081 (49.00 MB)
Non-trainable params: 14714688 (56.13 MB)
```

VGG19

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 32870465 (125.39 MB)
Trainable params: 12846081 (49.00 MB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

Hyperparameter Optimization

Bayes

- Transform (True / False)
- Generator batch size (8, 16, 24, 32, 64)
- Model batch size (8, 16, 24, 32, 64)
- Optimizer
  - Adam
  - RMSprop
- Optimizer learning rate
  - 0.001
  - 0.0001

Hyperparameter Optimization

VGG16

epoch/loss	epoch/val_loss	epoch/accuracy	epoch/val_accuracy	Test accuracy ▾	Test precision	Test recall
0.01634	0.1913	0.9997	0.9389	0.9248	0.9372	0.9106
0.02327	0.1857	0.9986	0.9362	0.9228	0.9194	0.9268
0.007381	0.22	1	0.9349	0.9228	0.9262	0.9187

- Transform = False
- Optimizer = Adam
- learning rate = 0.001

- Generator training batch size = 64
- Generator validation batch size = 32
- Model Training batch size = 64
- Model Validation batch size = 32

Hyperparameter Optimization

VGG19

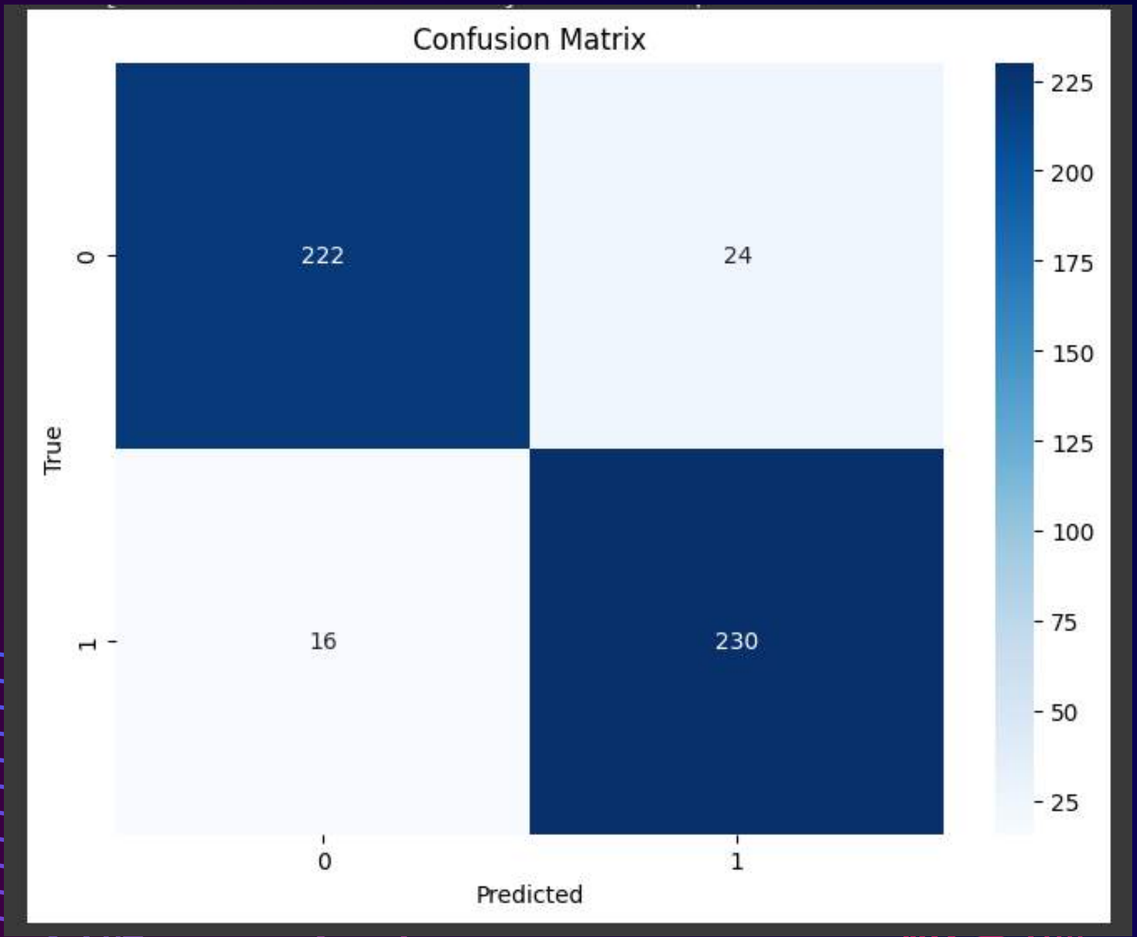
epoch/loss	epoch/val_loss	epoch/accuracy	epoch/val_accuracy	Test accuracy ▾	Test precision	Test recall
0.08696	0.2137	0.9668	0.9172	0.9228	0.9	0.9512
0.01826	0.2459	0.9986	0.9267	0.9167	0.9116	0.9228
0.08999	0.2181	0.9651	0.9281	0.9146	0.8835	0.9553

- Transform = False
- Optimizer = RMSprop
- learning rate = 0.0001
- Generator training batch size = 24
- Generator validation batch size = 8
- Model Training batch size = 64
- Model Validation batch size = 16

Evaluasi

VGG16

492 test data

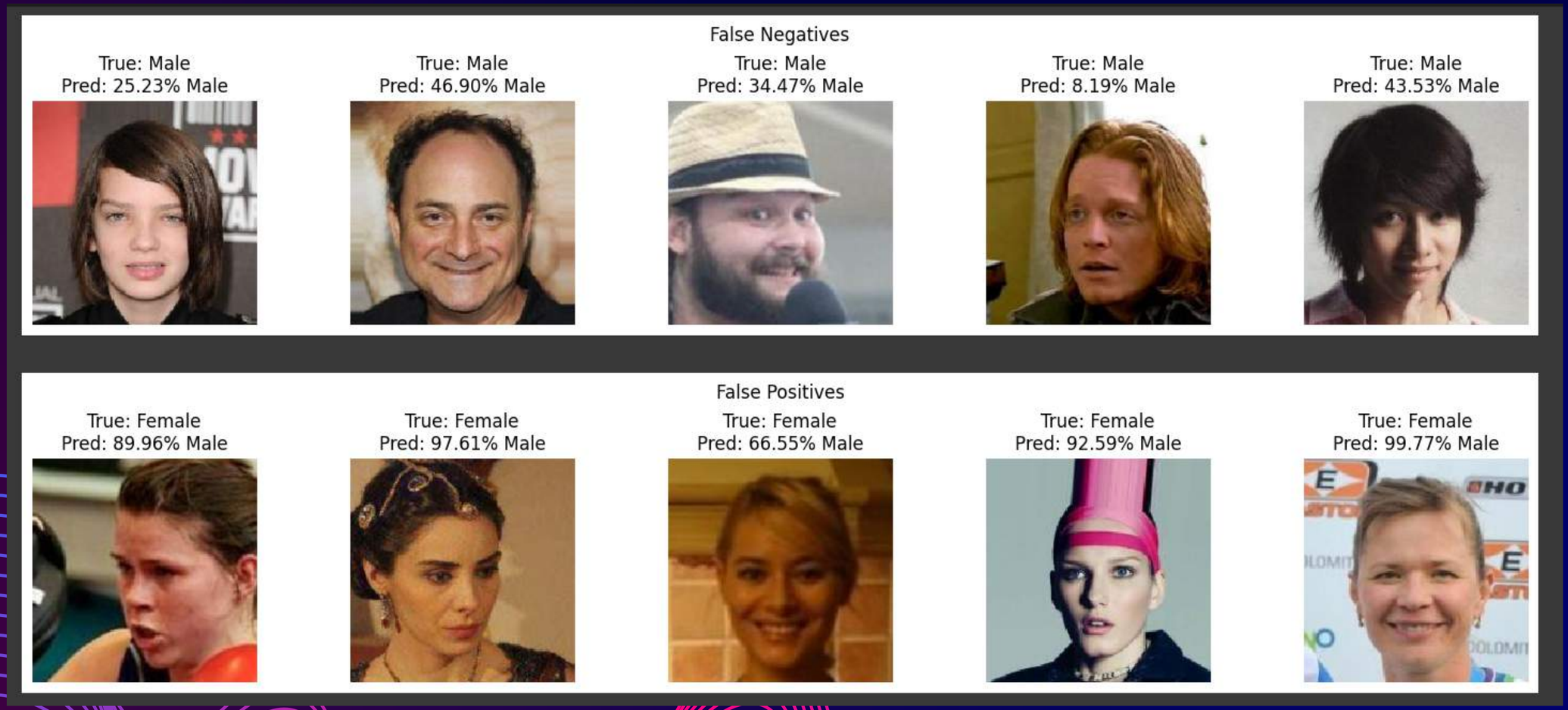



