In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from PIL import Image
from urllib import request
from io import BytesIO
```

In [2]:
```
import tensorflow as tf
import cv2
from tensorflow.keras import utils
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import Model
from keras.callbacks import ReduceLROnPlateau
```

In [3]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [4]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

0. Data Pre-Processing

In [5]:
```
# df = pd.read_csv('/cloud/msca-gcp/chuyucc/Flat/df_class_20_cleaned.csv',index_col=0)
```

In [6]:
```
# df.isnull().sum()
```

In [7]:
```
# df.dropna(subset=['SERIES'],inplace=True)
```

In [8]:
```
# df['MAKE1_MODEL'] = df['MAKE_1'] + " " + df['MODEL']
# df['MAKE1_MODEL_SERIE'] = df['MAKE_1'] + " " + df['MODEL'] + " " + df['SERIES']
```

In [9]:
```
# ## Create a function to collect top 5 models for each make

# def TopModel(make):
#     model_name = list(df[df['MAKE_1']==make]['MODEL'].value_counts().index)[:5]
#     return model_name
```

In [10]:
```
# ## Only use Model Year between 2011 and 2022

# df = df[df['MODEL_YEAR']>=2011]
```

In [11]:
```
# ## Only use front photos

# df = df[df['IMAGE_CAPTION']=='Front Photo']

# df = df.reset_index(drop=True)
```

In [12]:
```
# top_make_list = [
#     'FORD',
#     'CHEVROLET',
#     'TOYOTA',
#     'NISSAN',
#     'JEEP',
#     'HONDA',
#     'DODGE',
#     'MERCEDES-BENZ',
#     'RAM',
#     'HYUNDAI',
#     'GMC',
```

```
#     'KIA',
#     'VOLKSWAGEN',
#     'BMW',
#     'CHRYSLER',
#     'AUDI',
#     'SUBARU',
#     'BUICK',
#     'CADILLAC',
#     'MAZDA']
```

In [13]:
```
# c1_make_dict = {
#     'FORD': 0,
#     'CHEVROLET': 1,
#     'TOYOTA': 2,
#     'NISSAN': 3,
#     'JEEP': 4,
#     'HONDA': 5,
#     'DODGE': 6,
#     'MERCEDES-BENZ': 7,
#     'RAM': 8,
#     'HYUNDAI': 9,
#     'GMC': 10,
#     'KIA': 11,
#     'VOLKSWAGEN':12,
#     'BMW': 13,
#     'CHRYSLER': 14,
#     'AUDI': 15,
#     'SUBARU': 16,
#     'BUICK': 17,
#     'CADILLAC': 18,
#     'MAZDA':19,
#     'OTHER':20
# }
```

In [14]:
```
# df['c1_make'] = df['MAKE_1'].map(c1_make_dict)
```

In [15]:
```
# top_make_model_list = []
```

In [16]:
```
# ## Get top 100 Make-Model based on number of images avaiable

# for i in top_make_list:
#     for k in TopModel(str(i)):
#         top_make_model_list.append(i+' '+k)
```

In [17]:
```
# ## Create a new column 'MAKE1_MODEL1'

# for i in top_make_list:
#     df.loc[(~df['MAKE1_MODEL'].isin(top_make_model_list)&(df['MAKE_1']==i)),'MAKE1_MODEL1'] = str(i) + ' ' +
#     df.loc[(df['MAKE1_MODEL'].isin(top_make_model_list)&(df['MAKE_1']==i)),'MAKE1_MODEL1'] = df.loc[(df['MAK


# df.loc[df['MAKE_1']=='OTHER','MAKE1_MODEL1'] = 'OTHER OTHER'
```

In [18]:
```
# len(list(df['MAKE1_MODEL1'].value_counts().index))
```

In [19]:
```
# y_c2 = list(i for i in range(121))
```

In [20]:
```
# c2_make_model_dict = dict(zip(list(df['MAKE1_MODEL1'].value_counts().index),y_c2))
```

In [21]:
```
# df['c2_make_model'] = df['MAKE1_MODEL1'].map(c2_make_model_dict)
```

In [22]:
```
# df['c2_make_model'].value_counts()
```

In [23]:
```
# ## Function to capture the top 3 series for each Make-Model
```

```
# def TopSerie(model):
#     serie_name = list(df[df['MODEL']==model]['SERIES'].value_counts().index)[:3]
#     return serie_name
```

In [24]:
```
# ## Get Top 300 Series based on amount of images available

# top_make_model_serie_list = []
# count = []

# for n in top_make_list:
#     for k in TopModel(n):
#         for s in TopSerie(k):
#             ### Only select series that have a least 500 images
#             #if df[(df['MAKE_1']==n) & (df['MODEL']==k) & (df['SERIES']==s)].count()[0]>300:
#                 top_make_model_serie_list.append(n+' '+k+' '+s)
#                 count.append(df[(df['MAKE_1']==n) & (df['MODEL']==k) & (df['SERIES']==s)].count()[0]) ## To
```

In [25]:
```
# def classwithenoughimages(num):
#   start = 0

#   for i in count:
#     if i>= num:
#       start = start+1
#   return start
```

In [26]:
```
# ## 260 of 300 classes have at least 300 images

# classwithenoughimages(300)
```

In [27]:
```
# ## 213 of 300 classes have at least 500 images

# classwithenoughimages(500)
```

In [28]:
```
# ## Create a new column 'MAKE1_MODEL1_SERIE1'

# for i in top_make_model_list:
#   df.loc[(~df['MAKE1_MODEL_SERIE'].isin(top_make_model_serie_list)&(df['MAKE1_MODEL1']==i)),'MAKE1_MODEL1_SE
#   df.loc[(df['MAKE1_MODEL_SERIE'].isin(top_make_model_serie_list)&(df['MAKE1_MODEL1']==i)),'MAKE1_MODEL1_SER


# df.loc[(~df['MAKE1_MODEL1'].isin(top_make_model_list)),'MAKE1_MODEL1_SERIE1'] = df['MAKE1_MODEL1'] + ' ' + '

# df.loc[df['MAKE1_MODEL1']=='OTHER OTHER','MAKE1_MODEL1_SERIE1'] = 'OTHER OTHER OTHER'
```

In [29]:
```
# len(df['MAKE1_MODEL1_SERIE1'].unique()) ## There 421 unique values in MAKE1_MODEL1_SERIE1; 20*5*(3+1)+20+1
```

In [30]:
```
# y_c3 = list(i for i in range(421))
```

In [31]:
```
# c3_make_model_serie_dict = dict(zip(list(df['MAKE1_MODEL1_SERIE1'].value_counts().index),y_c3))
```

In [32]:
```
# df['c3_make_model_serie'] = df['MAKE1_MODEL1_SERIE1'].map(c3_make_model_serie_dict)
```

In [33]:
```
# df['c3_make_model_serie'].value_counts()
```

In [34]:
```
# c2 = []
# c3 = []

# for i in range(len(df['c3_make_model_serie'].unique())):
#   c3.append(i)
#   c2.append(df[df['c3_make_model_serie']==i]['c2_make_model'].unique().item())
```

In [35]:
```
# c3_to_c2 = dict(zip(c3,c2))
```

In [36]:
```
# df.to_csv('/content/drive/Shared drives/KAR Global/Branch/Threelayers/df_branch_threelayers.csv')
```

1. Image Downloading

In [37]:
```
pwd
```

Out[37]:
```
'/home/chuyucc/BCNN/Threelayers'
```

In [38]:
```
df = pd.read_csv('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/df_branch_threelayers.csv',index_col=0)
```

In [39]:
```
num_c3 = len(df['c3_make_model_serie'].unique())
```

In [40]:
```
num_c2 = len(df['c2_make_model'].unique())
```

In [41]:
```
# import urllib.request
# import os


# for i in range(num_c3):
#   path = os.path.sep.join(['/content/drive/Shared drives/KAR Global_1/Branch/Threelayers',str(i)])

#   if not os.path.exists(path):
#     os.mkdir(path)
#     os.chdir(path)

#     dfs = df[df['c3_make_model_serie']==i].reset_index(drop=True)

#     if len(dfs)>300: ### If there are more 300 images in the class, we would sample 300 only
#       dfc = dfs.sample(300).reset_index(drop=True)
#       for n in range(len(dfc)):
#         try:
#           urllib.request.urlretrieve(dfc['IMAGE_URL'][n], str(dfc['MAKE1_MODEL1_SERIE1'][n]) + " "+str(n))
#         except:
#           continue

#     else: ### collect all images for that class if there are fewer than 300 images
#       for n in range(len(dfs)):
#         try:
#           urllib.request.urlretrieve(dfs['IMAGE_URL'][n], str(dfs['MAKE1_MODEL1_SERIE1'][n]) + " "+str(n))
#         except:
#           continue

#   else:
#     continue
```

In [42]:
```
# import pathlib
# data_dir = pathlib.Path('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers')
```

In [43]:
```
# for i in range(421):
#   print('Folder '+ str(i) + ' has ' + str(len(list(data_dir.glob(str(i)+'/*')))))
```

In [44]:
```
# df[df['c3_make_model_serie']==419]['MODEL_YEAR'].value_counts()
```

In [45]:
```
# df[df['c3_make_model_serie']==419]
```

2. Image Processing

In [46]:
```
# import pathlib
# data_dir = pathlib.Path('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers')
```

In [47]:
```
# for i in range(30):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)

#   print('{} folder is done'.format(i))
```

In [48]:
```
# for i in range(30,50):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)

#   print('{} folder is done'.format(i))
```

In [49]:
```
# np.save('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers/X_0_50.npy',X)
# np.save('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers/y_0_50.npy',y)
```

In [50]:
```
# for i in range(50,100):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)

#   print('{} folder is done'.format(i))
```

In [51]:
```
# np.save('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers/X_50_100.npy',X)
# np.save('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers/y_50_100.npy',y)
```

In [52]:
```
# for i in range(100,150):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)

#   print('{} folder is done'.format(i))
```

In [53]:
```
# for i in range(150,200):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)

#   print('{} folder is done'.format(i))
```

In [54]:
```
# for i in range(200,300):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)

#   print('{} folder is done'.format(i))
```

In [55]:
```
# for i in range(300,421):
#   for n in list(data_dir.glob(str(i)+'/*')):
#     img = cv2.imread(str(n))
#     resized_img = cv2.resize(img,(224,224))
#     X.append(resized_img)
#     y.append(i)
```

```
#   print('{} folder is done'.format(i))
```

3. Modeling

In [56]:
```
X_0_50 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/X_0_50.npy')
X_50_100 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/X_50_100.npy')
X_100_200 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/X_100_200.npy')
X_200_300 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/X_200_300.npy')
X_300_421 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/X_300_421.npy')

y_0_50 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/y_0_50.npy')
y_50_100 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/y_50_100.npy')
y_100_200 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/y_100_200.npy')
y_200_300 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/y_200_300.npy')
y_300_421 = np.load('/cloud/msca-gcp/chuyucc/BCNN/Threelayers/y_300_421.npy')
```

In [57]:
```
X = np.append(X_0_50,X_50_100,axis=0)
y = np.append(y_0_50,y_50_100,axis=0)
```

In [58]:
```
X = np.append(X,X_100_200,axis=0)
y = np.append(y,y_100_200,axis=0)
```

In [59]:
```
X = np.append(X,X_200_300,axis=0)
y = np.append(y,y_200_300,axis=0)
```

In [60]:
```
X = np.append(X,X_300_421,axis=0)
y = np.append(y,y_300_421,axis=0)
```

In [61]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, stratify=y, test_size=0.2)
```

