CS 109B Advanced Topics in Data Science, Final Project, Milestone 4

Group 9 - Steve Robbins, Chad Tsang, and Ted Heuer

Harvard University
Spring 2017
Due Date: Wednesday, April 26th, 2017 at 11:59pm

Milestone 4: Deep learning, due Wednesday, April 26, 2017

For this milestone you will (finally) use deep learning to predict movie genres. You will train one small network from scratch on the posters only, and compare this one to a pre-trained network that you fine tune. Here (https://keras.io/getting-started/faq/#how-can-i-use-pre-trained-models-in-keras) is a description of how to use pretrained models in Keras.

You can try different architectures, initializations, parameter settings, optimization methods, etc. Be adventurous and explore deep learning! It can be fun to combine the features learned by the deep learning model with a SVM, or incorporate meta data into your deep learning model.

Note: Be mindful of the longer training times for deep models. Not only for training time, but also for the parameter tuning efforts. You need time to develop a feel for the different parameters and which settings work, which normalization you want to use, which model architecture you choose, etc.

It is great that we have GPUs via AWS to speed up the actual computation time, but you need to be mindful of your AWS credits. The GPU instances are not cheap and can accumulate costs rather quickly. Think about your model first and do some quick dry runs with a larger learning rate or large batch size on your local machine.

The notebook to submit this week should at least include:

- Complete description of the deep network you trained from scratch, including parameter settings, performance, features learned, etc.
- Complete description of the pre-trained network that you fine tuned, including parameter settings, performance, features learned, etc.
- Discussion of the results, how much improvement you gained with fine tuning, etc.
- Discussion of at least one additional exploratory idea you pursued

In [1]:

```
#!pip install keras
#!pip install tensorflow
#!pip install tensorflow.python
#!pip install h5py
```

In [1]:

```
from __future__ import print_function
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
import numpy as np
import pandas as pd
from scipy import misc
import time
from sklearn.preprocessing import MultiLabelBinarizer
```

In [2]:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential, Model
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropo
ut, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_a
rray, load_img
from keras.optimizers import SGD
from keras import backend as K
from keras.applications.inception_v3 import InceptionV3
from keras import regularizers
```

```
Using TensorFlow backend.
```

The following explicit setting of the random seed was not used in testing and tuning efforts, as it would create a propensity to fit a particular dataset. It was included only to make re-executions of the notebook match the exact text supplied in the analysis.

In [3]:

```
import random
random.seed(42)
```

Load, split, and prepare the data.

In [4]:

```
def load_poster_data(image_size, source_size = 'w92', verbose = False):
    # Loads the poster image data at the requested size, the assigned genre, and
the movie id.
    y_labels = pd.read_csv('y_labels_multiclass.csv')

    image_path = './posters/' + source_size + '/'
    posters = pd.DataFrame()
    for movie in y_labels.iterrows():
        row = movie[0]
        movie_id = movie[1]['movie_id']
        genre_id = int(movie[1]['genre_id'].replace('[', '').replace(']',''))
        try:
            image = misc.imread(image_path + str(movie_id) + '.jpg')
            image_resize = img_to_array(misc.imresize(image, image_size))
            if (image_resize.shape[2]==3):
                posters = posters.append({'movie_id' : movie_id,
                                          'genre_id' : genre_id,
                                          'poster' : image_resize}, ignore_index
= True)
        except IOError:
            if (verbose == True):
                print('Unable to load poster for movie #', movie_id)
    print('Loaded ', posters.shape[0], ' posters.')
    return posters
```

In [5]:

```
def stratified_sampler(dataset, observations):
    # Performs a stratified sample on the dataset and returns the number of obse
rvations
    # requested.
    #
    # Parameters:
    #    dataset:  The dataframe to sample, observing class relationships.
    #    observations:  The number of total target observations across all class
es.
    #
    # Returns:
    #    A pandas dataframe sampled from the dataset maintaining class relations
hips.
    class_weights = dataset.groupby("genre_id").agg(['count'])/len(dataset)
    class_sample_counts = class_weights * observations
    class_count = class_weights.shape[0]
    sampled = pd.DataFrame()
    for class_to_sample in class_sample_counts.iterrows():
        class_name = class_to_sample[0]
        desired_class_observations = class_to_sample[1][0]
        sampled_obs = dataset[dataset["genre_id"]==class_name].sample(int(desire
d_class_observations), replace="True")
        sampled = sampled.append(sampled_obs, ignore_index=True)
    return sampled, class_count
```

In [6]:

```
def reshape_and_normalize(data):
    # Reshape the dataset and normalize the 8-bit RGB values to floats.
    image_count = data.shape[0]
    temp = np.ndarray(shape=(image_count, data[0].shape[0], data[0].shape[1], 3)
)

    for index in range(0, image_count):
        try:
            temp[index] = data[index].reshape(data[0].shape[0], data[0].shape[1]
, 3)
        except ValueError:
            print(data[index].shape)
    # Since we use relu ubiquitously, normalize to [0,1]
    temp = temp.astype('float32')
    temp /= 255.0

    return temp
```

In [7]:

```
def normalize_responses(data):
    # Replace the genre ids to a sequential set from 0..classes
    unique_responses = np.sort(data["genre_id"].unique())
    data["genre_id"] = data["genre_id"].replace(unique_responses, range(0,len(un
ique_responses)), inplace=False)
    return data
```

In [8]:

```
def load_split_prepare_data(train_observations, test_observations, image_size, s
ample = 'stratified'):
    # Loads, splits, and prepares the data for use by a CNN model.
    #
    # Parameters:
    #    train_observations:  The dataframe to sample, observing class relations
hips.
    #    test_observations:  The number of total target observations across all
classes.
    #    sample:  The sampling method, currently only supports 'stratified'
    #
    # Returns:
    #    The training and testing datasets, with normalized images and categoric
al responses.
    #    The number of classes.
    posters_data = load_poster_data(image_size)
    posters = normalize_responses(posters_data)
    class_weights = posters.groupby("genre_id").agg(['count'])["movie_id"].to_di
ct()["count"]

    if (sample == 'stratified'):
        train_sample, class_count_train = stratified_sampler(posters, train_obse
rvations)
        remaining_posters = posters[~posters["movie_id"].isin(train_sample["movi
e_id"])]
        test_sample, class_count_test = stratified_sampler(remaining_posters, te
st_observations)
    else:
        raise('Unsupported sample method : ', sample)

    x_train = train_sample["poster"]
    y_train = train_sample["genre_id"]
    x_test = test_sample["poster"]
    y_test = test_sample["genre_id"]

    img_rows = x_train[0].shape[0]
    img_cols = x_train[0].shape[1]
    print('Classes : ', class_count_train)

    x_train = reshape_and_normalize(x_train)
    x_test = reshape_and_normalize(x_test)

    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    # Convert response to one hot encoding
    y_train = keras.utils.to_categorical(y_train, class_count_train)
    y_test = keras.utils.to_categorical(y_test, class_count_test)

    return (x_train, y_train), (x_test, y_test), class_weights
```

In [10]:

```
(x_train, y_train), (x_test, y_test), class_weights = load_split_prepare_data(tr
ain_observations = 5000,
                                                                              te
st_observations = 1000,
                                                                              im
age_size = (138,92),
                                                                              sa
mple='stratified')
classes = len(class_weights.keys())
```

```
Loaded  6824  posters.
Classes :  7
x_train shape: (4996, 138, 92, 3)
4996 train samples
996 test samples
```

Baseline (Simple) Multiclass CNN

In [16]:

```
final_activation_function = 'softmax'

input_activation_function = 'relu'
input_kernel_size = (5,5)
input_shape = (138, 92, 3)
pool_size = (3,3)

hidden_activation_function = 'relu'
hidden_kernel_size = (3,3)

loss_method = 'categorical_crossentropy'
# Learning rate manually tuned based on model performance, too high numerically
# results in nearly no improvement per epoch, and too low numerically doesn't
# cause the weights to change significantly (fine tuning)
optimizer = SGD(lr=0.1, momentum=0.9)
eval_metric = 'accuracy'

# smaller batch size means noisier gradient, but more updates per epoch
batch_size = 512
# number of iterations over the complete training data
epochs = 200
```

In [12]:

```
# create an empty network model
model = Sequential()

# Input Layer
model.add(Conv2D(16, kernel_size=input_kernel_size, activation=input_activation_
function, input_shape=input_shape))
model.add(MaxPooling2D(pool_size=pool_size))

# Hidden Layer(s)
model.add(Conv2D(32, kernel_size=hidden_kernel_size, activation=hidden_activatio
n_function))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=pool_size))

# Classification layer with Regularizers
model.add(Flatten())
model.add(Dense(64, activation=hidden_activation_function))
model.add(Dropout(0.5))
model.add(Dense(classes, activation=final_activation_function))

# Compile the model.
model.compile(loss=loss_method, optimizer=optimizer, metrics=[eval_metric])
```

In [13]:

```
debug_flag = 1
```

In [22]:

```
def train_evaluate_CNN(x_train, y_train, x_test, y_test, model, batch_size, epoc
hs, verbose, show_layers,
                       validation_split, weights_filename, plot_title, class_wei
ghts, augment_data = False):
    # Trains and evaluates the performance numerically and graphically of a comp
iled Convolutional Neural Net.
    #
    # Parameters:
    #    x_train, y_train:  The predictors and responses of the training set.
    #    x_test, y_test:  The predictors and responses of the test set.
    #    model:  The compiled CNN model.
    #    batch_size:  The batch size used in training.
    #    epochs:  The number of epochs to train.
    #    verbose:  If 1, will display the intermediate epoch training results.
    #    show_layers:  If True, will display the model layers and parameters.
    #    validation_split:  The percentage of the training set to use as a valid
ation set for tuning.
    #    weights_filename:  The name of the file to store the tuned weights in.
    #    plot_title:  The model title to include in the performance plot.
    #    class_weights:  The relative class weights.
    #    augment_data:  If true, will augment the input dataset dramatically inc
reasing size.
```

```
    #
    # Returns:
    #    test_accuracy:  The test accuracy (as a float)
    #    train_time:  The time spent training the model, in seconds.
    start_time = time.time()
    if (augment_data == True):
        datagen = ImageDataGenerator(
            featurewise_center=True,
            samplewise_center=False,
            featurewise_std_normalization=True,
            samplewise_std_normalization=False,
            zca_whitening=False,
            rotation_range=20,
            width_shift_range=0.2,
            height_shift_range=0.2,
            horizontal_flip=True,
            vertical_flip=False)
        datagen.fit(x_train)
        history = model.fit_generator(datagen.flow(x_train, y_train, batch_size=
batch_size),
                                      steps_per_epoch=16,
                                      epochs=epochs,
                                      verbose=verbose)
    else:
        history = model.fit(x_train, y_train,
                            batch_size=batch_size,
                            epochs=epochs,
                            verbose=verbose,
                            class_weight=class_weights,
                            validation_split = validation_split)
    end_time = time.time()
    train_time = end_time - start_time
    print('Time to train model with ', epochs, ' epochs is : ', train_time, ' se
conds.')
    # Evaluate the performance on the testing set.
    score = model.evaluate(x_test, y_test, verbose=verbose)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

    if (show_layers == 1):
        print(model.summary())

    model.save_weights(weights_filename + '_weights.h5')

    plt.plot(history.history['acc'])
    plt.axhline(y=score[1], color='r')
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.title("Training Accuracy, " + plot_title);
    print(plt.show())

    plt.plot(history.history['val_loss'])
    plt.xlabel("Epoch")
```

```
    plt.ylabel("Validation Loss")

    plt.title("Training Accuracy, " + plot_title);
    print(plt.show())

    # Return the TEST set accuracy.
    return(score[1], train_time)
```

In [15]:

```
experiment_results = pd.DataFrame()
```

In [18]:

```
# Test:  Scale class weights
scaled_weights = {0: 1.93229167,
                  1: 4.31770833,
                  2: 8.32291667, 3: 8.15625, 4: 7.33854167, 5: 1.0, 6: 4.4739583
3}
uniform_weights = {0: 1., 1: 1., 2: 1., 3: 1., 4: 1., 5: 1.0, 6: 1.}
```

In [17]:

```
# The actual training of the CNN using the parameters and model previously speci
fied.
# The validation set is a split of the stratified sampled training data.
experiment = 'Baseline Model, Multiclass'
accuracy, train_time = train_evaluate_CNN(x_train, y_train, x_test, y_test, mode
l,
                                          batch_size, epochs=100,
                                          verbose=debug_flag, show_layers=1,
                                          validation_split=0.15,
                                          weights_filename='baseline',
                                          class_weights=uniform_weights,
                                          plot_title=experiment)
experiment_results = experiment_results.append({'Name' : experiment,
                                                'Test Accuracy' : accuracy,
                                                'Train Time': train_time}, ignor
e_index=True)
```

