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

In [2]:
```
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.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from UtilNNDL import *
```

In [3]:
```
"""
#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[3]:
```
"\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 [4]:
```
"""
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[4]:
```
'\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 [5]:
```
#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 =
9)
print train_data.shape
print train_labels.shape
print test_data.shape
print test_labels.shape
```

```
(2108, 22, 1000)
(2108, 4)
(450, 22, 1000)
(450, 4)
```

In [6]:
```
test_labels[449]
```

Out[6]:
```
array([ 0.,  0.,  1.,  0.])
```

In [7]:
```
#assist numerical stability
train_data = train_data*(1e6)
test_data = test_data*(1e6)
```

In [8]:
```
#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
```

```
(2108, 1000, 22)
(450, 1000, 22)
(2108, 1000, 22)
(450, 1000, 22)
```

In [9]:
```
#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
```

```
(2108, 22, 1000)
(450, 22, 1000)
```

In [10]:
```
#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
```

```
(21080, 512, 22)
(21080, 4)
(4500, 512, 22)
(4500, 4)
```

Everything from this point down is Testing

In [11]:
```
# Modify test data to run model on another subject
_, test_data_orig, _, test_labels_orig = prepare_data(file_path,
                                                      num_test_samples=50,
                                                      verbose=False,
                                                      return_all=False,
                                                      num_files=9)
```

In [12]:
```
test_data_all = []
test_labels_all = []
for n in range(1,10):
    test_data = test_data_orig['A0{}T'.format(n)]
    test_labels = test_labels_orig['A0{}T'.format(n)]
    #print test_data.shape
    #print test_labels.shape

    #assist numerical stability
    test_data = test_data*(1e6)
    test_data = test_data.swapaxes(1,2)
    for i,a in enumerate(test_data):
        test_data[i] = bandpass_cnt(a, 4, 38, 250, filt_order=3)
    #print test_data.shape

    #standardize
    for i,a in enumerate(test_data):
        test_data[i] = exponential_running_standardize(a, factor_new=0.001,
init_block_size=1000, eps=1e-4)

    test_data = test_data.swapaxes(1,2)
    test_data_sliced, test_labels_sliced = create_window_data(test_data, te
st_labels, windows=10)

    test_data_sliced = test_data_sliced[:500,:]
    test_labels_sliced = test_labels_sliced[:500,:]

    test_data_sliced = test_data_sliced.swapaxes(1,2)

    #print test_data_sliced.shape
    #print test_labels_sliced.shape

    test_data_all.append(test_data_sliced)
    test_labels_all.append(test_labels_sliced)
```

In [16]:
```
whos
```

```
Variable                            Type        Data/Info
-----------------------------------------------------
Activation                          type        <class 'keras.layers.core.Act
ivation'>
Axes3D                              type        <class 'mpl_toolkits.mplot3d.
axes3d.Axes3D'>
BatchNormalization                  type        <class 'keras.layers.norm<...
>tion.BatchNormalization'>
Convolution2D                       type        <class 'keras.layers.convolut
ional.Conv2D'>
Dense                               type        <class 'keras.layers.core.Den
se'>
Dropout                             type        <class 'keras.layers.core.Dro
pout'>
Flatten                             type        <class 'keras.layers.core.Fla
tten'>
LSTM                                type        <class 'keras.layers.recurren
t.LSTM'>
MaxPooling2D                        type        <class 'keras.layers.pooling.
MaxPooling2D'>
Sequential                          type        <class 'keras.models.Sequenti
al'>
SimpleRNN                           type        <class 'keras.layers.recurren
t.SimpleRNN'>
Spectrogram                         type        <class 'kapre.time_frequency.
Spectrogram'>
StandardScaler                      type        <class 'sklearn.preproces<...
>ing.data.StandardScaler'>
a                                   ndarray     1000x22: 22000 elems, type `f
loat64`, 176000 bytes (171 kb)
bandpass_cnt                        function    <function bandpass_cnt at 0x7
fcdf993a8c0>
bs                                  int         21080
confusion_matrix                    function    <function confusion_matrix at
0x7fce042adaa0>
create_window_data                  function    <function create_window_data
at 0x7fcdf99afb90>
datetime                            type        <type 'datetime.datetime'>
display                             module      <module 'librosa.display'<...
>ges/librosa/display.pyc'>
exponential_running_standardize     function    <function exponential_run<...
>ardize at 0x7fcdf993a848>
f                                   int         22
file_path                           str         /home/carla/Downloads/pro<...
>atasets/project_datasets/
filter_is_stable                    function    <function filter_is_stable at
0x7fcdf993a938>
h5py                                module      <module 'h5py' from '/hom<...
>kages/h5py/__init__.pyc'>
i                                   int         49
itertools                           module      <module 'itertools' (built-in
)>
kapre                               module      <module 'kapre' from '/ho<...
>.egg/kapre/__init__.pyc'>
keras                               module      <module 'keras' from '/ho<...
>ages/keras/__init__.pyc'>
librosa                             module      <module 'librosa' from '/<...
>es/librosa/__init__.pyc'>
n                                   int         9
now                                 datetime    2018-03-18 03:02:21.129050
np                                  module      <module 'numpy' from '/us<...
>ages/numpy/__init__.pyc'>
pd                                  module      <module 'pandas' from '/h<...
```

