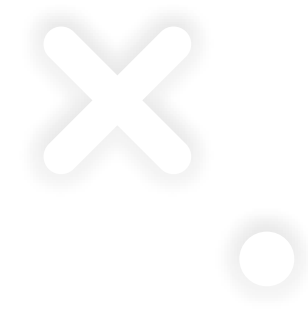
A decorative graphic consisting of three large, light gray 'X' marks and three small, light gray circles. One 'X' is in the top left, one is in the top right, and one is in the bottom right. The circles are positioned near the 'X' marks: one near the top left 'X', one near the top right 'X', and one near the bottom right 'X'.

# APR 2023 – Fil rouge

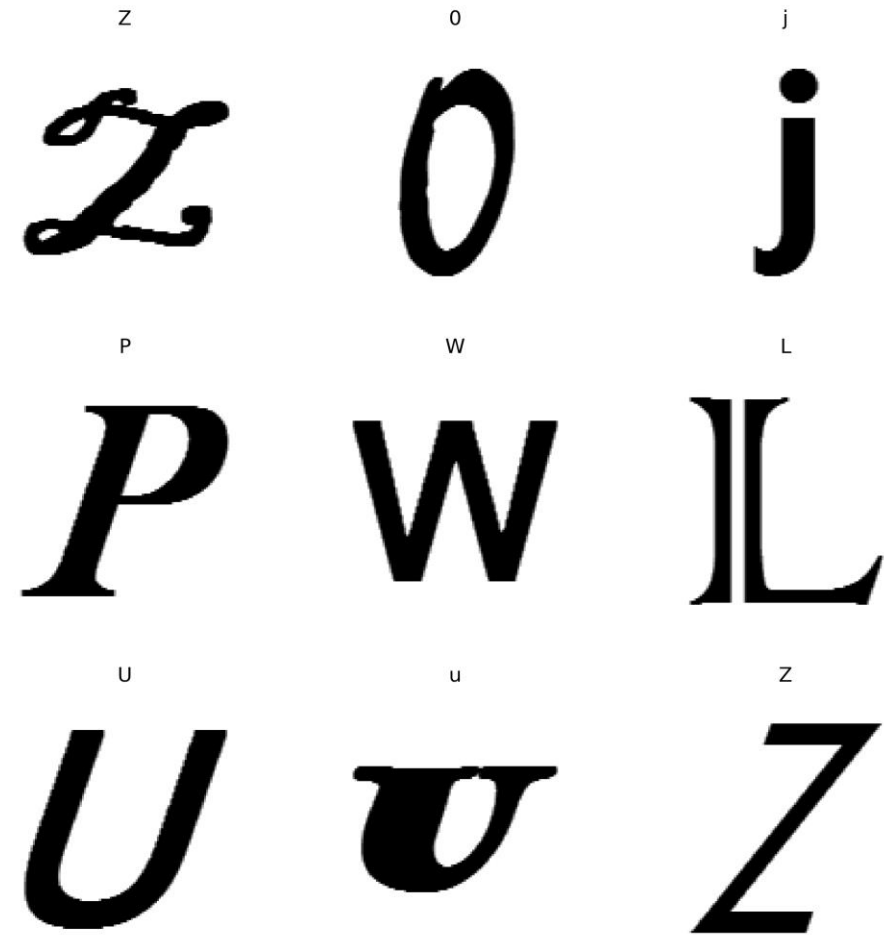
Tom Vaucourt

# Sommaire

1. Problème sélectionné et solution
  2. Modèle 1
  3. Data augmentation et dropout
  4. Modèle 2
  5. Modèle 3
  6. Modèle 4 : optimisation des hyperparamètres
  7. Prédictions sur de nouvelles données
- 

# Problème sélectionné

- + Classification d'images
- + Dataset : **Chars74K dataset**
  - 62992 images provenant de polices d'écriture
  - 0-9, a-z, A-Z = 62 classes
  - 128 x 128 pixels




Échantillons du dataset et labels correspondants

Référence : T. E. de Campos, B. R. Babu and M. Varma. **Character recognition in natural images**. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP), Lisbon, Portugal, February 2009*.