```
Train on 4246 samples, validate on 750 samples
Epoch 1/100
4246/4246 [==============================] - 7s - loss: 1.8109 - acc
: 0.2414 - val_loss: 4.2872 - val_acc: 0.0000e+00
Epoch 2/100
4246/4246 [==============================] - 2s - loss: 1.5839 - acc
: 0.2605 - val_loss: 4.2008 - val_acc: 0.0000e+00
Content manually removed for brevity.
Epoch 99/100
4246/4246 [==============================] - 2s - loss: 0.3739 - acc
: 0.8493 - val_loss: 11.5140 - val_acc: 0.0067
Epoch 100/100
4246/4246 [==============================] - 2s - loss: 0.3690 - acc
: 0.8545 - val_loss: 10.7931 - val_acc: 0.0053
Time to train model with  100  epochs is :  275.826075077  seconds.
960/996 [===========================>..] - ETA: 0s - ETA: 0sTest los
s: 3.84234348358
Test accuracy: 0.327309236948
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 134, 88, 16)       1216
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 44, 29, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 42, 27, 32)        4640
_________________________________________________________________
dropout_1 (Dropout)          (None, 42, 27, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 9, 32)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4032)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                258112
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 455
=================================================================
Total params: 264,423
Trainable params: 264,423
Non-trainable params: 0
_________________________________________________________________
None
```

None

None

Baseline Regularized Multiclass Model

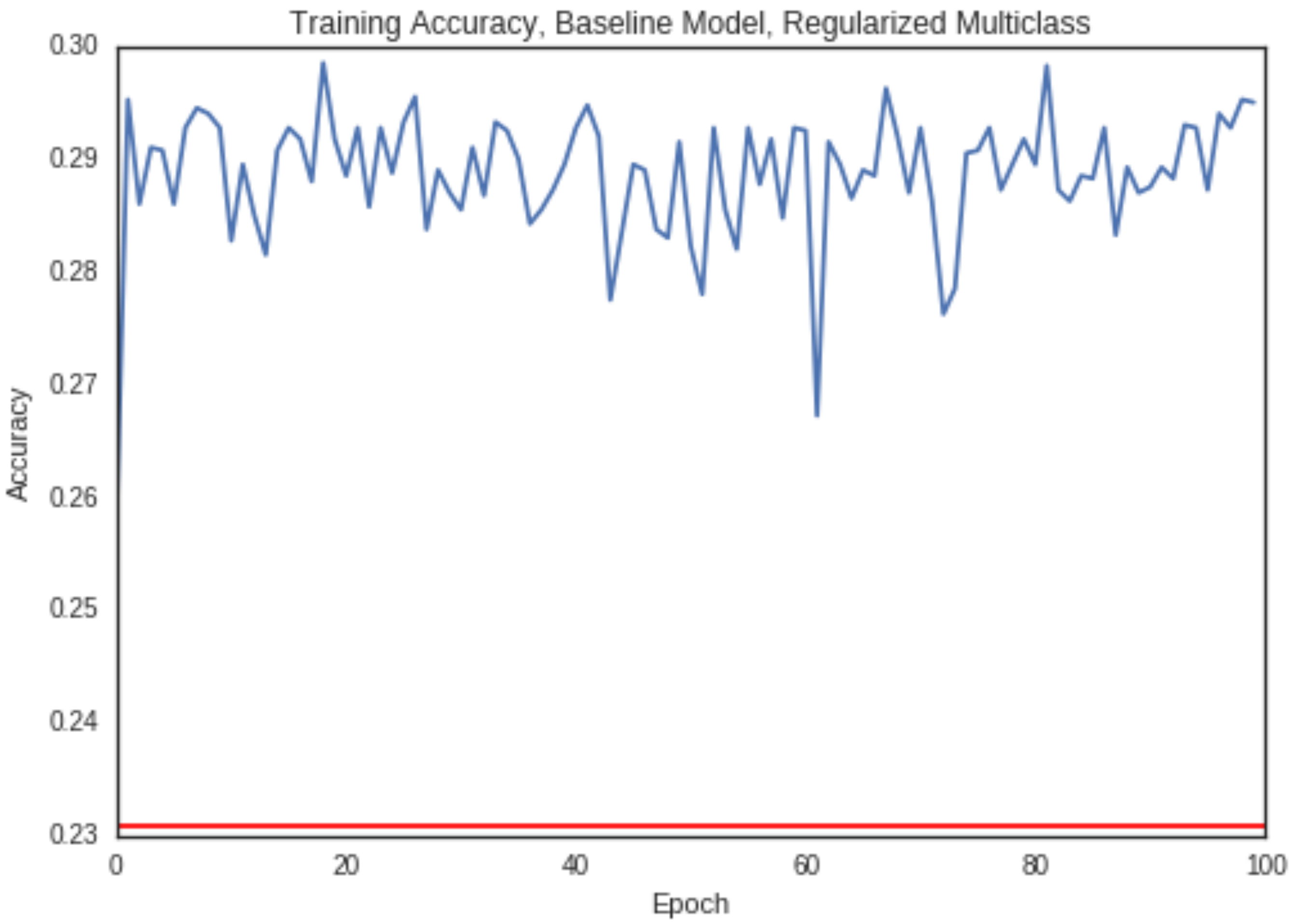
In [18]:

```
# create an empty network model
model = Sequential()

# Input Layer
model.add(Conv2D(16, kernel_size=input_kernel_size, activation=input_activation_
function, input_shape=input_shape))
model.add(MaxPooling2D(pool_size=pool_size))

# Hidden Layer(s)
model.add(Conv2D(32, kernel_size=hidden_kernel_size, activation=hidden_activatio
n_function))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=pool_size))

# Classification layer with Regularizers
model.add(Flatten())
model.add(Dense(64, activation=hidden_activation_function,
                activity_regularizer=regularizers.l1(0.1)))
model.add(Dropout(0.5))
model.add(Dense(classes, activation=final_activation_function))

# Compile the model.
model.compile(loss=loss_method, optimizer=optimizer, metrics=[eval_metric])
```

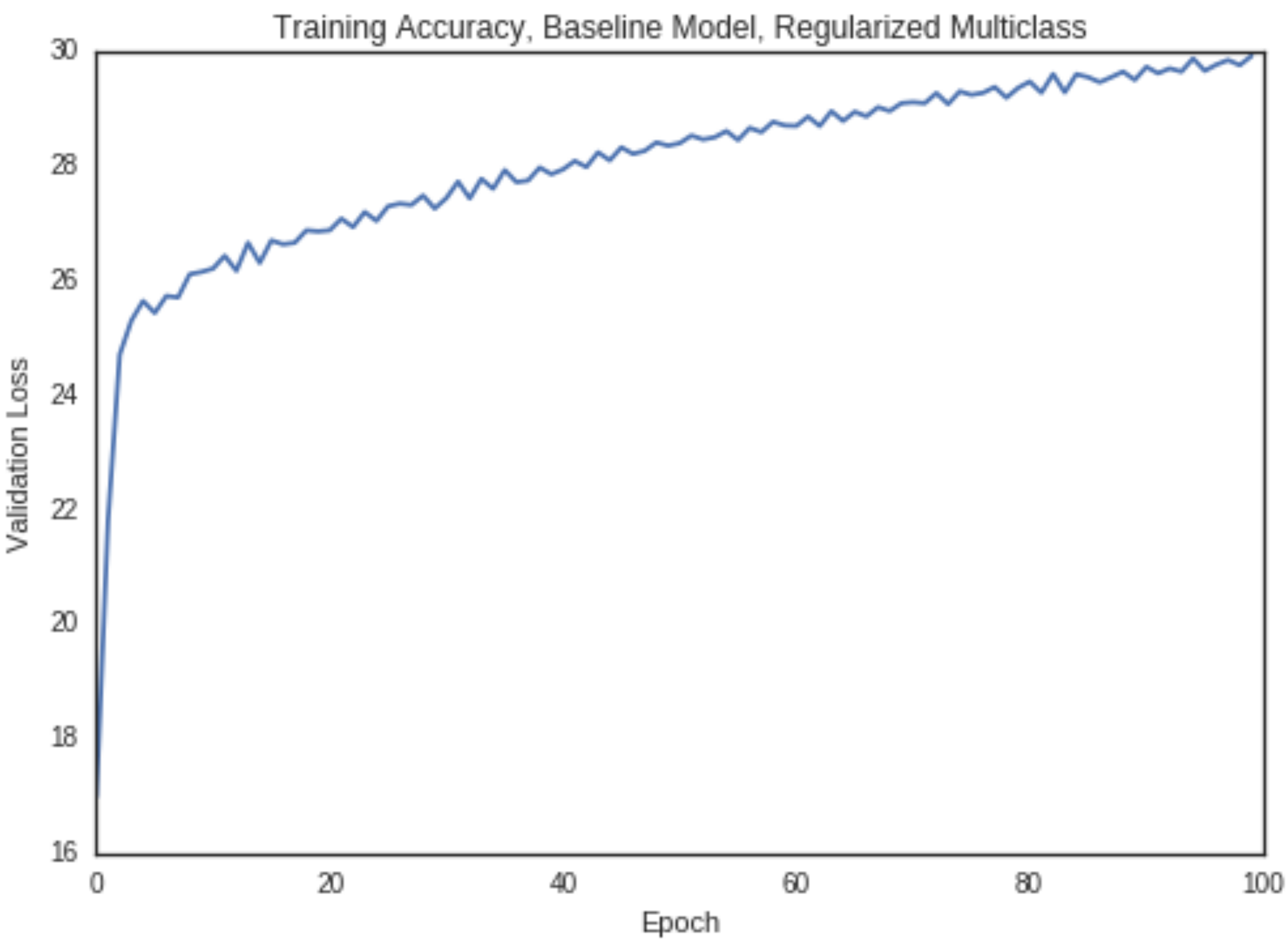
In [19]:

```
# The actual training of the CNN using the parameters and model previously speci
fied.
# The validation set is a split of the stratified sampled training data.
experiment = 'Baseline Model, Regularized Multiclass'
accuracy, train_time = train_evaluate_CNN(x_train, y_train, x_test, y_test, mode
l,
                                          batch_size, epochs=100,
                                          verbose=debug_flag, show_layers=1,
                                          validation_split=0.2,
                                          weights_filename='baseline_reg',
                                          class_weights=scaled_weights,
                                          plot_title=experiment)
experiment_results = experiment_results.append({'Name' : experiment,
                                                'Test Accuracy' : accuracy,
                                                'Train Time': train_time}, ignor
e_index=True)
```

```
Train on 3996 samples, validate on 1000 samples
Epoch 1/100
3996/3996 [==============================] - 3s - loss: 906.1696 - a
cc: 0.2578 - val_loss: 16.9834 - val_acc: 0.0000e+00
Epoch 2/100
3996/3996 [==============================] - 2s - loss: 9.9114 - acc
: 0.2953 - val_loss: 21.8222 - val_acc: 0.0000e+00
Content Manually Removed for Brevity.
Epoch 98/100
3996/3996 [==============================] - 2s - loss: 9.5470 - acc
: 0.2928 - val_loss: 29.8453 - val_acc: 0.0000e+00
Epoch 99/100
3996/3996 [==============================] - 2s - loss: 9.5449 - acc
: 0.2953 - val_loss: 29.7558 - val_acc: 0.0000e+00
Epoch 100/100
3996/3996 [==============================] - 2s - loss: 9.5439 - acc
: 0.2950 - val_loss: 29.9154 - val_acc: 0.0000e+00
Time to train model with  100  epochs is :  263.02725482  seconds.
992/996 [============================>.] - ETA: 0sTest loss: 2.77296
321363
Test accuracy: 0.230923694779
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 134, 88, 16)       1216
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 44, 29, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 42, 27, 32)        4640
_________________________________________________________________
dropout_3 (Dropout)          (None, 42, 27, 32)        0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 9, 32)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4032)              0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                258112
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 7)                 455
=================================================================
Total params: 264,423
Trainable params: 264,423
Non-trainable params: 0
_________________________________________________________________
None
```




```
None
```




```
None
```

Baseline (Simple) Model with Data Augmentation

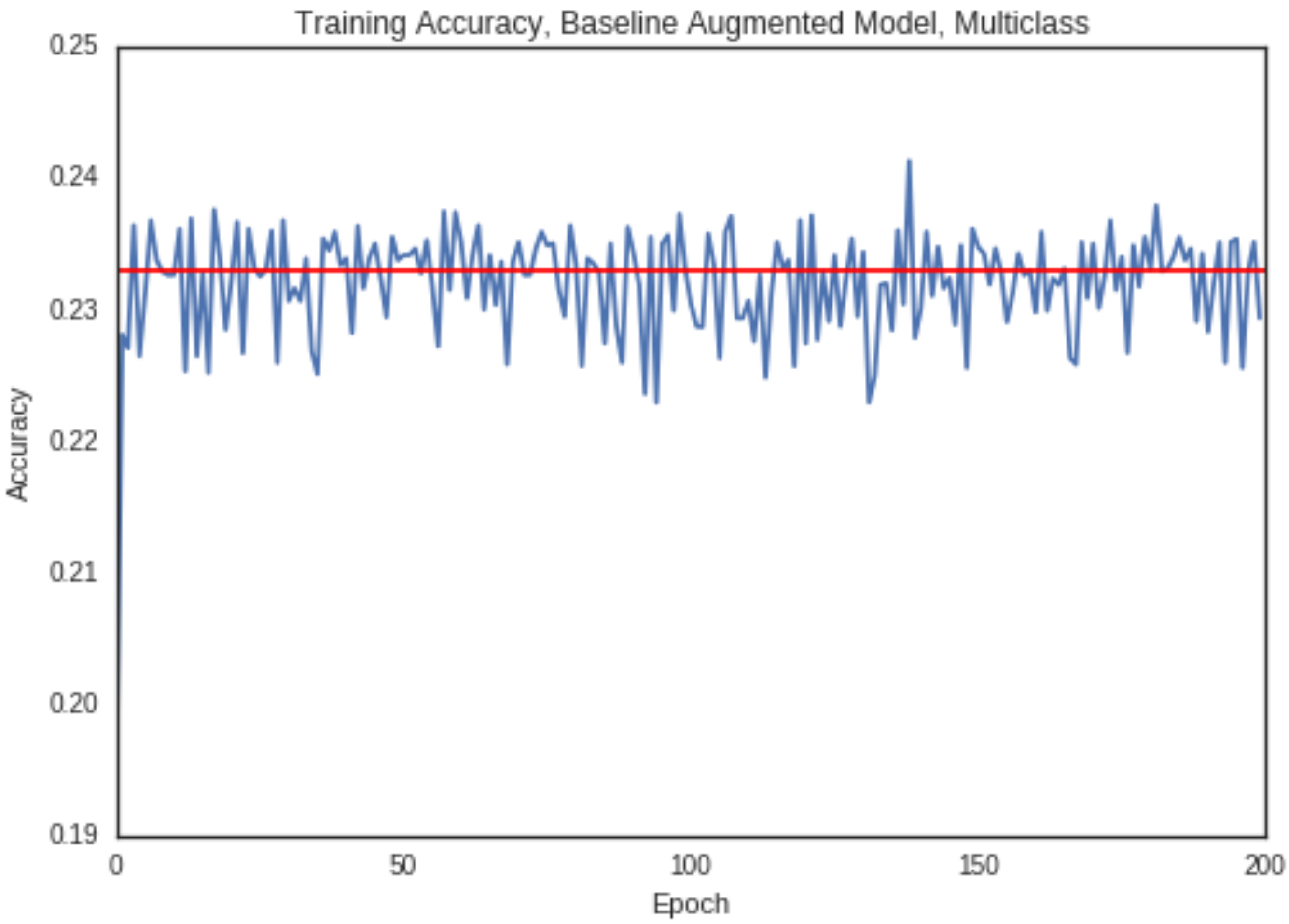
In [20]:

```
# Recreate the same model (to start "fresh") for using augmented dataset.
# create an empty network model
model_aug = Sequential()

# Input Layer
model_aug.add(Conv2D(16, kernel_size=input_kernel_size, activation=input_activat
ion_function, input_shape=input_shape))
model_aug.add(MaxPooling2D(pool_size=pool_size))

# Hidden Layer(s)
model_aug.add(Conv2D(32, kernel_size=hidden_kernel_size, activation=hidden_activ
ation_function))
model_aug.add(Dropout(0.25))
model_aug.add(MaxPooling2D(pool_size=pool_size))

# Classification layer
model_aug.add(Flatten())
model.add(Dense(64, activation=hidden_activation_function,
                activity_regularizer=regularizers.l1(0.1)))
model_aug.add(Dropout(0.5))
model_aug.add(Dense(classes, activation=final_activation_function))

# Compile the model.
model_aug.compile(loss=loss_method, optimizer=optimizer, metrics=[eval_metric])
```

In [21]:

```
# The actual training of the CNN using the parameters and model previously speci
fied.
# The validation set is a split of the stratified sampled training data.
experiment = 'Baseline Augmented Model, Multiclass'
accuracy, train_time = train_evaluate_CNN(x_train, y_train, x_test, y_test, mode
l_aug,
                                          batch_size, epochs=200,
                                          verbose=debug_flag,
                                          show_layers=1,
                                          validation_split=0.20,
                                          weights_filename='baseline_aug',
                                          plot_title=experiment,
                                          class_weights = scaled_weights,
                                          augment_data = True)
experiment_results = experiment_results.append({'Name' : experiment,
                                                'Test Accuracy' : accuracy,
                                                'Train Time': train_time}, ignor
e_index=True)
```

```
Epoch 1/200
16/16 [==============================] - 17s - loss: 2.8000 - acc: 0
.1940
Epoch 2/200
16/16 [==============================] - 16s - loss: 1.7984 - acc: 0
.2272
Content Manually Removed for Brevity.
Epoch 198/200
16/16 [==============================] - 16s - loss: 1.7769 - acc: 0
.2332
Epoch 199/200
16/16 [==============================] - 16s - loss: 1.7776 - acc: 0
.2354
Epoch 200/200
16/16 [==============================] - 16s - loss: 1.7778 - acc: 0
.2294
Time to train model with  200  epochs is :  3356.23466086  seconds.
992/996 [============================>.] - ETA: 0sTest loss: 1.78171
567457
Test accuracy: 0.232931726908
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_5 (Conv2D)            (None, 134, 88, 16)       1216      
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 44, 29, 16)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 42, 27, 32)        4640      
_________________________________________________________________
dropout_5 (Dropout)          (None, 42, 27, 32)        0         
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 9, 32)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 4032)              0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 4032)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 7)                 28231     
=================================================================
Total params: 34,087
Trainable params: 34,087
Non-trainable params: 0
_________________________________________________________________
None
```




```
None
```

Baseline Model with Additional Layer

In [22]:

```
# create an empty network model
model2 = Sequential()

# Input Layer
model2.add(Conv2D(16, kernel_size=input_kernel_size, activation=input_activation
_function, input_shape=input_shape))
model2.add(MaxPooling2D(pool_size=pool_size))

# Hidden Layer(s)
model2.add(Conv2D(32, kernel_size=hidden_kernel_size, activation=hidden_activati
on_function))
model2.add(Dropout(0.25))
model2.add(MaxPooling2D(pool_size=pool_size))

model2.add(Conv2D(48, kernel_size=hidden_kernel_size, activation=hidden_activati
on_function))
model2.add(Dropout(0.25))
model2.add(MaxPooling2D(pool_size=pool_size))

# Classification layer
model2.add(Flatten())
model2.add(Dense(64, activation=hidden_activation_function))
model2.add(Dropout(0.5))
model2.add(Dense(classes, activation=final_activation_function))

# Compile the model.
model2.compile(loss=loss_method, optimizer=optimizer, metrics=[eval_metric])
```

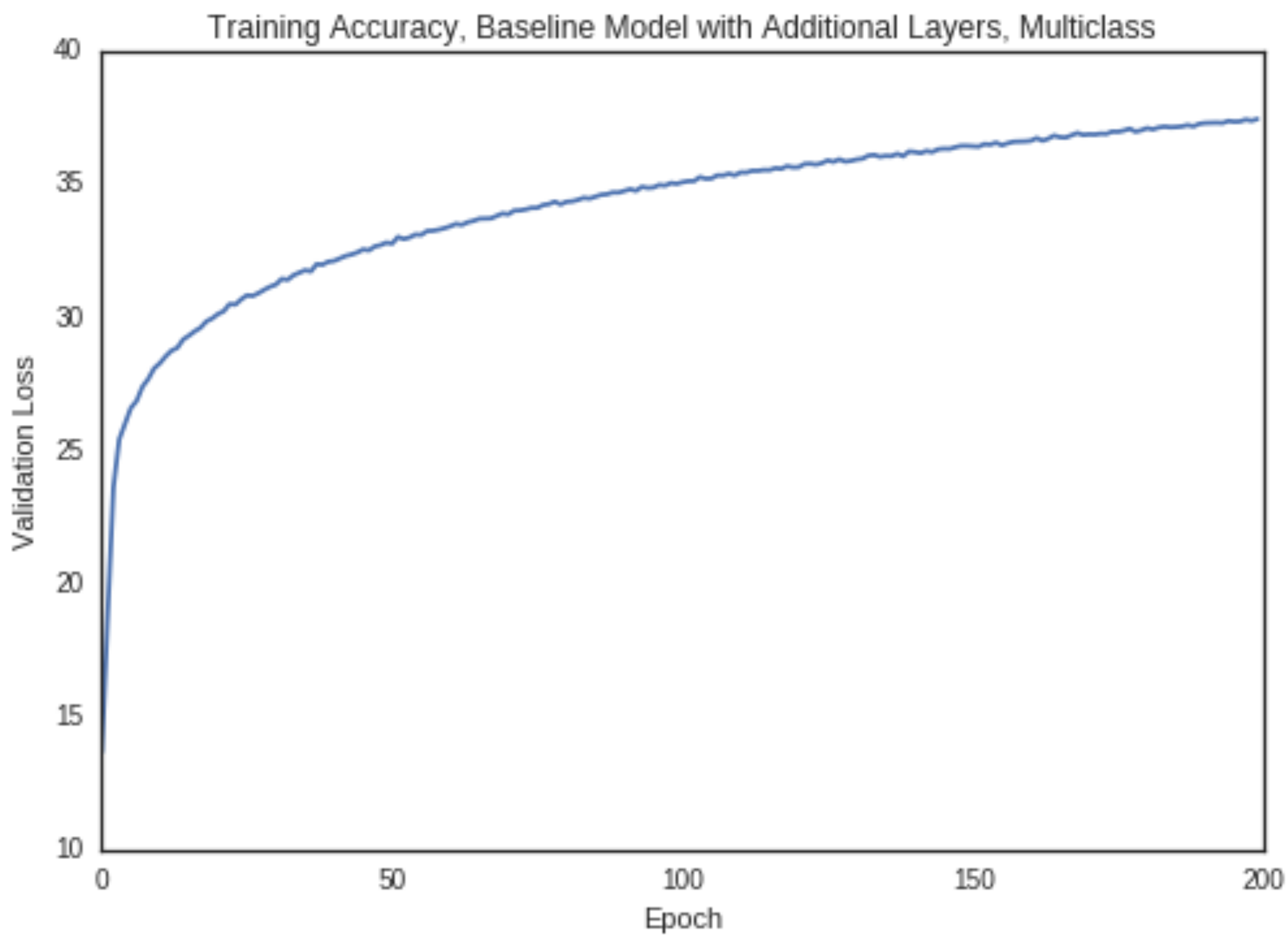
In [23]:

```
# The actual training of the CNN using the parameters and model previously speci
fied.
experiment = 'Baseline Model with Additional Layers, Multiclass'
accuracy, train_time = train_evaluate_CNN(x_train, y_train, x_test, y_test, mode
l2,
                                          batch_size, epochs=200,
                                          verbose=debug_flag, show_layers=True,
                                          validation_split=0.15,
                                          weights_filename='baseline_addl_layer'
,
                                          class_weights=scaled_weights,
                                          plot_title=experiment)
experiment_results = experiment_results.append({'Name' : experiment,
                                                'Test Accuracy' : accuracy,
                                                'Train Time': train_time}, ignor
e_index=True)
```

```
Train on 4246 samples, validate on 750 samples
Epoch 1/200
4246/4246 [==============================] - 3s - loss: 36.1881 - ac
c: 0.2602 - val_loss: 13.6715 - val_acc: 0.0000e+00
```

```
Epoch 2/200
4246/4246 [==============================] - 2s - loss: 11.3137 - ac
c: 0.2574 - val_loss: 18.7470 - val_acc: 0.0000e+00
Content Manually Removed for Brevity.
Epoch 198/200
4246/4246 [==============================] - 2s - loss: 9.6200 - acc
: 0.2715 - val_loss: 37.4238 - val_acc: 0.0000e+00
Epoch 199/200
4246/4246 [==============================] - 2s - loss: 9.6161 - acc
: 0.2701 - val_loss: 37.3785 - val_acc: 0.0000e+00
Epoch 200/200
4246/4246 [==============================] - 2s - loss: 9.6138 - acc
: 0.2767 - val_loss: 37.4442 - val_acc: 0.0000e+00
Time to train model with  200  epochs is :  555.364931822  seconds.
992/996 [============================>.] - ETA: 0sTest loss: 2.77694
278932
Test accuracy: 0.230923694779
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 134, 88, 16)       1216
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 44, 29, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 42, 27, 32)        4640
_________________________________________________________________
dropout_7 (Dropout)          (None, 42, 27, 32)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 9, 32)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 12, 7, 48)         13872
_________________________________________________________________
dropout_8 (Dropout)          (None, 12, 7, 48)         0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 4, 2, 48)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 384)               0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                24640
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 7)                 455
=================================================================
Total params: 44,823
Trainable params: 44,823
Non-trainable params: 0
_________________________________________________________________
None
```

None



None

Tune an existing CNN (Multi-class)

The following code is used with modification from the sample InceptionV3 code from the Keras documentation, https://keras.io/applications/ (https://keras.io/applications/)

In [24]:

```
# Create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer
predictions = Dense(classes, activation=final_activation_function)(x)

# this is the model we will train
model_tune = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model_tune.compile(optimizer='rmsprop', loss=loss_method,  metrics=[eval_metric]
)
```

In [25]:

```
# train the model on the new data for a few epochs
# fine-tune the model
experiment = 'Initial Tune of InceptionV3, Multiclass'
accuracy, train_time = train_evaluate_CNN(x_train, y_train, x_test, y_test, mode
l_tune,
                                          batch_size, epochs=10,
                                          verbose=debug_flag, show_layers=True,
                                          validation_split=0.20,
                                          weights_filename='inceptionv3_initial'
,
                                          class_weights=scaled_weights,
                                          plot_title=experiment)
experiment_results = experiment_results.append({'Name' : experiment,
                                                'Test Accuracy' : accuracy,
                                                'Train Time': train_time}, ignor
e_index=True)
```

```
Train on 3996 samples, validate on 1000 samples
Epoch 1/10
3996/3996 [==============================] - 18s - loss: 25.9208 - a
cc: 0.2803 - val_loss: 53.4514 - val_acc: 1.0000e-03
Epoch 2/10
```

