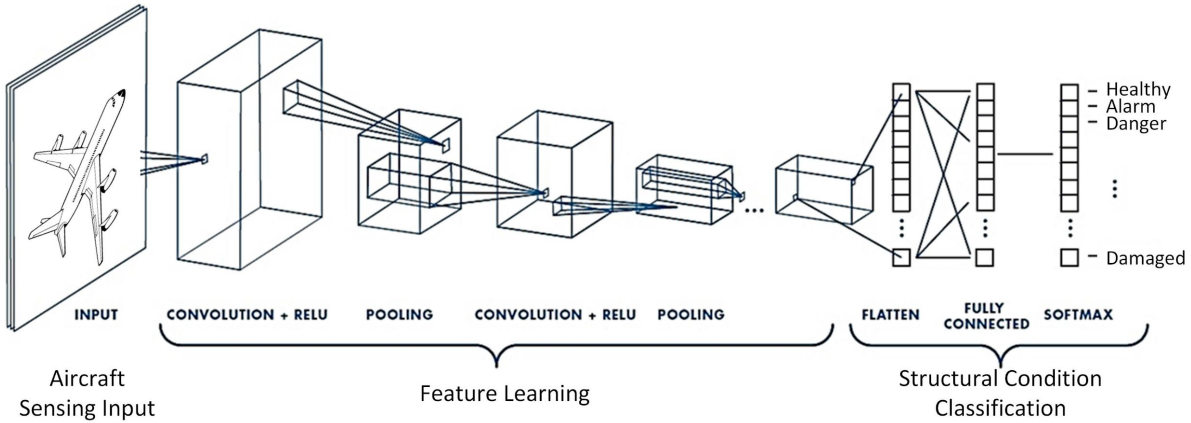
Facial Emotion Recognition

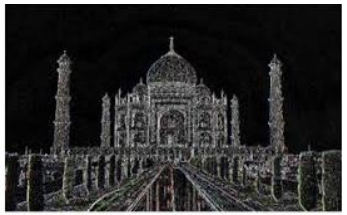
Team Members: Ash Tan, Frank Bruni, Daphne Yang, Jeff Zheng

Context

Machine Learning For Facial Emotion Recognition

Convolutional Neural Networks

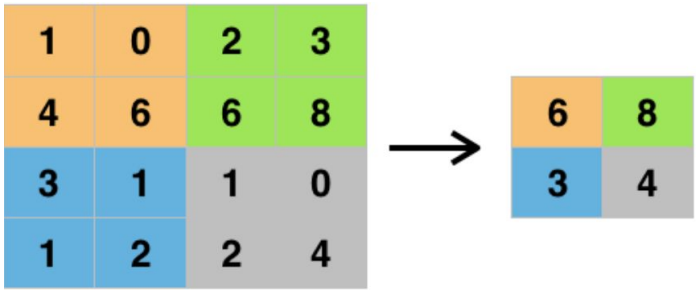




Convolutions

Pooling

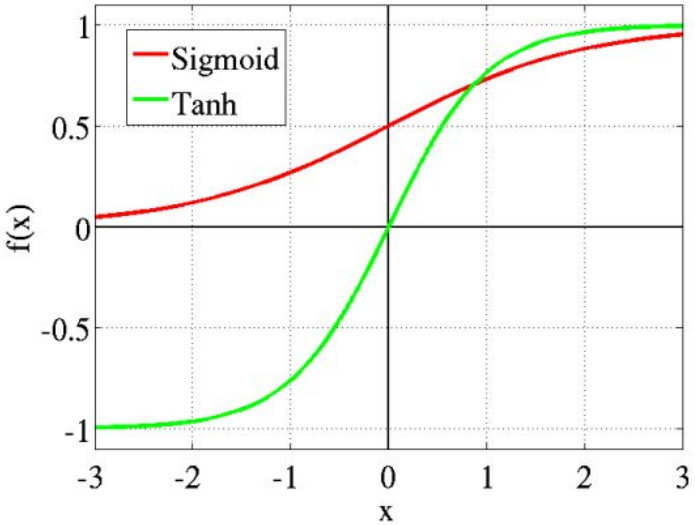
Max Pooling

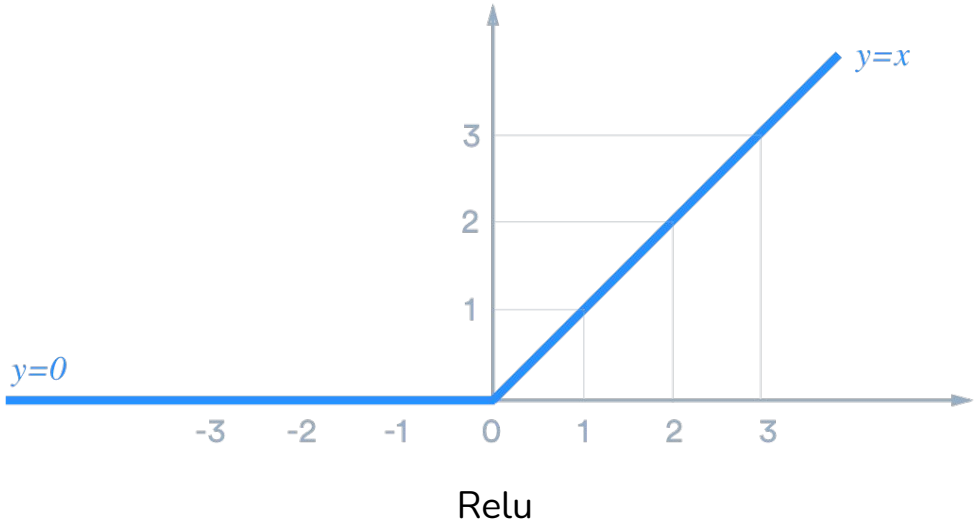


Average Pooling

Activation

- What is Activation?
- Sigmoid and Hyperbolic Tangent Activation
- Rectified Linear (Relu) Activation

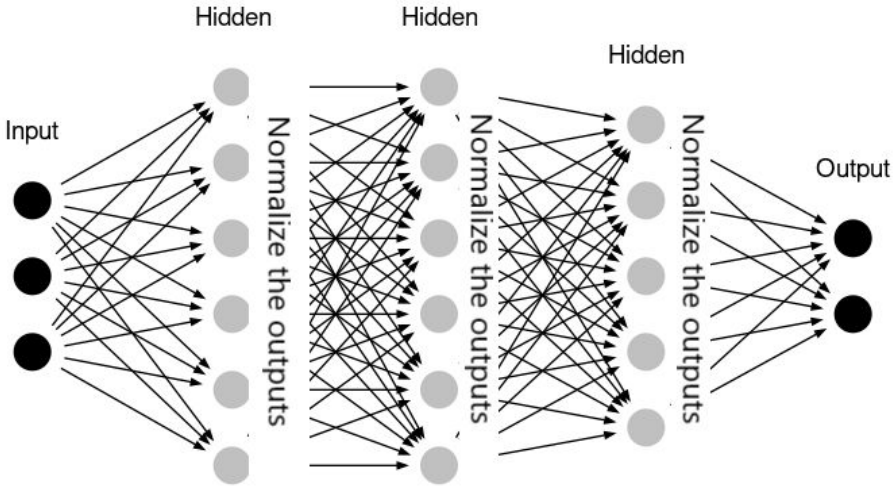




Relu

BatchNormalization

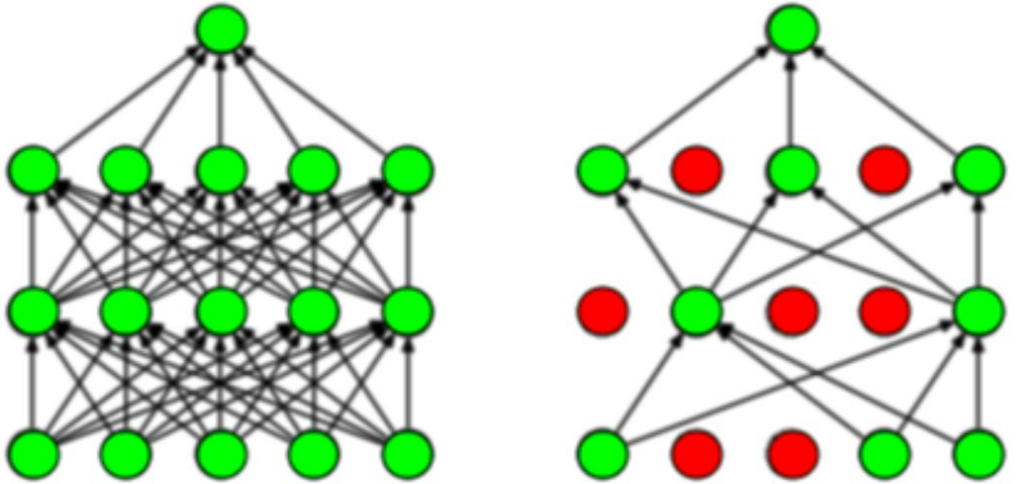
- Deep Network Challenge
- Batch Size
- Standardize Layer Inputs



$$x_{scaled} = \frac{x - mean}{sd}$$

Dropout

1. Problem With Overfitting
2. What is Dropout?
3. How to Dropout

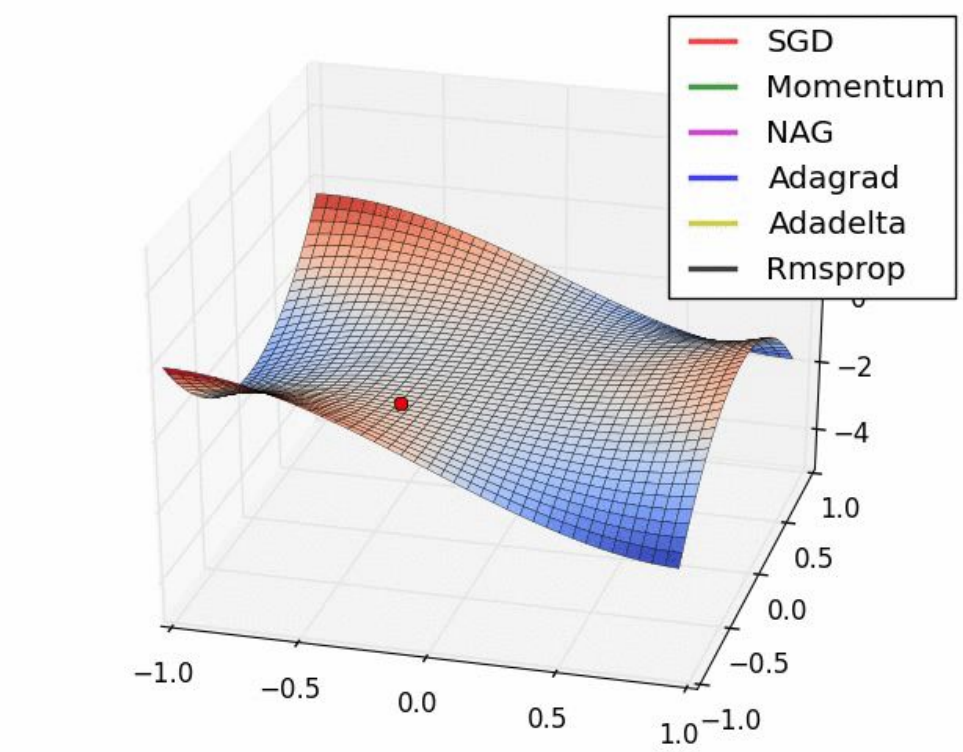


a) Standard Neural Net

b) After applying Dropout

Optimizer

- Stochastic Gradient Descent
  - Weights updated by following the negative gradient with randomness introduced
- RMSprop
  - A type of SGD based on a moving average of squared gradients
- Adam
  - A type of SGD based on adaptive estimation of first and second order moments



Why CNN?

Sklearn's MLP Classifier is a very useful perceptron-based neural net.

```
mlp = MLPClassifier(hidden_layer_sizes=(300,), max_iter=2000, alpha=1e-4,
                    solver='sgd', verbose=10, random_state=1,
                    learning_rate_init=.01)
mlp.fit(train, train_labels)
```

```
Iteration 1, loss = 1084.45769546
Iteration 2, loss = 1215.64234577
Iteration 3, loss = 1215.61754893
Iteration 4, loss = 1215.59748855
Iteration 5, loss = 1215.57885807
Iteration 6, loss = 1215.56073009
Iteration 7, loss = 1215.54295232
Iteration 8, loss = 1215.52517101
Iteration 9, loss = 1215.50749831
Iteration 10, loss = 1215.48994226
Iteration 11, loss = 1215.47231058
Iteration 12, loss = 1215.45469211
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

```
MLPClassifier(hidden_layer_sizes=(300,), learning_rate_init=0.01, max_iter=2000,
              random_state=1, solver='sgd', verbose=10)
```

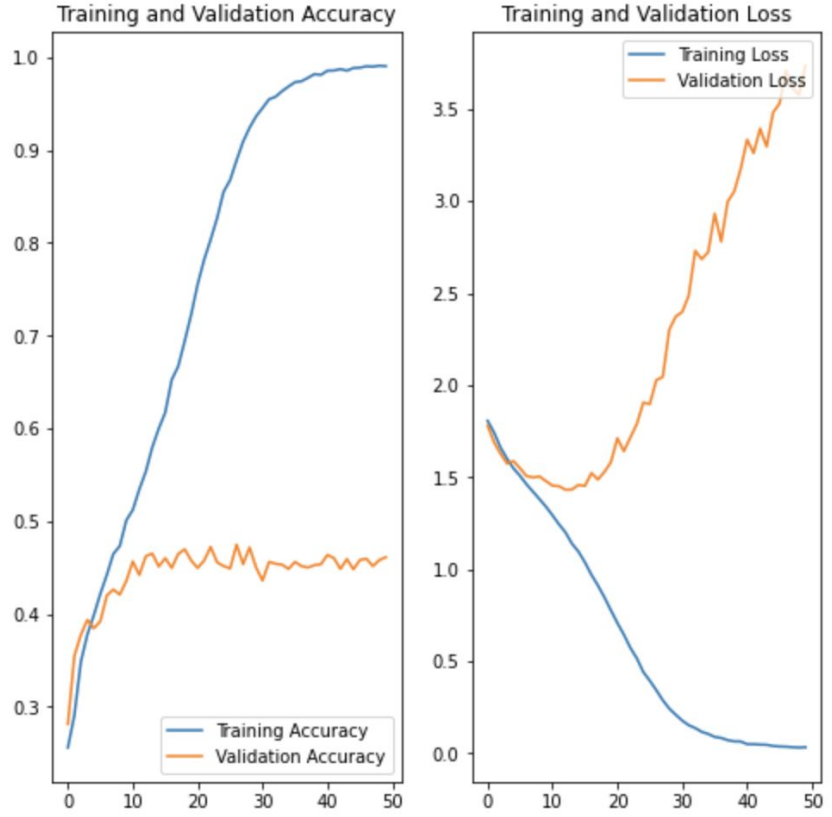
```
print("Training set score: %f" % mlp.score(train, train_labels))
print("Test set score: %f" % mlp.score(test, test_labels))
```

```
Training set score: 0.251385
Test set score: 0.249652
```

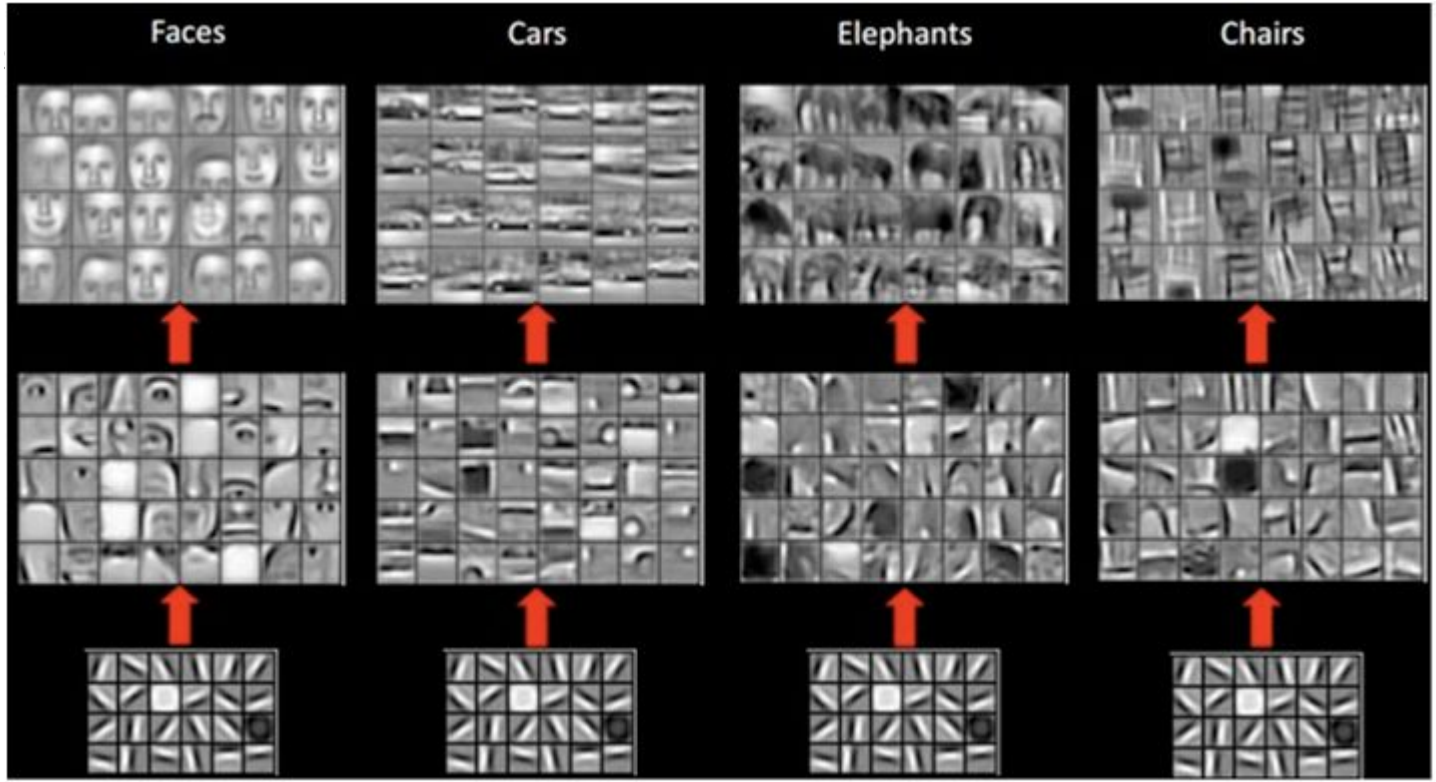
‘Default’ NN (demo)

```
# build model
model = Sequential()
model.add(Conv2D(48, (3, 3), padding='same',
                 input_shape=(48,48,3)))
model.add(Activation('relu'))
model.add(Conv2D(48, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(48*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(48*2*8))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
model.compile(optimizers.RMSprop(lr=0.0001, decay=1e-6),loss="categorical_crossentropy",metrics=["accuracy"])
```

```
[3.6171019077301025, 0.46320563554763794]
```



Improving our NN



Improved NN

```
model = Sequential()
model.add(Conv2D(48, (3, 3), padding='same',
                 input_shape=(48,48,3)))
model.add(Activation('relu'))


model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(48, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(48, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(48*2*8))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))

model.compile(optimizers.Adam(lr=0.0001, decay=1e-7),loss="categorical_crossentropy",metrics=["accuracy"])
```

Added BatchNormalization

Bigger layers

Added BatchNormalization

More layers

Better optimizer

```
[2.0992813110351562, 0.5132319927215576]
```

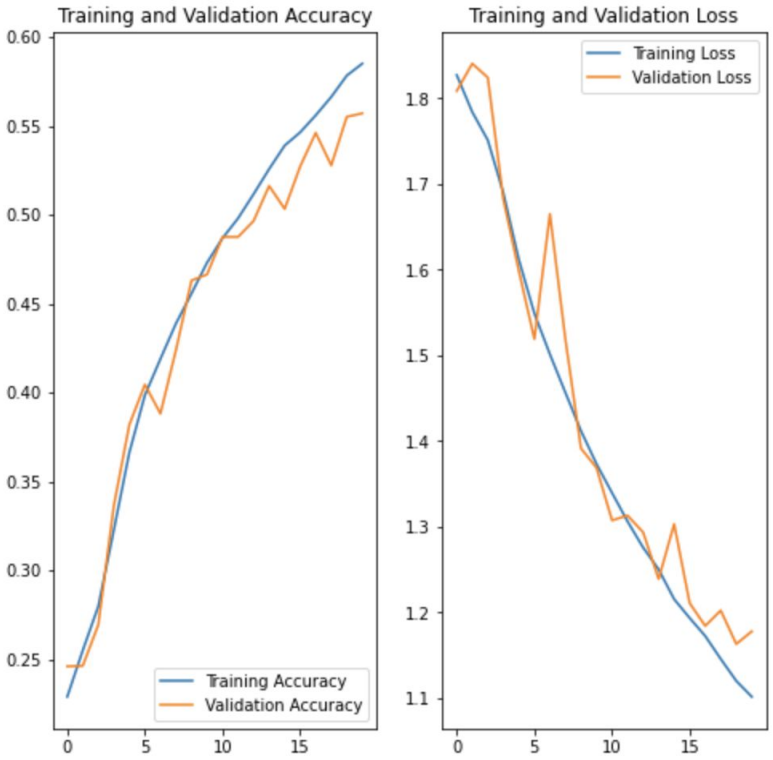
Adding dropout: looks promising at 20 epochs

```
model = Sequential()
model.add(Conv2D(48, (3, 3), padding='same',
                 input_shape=(48,48,3)))
model.add(Activation('relu'))

model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(48, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(48*2*8))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))

model.compile(optimizers.Adam(lr=0.0001, decay=1e-7),loss="categorical_crossentropy",metrics=["accuracy"])
```

Added Dropout

Added Dropout

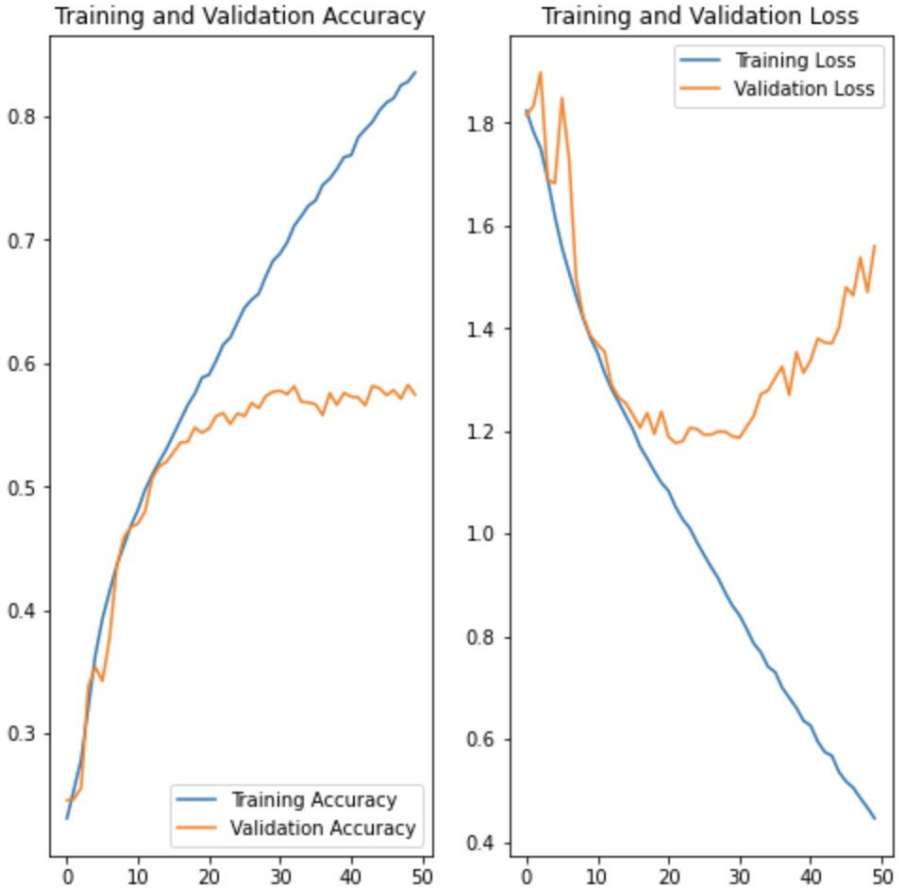
```
[1.22072172164917, 0.5391328930854797]
```



But 50 epochs:

```
model = Sequential()
model.add(Conv2D(48, (3, 3), padding='same',
                 input_shape=(48,48,3)))
model.add(Activation('relu'))

model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(48, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(48*2*8))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))

model.compile(optimizers.Adam(lr=0.0001, decay=1e-7),loss="categorical_crossentropy",metrics=["accuracy"])
```

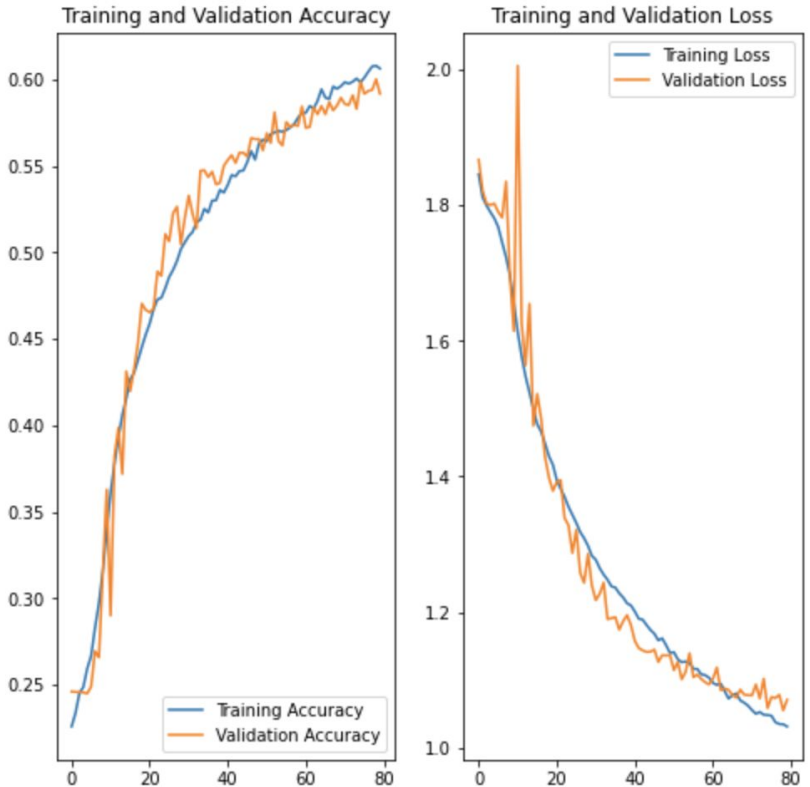
```
[1.6314417123794556, 0.5675675868988037]
```



Solution: More dropout!

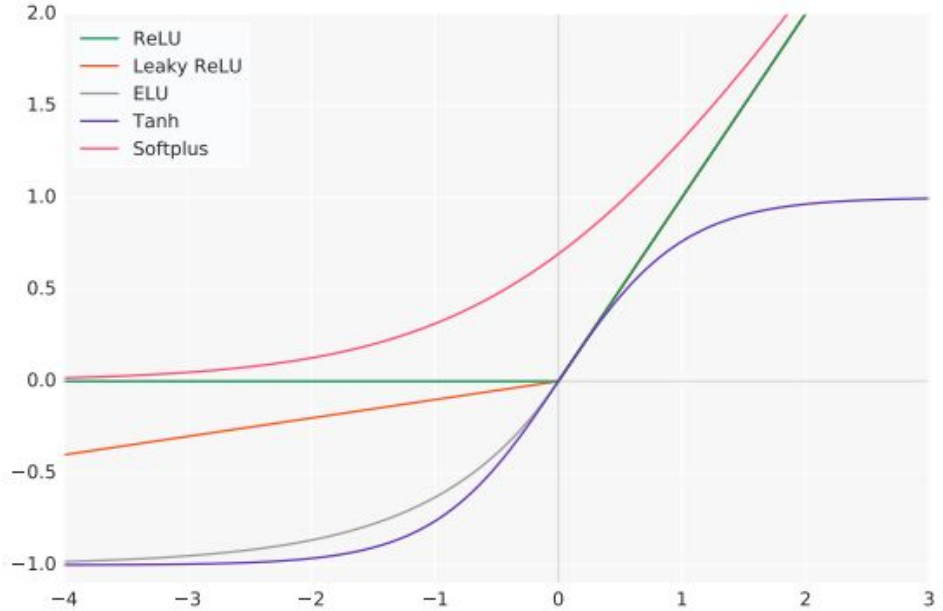
```
model = Sequential()
model.add(Conv2D(48, (3, 3), padding='same', input_shape=(48,48,3)))
model.add(Activation('relu'))

model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Dropout(0.4))

model.add(Conv2D(48*2*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48*2*2, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(48*2, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Conv2D(48, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(48*2*8))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(7, activation='softmax'))
model.compile(optimizers.Adam(lr=0.0001, decay=1e-7),loss="categorical_crossentropy",metrics=["accuracy"])
```

```
[1.1286063194274902, 0.5805180072784424]
```

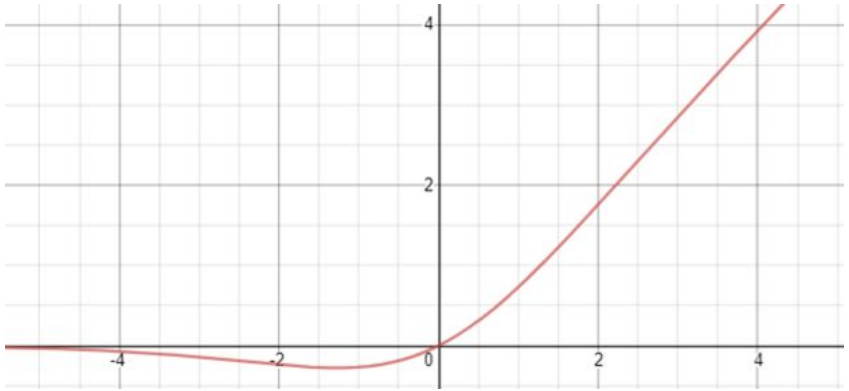


How to improve learning?

Traditional Activation Functions

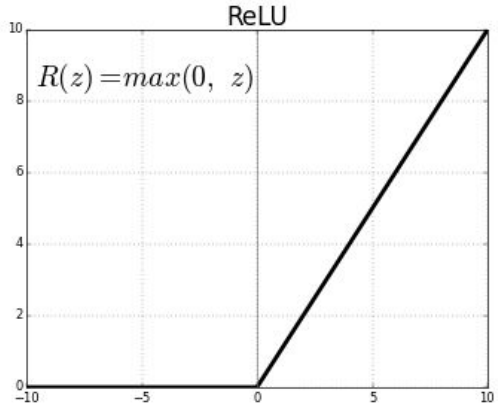


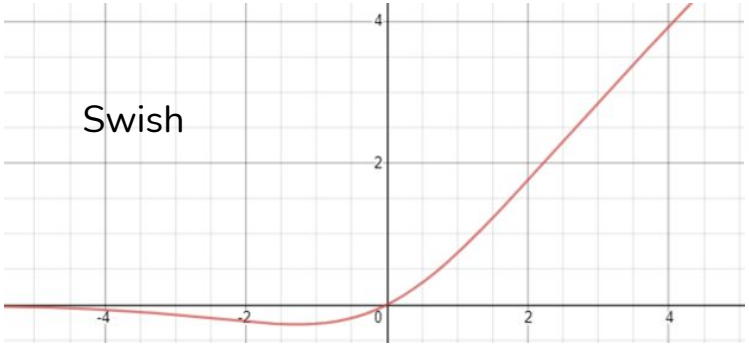
Swish

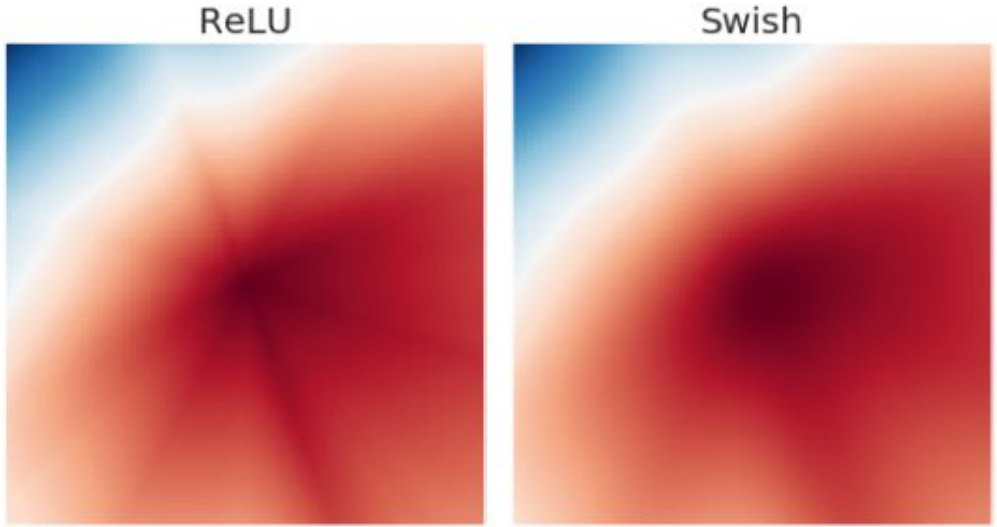


$$f(x) = x * (1 + \exp(-x))^{-1}$$

ReLU vs Swish



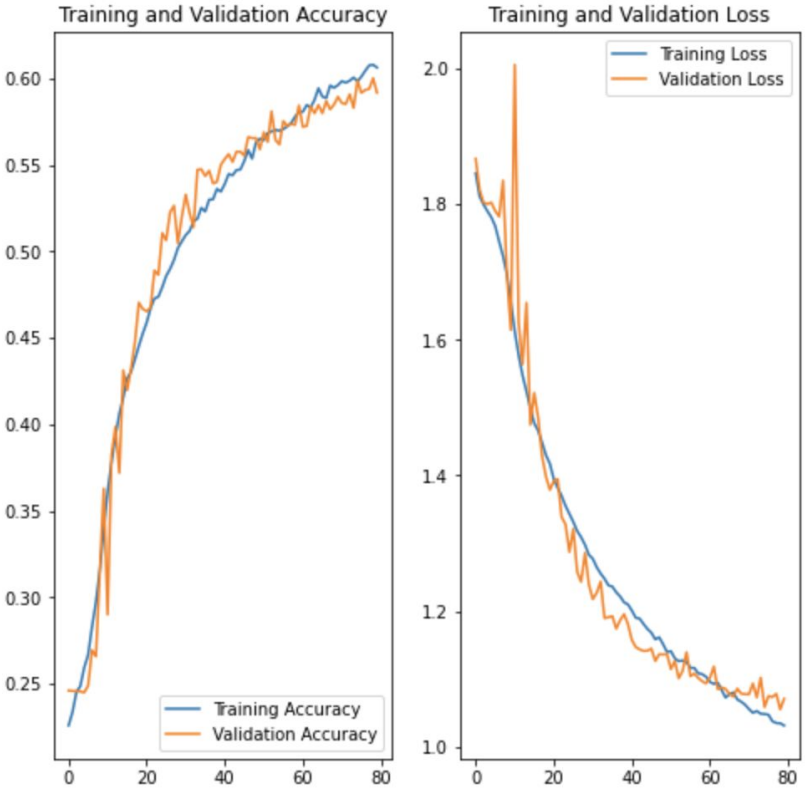




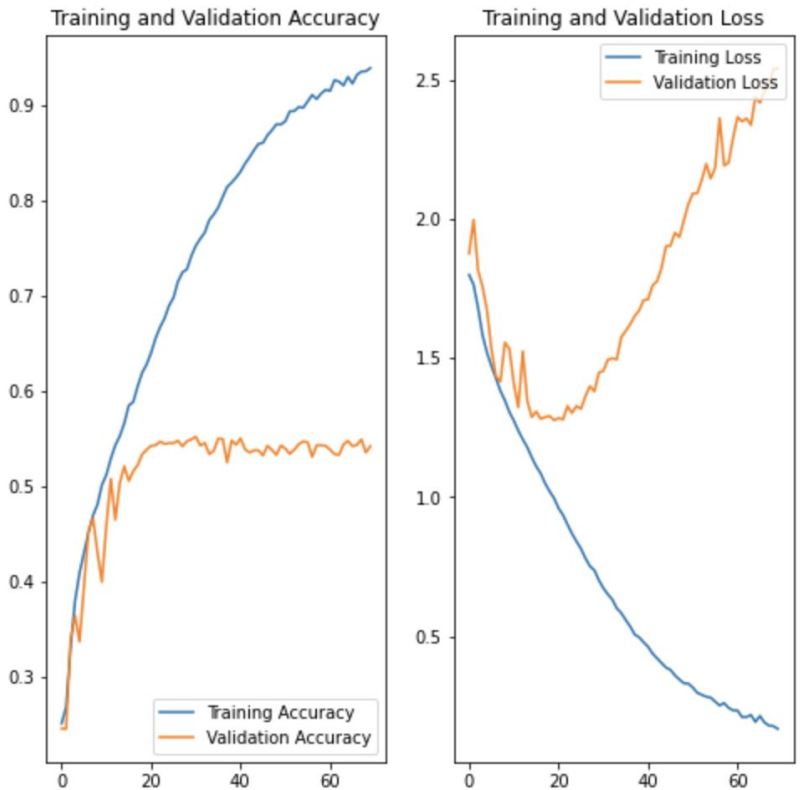
Output landscape of each activation function

Relu versus Swish

[1.1286063194274902, 0.5805180072784424]

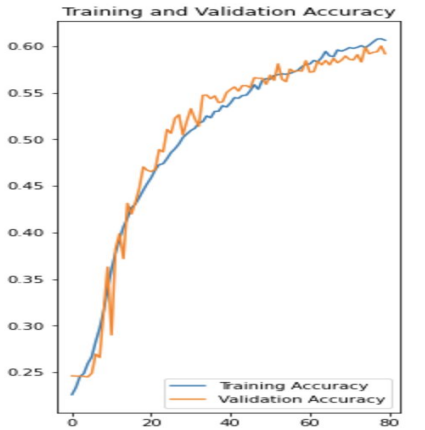


[2.7117035388946533, 0.5374436974525452]

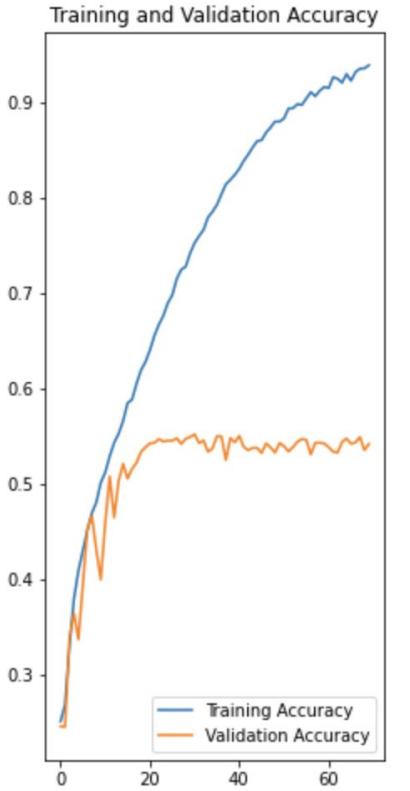


Relu versus Swish (accuracy)

[1.1286063194274902, 0.580518007

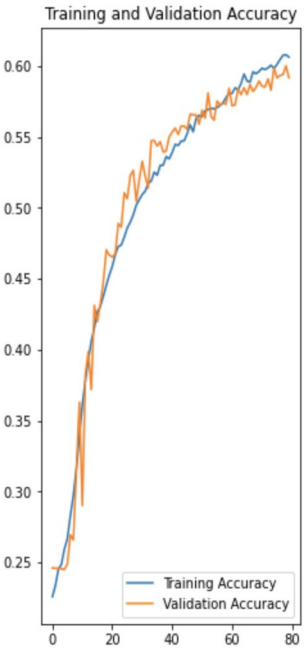


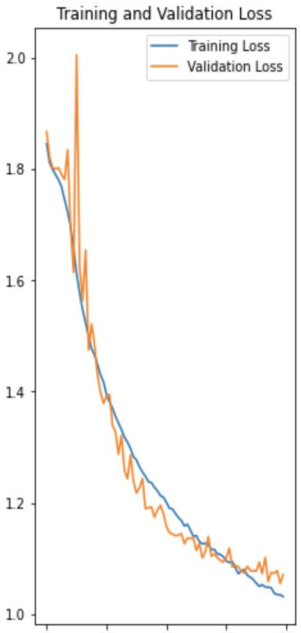
[2.7117035388946533, 0.5374436



Relu

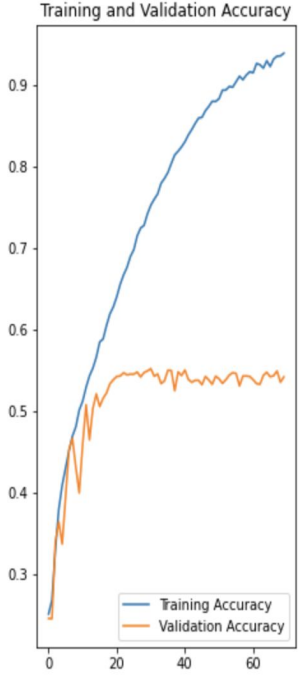
[1.1286063194274902, 0.5805180072784424]

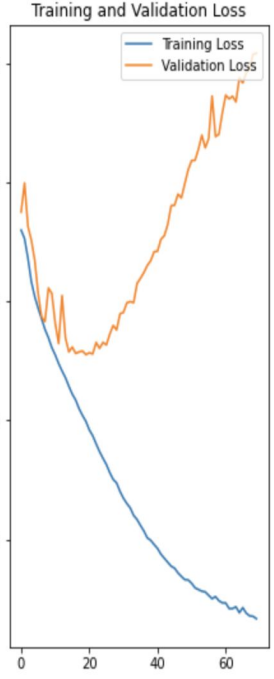




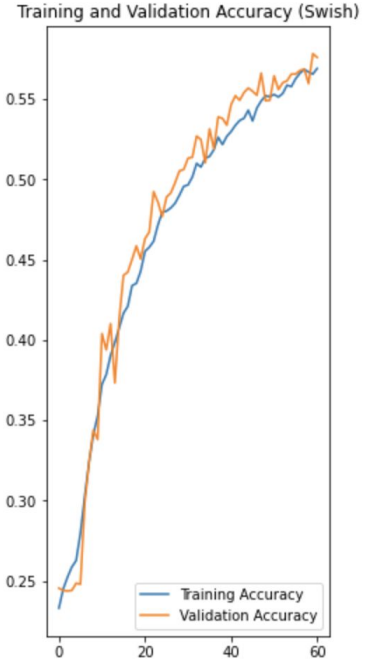
Swish

[2.7117035388946533, 0.5374436974525452]



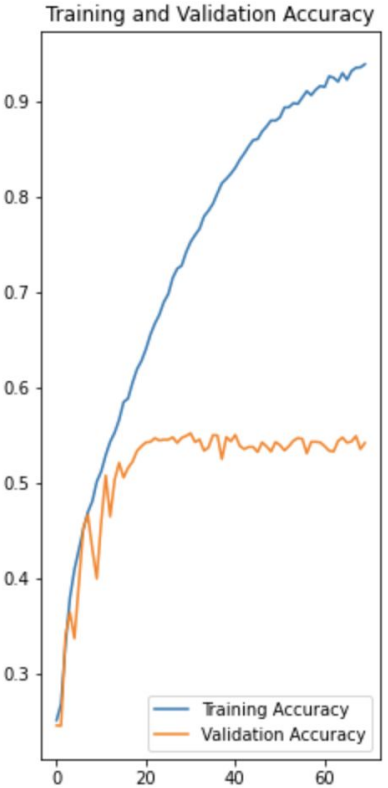


Swish with more dropout



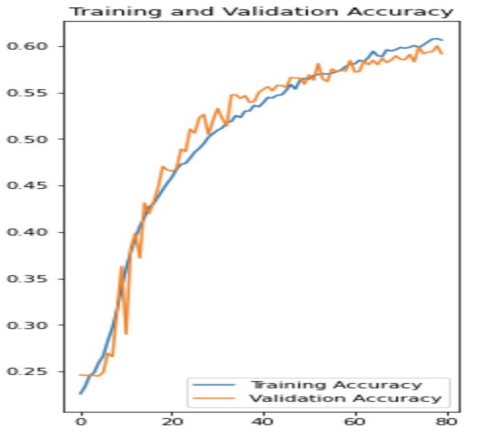
Swish

[2.7117035388946533, 0.53744369



Relu

[1.1286063194274902, 0.580518007



Swish with more dropout

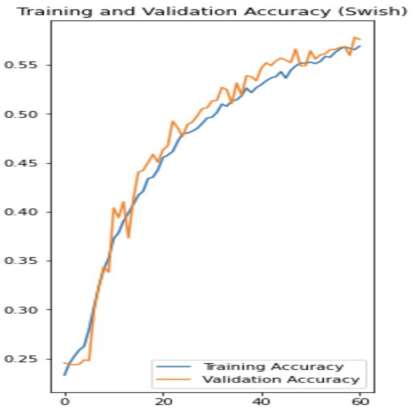


Image processing?

```
approximation = pca.inverse_transform(
```

```
#Original Image
fig = plt.figure(frameon=False)
# print(i)
plt.imshow(np.reshape(pixelarray[1
plt.axis('off')
```

```
(-0.5, 47.5, 47.5, -0.5)
```

```
# Normalized Image
fig = plt.figure(frameon=False)
# print(i)
plt.imshow(np.reshape(normalized
plt.axis('off')
```

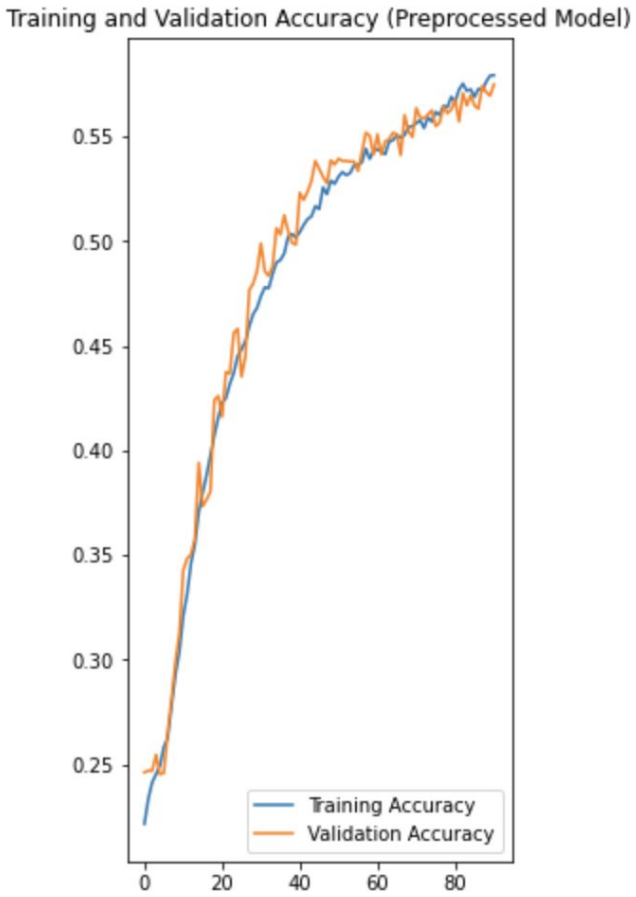
```
(-0.5, 47.5, 47.5, -0.5)
```

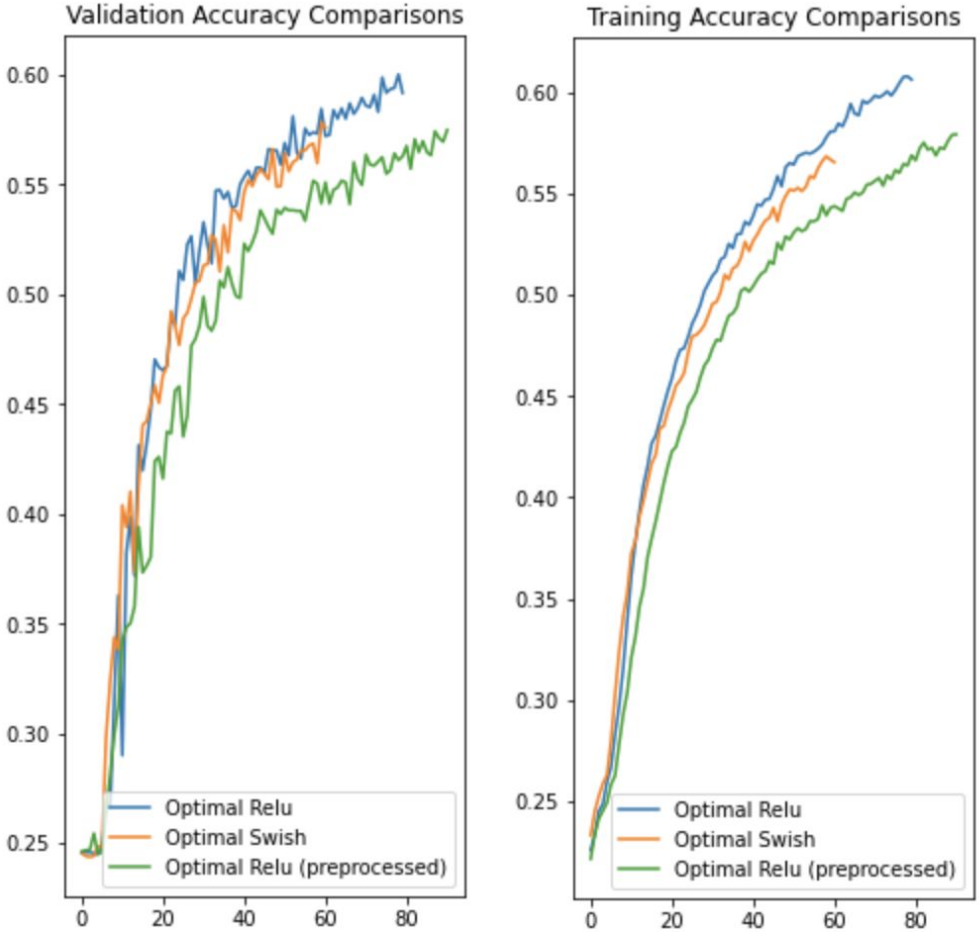
```
# 98% variance principal components
fig = plt.figure(frameon=False)
# print(i)
plt.imshow(np.reshape(approximation[1]
plt.axis('off')
```

```
(-0.5, 47.5, 47.5, -0.5)
```

Processed-Image model

- Slower learning
- Reached 0.57 training and validation accuracy at 90th epoch
- Has potential - slower plateauing?
- Overall, not super impressive




Validation Accuracy Comparisons
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
20
40
60
80
Optimal Relu
Optimal Swish
Optimal Relu (preprocessed)
Training Accuracy Comparisons
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
20
40
60
80
Optimal Relu
Optimal Swish
Optimal Relu (preprocessed)

Overview Data Notebooks Discussion Leaderboard Rules

https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/leaderboard

#	△pub	Team Name	Notebook	Team Members	Score	Entries	Last
1	—	RBM			0.71161	5	8y
2	—	Unsupervised			0.69267	8	8y
3	—	Maxim Milakov			0.68821	7	8y
4	—	Radu+Marius+Cristi			0.67483	6	8y
5	—	Lor.Voldy			0.65254	2	8y
6	▲1	ryank			0.65087	2	8y
7	▼1	Eric Cartman			0.64474	1	8y
8	—	Xavier Bouthillier			0.64224	1	8y
9	▲1	Alejandro Dubrovsky			0.63109	5	8y
10	▼1	sayit			0.62190	2	8y
11	—	jaberg			0.61967	6	8y
12	▲1	bulbuloglu			0.59654	8	8y
13	▼1	kg			0.59208	4	8y
14	—	Liu			0.58623	8	8y
15	▲1	12AngryBird			0.57899	2	8y

Us!