# × Solution

- +Élaboration de réseaux de neurones convolutifs
- +Outils : Kaggle, Nvidia P100
- +Taille des batches : 64 (meilleure utilisation de la mémoire)



# Modèle 1 : définition

- + Réseau simple inspiré de <https://www.tensorflow.org/tutorials/images/classification>
- + Initialement pour des images 180 x 180 donc pas de modifications de *kernel\_size* ou *strides*
- + Couche de sortie entièrement connectée de 62 neurones

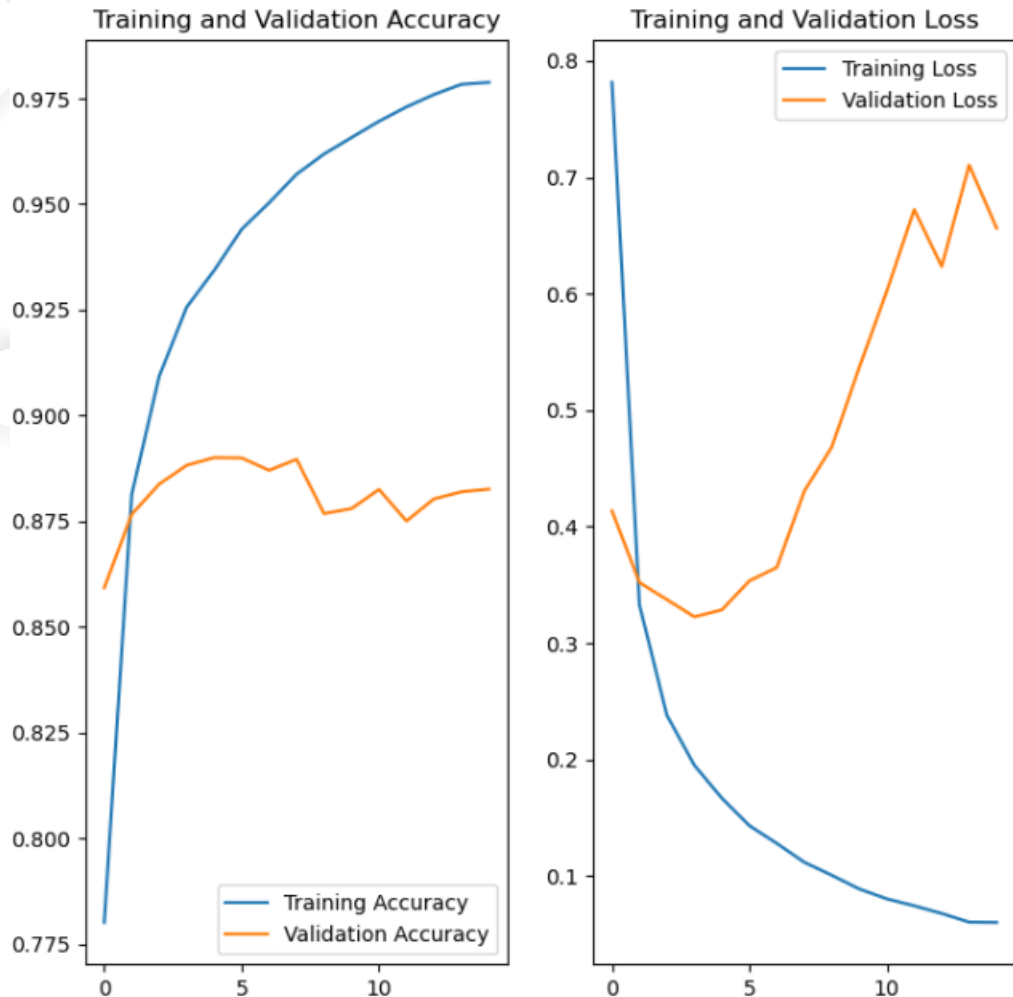
```
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes)
])
```

# Modèle 1 : compilation

- + Optimiseur : algorithme d'Adam
  - Descente de gradient stochastique
  - Estimation du moment de premier et second ordre
  - On conserve les paramètres par défaut pour l'instant
- + Loss : SparseCategoricalCrossentropy car les labels sont des entiers
- + Metric : accuracy pour un problème de classification

```
model.compile(  
    optimizer='adam',  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    #on veut une probabilité d'appartenance aux classes  
    metrics=['accuracy'])
```

# Modèle 1 : entraînement



```
history = model.fit(  
    train_ds,  
    validation_data=val_ds,  
    epochs=epochs  
)
```

- + On commence par 15 époques (7 mins)
- + On observe du **surapprentissage** à partir de l'époque 5, ce qui est rapide
- + Précision de 89% et perte de 0,33 sur le dataset de validation
- + Comment empêcher ce **surapprentissage** ?