```
3996/3996 [==============================] - 7s - loss: 11.7539 - ac
c: 0.3418 - val_loss: 62.3071 - val_acc: 0.0000e+00
Epoch 3/10
3996/3996 [==============================] - 7s - loss: 11.1904 - ac
c: 0.3493 - val_loss: 51.1568 - val_acc: 1.0000e-03
Epoch 4/10
3996/3996 [==============================] - 7s - loss: 9.9062 - acc
: 0.3794 - val_loss: 47.0615 - val_acc: 0.2240
Epoch 5/10
3996/3996 [==============================] - 7s - loss: 9.1908 - acc
: 0.4464 - val_loss: 50.8395 - val_acc: 0.0230
Epoch 6/10
3996/3996 [==============================] - 7s - loss: 9.4595 - acc
: 0.4429 - val_loss: 46.9311 - val_acc: 0.1990
Epoch 7/10
3996/3996 [==============================] - 7s - loss: 9.1164 - acc
: 0.4437 - val_loss: 48.8320 - val_acc: 0.0040
Epoch 8/10
3996/3996 [==============================] - 7s - loss: 9.1507 - acc
: 0.4535 - val_loss: 43.8139 - val_acc: 0.1840
Epoch 9/10
3996/3996 [==============================] - 7s - loss: 7.7951 - acc
: 0.5060 - val_loss: 51.8179 - val_acc: 0.0000e+00
Epoch 10/10
3996/3996 [==============================] - 7s - loss: 7.9258 - acc
: 0.4992 - val_loss: 44.9243 - val_acc: 0.0410
Time to train model with  10  epochs is :  89.0398108959  seconds.
996/996 [==============================] - 3s
Test loss: 3.71217436197
Test accuracy: 0.281124497992
____________________________________________________________________
________________________________
Layer (type)                     Output Shape          Param #     C
onnected to
====================================================================
================================
input_1 (InputLayer)             (None, None, None, 3) 0
____________________________________________________________________
________________________________
conv2d_10 (Conv2D)               (None, None, None, 32 864
____________________________________________________________________
________________________________
batch_normalization_1 (BatchNorm (None, None, None, 32 96
____________________________________________________________________
________________________________
activation_1 (Activation)        (None, None, None, 32 0
____________________________________________________________________
________________________________
conv2d_11 (Conv2D)               (None, None, None, 32 9216
____________________________________________________________________
________________________________
batch_normalization_2 (BatchNorm (None, None, None, 32 96
____________________________________________________________________
```

```
________________________________
activation_2 (Activation)        (None, None, None, 32 0
__________________________________________________________________
________________________________
conv2d_12 (Conv2D)               (None, None, None, 64 18432
__________________________________________________________________
________________________________
batch_normalization_3 (BatchNorm (None, None, None, 64 192
__________________________________________________________________
________________________________
activation_3 (Activation)        (None, None, None, 64 0
__________________________________________________________________
________________________________
max_pooling2d_10 (MaxPooling2D)  (None, None, None, 64 0
__________________________________________________________________
________________________________
conv2d_13 (Conv2D)               (None, None, None, 80 5120
__________________________________________________________________
________________________________
batch_normalization_4 (BatchNorm (None, None, None, 80 240
__________________________________________________________________
________________________________
activation_4 (Activation)        (None, None, None, 80 0
__________________________________________________________________
________________________________
conv2d_14 (Conv2D)               (None, None, None, 19 138240
__________________________________________________________________
________________________________
batch_normalization_5 (BatchNorm (None, None, None, 19 576
__________________________________________________________________
________________________________
activation_5 (Activation)        (None, None, None, 19 0
__________________________________________________________________
________________________________
max_pooling2d_11 (MaxPooling2D)  (None, None, None, 19 0
__________________________________________________________________
________________________________
conv2d_18 (Conv2D)               (None, None, None, 64 12288
__________________________________________________________________
________________________________
batch_normalization_9 (BatchNorm (None, None, None, 64 192
__________________________________________________________________
________________________________
activation_9 (Activation)        (None, None, None, 64 0
__________________________________________________________________
________________________________
conv2d_16 (Conv2D)               (None, None, None, 48 9216
__________________________________________________________________
________________________________
conv2d_19 (Conv2D)               (None, None, None, 96 55296
__________________________________________________________________
________________________________
batch_normalization_7 (BatchNorm (None, None, None, 48 144
```

batch_normalization_10 (BatchNor	(None, None, None, 96	288
activation_7 (Activation)	(None, None, None, 48	0
activation_10 (Activation)	(None, None, None, 96	0
average_pooling2d_1 (AveragePool	(None, None, None, 19	0
conv2d_15 (Conv2D)	(None, None, None, 64	12288
conv2d_17 (Conv2D)	(None, None, None, 64	76800
conv2d_20 (Conv2D)	(None, None, None, 96	82944
conv2d_21 (Conv2D)	(None, None, None, 32	6144
batch_normalization_6 (BatchNorm	(None, None, None, 64	192
batch_normalization_8 (BatchNorm	(None, None, None, 64	192
batch_normalization_11 (BatchNor	(None, None, None, 96	288
batch_normalization_12 (BatchNor	(None, None, None, 32	96
activation_6 (Activation)	(None, None, None, 64	0
activation_8 (Activation)	(None, None, None, 64	0
activation_11 (Activation)	(None, None, None, 96	0
activation_12 (Activation)	(None, None, None, 32	0
mixed0 (Concatenate)	(None, None, None, 25	0

```
conv2d_25 (Conv2D)               (None, None, None, 64 16384
________________________________________________________________
batch_normalization_16 (BatchNor (None, None, None, 64 192
________________________________________________________________
activation_16 (Activation)       (None, None, None, 64 0
________________________________________________________________
conv2d_23 (Conv2D)               (None, None, None, 48 12288
________________________________________________________________
conv2d_26 (Conv2D)               (None, None, None, 96 55296
________________________________________________________________
batch_normalization_14 (BatchNor (None, None, None, 48 144
________________________________________________________________
batch_normalization_17 (BatchNor (None, None, None, 96 288
________________________________________________________________
activation_14 (Activation)       (None, None, None, 48 0
________________________________________________________________
activation_17 (Activation)       (None, None, None, 96 0
________________________________________________________________
average_pooling2d_2 (AveragePool (None, None, None, 25 0
________________________________________________________________
conv2d_22 (Conv2D)               (None, None, None, 64 16384
________________________________________________________________
conv2d_24 (Conv2D)               (None, None, None, 64 76800
________________________________________________________________
conv2d_27 (Conv2D)               (None, None, None, 96 82944
________________________________________________________________
conv2d_28 (Conv2D)               (None, None, None, 64 16384
________________________________________________________________
batch_normalization_13 (BatchNor (None, None, None, 64 192
________________________________________________________________
batch_normalization_15 (BatchNor (None, None, None, 64 192
________________________________________________________________
batch_normalization_18 (BatchNor (None, None, None, 96 288
________________________________________________________________
batch_normalization_19 (BatchNor (None, None, None, 64 192
________________________________________________________________
```

```
activation_13 (Activation)       (None, None, None, 64 0

activation_15 (Activation)       (None, None, None, 64 0

activation_18 (Activation)       (None, None, None, 96 0

activation_19 (Activation)       (None, None, None, 64 0

mixed1 (Concatenate)             (None, None, None, 28 0

conv2d_32 (Conv2D)               (None, None, None, 64 18432

batch_normalization_23 (BatchNor (None, None, None, 64 192

activation_23 (Activation)       (None, None, None, 64 0

conv2d_30 (Conv2D)               (None, None, None, 48 13824

conv2d_33 (Conv2D)               (None, None, None, 96 55296

batch_normalization_21 (BatchNor (None, None, None, 48 144

batch_normalization_24 (BatchNor (None, None, None, 96 288

activation_21 (Activation)       (None, None, None, 48 0

activation_24 (Activation)       (None, None, None, 96 0

average_pooling2d_3 (AveragePool (None, None, None, 28 0

conv2d_29 (Conv2D)               (None, None, None, 64 18432

conv2d_31 (Conv2D)               (None, None, None, 64 76800

conv2d_34 (Conv2D)               (None, None, None, 96 82944
```

conv2d_35 (Conv2D)	(None, None, None, 64	18432
batch_normalization_20 (BatchNor	(None, None, None, 64	192
batch_normalization_22 (BatchNor	(None, None, None, 64	192
batch_normalization_25 (BatchNor	(None, None, None, 96	288
batch_normalization_26 (BatchNor	(None, None, None, 64	192
activation_20 (Activation)	(None, None, None, 64	0
activation_22 (Activation)	(None, None, None, 64	0
activation_25 (Activation)	(None, None, None, 96	0
activation_26 (Activation)	(None, None, None, 64	0
mixed2 (Concatenate)	(None, None, None, 28	0
conv2d_37 (Conv2D)	(None, None, None, 64	18432
batch_normalization_28 (BatchNor	(None, None, None, 64	192
activation_28 (Activation)	(None, None, None, 64	0
conv2d_38 (Conv2D)	(None, None, None, 96	55296
batch_normalization_29 (BatchNor	(None, None, None, 96	288
activation_29 (Activation)	(None, None, None, 96	0
conv2d_36 (Conv2D)	(None, None, None, 38	995328

```
conv2d_39 (Conv2D)              (None, None, None, 96 82944
________________________________________________________________________
________________________________
batch_normalization_27 (BatchNor (None, None, None, 38 1152
________________________________________________________________________
________________________________
batch_normalization_30 (BatchNor (None, None, None, 96 288
________________________________________________________________________
________________________________
activation_27 (Activation)      (None, None, None, 38 0
________________________________________________________________________
________________________________
activation_30 (Activation)      (None, None, None, 96 0
________________________________________________________________________
________________________________
max_pooling2d_12 (MaxPooling2D) (None, None, None, 28 0
________________________________________________________________________
________________________________
mixed3 (Concatenate)            (None, None, None, 76 0
________________________________________________________________________
________________________________
conv2d_44 (Conv2D)              (None, None, None, 12 98304
________________________________________________________________________
________________________________
batch_normalization_35 (BatchNor (None, None, None, 12 384
________________________________________________________________________
________________________________
activation_35 (Activation)      (None, None, None, 12 0
________________________________________________________________________
________________________________
conv2d_45 (Conv2D)              (None, None, None, 12 114688
________________________________________________________________________
________________________________
batch_normalization_36 (BatchNor (None, None, None, 12 384
________________________________________________________________________
________________________________
activation_36 (Activation)      (None, None, None, 12 0
________________________________________________________________________
________________________________
conv2d_41 (Conv2D)              (None, None, None, 12 98304
________________________________________________________________________
________________________________
conv2d_46 (Conv2D)              (None, None, None, 12 114688
________________________________________________________________________
________________________________
batch_normalization_32 (BatchNor (None, None, None, 12 384
________________________________________________________________________
________________________________
batch_normalization_37 (BatchNor (None, None, None, 12 384
________________________________________________________________________
________________________________
activation_32 (Activation)      (None, None, None, 12 0
________________________________________________________________________
```

```
activation_37 (Activation)       (None, None, None, 12 0
conv2d_42 (Conv2D)               (None, None, None, 12 114688
conv2d_47 (Conv2D)               (None, None, None, 12 114688
batch_normalization_33 (BatchNor (None, None, None, 12 384
batch_normalization_38 (BatchNor (None, None, None, 12 384
activation_33 (Activation)       (None, None, None, 12 0
activation_38 (Activation)       (None, None, None, 12 0
average_pooling2d_4 (AveragePool (None, None, None, 76 0
conv2d_40 (Conv2D)               (None, None, None, 19 147456
conv2d_43 (Conv2D)               (None, None, None, 19 172032
conv2d_48 (Conv2D)               (None, None, None, 19 172032
conv2d_49 (Conv2D)               (None, None, None, 19 147456
batch_normalization_31 (BatchNor (None, None, None, 19 576
batch_normalization_34 (BatchNor (None, None, None, 19 576
batch_normalization_39 (BatchNor (None, None, None, 19 576
batch_normalization_40 (BatchNor (None, None, None, 19 576
activation_31 (Activation)       (None, None, None, 19 0
activation_34 (Activation)       (None, None, None, 19 0
```

```
activation_39 (Activation)       (None, None, None, 19 0
activation_40 (Activation)       (None, None, None, 19 0
mixed4 (Concatenate)             (None, None, None, 76 0
conv2d_54 (Conv2D)               (None, None, None, 16 122880
batch_normalization_45 (BatchNor (None, None, None, 16 480
activation_45 (Activation)       (None, None, None, 16 0
conv2d_55 (Conv2D)               (None, None, None, 16 179200
batch_normalization_46 (BatchNor (None, None, None, 16 480
activation_46 (Activation)       (None, None, None, 16 0
conv2d_51 (Conv2D)               (None, None, None, 16 122880
conv2d_56 (Conv2D)               (None, None, None, 16 179200
batch_normalization_42 (BatchNor (None, None, None, 16 480
batch_normalization_47 (BatchNor (None, None, None, 16 480
activation_42 (Activation)       (None, None, None, 16 0
activation_47 (Activation)       (None, None, None, 16 0
conv2d_52 (Conv2D)               (None, None, None, 16 179200
conv2d_57 (Conv2D)               (None, None, None, 16 179200
```

```
batch_normalization_43 (BatchNor (None, None, None, 16 480
________________________________________________________________________________
________________________________
batch_normalization_48 (BatchNor (None, None, None, 16 480
________________________________________________________________________________
________________________________
activation_43 (Activation)       (None, None, None, 16 0
________________________________________________________________________________
________________________________
activation_48 (Activation)       (None, None, None, 16 0
________________________________________________________________________________
________________________________
average_pooling2d_5 (AveragePool (None, None, None, 76 0
________________________________________________________________________________
________________________________
conv2d_50 (Conv2D)               (None, None, None, 19 147456
________________________________________________________________________________
________________________________
conv2d_53 (Conv2D)               (None, None, None, 19 215040
________________________________________________________________________________
________________________________
conv2d_58 (Conv2D)               (None, None, None, 19 215040
________________________________________________________________________________
________________________________
conv2d_59 (Conv2D)               (None, None, None, 19 147456
________________________________________________________________________________
________________________________
batch_normalization_41 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
batch_normalization_44 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
batch_normalization_49 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
batch_normalization_50 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
activation_41 (Activation)       (None, None, None, 19 0
________________________________________________________________________________
________________________________
activation_44 (Activation)       (None, None, None, 19 0
________________________________________________________________________________
________________________________
activation_49 (Activation)       (None, None, None, 19 0
________________________________________________________________________________
________________________________
activation_50 (Activation)       (None, None, None, 19 0
________________________________________________________________________________
________________________________
mixed5 (Concatenate)             (None, None, None, 76 0
________________________________________________________________________________
```

