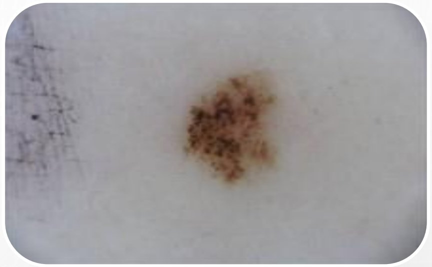
SKIN CANCER DETECTION MODELS

CLASSIFY CANCER IMAGES AS MALIGNANT OR BENIGN

DATA

- TRAINING DATAT SET: 2000 DERMOSCOPIC IMAGES
  - CLASS LABELS – MALIGNANT/BENIGN (1/0) FOR MELANOMA
  - 374 MALIGNANT
  - 1626 BENIGN

- VALIDATION DATA SET: 150 DERMOSCOPIC IMAGES
  - CLASS LABELS – MALIGNANT/BENIGN (1/0) FOR MELANOMA
  - 30 MALIGNANT
  - 120 BENIGN




DATA PREPARATION

- RESIZING OF IMAGES - CONVERTED TO 224 X 224 X 3
- ROTATION OF IMAGES – 180 AND 90 DEGREES
- UPSCALING OF MALIGNANT IMAGES
- CONVERTED IMAGE PIXELS TO A VECTOR
- DEVELOPED TRAIN DATA (.H5 FILES) – PREDICTOR MATRIX AND LABEL VECTOR
- STANDARDIZED PREDICTORS

MODELS DEVELOPED

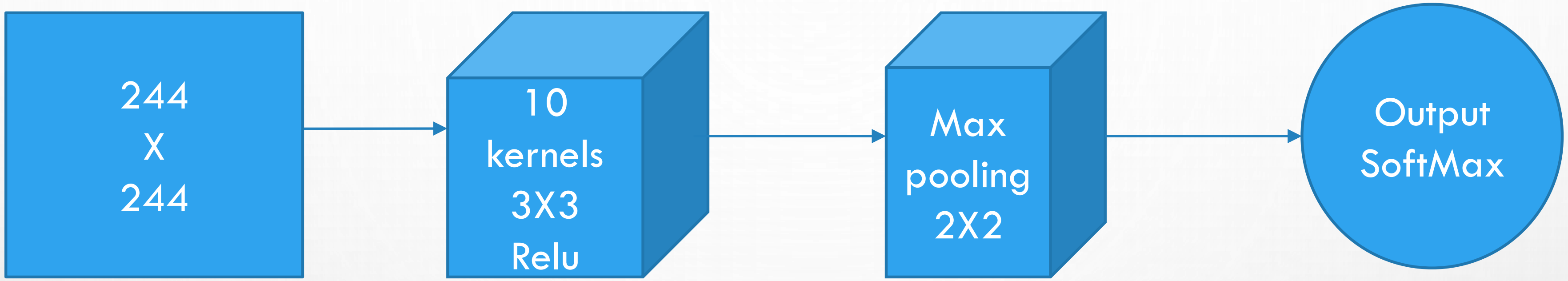
I. DEEP MODELS USING KERAS LIBRARIES

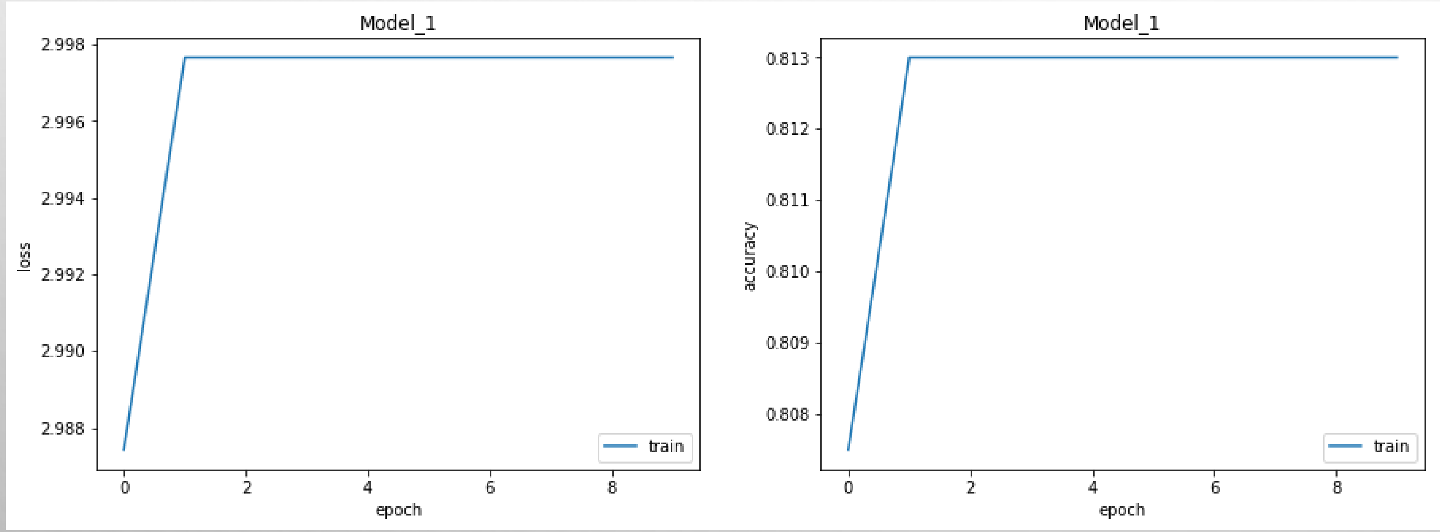
II. SHALLOW NEURAL NETWORK – 2 LAYERS

III. DEEP NEURAL NETWORK – 5 LAYERS

OPTIMIZER = Adam()
METRICS =['accuracy']
LOSS = 'binary_crossentropy'
KERNEL_INITIAL ='glorot_uniform'