# Data augmentation

- + Augmenter la diversité et la taille du jeu de données d'entraînement en appliquant des transformations aléatoires mais réalistes
- + Transformations :
  - Rotation :  $0.02 * 2\pi \text{ rad} = 7,2^\circ$
  - Zoom : 10% en largeur ou hauteur car les caractères sont déjà assez normalisés en taille
  - Fill\_mode : constant, toutes les valeurs extrapolées sont égales à fill\_value = 255 donc du blanc

```
data_augmentation = tf.keras.Sequential(  
    [  
        tf.keras.layers.RandomRotation(0.02, fill_mode='constant', fill_value=255.0),  
        tf.keras.layers.RandomZoom(height_factor=0.1, width_factor=0.1, fill_mode='constant', fill_value=255.0),  
    ]  
)
```

*m m m*

*m m m*

*m m m*

*Effets de l'augmentation des données*



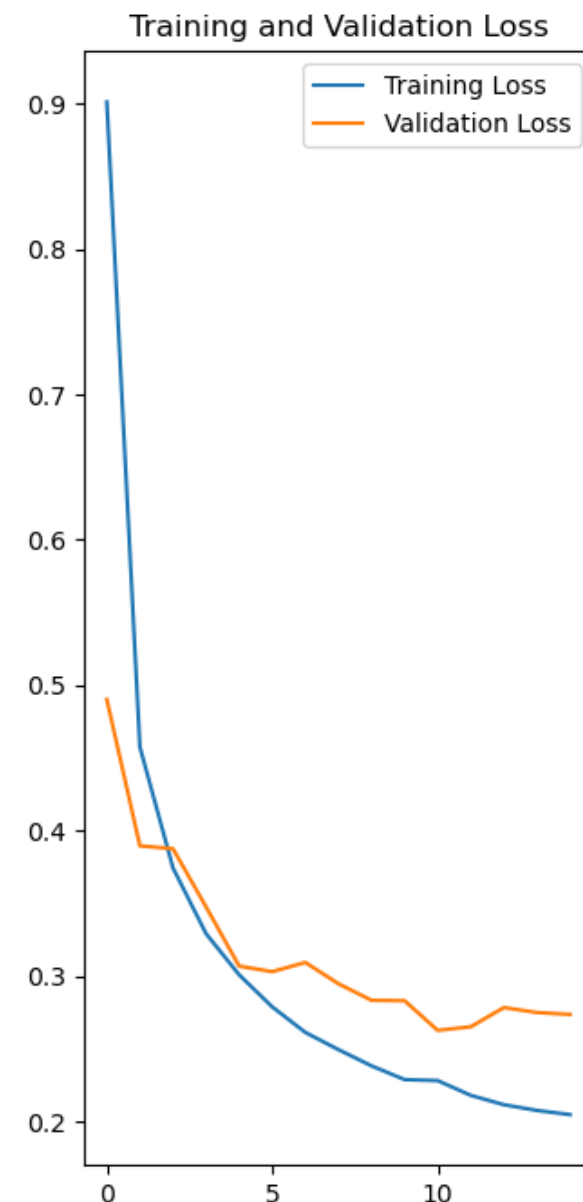
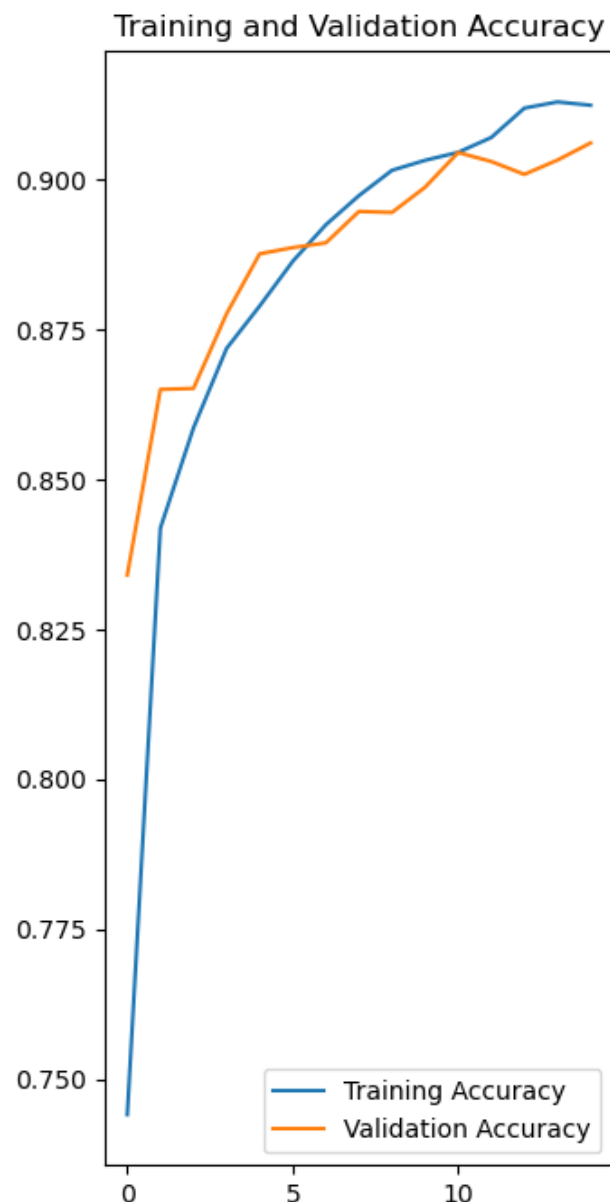
# Dropout


```
tf.keras.layers.Dropout(0.2)
```

