In [295]:
```
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Dropout
from keras.layers import Conv2D,LSTM,BatchNormalization,MaxPooling2D,Reshape
from tensorflow.compat.v1.keras.layers import CuDNNLSTM
from tensorflow.keras.regularizers import L1L2
from keras.utils import to_categorical
import matplotlib.pyplot as plt
```

In [296]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [297]:
```
def data_prep(X,y,sub_sample,average,noise):

    total_X = None
    total_y = None

    # Trimming the data (sample,22,1000) -> (sample,22,500)
    X = X[:,:,0:500]
    print('Shape of X after trimming:',X.shape)

    # Maxpooling the data (sample,22,1000) -> (sample,22,500/sub_sample)
    X_max = np.max(X.reshape(X.shape[0], X.shape[1], -1, sub_sample), axis=3)


    total_X = X_max
    total_y = y
    print('Shape of X after maxpooling:',total_X.shape)

    # Averaging + noise
    X_average = np.mean(X.reshape(X.shape[0], X.shape[1], -1, average),axis=3)
    X_average = X_average + np.random.normal(0.0, 0.5, X_average.shape)

    total_X = np.vstack((total_X, X_average))
    total_y = np.hstack((total_y, y))
    print('Shape of X after averaging+noise and concatenating:',total_X.shape)

    # Subsampling

    for i in range(sub_sample):

        X_subsample = X[:, :, i::sub_sample] + (np.random.normal(0.0, 0.5, X[:

        total_X = np.vstack((total_X, X_subsample))
        total_y = np.hstack((total_y, y))


    print('Shape of X after subsampling and concatenating:',total_X.shape)
    return total_X,total_y
```