CCN MODEL 1



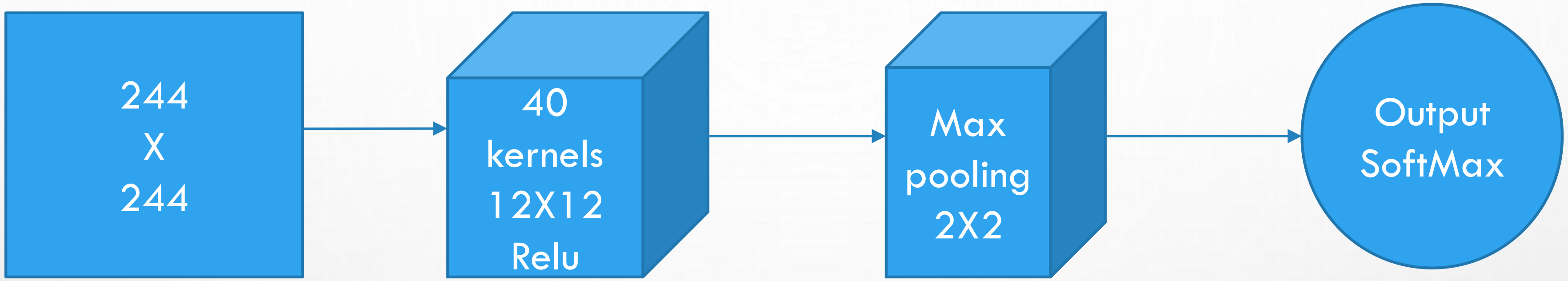


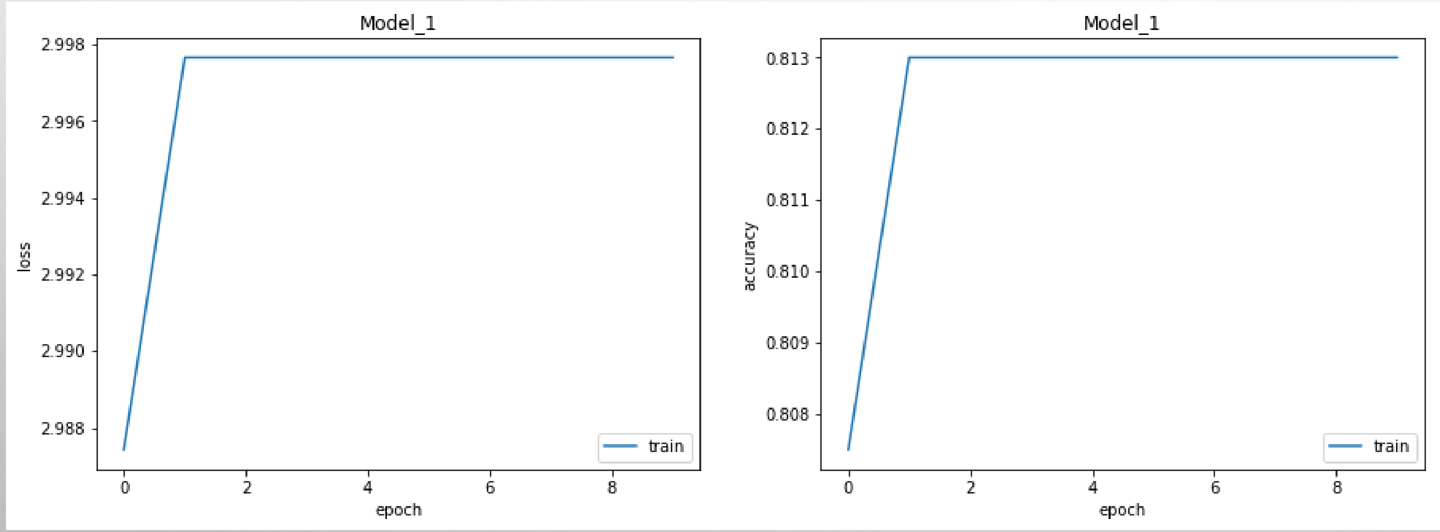
Epoch 2/10
loss: 2.9977
acc: 0.8130

deepPredict(model1,x_test,y_test)
Test score: 3.20604784648
Test accuracy: 0.800000000795

OPTIMIZER = Adam()
METRICS =['accuracy']
LOSS = 'binary_crossentropy'
KERNEL_INITIAL ='glorot_uniform'

CCN MODEL 2



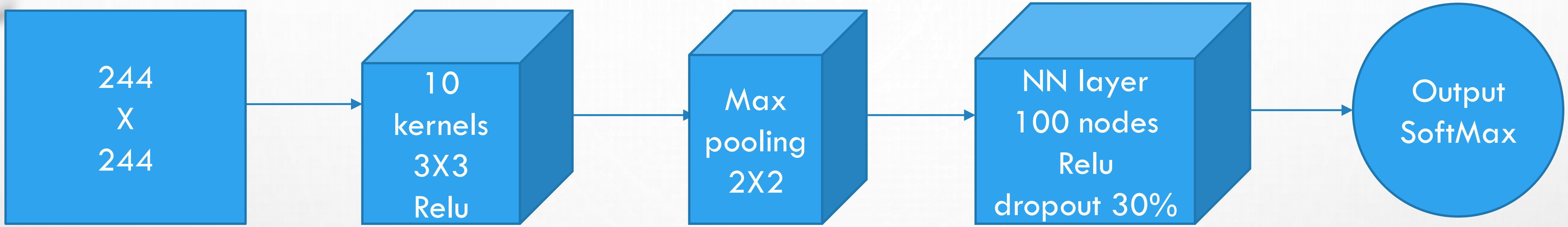


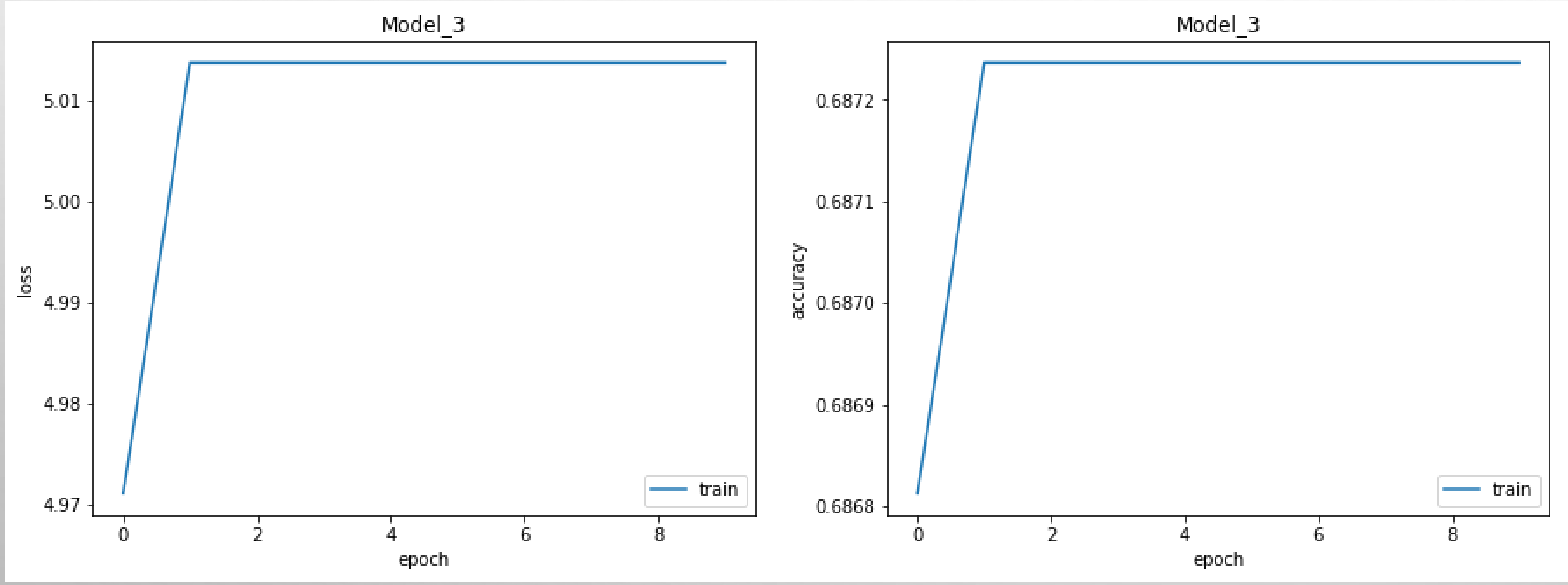
Epoch 2/10
loss: 2.9977
acc: 0.8130

deepPredict(model1,x_test,y_test)
Test score: 3.20604784648
Test accuracy: 0.800000000795

OPTIMIZER = Adam()
METRICS =['accuracy']
LOSS = 'binary_crossentropy'
KERNEL_INITIAL ='glorot_uniform'

