In [1]:
```
# Import GeneralPackages
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
import warnings
import random
from scipy.stats import reciprocal
from PIL import Image
import cv2
import pandas as pd
import joblib

# Import Sklearn Packages
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split as split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
```

In [2]:
```
# Import Tensorflow packages
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.wrappers.scikit_learn import KerasClassifier
```

Problem 1 - Flower Species Classification

In [3]:
```
# Define Class Names
class_names = ['Roses', 'Magnolias', 'Lilies', 'Sunflowers', 'Orchids',
               'Marigold', 'Hibiscus', 'Firebush', 'Pentas', 'Bougainvillea']
```

In [4]:
```
# Loading Training Data
X_train = np.load('flower_species_classification/data_train.npy').T
t_train = np.load('flower_species_classification/labels_train.npy')

# Load Test Data
X_test = np.load('flower_species_classification/data_test.npy').T
t_test = np.load('flower_species_classification/labels_test.npy')

print(X_train.shape, t_train.shape)
```

```
(1658, 270000) (1658,)
```

In [5]:
```
# Perform Stratified Test Train Split
X_training, X_val, t_training, t_val = train_test_split(X_train,
                                                        t_train,
                                                        shuffle=True,
                                                        stratify=t_train,
                                                        test_size=0.20,
                                                      random_state = 42)
print(f"X_training Shape: {X_training.shape}")
print(f"X_val Shape: {X_val.shape}")
```

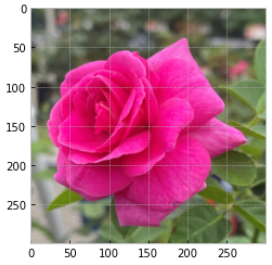
```
X_training Shape: (1326, 270000)
X_val Shape: (332, 270000)
```

In [6]:
```
# Scale Data
scaler = MinMaxScaler()
X_training = scaler.fit_transform(X_training)
X_val = scaler.fit_transform(X_val)

# Reshape Image array to accomodate Neural Network
X_train_rs = X_training.reshape([1326,300,300,3])
X_val_rs = X_val.reshape([332,300,300,3])
print(X_train_rs.shape)
print(X_val_rs.shape)
```

```
(1326, 300, 300, 3)
(332, 300, 300, 3)
```

In [7]:
```
plt.imshow(X_train_rs[0])
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x2b54d45c2ca0>
```



Xception Model

In [7]:
```
# Import Xception CNN Model
base_model = keras.applications.Xception(weights='imagenet', # Load weights pre-trained on ImageNet.
                                         input_shape=(150, 150, 3),
                                         include_top=False)
```

```
2022-12-09 18:07:18.295757: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use
the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-09 18:07:18.824130: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 79111 MB memory:  -> de
vice: 0, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:b7:00.0, compute capability: 8.0
```

Base Model

In [8]:
```
# Define Base Model: Xception Weights are not Trainable
base_model.trainable = False

IMG_SIZE = 150 # Xception Image Size
inputs = keras.Input(shape=(300, 300, 3)) # Input Image Size from Flower Data
inputs_resized = keras.layers.Resizing(IMG_SIZE, IMG_SIZE)(inputs) # 150-150-3 # Input Layer
x = base_model(inputs_resized, training=False) # resizing input to match pretrained model
x_pooling = keras.layers.GlobalAveragePooling2D()(x)
outputs = keras.layers.Dense(10, activation='softmax')(x_pooling) # Output Layer
model_Base = tf.keras.Model(inputs, outputs)

# Summary of Model Layers and Parameters
model_Base.summary()

warnings.filterwarnings('ignore')
```

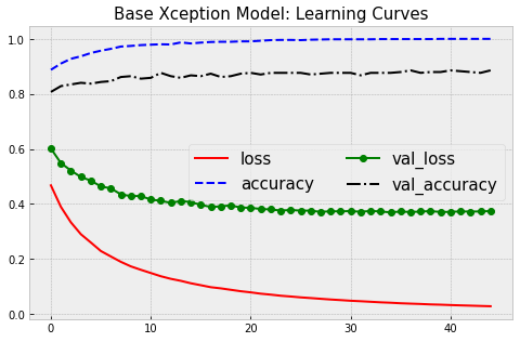
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 300, 300, 3)]     0

 resizing (Resizing)         (None, 150, 150, 3)       0

 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 20,881,970
Trainable params: 20,490
Non-trainable params: 20,861,480
_________________________________________________________________
```

In [10]:
```
# Compile Base Model
model_Base.compile(optimizer=keras.optimizers.Nadam(),
              loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# Fit Base Model to Training Data
model_Base.fit(X_train_rs,t_training, epochs=50, batch_size=32,
          validation_data=(X_val_rs, t_val),
          callbacks=[keras.callbacks.EarlyStopping(patience=10)])

warnings.filterwarnings("ignore")
```

```
Epoch 1/50
42/42 [==============================] - 3s 36ms/step - loss: 0.4676 - accuracy: 0.8876 - val_loss: 0.6016 - val_accuracy: 0.8072
Epoch 2/50
42/42 [==============================] - 1s 26ms/step - loss: 0.3894 - accuracy: 0.9103 - val_loss: 0.5486 - val_accuracy: 0.8283
Epoch 3/50
42/42 [==============================] - 1s 26ms/step - loss: 0.3333 - accuracy: 0.9276 - val_loss: 0.5216 - val_accuracy: 0.8343
Epoch 4/50
42/42 [==============================] - 1s 27ms/step - loss: 0.2901 - accuracy: 0.9367 - val_loss: 0.4981 - val_accuracy: 0.8404
Epoch 5/50
42/42 [==============================] - 1s 29ms/step - loss: 0.2599 - accuracy: 0.9487 - val_loss: 0.4849 - val_accuracy: 0.8373
Epoch 6/50
42/42 [==============================] - 1s 29ms/step - loss: 0.2290 - accuracy: 0.9570 - val_loss: 0.4637 - val_accuracy: 0.8434
Epoch 7/50
42/42 [==============================] - 1s 27ms/step - loss: 0.2092 - accuracy: 0.9638 - val_loss: 0.4567 - val_accuracy: 0.8464
Epoch 8/50
42/42 [==============================] - 1s 26ms/step - loss: 0.1897 - accuracy: 0.9721 - val_loss: 0.4351 - val_accuracy: 0.8614
Epoch 9/50
42/42 [==============================] - 1s 26ms/step - loss: 0.1732 - accuracy: 0.9744 - val_loss: 0.4285 - val_accuracy: 0.8645
Epoch 10/50
42/42 [==============================] - 1s 27ms/step - loss: 0.1608 - accuracy: 0.9774 - val_loss: 0.4292 - val_accuracy: 0.8554
Epoch 11/50
42/42 [==============================] - 1s 27ms/step - loss: 0.1491 - accuracy: 0.9789 - val_loss: 0.4158 - val_accuracy: 0.8584
Epoch 12/50
42/42 [==============================] - 1s 27ms/step - loss: 0.1374 - accuracy: 0.9804 - val_loss: 0.4117 - val_accuracy: 0.8765
Epoch 13/50
42/42 [==============================] - 1s 26ms/step - loss: 0.1280 - accuracy: 0.9796 - val_loss: 0.4039 - val_accuracy: 0.8645
Epoch 14/50
42/42 [==============================] - 1s 26ms/step - loss: 0.1207 - accuracy: 0.9872 - val_loss: 0.4109 - val_accuracy: 0.8584
Epoch 15/50
42/42 [==============================] - 1s 26ms/step - loss: 0.1118 - accuracy: 0.9834 - val_loss: 0.4068 - val_accuracy: 0.8675
Epoch 16/50
42/42 [==============================] - 1s 26ms/step - loss: 0.1048 - accuracy: 0.9864 - val_loss: 0.3967 - val_accuracy: 0.8645
Epoch 17/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0975 - accuracy: 0.9887 - val_loss: 0.3892 - val_accuracy: 0.8735
Epoch 18/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0934 - accuracy: 0.9894 - val_loss: 0.3911 - val_accuracy: 0.8614
Epoch 19/50
42/42 [==============================] - 1s 29ms/step - loss: 0.0881 - accuracy: 0.9894 - val_loss: 0.3949 - val_accuracy: 0.8645
Epoch 20/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0829 - accuracy: 0.9910 - val_loss: 0.3870 - val_accuracy: 0.8735
Epoch 21/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0788 - accuracy: 0.9910 - val_loss: 0.3860 - val_accuracy: 0.8765
Epoch 22/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0740 - accuracy: 0.9932 - val_loss: 0.3817 - val_accuracy: 0.8705
Epoch 23/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0705 - accuracy: 0.9955 - val_loss: 0.3803 - val_accuracy: 0.8765
Epoch 24/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0665 - accuracy: 0.9962 - val_loss: 0.3760 - val_accuracy: 0.8765
Epoch 25/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0637 - accuracy: 0.9962 - val_loss: 0.3782 - val_accuracy: 0.8765
Epoch 26/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0606 - accuracy: 0.9955 - val_loss: 0.3758 - val_accuracy: 0.8765
Epoch 27/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0581 - accuracy: 0.9970 - val_loss: 0.3757 - val_accuracy: 0.8705
Epoch 28/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0554 - accuracy: 0.9977 - val_loss: 0.3712 - val_accuracy: 0.8735
Epoch 29/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0528 - accuracy: 0.9985 - val_loss: 0.3736 - val_accuracy: 0.8765
Epoch 30/50
42/42 [==============================] - 1s 29ms/step - loss: 0.0507 - accuracy: 0.9985 - val_loss: 0.3730 - val_accuracy: 0.8765
Epoch 31/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0483 - accuracy: 0.9985 - val_loss: 0.3744 - val_accuracy: 0.8765
Epoch 32/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0465 - accuracy: 0.9985 - val_loss: 0.3715 - val_accuracy: 0.8675
Epoch 33/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0446 - accuracy: 0.9985 - val_loss: 0.3747 - val_accuracy: 0.8765
Epoch 34/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0426 - accuracy: 0.9992 - val_loss: 0.3728 - val_accuracy: 0.8765
Epoch 35/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0411 - accuracy: 0.9992 - val_loss: 0.3695 - val_accuracy: 0.8765
Epoch 36/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0392 - accuracy: 0.9992 - val_loss: 0.3726 - val_accuracy: 0.8795
Epoch 37/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0378 - accuracy: 0.9992 - val_loss: 0.3707 - val_accuracy: 0.8855
Epoch 38/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0366 - accuracy: 0.9992 - val_loss: 0.3725 - val_accuracy: 0.8765
Epoch 39/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0350 - accuracy: 0.9992 - val_loss: 0.3740 - val_accuracy: 0.8795
Epoch 40/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0340 - accuracy: 1.0000 - val_loss: 0.3709 - val_accuracy: 0.8795
Epoch 41/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0327 - accuracy: 1.0000 - val_loss: 0.3712 - val_accuracy: 0.8855
Epoch 42/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0315 - accuracy: 1.0000 - val_loss: 0.3731 - val_accuracy: 0.8825
Epoch 43/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0305 - accuracy: 1.0000 - val_loss: 0.3722 - val_accuracy: 0.8795
Epoch 44/50
42/42 [==============================] - 1s 27ms/step - loss: 0.0294 - accuracy: 1.0000 - val_loss: 0.3730 - val_accuracy: 0.8765
Epoch 45/50
42/42 [==============================] - 1s 28ms/step - loss: 0.0284 - accuracy: 1.0000 - val_loss: 0.3741 - val_accuracy: 0.8855
```

In [12]:

```python
# Display Learning Curve for Train and Validation Data of the Standard Xception Model
key_names = list(model_Base.history.history.keys())
colors = ['-r','--b','-og','-.k']
plt.figure(figsize=(8,5))
for i in range(len(key_names)):
    plt.plot(model_Base.history.history[key_names[i]], colors[i], label=key_names[i])
    plt.legend(fontsize=15,ncol=2)
    plt.title('Base Xception Model: Learning Curves', size=15);
```



Fine-Tuning Modified Xception Model

In [7]:
```
# Function For Establishing Transfer Learning Model with Tunable Parameters
def create_model(train_layer, LR):
    # Import Xception Model
    base_model = keras.applications.Xception(weights='imagenet', # Load weights pre-trained on ImageNet.
                                             input_shape=(150, 150, 3),
                                             include_top=False)
    # Xception Weights are Trainable
    base_model.trainable = True

    # Let's take a look to see how many layers are in the base model
#     print("Number of layers in the base model: ", len(base_model.layers))

    # Fine-tune from this layer onwards
    fine_tune_at = train_layer

    # Freeze all the layers before the `fine_tune_at` layer
    for layer in base_model.layers[:fine_tune_at]:
        layer.trainable = False

    IMG_SIZE = 150 # Xception Default Image Size
    inputs = keras.Input(shape=(300, 300, 3)) # Input Image Size from Flowers Dataset

    inputs_resized = keras.layers.Resizing(IMG_SIZE, IMG_SIZE)(inputs) # 150-150-3 # Input Layer
    x = base_model(inputs_resized, training=False) # resizing input to match pretrained model
    x_pooling = keras.layers.GlobalAveragePooling2D()(x)
    batch = keras.layers.BatchNormalization()(x_pooling)
    outputs = keras.layers.Dense(10, activation='softmax')(batch) # Output Layer
    model = tf.keras.Model(inputs, outputs)

    # Compile Model
    model.compile(optimizer=keras.optimizers.Adam(learning_rate = LR), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model

# Hyperparameter Values to be tuned
tuning_layer = [80, 100,130]
batch_size = [16,32, 40]
LR = [0.001, 0.0001]

# Performs Random CV Search On hyperparameters
for i in range(8):
    tl = random.choice(tuning_layer)
    bs = random.choice(batch_size)
    lr = random.choice(LR)

    # Establish Model With randomized hyperparameter Values
    keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
                                epochs = 10,
                                batch_size = bs,
                               verbose = 0)


    # Perform Cross Validation Scoring on Model
    score = cross_val_score(keras_clf, X_train_rs, t_training, cv=2, scoring='accuracy')
    print(f"Tuning Layer: {tl} | Batch Size: {bs} | Learning Rate {lr}")
    print(f"CV Score: {score.mean()}\n")
```

```
/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
2022-12-09 16:15:29.805760: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-09 16:15:30.344991: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 79111 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:bd:00.0, compute capability: 8.0
2022-12-09 16:15:35.428136: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8201
2022-12-09 16:15:36.531009: W tensorflow/stream_executor/gpu/asm_compiler.cc:80] Couldn't get ptxas version string: INTERNAL: Running ptxas --version returned 32512
2022-12-09 16:15:36.718477: W tensorflow/stream_executor/gpu/redzone_allocator.cc:314] INTERNAL: ptxas exited with non-zero error code 32512, output:
Relying on driver to perform ptx compilation.
Modify $PATH to customize ptxas location.
This message will be only logged once.
2022-12-09 16:15:37.760971: I tensorflow/stream_executor/cuda/cuda_blas.cc:1774] TensorFloat-32 will be used for the matrix multiplication. This will only be logged once.
Tuning Layer: 100 | Batch Size: 32 | Learning Rate 0.0001
CV Score: 0.8725490196078431

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
Tuning Layer: 130 | Batch Size: 16 | Learning Rate 0.001
CV Score: 0.8220211161387632

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
Tuning Layer: 130 | Batch Size: 40 | Learning Rate 0.0001
CV Score: 0.641025641025641

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
```

```
Tuning Layer: 130 | Batch Size: 40 | Learning Rate 0.001
CV Score: 0.8076923076923077

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
Tuning Layer: 100 | Batch Size: 40 | Learning Rate 0.001
CV Score: 0.603318250377074

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
Tuning Layer: 130 | Batch Size: 32 | Learning Rate 0.0001
CV Score: 0.691553544494721

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
Tuning Layer: 130 | Batch Size: 32 | Learning Rate 0.0001
CV Score: 0.6704374057315233

/scratch/local/53498934/ipykernel_120167/3918662170.py:40: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  keras_clf = KerasClassifier(build_fn = lambda: create_model(train_layer = tl, LR = lr),
Tuning Layer: 80 | Batch Size: 16 | Learning Rate 0.001
CV Score: 0.8438914027149321
```

Final Model

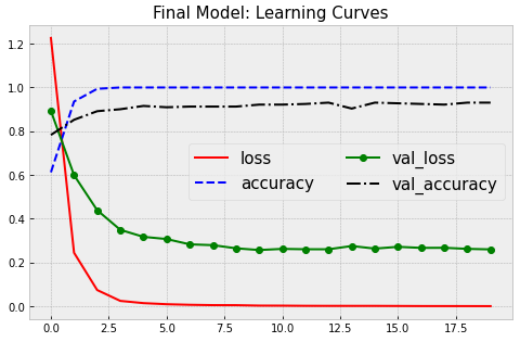
In [13]:
```python
warnings.filterwarnings('ignore')
# Final Model as determined by random CV Search