In [298]:
```
X_test = np.load("/content/drive/MyDrive/cs/X_test.npy")
y_test = np.load("/content/drive/MyDrive/cs/y_test.npy")
person_train_valid = np.load("/content/drive/MyDrive/cs/person_train_valid.npy
X_train_valid = np.load("/content/drive/MyDrive/cs/X_train_valid.npy")
y_train_valid = np.load("/content/drive/MyDrive/cs/y_train_valid.npy")
person_test = np.load("/content/drive/MyDrive/cs/person_test.npy")

print(X_test.shape)
print(y_test.shape)
print(person_train_valid.shape)
print(X_train_valid.shape)
print(y_train_valid.shape)
print(person_test.shape)

y_train_valid -= 769
y_test -= 769

## Random splitting and reshaping the data
# First generating the training and validation indices using random splitting

ind_valid = np.random.choice(2115, 375, replace=False)
ind_train = np.array(list(set(range(2115)).difference(set(ind_valid))))

# Creating the training and validation sets using the generated indices
(X_train, X_valid) = X_train_valid[ind_train], X_train_valid[ind_valid]
(y_train, y_valid) = y_train_valid[ind_train], y_train_valid[ind_valid]
person_train, person_valid = person_train_valid[ind_train], person_train_valid

## Preprocessing the dataset
x_train,y_train = data_prep(X_train,y_train,2,2,True)
x_valid,y_valid = data_prep(X_valid,y_valid,2,2,True)
X_test_prep,y_test_prep = data_prep(X_test,y_test,2,2,True)

print('Shape of training set:',x_train.shape)
print('Shape of validation set:',x_valid.shape)
print('Shape of training labels:',y_train.shape)
print('Shape of validation labels:',y_valid.shape)
print('Shape of testing set:',X_test_prep.shape)
print('Shape of testing labels:',y_test_prep.shape)

# Converting the labels to categorical variables for multiclass classification
y_train = to_categorical(y_train, 4)
y_valid = to_categorical(y_valid, 4)
y_test = to_categorical(y_test_prep, 4)
print('Shape of training labels after categorical conversion:',y_train.shape)
print('Shape of validation labels after categorical conversion:',y_valid.shape
print('Shape of test labels after categorical conversion:',y_test.shape)

# Adding width of the segment to be 1
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2]
x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_train.shape[2]
x_test = X_test_prep.reshape(X_test_prep.shape[0], X_test_prep.shape[1], X_tes
print('Shape of training set after adding width info:',x_train.shape)
print('Shape of validation set after adding width info:',x_valid.shape)
print('Shape of test set after adding width info:',x_test.shape)

# Reshaping the training and validation dataset
```

```
x_train = np.swapaxes(x_train, 1,3)
x_train = np.swapaxes(x_train, 1,2)
x_valid = np.swapaxes(x_valid, 1,3)
x_valid = np.swapaxes(x_valid, 1,2)
x_test = np.swapaxes(x_test, 1,3)
x_test = np.swapaxes(x_test, 1,2)
print('Shape of training set after dimension reshaping:',x_train.shape)
print('Shape of validation set after dimension reshaping:',x_valid.shape)
print('Shape of test set after dimension reshaping:',x_test.shape)
```

```
(443, 22, 1000)
(443,)
(2115, 1)
(2115, 22, 1000)
(2115,)
(443, 1)
Shape of X after trimming: (1740, 22, 500)
Shape of X after maxpooling: (1740, 22, 250)
Shape of X after averaging+noise and concatenating: (3480, 22, 250)
Shape of X after subsampling and concatenating: (6960, 22, 250)
Shape of X after trimming: (375, 22, 500)
Shape of X after maxpooling: (375, 22, 250)
Shape of X after averaging+noise and concatenating: (750, 22, 250)
Shape of X after subsampling and concatenating: (1500, 22, 250)
Shape of X after trimming: (443, 22, 500)
Shape of X after maxpooling: (443, 22, 250)
Shape of X after averaging+noise and concatenating: (886, 22, 250)
Shape of X after subsampling and concatenating: (1772, 22, 250)
Shape of training set: (6960, 22, 250)
Shape of validation set: (1500, 22, 250)
Shape of training labels: (6960,)
Shape of validation labels: (1500,)
Shape of testing set: (1772, 22, 250)
Shape of testing labels: (1772,)
Shape of training labels after categorical conversion: (6960, 4)
Shape of validation labels after categorical conversion: (1500, 4)
Shape of test labels after categorical conversion: (1772, 4)
Shape of training set after adding width info: (6960, 22, 250, 1)
Shape of validation set after adding width info: (1500, 22, 250, 1)
Shape of test set after adding width info: (1772, 22, 250, 1)
Shape of training set after dimension reshaping: (6960, 250, 1, 22)
Shape of validation set after dimension reshaping: (1500, 250, 1, 22)
Shape of test set after dimension reshaping: (1772, 250, 1, 22)
```

In [299]:
```
person_train = np.vstack((person_train, person_train))
person_train = np.vstack((person_train, person_train))
print("Shape of person_train:", person_train.shape)

person_valid = np.vstack((person_valid, person_valid))
person_valid = np.vstack((person_valid, person_valid))
print("Shape of person_valid:", person_valid.shape)

person_test = np.vstack((person_test, person_test))
person_test = np.vstack((person_test, person_test))
print("Shape of person_test:", person_test.shape)
```

```
Shape of person_train: (6960, 1)
Shape of person_valid: (1500, 1)
Shape of person_test: (1772, 1)
```

