DAFTAR PUSTAKA

Agarwal, S., Farid, H., El-Gaaly, T., & Lim, S. N. (2020). Detecting Deep-Fake Videos from Appearance and Behavior. *2020 IEEE International Workshop on Information Forensics and Security, WIFS 2020*. https://doi.org/10.1109/WIFS49906.2020.9360904

Anjani, I. A., Pratiwi, Y. R., & Norfa Bagas Nurhuda, S. (2021). Implementation of Deep Learning Using Convolutional Neural Network Algorithm for Classification Rose Flower. *Journal of Physics: Conference Series*, *1842*(1). https://doi.org/10.1088/1742-6596/1842/1/012002

Annapurani, K., & Ravilla, D. (2019). CNN based image classification model. *International Journal of Innovative Technology and Exploring Engineering*, *8*(11 Special Issue), 1106–1114. https://doi.org/10.35940/ijitee.K1225.09811S19

Ayunita, D., Arief, D. S., & Mirdanies, M. (2012). Pencitraan dan Pemrograman Berdasarkan Perhitungan Aplikasi Volume Paket Logistik. *Jom FTEKNIK*, *6*, 1–7.

Brunetti, A., Buongiorno, D., Trotta, G. F., & Bevilacqua, V. (2018). Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. *Neurocomputing*, *300*, 17–33. https://doi.org/10.1016/j.neucom.2018.01.092

Caelen, O. (2017). A Bayesian interpretation of the confusion matrix. *Annals of Mathematics and Artificial Intelligence*, *81*(3–4), 429–450. https://doi.org/10.1007/s10472-017-9564-8

Deepfake Detection Challenge. (2020). Retrieved from Https://www.kaggle.com website: https://www.kaggle.com/competitions/deepfake-detection-challenge/overview/evaluation

Guera, D., & Delp, E. J. (2019). Deepfake Video Detection Using Recurrent Neural Networks. *Proceedings of AVSS 2018 - 2018 15th IEEE International Conference on Advanced Video and Signal-Based Surveillance*. https://doi.org/10.1109/AVSS.2018.8639163

Lattifia, T., Wira, P., Kadek, N., & Rusjayanthi, D. (2022). *Model Prediksi Cuaca Menggunakan Metode LSTM*. *3*(1).

Li, X., Li, S., Li, J., Yao, J., & Xiao, X. (2021). Detection of fake-video uploaders on social media using Naive Bayesian model with social cues. *Scientific Reports*, *11*(1), 1–11. https://doi.org/10.1038/s41598-021-95514-5

Megawan, S., Lestari, W. S., & Halim, A. (2022). *Deteksi Non-Spoofing Wajah pada Video secara Real Time Menggunakan Faster R-CNN*. *3*(3).

https://doi.org/10.47065/josh.v3i3.1519

Mulyawan, H., Samsono, M. Z. H., & Setiawardhana. (2011). *Identifikasi Dan Tracking Objek Berbasis Image*. 1–5. Retrieved from http://repo.pens.ac.id/1324/1/Paper_TA_MBAH.pdf

Muttaqin, F. A., & Bachtiar, A. M. (2016). Implementasi Teks Mining Pada Aplikasi Pengawasan Penggunaan Internet Anak "Dodo Kids Browser." *Jurnal Ilmiah Komputer Dan Informatika*, 1–8.

Naiemi, F., Ghods, V., & Khalesi, H. (2019). An efficient character recognition method using enhanced HOG for spam image detection. *Soft Computing*, *23*(22), 11759–11774. https://doi.org/10.1007/s00500-018-03728-z

Najemi, A., Munandar, T. I., & Prayudi, A. H. (2021). Bahaya penyampaian berita bohong melalui media sosial. *Jurnal Karya Abdi Masyarakat*, *5*(3), 575–582. Retrieved from https://online-journal.unja.ac.id/JKAM/article/view/16646

Nakisa, B., Rastgoo, M. N., Rakotonirainy, A., Maire, F., & Chandran, V. (2018). Long short term memory hyperparameter optimization for a neural network based emotion recognition framework. *IEEE Access*, *6*(September), 49325–49338. https://doi.org/10.1109/ACCESS.2018.2868361

Pan, D., Sun, L., Wang, R., Zhang, X., & Sinnott, R. O. (2020). Deepfake Detection through Deep Learning. *Proceedings - 2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, BDCAT 2020*, 134–143. https://doi.org/10.1109/BDCAT50828.2020.00001

Pant, G., Yadav, D. P., & Gaur, A. (2020). ResNeXt convolution neural network topology-based deep learning model for identification and classification of Pediastrum. *Algal Research*, *48*(April), 101932. https://doi.org/10.1016/j.algal.2020.101932

Rahmadhany, A., Aldila Safitri, A., & Irwansyah, I. (2021). Fenomena Penyebaran Hoax dan Hate Speech pada Media Sosial. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *3*(1), 30–43. https://doi.org/10.47233/jteksis.v3i1.182

Ranjan, P., Patil, S., & Kazi, F. (2020). Improved generalizability of deep-fakes detection using transfer learning based CNN framework. *Proceedings - 3rd International Conference on Information and Computer Technologies, ICICT 2020*, 86–90. https://doi.org/10.1109/ICICT50521.2020.00021

Religia, Y. (2019). *Feature Extraction untuk Klasifikasi Pengenalan Wajah Menggunakan Support Vector Machine dan K-Nearest Neighbor*. *14*(September), 85–92.