CCN MODEL 3



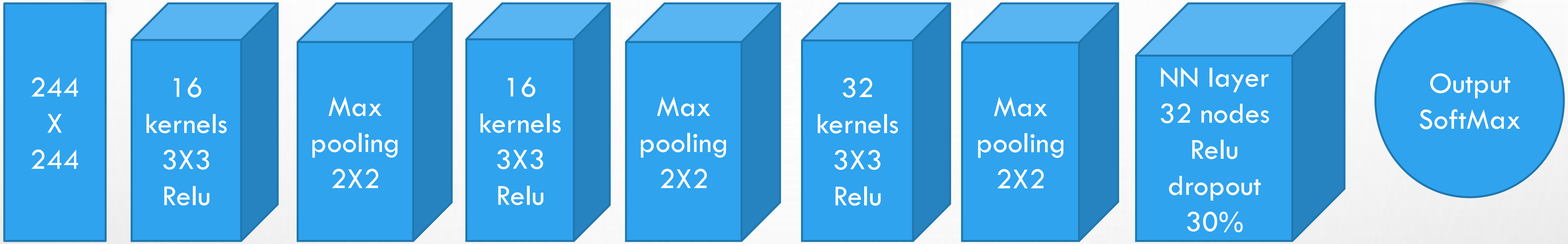


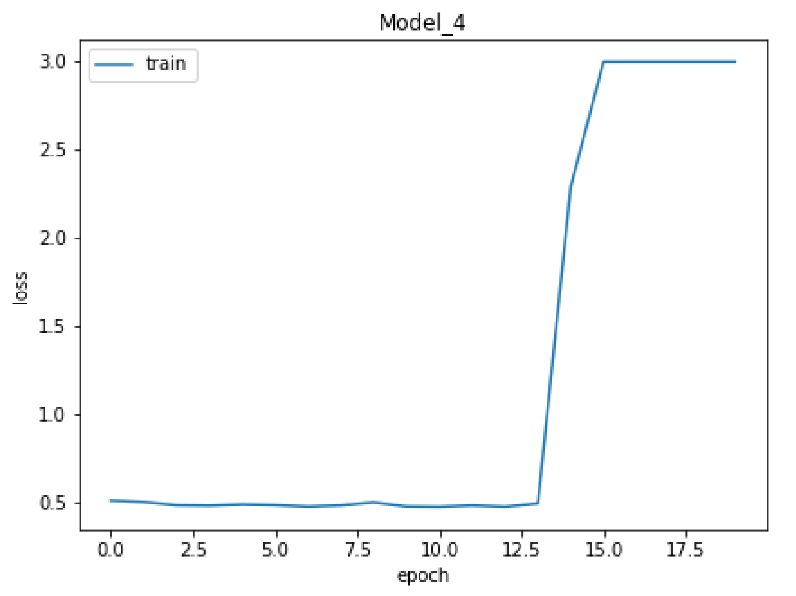
Epoch 10/10
loss: 5.0137
acc: 0.6872

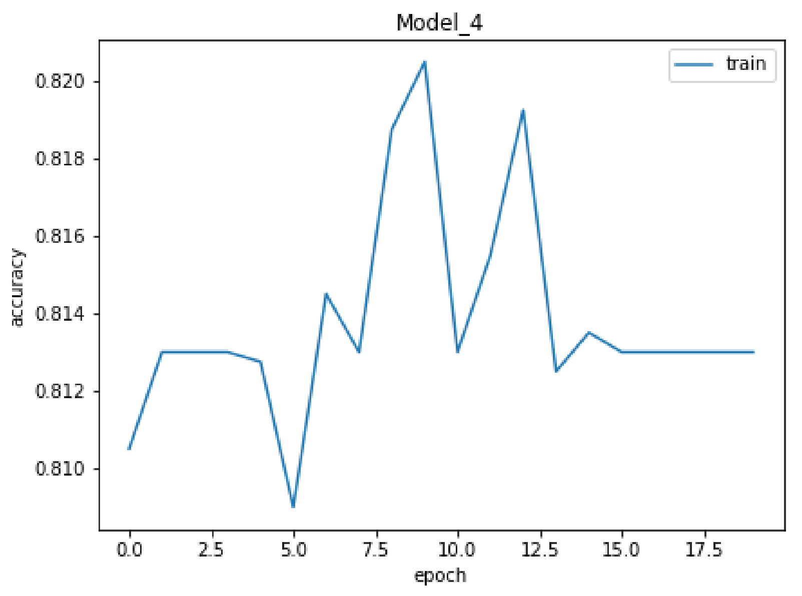
deepPredict(model1,x_test,y_test)
Test score: 3.20604784648
Test accuracy: 0.800000000795

OPTIMIZER = rmsprop
METRICS =['accuracy']
LOSS = 'binary_crossentropy'

CCN MODEL 4





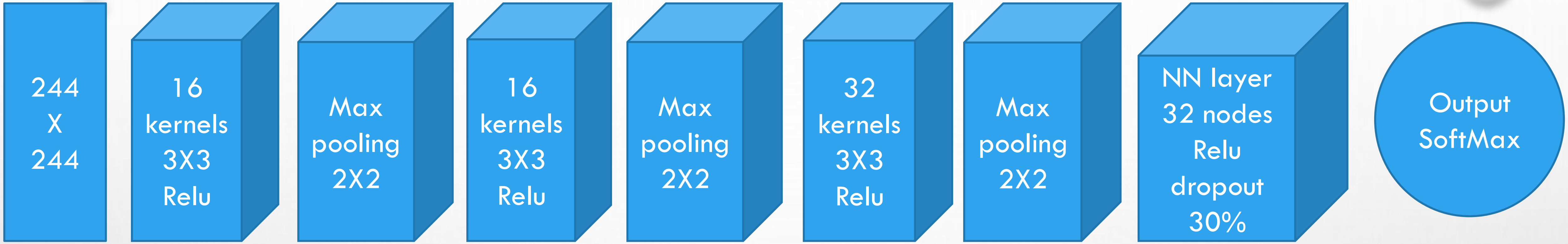


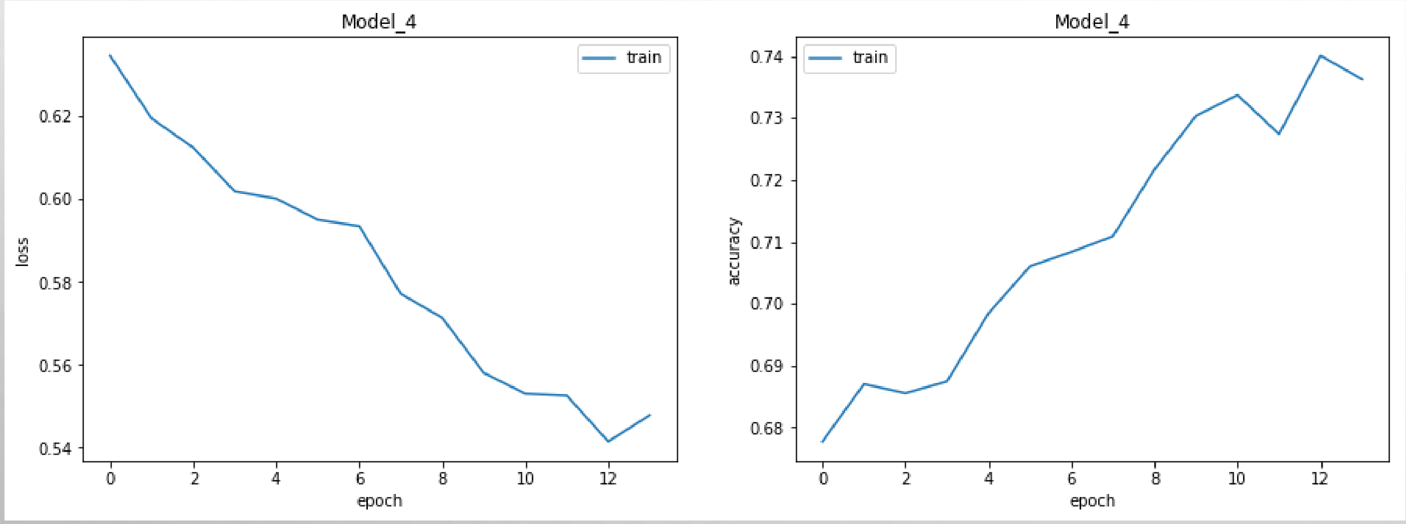
Epoch 10/10
loss: 0.4868
acc: 0.8183

deepPredict(model1,x_test,y_test)
Test score: 0.488429104487
Test accuracy: 0.800000000795

