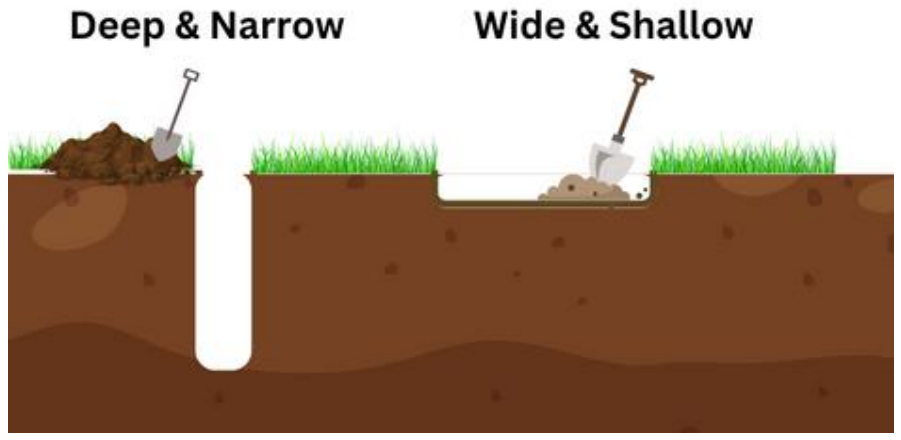
Comparing Neural Network Performance- Deep and Narrow vs Shallow and Wide

By- **Parth Ghumare**
Graduate Student Assistant,
The University of Texas at Dallas

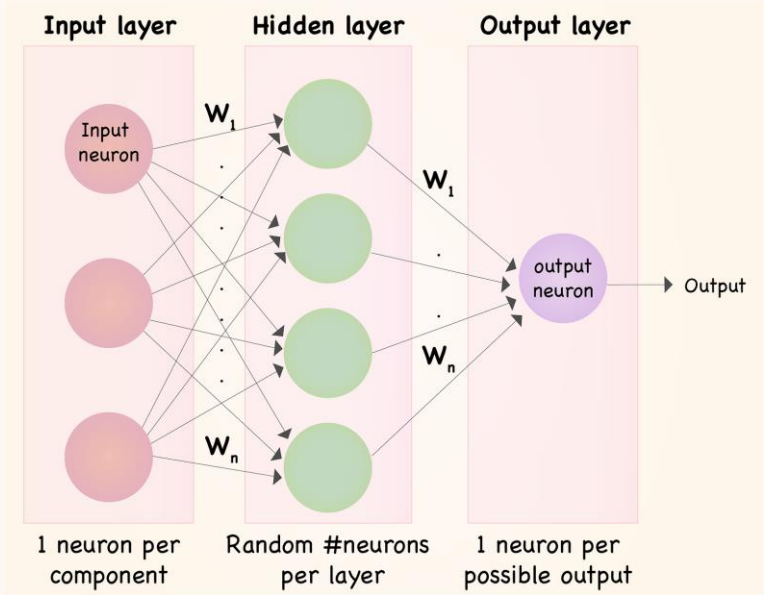
Executive Summary

- The goal of this project is to evaluate the performance of deep and narrow versus shallow and wide neural networks on specific tasks to determine which architecture offers the best trade-off between accuracy, training time, and computational efficiency.

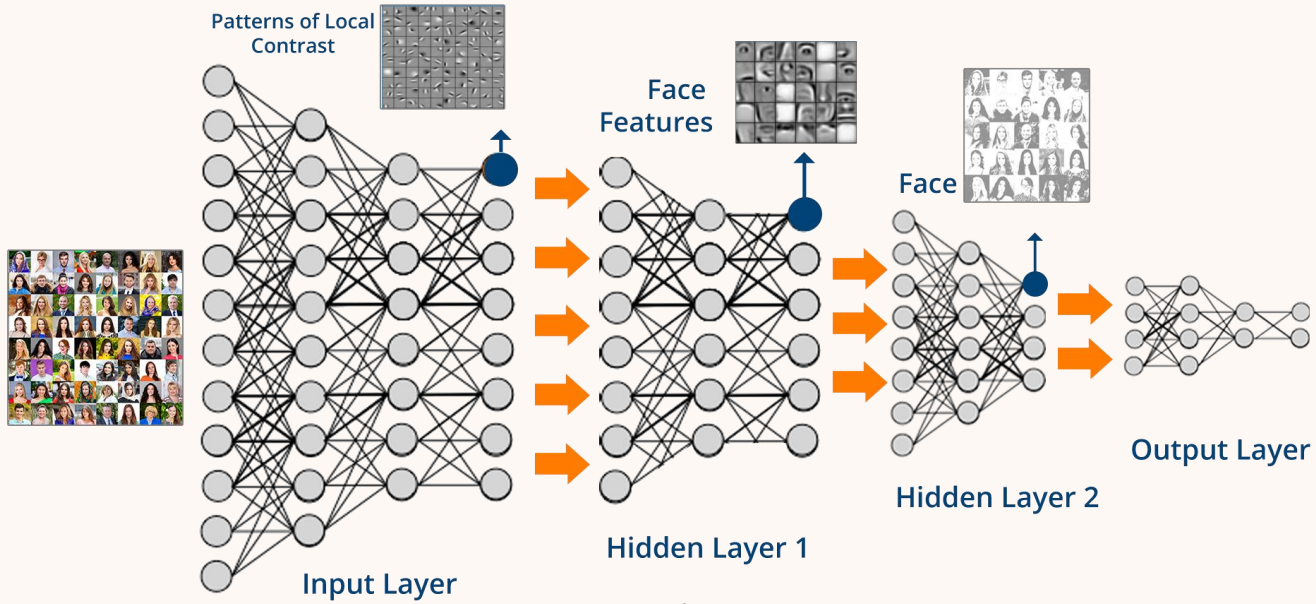


Primary Understanding

- The world of neural networks presents an intricate panorama of architectures, methodologies, and applications. Neural networks are like the brains of artificial intelligence, designed to recognize patterns and make decisions based on data. They are divided into two main types: deep and shallow. While both are instrumental in tasks across machine learning domains, distinguishing between the two can unlock nuanced perspectives on their design principles and potential applicability.



