Exploring Shortcut Learning with MedMNIST

Caroline Hixon

Review: Shortcut Learning

Shortcut Learning in Deep Neural Networks. Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, Felix A. (2020).

Shortcuts revealed when tested under slightly different circumstances → use o.o.d. data

Where to find shortcuts:

- *Overfitting features* appear when performs well on training, but not i.i.d. validation
- *Shortcut features* appear when perform well on training and i.i.d., but not o.o.d
- *Intended features* appear when succeed with o.o.d. tests

Methodology

Models

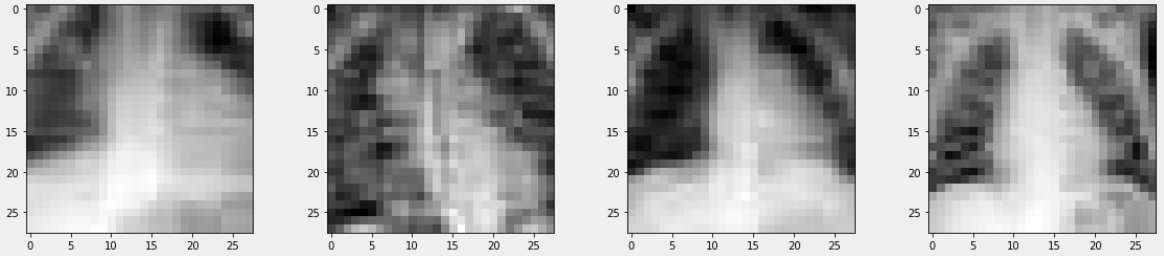
- Simple FC neural network
- Baseline CNN
- CNN with dropout
- ResNet

Testing Environments

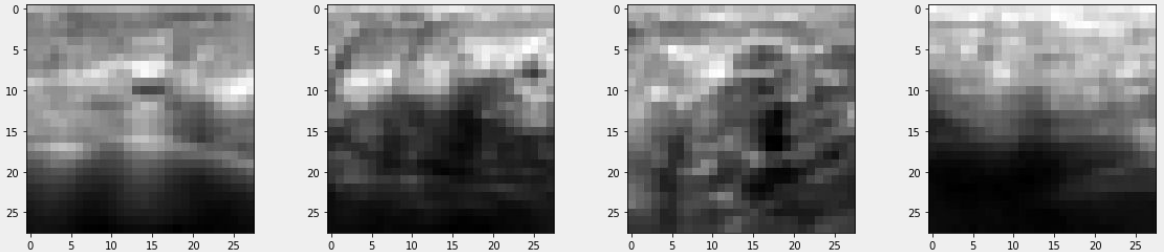
- No augmentation
- Test set augmentation (random shifts, rotations)
- Train and test set augmentation (random shifts, rotations)

Data: MedMNIST

- pneumoniaMNIST
  - Chest X-Ray
  - Binary class (2)
  - 5,856 samples



- breastMNIST
  - Breast ultrasound
  - Binary class (2)
  - 780 samples



Models

- Simple FC Neural Network
  - 3 dense layers (neurons = 300, 100, 10), relu
- Baseline CNN
  - 1 Conv2D layer, kernel size 3, relu + maxpool of size 2
- CNN with dropout
  - Two Conv2D layers, kernel size 3, relu + maxpool of size 2
  - Batch normalization
  - Dropout = 0.5
- ResNet
  - 4 layers with residual units

Results: No Augmentation

- Highest accuracy in each model (bold)
  - Train accuracy for all pneumonia
  - train/test accuracy for breast
- Highest accuracy from all models (blue)
  - ResNet for both
- Highest test accuracy (green)
  - Pneumonia: CNN with dropout
  - Breast: Simple NN
- Simple and ResNet: val and test accuracy drop significantly from train accuracy

Model	Dataset	train acc	val acc	test acc
Simple NN	Pneumoina	**94.35**	76.34	63.3
	Breast	75.8	75.6	**78.8**
Baseline CNN	Pneumoina	**95.31**	95.61	81.73
	Breast	77.1	75.6	**75.64**
CNN with dropout	Pneumoina	**97.37**	96.76	84.13
	Breast	**87.73**	46.15	48.08
ResNet	Pneumoina	**97.58**	75.76	62.66
	Breast	**95.4**	73	73.08

Results: Test set only Augmentation

- Highest accuracy in each model (bold)
  - Train accuracy for all pneumonia
  - train/val accuracy for breast
- Highest accuracy from all models (blue)
  - Pneumonia: CNN with dropout
  - Breast: ResNet
- Highest test accuracy (green)
  - Baseline CNN for both
- Breast: val and test accuracy drop significantly from train accuracy

Model	Dataset	train acc	val acc	test acc
Simple NN	Pneumoina	**94.33**	90.27	73.24
	Breast	75.8	62.8	61.53
Baseline CNN	Pneumoina	**95.31**	95.23	84.13
	Breast	78.75	**80.77**	79.49
CNN with dropout	Pneumoina	**97.39**	92.94	74.04
	Breast	**90.66**	71.79	68.59
ResNet	Pneumoina	**97.24**	85.31	67.47
	Breast	**93.59**	73.08	73.08

Results: Train and test Augmentation

- Highest accuracy in each model (bold)
  - Train accuracy for all pneumonia
  - One val for breast
- Highest accuracy from all models (blue)
  - ResNet for both
- Highest test accuracy (green)
  - Pneumonia: Baseline CNN
  - Breast: CNN with dropout
- Breast: val and test accuracy drop significantly from train accuracy

Model	Dataset	train acc	val acc	test acc
Simple NN	Pneumoina	**94.6**	93.89	85.42
	Breast	**78.94**	73.08	74.36
Baseline CNN	Pneumoina	**95.77**	94.66	86.38
	Breast	76.92	**78.21**	77.56
CNN with dropout	Pneumoina	**95.88**	63.36	75.48
	Breast	**86.26**	73.08	78.21
ResNet	Pneumoina	**97.77**	87.02	69.39
	Breast	**94.87**	73.08	73.08

Results: Comparing Models

- Highest train accuracy -
  - Pneumonia: 97.77, ResNet, train and test augmentation
  - Breast: 90.66, CNN with dropout, test augmentation

- Highest val accuracy -
  - Pneumonia: 96.76, CNN with dropout, no augmentation
  - Breast: 80.77, Baseline CNN, test augmentation

- Highest test accuracy
  - Pneumonia: 86.38, Baseline CNN, train and test augmentation
  - Breast: 79.49, Baseline CNN, test augmentation

Conclusions

- Convolutional NN help with obvious shortcuts (like paper showed)
- All test accuracies relatively high, but not close to train/val accuracies → shortcuts
- Seems the simple models work better for testing

Future work

- Expand to more MedMNIST datasets
- Try different NN architectures
- Use different variations of augmentation
- Test on o.o.d. data from different sources

```
[ ]  #BASIC SEQUENTIAL NN

     modelNN = keras.models.Sequential([
         keras.layers.Flatten(input_shape=[28,28]),
         keras.layers.Dense(300, activation='relu'),
         keras.layers.Dense(100, activation='relu'),
         keras.layers.Dense(10, activation='softmax')
     ])

     modelNN.summary()

     modelNN.compile(loss='sparse_categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
     historyNN = modelNN.fit(x_train, y_train, epochs=15, validation_data=(x_val, y_val))
```

```
[ ]  # BASELINE CNN

     epochs = 15
     batch_size = 100

     baseline_model = keras.Sequential([

             #first layer
             layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28,28,1)),
             layers.MaxPoo
                           # CNN with Dropout

             layers.Flatte
             layers.Dense(
             layers.Dense(
         ])

     baseline_model.summary

     opt = optimizers.SGD(
     baseline_model.compile
     historyCNN = baseline_
```

```
CNN_model = keras.Sequential([
        #first layer
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28,28,1)),
        layers.BatchNormalization(),
        layers.MaxPooling2D(pool_size=(2, 2)),
        #second layer
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu", input_shape=(28,28,1)),
        layers.BatchNormalization(),
        layers.MaxPooling2D(pool_size=(2, 2)),

        layers.Flatten(),
        layers.Dense(100, activation="relu"),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ])
```

```
## ResNet

inputs = Input(shape=input_size)
x = Conv2D(num_filters, padding='same', kernel_initializer='he_normal', kernel_size=7, strides=2, kernel_regularizer=l2(1e-4))(inputs)
x = BatchNormalization()(x)
x = Activation('relu')(x)

for i in range(num_blocks):
    for j in range(num_sub_blocks):
        strides = 1
        is_first_layer_but_not_first_block = j == 0 and i > 0
        if is_first_layer_but_not_first_block:
            strides = 2

        y = Conv2D(num_filters, kernel_size=3, padding='same', strides=strides, kernel_initializer='he_normal', kernel_regularizer=l2(1e-4))(x)
        y = BatchNormalization()(y)
        y = Activation('relu')(y)

        y = Conv2D(num_filters, kernel_size=3, padding='same', kernel_initializer='he_normal', kernel_regularizer=l2(1e-4))(y)
        y = BatchNormalization()(y)

        if is_first_layer_but_not_first_block:
            x = Conv2D(num_filters, kernel_size=1, padding='same', strides=2, kernel_initializer='he_normal', kernel_regularizer=l2(1e-4))(x)

        x = keras.layers.add([x, y])
        x = Activation('relu')(x)

    num_filters = 2 * num_filters

x = AveragePooling2D()(x)
y = Flatten()(x)
outputs = Dense(num_classes,activation='softmax', kernel_initializer='he_normal')(y)

modelrn = Model(inputs=inputs, outputs=outputs)
modelrn.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
modelrn.summary()
```