```
conv2d_64 (Conv2D)              (None, None, None, 16 122880
________________________________________________________________
batch_normalization_55 (BatchNor (None, None, None, 16 480
________________________________________________________________
activation_55 (Activation)      (None, None, None, 16 0
________________________________________________________________
conv2d_65 (Conv2D)              (None, None, None, 16 179200
________________________________________________________________
batch_normalization_56 (BatchNor (None, None, None, 16 480
________________________________________________________________
activation_56 (Activation)      (None, None, None, 16 0
________________________________________________________________
conv2d_61 (Conv2D)              (None, None, None, 16 122880
________________________________________________________________
conv2d_66 (Conv2D)              (None, None, None, 16 179200
________________________________________________________________
batch_normalization_52 (BatchNor (None, None, None, 16 480
________________________________________________________________
batch_normalization_57 (BatchNor (None, None, None, 16 480
________________________________________________________________
activation_52 (Activation)      (None, None, None, 16 0
________________________________________________________________
activation_57 (Activation)      (None, None, None, 16 0
________________________________________________________________
conv2d_62 (Conv2D)              (None, None, None, 16 179200
________________________________________________________________
conv2d_67 (Conv2D)              (None, None, None, 16 179200
________________________________________________________________
batch_normalization_53 (BatchNor (None, None, None, 16 480
________________________________________________________________
batch_normalization_58 (BatchNor (None, None, None, 16 480
________________________________________________________________
activation_53 (Activation)      (None, None, None, 16 0
________________________________________________________________
activation_58 (Activation)      (None, None, None, 16 0
```

```
average_pooling2d_6 (AveragePool (None, None, None, 76 0

conv2d_60 (Conv2D)               (None, None, None, 19 147456

conv2d_63 (Conv2D)               (None, None, None, 19 215040

conv2d_68 (Conv2D)               (None, None, None, 19 215040

conv2d_69 (Conv2D)               (None, None, None, 19 147456

batch_normalization_51 (BatchNor (None, None, None, 19 576

batch_normalization_54 (BatchNor (None, None, None, 19 576

batch_normalization_59 (BatchNor (None, None, None, 19 576

batch_normalization_60 (BatchNor (None, None, None, 19 576

activation_51 (Activation)       (None, None, None, 19 0

activation_54 (Activation)       (None, None, None, 19 0

activation_59 (Activation)       (None, None, None, 19 0

activation_60 (Activation)       (None, None, None, 19 0

mixed6 (Concatenate)             (None, None, None, 76 0

conv2d_74 (Conv2D)               (None, None, None, 19 147456

batch_normalization_65 (BatchNor (None, None, None, 19 576

activation_65 (Activation)       (None, None, None, 19 0
```

```
conv2d_75 (Conv2D)              (None, None, None, 19 258048
________________________________________________________________________________
________________________________
batch_normalization_66 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
activation_66 (Activation)      (None, None, None, 19 0
________________________________________________________________________________
________________________________
conv2d_71 (Conv2D)              (None, None, None, 19 147456
________________________________________________________________________________
________________________________
conv2d_76 (Conv2D)              (None, None, None, 19 258048
________________________________________________________________________________
________________________________
batch_normalization_62 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
batch_normalization_67 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
activation_62 (Activation)      (None, None, None, 19 0
________________________________________________________________________________
________________________________
activation_67 (Activation)      (None, None, None, 19 0
________________________________________________________________________________
________________________________
conv2d_72 (Conv2D)              (None, None, None, 19 258048
________________________________________________________________________________
________________________________
conv2d_77 (Conv2D)              (None, None, None, 19 258048
________________________________________________________________________________
________________________________
batch_normalization_63 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
batch_normalization_68 (BatchNor (None, None, None, 19 576
________________________________________________________________________________
________________________________
activation_63 (Activation)      (None, None, None, 19 0
________________________________________________________________________________
________________________________
activation_68 (Activation)      (None, None, None, 19 0
________________________________________________________________________________
________________________________
average_pooling2d_7 (AveragePool (None, None, None, 76 0
________________________________________________________________________________
________________________________
conv2d_70 (Conv2D)              (None, None, None, 19 147456
________________________________________________________________________________
________________________________
conv2d_73 (Conv2D)              (None, None, None, 19 258048
________________________________________________________________________________
```

```
________________________________
conv2d_78 (Conv2D)               (None, None, None, 19 258048
__________________________________________________________________
________________________________
conv2d_79 (Conv2D)               (None, None, None, 19 147456
__________________________________________________________________
________________________________
batch_normalization_61 (BatchNor (None, None, None, 19 576
__________________________________________________________________
________________________________
batch_normalization_64 (BatchNor (None, None, None, 19 576
__________________________________________________________________
________________________________
batch_normalization_69 (BatchNor (None, None, None, 19 576
__________________________________________________________________
________________________________
batch_normalization_70 (BatchNor (None, None, None, 19 576
__________________________________________________________________
________________________________
activation_61 (Activation)       (None, None, None, 19 0
__________________________________________________________________
________________________________
activation_64 (Activation)       (None, None, None, 19 0
__________________________________________________________________
________________________________
activation_69 (Activation)       (None, None, None, 19 0
__________________________________________________________________
________________________________
activation_70 (Activation)       (None, None, None, 19 0
__________________________________________________________________
________________________________
mixed7 (Concatenate)             (None, None, None, 76 0
__________________________________________________________________
________________________________
conv2d_82 (Conv2D)               (None, None, None, 19 147456
__________________________________________________________________
________________________________
batch_normalization_73 (BatchNor (None, None, None, 19 576
__________________________________________________________________
________________________________
activation_73 (Activation)       (None, None, None, 19 0
__________________________________________________________________
________________________________
conv2d_83 (Conv2D)               (None, None, None, 19 258048
__________________________________________________________________
________________________________
batch_normalization_74 (BatchNor (None, None, None, 19 576
__________________________________________________________________
________________________________
activation_74 (Activation)       (None, None, None, 19 0
__________________________________________________________________
________________________________
conv2d_80 (Conv2D)               (None, None, None, 19 147456
```

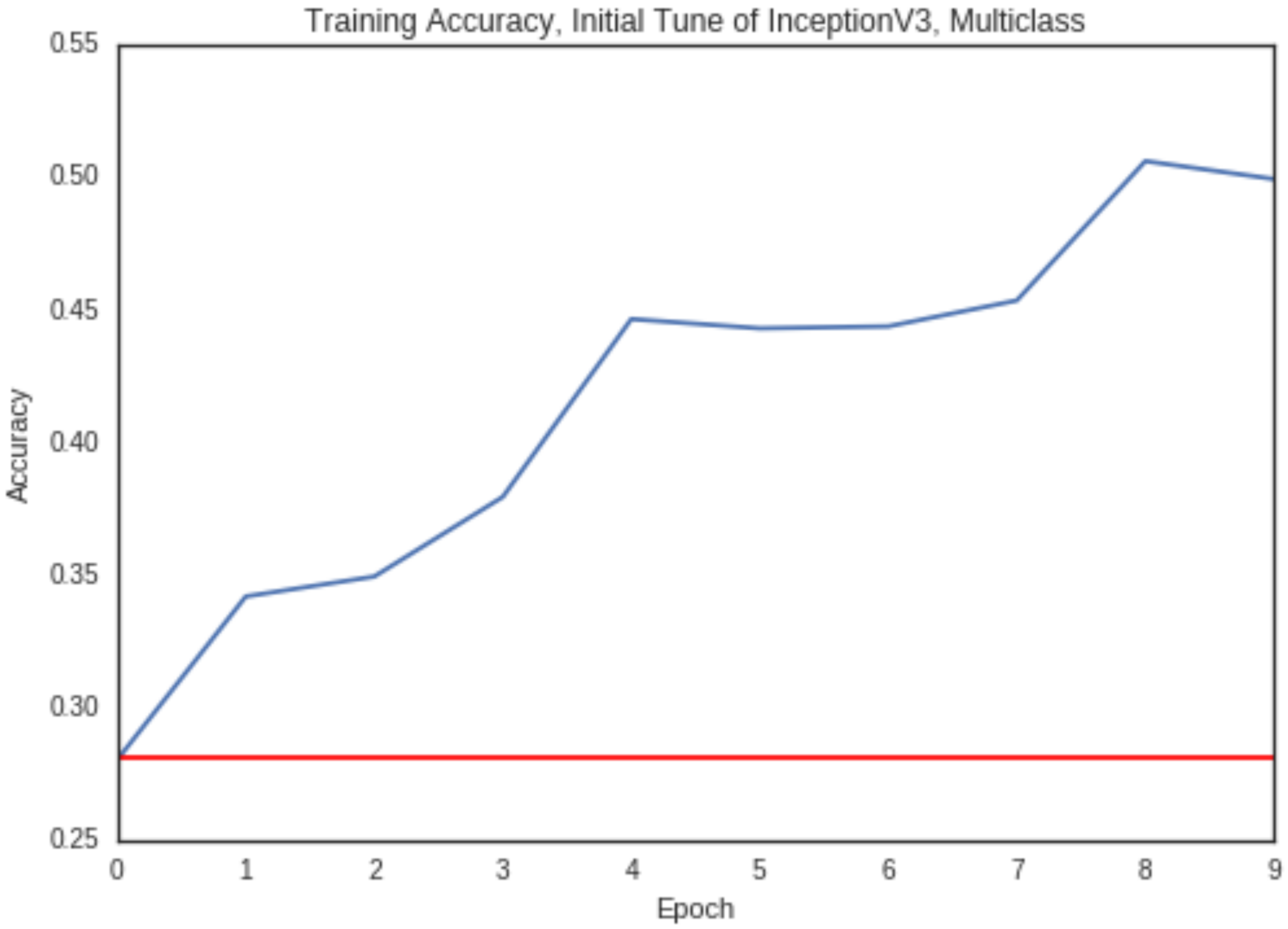
```
conv2d_84 (Conv2D)              (None, None, None, 19 258048
batch_normalization_71 (BatchNor (None, None, None, 19 576
batch_normalization_75 (BatchNor (None, None, None, 19 576
activation_71 (Activation)      (None, None, None, 19 0
activation_75 (Activation)      (None, None, None, 19 0
conv2d_81 (Conv2D)              (None, None, None, 32 552960
conv2d_85 (Conv2D)              (None, None, None, 19 331776
batch_normalization_72 (BatchNor (None, None, None, 32 960
batch_normalization_76 (BatchNor (None, None, None, 19 576
activation_72 (Activation)      (None, None, None, 32 0
activation_76 (Activation)      (None, None, None, 19 0
max_pooling2d_13 (MaxPooling2D)  (None, None, None, 76 0
mixed8 (Concatenate)            (None, None, None, 12 0
conv2d_90 (Conv2D)              (None, None, None, 44 573440
batch_normalization_81 (BatchNor (None, None, None, 44 1344
activation_81 (Activation)      (None, None, None, 44 0
conv2d_87 (Conv2D)              (None, None, None, 38 491520
```

