CM3015 MLNN - Final

IMDB movie dataset

1. Defining the problem and assembling a dataset

What is the input data?

The selected input data for this machine learning problem will be the IMDB movieset from keras.datasets. The dataset contains 50,000 reviews split into train and test batches. Each batch contains 25,000 reviews. The reviews are labeled as positive or negative in a ratio of 50/50. The reviews have also been preprocessed with each review encoded as a list of word indexes.

For the collection of the data method of the data from the IMDB website, each movie will have a maximum of 30 reviews. The negative labeled reviews have a score of <= 4/10, and the positive reviews are scored >= 7/10. This helps to ensures that the reviews that are more neutral are not included.

For the purpose of this project, the data (train_data, test_data) will be transformed from an array of lists representing a list of the index of words ranked in the overall frequency of the dataset (already-tokenized bag of words (BoW)), which was provided by the owners of this dataset.

The label (train_labels, test_labels) is either 0 or 1, where 0 is negative and 1 is positive.

What type of problem?

This type of problem is a binary classification problem. There would only be two possible outputs and that would either be 0 or 1, where 0 stands for negative and 1 stands for positive.

What are you hoping to predict?

Given a new or unseen movie review (according to the model), the aim is to predict if the movie reviews as positive or negative to a high level of accuracy.

The unseen movie reviews will be the separated test set, which contains 25,000 movie reviews with their correct labels. After finding the best performing model, we will evaluate with the test set to see the performance.

---

2. Choosing a measure of success

Accuracy will be the main measure of success for the model. The reason is because in this binary classification we can only have 2 outcomes either positive or negative label. Accuracy measures how often the classifier predicts the correct label. The reference for the baseline model will be based on accuracy score.

A confusion matrix for step 8 (testing) will be created and plotted, which will also let us visualize the performance of the model and provide us with more metrics such as precision, recall, and f1-score. These metrics will allow us to further example how the model performs in the context of predicting the labels.

3.Deciding on an evaluation protocol

The evaluation method used is the hold-out validation. Before we train the model, we will take the training data and training labels to split into a partial train and validation train set. The validation set will include 10,000 samples, and the partial train will include the remaining 15,000 samples.

This method will allow us to train the model with the partial train data, and monitor the performance (loss and accuracy) of the model by running a validation set after training. This will allow us to fine tune the parameters before fully testing it on the unseen test set.

The advantage of this is that we can get a preliminary view into how the model will perform on unseen data. Since the dataset is quite large with enough samples (25,000 for train, 25,000 for test), the hold-out validation method is a good choice.

---

4. Preparing your data

Import libraries and load dataset

Import libraries and load dataset

In [1]:
```
#import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import seaborn as sns

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics
from keras import regularizers
```

In [2]:
```
# load dataset and limit number of words to 10000
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

Manipulate into tensors and normalize data

We can see that the data from the "keras.datasets import imdb" is organized into an array of lists with the shape of 25,000 items. Each list contains the number corresponding to the already-tokenized bag of words (BoW) features.

In [3]:
```
train_data[0:2]
```

Out[3]:
```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43,
838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546,
38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38,
76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16,
480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77,
52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400,
317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22,
21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16
, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394,
20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300,
1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952,
46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7
, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2
, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837,
131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16
, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95])],
      dtype=object)
```

In [4]:
```
train_data.shape
```

Out[4]:
```
(25000,)
```

Here we will use the vectorize_sequences function to encode the train and test data into a binary matrix in the size of 25,000 by 10,000. Then the training and test data will be vectorized by passing through the function that creates an all-zero matrix of the specified shape while setting specific indices to 1. This also effectively normalizes the train and test data at the same time.

The corresponding labels will also be vectorized with the np.asarray and .astype('float32').

In [5]:
```
# encoding the interger into binary matrix
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

# vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
x_train.shape
```

Out[5]:
```
(25000, 10000)
```

The end result after encoding is a 25,000 by 10,000 tensor for each train and test set. This represents the 25,000 movie reviews by the 10,000 possible words.

The input data is now prepared and in the form of vectors. The labels are scalers (1 or 0). Everything is now ready for the next step.

---

5. Developing a model that does better than a baseline

3. Developing a model that does better than a baseline

Common-sense baseline:

For this dataset, the positive and negative review labels are at a ratio of 50/50 each. So, the baseline common sense would be to beat 50% and we will aim to create a model that achieve slightly over that.

The smallest model that beats common-sense baseline

We will create a model with 2 sequential hidden layers, each with relu activation functions. The first layer will have 4 hidden units and the second will have 2 hidden units.

For loss function, we will select the binary_crossentropy. rmsorop will be the optimizer and accuracy will be observed during training. All these parameters are recommended in DLWP Chapter 3.4.

We will also set aside the validation set with 10,000 samples from the train set, leaving 15,000 samples as the partial train set.

In [6]:

```
# split validation set
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

# define model
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(2, activation='relu'))

# compile model
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 40004

 dense_1 (Dense)             (None, 2)                 10

=================================================================
Total params: 40014 (156.30 KB)
Trainable params: 40014 (156.30 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [7]:

```
#train mode and evaluate with test set
history = model.fit(partial_x_train,
    partial_y_train,
    epochs=5,
    batch_size=512,
    validation_data=(x_val, y_val), verbose=0)
#get results of model with test data
results = model.evaluate(x_test, y_test)
model.reset_states()
```

```
782/782 [==============================] - 1s 2ms/step - loss: 0.4515 - accuracy: 0.5662
```

Results of the baseline model:

We can see that the reported numbers after training and evaluating returns 2 metrics:

- loss: 0.4515
- acc: 0.5662

So far, the accuracy is slightly above our baseline of 50%. And that should be a good starting point for expanding the model.

In [8]:

```
#get history
history_dict = history.history
history_dict.keys()
```

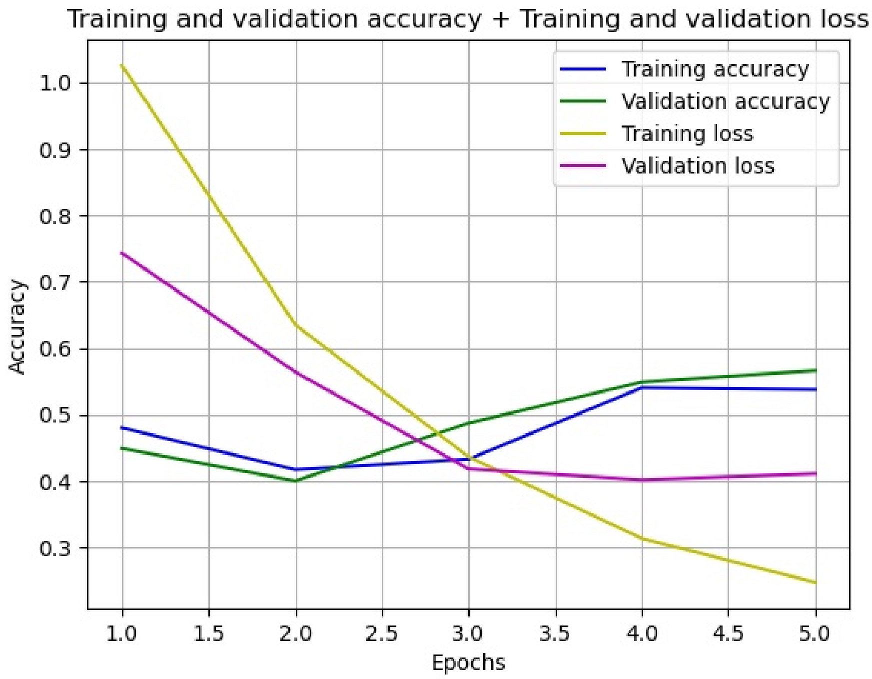
Out[8]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [9]:

```
# Let's plot training and validation accuracy as well as loss.
def plot_history(history):
    accuracy = history.history['accuracy']
    val_accuracy = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1,len(accuracy) + 1)
```

```
    # Plot accuracy
    plt.figure(1)
    plt.plot(epochs, accuracy, 'b', label='Training accuracy')
    plt.plot(epochs, val_accuracy, 'g', label='Validation accuracy')
    plt.plot(epochs, loss, 'y', label='Training loss')
    plt.plot(epochs, val_loss, 'm', label='Validation loss')
    plt.title('Training and validation accuracy + Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.grid()
    plt.legend()
plot_history(history)
```



Training data interpretation - accuracy and loss

From the plot, we can see that the training and validation accuracy starts at below 0.5, then rises slightly above the baseline and then settles in at around 0.55. This is similar to the result when we run model.evaluate to test the model on the test set.

Loss for both training and validation is steadily decreasing but hints at a divergence at around 3 epochs.

Overall, the performance is not great, but able to meet the requirements of a small baseline model.

---

6. Scaling up: developing a model that overfits

We will start by creating a function to run multiple models sequentially with different parameters. This will speed up the process of testing and interpreting the results. Overall, this allows us to test different batches of configurations with less lines of code as it will be modular.

3 layer model with differing hidden units per layer

We will start off with a 3 layer model consisting of 2 relu layers and ending in 1 sigmoid layer. We will first explore the effect of network size in relation to the hidden units per layer starting with 16 units and ending in 256 units.

In [10]:
```
# define function that can run sequential models back to back
def build_model(layer_1_units, layer_2_units, layer_3_units):
    model = models.Sequential()
    model.add(layers.Dense(layer_1_units, activation='relu', input_shape=(10000,)))
    model.add(layers.Dense(layer_2_units, activation='relu'))
    model.add(layers.Dense(layer_3_units, activation='sigmoid'))
    model.compile(optimizer='rmsprop',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        c = ['\b|', '\b/', '\b-', '\b\\']
        print(c[epoch % 4], end='')
    def on_epoch_end(self, epoch, logs=None):
        print('\b', end='')
```

In [11]:
```
histories = {}
#can input number of hidden units for each layer. Here we will test an increasing large number of hidden units
for i in [16, 32, 64, 128, 256]:
```

```
    model = build_model(i, i, 1)
    model_name = str(i) + '-' + str(i) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
    model.reset_states()
```

```
Training 16-16-1
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_2 (Dense)             (None, 16)                160016    
                                                                 
 dense_3 (Dense)             (None, 16)                272       
                                                                 
 dense_4 (Dense)             (None, 1)                 17        
                                                                 
=================================================================
Total params: 160305 (626.19 KB)
Trainable params: 160305 (626.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-32-1
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_5 (Dense)             (None, 32)                320032    
                                                                 
 dense_6 (Dense)             (None, 32)                1056      
                                                                 
 dense_7 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 321121 (1.22 MB)
Trainable params: 321121 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 64-64-1
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_8 (Dense)             (None, 64)                640064    
                                                                 
 dense_9 (Dense)             (None, 64)                4160      
                                                                 
 dense_10 (Dense)            (None, 1)                 65        
                                                                 
=================================================================
Total params: 644289 (2.46 MB)
Trainable params: 644289 (2.46 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 128-128-1
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_11 (Dense)            (None, 128)               1280128   
                                                                 
 dense_12 (Dense)            (None, 128)               16512     
                                                                 
 dense_13 (Dense)            (None, 1)                 129       
                                                                 
=================================================================
Total params: 1296769 (4.95 MB)
Trainable params: 1296769 (4.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 256-256-1
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_14 (Dense)            (None, 256)               2560256   
                                                                 
 dense_15 (Dense)            (None, 256)               65792     
                                                                 
 dense_16 (Dense)            (None, 1)                 257       
                                                                 
=================================================================
Total params: 2626305 (10.02 MB)
Trainable params: 2626305 (10.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [11]:
```
history_dict = history.history
history_dict.keys()
```

Out[11]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

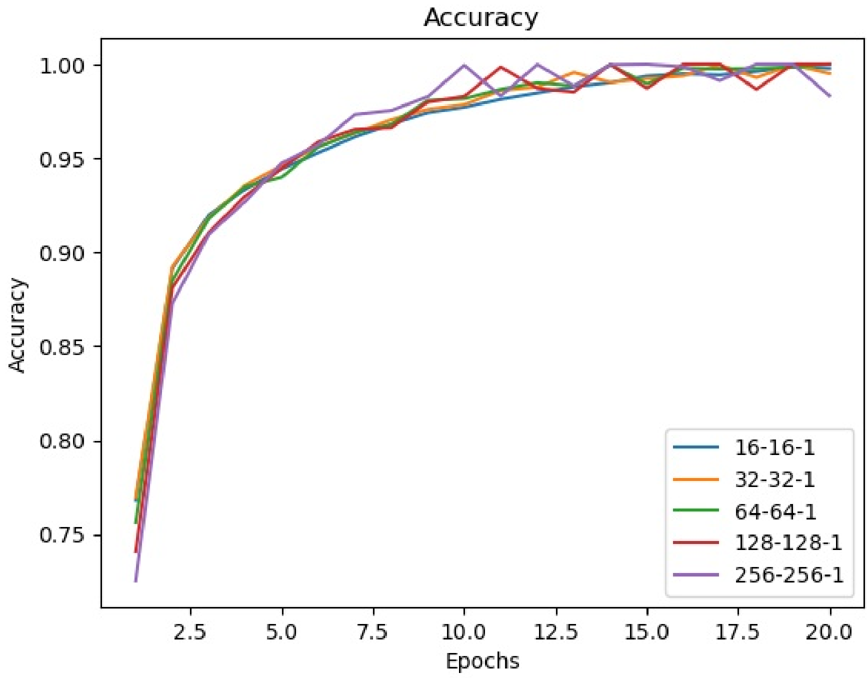
Modular code for plotting

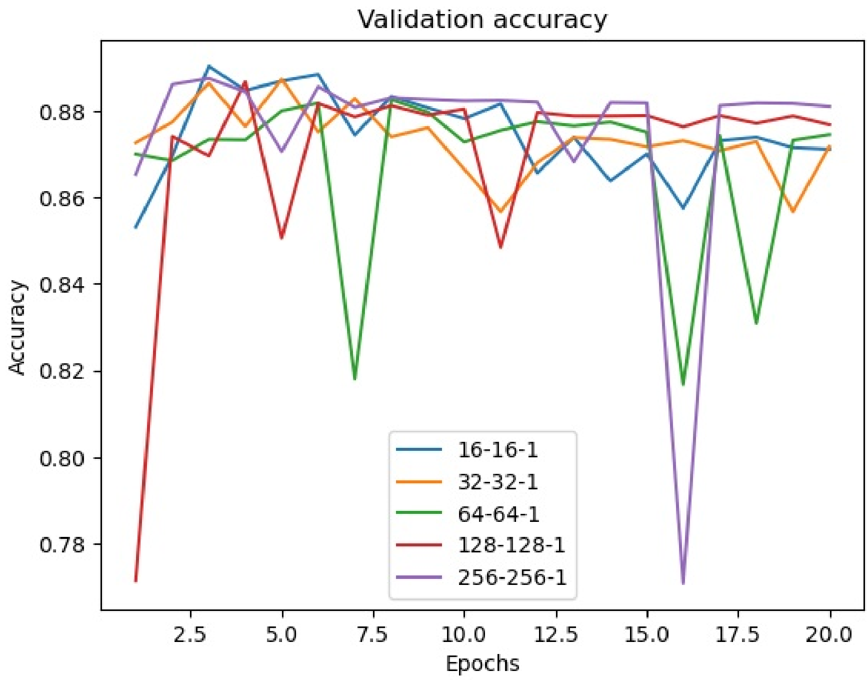
In [12]:
```
# plot training and validation loss helper functions for future use
acc = history_dict['accuracy']
#define training accuracy
def plot_acc_comparison(acc_a, label_a, acc_b, label_b, acc_c, label_c, acc_d, label_d, acc_e, label_e, y_label
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc_a, label=label_a)
    plt.plot(epochs, acc_b, label=label_b)
    plt.plot(epochs, acc_c, label=label_c)
    plt.plot(epochs, acc_d, label=label_d)
    plt.plot(epochs, acc_e, label=label_e)
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.show()

