Run the first 4 cells to import data and create training and testing set

In [1]:
```
import numpy as np
import h5py
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import SimpleRNN,LSTM, Dense, Activation
from keras.utils import to_categorical
#from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from UtilNNDL import *
```

```
Using TensorFlow backend.
```

In [2]:
```
"""
#file_path = '/home/carla/Downloads/project_datasets/project_datasets/'
file_path = '/home/kunal/Desktop/FinalProject/datasets/A01T_slice.mat'

A01T = h5py.File(file_path,'r')
data = np.copy(A01T['image'])
data = np.transpose(data,(0,2,1))
data = data[:,:,:22]
labels = np.copy(A01T['type'])
labels = labels[0,0:data.shape[0]:1]
labels = np.asarray(labels, dtype=np.int32)

a = data[:56]
b = data[57:]
data = np.vstack((a,b))
a = labels[:56]
b = labels[57:]
labels = np.hstack((a,b))
#enc = OneHotEncoder()
#enc_labels = enc.fit_transform(labels.reshape(-1,1)).toarray()
enc_labels = to_categorical(labels-769, num_classes=4)
print(enc_labels)

#scaler = StandardScaler()
#data = scaler.fit_transform(data,enc_labels)
"""
```

Out[2]:
```
"\n#file_path = '/home/carla/Downloads/project_datasets/project_datasets/'\
nfile_path = '/home/kunal/Desktop/FinalProject/datasets/A01T_slice.mat'\n\n
A01T = h5py.File(file_path,'r')\ndata = np.copy(A01T['image'])\ndata = np.t
ranspose(data,(0,2,1))\ndata = data[:,:,:22]\nlabels = np.copy(A01T['type']
)\nlabels = labels[0,0:data.shape[0]:1]\nlabels = np.asarray(labels, dtype=
np.int32)\n\na = data[:56]\nb = data[57:]\ndata = np.vstack((a,b))\na = lab
els[:56]\nb = labels[57:]\nlabels = np.hstack((a,b))\n#enc = OneHotEncoder(
)\n#enc_labels = enc.fit_transform(labels.reshape(-1,1)).toarray()\nenc_lab
els = to_categorical(labels-769, num_classes=4)\nprint(enc_labels)\n\n#scal
er = StandardScaler()\n#data = scaler.fit_transform(data,enc_labels)\n"
```

In [3]:
```
"""
bs, t, f = data.shape
np.random.seed(42)
shuffle = np.random.choice(bs,bs,replace=False)

train_samples = 237
train_data = data[shuffle[:train_samples],:,:]
train_labels = enc_labels[shuffle[:train_samples]]
test_data = data[shuffle[train_samples:],:,:]
test_labels =enc_labels[shuffle[train_samples:]]

train_data = np.transpose(train_data,(0,2,1))
test_data = np.transpose(test_data,(0,2,1))

train_data,train_labels = create_window_data(train_data,train_labels)
test_data,test_labels = create_window_data(test_data,test_labels)

train_data = np.transpose(train_data,(0,2,1))
test_data = np.transpose(test_data,(0,2,1))

bs, t, f = train_data.shape
"""
```

Out[3]:
```
'\nbs, t, f = data.shape\nnp.random.seed(42)\nshuffle = np.random.choice(bs
,bs,replace=False)\n\ntrain_samples = 237\ntrain_data = data[shuffle[:train
_samples],:,:]\ntrain_labels = enc_labels[shuffle[:train_samples]]\ntest_da
ta = data[shuffle[train_samples:],:,:]\ntest_labels =enc_labels[shuffle[tra
in_samples:]]\n\ntrain_data = np.transpose(train_data,(0,2,1))\ntest_data =
np.transpose(test_data,(0,2,1))\n\ntrain_data,train_labels = create_window_
data(train_data,train_labels)\ntest_data,test_labels = create_window_data(t
est_data,test_labels)\n\ntrain_data = np.transpose(train_data,(0,2,1))\ntes
t_data = np.transpose(test_data,(0,2,1))\n\nbs, t, f = train_data.shape\n'
```

In [4]:
```
#Prepare the data by taking out nans and dividing into test and train
#file_path = '/home/carla/Downloads/project_datasets/project_datasets/'
file_path = '/home/kunal/Desktop/FinalProject/datasets/'
train_data, test_data, train_labels, test_labels = prepare_data(file_path,
                                                                num_test_sa
mples = 50,
                                                                verbose= Fa
lse,
                                                                return_all=
True,
                                                                num_files =
1)
print train_data.shape
print train_labels.shape
print test_data.shape
print test_labels.shape
```

```
(237, 22, 1000)
(237, 4)
(50, 22, 1000)
(50, 4)
```

In [5]:
```
#assist numerical stability
train_data = train_data*(1e6)
test_data = test_data*(1e6)
```

In [6]:
```
#Bandpass filter the data
train_data = train_data.swapaxes(1,2)
test_data = test_data.swapaxes(1,2)
print train_data.shape
print test_data.shape
for i,a in enumerate(train_data):
    train_data[i] = bandpass_cnt(a, 4, 38, 250, filt_order=3)
for i,a in enumerate(test_data):
    test_data[i] = bandpass_cnt(a, 4, 38, 250, filt_order=3)
print train_data.shape
print test_data.shape
```

```
(237, 1000, 22)
(50, 1000, 22)
(237, 1000, 22)
(50, 1000, 22)
```

In [7]:
```
#Standardize the data
for i,a in enumerate(train_data):
    train_data[i] = exponential_running_standardize(a, factor_new=0.001, in
it_block_size=1000, eps=1e-4)
for i,a in enumerate(test_data):
    test_data[i] = exponential_running_standardize(a, factor_new=0.001, ini
t_block_size=1000, eps=1e-4)
train_data = train_data.swapaxes(1,2)
test_data = test_data.swapaxes(1,2)
print train_data.shape
print test_data.shape
```

```
(237, 22, 1000)
(50, 22, 1000)
```

In [8]:
```
#Augment the data into a bigger set by windowing
train_data_sliced, train_labels_sliced = create_window_data(train_data, tra
in_labels, windows=10,window_size=512)
test_data_sliced, test_labels_sliced = create_window_data(test_data, test_l
abels, windows=10,window_size=512)

train_data_sliced = train_data_sliced.swapaxes(1,2)
test_data_sliced = test_data_sliced.swapaxes(1,2)

bs,t,f = train_data_sliced.shape

print train_data_sliced.shape
print train_labels_sliced.shape
print test_data_sliced.shape
print test_labels_sliced.shape
```

```
(2370, 512, 22)
(2370, 4)
(500, 512, 22)
(500, 4)
```