```
single_img_inference_latency(app)
```

```
1/1 [==============================] - 0s 39ms/step
Inference time: 0.08886885643005371 seconds
```

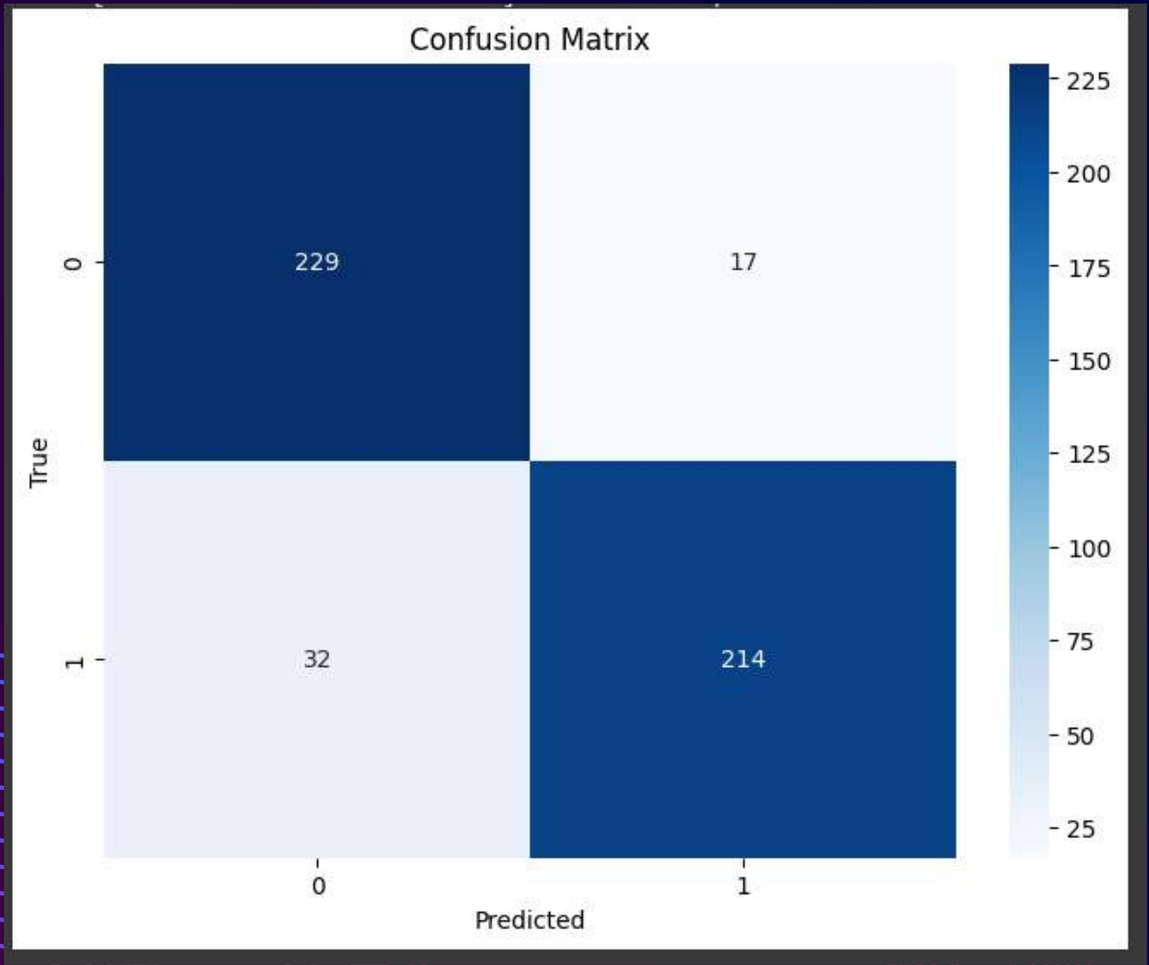
Evaluasi

VGG16



Evaluasi

VGG19 492 test data

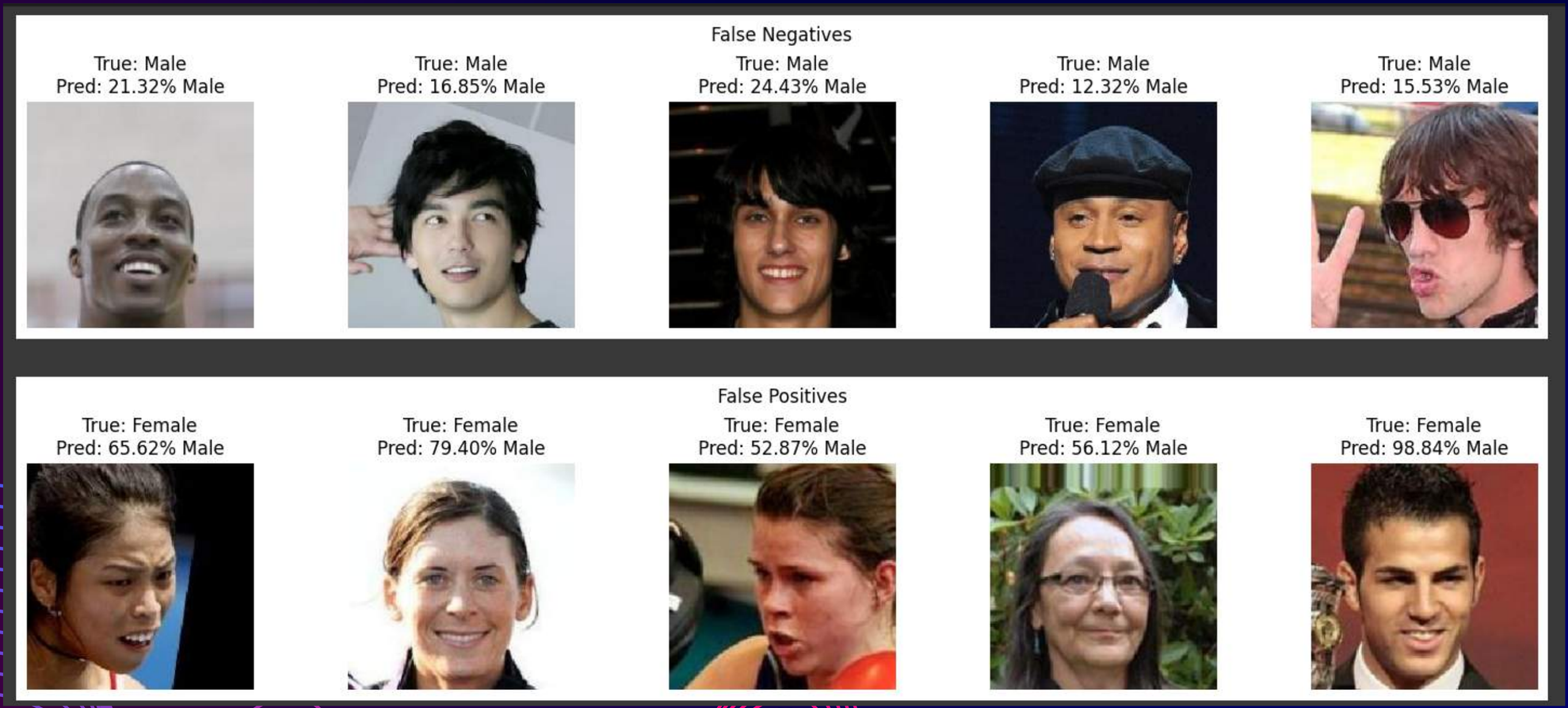



```
single_img_inference_latency(app)
```

```
1/1 [==============================] - 0s 39ms/step
Inference time: 0.14367103576660156 seconds
```

Evaluasi

VGG19



GoogLeNet

- Preprocessing
- Arsitektur Model
- Hyperparameter
- Evaluasi
  - accuracy

Preprocessing

Change the Feature Value -1 to 0

```
data.loc[data['Male'] == -1,'Male'] = 0
data.loc[data['Male'] == 1,'Male'] = 1

col_list = ['image_id', 'Male']
data = data[col_list]

data = data.reset_index(drop=True)

data.head()
```

	image_id	Male
0	000051.jpg	1
1	000052.jpg	1
2	000065.jpg	1
3	000166.jpg	1
4	000198.jpg	0

Preprocessing

Split into train and test

```
# split the data into train and test sets with a 80:20 ratio
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

```
[16] train_data
```

	image_id	Male
4227	171265.jpg	1
4676	189513.jpg	1
800	032356.jpg	0
3671	148050.jpg	0
4193	170130.jpg	0
...	...	...
4426	179336.jpg	0
466	019169.jpg	1
3092	126186.jpg	0
3772	151869.jpg	0
860	035059.jpg	0

4000 rows × 2 columns

Preprocessing

Transform the train dataset

```
transform_train = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
```

