nn

November 30, 2019

In [3]:
```
from keras.callbacks import ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn import metrics
from matplotlib import pyplot as plt
import pandas as pd
import pickle

data = None

with open("../pickles/preprocessed_data_nn.pkl","rb") as f:
    data = pickle.load(f)

X = data.drop('log_price', 1)
y = data['log_price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
Using TensorFlow backend.
```

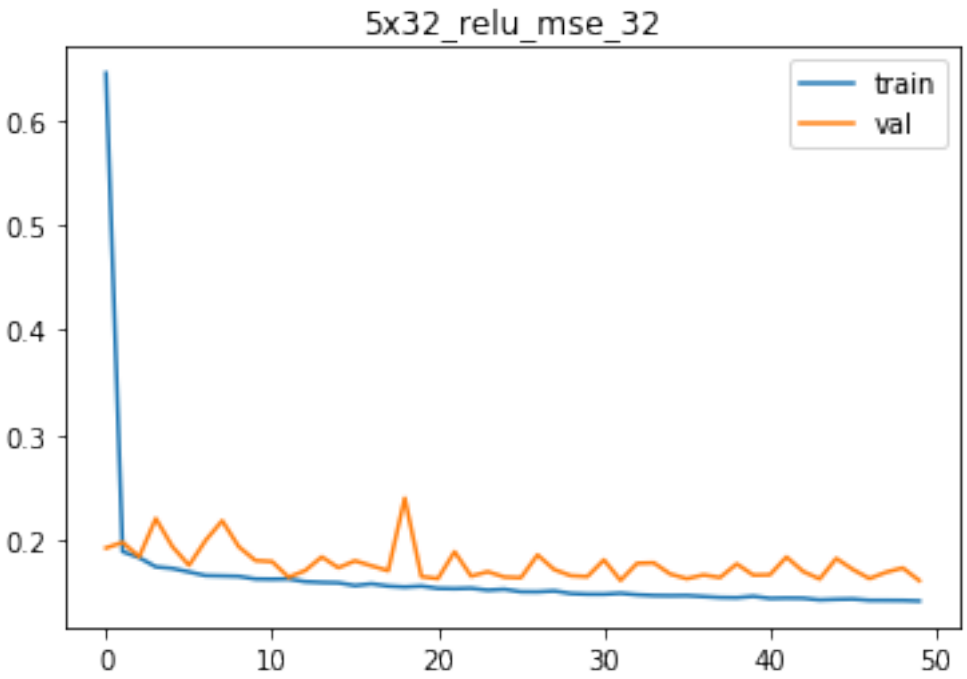
In [4]:
```
def build_NN(X_train, y_train, layer_width, layers, activation_fn, batch, loss_fn, epo
    NN_model = Sequential()

    # The Hidden Layers :
    NN_model.add(Dense(32, kernel_initializer='normal',input_dim = data.shape[1]-1, ac
    for i in range(layers):
        NN_model.add(Dense(layer_width, kernel_initializer='normal',activation=activat

    # The Output Layer :
    NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))

    # Compile the network :
    NN_model.compile(loss = loss_fn, optimizer='adam', metrics=['mse','mae'])
    #NN_model.summary()
```

```
            prefix = str(layers + 1) + 'x' + str(layer_width) + '_' + activation_fn + '_' + los
            checkpoint_name = 'checkpoints/'+prefix+'__{val_loss:.5f}.hdf5'  # Depth x width_l
            checkpoint = ModelCheckpoint(checkpoint_name, monitor='val_loss', save_best_only =
            callbacks_list = [checkpoint]

            hist = NN_model.fit(X_train, y_train, epochs=epoch, batch_size=batch, verbose = 0,
            return NN_model, hist, prefix
```

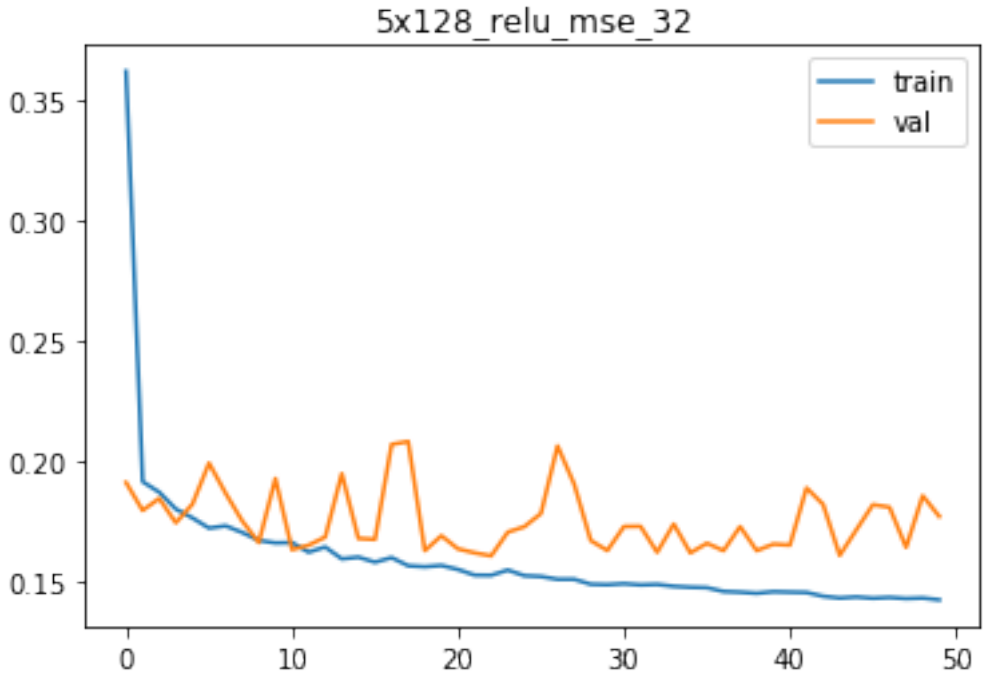
In [5]:
```
def get_mse(model, X_test, y_test):
    preds = NN_model.predict(X_test)
    return metrics.mean_squared_error(preds,y_test)
```