Everything from this point down is Testing

In [18]:
```
model = Sequential([
    LSTM(100, input_shape=(t,f)),
    Dense(32),
    Activation('relu'),
    #Dense(64),
    #Activation('relu'),
    Dense(32),
    Activation('relu'),
    Dense(4),
    Activation('softmax'),
])

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

hist = model.fit(train_data,train_labels,epochs=15,validation_split=0.25,ba
tch_size=32,verbose=0)
test_score = model.evaluate(test_data, test_labels, batch_size=32)

print(test_score)

plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-18-4798cfa1d3bb> in <module>()
     15              metrics=['accuracy'])
     16
---> 17 hist = model.fit(train_data,train_labels,epochs=15,validation_split
=0.25,batch_size=32,verbose=0)
     18 test_score = model.evaluate(test_data, test_labels, batch_size=32)
     19

/home/carla/Documents/tensorflow/local/lib/python2.7/site-packages/keras/mo
dels.pyc in fit(self, x, y, batch_size, epochs, verbose, callbacks, validat
ion_split, validation_data, shuffle, class_weight, sample_weight, initial_e
poch, steps_per_epoch, validation_steps, **kwargs)
    961                                 initial_epoch=initial_epoch,
    962                                 steps_per_epoch=steps_per_epoch,
--> 963                                 validation_steps=validation_steps)
    964
    965     def evaluate(self, x=None, y=None,

/home/carla/Documents/tensorflow/local/lib/python2.7/site-packages/keras/en
gine/training.pyc in fit(self, x, y, batch_size, epochs, verbose, callbacks
, validation_split, validation_data, shuffle, class_weight, sample_weight,
initial_epoch, steps_per_epoch, validation_steps, **kwargs)
   1710                                 initial_epoch=initial_epoch,
   1711                                 steps_per_epoch=steps_per_epoch,
-> 1712                                 validation_steps=validation_steps)
   1713
   1714     def evaluate(self, x=None, y=None,

/home/carla/Documents/tensorflow/local/lib/python2.7/site-packages/keras/en
gine/training.pyc in _fit_loop(self, f, ins, out_labels, batch_size, epochs
, verbose, callbacks, val_f, val_ins, shuffle, callback_metrics, initial_ep
och, steps_per_epoch, validation_steps)
   1233                         ins_batch[i] = ins_batch[i].toarray()
   1234
-> 1235                     outs = f(ins_batch)
   1236                     if not isinstance(outs, list):
   1237                         outs = [outs]

/home/carla/Documents/tensorflow/local/lib/python2.7/site-packages/keras/ba
ckend/tensorflow_backend.pyc in __call__(self, inputs)
   2473         session = get_session()
   2474         updated = session.run(fetches=fetches, feed_dict=feed_dict,
-> 2475                               **self.session_kwargs)
   2476         return updated[:len(self.outputs)]
   2477

/home/carla/Documents/tensorflow/local/lib/python2.7/site-packages/tensorfl
ow/python/client/session.pyc in run(self, fetches, feed_dict, options, run_
metadata)
    893     try:
    894       result = self._run(None, fetches, feed_dict, options_ptr,
--> 895                          run_metadata_ptr)
    896       if run_metadata:
    897         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

/home/carla/Documents/tensorflow/local/lib/python2.7/site-packages/tensorfl
ow/python/client/session.pyc in _run(self, handle, fetches, feed_dict, opti
ons, run_metadata)
   1126     if final_fetches or final_targets or (handle and feed_dict_tens
or):
   1127       results = self._do_run(handle, final_targets, final_fetches,
```

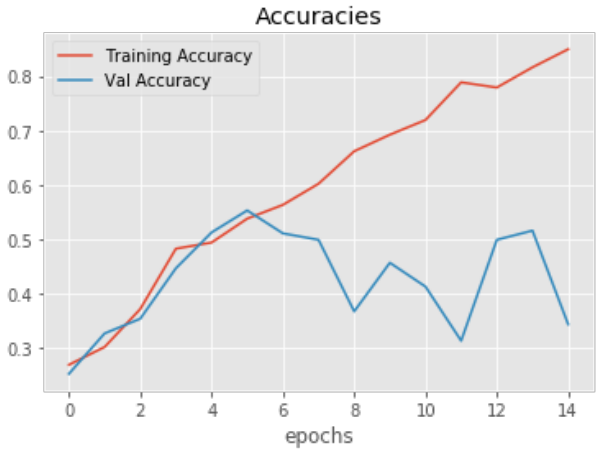
Modified VGGnet for this type of data

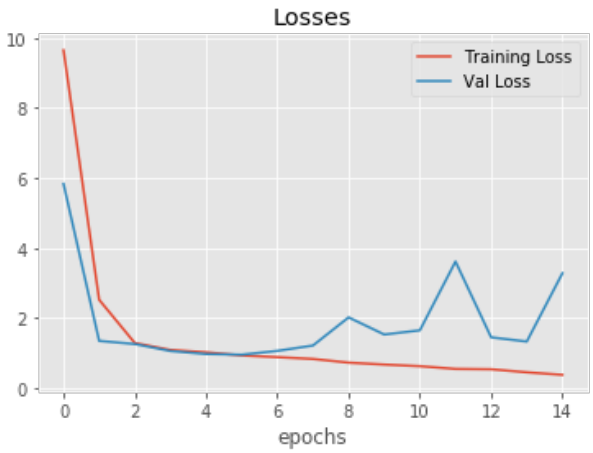
Modified VGG net to handle our input i.e. replace 2D with 1D, etc.(need to check dimensions and might need to transpose input to original shape)

Original VGGnet implementation can be found at hte address below

In [66]:
```
### VGGnet
# https://keras.io/getting-started/sequential-model-guide/#examples

import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv1D, MaxPooling1D, BatchNormalization
from keras.optimizers import SGD

#norm_train = np.transpose((-np.mean(train_data,axis=2)+np.transpose(train_
data,(2,0,1)))/np.std(train_data,axis=2),(1,2,0))
#norm_test = np.transpose((-np.mean(test_data,axis=2)+np.transpose(test_dat
a,(2,0,1)))/np.std(test_data,axis=2),(1,2,0))

model = Sequential()
#model.add(LSTM(100, input_shape=(t,f)))
model.add(Conv1D(32, 4, activation='relu',input_shape=(t,f)))             #
Originally 32 each
model.add(BatchNormalization())
model.add(Conv1D(32, 4, activation='relu'))
model.add(MaxPooling1D())
model.add(Dropout(0.25))

model.add(Conv1D(64, 4, activation='relu'))
#Originally 64 each
model.add(BatchNormalization())
model.add(Conv1D(64, 4, activation='relu'))
model.add(MaxPooling1D())
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['ac
curacy'])

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

hist = model.fit(train_data_sliced,train_labels_sliced,epochs=15,validation
_split=0.25,batch_size=64,verbose=0)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=64)

print(test_score)
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
500/500 [==============================] - 0s 139us/step
[3.3455803813934328, 0.36599999904632569]
```

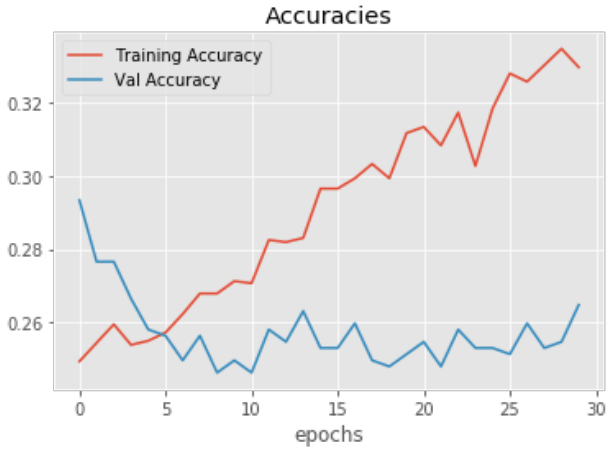


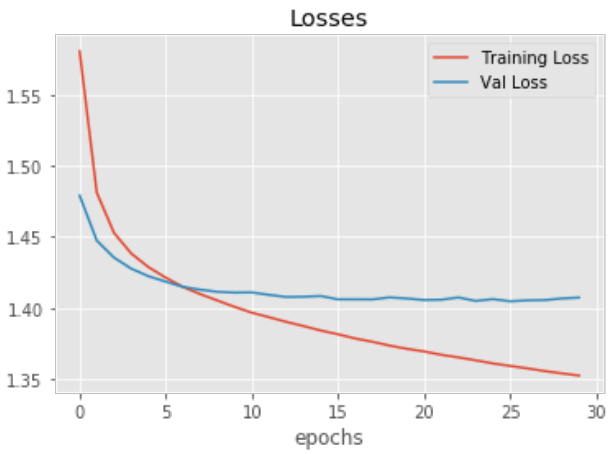


Simple RNN model

In [12]:
```
model = Sequential([
    SimpleRNN(64, input_shape=(t,f)),
    #Dense(32),
    #BatchNormalization(),
    #Activation('relu'),
    Dense(4),
    Activation('softmax'),
])

model.compile(optimizer = 'sgd',
              loss = 'categorical_crossentropy',
              metrics=['accuracy'])

hist = model.fit(train_data_sliced,train_labels_sliced,epochs=30,validation
_split=0.25,batch_size=64,verbose=0)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=32)

print(test_score)
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
500/500 [==============================] - 1s 1ms/step
[1.4065807809829711, 0.25600000023841857]
```