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,
```

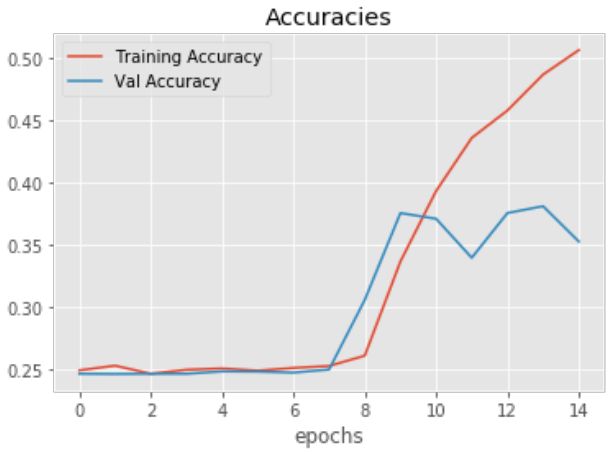
Modified VGGnet for this type of data

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 [17]:
```
### 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_si
ze=64)
#print(test_score)

conf = []
test_score = []

for i in range(9):
    print "Subject {}".format(i+1)
    test_score_i = model.evaluate(test_data_all[i], test_labels_all[i], bat
ch_size=64)
    print "Test Score: {}".format(test_score_i)
    test_predict = model.predict(test_data_all[i], batch_size=16)
    cm = confusion_matrix(np.argmax(test_labels_all[i],axis=1),np.argmax(te
st_predict,axis=1))
    conf.append(cm)
    test_score.append(test_score_i)
```

```
Subject 1
500/500 [==============================] - 0s 608us/step
Test Score: [1.1591282806396483, 0.41999999904632568]
Subject 2
500/500 [==============================] - 0s 107us/step
Test Score: [1.5540989713668822, 0.26200000017881392]
Subject 3
500/500 [==============================] - 0s 115us/step
Test Score: [1.1912931404113769, 0.47800000143051147]
Subject 4
500/500 [==============================] - 0s 112us/step
Test Score: [1.6169553489685058, 0.2960000002384186]
Subject 5
500/500 [==============================] - 0s 124us/step
Test Score: [1.5434520225524901, 0.25800000002980233]
Subject 6
500/500 [==============================] - 0s 134us/step
Test Score: [1.6908332347869872, 0.27600000002980229]
Subject 7
500/500 [==============================] - 0s 129us/step
Test Score: [1.5334240589141845, 0.28800000035762785]
Subject 8
500/500 [==============================] - 0s 116us/step
Test Score: [1.483209119796753, 0.3440000000298023]
Subject 9
500/500 [==============================] - 0s 117us/step
Test Score: [1.515034086227417, 0.28000000071525571]
```

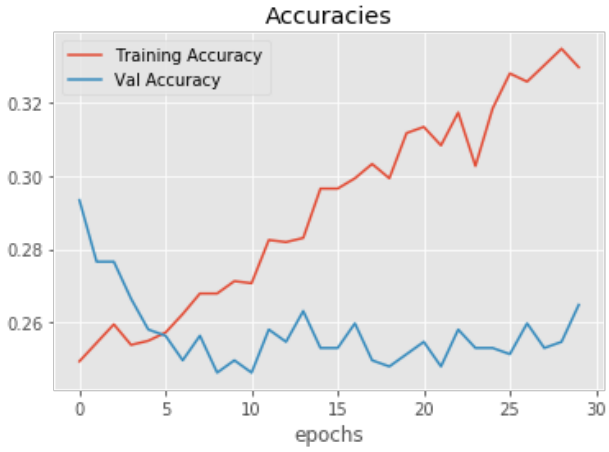


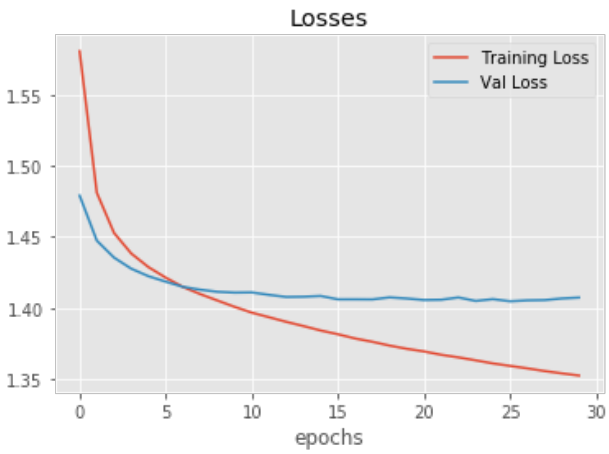
In [18]:
```
np.save('Best Models/Variables/VGG_hist_All',hist.history)
np.save('Best Models/Variables/VGG_testacc_All',test_score)
np.save('Best Models/Variables/VGG_conf_All',conf)
```

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]
```



