بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Imports

```
!pip install ipython-autotime
```

```
Requirement already satisfied: ipython-autotime in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: ipython in /usr/local/lib/python3.6/dist-packages (from i
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.6/
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/lib/python
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from trait
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages (from p
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/dist-packages
```

```
# necessary imports
import os
import cv2
import numpy as np
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
from tqdm import tqdm

import matplotlib.pyplot as plt
%matplotlib inline

from google.colab.patches import cv2_imshow

%load_ext autotime
```

```
time: 109 µs (started: 2021-01-05 21:01:52 +00:00)
```

```
img_width = 150
img_width = 150
```

```
time: 1.19 ms (started: 2021-01-05 21:01:52 +00:00)
```

VGG Model

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
```

```
time: 3.05 s (started: 2021-01-05 21:01:52 +00:00)
```

```
# load VGG16 model without classification layers
model = VGG16(include_top=False, input_shape=(150, 150, 3))
```

```
time: 2.67 s (started: 2021-01-05 21:01:55 +00:00)
```

```
# add new classification layers
flat1 = Flatten()(model.layers[-1].output) # flatten last layer
class1 = Dense(1024, activation='relu')(flat1) # add FC layer on previous layer
output = Dense(6, activation='softmax')(class1) # add softmax layer
```

```
time: 21 ms (started: 2021-01-05 21:01:58 +00:00)
```

```
# define the new model
model = Model(inputs=model.inputs, outputs=output)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
```

```
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              8389632
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 6150
=================================================================
Total params: 23,110,470
Trainable params: 23,110,470
Non-trainable params: 0
_________________________________________________________________
time: 11.7 ms (started: 2021-01-05 21:01:58 +00:00)
```

Loading Data

```
# !unzip "/content/drive/MyDrive/CV/Assignment 3/intel-image-classification.zip" -d "/content
```

```
time: 940 µs (started: 2021-01-05 21:01:58 +00:00)
```

```
# !unzip "/content/drive/MyDrive/CV/Assignment 3/Test_data.zip" -d "/content/drive/MyDrive/CV
```

```
time: 574 µs (started: 2021-01-05 21:01:58 +00:00)
```

```
# A function to load data from a given directory
def load_data(data_dir):
  data = []
  labels = []
  class_dirs = os.listdir(data_dir)

  for direc in class_dirs:
    # i=0
    class_dir = os.path.join(data_dir, direc)
    for imagepath in tqdm(list(paths.list_images(class_dir))):
      image = cv2.imread(imagepath)
```

```
        image = cv2.imread(imagepath)
        image = cv2.resize(image, (img_width, img_height))  # incase images not of same size
        data.append(image)
        labels.append(direc)
        # i = i+1
        # if (i==10):
        #   break
    # normalizing and converting to numpy array format
    data = np.array(data, dtype='float')/255.0
    labels = np.array(labels)
    return data, labels
```

time: 12 ms (started: 2021-01-05 21:01:58 +00:00)

```
train_dir = "/content/drive/MyDrive/CV/Assignment 3/seg_train/seg_train/"
test_dir = "/content/drive/MyDrive/CV/Assignment 3/seg_test/seg_test/"
pred_dir = "/content/drive/MyDrive/CV/Assignment 3/pred/seg_pred/seg_pred/"
```

time: 1.16 ms (started: 2021-01-05 21:01:58 +00:00)

Compile the model

```
from keras.optimizers import SGD
sgd = SGD(lr=0.001, decay=1e-7, momentum=.9)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
```

time: 21.8 ms (started: 2021-01-05 21:01:58 +00:00)

```
from keras.preprocessing.image import ImageDataGenerator
```

time: 2.52 ms (started: 2021-01-05 21:01:58 +00:00)