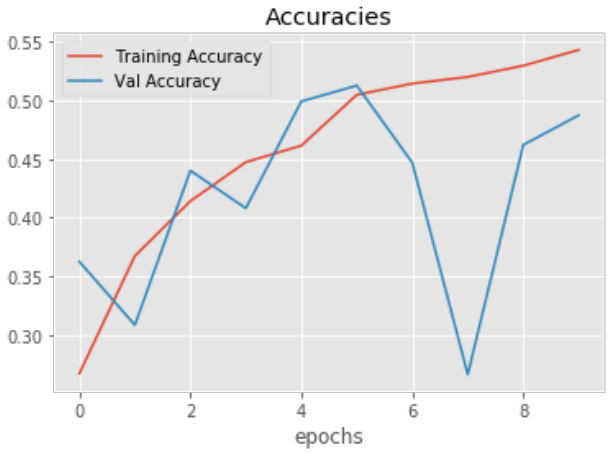


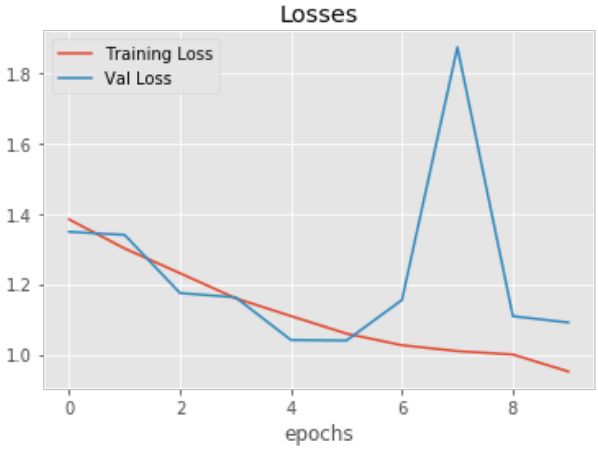
CHRONONET PAPER

C-RNN implementation (Figure 1b)

In [67]:
```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv1D, MaxPooling1D, BatchNormalization,GRU
from keras.optimizers import SGD
from keras.initializers import glorot_normal


model = Sequential()

model.add(Conv1D(32, 4, strides=2,activation='relu',input_shape=(t,f)))
model.add(Conv1D(32, 4, strides=2,activation='relu'))
model.add(Conv1D(32, 4, strides=2,activation='relu'))

#model.add(Flatten())

model.add(GRU(32,activation='tanh',return_sequences=True,kernel_initializer
=glorot_normal()))
model.add(GRU(32,activation='tanh',return_sequences=True,kernel_initializer
=glorot_normal()))
model.add(GRU(32,activation='tanh',return_sequences=True,kernel_initializer
=glorot_normal()))
model.add(GRU(32,activation='tanh',kernel_initializer=glorot_normal()))

#model.add(Dense(64,activation = 'relu'))
#model.add(Dense(32,activation = 'relu'))
model.add(Dense(4, activation='softmax'))

#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd)

#model.add()

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

hist = model.fit(train_data_sliced,train_labels_sliced,epochs=10,validation
_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced,test_labels_sliced, batch_size
=64)
print(test_score)
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
Train on 1777 samples, validate on 593 samples
Epoch 1/10
1777/1777 [==============================] - 17s 10ms/step - loss: 1.3835 -
acc: 0.2673 - val_loss: 1.3483 - val_acc: 0.3626
Epoch 2/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.3011 - a
cc: 0.3675 - val_loss: 1.3396 - val_acc: 0.3086
Epoch 3/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.2308 - a
cc: 0.4142 - val_loss: 1.1744 - val_acc: 0.4401
Epoch 4/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.1597 - a
cc: 0.4474 - val_loss: 1.1623 - val_acc: 0.4081
Epoch 5/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.1090 - a
cc: 0.4615 - val_loss: 1.0412 - val_acc: 0.4992
Epoch 6/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.0594 - a
cc: 0.5048 - val_loss: 1.0397 - val_acc: 0.5126
Epoch 7/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.0264 - a
cc: 0.5144 - val_loss: 1.1548 - val_acc: 0.4469
Epoch 8/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.0096 - a
cc: 0.5200 - val_loss: 1.8726 - val_acc: 0.2664
Epoch 9/10
1777/1777 [==============================] - 7s 4ms/step - loss: 1.0000 - a
cc: 0.5295 - val_loss: 1.1088 - val_acc: 0.4621
Epoch 10/10
1777/1777 [==============================] - 7s 4ms/step - loss: 0.9516 - a
cc: 0.5431 - val_loss: 1.0908 - val_acc: 0.4874
500/500 [==============================] - 0s 880us/step
[0.95397258472442625, 0.53600000000000003]
```

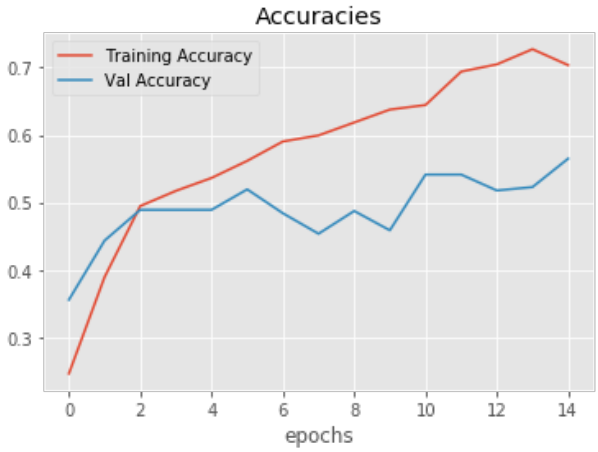




Implementation of Figure 1b but adding regularization structures like that found in VGGnet