In [6]:
```
def show_plot(hist, name):
    # plot loss during training
    plt.title(name)
    plt.plot(hist.history['loss'], label='train')
    plt.plot(hist.history['val_loss'], label='val')
    plt.legend()
    plt.show()
```

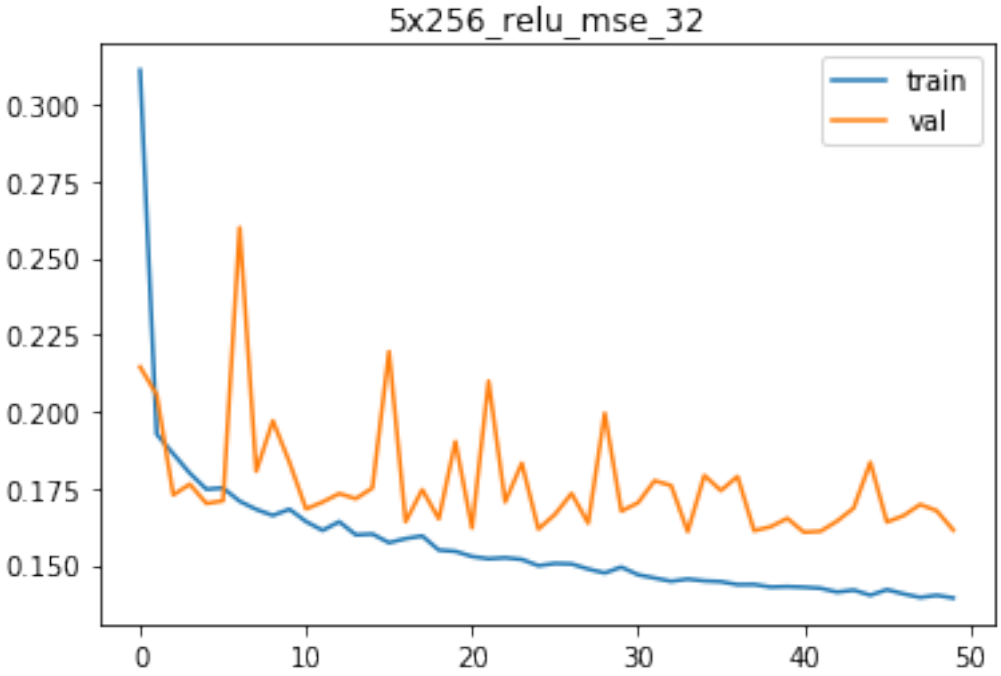
In [5]:
```
NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'relu', 32, 'mse') # width, d
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))
```