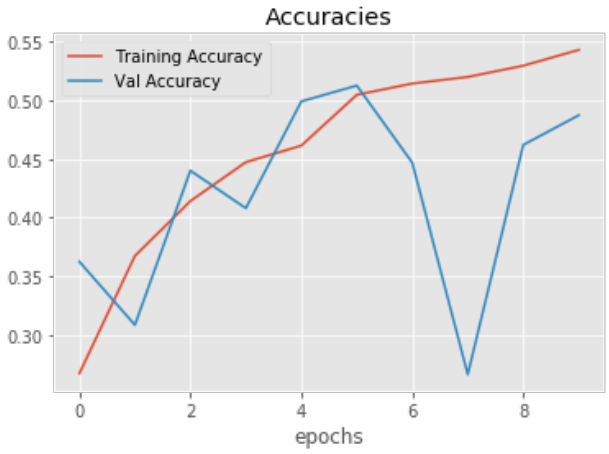


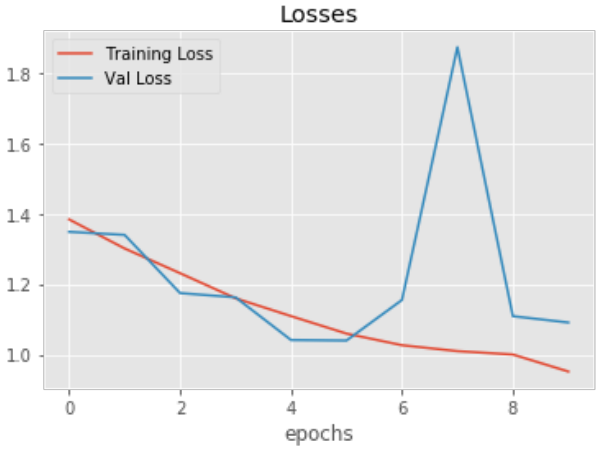
CHRONONET PAPER

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]
```

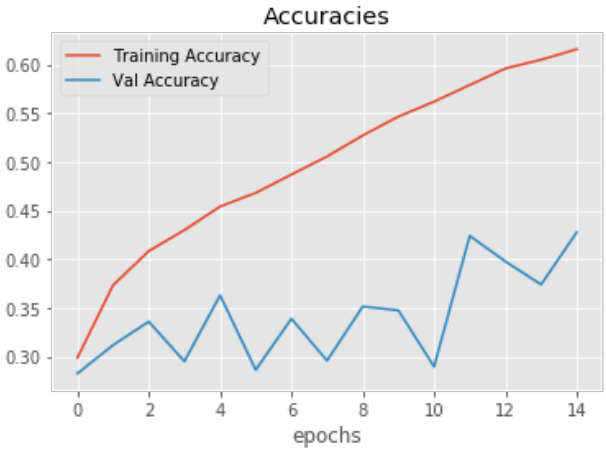


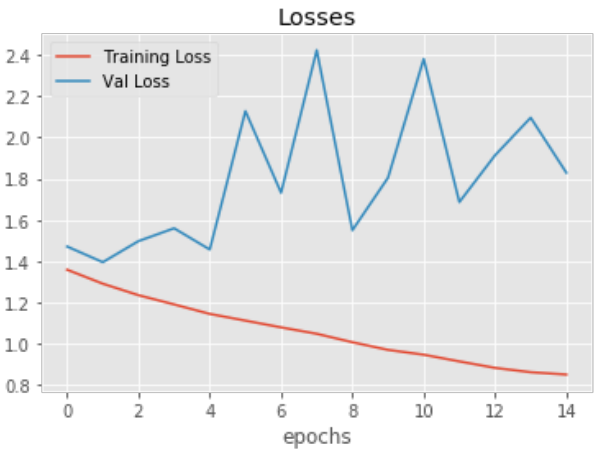


Implementation of Figure 1b but adding regularization structures like that found in VGGnet

In [26]:
```
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_si
ze=16)
#print(test_score)

conf = []
test_score = []

for i in range(9):
    print "Subject {}".format(i+1)
    test_score_i = model.evaluate(test_data_all[i], test_labels_all[i], bat
ch_size=64)
    print "Test Score: {}".format(test_score_i)
    test_predict = model.predict(test_data_all[i], batch_size=16)
    cm = confusion_matrix(np.argmax(test_labels_all[i],axis=1),np.argmax(te
st_predict,axis=1))
```

