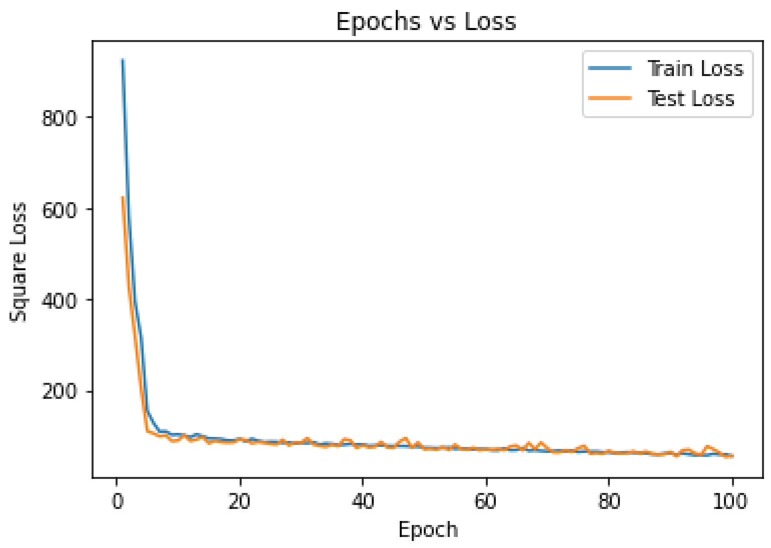
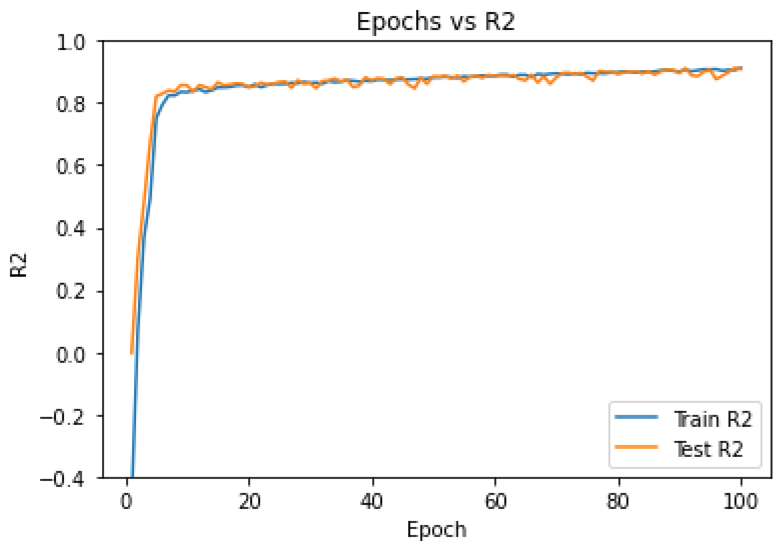
Abelardo Riojas
STA 5635 Spring 2022
Prof. Barbu

