CNN's

Luis Garduno

Brief Business Understanding

`About STL-10`

Inspired by the CIFAR-10 dataset, STL-10 is a dataset containing a combination of images (gathered from ImageNet) of animals and transportation objects.

Within the dataset there are 6 animal & 4 transportation object classes:

- **Animals** : bird, cat, deer, dog, horse, monkey
- **Transportation Objects** : airplane, car, ship, truck

The dataset contains 3 folders that will be used at specified times:

- **Train** : 5000 images used to train the algorithm
- **Test** : 8000 images used to test an algorithm (800 images per class)
- **Unlabeled** : 100,000 unlabeled image files

Aside from having not having identical classes, another difference between the datasets, is that the images in STL-10 are 3x's the resolution of CIFAR-10's images (96x96 versus 32x32).

STL-10 is specifically an image recognition dataset. The dataset is intended to be used for developing unsupervised feature learning, deep learning, self-taught algorithms. That being said, the primary prediction task is to determine the type of animal or transportation object found in each of the pictures in the Unlabeled folder. Something that should be noted about the "Unlabeled" folder, aside from it containing the the classes mentioned above, it additionally includes other types of animals (bears, rabbits, etc.) and transportation objects[trains, buses, etc.).

`Measuring Success`

One reason this data is important is if trained correctly & the prediction task is achieved, third parties that use image captcha's for their websites, networks, etc. could use this data as a way to visualize how captcha's can be bypassed by **unsupervised** feature learning, which essentially defeats the purpose of having a captcha test.

In order for this data to be of use to third parties using captcha's, I believe the prediction algorithm will have to render at least an 80% accuracy. The reason it isn't 90% is because if the prediction algorithm selects a wrong image, or doesn't recognize an image, often times captcha test's will let you get away with about 2 or less errors.

- Choose & explain what metric(s) I will use to evaluate my algorithm’s performance.
- I should give a detailed argument for why this (these) metric(s) are appropriate on your data.
- That is, why is the metric appropriate for the task (e.g., in terms of the business case for the task).
- (accuracy is RARELY the best evaluation metric to use. Think deeply about an appropriate measure of performance.)

Dataset : STL-10 Kaggle Dataset

Question Of Interest : Identify the type of animal or transportation object shown in the picture

---

1.1 Data Preparation

In [1]:

```
import glob
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from PIL import Image
from tkinter import Tcl
from skimage.io import imshow
from sklearn import metrics as mt
from sklearn import preprocessing

%matplotlib inline

le = preprocessing.LabelEncoder()

# Load list of labels/classes
with open('../data/stl10_binary/class_names.txt', 'r') as f:
    labels = [line.rstrip() for line in f]
```

```
print("=========== Classes ===========")
for i in range(len(labels)): print(i,":", labels[i])
```

```
=========== Classes ===========
0 : airplane
1 : bird
2 : car
3 : cat
4 : deer
5 : dog
6 : horse
7 : monkey
8 : ship
9 : truck
```

** Note : The dataset is found to be composed of 3 primary folders: Train, Test, & Unlabeled. Because of this, the way we load in our data, will also be the way we 'split' our data into training and testing sets.

In [2]:

```
# Training Dataset ###############################################################################
""" Reading in the TRAINING dataset (sorted by filename) into a numpy array """
file_train = list(Tcl().call('lsort', '-dict', glob.glob('../data/STL10/Train/*.png')))
rgb_train_matrix = np.array([np.array(Image.open(file)) for file in file_train])
_, h, w, c = rgb_train_matrix.shape

""" Gather label's indices # (Training | y)"""
with open('../data/stl10_binary/train_y.bin', 'rb') as f:
    yTrain_temp = np.fromfile(f, dtype=np.uint8)
    le.fit(yTrain_temp)
    y_train = le.transform(yTrain_temp)

""" Create new numpy array w/ re-colored greyscaled images | TRAINING """
greyscale_train_matrix = np.array([np.array(Image.open(file).convert("L")) for file in file_train])
print

df_train = pd.DataFrame({'' : ['# of Samples','# of Features','Image Resolution','# of Channels','Image
Size']})
df_train['Original Data']  = [_, h*w*c, '{} x {}'.format(h,w), c, str(h*w*c) + 'px\'s']
df_train['Greyscale Data']  = [_, h*w, '{} x {}'.format(h,w), 1, str(h*w) + 'px\'s']

rgb_train_vec = rgb_train_matrix.reshape((_,h*w*c))
greyscale_train_vec = greyscale_train_matrix.reshape((_,h*w))
X_train = greyscale_train_vec

print("Training Data")
df_train
```

```
Training Data
```

Out[2]:

		Original Data	Greyscale Data
0	# of Samples	5000	5000
1	# of Features	27648	9216
2	Image Resolution	96 x 96	96 x 96
3	# of Channels	3	1
4	Image Size	27648px's	9216px's

In [3]:

```
# Testing Dataset ###############################################################################
""" Reading in the TESTING dataset (sorted by filename) into a numpy array """
file_test = list(Tcl().call('lsort', '-dict', glob.glob('../data/STL10/Test/*.png')))
rgb_test_matrix = np.array([np.array(Image.open(file)) for file in file_test])
_, h, w, c = rgb_test_matrix.shape

""" Gather label's indices # (Testing | y)"""
with open('../data/stl10_binary/test_y.bin', 'rb') as f:
    yTest_temp = np.fromfile(f, dtype=np.uint8)
    le.fit(yTest_temp)
    y_test = le.transform(yTest_temp)

""" Create new numpy array w/ re-colored greyscaled images | TESTING """
greyscale_test_matrix = np.array([np.array(Image.open(file).convert("L")) for file in file_test])

df_test = pd.DataFrame({'' : ['# of Samples','# of Features','Image Resolution','# of Channels','Image Size']})
df_test['Original Data']  = [_, h*w*c, '{} x {}'.format(h,w), c, str(h*w*c) + 'px\'s']
df_test['Greyscale Data']  = [_, h*w, '{} x {}'.format(h,w), 1, str(h*w) + 'px\'s']

rgb_test_vec = rgb_test_matrix.reshape((_,h*w*c))
greyscale_test_vec = greyscale_test_matrix.reshape((_,h*w))
```

```
X_test = greyscale_test_vec

print("Test Data")
df_test
```

```
Test Data
```

Out[3]:

		Original Data	Greyscale Data
0	# of Samples	8000	8000
1	# of Features	27648	9216
2	Image Resolution	96 x 96	96 x 96
3	# of Channels	3	1
4	Image Size	27648px's	9216px's

In [4]:

```
print("\n[Train] Original Matrix Shape   :  Before", rgb_train_matrix.shape, "----->  After",
rgb_train_vec.shape)
print("[Train] Greyscale Matrix Shape  :  Before", greyscale_train_matrix.shape, "   ----->
After",X_train.shape)

print()

print("\n[Test] Original Matrix Shape   :  Before", rgb_test_matrix.shape, "----->  After", rgb_test_vec.shape)
print("[Test] Greyscale Matrix Shape  :  Before", greyscale_test_matrix.shape, "   ----->  After",
X_test.shape)
```

```
[Train] Original Matrix Shape   :  Before (5000, 96, 96, 3) ----->  After (5000, 27648)
[Train] Greyscale Matrix Shape  :  Before (5000, 96, 96)    ----->  After (5000, 9216)


[Test] Original Matrix Shape   :  Before (8000, 96, 96, 3) ----->  After (8000, 27648)
[Test] Greyscale Matrix Shape  :  Before (8000, 96, 96)    ----->  After (8000, 9216)
```

We begin by reading the dataset's (train & test) into numpy array's, but because they contain colored images, it would be optimal to turn these array's into only containing grayscale values so we are able to compute faster.