Saleem, M. A., Senan, N., Wahid, F., Aamir, M., Samad, A., & Khan, M. (2022). Comparative Analysis of Recent Architecture of Convolutional Neural Network. *Mathematical Problems in Engineering*, *2022*. https://doi.org/10.1155/2022/7313612

Saputro, I. W., & Sari, B. W. (2020). Naïve Bayes Algorithm Performance Test for Student Study Prediction. *Creative Information Technology Journal*, *6*(1), 1.

Sindar, A. (2018). Optical Systems and Digital Image Acquisition. *Jurnal Insitusi Politeknik Ganesha Medan Juripol*, *1*(1), 61–67.

Suganthi, S. T., Ayoobkhan, M. U. A., Kumar, V. K., Bacanin, N., Venkatachalam, K., Stepán, H., & Pavel, T. (2022). Deep learning model for deep fake face recognition and detection. *PeerJ Computer Science*, *8*, 1–20. https://doi.org/10.7717/PEERJ-CS.881

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Tech, V., Seth, A., & Tech, V. (2021). *Detection of Deep-fakes in Videos using CNN and Transformers Detection of Deep-fakes in Videos using CNN and Transformers*. (July). https://doi.org/10.13140/RG.2.2.23238.60480

Victoria, O., & Solihin, I. P. (2018). Pendeteksi Wajah Secara Realtime Menggunakan Metode Eigenface. *SEINASI-KESI (Seminar Nasional Informatika, Sistem Informasi Dan Keamanan Siber)*, 126–131.

Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., & Zhou, J. (2018). CosFace: Large Margin Cosine Loss for Deep Face Recognition. *Cvpr*, 5265–5274.

Westerlund, M. (2019). The emergence of deepfake technology: A review. *Technology Innovation Management Review*, *9*(11), 39–52. https://doi.org/10.22215/TIMREVIEW/1282

Wiranda, L., & Sadikin, M. (2019). Penerapan Long Short Term Memory Pada Data Time Series Untuk Memprediksi Penjualan Produk Pt. Metiska Farma. *Jurnal Nasional Pendidikan Teknik Informatika*, *8*(3), 184–196.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-Janua*, 5987–5995. https://doi.org/10.1109/CVPR.2017.634

Zhang, Q., Zhang, M., Chen, T., Sun, Z., Ma, Y., & Yu, B. (2019). Recent advances in convolutional neural network acceleration. *Neurocomputing*, *323*, 37–51. https://doi.org/10.1016/j.neucom.2018.09.038

LAMPIRAN

LAMPIRAN

Lampiran 1 *Dataset* video dalam bentuk *file* .mp4

Lampiran 2 *Splitting* video

1. *Split* 10 *frames*

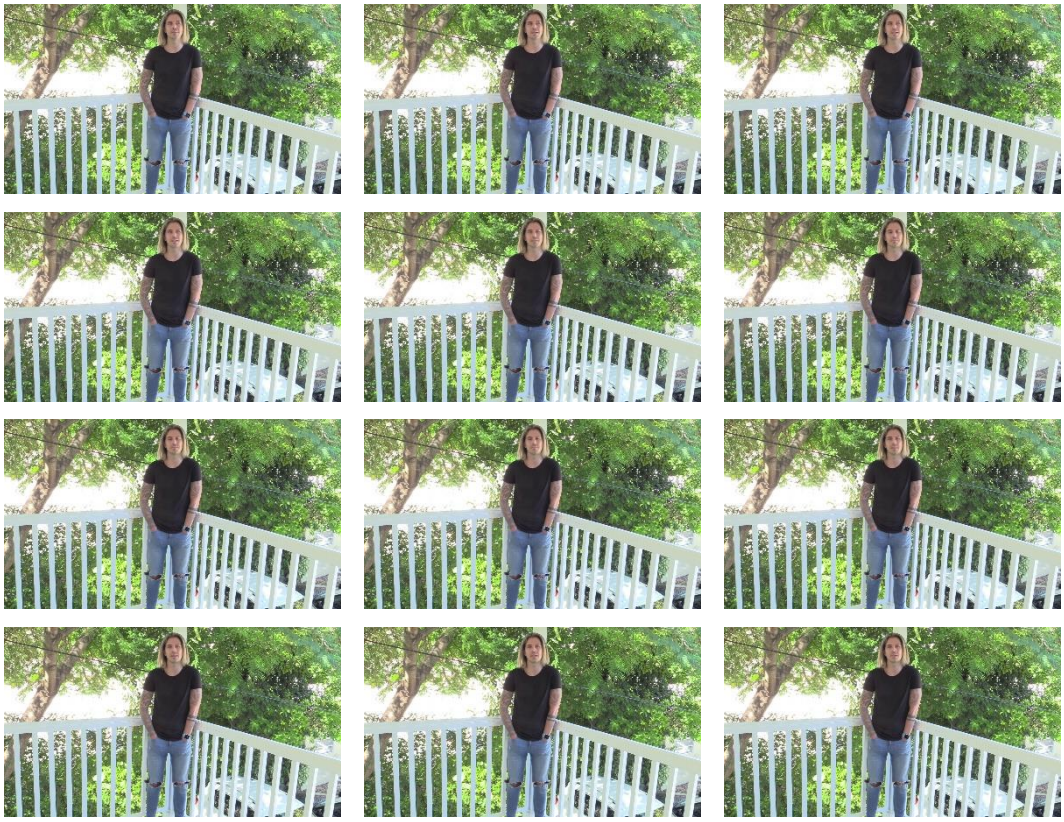
*Preprossed*_1 *Preprossed*_2 *Preprossed*_3