```
img_height = 150
img_width = 150
batch_size = 16
nb_epochs = 100
```

time: 3.57 ms (started: 2021-01-05 21:01:58 +00:00)

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2, # zoom
    rotation_range=10, # rotation
    width_shift_range=0.2, # horizontal shift
    height_shift_range=0.2, # vertical shift
    horizontal_flip=True) # horizontal flip
```

```
    horizontal_flip=True) # horizontal flip
    # channel_shift_range = [-0.1, 0.1]
    # )
    # ,validation_split=0.3) # set validation split
```

```
 time: 2.83 ms (started: 2021-01-05 21:05:44 +00:00)
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=True,
    class_mode='categorical',
    interpolation="nearest")
    # subset='training') # set as training data
```

```
 Found 14033 images belonging to 6 classes.
 time: 336 ms (started: 2021-01-05 21:05:47 +00:00)
```

```
val_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
 time: 805 µs (started: 2021-01-05 21:02:00 +00:00)
```

```
validation_generator = val_datagen.flow_from_directory(
    test_dir, # directory for validation data
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
    # subset='validation') # set as validation data
```

```
 Found 3000 images belonging to 6 classes.
 time: 421 ms (started: 2021-01-05 21:02:00 +00:00)
```

```
# validation_generator[0]
```

```
 time: 703 µs (started: 2021-01-05 21:02:00 +00:00)
```

```
H = model.fit(
    train_generator,
    steps_per_epoch = train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs = nb_epochs)
```

```
 Epoch 73/100
 877/877 [==============================] - 130s 148ms/step - loss: 0.0127 - accuracy:
 Epoch 74/100
 877/877 [==============================] - 130s 148ms/step - loss: 0.0113 - accuracy:
 Epoch 75/100
 877/877 [==============================] - 129s 148ms/step - loss: 0.0119 - accuracy:
```

```
Epoch 76/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0179 - accuracy:
Epoch 77/100
877/877 [==============================] - 130s 148ms/step - loss: 0.0127 - accuracy:
Epoch 78/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0187 - accuracy:
Epoch 79/100
877/877 [==============================] - 130s 148ms/step - loss: 0.0116 - accuracy:
Epoch 80/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0163 - accuracy:
Epoch 81/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0172 - accuracy:
Epoch 82/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0263 - accuracy:
Epoch 83/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0156 - accuracy:
Epoch 84/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0099 - accuracy:
Epoch 85/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0181 - accuracy:
Epoch 86/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0175 - accuracy:
Epoch 87/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0110 - accuracy:
Epoch 88/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0095 - accuracy:
Epoch 89/100

877/877 [==============================] - 129s 148ms/step - loss: 0.0086 - accuracy:
Epoch 90/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0125 - accuracy:
Epoch 91/100
877/877 [==============================] - 130s 148ms/step - loss: 0.0141 - accuracy:
Epoch 92/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0123 - accuracy:
Epoch 93/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0071 - accuracy:
Epoch 94/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0084 - accuracy:
Epoch 95/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0114 - accuracy:
Epoch 96/100
877/877 [==============================] - 130s 148ms/step - loss: 0.0127 - accuracy:
Epoch 97/100
877/877 [==============================] - 129s 148ms/step - loss: 0.0131 - accuracy:
Epoch 98/100
877/877 [==============================] - 130s 148ms/step - loss: 0.0074 - accuracy:
Epoch 99/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0062 - accuracy:
Epoch 100/100
877/877 [==============================] - 129s 147ms/step - loss: 0.0225 - accuracy:
time: 4h 8min 12s (started: 2021-01-05 21:06:29 +00:00)
```

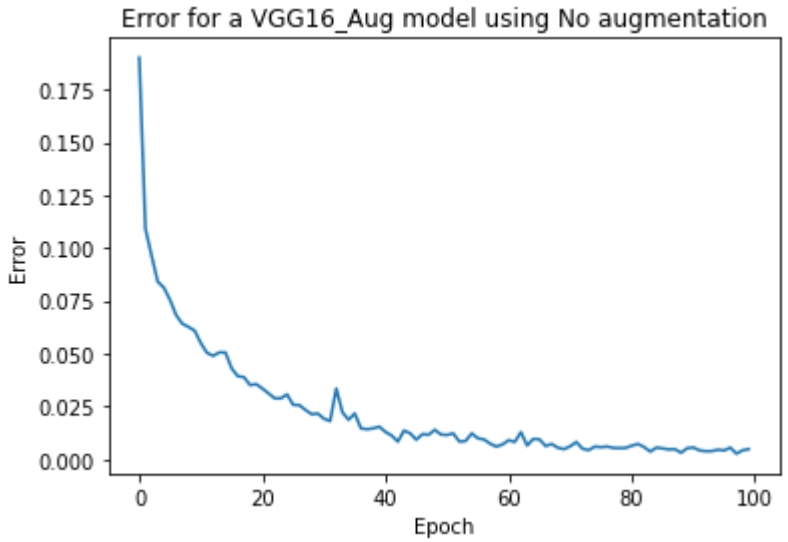
```
# save the model's trained weights
model.save_weights('/content/drive/MyDrive/CV/Assignment 3/vgg_aug_transfer_trained_wts.h5')
```

```
time: 3.02 s (started: 2021-01-06 01:14:45 +00:00)
```

```
# model.load_weights('/content/drive/MyDrive/CV/Assignment 3/vgg_aug_transfer_trained_wts.h5'
```