```
Train on 15810 samples, validate on 5270 samples
Epoch 1/15
15810/15810 [==============================] - 34s 2ms/step - loss: 1.3592
- acc: 0.2988 - val_loss: 1.4715 - val_acc: 0.2825
Epoch 2/15
15810/15810 [==============================] - 34s 2ms/step - loss: 1.2914
- acc: 0.3732 - val_loss: 1.3956 - val_acc: 0.3118
Epoch 3/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.2358
- acc: 0.4084 - val_loss: 1.4977 - val_acc: 0.3359
Epoch 4/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.1911
- acc: 0.4300 - val_loss: 1.5600 - val_acc: 0.2951
Epoch 5/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.1451
- acc: 0.4541 - val_loss: 1.4569 - val_acc: 0.3630
Epoch 6/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.1126
- acc: 0.4682 - val_loss: 2.1275 - val_acc: 0.2863
Epoch 7/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.0799
- acc: 0.4870 - val_loss: 1.7322 - val_acc: 0.3389
Epoch 8/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.0488
- acc: 0.5056 - val_loss: 2.4226 - val_acc: 0.2958
Epoch 9/15
15810/15810 [==============================] - 33s 2ms/step - loss: 1.0079
- acc: 0.5273 - val_loss: 1.5504 - val_acc: 0.3514
Epoch 10/15
15810/15810 [==============================] - 33s 2ms/step - loss: 0.9708
- acc: 0.5466 - val_loss: 1.8052 - val_acc: 0.3474
Epoch 11/15
15810/15810 [==============================] - 33s 2ms/step - loss: 0.9475
- acc: 0.5619 - val_loss: 2.3804 - val_acc: 0.2896
Epoch 12/15
15810/15810 [==============================] - 33s 2ms/step - loss: 0.9150
- acc: 0.5791 - val_loss: 1.6868 - val_acc: 0.4241
Epoch 13/15
15810/15810 [==============================] - 33s 2ms/step - loss: 0.8837
- acc: 0.5961 - val_loss: 1.9134 - val_acc: 0.3977
Epoch 14/15
15810/15810 [==============================] - 33s 2ms/step - loss: 0.8627
- acc: 0.6051 - val_loss: 2.0957 - val_acc: 0.3740
Epoch 15/15
15810/15810 [==============================] - 33s 2ms/step - loss: 0.8514
- acc: 0.6159 - val_loss: 1.8291 - val_acc: 0.4277
Subject 1
500/500 [==============================] - 0s 518us/step
Test Score: [1.3200358228683471, 0.47800000143051147]
Subject 2
500/500 [==============================] - 0s 537us/step
Test Score: [2.2127631096839906, 0.26800000000000002]
Subject 3
500/500 [==============================] - 0s 521us/step
Test Score: [1.5144780712127686, 0.47999999999999998]
Subject 4
500/500 [==============================] - 0s 529us/step
Test Score: [1.7753049964904786, 0.32199999809265134]
Subject 5
500/500 [==============================] - 0s 519us/step
Test Score: [2.386250057220459, 0.29200000119209291]
Subject 6
500/500 [==============================] - 0s 523us/step
```



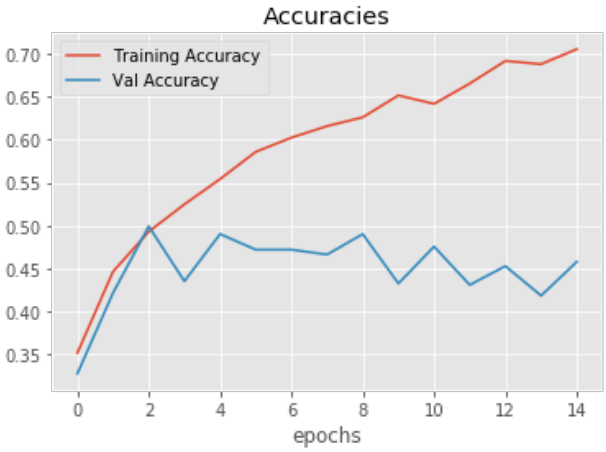


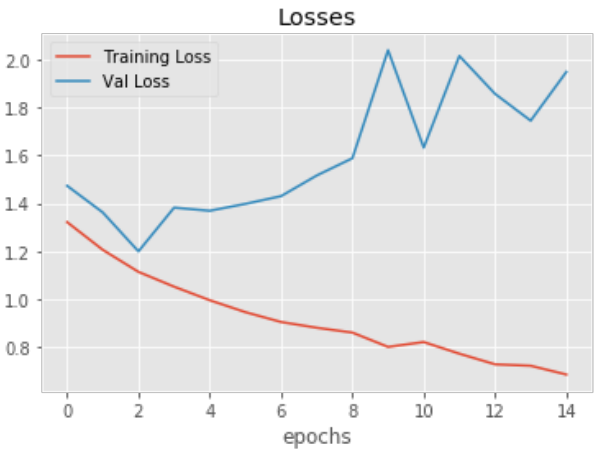
In [27]:
```
np.save('Best Models/Variables/CRNNGRU_hist_All',hist.history)
np.save('Best Models/Variables/CRNNGRU_testacc_All',test_score)
np.save('Best Models/Variables/CRNNGRU_conf_All',conf)
```

Replaced GRU with LSTM

In [28]:
```
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_si
ze=64)
#print "Test Results are ", test_score

conf = []
test_score = []

for i in range(9):
    print "Subject {}".format(i+1)
    test_score_i = model.evaluate(test_data_all[i], test_labels_all[i], bat
ch_size=64)
    print "Test Score: {}".format(test_score_i)
    test_predict = model.predict(test_data_all[i], batch_size=16)
    cm = confusion_matrix(np.argmax(test_labels_all[i],axis=1),np.argmax(te
st_predict,axis=1))
    conf.append(cm)
    test_score.append(test_score_i)