Preprocessing

Create Custom Dataset and Data Loader

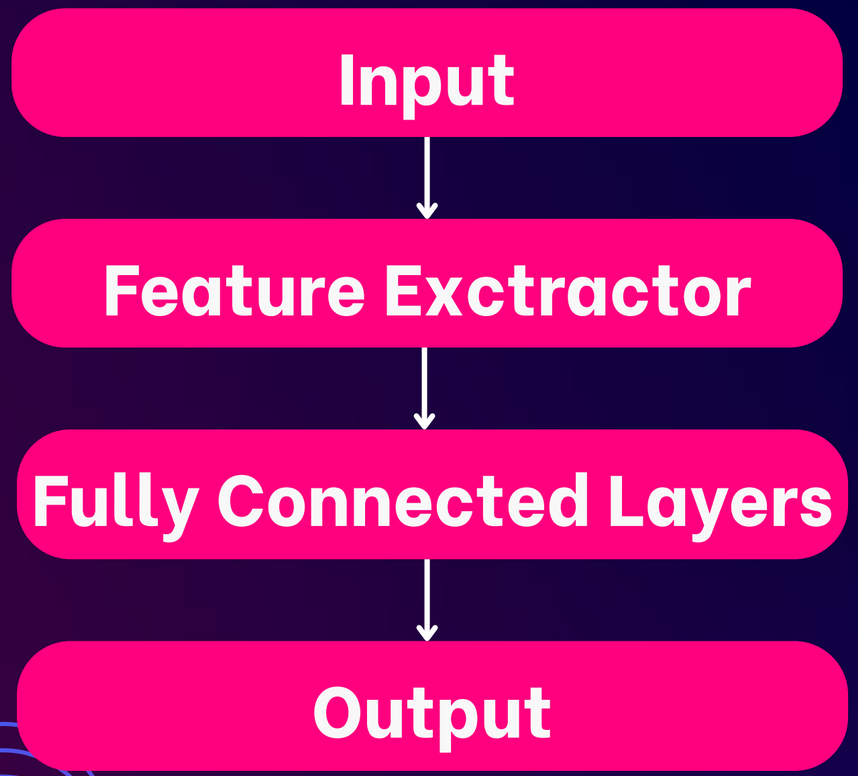
Batch Size 16

```python
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_name = IMG_PATH + '/' + self.dataframe.iloc[idx, 0]
        image = cv2.imread(img_name)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        if self.transform:
            image = self.transform(image)

        label = self.dataframe.iloc[idx, 1]
        return image, torch.tensor(label)
```

```python
BATCH_SIZE = 16
train_dataset = CustomDataset(train_df, transform=transform_train)
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
```

```python
test_dataset = CustomDataset(test_df, transform=transform_test)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

Arsitektur Model



218 x 178 x 3

GoogleNet with pretrained weights

Flatten & 3 Dense layer with relu activation

Dense 1 neuron activation sigmoid
Binary cross entropy loss

Load Model GoogleNet with pretrained weight

```
model = models.googlenet(weights=model.GoogLeNet_Weights.IMAGENET1K_V1)
model.fc=nn.Sequential(
    nn.Linear(in_features=1024,out_features=512),
    nn.ReLU(),
    nn.Linear(in_features=512,out_features=128),
    nn.ReLU(),
    nn.Linear(in_features=128,out_features=32),
    nn.ReLU(),
    nn.Linear(in_features=32,out_features=1),
    nn.Sigmoid()
)
model.to('cuda');
```

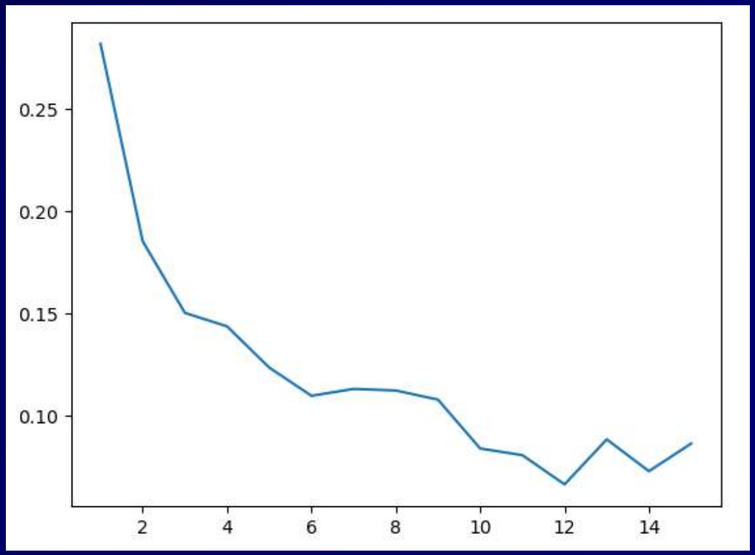
```
        Dropout-195                 [-1, 1024]               0
         Linear-196                  [-1, 512]         524,800
           ReLU-197                  [-1, 512]               0
         Linear-198                  [-1, 128]          65,664
           ReLU-199                  [-1, 128]               0
         Linear-200                   [-1, 32]           4,128
           ReLU-201                   [-1, 32]               0
         Linear-202                    [-1, 1]              33
        Sigmoid-203                    [-1, 1]               0
================================================================
Total params: 6,194,529
Trainable params: 6,194,529
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.44
Forward/backward pass size (MB): 71.30
Params size (MB): 23.63
Estimated Total Size (MB): 95.38
----------------------------------------------------------------
```