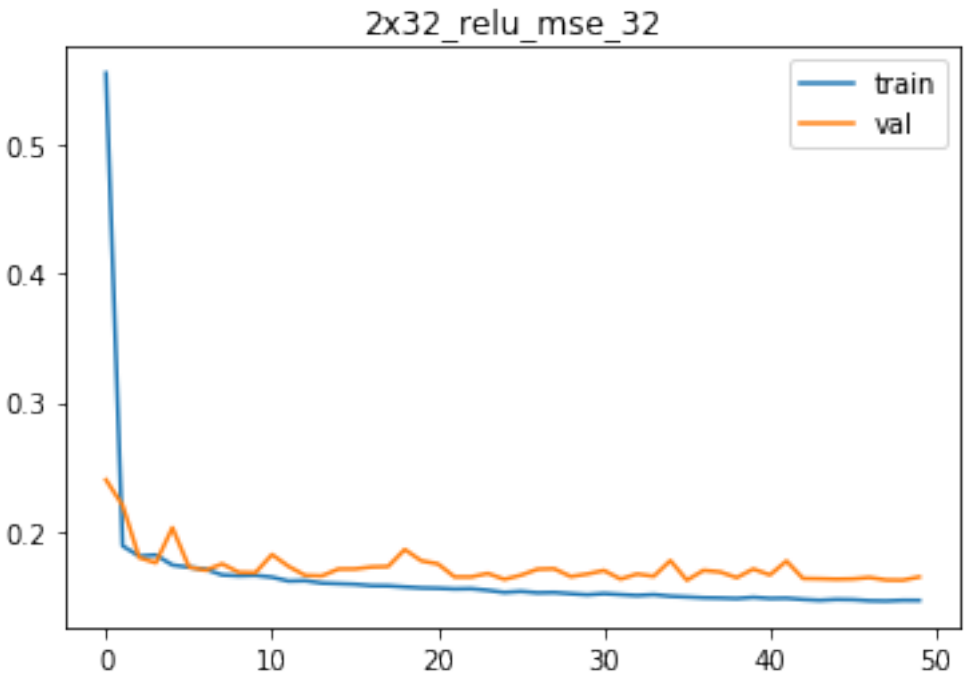

```
0.15417947076354424
```



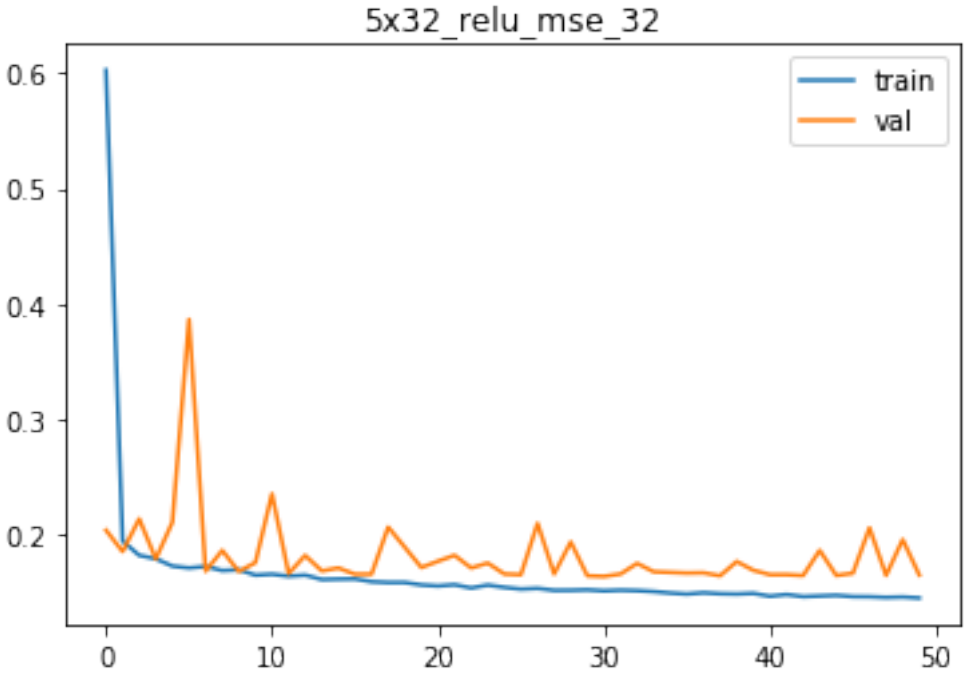

```
0.17042396764564222
```



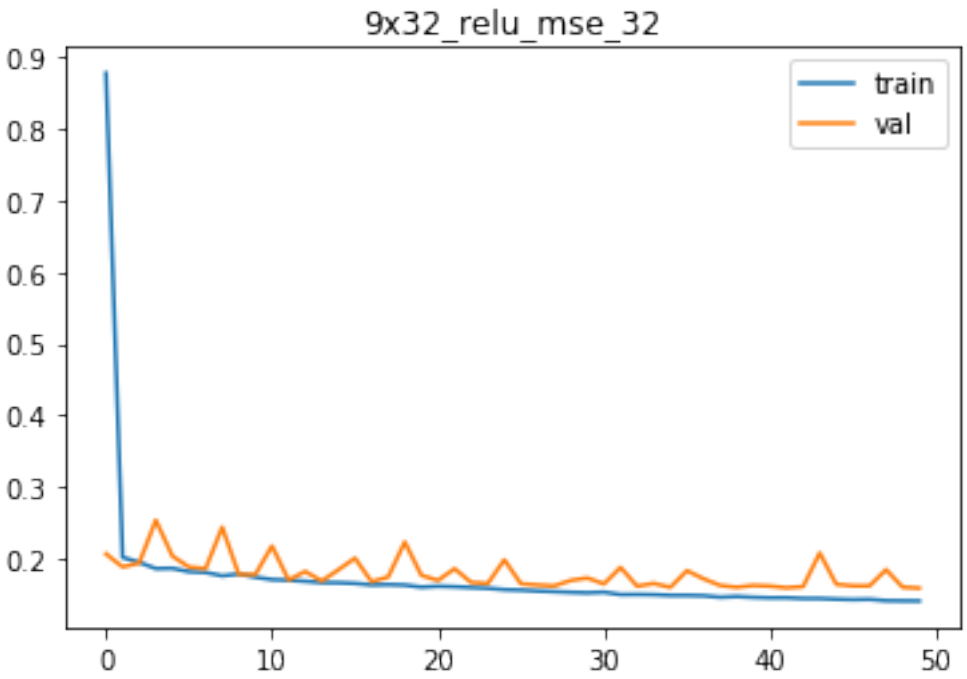
0.15479445607862985