*Preprossed*_4 *Preprossed*_5 *Preprossed*_6

*Preprossed*_7 *Preprossed*_8 *Preprossed*_9

*Preprossed*_10

2. *Split* 60 *frames*



Lampiran 3 *Face detection*

Lampiran 4 *Cropped face*

1. 10 *frames*

*Cropped_face_*1 *Cropped_face_*2 *Cropped_face_*3 *Cropped_face_*4


*Cropped_face*_5


*Cropped_face*_6

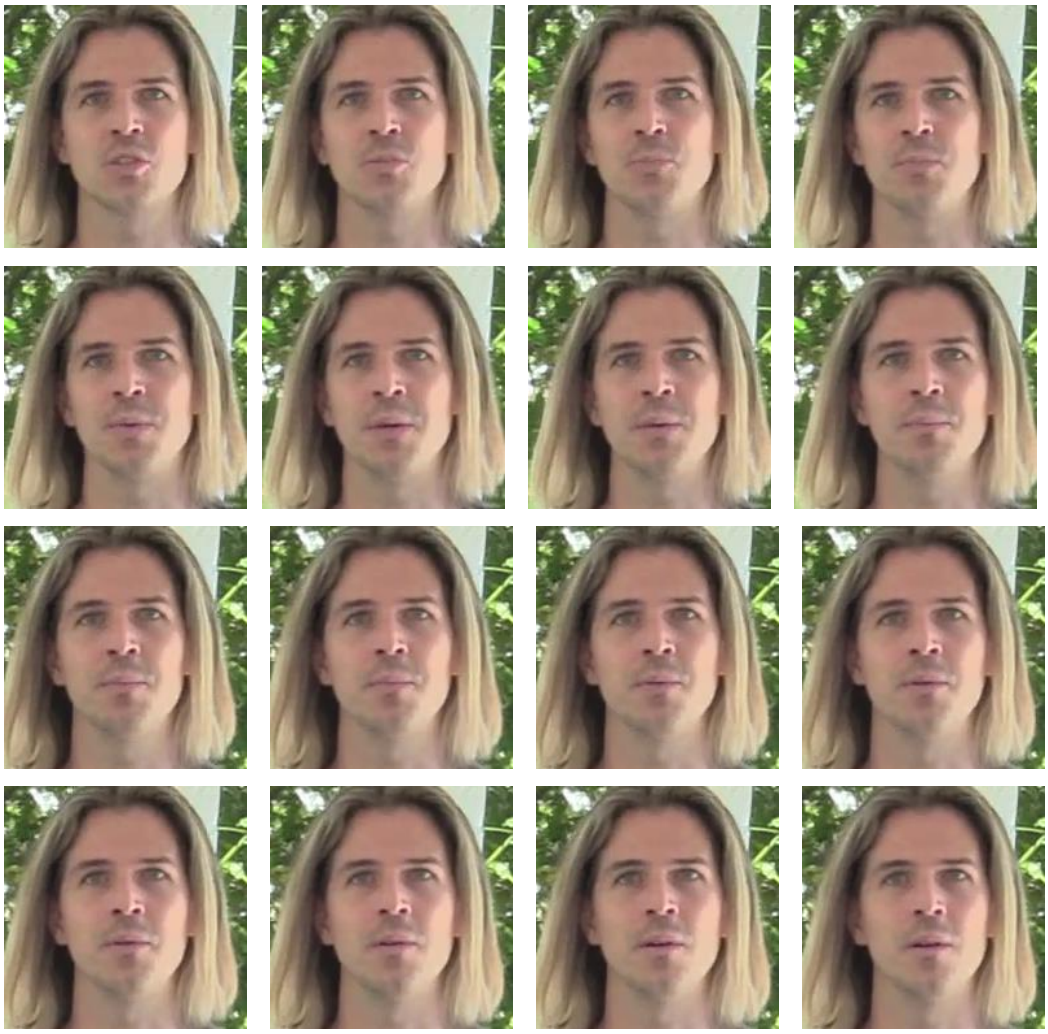

*Cropped_face*_7

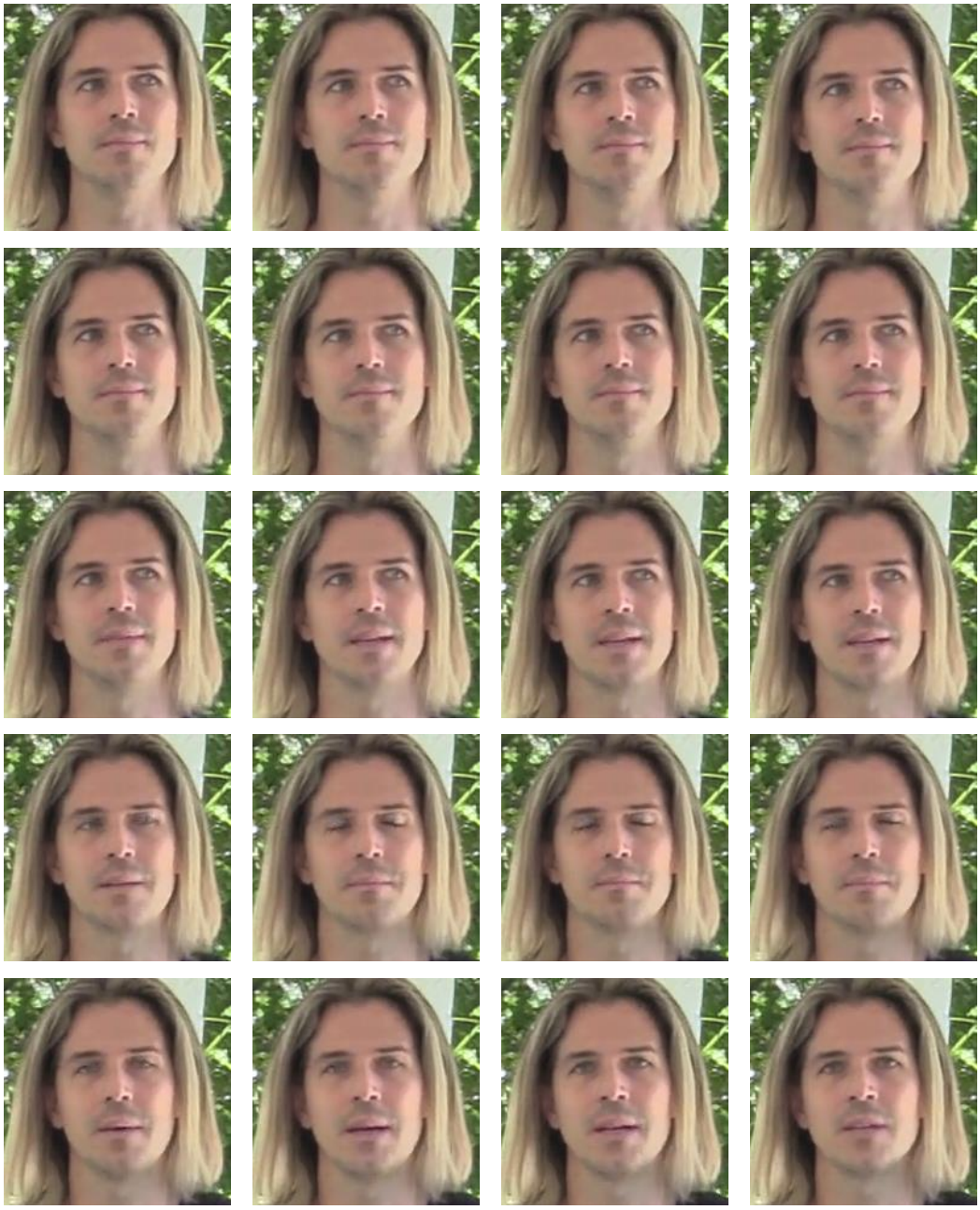

*Cropped_face*_8


*Cropped_face*_9

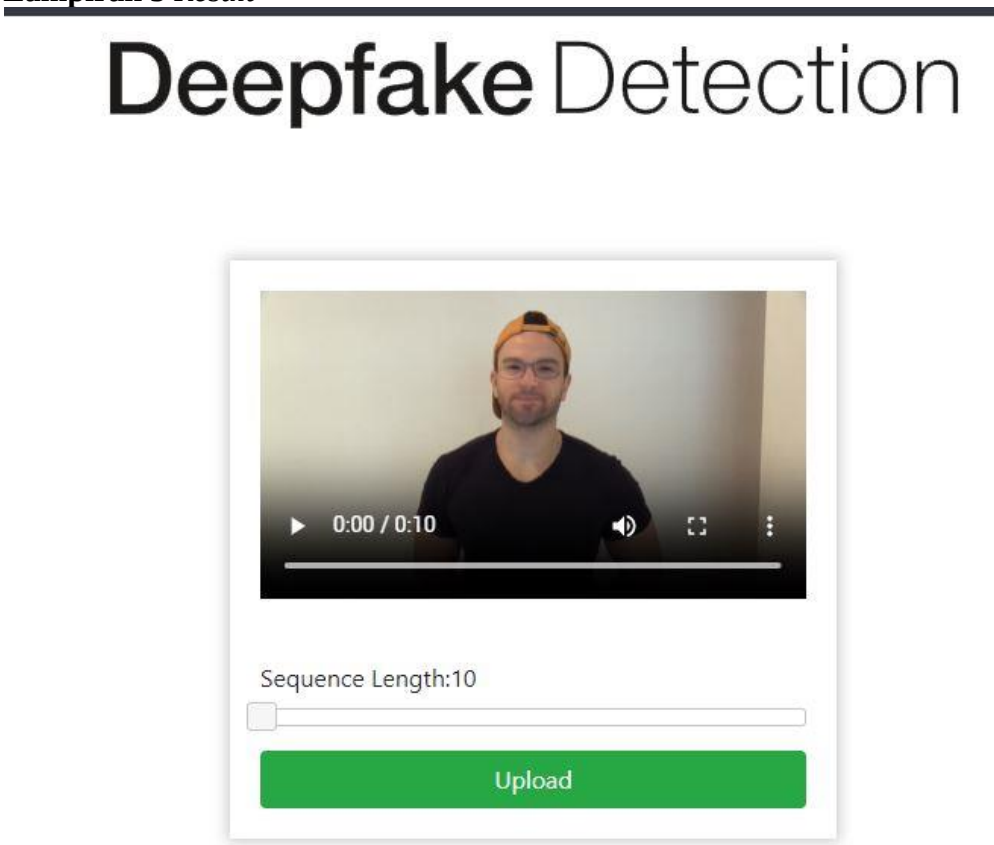

*Cropped_face*_10

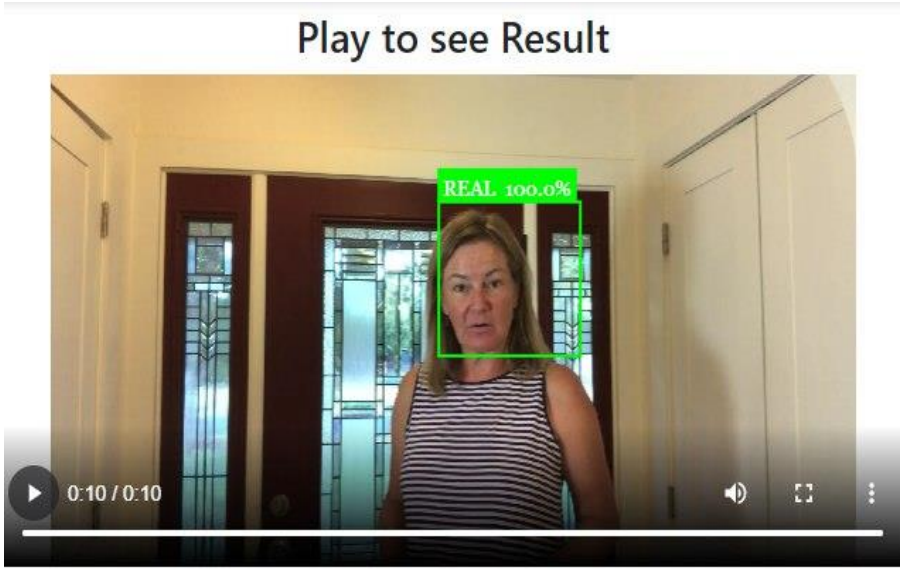
2. 60 *frames*

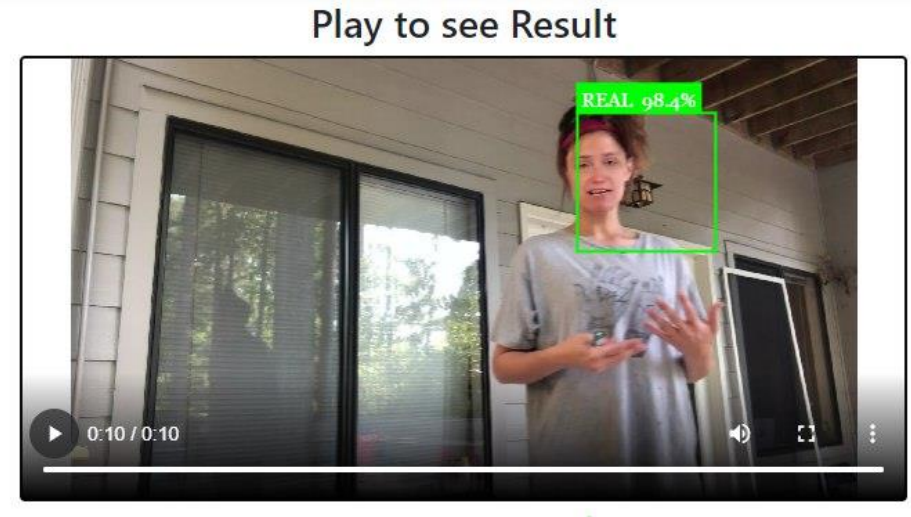




Lampiran 5 *Result*




Play to see Result
REAL 100.0%
0:10 / 0:10


Play to see Result
REAL 98.4%
0:10 / 0:10


Play to see Result
FAKE 100.0%
0:02 / 0:10

Lampiran 6 *Source code*

```
from django.shortcuts import render, redirect
import torch
import torchvision
from torchvision import transforms, models
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
from torch.autograd import Variable
import time
import sys
from torch import nn
import json
import glob
import copy
from torchvision import models
import shutil
from PIL import Image as pImage
import time
from django.conf import settings
from .forms import VideoUploadForm

index_template_name = 'index.html'
predict_template_name = 'predict.html'

im_size = 112
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
sm = nn.Softmax()
inv_normalize =  transforms.Normalize(mean=-
1*np.divide(mean,std),std=np.divide([1,1,1],std))
```

```
train_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_si
ze,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mea
n,std)])

class Model(nn.Module):

    def __init__(self, num_classes,latent_dim= 2048,
lstm_layers=1 , hidden_dim = 2048, bidirectional = False):
        super(Model, self).__init__()
        model = models.resnext50_32x4d(pretrained = True)
        self.model = nn.Sequential(*list(model.children())[:-2])
        self.lstm = nn.LSTM(latent_dim,hidden_dim,
lstm_layers,  bidirectional)
        self.relu = nn.LeakyReLU()
        self.dp = nn.Dropout(0.4)
        self.linear1 = nn.Linear(2048,num_classes)
        self.avgpool = nn.AdaptiveAvgPool2d(1)

    def forward(self, x):
        batch_size,seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length, c, h, w)
        fmap = self.model(x)
        x = self.avgpool(fmap)
        x = x.view(batch_size,seq_length,2048)
        x_lstm,_ = self.lstm(x,None)