In [68]:
```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv1D, MaxPooling1D, BatchNormalization,GRU
from keras.optimizers import SGD

#norm_train = np.transpose((-np.mean(train_data,axis=2)+np.transpose(train_
data,(2,0,1)))/np.std(train_data,axis=2),(1,2,0))
model = Sequential()

model.add(Conv1D(32, 4, strides=2,activation='relu',input_shape=(t,f)))
model.add(BatchNormalization())                              #From VGGnet
model.add(Conv1D(32, 4, strides=2,activation='relu'))
model.add(BatchNormalization())                              #From VGGnet
model.add(Conv1D(32, 4, strides=2,activation='relu'))
model.add(MaxPooling1D())                                    #From VGGnet
model.add(Dropout(0.25))                                     #From VGGnet
#model.add(Flatten())

model.add(GRU(32,activation='tanh',return_sequences=True))
model.add(GRU(32,activation='tanh',return_sequences=True))
model.add(GRU(32,activation='tanh',return_sequences=True))  #removed becaus
e of overfitting problem to small sample size
model.add(GRU(32,activation='tanh'))

#model.add(Dense(256, activation='relu'))                    #From VGGnet, b
ut makes model suck
#model.add(Dropout(0.5))                                     #From VGGnet, b
ut makes model suck
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(4, activation='softmax'))

# From VGGnet, works well for some reason
#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd,metrics=['acc
uracy'])

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=15,validation
_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=16)
print(test_score)

plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

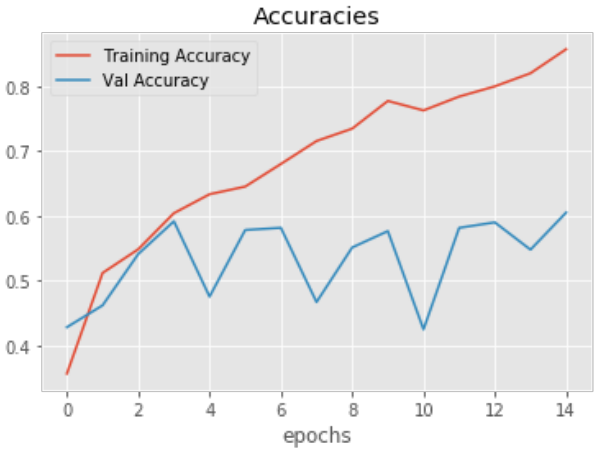
```
Train on 1777 samples, validate on 593 samples
Epoch 1/15
1777/1777 [==============================] - 14s 8ms/step - loss: 1.3872 -
acc: 0.2465 - val_loss: 1.3790 - val_acc: 0.3558
Epoch 2/15
1777/1777 [==============================] - 4s 2ms/step - loss: 1.2981 - a
cc: 0.3894 - val_loss: 1.1695 - val_acc: 0.4435
Epoch 3/15
1777/1777 [==============================] - 4s 2ms/step - loss: 1.0663 - a
cc: 0.4947 - val_loss: 1.0707 - val_acc: 0.4890
Epoch 4/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.9872 - a
cc: 0.5172 - val_loss: 1.0243 - val_acc: 0.4890
Epoch 5/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.9360 - a
cc: 0.5363 - val_loss: 1.0651 - val_acc: 0.4890
Epoch 6/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.8876 - a
cc: 0.5616 - val_loss: 1.0533 - val_acc: 0.5194
Epoch 7/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.8454 - a
cc: 0.5903 - val_loss: 1.1391 - val_acc: 0.4840
Epoch 8/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.8399 - a
cc: 0.5993 - val_loss: 1.5515 - val_acc: 0.4536
Epoch 9/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.8013 - a
cc: 0.6185 - val_loss: 1.4747 - val_acc: 0.4874
Epoch 10/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.7724 - a
cc: 0.6376 - val_loss: 1.5512 - val_acc: 0.4587
Epoch 11/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.7468 - a
cc: 0.6443 - val_loss: 1.1060 - val_acc: 0.5413
Epoch 12/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.6889 - a
cc: 0.6939 - val_loss: 1.2999 - val_acc: 0.5413
Epoch 13/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.6846 - a
cc: 0.7046 - val_loss: 1.3809 - val_acc: 0.5177
Epoch 14/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.6359 - a
cc: 0.7271 - val_loss: 1.3182 - val_acc: 0.5228
Epoch 15/15
1777/1777 [==============================] - 4s 2ms/step - loss: 0.6637 - a
cc: 0.7034 - val_loss: 1.1776 - val_acc: 0.5649
500/500 [==============================] - 1s 2ms/step
[1.0790683217048644, 0.56399999999999995]
```

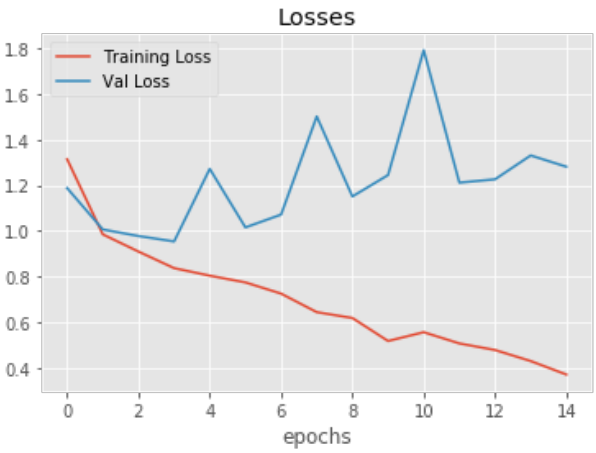


Replaced GRU with LSTM

In [69]:
```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv1D, MaxPooling1D, BatchNormalization,GRU,LSTM
from keras.optimizers import SGD

#norm_train = np.transpose((-np.mean(train_data,axis=2)+np.transpose(train_
data,(2,0,1)))/np.std(train_data,axis=2),(1,2,0))
model = Sequential()

model.add(Conv1D(32, 4, strides=2,activation='relu',input_shape=(t,f)))
model.add(BatchNormalization())                              #From VGGnet
model.add(Conv1D(32, 4, strides=2,activation='relu'))
model.add(BatchNormalization())                              #From VGGnet
model.add(Conv1D(32, 4, strides=2,activation='relu'))
model.add(MaxPooling1D())                                    #From VGGnet
model.add(Dropout(0.25))                                     #From VGGnet
#model.add(Flatten())

model.add(LSTM(32,activation='tanh',return_sequences=True))
model.add(LSTM(32,activation='tanh',return_sequences=True))
model.add(LSTM(32,activation='tanh',return_sequences=True))
model.add(LSTM(32,activation='tanh'))

#model.add(Dense(256, activation='relu'))                    #From VGGnet, b
ut makes model suck
#model.add(Dropout(0.5))                                     #From VGGnet, b
ut makes model suck
model.add(Dense(4, activation='softmax'))

# From VGGnet, works well for some reason
#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(loss='categorical_crossentropy', optimizer=sgd,metrics=['acc
uracy'])

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=15,validation
_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=64)

print "Test Results are ", test_score
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
Train on 1777 samples, validate on 593 samples
Epoch 1/15
1777/1777 [==============================] - 16s 9ms/step - loss: 1.3144 -
acc: 0.3568 - val_loss: 1.1880 - val_acc: 0.4283
Epoch 2/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.9852 - a
cc: 0.5121 - val_loss: 1.0066 - val_acc: 0.4621
Epoch 3/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.9097 - a
cc: 0.5487 - val_loss: 0.9779 - val_acc: 0.5413
Epoch 4/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.8371 - a
cc: 0.6044 - val_loss: 0.9541 - val_acc: 0.5919
Epoch 5/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.8041 - a
cc: 0.6337 - val_loss: 1.2722 - val_acc: 0.4755
Epoch 6/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.7746 - a
cc: 0.6455 - val_loss: 1.0156 - val_acc: 0.5784
Epoch 7/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.7254 - a
cc: 0.6804 - val_loss: 1.0717 - val_acc: 0.5818
Epoch 8/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.6443 - a
cc: 0.7158 - val_loss: 1.5019 - val_acc: 0.4671
Epoch 9/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.6186 - a
cc: 0.7349 - val_loss: 1.1512 - val_acc: 0.5514
Epoch 10/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.5178 - a
cc: 0.7777 - val_loss: 1.2450 - val_acc: 0.5767
Epoch 11/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.5564 - a
cc: 0.7631 - val_loss: 1.7922 - val_acc: 0.4250
Epoch 12/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.5071 - a
cc: 0.7845 - val_loss: 1.2118 - val_acc: 0.5818
Epoch 13/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.4784 - a
cc: 0.8002 - val_loss: 1.2263 - val_acc: 0.5902
Epoch 14/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.4300 - a
cc: 0.8205 - val_loss: 1.3309 - val_acc: 0.5481
Epoch 15/15
1777/1777 [==============================] - 5s 3ms/step - loss: 0.3709 - a
cc: 0.8576 - val_loss: 1.2816 - val_acc: 0.6054
500/500 [==============================] - 0s 655us/step
Test Results are  [1.1562326831817626, 0.59399999952316285]
```

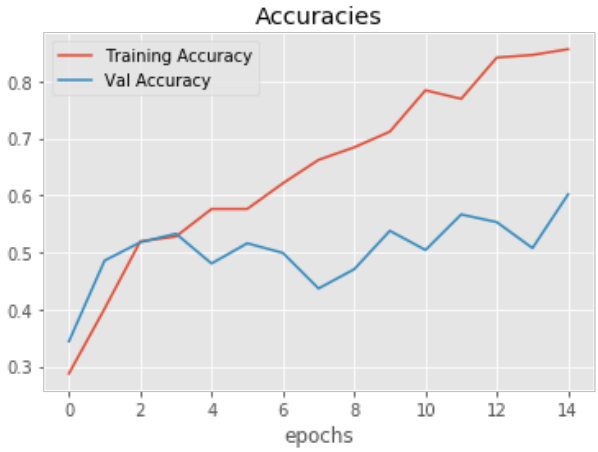

Accuracies
Training Accuracy
Val Accuracy
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
10
12
14
epochs


Losses
Training Loss
Val Loss
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
2
4
6
8
10
12
14
epochs

IC-RNN

In [70]:
```
from keras.layers import Input,Dense,concatenate,Flatten,GRU,Conv1D
from keras.models import Model

inputs= Input(shape=(t,f))