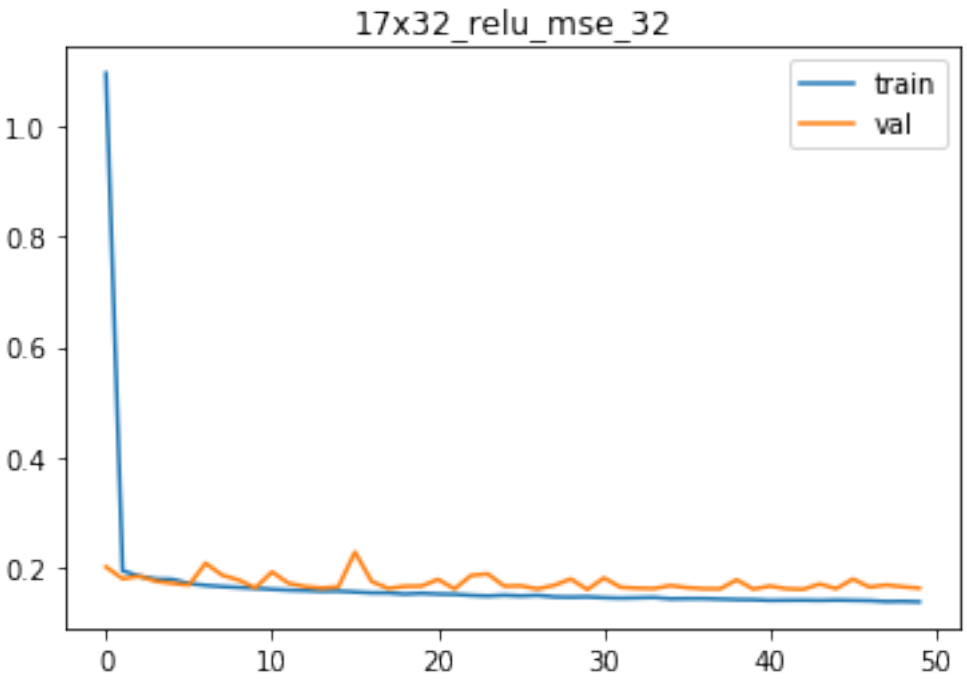
0.15829847452563972



0.15839866361927513

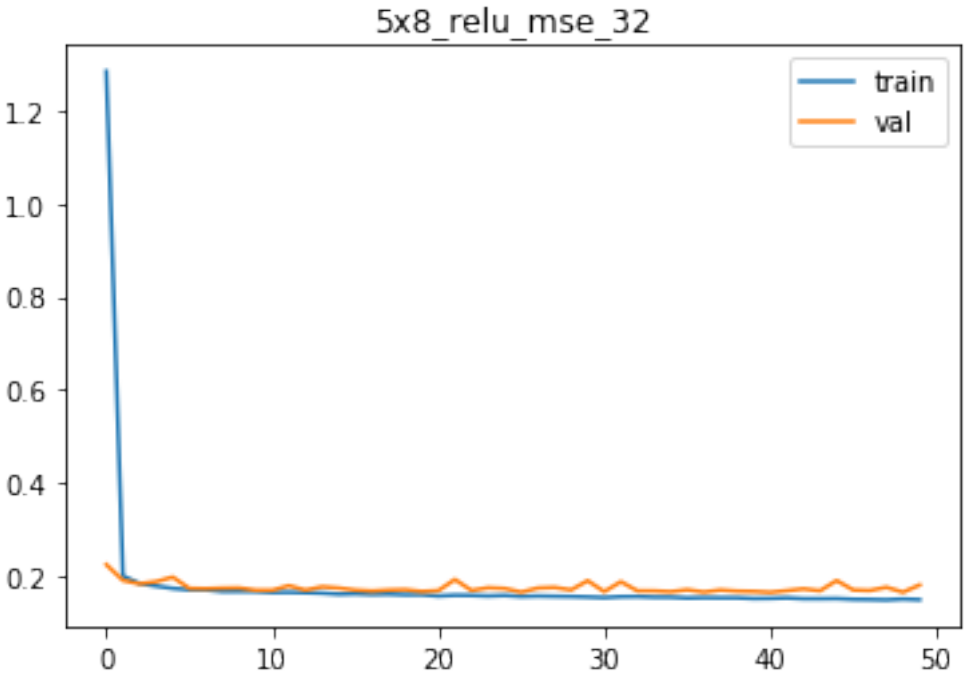


0.15302125004359599

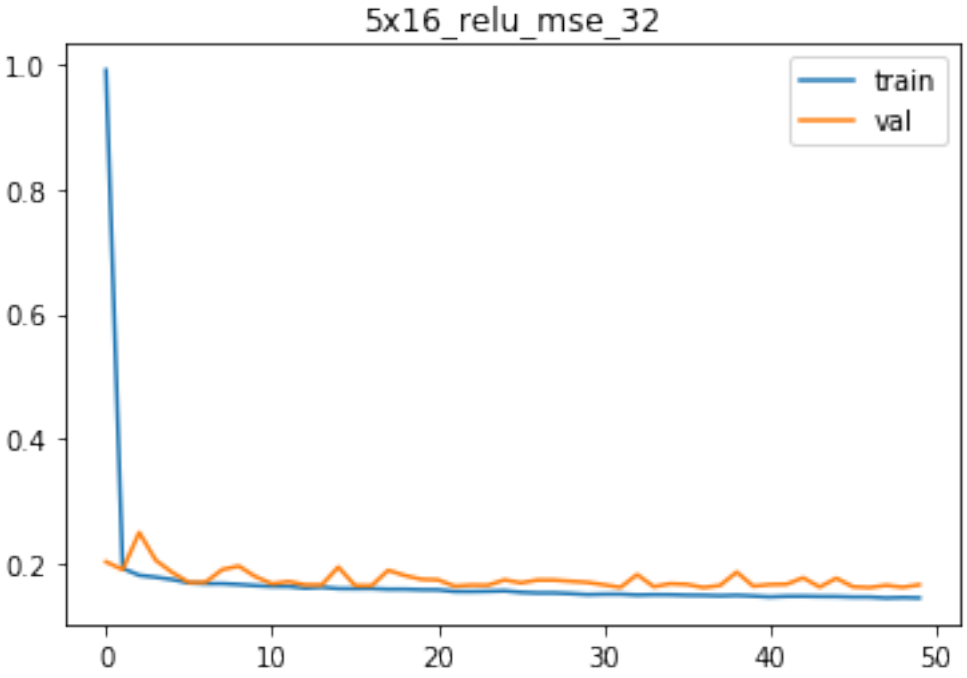



```
0.15822405492642272
```

In [6]:
```
# 32 width is optimal. 9 depth is optimal.

# Checking for smaller widths
NN_model, hist, name = build_NN(X_train, y_train, 8, 4, 'relu', 32, 'mse') # width, de
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))
```