- + Désactiver aléatoirement une fraction des neurones de la couche précédente lors de l'entraînement d'un réseau de neurones
  - réduire le surapprentissage
  - améliorer la robustesse du modèle en le forçant à apprendre des caractéristiques redondantes et indépendantes
- + On le fixe ici à 20% pour commencer

# Modèle 2 = Modèle 1 + data augmentation

- + Toujours 15 époques
- + Convergence après 10 époques
- + Précision de 90% et perte de 0,26 sur le dataset de validation
- + Léger gain mais entraînement 2x plus long





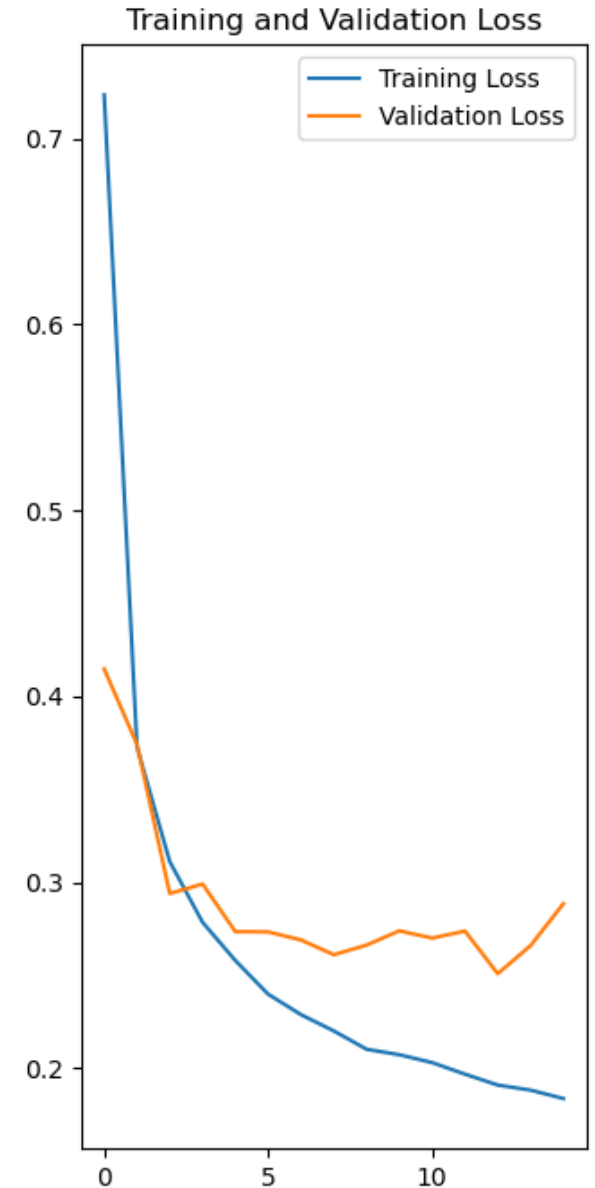
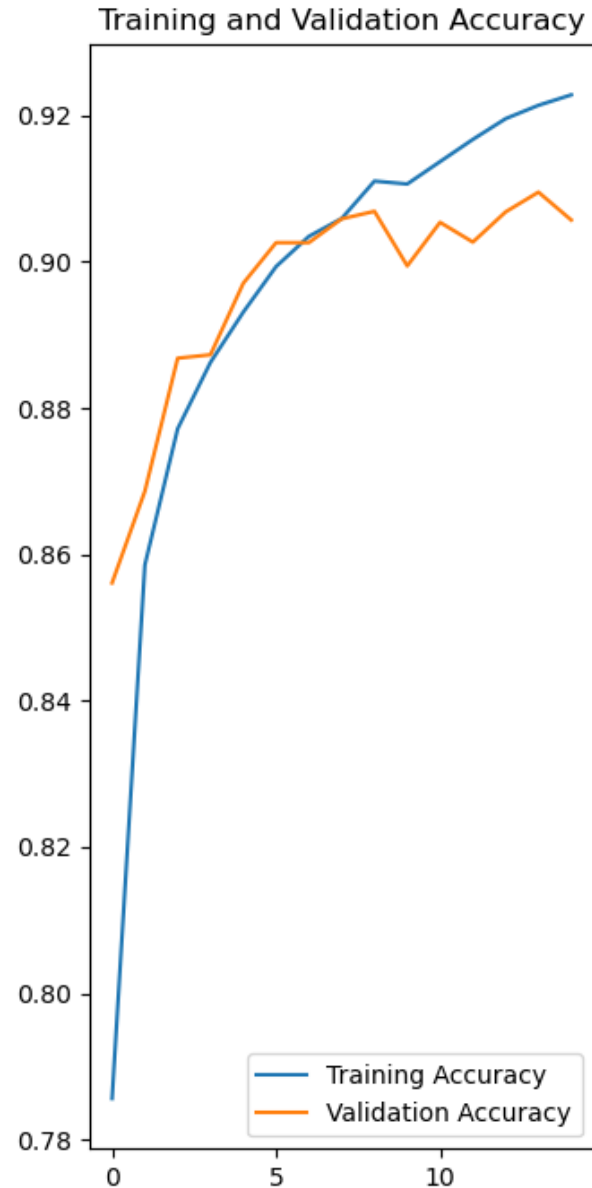
# Modèle 3 : définition

- + Réseau simple inspiré de <https://www.kaggle.com/code/anuraggupta29/optical-character-recognition-chars74k-dataset>
- + Changements : filtres des Conv2D, pool\_size des MaxPooling2D
- + Augmentation de la taille de la dernière couche cachée à 310 neurones
- + On augmente le dropout car plus de neurones
- + On garde la data augmentation

```
model_3 = tf.keras.Sequential([
    data_augmentation,
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 4, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size = 4, strides = 2),
    tf.keras.layers.Conv2D(64, 4, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size = 4, strides = 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.4), #on ajoute également un layer de dropout
    tf.keras.layers.Dense(310, activation='relu'),
    tf.keras.layers.Dense(num_classes, name="outputs")
])
```

# Modèle 3

- + Convergence après 10 époques
- + Précision de 91% et perte de 0,25 sur le dataset de validation
- + Léger gain, entraînement d'une durée similaire
- + Ces paramètres semblent plus adaptés, essayons de les optimiser