```
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

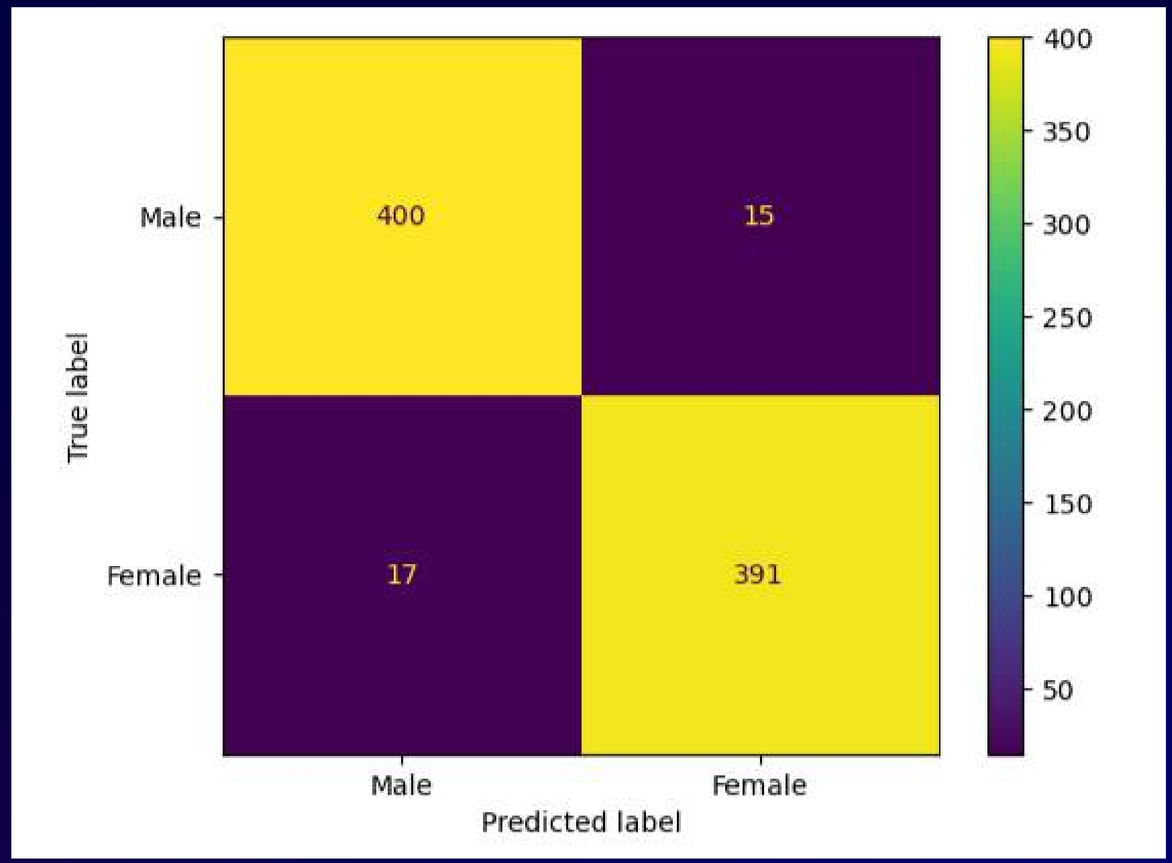
Training and Testing

```python
epochs = 15
model.train()
loss_vals =  []
for epoch in range(epochs):
    running_loss = 0.0
    epoch_loss =  []
    for i, data in enumerate(train_loader, 0):
        # Get the inputs
        inputs, labels = data
        inputs, labels = inputs.to('cuda'), labels.to('cuda')

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward + backward + optimize
        outputs = model(inputs).reshape(-1)
        loss = criterion(outputs, labels.float())
        loss.backward()
        optimizer.step()

        # Print statistics
        epoch_loss.append(loss.item())
        running_loss += loss.item()

    loss_vals.append(sum(epoch_loss)/len(epoch_loss))
    print(f'Epoch: {epoch + 1}, Loss: {running_loss}')


# Test the model
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        images, labels = images.to('cuda'), labels.to('cuda')
        outputs = model(images).reshape(-1)
        predicted = (outputs > 0.5).float()
        total += labels.size(0)
        correct += (predicted == labels.float()).sum().item()

print('Accuracy of the network on the test images: %d %%' % (
    100 * correct / total))

print('Finished Training')
```

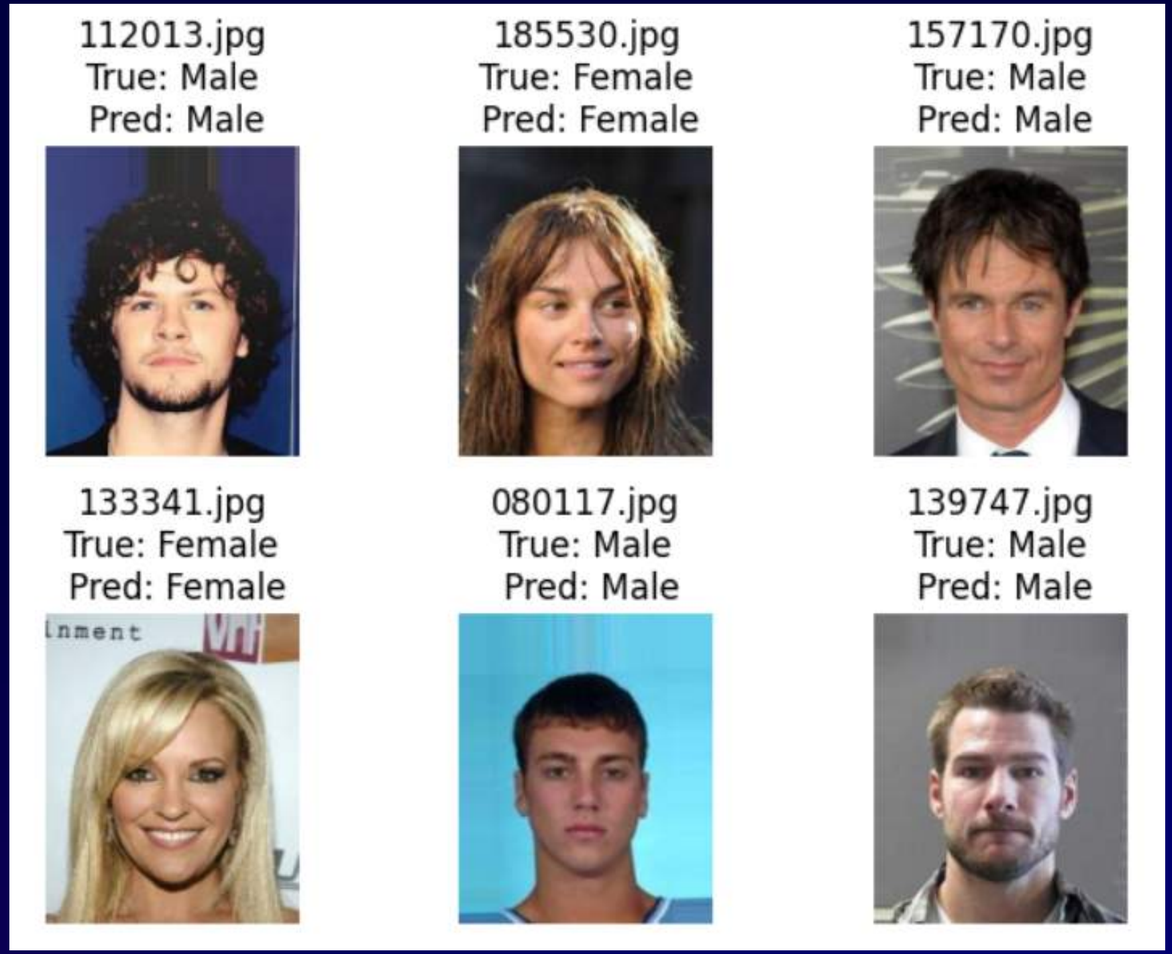
```
Epoch: 1, Loss: 62.806739926338196
Epoch: 2, Loss: 35.93328795861453
Epoch: 3, Loss: 30.24664714373648
Epoch: 4, Loss: 23.93996395211434
Epoch: 5, Loss: 24.083360778633505
Epoch: 6, Loss: 16.12126322754193
Epoch: 7, Loss: 20.023520489688963
Epoch: 8, Loss: 17.945520065593882
Epoch: 9, Loss: 21.853983500332106
Epoch: 10, Loss: 20.12791301868856
Epoch: 11, Loss: 17.578846291376976
Epoch: 12, Loss: 19.40000972955022
Epoch: 13, Loss: 15.142957859905437
Epoch: 14, Loss: 10.631023452500813
Epoch: 15, Loss: 13.95529675357102
Accuracy of the network on the test images: 97 %
Finished Training
```