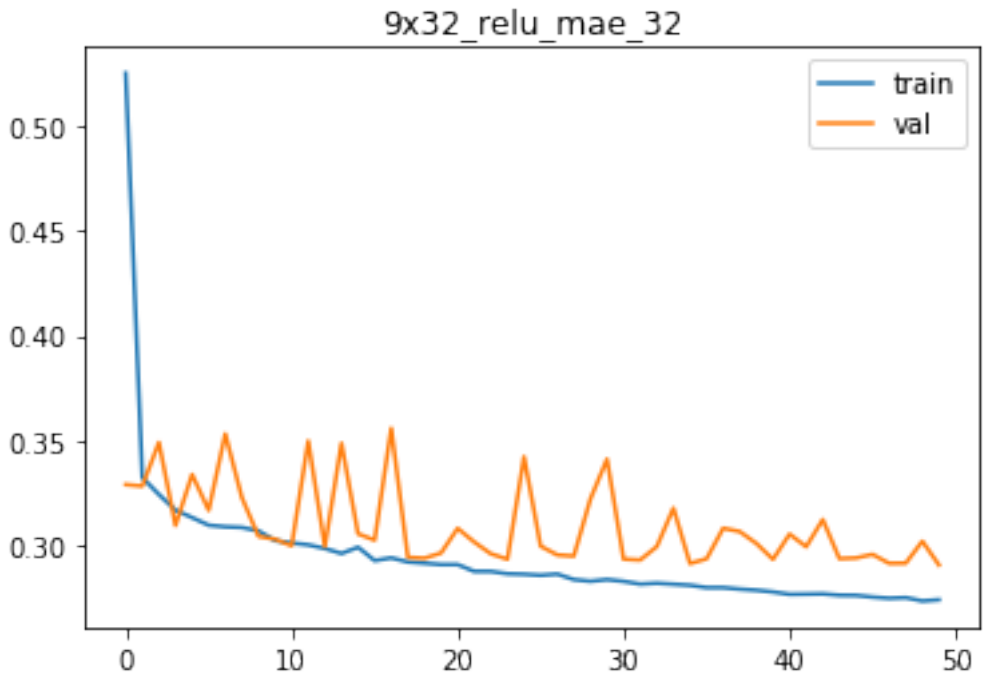



```
0.17307329965649632
```

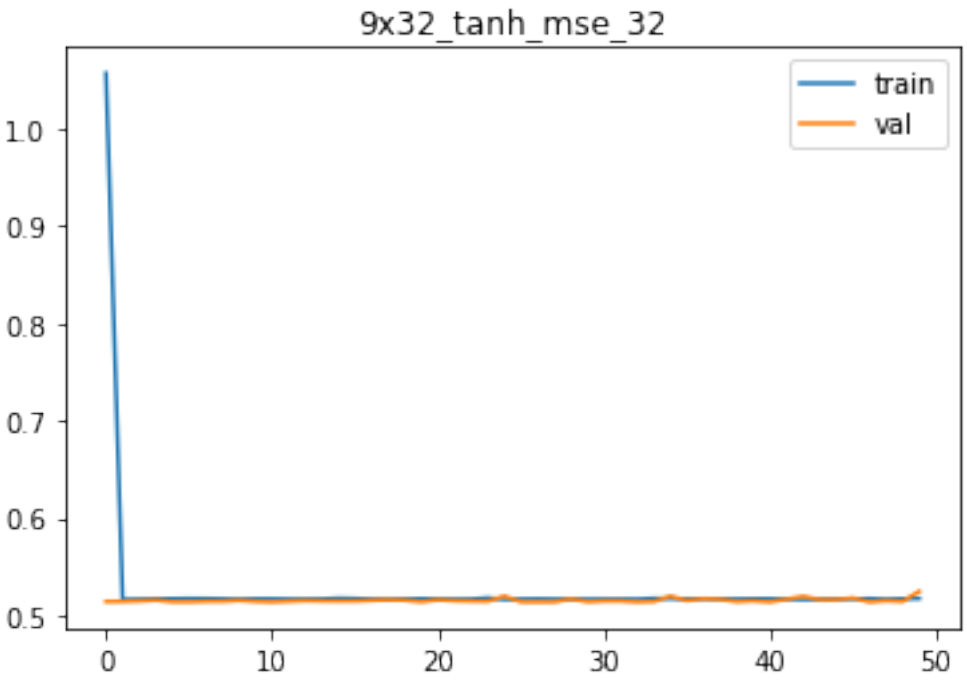


```
0.1594820421067284
```

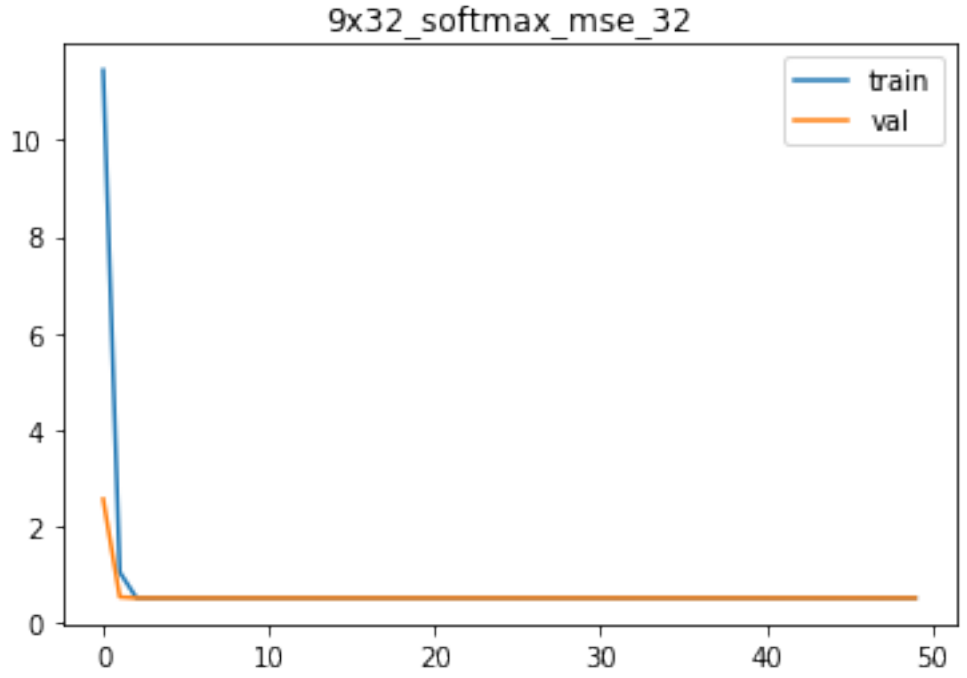
In [7]:
```
# 32 width is optimal.

# checking for other parameters
NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'relu', 32, 'mae') # width, d
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))

NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'tanh', 32, 'mse') # width, d
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))

NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'softmax', 32, 'mse') # width
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))
```




```
0.1581170385476567
```