# Modèle 4 : définition

- + On va optimiser les hyperparamètres de ce modèle
  - Nombre de filtres dans les 3 couches Conv2D (16/32 et 16/32/48/64 pour la dernière)
  - Taux de dropout (entre 0,3 et 0,7)
  - Nombre de neurones dans la dernière couche cachée dense (entre 256 et 512 par pas de 32)
- + On doit créer un constructeur que le tuner va appeler

```
def model_builder(hp):  
    model = tf.keras.Sequential([  
        data_augmentation,  
        tf.keras.layers.Rescaling(1./255)])  
  
    hp_filters_1 = hp.Int('filters_hp_1', min_value = 16, max_value = 32, step = 16)  
    model.add(tf.keras.layers.Conv2D(hp_filters_1, 4, padding='same', activation='relu'))  
    model.add(tf.keras.layers.MaxPooling2D(pool_size = 4, strides = 2))  
  
    hp_filters_2 = hp.Int('filters_hp_2', min_value = 16, max_value = 32, step = 16)  
    model.add(tf.keras.layers.Conv2D(hp_filters_2, 4, padding='same', activation='relu'))  
    model.add(tf.keras.layers.MaxPooling2D(pool_size = 4, strides = 2))  
  
    hp_filters_3 = hp.Int('filters_hp_3', min_value = 16, max_value = 64, step = 16)  
    model.add(tf.keras.layers.Conv2D(hp_filters_3, 4, padding='same', activation='relu'))  
    model.add(tf.keras.layers.MaxPooling2D(pool_size = 4, strides = 2))  
  
    model.add(tf.keras.layers.Flatten())  
  
    hp_dropout = hp.Float('dropout', 0.3, 0.7, step=0.1)  
    model.add(tf.keras.layers.Dropout(hp_dropout))  
  
    hp_units = hp.Int('units', min_value=256, max_value=512, step=32)  
    model.add(tf.keras.layers.Dense(units=hp_units, activation='relu'))  
  
    model.add(tf.keras.layers.Dense(num_classes, name="outputs"))  
  
    #hp_learning_rate = hp.Choice('learning_rate', values=[1e-3, 1e-4])  
    model.compile(optimizer=tf.keras.optimizers.Adam(), #learning_rate=hp_learning_rate),  
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
                  metrics=['accuracy'])  
  
    return model
```