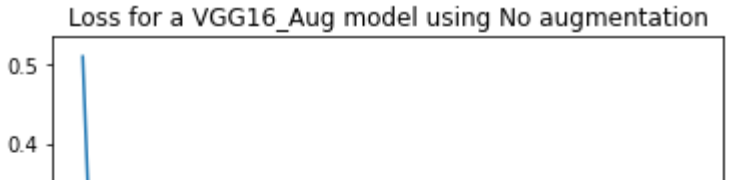
```
time: 1.35 ms (started: 2021-01-06 01:14:48 +00:00)
```

```
simple_acc = H.history['accuracy']
plt.plot([1 - acc for acc in simple_acc])
plt.title('Error for a VGG16_Aug model using No augmentation')
plt.ylabel('Error')
plt.xlabel('Epoch')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/simple_acc_error.png')
plt.show()
```

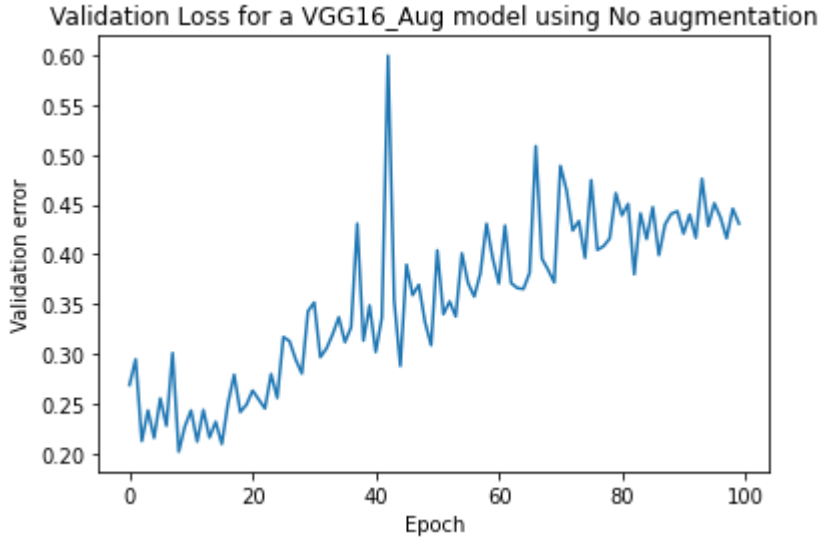



```
time: 208 ms (started: 2021-01-06 01:14:48 +00:00)
```

```
simple_loss = H.history['loss']
plt.plot([los for los in simple_loss])
plt.title('Loss for a VGG16_Aug model using No augmentation')
plt.ylabel('Error')
plt.xlabel('Epoch')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/simple_loss_error.png')
plt.show()
```