# First Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(inputs
)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(inputs
)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(inputs
)
x = concatenate([tower1,tower2,tower3],axis=2)

# Second Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
x = concatenate([tower1,tower2,tower3],axis=2)

# Third Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
x = concatenate([tower1,tower2,tower3],axis=2)


x = GRU(32,activation='tanh',return_sequences=True)(x)
x = GRU(32,activation='tanh',return_sequences=True)(x)
x = GRU(32,activation='tanh',return_sequences=True)(x)
x = GRU(32,activation='tanh')(x)


predictions = Dense(4,activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=15,validation
_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=64)

print "Test Results are ", test_score
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

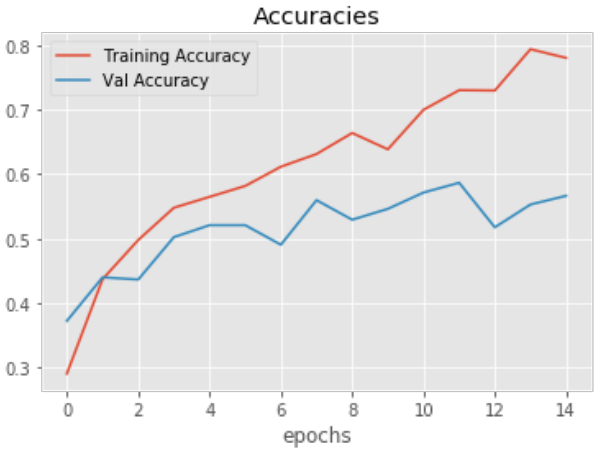
```
Train on 1777 samples, validate on 593 samples
Epoch 1/15
1777/1777 [==============================] - 19s 10ms/step - loss: 1.3920 -
acc: 0.2876 - val_loss: 1.3459 - val_acc: 0.3440
Epoch 2/15
1777/1777 [==============================] - 8s 4ms/step - loss: 1.2776 - a
cc: 0.4012 - val_loss: 1.1440 - val_acc: 0.4857
Epoch 3/15
1777/1777 [==============================] - 8s 4ms/step - loss: 1.0378 - a
cc: 0.5194 - val_loss: 1.0521 - val_acc: 0.5177
Epoch 4/15
1777/1777 [==============================] - 8s 4ms/step - loss: 1.0068 - a
cc: 0.5279 - val_loss: 1.0073 - val_acc: 0.5329
Epoch 5/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.9075 - a
cc: 0.5763 - val_loss: 1.0471 - val_acc: 0.4806
Epoch 6/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.8892 - a
cc: 0.5763 - val_loss: 0.9707 - val_acc: 0.5160
Epoch 7/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.8196 - a
cc: 0.6213 - val_loss: 1.2897 - val_acc: 0.4992
Epoch 8/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.7514 - a
cc: 0.6624 - val_loss: 1.4060 - val_acc: 0.4368
Epoch 9/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.7180 - a
cc: 0.6843 - val_loss: 1.3166 - val_acc: 0.4705
Epoch 10/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.6745 - a
cc: 0.7119 - val_loss: 1.0606 - val_acc: 0.5379
Epoch 11/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.5457 - a
cc: 0.7845 - val_loss: 1.2745 - val_acc: 0.5042
Epoch 12/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.5444 - a
cc: 0.7693 - val_loss: 1.1897 - val_acc: 0.5666
Epoch 13/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.4342 - a
cc: 0.8419 - val_loss: 1.4807 - val_acc: 0.5531
Epoch 14/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.3879 - a
cc: 0.8464 - val_loss: 1.3849 - val_acc: 0.5076
Epoch 15/15
1777/1777 [==============================] - 8s 4ms/step - loss: 0.3739 - a
cc: 0.8565 - val_loss: 1.1536 - val_acc: 0.6020
500/500 [==============================] - 0s 960us/step
Test Results are  [1.1252464733123779, 0.57800000023841858]
```



IC-RNN Testing

In [81]:
```
from keras.layers import Input,Dense,concatenate,Flatten,GRU,Conv1D,Bidirectional
from keras.models import Model

inputs= Input(shape=(t,f))

# First Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(inputs)
tower1 = BatchNormalization()(tower1)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(inputs)
tower2 = BatchNormalization()(tower2)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(inputs)
tower3 = BatchNormalization()(tower3)
#tower4 = MaxPooling1D()(inputs)
x = concatenate([tower1,tower2,tower3],axis=2)
x = Dropout(0.5)(x)

# Second Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower1 = BatchNormalization()(tower1)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower2 = BatchNormalization()(tower2)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
tower3 = BatchNormalization()(tower3)
#tower4 = MaxPooling1D()(x)
x = concatenate([tower1,tower2,tower3],axis=2)
x = Dropout(0.5)(x)


# Third Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower1 = BatchNormalization()(tower1)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower2 = BatchNormalization()(tower2)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
tower3 = BatchNormalization()(tower3)
#tower4 = MaxPooling1D()(x)
x = concatenate([tower1,tower2,tower3],axis=2)
x = Dropout(0.5)(x)


x = (GRU(32,activation='tanh',return_sequences=True))(x)
x = (GRU(32,activation='tanh',return_sequences=True))(x)
x = (GRU(32,activation='tanh',return_sequences=True))(x)
x = (GRU(32,activation='tanh'))(x)


predictions = Dense(4,activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=15,validation_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_size=64)
```

```
Train on 1777 samples, validate on 593 samples
Epoch 1/15
1777/1777 [==============================] - 24s 13ms/step - loss: 1.3777 -
acc: 0.2909 - val_loss: 1.2985 - val_acc: 0.3727
Epoch 2/15
1777/1777 [==============================] - 8s 5ms/step - loss: 1.1911 - a
cc: 0.4373 - val_loss: 1.1600 - val_acc: 0.4401
Epoch 3/15
1777/1777 [==============================] - 8s 5ms/step - loss: 1.0423 - a
cc: 0.4980 - val_loss: 1.1743 - val_acc: 0.4368
Epoch 4/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.9443 - a
cc: 0.5481 - val_loss: 1.0311 - val_acc: 0.5025
Epoch 5/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.9100 - a
cc: 0.5650 - val_loss: 1.0132 - val_acc: 0.5211
Epoch 6/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.8686 - a
cc: 0.5819 - val_loss: 1.0369 - val_acc: 0.5211
Epoch 7/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.8379 - a
cc: 0.6117 - val_loss: 1.1766 - val_acc: 0.4907
Epoch 8/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.7892 - a
cc: 0.6314 - val_loss: 1.0153 - val_acc: 0.5599
Epoch 9/15
1777/1777 [==============================] - 9s 5ms/step - loss: 0.7515 - a
cc: 0.6640 - val_loss: 1.0862 - val_acc: 0.5295
Epoch 10/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.7640 - a
cc: 0.6387 - val_loss: 1.1076 - val_acc: 0.5464
Epoch 11/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.6772 - a
cc: 0.7001 - val_loss: 1.0621 - val_acc: 0.5717
Epoch 12/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.6381 - a
cc: 0.7304 - val_loss: 1.0481 - val_acc: 0.5868
Epoch 13/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.6075 - a
cc: 0.7299 - val_loss: 1.3071 - val_acc: 0.5177
Epoch 14/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.5129 - a
cc: 0.7940 - val_loss: 1.1969 - val_acc: 0.5531
Epoch 15/15
1777/1777 [==============================] - 8s 5ms/step - loss: 0.5112 - a
cc: 0.7805 - val_loss: 1.1992 - val_acc: 0.5666
500/500 [==============================] - 0s 928us/step
Test Results are  [1.0601895842552185, 0.5840000004768372]
```