Shallow and Wide NN

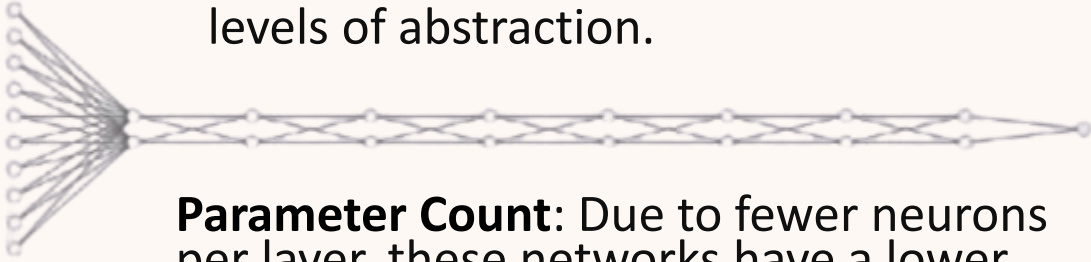


Deep and Narrow NN

Primary Understanding of Deep and Narrow NN

- **Definition**: A "deep" neural network refers to a network that contains a substantial number of layers. These layers include hidden layers between the input and output layers.

- **Layer Depth**: These networks have a significant number of layers, which is characteristic of deep neural networks. The depth allows for the processing of data through multiple levels of abstraction.



- **Neurons Per Layer**: Despite the depth, each layer has a relatively small number of neurons, which defines the 'narrow' aspect. This contrasts with wide networks that have many neurons in each layer.

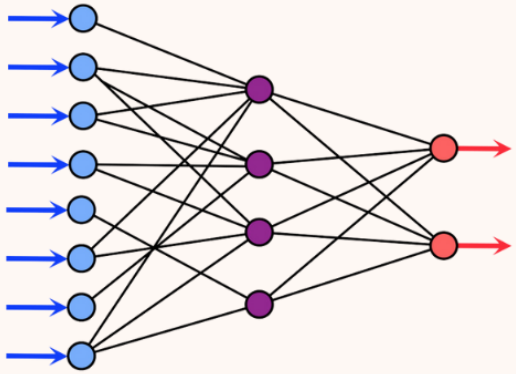
Parameter Count: Due to fewer neurons per layer, these networks have a lower number of parameters compared to wide networks, which can lead to less computational demand during training and inference, and potentially faster learning, albeit possibly at the expense of the network's ability to capture more complex patterns.

Primary Understanding of Shallow and Wide NN

- **Definition**: "shallow" neural network refers to a network with a limited number of layers, typically just one or two hidden layers between the input and output layers.

- **Parameter Count:** With many neurons in each layer, wide networks have a high number of parameters, which can lead to greater computational demand during training and inference, but can also allow for the capturing of complex patterns without needing deeper architectures.

- **Layer Depth**: These networks have a minimal number of layers, often just the input layer, one or two hidden layers, and the output layer.



- **Neurons Per Layer**: In each of the few layers, there is a high number of neurons, which gives the network its "wide" attribute.

Model Design

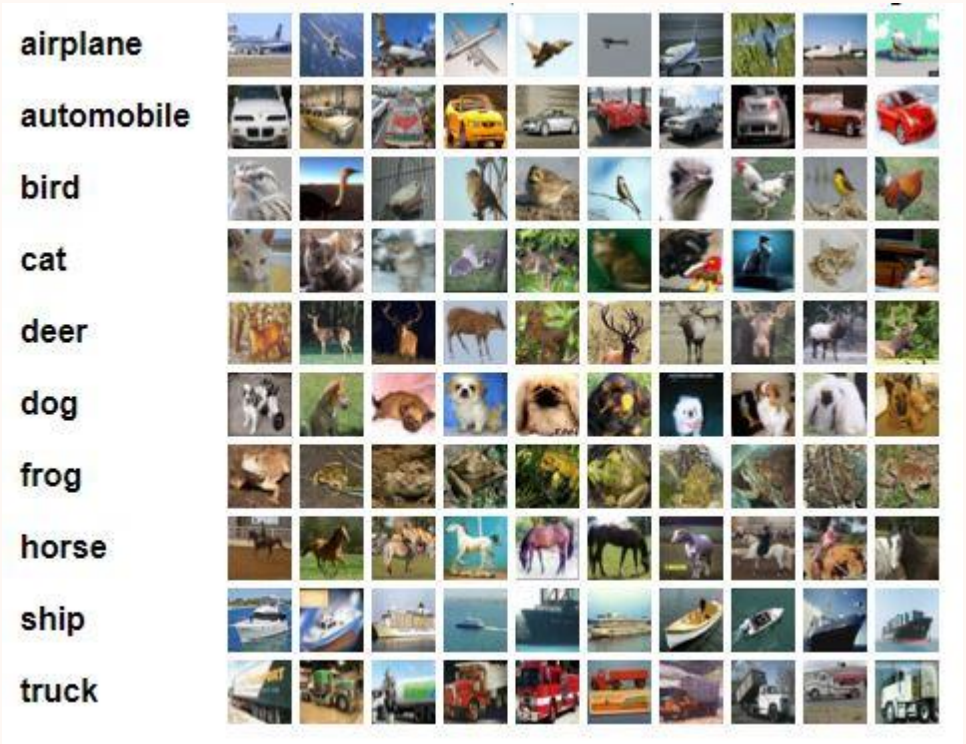
- Create two types of convolutional neural network (CNN) models and then comparing two different architectures of neural networks, one that is deep but narrow, and the other that is wider but shallower. The key aspect of this comparison is to keep the total number of trainable parameters approximately the same in both architectures. This setup will allow us to explore whether the depth or width of a network has a more significant impact on performance, given a constant parameter budget.