OPTIMIZER = rmsprop
METRICS =['accuracy']
LOSS = 'binary_crossentropy'

CCN MODEL 4 IMAGES ROTATED (180)



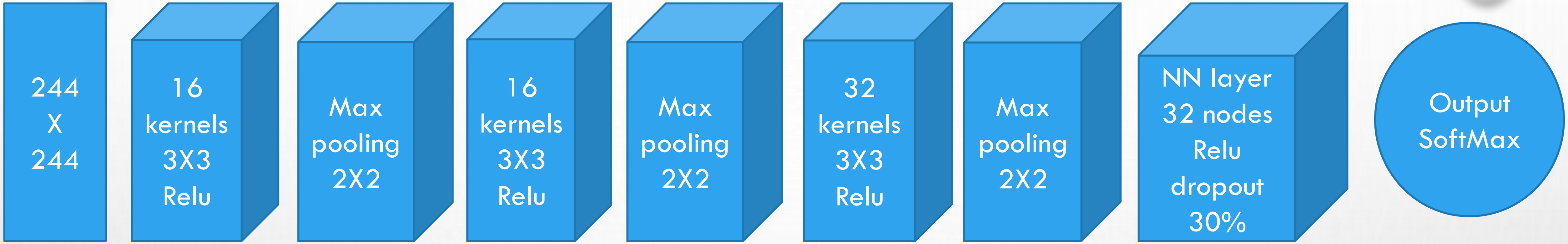


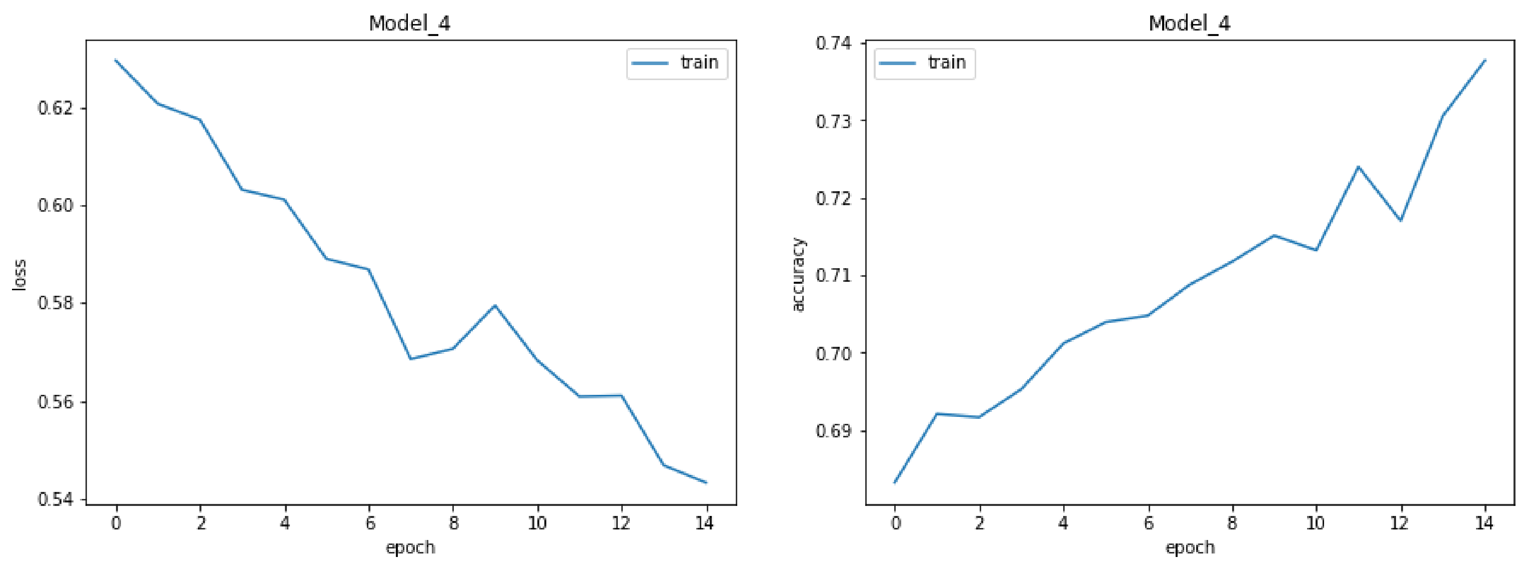
Epoch 13/14
loss: 0.5415
acc: 0.7401

deepPredict(model1,x_test,y_test)
Test score: 0.5447554938
Test accuracy: 0.769999996026

OPTIMIZER = rmsprop
METRICS =['accuracy']
LOSS = 'binary_crossentropy'

CCN MODEL 4 IMAGES ROTATED (90)