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

```
Train on 15810 samples, validate on 5270 samples
Epoch 1/15
15810/15810 [==============================] - 42s 3ms/step - loss: 1.3210
- acc: 0.3512 - val_loss: 1.4728 - val_acc: 0.3271
Epoch 2/15
15810/15810 [==============================] - 41s 3ms/step - loss: 1.2052
- acc: 0.4462 - val_loss: 1.3615 - val_acc: 0.4213
Epoch 3/15
15810/15810 [==============================] - 40s 3ms/step - loss: 1.1132
- acc: 0.4927 - val_loss: 1.1989 - val_acc: 0.4991
Epoch 4/15
15810/15810 [==============================] - 40s 3ms/step - loss: 1.0513
- acc: 0.5247 - val_loss: 1.3815 - val_acc: 0.4349
Epoch 5/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.9946
- acc: 0.5541 - val_loss: 1.3685 - val_acc: 0.4899
Epoch 6/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.9448
- acc: 0.5859 - val_loss: 1.3974 - val_acc: 0.4717
Epoch 7/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.9037
- acc: 0.6025 - val_loss: 1.4297 - val_acc: 0.4717
Epoch 8/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.8797
- acc: 0.6160 - val_loss: 1.5159 - val_acc: 0.4660
Epoch 9/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.8595
- acc: 0.6262 - val_loss: 1.5880 - val_acc: 0.4899
Epoch 10/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.7997
- acc: 0.6518 - val_loss: 2.0395 - val_acc: 0.4323
Epoch 11/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.8200
- acc: 0.6419 - val_loss: 1.6323 - val_acc: 0.4753
Epoch 12/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.7712
- acc: 0.6657 - val_loss: 2.0156 - val_acc: 0.4306
Epoch 13/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.7268
- acc: 0.6921 - val_loss: 1.8572 - val_acc: 0.4526
Epoch 14/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.7209
- acc: 0.6883 - val_loss: 1.7449 - val_acc: 0.4180
Epoch 15/15
15810/15810 [==============================] - 40s 3ms/step - loss: 0.6841
- acc: 0.7058 - val_loss: 1.9488 - val_acc: 0.4575
Subject 1
500/500 [==============================] - 0s 616us/step
Test Score: [0.80047360515594479, 0.7000000023841858]
Subject 2
500/500 [==============================] - 0s 587us/step
Test Score: [1.8585475826263427, 0.33200000023841858]
Subject 3
500/500 [==============================] - 0s 588us/step
Test Score: [1.1203386631011962, 0.63400000000000001]
Subject 4
500/500 [==============================] - 0s 599us/step
Test Score: [1.6286471433639527, 0.40200000071525571]
Subject 5
500/500 [==============================] - 0s 590us/step
Test Score: [1.57941792011261, 0.36400000143051148]
Subject 6
500/500 [==============================] - 0s 578us/step
```



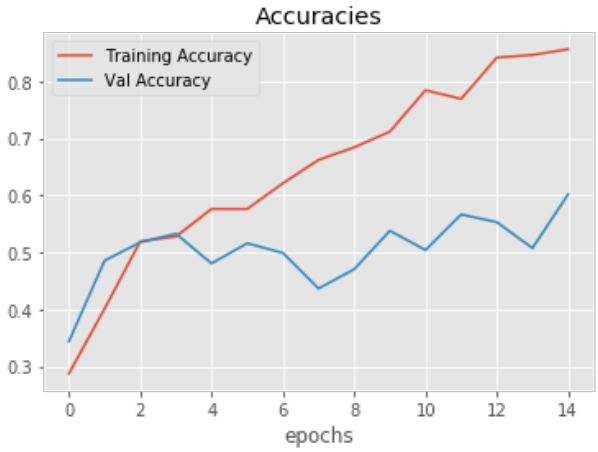


In [29]:
```
np.save('Best Models/Variables/CRNNLSTM_hist_All',hist.history)
np.save('Best Models/Variables/CRNNLSTM_testacc_All',test_score)
np.save('Best Models/Variables/CRNNLSTM_conf_All',conf)
```

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')
```

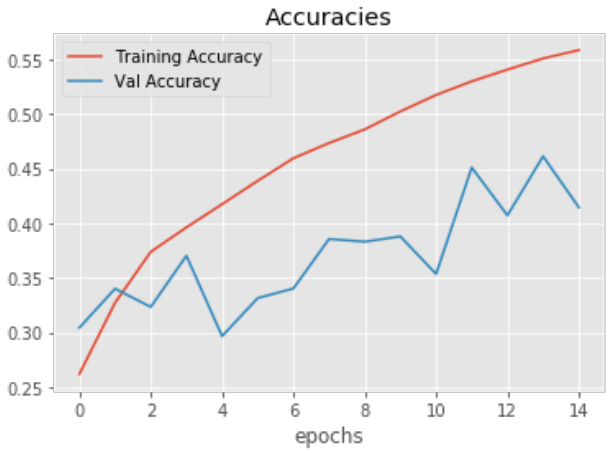
```
Train on 1777 samples, validate on 593 samples
Epoch 1/15
1777/1777 [==============================] - 19s 10ms/step - loss: 1.3920 -
acc: 0.2876 - val_loss: 1.3459 - val_acc: 0.3440
Epoch 2/15
1777/1777 [==============================] - 8s 4ms/step - loss: 1.2776 - a
cc: 0.4012 - val_loss: 1.1440 - 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 [19]:
```
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=0)
#test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_size=64)
```

```
Subject 1
500/500 [==============================] - 0s 917us/step
Test Score: [1.0620317478179933, 0.57200000190734868]
Subject 2
500/500 [==============================] - 0s 814us/step
Test Score: [1.7051339035034179, 0.34400000017881394]
Subject 3
500/500 [==============================] - 0s 811us/step
Test Score: [1.4872502994537353, 0.46600000005960462]
Subject 4
500/500 [==============================] - 0s 854us/step
Test Score: [1.660764591217041, 0.37600000095367431]
Subject 5
500/500 [==============================] - 0s 883us/step
Test Score: [1.8736546630859374, 0.27400000017881393]
Subject 6
500/500 [==============================] - 0s 806us/step
Test Score: [2.0726626338958742, 0.2980000001192093]
Subject 7
500/500 [==============================] - 0s 882us/step
Test Score: [1.6862463340759277, 0.38400000035762788]
Subject 8
500/500 [==============================] - 0s 836us/step
Test Score: [1.6732919826507568, 0.39000000000000001]
Subject 9
500/500 [==============================] - 0s 867us/step
Test Score: [2.1447908878326416, 0.30000000017881395]
```