(iii)(CNN-LSTM) Defining the architecture of the hybrid CNN-LSTM model

In [300]:
```
def cnn_lstm(time_period=250):
    hybrid_cnn_lstm_model = Sequential()

    # Conv. block 1
    hybrid_cnn_lstm_model.add(Conv2D(filters=25, kernel_size=(10,1), kernel_re
    hybrid_cnn_lstm_model.add(MaxPooling2D(pool_size=(4,1), padding='same'))
    hybrid_cnn_lstm_model.add(BatchNormalization())
    hybrid_cnn_lstm_model.add(Dropout(0.5))

    # Conv. block 2
    hybrid_cnn_lstm_model.add(Conv2D(filters=50, kernel_size=(10,1), kernel_re
    hybrid_cnn_lstm_model.add(MaxPooling2D(pool_size=(4,1), padding='same'))
    hybrid_cnn_lstm_model.add(BatchNormalization())
    hybrid_cnn_lstm_model.add(Dropout(0.5))

    # Conv. block 3
    hybrid_cnn_lstm_model.add(Conv2D(filters=100, kernel_size=(10,1), kernel_r
    hybrid_cnn_lstm_model.add(MaxPooling2D(pool_size=(4,1), padding='same'))
    hybrid_cnn_lstm_model.add(BatchNormalization())
    hybrid_cnn_lstm_model.add(Dropout(0.5))

    # Conv. block 4
    hybrid_cnn_lstm_model.add(Conv2D(filters=200, kernel_size=(10,1), kernel_r
    hybrid_cnn_lstm_model.add(MaxPooling2D(pool_size=(4,1), padding='same'))
    hybrid_cnn_lstm_model.add(BatchNormalization())
    hybrid_cnn_lstm_model.add(Dropout(0.5))

    # FC+LSTM layers
    hybrid_cnn_lstm_model.add(Flatten())
    hybrid_cnn_lstm_model.add(Dense((100)))
    hybrid_cnn_lstm_model.add(Reshape((100,1)))
    hybrid_cnn_lstm_model.add(CuDNNLSTM(25, return_sequences=False))
    hybrid_cnn_lstm_model.add(Dropout(0.5))

    hybrid_cnn_lstm_model.add(Dense(4, kernel_regularizer=L1L2(l1=0, l2=1e-3),

    # hybrid_cnn_lstm_model.summary()

    return hybrid_cnn_lstm_model
```

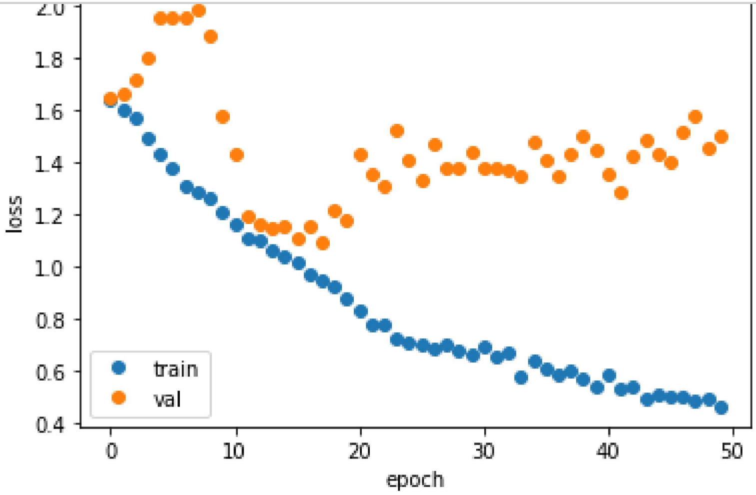
In [301]:
```
def plot_results(res):
    # Plotting accuracy trajectory
    plt.plot(res.history['accuracy'])
    plt.plot(res.history['val_accuracy'])
    plt.title('Hybrid CNN-LSTM model accuracy trajectory')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()

    # Plotting loss trajectory
    plt.plot(res.history['loss'],'o')
    plt.plot(res.history['val_loss'],'o')
    plt.title('Hybrid CNN-LSTM model loss trajectory')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='lower left')
    plt.show()

def run_model_all():
    model = cnn_lstm()

    learning_rate = 1e-3
    epochs = 50

    model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.A

    results = model.fit(x_train, y_train, epochs=epochs, batch_size=64, valida
    plot_results(results)

    train_score = model.evaluate(x_train, y_train)
    print('Train accuracy of the hybrid CNN-LSTM model:', train_score[1])

    test_score = model.evaluate(x_test, y_test)
    print('Test accuracy of the hybrid CNN-LSTM model:', test_score[1])

def run_model_s1(train_s1):
    if train_s1:
        x_train_cur = x_train[list(np.where(person_train==0)[0])]
        x_valid_cur = x_valid[list(np.where(person_valid==0)[0])]
        y_train_cur = y_train[list(np.where(person_train==0)[0])]
        y_valid_cur = y_valid[list(np.where(person_valid==0)[0])]
    else:
        x_train_cur, x_valid_cur = x_train, x_valid
        y_train_cur, y_valid_cur = y_train, y_valid
    x_test_cur = x_test[list(np.where(person_test==0)[0])]
    y_test_cur = y_test[list(np.where(person_test==0)[0])]

    model = cnn_lstm()

    learning_rate = 1e-3
    epochs = 50

    model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.A

    results = model.fit(x_train_cur, y_train_cur, epochs=epochs, batch_size=64
    plot_results(results)
```

```
    train_score = model.evaluate(x_train_cur, y_train_cur)
    print('Train accuracy of the hybrid CNN-LSTM model, subject 1:', train_sco

    test_score = model.evaluate(x_test_cur, y_test_cur)
    print('Test accuracy of the hybrid CNN-LSTM model, subject 1:', test_score

def run_model_over_time(time_period):
    x_train_cur = x_train[:, :time_period, :, :]
    x_valid_cur = x_valid[:, :time_period, :, :]
    x_test_cur = x_test[:, :time_period, :, :]
    y_train_cur, y_valid_cur, y_test_cur = y_train, y_valid, y_test

    model = cnn_lstm(time_period)

    learning_rate = 1e-3
    epochs = 50

    model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.

    results = model.fit(x_train_cur, y_train_cur, epochs=epochs, batch_size=64
    # plot_results(results)

    train_score = model.evaluate(x_train_cur, y_train_cur)
    print('Train accuracy of the hybrid CNN-LSTM model:', train_score[1])

    test_score = model.evaluate(x_test_cur, y_test_cur)
    print('Test accuracy of the hybrid CNN-LSTM model:', test_score[1])

    return train_score, test_score
```