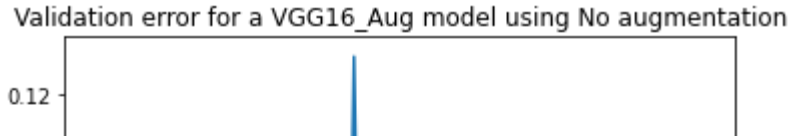



```
simple_val_loss = H.history['val_loss']
plt.plot([los for los in simple_val_loss])
plt.title('Validation Loss for a VGG16_Aug model using No augmentation')
plt.ylabel('Validation error')
plt.xlabel('Epoch')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/simple_Validation_loss_error.pn
plt.show()
```




```
time: 197 ms (started: 2021-01-06 01:14:48 +00:00)
```

```
simple_val_acc = H.history['val_accuracy']
plt.plot([1 - acc for acc in simple_val_acc])
plt.title('Validation error for a VGG16_Aug model using No augmentation')
plt.ylabel('Validation error')
plt.xlabel('Epoch')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/simple_Validation_error.png')
plt.show()
```




```
# test_datagen = ImageDataGenerator()
```

```
time: 510 µs (started: 2021-01-06 01:14:48 +00:00)
```

```
# test_generator = test_datagen.flow_from_directory(
#     pred_dir, # directory for prediction data
#     target_size=(img_height, img_width),
#     batch_size=batch_size,
#     class_mode='categorical',
#     subset='Prediction') # set as validation data
```

```
print('loading pred images')
X_test, y_test = load_data(pred_dir)
```

```
  2%|▏          | 33/1330 [00:00<00:04, 322.29it/s]loading pred images
100%|██████████| 1330/1330 [00:04<00:00, 323.82it/s]
100%|██████████| 1297/1297 [00:03<00:00, 361.66it/s]
100%|██████████| 1128/1128 [00:03<00:00, 324.01it/s]
100%|██████████| 1166/1166 [00:03<00:00, 336.23it/s]
100%|██████████| 1236/1236 [00:03<00:00, 345.48it/s]
100%|██████████| 1144/1144 [00:03<00:00, 339.00it/s]
time: 23.2 s (started: 2021-01-06 01:14:48 +00:00)
```

```
lb = LabelBinarizer()
y_test = lb.fit_transform(y_test)
```

```
time: 13.7 ms (started: 2021-01-06 01:15:12 +00:00)
```

```
score = model.evaluate(X_test, y_test, batch_size=64)
print('Test Loss = ', score[0])
print('Test Accuracy = ', score[1])
```

```
115/115 [==============================] - 16s 126ms/step - loss: 1.0472 - accuracy: 0.8
Test Loss =  1.0471714735031128
Test Accuracy =  0.8371455669403076
time: 17.2 s (started: 2021-01-06 01:15:12 +00:00)
```

```
'''CONFUSION MATRIX'''
# Making prediction
y_pred = model.predict(X_test)
y_true = np.argmax(y_test, axis=-1)

# Plotting the confusion matrix
from sklearn.metrics import confusion_matrix
```

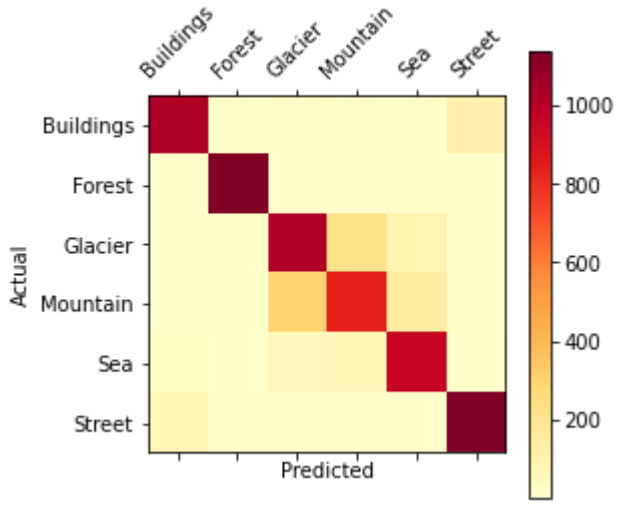
```
confusion_mtx = confusion_matrix(y_true, np.argmax(y_pred, axis=1))
```

```
time: 18.5 s (started: 2021-01-06 01:15:29 +00:00)
```

```
confusion_mtx
```

```
array([[1022,    4,    8,    2,    3,  105],
       [   5, 1140,    8,    3,    8,    2],
       [   3,    2, 1017,  216,   91,    1],
       [   6,    4,  305,  834,  147,    1],
       [  21,    5,   59,   69,  962,   12],
       [  68,   11,    8,    6,    6, 1137]])time: 5.31 ms (started: 2021-01-06 01:15:47
```

```
def plot_confusion_matrix(df_confusion, title='Confusion matrix', cmap=plt.cm.YlOrRd):
  plt.matshow(df_confusion, cmap=cmap) # imshow
  plt.colorbar()
  tick_marks = np.arange(6)
  names = ["Buildings", "Forest", "Glacier", "Mountain", "Sea", "Street"]
  plt.xticks(tick_marks, names, rotation=45)
  plt.yticks(tick_marks, names)
  plt.ylabel("Actual")
  plt.xlabel("Predicted")
#call function
plot_confusion_matrix(confusion_mtx)
```