        return fmap,self.dp(self.linear1(x_lstm[:,-1,:]))

class validation_dataset(Dataset):
    def __init__(self,video_names,sequence_length=60,transform =
None):
        self.video_names = video_names
        self.transform = transform
```

```
        self.count = sequence_length

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

    def __getitem__(self,idx):
        video_path = self.video_names[idx]
        frames = []
        a = int(100/self.count)
        first_frame = np.random.randint(0,a)
        for i,frame in
enumerate(self.frame_extract(video_path)):
            #if(i % a == first_frame):
            faces = face_recognition.face_locations(frame)
            try:
              top,right,bottom,left = faces[0]
              frame = frame[top:bottom,left:right,:]
            except:
              pass
            frames.append(self.transform(frame))
            if(len(frames) == self.count):
                break
        """
        for i,frame in
enumerate(self.frame_extract(video_path)):
            if(i % a == first_frame):
                frames.append(self.transform(frame))
        """
        # if(len(frames)<self.count):
        #   for i in range(self.count-len(frames)):
        #         frames.append(self.transform(frame))
        #print("no of frames", self.count)
        frames = torch.stack(frames)
        frames = frames[:self.count]
        return frames.unsqueeze(0)
```

```
    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
      while success:
          success, image = vidObj.read()
          if success:
              yield image

def im_convert(tensor, video_file_name):
    """ Display a tensor as an image. """
    image = tensor.to("cpu").clone().detach()
    image = image.squeeze()
    image = inv_normalize(image)
    image = image.numpy()
    image = image.transpose(1,2,0)
    image = image.clip(0, 1)
    # This image is not used
    # cv2.imwrite(os.path.join(settings.PROJECT_DIR,
'uploaded_images', video_file_name+'_convert_2.png'),image*255)
    return image

def im_plot(tensor):
    image = tensor.cpu().numpy().transpose(1,2,0)
    b,g,r = cv2.split(image)
    image = cv2.merge((r,g,b))
    image = image*[0.22803, 0.22145, 0.216989] +  [0.43216,
0.394666, 0.37645]
    image = image*255.0
    plt.imshow(image.astype(int))
    plt.show()

def predict(model,img,path = './', video_file_name=""):
  fmap,logits = model(img.to('cuda'))