#define validation accuracy
def plot_val_acc_comparison(val_acc_a, label_a, val_acc_b, label_b, val_acc_c, label_c, val_acc_d, label_d,val_
                        y_label):
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, val_acc_a, label=label_a)
    plt.plot(epochs, val_acc_b, label=label_b)
    plt.plot(epochs, val_acc_c, label=label_c)
    plt.plot(epochs, val_acc_d, label=label_d)
    plt.plot(epochs, val_acc_e, label=label_e)
    plt.title('Validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.show()

#define validation loss
def plot_loss_comparison(loss_a, label_a, loss_b, label_b, loss_c, label_c, loss_d, label_d, loss_e, label_e, y
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, loss_a, label=label_a)
    plt.plot(epochs, loss_b, label=label_b)
    plt.plot(epochs, loss_c, label=label_c)
    plt.plot(epochs, loss_d, label=label_d)
    plt.plot(epochs, loss_e, label=label_e)
    plt.title('Validation loss')
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.show()

#define training loss
def plot_training_loss_comparison(tloss_a, label_a, tloss_b, label_b, tloss_c, label_c, tloss_d, label_d, tloss
                              y_label):
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, tloss_a, label=label_a)
    plt.plot(epochs, tloss_b, label=label_b)
    plt.plot(epochs, tloss_c, label=label_c)
    plt.plot(epochs, tloss_d, label=label_d)
    plt.plot(epochs, tloss_e, label=label_e)
    plt.title('Training loss')
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.show()

#plot for network a
def plot_net_a(acc , label_a, val_acc , label_b, tloss, label_c, loss, label_d, y_label):
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'b', label=label_a)
    plt.plot(epochs, val_acc, 'g', label=label_b)
    plt.plot(epochs, tloss, 'y', label=label_c)
    plt.plot(epochs, loss, 'm', label=label_d)
    plt.title('Network A: ' + net_a)
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.grid()
    plt.show()
#plot for network b
def plot_net_b(acc , label_a, val_acc , label_b, tloss, label_c, loss, label_d, y_label):
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'b', label=label_a)
    plt.plot(epochs, val_acc, 'g', label=label_b)
    plt.plot(epochs, tloss, 'y', label=label_c)
    plt.plot(epochs, loss, 'm', label=label_d)
    plt.title('Network B: ' + net_b)
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.grid()
    plt.show()
#plot for network c
def plot_net_c(acc , label_a, val_acc , label_b, tloss, label_c, loss, label_d, y_label):
```

```
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'b', label=label_a)
    plt.plot(epochs, val_acc, 'g', label=label_b)
    plt.plot(epochs, tloss, 'y', label=label_c)
    plt.plot(epochs, loss, 'm', label=label_d)
    plt.title('Network C: ' + net_c)
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.grid()
    plt.show()
#plot for network d
def plot_net_d(acc , label_a, val_acc , label_b, tloss, label_c, loss, label_d, y_label):
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'b', label=label_a)
    plt.plot(epochs, val_acc, 'g', label=label_b)
    plt.plot(epochs, tloss, 'y', label=label_c)
    plt.plot(epochs, loss, 'm', label=label_d)
    plt.title('Network D: ' + net_d)
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.grid()
    plt.show()
#plot for network e
def plot_net_e(acc , label_a, val_acc , label_b, tloss, label_c, loss, label_d, y_label):
    epochs = range(1, len(acc) + 1)
    plt.plot(epochs, acc, 'b', label=label_a)
    plt.plot(epochs, val_acc, 'g', label=label_b)
    plt.plot(epochs, tloss, 'y', label=label_c)
    plt.plot(epochs, loss, 'm', label=label_d)
    plt.title('Network E: ' + net_e)
    plt.xlabel('Epochs')
    plt.ylabel(y_label)
    plt.legend()
    plt.grid()
    plt.show()
```

Accuracy and validation accuracy

In [14]:
```
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e, = '16-16-1', '32-32-1', '64-64-1', '128-128-1', '256-256-1'
#plotting accuracy
plot_acc_comparison(acc_a=histories[net_a].history['accuracy'], label_a=net_a,
                     acc_b=histories[net_b].history['accuracy'], label_b=net_b,
                     acc_c=histories[net_c].history['accuracy'], label_c=net_c,
                    acc_d=histories[net_d].history['accuracy'], label_d=net_d,
                    acc_e=histories[net_e].history['accuracy'], label_e=net_e, y_label='Accuracy')
#plotting validation accuracy
plot_val_acc_comparison(val_acc_a=histories[net_a].history['val_accuracy'], label_a=net_a,
                     val_acc_b=histories[net_b].history['val_accuracy'], label_b=net_b,
                     val_acc_c=histories[net_c].history['val_accuracy'], label_c=net_c,
                        val_acc_d=histories[net_d].history['val_accuracy'], label_d=net_d,
                        val_acc_e=histories[net_e].history['val_accuracy'], label_e=net_e, y_label='Accuracy')
```



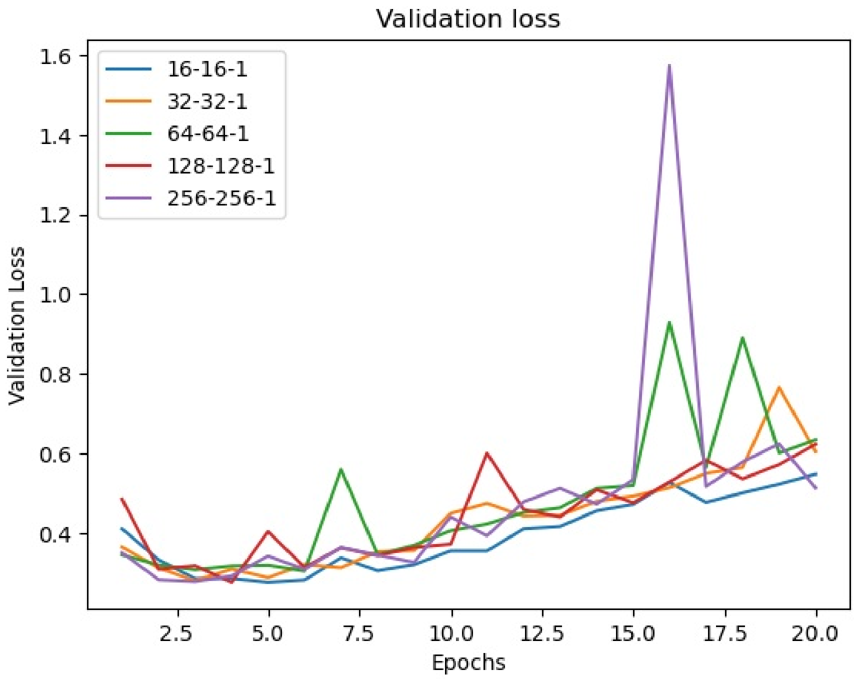


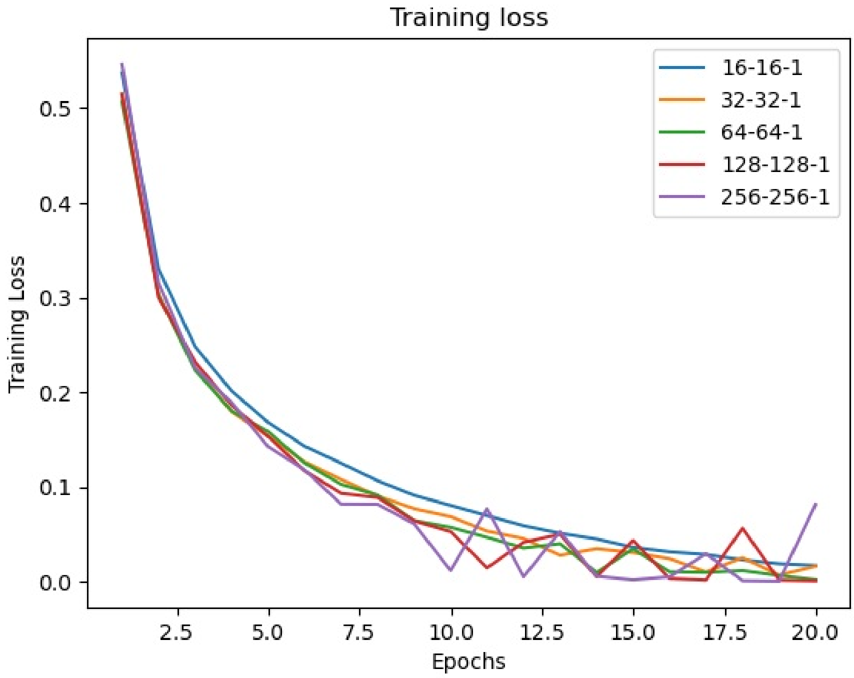
Accuracy plots interpetation

Accuracy and validation accuracy performed similarly for all 5 models. Not much noticeable variance can be noted.

Loss and validation loss

In [15]:
```
# plotting loss and validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
                     loss_e=histories[net_e].history['val_loss'], label_e=net_e,
                        y_label='Validation Loss')

plot_training_loss_comparison(tloss_a=histories[net_a].history['loss'], label_a=net_a,
                     tloss_b=histories[net_b].history['loss'], label_b=net_b,
                     tloss_c=histories[net_c].history['loss'], label_c=net_c,
                    tloss_d=histories[net_d].history['loss'], label_d=net_d,
                    tloss_e=histories[net_e].history['loss'], label_e=net_e,
                        y_label='Training Loss')
```





Loss and validation loss interpretation

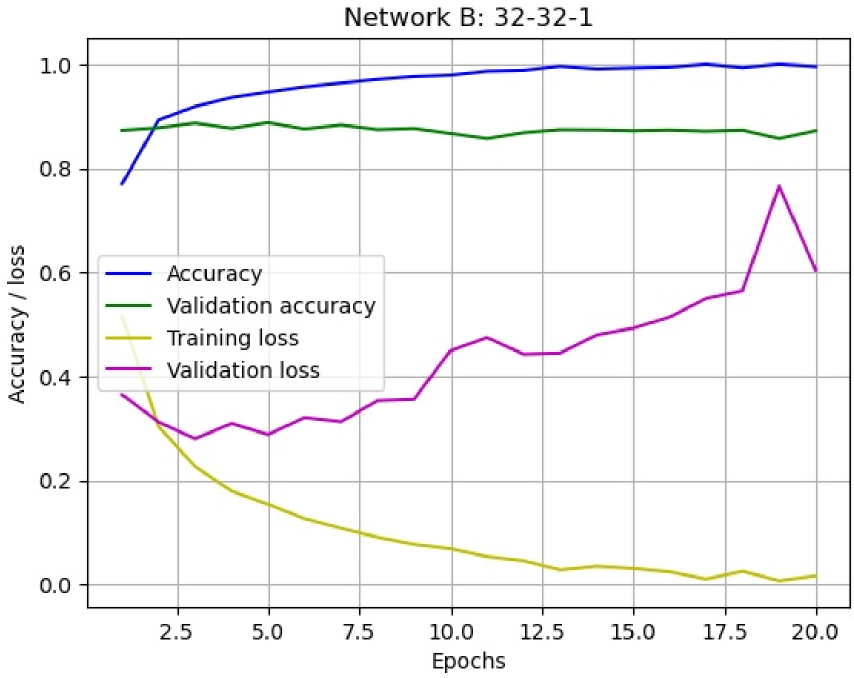
Validation loss shows that some models overtrain earlier, and some models will overtrain harder.

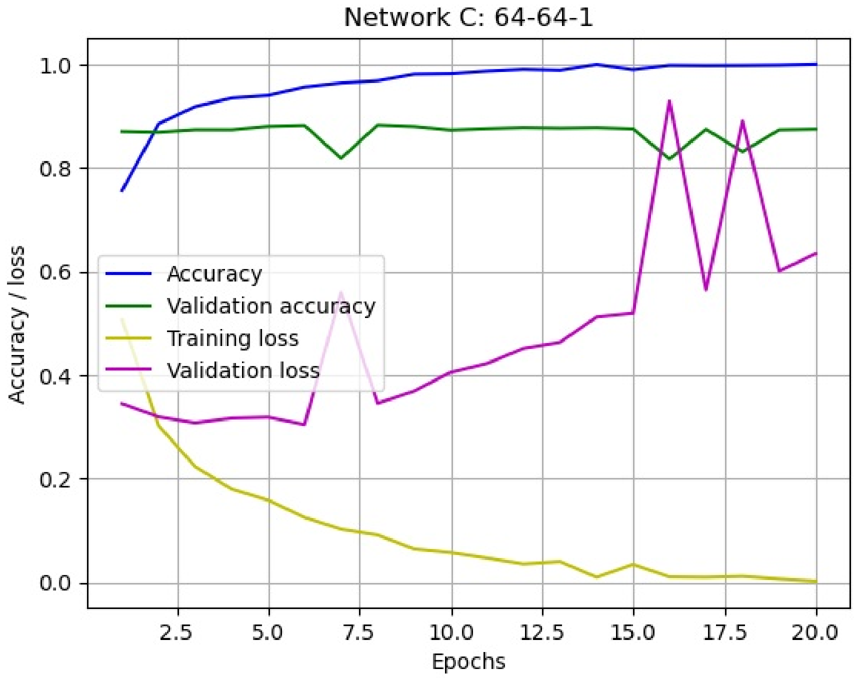
Training loss steadily declines for all models regardless of size.

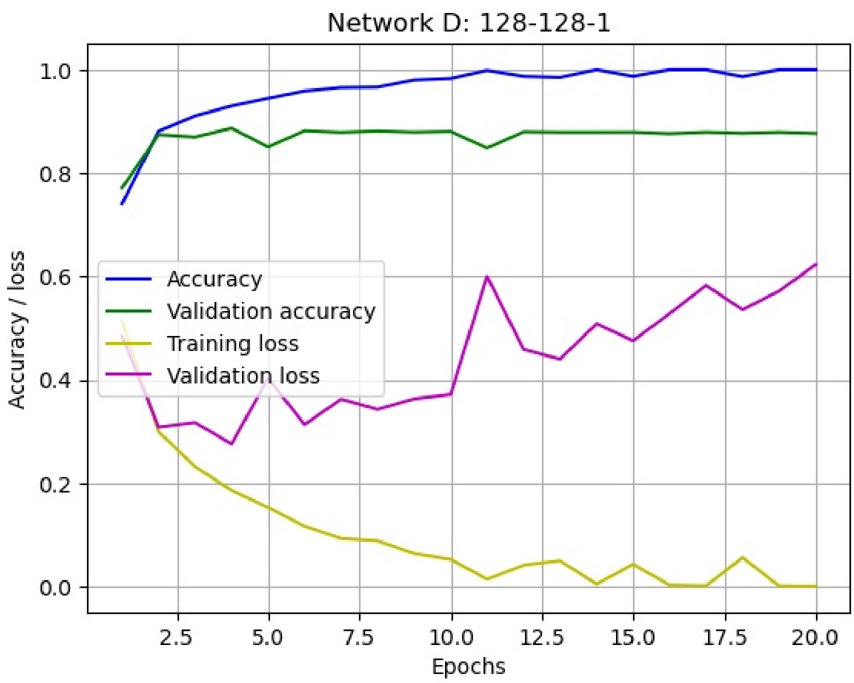
Plotting training and validation accuracy as well as loss of a few interesting models

We will start to examine the data for the middle range of networks -- 32, 64, and 128.

In [16]:
```
# selecting network b, c, d
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')

plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')

plot_net_d(acc=histories[net_d].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_d].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_d].history['loss'], label_c="Training loss",
            loss=histories[net_d].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Network b, c, d interpretation

Overfitting

Overfitting is when the model performs well on the training data, but performs poorly on test, validation, or unseen data. This is because the model becomes too specialized in solving the training data, and it can be said that the model memorizes the training data. This causes it to perform poorly on unseen datasets.

The overfitting of the models is quite clear in these 3 examples. By looking at the training and validation loss curves, we can see that there is a divergence early on in all 3 models.

To explain, the training loss curve decreases at a normal rate reaching near 0 by the end.

Lets try adding more layers

Let's try to add more layers and see if the model overtrains faster or more severely. We will add 1 more hidden relu layer and observe the results.

In [17]:
```
# define model
def build_model(layer_1_units, layer_2_units, layer_3_units, layer_4_units):
    model = models.Sequential()
    model.add(layers.Dense(layer_1_units, activation='relu', input_shape=(10000,)))
    model.add(layers.Dense(layer_2_units, activation='relu'))
    model.add(layers.Dense(layer_3_units, activation='relu'))
    model.add(layers.Dense(layer_4_units, activation='sigmoid'))
    model.compile(optimizer='rmsprop',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        c = ['\b|', '\b/', '\b-', '\b\\']
        print(c[epoch % 4], end='')
    def on_epoch_end(self, epoch, logs=None):
        print('\b', end='')
```

In [18]:
```
histories = {}
for i in [16, 32, 64, 128, 256]:
    model = build_model(i, i, i, 1)
    model_name = str(i) + '-' + str(i) + '-' + str(i) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
    model.reset_states()
```

```
Training 16-16-16-1
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_17 (Dense)            (None, 16)                160016

 dense_18 (Dense)            (None, 16)                272

 dense_19 (Dense)            (None, 16)                272

 dense_20 (Dense)            (None, 1)                 17

=================================================================
Total params: 160577 (627.25 KB)
Trainable params: 160577 (627.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-32-32-1
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 32)                320032

 dense_22 (Dense)            (None, 32)                1056

 dense_23 (Dense)            (None, 32)                1056

 dense_24 (Dense)            (None, 1)                 33

=================================================================
Total params: 322177 (1.23 MB)
```

```
Trainable params: 322177 (1.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 64-64-64-1
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_25 (Dense)            (None, 64)                640064    
                                                                 
 dense_26 (Dense)            (None, 64)                4160      
                                                                 
 dense_27 (Dense)            (None, 64)                4160      
                                                                 
 dense_28 (Dense)            (None, 1)                 65        
                                                                 
=================================================================
Total params: 648449 (2.47 MB)
Trainable params: 648449 (2.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 128-128-128-1
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_29 (Dense)            (None, 128)               1280128   
                                                                 
 dense_30 (Dense)            (None, 128)               16512     
                                                                 
 dense_31 (Dense)            (None, 128)               16512     
                                                                 
 dense_32 (Dense)            (None, 1)                 129       
                                                                 
=================================================================
Total params: 1313281 (5.01 MB)
Trainable params: 1313281 (5.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 256-256-256-1
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_33 (Dense)            (None, 256)               2560256   
                                                                 
 dense_34 (Dense)            (None, 256)               65792     
                                                                 
 dense_35 (Dense)            (None, 256)               65792     
                                                                 
 dense_36 (Dense)            (None, 1)                 257       
                                                                 
=================================================================
Total params: 2692097 (10.27 MB)
Trainable params: 2692097 (10.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

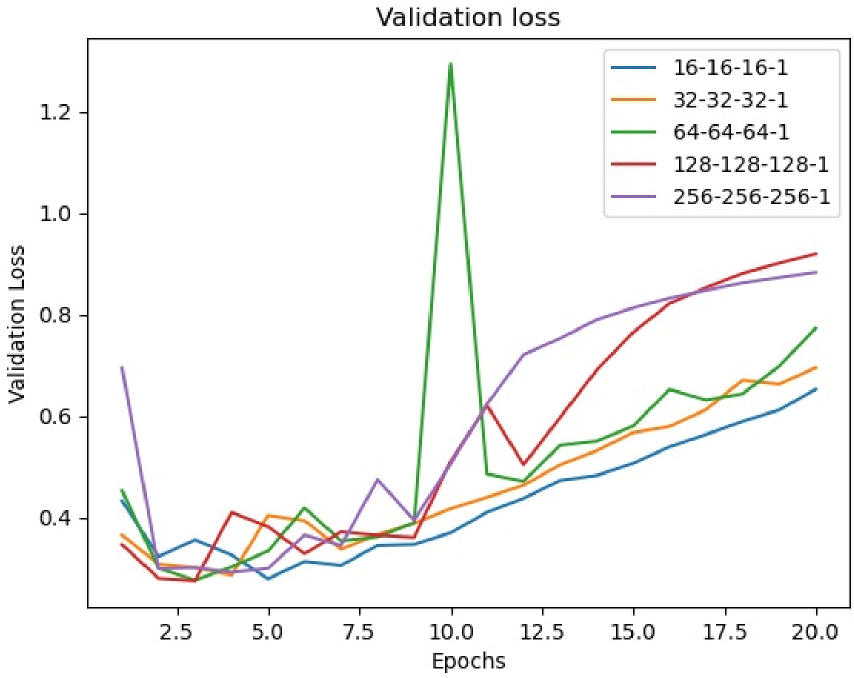
In [19]:

```
history_dict = history.history
```

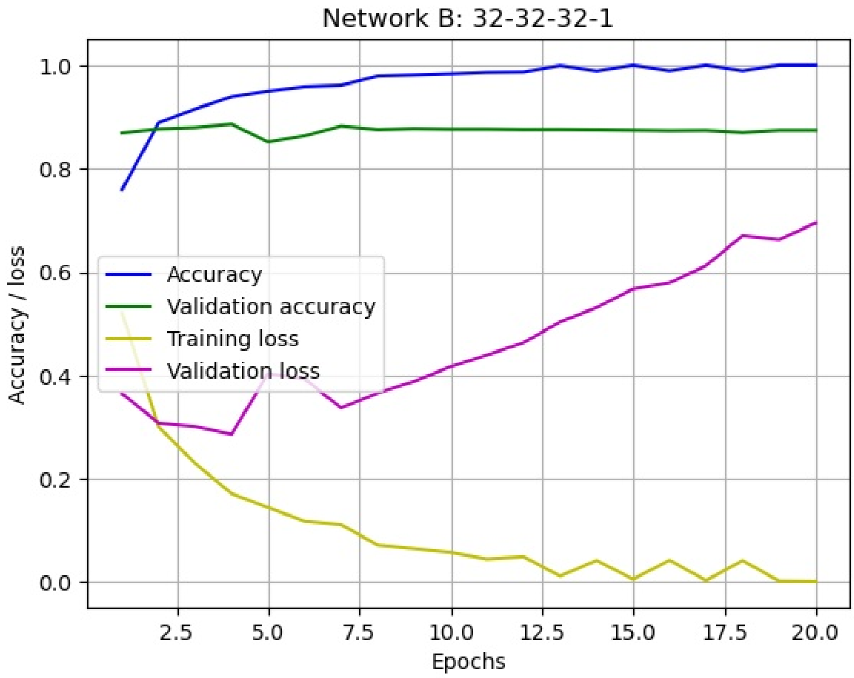
Validation loss

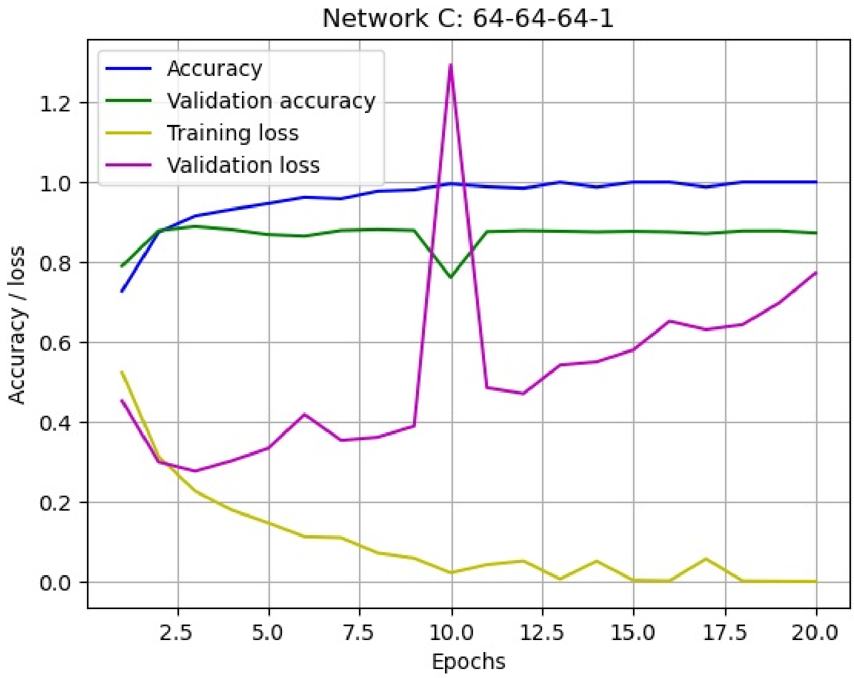
In [20]:

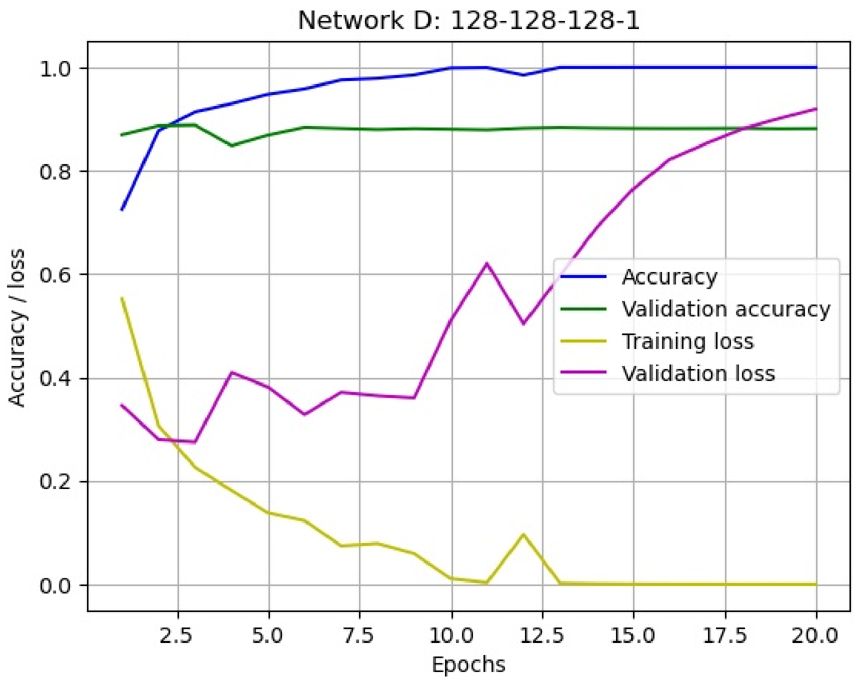
```
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e, = '16-16-16-1', '32-32-32-1', '64-64-64-1', '128-128-128-1', '256-256-256-1'
#plotting validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
                     loss_e=histories[net_e].history['val_loss'], label_e=net_e,
                        y_label='Validation Loss')
```



In [22]:
```
# plotting training and validation accuracy as well as loss of a few interesting models

plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')

plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_d(acc=histories[net_d].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_d].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_d].history['loss'], label_c="Training loss",
            loss=histories[net_d].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Network a, b, c interpretation

The result of adding another hidden layer is quite evident. We can see that the model is overtraining more severely as all 3 models of interest and went over 0.6 validation loss at the end. The previous 3 hidden layer model barley reached or did not go over 0.6 by the end of 20 epochs.

We can also see that the models are overtraining faster, with the batch from 4 layers starting at around 3 epochs. The previous 3 hidden layer model started at around 5 epochs.

Statistical power and overtraining achieved

The 4 hidden layer model can meet the requirements and we can move to the next phase.

7. Regularizing your model and tuning your hyperparameters

Dropout layers

Moving on to the next phase of taming the overfitting of the batch of models from the previous section. Lets start by adding some dropout layers between the relu layers. It is recommended to use a number between 0.2 and 0.5, so let's start high with 0.5.

In [27]:
```
# define model
def build_model(layer_1_units, layer_2_units, layer_3_units, layer_4_units):
    model = models.Sequential()
    model.add(layers.Dense(layer_1_units, activation='relu', input_shape=(10000,)))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_2_units, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_3_units, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_4_units, activation='sigmoid'))
    model.compile(optimizer='rmsprop',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        c = ['\b|', '\b/', '\b-', '\b\\']
        print(c[epoch % 4], end='')
    def on_epoch_end(self, epoch, logs=None):
        print('\b', end='')
```

In [43]:
```
histories = {}
model.reset_states()
for i in [16, 32, 64, 128, 256]:
    model = build_model(i, i, i, 1)
    model_name = str(i) + '-' + str(i) + '-' + str(i) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

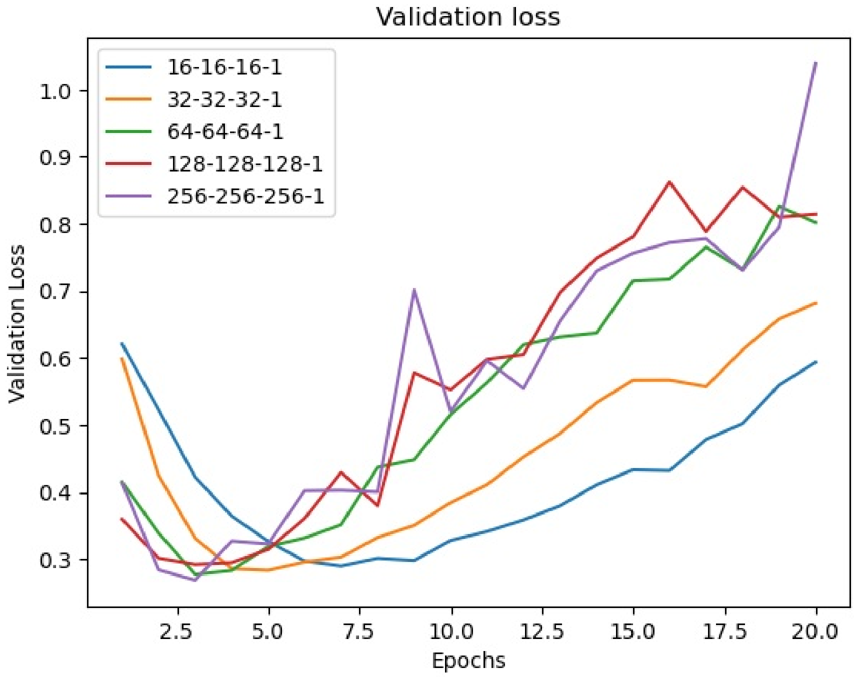
```
Training 16-16-16-1
Model: "sequential_34"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_129 (Dense)           (None, 16)                160016

 dropout_69 (Dropout)        (None, 16)                0

 dense_130 (Dense)           (None, 16)                272

 dropout_70 (Dropout)        (None, 16)                0

 dense_131 (Dense)           (None, 16)                272

 dropout_71 (Dropout)        (None, 16)                0

 dense_132 (Dense)           (None, 1)                 17

=================================================================
Total params: 160577 (627.25 KB)
Trainable params: 160577 (627.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-32-32-1
Model: "sequential_35"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_133 (Dense)           (None, 32)                320032

 dropout_72 (Dropout)        (None, 32)                0

 dense_134 (Dense)           (None, 32)                1056

 dropout_73 (Dropout)        (None, 32)                0

 dense_135 (Dense)           (None, 32)                1056
```

```
 dropout_74 (Dropout)        (None, 32)                0

 dense_136 (Dense)           (None, 1)                 33

=================================================================
Total params: 322177 (1.23 MB)
Trainable params: 322177 (1.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 64-64-64-1
Model: "sequential_36"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_137 (Dense)           (None, 64)                640064

 dropout_75 (Dropout)        (None, 64)                0

 dense_138 (Dense)           (None, 64)                4160

 dropout_76 (Dropout)        (None, 64)                0

 dense_139 (Dense)           (None, 64)                4160

 dropout_77 (Dropout)        (None, 64)                0

 dense_140 (Dense)           (None, 1)                 65

=================================================================
Total params: 648449 (2.47 MB)
Trainable params: 648449 (2.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 128-128-128-1
Model: "sequential_37"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_141 (Dense)           (None, 128)               1280128

 dropout_78 (Dropout)        (None, 128)               0

 dense_142 (Dense)           (None, 128)               16512

 dropout_79 (Dropout)        (None, 128)               0

 dense_143 (Dense)           (None, 128)               16512

 dropout_80 (Dropout)        (None, 128)               0

 dense_144 (Dense)           (None, 1)                 129

=================================================================
Total params: 1313281 (5.01 MB)
Trainable params: 1313281 (5.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 256-256-256-1
Model: "sequential_38"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_145 (Dense)           (None, 256)               2560256

 dropout_81 (Dropout)        (None, 256)               0

 dense_146 (Dense)           (None, 256)               65792

 dropout_82 (Dropout)        (None, 256)               0

 dense_147 (Dense)           (None, 256)               65792

 dropout_83 (Dropout)        (None, 256)               0

 dense_148 (Dense)           (None, 1)                 257

=================================================================
Total params: 2692097 (10.27 MB)
Trainable params: 2692097 (10.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

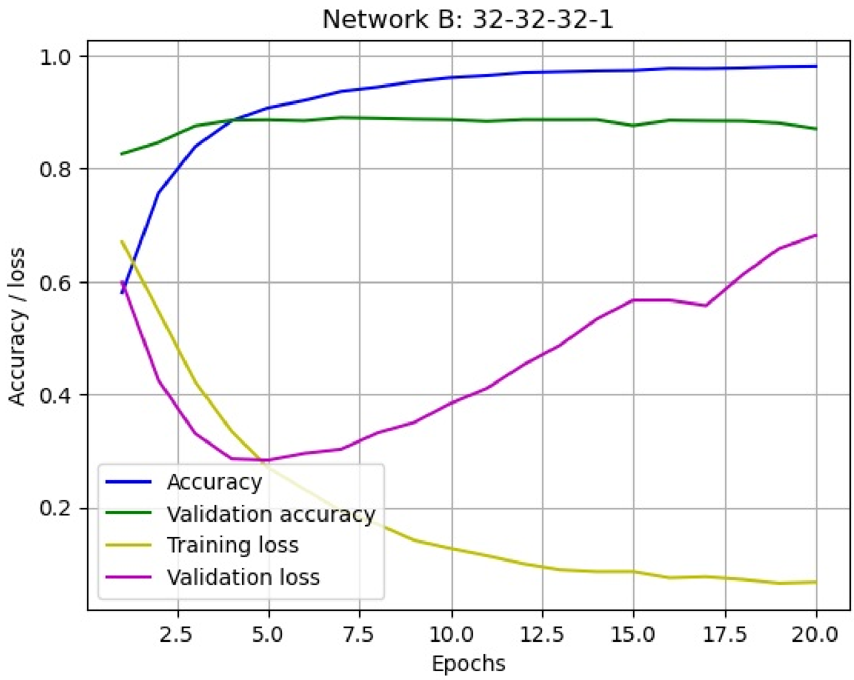
In [44]:

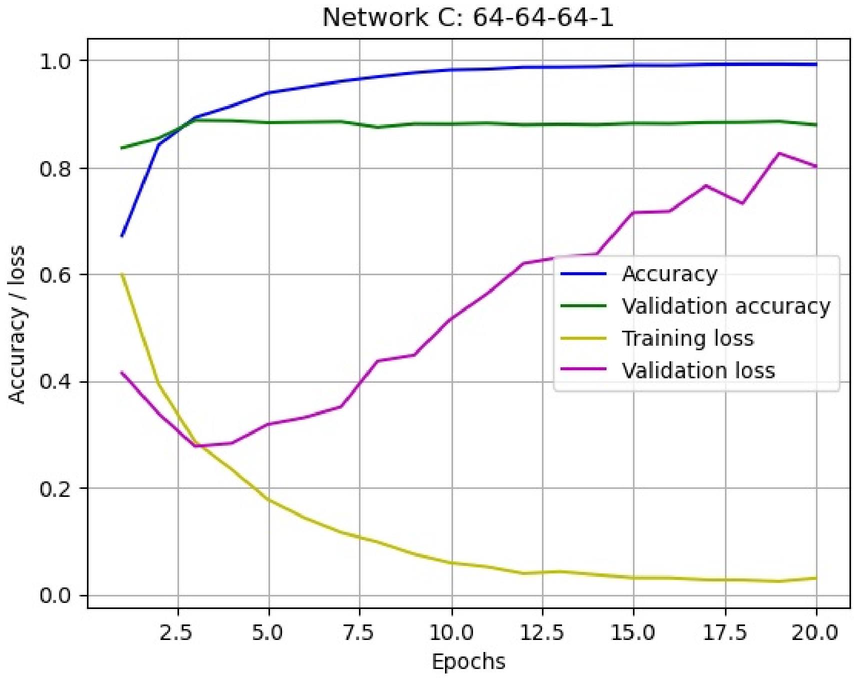
```python
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e, = '16-16-16-1', '32-32-32-1', '64-64-64-1', '128-128-128-1', '256-256-256-1'
#plotting validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
```

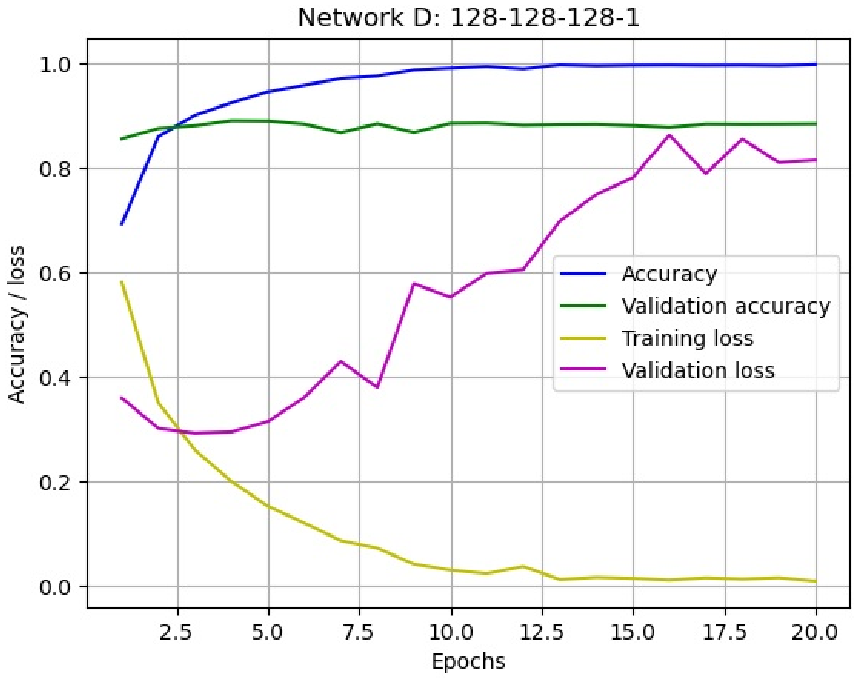
```
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
                     loss_e=histories[net_e].history['val_loss'], label_e=net_e,
                        y_label='Validation Loss')
```



In [45]:
```
# plotting training and validation accuracy as well as loss of a few interesting models

plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')

plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_d(acc=histories[net_d].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_d].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_d].history['loss'], label_c="Training loss",
            loss=histories[net_d].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation

The effect seen from the aggressive dropout did have some effect, but not a very noticeable one. We can see that it benefited the smaller networks more, and we can see that the overfitting happened later for the 16-16-16-1 network. The drop in validation loss was also also steeper, with all networks showing a more spread out grouping.

Perhaps we need to combine more regularization methods to better tame the overfitting.

Focus on the smaller networks

At this point, it is noticed that the larger networks are overfitting more severely for our problem, so we will just focus on the smaller networks as they would be easier to tame. The end result will most definitely not use one of the larger models.

Weight regularization

As stated in the DLWP 4.4.2, there are two kinds of weight regularization:

- L1 regularization
- L2 regularization

There are options to try either one or both at the same time with the network. We will start with L1 first.

L1 regularization

In [47]:
```
# define model
def build_model(layer_1_units, layer_2_units, layer_3_units, layer_4_units):
    model = models.Sequential()
    model.add(layers.Dense(layer_1_units, kernel_regularizer=regularizers.l1(0.001), activation='relu', input_s
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_2_units, kernel_regularizer=regularizers.l1(0.001), activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_3_units, kernel_regularizer=regularizers.l1(0.001), activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_4_units, activation='sigmoid'))
    model.compile(optimizer='rmsprop',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        c = ['\b|', '\b/', '\b-', '\b\\']
        print(c[epoch % 4], end='')
    def on_epoch_end(self, epoch, logs=None):
        print('\b', end='')
```

In [51]:
```
histories = {}
model.reset_states()
for i in [16, 32, 64, 128, 256]:
    model = build_model(i, i, i, 1)
    model_name = str(i) + '-' + str(i) + '-' + str(i) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

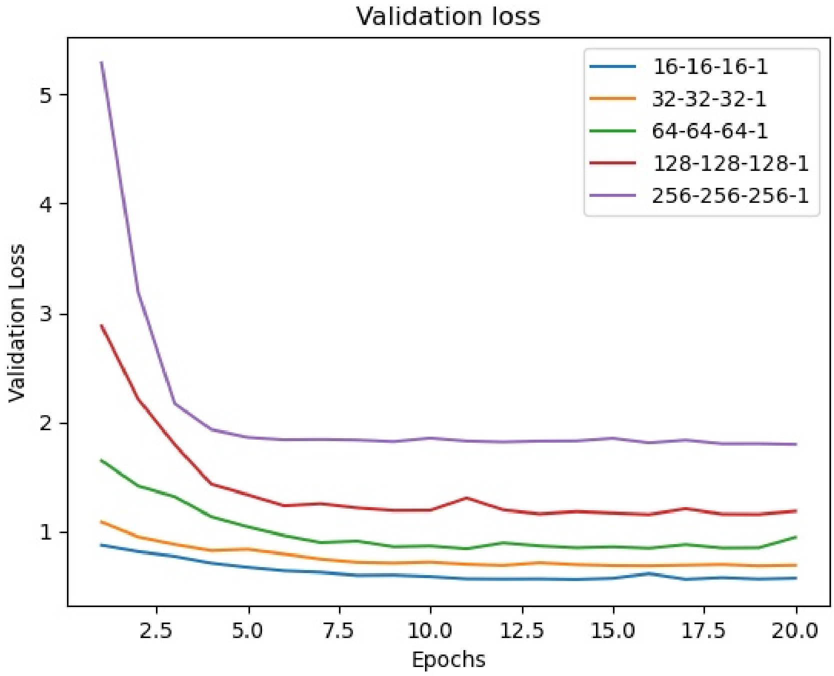
```
Training 16-16-16-1
Model: "sequential_42"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_161 (Dense)           (None, 16)                160016    
                                                                 
 dropout_93 (Dropout)        (None, 16)                0         
                                                                 
 dense_162 (Dense)           (None, 16)                272       
                                                                 
 dropout_94 (Dropout)        (None, 16)                0         
                                                                 
 dense_163 (Dense)           (None, 16)                272       
                                                                 
 dropout_95 (Dropout)        (None, 16)                0         
                                                                 
 dense_164 (Dense)           (None, 1)                 17        
                                                                 
=================================================================
Total params: 160577 (627.25 KB)
Trainable params: 160577 (627.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-32-32-1
Model: "sequential_43"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_165 (Dense)           (None, 32)                320032    
                                                                 
 dropout_96 (Dropout)        (None, 32)                0         
                                                                 
 dense_166 (Dense)           (None, 32)                1056      
                                                                 
 dropout_97 (Dropout)        (None, 32)                0         
                                                                 
 dense_167 (Dense)           (None, 32)                1056      
                                                                 
 dropout_98 (Dropout)        (None, 32)                0         
```

```
 dense_168 (Dense)           (None, 1)                 33

=================================================================
Total params: 322177 (1.23 MB)
Trainable params: 322177 (1.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 64-64-64-1
Model: "sequential_44"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_169 (Dense)           (None, 64)                640064    

 dropout_99 (Dropout)        (None, 64)                0         

 dense_170 (Dense)           (None, 64)                4160      

 dropout_100 (Dropout)       (None, 64)                0         

 dense_171 (Dense)           (None, 64)                4160      

 dropout_101 (Dropout)       (None, 64)                0         

 dense_172 (Dense)           (None, 1)                 65        

=================================================================
Total params: 648449 (2.47 MB)
Trainable params: 648449 (2.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 128-128-128-1
Model: "sequential_45"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_173 (Dense)           (None, 128)               1280128   

 dropout_102 (Dropout)       (None, 128)               0         

 dense_174 (Dense)           (None, 128)               16512     

 dropout_103 (Dropout)       (None, 128)               0         

 dense_175 (Dense)           (None, 128)               16512     

 dropout_104 (Dropout)       (None, 128)               0         

 dense_176 (Dense)           (None, 1)                 129       

=================================================================
Total params: 1313281 (5.01 MB)
Trainable params: 1313281 (5.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 256-256-256-1
Model: "sequential_46"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_177 (Dense)           (None, 256)               2560256   

 dropout_105 (Dropout)       (None, 256)               0         

 dense_178 (Dense)           (None, 256)               65792     

 dropout_106 (Dropout)       (None, 256)               0         

 dense_179 (Dense)           (None, 256)               65792     

 dropout_107 (Dropout)       (None, 256)               0         

 dense_180 (Dense)           (None, 1)                 257       

=================================================================
Total params: 2692097 (10.27 MB)
Trainable params: 2692097 (10.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

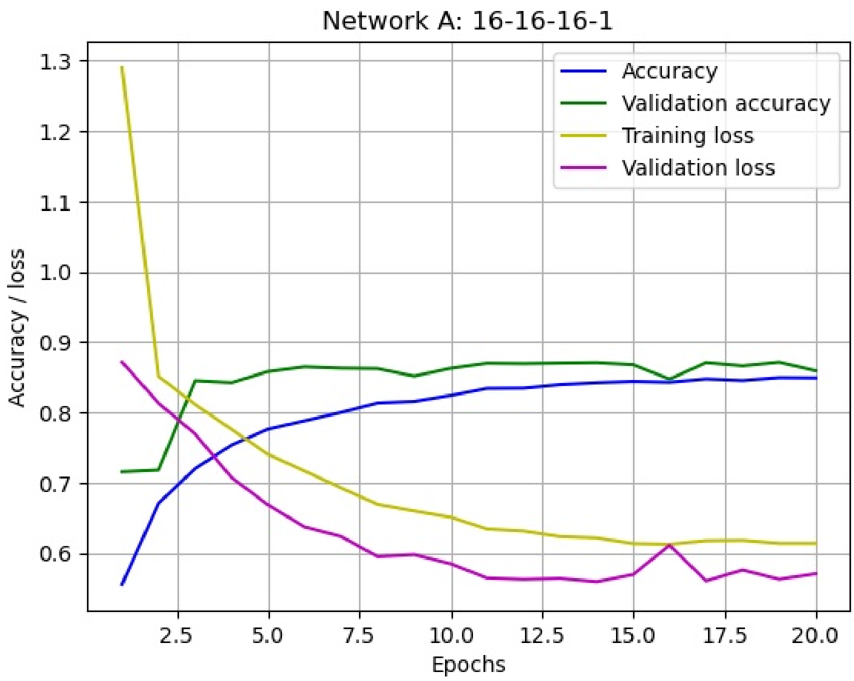
In [52]:

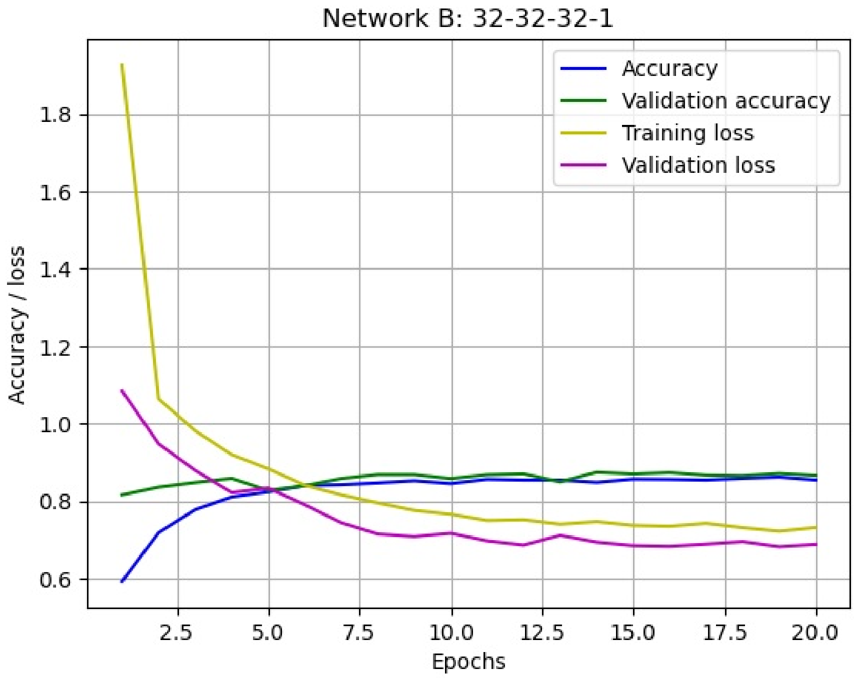
```
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e, = '16-16-16-1', '32-32-32-1', '64-64-64-1', '128-128-128-1', '256-256-256-1'
#plotting validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
                     loss_e=histories[net_e].history['val_loss'], label_e=net_e,
```

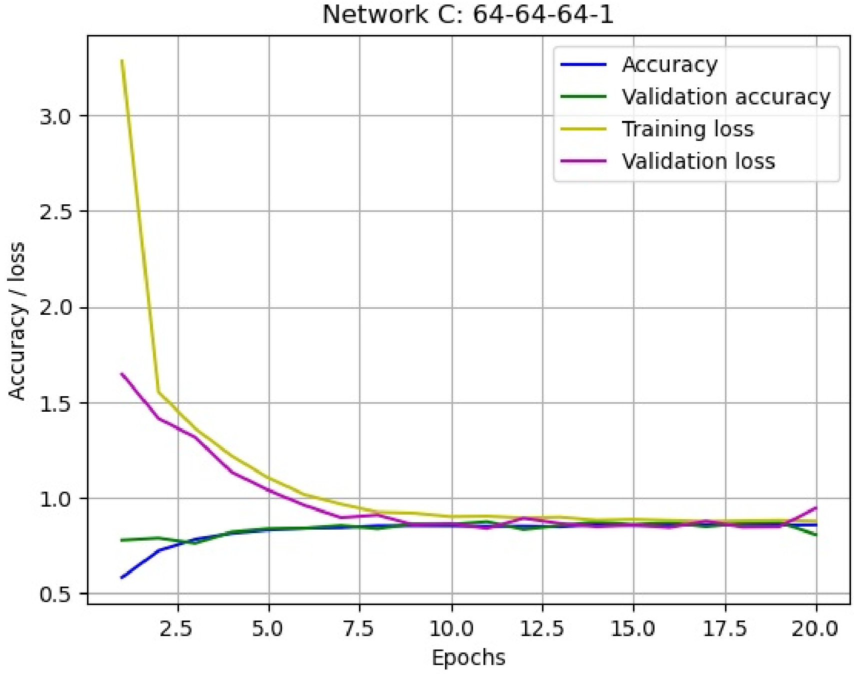
```
                         y_label='Validation Loss')
```



In [53]:
```
# plotting training and validation accuracy as well as loss of a few interesting models
plot_net_a(acc=histories[net_a].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_a].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_a].history['loss'], label_c="Training loss",
            loss=histories[net_a].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation

Not quite sure how to interpret the results, or maybe it is not the correct method for what we are working on, but let's move on to L2 and see if something else happens.

L2 regularization

In [77]:
```
# define model
def build_model(layer_1_units, layer_2_units, layer_3_units, layer_4_units):
    model = models.Sequential()
    model.add(layers.Dense(layer_1_units, kernel_regularizer=regularizers.l2(0.001), activation='relu', input_s
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_2_units, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
```

```python
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_3_units, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(layer_4_units, activation='sigmoid'))
    model.compile(optimizer='rmsprop',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        c = ['\b|', '\b/', '\b-', '\b\\']
        print(c[epoch % 4], end='')
    def on_epoch_end(self, epoch, logs=None):
        print('\b', end='')
```

In [55]:

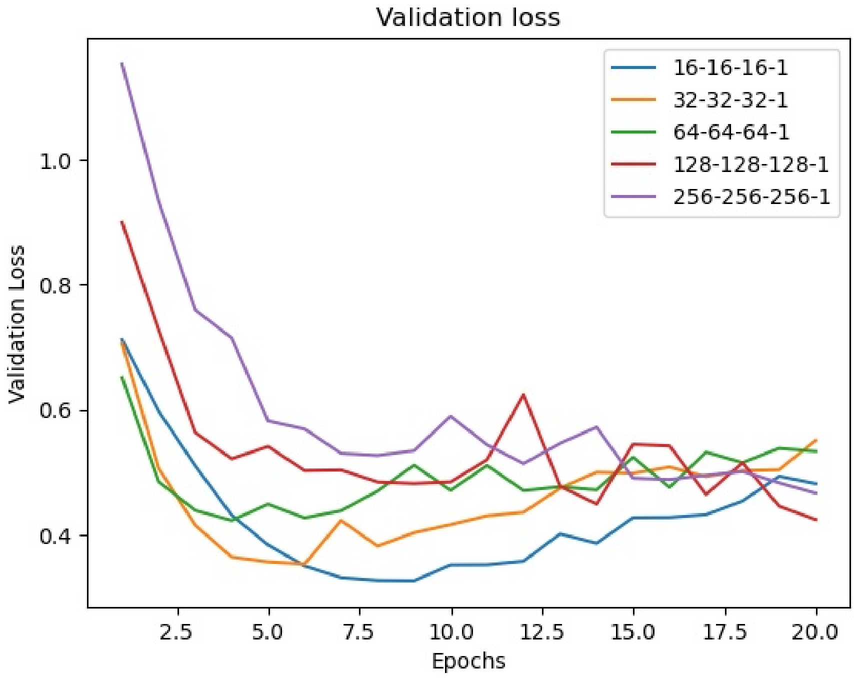
```python
histories = {}
model.reset_states()
for i in [16, 32, 64, 128, 256]:
    model = build_model(i, i, i, 1)
    model_name = str(i) + '-' + str(i) + '-' + str(i) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

```
Training 16-16-16-1
Model: "sequential_47"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_181 (Dense)           (None, 16)                160016    
                                                                 
 dropout_108 (Dropout)       (None, 16)                0         
                                                                 
 dense_182 (Dense)           (None, 16)                272       
                                                                 
 dropout_109 (Dropout)       (None, 16)                0         
                                                                 
 dense_183 (Dense)           (None, 16)                272       
                                                                 
 dropout_110 (Dropout)       (None, 16)                0         
                                                                 
 dense_184 (Dense)           (None, 1)                 17        
                                                                 
=================================================================
Total params: 160577 (627.25 KB)
Trainable params: 160577 (627.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-32-32-1
Model: "sequential_48"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_185 (Dense)           (None, 32)                320032    
                                                                 
 dropout_111 (Dropout)       (None, 32)                0         
                                                                 
 dense_186 (Dense)           (None, 32)                1056      
                                                                 
 dropout_112 (Dropout)       (None, 32)                0         
                                                                 
 dense_187 (Dense)           (None, 32)                1056      
                                                                 
 dropout_113 (Dropout)       (None, 32)                0         
                                                                 
 dense_188 (Dense)           (None, 1)                 33        
                                                                 
=================================================================
Total params: 322177 (1.23 MB)
Trainable params: 322177 (1.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 64-64-64-1
Model: "sequential_49"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_189 (Dense)           (None, 64)                640064    
                                                                 
 dropout_114 (Dropout)       (None, 64)                0         
```

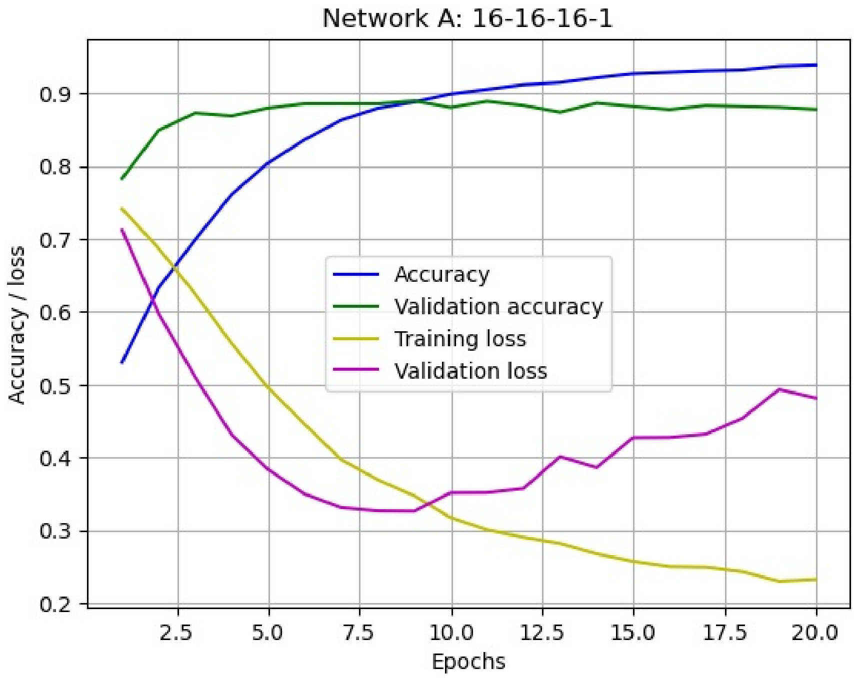
```
 dense_190 (Dense)           (None, 64)                4160

 dropout_115 (Dropout)       (None, 64)                0

 dense_191 (Dense)           (None, 64)                4160

 dropout_116 (Dropout)       (None, 64)                0

 dense_192 (Dense)           (None, 1)                 65

=================================================================
Total params: 648449 (2.47 MB)
Trainable params: 648449 (2.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 128-128-128-1
Model: "sequential_50"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_193 (Dense)           (None, 128)               1280128

 dropout_117 (Dropout)       (None, 128)               0

 dense_194 (Dense)           (None, 128)               16512

 dropout_118 (Dropout)       (None, 128)               0

 dense_195 (Dense)           (None, 128)               16512

 dropout_119 (Dropout)       (None, 128)               0

 dense_196 (Dense)           (None, 1)                 129

=================================================================
Total params: 1313281 (5.01 MB)
Trainable params: 1313281 (5.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 256-256-256-1
Model: "sequential_51"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_197 (Dense)           (None, 256)               2560256

 dropout_120 (Dropout)       (None, 256)               0

 dense_198 (Dense)           (None, 256)               65792

 dropout_121 (Dropout)       (None, 256)               0

 dense_199 (Dense)           (None, 256)               65792

 dropout_122 (Dropout)       (None, 256)               0

 dense_200 (Dense)           (None, 1)                 257

=================================================================
Total params: 2692097 (10.27 MB)
Trainable params: 2692097 (10.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

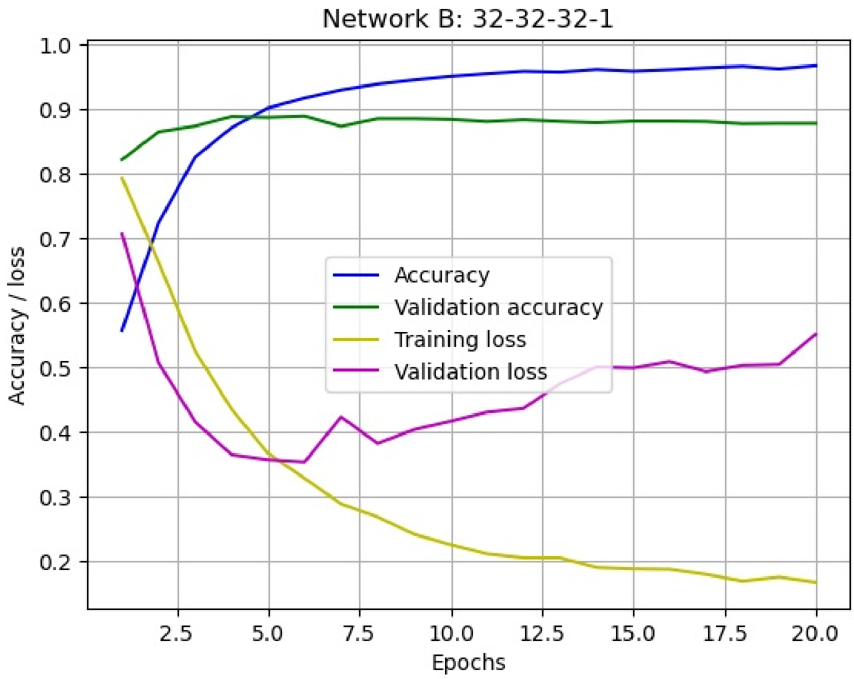
In [56]:

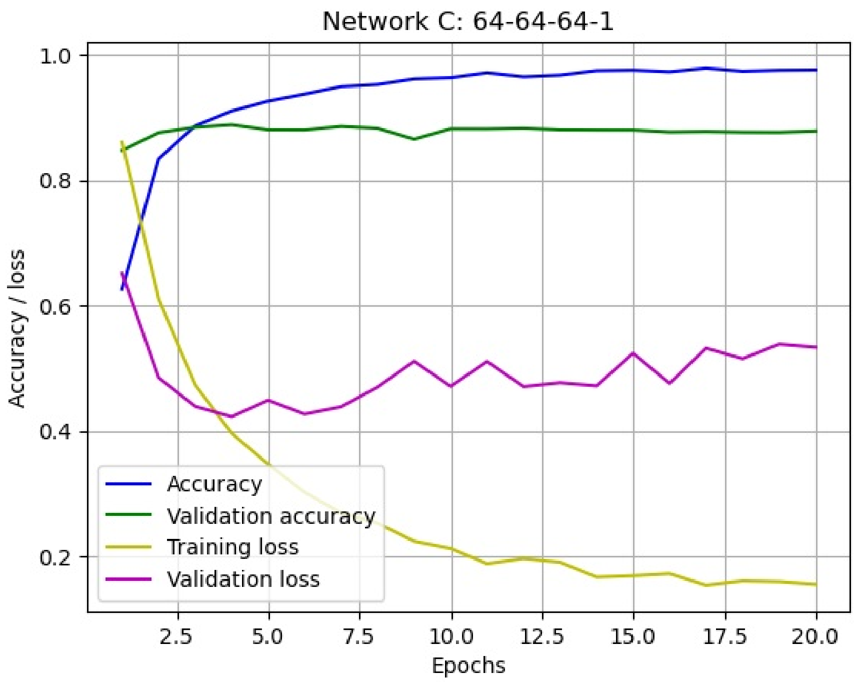
```python
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e, = '16-16-16-1', '32-32-32-1', '64-64-64-1', '128-128-128-1', '256-256-256-1'
#plotting validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
                     loss_e=histories[net_e].history['val_loss'], label_e=net_e,
                        y_label='Validation Loss')
```