Confusion Matrix

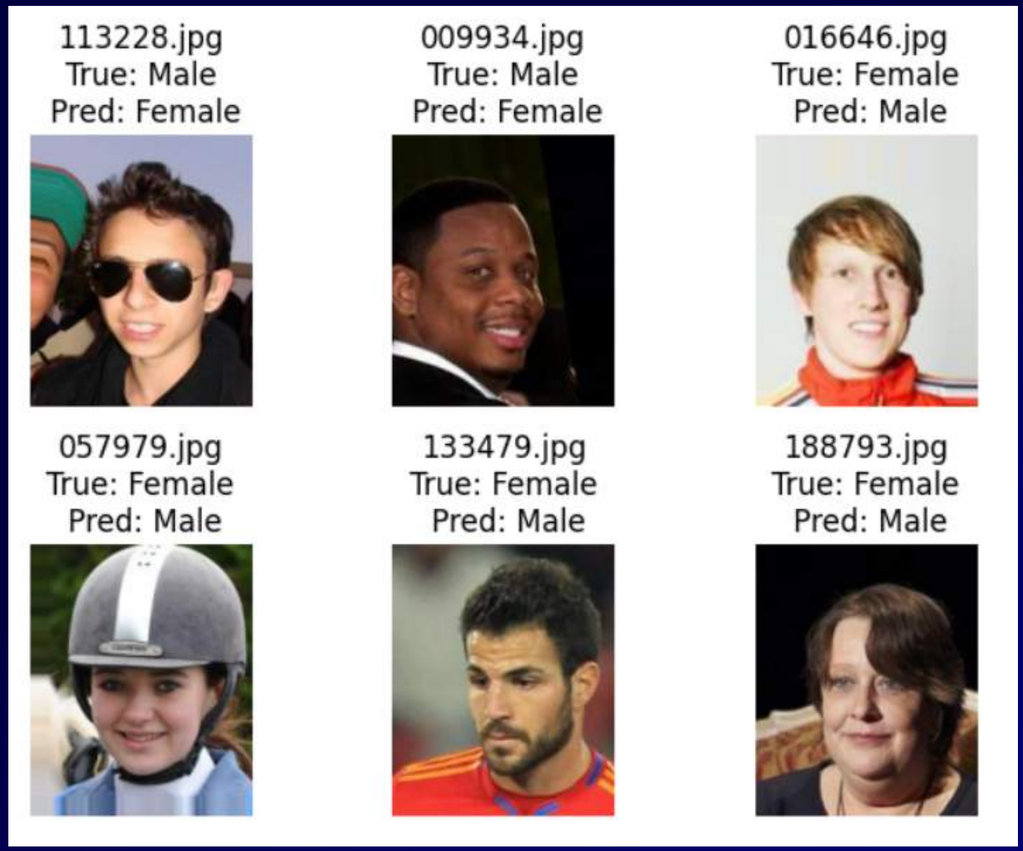


Evaluation


112013.jpg
True: Male
Pred: Male
185530.jpg
True: Female
Pred: Female
157170.jpg
True: Male
Pred: Male
133341.jpg
True: Female
Pred: Female
080117.jpg
True: Male
Pred: Male
139747.jpg
True: Male
Pred: Male

Evaluation

There is some false labelling.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	image_id	5_o_Clock	Arched_Ey	Attractive	Bags_Und	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	Blond_Hai	Blurry	Brown_Ha	Bushy_Eye	Chubby	Double_Ch	Eyeglasses	Goatee	Gray_Hair	Heavy_Ma	High_Chee	Male
133480	133479.jpg	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	image_id	5_o_Clock	Arched_Ey	Attractive	Bags_Und	Bald	Bangs	Big_Lips	Big_Nose	Black_Hair	Blond_Hai	Blurry	Brown_Ha	Bushy_Eye	Chubby	Double_Ch	Eyeglasses	Goatee	Gray_Hair	Heavy_Ma	High_Chee	Male
16647	016646.jpg	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1