- **Deep and Narrow Network**: This will have many layers but fewer parameters per layer.
- **Wide and Shallow Network**: This will have fewer layers but more parameters per layer.

DATA SOURCE1 (IMAGE CLASSIFICATION)

•The CIFAR-10 dataset consists of 60000 32x32 px color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
•Source: https://www.cs.toronto.edu/~kriz/cifar.html

```
[ ] (X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
    X_train.shape

    (50000, 32, 32, 3)

[ ] # Check the shape of the datasets
    print("X_train shape:", X_train.shape)
    print("y_train shape:", y_train.shape)
    print("X_test shape:", X_test.shape)
    print("y_test shape:", y_test.shape)

    X_train shape: (50000, 32, 32, 3)
    y_train shape: (50000, 1)
    X_test shape: (10000, 32, 32, 3)
    y_test shape: (10000, 1)

[ ] classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

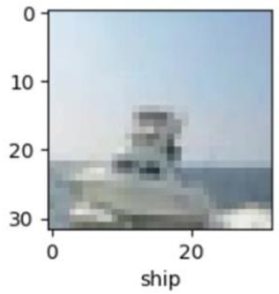


Data Preprocessing

- We'll normalize the pixel values to be 0 to 1 by dividing by 255 (since pixel values range from 0 to 255). For the labels, we'll use one-hot encoding if necessary, depending on the model requirements.

```
[ ] def plot_sample(X, y, index):
        plt.figure(figsize = (15,2))
        plt.imshow(X[index])
        plt.xlabel(classes[y[index]])
```

```
[ ] plot_sample(X_train, y_train, 100)
```

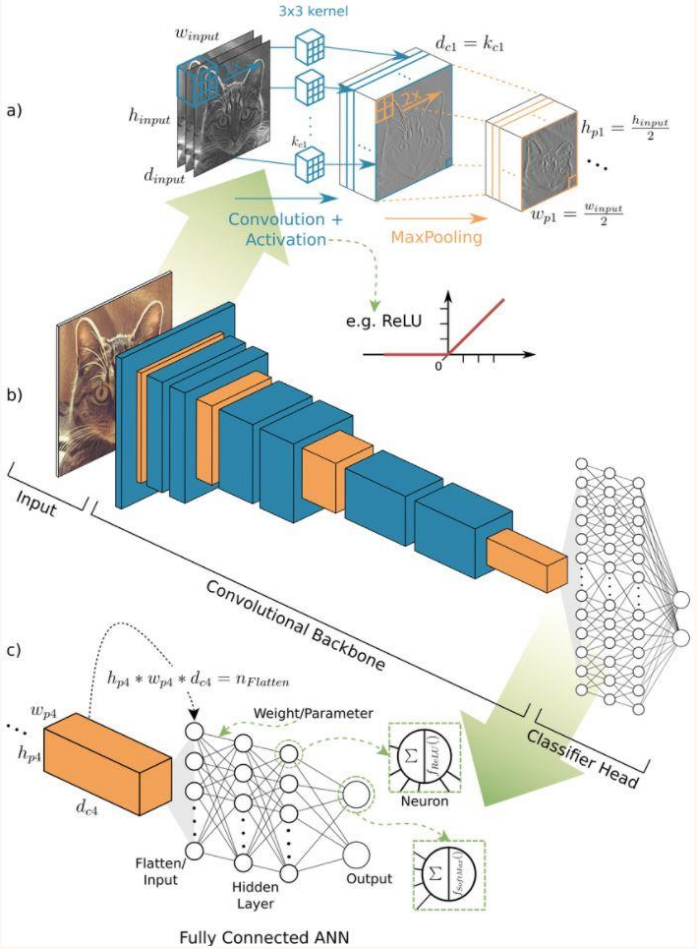



```
[ ] from tensorflow.keras.utils import to_categorical

    # Normalizing
    X_train = X_train/255
    X_test = X_test/255
    # One-Hot-Encoding
    Y_train_en = to_categorical(y_train,10)
    Y_test_en = to_categorical(y_test,10)
```

Deep and Narrow Model (CIFAR10)

```
[ ] # Deep and Narrow Model
    def create_deep_model():
        model = Sequential()
        model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=X_train.shape[1:]))
        model.add(BatchNormalization())
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))
        model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
        model.add(BatchNormalization())
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.3))

        model.add(Flatten())
        model.add(Dense(64, activation='relu'))
        model.add(Dense(10, activation='softmax', kernel_regularizer=l2(0.01)))
        model.compile(loss ='categorical_crossentropy', optimizer ='adam', metrics =['accuracy'])
        return model
```

```
[ ] #summery of Deep and Narrow Model
    deep_model = create_deep_model()
    deep_model.summary()
    history_deep = deep_model.fit(X_train, Y_train_en, epochs = 25,batch_size=128, verbose=1,validation_data=(X_test,Y_test_en))
```



Shallow and Wide Model (CIFAR10)

```
# Shallow and Wide Model
def create_wide_model():
  model = Sequential()
  model.add(Conv2D(128, (3, 3), padding='same', activation='relu', input_shape=X_train.shape[1:]))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(0.2))
  model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(0.2))
  model.add(Flatten())
  model.add(Dense(10, activation='softmax', kernel_regularizer=l2(0.01)))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```

```
[ ] wide_model = create_wide_model()
    wide_model.summary()
    history_wide = wide_model.fit(X_train, Y_train_en, epochs = 25,batch_size=128, verbose=1,validation_data=(X_test,Y_test_en))
```

Comparison CIFAR10

```
=================================================================
Total params: 261770 (1022.54 KB)
Trainable params: 261258 (1020.54 KB)
Non-trainable params: 512 (2.00 KB)
_________________________________________________________________
Epoch 1/25
391/391 [==============================] - 25s 31ms/step - loss: 1.5522 - accuracy: 0.4853 - val_loss: 3.0361 - val_accuracy: 0.2309
Epoch 2/25
391/391 [==============================] - 11s 28ms/step - loss: 0.9998 - accuracy: 0.6607 - val_loss: 0.8817 - val_accuracy: 0.7038
Epoch 3/25
391/391 [==============================] - 11s 29ms/step - loss: 0.8004 - accuracy: 0.7256 - val_loss: 0.7872 - val_accuracy: 0.7372
Epoch 4/25
391/391 [==============================] - 11s 29ms/step - loss: 0.7003 - accuracy: 0.7617 - val_loss: 0.9161 - val_accuracy: 0.6944
Epoch 5/25
391/391 [==============================] - 11s 28ms/step - loss: 0.6274 - accuracy: 0.7890 - val_loss: 0.7464 - val_accuracy: 0.7505
Epoch 6/25
391/391 [==============================] - 11s 28ms/step - loss: 0.5737 - accuracy: 0.8052 - val_loss: 0.6681 - val_accuracy: 0.7776
Epoch 7/25
391/391 [==============================] - 11s 28ms/step - loss: 0.5258 - accuracy: 0.8222 - val_loss: 0.6694 - val_accuracy: 0.7772
Epoch 8/25
391/391 [==============================] - 11s 28ms/step - loss: 0.4872 - accuracy: 0.8359 - val_loss: 0.6836 - val_accuracy: 0.7768
Epoch 9/25
391/391 [==============================] - 11s 28ms/step - loss: 0.4521 - accuracy: 0.8478 - val_loss: 0.7007 - val_accuracy: 0.7752
Epoch 10/25
391/391 [==============================] - 11s 28ms/step - loss: 0.4175 - accuracy: 0.8588 - val_loss: 0.6678 - val_accuracy: 0.7860
Epoch 11/25
391/391 [==============================] - 11s 29ms/step - loss: 0.3880 - accuracy: 0.8705 - val_loss: 0.6205 - val_accuracy: 0.8043
Epoch 12/25
391/391 [==============================] - 11s 29ms/step - loss: 0.3666 - accuracy: 0.8759 - val_loss: 0.6465 - val_accuracy: 0.7964
Epoch 13/25
391/391 [==============================] - 11s 29ms/step - loss: 0.3431 - accuracy: 0.8853 - val_loss: 0.6382 - val_accuracy: 0.7975
Epoch 14/25
391/391 [==============================] - 11s 28ms/step - loss: 0.3216 - accuracy: 0.8935 - val_loss: 0.6547 - val_accuracy: 0.8023
Epoch 15/25
391/391 [==============================] - 11s 28ms/step - loss: 0.3009 - accuracy: 0.8989 - val_loss: 0.6406 - val_accuracy: 0.8083
Epoch 16/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2915 - accuracy: 0.9040 - val_loss: 0.6444 - val_accuracy: 0.8017
Epoch 17/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2694 - accuracy: 0.9119 - val_loss: 0.6547 - val_accuracy: 0.8061
Epoch 18/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2576 - accuracy: 0.9149 - val_loss: 0.6835 - val_accuracy: 0.7977
Epoch 19/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2518 - accuracy: 0.9182 - val_loss: 0.6673 - val_accuracy: 0.8045
Epoch 20/25
391/391 [==============================] - 11s 29ms/step - loss: 0.2395 - accuracy: 0.9200 - val_loss: 0.7180 - val_accuracy: 0.7987
Epoch 21/25
391/391 [==============================] - 11s 29ms/step - loss: 0.2270 - accuracy: 0.9253 - val_loss: 0.6927 - val_accuracy: 0.8057
Epoch 22/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2140 - accuracy: 0.9306 - val_loss: 0.6731 - val_accuracy: 0.8122
Epoch 23/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2067 - accuracy: 0.9335 - val_loss: 0.7700 - val_accuracy: 0.7959
Epoch 24/25
391/391 [==============================] - 11s 28ms/step - loss: 0.2002 - accuracy: 0.9360 - val_loss: 0.6658 - val_accuracy: 0.8171
Epoch 25/25
391/391 [==============================] - 11s 29ms/step - loss: 0.1966 - accuracy: 0.9367 - val_loss: 0.7288 - val_accuracy: 0.7994
```

Deep and Narrow Model

```
=================================================================
Total params: 234122 (914.54 KB)
Trainable params: 233610 (912.54 KB)
Non-trainable params: 512 (2.00 KB)
_________________________________________________________________
Epoch 1/25
391/391 [==============================] - 12s 27ms/step - loss: 1.9822 - accuracy: 0.4645 - val_loss: 1.8602 - val_accuracy: 0.3833
Epoch 2/25
391/391 [==============================] - 10s 24ms/step - loss: 1.3789 - accuracy: 0.6024 - val_loss: 1.4981 - val_accuracy: 0.5623
Epoch 3/25
391/391 [==============================] - 10s 25ms/step - loss: 1.1772 - accuracy: 0.6523 - val_loss: 1.2853 - val_accuracy: 0.6305
Epoch 4/25
391/391 [==============================] - 10s 25ms/step - loss: 1.0599 - accuracy: 0.6828 - val_loss: 1.4249 - val_accuracy: 0.5845
Epoch 5/25
391/391 [==============================] - 10s 24ms/step - loss: 0.9811 - accuracy: 0.7085 - val_loss: 1.1478 - val_accuracy: 0.6596
Epoch 6/25
391/391 [==============================] - 10s 25ms/step - loss: 0.9371 - accuracy: 0.7240 - val_loss: 1.2508 - val_accuracy: 0.6225
Epoch 7/25
391/391 [==============================] - 10s 25ms/step - loss: 0.9086 - accuracy: 0.7338 - val_loss: 1.6896 - val_accuracy: 0.5316
Epoch 8/25
391/391 [==============================] - 9s 24ms/step - loss: 0.8826 - accuracy: 0.7448 - val_loss: 1.0068 - val_accuracy: 0.7068
Epoch 9/25
391/391 [==============================] - 10s 25ms/step - loss: 0.8559 - accuracy: 0.7569 - val_loss: 0.9983 - val_accuracy: 0.7158
Epoch 10/25
391/391 [==============================] - 10s 25ms/step - loss: 0.8400 - accuracy: 0.7646 - val_loss: 1.0081 - val_accuracy: 0.7164
Epoch 11/25
391/391 [==============================] - 9s 24ms/step - loss: 0.8321 - accuracy: 0.7717 - val_loss: 1.1418 - val_accuracy: 0.6892
Epoch 12/25
391/391 [==============================] - 10s 25ms/step - loss: 0.8268 - accuracy: 0.7763 - val_loss: 0.9786 - val_accuracy: 0.7181
Epoch 13/25
391/391 [==============================] - 10s 25ms/step - loss: 0.8206 - accuracy: 0.7810 - val_loss: 1.4985 - val_accuracy: 0.6136
Epoch 14/25
391/391 [==============================] - 10s 24ms/step - loss: 0.8067 - accuracy: 0.7862 - val_loss: 0.9706 - val_accuracy: 0.7331
Epoch 15/25
391/391 [==============================] - 10s 24ms/step - loss: 0.8039 - accuracy: 0.7900 - val_loss: 1.0724 - val_accuracy: 0.7000
Epoch 16/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7947 - accuracy: 0.7942 - val_loss: 1.0038 - val_accuracy: 0.7331
Epoch 17/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7944 - accuracy: 0.7940 - val_loss: 0.9814 - val_accuracy: 0.7397
Epoch 18/25
391/391 [==============================] - 9s 24ms/step - loss: 0.7750 - accuracy: 0.8021 - val_loss: 1.0599 - val_accuracy: 0.7092
Epoch 19/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7746 - accuracy: 0.8037 - val_loss: 1.0266 - val_accuracy: 0.7347
Epoch 20/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7681 - accuracy: 0.8060 - val_loss: 1.0978 - val_accuracy: 0.7036
Epoch 21/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7610 - accuracy: 0.8090 - val_loss: 1.1713 - val_accuracy: 0.6998
Epoch 22/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7562 - accuracy: 0.8099 - val_loss: 0.9909 - val_accuracy: 0.7402
Epoch 23/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7495 - accuracy: 0.8151 - val_loss: 0.9867 - val_accuracy: 0.7420
Epoch 24/25
391/391 [==============================] - 10s 24ms/step - loss: 0.7384 - accuracy: 0.8200 - val_loss: 0.9734 - val_accuracy: 0.7509
Epoch 25/25
391/391 [==============================] - 10s 25ms/step - loss: 0.7439 - accuracy: 0.8186 - val_loss: 0.9874 - val_accuracy: 0.7484
```

Shallow and Wide Model

DATA SOURCE2 (IMDB)

•The IMDB dataset which is a standard dataset used for binary sentiment classification. As of April 2023, this dataset contains 50,000 reviews split evenly into 25,000 reviews for training and 25,000 for testing. Each of these sets has an equal number of positive and negative reviews.

•Source: https://keras.io/api/datasets/imdb/

```
[ ]  #Load and Prepare the IMDb Dataset
     vocab_size = 10000
     max_review_length = 500
     (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
     x_train = pad_sequences(x_train, maxlen=max_review_length)
     x_test = pad_sequences(x_test, maxlen=max_review_length)
```

```
[ ]  # Get the word index and display how words are mapped to integers
     word_index = imdb.get_word_index()
     print({k: word_index[k] for k in list(word_index)[:5]})
```

```
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951}
```

```
[ ]  # Print the size of each dataset
     print("Training Data:")
     print("x_train shape:", x_train.shape)
     print("y_train shape:", y_train.shape)

     print("\nTesting Data:")
     print("x_test shape:", x_test.shape)
     print("y_test shape:", y_test.shape)
```

```
Training Data:
x_train shape: (25000, 500)
y_train shape: (25000,)

Testing Data:
x_test shape: (25000, 500)
y_test shape: (25000,)
```

Data Preprocessing IMDB

- The IMDB dataset in Keras is preprocessed for sentiment analysis. Each movie review is tokenized and encoded as a sequence of word indexes, represented as integers. Additionally, every review in the training dataset is labeled as either positive or negative

```
[ ] # Display the structure of the first few reviews
    for i in range(2):
        print(f"Review {i + 1}:")
        print("Words (Encoded):", x_train[i])
        print("Label:", "Positive" if y_train[i] == 1 else "Negative")
        print()
```

```
Review 1:
Words (Encoded):    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    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]
Label: Positive
```

Deep and Narrow Model (IMDB)

```python
def deep_narrow_model():
    model = Sequential()
    model.add(Embedding(vocab_size, 300, input_length=max_review_length))

    model.add(Conv1D(64, kernel_size=3, activation='relu'))
    model.add(GlobalAveragePooling1D())

    # Increasing model complexity and reducing regularization
    model.add(Dense(32, activation='relu')) # Increased neurons
    model.add(BatchNormalization())
    model.add(Dropout(0.5)) # Reduced dropout

    model.add(Dense(32, activation='relu')) # Increased neurons
    model.add(BatchNormalization())
    model.add(Dropout(0.5)) # Reduced dropout
    # Output layer with sigmoid activation
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```python
deep_model = deep_narrow_model()
deep_model.summary()
history_deep = deep_model.fit(x_train, y_train, epochs=15,batch_size=128, verbose=1, validation_data=(x_test, y_test))
```

Shallow and Wide Model (IMDB)

```
[ ] def wide_shallow_model():
        model = Sequential()
        model.add(Embedding(vocab_size, 300, input_length=max_review_length)) # Wide embedding layer
        model.add(GlobalAveragePooling1D())

        model.add(Dense(512, activation='relu'))
        model.add(BatchNormalization()) # Added Batch Normalization
        model.add(Dropout(0.5))
        model.add(Dense(1, activation='sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

        return model
```

```
[ ] # Wide and Shallow Model
    wide_model = wide_shallow_model()
    wide_model.summary()
    history_wide= wide_model.fit(x_train, y_train, epochs=15,batch_size=128, verbose=1, validation_data=(x_test, y_test))
```

Comparison IMDB

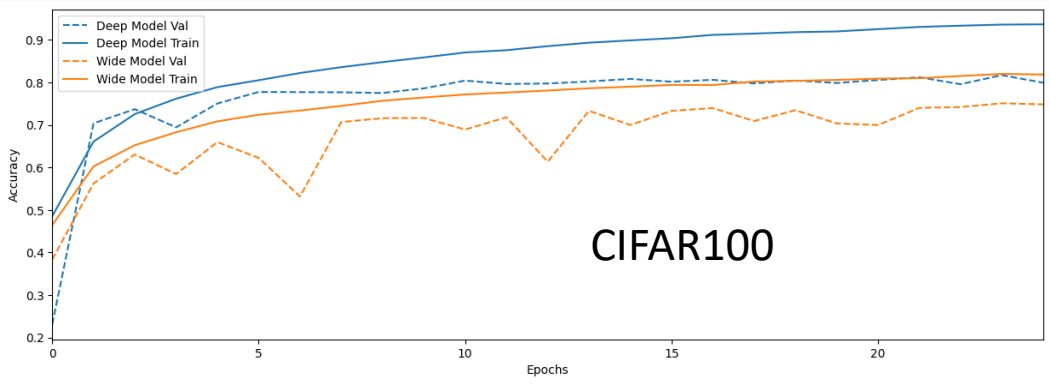
```
=================================================================
Total params: 3061089 (11.68 MB)
Trainable params: 3060961 (11.68 MB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
Epoch 1/15
196/196 [==============================] - 47s 166ms/step - loss: 0.5214 - accuracy: 0.7538 - val_loss: 0.5797 - val_accuracy: 0.7392
Epoch 2/15
196/196 [==============================] - 25s 127ms/step - loss: 0.2514 - accuracy: 0.9048 - val_loss: 0.3692 - val_accuracy: 0.8611
Epoch 3/15
196/196 [==============================] - 18s 93ms/step - loss: 0.1853 - accuracy: 0.9349 - val_loss: 0.3172 - val_accuracy: 0.8680
Epoch 4/15
196/196 [==============================] - 15s 78ms/step - loss: 0.1460 - accuracy: 0.9506 - val_loss: 0.3714 - val_accuracy: 0.8669
Epoch 5/15
196/196 [==============================] - 13s 65ms/step - loss: 0.1166 - accuracy: 0.9610 - val_loss: 0.4067 - val_accuracy: 0.8669
Epoch 6/15
196/196 [==============================] - 12s 60ms/step - loss: 0.0901 - accuracy: 0.9720 - val_loss: 0.5261 - val_accuracy: 0.8548
Epoch 7/15
196/196 [==============================] - 9s 45ms/step - loss: 0.0648 - accuracy: 0.9801 - val_loss: 0.6151 - val_accuracy: 0.8559
Epoch 8/15
196/196 [==============================] - 9s 44ms/step - loss: 0.0569 - accuracy: 0.9814 - val_loss: 0.7612 - val_accuracy: 0.8304
Epoch 9/15
196/196 [==============================] - 9s 47ms/step - loss: 0.0434 - accuracy: 0.9860 - val_loss: 0.6647 - val_accuracy: 0.8580
Epoch 10/15
196/196 [==============================] - 8s 40ms/step - loss: 0.0408 - accuracy: 0.9872 - val_loss: 0.7588 - val_accuracy: 0.8551
Epoch 11/15
196/196 [==============================] - 8s 39ms/step - loss: 0.0361 - accuracy: 0.9880 - val_loss: 0.8349 - val_accuracy: 0.8518
Epoch 12/15
196/196 [==============================] - 8s 42ms/step - loss: 0.0346 - accuracy: 0.9891 - val_loss: 0.7869 - val_accuracy: 0.8583
Epoch 13/15
196/196 [==============================] - 7s 34ms/step - loss: 0.0375 - accuracy: 0.9870 - val_loss: 0.9361 - val_accuracy: 0.8545
Epoch 14/15
196/196 [==============================] - 6s 32ms/step - loss: 0.0273 - accuracy: 0.9913 - val_loss: 0.9176 - val_accuracy: 0.8579
Epoch 15/15
196/196 [==============================] - 7s 36ms/step - loss: 0.0239 - accuracy: 0.9930 - val_loss: 0.8686 - val_accuracy: 0.8595
```

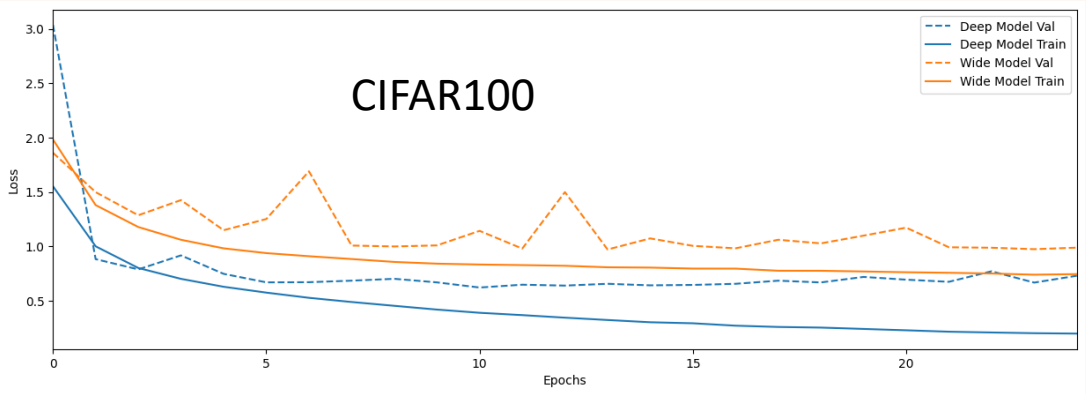
Deep and Narrow Model

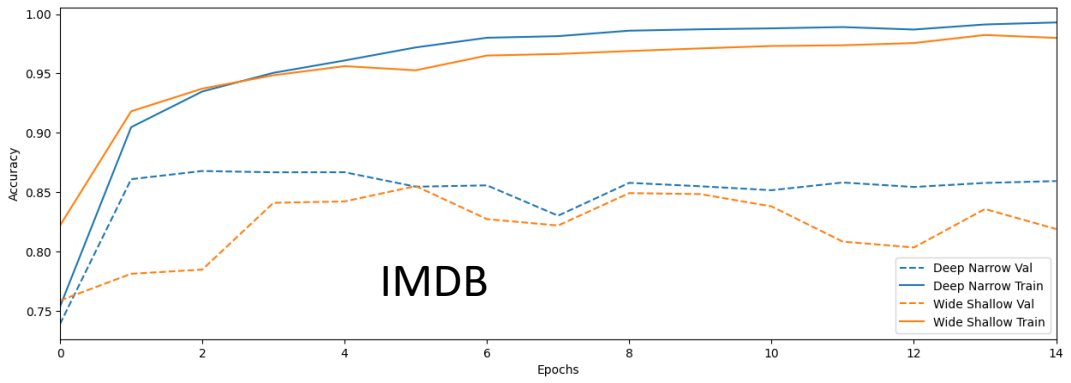

```
=================================================================
Total params: 3156673 (12.04 MB)
Trainable params: 3155649 (12.04 MB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
Epoch 1/15
196/196 [==============================] - 24s 116ms/step - loss: 0.3935 - accuracy: 0.8221 - val_loss: 0.5985 - val_accuracy: 0.7590
Epoch 2/15
196/196 [==============================] - 14s 74ms/step - loss: 0.2147 - accuracy: 0.9182 - val_loss: 0.4679 - val_accuracy: 0.7815
Epoch 3/15
196/196 [==============================] - 13s 65ms/step - loss: 0.1665 - accuracy: 0.9373 - val_loss: 0.4331 - val_accuracy: 0.7850
Epoch 4/15
196/196 [==============================] - 7s 38ms/step - loss: 0.1405 - accuracy: 0.9485 - val_loss: 0.4068 - val_accuracy: 0.8412
Epoch 5/15
196/196 [==============================] - 7s 37ms/step - loss: 0.1199 - accuracy: 0.9562 - val_loss: 0.4939 - val_accuracy: 0.8424
Epoch 6/15
196/196 [==============================] - 7s 35ms/step - loss: 0.1194 - accuracy: 0.9527 - val_loss: 0.4724 - val_accuracy: 0.8552
Epoch 7/15
196/196 [==============================] - 7s 36ms/step - loss: 0.0928 - accuracy: 0.9651 - val_loss: 0.6442 - val_accuracy: 0.8275
Epoch 8/15
196/196 [==============================] - 5s 24ms/step - loss: 0.0875 - accuracy: 0.9664 - val_loss: 0.8108 - val_accuracy: 0.8221
Epoch 9/15
196/196 [==============================] - 6s 32ms/step - loss: 0.0836 - accuracy: 0.9689 - val_loss: 0.6634 - val_accuracy: 0.8494
Epoch 10/15
196/196 [==============================] - 4s 23ms/step - loss: 0.0795 - accuracy: 0.9711 - val_loss: 0.6925 - val_accuracy: 0.8485
Epoch 11/15
196/196 [==============================] - 4s 20ms/step - loss: 0.0708 - accuracy: 0.9732 - val_loss: 0.7744 - val_accuracy: 0.8382
Epoch 12/15
196/196 [==============================] - 4s 20ms/step - loss: 0.0672 - accuracy: 0.9738 - val_loss: 0.9860 - val_accuracy: 0.8086
Epoch 13/15
196/196 [==============================] - 4s 21ms/step - loss: 0.0636 - accuracy: 0.9756 - val_loss: 1.1972 - val_accuracy: 0.8036
Epoch 14/15
196/196 [==============================] - 3s 14ms/step - loss: 0.0493 - accuracy: 0.9824 - val_loss: 0.9677 - val_accuracy: 0.8360
Epoch 15/15
196/196 [==============================] - 4s 19ms/step - loss: 0.0524 - accuracy: 0.9800 - val_loss: 1.0770 - val_accuracy: 0.8191
```

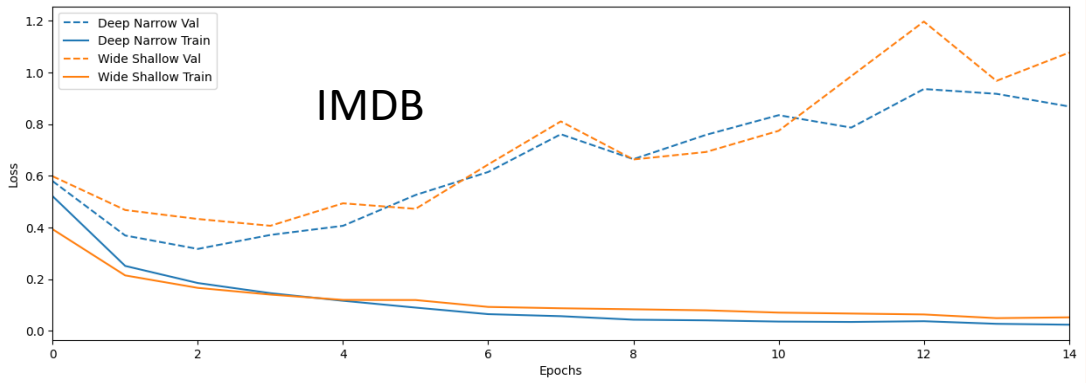
Shallow and Wide Model

Insights from Graphical Representations









Take-aways about Shallow and Wide NN

- First, in principle, there is no reason you need deep neural nets at all. **A sufficiently wide neural network with just a single hidden layer can approximate any (reasonable) function given enough training data**. There are, however, a few difficulties with using an extremely wide, shallow network.

- **The main issue is that these very wide, shallow networks are very good at memorization, but not so good at generalization**. So, if you train the network with every possible input value, a super wide network could eventually memorize the corresponding output value that you want. But that's not useful because for any practical application you won't have every possible input value to train with.

- One could argue that the trade-offs between shallow and deep networks resonate with the age-old tension between parsimony and complexity. **Shallow networks, with their limited architecture, often shine in tasks where the data is less voluminous,** or the underlying relationships are not profoundly nested. Their training is computationally less demanding and might offer more interpretable models, a boon in domains where transparency is paramount.

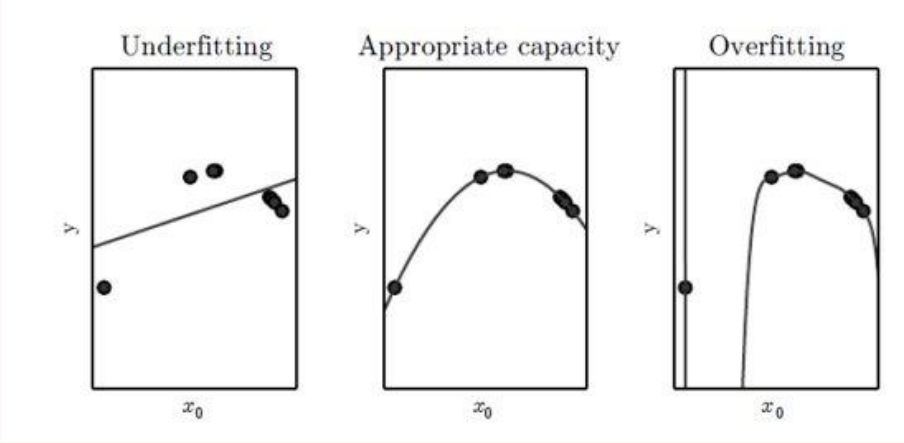
- **Aside from the specter of overfitting, the wider your network, the longer it will take to train**. Deep networks already can be very computationally expensive to train, so there's a strong incentive to make them wide enough that they work well, but no wider.

Take-aways about Deep and Narrow NN

- **Deep networks, with their *capacity* to grasp** intricate data intricacies, tend to outperform in scenarios inundated with vast datasets and underlying complex structures. Their ability to autonomously derive features and adaptively learn from data nuances positions them as a formidable tool in AI challenges.

- **The advantage of multiple layers is that they can learn features at various levels of abstraction**. For example, if you train a deep convolutional neural network to classify images, you will find that the first layer will train itself to recognize very basic things like edges, the next layer will train itself to recognize collections of edges such as shapes, the next layer will train itself to recognize collections of shapes like eyes or noses, and the next layer will learn even higher-order features like faces. **Multiple layers are much better at generalizing because they learn all the intermediate features** between the raw data and the high-level classification.

- Increasing the number of hidden layers much more than the sufficient number of layers will **cause accuracy in the test set to decrease**. It will cause your network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data. The left picture they try to fit a linear function to the data. This function is not complex enough to correctly represent the data, and it suffers from a bias(underfitting) problem. In the middle picture, the model has the appropriate complexity to accurately represent the data and to generalize, since it has learned the trend that this data follows (the data was synthetically created and has an inverted parabola shape). In the right picture, the model fits to the data, but it overfits to it, it hasn't learnt the trend and thus it is not able to generalize to new data.



Conclusion

We tested the performance of both types of neural network on two different datasets. One application was image recognition, and the other was NLP. In both the cases we saw that the deep and narrow model out-performed the shallow and wide model, despite having slightly a smaller number of parameters. This gives us an idea of how more layers in a neural network helps us better recognize the patterns in data and thus, has better generalization. On the other hand, more nodes and smaller number of layers lag in generalizing.

In practice, the choice between a deep and narrow network versus a wide and shallow network is often a trade-off based on the specific requirements of the task, the nature of the input data, the availability of training data, and computational constraints. In some advanced architectures, a combination of both approaches is used to harness the strengths of each. For example, in some modern CNN architectures, there are initial layers that are wide and shallow to capture a broad range of features, followed by deeper layers to capture more abstract representations.

Some tips to choose between Deep and Narrow or Wide and Shallow :

1. **Assess Problem Complexity and Data Characteristics**:
    - Determine the complexity of your task. Use shallow networks for simpler problems and deep networks for more complex tasks involving hierarchical data structures.
    - Analyze your data. Consider the volume and variety of features. Deep networks are typically more suited for large and complex datasets.
2. **Evaluate Computational Resources and Performance Needs**:
    - Consider the available computational power and memory. Deep networks require more resources.
    - Identify the performance and accuracy requirements of your task. Deep networks often provide higher accuracy for complex tasks, but shallow networks are more efficient and faster to train (But not always).
3. **Risk of Overfitting and Model Interpretability**:
    - Assess the risk of overfitting, especially if you have a limited amount of data. Shallow networks are less prone to overfitting.
    - If you need a more interpretable model, shallow networks are generally easier to analyze and understand.
4. **Experiment and Validate**:
    - Conduct experiments with both architectures to compare their performance on your specific task.
    - Use validation techniques to evaluate the models and refine your choice based on results.

References

- [1312.6184] Do Deep Nets Really Need to be Deep? (arxiv.org)
- [1608.08225] Why does deep and cheap learning work so well? (arxiv.org)
- 2003.00777.pdf (arxiv.org)
- Do Wide and Deep Networks Learn the Same Things?

Thank you