```
conv2d_91 (Conv2D)              (None, None, None, 38 1548288
________________________________________________________________
batch_normalization_78 (BatchNor (None, None, None, 38 1152
________________________________________________________________
batch_normalization_82 (BatchNor (None, None, None, 38 1152
________________________________________________________________
activation_78 (Activation)      (None, None, None, 38 0
________________________________________________________________
activation_82 (Activation)      (None, None, None, 38 0
________________________________________________________________
conv2d_88 (Conv2D)              (None, None, None, 38 442368
________________________________________________________________
conv2d_89 (Conv2D)              (None, None, None, 38 442368
________________________________________________________________
conv2d_92 (Conv2D)              (None, None, None, 38 442368
________________________________________________________________
conv2d_93 (Conv2D)              (None, None, None, 38 442368
________________________________________________________________
average_pooling2d_8 (AveragePool (None, None, None, 12 0
________________________________________________________________
conv2d_86 (Conv2D)              (None, None, None, 32 409600
________________________________________________________________
batch_normalization_79 (BatchNor (None, None, None, 38 1152
________________________________________________________________
batch_normalization_80 (BatchNor (None, None, None, 38 1152
________________________________________________________________
batch_normalization_83 (BatchNor (None, None, None, 38 1152
________________________________________________________________
batch_normalization_84 (BatchNor (None, None, None, 38 1152
________________________________________________________________
conv2d_94 (Conv2D)              (None, None, None, 19 245760
________________________________________________________________
batch_normalization_77 (BatchNor (None, None, None, 32 960
________________________________________________________________
activation_79 (Activation)      (None, None, None, 38 0
________________________________________________________________
```

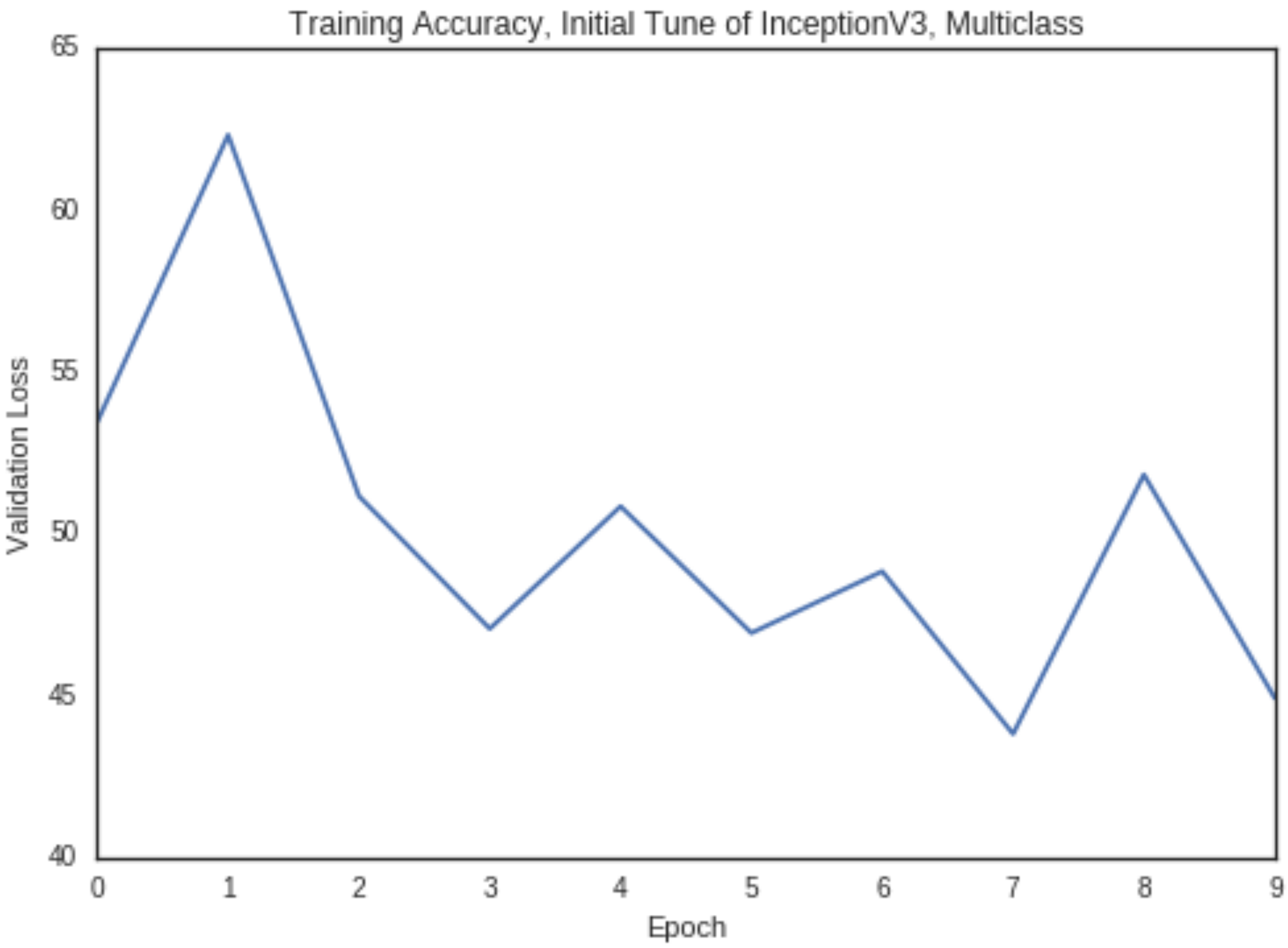
```
activation_80 (Activation)      (None, None, None, 38 0
activation_83 (Activation)      (None, None, None, 38 0
activation_84 (Activation)      (None, None, None, 38 0
batch_normalization_85 (BatchNor (None, None, None, 19 576
activation_77 (Activation)      (None, None, None, 32 0
mixed9_0 (Concatenate)          (None, None, None, 76 0
concatenate_1 (Concatenate)     (None, None, None, 76 0
activation_85 (Activation)      (None, None, None, 19 0
mixed9 (Concatenate)            (None, None, None, 20 0
conv2d_99 (Conv2D)              (None, None, None, 44 917504
batch_normalization_90 (BatchNor (None, None, None, 44 1344
activation_90 (Activation)      (None, None, None, 44 0
conv2d_96 (Conv2D)              (None, None, None, 38 786432
conv2d_100 (Conv2D)             (None, None, None, 38 1548288
batch_normalization_87 (BatchNor (None, None, None, 38 1152
batch_normalization_91 (BatchNor (None, None, None, 38 1152
activation_87 (Activation)      (None, None, None, 38 0
activation_91 (Activation)      (None, None, None, 38 0
```

```
conv2d_97 (Conv2D)              (None, None, None, 38 442368

conv2d_98 (Conv2D)              (None, None, None, 38 442368

conv2d_101 (Conv2D)             (None, None, None, 38 442368

conv2d_102 (Conv2D)             (None, None, None, 38 442368

average_pooling2d_9 (AveragePool (None, None, None, 20 0

conv2d_95 (Conv2D)              (None, None, None, 32 655360

batch_normalization_88 (BatchNor (None, None, None, 38 1152

batch_normalization_89 (BatchNor (None, None, None, 38 1152

batch_normalization_92 (BatchNor (None, None, None, 38 1152

batch_normalization_93 (BatchNor (None, None, None, 38 1152

conv2d_103 (Conv2D)             (None, None, None, 19 393216

batch_normalization_86 (BatchNor (None, None, None, 32 960

activation_88 (Activation)      (None, None, None, 38 0

activation_89 (Activation)      (None, None, None, 38 0

activation_92 (Activation)      (None, None, None, 38 0

activation_93 (Activation)      (None, None, None, 38 0

batch_normalization_94 (BatchNor (None, None, None, 19 576
```

```
activation_86 (Activation)       (None, None, None, 32 0
__________________________________________________________________________________________________
mixed9_1 (Concatenate)           (None, None, None, 76 0
__________________________________________________________________________________________________
concatenate_2 (Concatenate)      (None, None, None, 76 0
__________________________________________________________________________________________________
activation_94 (Activation)       (None, None, None, 19 0
__________________________________________________________________________________________________
mixed10 (Concatenate)            (None, None, None, 20 0
__________________________________________________________________________________________________
global_average_pooling2d_1 (Glob (None, 2048)          0
__________________________________________________________________________________________________
dense_9 (Dense)                  (None, 1024)          2098176
__________________________________________________________________________________________________
dense_10 (Dense)                 (None, 7)             7175
==================================================================================================
Total params: 23,908,135
Trainable params: 2,105,351
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
None
```



None



None

In [26]:

```
# at this point, the top layers are well trained and we can start fine-tuning
# convolutional layers from inception V3. We will freeze the bottom N layers
# and train the remaining top layers.

# let's visualize layer names and layer indices to see how many layers
# we should freeze:
for i, layer in enumerate(base_model.layers):
   print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 172 layers and unfreeze the rest:
for layer in model.layers[:172]:
   layer.trainable = False
for layer in model.layers[172:]:
   layer.trainable = True

# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
model.compile(optimizer=SGD(lr=0.0001, momentum=0.8), loss=loss_method,  metrics
=[eval_metric])
```

```
0 input_1
1 conv2d_10
2 batch_normalization_1
3 activation_1
```

```
4 conv2d_11
5 batch_normalization_2
6 activation_2
7 conv2d_12
8 batch_normalization_3
9 activation_3
10 max_pooling2d_10
11 conv2d_13
12 batch_normalization_4
13 activation_4
14 conv2d_14
15 batch_normalization_5
16 activation_5
17 max_pooling2d_11
18 conv2d_18
19 batch_normalization_9
20 activation_9
21 conv2d_16
22 conv2d_19
23 batch_normalization_7
24 batch_normalization_10
25 activation_7
26 activation_10
27 average_pooling2d_1
28 conv2d_15
29 conv2d_17
30 conv2d_20
31 conv2d_21
32 batch_normalization_6
33 batch_normalization_8
34 batch_normalization_11
35 batch_normalization_12
36 activation_6
37 activation_8
38 activation_11
39 activation_12
40 mixed0
41 conv2d_25
42 batch_normalization_16
43 activation_16
44 conv2d_23
45 conv2d_26
46 batch_normalization_14
47 batch_normalization_17
48 activation_14
49 activation_17
50 average_pooling2d_2
51 conv2d_22
52 conv2d_24
53 conv2d_27
54 conv2d_28
55 batch_normalization_13
56 batch_normalization_15
```

```
57 batch_normalization_18
58 batch_normalization_19
59 activation_13
60 activation_15
61 activation_18
62 activation_19
63 mixed1
64 conv2d_32
65 batch_normalization_23
66 activation_23
67 conv2d_30
68 conv2d_33
69 batch_normalization_21
70 batch_normalization_24
71 activation_21
72 activation_24
73 average_pooling2d_3
74 conv2d_29
75 conv2d_31
76 conv2d_34
77 conv2d_35
78 batch_normalization_20
79 batch_normalization_22
80 batch_normalization_25
81 batch_normalization_26
82 activation_20
83 activation_22
84 activation_25
85 activation_26
86 mixed2
87 conv2d_37
88 batch_normalization_28
89 activation_28
90 conv2d_38
91 batch_normalization_29
92 activation_29
93 conv2d_36
94 conv2d_39
95 batch_normalization_27
96 batch_normalization_30
97 activation_27
98 activation_30
99 max_pooling2d_12
100 mixed3
101 conv2d_44
102 batch_normalization_35
103 activation_35
104 conv2d_45
105 batch_normalization_36
106 activation_36
107 conv2d_41
108 conv2d_46
109 batch_normalization_32
```

```
110 batch_normalization_37
111 activation_32
112 activation_37
113 conv2d_42
114 conv2d_47
115 batch_normalization_33
116 batch_normalization_38
117 activation_33
118 activation_38
119 average_pooling2d_4
120 conv2d_40
121 conv2d_43
122 conv2d_48
123 conv2d_49
124 batch_normalization_31
125 batch_normalization_34
126 batch_normalization_39
127 batch_normalization_40
128 activation_31
129 activation_34
130 activation_39
131 activation_40
132 mixed4
133 conv2d_54
134 batch_normalization_45
135 activation_45
136 conv2d_55
137 batch_normalization_46
138 activation_46
139 conv2d_51
140 conv2d_56
141 batch_normalization_42
142 batch_normalization_47
143 activation_42
144 activation_47
145 conv2d_52
146 conv2d_57
147 batch_normalization_43
148 batch_normalization_48
149 activation_43
150 activation_48
151 average_pooling2d_5
152 conv2d_50
153 conv2d_53
154 conv2d_58
155 conv2d_59
156 batch_normalization_41
157 batch_normalization_44
158 batch_normalization_49
159 batch_normalization_50
160 activation_41
161 activation_44
162 activation_49
```

```
163 activation_50
164 mixed5
165 conv2d_64
166 batch_normalization_55
167 activation_55
168 conv2d_65
169 batch_normalization_56
170 activation_56
171 conv2d_61
172 conv2d_66
173 batch_normalization_52
174 batch_normalization_57
175 activation_52
176 activation_57
177 conv2d_62
178 conv2d_67
179 batch_normalization_53
180 batch_normalization_58
181 activation_53
182 activation_58
183 average_pooling2d_6
184 conv2d_60
185 conv2d_63
186 conv2d_68
187 conv2d_69
188 batch_normalization_51
189 batch_normalization_54
190 batch_normalization_59
191 batch_normalization_60
192 activation_51
193 activation_54
194 activation_59
195 activation_60
196 mixed6
197 conv2d_74
198 batch_normalization_65
199 activation_65
200 conv2d_75
201 batch_normalization_66
202 activation_66
203 conv2d_71
204 conv2d_76
205 batch_normalization_62
206 batch_normalization_67
207 activation_62
208 activation_67
209 conv2d_72
210 conv2d_77
211 batch_normalization_63
212 batch_normalization_68
213 activation_63
214 activation_68
215 average_pooling2d_7
```

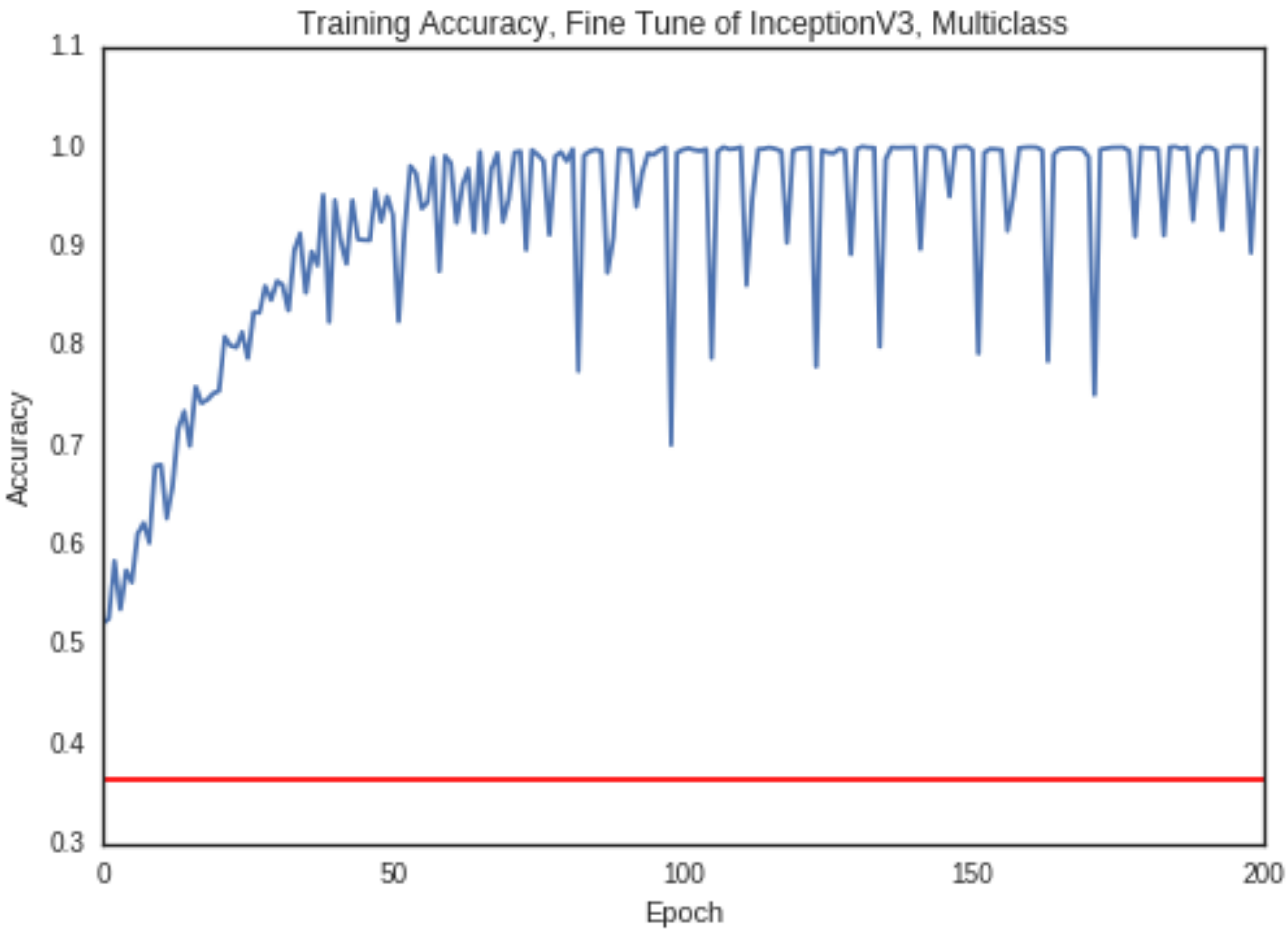
```
216 conv2d_70
217 conv2d_73
218 conv2d_78
219 conv2d_79
220 batch_normalization_61
221 batch_normalization_64
222 batch_normalization_69
223 batch_normalization_70
224 activation_61
225 activation_64
226 activation_69
227 activation_70
228 mixed7
229 conv2d_82
230 batch_normalization_73
231 activation_73
232 conv2d_83
233 batch_normalization_74
234 activation_74
235 conv2d_80
236 conv2d_84
237 batch_normalization_71
238 batch_normalization_75
239 activation_71
240 activation_75
241 conv2d_81
242 conv2d_85
243 batch_normalization_72
244 batch_normalization_76
245 activation_72
246 activation_76
247 max_pooling2d_13
248 mixed8
249 conv2d_90
250 batch_normalization_81
251 activation_81
252 conv2d_87
253 conv2d_91
254 batch_normalization_78
255 batch_normalization_82
256 activation_78
257 activation_82
258 conv2d_88
259 conv2d_89
260 conv2d_92
261 conv2d_93
262 average_pooling2d_8
263 conv2d_86
264 batch_normalization_79
265 batch_normalization_80
266 batch_normalization_83
267 batch_normalization_84
268 conv2d_94
```