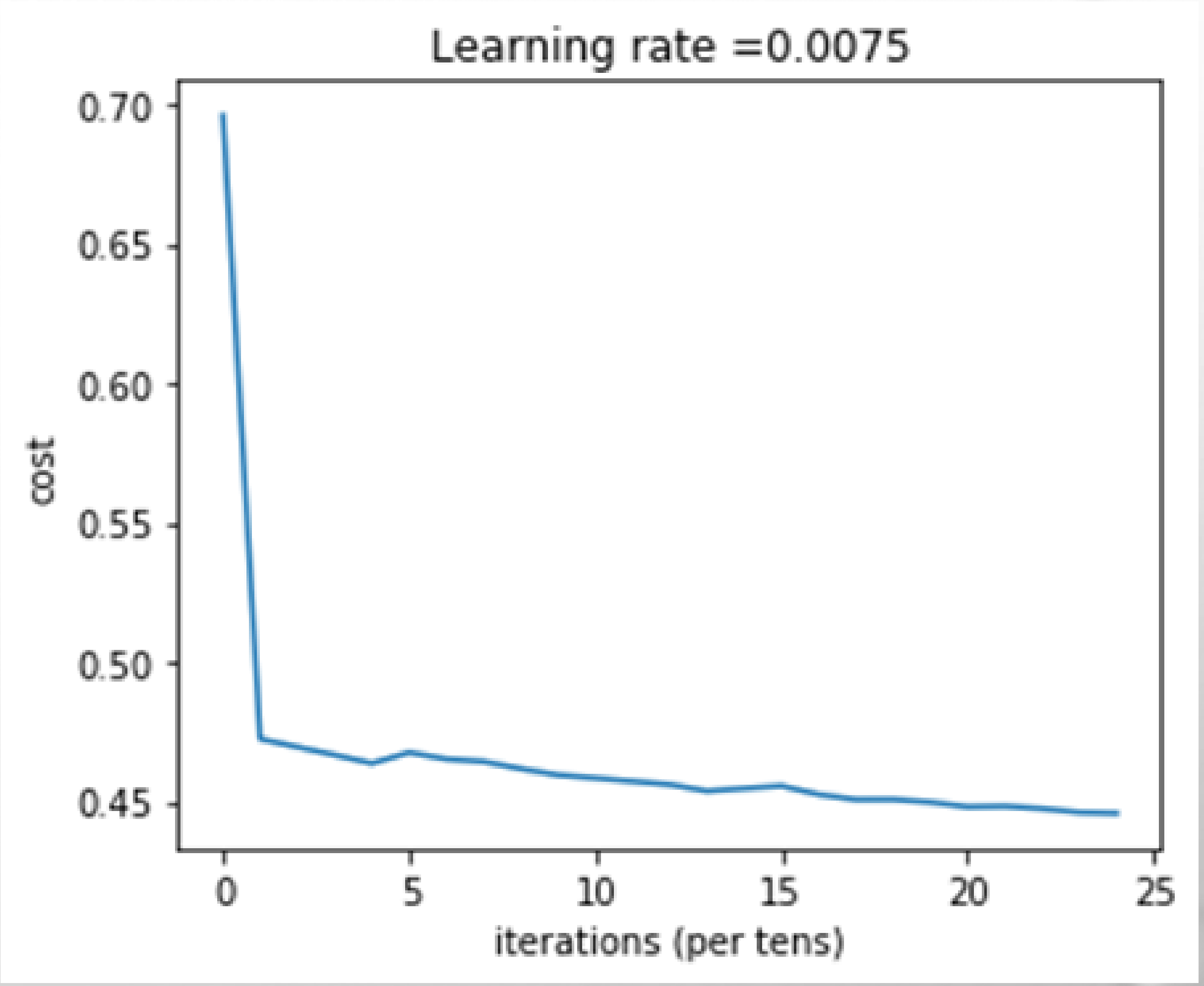
Epoch 13/14
loss: 0.5433
acc: 0.7377

deepPredict(model1,x_test,y_test)
Test score: 0.527270306746
Test accuracy: 0.77666667064

SHALLOW NEURAL NETWORK

Cost after iteration 0: 0.6960335407116954
Cost after iteration 100: 0.4724549661840295
Cost after iteration 200: 0.4695814213451869

Cost after iteration 2200: 0.4474359727229443
Cost after iteration 2300: 0.4461306727321353
Cost after iteration 2400: 0.4458236539266459



predictions_train = predict(X, Y, parameters)
Accuracy: 0.82

for shallow 2 layer model [150528, 7, 1]

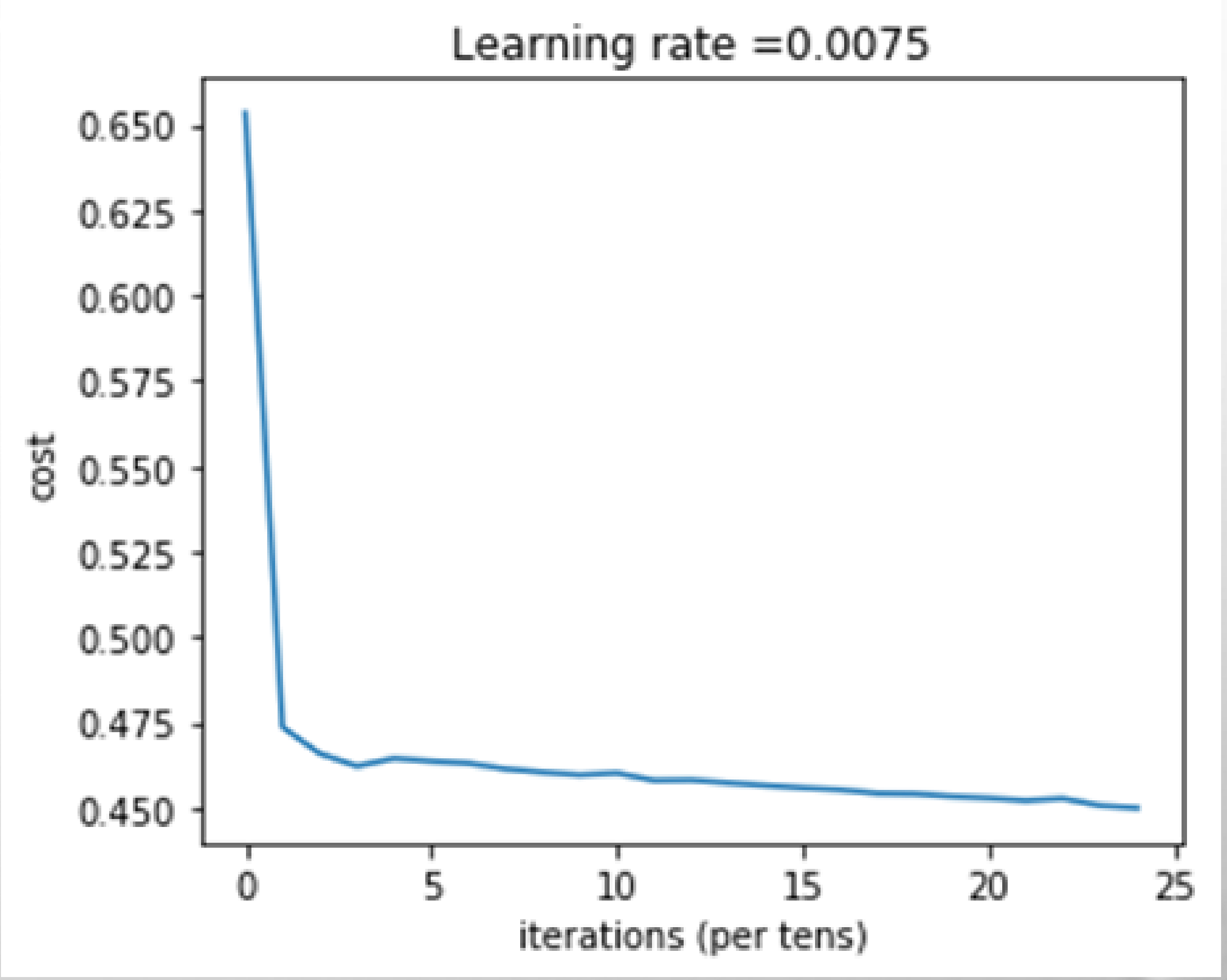
DEEP NEURAL NETWORK

Cost after iteration 0: 0.653496
Cost after iteration 100: 0.473703
Cost after iteration 200: 0.465915