Question 1

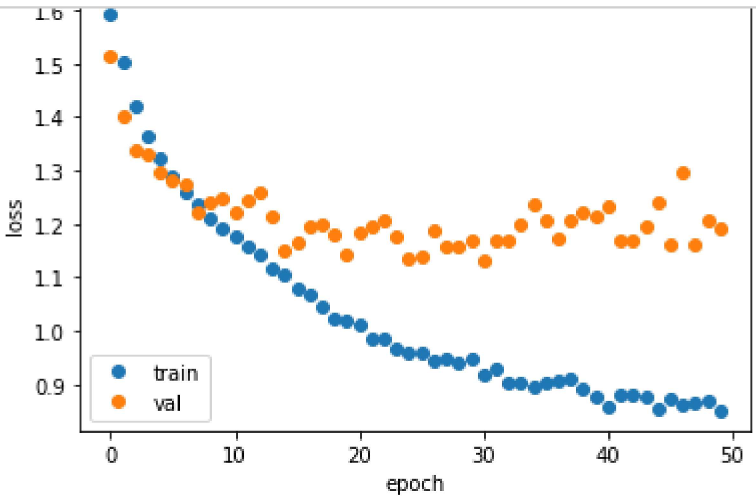
Optimize the classification accuracy for subject 1. Does it help to train across all subjects?

In [302]:
```
## Train/test on subject 1
run_model_s1(train_s1=True)
```




```
24/24 [==============================] - 0s 6ms/step - loss: 0.3692 - accur
acy: 0.9714
Train accuracy of the hybrid CNN-LSTM model, subject 1: 0.9713541865348816
7/7 [==============================] - 0s 17ms/step - loss: 1.8941 - accura
```

In [303]:
```
## Train on all, test on subject 1
run_model_s1(train_s1=False)
```