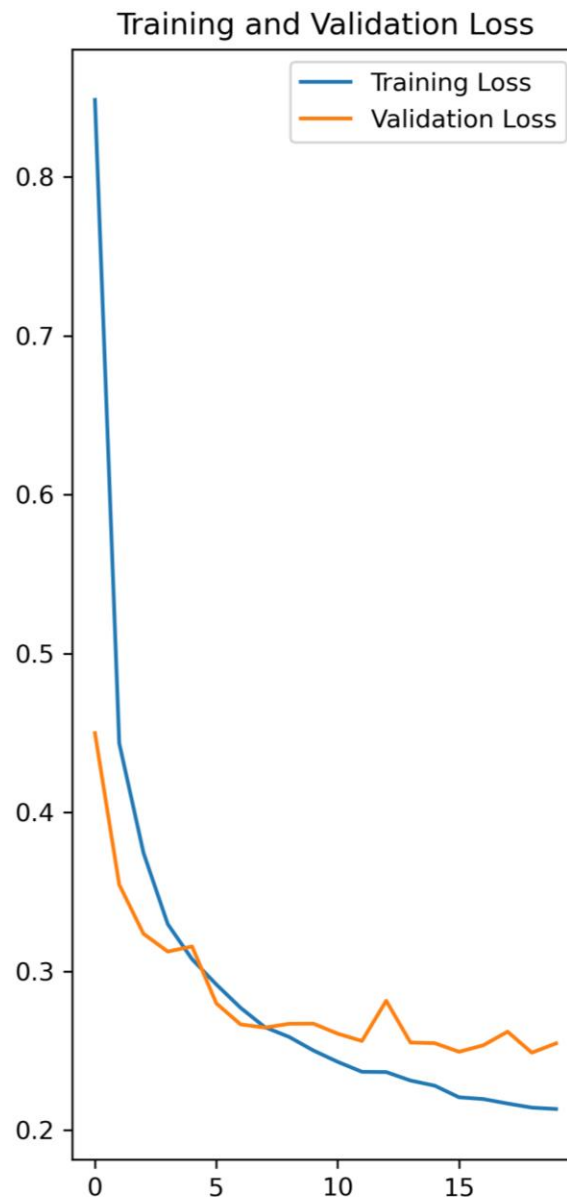
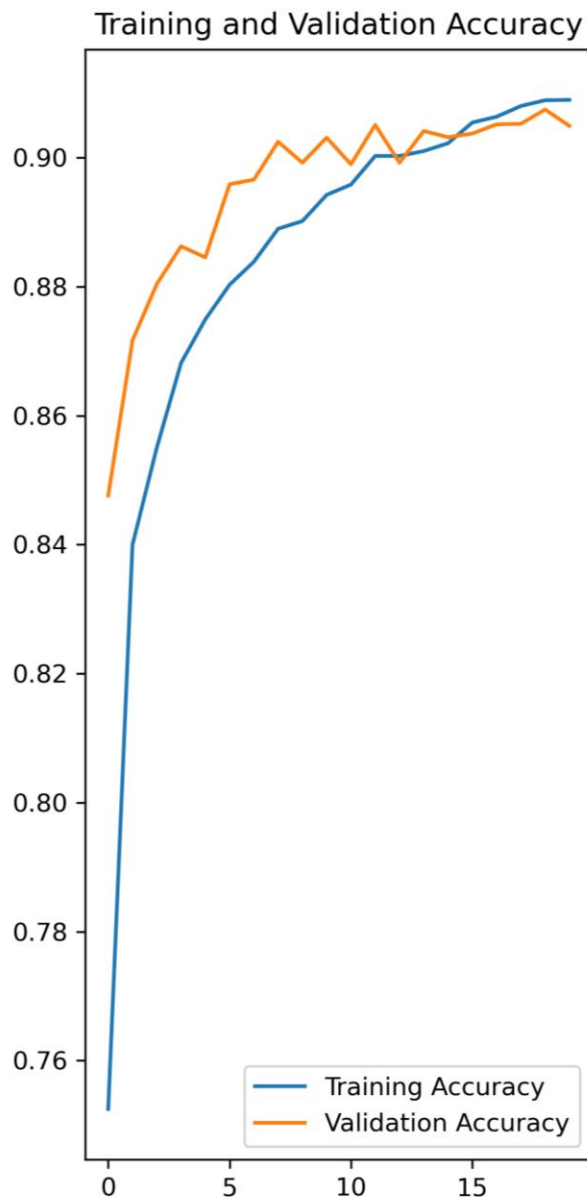
# Optimisation des hyperparamètres

- + On va utiliser le tuner **Hyperband** de la bibliothèque **keras\_tuner** pour la rapidité
- + Divise les modèles en brackets. Les modèles les plus performants de chaque bracket sont promus au tour suivant, où ils sont entraînés pour plus d'époques, jusqu'à ce qu'un seul modèle reste.
- + Performance : précision sur le dataset de validation
- + Paramètres:
  - max\_epochs : nombre max d'époques d'entraînement pour un seul modèle
  - factor : réduction du nombre de modèles et époques à chaque nouveau bracket

```
import keras_tuner as kt
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='model_4',
                     project_name='model_4_tuning')
```

# Modèle 4 : résultats

- + 1h11 de recherche
- + Hyperparamètres:
  - filters\_hp\_1 : 16
  - filters\_hp\_2 : 32
  - filters\_hp\_3 : 64
  - dropout : 0.6
  - units : 288
- + Précision de 91% et perte de 0,25 sur le dataset de validation
- + On a obtenu un modèle aussi performant que le précédent mais qui converge en 19 époques...
- + Les époques sont cependant plus rapides que le modèle 3, similaires au modèle 2.



# Prédictions sur des nouvelles données

- + On utilise MINST et le dataset Alphabet Characters Fonts Dataset (<https://www.kaggle.com/datasets/thomasqazwsxedc/alphabet-characters-fonts-dataset>)
- + On va essayer de lui faire reconnaître les textes « APR » et « 2023 »
- + On choisit une police au hasard pour chaque lettre dans le Alphabet Characters Fonts Dataset
- + On choisit une police au hasard pour chaque chiffre dans le dataset MINST