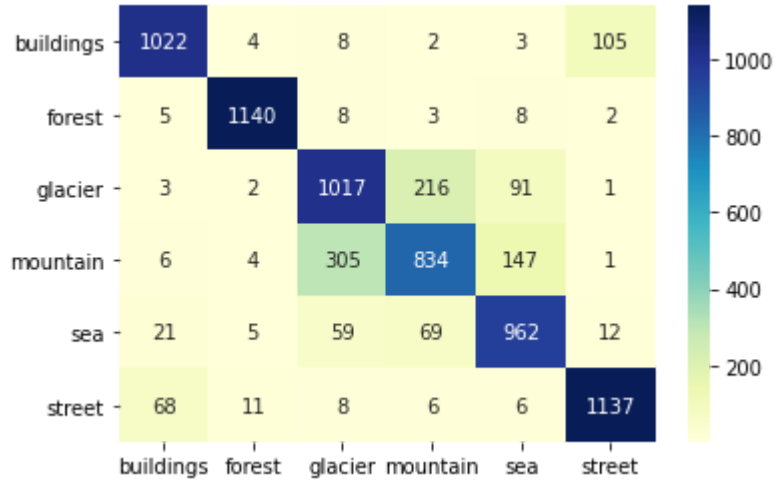



```
time: 166 ms (started: 2021-01-06 01:15:47 +00:00)
```

```
import seaborn as sns

class_names = ['buildings','street','forest','glacier','mountain','sea']

class_names = sorted(class_names)

sns.heatmap(confusion_mtx, xticklabels=class_names, yticklabels=class_names,
            annot=True, fmt='d', cmap="YlGnBu")
```

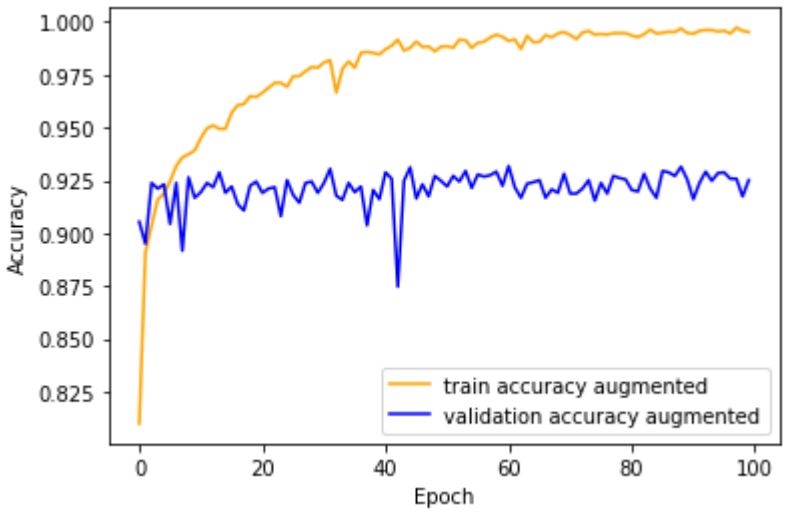
```
annot=True, fmt='d', cmap='Reds')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f37a223c5f8>
```




```
time: 348 ms (started: 2021-01-06 01:15:48 +00:00)
```