  img = im_convert(img[:,-1,:,:,:], video_file_name)
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
```

```
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  confidence = logits[:,int(prediction.item())].item()*100
  print('confidence of
prediction:',logits[:,int(prediction.item())].item()*100)
  return [int(prediction.item()),confidence]

def plot_heat_map(i, model, img, path = './',
video_file_name=''):
  fmap,logits = model(img.to('cuda'))
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  idx = np.argmax(logits.detach().cpu().numpy())
  bz, nc, h, w = fmap.shape
  #out = np.dot(fmap[-1].detach().cpu().numpy().reshape((nc,
h*w)).T,weight_softmax[idx,:].T)
  out = np.dot(fmap[i].detach().cpu().numpy().reshape((nc,
h*w)).T,weight_softmax[idx,:].T)
  predict = out.reshape(h,w)
  predict = predict - np.min(predict)
  predict_img = predict / np.max(predict)
  predict_img = np.uint8(255*predict_img)
  out = cv2.resize(predict_img, (im_size,im_size))
  heatmap = cv2.applyColorMap(out, cv2.COLORMAP_JET)
  img = im_convert(img[:,-1,:,:,:], video_file_name)
  result = heatmap * 0.5 + img*0.8*255
  # Saving heatmap - Start
  heatmap_name = video_file_name+"_heatmap_"+str(i)+".png"
  image_name = os.path.join(settings.PROJECT_DIR,
'uploaded_images', heatmap_name)
  cv2.imwrite(image_name,result)
  # Saving heatmap - End
  result1 = heatmap * 0.5/255 + img*0.8
  r,g,b = cv2.split(result1)
```

```
  result1 = cv2.merge((r,g,b))
  return image_name

# Model Selection
def get_accurate_model(sequence_length):
    model_name = []
    sequence_model = []
    final_model = ""
    list_models = glob.glob(os.path.join(settings.PROJECT_DIR,
"models", "*.pt"))
    for i in list_models:
        model_name.append(i.split("\\")[-1])
    for i in model_name:
        try:
            seq = i.split("_")[3]
            if (int(seq) == sequence_length):
                sequence_model.append(i)
        except:
            pass

    if len(sequence_model) > 1:
        accuracy = []
        for i in sequence_model:
            acc = i.split("_")[1]
            accuracy.append(acc)
        max_index = accuracy.index(max(accuracy))
        final_model = sequence_model[max_index]
    else:
        final_model = sequence_model[0]
    return final_model

ALLOWED_VIDEO_EXTENSIONS =
set(['mp4','gif','webm','avi','3gp','wmv','flv','mkv'])

def allowed_video_file(filename):
    #print("filename" ,filename.rsplit('.',1)[1].lower())