# Model Weights are Trainable
base_model.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

IMG_SIZE = 150 # Xception Defaulkt Image Size
inputs = keras.Input(shape=(300, 300, 3)) # Input Image Size

inputs_resized = keras.layers.Resizing(IMG_SIZE, IMG_SIZE)(inputs) # 150-150-3 # Input layer
x = base_model(inputs_resized, training=False) # resizing input to match pretrained model
x_pooling = keras.layers.GlobalAveragePooling2D()(x)
batch = keras.layers.BatchNormalization()(x_pooling)
outputs = keras.layers.Dense(10, activation='softmax')(batch) # Output Layer
model_final = tf.keras.Model(inputs, outputs)

model_final.summary()

warnings.filterwarnings("ignore")
```

```
Number of layers in the base model:  132
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 300, 300, 3)]     0

 resizing_1 (Resizing)       (None, 150, 150, 3)       0

 xception (Functional)       (None, 5, 5, 2048)        20861480

 global_average_pooling2d_1   (None, 2048)             0
 (GlobalAveragePooling2D)

 batch_normalization_4 (Batc  (None, 2048)             8192
 hNormalization)

 dense_1 (Dense)             (None, 10)                20490

=================================================================
Total params: 20,890,162
Trainable params: 9,502,930
Non-trainable params: 11,387,232
_________________________________________________________________
```

In [14]:
```python
# Compile Final Model
model_final.compile(optimizer=keras.optimizers.Nadam(learning_rate = 0.0001),
              loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# Fit Final Model
model_final.fit(X_train_rs,t_training, epochs=100, batch_size=32,
          validation_data=(X_val_rs, t_val),
          callbacks=[keras.callbacks.EarlyStopping(patience=10)])

warnings.filterwarnings("ignore")
```

```
Epoch 1/100
42/42 [==============================] - 6s 64ms/step - loss: 1.2261 - accuracy: 0.6124 - val_loss: 0.8932 - val_accuracy: 0.7831
Epoch 2/100
42/42 [==============================] - 2s 40ms/step - loss: 0.2448 - accuracy: 0.9359 - val_loss: 0.5993 - val_accuracy: 0.8524
Epoch 3/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0743 - accuracy: 0.9940 - val_loss: 0.4392 - val_accuracy: 0.8916
Epoch 4/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0247 - accuracy: 1.0000 - val_loss: 0.3495 - val_accuracy: 0.9006
Epoch 5/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0146 - accuracy: 1.0000 - val_loss: 0.3174 - val_accuracy: 0.9157
Epoch 6/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0095 - accuracy: 1.0000 - val_loss: 0.3074 - val_accuracy: 0.9096
Epoch 7/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0069 - accuracy: 1.0000 - val_loss: 0.2831 - val_accuracy: 0.9127
Epoch 8/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0054 - accuracy: 1.0000 - val_loss: 0.2794 - val_accuracy: 0.9127
Epoch 9/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0052 - accuracy: 1.0000 - val_loss: 0.2650 - val_accuracy: 0.9127
Epoch 10/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0032 - accuracy: 1.0000 - val_loss: 0.2572 - val_accuracy: 0.9217
Epoch 11/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.2624 - val_accuracy: 0.9217
Epoch 12/100
42/42 [==============================] - 2s 38ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.2607 - val_accuracy: 0.9247
Epoch 13/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.2606 - val_accuracy: 0.9307
Epoch 14/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.2755 - val_accuracy: 0.9036
Epoch 15/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.2637 - val_accuracy: 0.9307
Epoch 16/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.2718 - val_accuracy: 0.9277
Epoch 17/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.2670 - val_accuracy: 0.9247
Epoch 18/100
42/42 [==============================] - 2s 39ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.2674 - val_accuracy: 0.9217
Epoch 19/100
42/42 [==============================] - 2s 39ms/step - loss: 9.4212e-04 - accuracy: 1.0000 - val_loss: 0.2625 - val_accuracy: 0.9307
Epoch 20/100
42/42 [==============================] - 2s 40ms/step - loss: 7.8824e-04 - accuracy: 1.0000 - val_loss: 0.2601 - val_accuracy: 0.9307
```

In [15]:
```
# Display Learning Curve for Train and Validation Data
key_names = list(model_final.history.history.keys())
colors = ['-r','--b','-og','-.k']
plt.figure(figsize=(8,5))
for i in range(len(key_names)):
    plt.plot(model_final.history.history[key_names[i]], colors[i], label=key_names[i])
    plt.legend(fontsize=15,ncol=2)
    plt.title('Final Model: Learning Curves', size=15);
```



In [11]:
```
# Save Final Model File
model_final.save('flower_model.h5')
```

Problem 2: Car Detection

In [3]:
```
# Displays Learning Curve for Train and Validation Data
def LearningCurve(model):
    key_names = list(model.history.history.keys())
    colors = ['-r','--b','-og','-.k']
    plt.figure(figsize=(8,5))
    for i in range(len(key_names)):
        plt.plot(model.history.history[key_names[i]], colors[i], label=key_names[i])
        plt.legend(fontsize=15,ncol=2)
        plt.title('Final Model: Learning Curves', size=15);
```

In [4]:
```
# Training Data with Cars in them and associated bounding boxes
bbox = pd.read_csv('car_detection_dataset/train_bounding_boxes.csv')
bbox
```

Out[4]:

	image	xmin	ymin	xmax	ymax
0	vid_4_1000.jpg	281.259045	187.035071	327.727931	223.225547
1	vid_4_10000.jpg	15.163531	187.035071	120.329957	236.430180
2	vid_4_10040.jpg	239.192475	176.764801	361.968162	236.430180
3	vid_4_10020.jpg	496.483358	172.363256	630.020260	231.539575
4	vid_4_10060.jpg	16.630970	186.546010	132.558611	238.386422
...	...	...	...	...	...
554	vid_4_9860.jpg	0.000000	198.321729	49.235251	236.223284
555	vid_4_9880.jpg	329.876184	156.482351	536.664239	250.497895
556	vid_4_9900.jpg	0.000000	168.295823	141.797524	239.176652
557	vid_4_9960.jpg	487.428988	172.233646	616.917699	228.839864
558	vid_4_9980.jpg	221.558631	182.570434	348.585579	238.192196

559 rows × 5 columns

Classification Neural Network

In [5]:
```
from PIL import Image
import glob

# Extracts Image Number from training data with Cars
train_car = np.zeros([len(bbox)])
for i in range(len(bbox)):

    # Extracts XXXXX value from 'vid_4_XXXXX.jpg' file name
    file_name = bbox['image'][i]
    position1= file_name.index('4_')
    position2=file_name.index('.')
    file_number= file_name[position1+2:position2]
    train_car[i] = file_number # XXXXX Value for all Training images with cars


# Extract all Training Images with associated Image Number
train_all = []
image_list = []
# Extracts ALL training images
for file_name in glob.glob('car_detection_dataset/training_images/*.jpg'):

    # Extracts XXXXX value from 'vid_4_XXXXX.jpg' file name

    position1= file_name.index('4_')
    position2=file_name.index('.')
    file_number= file_name[position1+2:position2]
    train_all.append(file_number) # XXXXX Value for ALL Training images

    im=np.array(Image.open(file_name)).flatten()
    image_list.append(im) # ALL Training images in the form of a flattened array

image_list = np.array(image_list)

# Define whether or not image has car
training_all = np.zeros([len(train_all), 2])
training_all[:,0] = train_all
for i in range(len(train_car)):
    car = np.where(train_car[i] == training_all[:,0])
    training_all[car,1] = 1
training_all = training_all.astype(int) # 2 column array specifying XXXXX value for all training samples and 0 or 1 if car is present
```

In [8]:
```
print(training_all)
print(image_list.shape)
joblib.dump(training_all, 'training_all.pkl')
joblib.dump(image_list, 'image_list.pkl')
```