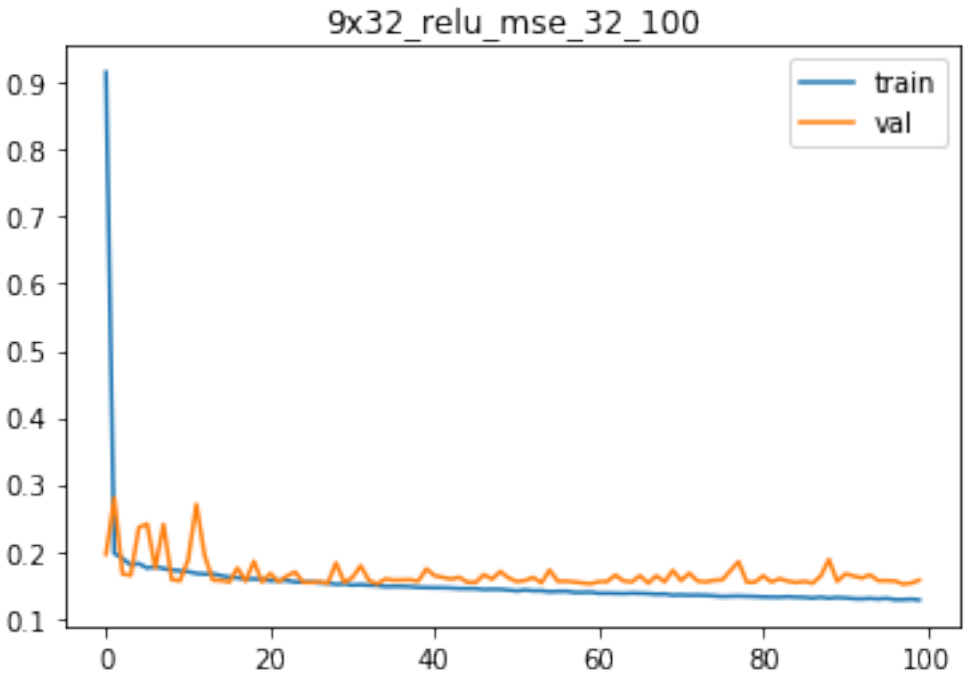



```
0.5238097607726891
```



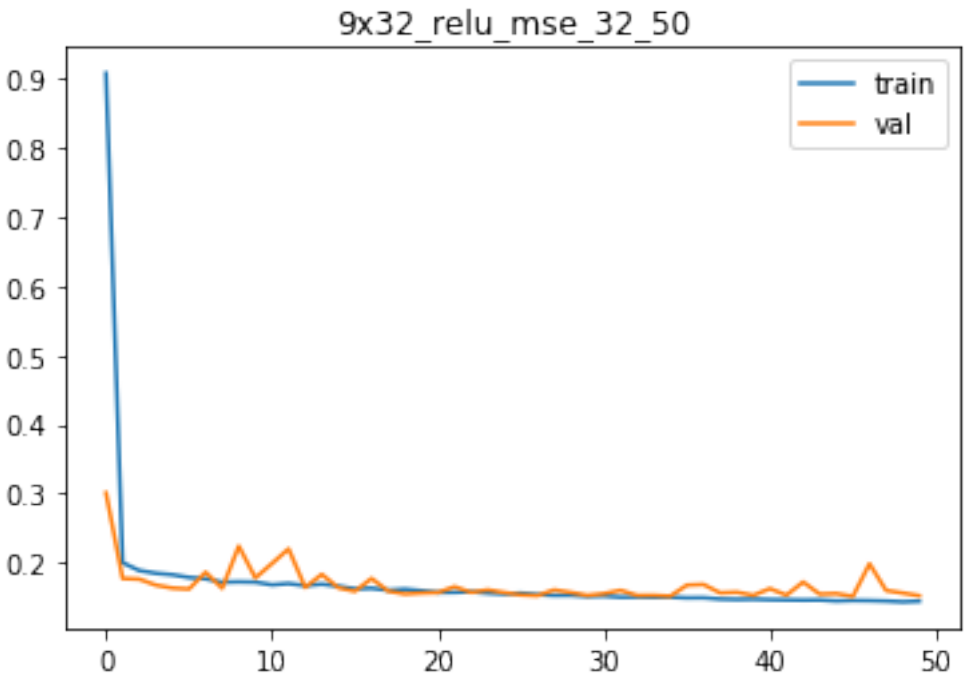
```
0.5158103607565511
```

In [8]:
```
# Relu + linear is best. There's no overfitting.

# Identifying best epochs
NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'relu', 32, 'mse', 100) # wid
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))
```