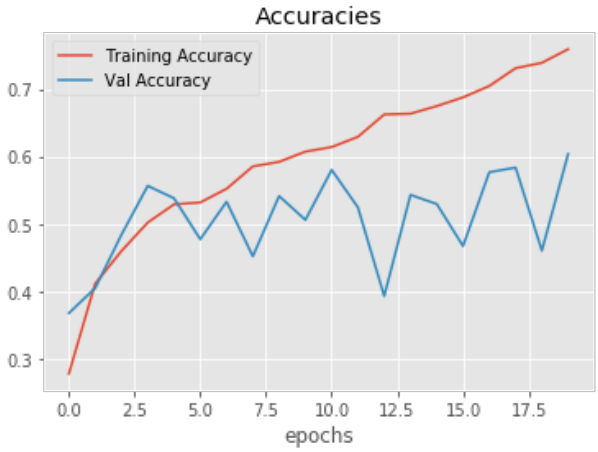


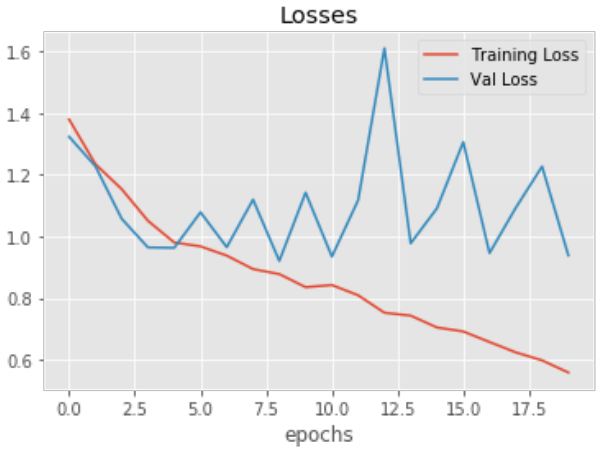
In [20]:
```
np.save('Best Models/Variables/ICRNN_hist_All',hist.history)
np.save('Best Models/Variables/ICRNN_testacc_All',test_score)
np.save('Best Models/Variables/ICRNN_conf_All',conf)
```

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]
```

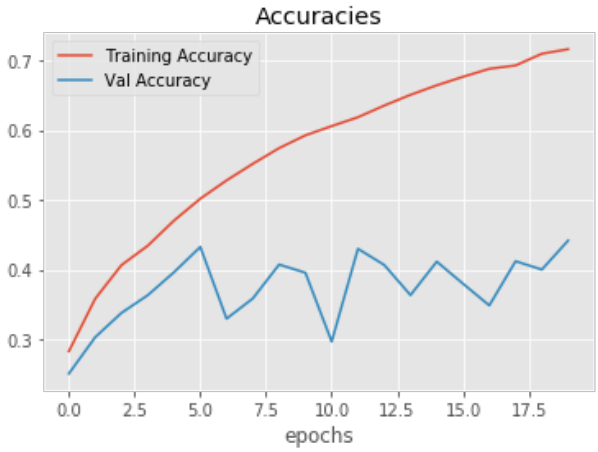


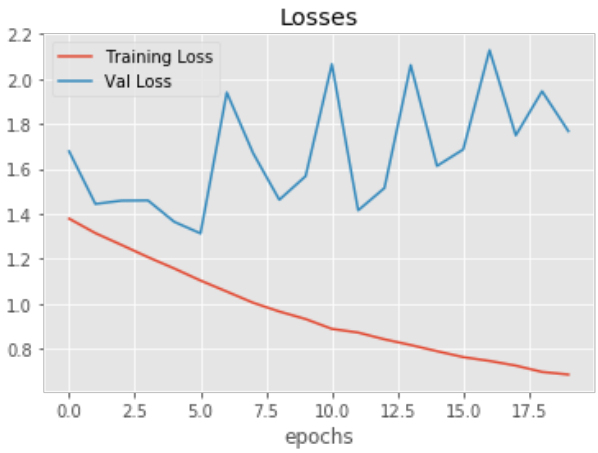


C-DRNN Testing

In [33]:
```
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_si
ze=32)
#print "Test Results are ", test_score

conf = []
test_score = []

for i in range(9):
    print "Subject {}".format(i+1)
    test_score_i = model.evaluate(test_data_all[i], test_labels_all[i], bat
ch_size=64)
    print "Test Score: {}".format(test_score_i)
    test_predict = model.predict(test_data_all[i], batch_size=16)
    cm = confusion_matrix(np.argmax(test_labels_all[i],axis=1),np.argmax(te
st_predict,axis=1))
    conf.append(cm)
    test_score.append(test_score_i)