```

```
    if (filename.rsplit('.',1)[1].lower() in
ALLOWED_VIDEO_EXTENSIONS):
        return True
    else:
        return False
def index(request):
    if request.method == 'GET':
        video_upload_form = VideoUploadForm()
        if 'file_name' in request.session:
            del request.session['file_name']
        if 'preprocessed_images' in request.session:
            del request.session['preprocessed_images']
        if 'faces_cropped_images' in request.session:
            del request.session['faces_cropped_images']
        return render(request, index_template_name, {"form":
video_upload_form})
    else:
        video_upload_form = VideoUploadForm(request.POST,
request.FILES)
        if video_upload_form.is_valid():
            video_file =
video_upload_form.cleaned_data['upload_video_file']
            video_file_ext = video_file.name.split('.')[-1]
            sequence_length =
video_upload_form.cleaned_data['sequence_length']
            video_content_type =
video_file.content_type.split('/')[0]
            if video_content_type in settings.CONTENT_TYPES:
                if video_file.size >
int(settings.MAX_UPLOAD_SIZE):
                    video_upload_form.add_error("upload_video_fi
le", "Maximum file size 100 MB")
                    return render(request, index_template_name,
{"form": video_upload_form})

            if sequence_length <= 0:
```

```
                video_upload_form.add_error("sequence_length",
"Sequence Length must be greater than 0")
                return render(request, index_template_name,
{"form": video_upload_form})

            if allowed_video_file(video_file.name) == False:
                video_upload_form.add_error("upload_video_file",
"Only video files are allowed ")
                return render(request, index_template_name,
{"form": video_upload_form})

            saved_video_file =
'uploaded_file_'+str(int(time.time()))+"."+video_file_ext
            with open(os.path.join(settings.PROJECT_DIR,
'uploaded_videos', saved_video_file), 'wb') as vFile:
                shutil.copyfileobj(video_file, vFile)

            request.session['file_name'] =
os.path.join(settings.PROJECT_DIR, 'uploaded_videos',
saved_video_file)
            request.session['sequence_length'] = sequence_length
            return redirect('ml_app:predict')
        else:
            return render(request, index_template_name, {"form":
video_upload_form})

def predict_page(request):