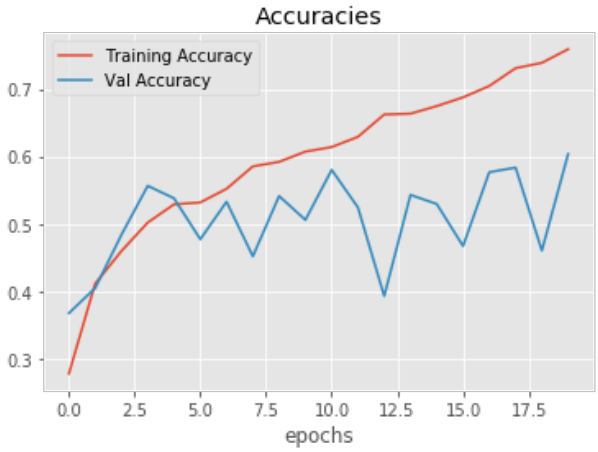

Accuracies
Training Accuracy
Val Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8
10
12
14
epochs

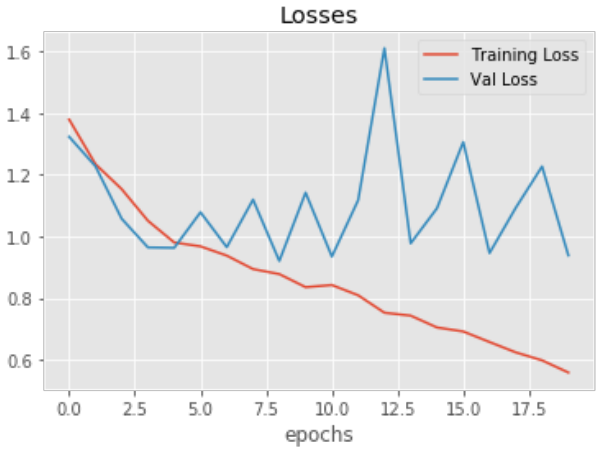

Losses
Training Loss
Val Loss
1.4
1.2
1.0
0.8
0.6
0
2
4
6
8
10
12
14
epochs

C-DRNN

In [83]:
```
from keras.layers import Input,Dense,concatenate,Flatten,GRU,Conv1D
from keras.models import Model

inputs= Input(shape=(t,f))

x = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(inputs)
x = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
x = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)

res1 = GRU(32,activation='tanh',return_sequences=True)(x)
res2 = GRU(32,activation='tanh',return_sequences=True)(res1)

res1_2 = concatenate([res1,res2],axis=2)

res3 = GRU(32,activation='tanh',return_sequences=True)(res1_2)

x = concatenate([res1,res2,res3])

x = GRU(32,activation='tanh')(x)
predictions = Dense(4,activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)
#print(model.summary())
model.compile(optimizer = 'rmsprop',
              loss = 'categorical_crossentropy',
              metrics=['accuracy'])

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=20,validation
_split=0.25,batch_size=32,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=32)

print "Test Results are ", test_score
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
Train on 1777 samples, validate on 593 samples
Epoch 1/20
1777/1777 [==============================] - 30s 17ms/step - loss: 1.3788 -
acc: 0.2780 - val_loss: 1.3228 - val_acc: 0.3676
Epoch 2/20
1777/1777 [==============================] - 15s 8ms/step - loss: 1.2343 -
acc: 0.4114 - val_loss: 1.2274 - val_acc: 0.4047
Epoch 3/20
1777/1777 [==============================] - 15s 8ms/step - loss: 1.1532 -
acc: 0.4598 - val_loss: 1.0572 - val_acc: 0.4840
Epoch 4/20
1777/1777 [==============================] - 15s 8ms/step - loss: 1.0506 -
acc: 0.5020 - val_loss: 0.9643 - val_acc: 0.5565
Epoch 5/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.9803 -
acc: 0.5290 - val_loss: 0.9630 - val_acc: 0.5379
Epoch 6/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.9677 -
acc: 0.5318 - val_loss: 1.0782 - val_acc: 0.4772
Epoch 7/20
1777/1777 [==============================] - 15s 8ms/step - loss: 0.9377 -
acc: 0.5521 - val_loss: 0.9651 - val_acc: 0.5329
Epoch 8/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.8941 -
acc: 0.5853 - val_loss: 1.1192 - val_acc: 0.4519
Epoch 9/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.8779 -
acc: 0.5920 - val_loss: 0.9204 - val_acc: 0.5413
Epoch 10/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.8355 -
acc: 0.6072 - val_loss: 1.1417 - val_acc: 0.5059
Epoch 11/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.8425 -
acc: 0.6140 - val_loss: 0.9347 - val_acc: 0.5801
Epoch 12/20
1777/1777 [==============================] - 15s 8ms/step - loss: 0.8093 -
acc: 0.6292 - val_loss: 1.1173 - val_acc: 0.5245
Epoch 13/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.7527 -
acc: 0.6624 - val_loss: 1.6097 - val_acc: 0.3929
Epoch 14/20
1777/1777 [==============================] - 16s 9ms/step - loss: 0.7433 -
acc: 0.6635 - val_loss: 0.9775 - val_acc: 0.5430
Epoch 15/20
1777/1777 [==============================] - 16s 9ms/step - loss: 0.7050 -
acc: 0.6747 - val_loss: 1.0911 - val_acc: 0.5295
Epoch 16/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6920 -
acc: 0.6877 - val_loss: 1.3057 - val_acc: 0.4671
Epoch 17/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6578 -
acc: 0.7046 - val_loss: 0.9459 - val_acc: 0.5767
Epoch 18/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6243 -
acc: 0.7310 - val_loss: 1.0929 - val_acc: 0.5835
Epoch 19/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.5984 -
acc: 0.7389 - val_loss: 1.2265 - val_acc: 0.4604
Epoch 20/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.5590 -
acc: 0.7591 - val_loss: 0.9386 - val_acc: 0.6037
500/500 [==============================] - 1s 2ms/step
Test Results are  [0.86857534575462336, 0.59999999999999998]
```