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 15810 samples, validate on 5270 samples
Epoch 1/20
15810/15810 [==============================] - 133s 8ms/step - loss: 1.3784
- acc: 0.2832 - val_loss: 1.6776 - val_acc: 0.2512
Epoch 2/20
15810/15810 [==============================] - 130s 8ms/step - loss: 1.3139
- acc: 0.3583 - val_loss: 1.4438 - val_acc: 0.3032
Epoch 3/20
15810/15810 [==============================] - 129s 8ms/step - loss: 1.2611
- acc: 0.4066 - val_loss: 1.4585 - val_acc: 0.3381
Epoch 4/20
15810/15810 [==============================] - 130s 8ms/step - loss: 1.2071
- acc: 0.4343 - val_loss: 1.4595 - val_acc: 0.3634
Epoch 5/20
15810/15810 [==============================] - 130s 8ms/step - loss: 1.1566
- acc: 0.4703 - val_loss: 1.3643 - val_acc: 0.3964
Epoch 6/20
15810/15810 [==============================] - 130s 8ms/step - loss: 1.1030
- acc: 0.5017 - val_loss: 1.3127 - val_acc: 0.4326
Epoch 7/20
15810/15810 [==============================] - 130s 8ms/step - loss: 1.0539
- acc: 0.5279 - val_loss: 1.9397 - val_acc: 0.3298
Epoch 8/20
15810/15810 [==============================] - 130s 8ms/step - loss: 1.0042
- acc: 0.5517 - val_loss: 1.6704 - val_acc: 0.3588
Epoch 9/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.9651
- acc: 0.5741 - val_loss: 1.4623 - val_acc: 0.4074
Epoch 10/20
15810/15810 [==============================] - 129s 8ms/step - loss: 0.9316
- acc: 0.5925 - val_loss: 1.5672 - val_acc: 0.3956
Epoch 11/20
15810/15810 [==============================] - 129s 8ms/step - loss: 0.8879
- acc: 0.6057 - val_loss: 2.0654 - val_acc: 0.2970
Epoch 12/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.8715
- acc: 0.6184 - val_loss: 1.4152 - val_acc: 0.4300
Epoch 13/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.8414
- acc: 0.6351 - val_loss: 1.5144 - val_acc: 0.4068
Epoch 14/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.8161
- acc: 0.6505 - val_loss: 2.0614 - val_acc: 0.3636
Epoch 15/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.7886
- acc: 0.6641 - val_loss: 1.6128 - val_acc: 0.4116
Epoch 16/20
15810/15810 [==============================] - 129s 8ms/step - loss: 0.7622
- acc: 0.6763 - val_loss: 1.6864 - val_acc: 0.3799
Epoch 17/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.7452
- acc: 0.6878 - val_loss: 2.1279 - val_acc: 0.3488
Epoch 18/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.7244
- acc: 0.6926 - val_loss: 1.7490 - val_acc: 0.4121
Epoch 19/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.6963
- acc: 0.7094 - val_loss: 1.9451 - val_acc: 0.4002
Epoch 20/20
15810/15810 [==============================] - 130s 8ms/step - loss: 0.6847
- acc: 0.7159 - val_loss: 1.7676 - val_acc: 0.4417
Subject 1
500/500 [==============================] - 0s 866us/step
```



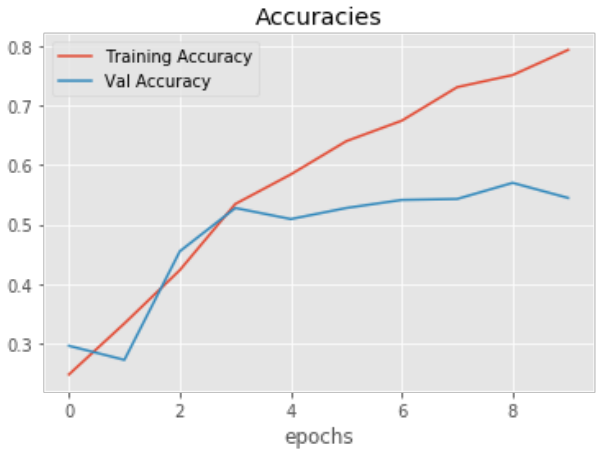


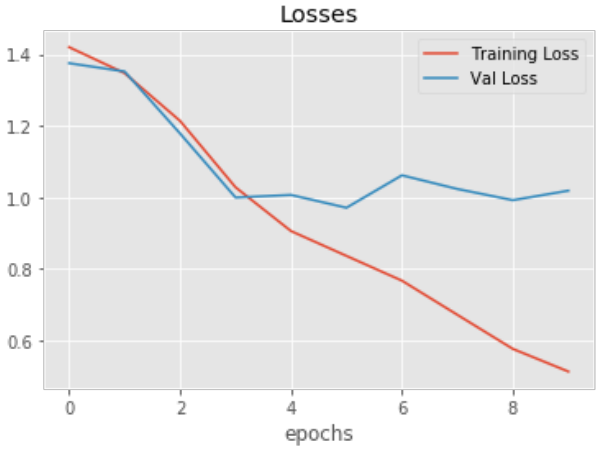
In [34]:
```
np.save('Best Models/Variables/CRDNN_hist_All',hist.history)
np.save('Best Models/Variables/CRDNN_testacc_All',test_score)
np.save('Best Models/Variables/CRDNN_conf_All',conf)
```

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, 0.5740000023841858]
```



