AIML Capstone Project: CV - Car Detection

Interim Report

Arun Sharma (Mentor)
Aravindan T
Jimmy R Fernandes
Kunal Pandya
Priya Balachandar
Rajeshkanna Ala

Table of Contents

Summary and Problem Statement

Computer vision can be used to automate supervision and generate action appropriate action trigger if the event is predicted from the image of interest. For example a car moving on the road can be easily identified by a camera as make of the car, type, colour, number plates etc.

DATA DESCRIPTION*:*

The dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split.

Classes are typically at the level of Make, Model, Year, e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.

- Train Images: Consists of real images of cars as per the make and year of the car
- Test Images: Consists of real images of cars as per the make and year of the car.
- Train Annotation: Consists of bounding box region for training images.
- Test Annotation: Consists of bounding box region for testing images.

Dataset:
https://drive.google.com/drive/folders/1y6JWx2CpsOuka00uePe72jNgr7F9sK45?usp=sharing,

PROJECT OBJECTIVE*:* Design a DL based car identification model

EDA and Pre-processing

Step #1: **Importing the dataset in dataframes**

```
#Different car labels
car_names = pd.read_csv( 'Car names and make.csv', header=None, names = ['CarLabe
l'] )

#Train data
train_data = pd.read_csv( 'Annotations/Train Annotations.csv', skiprows=1, names
= ['ImageName', 'X1', 'Y1', 'X2' , 'Y2', 'Class'] )

#Test data
test_data = pd.read_csv( 'Annotations/Test Annotation.csv' , skiprows=1, names =
['ImageName', 'X1', 'Y1', 'X2' , 'Y2', 'Class'])
```

Display few records

```
                    CarLabel
0  AM General Hummer SUV 2000
1         Acura RL Sedan 2012
2         Acura TL Sedan 2012
3       Acura TL Type-S 2008
4       Acura TSX Sedan 2012

   ImageName   X1   Y1    X2    Y2  Class
0  00001.jpg   39  116   569   375     14
1  00002.jpg   36  116   868   587      3
2  00003.jpg   85  109   601   381     91
3  00004.jpg  621  393  1484  1096    134
4  00005.jpg   14   36   133    99    106

   ImageName   X1   Y1   X2   Y2  Class
0  00001.jpg   30   52  246  147    181
1  00002.jpg  100   19  576  203    103
2  00003.jpg   51  105  968  659    145
3  00004.jpg   67   84  581  407    187

4  00005.jpg  140  151  593  339    185
```

Problem faced: While trying to read few images using cv2, we were getting return type as None. After troubleshooting, we found that this is due to folder name of images. Some of folder names have a '/' in it. Therefore, we decided to update such names as part of pre-processing step.

Step #2: **Find class name with '/' and update**

```
for i in range(len(car_names)) :
  if '/' in car_names.loc[i,"CarLabel"]:
    print(car_names.loc[i,"CarLabel"])
    print(i)
```

```
Ram C/V Cargo Van Minivan 2012
173
```

Thus, there was only 1 class with '/' in it's name.

```
#Replace '/' with '-' in the name
car_names.loc[173,'CarLabel'] = 'Ram C-V Cargo Van Minivan 2012'
```

Step #3: Map training and test images to corresponding classes and annotations.

```
car_names['Class'] = car_names.index + 1
car_train_df = pd.merge(train_data, car_names, how = 'left', left_on='Class', rig
ht_on='Class' )
car_train_df.head()
```

```
car_test_df = pd.merge(test_data, car_names, how = 'left', left_on='Class', right
_on='Class' )
car_test_df.head()
```

Display few records

	ImageName	X1	Y1	X2	Y2	Class	CarLabel
0	00001.jpg	30	52	246	147	181	Suzuki Aerio Sedan 2007
1	00002.jpg	100	19	576	203	103	Ferrari 458 Italia Convertible 2012
2	00003.jpg	51	105	968	659	145	Jeep Patriot SUV 2012
3	00004.jpg	67	84	581	407	187	Toyota Camry Sedan 2012
4	00005.jpg	140	151	593	339	185	Tesla Model S Sedan 2012

Step #4: **Exploratory Data Analysis**

- **For each car image label, separate year, make, model and body**

```
import nltk
nltk.download('punkt')

#Different body types
car_body_type = ["suv","sedan","type-s","type-
r","convertible","coupe","wagon","hatchback","cab","supercab","van","minivan"]
car_body_type = [item.lower() for item in car_body_type]

#Different car make
car_make = ['am', 'general', 'acura', 'aston', 'martin', 'audi', 'bmw', 'bentley',
'bugatti', 'buick', 'cadillac', 'chevrolet', 'chrysler', 'daewoo',
            'dodge', 'eagle', 'fiat', 'ferrari', 'fisker', 'ford', 'gmc', 'geo', 'h
onda', 'hyundai', 'infiniti', 'isuzu',
            'jaguar', 'jeep', 'lamborghini', 'land', 'rover', 'lincoln', 'mini', 'c
ooper', 'maybach', 'mazda', 'mclaren', 'mercedes-benz',
            'mitsubishi', 'nissan', 'plymouth', 'porsche', 'ram', 'rolls-
royce', 'scion', 'spyker', 'suzuki', 'tesla', 'toyota', 'volkswagen', 'volvo', 'sma
rt']
car_make = [item.lower() for item in car_make]

#Define dataframe to store results
eda_df = car_test_df.copy()
eda_df["year"] = None
eda_df["make"] = None
eda_df["model"] = None
eda_df["body"] = None

pattern="[0-9][0-9][0-9][0-9]"
for col in eda_df.columns:
    if col == 'CarLabel':
        for index, row in eda_df.iterrows():
            wordsl = word_tokenize(row[col].lower())
            print(index, wordsl)

            if len(wordsl)>1:
                year = re.findall(pattern,wordsl[len(wordsl)-1])#row[0])
                if year:
                    eda_df.loc[index, 'year'] = year[0]
                body = list(set(wordsl).intersection(car_body_type))
                if body:
                    eda_df.loc[index, 'body'] = body[0]
                make = list(set(wordsl).intersection(car_make))
                if make:
```

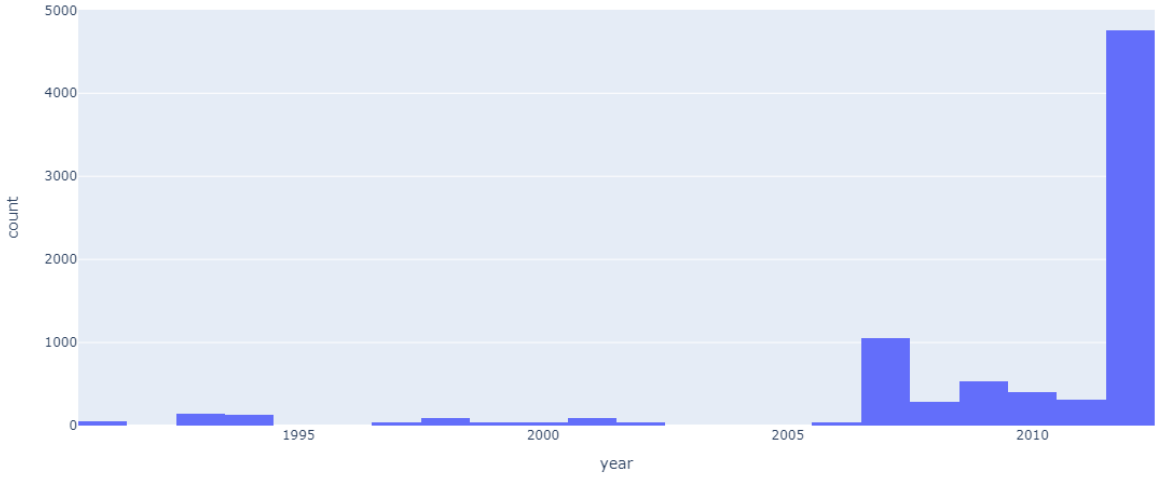
```
            eda_df.loc[index, 'make'] = ' '.join(make)
    iden=year + body + make
    print(iden)
    model = list(set(wordsl).difference(iden))
    print(model)
    if model:
            eda_df.loc[index, 'model'] = ' '.join(model)
```

Display Few records:

	ImageName	X1	Y1	X2	Y2	Class	CarLabel	year	make	model	body
0	00001.jpg	30	52	246	147	181	Suzuki Aerio Sedan 2007	2007	suzuki	aerio	sedan
1	00002.jpg	100	19	576	203	103	Ferrari 458 Italia Convertible 2012	2012	ferrari	458 italia	convertible
2	00003.jpg	51	105	968	659	145	Jeep Patriot SUV 2012	2012	jeep	patriot	suv
3	00004.jpg	67	84	581	407	187	Toyota Camry Sedan 2012	2012	toyota	camry	sedan
4	00005.jpg	140	151	593	339	185	Tesla Model S Sedan 2012	2012	tesla	s model	sedan

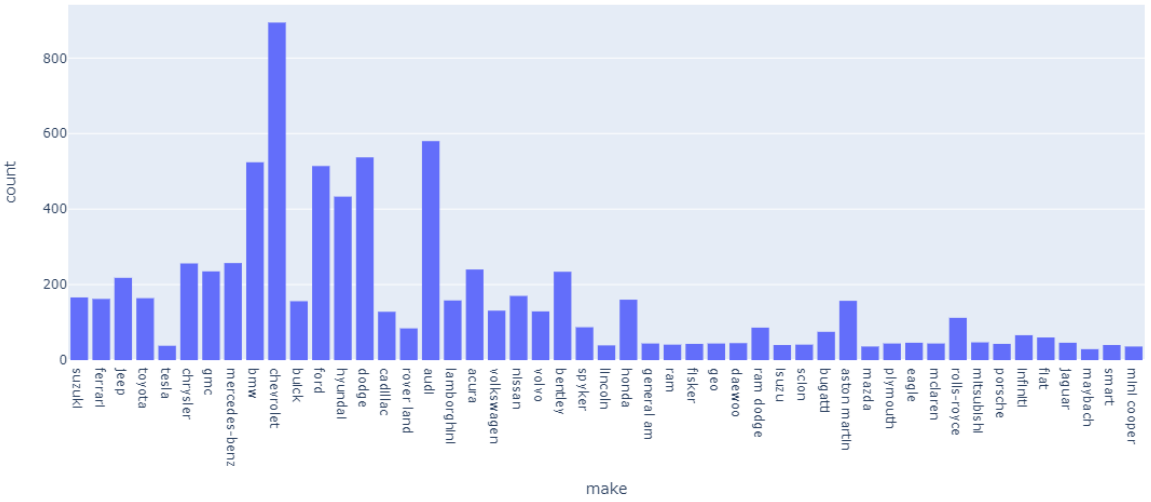
- **Display bar charts for different columns**

  - Count of different cars by year



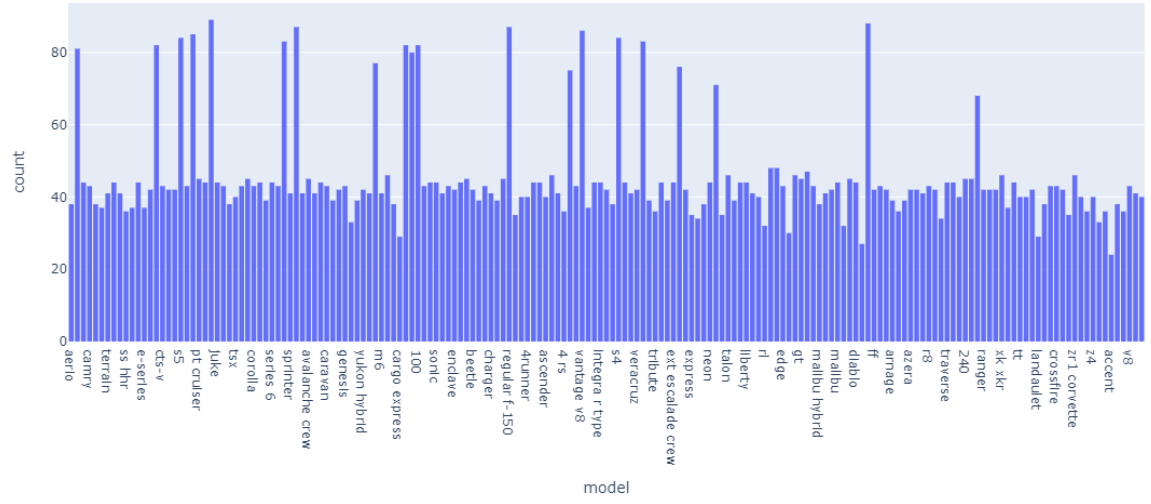
As seen, most of cars are of make year between 2007 – 2012

➢ Count of different cars by make



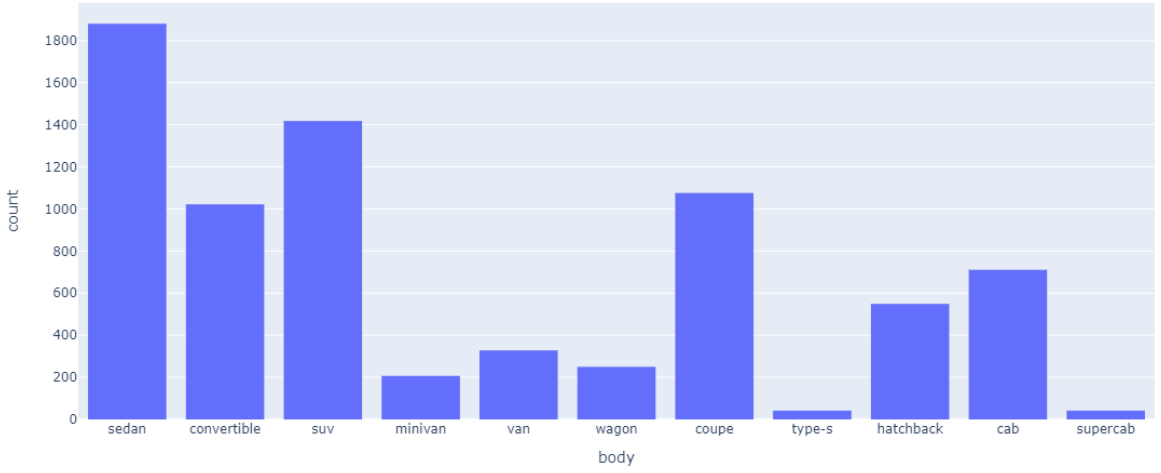
As seen, most of cars are of make Chevrolet, audi, bmw and dodge. ‘Mini Cooper’, ‘Smart’ and ‘Jaguar’ are less represented. Therefore, an effective model would be the one which can identify cars belonging to these classes.

➢ Count of different cars by model



As seen, cars evenly belong to different models, with some models having more number of cars.

➢ Count of cars by different body types

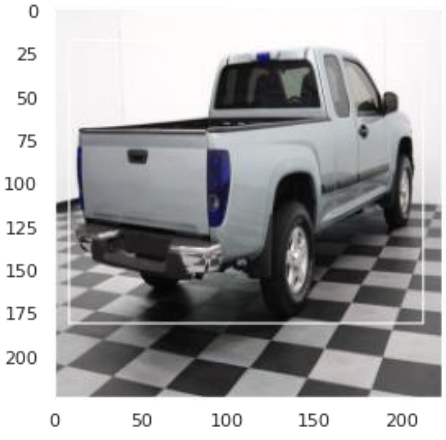
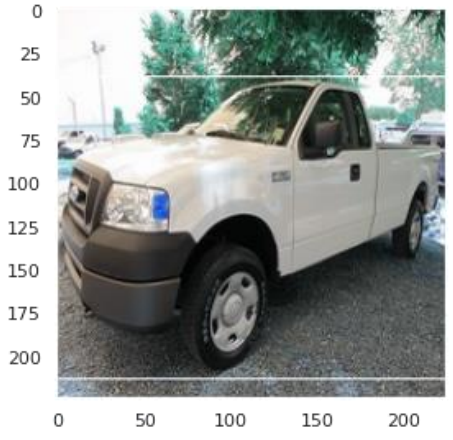
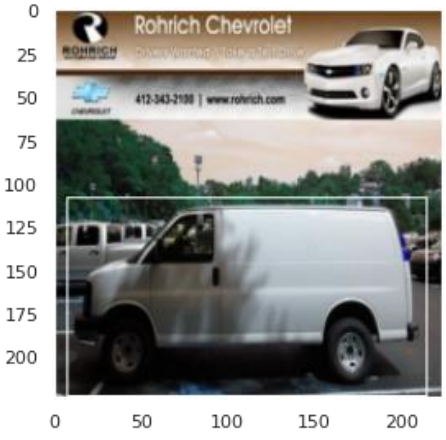


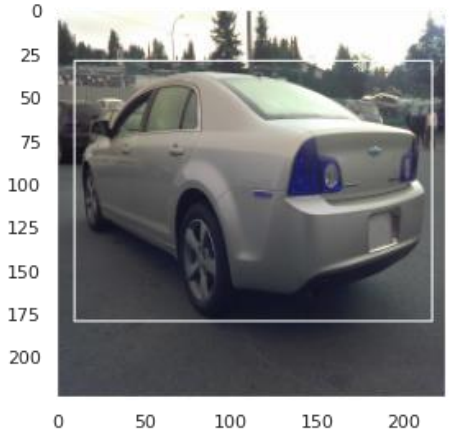
As seen, most of cars are of type Sedan

- **Display images with bounding box.**

```
IMAGE_SIZE = 224
IMAGE_HEIGHT = IMAGE_SIZE
IMAGE_WIDTH = IMAGE_SIZE

HEIGHT_CELLS = 28
WIDTH_CELLS = 28


print ( 'Generating bounding boxes images for Eg Train Data')

i = 1
      plt.figure(figsize=(20,20))
      for no in [0 , 6, 67, 89 , 99, 340 ]:
```

```
    eg_car  = car_train_df.iloc[ no ]

    path = 'Car Images/Train Images/{0}/{1}'.format( eg_car['CarLabel'], eg
_car['ImageName'] )
    img = cv2.imread( path )
    img_shape = img.shape
    img = cv2.resize(img, dsize = (IMAGE_SIZE, IMAGE_SIZE), interpolation=c
v2.INTER_AREA)


    x1 = int(eg_car['X1']   * IMAGE_SIZE  / img_shape[1] -
3 )           # Normalize bounding box by image size
    y1 = int(eg_car['Y1']   * IMAGE_SIZE / img_shape[0] - 3 )        # Norm
alize bounding box by image size
    x2 = int(eg_car['X2']   * IMAGE_SIZE / img_shape[1] + 3)           # No
rmalize bounding box by image size
    y2 = int(eg_car['Y2']   * IMAGE_SIZE / img_shape[0] + 3 )          # No
rmalize bounding box by image size


    cv2.rectangle(img, (x1, y1), (x2, y2), (255,255,255) )

    i +=1
    plt.subplot(4,2,i+1)
    plt.grid(False)
    plt.imshow(img);
```



- **Save training images with bounding box.**

  We will extract bounding boxes and then save those as images

```
for i in range(len(car_train_df)):
  eg_car  = car_train_df.iloc[i]
  source_path = 'Car Images/Train Images/{0}/{1}'.format( eg_car['CarLabel'], eg_
car['ImageName'] )
```

```
  dest_path = 'Car Images/Train Images Annoted/{0}/{1}'.format( eg_car['CarLabel'
], eg_car['ImageName'] )

  image = cv2.imread(source_path)
  if image is None:
    print(source_path)

  x1 = int(eg_car['X1'])
  y1 = int(eg_car['Y1'])
  x2 = int(eg_car['X2'])
  y2 = int(eg_car['Y2'])

  im2 = image[y1:y2,x1:x2]
  im2 = cv2.resize(im2, (IMAGE_SIZE, IMAGE_SIZE))

  destdirname = 'Car Images/Train Images Annoted/{0}'.format( eg_car['CarLabel'])
  destfilename= eg_car['ImageName']

  if not os.path.exists(destdirname):
    os.mkdir(destdirname)

  cv2.imwrite(os.path.join(destdirname, destfilename), im2)
```

Display few cropped training images

```
eg_car  = car_train_df.iloc[8]
path = 'Car Images/Train Images Annoted/{0}/{1}'.format( eg_car['CarLabel'], eg_c
ar['ImageName'] )
img = cv2.imread( path )
plt.grid(False)
plt.imshow(img)
```

```
eg_car  = car_train_df.iloc[16]
path = 'Car Images/Train Images Annoted/{0}/{1}'.format( eg_car['CarLabel'], eg_c
ar['ImageName'] )
img = cv2.imread( path )
plt.grid(False)
plt.imshow(img)
```

Do the same for test images.

- **Load the cropped train & test images using ImageDataGenerator**

```
train_path = 'Car Images/Train Images Annoted'

test_path = 'Car Images/Test Images Annoted'

BATCH_SIZE = 32
IMG_SIZE = (224, 224)

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=IMG_SIZE,
```

```
    batch_size=BATCH_SIZE,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical')
```

```
Found 8144 images belonging to 197 classes.
Found 8041 images belonging to 197 classes.
```

Train Different Models

For this problem, we tried following models:

1) CNN Classifier
2) ResNet50
3) VGG16
4) ResNet with custom FC layer
5) InceptionResNetV2

A. CNN Classifier

1. Create the model

```
# Initialising the CNN classifier
classifier = Sequential()

INPUT_SIZE = (224, 224, 3)

# Add a Convolution layer with 32 kernels of 3X3 shape with activation function R
eLU
classifier.add(Conv2D(32, (3, 3), input_shape = INPUT_SIZE, activation = 'relu',
padding = 'same'))

# Add a Max Pooling layer of size 2X2
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Add another Convolution layer with 32 kernels of 3X3 shape with activation func
tion ReLU
classifier.add(Conv2D(32, (3, 3), activation = 'relu', padding = 'same'))

# Adding another pooling layer
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Add another Convolution layer with 32 kernels of 3X3 shape with activation func
tion ReLU
classifier.add(Conv2D(32, (3, 3), activation = 'relu', padding = 'same'))

# Adding another pooling layer
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Flattening the layer before fully connected layers
classifier.add(Flatten())
```

```
# Adding a fully connected layer with 512 neurons
classifier.add(Dense(units = 512, activation = 'relu'))

# Adding dropout with probability 0.5
classifier.add(Dropout(0.5))


# Adding a fully connected layer with 128 neurons
classifier.add(Dense(units = 128, activation = 'relu'))


# The final output layer with output size 197 classes for  the categorical classi
fcation
classifier.add(Dense(units = 197, activation = 'softmax'))
```

2. Summary

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 32)      9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 32)        9248
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 28, 28, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               12845568
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_5 (Dense)              (None, 197)               25413
=================================================================
Total params: 12,956,037
Trainable params: 12,956,037
Non-trainable params: 0
```

---

3. Define Optimizer

```
opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=
0.001, amsgrad=False)
classifier.compile(optimizer = opt, loss = 'categorical_crossentropy', metrics
 = ['accuracy'])
```

4. Training [Forward pass and Backpropagation]

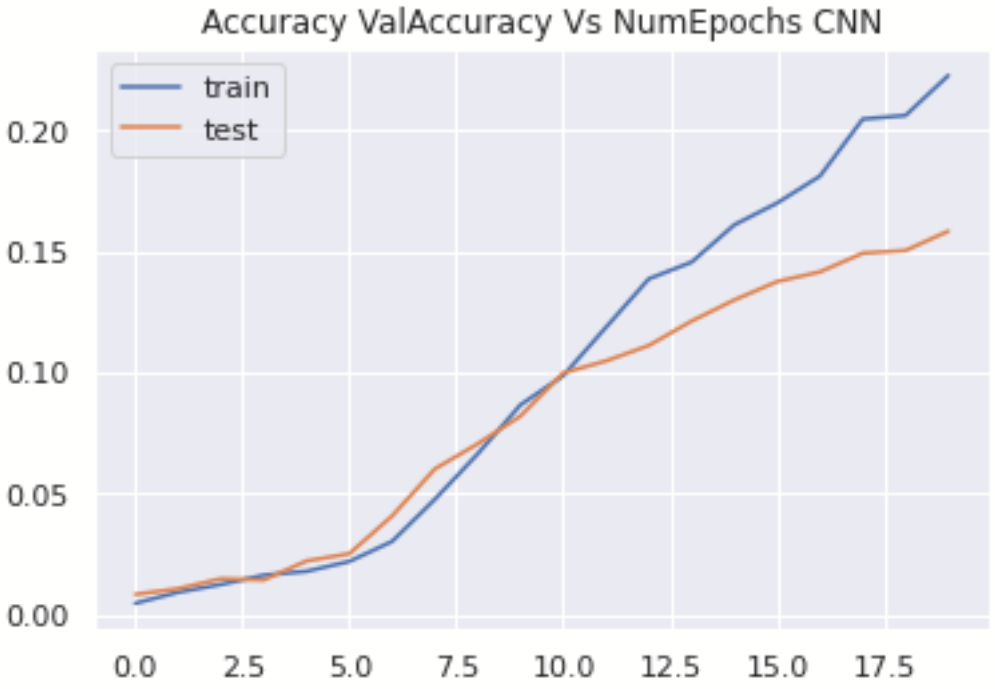
```
#Early stopping
early = EarlyStopping(monitor='val_accuracy',min_delta=0,patience=40,verbose=1,mo
de='auto')

# There are 3823 training images and 500 test images in total
hist_CNNClassifier = classifier.fit_generator(train_generator,
                     steps_per_epoch = int(train_generator.samples/BATCH_SIZE),
                     epochs = 20,
                     validation_data = validation_generator,
                    validation_steps = int(validation_generator.samples/BATCH_SIZE
                   ),
                     callbacks = [early])
```

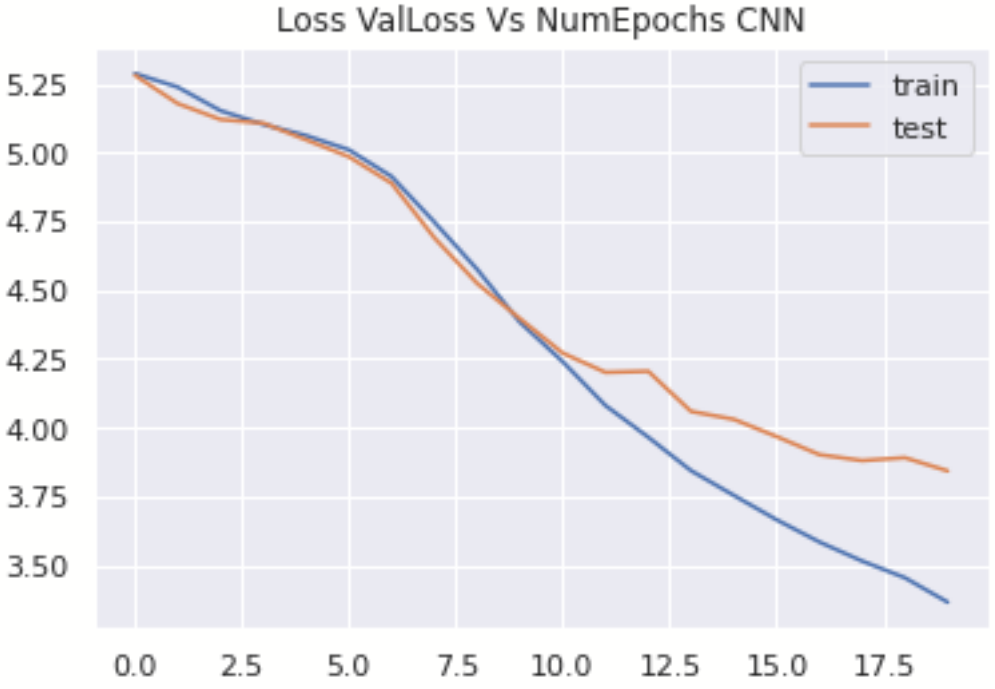
```
Epoch 1/20
254/254 [==============================] - 2523s 10s/step - loss: 5.2871 -
accuracy: 0.0048 - val_loss: 5.2809 - val_accuracy: 0.0085
Epoch 2/20
254/254 [==============================] - 188s 739ms/step - loss: 5.2378 -
accuracy: 0.0094 - val_loss: 5.1779 - val_accuracy: 0.0108
Epoch 3/20
254/254 [==============================] - 187s 736ms/step - loss: 5.1521 -
accuracy: 0.0126 - val_loss: 5.1202 - val_accuracy: 0.0149
Epoch 4/20
254/254 [==============================] - 185s 728ms/step - loss: 5.1024 -
accuracy: 0.0164 - val_loss: 5.1068 - val_accuracy: 0.0144
Epoch 5/20
254/254 [==============================] - 184s 724ms/step - loss: 5.0617 -
accuracy: 0.0180 - val_loss: 5.0464 - val_accuracy: 0.0223
Epoch 6/20
254/254 [==============================] - 182s 719ms/step - loss: 5.0102 -
accuracy: 0.0221 - val_loss: 4.9850 - val_accuracy: 0.0253
Epoch 7/20
254/254 [==============================] - 184s 726ms/step - loss: 4.9130 -
accuracy: 0.0303 - val_loss: 4.8883 - val_accuracy: 0.0408
Epoch 8/20
254/254 [==============================] - 184s 724ms/step - loss: 4.7473 -
accuracy: 0.0477 - val_loss: 4.6899 - val_accuracy: 0.0603
Epoch 9/20
254/254 [==============================] - 184s 725ms/step - loss: 4.5756 -
accuracy: 0.0664 - val_loss: 4.5243 - val_accuracy: 0.0706
Epoch 10/20
254/254 [==============================] - 184s 725ms/step - loss: 4.3834 -
accuracy: 0.0867 - val_loss: 4.3964 - val_accuracy: 0.0820
Epoch 11/20
```

```
254/254 [==============================] - 184s 725ms/step - loss: 4.2401 -
accuracy: 0.0987 - val_loss: 4.2711 - val_accuracy: 0.0999
Epoch 12/20
254/254 [==============================] - 183s 723ms/step - loss: 4.0806 -
accuracy: 0.1187 - val_loss: 4.2011 - val_accuracy: 0.1048
Epoch 13/20
254/254 [==============================] - 183s 722ms/step - loss: 3.9654 -
accuracy: 0.1387 - val_loss: 4.2050 - val_accuracy: 0.1112
Epoch 14/20
254/254 [==============================] - 184s 727ms/step - loss: 3.8438 -
accuracy: 0.1456 - val_loss: 4.0591 - val_accuracy: 0.1213
Epoch 15/20
254/254 [==============================] - 185s 728ms/step - loss: 3.7540 -
accuracy: 0.1610 - val_loss: 4.0304 - val_accuracy: 0.1300
Epoch 16/20
254/254 [==============================] - 184s 726ms/step - loss: 3.6654 -
accuracy: 0.1700 - val_loss: 3.9678 - val_accuracy: 0.1376
Epoch 17/20
254/254 [==============================] - 183s 722ms/step - loss: 3.5846 -
accuracy: 0.1811 - val_loss: 3.9020 - val_accuracy: 0.1416
Epoch 18/20
254/254 [==============================] - 183s 723ms/step - loss: 3.5153 -
accuracy: 0.2046 - val_loss: 3.8804 - val_accuracy: 0.1493
Epoch 19/20
254/254 [==============================] - 184s 725ms/step - loss: 3.4546 -
accuracy: 0.2061 - val_loss: 3.8908 - val_accuracy: 0.1504
Epoch 20/20
      254/254 [==============================] - 184s 725ms/step - loss: 3.3658 -
      accuracy: 0.2226 - val_loss: 3.8429 - val_accuracy: 0.1584
```

5. Accuracy and Loss for Training and Validation

```
train_loss = hist_CNNClassifier.history['loss']
val_loss   = hist_CNNClassifier.history['val_loss']

xc = hist_CNNClassifier.epoch
plt.title("Accuracy ValAccuracy Vs NumEpochs CNN")
plt.plot(xc,hist_CNNClassifier.history['accuracy'], label='train')
plt.plot(xc,hist_CNNClassifier.history['val_accuracy'], label='test')
plt.legend()
plt.show()

plt.figure()
plt.title("Loss ValLoss Vs NumEpochs CNN")
plt.plot(xc, train_loss,label='train')
plt.plot(xc, val_loss,label='test')
plt.legend()
plt.show
```





Graph shows that model tends to increase both training and validation accuracy, and decrease training and validation loss with each epoch.

6. Evaluation

```
train_acc = classifier.evaluate_generator(train_generator,steps = int(train_ge
nerator.samples/BATCH_SIZE))
val_acc = classifier.evaluate_generator(validation_generator, steps = int(vali
dation_generator.samples/BATCH_SIZE))
```

```
print(train_acc[1])
print(val_acc[1])
```

```
0.3591289222240448
0.15861554443836212
```

Final evaluation shows that overall training accuracy is only around 36% and validation accuracy is around 16%. Therefore, this reveals both high bias and high variance issues.

7. Store result in a dataframe for final comparison of different models

```
#Store the Performance Matrix for each model in a dataframe for final comparis
on
resultsDf = pd.DataFrame({'Model':['CNN'], 'Train_Accuracy': train_acc[1],'Tes
t_Accuracy': val_acc[1],
                          })
resultsDf = resultsDf[['Model', 'Train_Accuracy','Test_Accuracy']]
resultsDf
```

	Model	Train_Accuracy	Test_Accuracy
0	CNN	0.359129	0.158616

8. Pickle the model for future use

```
classifier.save('./classifier.h5')

classifier.save_weights('./classifier_weights.h5')
```

B. ResNet50

1. Creating the model

```
resnet50 = resnet50
conv_model = resnet50.ResNet50(weights='imagenet', include_top=False, input_shape
=(224,224,3))
x = Flatten()(conv_model.layers[-1].output)
```

```
x = Dense(512, activation='relu')(x)
x = Dense(224, activation='relu')(x)
x = Dense(224, activation='relu')(x)
x = Dense(224, activation='relu')(x)
x = Dense(224, activation='relu')(x)
x = Dense(224, activation='sigmoid')(x)
x = Dense(224, activation='sigmoid')(x)
predictions = Dense(197, activation='softmax')(x)

full_model = Model(inputs=conv_model.input, outputs=predictions)
```

2. Summary of model

```
full_model.summary()

Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)      0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, 230, 230, 3)       0
_________________________________________________________________
conv1_conv (Conv2D)          (None, 112, 112, 64)      9472
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 64)      256
_________________________________________________________________
conv1_relu (Activation)      (None, 112, 112, 64)      0
_________________________________________________________________
pool1_pad (ZeroPadding2D)    (None, 114, 114, 64)      0
_________________________________________________________________
pool1_pool (MaxPooling2D)    (None, 56, 56, 64)        0
_________________________________________________________________
conv2_block1_1_conv (Conv2D) (None, 56, 56, 64)        4160
_________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)     256
conv2_block1_1_conv[0][0]
_________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)     0
conv2_block1_1_bn[0][0]
_________________________________________________________________
conv2_block1_2_conv (Conv2D) (None, 56, 56, 64)        36928
conv2_block1_1_relu[0][0]
```

```
__________________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block1_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)   0
conv2_block1_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 56, 56, 256)  16640       pool1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_3_conv (Conv2D)    (None, 56, 56, 256)  16640
conv2_block1_2_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block1_0_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block1_3_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_add (Add)          (None, 56, 56, 256)  0
conv2_block1_0_bn[0][0]

conv2_block1_3_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_out (Activation)   (None, 56, 56, 256)  0
conv2_block1_add[0][0]
__________________________________________________________________________________________________
conv2_block2_1_conv (Conv2D)    (None, 56, 56, 64)   16448
conv2_block1_out[0][0]
__________________________________________________________________________________________________
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block2_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_1_relu (Activation (None, 56, 56, 64)   0
conv2_block2_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_2_conv (Conv2D)    (None, 56, 56, 64)   36928
conv2_block2_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_2_relu (Activation (None, 56, 56, 64)   0
conv2_block2_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_3_conv (Conv2D)    (None, 56, 56, 256)  16640
conv2_block2_2_relu[0][0]
```

```
__________________________________________________________________________________________________
__________________
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block2_3_conv[0][0]
__________________________________________________________________________________________________
__________________
conv2_block2_add (Add)          (None, 56, 56, 256)  0
conv2_block1_out[0][0]

conv2_block2_3_bn[0][0]
__________________________________________________________________________________________________
__________________
conv2_block2_out (Activation)   (None, 56, 56, 256)  0
conv2_block2_add[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_1_conv (Conv2D)    (None, 56, 56, 64)   16448
conv2_block2_out[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_1_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block3_1_conv[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_1_relu (Activation (None, 56, 56, 64)   0
conv2_block3_1_bn[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_2_conv (Conv2D)    (None, 56, 56, 64)   36928
conv2_block3_1_relu[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_2_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block3_2_conv[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_2_relu (Activation (None, 56, 56, 64)   0
conv2_block3_2_bn[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_3_conv (Conv2D)    (None, 56, 56, 256)  16640
conv2_block3_2_relu[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_3_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block3_3_conv[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_add (Add)          (None, 56, 56, 256)  0
conv2_block2_out[0][0]

conv2_block3_3_bn[0][0]
__________________________________________________________________________________________________
__________________
conv2_block3_out (Activation)   (None, 56, 56, 256)  0
conv2_block3_add[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_1_conv (Conv2D)    (None, 28, 28, 128)  32896
conv2_block3_out[0][0]
```

```
__________________________________________________________________________________________________
__________________
conv3_block1_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block1_1_conv[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_1_relu (Activation (None, 28, 28, 128)  0
conv3_block1_1_bn[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block1_1_relu[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block1_2_conv[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_2_relu (Activation (None, 28, 28, 128)  0
conv3_block1_2_bn[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_0_conv (Conv2D)    (None, 28, 28, 512)  131584
conv2_block3_out[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block1_2_relu[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_0_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block1_0_conv[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block1_3_conv[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_add (Add)          (None, 28, 28, 512)  0
conv3_block1_0_bn[0][0]

conv3_block1_3_bn[0][0]
__________________________________________________________________________________________________
__________________
conv3_block1_out (Activation)   (None, 28, 28, 512)  0
conv3_block1_add[0][0]
__________________________________________________________________________________________________
__________________
conv3_block2_1_conv (Conv2D)    (None, 28, 28, 128)  65664
conv3_block1_out[0][0]
__________________________________________________________________________________________________
__________________
conv3_block2_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block2_1_conv[0][0]
__________________________________________________________________________________________________
__________________
conv3_block2_1_relu (Activation (None, 28, 28, 128)  0
conv3_block2_1_bn[0][0]
__________________________________________________________________________________________________
__________________
```

```
conv3_block2_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block2_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block2_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block2_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block2_2_relu (Activation (None, 28, 28, 128)  0
conv3_block2_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block2_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block2_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block2_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block2_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block2_add (Add)          (None, 28, 28, 512)  0
conv3_block1_out[0][0]

conv3_block2_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block2_out (Activation)   (None, 28, 28, 512)  0
conv3_block2_add[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_1_conv (Conv2D)    (None, 28, 28, 128)  65664
conv3_block2_out[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block3_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_1_relu (Activation (None, 28, 28, 128)  0
conv3_block3_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block3_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block3_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_2_relu (Activation (None, 28, 28, 128)  0
conv3_block3_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block3_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block3_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block3_3_conv[0][0]
```

```
__________________________________________________________________________________________________
conv3_block3_add (Add)          (None, 28, 28, 512)  0
conv3_block2_out[0][0]

conv3_block3_3_bn[0][0]
__________________________________________________________________________________________________
conv3_block3_out (Activation)   (None, 28, 28, 512)  0
conv3_block3_add[0][0]
__________________________________________________________________________________________________
conv3_block4_1_conv (Conv2D)    (None, 28, 28, 128)  65664
conv3_block3_out[0][0]
__________________________________________________________________________________________________
conv3_block4_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block4_1_conv[0][0]
__________________________________________________________________________________________________
conv3_block4_1_relu (Activation (None, 28, 28, 128)  0
conv3_block4_1_bn[0][0]
__________________________________________________________________________________________________
conv3_block4_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block4_1_relu[0][0]
__________________________________________________________________________________________________
conv3_block4_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block4_2_conv[0][0]
__________________________________________________________________________________________________
conv3_block4_2_relu (Activation (None, 28, 28, 128)  0
conv3_block4_2_bn[0][0]
__________________________________________________________________________________________________
conv3_block4_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block4_2_relu[0][0]
__________________________________________________________________________________________________
conv3_block4_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block4_3_conv[0][0]
__________________________________________________________________________________________________
conv3_block4_add (Add)          (None, 28, 28, 512)  0
conv3_block3_out[0][0]

conv3_block4_3_bn[0][0]
__________________________________________________________________________________________________
conv3_block4_out (Activation)   (None, 28, 28, 512)  0
conv3_block4_add[0][0]
__________________________________________________________________________________________________
conv4_block1_1_conv (Conv2D)    (None, 14, 14, 256)  131328
conv3_block4_out[0][0]
__________________________________________________________________________________________________
conv4_block1_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block1_1_conv[0][0]
```

```
__________________________________________________________________________________________________
____________________
conv4_block1_1_relu (Activation (None, 14, 14, 256)  0
conv4_block1_1_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block1_1_relu[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block1_2_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_2_relu (Activation (None, 14, 14, 256)  0
conv4_block1_2_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_0_conv (Conv2D)    (None, 14, 14, 1024) 525312
conv3_block4_out[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block1_2_relu[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_0_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block1_0_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block1_3_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_add (Add)          (None, 14, 14, 1024) 0
conv4_block1_0_bn[0][0]

conv4_block1_3_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_out (Activation)   (None, 14, 14, 1024) 0
conv4_block1_add[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block1_out[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block2_1_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_1_relu (Activation (None, 14, 14, 256)  0
conv4_block2_1_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block2_1_relu[0][0]
__________________________________________________________________________________________________
____________________
```

```
conv4_block2_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv4_block2_2_relu (Activation (None, 14, 14, 256)  0
conv4_block2_2_bn[0][0]
__________________________________________________________________________________________________
conv4_block2_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block2_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block2_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block2_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block2_add (Add)          (None, 14, 14, 1024) 0
conv4_block1_out[0][0]

conv4_block2_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block2_out (Activation)   (None, 14, 14, 1024) 0
conv4_block2_add[0][0]
__________________________________________________________________________________________________
conv4_block3_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block2_out[0][0]
__________________________________________________________________________________________________
conv4_block3_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block3_1_conv[0][0]
__________________________________________________________________________________________________
conv4_block3_1_relu (Activation (None, 14, 14, 256)  0
conv4_block3_1_bn[0][0]
__________________________________________________________________________________________________
conv4_block3_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block3_1_relu[0][0]
__________________________________________________________________________________________________
conv4_block3_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block3_2_conv[0][0]
__________________________________________________________________________________________________
conv4_block3_2_relu (Activation (None, 14, 14, 256)  0
conv4_block3_2_bn[0][0]
__________________________________________________________________________________________________
conv4_block3_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block3_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block3_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block3_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block3_add (Add)          (None, 14, 14, 1024) 0
conv4_block2_out[0][0]
```

```
conv4_block3_3_bn[0][0]
__________________________________________________________________________________________________
__________________
conv4_block3_out (Activation)   (None, 14, 14, 1024) 0
conv4_block3_add[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block3_out[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block4_1_conv[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_1_relu (Activation (None, 14, 14, 256)  0
conv4_block4_1_bn[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block4_1_relu[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block4_2_conv[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_2_relu (Activation (None, 14, 14, 256)  0
conv4_block4_2_bn[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block4_2_relu[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block4_3_conv[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_add (Add)          (None, 14, 14, 1024) 0
conv4_block3_out[0][0]

conv4_block4_3_bn[0][0]
__________________________________________________________________________________________________
__________________
conv4_block4_out (Activation)   (None, 14, 14, 1024) 0
conv4_block4_add[0][0]
__________________________________________________________________________________________________
__________________
conv4_block5_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block4_out[0][0]
__________________________________________________________________________________________________
__________________
conv4_block5_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block5_1_conv[0][0]
__________________________________________________________________________________________________
__________________
conv4_block5_1_relu (Activation (None, 14, 14, 256)  0
conv4_block5_1_bn[0][0]
```

```
____________________________________________________________________________________________
____________________
conv4_block5_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block5_1_relu[0][0]
____________________________________________________________________________________________
____________________
conv4_block5_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block5_2_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block5_2_relu (Activation (None, 14, 14, 256)  0
conv4_block5_2_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block5_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block5_2_relu[0][0]
____________________________________________________________________________________________
____________________
conv4_block5_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block5_3_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block5_add (Add)          (None, 14, 14, 1024) 0
conv4_block4_out[0][0]

conv4_block5_3_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block5_out (Activation)   (None, 14, 14, 1024) 0
conv4_block5_add[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block5_out[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block6_1_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_1_relu (Activation (None, 14, 14, 256)  0
conv4_block6_1_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block6_1_relu[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block6_2_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_2_relu (Activation (None, 14, 14, 256)  0
conv4_block6_2_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block6_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block6_2_relu[0][0]
____________________________________________________________________________________________
____________________
```

```
conv4_block6_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block6_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block6_add (Add)          (None, 14, 14, 1024) 0
conv4_block5_out[0][0]

conv4_block6_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block6_out (Activation)   (None, 14, 14, 1024) 0
conv4_block6_add[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_1_conv (Conv2D)    (None, 7, 7, 512)    524800
conv4_block6_out[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block1_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_1_relu (Activation (None, 7, 7, 512)    0
conv5_block1_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_2_conv (Conv2D)    (None, 7, 7, 512)    2359808
conv5_block1_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block1_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_2_relu (Activation (None, 7, 7, 512)    0
conv5_block1_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_0_conv (Conv2D)    (None, 7, 7, 2048)   2099200
conv4_block6_out[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624
conv5_block1_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_0_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block1_0_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_3_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block1_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block1_add (Add)          (None, 7, 7, 2048)   0
conv5_block1_0_bn[0][0]

conv5_block1_3_bn[0][0]
__________________________________________________________________________________________
__________________
```

```
conv5_block1_out (Activation)   (None, 7, 7, 2048)   0
conv5_block1_add[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)    1049088
conv5_block1_out[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block2_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_1_relu (Activation (None, 7, 7, 512)    0
conv5_block2_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)    2359808
conv5_block2_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block2_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_2_relu (Activation (None, 7, 7, 512)    0
conv5_block2_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624
conv5_block2_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block2_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_add (Add)          (None, 7, 7, 2048)   0
conv5_block1_out[0][0]

conv5_block2_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_out (Activation)   (None, 7, 7, 2048)   0
conv5_block2_add[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088
conv5_block2_out[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block3_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_1_relu (Activation (None, 7, 7, 512)    0
conv5_block3_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808
conv5_block3_1_relu[0][0]
```

```
__________________________________________________________________________________________________
__________________
conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block3_2_conv[0][0]
__________________________________________________________________________________________________
__________________
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0
conv5_block3_2_bn[0][0]
__________________________________________________________________________________________________
__________________
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624
conv5_block3_2_relu[0][0]
__________________________________________________________________________________________________
__________________
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block3_3_conv[0][0]
__________________________________________________________________________________________________
__________________
conv5_block3_add (Add)          (None, 7, 7, 2048)   0
conv5_block2_out[0][0]

conv5_block3_3_bn[0][0]
__________________________________________________________________________________________________
__________________
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0
conv5_block3_add[0][0]
__________________________________________________________________________________________________
__________________
flatten_5 (Flatten)             (None, 100352)       0
conv5_block3_out[0][0]
__________________________________________________________________________________________________
__________________
dense_18 (Dense)                (None, 512)          51380736    flatten_5[0][0]
__________________________________________________________________________________________________
__________________
dense_19 (Dense)                (None, 224)          114912      dense_18[0][0]
__________________________________________________________________________________________________
__________________
dense_20 (Dense)                (None, 224)          50400       dense_19[0][0]
__________________________________________________________________________________________________
__________________
dense_21 (Dense)                (None, 224)          50400       dense_20[0][0]
__________________________________________________________________________________________________
__________________
dense_22 (Dense)                (None, 224)          50400       dense_21[0][0]
__________________________________________________________________________________________________
__________________
dense_23 (Dense)                (None, 224)          50400       dense_22[0][0]
__________________________________________________________________________________________________
__________________
dense_24 (Dense)                (None, 224)          50400       dense_23[0][0]
__________________________________________________________________________________________________
__________________
dense_25 (Dense)                (None, 197)          44325       dense_24[0][0]
==================================================================================================
==================
Total params: 75,379,685
Trainable params: 75,326,565
Non-trainable params: 53,120
     __________________________________________________________________________________________
__________________________
```

3. Define optimizer

```
opt= Adam(learning_rate=0.001)
```

4. Training [Forward pass and Backpropagation]

```
#Compile
full_model.compile(optimizer= opt, loss = 'categorical_crossentropy', metrics
= ['accuracy'])


#Early stopping
early = EarlyStopping(monitor='val_accuracy',min_delta=0,patience=40,verbose=1
,mode='auto')


res_classifier=full_model.fit_generator(train_generator,steps_per_epoch = 2, e
pochs =30, validation_data = validation_generator,
            validation_steps = 1,callbacks = [early])
```

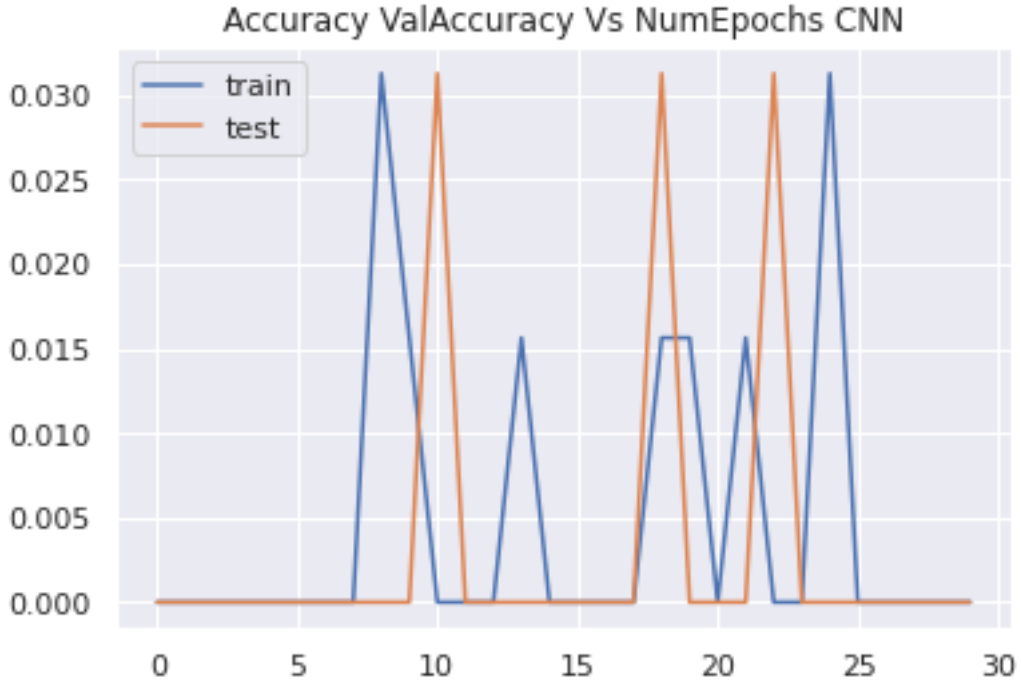
```
Epoch 1/30
2/2 [==============================] - 11s 2s/step - loss: 5.3621 - accuracy:
0.0000e+00 - val_loss: 5.5525 - val_accuracy: 0.0000e+00
Epoch 2/30
2/2 [==============================] - 2s 914ms/step - loss: 5.4601 -
accuracy: 0.0000e+00 - val_loss: 5.7100 - val_accuracy: 0.0000e+00
Epoch 3/30
2/2 [==============================] - 2s 1s/step - loss: 5.4117 - accuracy:
0.0000e+00 - val_loss: 5.5345 - val_accuracy: 0.0000e+00
Epoch 4/30
2/2 [==============================] - 2s 993ms/step - loss: 5.3791 -
accuracy: 0.0000e+00 - val_loss: 5.5610 - val_accuracy: 0.0000e+00
Epoch 5/30
2/2 [==============================] - 2s 1s/step - loss: 5.5947 - accuracy:
0.0000e+00 - val_loss: 5.6320 - val_accuracy: 0.0000e+00
Epoch 6/30
2/2 [==============================] - 2s 1s/step - loss: 5.3780 - accuracy:
0.0000e+00 - val_loss: 5.4623 - val_accuracy: 0.0000e+00
Epoch 7/30
2/2 [==============================] - 2s 1s/step - loss: 5.4169 - accuracy:
0.0000e+00 - val_loss: 5.5907 - val_accuracy: 0.0000e+00
Epoch 8/30
2/2 [==============================] - 2s 1s/step - loss: 5.4999 - accuracy:
0.0000e+00 - val_loss: 5.4237 - val_accuracy: 0.0000e+00
```

```
Epoch 9/30
2/2 [==============================] - 2s 965ms/step - loss: 5.4621 -
accuracy: 0.0312 - val_loss: 5.4819 - val_accuracy: 0.0000e+00
Epoch 10/30
2/2 [==============================] - 2s 1s/step - loss: 5.4418 - accuracy:
0.0156 - val_loss: 5.4853 - val_accuracy: 0.0000e+00
Epoch 11/30
2/2 [==============================] - 2s 1s/step - loss: 5.5179 - accuracy:
0.0000e+00 - val_loss: 5.3407 - val_accuracy: 0.0312
Epoch 12/30
2/2 [==============================] - 2s 955ms/step - loss: 5.3203 -
accuracy: 0.0000e+00 - val_loss: 5.5048 - val_accuracy: 0.0000e+00
Epoch 13/30
2/2 [==============================] - 2s 1s/step - loss: 5.3476 - accuracy:
0.0000e+00 - val_loss: 5.3331 - val_accuracy: 0.0000e+00
Epoch 14/30
2/2 [==============================] - 2s 954ms/step - loss: 5.2757 -
accuracy: 0.0156 - val_loss: 5.3322 - val_accuracy: 0.0000e+00
Epoch 15/30
2/2 [==============================] - 2s 1s/step - loss: 5.3678 - accuracy:
0.0000e+00 - val_loss: 5.4033 - val_accuracy: 0.0000e+00
Epoch 16/30
2/2 [==============================] - 2s 1s/step - loss: 5.4252 - accuracy:
0.0000e+00 - val_loss: 5.3253 - val_accuracy: 0.0000e+00
Epoch 17/30
2/2 [==============================] - 2s 1s/step - loss: 5.4463 - accuracy:
0.0000e+00 - val_loss: 5.3175 - val_accuracy: 0.0000e+00
Epoch 18/30
2/2 [==============================] - 2s 1s/step - loss: 5.4036 - accuracy:
0.0000e+00 - val_loss: 5.3590 - val_accuracy: 0.0000e+00
Epoch 19/30
2/2 [==============================] - 2s 987ms/step - loss: 5.3229 -
accuracy: 0.0156 - val_loss: 5.3885 - val_accuracy: 0.0312
Epoch 20/30
2/2 [==============================] - 2s 1s/step - loss: 5.2921 - accuracy:
0.0156 - val_loss: 5.3115 - val_accuracy: 0.0000e+00
Epoch 21/30
2/2 [==============================] - 2s 957ms/step - loss: 5.3349 -
accuracy: 0.0000e+00 - val_loss: 5.2952 - val_accuracy: 0.0000e+00
Epoch 22/30
2/2 [==============================] - 2s 993ms/step - loss: 5.3338 -
accuracy: 0.0156 - val_loss: 5.3405 - val_accuracy: 0.0000e+00
Epoch 23/30
2/2 [==============================] - 2s 994ms/step - loss: 5.3303 -
accuracy: 0.0000e+00 - val_loss: 5.3367 - val_accuracy: 0.0312
Epoch 24/30
2/2 [==============================] - 2s 1s/step - loss: 5.3819 - accuracy:
0.0000e+00 - val_loss: 5.2927 - val_accuracy: 0.0000e+00
Epoch 25/30
2/2 [==============================] - 2s 1s/step - loss: 5.3205 - accuracy:
0.0312 - val_loss: 5.3621 - val_accuracy: 0.0000e+00
Epoch 26/30
2/2 [==============================] - 2s 1s/step - loss: 5.3144 - accuracy:
0.0000e+00 - val_loss: 5.3109 - val_accuracy: 0.0000e+00
Epoch 27/30
2/2 [==============================] - 2s 1s/step - loss: 5.3487 - accuracy:
0.0000e+00 - val_loss: 5.2597 - val_accuracy: 0.0000e+00
Epoch 28/30
2/2 [==============================] - 2s 958ms/step - loss: 5.3365 -
accuracy: 0.0000e+00 - val_loss: 5.3778 - val_accuracy: 0.0000e+00
Epoch 29/30
```

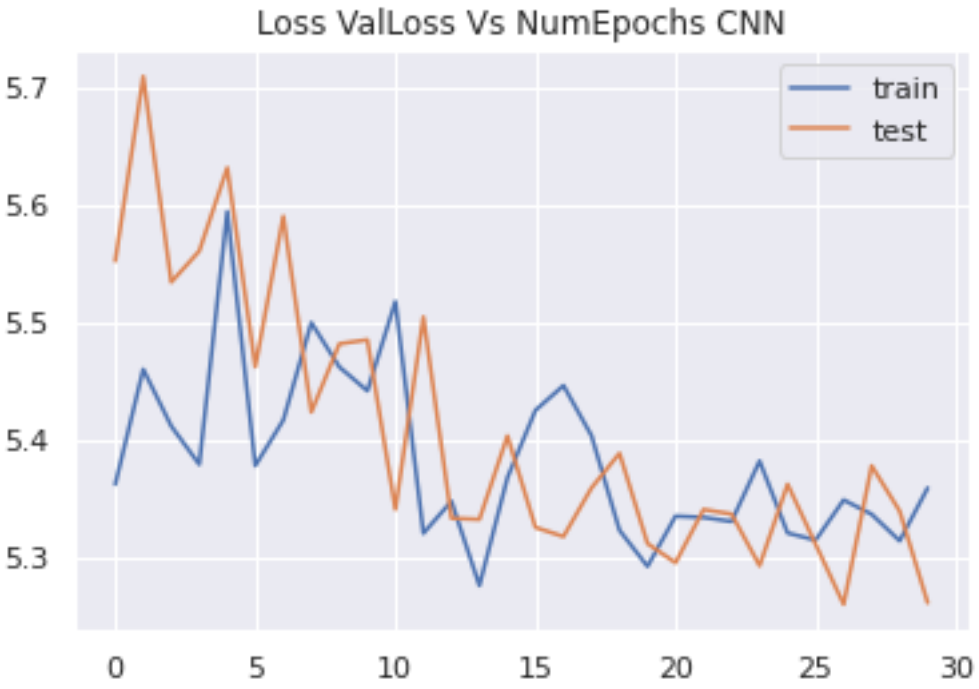
```
2/2 [==============================] - 2s 1s/step - loss: 5.3138 - accuracy:
0.0000e+00 - val_loss: 5.3392 - val_accuracy: 0.0000e+00
Epoch 30/30
        2/2 [==============================] - 2s 1s/step - loss: 5.3589 -
        accuracy: 0.0000e+00 - val_loss: 5.2610 - val_accuracy: 0.0000e+00
```

5. Plot Accuracy and Loss for Training and Validation

```
train_loss = res_classifier.history['loss']
val_loss   = res_classifier.history['val_loss']

xc = res_classifier.epoch
plt.title("Accuracy ValAccuracy Vs NumEpochs CNN")
plt.plot(xc,res_classifier.history['accuracy'], label='train')
plt.plot(xc,res_classifier.history['val_accuracy'], label='test')
plt.legend()
plt.show()

plt.figure()
plt.title("Loss ValLoss Vs NumEpochs CNN")
plt.plot(xc, train_loss,label='train')
plt.plot(xc, val_loss,label='test')
plt.legend()
plt.show
```



Plot shows that model tries to touch peaks and troughs with increasing epochs.



Plot shows that model tries to reduce loss for both training and validation dataset with each epoch.

6. Evaluation

```
train_acc = full_model.evaluate_generator(train_generator,steps = int(train_ge
nerator.samples/BATCH_SIZE))
val_acc = full_model.evaluate_generator(validation_generator, steps = int(vali
dation_generator.samples/BATCH_SIZE))
```

```
print(train_acc[1])
print(val_acc[1])
```

```
0.005290354136377573
0.005229083821177483
```

Thus, this model performs very poorly on both training and validation dataset.

7. **Adding results to dataframe for final comparison**

```
#Adding Performance metrics of ResNet50 to the list
tempResultsDf = pd.DataFrame({'Model':['ResNet50'],  'Train_Accuracy': train_a
cc[1],'Test_Accuracy': val_acc[1]})
resultsDf = pd.concat([resultsDf, tempResultsDf])
resultsDf = resultsDf[['Model', 'Train_Accuracy','Test_Accuracy']]
resultsDf
```

	Model	Train_Accuracy	Test_Accuracy
0	CNN	0.359129	0.158616
0	ResNet50	0.005290	0.005229

C. VGG16

1. Creating the model

```
print ( 'VGG with custom FC Layers')
vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape= (224,224,
3))
# Freeze all the layers except for the last layer:
for layer in vgg_conv.layers:
    layer.trainable = False

x = Flatten()(vgg_conv.output)
x = Dense(197, activation='softmax')(x)
vgg_model = Model(vgg_conv.input, x)
```

```
VGG with custom FC Layers
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
        58900480/58889256 [==============================] - 1s 0us/step
```

2. **Summary of model**

```
vgg_model.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_7 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_27 (Dense)             (None, 197)               4942533
=================================================================
Total params: 19,657,221
Trainable params: 4,942,533
Non-trainable params: 14,714,688
```

3. Training [Forward pass and Backpropagation]

```
#Compile the model with Adam optimizer
vgg_model.compile(optimizer = Adam(learning_rate=0.001), loss = 'categorical_c
rossentropy', metrics = ['accuracy'])
```

```
early = EarlyStopping(monitor='val_accuracy',min_delta=0.01,patience=20,verbos
e=1,mode='auto')

#Training
vgg_classifier = vgg_model.fit_generator(train_generator, epochs =30, validati
on_data = validation_generator, callbacks = [early]  )
```

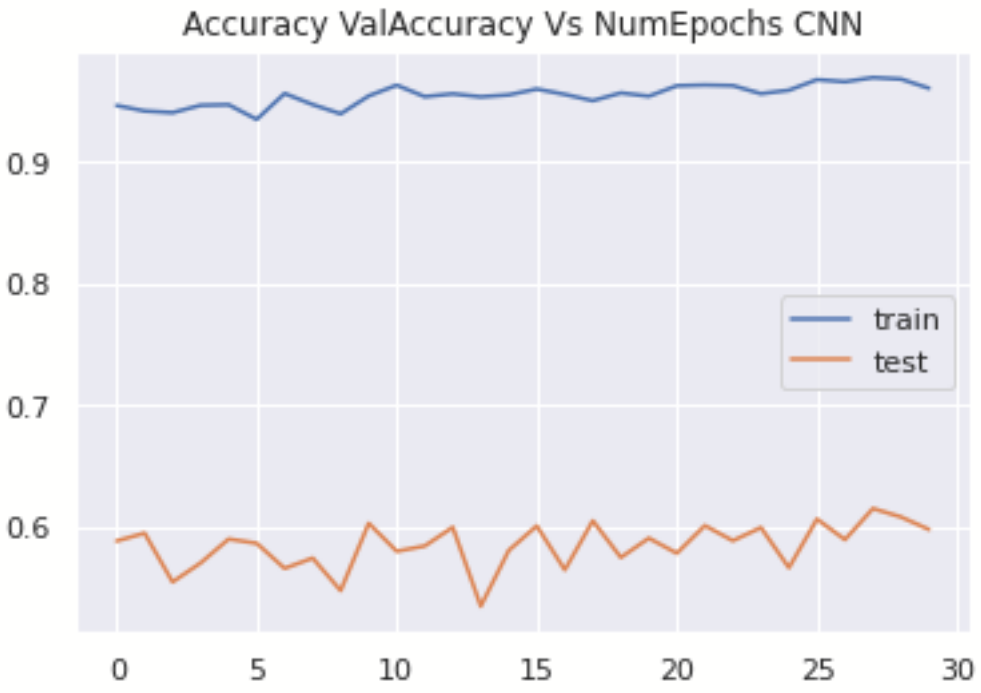
```
Epoch 1/30
255/255 [==============================] - 187s 731ms/step - loss: 0.3102 -
accuracy: 0.9467 - val_loss: 4.6444 - val_accuracy: 0.5884
Epoch 2/30
255/255 [==============================] - 183s 718ms/step - loss: 0.3438 -
accuracy: 0.9422 - val_loss: 4.7269 - val_accuracy: 0.5948
Epoch 3/30
255/255 [==============================] - 181s 711ms/step - loss: 0.3808 -
accuracy: 0.9408 - val_loss: 5.6650 - val_accuracy: 0.5547
Epoch 4/30
255/255 [==============================] - 180s 708ms/step - loss: 0.3101 -
accuracy: 0.9468 - val_loss: 5.3064 - val_accuracy: 0.5703
Epoch 5/30
255/255 [==============================] - 181s 709ms/step - loss: 0.3306 -
accuracy: 0.9473 - val_loss: 5.1069 - val_accuracy: 0.5900
Epoch 6/30
255/255 [==============================] - 180s 707ms/step - loss: 0.4257 -
accuracy: 0.9353 - val_loss: 5.2278 - val_accuracy: 0.5865
Epoch 7/30
255/255 [==============================] - 180s 708ms/step - loss: 0.2552 -
accuracy: 0.9565 - val_loss: 5.8028 - val_accuracy: 0.5659
Epoch 8/30
255/255 [==============================] - 180s 707ms/step - loss: 0.3380 -
accuracy: 0.9474 - val_loss: 5.7594 - val_accuracy: 0.5743
Epoch 9/30
255/255 [==============================] - 180s 705ms/step - loss: 0.3781 -
accuracy: 0.9398 - val_loss: 6.2191 - val_accuracy: 0.5476
Epoch 10/30
255/255 [==============================] - 180s 706ms/step - loss: 0.3021 -
accuracy: 0.9546 - val_loss: 4.9970 - val_accuracy: 0.6029
Epoch 11/30
255/255 [==============================] - 181s 709ms/step - loss: 0.2226 -
accuracy: 0.9634 - val_loss: 5.5075 - val_accuracy: 0.5799
Epoch 12/30
255/255 [==============================] - 181s 708ms/step - loss: 0.3211 -
accuracy: 0.9540 - val_loss: 5.6694 - val_accuracy: 0.5843
Epoch 13/30
255/255 [==============================] - 182s 715ms/step - loss: 0.2927 -
accuracy: 0.9563 - val_loss: 5.5035 - val_accuracy: 0.5997
Epoch 14/30
255/255 [==============================] - 183s 717ms/step - loss: 0.3017 -
accuracy: 0.9538 - val_loss: 6.6898 - val_accuracy: 0.5348
Epoch 15/30
255/255 [==============================] - 182s 715ms/step - loss: 0.3076 -
accuracy: 0.9554 - val_loss: 5.8260 - val_accuracy: 0.5806
Epoch 16/30
255/255 [==============================] - 183s 717ms/step - loss: 0.2479 -
accuracy: 0.9602 - val_loss: 5.4916 - val_accuracy: 0.6008
Epoch 17/30
```

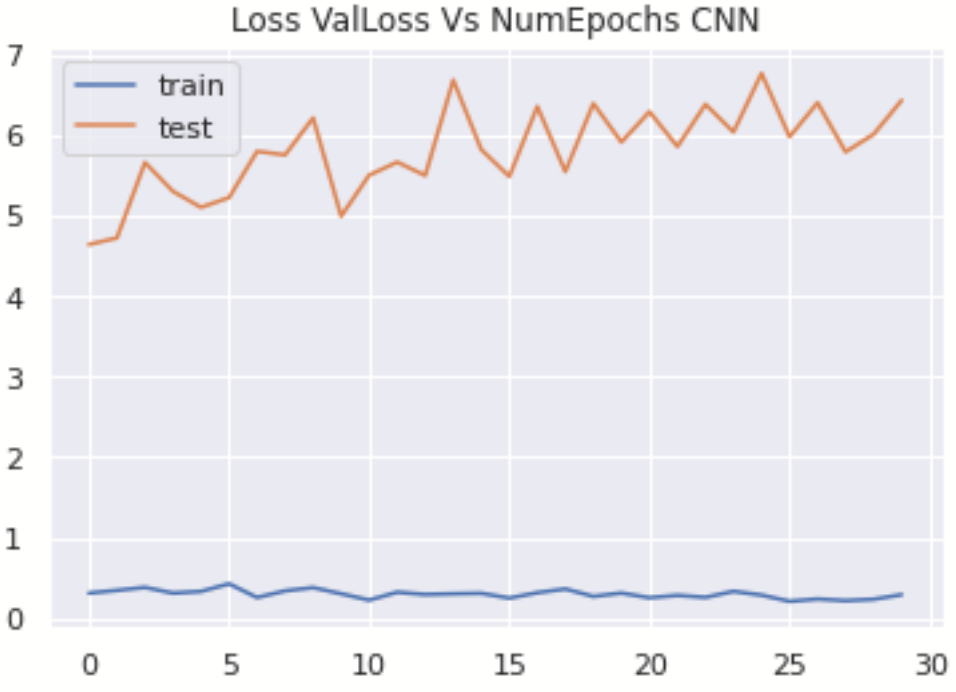
```
255/255 [==============================] - 184s 723ms/step - loss: 0.3149 -
accuracy: 0.9558 - val_loss: 6.3606 - val_accuracy: 0.5646
Epoch 18/30
255/255 [==============================] - 184s 720ms/step - loss: 0.3615 -
accuracy: 0.9506 - val_loss: 5.5514 - val_accuracy: 0.6050
Epoch 19/30
255/255 [==============================] - 181s 711ms/step - loss: 0.2693 -
accuracy: 0.9570 - val_loss: 6.3985 - val_accuracy: 0.5748
Epoch 20/30
255/255 [==============================] - 182s 715ms/step - loss: 0.3107 -
accuracy: 0.9543 - val_loss: 5.9179 - val_accuracy: 0.5907
Epoch 21/30
255/255 [==============================] - 182s 713ms/step - loss: 0.2523 -
accuracy: 0.9629 - val_loss: 6.2966 - val_accuracy: 0.5783
Epoch 22/30
255/255 [==============================] - 182s 713ms/step - loss: 0.2820 -
accuracy: 0.9635 - val_loss: 5.8626 - val_accuracy: 0.6010
Epoch 23/30
255/255 [==============================] - 181s 709ms/step - loss: 0.2569 -
accuracy: 0.9630 - val_loss: 6.3886 - val_accuracy: 0.5886
Epoch 24/30
255/255 [==============================] - 181s 711ms/step - loss: 0.3322 -
accuracy: 0.9564 - val_loss: 6.0438 - val_accuracy: 0.5994
Epoch 25/30
255/255 [==============================] - 182s 714ms/step - loss: 0.2849 -
accuracy: 0.9592 - val_loss: 6.7721 - val_accuracy: 0.5663
Epoch 26/30
255/255 [==============================] - 181s 710ms/step - loss: 0.2081 -
accuracy: 0.9680 - val_loss: 5.9842 - val_accuracy: 0.6064
Epoch 27/30
255/255 [==============================] - 183s 717ms/step - loss: 0.2362 -
accuracy: 0.9662 - val_loss: 6.4080 - val_accuracy: 0.5895
Epoch 28/30
255/255 [==============================] - 182s 713ms/step - loss: 0.2165 -
accuracy: 0.9697 - val_loss: 5.7963 - val_accuracy: 0.6151
Epoch 29/30
255/255 [==============================] - 182s 713ms/step - loss: 0.2329 -
accuracy: 0.9684 - val_loss: 6.0159 - val_accuracy: 0.6081
Epoch 30/30
        255/255 [==============================] - 180s 707ms/step - loss:
        0.2903 - accuracy: 0.9608 - val_loss: 6.4375 - val_accuracy: 0.5978
```

4. Plot Accuracy and Loss

```
train_loss = vgg_classifier.history['loss']
val_loss   = vgg_classifier.history['val_loss']

xc = vgg_classifier.epoch
plt.title("Accuracy ValAccuracy Vs NumEpochs CNN")
plt.plot(xc,vgg_classifier.history['accuracy'], label='train')
plt.plot(xc,vgg_classifier.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```

```
plt.figure()
plt.title("Loss ValLoss Vs NumEpochs CNN")
plt.plot(xc, train_loss,label='train')
plt.plot(xc, val_loss,label='test')
plt.legend()
plt.show
```





Plot shows that both training and validation accuracy, and training and validation loss remains more or less constant over epochs.

5. Evaluation

```
train_acc = vgg_model.evaluate_generator(train_generator,steps = int(train_gen
erator.samples/BATCH_SIZE))
val_acc = vgg_model.evaluate_generator(validation_generator, steps = int(valid
ation_generator.samples/BATCH_SIZE))

print(train_acc[1])
print(val_acc[1])
```

```
0.9386072754859924
0.5639940500259399
```

Therefore, model shows a high training accuracy of around 94%. But validation accuracy is low at 56%. This shows a high variance problem.

6. Add result to dataframe for final comparison

```
#Adding Performance metrics of ResNet50 to the list
tempResultsDf = pd.DataFrame({'Model':['VGG16'],  'Train_Accuracy': train_acc[
1],'Test_Accuracy': val_acc[1]})
resultsDf = pd.concat([resultsDf, tempResultsDf])
resultsDf = resultsDf[['Model', 'Train_Accuracy','Test_Accuracy']]
resultsDf
```

	Model	Train_Accuracy	Test_Accuracy
0	CNN	0.359129	0.158616
0	ResNet50	0.005290	0.005229
0	VGG16	0.938607	0.563994

7. Save model for future use

```
vgg_model.save('./vgg.h5')

vgg_model.save_weights('./vgg_weights.h5')
```

D. ResNet50 with custom FC layer

1. Creating the model

```
resnet_conv = ResNet50(weights='imagenet', include_top=False, input_shape=(224
,224,3))

# Freeze all the layers except for the last layer:
for layer in resnet_conv.layers:
    layer.trainable = False


x2 = Flatten()(resnet_conv.output)
x2 = Dense(197, activation='sigmoid')(x2)
resnet = Model(resnet_conv.input, x2)
```

2. Summary of model

```
resnet.summary()

Model: "model_4"
_____________________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #     Connected to
=============================================================================
=====================
input_5 (InputLayer)         [(None, 224, 224, 3)  0
_____________________________________________________________________________
_____________________
conv1_pad (ZeroPadding2D)    (None, 230, 230, 3)   0           input_5[0][0]
_____________________________________________________________________________
_____________________
```

```
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472
conv1_pad[0][0]
__________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256
conv1_conv[0][0]
__________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0
conv1_bn[0][0]
__________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0
conv1_relu[0][0]
__________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0
pool1_pad[0][0]
__________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 56, 56, 64)   4160
pool1_pool[0][0]
__________________________________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block1_1_conv[0][0]
__________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)   0
conv2_block1_1_bn[0][0]
__________________________________________________________________________________________
conv2_block1_2_conv (Conv2D)    (None, 56, 56, 64)   36928
conv2_block1_1_relu[0][0]
__________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block1_2_conv[0][0]
__________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)   0
conv2_block1_2_bn[0][0]
__________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 56, 56, 256)  16640
pool1_pool[0][0]
__________________________________________________________________________________________
conv2_block1_3_conv (Conv2D)    (None, 56, 56, 256)  16640
conv2_block1_2_relu[0][0]
__________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block1_0_conv[0][0]
__________________________________________________________________________________________
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block1_3_conv[0][0]
__________________________________________________________________________________________
```

```
conv2_block1_add (Add)          (None, 56, 56, 256)  0
conv2_block1_0_bn[0][0]

conv2_block1_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block1_out (Activation)   (None, 56, 56, 256)  0
conv2_block1_add[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_1_conv (Conv2D)    (None, 56, 56, 64)   16448
conv2_block1_out[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block2_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_1_relu (Activation (None, 56, 56, 64)   0
conv2_block2_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_2_conv (Conv2D)    (None, 56, 56, 64)   36928
conv2_block2_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block2_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_2_relu (Activation (None, 56, 56, 64)   0
conv2_block2_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_3_conv (Conv2D)    (None, 56, 56, 256)  16640
conv2_block2_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block2_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_add (Add)          (None, 56, 56, 256)  0
conv2_block1_out[0][0]

conv2_block2_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block2_out (Activation)   (None, 56, 56, 256)  0
conv2_block2_add[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_1_conv (Conv2D)    (None, 56, 56, 64)   16448
conv2_block2_out[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_1_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block3_1_conv[0][0]
__________________________________________________________________________________________
__________________
```

```
conv2_block3_1_relu (Activation (None, 56, 56, 64)   0
conv2_block3_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_2_conv (Conv2D)    (None, 56, 56, 64)   36928
conv2_block3_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_2_bn (BatchNormali (None, 56, 56, 64)   256
conv2_block3_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_2_relu (Activation (None, 56, 56, 64)   0
conv2_block3_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_3_conv (Conv2D)    (None, 56, 56, 256)  16640
conv2_block3_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_3_bn (BatchNormali (None, 56, 56, 256)  1024
conv2_block3_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_add (Add)          (None, 56, 56, 256)  0
conv2_block2_out[0][0]

conv2_block3_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv2_block3_out (Activation)   (None, 56, 56, 256)  0
conv2_block3_add[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_1_conv (Conv2D)    (None, 28, 28, 128)  32896
conv2_block3_out[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block1_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_1_relu (Activation (None, 28, 28, 128)  0
conv3_block1_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block1_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block1_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_2_relu (Activation (None, 28, 28, 128)  0
conv3_block1_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block1_0_conv (Conv2D)    (None, 28, 28, 512)  131584
conv2_block3_out[0][0]
```

```
__________________________________________________________________________________________________
conv3_block1_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block1_2_relu[0][0]
__________________________________________________________________________________________________
conv3_block1_0_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block1_0_conv[0][0]
__________________________________________________________________________________________________
conv3_block1_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block1_3_conv[0][0]
__________________________________________________________________________________________________
conv3_block1_add (Add)          (None, 28, 28, 512)  0
conv3_block1_0_bn[0][0]

conv3_block1_3_bn[0][0]
__________________________________________________________________________________________________
conv3_block1_out (Activation)   (None, 28, 28, 512)  0
conv3_block1_add[0][0]
__________________________________________________________________________________________________
conv3_block2_1_conv (Conv2D)    (None, 28, 28, 128)  65664
conv3_block1_out[0][0]
__________________________________________________________________________________________________
conv3_block2_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block2_1_conv[0][0]
__________________________________________________________________________________________________
conv3_block2_1_relu (Activation (None, 28, 28, 128)  0
conv3_block2_1_bn[0][0]
__________________________________________________________________________________________________
conv3_block2_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block2_1_relu[0][0]
__________________________________________________________________________________________________
conv3_block2_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv3_block2_2_relu (Activation (None, 28, 28, 128)  0
conv3_block2_2_bn[0][0]
__________________________________________________________________________________________________
conv3_block2_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block2_2_relu[0][0]
__________________________________________________________________________________________________
conv3_block2_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block2_3_conv[0][0]
__________________________________________________________________________________________________
conv3_block2_add (Add)          (None, 28, 28, 512)  0
conv3_block1_out[0][0]

conv3_block2_3_bn[0][0]
```

```
__________________________________________________________________________________________________
____________________
conv3_block2_out (Activation)   (None, 28, 28, 512)  0
conv3_block2_add[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_1_conv (Conv2D)    (None, 28, 28, 128)  65664
conv3_block2_out[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block3_1_conv[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_1_relu (Activation (None, 28, 28, 128)  0
conv3_block3_1_bn[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block3_1_relu[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block3_2_conv[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_2_relu (Activation (None, 28, 28, 128)  0
conv3_block3_2_bn[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block3_2_relu[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block3_3_conv[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_add (Add)          (None, 28, 28, 512)  0
conv3_block2_out[0][0]

conv3_block3_3_bn[0][0]
__________________________________________________________________________________________________
____________________
conv3_block3_out (Activation)   (None, 28, 28, 512)  0
conv3_block3_add[0][0]
__________________________________________________________________________________________________
____________________
conv3_block4_1_conv (Conv2D)    (None, 28, 28, 128)  65664
conv3_block3_out[0][0]
__________________________________________________________________________________________________
____________________
conv3_block4_1_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block4_1_conv[0][0]
__________________________________________________________________________________________________
____________________
conv3_block4_1_relu (Activation (None, 28, 28, 128)  0
conv3_block4_1_bn[0][0]
__________________________________________________________________________________________________
____________________
```

```
conv3_block4_2_conv (Conv2D)    (None, 28, 28, 128)  147584
conv3_block4_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block4_2_bn (BatchNormali (None, 28, 28, 128)  512
conv3_block4_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block4_2_relu (Activation (None, 28, 28, 128)  0
conv3_block4_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block4_3_conv (Conv2D)    (None, 28, 28, 512)  66048
conv3_block4_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv3_block4_3_bn (BatchNormali (None, 28, 28, 512)  2048
conv3_block4_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv3_block4_add (Add)          (None, 28, 28, 512)  0
conv3_block3_out[0][0]

conv3_block4_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv3_block4_out (Activation)   (None, 28, 28, 512)  0
conv3_block4_add[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_1_conv (Conv2D)    (None, 14, 14, 256)  131328
conv3_block4_out[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block1_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_1_relu (Activation (None, 14, 14, 256)  0
conv4_block1_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block1_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block1_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_2_relu (Activation (None, 14, 14, 256)  0
conv4_block1_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_0_conv (Conv2D)    (None, 14, 14, 1024) 525312
conv3_block4_out[0][0]
__________________________________________________________________________________________
__________________
conv4_block1_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block1_2_relu[0][0]
```

```
__________________________________________________________________________________________________
____________________
conv4_block1_0_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block1_0_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block1_3_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_add (Add)          (None, 14, 14, 1024) 0
conv4_block1_0_bn[0][0]

conv4_block1_3_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block1_out (Activation)   (None, 14, 14, 1024) 0
conv4_block1_add[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block1_out[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block2_1_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_1_relu (Activation (None, 14, 14, 256)  0
conv4_block2_1_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block2_1_relu[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block2_2_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_2_relu (Activation (None, 14, 14, 256)  0
conv4_block2_2_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block2_2_relu[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block2_3_conv[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_add (Add)          (None, 14, 14, 1024) 0
conv4_block1_out[0][0]

conv4_block2_3_bn[0][0]
__________________________________________________________________________________________________
____________________
conv4_block2_out (Activation)   (None, 14, 14, 1024) 0
conv4_block2_add[0][0]
```

```
____________________________________________________________________________________________
____________________
conv4_block3_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block2_out[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block3_1_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_1_relu (Activation (None, 14, 14, 256)  0
conv4_block3_1_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block3_1_relu[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block3_2_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_2_relu (Activation (None, 14, 14, 256)  0
conv4_block3_2_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block3_2_relu[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block3_3_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_add (Add)          (None, 14, 14, 1024) 0
conv4_block2_out[0][0]

conv4_block3_3_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block3_out (Activation)   (None, 14, 14, 1024) 0
conv4_block3_add[0][0]
____________________________________________________________________________________________
____________________
conv4_block4_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block3_out[0][0]
____________________________________________________________________________________________
____________________
conv4_block4_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block4_1_conv[0][0]
____________________________________________________________________________________________
____________________
conv4_block4_1_relu (Activation (None, 14, 14, 256)  0
conv4_block4_1_bn[0][0]
____________________________________________________________________________________________
____________________
conv4_block4_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block4_1_relu[0][0]
____________________________________________________________________________________________
____________________
```

```
conv4_block4_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block4_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block4_2_relu (Activation (None, 14, 14, 256)  0
conv4_block4_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block4_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block4_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv4_block4_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block4_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block4_add (Add)          (None, 14, 14, 1024) 0
conv4_block3_out[0][0]

conv4_block4_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block4_out (Activation)   (None, 14, 14, 1024) 0
conv4_block4_add[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block4_out[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block5_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_1_relu (Activation (None, 14, 14, 256)  0
conv4_block5_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block5_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block5_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_2_relu (Activation (None, 14, 14, 256)  0
conv4_block5_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block5_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block5_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv4_block5_add (Add)          (None, 14, 14, 1024) 0
conv4_block4_out[0][0]
```

```
conv4_block5_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block5_out (Activation)   (None, 14, 14, 1024) 0
conv4_block5_add[0][0]
__________________________________________________________________________________________________
conv4_block6_1_conv (Conv2D)    (None, 14, 14, 256)  262400
conv4_block5_out[0][0]
__________________________________________________________________________________________________
conv4_block6_1_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block6_1_conv[0][0]
__________________________________________________________________________________________________
conv4_block6_1_relu (Activation (None, 14, 14, 256)  0
conv4_block6_1_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_2_conv (Conv2D)    (None, 14, 14, 256)  590080
conv4_block6_1_relu[0][0]
__________________________________________________________________________________________________
conv4_block6_2_bn (BatchNormali (None, 14, 14, 256)  1024
conv4_block6_2_conv[0][0]
__________________________________________________________________________________________________
conv4_block6_2_relu (Activation (None, 14, 14, 256)  0
conv4_block6_2_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block6_2_relu[0][0]
__________________________________________________________________________________________________
conv4_block6_3_bn (BatchNormali (None, 14, 14, 1024) 4096
conv4_block6_3_conv[0][0]
__________________________________________________________________________________________________
conv4_block6_add (Add)          (None, 14, 14, 1024) 0
conv4_block5_out[0][0]

conv4_block6_3_bn[0][0]
__________________________________________________________________________________________________
conv4_block6_out (Activation)   (None, 14, 14, 1024) 0
conv4_block6_add[0][0]
__________________________________________________________________________________________________
conv5_block1_1_conv (Conv2D)    (None, 7, 7, 512)    524800
conv4_block6_out[0][0]
__________________________________________________________________________________________________
conv5_block1_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv5_block1_1_relu (Activation (None, 7, 7, 512)    0
conv5_block1_1_bn[0][0]
```

```
__________________________________________________________________________________________________
____________________
conv5_block1_2_conv (Conv2D)    (None, 7, 7, 512)    2359808
conv5_block1_1_relu[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block1_2_conv[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_2_relu (Activation (None, 7, 7, 512)    0
conv5_block1_2_bn[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_0_conv (Conv2D)    (None, 7, 7, 2048)   2099200
conv4_block6_out[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624
conv5_block1_2_relu[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_0_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block1_0_conv[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_3_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block1_3_conv[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_add (Add)          (None, 7, 7, 2048)   0
conv5_block1_0_bn[0][0]

conv5_block1_3_bn[0][0]
__________________________________________________________________________________________________
____________________
conv5_block1_out (Activation)   (None, 7, 7, 2048)   0
conv5_block1_add[0][0]
__________________________________________________________________________________________________
____________________
conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)    1049088
conv5_block1_out[0][0]
__________________________________________________________________________________________________
____________________
conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block2_1_conv[0][0]
__________________________________________________________________________________________________
____________________
conv5_block2_1_relu (Activation (None, 7, 7, 512)    0
conv5_block2_1_bn[0][0]
__________________________________________________________________________________________________
____________________
conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)    2359808
conv5_block2_1_relu[0][0]
__________________________________________________________________________________________________
____________________
conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block2_2_conv[0][0]
__________________________________________________________________________________________________
____________________
```

```
conv5_block2_2_relu (Activation (None, 7, 7, 512)    0
conv5_block2_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624
conv5_block2_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block2_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_add (Add)          (None, 7, 7, 2048)   0
conv5_block1_out[0][0]

conv5_block2_3_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block2_out (Activation)   (None, 7, 7, 2048)   0
conv5_block2_add[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_1_conv (Conv2D)    (None, 7, 7, 512)    1049088
conv5_block2_out[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block3_1_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_1_relu (Activation (None, 7, 7, 512)    0
conv5_block3_1_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_2_conv (Conv2D)    (None, 7, 7, 512)    2359808
conv5_block3_1_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block3_2_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_2_relu (Activation (None, 7, 7, 512)    0
conv5_block3_2_bn[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624
conv5_block3_2_relu[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192
conv5_block3_3_conv[0][0]
__________________________________________________________________________________________
__________________
conv5_block3_add (Add)          (None, 7, 7, 2048)   0
conv5_block2_out[0][0]

conv5_block3_3_bn[0][0]
__________________________________________________________________________________________
__________________
```

```
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0
conv5_block3_add[0][0]
__________________________________________________________________________________
_____________________
flatten_5 (Flatten)             (None, 100352)       0
conv5_block3_out[0][0]
__________________________________________________________________________________
_____________________
dense_14 (Dense)                (None, 197)          19769541
flatten_5[0][0]
==================================================================================
====================
Total params: 43,357,253
Trainable params: 19,769,541
Non-trainable params: 23,587,712
```

```
          ____________________________________________________________________________
   ___________________________________
```

3. Training [Forward pass and Backpropagation]

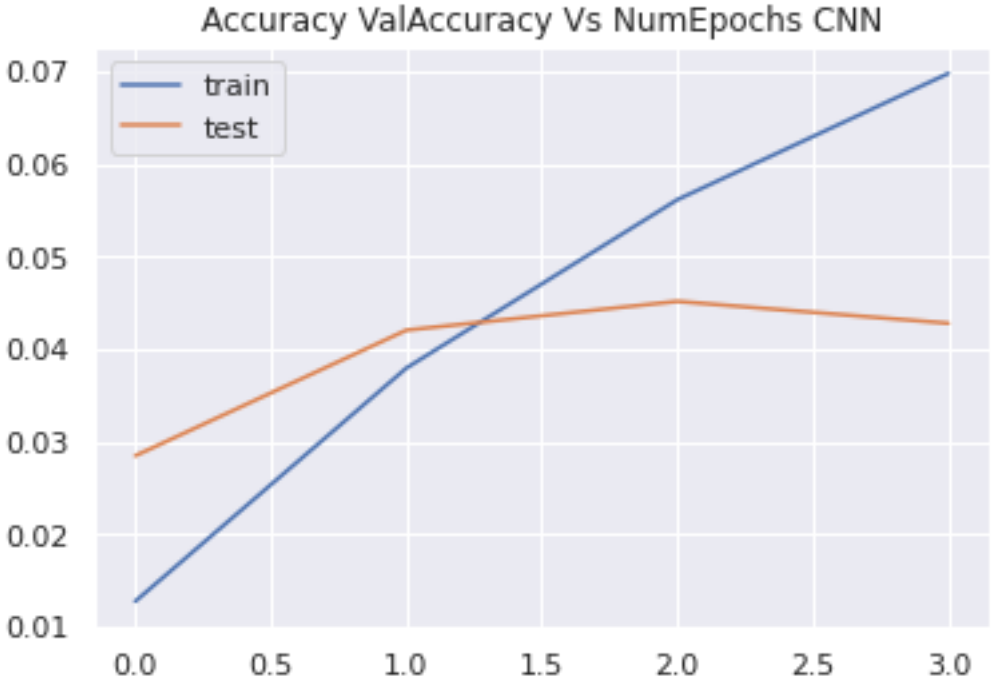
```
#Compile with optimizer
resnet.compile(optimizer = Adam(learning_rate=0.001), loss = 'categorical_cros
sentropy', metrics = ['accuracy'])

early = EarlyStopping(monitor='val_accuracy',min_delta=0.01,patience=2,verbose
=1,mode='auto')

#Training
resnet_classifier = resnet.fit_generator(train_generator,epochs =30, validatio
n_data = validation_generator, callbacks = [early]  )
```

```
Epoch 1/30
255/255 [==============================] - 188s 726ms/step - loss: 25.5408 -
accuracy: 0.0128 - val_loss: 14.5744 - val_accuracy: 0.0285
Epoch 2/30
255/255 [==============================] - 182s 716ms/step - loss: 13.3197 -
accuracy: 0.0379 - val_loss: 14.4397 - val_accuracy: 0.0420
Epoch 3/30
255/255 [==============================] - 183s 718ms/step - loss: 12.8443 -
accuracy: 0.0561 - val_loss: 13.0493 - val_accuracy: 0.0451
Epoch 4/30
255/255 [==============================] - 183s 718ms/step - loss: 12.7389 -
accuracy: 0.0697 - val_loss: 15.5320 - val_accuracy: 0.0428
         Epoch 00004: early stopping
```

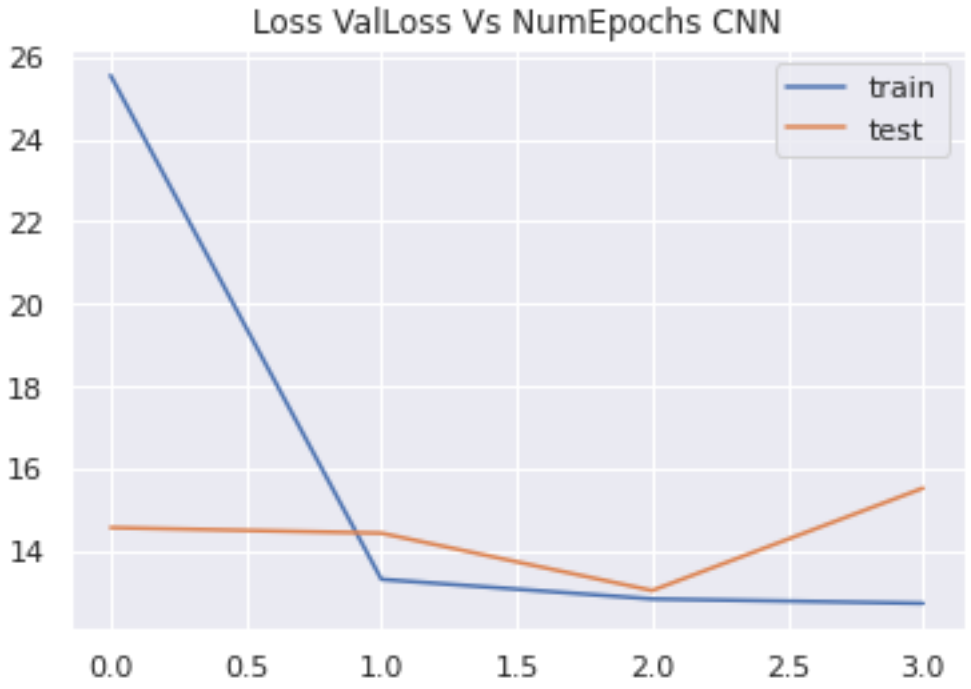
4. Plot Accuracy and Loss

```
train_loss = resnet_classifier.history['loss']
```

```
val_loss   = resnet_classifier.history['val_loss']

xc = resnet_classifier.epoch
plt.title("Accuracy ValAccuracy Vs NumEpochs CNN")
plt.plot(xc,resnet_classifier.history['accuracy'], label='train')
plt.plot(xc,resnet_classifier.history['val_accuracy'], label='test')
plt.legend()
plt.show()

plt.figure()
plt.title("Loss ValLoss Vs NumEpochs CNN")
plt.plot(xc, train_loss,label='train')
plt.plot(xc, val_loss,label='test')
plt.legend()
plt.show
```



As seen in the graph, training set accuracy continue to increase with each epoch. However, validation set accuracy doesn't change after few epochs.