```
[[13800     1]
 [ 4100     0]
 [10520     1]
 ...
 [26160     0]
 [18440     0]
 [ 7220     0]]
(1001, 770640)
```

Out[8]:
```
['image_list.pkl']
```

In [7]:
```
# Stratified Train Test Split Training Data
X_training, X_val, t_training, t_val = train_test_split(image_list,
                                                        training_all[:,1],
                                                        shuffle=True,
                                                        test_size=0.2,
                                                       stratify = training_all[:,1],
                                                      random_state = 42)
print(f"X_training Shape: {X_training.shape}")
print(f"X_val Shape: {X_val.shape}")
```

```
X_training Shape: (800, 770640)
X_val Shape: (201, 770640)
```

In [8]:
```
# Scale Training Data
scaler = MinMaxScaler()
X_training = scaler.fit_transform(X_training)
X_val = scaler.transform(X_val)

# Reshape Images to Accomodate Neural Network
X_train_rs = X_training.reshape(800,380,676,3)
X_val_rs = X_val.reshape(201,380,676,3)
```

In [9]:
```
# Import MobileNetV2 Transfer Learning Model
CNN_model = tf.keras.applications.MobileNetV2(weights = "imagenet",
                                              input_shape=(224, 224, 3),
                                              include_top=False)
```

```
# Model Weights Are Trainable
CNN_model.trainable = True

# Let's take a look to see how many layers are in the base model

# Fine-tune from this layer onwards
fine_tune_at = 224

# Freeze all the layers before the `fine_tune_at` layer
for layer in CNN_model.layers[:fine_tune_at]:
    layer.trainable = False


IMG_SIZE = 224 # MobileNetV2 Default Image Size
inputs = keras.Input(shape=(380, 676, 3)) # Input Image Size
inputs_resized = keras.layers.Resizing(IMG_SIZE, IMG_SIZE)(inputs) # 150-150-3 # Input layer
x = CNN_model(inputs_resized, training=False) # resizing input to match pretrained model
x_pooling = keras.layers.GlobalAveragePooling2D()(x)
outputs = keras.layers.Dense(2, activation='softmax')(x_pooling) # Output Layer
model_MNV2 = tf.keras.Model(inputs, outputs)

# Model Summary
model_MNV2.summary()

warnings.filterwarnings('ignore')
```

```
2022-12-10 01:52:00.859116: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-10 01:52:01.284448: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 79111 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:07:00.0, compute capability: 8.0
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 380, 676, 3)]     0

 resizing (Resizing)         (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [ ]:

```
# Hyperparameter Values Tested
# *Note: When running GridSearchCV, RandomSearchCV, and iterative tuning with for-loops, the kerenel
# would crash, due to memory limitations. The following hyperparameters were, therefore, implemented manually
# and evaluated based on the MSE of the validation set.
# batch_size [16, 32, 48]
# tuning_layer = [50, 100, 224] # Transfer learning layer to begin retraining weights
```

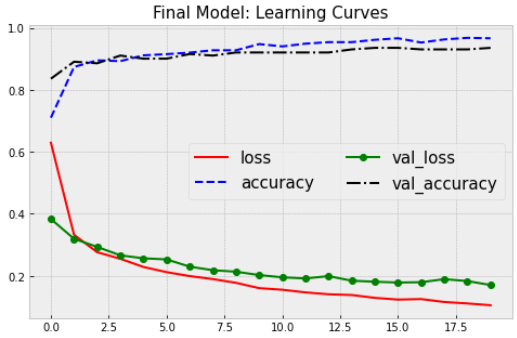
In [10]:

```
# Compile Car Classification Model
model_MNV2.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# Fit Car Classification
model_MNV2.fit(X_train_rs,t_training, epochs=20, batch_size=32,
          validation_data=(X_val_rs, t_val),
          callbacks=[keras.callbacks.EarlyStopping(patience=10)])

warnings.filterwarnings("ignore")
```

```
Epoch 1/20
2022-12-10 01:52:10.998681: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8201
2022-12-10 01:52:12.037517: W tensorflow/stream_executor/gpu/asm_compiler.cc:80] Couldn't get ptxas version string: INTERNAL: Running ptxas --version returned 32512
2022-12-10 01:52:12.223081: W tensorflow/stream_executor/gpu/redzone_allocator.cc:314] INTERNAL: ptxas exited with non-zero error code 32512, output:
Relying on driver to perform ptx compilation.
Modify $PATH to customize ptxas location.
This message will be only logged once.
 5/25 [=====>.....................] - ETA: 0s - loss: 0.9157 - accuracy: 0.5562
2022-12-10 01:52:13.165498: I tensorflow/stream_executor/cuda/cuda_blas.cc:1774] TensorFloat-32 will be used for the matrix multiplication. This will only be logged once.
```

```
25/25 [==============================] - 7s 115ms/step - loss: 0.6294 - accuracy: 0.7100 - val_loss: 0.3834 - val_accuracy: 0.8358
Epoch 2/20
25/25 [==============================] - 2s 63ms/step - loss: 0.3325 - accuracy: 0.8737 - val_loss: 0.3195 - val_accuracy: 0.8905
Epoch 3/20
25/25 [==============================] - 2s 62ms/step - loss: 0.2764 - accuracy: 0.8950 - val_loss: 0.2934 - val_accuracy: 0.8856
Epoch 4/20
25/25 [==============================] - 2s 62ms/step - loss: 0.2551 - accuracy: 0.8925 - val_loss: 0.2663 - val_accuracy: 0.9104
Epoch 5/20
25/25 [==============================] - 2s 62ms/step - loss: 0.2287 - accuracy: 0.9112 - val_loss: 0.2566 - val_accuracy: 0.9005
Epoch 6/20
25/25 [==============================] - 2s 63ms/step - loss: 0.2120 - accuracy: 0.9150 - val_loss: 0.2531 - val_accuracy: 0.9005
Epoch 7/20
25/25 [==============================] - 2s 68ms/step - loss: 0.1994 - accuracy: 0.9200 - val_loss: 0.2297 - val_accuracy: 0.9154
Epoch 8/20
25/25 [==============================] - 2s 78ms/step - loss: 0.1895 - accuracy: 0.9275 - val_loss: 0.2182 - val_accuracy: 0.9104
Epoch 9/20
25/25 [==============================] - 2s 63ms/step - loss: 0.1776 - accuracy: 0.9275 - val_loss: 0.2135 - val_accuracy: 0.9204
Epoch 10/20
25/25 [==============================] - 2s 62ms/step - loss: 0.1604 - accuracy: 0.9475 - val_loss: 0.2026 - val_accuracy: 0.9204
Epoch 11/20
25/25 [==============================] - 2s 62ms/step - loss: 0.1553 - accuracy: 0.9400 - val_loss: 0.1953 - val_accuracy: 0.9204
Epoch 12/20
25/25 [==============================] - 2s 62ms/step - loss: 0.1468 - accuracy: 0.9488 - val_loss: 0.1916 - val_accuracy: 0.9204
Epoch 13/20
25/25 [==============================] - 2s 67ms/step - loss: 0.1406 - accuracy: 0.9538 - val_loss: 0.1993 - val_accuracy: 0.9204
Epoch 14/20
25/25 [==============================] - 2s 72ms/step - loss: 0.1383 - accuracy: 0.9538 - val_loss: 0.1844 - val_accuracy: 0.9303
Epoch 15/20
25/25 [==============================] - 2s 68ms/step - loss: 0.1290 - accuracy: 0.9613 - val_loss: 0.1811 - val_accuracy: 0.9353
Epoch 16/20
25/25 [==============================] - 2s 62ms/step - loss: 0.1235 - accuracy: 0.9663 - val_loss: 0.1786 - val_accuracy: 0.9353
Epoch 17/20
25/25 [==============================] - 2s 63ms/step - loss: 0.1252 - accuracy: 0.9525 - val_loss: 0.1795 - val_accuracy: 0.9303
Epoch 18/20
25/25 [==============================] - 2s 62ms/step - loss: 0.1158 - accuracy: 0.9625 - val_loss: 0.1894 - val_accuracy: 0.9303
Epoch 19/20
25/25 [==============================] - 2s 62ms/step - loss: 0.1111 - accuracy: 0.9675 - val_loss: 0.1832 - val_accuracy: 0.9303
Epoch 20/20
25/25 [==============================] - 2s 65ms/step - loss: 0.1053 - accuracy: 0.9663 - val_loss: 0.1705 - val_accuracy: 0.9353
```

In [11]:
```
# Learning Curve for Tuned MobileNetV2 Classification Model
LearningCurve(model_MNV2)
```



In [18]:
```
model_MNV2.save('car_clf_model.h5')
```

Bounding Box Neural Network

In [5]:
```
N = len(bbox) # no. of training samples

# Create a numpy array with all images
for i in range(N):
    filename='car_detection_dataset/training_images/'+bbox['image'][i]
    image = np.array(Image.open(filename))
    image_col = image.ravel()[:,np.newaxis]

    if i==0:
        X_train = image_col
    else:
        X_train = np.hstack((X_train, image_col))

# Training feature matrices
X_train = X_train.T

# Training labels
t_train = bbox.drop('image', axis=1).round().to_numpy().astype(int)

X_train.shape, t_train.shape
```

Out[5]:
```
((559, 770640), (559, 4))
```

In [6]:
```
# Test Train Split of Training Data
X_training, X_val, t_training, t_val = train_test_split(X_train,
                                                        t_train,
                                                        shuffle=True,
                                                        test_size=0.2,
                                                    random_state = 42)
print(f"X_training Shape: {X_training.shape}")
print(f"X_val Shape: {X_val.shape}")
```

```
X_training Shape: (447, 770640)
X_val Shape: (112, 770640)
```

In [7]:
```
# Performs Scaling on Training and Validation Data
scaler = MinMaxScaler()
X_training = scaler.fit_transform(X_training)
X_val = scaler.transform(X_val)

# Reshape Data to Accomodate Neural Network
```

```
X_train_rs = X_training.reshape(447,380,676,3)
X_val_rs = X_val.reshape(112,380,676,3)
```

In [8]:
```
# Object Detection Nueral Network
from keras.layers import Dense, Activation, Flatten

# Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning
box_model = tf.keras.applications.MobileNetV2(weights = "imagenet",
                                              input_shape=(224, 224, 3),
                                              include_top=False)

box_model.trainable = True

# Let's take a look to see how many layers are in the base model

# Fine-tune from this Layer onwards
fine_tune_at = 120

# Freeze all the layers before the `fine_tune_at` layer
for layer in box_model.layers[:fine_tune_at]:
    layer.trainable = False

IMG_SIZE = 224 # MobileNetV2 default Image Size
inputs = keras.Input(shape=(380, 676, 3)) # Input Image Size
inputs_resized = keras.layers.Resizing(IMG_SIZE, IMG_SIZE)(inputs) # 150-150-3 # Input layer
x = box_model(inputs_resized, training=False)

flatten = Flatten()(x)
# construct a fully-connected layer header to output the predicted
# bounding box coordinates
bboxHead = Dense(128, activation="relu")(flatten)
bboxHead = Dense(64, activation="relu")(bboxHead)
bboxHead = Dense(32, activation="relu")(bboxHead)
bboxHead = Dense(4, activation="softplus")(bboxHead) # ouputs regression values for bounding box verticies
model = tf.keras.Model(inputs=inputs, outputs=bboxHead)

model.summary()
```

```
2022-12-10 13:45:09.102530: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-10 13:45:09.628683: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 79111 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:4e:00.0, compute capability: 8.0
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 380, 676, 3)]     0

 resizing (Resizing)         (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 62720)             0

 dense (Dense)               (None, 128)               8028288

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 4)                 132

=================================================================
Total params: 10,296,740
Trainable params: 9,663,652
Non-trainable params: 633,088
_________________________________________________________________
```

In [ ]:
```
# Hyperparameter Values Tested
# *Note: When running GridSearchCV, RandomSearchCV, and iterative tuning with for-Loops, the kerenel
# would crash, due to memory limitations. The following hyperparameters were, therefore, implemented manually
# and evaluated based on the MSE of the validation set. First, a manually tuned model is shown, then the final model
# given the best hyperparameters
# batch_size [16, 32, 48]
# epochs = [25, 50, 75]
# tuning_layer = [50, 100, 120] # Transfer learning layer to begin retraining weights
```

In [12]:
```
# Example of tuning model using various hyperparameters (tested manually)
opt = optimizer=tf.keras.optimizers.Adam(learning_rate = 0.001)
model.compile(loss="mse", optimizer=opt, metrics=keras.metrics.MeanSquaredError())
# train the network for bounding box regression
model.fit(X_train_rs,
          t_training,
          validation_data=(X_val_rs, t_val),
          batch_size=16,
          epochs=50,
          verbose=1)
```

```
Epoch 1/50
28/28 [==============================] - 4s 81ms/step - loss: 30399.3691 - mean_squared_error: 30399.3691 - val_loss: 20539.4980 - val_mean_squared_error: 20539.4980
Epoch 2/50
28/28 [==============================] - 1s 39ms/step - loss: 23534.6738 - mean_squared_error: 23534.6738 - val_loss: 20244.6289 - val_mean_squared_error: 20244.6289
Epoch 3/50
28/28 [==============================] - 1s 39ms/step - loss: 23087.6582 - mean_squared_error: 23087.6582 - val_loss: 19653.5645 - val_mean_squared_error: 19653.5645
Epoch 4/50
28/28 [==============================] - 1s 39ms/step - loss: 24672.6250 - mean_squared_error: 24672.6250 - val_loss: 21303.4219 - val_mean_squared_error: 21303.4219
Epoch 5/50
28/28 [==============================] - 1s 39ms/step - loss: 25026.5684 - mean_squared_error: 25026.5684 - val_loss: 19875.7656 - val_mean_squared_error: 19875.7656
Epoch 6/50
28/28 [==============================] - 1s 39ms/step - loss: 24916.1133 - mean_squared_error: 24916.1133 - val_loss: 22417.0898 - val_mean_squared_error: 22417.0898
Epoch 7/50
28/28 [==============================] - 1s 39ms/step - loss: 23302.2422 - mean_squared_error: 23302.2422 - val_loss: 21007.3848 - val_mean_squared_error: 21007.3848
Epoch 8/50
28/28 [==============================] - 1s 39ms/step - loss: 23122.5605 - mean_squared_error: 23122.5605 - val_loss: 20659.6621 - val_mean_squared_error: 20659.6621
Epoch 9/50
28/28 [==============================] - 1s 39ms/step - loss: 23501.6562 - mean_squared_error: 23501.6562 - val_loss: 20289.5918 - val_mean_squared_error: 20289.5918
Epoch 10/50
28/28 [==============================] - 1s 41ms/step - loss: 23824.4336 - mean_squared_error: 23824.4336 - val_loss: 24814.9707 - val_mean_squared_error: 24814.9707
Epoch 11/50
28/28 [==============================] - 1s 39ms/step - loss: 23119.2402 - mean_squared_error: 23119.2402 - val_loss: 20380.9375 - val_mean_squared_error: 20380.9375
Epoch 12/50
28/28 [==============================] - 1s 39ms/step - loss: 23228.9785 - mean_squared_error: 23228.9785 - val_loss: 22636.2441 - val_mean_squared_error: 22636.2441
Epoch 13/50
28/28 [==============================] - 1s 39ms/step - loss: 23268.2402 - mean_squared_error: 23268.2402 - val_loss: 19461.3418 - val_mean_squared_error: 19461.3418
Epoch 14/50
28/28 [==============================] - 1s 44ms/step - loss: 24047.6680 - mean_squared_error: 24047.6680 - val_loss: 20613.4395 - val_mean_squared_error: 20613.4395
Epoch 15/50
28/28 [==============================] - 1s 43ms/step - loss: 23209.2305 - mean_squared_error: 23209.2305 - val_loss: 19481.5137 - val_mean_squared_error: 19481.5137
Epoch 16/50
28/28 [==============================] - 1s 39ms/step - loss: 22810.9219 - mean_squared_error: 22810.9219 - val_loss: 22870.8301 - val_mean_squared_error: 22870.8301
Epoch 17/50
28/28 [==============================] - 1s 40ms/step - loss: 23014.0020 - mean_squared_error: 23014.0020 - val_loss: 21055.7324 - val_mean_squared_error: 21055.7324
Epoch 18/50
28/28 [==============================] - 1s 39ms/step - loss: 22883.9395 - mean_squared_error: 22883.9395 - val_loss: 20601.7812 - val_mean_squared_error: 20601.7812
Epoch 19/50
28/28 [==============================] - 1s 39ms/step - loss: 23058.9746 - mean_squared_error: 23058.9746 - val_loss: 19684.2285 - val_mean_squared_error: 19684.2285
Epoch 20/50
28/28 [==============================] - 1s 39ms/step - loss: 22999.4102 - mean_squared_error: 22999.4102 - val_loss: 20824.5469 - val_mean_squared_error: 20824.5469
Epoch 21/50
28/28 [==============================] - 1s 41ms/step - loss: 22775.4180 - mean_squared_error: 22775.4180 - val_loss: 20225.9043 - val_mean_squared_error: 20225.9043
Epoch 22/50
28/28 [==============================] - 1s 39ms/step - loss: 22923.3574 - mean_squared_error: 22923.3574 - val_loss: 21353.7910 - val_mean_squared_error: 21353.7910
Epoch 23/50
28/28 [==============================] - 1s 39ms/step - loss: 23047.6348 - mean_squared_error: 23047.6348 - val_loss: 20341.3887 - val_mean_squared_error: 20341.3887
Epoch 24/50
28/28 [==============================] - 1s 39ms/step - loss: 23165.8984 - mean_squared_error: 23165.8984 - val_loss: 20598.0137 - val_mean_squared_error: 20598.0137
Epoch 25/50
28/28 [==============================] - 1s 40ms/step - loss: 22818.7090 - mean_squared_error: 22818.7090 - val_loss: 19776.9199 - val_mean_squared_error: 19776.9199
Epoch 26/50
28/28 [==============================] - 1s 42ms/step - loss: 22945.2129 - mean_squared_error: 22945.2129 - val_loss: 21472.8164 - val_mean_squared_error: 21472.8164
Epoch 27/50
28/28 [==============================] - 1s 41ms/step - loss: 22811.8457 - mean_squared_error: 22811.8457 - val_loss: 19581.2305 - val_mean_squared_error: 19581.2305
Epoch 28/50
28/28 [==============================] - 1s 40ms/step - loss: 22896.2578 - mean_squared_error: 22896.2578 - val_loss: 22257.5977 - val_mean_squared_error: 22257.5977
Epoch 29/50
28/28 [==============================] - 1s 39ms/step - loss: 23148.0918 - mean_squared_error: 23148.0918 - val_loss: 20421.1836 - val_mean_squared_error: 20421.1836
Epoch 30/50
28/28 [==============================] - 1s 39ms/step - loss: 24032.5273 - mean_squared_error: 24032.5273 - val_loss: 20968.6035 - val_mean_squared_error: 20968.6035
Epoch 31/50
28/28 [==============================] - 1s 40ms/step - loss: 23313.2480 - mean_squared_error: 23313.2480 - val_loss: 20835.5410 - val_mean_squared_error: 20835.5410
Epoch 32/50
28/28 [==============================] - 1s 39ms/step - loss: 23239.2676 - mean_squared_error: 23239.2676 - val_loss: 19637.0156 - val_mean_squared_error: 19637.0156
Epoch 33/50
28/28 [==============================] - 1s 39ms/step - loss: 23070.3379 - mean_squared_error: 23070.3379 - val_loss: 19630.0371 - val_mean_squared_error: 19630.0371
Epoch 34/50
28/28 [==============================] - 1s 39ms/step - loss: 23151.8105 - mean_squared_error: 23151.8105 - val_loss: 20068.0312 - val_mean_squared_error: 20068.0312
Epoch 35/50
28/28 [==============================] - 1s 39ms/step - loss: 22843.2012 - mean_squared_error: 22843.2012 - val_loss: 22176.3574 - val_mean_squared_error: 22176.3574
Epoch 36/50
28/28 [==============================] - 1s 39ms/step - loss: 23283.0723 - mean_squared_error: 23283.0723 - val_loss: 21833.2773 - val_mean_squared_error: 21833.2773
Epoch 37/50
28/28 [==============================] - 1s 40ms/step - loss: 23043.4453 - mean_squared_error: 23043.4453 - val_loss: 21861.2090 - val_mean_squared_error: 21861.2090
Epoch 38/50
28/28 [==============================] - 1s 39ms/step - loss: 23239.9980 - mean_squared_error: 23239.9980 - val_loss: 20348.8652 - val_mean_squared_error: 20348.8652
Epoch 39/50
28/28 [==============================] - 1s 40ms/step - loss: 23136.4492 - mean_squared_error: 23136.4492 - val_loss: 21572.5410 - val_mean_squared_error: 21572.5410
Epoch 40/50
28/28 [==============================] - 1s 39ms/step - loss: 23018.0020 - mean_squared_error: 23018.0020 - val_loss: 20701.8242 - val_mean_squared_error: 20701.8242
Epoch 41/50
28/28 [==============================] - 1s 39ms/step - loss: 23057.7070 - mean_squared_error: 23057.7070 - val_loss: 21947.2246 - val_mean_squared_error: 21947.2246
Epoch 42/50
28/28 [==============================] - 1s 40ms/step - loss: 23182.6582 - mean_squared_error: 23182.6582 - val_loss: 20906.1543 - val_mean_squared_error: 20906.1543
Epoch 43/50
28/28 [==============================] - 1s 39ms/step - loss: 23566.3574 - mean_squared_error: 23566.3574 - val_loss: 21216.3027 - val_mean_squared_error: 21216.3027
Epoch 44/50
28/28 [==============================] - 1s 40ms/step - loss: 22769.2812 - mean_squared_error: 22769.2812 - val_loss: 21147.4727 - val_mean_squared_error: 21147.4727
Epoch 45/50
28/28 [==============================] - 1s 42ms/step - loss: 22865.9941 - mean_squared_error: 22865.9941 - val_loss: 20073.5488 - val_mean_squared_error: 20073.5488
Epoch 46/50
28/28 [==============================] - 1s 40ms/step - loss: 22919.0625 - mean_squared_error: 22919.0625 - val_loss: 24122.4023 - val_mean_squared_error: 24122.4023
Epoch 47/50
28/28 [==============================] - 1s 42ms/step - loss: 23806.7910 - mean_squared_error: 23806.7910 - val_loss: 21041.1152 - val_mean_squared_error: 21041.1152
Epoch 48/50
28/28 [==============================] - 1s 41ms/step - loss: 22847.4414 - mean_squared_error: 22847.4414 - val_loss: 20714.3008 - val_mean_squared_error: 20714.3008
Epoch 49/50
28/28 [==============================] - 1s 40ms/step - loss: 22929.2148 - mean_squared_error: 22929.2148 - val_loss: 21551.0215 - val_mean_squared_error: 21551.0215
Epoch 50/50
28/28 [==============================] - 1s 39ms/step - loss: 22817.6621 - mean_squared_error: 22817.6621 - val_loss: 20519.2871 - val_mean_squared_error: 20519.2871
```

Out[12]:
```
<keras.callbacks.History at 0x2b549e9d56d0>
```

In [9]:
```
# Tuned Model using optimal hyperparameters (tested manually)
opt = optimizer=tf.keras.optimizers.Adam(learning_rate = 0.0001)
model.compile(loss="mse", optimizer=opt, metrics=keras.metrics.MeanSquaredError())
# train the network for bounding box regression
model.fit(X_train_rs,
          t_training,
          validation_data=(X_val_rs, t_val),
          batch_size=32,
```

```
          epochs=75,
          verbose=1)
```

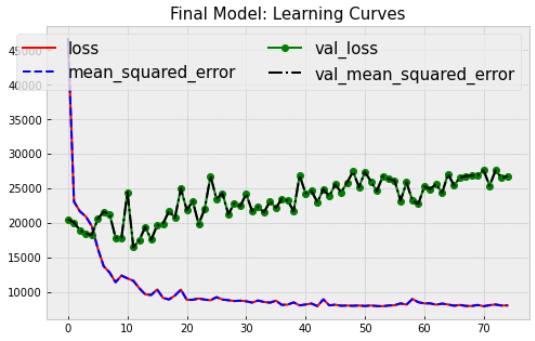
```
Epoch 1/75
2022-12-10 13:45:21.583474: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8201
2022-12-10 13:45:22.696277: W tensorflow/stream_executor/gpu/asm_compiler.cc:80] Couldn't get ptxas version string: INTERNAL: Running ptxas --version returned 32512
2022-12-10 13:45:22.892245: W tensorflow/stream_executor/gpu/redzone_allocator.cc:314] INTERNAL: ptxas exited with non-zero error code 32512, output:
Relying on driver to perform ptx compilation.
Modify $PATH to customize ptxas location.
This message will be only logged once.
2022-12-10 13:45:23.873933: I tensorflow/stream_executor/cuda/cuda_blas.cc:1774] TensorFloat-32 will be used for the matrix multiplication. This will only be logged once.
```

```
14/14 [==============================] - 8s 182ms/step - loss: 46571.1758 - mean_squared_error: 46571.1758 - val_loss: 20476.7402 - val_mean_squared_error: 20476.7402
Epoch 2/75
14/14 [==============================] - 1s 69ms/step - loss: 23025.8066 - mean_squared_error: 23025.8066 - val_loss: 20013.0820 - val_mean_squared_error: 20013.0820
Epoch 3/75
14/14 [==============================] - 1s 67ms/step - loss: 21624.5410 - mean_squared_error: 21624.5410 - val_loss: 18928.7363 - val_mean_squared_error: 18928.7363
Epoch 4/75
14/14 [==============================] - 1s 68ms/step - loss: 20879.5938 - mean_squared_error: 20879.5938 - val_loss: 18442.3145 - val_mean_squared_error: 18442.3145
Epoch 5/75
14/14 [==============================] - 1s 79ms/step - loss: 19491.4766 - mean_squared_error: 19491.4766 - val_loss: 18202.5371 - val_mean_squared_error: 18202.5371
Epoch 6/75
14/14 [==============================] - 1s 70ms/step - loss: 16284.7568 - mean_squared_error: 16284.7568 - val_loss: 20534.2031 - val_mean_squared_error: 20534.2031
Epoch 7/75
14/14 [==============================] - 1s 71ms/step - loss: 13691.3711 - mean_squared_error: 13691.3711 - val_loss: 21569.1680 - val_mean_squared_error: 21569.1680
Epoch 8/75
14/14 [==============================] - 1s 68ms/step - loss: 12795.1904 - mean_squared_error: 12795.1904 - val_loss: 21193.2285 - val_mean_squared_error: 21193.2285
Epoch 9/75
14/14 [==============================] - 1s 67ms/step - loss: 11377.6895 - mean_squared_error: 11377.6895 - val_loss: 17834.8457 - val_mean_squared_error: 17834.8457
Epoch 10/75
14/14 [==============================] - 1s 67ms/step - loss: 12351.3721 - mean_squared_error: 12351.3721 - val_loss: 17806.8398 - val_mean_squared_error: 17806.8398
Epoch 11/75
14/14 [==============================] - 1s 67ms/step - loss: 11943.1367 - mean_squared_error: 11943.1367 - val_loss: 24390.9824 - val_mean_squared_error: 24390.9824
Epoch 12/75
14/14 [==============================] - 1s 68ms/step - loss: 11594.0225 - mean_squared_error: 11594.0225 - val_loss: 16452.1719 - val_mean_squared_error: 16452.1719
Epoch 13/75
14/14 [==============================] - 1s 71ms/step - loss: 10514.3262 - mean_squared_error: 10514.3262 - val_loss: 17455.5234 - val_mean_squared_error: 17455.5234
Epoch 14/75
14/14 [==============================] - 1s 76ms/step - loss: 9627.5137 - mean_squared_error: 9627.5137 - val_loss: 19380.9316 - val_mean_squared_error: 19380.9316
Epoch 15/75
14/14 [==============================] - 1s 73ms/step - loss: 9538.1143 - mean_squared_error: 9538.1143 - val_loss: 17532.4082 - val_mean_squared_error: 17532.4082
Epoch 16/75
14/14 [==============================] - 1s 68ms/step - loss: 10338.4395 - mean_squared_error: 10338.4395 - val_loss: 19600.0527 - val_mean_squared_error: 19600.0527
Epoch 17/75
14/14 [==============================] - 1s 74ms/step - loss: 9117.9160 - mean_squared_error: 9117.9160 - val_loss: 19858.8320 - val_mean_squared_error: 19858.8320
Epoch 18/75
14/14 [==============================] - 1s 69ms/step - loss: 8891.3213 - mean_squared_error: 8891.3213 - val_loss: 21684.2773 - val_mean_squared_error: 21684.2773
Epoch 19/75
14/14 [==============================] - 1s 70ms/step - loss: 9463.5029 - mean_squared_error: 9463.5029 - val_loss: 20718.6895 - val_mean_squared_error: 20718.6895
Epoch 20/75
14/14 [==============================] - 1s 68ms/step - loss: 10315.7305 - mean_squared_error: 10315.7305 - val_loss: 24934.6152 - val_mean_squared_error: 24934.6152
Epoch 21/75
14/14 [==============================] - 1s 70ms/step - loss: 8847.9268 - mean_squared_error: 8847.9268 - val_loss: 21819.4414 - val_mean_squared_error: 21819.4414
Epoch 22/75
14/14 [==============================] - 1s 69ms/step - loss: 8852.2715 - mean_squared_error: 8852.2715 - val_loss: 23109.8477 - val_mean_squared_error: 23109.8477
Epoch 23/75
14/14 [==============================] - 1s 70ms/step - loss: 9031.8613 - mean_squared_error: 9031.8613 - val_loss: 19769.0469 - val_mean_squared_error: 19769.0469
Epoch 24/75
14/14 [==============================] - 1s 74ms/step - loss: 8875.7783 - mean_squared_error: 8875.7783 - val_loss: 21952.4277 - val_mean_squared_error: 21952.4277
Epoch 25/75
14/14 [==============================] - 1s 73ms/step - loss: 8776.7441 - mean_squared_error: 8776.7441 - val_loss: 26738.6934 - val_mean_squared_error: 26738.6934
Epoch 26/75
14/14 [==============================] - 1s 73ms/step - loss: 9223.4258 - mean_squared_error: 9223.4258 - val_loss: 23434.3438 - val_mean_squared_error: 23434.3438
Epoch 27/75
14/14 [==============================] - 1s 70ms/step - loss: 8888.8174 - mean_squared_error: 8888.8174 - val_loss: 24235.9688 - val_mean_squared_error: 24235.9688
Epoch 28/75
14/14 [==============================] - 1s 69ms/step - loss: 8781.3115 - mean_squared_error: 8781.3115 - val_loss: 21229.1270 - val_mean_squared_error: 21229.1270
Epoch 29/75
14/14 [==============================] - 1s 69ms/step - loss: 8668.3467 - mean_squared_error: 8668.3467 - val_loss: 22763.3848 - val_mean_squared_error: 22763.3848
Epoch 30/75
14/14 [==============================] - 1s 71ms/step - loss: 8718.2227 - mean_squared_error: 8718.2227 - val_loss: 22380.3477 - val_mean_squared_error: 22380.3477
Epoch 31/75
14/14 [==============================] - 1s 69ms/step - loss: 8633.5928 - mean_squared_error: 8633.5928 - val_loss: 24160.8711 - val_mean_squared_error: 24160.8711
Epoch 32/75
14/14 [==============================] - 1s 69ms/step - loss: 8443.7285 - mean_squared_error: 8443.7285 - val_loss: 21666.9199 - val_mean_squared_error: 21666.9199
Epoch 33/75
14/14 [==============================] - 1s 71ms/step - loss: 8742.9043 - mean_squared_error: 8742.9043 - val_loss: 22240.0039 - val_mean_squared_error: 22240.0039
Epoch 34/75
14/14 [==============================] - 1s 69ms/step - loss: 8535.5361 - mean_squared_error: 8535.5361 - val_loss: 21476.1934 - val_mean_squared_error: 21476.1934
Epoch 35/75
14/14 [==============================] - 1s 70ms/step - loss: 8426.3896 - mean_squared_error: 8426.3896 - val_loss: 23089.2695 - val_mean_squared_error: 23089.2695
Epoch 36/75
14/14 [==============================] - 1s 71ms/step - loss: 8716.5127 - mean_squared_error: 8716.5127 - val_loss: 22118.0801 - val_mean_squared_error: 22118.0801
Epoch 37/75
14/14 [==============================] - 1s 69ms/step - loss: 8108.5801 - mean_squared_error: 8108.5801 - val_loss: 23436.9688 - val_mean_squared_error: 23436.9688
Epoch 38/75
14/14 [==============================] - 1s 71ms/step - loss: 8150.4160 - mean_squared_error: 8150.4160 - val_loss: 23167.7090 - val_mean_squared_error: 23167.7090
Epoch 39/75
14/14 [==============================] - 1s 70ms/step - loss: 8466.8516 - mean_squared_error: 8466.8516 - val_loss: 21662.9336 - val_mean_squared_error: 21662.9336
Epoch 40/75
14/14 [==============================] - 1s 70ms/step - loss: 8035.8452 - mean_squared_error: 8035.8452 - val_loss: 26835.2344 - val_mean_squared_error: 26835.2344
Epoch 41/75
14/14 [==============================] - 1s 72ms/step - loss: 8165.8506 - mean_squared_error: 8165.8506 - val_loss: 24242.1660 - val_mean_squared_error: 24242.1660
Epoch 42/75
14/14 [==============================] - 1s 71ms/step - loss: 8303.7529 - mean_squared_error: 8303.7529 - val_loss: 24583.1387 - val_mean_squared_error: 24583.1387
Epoch 43/75
14/14 [==============================] - 1s 73ms/step - loss: 7930.8350 - mean_squared_error: 7930.8350 - val_loss: 22940.1309 - val_mean_squared_error: 22940.1309
Epoch 44/75
14/14 [==============================] - 1s 73ms/step - loss: 8908.8750 - mean_squared_error: 8908.8750 - val_loss: 24845.4648 - val_mean_squared_error: 24845.4648
Epoch 45/75
14/14 [==============================] - 1s 69ms/step - loss: 8041.7695 - mean_squared_error: 8041.7695 - val_loss: 23833.0625 - val_mean_squared_error: 23833.0625
Epoch 46/75
14/14 [==============================] - 1s 69ms/step - loss: 8138.7920 - mean_squared_error: 8138.7920 - val_loss: 25524.1836 - val_mean_squared_error: 25524.1836
Epoch 47/75
14/14 [==============================] - 1s 69ms/step - loss: 7982.1353 - mean_squared_error: 7982.1353 - val_loss: 24398.9785 - val_mean_squared_error: 24398.9785
Epoch 48/75
14/14 [==============================] - 1s 69ms/step - loss: 8015.8022 - mean_squared_error: 8015.8022 - val_loss: 25796.0625 - val_mean_squared_error: 25796.0625
Epoch 49/75
14/14 [==============================] - 1s 70ms/step - loss: 7973.4048 - mean_squared_error: 7973.4048 - val_loss: 27405.8301 - val_mean_squared_error: 27405.8301
Epoch 50/75
14/14 [==============================] - 1s 69ms/step - loss: 8021.6592 - mean_squared_error: 8021.6592 - val_loss: 25126.1230 - val_mean_squared_error: 25126.1230
Epoch 51/75
14/14 [==============================] - 1s 72ms/step - loss: 7946.4565 - mean_squared_error: 7946.4565 - val_loss: 27336.9062 - val_mean_squared_error: 27336.9062
Epoch 52/75
14/14 [==============================] - 1s 69ms/step - loss: 8023.7490 - mean_squared_error: 8023.7490 - val_loss: 25967.8027 - val_mean_squared_error: 25967.8027
Epoch 53/75
14/14 [==============================] - 1s 68ms/step - loss: 7942.2676 - mean_squared_error: 7942.2676 - val_loss: 24578.0449 - val_mean_squared_error: 24578.0449
Epoch 54/75
14/14 [==============================] - 1s 69ms/step - loss: 7917.3105 - mean_squared_error: 7917.3105 - val_loss: 26663.6035 - val_mean_squared_error: 26663.6035
Epoch 55/75
14/14 [==============================] - 1s 69ms/step - loss: 8018.3145 - mean_squared_error: 8018.3145 - val_loss: 26303.9570 - val_mean_squared_error: 26303.9570
Epoch 56/75
14/14 [==============================] - 1s 69ms/step - loss: 8058.2129 - mean_squared_error: 8058.2129 - val_loss: 26055.0273 - val_mean_squared_error: 26055.0273
```

```
Epoch 57/75
14/14 [==============================] - 1s 70ms/step - loss: 8321.0781 - mean_squared_error: 8321.0781 - val_loss: 23153.7539 - val_mean_squared_error: 23153.7539
Epoch 58/75
14/14 [==============================] - 1s 70ms/step - loss: 8145.0381 - mean_squared_error: 8145.0381 - val_loss: 25820.1738 - val_mean_squared_error: 25820.1738
Epoch 59/75
14/14 [==============================] - 1s 69ms/step - loss: 8958.6973 - mean_squared_error: 8958.6973 - val_loss: 23198.6152 - val_mean_squared_error: 23198.6152
Epoch 60/75
14/14 [==============================] - 1s 70ms/step - loss: 8490.6426 - mean_squared_error: 8490.6426 - val_loss: 22722.6426 - val_mean_squared_error: 22722.6426
Epoch 61/75
14/14 [==============================] - 1s 69ms/step - loss: 8323.3203 - mean_squared_error: 8323.3203 - val_loss: 25217.2754 - val_mean_squared_error: 25217.2754
Epoch 62/75
14/14 [==============================] - 1s 70ms/step - loss: 8329.2393 - mean_squared_error: 8329.2393 - val_loss: 24872.8262 - val_mean_squared_error: 24872.8262
Epoch 63/75
14/14 [==============================] - 1s 69ms/step - loss: 8133.8911 - mean_squared_error: 8133.8911 - val_loss: 25570.1152 - val_mean_squared_error: 25570.1152
Epoch 64/75
14/14 [==============================] - 1s 71ms/step - loss: 8293.3652 - mean_squared_error: 8293.3652 - val_loss: 24287.5723 - val_mean_squared_error: 24287.5723
Epoch 65/75
14/14 [==============================] - 1s 71ms/step - loss: 8163.4551 - mean_squared_error: 8163.4551 - val_loss: 27057.2188 - val_mean_squared_error: 27057.2188
Epoch 66/75
14/14 [==============================] - 1s 71ms/step - loss: 7988.2798 - mean_squared_error: 7988.2798 - val_loss: 25369.1934 - val_mean_squared_error: 25369.1934
Epoch 67/75
14/14 [==============================] - 1s 71ms/step - loss: 8099.1333 - mean_squared_error: 8099.1333 - val_loss: 26528.3770 - val_mean_squared_error: 26528.3770
Epoch 68/75
14/14 [==============================] - 1s 72ms/step - loss: 7956.7017 - mean_squared_error: 7956.7017 - val_loss: 26721.7188 - val_mean_squared_error: 26721.7188
Epoch 69/75
14/14 [==============================] - 1s 72ms/step - loss: 7949.2964 - mean_squared_error: 7949.2964 - val_loss: 26813.3770 - val_mean_squared_error: 26813.3770
Epoch 70/75
14/14 [==============================] - 1s 72ms/step - loss: 8097.1968 - mean_squared_error: 8097.1968 - val_loss: 26835.2812 - val_mean_squared_error: 26835.2812
Epoch 71/75
14/14 [==============================] - 1s 70ms/step - loss: 7914.6602 - mean_squared_error: 7914.6602 - val_loss: 27560.6973 - val_mean_squared_error: 27560.6973
Epoch 72/75
14/14 [==============================] - 1s 70ms/step - loss: 8063.7007 - mean_squared_error: 8063.7007 - val_loss: 25334.2988 - val_mean_squared_error: 25334.2988
Epoch 73/75
14/14 [==============================] - 1s 69ms/step - loss: 8164.9531 - mean_squared_error: 8164.9531 - val_loss: 27556.4336 - val_mean_squared_error: 27556.4336
Epoch 74/75
14/14 [==============================] - 1s 70ms/step - loss: 8014.2002 - mean_squared_error: 8014.2002 - val_loss: 26502.7949 - val_mean_squared_error: 26502.7949
Epoch 75/75
14/14 [==============================] - 1s 68ms/step - loss: 8027.6196 - mean_squared_error: 8027.6196 - val_loss: 26712.8848 - val_mean_squared_error: 26712.8848
```

Out[9]:
```
<keras.callbacks.History at 0x2b7c526cac40>
```

In [10]:
```
# Learning Curve for Bounding Box Detection Model
LearningCurve(model)
```



In [11]:
```
model.save('car_clf_box.h5')
```

```
/apps/tensorflow/2.7.0/lib/python3.9/site-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must override get_config. When loading, the custom mask layer must be passed to the custom_objects argument.
  layer_config = serialize_layer_fn(layer)
```