Homework #6

Student Name: Sam Crane

Student ID: 801101091

GitHub: https://github.com/samofuture/Intro-to-ML

In [ ]:
```
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import torch
import torch.optim as optim
import torch.nn as nn
# use seaborn plotting defaults
import seaborn as sns; sns.set()
print(torch.cuda.device_count())
print(f"Version: {torch.__version__}, GPU: {torch.cuda.is_available()}, NUM_GPU: {t


device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
1
Version: 2.1.1, GPU: True, NUM_GPU: 1
```

Problem 1

In [ ]:
```
def prep_data() -> pd.DataFrame:
    df = pd.read_csv("Housing.csv")
    furnish_encoder = LabelEncoder()
    df['mainroad'] = df['mainroad'].apply(lambda x: 1 if x == 'yes' else 0)
    df['guestroom'] = df['guestroom'].apply(lambda x: 1 if x == 'yes' else 0)
    df['basement'] = df['basement'].apply(lambda x: 1 if x == 'yes' else 0)
    df['hotwaterheating'] = df['hotwaterheating'].apply(lambda x: 1 if x == 'yes' e
    df['airconditioning'] = df['airconditioning'].apply(lambda x: 1 if x == 'yes' e
    df['prefarea'] = df['prefarea'].apply(lambda x: 1 if x == 'yes' else 0)
    # df['furnishingstatus'] = df['furnishingstatus'].apply(lambda x: 2 if x == 'fu
    df['furnishingstatus'] = furnish_encoder.fit_transform(df['furnishingstatus'])

    return df
```

In [ ]:
```
df = prep_data()
price = df.pop('price').to_numpy()
df
```

Out[ ]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheati
0	7420	4	2	3	1	0	0	
1	8960	4	4	4	1	0	0	
2	9960	3	2	2	1	0	1	
3	7500	4	2	2	1	0	1	
4	7420	4	1	2	1	1	1	
...	...	...	...	...	...	...	...	
540	3000	2	1	1	1	0	1	
541	2400	3	1	1	0	0	0	
542	3620	2	1	1	1	0	0	
543	2910	3	1	1	0	0	0	
544	3850	3	1	2	1	0	0	

545 rows × 12 columns

In [ ]:

```
scaler_x = StandardScaler()
data = scaler_x.fit_transform(df)

scaler_y = StandardScaler()
price = scaler_y.fit_transform(price.reshape(-1, 1))

X_train, X_test, Y_train, Y_test = train_test_split(data, price, test_size=0.2, ran

train_inputs = torch.tensor(X_train).float()
train_outputs = torch.tensor(Y_train).float()
Y_test = torch.tensor(Y_test).float()
X_test = torch.tensor(X_test).float()
```

In [ ]:

```
def loss_fn(t_p, t_c):
    squared_diffs = (t_p - t_c)**2
    return squared_diffs.mean()
```

In [ ]:

```
def training_loop(num_epochs, optimizer, model, t_in, t_out, v_in, v_out):
    for epoch in range(1, num_epochs+1):
        t_p = model(t_in)
        train_loss = loss_fn(t_p, t_out)
        v_p = model(v_in)
        val_loss = loss_fn(v_p, v_out)

        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        if epoch % 500 == 0:
```

```
            print(f"Epoch {epoch}:")
            print(f"\tTraining Loss: {float(train_loss)}")
            print(f"\tValidation Loss: {float(val_loss)}")

    return model.parameters()
```

In [ ]:
```
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import KFold

def training_loop_with_cv(num_epochs, optimizer, model, loss_fn, inputs, targets, n
    kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

    for fold, (train_idx, val_idx) in enumerate(kf.split(inputs, targets)):
        t_in, v_in = inputs[train_idx], inputs[val_idx]
        t_out, v_out = targets[train_idx], targets[val_idx]

        print(f"Training Fold {fold + 1}/{num_folds}:")

        for epoch in range(1, num_epochs + 1):
            # Training
            model.train()
            t_p = model(t_in)
            train_loss = loss_fn(t_p, t_out)

            optimizer.zero_grad()
            train_loss.backward()
            optimizer.step()

            # Validation
            model.eval()
            v_p = model(v_in)
            val_loss = loss_fn(v_p, v_out)

            if epoch % 500 == 0:
                print(f"Epoch {epoch}:")
                print(f"\tTraining Loss: {float(train_loss)}")
                print(f"\tValidation Loss: {float(val_loss)}")

    return model
```