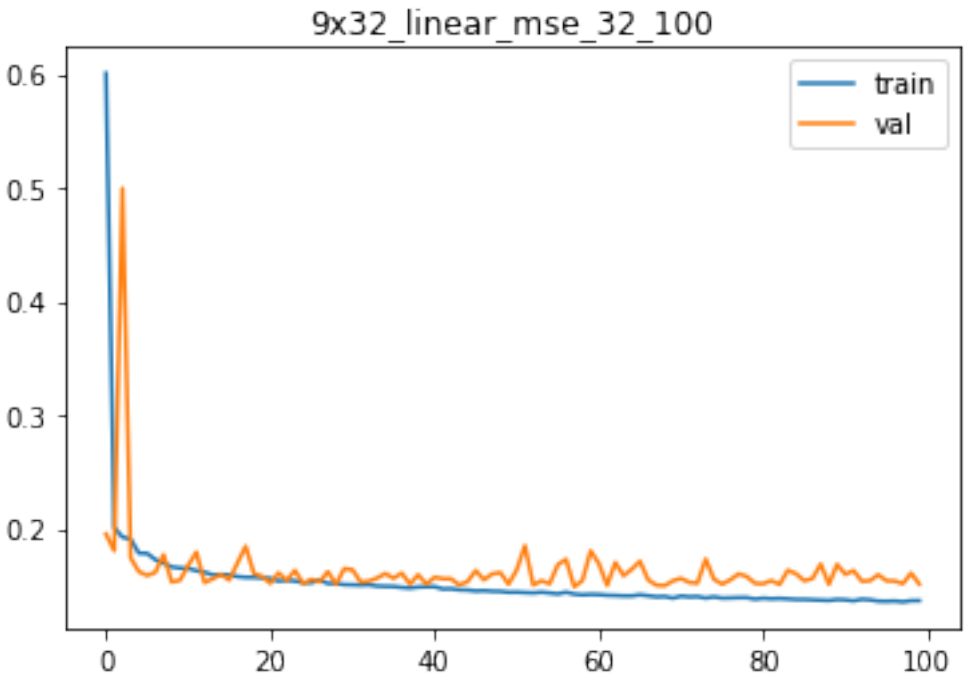

```
0.17154473146846
```

In [9]:
```
# 40 - 50 range seems to be the best epocs to avoid overfitting.

# Final model.
NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'relu', 32, 'mse', 50) # widt
show_plot(hist, name)
print(get_mse(NN_model, X_test, y_test))
preds = NN_model.predict(X_test)
# Print r score
print(metrics.r2_score(preds,y_test))
```



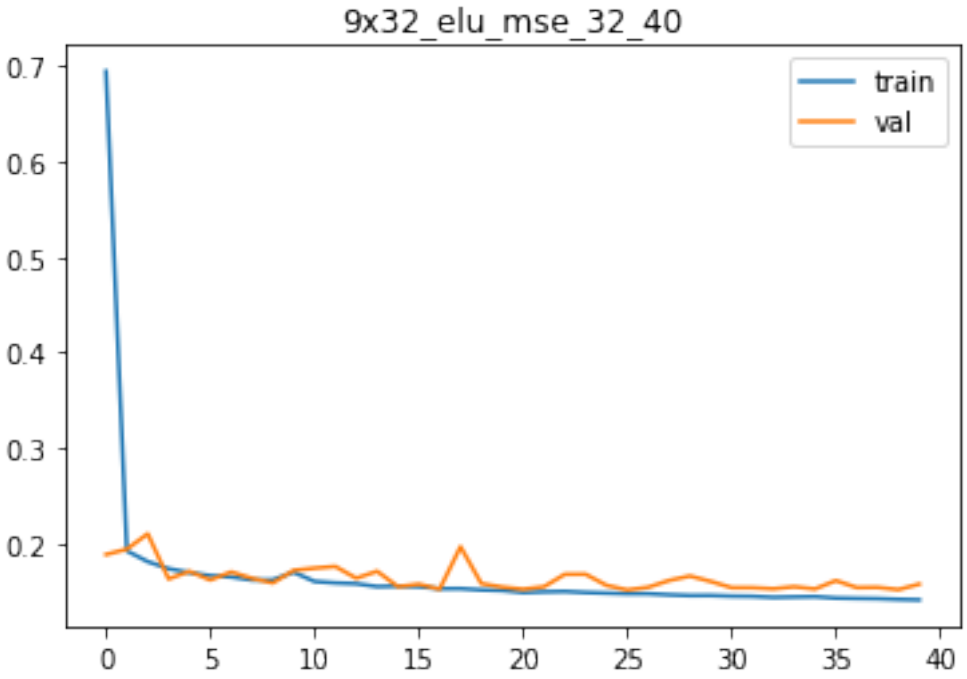

```
0.16519643511229784
0.5954635078887225
```

In [11]:
```
# Also trying with linea activation function

NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'linear', 32, 'mse', 100) #
show_plot(hist, name)
print("mse: ", get_mse(NN_model, X_test, y_test))
preds = NN_model.predict(X_test)
# Print r score
print("rscore: ",metrics.r2_score(preds,y_test))
```




```
0.1651841126215015
0.583996829733165
```

In [14]:
```
# Also trying with exponential linear units (elu) activation function

NN_model, hist, name = build_NN(X_train, y_train, 32, 8, 'elu', 32, 'mse', 40) # widt
show_plot(hist, name)
print("mse: ", get_mse(NN_model, X_test, y_test))
preds = NN_model.predict(X_test)
# Print r score
print("rscore: ",metrics.r2_score(preds,y_test))
```




```
mse:  0.16766955781076806
rscore:  0.5644491352951815
```

In [ ]: