```
from cnn_visualization.gradcam import Activation, Conv2D
from cnn_visualization import gradcam
from mxnet.gluon.nn import Dense, Dropout, BatchNorm
import mxnet as mx
import mxnet.ndarray as nd
from mxnet import gluon
from mxnet import autograd
from mxnet.gluon import nn
import numpy as np
import cv2

class mobilenet(mx.gluon.HybridBlock):
        def __init__(self, classes=3, **kwargs):
                super(mobilenet, self).__init__(**kwargs)
                with self.name_scope():
                        self.features = self._make_features()
                layers = [
                        Dropout(0.2),
                        Conv2D(16, kernel_size=(3,3), strides=(2,2), padding=(1,1)),
                        BatchNorm(axis=1, in_channels=16),
                        Dropout(0.2),
                        Activation("relu"),
                        Conv2D(8, kernel_size=(3,3), strides=(2,2), padding=(1,1)),
                        BatchNorm(axis=1, in_channels=8),
                        Dropout(0.2),
                        Activation("relu"),
                        Dense(8),
                        Activation("relu")
                ]
                for layer in layers:
                        self.features.add(layer)
                self.output = Dense(classes)

        def _make_features(self):
                featurizer = mx.gluon.nn.HybridSequential(prefix='')
                layers = [
                        Conv2D(32,          kernel_size=(3,3),          strides=(2,2),          padding=(1,1),
bias_initializer="zeros"),
                        BatchNorm(axis=1, in_channels=32),
                        Activation("relu"),
                ]
                for layer in layers:
                        featurizer.add(layer)
                return featurizer
```

```
    def hybrid_forward(self, F, x):
        x = self.features(x)
        x = self.output(x)
        return x
files = "2019-06-21T11-33-35"
rootpath = r".\vibration_model"
subpath1 = "models/%s/net"%(files)
parapath = os.path.join(rootpath, subpath1+"-0000.params")
loadpath = os.path.join(rootpath,subpath1)
cam_net = mobilenet()
cam_net.initialize()
syms_, arg_params, aux_params = mx.model.load_checkpoint(loadpath,0)
params = arg_params.copy()
params.update(aux_params)
for key in params.keys():
    param = params[key]
    cam_net.collect_params()[key].set_data(param)
last_conv_layer_name = cam_net.features[8]._name
print(last_conv_layer_name)

def _get_grad(net, image, class_id=None, conv_layer_name=None, image_grad=False):
    if image_grad:
        image.attach_grad()
        Conv2D.capture_layer_name = None
        Activation.set_guided_backprop(True)
    else:
        Conv2D.capture_layer_name = conv_layer_name
        Activation.set_guided_backprop(False)
    with autograd.record(train_mode=False):
        out = net(image)
    if class_id == None:
        model_output = out.asnumpy()
        class_id = np.argmax(model_output)
    one_hot_target = mx.nd.one_hot(mx.nd.array([class_id]), 3)
    out.backward(one_hot_target, train_mode=False)
    if image_grad:
        return image.grad[0].asnumpy()
    else:
        # Return the recorded convolution output and gradient
        conv_out = Conv2D.conv_output
        return conv_out[0].asnumpy(), conv_out.grad[0].asnumpy()

def get_conv_out_grad(net, image, class_id=None, conv_layer_name=None):
    return _get_grad(net, image, class_id, conv_layer_name, image_grad=False)
def get_image_grad(net, image, class_id=None):
    return _get_grad(net, image, class_id, image_grad=True)
def grad_to_image(gradient):
    gradient = gradient - gradient.min()
    gradient /= gradient.max()
    gradient = np.uint8(gradient * 255).transpose(1, 2, 0)
    gradient = gradient[..., ::-1]
    return gradient
def get_cam(imggrad, conv_out):
    weights = np.mean(imggrad, axis=(1, 2))
    cam = np.ones(conv_out.shape[1:], dtype=np.float32)
    for i, w in enumerate(weights):
        cam += w * conv_out[i, :, :]
    cam = cv2.resize(cam, (imggrad.shape[1], imggrad.shape[2]))
    cam = np.maximum(cam, 0)
    cam = (cam - np.min(cam)) / (np.max(cam) - np.min(cam))
    cam = np.uint8(cam * 255)
    return cam

def get_guided_grad_cam(cam, imggrad):
    return np.multiply(cam, imggrad)

def get_img_heatmap(orig_img, activation_map):
    heatmap = cv2.applyColorMap(activation_map, cv2.COLORMAP_JET)
    heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
    img_heatmap = np.float32(heatmap) + np.float32(orig_img)
    img_heatmap = img_heatmap / np.max(img_heatmap)
    img_heatmap *= 255
    return img_heatmap.astype(int)

def to_grayscale(cv2im):
    grayscale_im = np.sum(np.abs(cv2im), axis=0)
    im_max = np.percentile(grayscale_im, 99)
    im_min = np.min(grayscale_im)
    grayscale_im = np.clip((grayscale_im - im_min) / (im_max - im_min), 0, 1)
    grayscale_im = np.expand_dims(grayscale_im, axis=0)
    return grayscale_im

def visualize(net, preprocessed_img, orig_img, conv_layer_name):
    imggrad = get_image_grad(net, preprocessed_img)
    conv_out,    conv_out_grad    =    get_conv_out_grad(net,    preprocessed_img,
conv_layer_name=conv_layer_name)
```

```
    cam = get_cam(conv_out_grad, conv_out)
    cam = cv2.resize(cam, (imggrad.shape[1], imggrad.shape[2]))
    ggcam = get_guided_grad_cam(cam, imggrad)
    img_ggcam = grad_to_image(ggcam)
    img_heatmap = get_img_heatmap(orig_img, cam)
    ggcam_gray = to_grayscale(ggcam)
    img_ggcam_gray = np.squeeze(grad_to_image(ggcam_gray))
    return img_heatmap, img_ggcam, img_ggcam_gray

features = np.concatenate((train_fea, val_fea, test_fea))
labels = np.concatenate((train_label, val_label, test_label))
datas, datas_iter = enhanceData(features, labels, batch_size=1)

def run_inference(net, data):
    out = net(data)
    return out.argmax(axis=1).asnumpy()[0].astype(int)
def visualize2(net, arrayData, conv_layer_name):
    label_strs=['H','M','S']
    data = arrayData[0]
    label = arrayData[1]
    true_str = label_strs[label]
    preprocessed_img = data.transpose((2,0,1)).expand_dims(axis=0)
    pred_str = label_strs[run_inference(net, preprocessed_img)]
    orig_img = data.asnumpy()
    vizs = visualize(net, preprocessed_img, orig_img, conv_layer_name)
    return (true_str, pred_str, (orig_img, *vizs))

def show_images(pred_str, images):
    titles = [pred_str, 'Grad-CAM', 'Guided Grad-CAM', 'Saliency Map']
    num_images = len(images)
    fig=plt.figure(figsize=(15,15))
    rows, cols = 1, num_images
    for i in range(num_images):
        fig.add_subplot(rows, cols, i+1)
        plt.xlabel(titles[i])
        img = images[i].astype(np.uint8)
        if i==0:
            ax = plt.imshow(img[:,:,0])
        else:
            ax = plt.imshow(img, cmap='gray' if i==num_images-1 else None)
    plt.show()
```