```
269 batch_normalization_77
270 activation_79
271 activation_80
272 activation_83
273 activation_84
274 batch_normalization_85
275 activation_77
276 mixed9_0
277 concatenate_1
278 activation_85
279 mixed9
280 conv2d_99
281 batch_normalization_90
282 activation_90
283 conv2d_96
284 conv2d_100
285 batch_normalization_87
286 batch_normalization_91
287 activation_87
288 activation_91
289 conv2d_97
290 conv2d_98
291 conv2d_101
292 conv2d_102
293 average_pooling2d_9
294 conv2d_95
295 batch_normalization_88
296 batch_normalization_89
297 batch_normalization_92
298 batch_normalization_93
299 conv2d_103
300 batch_normalization_86
301 activation_88
302 activation_89
303 activation_92
304 activation_93
305 batch_normalization_94
306 activation_86
307 mixed9_1
308 concatenate_2
309 activation_94
310 mixed10
```

In [27]:

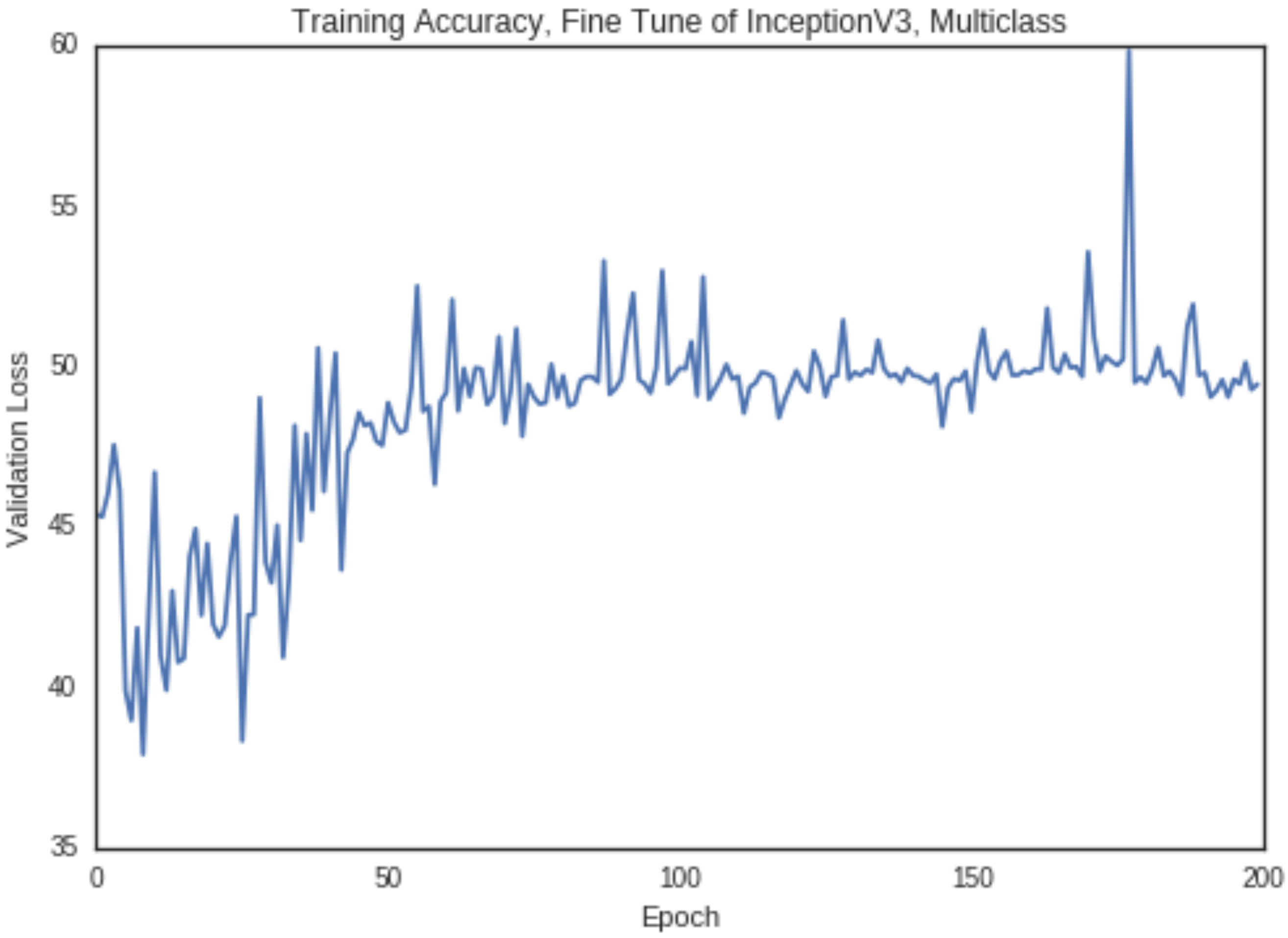
```
# Fine tune the new model, using the "fine" parameters of SGD previously defined
.
experiment = 'Fine Tune of InceptionV3, Multiclass'
accuracy, train_time = train_evaluate_CNN(x_train, y_train, x_test, y_test, mode
l_tune,
                                          batch_size, epochs=200,
                                          verbose=debug_flag, show_layers=False,
                                          validation_split=0.20,
                                          weights_filename='inceptionv3_finetune
',
                                          class_weights=scaled_weights,
                                          plot_title=experiment)
experiment_results = experiment_results.append({'Name' : experiment,
                                                'Test Accuracy' : accuracy,
                                                'Train Time': train_time}, ignor
e_index=True)
```

```
Train on 3996 samples, validate on 1000 samples
Epoch 1/200
3996/3996 [==============================] - 8s - loss: 7.9260 - acc
: 0.5210 - val_loss: 45.3977 - val_acc: 0.0020
Epoch 2/200
3996/3996 [==============================] - 7s - loss: 7.1913 - acc
: 0.5265 - val_loss: 45.3103 - val_acc: 0.0080
Content Manually Removed for Brevity.
Epoch 198/200
3996/3996 [==============================] - 7s - loss: 0.0427 - acc
: 0.9995 - val_loss: 50.1217 - val_acc: 0.1450
Epoch 199/200
3996/3996 [==============================] - 7s - loss: 3.1412 - acc
: 0.8934 - val_loss: 49.2745 - val_acc: 0.1530
Epoch 200/200
3996/3996 [==============================] - 7s - loss: 0.0584 - acc
: 0.9980 - val_loss: 49.4392 - val_acc: 0.1490
Time to train model with  200  epochs is :  1559.33376002  seconds.
992/996 [============================>.] - ETA: 0sTest loss: 4.57182
977477
Test accuracy: 0.364457831325
```



Training Accuracy, Fine Tune of InceptionV3, Multiclass
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Accuracy
0
50
100
150
200
Epoch

None

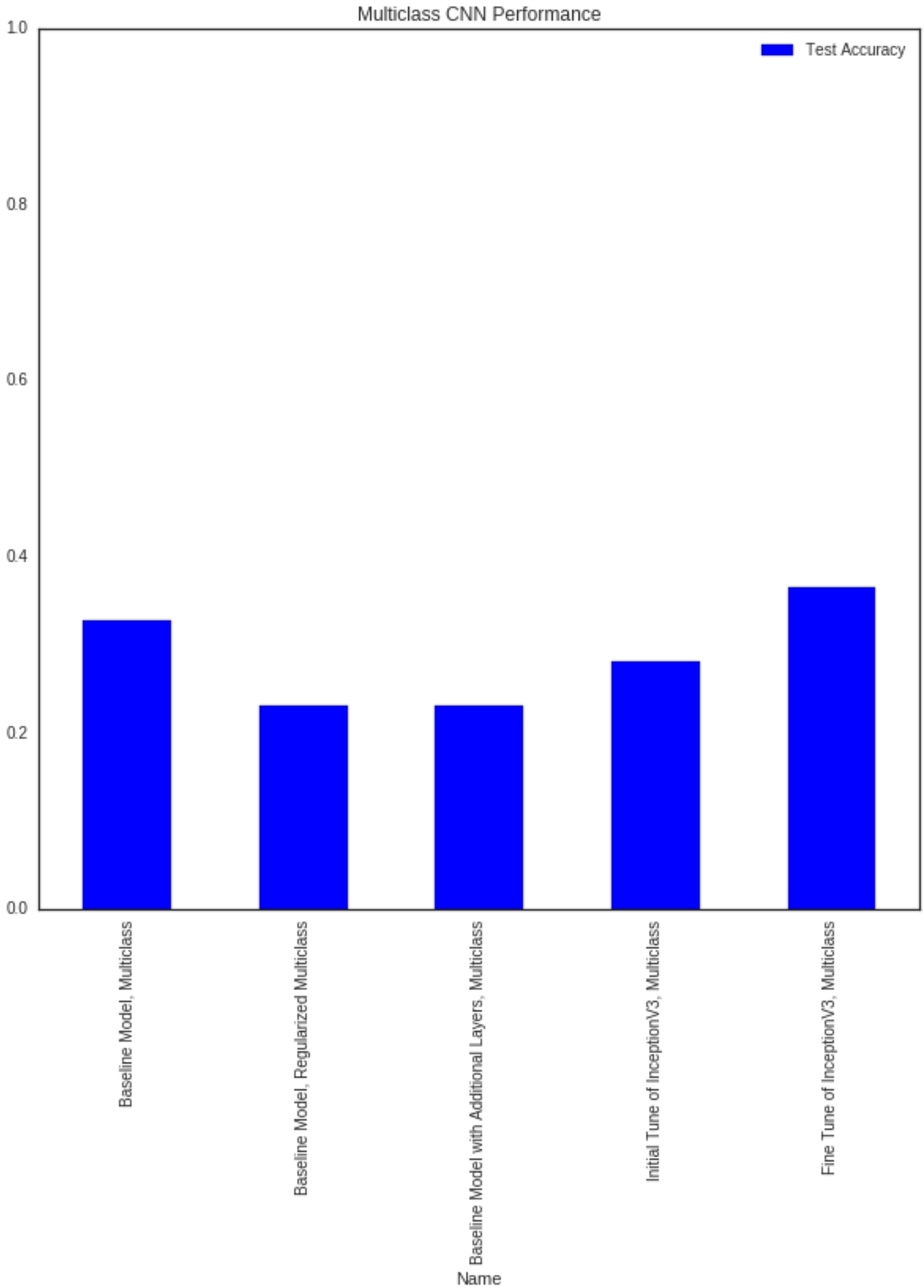


Training Accuracy, Fine Tune of InceptionV3, Multiclass
60
55
50
45
40
35
Validation Loss
0
50
100
150
200
Epoch

None

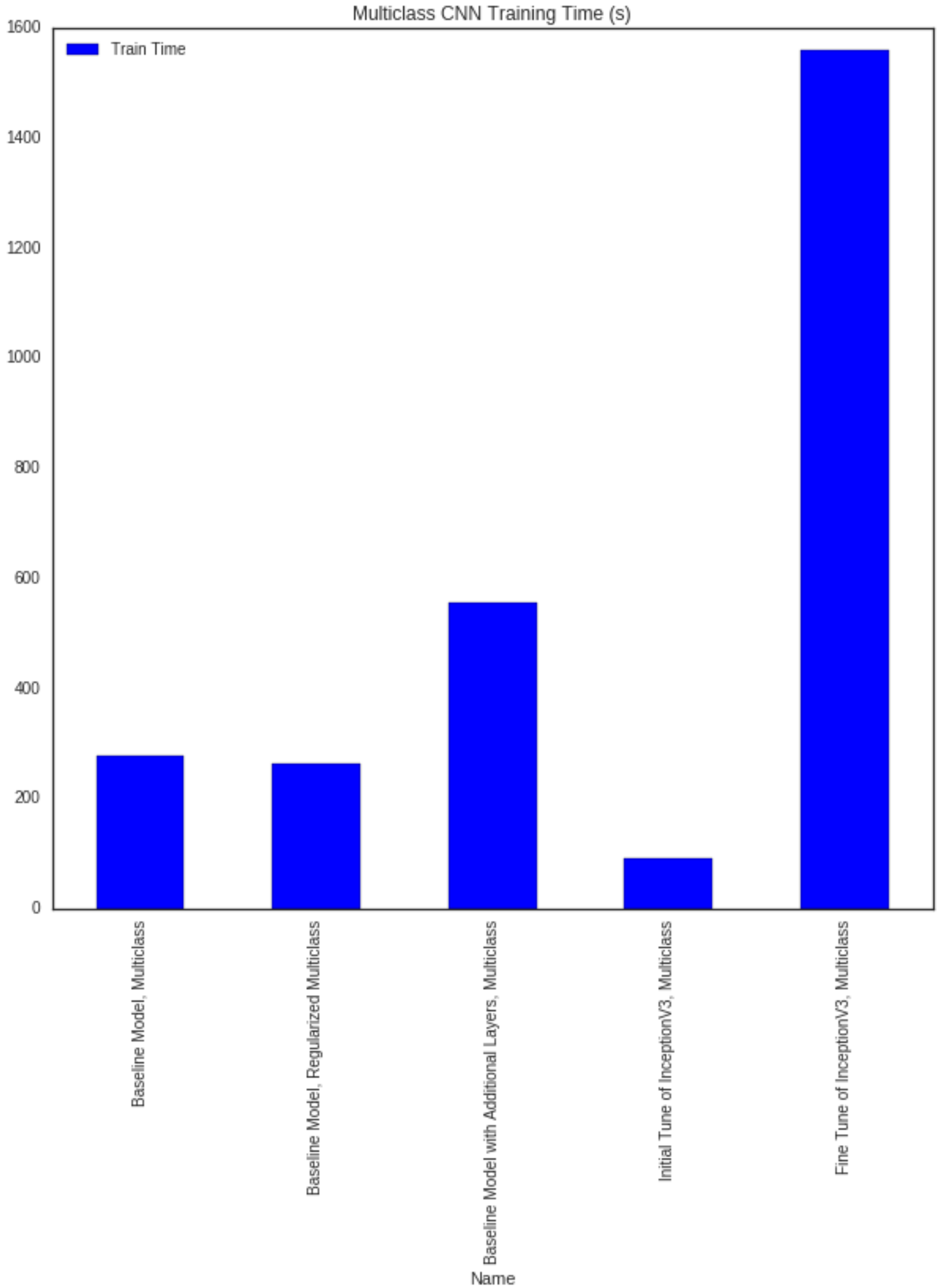
In [28]:

```
experiment_results.plot(x='Name', y='Test Accuracy', kind='bar',
                        title='Multiclass CNN Performance', ylim=(0,1.0), figsiz
e=(10,10));
```


Multiclass CNN Performance
Test Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Baseline Model, Multiclass
Baseline Model, Regularized Multiclass
Baseline Model with Additional Layers, Multiclass
Initial Tune of InceptionV3, Multiclass
Fine Tune of InceptionV3, Multiclass
Name

In [29]:

```
experiment_results.plot(x='Name', y='Train Time', kind='bar',
                        title='Multiclass CNN Training Time (s)', figsize=(10,10
));
```



Tuning Explored and Results Commentary

From the above results, we see that the baseline model without regularization nor class weighting steadily increased training accuracy per epoch, culminating to a peak of ~85%, however the validation loss function was increasing, indicating that the model is likely overfitting the training data. The accuracy on unseen test data (1000 movie posters) resulted in an accuracy of 32.7%. The parameters used were 100 epochs, with 15% of the 4000 stratified sample observations of training data used as validation, stochastic gradient descent with a learning rate of 0.1, relu activation functions and a final activation function of 'softmax'. Although dropout layers were employed to reduce overfitting, the model (based upon the shape of the validation loss) appears to have been overfitting, therefore the next experiment introduced regularization into the same model.

The next model employed regularization to the final activiation function and class weights, but otherwise unchanged from the previous model. The learning rate (slope of the training accuracy) seemed to oscillate about 0.29 (potential causes will be explained in the tuning section, below), indicating the model did not appear to be significantly learning. Futher exacerbating the results, the validation set loss metric was increasing across epochs, whereas one would hope that it would decrease over epochs. The resultant accuracy on the same unseen test set was only 23%, marginally better than a random guess of 1/classes = 14.3%.

The next model (not included in the performance summary plots) added synthetic data generation to the previous. This resulted in a significant reduction in training speed with an accuracy of 23%, indicating that the sampling and transformations of the original dataset did not improve the accuracy of the trained model. Given the simplicity (few layers) of the models explored so far, it was chosen to add more layers into the model for the next experiment.

The third model in the performance summary is identical to the second, with the addition of another trio of convolution, dropout and pooling layers within the hidden model. This resulted in no increase in test set accuracy at ~23%. This set of bespoke models, although out performing random guesses, is perhaps too simple to capture the information in the posters. As such, the next two models leverage a pre-trained CNN model with hundreds of layers.

The fourth model is the execution of the Keras Inception V3 model, with a few layers added and all of the Inception V3 weights frozen for 10 epochs. The intent of this was to quicly train the added layers. This resulted in a test set accuracy of 28% over just the 10 epochs, with a very quick learning rate.

The fifth model is the fine tune of the fourth model, using stocastic gradient descent with a very small learning rate (0.0001) and a momentum value of 0.8, trained across 200 epochs. The validation loss plot indicates that the loss was slightly decreasing after an initial rise, and the training accuracy plot is evidence of overfitting, evidenced by the extremely high training accuracy (over 99% for many epochs) and the occasional dramatic increase in loss/drop in accuracy for some epochs, especially considering the low learning rate (and thus very small changes in the weights). Fortunately, the accuracy of this model on the unseen test set exceeded 36%, compared to the random baseline of 14.3%.

But, is 36% accuracy "good"?

Given the datasets in previous assignments have been carefully prepared for pedagogical value, it is tempting to be accustomed to accuracies above 0.90. As such, 36% accuracy seems low at first impression. To convince myself in regard to the performance, a few reference points are:

- **Random Class Prediction** - Random class prediction with 7 classes would result in an accuracy of 1/7=14.3%
- **Predominant Class Prediction** - The predominant class prediction (maximum class weight / sum of the weights) is 8.32292 / 35.54167 = 23.42%.
- **"Expert Prediction** - I randomly examined 20 movie posters and set of genres used, using all of my *a priori* knowledge and ability to read the text, and attempted to "expertly" classify the posters. I was able to correctly classify 8 of the 20 posters, thus I had an accuracy of 40% (albeit with a small sample).
- **Milestone 3 Investigation** - For milestone three, we incorporated average color (R, G, and B) into our traditional models and PAL intensities, in addition to other metadata about the movies. These models in the multiclass setting achieved 32.3% for Random Forest and 26.4% for a Support Vector Machine with an RBF kernel.

As such, the accuracy achieved by the CNN is rather impressive, as it is similar to mine and exceeds the random sampling, however is only looking at the color and structural elements (such as edges and shapes). It is obvious that this is a real-world problem, and a quite challenging one at that.

Tuning Observations

This notebook has been executed hundreds of times with varying parameters using approximately 50 hours of Tesla K80 GPU time, attempting to develop an intuition about the parameters. The following is the result of this developed intuition:

- **Adding Layers** - The addition of layers increases the flexibility of the model, allowing it to learn quicker on the training set at the expense of additional computation time. Unfortunatley, without regularization it dramatically increases the chances of overfitting.
- **Epochs** - The models have been executed with between 10 and 500 epochs, however most of the models explored seem to plateau within 100 epochs. As such, most of the epochs have been set to 200.
- **Regularization** - Regularization was employed both with l1 and l2 regularization to the activation function and with dropout, which effectively performs implicit regularization by dropping data. Regularization proved to be very tricky to tune, as insufficient (or absent) regularization led to rapid overfitting on flexible models, whereas "over-regularization" resulted in models plateauing at very low training accuracies.
- **Learning Rate** - Learning rate (and method, SGD & rmsprop) impacted the slope of the learning rate, although too high of a learning rate resulted in instability, too low resulted in very fine tuning and very low learning rates. It became quite obvious that a relatively high learning rate, such as 0.1 with SGD, followed by iterative reduction in the rate performed well, such as the Inception V3 training.
- **Synthetic Data Generation** - Synethetic data generation was explored using the Keras ImageDataGenerator, creating images via cropping, roations, flips and various transformations to the original dataset. This significantly increased run time, and given the relatively large amount of

data available, did not prove to be worthwhile for this dataset given the constraints.

- **Class Weights** - Strangely, the use of class weights, representing the class proportions, did not *appear* to improve the accuracy of the model, possibly due to the use of stratified (class-proportionate) sampling of the training data.
- **Validation Splits** - Adjusting the validation splits from 15-30% had the anticipated effect on the training, with reductions in training data (due to a higher validation percentage) reducing the learning rate.
- **Drop Out Percentage** - The percentage of dropout, varying between 25% and 50% on layers, impacted both the learning rate and seemed to slightly reduce overfitting concerns.

Next Steps

As a follow-on from Milestone 3, we discovered that a multi-label approach generated more intuitive results than our multiclass with the classes being reduced using heuristics. We intend on creating a compositite stacking model for our final predictive model, with multi-label class probabilities being derived from the posters, but also intermediate data via other means, such as textaul analysis of the movie overviews and other metadata. As such, the next steps will be to train a multi-label model based upon the posters and generate and save the label probabilities as input to the final model, as follow.

Create the Convolutional Neural Net architecture, from scratch (Multi-label)

In [9]:

```
def load_poster_data_multilabel(image_size, source_size = 'w92', verbose = False
):
    # Loads the poster image data at the requested size, the assigned genre, and
the movie id.
    #
    y_labels = pd.read_csv('y_labels_multilabel.csv')

    image_path = './posters/' + source_size + '/'
    posters = pd.DataFrame()
    for movie in y_labels.iterrows():
        row = movie[0]
        movie_id = movie[1]['movie_id']
        genre_ids = movie[1]['genre_id'].replace('[', '').replace(']','').split(
',')
        try:
            image = misc.imread(image_path + str(movie_id) + '.jpg')
            image_resize = img_to_array(misc.imresize(image, image_size))
            if (image_resize.shape[2]==3):
                posters = posters.append({'movie_id' : movie_id,
                                          'genre_id' : genre_ids,
                                          'poster' : image_resize}, ignore_index
= True)
        except IOError:
            if (verbose == True):
                print('Unable to load poster for movie #', movie_id)
    print('Loaded ', posters.shape[0], ' posters.')
    return posters
```

In [10]:

```
temp = load_poster_data_multilabel((138,92), 'w92', False)
```

```
Loaded  6824  posters.
```

In [12]:

```
train_sample = temp.sample(5000, replace="False")
test_sample = temp.sample(1000, replace="False")

train_posters = train_sample["poster"]
test_posters = test_sample["poster"]

x_train = reshape_and_normalize(train_posters.as_matrix())
x_test = reshape_and_normalize(test_posters.as_matrix())


img_rows = x_train[0].shape[0]
img_cols = x_train[0].shape[1]

y_train = train_sample["genre_id"]
y_test = test_sample["genre_id"]

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (5000, 138, 92, 3)
5000 train samples
1000 test samples
```

In [13]:

```
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(y_train)
y_test = mlb.transform(y_test)
classes = max(y_train.shape[1], y_test.shape[1])
```

In [14]:

```
loss_method='binary_crossentropy'
final_activation_function = 'sigmoid'
```

In [17]:

```
# create an empty network model
model_ml = Sequential()

# Input Layer
model_ml.add(Conv2D(16, kernel_size=input_kernel_size, activation=input_activati
on_function, input_shape=input_shape))
model_ml.add(MaxPooling2D(pool_size=pool_size))

# Hidden Layer(s)
model_ml.add(Conv2D(32, kernel_size=hidden_kernel_size, activation=hidden_activa
tion_function))
model_ml.add(Dropout(0.25))
model_ml.add(MaxPooling2D(pool_size=pool_size))

# Classification layer
model_ml.add(Flatten())
model_ml.add(Dense(64, activation=hidden_activation_function))
model_ml.add(Dropout(0.5))
model_ml.add(Dense(classes, activation=final_activation_function))

# Compile the model.
model_ml.compile(loss=loss_method, optimizer=optimizer, metrics=[eval_metric])
```

In [19]:

```
test_genre_probabilities = np.array(model_ml.predict_proba(x_test))
```

In [ ]:

```
# We may wish to consider tuning an InceptionV3 or other pre-trained model for b
etter results.
```

References Used

https://keras.io/callbacks/ (https://keras.io/callbacks/)

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html)

https://keras.io/getting-started/faq/#how-can-i-use-keras-with-datasets-that-dont-fit-in-memory (https://keras.io/getting-started/faq/#how-can-i-use-keras-with-datasets-that-dont-fit-in-memory)

https://elitedatascience.com/keras-tutorial-deep-learning-in-python#step-4 (https://elitedatascience.com/keras-tutorial-deep-learning-in-python#step-4)

http://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/ (http://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/)

In [ ]:

```

```