In [57]:
```
# plotting training and validation accuracy as well as loss of a few interesting models
plot_net_a(acc=histories[net_a].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_a].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_a].history['loss'], label_c="Training loss",
            loss=histories[net_a].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_c(acc=histories[net_c]a.history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation

So, we can see the results are much more favorable from L2, and we can see that the validation loss flattens off at some level and does not increase (at least within 20 epochs). This is great progress as the overfitting at least is not out of control. We still need to find better ways of reducing the overfitting. The next idea would be to reduce the network size.

Reduce network size

Since the larger networks are still overfitting too much and will not make it to the final cut, we will remove 256 and add an 8 network. Also, we can try to taper the hidden units per layer by half for the 2nd and 3rd layer.

Taper the hidden units per layer

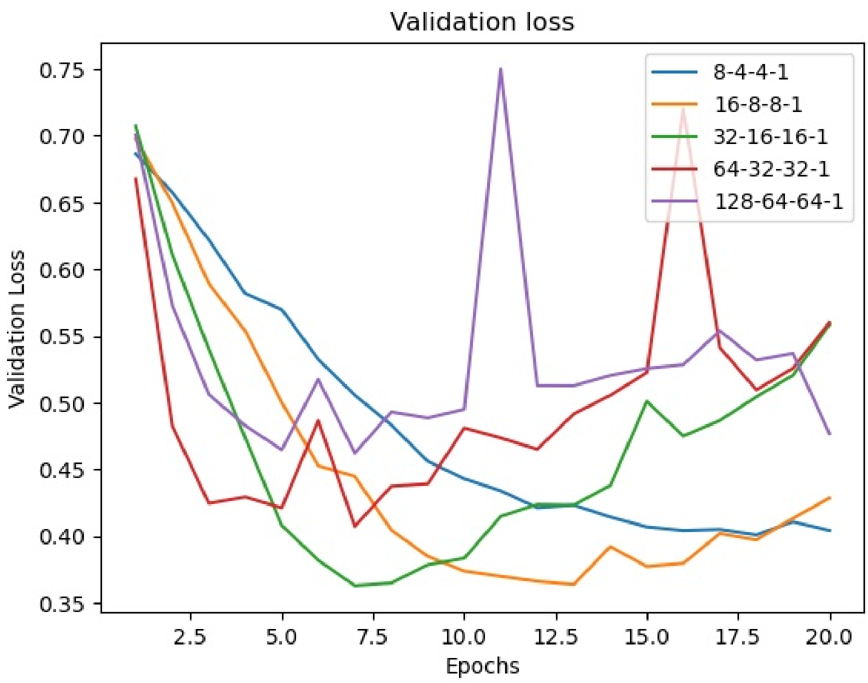
In [58]:
```
histories = {}
model.reset_states()
## use //2 to reduce the network of i
for i in [8, 16, 32, 64, 128]:
    model = build_model(i, i //2, i //2, 1)
    model_name = str(i) + '-' + str(i //2) + '-' + str(i //2) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

```
Training 8-4-4-1
Model: "sequential_52"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_201 (Dense)           (None, 8)                 80008

 dropout_123 (Dropout)       (None, 8)                 0

 dense_202 (Dense)           (None, 4)                 36

 dropout_124 (Dropout)       (None, 4)                 0

 dense_203 (Dense)           (None, 4)                 20

 dropout_125 (Dropout)       (None, 4)                 0

 dense_204 (Dense)           (None, 1)                 5

=================================================================
Total params: 80069 (312.77 KB)
Trainable params: 80069 (312.77 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-8-1
Model: "sequential_53"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_205 (Dense)           (None, 16)                160016

 dropout_126 (Dropout)       (None, 16)                0

 dense_206 (Dense)           (None, 8)                 136

 dropout_127 (Dropout)       (None, 8)                 0

 dense_207 (Dense)           (None, 8)                 72

 dropout_128 (Dropout)       (None, 8)                 0

 dense_208 (Dense)           (None, 1)                 9

=================================================================
Total params: 160233 (625.91 KB)
Trainable params: 160233 (625.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-16-16-1
Model: "sequential_54"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_209 (Dense)           (None, 32)                320032

 dropout_129 (Dropout)       (None, 32)                0

 dense_210 (Dense)           (None, 16)                528

 dropout_130 (Dropout)       (None, 16)                0

 dense_211 (Dense)           (None, 16)                272

 dropout_131 (Dropout)       (None, 16)                0

 dense_212 (Dense)           (None, 1)                 17

=================================================================
```

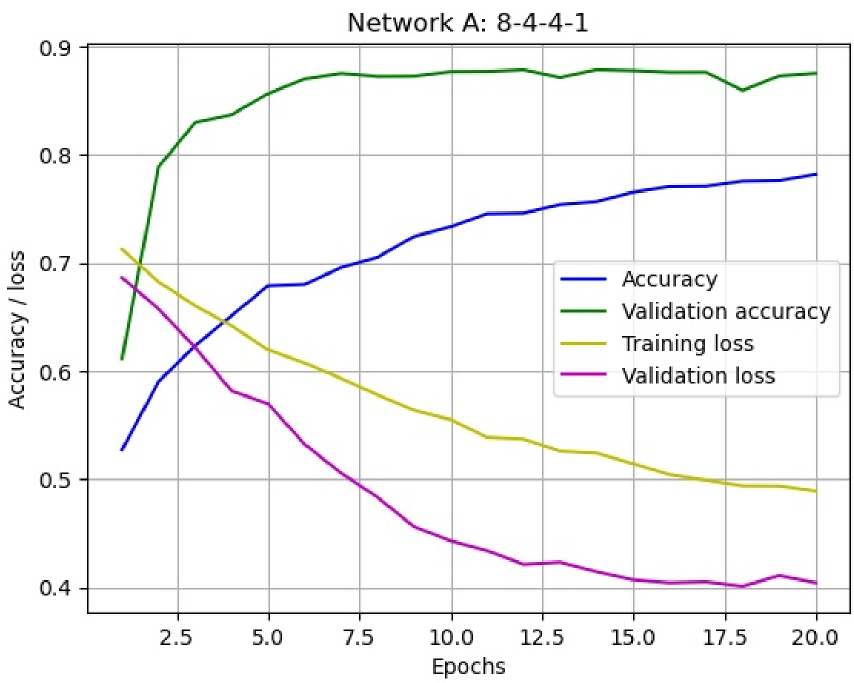
```
Total params: 320849 (1.22 MB)
Trainable params: 320849 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 64-32-32-1
Model: "sequential_55"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_213 (Dense)           (None, 64)                640064

 dropout_132 (Dropout)       (None, 64)                0

 dense_214 (Dense)           (None, 32)                2080

 dropout_133 (Dropout)       (None, 32)                0

 dense_215 (Dense)           (None, 32)                1056

 dropout_134 (Dropout)       (None, 32)                0

 dense_216 (Dense)           (None, 1)                 33

=================================================================
Total params: 643233 (2.45 MB)
Trainable params: 643233 (2.45 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 128-64-64-1
Model: "sequential_56"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_217 (Dense)           (None, 128)               1280128

 dropout_135 (Dropout)       (None, 128)               0

 dense_218 (Dense)           (None, 64)                8256

 dropout_136 (Dropout)       (None, 64)                0

 dense_219 (Dense)           (None, 64)                4160

 dropout_137 (Dropout)       (None, 64)                0

 dense_220 (Dense)           (None, 1)                 65

=================================================================
Total params: 1292609 (4.93 MB)
Trainable params: 1292609 (4.93 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

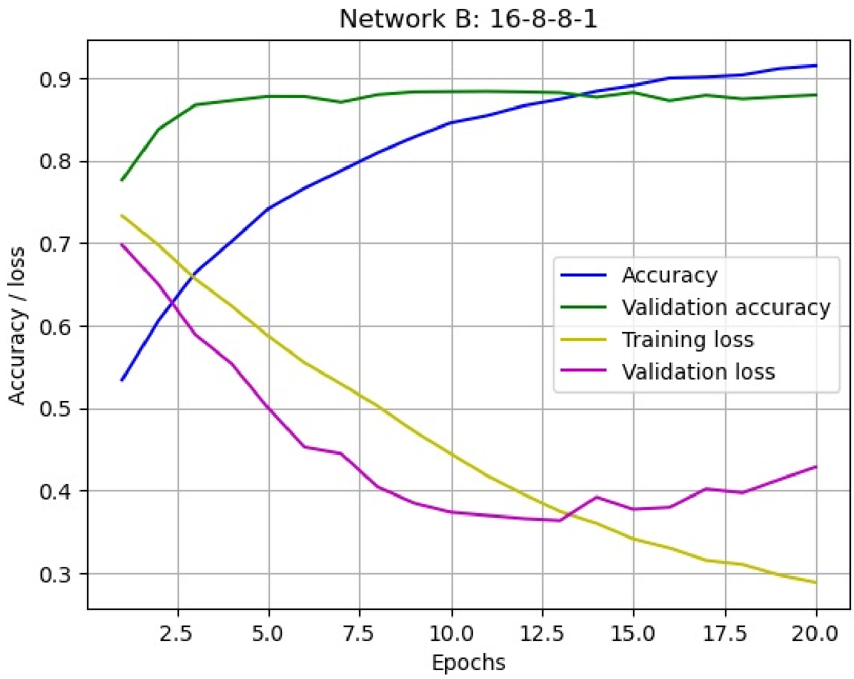
In [59]:

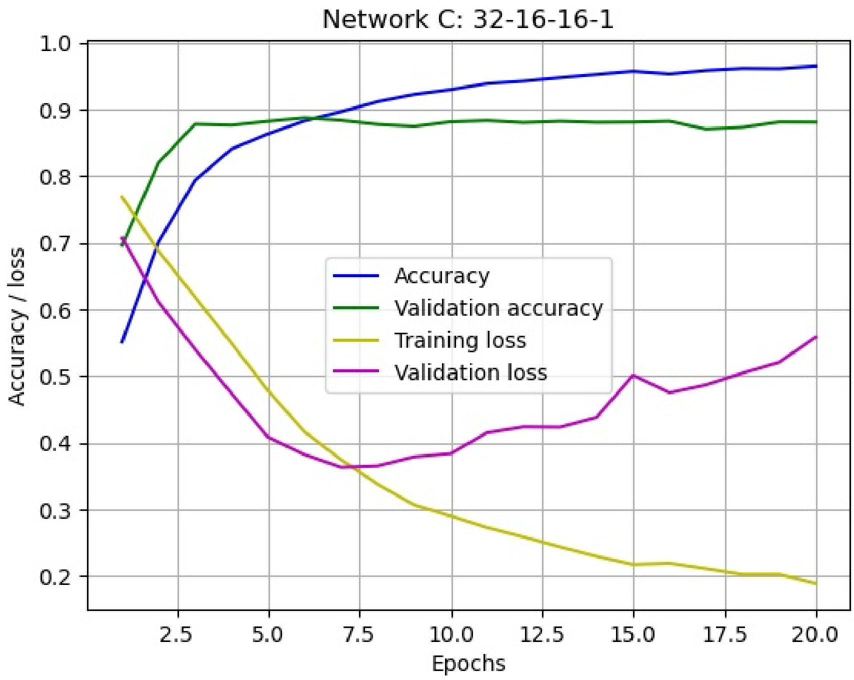
```
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e = '8-4-4-1', '16-8-8-1', '32-16-16-1', '64-32-32-1', '128-64-64-1'
#plotting validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
                     loss_e=histories[net_e].history['val_loss'], label_e=net_e,
                        y_label='Validation Loss')
```



In [61]:
```
# plotting training and validation accuracy as well as loss of a few interesting models
plot_net_a(acc=histories[net_a].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_a].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_a].history['loss'], label_c="Training loss",
            loss=histories[net_a].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation

Networks of interest:

- net_a (8-4-4-1)

We can see the that validation loss drops quite a bit and is actually lower than training loss. The accuracy is also quite low and it looks like this network is not powerful enough to solve our problem.

- net_b (16-8-8-1)

The training and validation loss drops together before diverging at around epoch 13. The network is still slightly overfitting. The accuracy is also not bad, but rises slowly.

- net_c (32-16-16-1)

The network is still overfitting.

Taper more aggressively

Let's try a batch of networks with 100% capacity for first layer, 50% capacity for second layer, and 25% capacity for third layer.

We will also increase the epochs to 50 to understand more about what is happening beyond epoch 20.

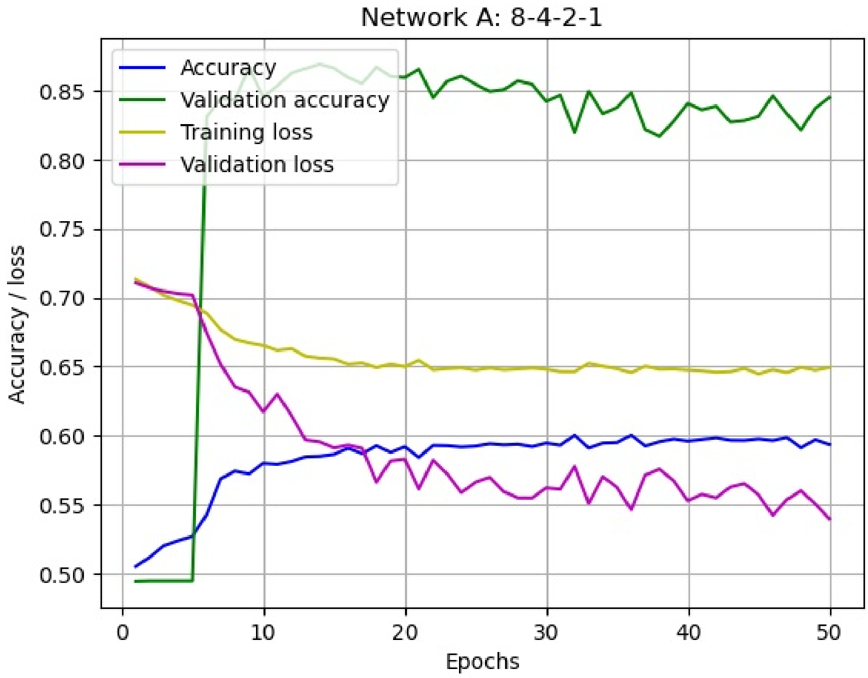
In [78]:

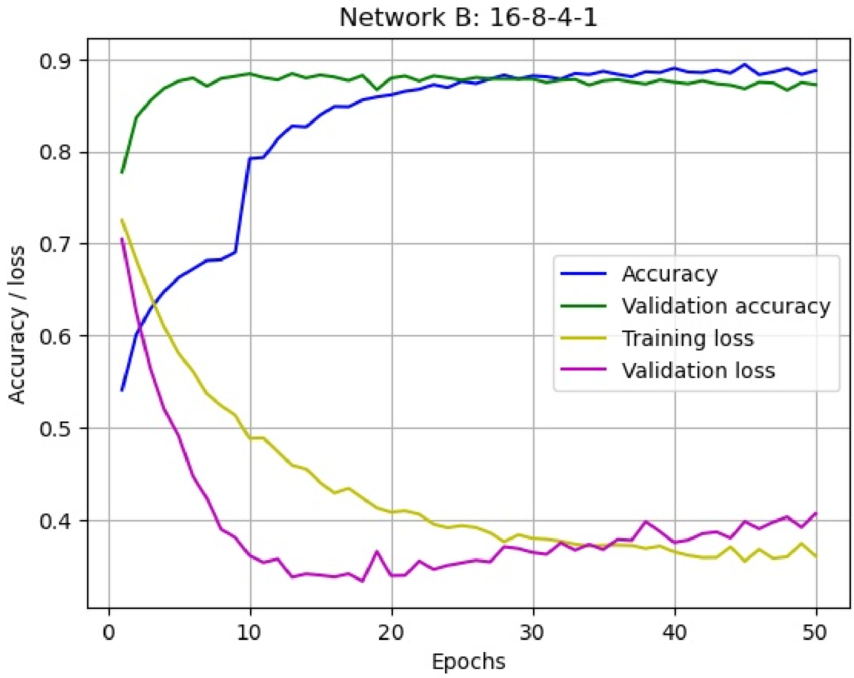
```
histories = {}
model.reset_states()
## use //2 and //4 to reduce the network of i
for i in [8, 16, 32]:
    model = build_model(i, i //2, i //4, 1)
    model_name = str(i) + '-' + str(i //2) + '-' + str(i //4) + '-' + str(1)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=50,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

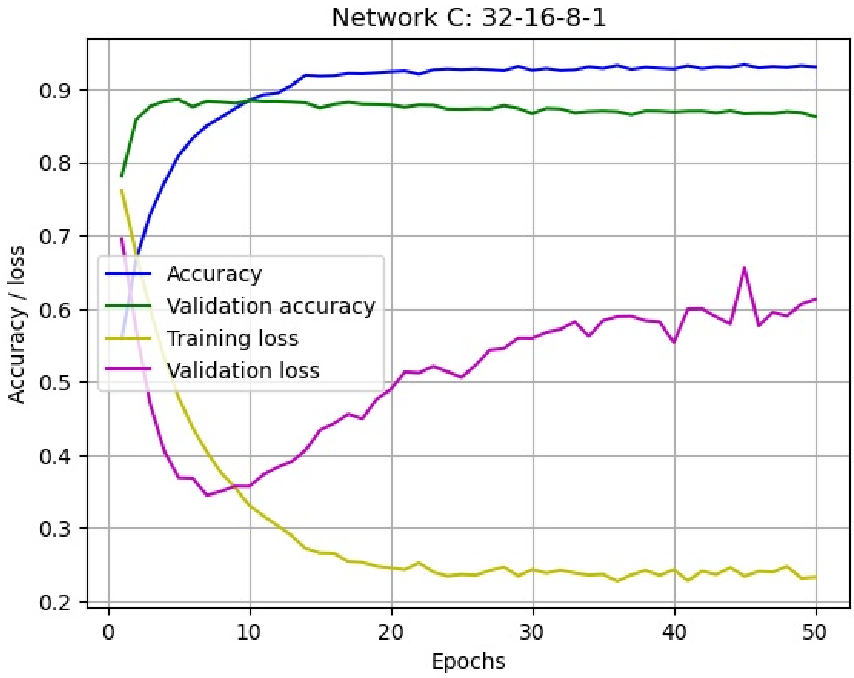
```
Training 8-4-2-1
Model: "sequential_72"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_281 (Dense)           (None, 8)                 80008     
                                                                 
 dropout_183 (Dropout)       (None, 8)                 0         
                                                                 
 dense_282 (Dense)           (None, 4)                 36        
                                                                 
 dropout_184 (Dropout)       (None, 4)                 0         
                                                                 
 dense_283 (Dense)           (None, 2)                 10        
                                                                 
 dropout_185 (Dropout)       (None, 2)                 0         
                                                                 
 dense_284 (Dense)           (None, 1)                 3         
                                                                 
=================================================================
Total params: 80057 (312.72 KB)
Trainable params: 80057 (312.72 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1
Model: "sequential_73"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_285 (Dense)           (None, 16)                160016    
                                                                 
 dropout_186 (Dropout)       (None, 16)                0         
                                                                 
 dense_286 (Dense)           (None, 8)                 136       
                                                                 
 dropout_187 (Dropout)       (None, 8)                 0         
                                                                 
 dense_287 (Dense)           (None, 4)                 36        
                                                                 
 dropout_188 (Dropout)       (None, 4)                 0         
                                                                 
 dense_288 (Dense)           (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 32-16-8-1
Model: "sequential_74"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_289 (Dense)           (None, 32)                320032    
                                                                 
 dropout_189 (Dropout)       (None, 32)                0         
                                                                 
 dense_290 (Dense)           (None, 16)                528       
                                                                 
 dropout_190 (Dropout)       (None, 16)                0         
                                                                 
 dense_291 (Dense)           (None, 8)                 136       
                                                                 
 dropout_191 (Dropout)       (None, 8)                 0         
                                                                 
 dense_292 (Dense)           (None, 1)                 9         
                                                                 
=================================================================
Total params: 320705 (1.22 MB)
Trainable params: 320705 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [79]:

```
#declare networks according to "histories[model_name]"
net_a, net_b, net_c = '8-4-2-1', '16-8-4-1', '32-16-8-1'

plot_net_a(acc=histories[net_a].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_a].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_a].history['loss'], label_c="Training loss",
            loss=histories[net_a].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation

Networks of interest:

- net_a (8-4-2-1)

The network is still not powerful enough

- net_b (16-8-4-1)

The validation loss drops much faster than the training loss, but then diverges at around epoch 35. Maybe it looks like it is slightly overfitting. The accuracy performance for both training and validation looks promising.

- net_c (32-16-8-1)

The network is still overfitting.

Reduce batch size

So far, we have specifically been working with batch size 512, lets see if we can get some more favorable results be decreasing the batch size number.

We will set the network size to a static net_b (16-8-4-1), and use i to run sequentially larger the batch sizes.

In [92]:
```
histories = {}
model.reset_states()
## use i to run different batch sizes, adding batch size to end of model name
for i in [64, 128, 256, 512, 1024]:
    model = build_model(16, 8, 4 , 1)
    model_name = str(16) + '-' + str(8) + '-' + str(4) + '-' + str(1) + '-|' + str(i)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=50,
                        batch_size=i,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

```
Training 16-8-4-1-|64
Model: "sequential_86"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_337 (Dense)           (None, 16)                160016    
                                                                 
 dropout_225 (Dropout)       (None, 16)                0         
                                                                 
 dense_338 (Dense)           (None, 8)                 136       
                                                                 
 dropout_226 (Dropout)       (None, 8)                 0         
                                                                 
 dense_339 (Dense)           (None, 4)                 36        
                                                                 
 dropout_227 (Dropout)       (None, 4)                 0         
                                                                 
 dense_340 (Dense)           (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1-|128
Model: "sequential_87"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_341 (Dense)           (None, 16)                160016    
                                                                 
 dropout_228 (Dropout)       (None, 16)                0         
                                                                 
 dense_342 (Dense)           (None, 8)                 136       
                                                                 
 dropout_229 (Dropout)       (None, 8)                 0         
                                                                 
 dense_343 (Dense)           (None, 4)                 36        
                                                                 
 dropout_230 (Dropout)       (None, 4)                 0         
                                                                 
 dense_344 (Dense)           (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
```

```
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1-|256
Model: "sequential_88"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_345 (Dense)           (None, 16)                160016    
                                                                 
 dropout_231 (Dropout)       (None, 16)                0         
                                                                 
 dense_346 (Dense)           (None, 8)                 136       
                                                                 
 dropout_232 (Dropout)       (None, 8)                 0         
                                                                 
 dense_347 (Dense)           (None, 4)                 36        
                                                                 
 dropout_233 (Dropout)       (None, 4)                 0         
                                                                 
 dense_348 (Dense)           (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1-|512
Model: "sequential_89"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_349 (Dense)           (None, 16)                160016    
                                                                 
 dropout_234 (Dropout)       (None, 16)                0         
                                                                 
 dense_350 (Dense)           (None, 8)                 136       
                                                                 
 dropout_235 (Dropout)       (None, 8)                 0         
                                                                 
 dense_351 (Dense)           (None, 4)                 36        
                                                                 
 dropout_236 (Dropout)       (None, 4)                 0         
                                                                 
 dense_352 (Dense)           (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1-|1024
Model: "sequential_90"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_353 (Dense)           (None, 16)                160016    
                                                                 
 dropout_237 (Dropout)       (None, 16)                0         
                                                                 
 dense_354 (Dense)           (None, 8)                 136       
                                                                 
 dropout_238 (Dropout)       (None, 8)                 0         
                                                                 
 dense_355 (Dense)           (None, 4)                 36        
                                                                 
 dropout_239 (Dropout)       (None, 4)                 0         
                                                                 
 dense_356 (Dense)           (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [88]:

```
history_dict = history.history
history_dict.keys()
```

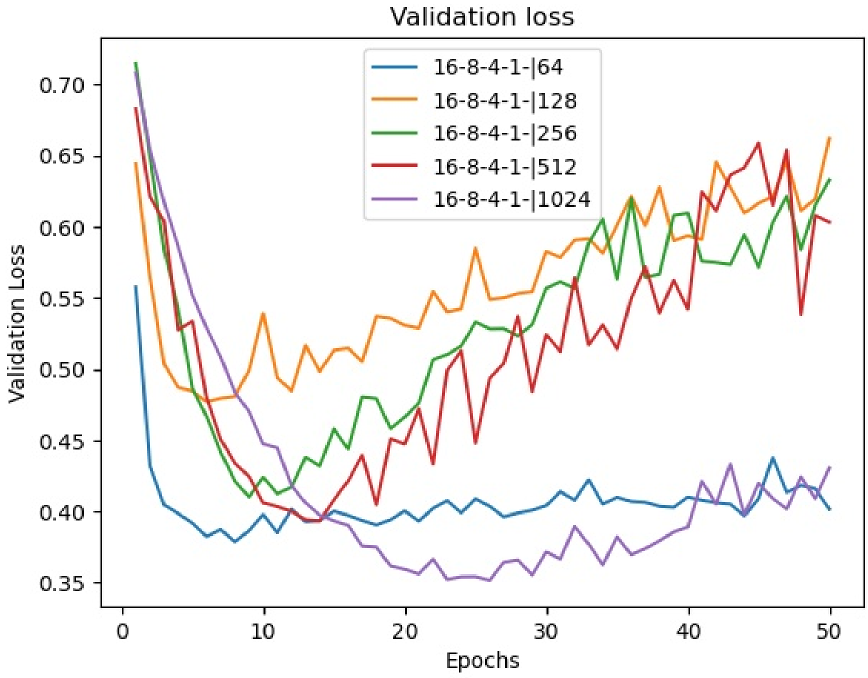
Out[88]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

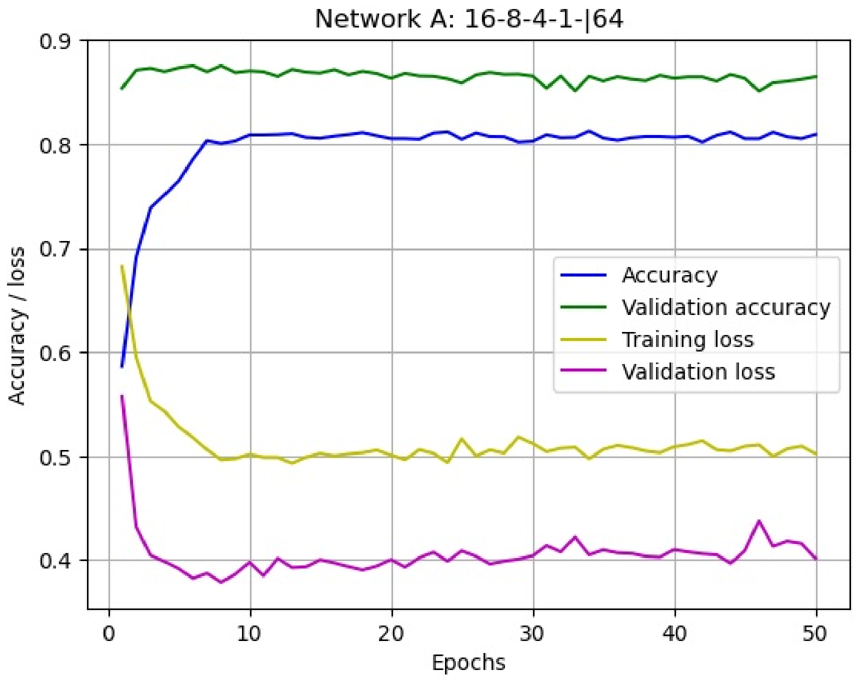
In [93]:

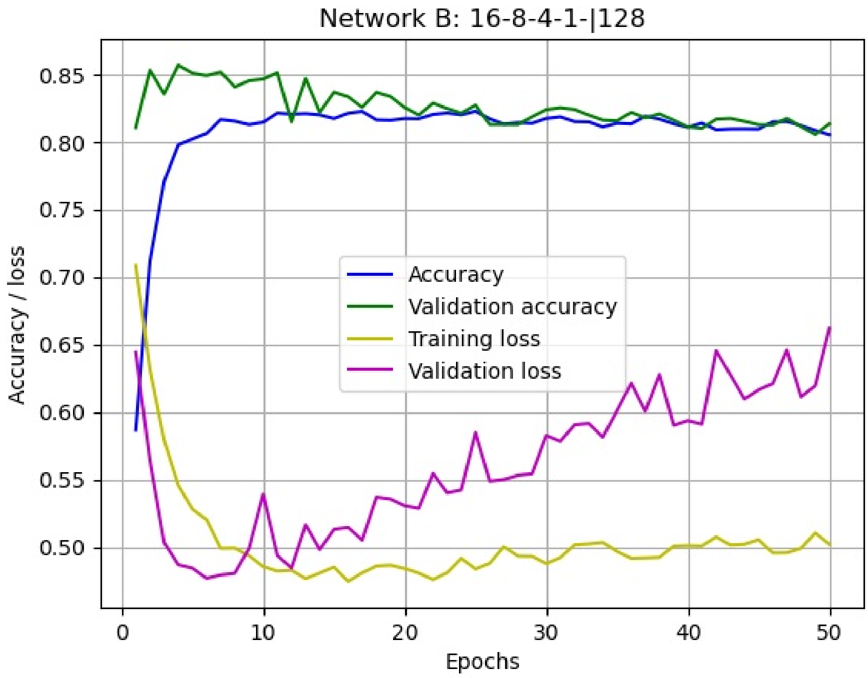
```
#declare networks according to "histories[model_name]"
net_a, net_b, net_c, net_d, net_e = '16-8-4-1-|64', '16-8-4-1-|128', '16-8-4-1-|256', '16-8-4-1-|512', '16-8-4-
#plotting validation loss
plot_loss_comparison(loss_a=histories[net_a].history['val_loss'], label_a=net_a,
                     loss_b=histories[net_b].history['val_loss'], label_b=net_b,
                     loss_c=histories[net_c].history['val_loss'], label_c=net_c,
                     loss_d=histories[net_d].history['val_loss'], label_d=net_d,
```

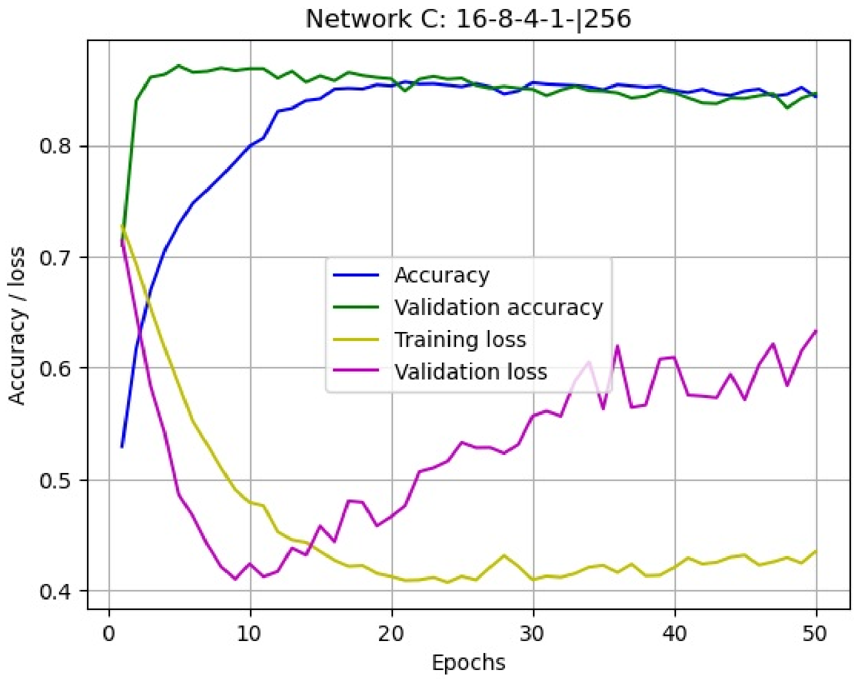
```
                         loss_e=histories[net_e].history['val_loss'], label_e=net_e,
                            y_label='Validation Loss')
```



In [94]:
```
plot_net_a(acc=histories[net_a].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_a].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_a].history['loss'], label_c="Training loss",
            loss=histories[net_a].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation

So, it looks like the larger the batch size, the easier it is for the network to overfit. But reducing the batch size looks to have a noticeable difference in the validation loss, specifically 64 and 128 as we can see it drops noticeably.

Lets try to reduce the dropout to (0.4) and also see if we can get a better result from an even smaller batch size.

In [18]:
```
# define model
def build_model(layer_1_units, layer_2_units, layer_3_units, layer_4_units):
    model = models.Sequential()
    model.add(layers.Dense(layer_1_units, kernel_regularizer=regularizers.l2(0.001), activation='relu', input_s
    model.add(layers.Dropout(0.4))
    model.add(layers.Dense(layer_2_units, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
    model.add(layers.Dropout(0.4))
```

```python
    model.add(layers.Dense(layer_3_units, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
    model.add(layers.Dropout(0.4))
    model.add(layers.Dense(layer_4_units, activation='sigmoid'))
    model.compile(optimizer='rmsprop',
        loss='binary_crossentropy',
        metrics=['accuracy'])
    return model

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        c = ['\b|', '\b/', '\b-', '\b\\']
        print(c[epoch % 4], end='')
    def on_epoch_end(self, epoch, logs=None):
        print('\b', end='')
```

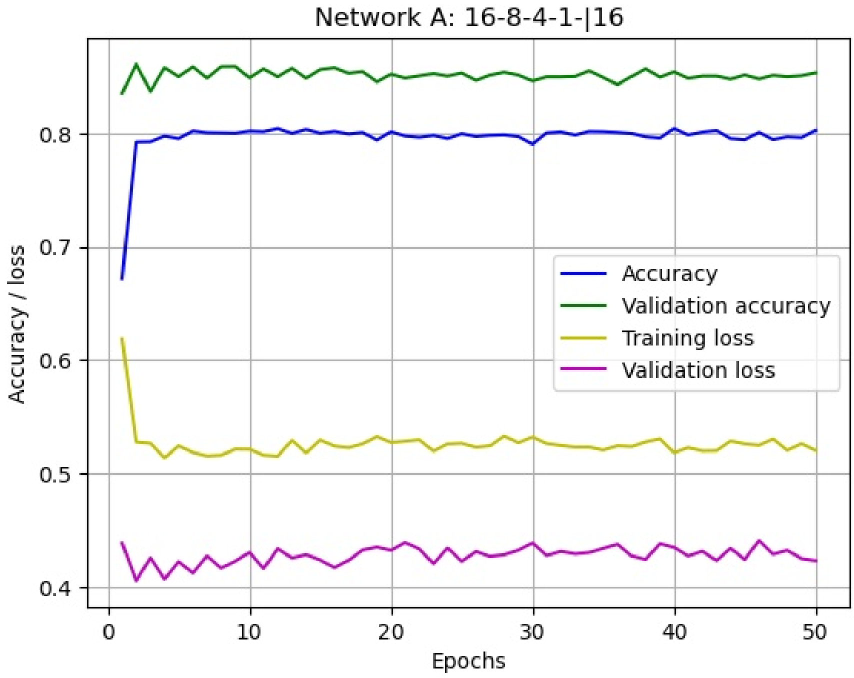
In [19]:

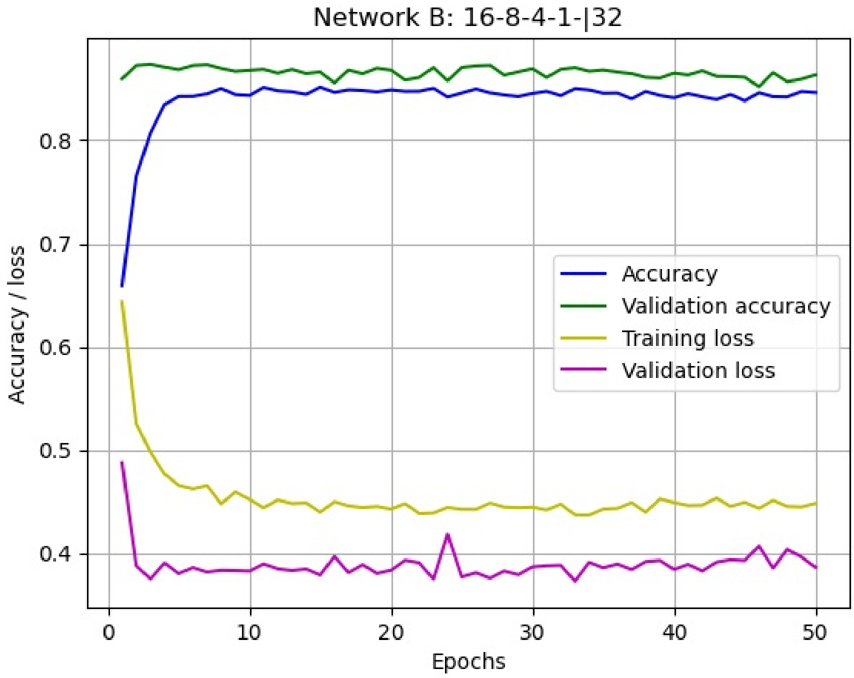
```python
histories = {}
model.reset_states()
## use i to run different batch sizes, adding batch size to end of model name
for i in [16, 32, 64]:
    model = build_model(16, 8, 4 , 1)
    model_name = str(16) + '-' + str(8) + '-' + str(4) + '-' + str(1) + '-|' + str(i)
    print('Training', model_name)
    history = model.fit(partial_x_train,
                       partial_y_train,
                        epochs=50,
                        batch_size=i,
                        validation_data=(x_val, y_val),
                        verbose = 0,
                        callbacks = [CustomCallback()])
    histories[model_name] = history
    model.summary()
```

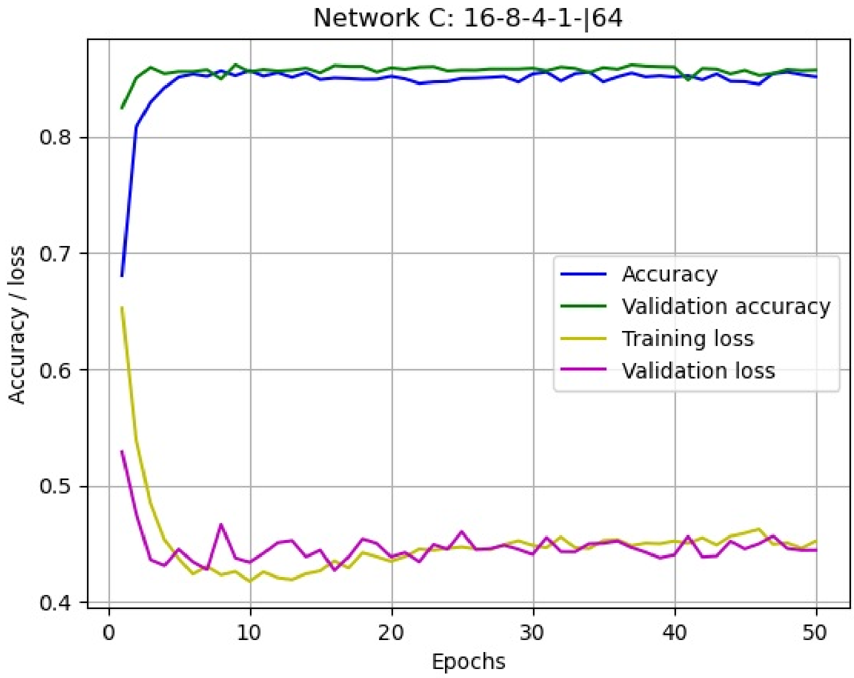
```
Training 16-8-4-1-|16
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_38 (Dense)            (None, 16)                160016    
                                                                 
 dropout_27 (Dropout)        (None, 16)                0         
                                                                 
 dense_39 (Dense)            (None, 8)                 136       
                                                                 
 dropout_28 (Dropout)        (None, 8)                 0         
                                                                 
 dense_40 (Dense)            (None, 4)                 36        
                                                                 
 dropout_29 (Dropout)        (None, 4)                 0         
                                                                 
 dense_41 (Dense)            (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1-|32
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_42 (Dense)            (None, 16)                160016    
                                                                 
 dropout_30 (Dropout)        (None, 16)                0         
                                                                 
 dense_43 (Dense)            (None, 8)                 136       
                                                                 
 dropout_31 (Dropout)        (None, 8)                 0         
                                                                 
 dense_44 (Dense)            (None, 4)                 36        
                                                                 
 dropout_32 (Dropout)        (None, 4)                 0         
                                                                 
 dense_45 (Dense)            (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Training 16-8-4-1-|64
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_46 (Dense)            (None, 16)                160016    
                                                                 
 dropout_33 (Dropout)        (None, 16)                0         
                                                                 
 dense_47 (Dense)            (None, 8)                 136       
                                                                 
 dropout_34 (Dropout)        (None, 8)                 0         
                                                                 
 dense_48 (Dense)            (None, 4)                 36        
                                                                 
 dropout_35 (Dropout)        (None, 4)                 0         
                                                                 
 dense_49 (Dense)            (None, 1)                 5         
                                                                 
=================================================================
Total params: 160193 (625.75 KB)
Trainable params: 160193 (625.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [20]:

```python
#declare networks according to "histories[model_name]"
net_a, net_b, net_c = '16-8-4-1-|16', '16-8-4-1-|32', '16-8-4-1-|64'

plot_net_a(acc=histories[net_a].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_a].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_a].history['loss'], label_c="Training loss",
            loss=histories[net_a].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_b(acc=histories[net_b].history['accuracy'], label_a="Accuracy",
           val_acc=histories[net_b].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_b].history['loss'], label_c="Training loss",
            loss=histories[net_b].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
plot_net_c(acc=histories[net_c].history['accuracy'], label_a="Accuracy",
            val_acc=histories[net_c].history['val_accuracy'], label_b="Validation accuracy",
            tloss=histories[net_c].history['loss'], label_c="Training loss",
            loss=histories[net_c].history['val_loss'], label_d="Validation loss", y_label='Accuracy / loss')
```







Interpretation and best network candidate

So far, it looks like net_c: 16-8-4-1-|64 is the best candidate so far. It exhibits good accuracy and the training and validation loss is stable and does not diverge even after 50 epochs.

---

8. Testing

Network net_c: 16-8-4-1-|64 is the best performing to solve the neural network solution. The network shows good accuracy for both training and validation, although not reaching 90%. When comparing the training and validation loss, we can see that both drops quickly and stabilizes without diverging.

So, we will save the parameters and retrain a new model for the optimal number of epochs and evaluate on the test data. From looking at the plots, we can see that around 7-10 epochs should be optimal.

Setting parameters for model

In [25]:

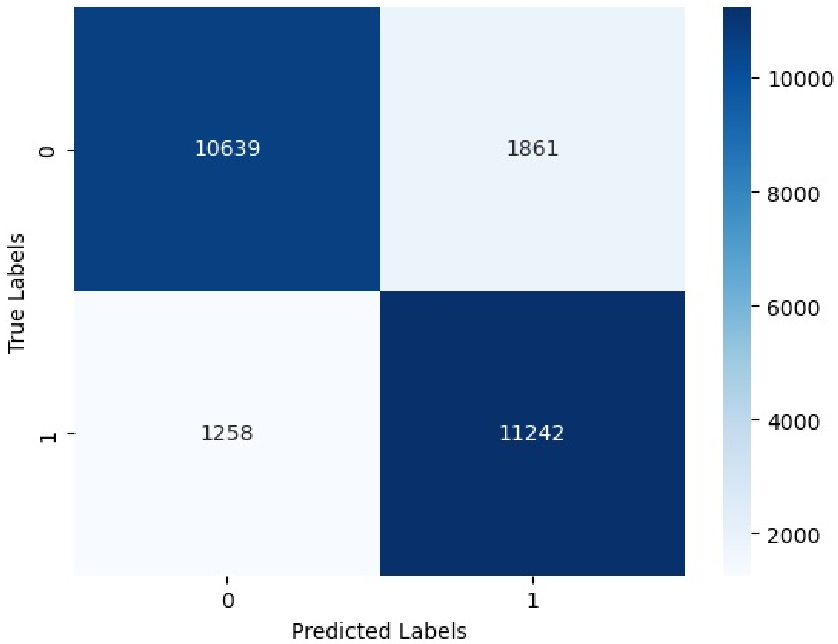
```
# define model Network net_c: 16-8-4-1-|64
model.reset_states()
model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001), activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(8, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(4, kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['accuracy'])
history = model.fit(x_train, y_train,
                    epochs=7,
                    batch_size=64,
                    verbose=0)
results = model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 1s 2ms/step - loss: 0.3723 - accuracy: 0.8752
```

In [27]:

```
#confusion matrix
y_pred = (model.predict(x_test)[:, 0] > 0.5).astype("int32")
conf_matrix = tf.math.confusion_matrix(labels=y_test, predictions=y_pred)
#SNS heatmap
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
782/782 [==============================] - 1s 1ms/step
```



In [28]:

```
classification_report = classification_report(y_test, y_pred)
print(classification_report)
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.85      0.87     12500
         1.0       0.86      0.90      0.88     12500

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000
```

Evaluation results

Accuracy and loss:

We can see that the accuracy achieved was 0.8752, with loss being 0.3723. This is much better than the baseline model we created in the beginning.

Other performance metrics:

After creating the confusion matrix with the tf.math.confusion_matrix and plotting it with SNS, we are able to visualize the performance. The Classification report also provided us with more metrics like precision, recall and f1-score.

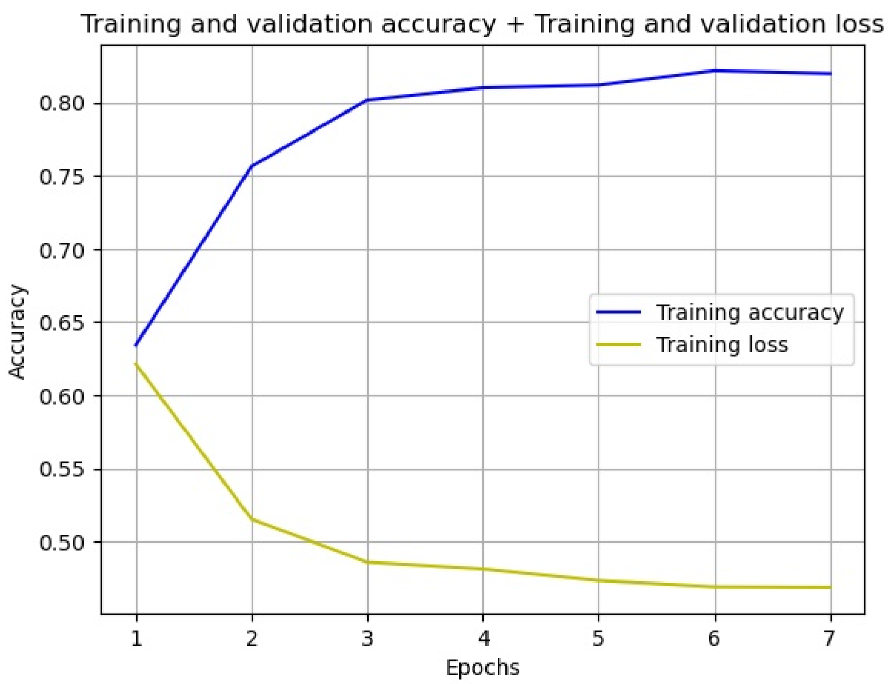
Overall, these results are quite favorable, but it seems that the model is better at predicting positive movie reviews as seen in the 0.90 recall score.

But when looking at precision, positive scores worse as it is 0.86. This is due to the fact that there is a much higher number of false positives as seen in the confusion matrix (1861, top right). The precision for false labels performed better at 0.89 as the model predicted less false negatives.

F1-score is very similar for both positive and negative labels with only 0.01 difference.

Plot accuracy and loss for training model

In [31]:
```
#get history
history_dict = history.history
history_dict.keys()


# Let's plot training and validation accuracy as well as loss.
def plot_history(history):
    accuracy = history.history['accuracy']
    loss = history.history['loss']
    epochs = range(1,len(accuracy) + 1)
    # Plot accuracy
    plt.figure(1)
    plt.plot(epochs, accuracy, 'b', label='Training accuracy')
    plt.plot(epochs, loss, 'y', label='Training loss')
    plt.title('Training accuracy + Training loss')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.grid()
    plt.legend()
plot_history(history)
```



Interpretation

From looking at the plot, we can see that 7 epochs are a good choice for training the model and it is the peak for both accuracy and loss. It is not clear if adding more epochs will increase or decrease the performance of the model.

Conclusion

The steps outline in the DLWP universal workflow of machine learning 4.5 has been followed. A baseline model was established then the model was expanded to create overfitting. Afterwards, the model was regularized and tuned with hyperparameters to find the best fitting model for the solution.