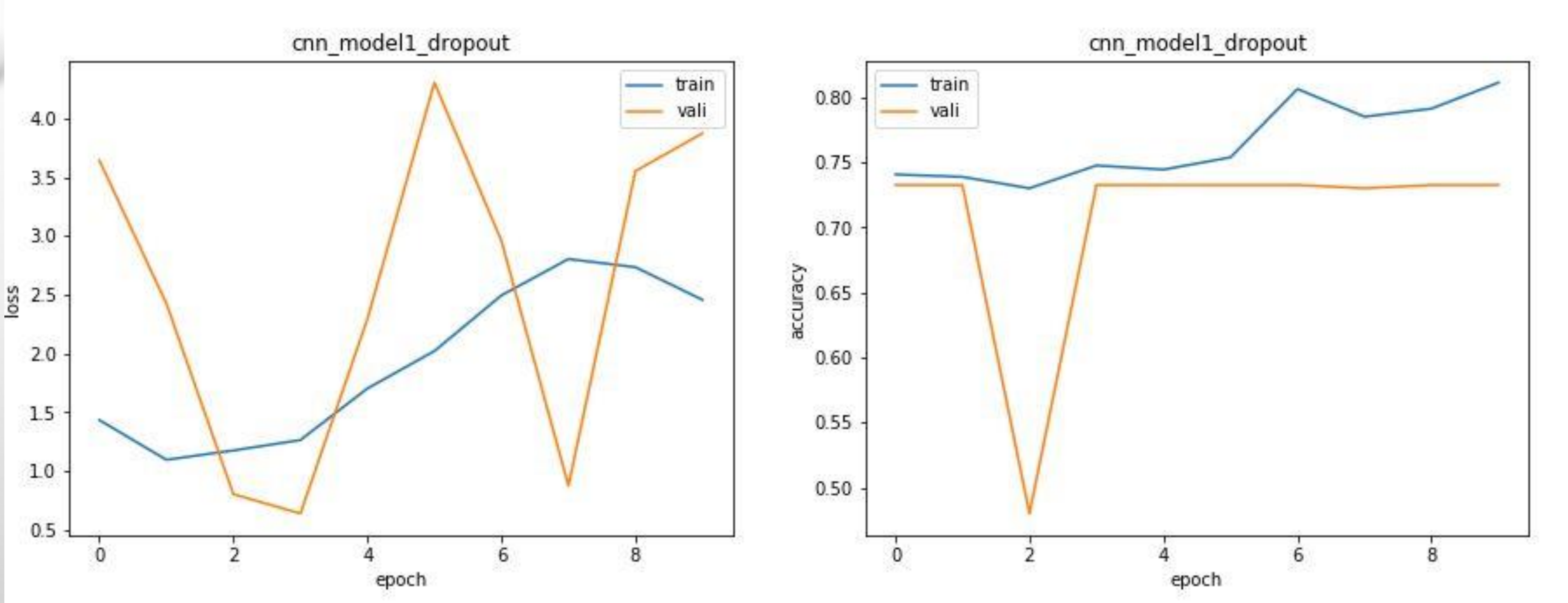
Cost after iteration 2200: 0.452765
Cost after iteration 2300: 0.450670
Cost after iteration 2400: 0.450000

pred_train = predict(X, Y, parameters)
Accuracy: 0.8145

for [150528, 20, 7, 5, 1] 5-layer model



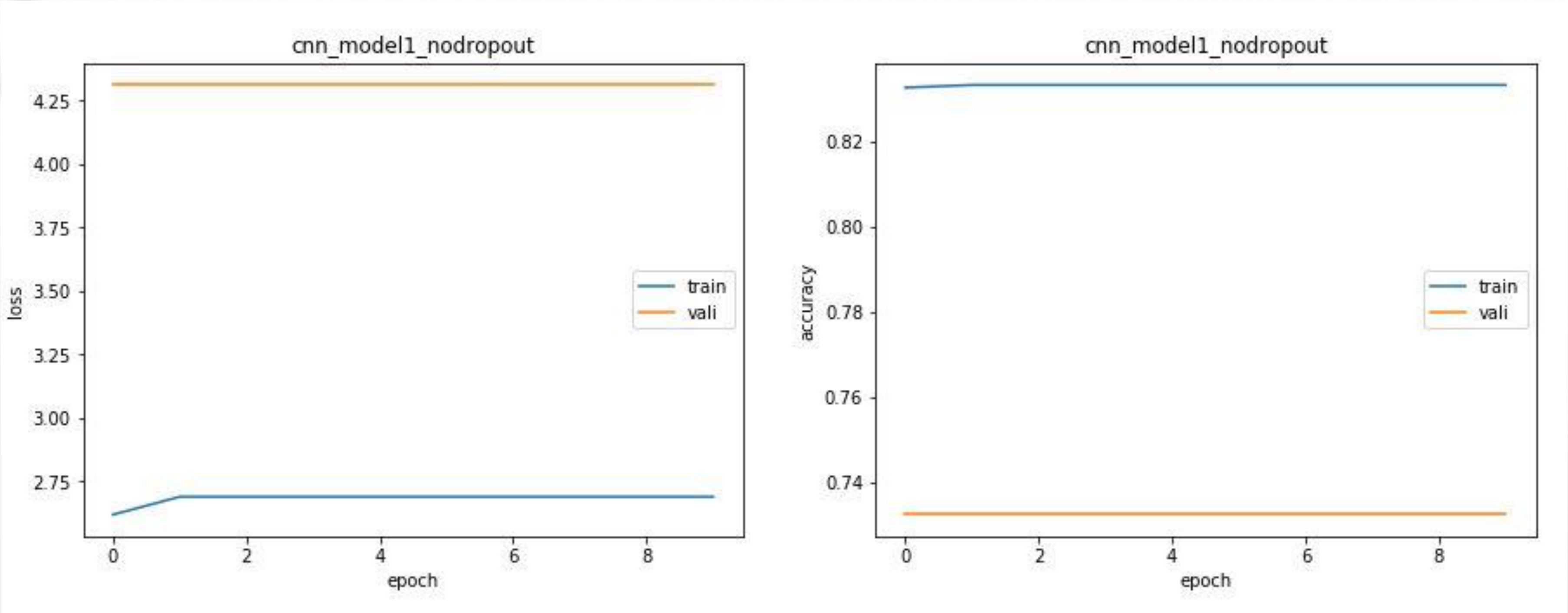
KERAS NEURAL NETWORK_1 RELU, 1 SOFTMAX



With dropout

```
Epoch 4/10
1550/1600 [============================>.] - ETA: 3s - loss: 1.2906 - acc: 0.7439 Epoch 00003: val_acc improved from 0.73250 to
0.73250, saving model to model1_dropout_1_03_0.73.hdf5
```