Then after doing so, the shape's of the original matrices and grayscale's are outputted to display the initial dimensions. To the right of these shapes, are the concatenated versions of those matrices.

The output of *cell 10* is created to better understand the differences between the 4 matrices, two containing *Original* color pictures, and the other two containing *Greyscaled* colored images. Here is where we notice the large distance between the image sizes for each matrix. Notice how each *Greyscaled* picture is 3 times smaller than the *Original* color pictures.

1.2 Visualizing Images

In [5]:

```
plt.style.use('ggplot')

def plot_gallery(images, titles, h, w, flag, n_row=3, n_col=6):
    plt.figure(figsize=(1.7 * n_col, 2.3 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=1.0, hspace=.25)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h,w)), cmap=plt.cm.gray)
        if flag:
            plt.title(labels[titles[i]], size=12)
        if not flag:
            plt.title(titles[i], size=12)
        plt.xticks(());  plt.yticks(())

plot_gallery(X_train, y_train, h, w, True)
```

Here we visualize 18 images within the training (greyscale) numpy array. This function will be helpful later on to output certain images given a certain certain array.

1.3 Creating Training & Test Data

Although the dataset that is being used (STL-10 : Kaggle) has already split the data into training & test set's, let's briefly see how this was done. The [original STL-10 dataset], uses 10 pre-defined folds on 500 training images, & 800 images for the test class, thus rendering out a total of 5,000 training images & 8000 test images.

The code below, heavily based on the code written by Martin Tutek, displays how the 5,000 training images are created.

In [6]:

```
# Don't run: For demonstration purposes only

from __future__ import print_function
from imageio import imsave
import os, sys, tarfile, errno, urllib

def read_labels(labels_path):
    """
    :param labels_path: path to the binary file containing labels from the STL-10 dataset
    :return: an array containing the labels
    """
    with open(labels_path, 'rb') as f:
        labels1 = np.fromfile(f, dtype=np.uint8)
        return labels1

def read_all_images(data_path):
    """
    :param data_path: the file containing the binary images from the STL-10 dataset
    :return: an array containing all the images
    """
    with open(data_path, 'rb') as f:
        # read file in uint8 chunks
        f = np.fromfile(f, dtype=np.uint8)

        # Force the data into 3x96x96 chunks, & the -1 is because of the size of the pictures heavily
        # depends on the input file so numpy is able to determine the size
        images = np.reshape(f, (-1, 3, 96, 96))

        # Transpose the images into a standard image format so that they can be display in python
        # Comment this line out if wanting to use a learning algorithm like CNN, since they like their channels
separated.
        images = np.transpose(images, (0, 3, 2, 1))
        return images

def save_images(images, labels1):
    i = 0
    for image in images:
        label = labels1[i]
```

```
        directory = '../data/img/' + str(label) + '/'
        os.makedirs(directory, exist_ok=True)
        filename = directory + str(i)
        imsave("%s.png" % filename, image, format="png")
        i += 1

# Path to the binary train file: image + labels
DATA_PATH, LABEL_PATH = '../data/stl10_binary/train_X.bin', '../data/stl10_binary/train_y.bin'

# Save new images
# save_images(read_all_images(DATA_PATH),read_labels(LABEL_PATH))
```

---

2. Modeling

2.1 Training w/ Data Expansion

In [7]:

```
# Import Keras libraries

import tensorflow.keras as keras
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import average, Reshape, Input, Add
from tensorflow.keras.layers import Conv2D, MaxPooling2D, concatenate
from tensorflow.keras.layers import SeparableConv2D, BatchNormalization
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [8]:

```
# Setup the training to use data expansion in Keras.

print("====== OLD Shapes ======")
print('Train :', X_train.shape, '\nTest  :', X_test.shape)

X_train, X_test = X_train/255.0 - 0.5, X_test/255.0 - 0.5

NUM_CLASSES = 10

# Fixing the sizes
X_train = np.expand_dims(X_train.reshape((-1, h, w)), axis=3)
X_test  = np.expand_dims(X_test.reshape((-1, h, w)),  axis=3)

# One hot encoding the output values
y_train_ohe = keras.utils.to_categorical(y_train, NUM_CLASSES)
y_test_ohe = keras.utils.to_categorical(y_test, NUM_CLASSES)

print("\n====== NEW Shapes ======")
print('Train :', X_train.shape, '\nTest  :', X_test.shape)
```

```
====== OLD Shapes ======
Train : (5000, 9216)
Test  : (8000, 9216)

====== NEW Shapes ======
Train : (5000, 96, 96, 1)
Test  : (8000, 96, 96, 1)
```

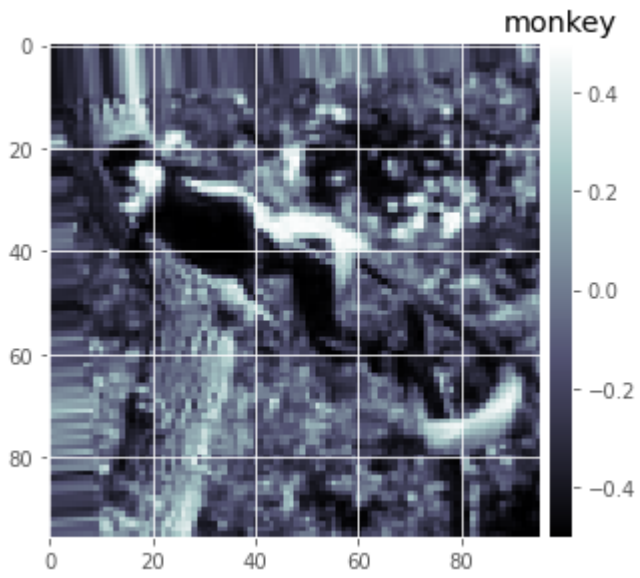
In [9]:

```
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=5,       # used, Int. Degree range for random rotations.
    width_shift_range=0.1,  # used, Float (fraction of total width). Range for random horizontal shifts.
    height_shift_range=0.1, # used, Float (fraction of total height). Range for random vertical shifts.
    shear_range=0.,         # Float. Shear Intensity (Shear angle in counter-clockwise direction as radians)
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=True,
```

```
    vertical_flip=False,
    rescale=None)

datagen.fit(X_train)

idx = 0
```

In [10]:

```
tmps = datagen.flow(X_train, y_train_ohe, batch_size=1)

for imgs in tmps:
    imshow(imgs[0].squeeze(),cmap='bone')
    plt.title(labels[np.argmax(imgs[1])])
    break
```



2.2 Exploring Convolutional Architectures

In [11]:

```
shpe = (h,w,1)
l2_lambda = 0.000001
input_holder = Input(shape=shpe)

shpe = (h,w,1)
input_holder = Input(shape=shpe)

# Graph comparing Training & Validation Vs. Accuracy & Training loss
def getChart(h):
    plt.figure(figsize=(10,4))
    plt.subplot(2,2,1)
    plt.plot(h.history['accuracy'])

    plt.ylabel('Accuracy %')
    plt.title('Training')
    plt.subplot(2,2,2)
    plt.plot(h.history['val_accuracy'])
    plt.title('Validation')

    plt.subplot(2,2,3)
    plt.plot(h.history['loss'])
    plt.ylabel('Training Loss')
    plt.xlabel('epochs')

    plt.subplot(2,2,4)
    plt.plot(h.history['val_loss'])
    plt.xlabel('epochs')

# Graph comparing Training & Validation Vs. Accuracy & Training loss
def getCharts(h1, h2, h3, h4):
    plt.figure(figsize=(10,8))
    plt.subplot(2,2,1)
    ax1 = sns.lineplot(data=h1.history['accuracy'], label='LeNet - AdaM', color='blue')
    sns.lineplot(data=h2.history['accuracy'], label='LeNet - RMSProp', color='red')
    sns.lineplot(data=h3.history['accuracy'], label='Xception - AdaM', color='green')
    sns.lineplot(data=h4.history['accuracy'], label='Xception - RMSProp ', color='darkorange')

    plt.ylabel('Accuracy %')
    plt.title('Training')
    plt.subplot(2,2,2)
    ax2 = sns.lineplot(data=h1.history['val_accuracy'], label='LeNet - AdaM', color='blue')
    sns.lineplot(data=h2.history['val_accuracy'], label='LeNet - RMSProp', color='red')
```

```
    sns.lineplot(data=h3.history['val_accuracy'], label='Xception - AdaM', color='green')
    sns.lineplot(data=h4.history['val_accuracy'], label='Xception - RMSProp', color='darkorange')
    plt.title('Validation')

    plt.subplot(2,2,3)
    ax3 = sns.lineplot(data=h1.history['loss'], label='LeNet - AdaM', color='blue')
    sns.lineplot(data=h2.history['loss'], label='LeNet - RMSProp', color='red')
    sns.lineplot(data=h3.history['loss'], label='Xception - AdaM', color='green')
    sns.lineplot(data=h4.history['loss'], label='Xception - RMSProp', color='darkorange')
    plt.ylabel('Training Loss')
    plt.xlabel('epochs')

    plt.subplot(2,2,4)
    ax4 = sns.lineplot(data=h1.history['val_loss'], label='LeNet - AdaM', color='blue')
    sns.lineplot(data=h2.history['val_loss'], label='LeNet - RMSProp', color='red')
    sns.lineplot(data=h3.history['val_loss'], label='Xception - AdaM', color='green')
    sns.lineplot(data=h4.history['val_loss'], label='Xception - RMSProp', color='darkorange')
    plt.xlabel('epochs')
```

2.2.1 Method 1: LeNet Architecture

In [12]:

```
##################################################################################
######################### LeNet Style Architecture : AdaM ##########################
##################################################################################

# Convolution Layer ================================================================
x1 = Conv2D(filters=32, input_shape=shpe,
            kernel_size=(3,3),kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(input_holder)
x1 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x1)

# Convolution Layer ================================================================
x1 = Conv2D(filters=32,
            kernel_size=(3,3), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x1)
x1_split = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x1)

# Convolution Layer ================================================================
x1 = Conv2D(filters=64,
            kernel_size=(1,1), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x1_split)

# Convolution Layer ================================================================
x1 = Conv2D(filters=64,
            kernel_size=(3,3),kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x1)

# Convolution Layer ================================================================
x1 = Conv2D(filters=32,
            kernel_size=(1,1), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x1)


# Now add back in the split layer, x_split (residual added in)
x1 = Add()([x1, x1_split])
x1 = Activation("relu")(x1)

x1 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x1)

x1 = Flatten()(x1)
x1 = Dropout(0.25)(x1)
x1 = Dense(256)(x1)
x1 = Activation("relu")(x1)
x1 = Dropout(0.5)(x1)
x1 = Dense(NUM_CLASSES)(x1)
x1 = Activation('softmax')(x1)
```

```
resnet_1 = Model(inputs=input_holder,outputs=x1)