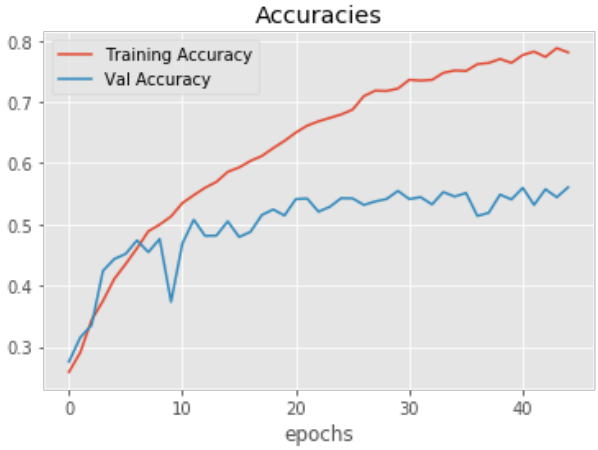


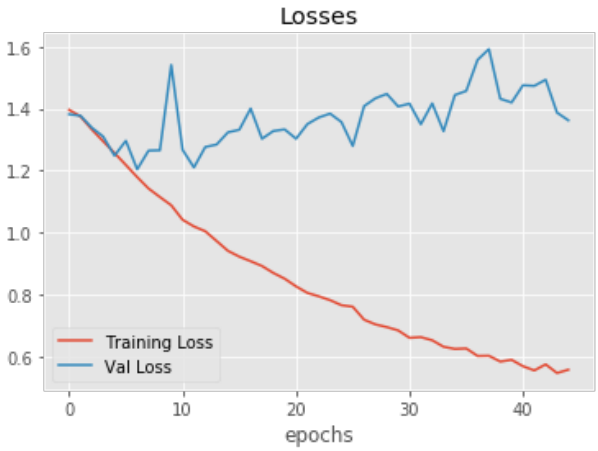
ChronoNet Model Testing

In [ ]:
```
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(0.45)(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(0.45)(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(0.45)(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=45,validation
_split=0.25,batch_size=64,verbose=1)
#test_score = model.evaluate(test_data_sliced, test_labels_sliced, batch_si
ze=64)
#print "Testing Accuracy is", test_score
```

In [59]:
```
conf = []
test_score = []

for i in range(9):
    print "Subject {}".format(i+1)
    test_score_i = model.evaluate(test_data_all[i], test_labels_all[i], bat
ch_size=64)
    print "Test Score: {}".format(test_score_i)
    test_predict = model.predict(test_data_all[i], batch_size=16)
    cm = confusion_matrix(np.argmax(test_labels_all[i],axis=1),np.argmax(te
st_predict,axis=1))
    conf.append(cm)
    test_score.append(test_score_i)

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')
```

```
Subject 1
500/500 [==============================] - 1s 1ms/step
Test Score: [0.61041759014129637, 0.75399999856948852]
Subject 2
500/500 [==============================] - 1s 1ms/step
Test Score: [1.6416517639160155, 0.45600000000000002]
Subject 3
500/500 [==============================] - 1s 1ms/step
Test Score: [1.1189512085914612, 0.68400000190734866]
Subject 4
500/500 [==============================] - 1s 1ms/step
Test Score: [1.5156427974700928, 0.44999999976158145]
Subject 5
500/500 [==============================] - 1s 1ms/step
Test Score: [1.6647456483840943, 0.44600000190734862]
Subject 6
500/500 [==============================] - 1s 1ms/step
Test Score: [1.5529703330993652, 0.48200000071525573]
Subject 7
500/500 [==============================] - 1s 1ms/step
Test Score: [1.2746698608398437, 0.59200000286102294]
Subject 8
500/500 [==============================] - 1s 1ms/step
Test Score: [1.7787003860473634, 0.49200000190734861]
Subject 9
500/500 [==============================] - 1s 1ms/step
Test Score: [2.5035202064514159, 0.30000000059604642]
```





In [62]:
```
#np.save('Best Models/Variables/ChronoNet_hist_All',hist.history)
#np.save('Best Models/Variables/ChronoNet_testacc_All',test_score)
#np.save('Best Models/Variables/ChronoNet_conf_All',conf)
```

In [64]:
```
np.load('Best Models/Variables/ChronoNet_conf_All.npy')
```

Out[64]:
```
array([[[114,   7,   9,   0],
        [ 23,  96,   1,   0],
        [  7,   6,  90,  27],
        [ 11,   3,  29,  77]],

       [[ 39,  14,  36,  31],
        [ 46,  38,  20,  36],
        [ 24,   4,  84,  18],
        [ 35,   0,   8,  67]],

       [[ 96,  11,  11,   2],
        [ 13,  75,   6,  26],
        [ 23,   4,  93,  10],
        [ 21,   1,  30,  78]],

       [[ 65,   6,  44,  15],
        [ 40,  47,  28,   5],
        [ 21,  15,  57,  27],
        [ 11,  15,  48,  56]],

       [[ 43,   8,  27,  42],
        [  7,  32,  22,  69],
        [  5,   3,  68,  54],
        [  9,   4,  27,  80]],

       [[ 91,  17,   3,   9],
        [ 55,  39,  17,  19],
        [ 17,  10,  52,  41],
        [ 31,  19,  21,  59]],

       [[ 83,  29,   6,   2],
        [ 22,  96,   8,   4],
        [ 22,  17,  66,  25],
        [ 14,   5,  50,  51]],

       [[ 40,   1,  45,  24],
        [  4,  56,  53,  17],
        [ 12,   6,  89,  23],
        [  5,  20,  44,  61]],

       [[ 46,  40,  24,  20],
        [ 10,  30,  22,  68],
        [ 28,  28,  20,  54],
        [  9,  28,  19,  54]]])
```