    if request.method == "GET":
        if 'file_name' not in request.session:
            return redirect("ml_app:home")
        if 'file_name' in request.session:
            video_file = request.session['file_name']
        if 'sequence_length' in request.session:
            sequence_length = request.session['sequence_length']
        path_to_videos = [video_file]
        video_file_name = video_file.split('\\')[-1]
```

```
        video_file_name_only = video_file_name.split('.')[0]
        video_dataset = validation_dataset(path_to_videos,
sequence_length=sequence_length,transform= train_transforms)
        model = Model(2).cuda()
        model_name = os.path.join(settings.PROJECT_DIR,'models',
get_accurate_model(sequence_length))
        models_location =
os.path.join(settings.PROJECT_DIR,'models')
        path_to_model = os.path.join(settings.PROJECT_DIR,
model_name)
        model.load_state_dict(torch.load(path_to_model))
        model.eval()
        start_time = time.time()
        # Start: Displaying preprocessing images
        print("<=== | Started Videos Splitting | ===>")
        preprocessed_images = []
        faces_cropped_images = []
        cap = cv2.VideoCapture(video_file)

        frames = []
        while(cap.isOpened()):
            ret, frame = cap.read()
            if ret==True:
                frames.append(frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()

        for i in range(1, sequence_length+1):
            frame = frames[i]
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = pImage.fromarray(image, 'RGB')
            image_name =
video_file_name_only+"_preprocessed_"+str(i)+'.png'
```

```
            image_path = os.path.join(settings.PROJECT_DIR,
'uploaded_images', image_name)
            img.save(image_path)
            preprocessed_images.append(image_name)
        print("<=== | Videos Splitting Done | ===>")
        print("--- %s seconds ---" % (time.time() - start_time))
        # End: Displaying preprocessing images

        # Start: Displaying Faces Cropped Images
        print("<=== | Started Face Cropping Each Frame | ===>")
        padding = 40
        faces_found = 0
        for i in range(1, sequence_length+1):
            frame = frames[i]
            #fig, ax = plt.subplots(1,1, figsize=(5, 5))
            face_locations =
face_recognition.face_locations(frame)
            if len(face_locations) == 0:
                continue