#plot_model(resnet_1,show_shapes=True,show_layer_names=True,rankdir='LR',expand_nested=False,dpi=96)
resnet_1.summary()
```

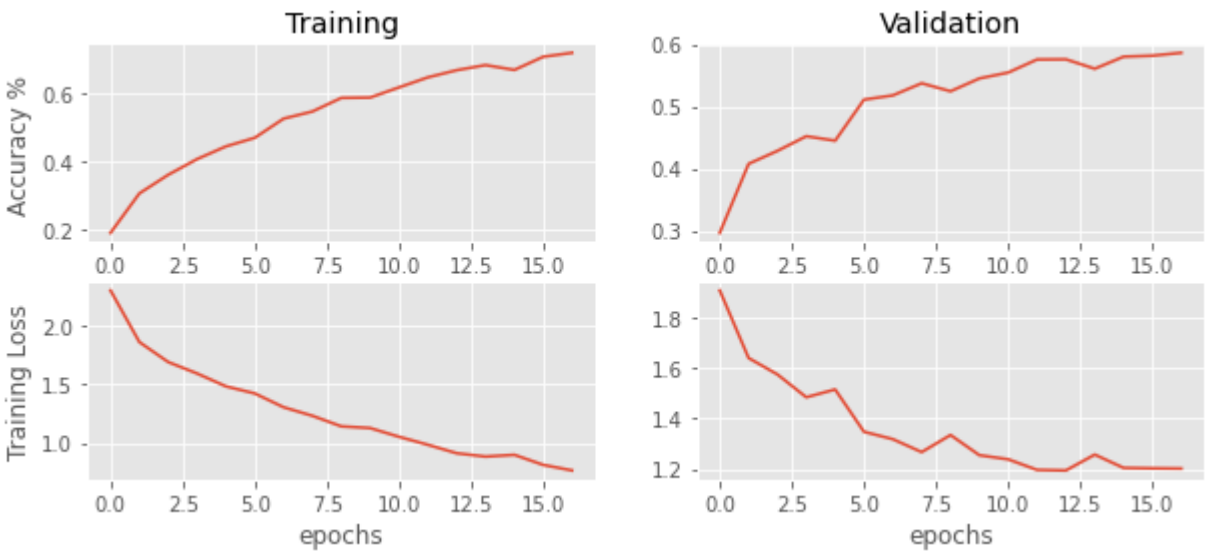
```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 96, 96, 1)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 96, 96, 32)   320         input_2[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 48, 48, 32)   0           conv2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 48, 48, 32)   9248        max_pooling2d[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 24, 24, 32)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 24, 24, 64)   2112        max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 24, 24, 64)   36928       conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 24, 24, 32)   2080        conv2d_3[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 24, 24, 32)   0           conv2d_4[0][0]
                                                                 max_pooling2d_1[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 24, 24, 32)   0           add[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 12, 12, 32)   0           activation[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 4608)         0           max_pooling2d_2[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 4608)         0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          1179904     dropout[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 256)          0           dense[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           activation_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 10)           2570        dropout_1[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 10)           0           dense_1[0][0]
==================================================================================================
Total params: 1,233,162
Trainable params: 1,233,162
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [13]:

```
%%time

resnet_1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history_resNet1 = resnet_1.fit(X_train, y_train_ohe, batch_size=128,
                               epochs=50, verbose=1, validation_data=(X_test,y_test_ohe),
                               callbacks=[EarlyStopping(monitor='val_loss', patience=4)])

yhat = np.round(resnet_1.predict(X_test))
print(mt.classification_report(y_test_ohe, yhat, zero_division=0))
getChart(history_resNet1)
```

```
Epoch 1/50
40/40 [==============================] - 23s 406ms/step - loss: 2.5983 - accuracy: 0.1485 - val_loss: 1.9087 -
val_accuracy: 0.2966
Epoch 2/50
40/40 [==============================] - 7s 176ms/step - loss: 1.9117 - accuracy: 0.2940 - val_loss: 1.6405 - v
al_accuracy: 0.4076
Epoch 3/50
40/40 [==============================] - 7s 178ms/step - loss: 1.7057 - accuracy: 0.3479 - val_loss: 1.5752 - v
al_accuracy: 0.4285
Epoch 4/50
40/40 [==============================] - 7s 177ms/step - loss: 1.6263 - accuracy: 0.3922 - val_loss: 1.4847 - v
al_accuracy: 0.4520
Epoch 5/50
40/40 [==============================] - 7s 175ms/step - loss: 1.5224 - accuracy: 0.4216 - val_loss: 1.5157 - v
al_accuracy: 0.4449
Epoch 6/50
40/40 [==============================] - 7s 176ms/step - loss: 1.4729 - accuracy: 0.4437 - val_loss: 1.3478 - v
al_accuracy: 0.5109
Epoch 7/50
40/40 [==============================] - 7s 178ms/step - loss: 1.3121 - accuracy: 0.5255 - val_loss: 1.3184 - v
al_accuracy: 0.5178
Epoch 8/50
40/40 [==============================] - 7s 177ms/step - loss: 1.2457 - accuracy: 0.5422 - val_loss: 1.2671 - v
al_accuracy: 0.5372
Epoch 9/50
40/40 [==============================] - 7s 175ms/step - loss: 1.1787 - accuracy: 0.5774 - val_loss: 1.3349 - v
al_accuracy: 0.5244
Epoch 10/50
40/40 [==============================] - 7s 176ms/step - loss: 1.1575 - accuracy: 0.5763 - val_loss: 1.2549 - v
al_accuracy: 0.5450
Epoch 11/50
40/40 [==============================] - 7s 178ms/step - loss: 1.0561 - accuracy: 0.6109 - val_loss: 1.2385 - v
al_accuracy: 0.5545
```

```
Epoch 12/50
40/40 [==============================] - 7s 178ms/step - loss: 0.9941 - accuracy: 0.6423 - val_loss: 1.1965 - v
al_accuracy: 0.5755
Epoch 13/50
40/40 [==============================] - 7s 174ms/step - loss: 0.9113 - accuracy: 0.6647 - val_loss: 1.1944 - v
al_accuracy: 0.5757
Epoch 14/50
40/40 [==============================] - 8s 199ms/step - loss: 0.8925 - accuracy: 0.6763 - val_loss: 1.2573 - v
al_accuracy: 0.5606
Epoch 15/50
40/40 [==============================] - 8s 214ms/step - loss: 0.9501 - accuracy: 0.6511 - val_loss: 1.2045 - v
al_accuracy: 0.5800
Epoch 16/50
40/40 [==============================] - 8s 205ms/step - loss: 0.8099 - accuracy: 0.7006 - val_loss: 1.2028 - v
al_accuracy: 0.5816
Epoch 17/50
40/40 [==============================] - 8s 194ms/step - loss: 0.7513 - accuracy: 0.7263 - val_loss: 1.2019 - v
al_accuracy: 0.5863
              precision    recall  f1-score   support

           0       0.81      0.76      0.79       800
           1       0.74      0.29      0.42       800
           2       0.81      0.70      0.75       800
           3       0.52      0.34      0.41       800
           4       0.69      0.39      0.50       800
           5       0.44      0.11      0.18       800
           6       0.71      0.49      0.58       800
           7       0.63      0.24      0.35       800
           8       0.79      0.69      0.74       800
           9       0.70      0.62      0.66       800

   micro avg       0.71      0.46      0.56      8000
   macro avg       0.68      0.46      0.54      8000
weighted avg       0.68      0.46      0.54      8000
 samples avg       0.46      0.46      0.46      8000

CPU times: user 1min 20s, sys: 11.7 s, total: 1min 31s
Wall time: 2min 25s
```



In [14]:

```
##################################################################################
######################### LeNet Style Architecture : RMSProp #####################
##################################################################################

# Convolution Layer ================================================================
x2 = Conv2D(filters=32, input_shape=shpe,
            kernel_size=(3,3),kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(input_holder)
x2 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x2)

# Convolution Layer ================================================================
x2 = Conv2D(filters=32,
            kernel_size=(3,3), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x2)
x2_split = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x2)

# Convolution Layer ================================================================
x2 = Conv2D(filters=64,
            kernel_size=(1,1), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x2_split)

# Convolution Layer ================================================================
x2 = Conv2D(filters=64,
            kernel_size=(3,3),kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x2)

# Convolution Layer ================================================================
```

```
x2 = Conv2D(filters=32,
            kernel_size=(1,1), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x2)

# now add back in the split layer, x_split (residual added in)
x2 = Add()([x2, x2_split])
x2 = Activation("relu")(x2)

x2 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x2)

x2 = Flatten()(x2)
x2 = Dropout(0.25)(x2)
x2 = Dense(256)(x2)
x2 = Activation("relu")(x2)
x2 = Dropout(0.5)(x2)
x2 = Dense(NUM_CLASSES)(x2)
x2 = Activation('softmax')(x2)

resnet_2 = Model(inputs=input_holder,outputs=x2)

#plot_model(resnet_2,show_shapes=True,show_layer_names=True,rankdir='LR',expand_nested=False,dpi=96)
resnet_2.summary()
```