In [62]:
```
## Use 20% of the training dataset as validation dataset

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, random_state=42, stratify= y_train, test_s
```

In [63]:
```
del(X)
del(y)
```

In [64]:
```
# X_train = X_train/255
# X_test = X_test/255
# X_val = X_val/255
```

In [65]:
```
# df = pd.read_csv('/content/drive/Shared drives/KAR Global_1/Branch/Threelayers/df_branch_threelayers.csv')
```

In [66]:
```
c2 = []
c3 = []

for i in range(len(df['c3_make_model_serie'].unique())):
  c3.append(i)
  c2.append(df[df['c3_make_model_serie']==i]['c2_make_model'].unique().item())
```

In [67]:
```
c3_to_c2 = dict(zip(c3,c2))
```

In [68]:
```
num_c3 = len(c3) ## This should be 421
num_c2 = len(c2) ## This should be 121
```

In [69]:
```
```

```
c2 = []
c1 = []

for i in range(len(df['c2_make_model'].unique())):
  c2.append(i)
  c1.append(df[df['c2_make_model']==i]['c1_make'].unique().item())
```

In [70]:
```
c2_to_c1 = dict(zip(c2,c1))
```

In [71]:
```
num_c1 = len(c1) ## This should be 21
```

In [72]:
```
y_c3_train = tf.keras.utils.to_categorical(y_train,num_c3)
y_c3_test = tf.keras.utils.to_categorical(y_test,num_c3)
y_c3_val = tf.keras.utils.to_categorical(y_val,num_c3)
```

In [73]:
```
y_c2_train = np.zeros((y_c3_train.shape[0],num_c2)).astype("float32")
y_c2_test = np.zeros((y_c3_test.shape[0],num_c2)).astype("float32")
y_c2_val = np.zeros((y_c3_val.shape[0],num_c2)).astype("float32")
```

In [74]:
```
for i in range(y_c2_train.shape[0]):
  y_c2_train[i][c3_to_c2[np.argmax(y_c3_train[i])]] = 1.0

for i in range(y_c2_test.shape[0]):
  y_c2_test[i][c3_to_c2[np.argmax(y_c3_test[i])]] = 1.0

for i in range(y_c2_val.shape[0]):
  y_c2_val[i][c3_to_c2[np.argmax(y_c3_val[i])]] = 1.0
```

In [75]:
```
y_c1_train = np.zeros((y_c2_train.shape[0],num_c1)).astype("float32")
y_c1_test = np.zeros((y_c2_test.shape[0],num_c1)).astype("float32")
y_c1_val = np.zeros((y_c2_val.shape[0],num_c1)).astype("float32")
```

In [76]:
```
for i in range(y_c1_train.shape[0]):
  y_c1_train[i][c2_to_c1[np.argmax(y_c2_train[i])]] = 1.0

for i in range(y_c1_test.shape[0]):
  y_c1_test[i][c2_to_c1[np.argmax(y_c2_test[i])]] = 1.0

for i in range(y_c1_val.shape[0]):
  y_c1_val[i][c2_to_c1[np.argmax(y_c2_val[i])]] = 1.0
```

In [77]:
```
import keras
from keras.models import Model
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Input
from keras.layers import BatchNormalization
from keras.initializers import he_normal
from keras import optimizers
from keras.callbacks import LearningRateScheduler, TensorBoard
from keras.utils.data_utils import get_file
from keras import backend as K
```

In [ ]:
```

```

In [78]:
```
def scheduler(epoch):
  learning_rate_init = 0.001
  if epoch > 55:
    learning_rate_init = 0.0002
  if epoch > 70:
    learning_rate_init = 0.00005
  return learning_rate_init
```