In [ ]:
```
housing_model = nn.Sequential(
                nn.Linear(len(df.columns), 32),
                nn.ReLU(),
                nn.Linear(32, 1))
optimizer = optim.SGD(housing_model.parameters(), lr=0.001)

training_loop(5000, optimizer, housing_model, train_inputs, train_outputs, X_test,
```

```
Epoch 500:
        Training Loss: 0.43353015184402466
        Validation Loss: 0.5522839426994324
Epoch 1000:
        Training Loss: 0.34394651651382446
        Validation Loss: 0.46478649973869324
Epoch 1500:
        Training Loss: 0.31984594464302063
        Validation Loss: 0.44427594542503357
Epoch 2000:
        Training Loss: 0.30723854899406433
        Validation Loss: 0.4347897171974182
Epoch 2500:
        Training Loss: 0.29896819591522217
        Validation Loss: 0.42940303683280945
Epoch 3000:
        Training Loss: 0.2929984927177429
        Validation Loss: 0.42578548192977905
Epoch 3500:
        Training Loss: 0.28828635811805725
        Validation Loss: 0.42303478717803955
Epoch 4000:
        Training Loss: 0.28439339995384216
        Validation Loss: 0.4209675192832947
Epoch 4500:
        Training Loss: 0.2810577154159546
        Validation Loss: 0.4191916286945343
Epoch 5000:
        Training Loss: 0.2779354453086853
        Validation Loss: 0.4173828959465027
```

Out[ ]:

```
<generator object Module.parameters at 0x0000023ED0791D20>
```

In [ ]:

```python
housing_model = nn.Sequential(
                nn.Linear(len(df.columns), 32),
                nn.ReLU(),
                nn.Linear(32, 1))
optimizer = optim.SGD(housing_model.parameters(), lr=0.001)

training_loop_with_cv(5000, optimizer, housing_model, loss_fn, train_inputs, train_
```

```
Training Fold 1/5:
Epoch 500:
        Training Loss: 0.4466761350631714
        Validation Loss: 0.44470709562301636
Epoch 1000:
        Training Loss: 0.34343641996383667
        Validation Loss: 0.33008602261543274
Epoch 1500:
        Training Loss: 0.31954967975616455
        Validation Loss: 0.3064955770969391
Epoch 2000:
        Training Loss: 0.3074979782104492
        Validation Loss: 0.2988772988319397
Epoch 2500:
        Training Loss: 0.2990087568759918
        Validation Loss: 0.29544419050216675
Epoch 3000:
        Training Loss: 0.2926625907421112
        Validation Loss: 0.2939257025718689
Epoch 3500:
        Training Loss: 0.2873786985874176
        Validation Loss: 0.2933778464794159
Epoch 4000:
        Training Loss: 0.28283005952835083
        Validation Loss: 0.2934132516384125
Epoch 4500:
        Training Loss: 0.2788689136505127
        Validation Loss: 0.2937256693840027
Epoch 5000:
        Training Loss: 0.27541428804397583
        Validation Loss: 0.29418814182281494
Training Fold 2/5:
Epoch 500:
        Training Loss: 0.2742505371570587
        Validation Loss: 0.28274160623550415
Epoch 1000:
        Training Loss: 0.27066999673843384
        Validation Loss: 0.28525418043136597
Epoch 1500:
        Training Loss: 0.26773667335510254
        Validation Loss: 0.2870347499847412
Epoch 2000:
        Training Loss: 0.2650340497493744
        Validation Loss: 0.2884601652622223
Epoch 2500:
        Training Loss: 0.2624594271183014
        Validation Loss: 0.28980302810668945
Epoch 3000:
        Training Loss: 0.259947270154953
        Validation Loss: 0.2909124493598938
Epoch 3500:
        Training Loss: 0.2574962079524994
        Validation Loss: 0.2919370234012604
Epoch 4000:
        Training Loss: 0.25511255860328674
        Validation Loss: 0.29308393597602844
```

```
Epoch 4500:
        Training Loss: 0.2527482509613037
        Validation Loss: 0.2942750155925751
Epoch 5000:
        Training Loss: 0.2504083514213562
        Validation Loss: 0.29550543427467346
Training Fold 3/5:
Epoch 500:
        Training Loss: 0.233327716588974
        Validation Loss: 0.35887157917022705
Epoch 1000:
        Training Loss: 0.22994662821292877
        Validation Loss: 0.36882466077804565
Epoch 1500:
        Training Loss: 0.22731490433216095
        Validation Loss: 0.3757067322731018
Epoch 2000:
        Training Loss: 0.22506628930568695
        Validation Loss: 0.3808686137199402
Epoch 2500:
        Training Loss: 0.22309818863868713
        Validation Loss: 0.38479289412498474
Epoch 3000:
        Training Loss: 0.2212824523448944
        Validation Loss: 0.38808673620224
Epoch 3500:
        Training Loss: 0.21962371468544006
        Validation Loss: 0.39074578881263733
Epoch 4000:
        Training Loss: 0.2180585414171219
        Validation Loss: 0.39309006929397583
Epoch 4500:
        Training Loss: 0.21660666167736053
        Validation Loss: 0.3947078585624695
Epoch 5000:
        Training Loss: 0.21524566411972046
        Validation Loss: 0.3964630961418152
Training Fold 4/5:
Epoch 500:
        Training Loss: 0.25040555000305176
        Validation Loss: 0.2273048311471939
Epoch 1000:
        Training Loss: 0.24664267897605896
        Validation Loss: 0.2334004044532776
Epoch 1500:
        Training Loss: 0.24386152625083923
        Validation Loss: 0.23786869645118713
Epoch 2000:
        Training Loss: 0.24140538275241852
        Validation Loss: 0.24157866835594177
Epoch 2500:
        Training Loss: 0.2392292469739914
        Validation Loss: 0.24478623270988464
Epoch 3000:
        Training Loss: 0.23725339770317078
        Validation Loss: 0.2477830946445465
```

```
Epoch 3500:
        Training Loss: 0.23546332120895386
        Validation Loss: 0.25030621886253357
Epoch 4000:
        Training Loss: 0.23370973765850067
        Validation Loss: 0.25262802839279175
Epoch 4500:
        Training Loss: 0.23196786642074585
        Validation Loss: 0.2549841105937958
Epoch 5000:
        Training Loss: 0.23034237325191498
        Validation Loss: 0.2569766938686371
Training Fold 5/5:
Epoch 500:
        Training Loss: 0.2502225339412689
        Validation Loss: 0.1641637533903122
Epoch 1000:
        Training Loss: 0.24779418110847473
        Validation Loss: 0.16780084371566772
Epoch 1500:
        Training Loss: 0.24593347311019897
        Validation Loss: 0.17067712545394897
Epoch 2000:
        Training Loss: 0.24424734711647034
        Validation Loss: 0.17314819991588593
Epoch 2500:
        Training Loss: 0.24266591668128967
        Validation Loss: 0.17531979084014893
Epoch 3000:
        Training Loss: 0.24117106199264526
        Validation Loss: 0.1774040013551712
Epoch 3500:
        Training Loss: 0.23971763253211975
        Validation Loss: 0.17939984798431396
Epoch 4000:
        Training Loss: 0.23827409744262695
        Validation Loss: 0.18126295506954193
Epoch 4500:
        Training Loss: 0.23689046502113342
        Validation Loss: 0.18294687569141388
Epoch 5000:
        Training Loss: 0.2355634868144989
        Validation Loss: 0.18456332385540009
```

Out[ ]:
```
Sequential(
  (0): Linear(in_features=12, out_features=32, bias=True)
  (1): ReLU()
  (2): Linear(in_features=32, out_features=1, bias=True)
)
```

The model complexity is orders of magnitude higher than linear regression. With each neuron in a neural network essentially being a linear regression on its own, this is a much more complex task. When we increased the number of layers for part B, the complexity only increased.

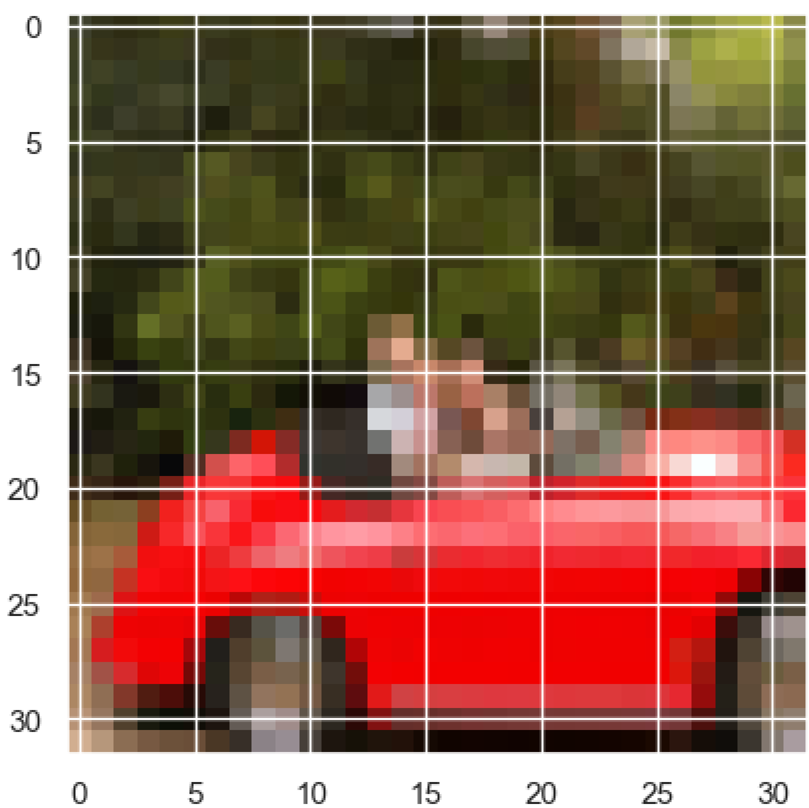
Compared to HW5, the first network performed worse than the linear regression. However, the second model was performing just as well, if not better than HW5.

Problem 2

In [ ]:
```
from torchvision import datasets, transforms
data_path = 'data-unversioned/p1ch7'
cifar10 = datasets.CIFAR10(data_path, train=True, download=True)
cifar10_val = datasets.CIFAR10(data_path, train=False, download=True)
```

```
c:\Users\Sam\anaconda3\Lib\site-packages\torchvision\io\image.py:13: UserWarning: Fa
iled to load image Python extension: 'Could not find module 'C:\Users\Sam\anaconda3
\Lib\site-packages\torchvision\image.pyd' (or one of its dependencies). Try using th
e full path with constructor syntax.'If you don't plan on using image functionality
from `torchvision.io`, you can ignore this warning. Otherwise, there might be someth
ing wrong with your environment. Did you have `libjpeg` or `libpng` installed before
building `torchvision` from source?
  warn(
Files already downloaded and verified
Files already downloaded and verified
```

In [ ]:
```
img, label = cifar10[99]
plt.imshow(img)
plt.show()
```



In [ ]:
```
to_tensor = transforms.ToTensor()

cifar10_tensor = datasets.CIFAR10(data_path, train=True, download=False, transform=
```

```
imgs = torch.stack([img_t for img_t, _ in cifar10_tensor], dim=3)
mean = imgs.view(3, -1).mean(dim=1)
std = imgs.view(3, -1).std(dim=1)

# print("M", type(mean))
# print("S", std)
# transforms.Normalize(mean, std)
```

In [ ]:
```
cifar10 = datasets.CIFAR10(
    data_path, train=True, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]))

# cifar10 = [(img, label) for img, label in cifar10]

cifar10_val = datasets.CIFAR10(
    data_path, train=False, download=False,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]))

# cifar10_val = [(img, label) for img, label in cifar10_val]
```

In [ ]:
```
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']

n_out = 10

cifar_model = nn.Sequential(
                    nn.Linear(3072, 512),
                    nn.ReLU(),
                    nn.Linear(512, n_out),
                    nn.Softmax(dim=1)
                )

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(cifar_model.parameters(), lr = 0.01)
num_epochs = 50
```

In [ ]:
```
cifar_model.to(device)
loss_fn.to(device)

for epoch in range(num_epochs):
    for img, label in cifar10:
        img = torch.Tensor(img)
        img = img.view(-1).unsqueeze(0)
        label = torch.tensor([label])
        img = img.to(device)
        out = cifar_model(img)
        loss = loss_fn(out, label.to(device))
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print("Epoch: %d, Loss: %f" % (epoch, float(loss)))
```

```
Epoch: 0, Loss: 2.461141
Epoch: 1, Loss: 1.536038
Epoch: 2, Loss: 2.454097
Epoch: 3, Loss: 2.461150
Epoch: 4, Loss: 2.461084
Epoch: 5, Loss: 2.461150
Epoch: 6, Loss: 2.461150
Epoch: 7, Loss: 2.461150
Epoch: 8, Loss: 2.461150
Epoch: 9, Loss: 2.461150
Epoch: 10, Loss: 2.461150
Epoch: 11, Loss: 2.461150
Epoch: 12, Loss: 2.461150
Epoch: 13, Loss: 2.456374
Epoch: 14, Loss: 2.461150
Epoch: 15, Loss: 2.461150
Epoch: 16, Loss: 2.461150
Epoch: 17, Loss: 2.460614
Epoch: 18, Loss: 2.461150
Epoch: 19, Loss: 2.461150
Epoch: 20, Loss: 2.461150
Epoch: 21, Loss: 2.461150
Epoch: 22, Loss: 2.461150
Epoch: 23, Loss: 2.461150
Epoch: 24, Loss: 2.461150
Epoch: 25, Loss: 2.460831
Epoch: 26, Loss: 2.461150
Epoch: 27, Loss: 2.461150
Epoch: 28, Loss: 2.461150
Epoch: 29, Loss: 2.461150
Epoch: 30, Loss: 2.461150
Epoch: 31, Loss: 2.461150
Epoch: 32, Loss: 2.461150
Epoch: 33, Loss: 2.461150
Epoch: 34, Loss: 2.461150
Epoch: 35, Loss: 2.461150
Epoch: 36, Loss: 2.461150
Epoch: 37, Loss: 2.461150
Epoch: 38, Loss: 2.461150
Epoch: 39, Loss: 2.461150
Epoch: 40, Loss: 2.461150
Epoch: 41, Loss: 2.461150
Epoch: 42, Loss: 2.461150
Epoch: 43, Loss: 2.461150
Epoch: 44, Loss: 2.461150
Epoch: 45, Loss: 2.461150
Epoch: 46, Loss: 2.461150
Epoch: 47, Loss: 2.461150
Epoch: 48, Loss: 2.461150
Epoch: 49, Loss: 2.461150
```

In [ ]:
```
train_loader = torch.utils.data.DataLoader(cifar10_val, batch_size=64, shuffle=Fals

correct = 0
total = 0
cifar_model.to(device)
loss_fn.to(device)
with torch.no_grad():
    for imgs, labels in train_loader:
        imgs = torch.Tensor(imgs).to(device)

        labels = torch.tensor(labels).to(device)

        outputs = cifar_model(imgs.view(imgs.shape[0], -1))
        _, predicted = torch.max(outputs, dim=1)
        total += labels.shape[0]
        correct += int((predicted == labels).sum())

print("Accuracy: %f" % (correct / total))
```

```
C:\Users\Sam\AppData\Local\Temp\ipykernel_35472\3286169161.py:11: UserWarning: To co
py construct from a tensor, it is recommended to use sourceTensor.clone().detach() o
r sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourc
eTensor).
  labels = torch.tensor(labels).to(device)
Accuracy: 0.365300
```

In [ ]:
```
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 8, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(8 * 8 * 8, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
        out = out.view(-1, 8 * 8 * 8)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [ ]:
```
model = nn.Sequential(
                    nn.Linear(3072, 512),
                    nn.ReLU(),
                    nn.Linear(512, 256),
                    nn.ReLU(),
                    nn.Linear(256, 128),
                    nn.ReLU(),
                    nn.Linear(128, n_out),
                    nn.Softmax(dim=1)
                )
```

In [ ]:
```python
import datetime  # <1>

def training_loop(n_epochs, optimizer, model, loss_fn, train_loader):
    for epoch in range(1, n_epochs + 1):  # <2>
        loss_train = 0.0
        for imgs, labels in train_loader:  # <3>

            outputs = model(imgs)  # <4>

            loss = loss_fn(outputs, labels)  # <5>

            optimizer.zero_grad()  # <6>

            loss.backward()  # <7>

            optimizer.step()  # <8>

            loss_train += loss.item()  # <9>

        if epoch == 1 or epoch % 10 == 0:
            print('{} Epoch {}, Training loss {}'.format(
                datetime.datetime.now(), epoch,
                loss_train / len(train_loader)))
```

In [ ]:
```python
train_loader = torch.utils.data.DataLoader(cifar10, batch_size=64, shuffle=True)

model = Net()  #  <2>
optimizer = optim.SGD(model.parameters(), lr=1e-2)  #  <3>
loss_fn = nn.CrossEntropyLoss()  #  <4>

training_loop(  # <5>
    n_epochs = 100,
    optimizer = optimizer,
    model = model,
    loss_fn = loss_fn,
    train_loader = train_loader,
)
```

```
2023-12-02 18:39:31.725474 Epoch 1, Training loss 2.0475708695933643
2023-12-02 18:42:43.357330 Epoch 10, Training loss 1.1856087433255238
2023-12-02 18:46:17.277680 Epoch 20, Training loss 1.0059958333554475
2023-12-02 18:49:52.609951 Epoch 30, Training loss 0.9336449801921844
2023-12-02 18:53:28.861767 Epoch 40, Training loss 0.8844456407420166
2023-12-02 18:57:01.941340 Epoch 50, Training loss 0.8441681743354139
2023-12-02 19:00:42.824281 Epoch 60, Training loss 0.8128206913199876
2023-12-02 19:04:21.251485 Epoch 70, Training loss 0.7862183522156743
2023-12-02 19:07:59.140729 Epoch 80, Training loss 0.7612364073391156
2023-12-02 19:11:38.193575 Epoch 90, Training loss 0.7391753197478517
2023-12-02 19:15:18.577279 Epoch 100, Training loss 0.7216365825565879
```

In [ ]:
```python
def validate(model, loader):
    correct = 0
    total = 0

    with torch.no_grad():  # <1>
```

```
        for imgs, labels in loader:
            outputs = model(imgs)
            _, predicted = torch.max(outputs, dim=1) # <2>
            total += labels.shape[0]  # <3>
            correct += int((predicted == labels).sum())  # <4>

    print("Accuracy: {:.2f}".format(correct / total))

val_loader = torch.utils.data.DataLoader(cifar10_val, batch_size=64, shuffle=False)
validate(model, val_loader)
```

```
Accuracy: 0.65
```

Part A

Training Time: 2 Hours Training Loss: 2.5144 Evaluation Accuracy: 0.36

Part B

Training Time: 3 Hours Training Loss: 0.72 Evaluation Accuracy: 0.65

I don't think the model trained long enough to achieve overfitting. The complexity of the network adds 128 + 256 = 384 more neurons to the already existing 522, making it more than 50% more complex.