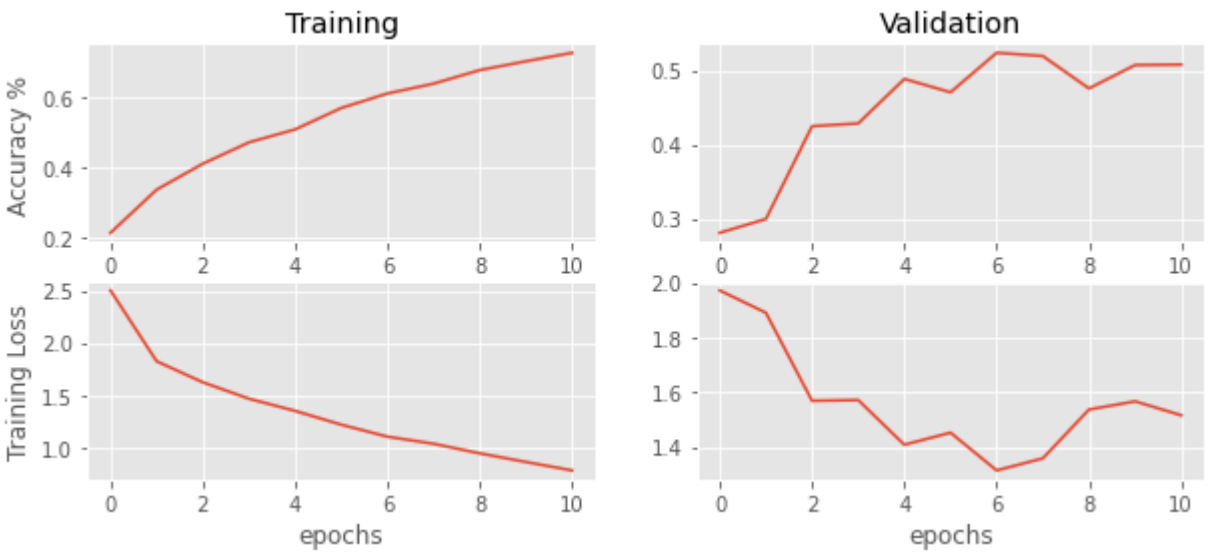
```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 96, 96, 1)]  0
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 96, 96, 32)   320         input_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 48, 48, 32)   0           conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 48, 48, 32)   9248        max_pooling2d_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 24, 24, 32)   0           conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 24, 24, 64)   2112        max_pooling2d_4[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 24, 24, 64)   36928       conv2d_7[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 24, 24, 32)   2080        conv2d_8[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 24, 24, 32)   0           conv2d_9[0][0]
                                                                 max_pooling2d_4[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, 24, 24, 32)   0           add_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 12, 12, 32)   0           activation_3[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 4608)         0           max_pooling2d_5[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 4608)         0           flatten_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          1179904     dropout_2[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 256)          0           dense_2[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 256)          0           activation_4[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 10)           2570        dropout_3[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 10)           0           dense_3[0][0]
==================================================================================================
Total params: 1,233,162
Trainable params: 1,233,162
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [15]:

```
%%time

resnet_2.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

history_resNet2 = resnet_2.fit(X_train, y_train_ohe, batch_size=128,
                               epochs=50, verbose=1, validation_data=(X_test,y_test_ohe),
                               callbacks=[EarlyStopping(monitor='val_loss', patience=4)])

yhat = np.round(resnet_2.predict(X_test))
print(mt.classification_report(y_test_ohe, yhat, zero_division=0))
getChart(history_resNet2)
```

```
Epoch 1/50
40/40 [==============================] - 8s 198ms/step - loss: 3.3166 - accuracy: 0.1609 - val_loss: 1.9711 - v
al_accuracy: 0.2806
Epoch 2/50
40/40 [==============================] - 7s 180ms/step - loss: 1.9114 - accuracy: 0.3033 - val_loss: 1.8896 - v
al_accuracy: 0.2993
```

```
Epoch 3/50
40/40 [==============================] - 7s 178ms/step - loss: 1.6728 - accuracy: 0.3885 - val_loss: 1.5684 - v
al_accuracy: 0.4244
Epoch 4/50
40/40 [==============================] - 7s 179ms/step - loss: 1.4955 - accuracy: 0.4655 - val_loss: 1.5710 - v
al_accuracy: 0.4281
Epoch 5/50
40/40 [==============================] - 7s 179ms/step - loss: 1.3683 - accuracy: 0.5021 - val_loss: 1.4080 - v
al_accuracy: 0.4881
Epoch 6/50
40/40 [==============================] - 7s 179ms/step - loss: 1.2230 - accuracy: 0.5635 - val_loss: 1.4520 - v
al_accuracy: 0.4701
Epoch 7/50
40/40 [==============================] - 7s 178ms/step - loss: 1.1010 - accuracy: 0.6187 - val_loss: 1.3142 - v
al_accuracy: 0.5234
Epoch 8/50
40/40 [==============================] - 7s 180ms/step - loss: 1.0203 - accuracy: 0.6369 - val_loss: 1.3583 - v
al_accuracy: 0.5190
Epoch 9/50
40/40 [==============================] - 7s 180ms/step - loss: 0.9364 - accuracy: 0.6774 - val_loss: 1.5363 - v
al_accuracy: 0.4754
Epoch 10/50
40/40 [==============================] - 7s 180ms/step - loss: 0.9145 - accuracy: 0.6847 - val_loss: 1.5663 - v
al_accuracy: 0.5069
Epoch 11/50
40/40 [==============================] - 7s 179ms/step - loss: 0.7999 - accuracy: 0.7211 - val_loss: 1.5154 - v
al_accuracy: 0.5075
              precision    recall  f1-score   support

           0       0.75      0.74      0.75       800
           1       0.60      0.31      0.41       800
           2       0.73      0.64      0.68       800
           3       0.77      0.04      0.07       800
           4       0.93      0.14      0.25       800
           5       0.31      0.45      0.36       800
           6       0.73      0.48      0.58       800
           7       0.74      0.08      0.14       800
           8       0.70      0.70      0.70       800
           9       0.55      0.68      0.60       800

   micro avg       0.61      0.43      0.50      8000
   macro avg       0.68      0.43      0.46      8000
weighted avg       0.68      0.43      0.46      8000
 samples avg       0.43      0.43      0.43      8000

CPU times: user 52.3 s, sys: 3.17 s, total: 55.5 s
Wall time: 1min 22s
```



2.2.2 Method 2: Xception Style Architecture

In [16]:

```
##########################################################################################
#################### Xception Style Architecture : AdaM ##########################
##########################################################################################

# Convolution Layer ==========================================================
x3 = Conv2D(filters=32, input_shape=shpe,
            kernel_size=(3,3), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(input_holder)
x3 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x3)

# Convolution Layer ==========================================================
x3 = Conv2D(filters=32,
            kernel_size=(3,3), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x3)
x3_split = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x3)

# Seperable Convoluation Layer ===============================================
x3 = SeparableConv2D(filters=32, input_shape=shpe,
                     kernel_size=(3,3), kernel_initializer='he_uniform',
                     kernel_regularizer=l2(l2_lambda), padding='same',
                     activation='relu', depth_multiplier = 1, # controls output channels
```

```
                    data_format="channels_last")(x3_split)
x3_split = Add()([x3, x3_split])

# Seperable Convoluation Layer ======================================================
x3 = SeparableConv2D(filters=32, input_shape=shpe,
                    kernel_size=(3,3), kernel_initializer='he_uniform',
                    kernel_regularizer=l2(l2_lambda), padding='same',
                    activation='relu', depth_multiplier = 1, # controls output channels
                    data_format="channels_last")(x3_split)
x3_split = Add()([x3, x3_split])


x3 = Activation("relu")(x3_split)
x3 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x3)

x3 = Flatten()(x3)
x3 = Dropout(0.25)(x3)
x3 = Dense(256, activation="relu")(x3)
x3 = Dropout(0.5)(x3)
x3 = Dense(NUM_CLASSES,activation="softmax")(x3)

xception_1 = Model(inputs=input_holder,outputs=x3)

#plot_model(xception_1,show_shapes=True,show_layer_names=True,rankdir='LR',expand_nested=False,dpi=96)
xception_1.summary()
```

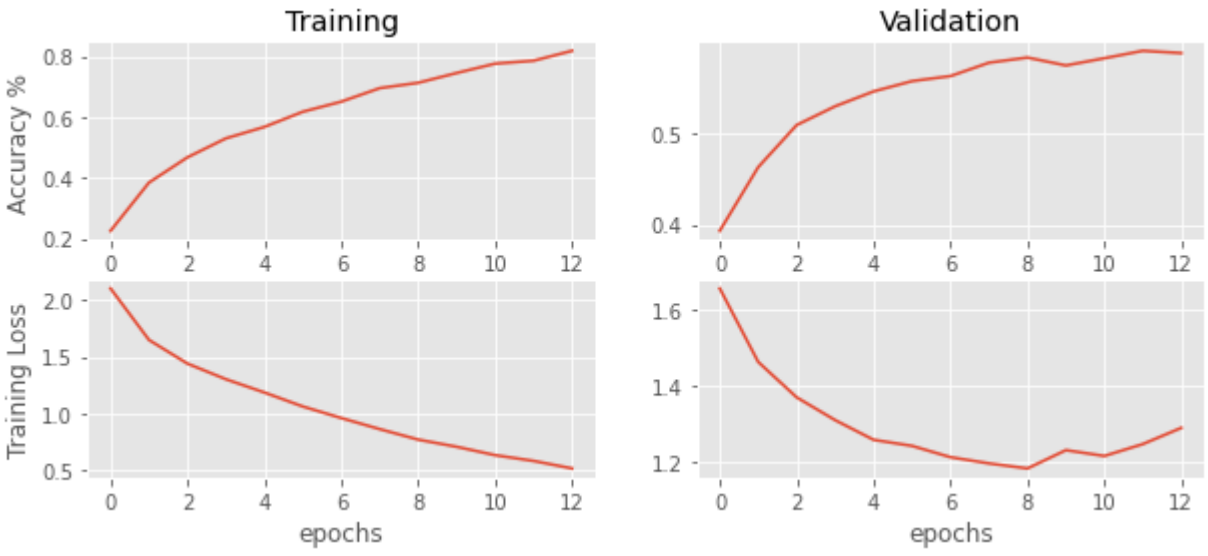
```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 96, 96, 1)]  0
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 96, 96, 32)   320         input_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_6 (MaxPooling2D)  (None, 48, 48, 32)   0           conv2d_10[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 48, 48, 32)   9248        max_pooling2d_6[0][0]
__________________________________________________________________________________________________
max_pooling2d_7 (MaxPooling2D)  (None, 24, 24, 32)   0           conv2d_11[0][0]
__________________________________________________________________________________________________
separable_conv2d (SeparableConv (None, 24, 24, 32)   1344        max_pooling2d_7[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 24, 24, 32)   0           separable_conv2d[0][0]
                                                                 max_pooling2d_7[0][0]
__________________________________________________________________________________________________
separable_conv2d_1 (SeparableCo (None, 24, 24, 32)   1344        add_2[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 24, 24, 32)   0           separable_conv2d_1[0][0]
                                                                 add_2[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 24, 24, 32)   0           add_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_8 (MaxPooling2D)  (None, 12, 12, 32)   0           activation_6[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 4608)         0           max_pooling2d_8[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 4608)         0           flatten_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 256)          1179904     dropout_4[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 256)          0           dense_4[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 10)           2570        dropout_5[0][0]
==================================================================================================
Total params: 1,194,730
Trainable params: 1,194,730
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [17]:

```
# speed up by training by not using augmentation, perhaps there are faster ways??
xception_1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history_Xception1 = xception_1.fit(X_train, y_train_ohe, batch_size=32,
                                   epochs=50, verbose=1, validation_data=(X_test,y_test_ohe),
                                   callbacks=[EarlyStopping(monitor='val_loss', patience=4)])

yhat = np.round(xception_1.predict(X_test))
print(mt.classification_report(y_test_ohe, yhat, zero_division=0))
getChart(history_Xception1)
```

```
Epoch 1/50
157/157 [==============================] - 12s 53ms/step - loss: 2.3859 - accuracy: 0.1649 - val_loss: 1.6569 -
val_accuracy: 0.3932
Epoch 2/50
157/157 [==============================] - 6s 41ms/step - loss: 1.7257 - accuracy: 0.3458 - val_loss: 1.4634 -
val_accuracy: 0.4629
Epoch 3/50
157/157 [==============================] - 6s 41ms/step - loss: 1.4418 - accuracy: 0.4666 - val_loss: 1.3702 -
```

```
val_accuracy: 0.5089
Epoch 4/50
157/157 [==============================] - 6s 41ms/step - loss: 1.3015 - accuracy: 0.5325 - val_loss: 1.3103 -
val_accuracy: 0.5294
Epoch 5/50
157/157 [==============================] - 6s 41ms/step - loss: 1.1779 - accuracy: 0.5663 - val_loss: 1.2585 -
val_accuracy: 0.5456
Epoch 6/50
157/157 [==============================] - 6s 41ms/step - loss: 1.0686 - accuracy: 0.6200 - val_loss: 1.2425 -
val_accuracy: 0.5569
Epoch 7/50
157/157 [==============================] - 6s 41ms/step - loss: 0.9531 - accuracy: 0.6561 - val_loss: 1.2127 -
val_accuracy: 0.5624
Epoch 8/50
157/157 [==============================] - 6s 41ms/step - loss: 0.8398 - accuracy: 0.7019 - val_loss: 1.1964 -
val_accuracy: 0.5770
Epoch 9/50
157/157 [==============================] - 6s 41ms/step - loss: 0.7494 - accuracy: 0.7254 - val_loss: 1.1832 -
val_accuracy: 0.5826
Epoch 10/50
157/157 [==============================] - 6s 41ms/step - loss: 0.6960 - accuracy: 0.7447 - val_loss: 1.2313 -
val_accuracy: 0.5740
Epoch 11/50
157/157 [==============================] - 6s 41ms/step - loss: 0.6208 - accuracy: 0.7839 - val_loss: 1.2160 -
val_accuracy: 0.5819
Epoch 12/50
157/157 [==============================] - 6s 41ms/step - loss: 0.5746 - accuracy: 0.7905 - val_loss: 1.2472 -
val_accuracy: 0.5900
Epoch 13/50
157/157 [==============================] - 6s 41ms/step - loss: 0.5162 - accuracy: 0.8179 - val_loss: 1.2899 -
val_accuracy: 0.5875
              precision    recall  f1-score   support

           0       0.83      0.77      0.80       800
           1       0.68      0.31      0.42       800
           2       0.82      0.71      0.76       800
           3       0.62      0.26      0.36       800
           4       0.55      0.56      0.56       800
           5       0.46      0.23      0.30       800
           6       0.62      0.60      0.61       800
           7       0.55      0.45      0.49       800
           8       0.80      0.68      0.73       800
           9       0.67      0.66      0.67       800

   micro avg       0.67      0.52      0.59      8000
   macro avg       0.66      0.52      0.57      8000
weighted avg       0.66      0.52      0.57      8000
 samples avg       0.52      0.52      0.52      8000
```



In [18]:

```
######################################################################################
##################### Xception Style Architecture : RMSProp ######################
######################################################################################

# Convolution Layer ==================================================================
x4 = Conv2D(filters=32, input_shape=shpe,
            kernel_size=(3,3), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(input_holder)
x4 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x4)

# Convolution Layer ==================================================================
x4 = Conv2D(filters=32,
            kernel_size=(3,3), kernel_initializer='he_uniform',
            kernel_regularizer=l2(l2_lambda), padding='same',
            activation='relu', data_format="channels_last")(x4)
x4_split = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x4)

# Seperable Convoluation Layer =======================================================
x4 = SeparableConv2D(filters=32, input_shape=shpe,
                     kernel_size=(3,3), kernel_initializer='he_uniform',
                     kernel_regularizer=l2(l2_lambda), padding='same',
                     activation='relu', depth_multiplier = 1, # controls output channels
                     data_format="channels_last")(x4_split)
```

```
x4_split = Add()([x4, x4_split])

# Seperable Convoluation Layer ======================================================
x4 = SeparableConv2D(filters=32, input_shape=shpe,
                     kernel_size=(3,3), kernel_initializer='he_uniform',
                     kernel_regularizer=l2(l2_lambda), padding='same',
                     activation='relu', depth_multiplier = 1, # controls output channels
                     data_format="channels_last")(x4_split)
x4_split = Add()([x4, x4_split])

x4 = Activation("relu")(x4_split)
x4 = MaxPooling2D(pool_size=(2, 2), data_format="channels_last")(x4)


x4 = Flatten()(x4)
x4 = Dropout(0.25)(x4)
x4 = Dense(256, activation="relu")(x4)
x4 = Dropout(0.5)(x4)
x4 = Dense(NUM_CLASSES,activation="softmax")(x4)

xception_2 = Model(inputs=input_holder,outputs=x4)

#plot_model(xception_2,show_shapes=True,show_layer_names=True,rankdir='LR',expand_nested=False,dpi=96)
```

```
xception_2.summary()
```

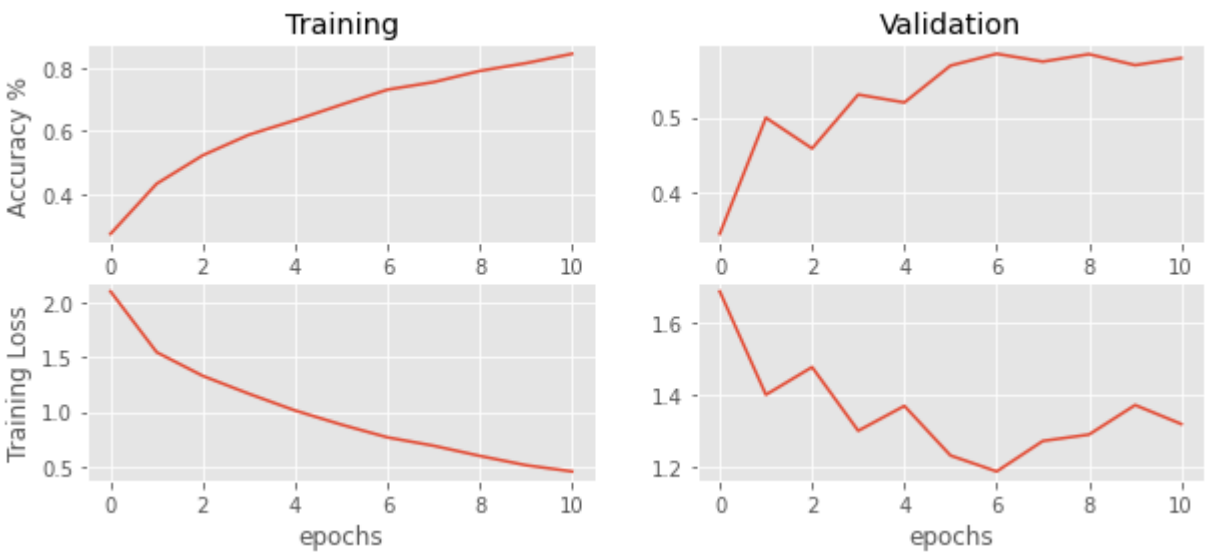
```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 96, 96, 1)]  0
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 96, 96, 32)   320         input_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_9 (MaxPooling2D)  (None, 48, 48, 32)   0           conv2d_12[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 48, 48, 32)   9248        max_pooling2d_9[0][0]
__________________________________________________________________________________________________
max_pooling2d_10 (MaxPooling2D) (None, 24, 24, 32)   0           conv2d_13[0][0]
__________________________________________________________________________________________________
separable_conv2d_2 (SeparableCo (None, 24, 24, 32)   1344        max_pooling2d_10[0][0]
__________________________________________________________________________________________________
add_4 (Add)                     (None, 24, 24, 32)   0           separable_conv2d_2[0][0]
                                                                 max_pooling2d_10[0][0]
__________________________________________________________________________________________________
separable_conv2d_3 (SeparableCo (None, 24, 24, 32)   1344        add_4[0][0]
__________________________________________________________________________________________________
add_5 (Add)                     (None, 24, 24, 32)   0           separable_conv2d_3[0][0]
                                                                 add_4[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 24, 24, 32)   0           add_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_11 (MaxPooling2D) (None, 12, 12, 32)   0           activation_7[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten)             (None, 4608)         0           max_pooling2d_11[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 4608)         0           flatten_3[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 256)          1179904     dropout_6[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 256)          0           dense_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 10)           2570        dropout_7[0][0]
==================================================================================================
Total params: 1,194,730
Trainable params: 1,194,730
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [19]:

```
# speed up by training by not using augmentation, perhaps there are faster ways??
xception_2.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

history_Xception2 = xception_2.fit(X_train, y_train_ohe, batch_size=32,
                                   epochs=50, verbose=1, validation_data=(X_test,y_test_ohe),
                                   callbacks=[EarlyStopping(monitor='val_loss', patience=4)])

yhat = np.round(xception_2.predict(X_test))
print(mt.classification_report(y_test_ohe, yhat, zero_division=0))
getChart(history_Xception2)
```

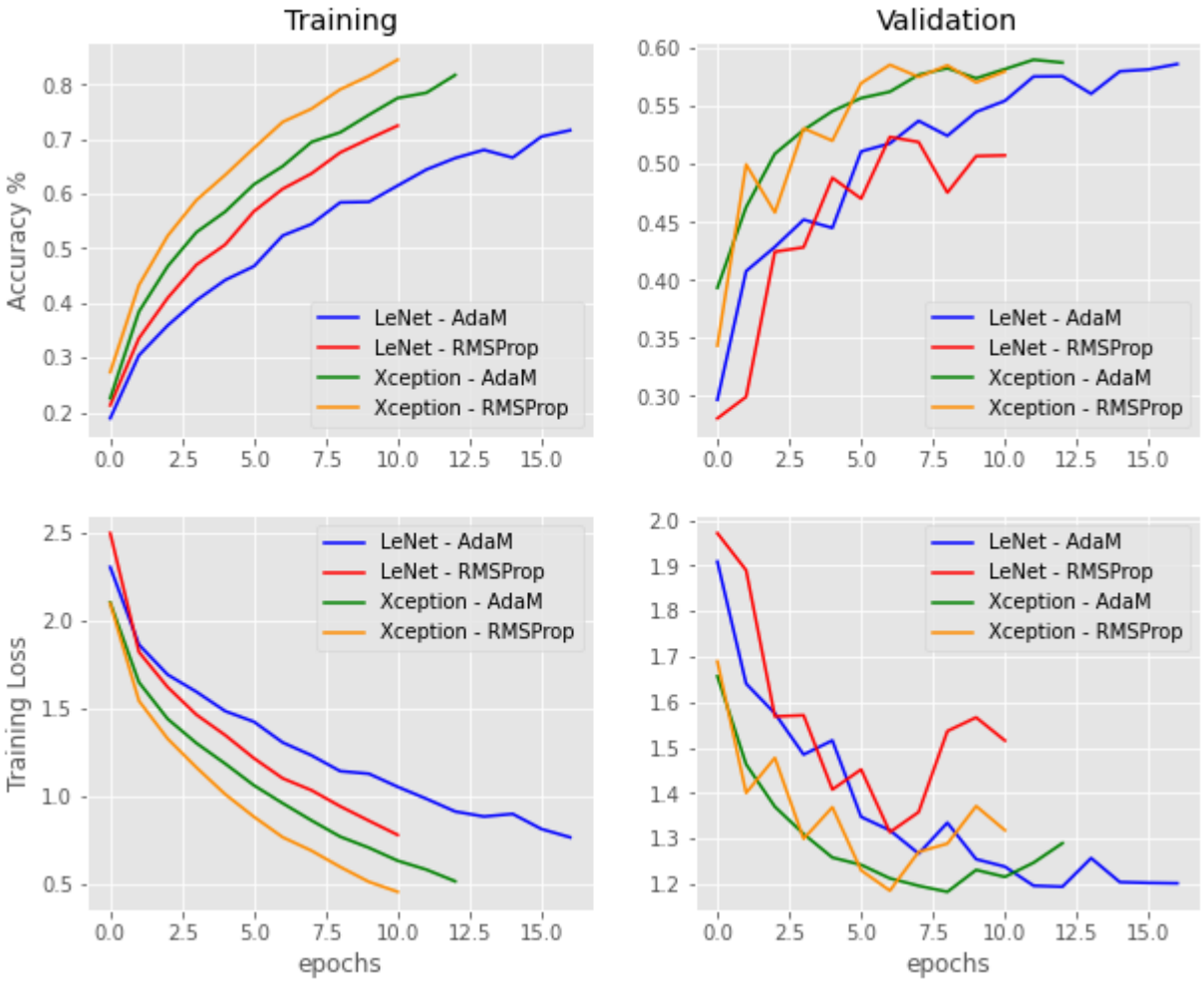
```
Epoch 1/50
157/157 [==============================] - 8s 45ms/step - loss: 2.7548 - accuracy: 0.2035 - val_loss: 1.6893 -
val_accuracy: 0.3434
Epoch 2/50
157/157 [==============================] - 7s 43ms/step - loss: 1.5989 - accuracy: 0.4165 - val_loss: 1.4003 -
val_accuracy: 0.4996
Epoch 3/50
157/157 [==============================] - 7s 43ms/step - loss: 1.3408 - accuracy: 0.5216 - val_loss: 1.4776 -
val_accuracy: 0.4582
```

```
Epoch 4/50
157/157 [==============================] - 7s 43ms/step - loss: 1.1468 - accuracy: 0.5899 - val_loss: 1.2996 -
val_accuracy: 0.5307
Epoch 5/50
157/157 [==============================] - 7s 43ms/step - loss: 1.0230 - accuracy: 0.6283 - val_loss: 1.3691 -
val_accuracy: 0.5201
Epoch 6/50
157/157 [==============================] - 7s 43ms/step - loss: 0.8473 - accuracy: 0.7016 - val_loss: 1.2303 -
val_accuracy: 0.5698
Epoch 7/50
157/157 [==============================] - 7s 43ms/step - loss: 0.7463 - accuracy: 0.7345 - val_loss: 1.1858 -
val_accuracy: 0.5856
Epoch 8/50
157/157 [==============================] - 7s 43ms/step - loss: 0.6728 - accuracy: 0.7634 - val_loss: 1.2710 -
val_accuracy: 0.5751
Epoch 9/50
157/157 [==============================] - 7s 43ms/step - loss: 0.5744 - accuracy: 0.8002 - val_loss: 1.2890 -
val_accuracy: 0.5851
Epoch 10/50
157/157 [==============================] - 7s 43ms/step - loss: 0.5197 - accuracy: 0.8159 - val_loss: 1.3714 -
val_accuracy: 0.5704
Epoch 11/50
157/157 [==============================] - 7s 43ms/step - loss: 0.4394 - accuracy: 0.8488 - val_loss: 1.3185 -
val_accuracy: 0.5799
              precision    recall  f1-score   support

           0       0.90      0.69      0.78       800
           1       0.64      0.34      0.44       800
           2       0.84      0.66      0.74       800
           3       0.55      0.26      0.35       800
           4       0.50      0.67      0.57       800
           5       0.50      0.19      0.28       800
           6       0.70      0.56      0.62       800
           7       0.50      0.46      0.48       800
           8       0.77      0.70      0.73       800
           9       0.63      0.72      0.67       800

   micro avg       0.65      0.53      0.58      8000
   macro avg       0.65      0.53      0.57      8000
weighted avg       0.65      0.53      0.57      8000
 samples avg       0.53      0.53      0.53      8000
```



2.3 Performance Analysis

In [20]:
```
getCharts(history_resNet1, history_resNet2, history_Xception1, history_Xception2)
```



References

Kaggle. STL-10. https://www.kaggle.com/jessicali9530/stl10 (Accessed 5-1-2021)

Adam Coates, Honglak Lee, Andrew Y. Ng An Analysis of Single Layer Networks in Unsupervised Feature Learning AISTATS, 2011.