In [79]:
```
class LossWeightsModifier(keras.callbacks.Callback):
  def __init__(self, alpha, beta, gamma):
```

```
    self.alpha = alpha
    self.beta = beta
    self.gamma = gamma

  def on_epoch_end(self, epoch, logs={}):
    if epoch == 13:
      K.set_value(self.alpha, 0.1)
      K.set_value(self.beta, 0.8)
      K.set_value(self.gamma, 0.1)

    if epoch == 23:
      K.set_value(self.alpha, 0.1)
      K.set_value(self.beta, 0.2)
      K.set_value(self.gamma, 0.7)

    if epoch == 33:
      K.set_value(self.alpha, 0)
      K.set_value(self.beta, 0)
      K.set_value(self.gamma, 1)
```

In [80]:
```
input_shape = (224,224,3)
```

In [81]:
```
#--- coarse 1 classes ---
coarse1_classes = num_c1 #21
#--- coarse 2 classes ---
coarse2_classes = num_c2 #121
#--- fine classes ---
num_classes  = num_c3 #421
```

In [82]:
```
alpha = K.variable(value=0.98, dtype="float32", name="alpha") # A1 in paper
beta = K.variable(value=0.01, dtype="float32", name="beta") # A2 in paper
gamma = K.variable(value=0.01, dtype="float32", name="gamma") # A3 in paper

img_input = Input(shape=input_shape, name='input')
```

```
2022-02-03 16:07:49.849136: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is opt
imized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-c
ritical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-02-03 16:07:50.463287: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:local
host/replica:0/task:0/device:GPU:0 with 38416 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-40GB, pci bus i
d: 0000:00:04.0, compute capability: 8.0
```

In [83]:
```
#--- block 1 ---
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(img_input)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)
```

In [84]:
```
#--- block 2 ---
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
```

In [85]:
```
#--- coarse 1 branch ---
c_1_bch = Flatten(name='c1_flatten')(x)
c_1_bch = Dense(256, activation='relu', name='c1_1')(c_1_bch)
c_1_bch = BatchNormalization()(c_1_bch)
c_1_bch = Dropout(0.5)(c_1_bch)
c_1_bch = Dense(256, activation='relu', name='c1_2')(c_1_bch)
c_1_bch = BatchNormalization()(c_1_bch)
c_1_bch = Dropout(0.5)(c_1_bch)
c_1_pred = Dense(coarse1_classes, activation='softmax', name='c1_predictions')(c_1_bch)
```

In [86]:
```
#--- block 3 ---
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
x = BatchNormalization()(x)
```

```
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
x = BatchNormalization()(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
```

In [87]:
```
#--- coarse 2 branch ---
c_2_bch = Flatten(name='c2_flatten')(x)
c_2_bch = Dense(1024, activation='relu', name='c2_1')(c_2_bch)
c_2_bch = BatchNormalization()(c_2_bch)
c_2_bch = Dropout(0.5)(c_2_bch)
c_2_bch = Dense(1024, activation='relu', name='c2_2')(c_2_bch)
c_2_bch = BatchNormalization()(c_2_bch)
c_2_bch = Dropout(0.5)(c_2_bch)
c_2_pred = Dense(coarse2_classes, activation='softmax', name='c2_predictions')(c_2_bch)
```

In [88]:
```
#--- block 4 ---
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
x = BatchNormalization()(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)
```

In [89]:
```
#--- block 5 ---
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
x = BatchNormalization()(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
x = BatchNormalization()(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
x = BatchNormalization()(x)
```

In [90]:
```
#--- fine block ---
x = Flatten(name='flatten')(x)
x = Dense(4096, activation='relu', name='c3_1')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(4096, activation='relu', name='c3_2')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
fine_pred = Dense(num_c3, activation='softmax', name='c3_predictions')(x)
```

In [91]:
```
model = Model(inputs = img_input, outputs = [c_1_pred, c_2_pred, fine_pred])
```