ResNet

- Preprocessing
- Arsitektur Model
- Hyperparameter
- Evaluasi
  - accuracy

Preprocessing

```
#pindahkan gambar sesuai gender , 1x aja
for i in range(len(df)):
  if(df.iloc[i]['Male']==1):
    print('Male= ',df.iloc[i]['image_id'])
    shutil.copy((os.path.join(IMG_PATH,(df.iloc[i]['image_id']))), male_folder_path)
  else:
    print('Female= ',df.iloc[i]['image_id'])
    shutil.copy((os.path.join(IMG_PATH,(df.iloc[i]['image_id']))), female_folder_path)
```

```
Male=  000051.jpg
Male=  000052.jpg
Male=  000065.jpg
Male=  000166.jpg
```

```
[10] # Generate dataset

     image_size = (180,180)
     batch_size = 128#32

     train_ds,val_ds =tf.keras.utils.image_dataset_from_directory(
         preprosess_dataset,
         validation_split=0.2,
         subset='both',
         seed = 123,
         image_size=image_size,
         batch_size=batch_size,
     )
```

```
Found 5000 files belonging to 2 classes.
Using 4000 files for training.
Using 1000 files for validation.
```

Preprocessing

```
normalization_layer = layers.Rescaling(1./255)

normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 0.9755738
```

```
img_height = 180
img_width = 180
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",input_shape=(img_height,img_width,3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

```
[ ] # Apply `data_augmentation` to the training images.
    train_ds = train_ds.map(  #cara ke-2
        lambda img, label: (data_augmentation(img), label),
        num_parallel_calls=tf.data.AUTOTUNE,
    )
    # Prefetching samples in GPU memory helps maximize GPU utilization.
    train_ds = train_ds.prefetch(tf.data.AUTOTUNE)
    val_ds = val_ds.prefetch(tf.data.AUTOTUNE)
```

Model - Resnet50 dengan activation Sigmoid

```
[ ] headModel = base_model1.output
    headModel = Flatten()(headModel)
    headModel = Dense(256, activation='relu', name='fc1',kernel_initializer=glorot_uniform(seed=0))(headModel)
    headModel = Dense(128, activation='relu', name='fc2',kernel_initializer=glorot_uniform(seed=0))(headModel)
    headModel = Dense( 1,activation='sigmoid', name='fc3',kernel_initializer=glorot_uniform(seed=0))(headModel)

[ ] model = Model(inputs=base_model1.input, outputs=headModel)

[ ] model.summary()
```

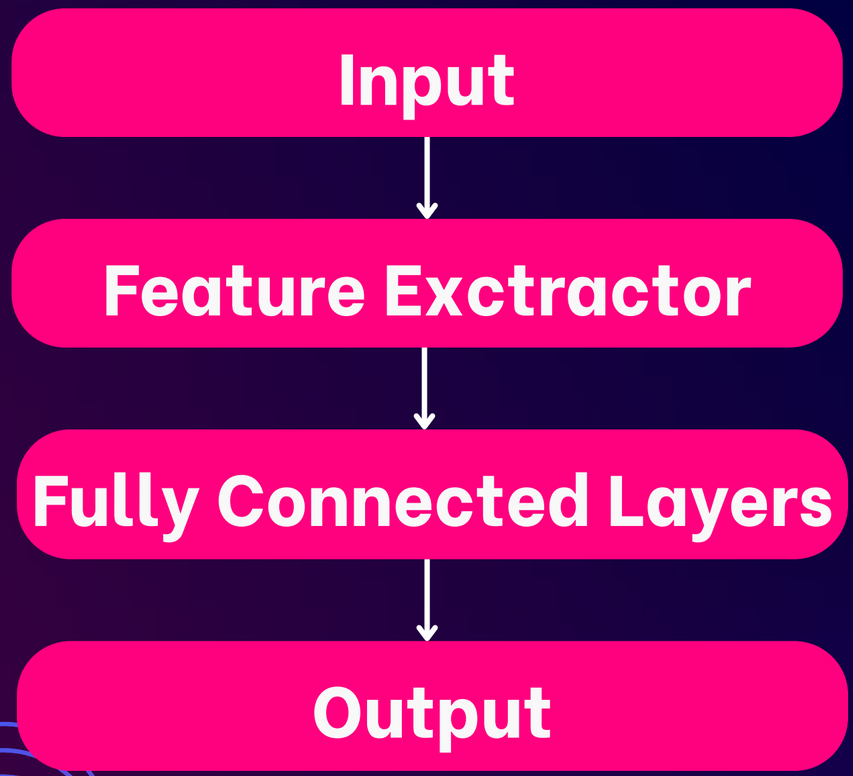
bn5a_branch2a (BatchNormalization)	(None, 6, 6, 512)	2048	['res5a_branch2a[0][0]']
activation_138 (Activation)	(None, 6, 6, 512)	0	['bn5a_branch2a[0][0]']
res5a_branch2b (Conv2D)	(None, 6, 6, 512)	2359808	['activation_138[0][0]']
bn5a_branch2b (BatchNormalization)	(None, 6, 6, 512)	2048	['res5a_branch2b[0][0]']
activation_139 (Activation)	(None, 6, 6, 512)	0	['bn5a_branch2b[0][0]']
res5a_branch2c (Conv2D)	(None, 6, 6, 2048)	1050624	['activation_139[0][0]']

Model - Resnet50 dengan activation Softmax

```
base_model1 = ResNet50(input_shape=(180, 180, 3))

headModel = base_model1.output
headModel = Flatten()(headModel)
headModel = Dense(256, activation='relu', name='fc1',kernel_initializer=glorot_uniform(seed=0))(headModel)
headModel = Dense(128, activation='relu', name='fc2',kernel_initializer=glorot_uniform(seed=0))(headModel)
headModel = Dense( 1,activation='softmax', name='fc3',kernel_initializer=glorot_uniform(seed=0))(headModel)

model = Model(inputs=base_model1.input, outputs=headModel)

model.summary()
```

res5b_branch2b (Conv2D)	(None, 6, 6, 512)	2359808	['activation_43[0][0]']

Arsitektur Model



180x180x3

RESnet50 with pretrained weights

Flatten & 2 Dense layer with relu activation

Dense 1 neuron activation sigmoid
Binary cross entropy loss

Train Model- Resnet50 dengan activation Sigmoid