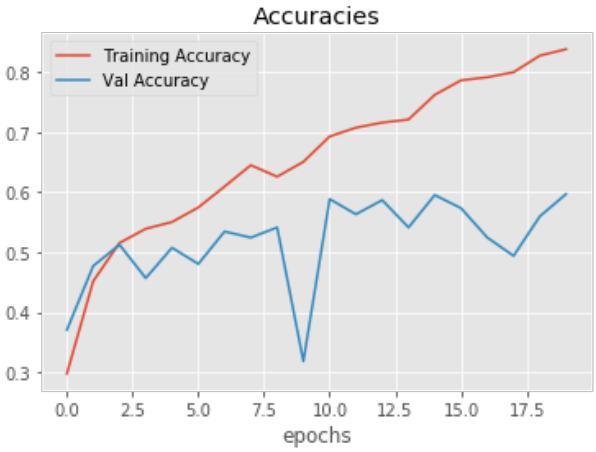


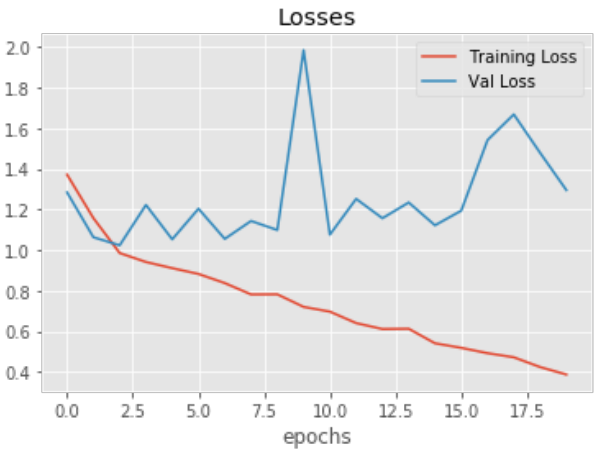


C-DRNN Testing

In [85]:
```
from keras.layers import Input,Dense,concatenate,Flatten,GRU,Conv1D,BatchNo
rmalization,Dropout
from keras.models import Model

inputs= Input(shape=(t,f))

x = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(inputs)
x = BatchNormalization()(x)
x = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
x = BatchNormalization()(x)
x = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
x = Dropout(0.5)(x)

res1 = GRU(32,activation='tanh',return_sequences=True)(x)
res2 = GRU(32,activation='tanh',return_sequences=True)(res1)

res1_2 = concatenate([res1,res2],axis=2)

res3 = GRU(32,activation='tanh',return_sequences=True)(res2)

x = concatenate([res1,res2,res3])

x = GRU(32,activation='tanh')(x)
predictions = Dense(4,activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=20,validation
_split=0.25,batch_size=32,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=32)

print "Test Results are ", test_score
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
Train on 1777 samples, validate on 593 samples
Epoch 1/20
1777/1777 [==============================] - 32s 18ms/step - loss: 1.3716 -
acc: 0.2983 - val_loss: 1.2838 - val_acc: 0.3710
Epoch 2/20
1777/1777 [==============================] - 15s 9ms/step - loss: 1.1575 -
acc: 0.4519 - val_loss: 1.0632 - val_acc: 0.4772
Epoch 3/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.9854 -
acc: 0.5155 - val_loss: 1.0235 - val_acc: 0.5126
Epoch 4/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.9412 -
acc: 0.5391 - val_loss: 1.2226 - val_acc: 0.4570
Epoch 5/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.9109 -
acc: 0.5504 - val_loss: 1.0529 - val_acc: 0.5076
Epoch 6/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.8823 -
acc: 0.5746 - val_loss: 1.2038 - val_acc: 0.4806
Epoch 7/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.8374 -
acc: 0.6095 - val_loss: 1.0551 - val_acc: 0.5346
Epoch 8/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.7813 -
acc: 0.6449 - val_loss: 1.1435 - val_acc: 0.5245
Epoch 9/20
1777/1777 [==============================] - 15s 8ms/step - loss: 0.7818 -
acc: 0.6258 - val_loss: 1.0988 - val_acc: 0.5413
Epoch 10/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.7199 -
acc: 0.6505 - val_loss: 1.9853 - val_acc: 0.3187
Epoch 11/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6966 -
acc: 0.6927 - val_loss: 1.0761 - val_acc: 0.5885
Epoch 12/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6401 -
acc: 0.7074 - val_loss: 1.2524 - val_acc: 0.5632
Epoch 13/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6108 -
acc: 0.7158 - val_loss: 1.1570 - val_acc: 0.5868
Epoch 14/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.6122 -
acc: 0.7209 - val_loss: 1.2346 - val_acc: 0.5413
Epoch 15/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.5405 -
acc: 0.7620 - val_loss: 1.1221 - val_acc: 0.5953
Epoch 16/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.5181 -
acc: 0.7862 - val_loss: 1.1942 - val_acc: 0.5734
Epoch 17/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.4918 -
acc: 0.7912 - val_loss: 1.5421 - val_acc: 0.5245
Epoch 18/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.4716 -
acc: 0.7997 - val_loss: 1.6688 - val_acc: 0.4941
Epoch 19/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.4234 -
acc: 0.8272 - val_loss: 1.4796 - val_acc: 0.5599
Epoch 20/20
1777/1777 [==============================] - 15s 9ms/step - loss: 0.3865 -
acc: 0.8379 - val_loss: 1.2957 - val_acc: 0.5970
500/500 [==============================] - 1s 2ms/step
Test Results are  [1.1160675964355469, 0.60999999904632574]
```

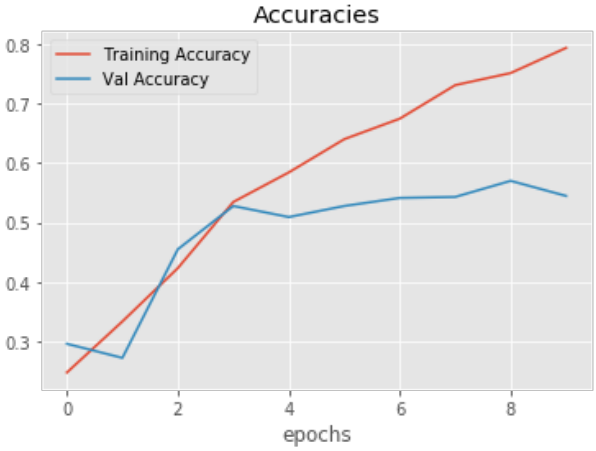

Accuracies
Training Accuracy
Val Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs

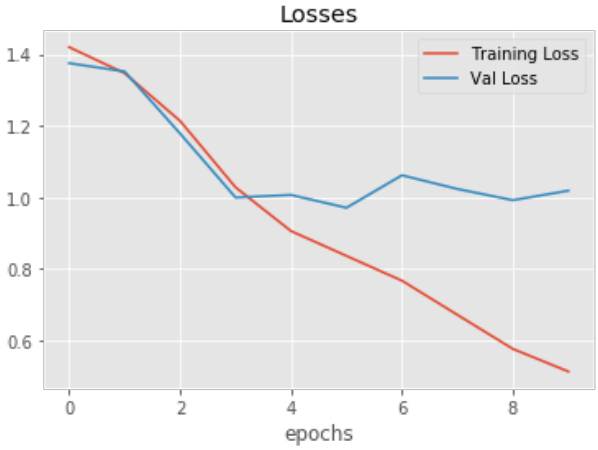

Losses
Training Loss
Val Loss
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs

ChronoNet

In [86]:
```
from keras.layers import Input,Dense,concatenate,Flatten,GRU,Conv1D
from keras.models import Model

inputs= Input(shape=(t,f))

# First Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(inputs
)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(inputs
)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(inputs
)
x = concatenate([tower1,tower2,tower3],axis=2)

# Second Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
x = concatenate([tower1,tower2,tower3],axis=2)

# Third Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
x = concatenate([tower1,tower2,tower3],axis=2)

res1 = GRU(32,activation='tanh',return_sequences=True)(x)
res2 = GRU(32,activation='tanh',return_sequences=True)(res1)

res1_2 = concatenate([res1,res2],axis=2)

res3 = GRU(32,activation='tanh',return_sequences=True)(res1_2)

x = concatenate([res1,res2,res3])

x = GRU(32,activation='tanh')(x)
predictions = Dense(4,activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=10,validation
_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=64)

print "Testing Accuracy is", test_score
plot_hist([hist.history['acc'],hist.history['val_acc']],['Training Accuracy
','Val Accuracy'],title='Accuracies')
plot_hist([hist.history['loss'],hist.history['val_loss']],['Training Loss',
'Val Loss'],title='Losses')
```

```
Train on 1777 samples, validate on 593 samples
Epoch 1/10
1777/1777 [==============================] - 24s 14ms/step - loss: 1.4201 -
acc: 0.2487 - val_loss: 1.3753 - val_acc: 0.2968
Epoch 2/10
1777/1777 [==============================] - 8s 4ms/step - loss: 1.3475 - a
cc: 0.3343 - val_loss: 1.3522 - val_acc: 0.2732
Epoch 3/10
1777/1777 [==============================] - 8s 4ms/step - loss: 1.2141 - a
cc: 0.4237 - val_loss: 1.1789 - val_acc: 0.4553
Epoch 4/10
1777/1777 [==============================] - 8s 4ms/step - loss: 1.0286 - a
cc: 0.5346 - val_loss: 1.0001 - val_acc: 0.5278
Epoch 5/10
1777/1777 [==============================] - 8s 4ms/step - loss: 0.9063 - a
cc: 0.5841 - val_loss: 1.0072 - val_acc: 0.5093
Epoch 6/10
1777/1777 [==============================] - 8s 4ms/step - loss: 0.8369 - a
cc: 0.6398 - val_loss: 0.9714 - val_acc: 0.5278
Epoch 7/10
1777/1777 [==============================] - 8s 4ms/step - loss: 0.7677 - a
cc: 0.6742 - val_loss: 1.0621 - val_acc: 0.5413
Epoch 8/10
1777/1777 [==============================] - 8s 4ms/step - loss: 0.6722 - a
cc: 0.7304 - val_loss: 1.0238 - val_acc: 0.5430
Epoch 9/10
1777/1777 [==============================] - 8s 4ms/step - loss: 0.5770 - a
cc: 0.7507 - val_loss: 0.9926 - val_acc: 0.5700
Epoch 10/10
1777/1777 [==============================] - 8s 4ms/step - loss: 0.5137 - a
cc: 0.7929 - val_loss: 1.0191 - val_acc: 0.5447
500/500 [==============================] - 0s 910us/step
Testing Accuracy is [0.90687944078445437, 0.57400000238418581]
```