As seen above, training dataset loss shows a sharp drop after initial few epochs and then becomes constant. Validation dataset loss shows a marginal drop after few epochs and then starts to increase again.

5. Evaluation

```
train_acc = resnet.evaluate_generator(train_generator,steps = int(train_genera
tor.samples/BATCH_SIZE))
val_acc = resnet.evaluate_generator(validation_generator, steps = int(validati
on_generator.samples/BATCH_SIZE))

print(train_acc[1])
print(val_acc[1])
```

```
0.07221949100494385
0.04282868653535843
```

As seen, this model gives a very low training and validation accuracy of 7% and 4% respectively.

6. Adding result to dataframe for comparison

```
#Adding Performance metrics of Custom ResNet50 to the list
tempResultsDf = pd.DataFrame({'Model':['ResNet Custom FC'],  'Train_Accuracy':
 train_acc[1],'Test_Accuracy': val_acc[1]})
```

```
resultsDf = pd.concat([resultsDf, tempResultsDf])
resultsDf = resultsDf[['Model', 'Train_Accuracy','Test_Accuracy']]
resultsDf
```

	Model	Train_Accuracy	Test_Accuracy
0	CNN	0.359129	0.158616
0	ResNet50	0.005290	0.005229
0	VGG16	0.938607	0.563994
0	ResNet Custom FC	0.072219	0.042829

7. Save model for future use

```
resnet.save('./resnet.h5')

resnet.save_weights('./resnet_weights.h5')
```

E. InceptionResNetV2

1. Creating the model

```
base_model = InceptionResNetV2(include_top=False, input_shape = INPUT_SIZE)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219062272/219055592 [==============================] - 2s 0us/step
219070464/219055592 [==============================] - 2s 0us/step
```

```
classification_model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
```

```
    tf.keras.layers.Dense(197, activation='softmax')
])
```

2. Summary of model

```
classification_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_resnet_v2 (Functio (None, 5, 5, 1536)        54336736
_________________________________________________________________
global_average_pooling2d (Gl (None, 1536)              0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               196736
_________________________________________________________________
batch_normalization_203 (Bat (None, 128)               512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 197)               25413
=================================================================
Total params: 54,559,397
Trainable params: 54,498,597
Non-trainable params: 60,800
```

3. Define optimizer

```
lr=0.001
classification_model.compile(loss='categorical_crossentropy', optimizer=Adam(l
r=lr), metrics=['accuracy'])
```

4. Early stopping and Model Checkpoint

```
patience = 1
stop_patience = 3
factor = 0.5

callbacks = [
    tf.keras.callbacks.ModelCheckpoint("classify_model.h5", save_best_only=Tru
e, verbose = 0),
    tf.keras.callbacks.EarlyStopping(patience=stop_patience, monitor='val_loss
', verbose=1),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor, pa
tience=patience, verbose=1)
]
```

5. Training [Forward pass and Backpropagation]

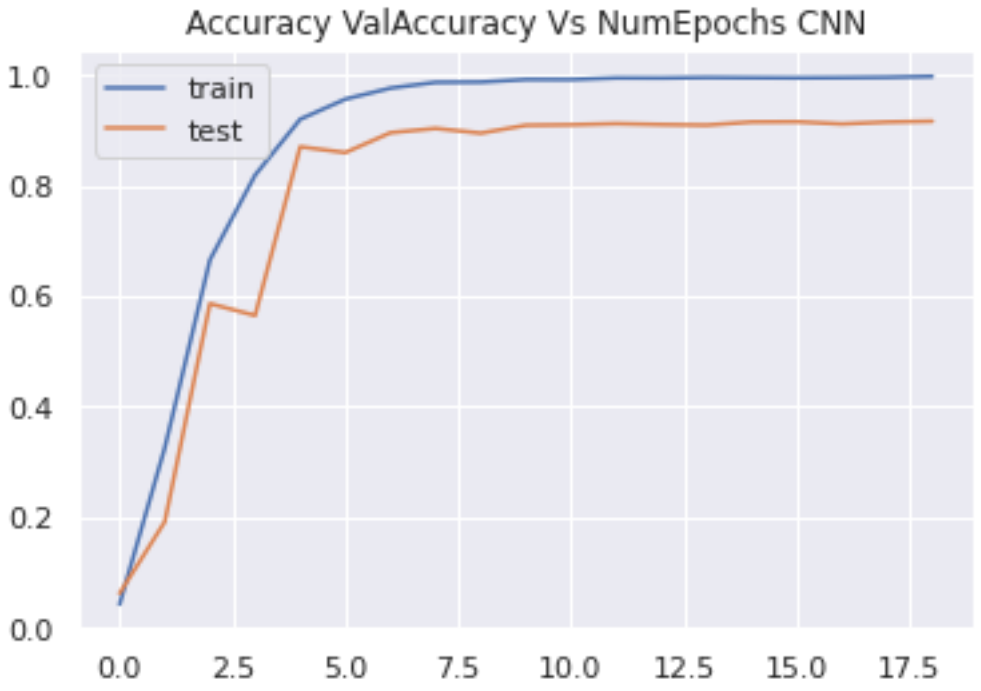
```
epochs = 30
history = classification_model.fit(train_generator, validation_data=validation
_generator, epochs=epochs, callbacks=callbacks, verbose=1)
```

```
Epoch 1/30
255/255 [==============================] - 219s 774ms/step - loss: 5.0064 -
accuracy: 0.0424 - val_loss: 4.8299 - val_accuracy: 0.0622
Epoch 2/30
255/255 [==============================] - 193s 756ms/step - loss: 3.0798 -
accuracy: 0.3256 - val_loss: 3.7128 - val_accuracy: 0.1919
Epoch 3/30
255/255 [==============================] - 192s 754ms/step - loss: 1.4914 -
accuracy: 0.6661 - val_loss: 1.6520 - val_accuracy: 0.5869
Epoch 4/30
255/255 [==============================] - 193s 755ms/step - loss: 0.7913 -
accuracy: 0.8196 - val_loss: 1.7750 - val_accuracy: 0.5655

Epoch 00004: ReduceLROnPlateau reducing learning rate to
0.0005000000237487257.
Epoch 5/30
255/255 [==============================] - 190s 745ms/step - loss: 0.3456 -
accuracy: 0.9209 - val_loss: 0.5200 - val_accuracy: 0.8710
Epoch 6/30
255/255 [==============================] - 193s 753ms/step - loss: 0.2035 -
accuracy: 0.9573 - val_loss: 0.5306 - val_accuracy: 0.8605

Epoch 00006: ReduceLROnPlateau reducing learning rate to
0.0002500000118743628.
Epoch 7/30
255/255 [==============================] - 191s 748ms/step - loss: 0.1194 -
accuracy: 0.9773 - val_loss: 0.3848 - val_accuracy: 0.8965
Epoch 8/30
255/255 [==============================] - 193s 755ms/step - loss: 0.0785 -
accuracy: 0.9877 - val_loss: 0.3624 - val_accuracy: 0.9042
Epoch 9/30
255/255 [==============================] - 193s 756ms/step - loss: 0.0683 -
accuracy: 0.9882 - val_loss: 0.3795 - val_accuracy: 0.8957

Epoch 00009: ReduceLROnPlateau reducing learning rate to
0.0001250000059371814.
Epoch 10/30
161/255 [=================>............] - ETA: 50s - loss: 0.0471 - accuracy:
0.9930
```

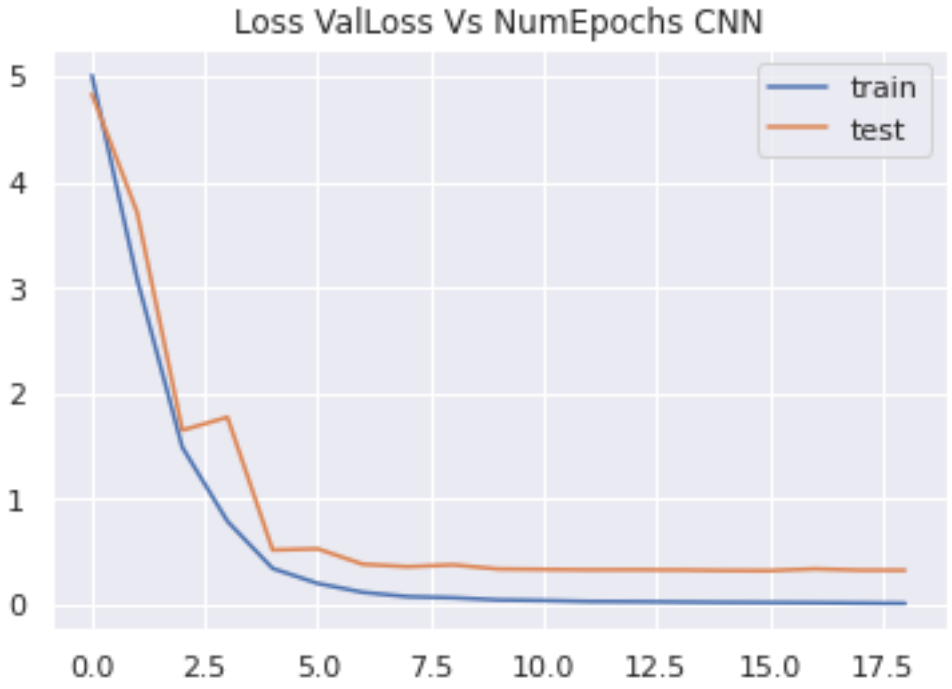
6. Plot Accuracy and Loss for Training and Validation

```
train_loss = history.history['loss']
val_loss   = history.history['val_loss']

xc = history.epoch
plt.title("Accuracy ValAccuracy Vs NumEpochs CNN")
```

```
plt.plot(xc,history.history['accuracy'], label='train')
plt.plot(xc,history.history['val_accuracy'], label='test')
plt.legend()
plt.show()

plt.figure()
plt.title("Loss ValLoss Vs NumEpochs CNN")
plt.plot(xc, train_loss,label='train')
plt.plot(xc, val_loss,label='test')
plt.legend()
plt.show
```



As seen from the graph above, both training and validation accuracy continue to increase for initial epochs and then becomes constant after reaching near 100%. This shows that we could have probably trained model for lesser number of epochs.



As seen from the graph above, both training and validation loss continue to decrease for initial epochs and then becomes constant after reaching near 0. This shows that we could have probably trained model for lesser number of epochs.

7. Evaluation

```
train_acc = classification_model.evaluate_generator(train_generator,steps = int(train_generator.samples/BATCH_SIZE))
val_acc = classification_model.evaluate_generator(validation_generator, steps = int(validation_generator.samples/BATCH_SIZE))

print(train_acc[1])
print(val_acc[1])
```

```
0.9977854490280151
0.9172061681747437
```

Thus model works really great and shows a near perfect accuracy of 99.77% for training dataset, and very high accuracy of 91.7% for validation dataset.

8. Adding result to dataframe for comparison

```
#Adding Performance metrics of InceptionResNetv2 to the list
tempResultsDf = pd.DataFrame({'Model':['InceptionResNetv2'],  'Train_Accuracy'
: train_acc[1],'Test_Accuracy': val_acc[1]})
resultsDf = pd.concat([resultsDf, tempResultsDf])
resultsDf = resultsDf[['Model', 'Train_Accuracy','Test_Accuracy']]
resultsDf
```

	Model	Train_Accuracy	Test_Accuracy
0	CNN	0.359129	0.158616
0	ResNet50	0.005290	0.005229
0	VGG16	0.938607	0.563994
0	ResNet Custom FC	0.072219	0.042829
0	InceptionResNetv2	0.997785	0.917206

Comparing Models

```
resultsDf
```

	Model	Train_Accuracy	Test_Accuracy
0	CNN	0.359129	0.158616
0	ResNet50	0.005290	0.005229
0	VGG16	0.938607	0.563994
0	ResNet Custom FC	0.072219	0.042829
0	InceptionResNetv2	0.997785	0.917206

As seen from the table above, we tried different models for this classification problem. *InceptionResNetv2* gives the best accuracy. Therefore, it is our final selected model.

```
final_model = classification_model
```

- **Pickle model for future use**

```
final_model.save('./final_model.h5')
```

Predictions

Let us use final model to predict some test car images

```
final_model = keras.models.load_model('final_model.h5')
```

```
from google.colab import files

uploaded = files.upload()
```

Saving test4.jpg to test4.jpg

```
path = 'test1.jpg'
img = cv2.imread( path )
plt.grid(False)
plt.imshow(img)
```


```
from tensorflow.keras.utils import img_to_array, load_img
import cv2
img = cv2.resize(img, (224,224),)
img.shape
```

(224, 224, 3)

```
pixels = img.astype('float32')
pixels /= 255.0
```

```
print(pixels.shape)
```
(224, 224, 3)

```
#Expanding the dimensions of the numpy array to match the dimension expected by
predict method
pixels = np.expand_dims(pixels, axis=0)
print(pixels.shape)
```

(1, 224, 224, 3)

```
prediction = final_model.predict(pixels)
prediction = np.argmax(prediction, axis = 1)
print(prediction)
```
[135]

```
predicted_label = car_names[car_names['Class'] == prediction[0]]
print(predicted_label)
```

	CarLabel	Class
134	Hyundai Elantra Sedan 2007	135

Thus our model is able to make correct prediction.

Display Few records:

Data Preprocessing & Exploratory Data Analysis

The original dataset defined a 'class' as the combination of make, model, and year. This yields 196 individual and unique classes. An example of one of these classes is shown below (Type is a subdivision in the Model level). The class levels were parsed into the components.

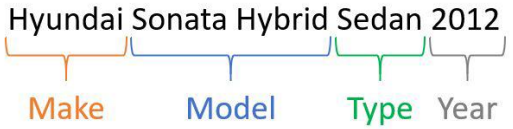


Figure 1 - Example class and separable characteristics.

The following table provides specific descriptive statistics from the entire original dataset. Due to the way that the image of this dataset was created, a thorough Exploratory Data Analysis of the original class distributions was highly desired.

									Color Channels
Type	Integer	String	String	String	String	Integer	Integer	Integer	Integer
Uniques	196	196	49	177	13	16	Several	Several	3
Mean	-	-	-	-	-	2009.56	308	573	-
Std Dev	-	-	-	-	-	4.43	214	375	-

Table 1 - Descriptive Statistics for the Stanford Cars Dataset.

The dataset contained no missing values, so no imputations or data removal was required. Class labels were split to explore the unique labels in Make, Model, Type and Year. The string-formatted labels were split by a space, then categorized into Make, Model, Type and Year levels. Automating this was challenging, since vehicle make and model strings vary widely in length and character type. Due to the lack of domain knowledge, there may exist miscategorized class information; however, this extraction of class label levels were performed to the best of our abilities. Because of the high number of classes, the levels of class labels were analyzed with the hopes of reducing the total class number. Initially the class labels were analyzed by human eyes. Our group believed that there were no multiple Make+Model yearly labels after the first examination. However, this was in fact not correct,

proven by the Model vs Year scatterplot per Make graph demonstrated in this link: https://rshinoha.shinyapps.io/jitter_plot_of_car_makes_across_years_1990-2012/.

While the initial expectations were to create models that classifies 196 classes of car models found in the dataset, this appeared to be not feasible in terms of computational power of most team members' devices. In addition, the authors stated in their research that vehicles with

similar visual features were merged into the same Year level. In order to reduce cost and achieve more accurate class labels, the Year level was dropped. There were a few duplicate Make+Model class labels due to this class engineering. These duplicate classes were simply fused together. This Year removal only resulted in a slight decrease in total class number, from 196 to 189.

Although the source data provided pre-split training and test sets, the 50/50 partitioning was not ideal for the purposes of this study. Therefore, the two sets of data were concatenated in order to re-generate a custom training-testing split. Due to the class imbalance issue shown below, random undersampling without replacement was used to create the final training-testing split. A random 80/20 split was performed on the concatenated Make+Model data, where the 80 was the under-sampled training set and the 20 was the imbalanced testing set (for the state-of-the-art CNN models, 70/30 was used). The testing set was kept imbalanced in order to preserve the original data integrity. This would also provide accurate true testing performance due to the maintained data integrity. Undersampling rather than oversampling (or SMOTE) was utilized because of the restrictions in terms of computing power and time costs.

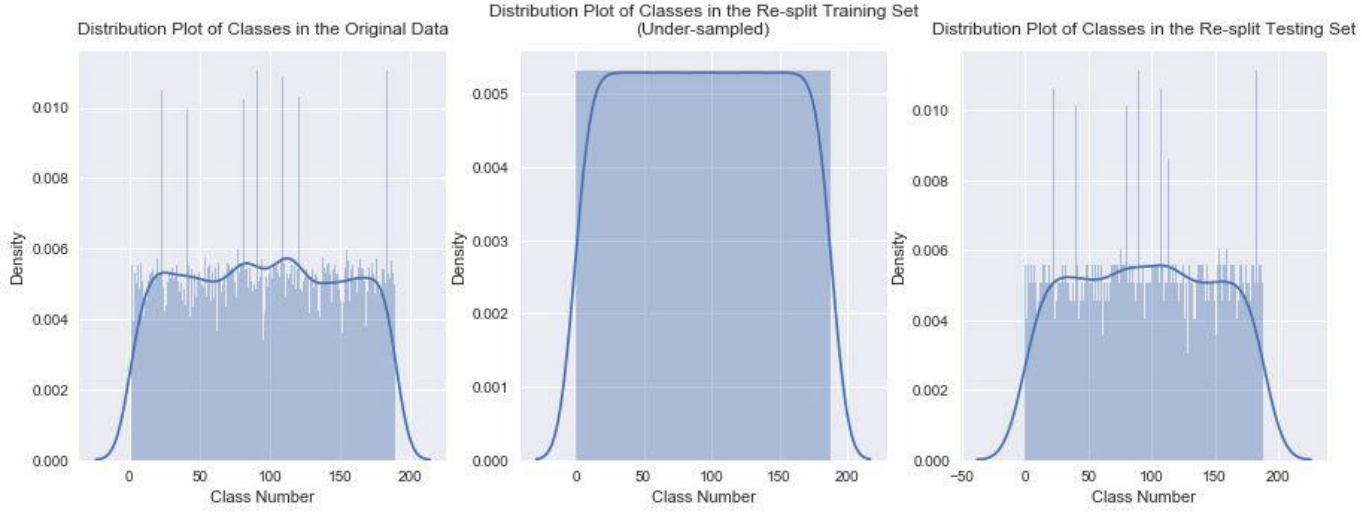


Figure 2 - Class distribution graphs comparing the original and under-sampled training/test data using the Make+Model class labels.

The Stanford Cars Dataset had images that had no apparent order or structure. There is high variability in how the photographs were taken. The angles, elevations, and rotations of the target vehicle vary widely between images, resulting in a more complex classification problem.

Preprocessing steps were dependent both on the modeling method and the available resources. The original dataset contained a collection of images with varying heights, widths, and color channels. In order to simplify problem and model complexity, images were normalized and resized to a particular height and width deemed suitable for model construction. For non-deep-learning models, images were downsized to 100x50x3. Grayscale

was also applied to the images, producing size of 100x50. For deep-learning models, an image size of 224x224x3 was selected as the standard input size. Due to the computation heavy nature of image data, all building and analysis of models were executed on AWS EC2. Depending on the computation need, the instance size was determined. From the exploratory phase, many class imbalances were revealed. Having identify those imbalances allowed to better understand the dataset and the obtained results and errors.

Methods

XGBoost

Decision tree based ensemble learners are some of the most generalizable algorithms for classification tasks. Methods like Boosted Trees and Random Forests consistently perform well in many applications including image classification, of which boosted trees such as XGBoost performing better in Caruna and Niculescu-Mizil's paper. Based on these observations, we attempted to use XGBoost as our decision tree-based ensemble learner model for this particular task. All model inputs were converted to 100x50 pixel grayscale images. While initial runs of XGBoost showed promise, performing hyperparameter tuning proved to be too computationally expensive despite using large EC2 instance (m5.2xlarge instance: vCPU: 8, memory: 32 GiB, dedicated EBS bandwidth: up to 3,500 Mbps, network performance: up to 10 Gbps); frequent memory issues and weak internet connection prevented any hyperparameter tuning iterations to run to completion. The issues arose due to the nature of boosting ensemble methods. Boosting utilizes weight systems in order to focus on difficult instances during its sequential training. Certain weak learners are prioritized during the prediction using weight systems as well. This ultimately caused memory errors, since the EC2 instance could only handle so much variables during individual runs. On the other hand, bagging simply utilizes parallel training and averaging during the prediction phase, which reduces a lot of computing cost. Therefore, we focused on using random forests as our decision tree-based ensemble learner using the same 100x50 grayscale images.

Random Forests

While better than XGBoost, the sklearn Random Forest Classifier struggled with memory issues. In order to minimize data and time loss, we used sklearn's RandomizedSearchCV() function to run a wide range of hyperparameters using 5-fold cross-validations. RandomizedSearchCV() chooses a random combination of hyperparameters based on a given set of ranges of hyperparameters. *Table 2* shows the initial hyperparameters for RandomizedSearchCV(). To further prevent data and time loss, we ran single RandomizedSearchCV() per run using different seed values.

With four running m5.2xlarge instances, we were able to run approximately 70 sets of hyperparameters. However, a subset of hyperparameters failed to run to completion due to memory issues, which suggests certain hyperparameter combinations were excluded from further analysis. *Figure 13 (Appendix)* shows a scatterplot matrix comparing the different hyperparameters with respect to the mean test scores of each 5-fold cross-validation. Based on this scatterplot matrix, we narrowed the hyperparameter ranges to use in a grid search.

Like RandomizedSearchCV(), we ran each hyperparameter combination of GridSearchCV (5-folds cross-validation) in separate lines of code in an attempt to minimize data loss. *Figure*

14 (Appendix) shows a scatterplot matrix comparing the mean test scores of 5-fold cross-validations with respect to the different max_depth and min_samples_split hyperparameters. Based on the observations of the scatterplot matrix, we chose the best hyperparameters for the final model as shown in the "Best Value" column of *Table 2*.

Parameters	RandomizedSearchCV()	GridSearchCV	Best Value
n_estimators	200, 400, 600, 800, 1000, 1200, 1400	800	800
max_features	'sqrt'	'sqrt'	'sqrt'
max_depth	10, 20, 30, 40, 50, 60, 70, 80, 90, 100	30, 40, 50, 60, 70	60
min_samples_split	2, 5, 10	2, 3, 4, 5	3
min_samples_leaf	1, 2, 4	1	1
bootstrap	True, False	False	False

Table 2 - Random Forest Hyperparameters

Support Vector Machines

Support Vector Machines (SVM) is a supervised learning technique and was popular in image classification before the adoption of deep learning. In this project, we used it as one of our traditional (non-deep-learning) algorithms. Several different feature extraction methods were applied during modeling. Before tuning the parameters, the default SVM model was used, where C = 1, gamma = 'auto', and kernel = 'rbf' to fit with different feature extraction algorithms to find an optimal combination. The testing set performance is shown below, where 'edge' means Canny edge detection, 'Gaussian' means adaptive gaussian thresholding and 'Mean' indicates adaptive mean thresholding.