```
[ ] opt = SGD(lr=1e-3, momentum=0.9)
    model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])

    WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e

[ ] es=EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=20)

[ ] mc = ModelCheckpoint(os.path.join(BASIC_PATH,'best_model1.h5'), monitor='val_accuracy', mode='max',
        save_best_only=True)

    H = model.fit_generator(train_ds,validation_data=val_ds,epochs=100,verbose=1,callbacks=[mc,es])

    =============] - 42s 280ms/step - loss: 0.3463 - accuracy: 0.8495 - val_loss: 0.1892 - val_accuracy: 0.9290
    =============] - 32s 252ms/step - loss: 0.2306 - accuracy: 0.9038 - val_loss: 0.1987 - val_accuracy: 0.9080
    =============] - 30s 242ms/step - loss: 0.2093 - accuracy: 0.9140 - val_loss: 0.1898 - val_accuracy: 0.9180
    =============] - 34s 271ms/step - loss: 0.1929 - accuracy: 0.9155 - val_loss: 0.1678 - val_accuracy: 0.9390
    =============] - 38s 304ms/step - loss: 0.1816 - accuracy: 0.9280 - val_loss: 0.1766 - val_accuracy: 0.9420
    =============] - 31s 245ms/step - loss: 0.1708 - accuracy: 0.9327 - val_loss: 0.1547 - val_accuracy: 0.9380
    =============] - 30s 240ms/step - loss: 0.1639 - accuracy: 0.9360 - val_loss: 0.1721 - val_accuracy: 0.9400
    =============] - 36s 283ms/step - loss: 0.1533 - accuracy: 0.9340 - val_loss: 0.1498 - val_accuracy: 0.9480
```

```
32/32 [==============================] - 23s 710ms/step - loss: 0.0274 - accuracy: 0.9910 - val_l
Epoch 64/150
32/32 [==============================] - 24s 737ms/step - loss: 0.0258 - accuracy: 0.9902 - val_l
Epoch 64: early stopping
```

```
loss, acc = model_resnet50.evaluate(val_ds)  # returns loss and metrics
print("loss: %.2f" % loss)
print("acc: %.2f" % acc)
```

```
8/8 [==============================] - 1s 93ms/step - loss: 0.2627 - accuracy: 0.9400
loss: 0.26
acc: 0.94
```

TrainModel - Resnet50 dengan activation Softmax

```
[46] opt = SGD(lr=1e-3, momentum=0.9)
     model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])

     WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer

[47] es=EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=20)

[48] mc = ModelCheckpoint(os.path.join(BASIC_PATH,'best_model1.h5'), monitor='val_accuracy', mode='max',
         save_best_only=True)

[49] H = model.fit_generator(train_ds,validation_data=val_ds,epochs=150,verbose=1,callbacks=[mc,es])

     Epoch 1/150
     <ipython-input-49-8a6227a60bab>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed ir
       H = model.fit_generator(train_ds,validation_data=val_ds,epochs=150,verbose=1,callbacks=[mc,es])
     32/32 [==============================] - 77s 2s/step - loss: 1.1080 - accuracy: 0.4042 - val_loss: 0.6885
     Epoch 2/150
     32/32 [==============================] - 37s 1s/step - loss: 0.6696 - accuracy: 0.4042 - val_loss: 0.6902
     Epoch 3/150
```

```
[50] loss, acc = model.evaluate(val_ds)  # returns loss and metrics
     print("loss: %.2f" % loss)
     print("acc: %.2f" % acc)

     8/8 [==============================] - 2s 230ms/step - loss: 0.6175 - accuracy: 0.4300
     loss: 0.62
     acc: 0.43
```

Resnet50 dengan activation Sigmoid menghasilkan akurasi yg lebih tinggi

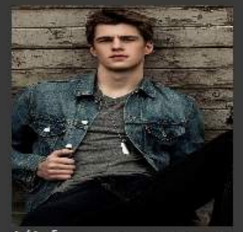
Prediksi - Resnet50 dengan activation Sigmoid

Pria

```
predict_image('pria1.jpg')
```

```
1/1 [==============================] - 0s 37ms/step
This image is 18.78% female and 81.22% male.
```

```
[44] predict_image('pria.jpg')
```



```
1/1 [==============================] - 0s 26ms/step
This image is 79.74% female and 20.26% male.
```

```
[45] predict_image('pria2.jpg')
```

```
1/1 [==============================] - 0s 45ms/step
This image is 0.00% female and 100.00% male.
```

```
[53] predict_image('pria4.jpg')
```

```
1/1 [==============================] - 0s 76ms/step
This image is 0.00% female and 100.00% male.
```

prediksi salah

Wanita

```
[46] predict_image('wanita.jpg')
```

```
1/1 [==============================] - 0s 37ms/step
This image is 100.00% female and 0.00% male.
```

```
[47] predict_image('wanita2.jpg')
```

```
1/1 [==============================] - 0s 64ms/step
This image is 100.00% female and 0.00% male.
```

```
[49] predict_image('wanita4.jpg')
```

```
1/1 [==============================] - 0s 90ms/step
This image is 6.05% female and 93.95% male.
```

```
predict_image('wanita5.jpg')
```

```
1/1 [==============================] - 0s 53ms/step
This image is 93.73% female and 6.27% male.
```

prediksi salah

Model Comparison

Model	Test Accuracy	Weights Size
VGG16	92.48%	105 MB
VGG19	92.28%	125 MB
GoogLeNet	97%	95 MB
ResNet	94.80%	108 MB

Kesimpulan dan Saran

- Semua model mampu memberikan hasil akurasi cukup tinggi **> 90%**
- Model terbaik dalam project ini **GoogLeNet** dengan akurasi test terbesar **97%** weights size terkecil **95 MB**
- Untuk improve hasil, dataset bisa ditambah variasi image wajah dengan lebih banyak data dari sudut pandang yang berbeda