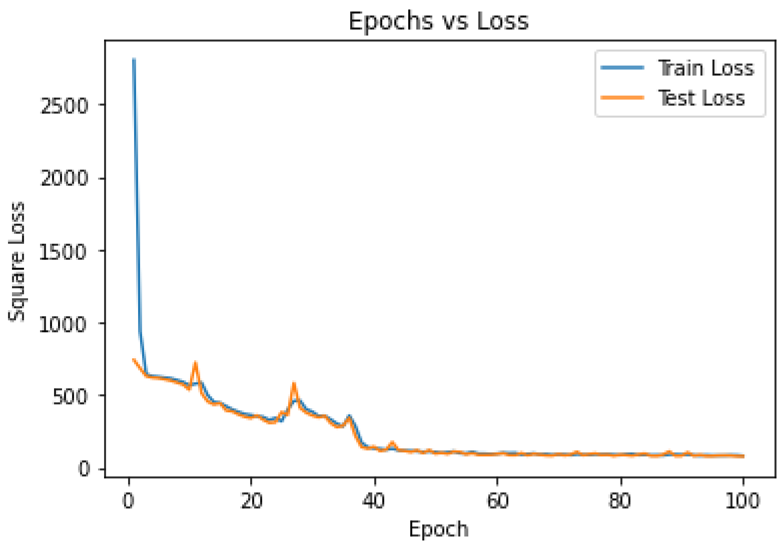
Homework 8 Report

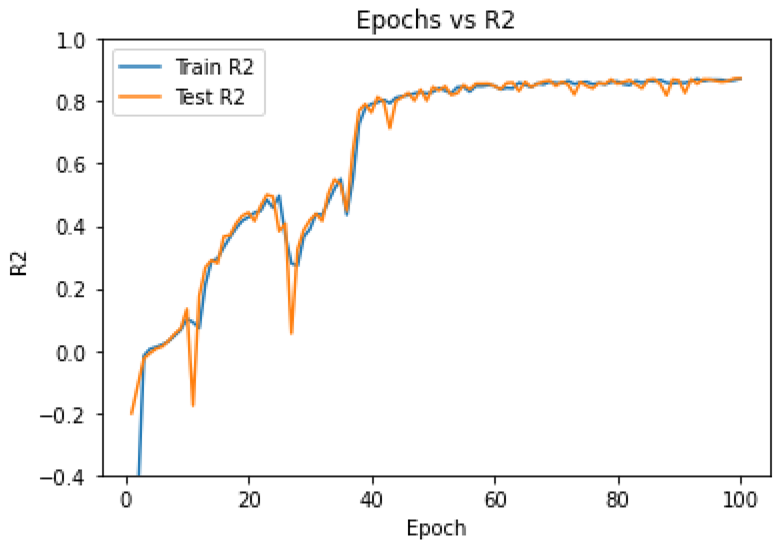
1. Train a CNN for 100 epochs with momentum 0.9 using the square loss (1). Use the SGD optimizer with an appropriate learning rate and λ = 0.0001 (weight decay). Start with minibatch size 64 and double it every 20 epochs and try to obtain a good training R 2 (at least 0.9). You might also need to reduce the learning rate as training progresses. Show a plot of the loss function vs epoch number for the training set and the test set. Show another plot of the training and test R 2 vs epoch number.





2. Repeat point a) with the same CNN architecture, but with a fixed minibatch of 1024 and a fixed learning rate. It’s ok if you cannot get a training R 2 of at least 0.9 in this case.

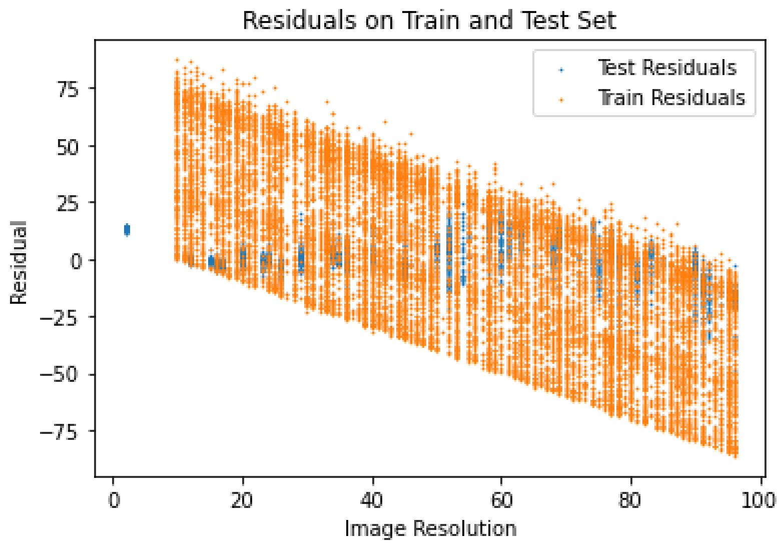




3. Report the CNN architecture in a table, where each row describes one layer, including the layer description, size and number of parameters, and the last row containing the total number of parameters.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 64, 64, 3)]       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 62, 16)        448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 31, 31, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 29, 29, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 11,425
Trainable params: 11,425
Non-trainable params: 0
_________________________________________________________________
```

4. Plot with two different colors the train and test residuals for the model obtained in part 1.



Code below:

```
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
```

```
import tensorflow as tf
from keras import backend as K

info=[]
for filename in os.listdir('train'):
    info.append([os.path.join('train', filename), int(filename[:2])])

train_df = pd.DataFrame(info, columns=['filepath','resolution'])

info=[]
for filename in os.listdir('test'):
    info.append([os.path.join('test', filename), int(filename[:2])])

test_df = pd.DataFrame(info, columns=['filepath','resolution'])