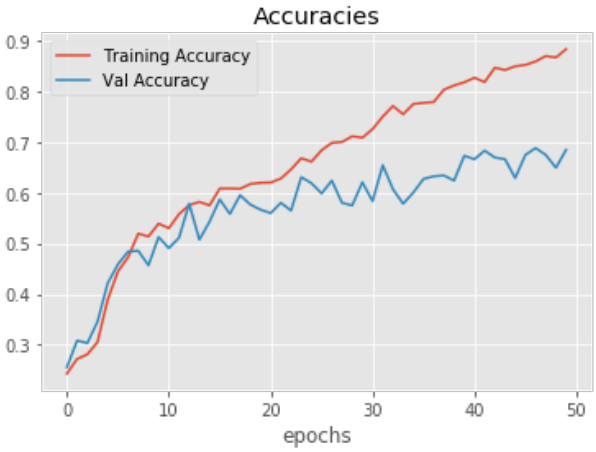




ChronoNet Model Testing

In [9]:
```
from keras.layers import Input,Dense,concatenate,Flatten,GRU,Conv1D
from keras.models import Model

inputs= Input(shape=(t,f))

# First Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(inputs
)
tower1 = BatchNormalization()(tower1)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(inputs
)
tower2 = BatchNormalization()(tower2)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(inputs
)
tower3 = BatchNormalization()(tower3)
x = concatenate([tower1,tower2,tower3],axis=2)
x = Dropout(0.55)(x)

# Second Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower1 = BatchNormalization()(tower1)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower2 = BatchNormalization()(tower2)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
tower3 = BatchNormalization()(tower3)
x = concatenate([tower1,tower2,tower3],axis=2)
x = Dropout(0.55)(x)

# Third Inception
tower1 = Conv1D(32, 2, strides=2,activation='relu',padding="causal")(x)
tower1 = BatchNormalization()(tower1)
tower2 = Conv1D(32, 4, strides=2,activation='relu',padding="causal")(x)
tower2 = BatchNormalization()(tower2)
tower3 = Conv1D(32, 8, strides=2,activation='relu',padding="causal")(x)
tower3 = BatchNormalization()(tower3)
x = concatenate([tower1,tower2,tower3],axis=2)
x = Dropout(0.55)(x)

res1 = GRU(32,activation='tanh',return_sequences=True)(x)
res2 = GRU(32,activation='tanh',return_sequences=True)(res1)

res1_2 = concatenate([res1,res2],axis=2)

res3 = GRU(32,activation='tanh',return_sequences=True)(res1_2)

x = concatenate([res1,res2,res3])

x = GRU(32,activation='tanh')(x)
predictions = Dense(4,activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)

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

#hist.history is a dictionary with all accs and losses
hist = model.fit(train_data_sliced,train_labels_sliced,epochs=50,validation
_split=0.25,batch_size=64,verbose=1)
test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_siz
e=64)

print "Testing Accuracy is", test_score
```

```
WARNING:tensorflow:From /home/kunal/Desktop/FinalProject/venv/local/lib/pyt
hon2.7/site-packages/tensorflow/python/util/deprecation.py:497: calling con
v1d (from tensorflow.python.ops.nn_ops) with data_format=NHWC is deprecated
and will be removed in a future version.
Instructions for updating:
`NHWC` for data_format is deprecated, use `NWC` instead
Train on 1777 samples, validate on 593 samples
Epoch 1/50
1777/1777 [==============================] - 21s 12ms/step - loss: 1.4205 -
acc: 0.2437 - val_loss: 1.4091 - val_acc: 0.2563
Epoch 2/50
1777/1777 [==============================] - 21s 12ms/step - loss: 1.3939 -
acc: 0.2724 - val_loss: 1.3933 - val_acc: 0.3086
Epoch 3/50
1777/1777 [==============================] - 25s 14ms/step - loss: 1.3852 -
acc: 0.2814 - val_loss: 1.3671 - val_acc: 0.3035
Epoch 4/50
1777/1777 [==============================] - 23s 13ms/step - loss: 1.3581 -
acc: 0.3056 - val_loss: 1.3402 - val_acc: 0.3457
Epoch 5/50
1777/1777 [==============================] - 23s 13ms/step - loss: 1.2530 -
acc: 0.3889 - val_loss: 1.2564 - val_acc: 0.4216
Epoch 6/50
1777/1777 [==============================] - 26s 15ms/step - loss: 1.1536 -
acc: 0.4446 - val_loss: 1.1384 - val_acc: 0.4587
Epoch 7/50
1777/1777 [==============================] - 29s 16ms/step - loss: 1.0491 -
acc: 0.4727 - val_loss: 1.0734 - val_acc: 0.4840
Epoch 8/50
1777/1777 [==============================] - 25s 14ms/step - loss: 0.9717 -
acc: 0.5194 - val_loss: 1.0787 - val_acc: 0.4857
Epoch 9/50
1777/1777 [==============================] - 21s 12ms/step - loss: 0.9869 -
acc: 0.5138 - val_loss: 1.1175 - val_acc: 0.4570
Epoch 10/50
1777/1777 [==============================] - 20s 11ms/step - loss: 0.9459 -
acc: 0.5391 - val_loss: 1.0410 - val_acc: 0.5126
Epoch 11/50
1777/1777 [==============================] - 17s 10ms/step - loss: 0.9290 -
acc: 0.5301 - val_loss: 1.0510 - val_acc: 0.4907
Epoch 12/50
1777/1777 [==============================] - 17s 10ms/step - loss: 0.9093 -
acc: 0.5577 - val_loss: 1.0248 - val_acc: 0.5110
Epoch 13/50
1777/1777 [==============================] - 17s 10ms/step - loss: 0.9020 -
acc: 0.5757 - val_loss: 0.9908 - val_acc: 0.5784
Epoch 14/50
1777/1777 [==============================] - 17s 10ms/step - loss: 0.8903 -
acc: 0.5819 - val_loss: 1.0287 - val_acc: 0.5076
Epoch 15/50
1777/1777 [==============================] - 17s 10ms/step - loss: 0.8806 -
acc: 0.5751 - val_loss: 1.0191 - val_acc: 0.5430
Epoch 16/50
1777/1777 [==============================] - 19s 11ms/step - loss: 0.8457 -
acc: 0.6083 - val_loss: 0.9532 - val_acc: 0.5868
Epoch 17/50
1777/1777 [==============================] - 21s 12ms/step - loss: 0.8321 -
acc: 0.6083 - val_loss: 1.0112 - val_acc: 0.5582
Epoch 18/50
1777/1777 [==============================] - 17s 10ms/step - loss: 0.8362 -
acc: 0.6078 - val_loss: 0.9233 - val_acc: 0.5953
Epoch 19/50
1777/1777 [==============================] - 19s 11ms/step - loss: 0.8190 -
```


Accuracies
Training Accuracy
Val Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
epochs


Losses
Training Loss
Val Loss
1.4
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50
epochs