            top, right, bottom, left = face_locations[0]
            frame_face = frame[top-padding:bottom+padding, left-
padding:right+padding]
            image = cv2.cvtColor(frame_face, cv2.COLOR_BGR2RGB)

            img = pImage.fromarray(image, 'RGB')
            image_name =
video_file_name_only+"_cropped_faces_"+str(i)+'.png'
            image_path = os.path.join(settings.PROJECT_DIR,
'uploaded_images',
video_file_name_only+"_cropped_faces_"+str(i)+'.png')
            img.save(image_path)
            faces_found = faces_found + 1
            faces_cropped_images.append(image_name)
        print("<=== | Face Cropping Each Frame Done | ===>")
        print("--- %s seconds ---" % (time.time() - start_time))
```

```
        # No face is detected
        if faces_found == 0:
            return render(request, predict_template_name,
{"no_faces": True})

        # End: Displaying Faces Cropped Images
        try:
            heatmap_images = []
            for i in range(0, len(path_to_videos)):
                output = ""
                print("<=== | Started Predicition | ===>")
                prediction = predict(model, video_dataset[i],
'./', video_file_name_only)
                confidence = round(prediction[1], 1)
                print("<=== |  Predicition Done | ===>")
                # print("<=== | Heat map creation started |
===>")
                # for j in range(0, sequence_length):
                #     heatmap_images.append(plot_heat_map(j,
model, video_dataset[i], './', video_file_name_only))
                if prediction[0] == 1:
                    output = "REAL"
                else:
                    output = "FAKE"
                print("Prediction : " ,
prediction[0],"==",output ,"Confidence : " , confidence)
                print("--- %s seconds ---" % (time.time() -
start_time))

            return render(request, predict_template_name,
{'preprocessed_images': preprocessed_images, 'heatmap_images':
heatmap_images, "faces_cropped_images": faces_cropped_images,
"original_video": video_file_name, "models_location":
models_location, "output": output, "confidence": confidence})
        except:
            return render(request, 'cuda_full.html')
```

```
def about(request):
    return render(request, about_template_name)

def handler404(request,exception):
    return render(request, '404.html', status=404)
def cuda_full(request):
    return render(request, 'cuda_full.html')
```