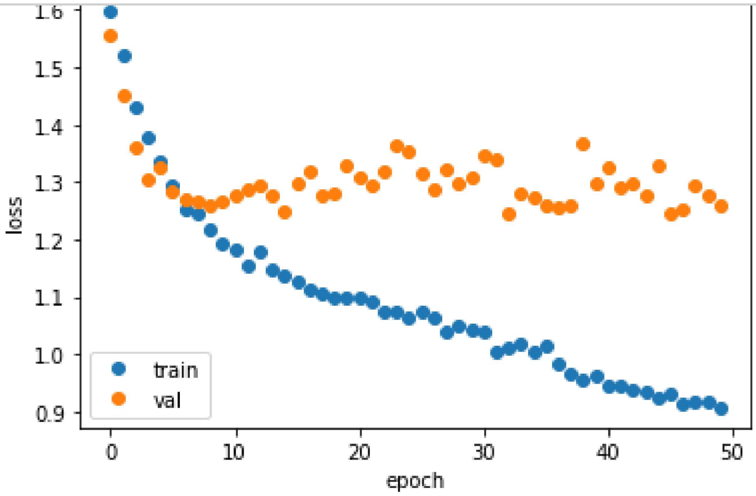



```
218/218 [==============================] - 2s 7ms/step - loss: 0.4338 - acc
uracy: 0.9899
Train accuracy of the hybrid CNN-LSTM model, subject 1: 0.9899425506591797
7/7 [==============================] - 0s 7ms/step - loss: 1.6879 - accurac
```

Question 2

Optimize the classification accuracy across all subjects. How does the classifier do? Do you notice any interesting trends?

In [304]:
```
## Train/test on all subjects
run_model_all()
```




```
218/218 [==============================] - 1s 6ms/step - loss: 0.4970 - acc
uracy: 0.9662
Train accuracy of the hybrid CNN-LSTM model: 0.9662356376647949
56/56 [==============================] - 0s 6ms/step - loss: 1.2765 - accur
```

Question 3

Evaluate the classification accuracy as a function of time (e.g., does it increase as you have data over longer periods of time? how much time is equired to get a reasonable classification accuracy?)

In [305]:
```
train_scores = []
test_scores = []

for t in range(25, 251, 25):
    print("Time period", t)
    train_score, test_score = run_model_over_time(time_period=t)
    train_scores.append(train_score[1])
    test_scores.append(test_score[1])
    print("=================================================================

print("Train accuracies:", train_scores)
print("Test accuracies:", test_scores)
print("Best accuracy: {}".format(max(test_scores)))
print("Best time period: {}".format(25 * (1 + np.argmax(test_scores))))

plt.plot(range(25, 251, 25), train_scores, label='train accuracies')
plt.plot(range(25, 251, 25), test_scores, label='test accuracies')
plt.title("Classification Accuracy over Time for CNN+LSTM")
plt.legend()
plt.show()
```

```
curacy: 0.7769 - val_loss: 1.2323 - val_accuracy: 0.6560
Epoch 48/50
109/109 [==============================] - 1s 11ms/step - loss: 0.9366 - ac
curacy: 0.7733 - val_loss: 1.1688 - val_accuracy: 0.6713
Epoch 49/50
109/109 [==============================] - 1s 13ms/step - loss: 0.9287 - ac
curacy: 0.7787 - val_loss: 1.2059 - val_accuracy: 0.6513
Epoch 50/50
109/109 [==============================] - 2s 17ms/step - loss: 0.9089 - ac
curacy: 0.7828 - val_loss: 1.2173 - val_accuracy: 0.6493
218/218 [==============================] - 2s 8ms/step - loss: 0.5279 - acc
uracy: 0.9532
Train accuracy of the hybrid CNN-LSTM model: 0.9531609416007996
56/56 [==============================] - 0s 7ms/step - loss: 1.2371 - accur
acy: 0.6586
Test accuracy of the hybrid CNN-LSTM model: 0.6585778594017029
========================================================================
Train accuracies: [0.6719827651977539, 0.820258617401123, 0.812212646007537
8, 0.8237069249153137, 0.915517270565033, 0.9298850297927856, 0.96637928485
87036, 0.9767241477966309, 0.9816091656684875, 0.9531609416007996]
```

In [305]: