In [1]:

```
import tensorflow as tf
#import tensorflow_datasets as tfds
```

Baseline model

In [2]:

```
# Load Data
# tfds.list_builders()
# (train, test), info = tfds.load("mnist",split=['train', 'test'], with_info=True, as_supervised=True)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

In [3]:

```
# Baseline model definition
model = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu'),
                             tf.keras.layers.Dense(16, activation='relu'),
                             tf.keras.layers.Dense(10, activation='softmax')
])
```

In [4]:

```
# Baseline model compilation
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy']
             )
```

In [ ]:

```
# Baseline model fitting
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=100,
                    validation_data=(x_test, y_test),
                    verbose=2
                    )
```

In [6]:

```
# Baseline model evaluation
model.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2953 - sparse_categorical_accuracy: 0.9446 - 320ms/epoch - 1ms/step
```

Out[6]:

```
[0.2952946126461029, 0.944599986076355]
```

The baseline model with `kernel_initializer='glorot_uniform'` , `bias_initializer='zeros'`

Try different kernel initializers

Model with `glorot_normal` kernel initializer

In [7]:

```
model_kernel1 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='glorot_normal'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='glorot_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='glorot_normal')
])
```

In [8]:

```
model_kernel1.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_kernel1 = model_kernel1.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [10]:

```
model_kernel1.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2941 - sparse_categorical_accuracy: 0.9271 - 329ms/epoch - 1ms/step
```

Out[10]:

```
[0.29405325651168823, 0.9271000027656555]
```

Model with `Ones` kernel initializer

In [11]:

```
model_kernel2 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='Ones'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='Ones'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='Ones')
])
```

In [12]:

```
model_kernel2.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_kernel2 = model_kernel2.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [14]:

```
model_kernel2.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 1.5867 - sparse_categorical_accuracy: 0.3872 - 321ms/epoch - 1ms/step
```

Out[14]:

```
[1.5867396593093872, 0.38719999790191650]
```

Model with `random_uniform` initializer

In [15]:

```
model_kernel3 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='random_uniform'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='random_uniform'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='random_uniform')
])
```

In [16]:

```
model_kernel3.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_kernel3 = model_kernel3.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [18]:

```
model_kernel3.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.3245 - sparse_categorical_accuracy: 0.9483 - 318ms/epoch - 1ms/step
```

Out[18]:

```
[0.32448622584342957, 0.9483000040054321]
```

Model with `random_normal` initializer

In [19]:

```
model_kernel4 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='random_normal'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='random_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='random_normal')
])
```

In [20]:

```
model_kernel4.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_kernel4 = model_kernel4.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [22]:

```
model_kernel4.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.3149 - sparse_categorical_accuracy: 0.9517 - 317ms/epoch - 1ms/step
```

Out[22]:

```
[0.31487616896629333, 0.95169997215271]
```

Model with `he_uniform` initializer

In [23]:

```
model_kernel5 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='he_uniform'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='he_uniform'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='he_uniform')
])
```

In [24]:

```
model_kernel5.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_kernel5 = model_kernel5.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [26]:

```
model_kernel5.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.6595 - sparse_categorical_accuracy: 0.8030 - 345ms/epoch - 1ms/step
```

Out[26]:

```
[0.6594504714012146, 0.8029999732971191]
```

Model with `he_normal` initializer

In [27]:

```
model_kernel6 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='he_normal'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='he_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='he_normal')
])
```

In [28]:

```
model_kernel6.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_kernel6 = model_kernel6.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [30]:

```
model_kernel6.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.4258 - sparse_categorical_accuracy: 0.8842 - 329ms/epoch - 1ms/step
```

Out[30]:

```
[0.425766259431839, 0.8841999769210815]
```

Try different bias initializers

Model with `glorot_normal` bias initializer

In [31]:

```
model_bias1 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='glorot_normal'),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='glorot_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', bias_initializer='glorot_normal')
])
```

In [32]:

```
model_bias1.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_bias1 = model_bias1.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [34]:

```
model_bias1.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2926 - sparse_categorical_accuracy: 0.9159 - 326ms/epoch - 1ms/step
```

Out[34]:

```
[0.29257574677467346, 0.9158999919891357]
```

Model with `Ones` bias initializer

In [35]:

```
model_bias2 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='Ones'),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='Ones'),
                             tf.keras.layers.Dense(10, activation='softmax', bias_initializer='Ones')
])
```

In [36]:

```
model_bias2.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_bias2 = model_bias2.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [38]:

```
model_bias2.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2843 - sparse_categorical_accuracy: 0.9292 - 329ms/epoch - 1ms/step
```

Out[38]:

```
[0.2843486964702606, 0.9291999936103821]
```

Model with `random_uniform` bias initializer

In [39]:

```
model_bias3 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='random_uniform'),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='random_uniform'),
                             tf.keras.layers.Dense(10, activation='softmax', bias_initializer='random_uniform')
])
```

In [40]:

```
model_bias3.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_bias3 = model_bias3.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [42]:

```
model_bias3.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.3218 - sparse_categorical_accuracy: 0.9416 - 322ms/epoch - 1ms/step
```

Out[42]:

```
[0.321821004152298, 0.9416000247001648]
```

Model with `random_normal` bias initializer

In [43]:

```
model_bias4 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='random_normal'),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='random_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', bias_initializer='random_normal')
])
```

In [44]:

```
model_bias4.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_bias4 = model_bias4.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [46]:

```
model_bias4.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2743 - sparse_categorical_accuracy: 0.9432 - 338ms/epoch - 1ms/step
```

Out[46]:

```
[0.27433234453201294, 0.9431999921798706]
```

Model with `he_uniform` bias initializer

In [47]:

```
model_bias5 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='he_uniform'),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='he_uniform'),
                             tf.keras.layers.Dense(10, activation='softmax', bias_initializer='he_uniform')
])
```

In [48]:

```
model_bias5.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_bias5 = model_bias5.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [50]:

```
model_bias5.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2762 - sparse_categorical_accuracy: 0.9336 - 322ms/epoch - 1ms/step
```

Out[50]:

```
[0.27617576718330383, 0.9336000084877014]
```

Model with `he_normal` bias initializer

In [51]:

```
model_bias6 = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='he_normal'),
                             tf.keras.layers.Dense(16, activation='relu', bias_initializer='he_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', bias_initializer='he_normal')
])
```

In [52]:

```
model_bias6.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy']
                   )
```

In [ ]:

```
history_bias6 = model_bias6.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [54]:

```
model_bias6.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2956 - sparse_categorical_accuracy: 0.9247 - 325ms/epoch - 1ms/step
```

Out[54]:

```
[0.2956417500972748, 0.9247000217437744]
```

Best initializers in these choices

Model with `kernel_initializer='random_normal' or 'random_uniform'`, `bias_initializer='zeros' or 'random_normal'`

In [55]:

```
model_init = tf.keras.Sequential([
                             tf.keras.layers.Flatten(input_shape=(28, 28)),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='random_normal', bia
s_initializer='random_normal'),
                             tf.keras.layers.Dense(16, activation='relu', kernel_initializer='random_normal', bia
s_initializer='random_normal'),
                             tf.keras.layers.Dense(10, activation='softmax', kernel_initializer='random_normal',
bias_initializer='random_normal')
])
```

In [56]:

```
model_init.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['sparse_categorical_accuracy']
                  )
```

In [ ]:

```
history_init = model_init.fit(x_train, y_train,
                                batch_size=128,
                                epochs=100,
                                validation_data=(x_test, y_test),
                                verbose=2
                                )
```

In [58]:

```
model_init.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.2948 - sparse_categorical_accuracy: 0.9521 - 321ms/epoch - 1ms/step
```

Out[58]:

```
[0.2948480546474457, 0.9520999789237976]
```

Plots

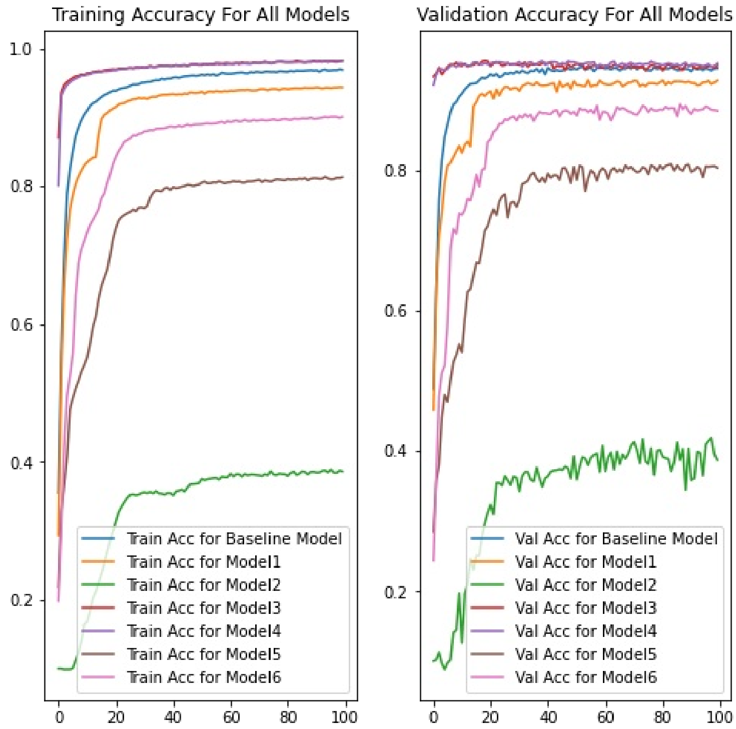
In [59]:

```
import matplotlib.pyplot as plt
```

Plots for different kernel initializers

In [60]:

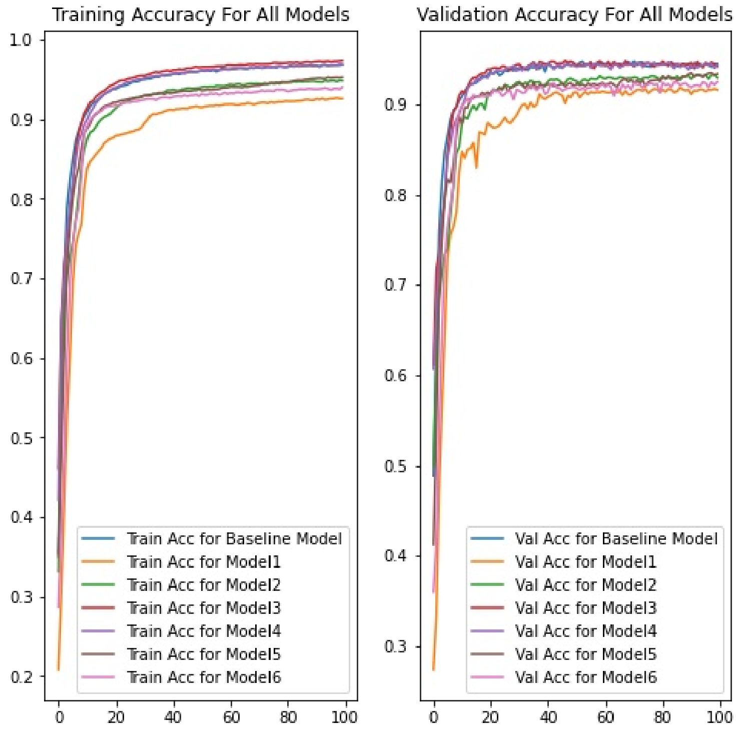
```
training_accuracy_kernel0 = history.history['sparse_categorical_accuracy']
validation_accuracy_kernel0 = history.history['val_sparse_categorical_accuracy']

training_accuracy_kernel1 = history_kernel1.history['sparse_categorical_accuracy']
validation_accuracy_kernel1 = history_kernel1.history['val_sparse_categorical_accuracy']

training_accuracy_kernel2 = history_kernel2.history['sparse_categorical_accuracy']
validation_accuracy_kernel2 = history_kernel2.history['val_sparse_categorical_accuracy']

training_accuracy_kernel3 = history_kernel3.history['sparse_categorical_accuracy']
validation_accuracy_kernel3 = history_kernel3.history['val_sparse_categorical_accuracy']

training_accuracy_kernel4 = history_kernel4.history['sparse_categorical_accuracy']
validation_accuracy_kernel4 = history_kernel4.history['val_sparse_categorical_accuracy']

training_accuracy_kernel5 = history_kernel5.history['sparse_categorical_accuracy']
validation_accuracy_kernel5 = history_kernel5.history['val_sparse_categorical_accuracy']

training_accuracy_kernel6 = history_kernel6.history['sparse_categorical_accuracy']
validation_accuracy_kernel6 = history_kernel6.history['val_sparse_categorical_accuracy']

epochs_range=range(100)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, training_accuracy_kernel0, label='Train Acc for Baseline Model')
plt.plot(epochs_range, training_accuracy_kernel1, label='Train Acc for Model1')
plt.plot(epochs_range, training_accuracy_kernel2, label='Train Acc for Model2')
plt.plot(epochs_range, training_accuracy_kernel3, label='Train Acc for Model3')
plt.plot(epochs_range, training_accuracy_kernel4, label='Train Acc for Model4')
plt.plot(epochs_range, training_accuracy_kernel5, label='Train Acc for Model5')
plt.plot(epochs_range, training_accuracy_kernel6, label='Train Acc for Model6')
plt.legend(loc='lower right')
plt.title('Training Accuracy For All Models')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, validation_accuracy_kernel0, label='Val Acc for Baseline Model')
plt.plot(epochs_range, validation_accuracy_kernel1, label='Val Acc for Model1')
plt.plot(epochs_range, validation_accuracy_kernel2, label='Val Acc for Model2')
plt.plot(epochs_range, validation_accuracy_kernel3, label='Val Acc for Model3')
plt.plot(epochs_range, validation_accuracy_kernel4, label='Val Acc for Model4')
plt.plot(epochs_range, validation_accuracy_kernel5, label='Val Acc for Model5')
plt.plot(epochs_range, validation_accuracy_kernel6, label='Val Acc for Model6')
plt.legend(loc='lower right')
plt.title('Validation Accuracy For All Models')
plt.show()
```



Plots for different bias initializers

In [61]:

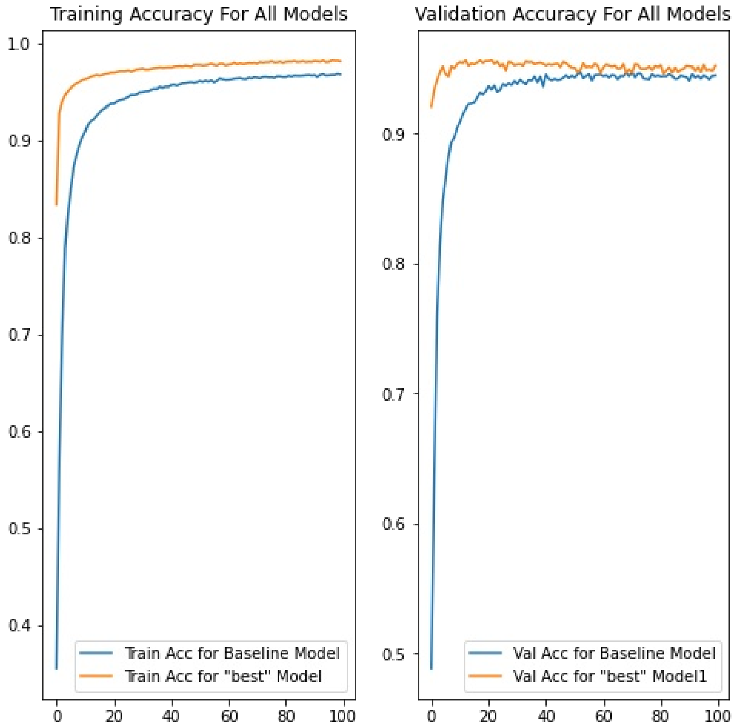
```
training_accuracy_bias0 = history.history['sparse_categorical_accuracy']
validation_accuracy_bias0 = history.history['val_sparse_categorical_accuracy']

training_accuracy_bias1 = history_bias1.history['sparse_categorical_accuracy']
validation_accuracy_bias1 = history_bias1.history['val_sparse_categorical_accuracy']

training_accuracy_bias2 = history_bias2.history['sparse_categorical_accuracy']
validation_accuracy_bias2 = history_bias2.history['val_sparse_categorical_accuracy']

training_accuracy_bias3 = history_bias3.history['sparse_categorical_accuracy']
validation_accuracy_bias3 = history_bias3.history['val_sparse_categorical_accuracy']

training_accuracy_bias4 = history_bias4.history['sparse_categorical_accuracy']
validation_accuracy_bias4 = history_bias4.history['val_sparse_categorical_accuracy']

training_accuracy_bias5 = history_bias5.history['sparse_categorical_accuracy']
validation_accuracy_bias5 = history_bias5.history['val_sparse_categorical_accuracy']

training_accuracy_bias6 = history_bias6.history['sparse_categorical_accuracy']
validation_accuracy_bias6 = history_bias6.history['val_sparse_categorical_accuracy']

epochs_range=range(100)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, training_accuracy_bias0, label='Train Acc for Baseline Model')
plt.plot(epochs_range, training_accuracy_bias1, label='Train Acc for Model1')
plt.plot(epochs_range, training_accuracy_bias2, label='Train Acc for Model2')
plt.plot(epochs_range, training_accuracy_bias3, label='Train Acc for Model3')
plt.plot(epochs_range, training_accuracy_bias4, label='Train Acc for Model4')
plt.plot(epochs_range, training_accuracy_bias5, label='Train Acc for Model5')
plt.plot(epochs_range, training_accuracy_bias6, label='Train Acc for Model6')
plt.legend(loc='lower right')
plt.title('Training Accuracy For All Models')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, validation_accuracy_bias0, label='Val Acc for Baseline Model')
plt.plot(epochs_range, validation_accuracy_bias1, label='Val Acc for Model1')
plt.plot(epochs_range, validation_accuracy_bias2, label='Val Acc for Model2')
plt.plot(epochs_range, validation_accuracy_bias3, label='Val Acc for Model3')
plt.plot(epochs_range, validation_accuracy_bias4, label='Val Acc for Model4')
plt.plot(epochs_range, validation_accuracy_bias5, label='Val Acc for Model5')
plt.plot(epochs_range, validation_accuracy_bias6, label='Val Acc for Model6')
plt.legend(loc='lower right')
plt.title('Validation Accuracy For All Models')
plt.show()
```



Plots for different initializers

In [63]:

```
training_accuracy0 = history.history['sparse_categorical_accuracy']
validation_accuracy0 = history.history['val_sparse_categorical_accuracy']

training_accuracy_init = history_init.history['sparse_categorical_accuracy']
validation_accuracy_init = history_init.history['val_sparse_categorical_accuracy']

epochs_range=range(100)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, training_accuracy0, label='Train Acc for Baseline Model')
plt.plot(epochs_range, training_accuracy_init, label='Train Acc for "best" Model')
plt.legend(loc='lower right')
plt.title('Training Accuracy For All Models')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, validation_accuracy0, label='Val Acc for Baseline Model')
plt.plot(epochs_range, validation_accuracy_init, label='Val Acc for "best" Model1')
plt.legend(loc='lower right')
plt.title('Validation Accuracy For All Models')
plt.show()
```


Training Accuracy For All Models
Train Acc for Baseline Model
Train Acc for "best" Model
Validation Accuracy For All Models
Val Acc for Baseline Model
Val Acc for "best" Model1