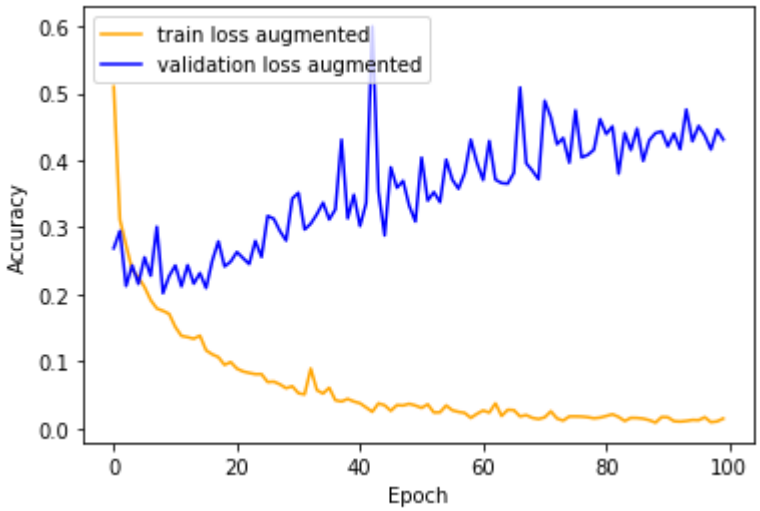
```
fig = plt.figure()
# fig.patch.set_facecolor('white')

plt.plot(H.history['accuracy'],
         label='train accuracy augmented',
         c='orange', ls='-')
plt.plot(H.history['val_accuracy'],
         label='validation accuracy augmented',
         c='blue',ls='-')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/Aug_Validation_error.png')
plt.show()
```




```
time: 207 ms (started: 2021-01-06 01:30:02 +00:00)
```

```
fig = plt.figure()
```

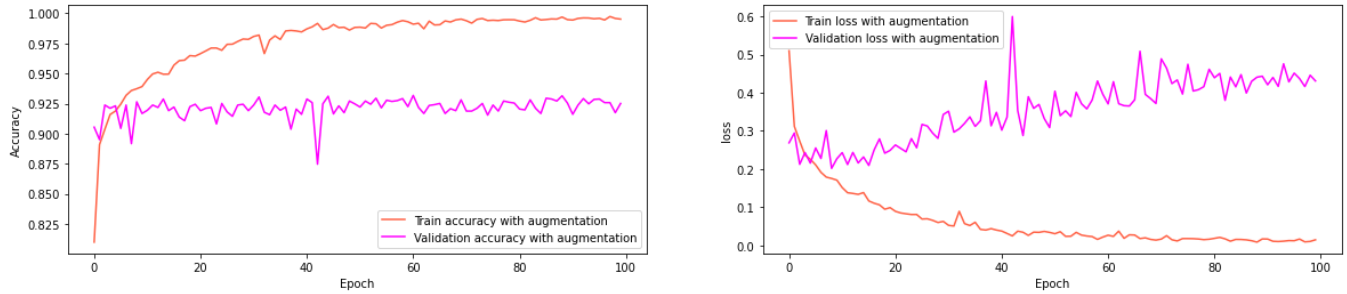
```
plt.plot(H.history['loss'],
         label='train loss augmented',
         c='orange', ls='-')
plt.plot(H.history['val_loss'],
         label='validation loss augmented',
         c='blue',ls='-')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/Aug_Validation_&loss.png')
plt.show()
```




```
time: 223 ms (started: 2021-01-06 01:29:23 +00:00)
```

```
fig, axis = plt.subplots(1, 2, figsize=(20, 4))


axis[0].plot(H.history['accuracy'],
         label='Train accuracy with augmentation',
         c='tomato', ls='-')
axis[0].plot(H.history['val_accuracy'],
         label='Validation accuracy with augmentation',
         c='magenta', ls='-')

axis[0].set_xlabel('Epoch')
axis[0].set_ylabel('Accuracy')
axis[0].legend(loc='lower right')


axis[1].plot(H.history['loss'],
         label='Train loss with augmentation',
         c='tomato', ls='-')
axis[1].plot(H.history['val_loss'],
         label='Validation loss with augmentation',
         c='magenta', ls='-')
```

```
axis[1].set_xlabel('Epoch')
axis[1].set_ylabel('loss')
axis[1].legend(loc='upper left')
plt.savefig('/content/drive/MyDrive/CV/Assignment 3/VGG16_Aug/Aug_Validation_error&loss.png')
plt.show()
```




```
time: 452 ms (started: 2021-01-06 01:28:34 +00:00)
```