In [92]:
```
sgd = tf.keras.optimizers.SGD(lr=0.003, momentum=0.9, nesterov=True)
nadam = tf.keras.optimizers.Nadam(
    learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07)
adadelta = tf.keras.optimizers.Adadelta(
    learning_rate=0.001, rho=0.95, epsilon=1e-07)
```

```
/software-msca/ivy2/env/2021.05/ML2/lib/python3.9/site-packages/keras/optimizer_v2/gradient_descent.py:102: Us
erWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

In [93]:
```
model.compile(loss='categorical_crossentropy',
              optimizer=adadelta,
              loss_weights=[alpha, beta, gamma],
              # optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

In [94]:
```
log_filepath = '/cloud/msca-gcp/chuyucc/BCNN/Threelayers/log'
```

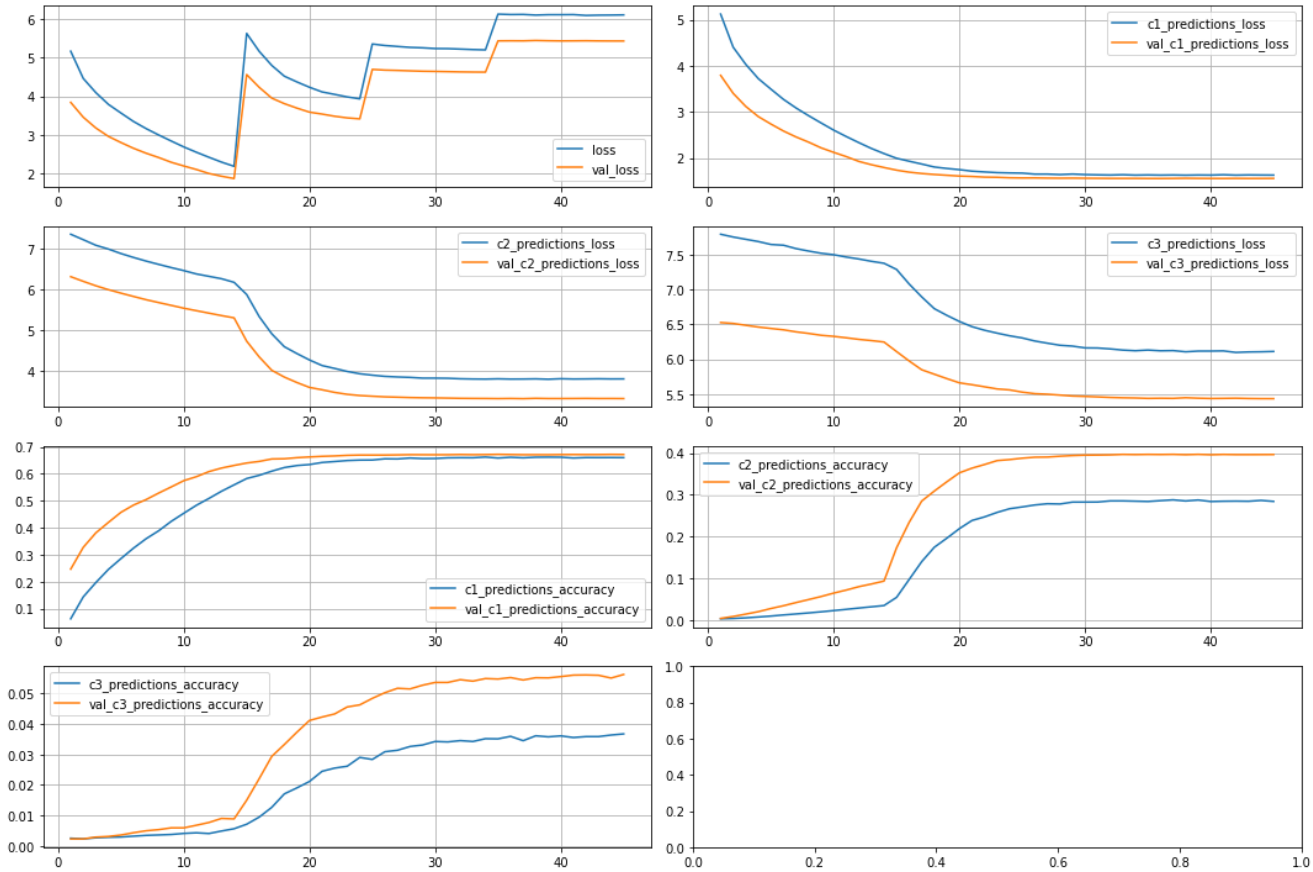
In [95]:
```
tb_cb = TensorBoard(log_dir=log_filepath, histogram_freq=0)
```