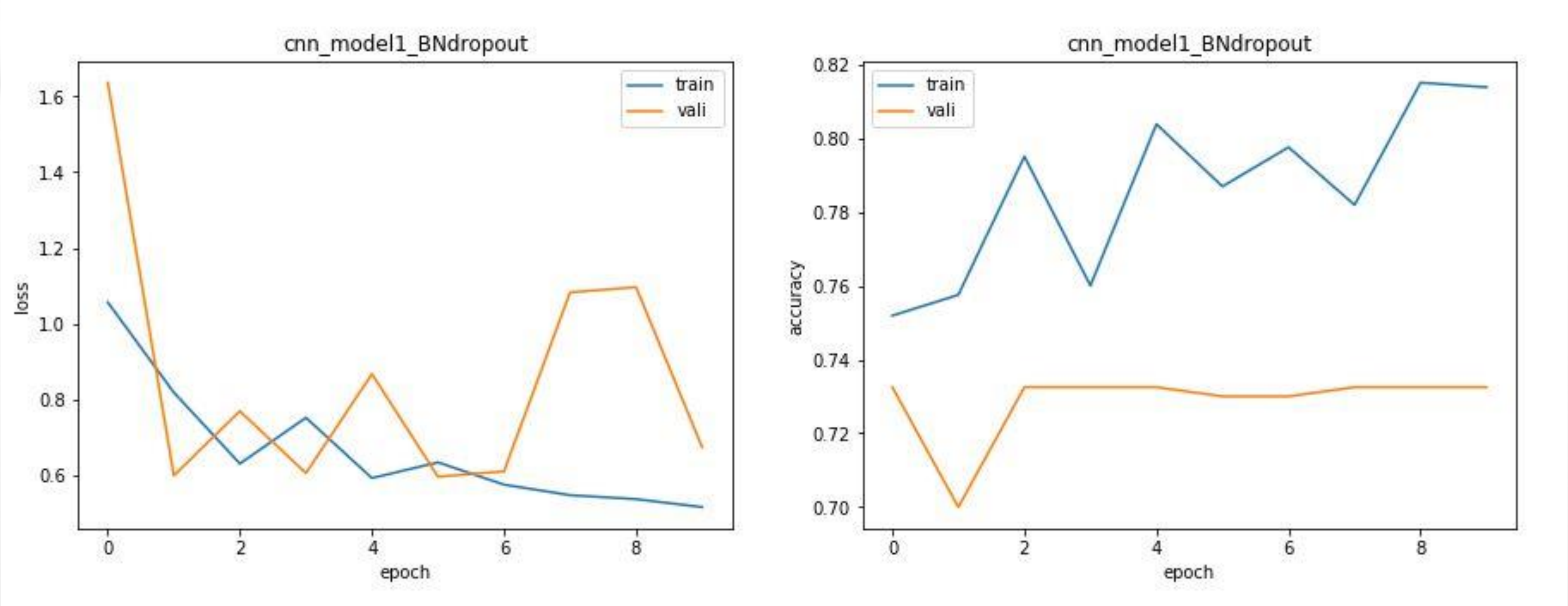
KERAS NEURAL NETWORK_1 RELU, 1 SOFTMAX



Without dropout

```
Epoch 1/10
1550/1600 [============================>.] - ETA: 3s - loss: 2.6622 - acc: 0.8297  Epoch 00000: val_acc improved from -inf to 0.73250, saving model to model1_dropout_0_00_0.73.hdf5
1600/1600 [==============================] - 126s - loss: 2.6193 - acc: 0.8325 - val_loss: 4.3116 - val_acc: 0.7325
```

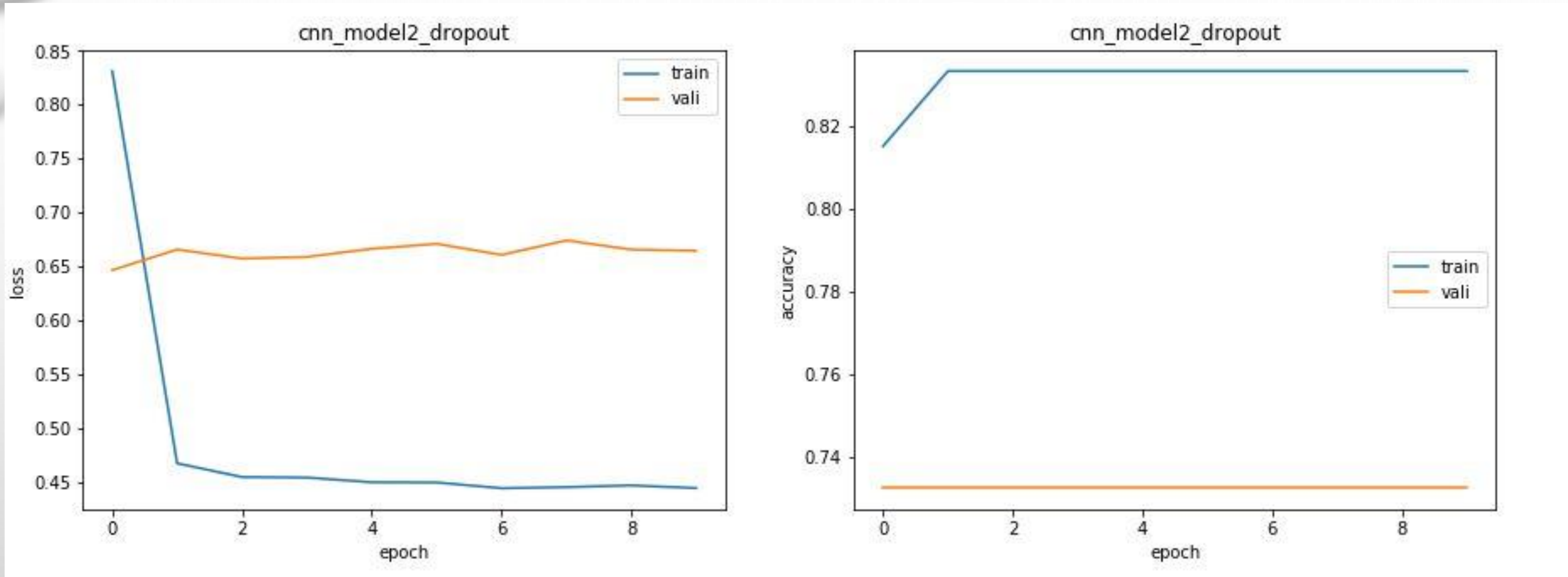
BATCH NORMALIZATION



Dropout & batch normalization -

```
Epoch 9/10
1550/1600 [============================>.] - ETA: 4s - loss: 0.5412 - acc: 0.8123  Epoch 00008: val_acc did not improve
1600/1600 [==============================] - 143s - loss: 0.5356 - acc: 0.8150 - val_loss: 1.0959 - val_acc: 0.7325
```

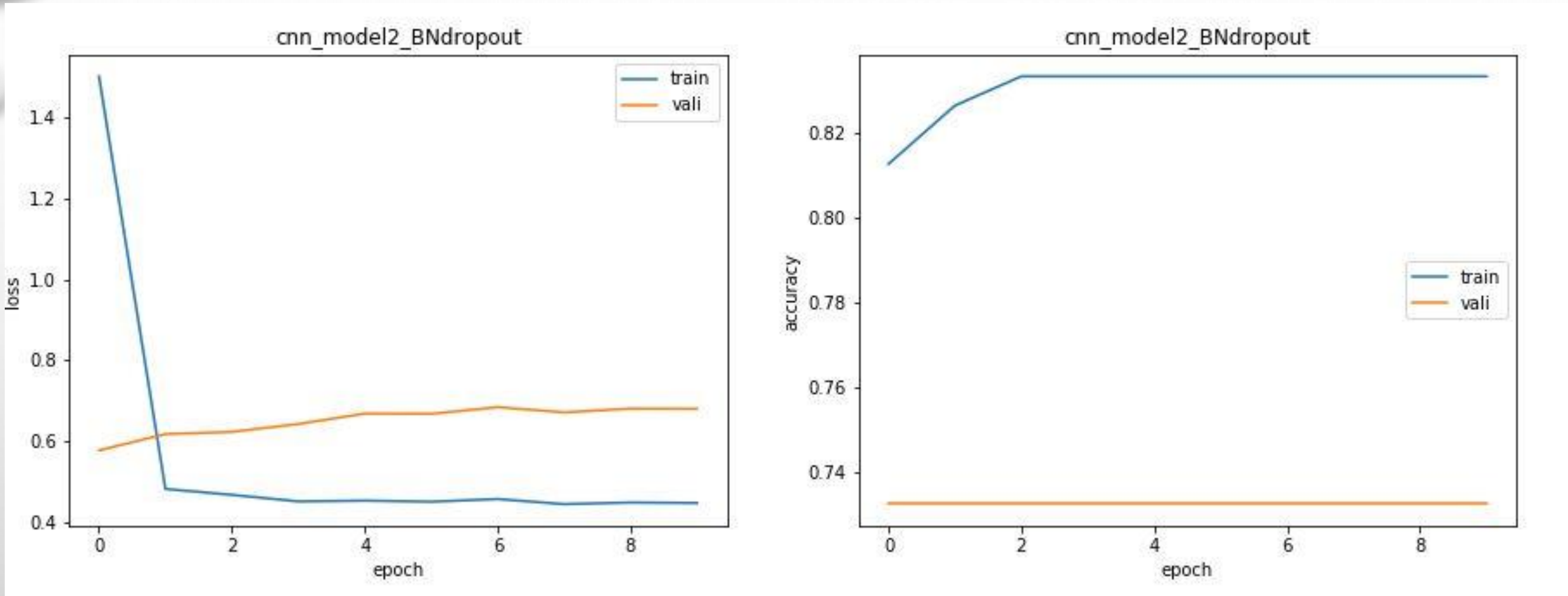
KERAS NEURAL NETWORK_3 CONVO, 1 OUTPUT



Without dropout

```
Epoch 2/10
1550/1600 [============================>.] - ETA: 4s - loss: 0.4626 - acc: 0.8342  Epoch 00001: val_acc did not improve
1600/1600 [==============================] - 175s - loss: 0.4675 - acc: 0.8331 - val_loss: 0.6657 - val_acc: 0.7325
```

BATCH NORMALIZATION



Dropout and batch normalization

```
Epoch 10/10
1550/1600 [============================>.] - ETA: 5s - loss: 0.4480 - acc: 0.8323  Epoch 00009: val_acc did not improve
1600/1600 [==============================] - 197s - loss: 0.4467 - acc: 0.8331 - val_loss: 0.6797 - val_acc: 0.7325
```

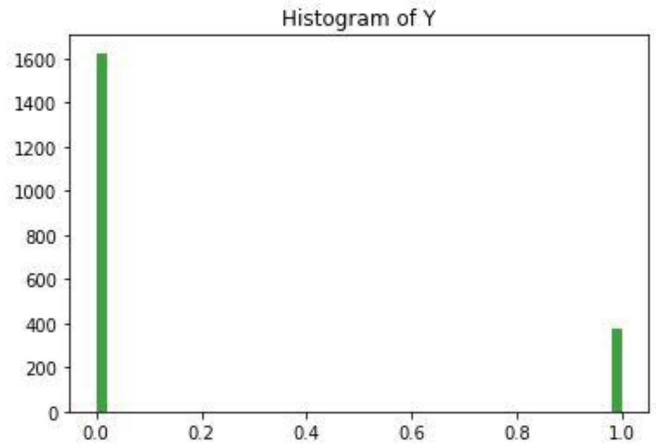
STEPS FORWARD

1) ADDRESS CLASS IMBALANCE – ACCURACY IS IMPROVED JUST A LITTLE OVER BASELINE
2) DEEPER NETWORKS USUALLY LEARN BETTER – NOT ABLE TO RUN DEEPER NETWORKS WITH CURRENT HARDWARE

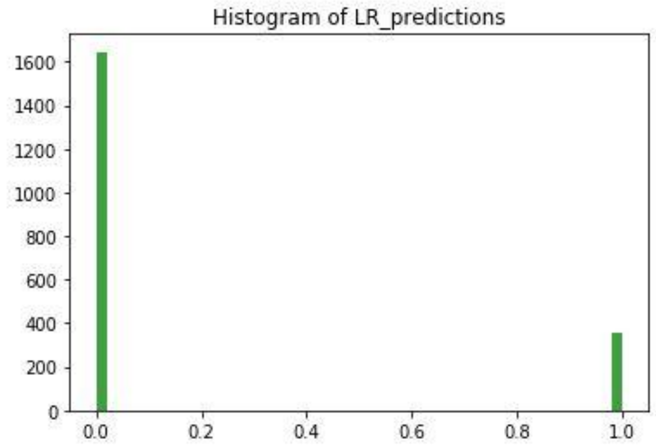
NEED BIG DATA TOO……2000 IMAGES ISN'T GOOD ENOUGH DATA FOR NEURAL

LOGISTIC

```
plt.title('Histogram of Y')
    ...: n, bins, patches = plt.hist(Y[:], 50, facecolor='green', alpha=0.75)
```




```
In [99]: plt.title('Histogram of LR_predictions')
    ...: n, bins, patches = plt.hist(LR_predictions[:], 50, facecolor='green', alpha=0.75)
```



Confusion Matrix: confusion_matrix(y_true , y_pred)

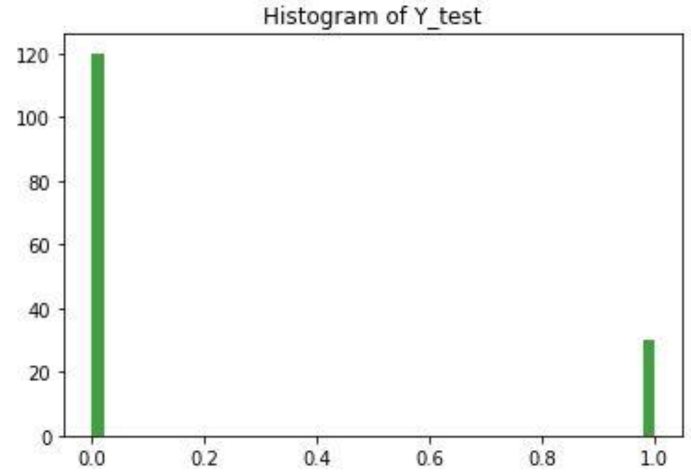
```
array([[1624, 2],
       [ 21, 353]], dtype=int64)
```

```
Accuracy = ((1624+353)/2000)*100
    ...: print('Accuracy =' + str(Accuracy) + "%")
```

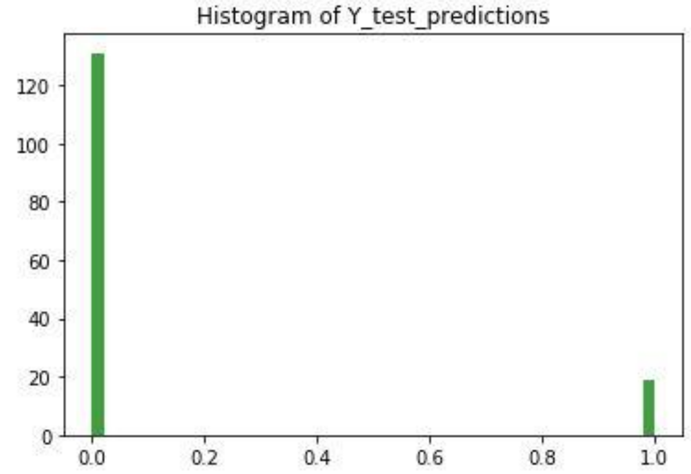
Accuracy =98.85 % on Training Data

LOGISTIC

```
In [34]: plt.title('Histogram of Y_test')
    ...: n, bins, patches = plt.hist(Y_test[:], 50, facecolor='green', alpha=0.75)
```




```
In [35]: plt.title('Histogram of Y_test_predictions')
    ...: n, bins, patches = plt.hist(Y_test_predictions[:], 50, facecolor='green', alpha=0.75)
```



Confusion Matrix: confusion_matrix(y_true , y_pred)

array([[109, 11],
 [22, 8]], dtype=int64)

Accuracy = ((109+8)/150)*100
...: print('Accuracy = ' + str(Accuracy) + "%")

Accuracy = 78.0 % on Test data

Upsampled the data

Original dataset shape Counter({0: 1626, 1: 374})

Resampled dataset shape Counter({0: 1626, 1: 1626})

array([[1626, 0],
 [0, 1626]], dtype=int64)

Accuracy on upsampled data = 100 % on Test data

array([[108, 12],
 [23, 7]], dtype=int64)

Accuracy of upsampled model on Test data = 76.66 %

QUESTIONS ?