Preconditioning	Accuracy	Recall	Precision	F1 Score	Time (s)
Grayscale+edge	0.0050	0.0050	0.0000	0.0001	9.03
Grayscale+edge+PCA	0.0030	0.0030	0.0023	0.0024	11.77
RGB+PCA	0.0675	0.0675	0.0735	0.0971	279.82
GrayScale+PCA	0.0599	0.0599	0.0727	0.0490	241.62
GrayScale+Gaussian+PCA	0.0358	0.0358	0.0454	0.0293	566.93
GrayScale+Mean+PCA	0.0363	0.0363	0.0468	0.0294	536.77

Table 3 - SVM Test Set Performances

As shown in *Table 3*, PCA on RGB images had the best performance. However, according to our experiment during modeling, sometimes model performance of PCA on grayscale images

could surpass that of PCA on RGB images. To eliminate the randomness, we did parameter tuning on both models, run training and testing splits 30 times, took the mean of the performance results and made comparisons. After parameter tuning, C = 10, gamma = 0.0001, kernel = 'rbf' for 'GrayScale+PCA', and C = 100, gamma = 0.00001, kernel = 'rbf' for 'RGB+PCA'. The mean of performance results is shown below:

Tuned Parameters	Accuracy	Recall	Precision	F1 Score
GrayScale + PCA	0.0657	0.0657	0.0777	0.0581
RGB + PCA	0.0856	0.0856	0.0849	0.0768

Table 4 - Effect of Image Color Scheme on Model Performance

As *Table 4* shows, color input images have a slight advantage over grayscale versions. Using this approach, experiments were performed on different sizes of image data. The graph of performance results is shown below. Since accuracy and recall are the same, only accuracy was plotted.

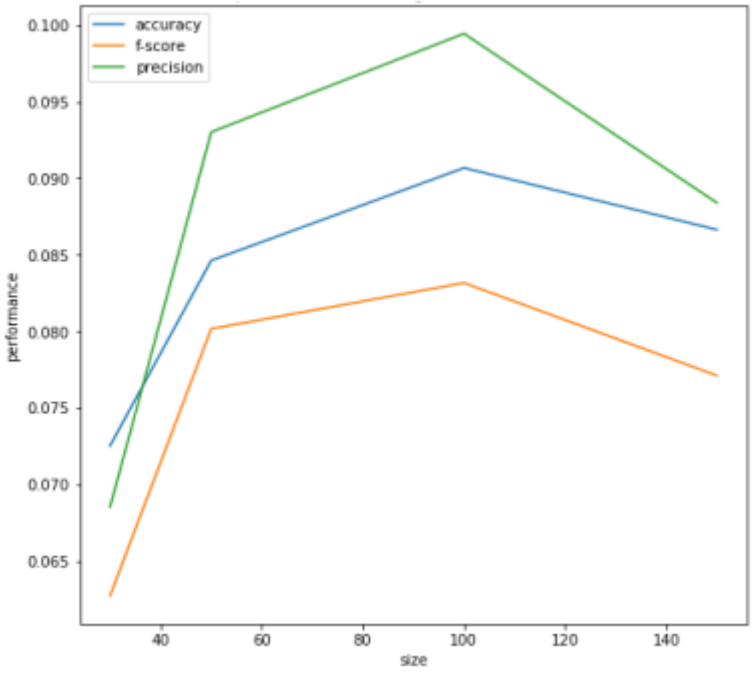


Figure 3 - Model performance on image data of different sizes.

Since downsizing is not a lossless process, it is not surprising that inputs as small as 50x25 and 30x15 resulted in worse performance. However, when the image size was decreased from 150x75 to 100x50, the performance was improved. This abnormal pattern reveals the weak capability of SVM in our non-structured image data.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are an extension of neural nets adapted for image classification. The model is composed of layers of filters and compressions in sequence. Filter weights are adjusted like those in the neural net for similar purposes. With an appropriate amount of training samples and adequately-tuned parameters, a CNN will automate the feature selection process and accurately classify the input.

Prior to model building, a concept called One-Cycle-Policy was researched in order to facilitate the training process (as discussed in https://arxiv.org/abs/1803.09820). This policy essentially utilizes the cyclic approach of learning rate and weight decay parameters to speed up the learning phase of the convolutional neural network models. Learning rate can be interpreted as the amount that the weights are updated during training (backpropagation). A small learning rate can result in slower training process but globally optimal set of weights and a large learning rate can result in faster training process but sub-optimal set of weights. Weight decay is a regularization term/parameter that penalizes big weights during the weight update phase. The best outcome is yielded when the learning rate that results in just before the global loss minima is selected. This can be determined after examining the Loss vs. Log Learning Rate plot. Different weight decay parameter can be graphed in this plot for analysis. Our team experimented with this policy on our models in order to demonstrate its effectiveness.

The initial custom convolutional neural networks with 1~3 ConvNet layers (conv-conv-pool) with relu activations followed by dense(512), dropout and dense(189) fully connected layers performed poorly. All of these exploratory custom CNNs were trained after tuning the learning rate and fit on regular Stanford Cars images (under-sampled). The 3rd model (3 ConvNet layers) had the highest validation accuracy of about 5% post learning rate tuning. These initial set of models illustrated that simple CNN models are not able to perform well on this complex and non-structured car image data. Due to this, the outputs and graphs related to these are not documented here. From here, our team started building deeper custom CNNs and exploring other state-of-the-art architectures such as Xception, ResNet34 and ResNet50.

The following custom architecture was inspired by the VGG16 CNN model, though it is more compact due to hardware limitations. Using several layers of filters allows the model to consider a large space of possible weights to explore. *Figure 4* shows the training history of the model. Validation accuracy (blue) approaches 30% while the top-5 validation accuracy (black) levels off near 50%. These metrics, while not wholly descriptive, provide valuable feedback during an iterative model development process.

In addition to a homegrown CNN architecture, state-of-the-art CNN models such as Xception, ResNet34 and ResNet50 were explored and analyzed. The state-of-the-art CNN models come pre-trained on ImageNet data and come only with the weights for the Convolution layers. These models are trained on large visual databases, which are often used

in object recognition. Due to this fact, the first step of implementing these models are adding custom fully connected (FC) layers. Once the top FC layers are added, the model weights are frozen in order to train only the top layers. This process makes these pre-trained models suitable for any image data of interest. After the top layers are trained for a small number of epochs, model is unfrozen to begin full learning. For all state-of-the-art models, this initial setup was performed. In addition, an Adam optimizer was used since it performs faster than other optimizers.

Image Augmentation

Due to the curse of dimensionality, a large number of images are required to produce satisfactory classifiers. In lieu of collecting more samples manually, the CNN models were trained with synthetic images. These images are slightly distorted (mirrored, rotated, and sheared to some extent) in order to help the model better generalize the input space. Consequently, this helps combat overfitting since the model is exposed to more than just the 'real' training images. Test images were left unmodified in order to evaluate models with actual image data.

Custom CNN

Filter Layers

Convolution layers contain trainable kernels (sets of weights) which transform patches of the input image. These are essentially different kinds of image filters that the model adjusts during training in order to optimize the chosen loss function. A well-trained model contains an array of filters that effectively discriminate between different classes of images. In the Keras package, convolutional layers have several user-defined inputs such as stride length and kernel dimensions. Altering these two parameters in particular can influence model learning and performance, but were not explored in depth here. The default kernel size of 3x3 was used for each models' filter layers. Max-pooling is used after each block of convolution layers. The pooling process summarizes a window of pixels into a single output value. In the case of max-pooling, the window will be condensed into the highest value. This helps emphasize and maintain areas of large contrast from layer to layer.

Deep Neural Network

During model development, larger and deeper neural networks tended to have an edge over smaller ones. The neural network model selected attempts to make use of the available resources by using a relatively large number of nodes in the hidden layers. The best performing combination of optimization algorithm and loss function proved to be Adagrad and categorical cross-entropy, respectively. The Adagrad algorithm is a gradient-descent method which has an adaptive learning rate and can compensate for class imbalances.

Dropout is utilized to increase problem-space generalization by forcing the model to consider more than the output of just a few nodes. A rate of 0.25 provided a good balance between

training time and validation accuracy. Linear activations are used between layers. This identity mapping preserves the variance of the data. The final layer applies a Softmax transformation to the final summations. This transforms the output into a set of likelihood values, the highest of which belongs to the best guess.

Development

The Keras package readily allows for two metrics to be monitored during training- accuracy and top-5 accuracy. The top-5 accuracy metric provides another level of feedback during training, since accuracy alone does not give credit to close guesses.

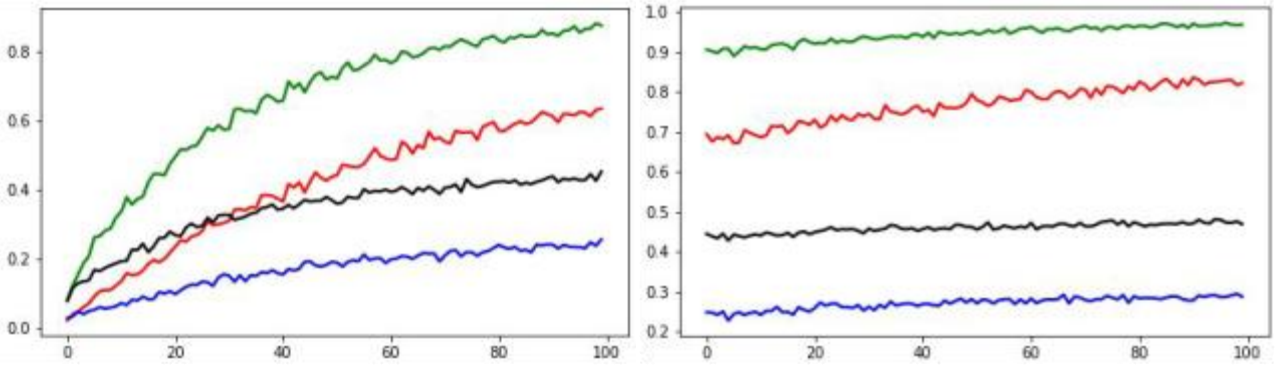


Figure 4 - Custom CNN training history. The blue and black lines show, respectively, the validation and top-5 validation accuracies as the model trains. Eventually, a cumulative penalty parameter in the Adagrad algorithm makes performance plateau by dampening the learning rate.

Xception

The fully connected layers of the Xception model had GlobalAveragePooling2D, Dense(2048) with relu, Dropout(0.4) and Dense(189) with a softmax layer. With a slight augmentation process performed on the training images using ImageDataGenerator in keras and the learning rate tuning, the Xception model was able to achieve a little above 30% validation accuracy. Due to the setup process, the learning rate was tuned twice, once before and once after the training of the top-layers. There was a big RAM and kernel error issue, so the model was saved after 10 epochs of training and loaded again each time. This model resulted in extremely long computing time. Although the Xception model could have improved with further training, this model was discontinued because of the computation limitations. The outputs from the Xception model can be found in the Miscellaneous Graphs section. To combat the issue of lack of resources, models that are available in the fastai module were the highest priority, since the fastai module offers the One-Cycle-Policy. This led to the ResNet34 model.

ResNet34

The initial setup for the ResNet34 model was done utilizing the fastai module. The fastai module does the freezing and adding of the FC layers automatically when loading the

pre-trained models. These FC layers were larger than the one added to the Xception model. They had multiple BatchNorms, Dropouts and Dense layers with linear and relu activations. The

One-Cycle-Policy learning rate tuning process was performed, in addition to implementing the weight decay parameter for the ResNet34 model. This model was able to achieve 0.68 validation accuracy after just 7 epochs during the top-layer training. This proved the effectiveness of the One-Cycle-Policy in terms of time costs. Excluding the top layer training phase, this model was trained for a total of 100 epochs. The loss vs. learning rate graphs used in determining the optimal learning rate for the One-Cycle-Policy is demonstrated below. Both the top layer training and full training used the One-Cycle-Policy.

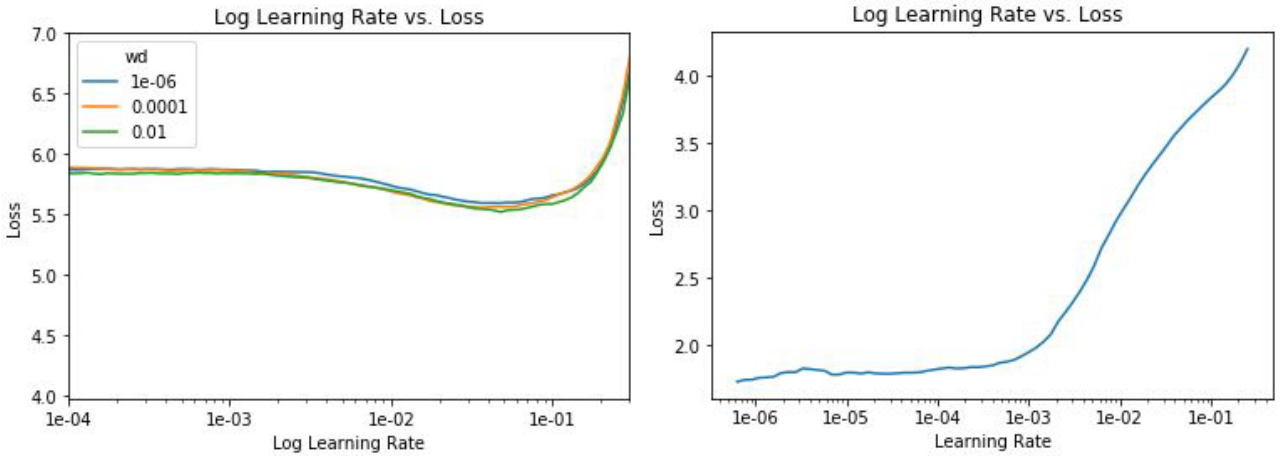
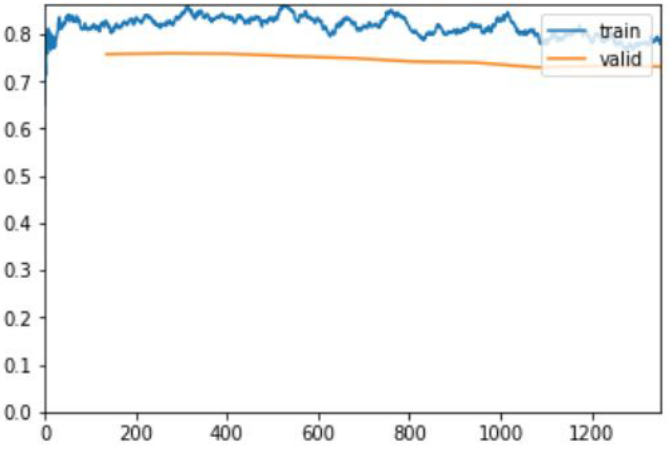


Figure 5 - ResNet34 Log Learning Rate vs Loss graph before and after training the FC layers.

Some augmentations and transformations were applied on the image data for the ResNet34 model. This helps prevent overfitting, since this process creates variability in the inputs. The parameters used for the augmentations are do_flip = True, flip_vert, False, max_rotate = 90. A preview of some of the images are illustrated in *Figure 6*.

epoch	train_loss	valid_loss	accuracy	time
0	0.828971	0.757601	0.789338	47:57
1	0.823475	0.759355	0.789338	47:17
2	0.834208	0.758517	0.786248	47:51
3	0.851897	0.753168	0.785733	47:02
4	0.821245	0.749253	0.790883	47:52
5	0.804125	0.741682	0.790111	46:54
6	0.818743	0.739885	0.791141	48:05
7	0.801630	0.729439	0.791398	47:00
8	0.798832	0.732448	0.792944	47:57
9	0.780111	0.731119	0.793974	46:53




```
The precision score is 0.8100481150660742
The recall score is 0.7939737316507854
The fscore score is 0.7940082428478917
```

Figure 6 - ResNet34 final (10th) model performance.

After 100 epochs of training, the ResNet34 model was able to obtain 0.78 training loss and 0.73 testing loss, as illustrated in the table and graph above. Compared to the previous loss vs. iteration graphs that are attached under the Miscellaneous Graphs section at the end of this paper, the gap between the training loss and the testing loss was decreasing during the last 10 epochs of training. The large gap between training and testing loss could be interpreted as the model not being able to perform similarly on the training and the testing set. Small gap between the training and the testing loss could be interpreted as the model's ability to perform similarly regardless of training/testing set it uses. In ResNet34's case, the training loss was much higher than the testing loss during the beginning stages of the learning phase. This meant that the model had room for improvement by learning more from the training data. After 100 epochs of fitting, both the training and testing loss were improved (decreased). In addition, the gap between the two loss lines also narrowed down. The training of ResNet34 was successful, since it was able to bring the training and testing loss down and increased the model accuracy. The cross and/or the divergence of the training and testing loss lines indicates overfitting. This did not occur during the 100 epochs of model fitting. There is potential room for improvement in terms of loss and accuracy scores. However, since 10 epochs take about 8 hours and there was less than 0.005 increase in model validation accuracy during the last 10 epochs, the training process was stopped after 100 epochs to save computation and time costs.

```
[('Bentley Continental GT Coupe', 'Bentley Continental Flying Spur Sedan', 19),
 ('GMC Savana Van', 'Chevrolet Express Van', 15),
 ('Dodge Sprinter Cargo Van', 'Mercedes-Benz Sprinter Van', 6)]
```

Figure 7 - ResNet34 most confused images.

Although the confusion matrix in the Appendix is not very readable, it is clear that the model performance is high based on the diagonal actual vs. predicted line. There are two spots that are not a part of this diagonal line, which are the first two lines of the most confused images summary above. According to the most confused images, the misclassified labels are in fact very similar in shape. The most confused (19 counts) image labels both belonged to the Bentley Make family. The second most confused (15 counts) image labels were both vans. This indicates that the misclassified images are not by chance, but actually due to the similarities in shape. The heatmaps in the Appendix portray images with high loss scores and highlight areas that the ResNet34 model believes that are important. This could be used to analyze why the model is classifying incorrectly more in depth. Although direct interpretation is not obtainable, it could be concluded that more variety of training images and further training will be beneficial by the heatmap images that has no important areas. The same could

be said by the heatmap images that do not focus on the Make logo.

ResNet50

Since ResNet34 using One-Cycle-Policy was extremely successful in classifying

the Make+Model level, ResNet50 was explored after development of the ResNet34 model. There

are a few differences between the ResNet34 and ResNet50 architectures. The main differences are the numbers and sizes of the convolution layers. As the name suggests, ResNet50 has a deeper architecture than ResNet34. As shown in the ResNet family comparison graph below, only the conv2.x ~ conv5.x are different. These layers have smaller window sizes and more Conv filters and layers, which can be interpreted as focusing on smaller pixel windows more heavily.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 5 - ResNet-family architectures.

Some augmentations and transformations were applied on the image data for the ResNet50 model. Same augmentation parameters were used as the ResNet34 model, which are do_flip = True, flip_vert, False, max_rotate = 90. However, the resize method was changed to 'squish' rather than the default 'crop'. The default 'crop' method crops the images to obtain the predefined image shape, but the 'squish' method squishes the images to the predefined image shape. This could include some important information but also lose some due to the loss of the original proportions. A preview of some of the images are portrayed in *Figure 16 (Appendix)*.

Similar to the ResNet34 model, the initial setup for the ResNet50 model was done utilizing the fastai module. The FC layers had multiple BatchNorms, Dropouts and Dense layers with linear and relu activations. Both learning rate and weight decay were used for the ResNet50 model as well. The loss vs. learning rate graphs used in determining the optimal learning rate for the One-Cycle-Policy is demonstrated below. Both the top layer training and full training used the One-Cycle-Policy. During the top-layer training using the One-Cycle-Policy, the ResNet50 model was able to achieve 0.57 validation loss after 10 epochs of top-layer training. This is extremely significant, since ResNet34 was only able to obtain 0.73 after 10 epochs of top-layer and 100 epochs of full training. Although the top-layer training of ResNet50 illustrated powerful performance increase in small number of epochs, the full training did not result in as much improvement. During the full training of 10 epochs, the validation loss ultimately decreased to 0.42. The interesting part about ResNet50 was that the

validation loss and the training loss already crossed each other during the top-layer training. Then the 10 epochs of full training resulted in the validation loss and the training loss diverging from each other. By the end of the full training, the validation loss seemed to be not improving even though the training loss was going down. This is an apparent sign of overfitting, hence the training was ended. The short full training means that the pre-trained set of weights already work well on the Stanford Cars data, so further training (or weight updates) is not necessary.

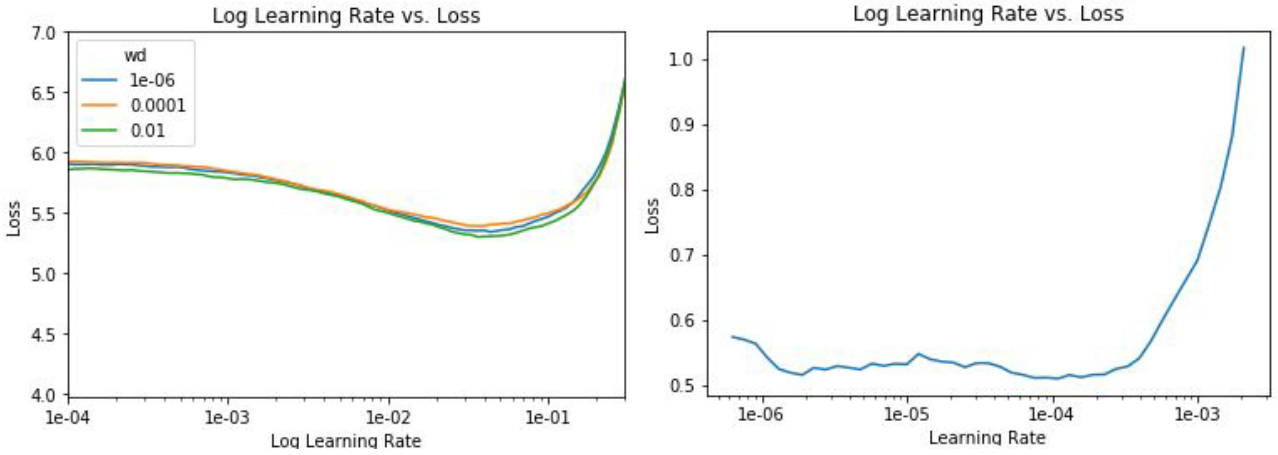
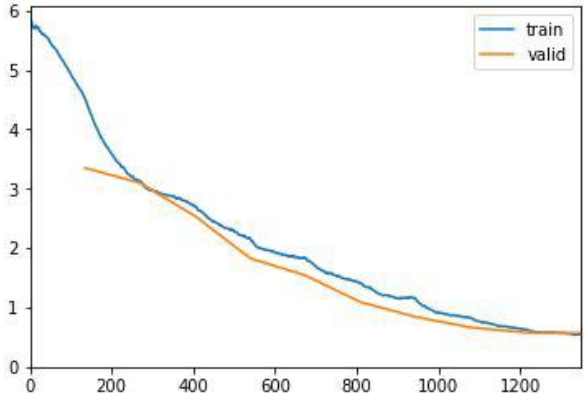
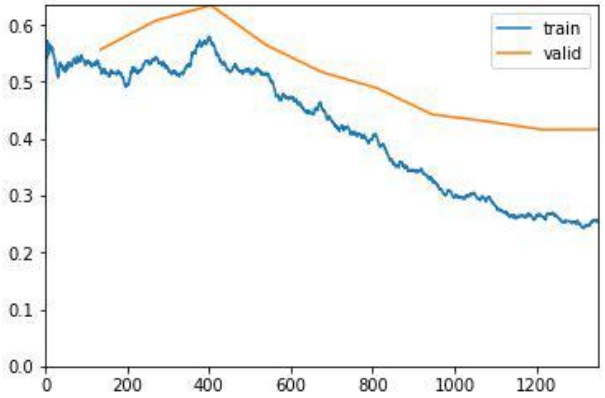


Figure 8 - ResNet50 Log Learning Rate vs. Loss graph before and after training the FC layers.

epoch	train_loss	valid_loss	accuracy	time
0	4.534760	3.349324	0.257533	1:04:04
1	3.134347	3.097307	0.278651	1:10:56
2	2.708412	2.537358	0.355138	1:10:16
3	2.165936	1.827119	0.507597	1:10:22
4	1.841726	1.539408	0.581252	1:11:16
5	1.419916	1.089868	0.696369	1:10:57
6	1.149049	0.841191	0.757147	1:11:00
7	0.824681	0.663156	0.814319	1:11:03
8	0.629282	0.578202	0.843678	1:10:17
9	0.547708	0.570551	0.843420	1:10:21

epoch	train_loss	valid_loss	accuracy	time
0	0.525315	0.556713	0.849086	57:36
1	0.543948	0.607696	0.831316	1:28:48
2	0.572520	0.634493	0.819212	1:28:36
3	0.514778	0.564616	0.837754	1:27:35
4	0.460670	0.517179	0.857842	1:28:57
5	0.404159	0.489222	0.858099	1:28:48
6	0.327171	0.442721	0.873036	1:27:26
7	0.294380	0.430297	0.879990	1:28:53
8	0.262758	0.415979	0.881020	1:29:26
9	0.254132	0.416259	0.880247	1:28:58





```
The precision score is 0.891808046663706
The recall score is 0.880247231522019
The fscore score is 0.8815954450168534
```

Figure 9 - ResNet50 top-layer and full training model performance.

ResNet50 resulted in significant improvement in precision, recall and F1-score. All three of these performance metrics were above 0.88. As shown in the confusion matrix in the Appendix, the diagonal actual vs. predicted line is slightly more apparent than the ResNet34 model. According to the most confused images summary below, the overall misclassified count has decreased. In fact, the Bentley family misclassification count went down to 8 from 19. All the misclassified labels are still very similar in shape, which indicates that the errors made by the model is due to the similarities of vehicle shapes. In *Figure 20*, the images that have no important areas disappeared. This means that the ResNet50 model is picking up important features from vehicles, due to the deeper architecture and the learning. However, there are still heatmap images that do not focus on the Make logo. It could be appropriate to conclude that even though ResNet50 model performs better, more variety of training images and further training will still be beneficial.

```
[('GMC Savana Van', 'Chevrolet Express Van', 16),
 ('Bentley Continental GT Coupe', 'Bentley Continental Flying Spur Sedan', 8),
 ('Audi S5 Coupe', 'Audi A5 Coupe', 6),
 ('Chevrolet Express Van', 'Chevrolet Express Cargo Van', 6)]
```

Figure 10: ResNet50 most confused images.

Analysis of Results

Several metrics were used to compare model performance. In addition to overall accuracy, weighted precision, recall, and F1-score were all calculated. The F1-score describes a balance of precision and recall. The weighted F1-score for a multi-class problem takes into account the class imbalance, and generates a fairer performance score. These metrics are provided in *Table 6*.

Model	F1 Score	Accuracy	Precision	Recall	Image Size
SVM	0.08	0.09	0.08	0.09	50 x 100
Random Forest	0.06	0.06	0.07	0.06	50 x 100
CNN (Custom)	0.29	0.29	0.33	0.27	224 x 224
CNN (ResNet34)	0.79	0.79	0.81	0.79	224 x 224
CNN (ResNet50)	0.88	0.88	0.89	0.88	224 x 224

Table 6 - Summary of model performances on Make & Model classification

SVM and Random Forest models yielded poor performance despite rigorous hyperparameter tuning. Both models produced similar performances that were better than random guessing but were considerably inferior to all CNN models. It should be noted that different models had access to different amounts of resources. This is reflected in the input image size column in *Table 6*. ResNet50 was by far the best performing model that had the highest weighted precision, recall and F1-score. Moreover, it took way less time cost than the ResNet34 model, since the pre-trained weights were extremely suitable for the Stanford Cars dataset. Considering both the cost and performance, it is obvious that ResNet50 is the top performing model.

Validation & Testing

To further investigate model performance on the real world data, we collected thirty additional images that were not included in the Stanford Cars dataset to gauge the real-world performance of the ResNet50 model. It is clear that more images in this ultimate testing phase would be ideal; however, these images were gathered and reviewed by the team manually. Due to the significant cost of the gathering of real-world data requires, only thirty images were used. In the original Stanford Cars dataset, the images were collected mainly from Google and Flickr, and went through a vehicle detection phase, which provided the coordinates of bounding boxes of the object of interest. This preprocessing phase removed a lot of the background noise. However, the final testing images were images that included the background, which explains the decrease in performance measures.

Model	F1 Score	Accuracy	Precision	Recall	Image Size
CNN (ResNet50)	0.73	0.73	0.73	0.73	224 x 224

Table 7 - ResNet50 model validation performance.

In addition to the background noise which is the key suspect, there were other factors that played a role in this decrease. The final testing set was collected from Google, which resulted in having access to newer yearly model images. The ResNet50 model was trained on the Stanford Cars data that was developed years ago. This means that the model was learning from older models of cars. These slight differences between the old and the new generations of vehicles definitely had a negative impact on the model performance. The performance gap may also have been led by the small-sample bias and the unstructured nature of these additional images.

Actual	Predicted
Audi A5 Coupe	Chevrolet TrailBlazer SS
Audi S6 Sedan	Mitsubishi Lancer Sedan
Dodge Challenger SRT8	Chevrolet Camaro Convertible
Ferrari 458 Italia Coupe	Geo Metro Convertible
Hyundai Sonata Sedan	Ford Fiesta Sedan
Land Rover Range Rover SUV	Cadillac SRX SUV
Mercedes-Benz SL-Class Coupe	BMW 1 Series Convertible
Rolls-Royce Phantom Drophead Coupe Convertible	Rolls-Royce Phantom Sedan

Table 8 - Misclassified Validation Images

The ResNet50 model performed fairly well on the validation set. In the misclassified validation images table above, most of the misclassified images are in fact very similar in overall shape. As discussed in the ResNet50 section, this means that the model is not making mistakes by random guess but due to the presence of certain objects in each image.

Discussion & Conclusions

Results

The team had expected CNNs would perform better than non-deep-learning models. However, we were surprised with the gap in performance between the two groups of algorithms, with state-of-the-art CNN architectures achieving great success despite the difficult task of classifying vehicles. The performances achieved in the ResNets provide strong evidence in promoting further research in creating a more complex car classification model to deploy to be used in a car identification app in the future.

The SVM and Random Forest non-CNN models performed considerably worse than CNN models. This was likely due to the nature of the models, inherently being unable to retain positional information about the pixels in a 3D recognition task. While these algorithms have shown success in simpler 2D image classification tasks such as the MNIST digits classification, the images found in the Stanford Cars dataset proved to be too complex for the two algorithms. Comparing the two non-CNN models, SVMs had slightly higher performance metrics (F1 score = 0.08, etc). This may be due to SVM's flexibility in the shape of their decision boundaries in higher dimensions, especially with complex kernels such as RBF. Given a more rigorous round of feature extractions more applicable to image classifications, we may have been able to improve the performance for SVMs. However, it would not reach the performances of CNN models, making further research on improving performance unnecessary. Similar problems arose for Random Forests as well, but with even larger consequences (F1 score = 0.06, etc). This was likely due to how decision tree decision boundaries are straight lines perpendicular to the axis of the feature, which limits the flexibility of decision boundaries. Furthermore, the combination of the large feature set and how Random Forests take subsets of features most likely exacerbated the situation. Considering how the edges found on cars are likely the most important features in classifying the different makes and models, the relevant classification features most likely occupy a very small portion of the feature set. Therefore, it is likely that many of the decision trees in the RF did not contain the important pixels of the images, making it difficult for Random Forest to create a relevant model.

Ethics and Social Acceptance

As image-recognition becomes more ubiquitous, it raises many questions about the ethical use of such capability. As recently employed in China, image-recognition methods can be used to spy on the population, giving rise to many questioning regarding privacy and it being violated. By not being regulated and not protecting personal information, such use of data lies on the border of ethics, the ownership of the data and what is socially acceptable. Regarding the dataset the authors used, one solution would have been to blur out license plates and faces during data collection for instance.

Conclusion

Among all machine learning methods used, CNNs are the models providing the best vehicle recognition model with precision, recall and f-score all above 0.88 on the validation set and 0.73 on additional testing images. This approach should be further explored by adding a substantial number of images to train the models as well as adding more classes (car model) in order to make this model more relevant from the application development business perspective.

Appendix

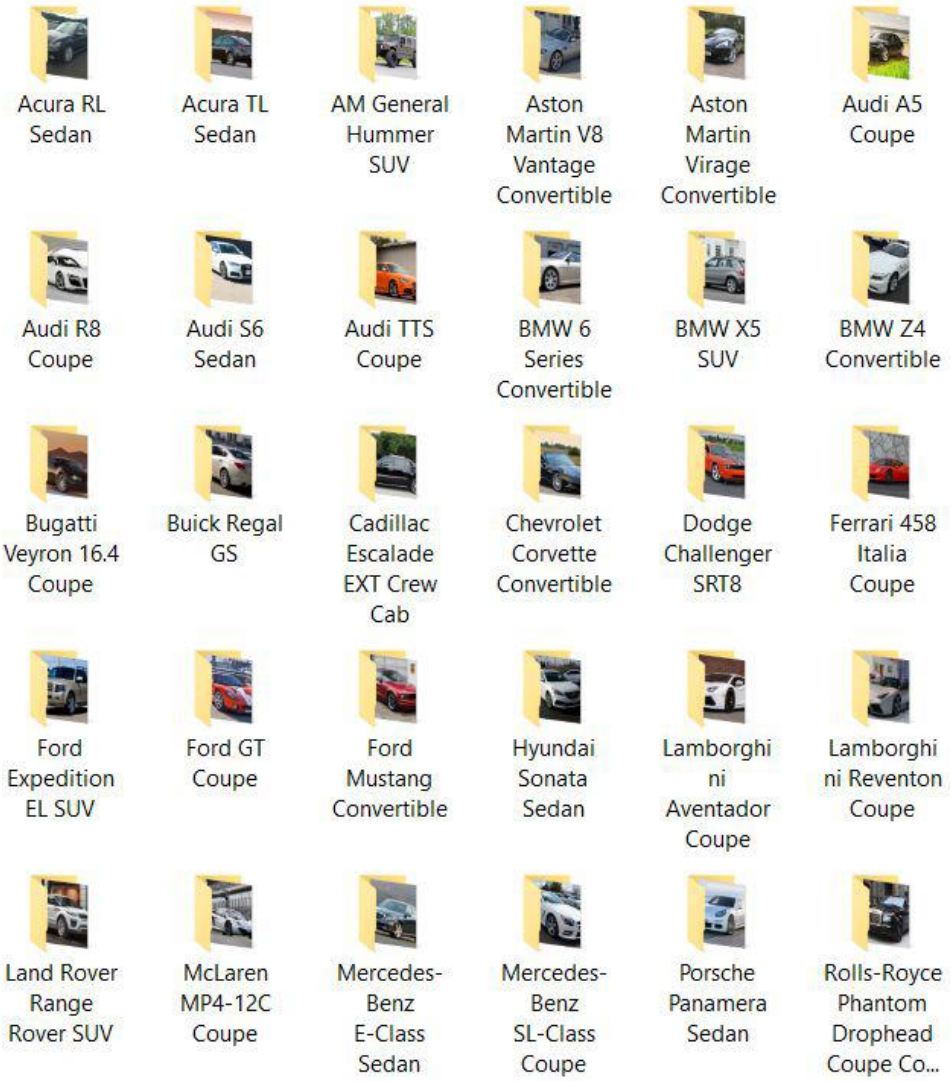


Figure 11 - Additional 30 images withheld for final model validation.

Figure 12 - Sample images from each Make+Model class.

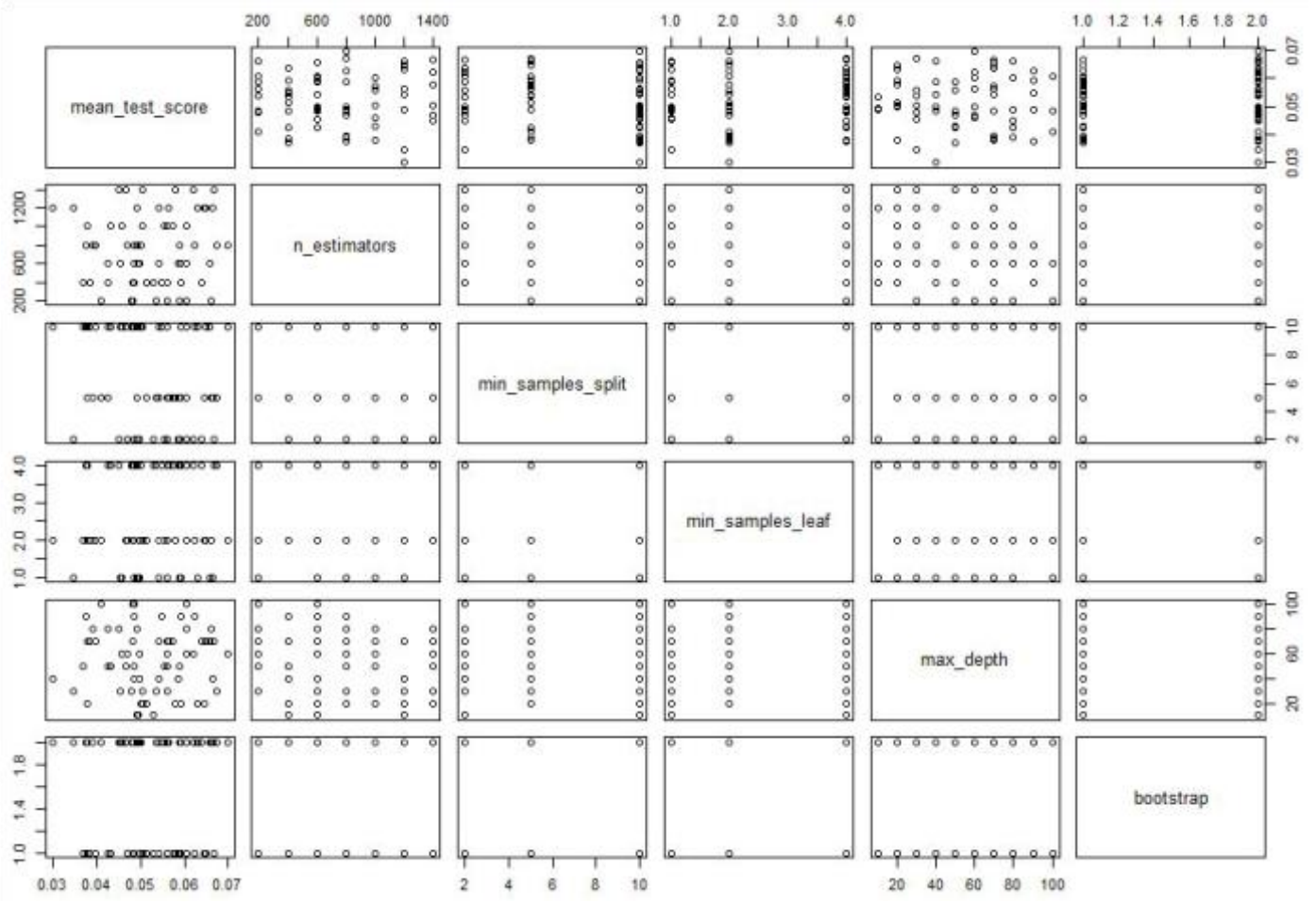


Figure 13 - Scatterplot matrix of mean test scores of 5-fold cross-validations with respect to RandomizedSearchCV() hyperparameters.

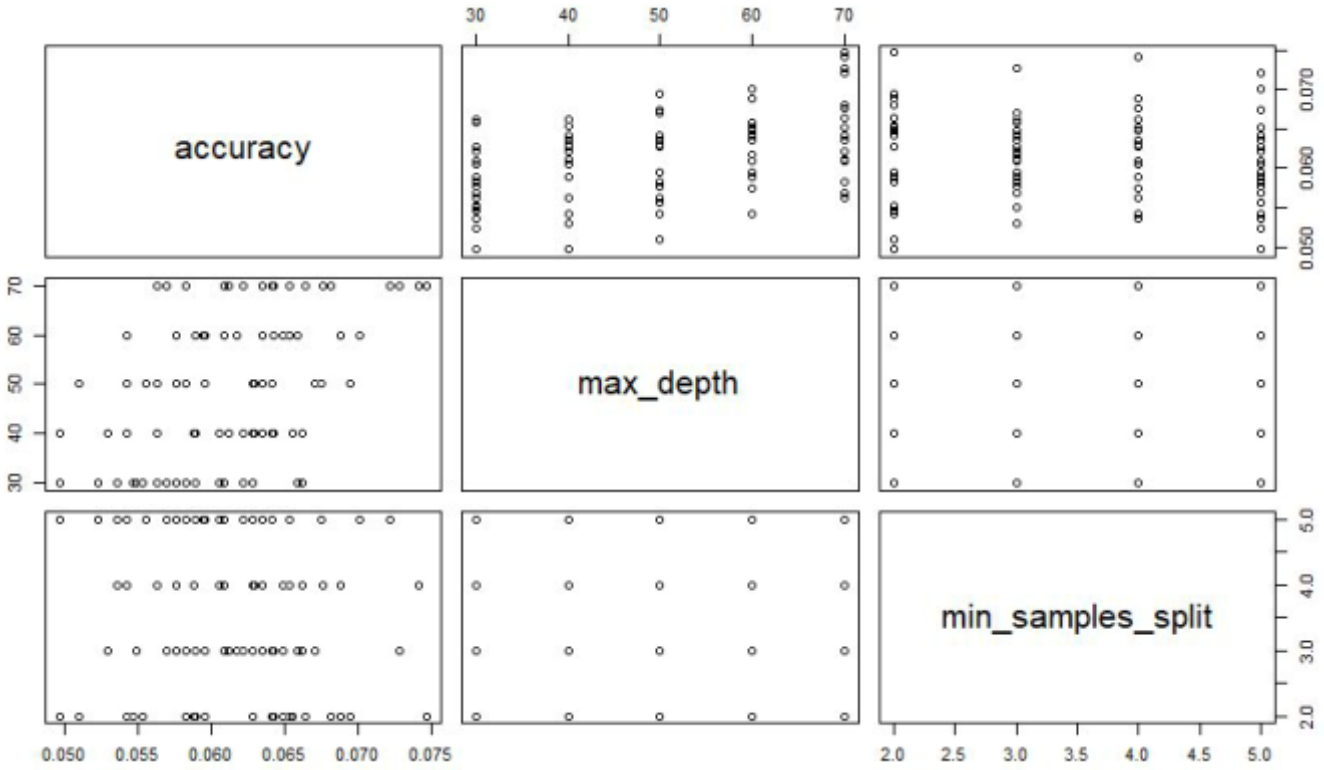


Figure 14 - Scatterplot matrix of mean test scores (accuracy) of 5-fold cross-validations with respect to the two grid search hyperparameters.

	Layer	Output Shape
1	Input Image	224 x 224
64	2D Convolutions (3x3) MaxPooling (2x2)	111 x 111 x 64
128	2D Convolutions (3x3) MaxPooling (2x2)	54 x 54 x 128
256	2D Convolutions (3x3) MaxPooling (2x2)	26 x 26 x 256
512	2D Convolutions (3x3) MaxPooling (2x2)	12 x 12 x 512
64	2D Convolutions (3x3) MaxPooling (2x2)	5 x 5 x 64
1	Flatten	1600
4608	Dense (Linear) Dropout (0.25)	4608
4608	Dense (Linear) Dropout (0.25)	4608
189	Dense (Softmax)	189

Figure 15 - Custom CNN model structure.

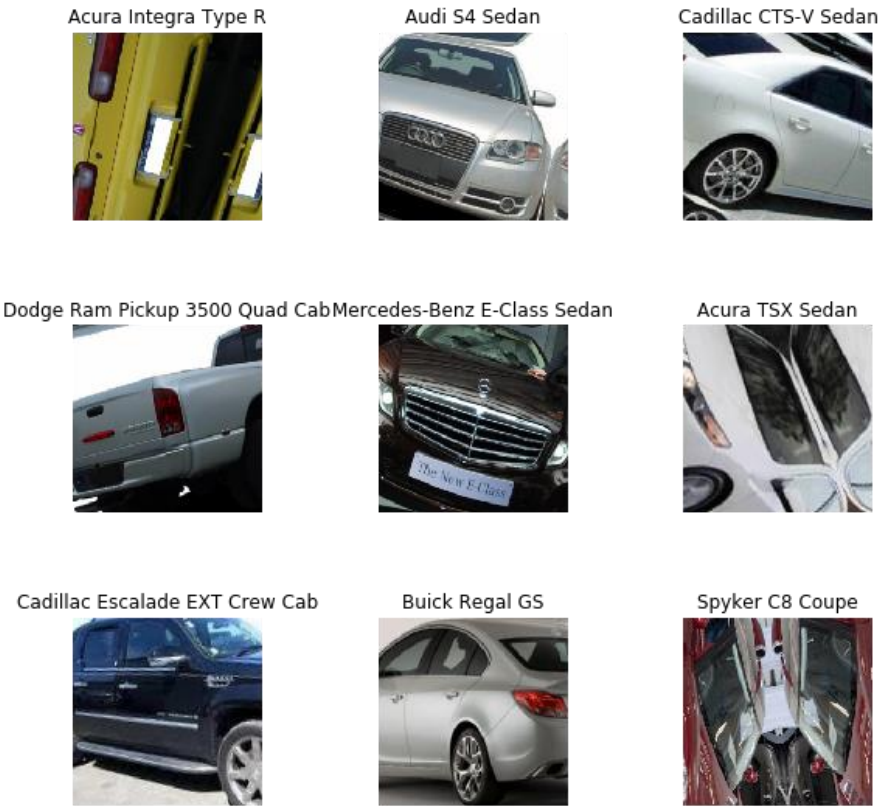
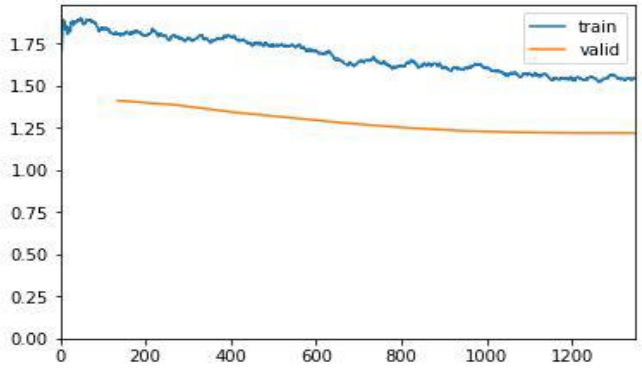


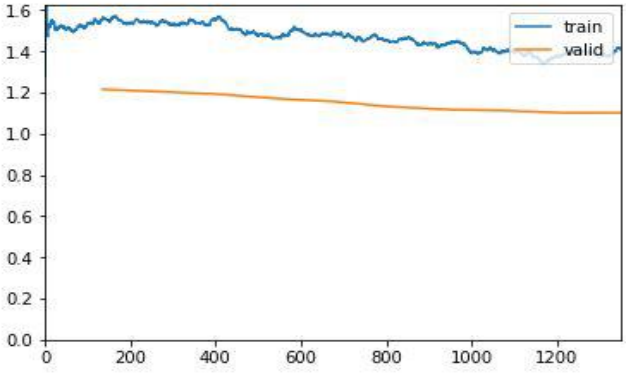
Figure 16 - Preview of the ResNet34 training images with augmentation/transformation.

Figure 17 - Preview of the ResNet50 training images with augmentation/transformation.