In [96]:
```
import numpy as np
from tensorflow import keras
from matplotlib import pyplot as plt
from IPython.display import clear_output

class PlotLearning(keras.callbacks.Callback):
    """
    Callback to plot the learning curves of the model during training.
    """
    def on_train_begin(self, logs={}):
        self.metrics = {}
        for metric in logs:
            self.metrics[metric] = []


    def on_epoch_end(self, epoch, logs={}):
        # Storing metrics
        for metric in logs:
            if metric in self.metrics:
                self.metrics[metric].append(logs.get(metric))
            else:
                self.metrics[metric] = [logs.get(metric)]

        # Plotting
        metrics = [x for x in logs if 'val' not in x]

        f, axs = plt.subplots(4, int(len(metrics)/3), figsize=(15,10))
        clear_output(wait=True)

        for i, metric in enumerate(metrics):
            axs[i//2,i%2].plot(range(1, epoch + 2),
                        self.metrics[metric],
                        label=metric)
            if logs['val_' + metric]:
                axs[i//2,i%2].plot(range(1, epoch + 2),
                            self.metrics['val_' + metric],
                            label='val_' + metric)

            axs[i//2,i%2].legend()
            axs[i//2,i%2].grid()

        path = '/home/chuyucc/BCNN/Plot/'
        plt.savefig(path+f'accuracy_superclass_epoch{i}'+'.png')

        plt.tight_layout()
        plt.show()
```

In [97]:
```
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=3, factor=0.5, min_lr=1e-6, verbose=1)
change_lr = LearningRateScheduler(scheduler)
change_lw = LossWeightsModifier(alpha, beta,gamma)
cbks = [PlotLearning(), rlrop, tb_cb, change_lw]
```

In [98]:
```
# try reducing the batch size
```

In [ ]:
```
model.fit(X_train, [y_c1_train, y_c2_train, y_c3_train],
          batch_size=64,
          epochs=60,
          verbose=1,
          callbacks=cbks,
          validation_data=(X_val, [y_c1_val, y_c2_val, y_c3_val]))
```




```
1172/1172 [==============================] - 210s 179ms/step - loss: 6.1117 - c1_predictions_loss: 1.6270 - c2
_predictions_loss: 3.7919 - c3_predictions_loss: 6.1117 - c1_predictions_accuracy: 0.6604 - c2_predictions_acc
uracy: 0.2839 - c3_predictions_accuracy: 0.0368 - val_loss: 5.4348 - val_c1_predictions_loss: 1.5549 - val_c2_
predictions_loss: 3.3052 - val_c3_predictions_loss: 5.4348 - val_c1_predictions_accuracy: 0.6720 - val_c2_pred
ictions_accuracy: 0.3962 - val_c3_predictions_accuracy: 0.0563 - lr: 1.0000e-06
Epoch 46/60
 230/1172 [====>.........................] - ETA: 2:38 - loss: 6.1065 - c1_predictions_loss: 1.6249 - c2_predi
ctions_loss: 3.8039 - c3_predictions_loss: 6.1065 - c1_predictions_accuracy: 0.6632 - c2_predictions_accuracy:
0.2840 - c3_predictions_accuracy: 0.0352
```

In [ ]:
```
score = model.evaluate(X_test, [y_c1_test, y_c2_test, y_c3_test], verbose=0)
print('score is: ', score)
```

In [ ]:
```

```