#loading in images
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)
#loading in images
test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
)

train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col ='filepath',
    y_col='resolution',
    color_mode='rgb',
    class_mode='raw',
    batch_size=64,
    seed=42,
    shuffle=True,
    subset='training'
)
val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col ='filepath',
    y_col='resolution',
    color_mode='rgb',
    class_mode='raw',
    batch_size=64,
    seed =42,
    shuffle=True,
    subset='validation'
)
test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col ='filepath',
    y_col='resolution',
    color_mode='rgb',
    class_mode='raw',
    batch_size=32,
    shuffle=False
)

def coeff_determination(y_true, y_pred):
    SS_res =  K.sum(K.square( y_true-y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )

inputs = tf.keras.Input(shape=(64,64,3))
x = tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3),
activation='relu')(inputs)
x = tf.keras.layers.MaxPool2D()(x)
x = tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3),
activation='relu')(x)
x = tf.keras.layers.MaxPool2D()(x)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='linear')(x)


model = tf.keras.Model(inputs=inputs, outputs=outputs)

alpha = 0.0001  # weight decay coefficient

for layer in model.layers:
    if isinstance(layer, tf.keras.layers.Conv2D) or isinstance(layer,
tf.keras.layers.Dense):
        layer.add_loss(lambda layer=layer:
tf.keras.regularizers.l2(alpha)(layer.kernel))
    if hasattr(layer, 'bias_regularizer') and layer.use_bias:
        layer.add_loss(lambda layer=layer:
tf.keras.regularizers.l2(alpha)(layer.bias))

model.compile(
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.0001, momentum=0.9,
name='SGD'),
    loss = 'mse',
    metrics = [coeff_determination]
)

model.summary()

#histories
history1 = model.fit(
    train_images,
    validation_data=val_images,
    epochs=20,
    verbose=1
)
train_generator.batch_size= 128
history2 = model.fit(
    train_images,
    validation_data=val_images,
    epochs=20,
    verbose=1
)
train_generator.batch_size= 256
history3 = model.fit(
    train_images,
    validation_data=val_images,
    epochs=20,
    verbose=1
)
train_generator.batch_size= 512
history4 = model.fit(
    train_images,
    validation_data=val_images,
    epochs=20,
    verbose=1
)
train_generator.batch_size= 1024
history5 = model.fit(
    train_images,
    validation_data=val_images,
    epochs=20,
    verbose=1
)

histories = [history1, history2, history3, history4, history5]
loss_train = []
loss_test = []
r2_train = []
r2_test = []

for history in histories:
    history = history.history
    loss_train.append(history['loss'])
    loss_test.append(history['val_loss'])
    r2_train.append(history['coeff_determination'])
    r2_test.append(history['val_coeff_determination'])
loss_train = np.array(loss_train).flatten()
loss_test = np.array(loss_test).flatten()
r2_train = np.array(r2_train).flatten()
r2_test = np.array(r2_test).flatten()

import matplotlib.pyplot as plt

plt.plot(list(range(1,101)), loss_train)
plt.plot(list(range(1,101)), loss_test)
plt.xlabel('Epoch')
plt.ylabel('Square Loss')
plt.title('Epochs vs Loss')
plt.legend(['Train Loss', 'Test Loss'])
plt.show()

plt.plot(list(range(1,101)), r2_train)
plt.plot(list(range(1,101)), r2_test)
plt.xlabel('Epoch')
plt.ylabel('R2')
plt.title('Epochs vs R2')
plt.ylim([-.4, 1])
plt.legend(['Train R2', 'Test R2'])
plt.show()

preds = np.squeeze(model.predict(test_images))
preds_train = np.squeeze(model.predict(train_images))
```

```
y_true = test_images.labels
y_true_train = train_image.labels

residuals_test = preds - y_true
residuals_train = preds_train - y_true_train

plt.scatter(y_true, residuals_test, s=.5)
plt.scatter(y_true_train, residuals_train, s=.5)
plt.xlabel('Image Resolution')
plt.ylabel('Residual')
plt.title('Residuals on Train and Test Set')
plt.legend(['Test Residuals', 'Train Residuals'])
her song pianoher song piano
```