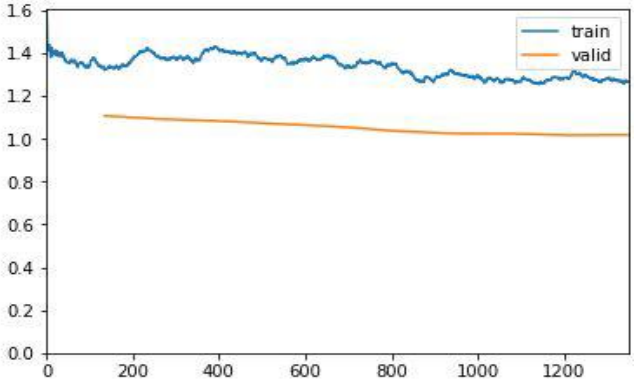
epoch	train_loss	valid_loss	accuracy	time
0	1.808390	1.410663	0.632243	49:07
1	1.790068	1.385608	0.638939	50:36
2	1.781615	1.342545	0.654133	48:59
3	1.734212	1.309546	0.659284	49:18
4	1.619155	1.276190	0.664692	48:52
5	1.619910	1.250240	0.673191	49:28
6	1.594307	1.232061	0.673963	49:26
7	1.550314	1.222225	0.679372	49:52
8	1.552493	1.218435	0.675766	49:52
9	1.542795	1.218609	0.678342	49:56



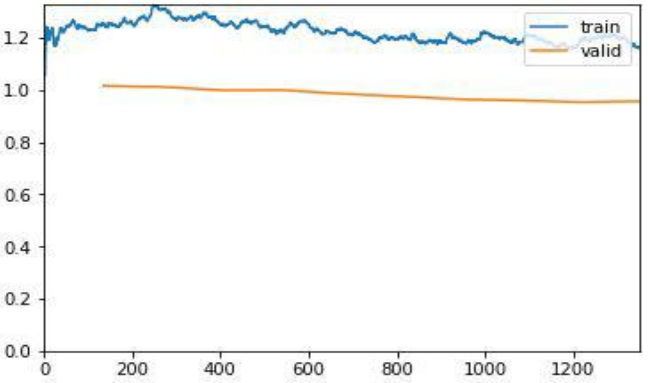
epoch	train_loss	valid_loss	accuracy	time
0	1.557081	1.216002	0.675251	14:58
1	1.545179	1.204483	0.681689	14:48
2	1.564081	1.192634	0.684522	14:51
3	1.465772	1.171620	0.688643	26:22
4	1.483997	1.156955	0.693021	47:19
5	1.449640	1.131565	0.704095	48:21
6	1.440592	1.117790	0.703580	46:59
7	1.400102	1.112542	0.704095	49:10
8	1.384001	1.101945	0.708215	47:01
9	1.410389	1.102118	0.707443	48:00



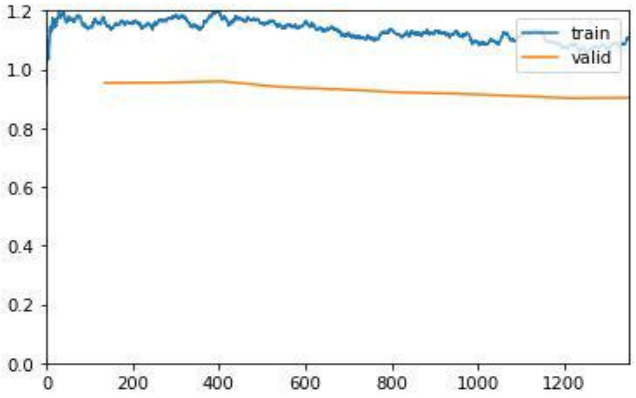
epoch	train_loss	valid_loss	accuracy	time
0	1.321524	1.107562	0.706155	37:22
1	1.378396	1.091762	0.708215	46:38
2	1.416011	1.083155	0.712078	48:37
3	1.387095	1.069836	0.714654	47:32
4	1.357621	1.057059	0.720577	47:41
5	1.350469	1.036730	0.725728	48:44
6	1.307412	1.025207	0.729591	46:43
7	1.274621	1.024214	0.730363	48:47
8	1.290659	1.016932	0.732423	46:05
9	1.266117	1.018915	0.731393	48:49



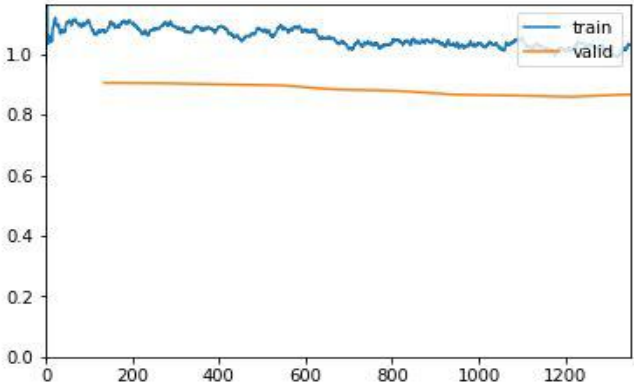
epoch	train_loss	valid_loss	accuracy	time
0	1.250747	1.015860	0.732938	47:46
1	1.307564	1.010971	0.730621	47:41
2	1.249875	0.998471	0.736029	47:18
3	1.230348	0.999026	0.734484	48:15
4	1.232139	0.984812	0.734226	46:46
5	1.207032	0.974829	0.742982	48:10
6	1.189793	0.963933	0.742725	46:56
7	1.181206	0.958936	0.743497	47:41
8	1.183900	0.953023	0.745815	47:05
9	1.159541	0.956321	0.744785	47:41



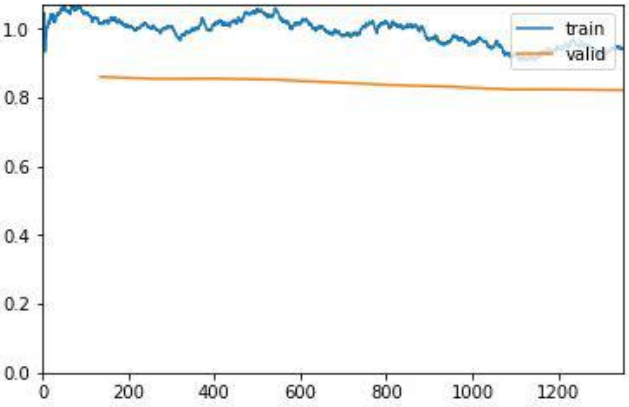
epoch	train_loss	valid_loss	accuracy	time
0	1.168369	0.954173	0.746073	47:08
1	1.166362	0.954944	0.741952	47:26
2	1.184143	0.959483	0.745043	47:08
3	1.153834	0.940191	0.748648	47:51
4	1.137994	0.932312	0.747360	47:12
5	1.123653	0.922264	0.751481	47:53
6	1.118429	0.917528	0.754829	47:37
7	1.097875	0.909929	0.753799	47:31
8	1.086187	0.902132	0.755086	48:07
9	1.109474	0.903756	0.754314	46:52



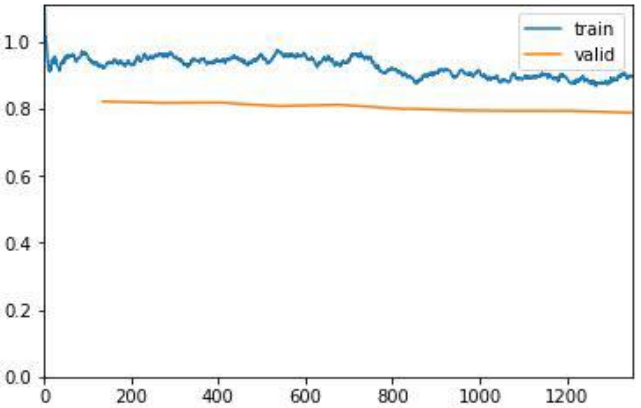
epoch	train_loss	valid_loss	accuracy	time
0	1.074703	0.904523	0.757404	14:52
1	1.097129	0.903359	0.757404	14:50
2	1.092081	0.899894	0.756374	14:58
3	1.088593	0.896544	0.756631	24:34
4	1.035675	0.882772	0.761267	47:48
5	1.048380	0.878450	0.761267	47:56
6	1.029200	0.866074	0.764873	49:15
7	1.038603	0.862865	0.762297	47:42
8	1.012528	0.858699	0.764873	47:58
9	1.029502	0.866741	0.763842	49:20



epoch	train_loss	valid_loss	accuracy	time
0	1.014698	0.859986	0.764873	43:32
1	0.995521	0.854635	0.768736	48:41
2	1.018612	0.855251	0.766675	46:24
3	1.044091	0.851986	0.768478	48:38
4	0.986598	0.844310	0.773114	46:36
5	1.006567	0.836668	0.772341	48:20
6	0.961997	0.831541	0.773371	46:45
7	0.927437	0.823986	0.776719	48:14
8	0.946067	0.823031	0.777749	47:13
9	0.940417	0.821491	0.776204	48:06

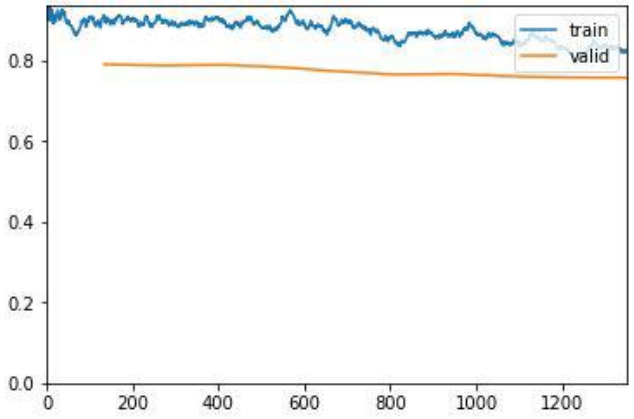


epoch	train_loss	valid_loss	accuracy	time
0	0.925195	0.822583	0.777492	47:00
1	0.960630	0.818020	0.777749	48:39
2	0.940481	0.818945	0.778779	46:50
3	0.971666	0.808305	0.777234	48:22
4	0.932030	0.812829	0.778264	46:55
5	0.917777	0.801429	0.779294	48:02
6	0.895212	0.796258	0.783672	47:22
7	0.905308	0.794433	0.785733	47:41
8	0.881446	0.794623	0.783157	48:14
9	0.898755	0.788267	0.786505	47:41



epoch	train_loss	valid_loss	accuracy	time
0	0.907949	0.790865	0.783672	42:28
1	0.899129	0.787927	0.781097	47:36
2	0.896608	0.789776	0.780325	48:17
3	0.888053	0.784256	0.786763	46:54
4	0.900395	0.773797	0.784960	48:36
5	0.851556	0.765870	0.787020	46:53
6	0.869491	0.767473	0.788308	48:45
7	0.841771	0.761102	0.789853	46:46
8	0.824650	0.758777	0.788566	48:22
9	0.825978	0.757898	0.789596	47:00

epoch	train_loss	valid_loss	accuracy	time
0	0.828971	0.757601	0.789338	47:57
1	0.823475	0.759355	0.789338	47:17
2	0.834208	0.758517	0.786248	47:51
3	0.851897	0.753168	0.785733	47:02
4	0.821245	0.749253	0.790883	47:52
5	0.804125	0.741682	0.790111	46:54
6	0.818743	0.739885	0.791141	48:05
7	0.801630	0.729439	0.791398	47:00
8	0.798832	0.732448	0.792944	47:57
9	0.780111	0.731119	0.793974	46:53



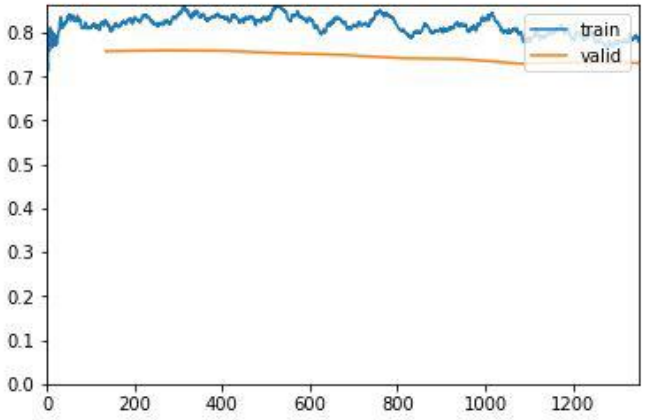


Figure 18 - ResNet34 Training Outputs (10 phases of full training, 10 epochs each, excluding the top layer training).

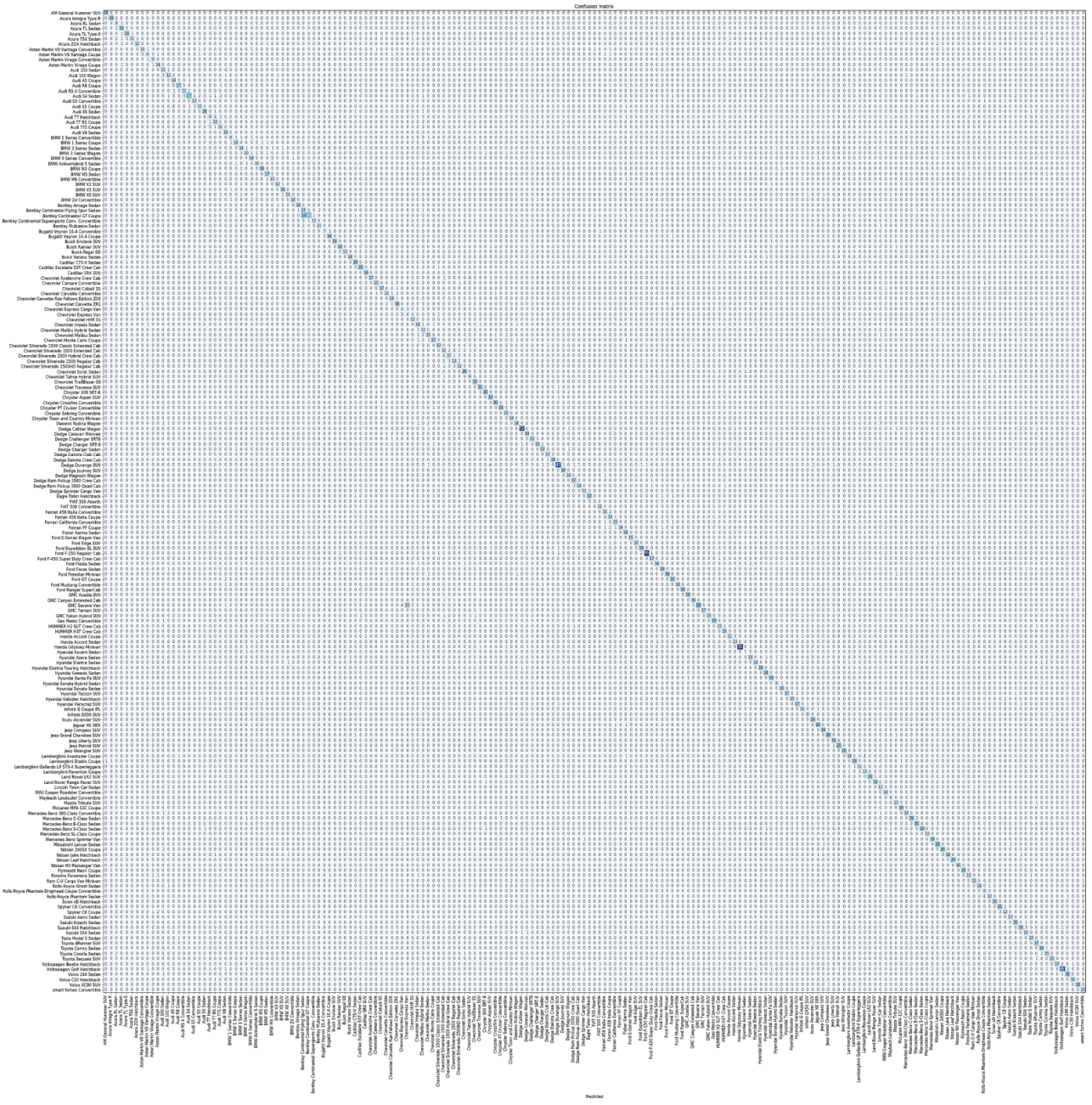


Figure 19 - ResNet34 Confusion Matrix.

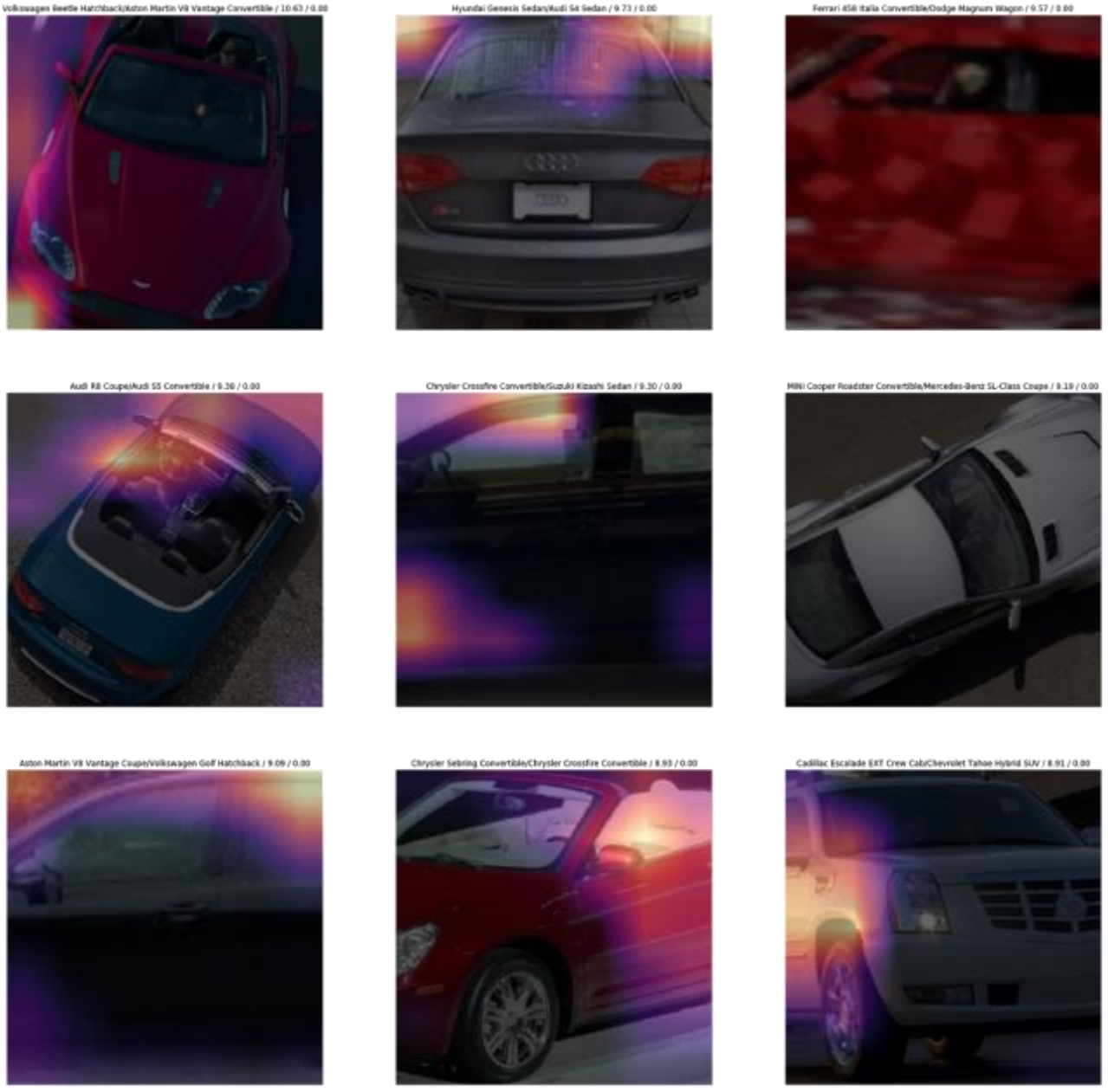


Figure 20 - Heatmap of areas of focus for the top-loss images by the ResNet34 model.

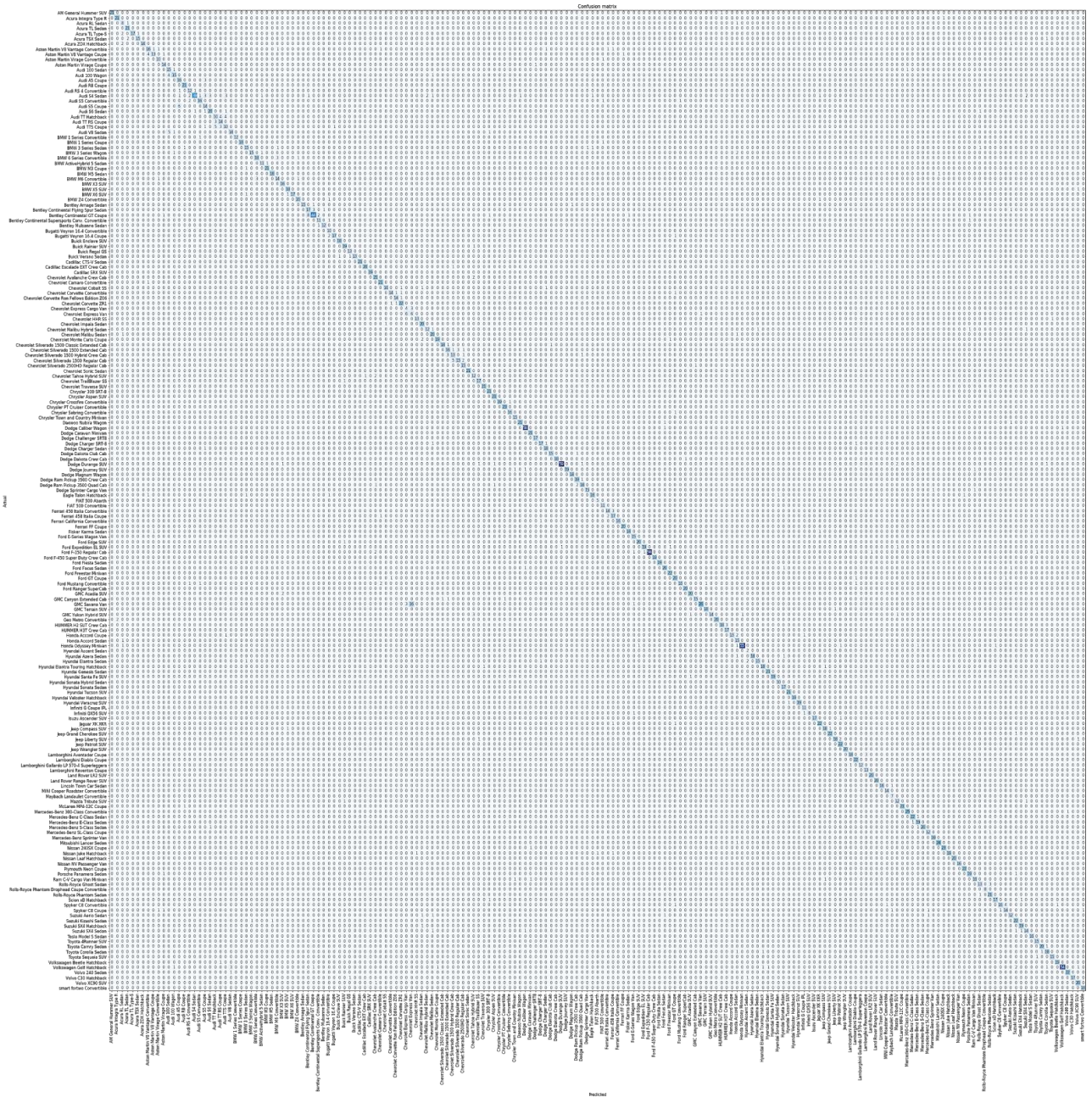


Figure 21 - ResNet50 Confusion Matrix.

Figure 22 - Heatmap of areas of focus for the top-loss images by the ResNet50 model.